REALIZAÇÃO

SBC
Sociedade Brasileira de Computação

WebMedia 2019

XXV SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB

29 DE OUTUBRO A 01 DE NOVEMBRO

Rio de Janeiro - RJ

MINICURSOS

ORGANIZADORES

Manoel Carvalho Marques Neto (IFBA) • Glauco Fiorott Amorim (CEFET/RJ)
Joel dos Santos (CEFET/RJ) • Débora C. Muchaluat-Saade (UFF) • Roberto Willrich (UFSC)

PATROCÍNIO

nic.br CAPES CNPq FAPERJ
cgi.br IBM Research Google

COOPERAÇÃO

acm In-Cooperation
sigweb
SIGMM

ORGANIZAÇÃO

CEFET/RJ
PPCIC Programa de Pós Graduação em Ciência da Computação
Grupo de Pesquisa em MultiMídia
Universidade Federal Fluminense
Instituto de Computação
Laboratório MídiaCom

Editora
Sociedade Brasileira de Computação (SBC)

XXV Simpósio Brasileiro de Sistemas Multimídia e Web

De 29 de outubro a 1 de novembro de 2019

Rio de Janeiro, Brasil

ANAIS

MINICURSOS

Organizadores

Manoel C. M. Neto (IFBA)
Glauco Fiorott Amorim (CEFET/RJ)
Joel dos Santos (CEFET/RJ)
Débora C. Muchaluat-Saade (UFF)
Roberto Willrich (UFSC)

Realização

Sociedade Brasileira de Computação – SBC

Em cooperação com

ACM/SIGWEB e ACM/SIGMM

Organização

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Universidade Federal Fluminense (UFF)

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

S612 XXV Simpósio Brasileiro de Sistemas Multimídia e Web. Minicursos (1. 2019 : Rio de Janeiro, RJ). [recurso eletrônico] / WebMedia 2019. XXV Simpósio Brasileiro de Sistemas Multimídia e Web, 29 de outubro a 01 de novembro no CEFET/RJ, RJ. - Rio de Janeiro, CEFET/RJ, UFF, SBC, 2019.
xi, 203p. : il.color.

ISBN 978-85-7669-481-6
Disponível em: https://sol.sbr.org.br/livros/index.php/sbc/catalog
Em cooperação com ACM/SIGWEB e ACM/SIGMM.

1. Multimídia. 2. Web. 3. Inteligência artificial. 4. Fluxo de dados (Computadores). 5. Internet das coisas. 6. Mineração de dados. I. Centro Federal de Educação Tecnológica Celso Suckow da Fonseca. II. Universidade Federal Fluminense. III Título.

CDD 006.7

Elaborada pelo bibliotecário Leandro Mota de Menezes CRB-7/5281

Prefácio

Tradicionalmente, o Simpósio Brasileiro de Sistemas Multimídia e Web oferece à sua comunidade minicursos de curta duração relacionados a temas que norteiam os últimos avanços na área de multimídia e web. Este também é o caso do XXV Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2019), onde os minicursos têm a duração de 4 horas e permitem que participantes recebam informações sobre novas tecnologias e tópicos atuais de pesquisa em áreas correlatas ao evento. Assim, minicursos aparecem como uma excelente oportunidade para familiarização dos congressistas com novos temas de pesquisa que podem vir a ser úteis em suas vidas profissionais.

Para o Webmedia 2019, o processo de seleção de minicursos foi feito a partir de várias chamadas públicas divulgadas na lista eletrônica de emails da SBC, bem como ampla divulgação no site oficial do evento e em redes sociais. Foram recebidas 8 (oito) propostas de minicursos, avaliadas por comitê composto de professores com conhecimento nos temas abordados. Cada minicurso foi avaliado por quatro avaliadores, que pontuaram notas para quesitos como relevância para o evento, expectativa do público, atualidade e conteúdo de cada minicurso. Ao final, cinco propostas foram selecionadas, cujos conteúdos abordados constituem os capítulos deste livro.

O primeiro capítulo, intitulado *Semântica e Multimídia: Uma Introdução à Inteligência Artificial Simbólica a Aplicada à Multimídia*, aborda inteligência artificial (IA) simbólica aplicada à multimídia. O minicurso está dividido em três partes. Na primeira parte, apresentam-se as noções gerais de IA simbólica e discutem-se os desafios envolvidos na construção de aplicações multimídia inteligentes, isto é, aplicações enriquecidas com consulta, raciocínio e recomendação semântica, Na segunda parte, são discutidas as principais tecnologias de AI simbólica e a sua aplicação em multimídia. Em particular, apresentam-se as linguagens de representação de conhecimento RDF(S) e OWL, as lógicas de descrição (DLs) que fundamentam essas linguagens, a linguagem de consulta SPARQL, e os vocabulários e extensões que nos permitem aplicar essas tecnologias à dados multimídia. Na terceira e última parte, apresenta-se o modelo de Hiperconhecimento: um modelo híbrido para representação de conhecimento e conteúdo. O modelo de Hiperconhecimento estende o NCM, um modelo clássico de hipermídia, com a noção de elos semânticos, permitindo a representação integrada de conteúdo multimídia e de sua descrição semântica.

O Capítulo dois, *An Introduction to Data Stream Processing: A Complex Event Processing Approach*, tem como objetivo apresentar o modelo de programação de processamento de eventos complexos (CEP) como meio de lidar com as especificidades de fluxos de dados. Especificamente, o minicurso prepara o participante para: (1) entender os conceitos e problemas fundamentais de processamento de fluxo de dados e o modelo CEP; (2) apresentar a tecnologia Esper e a sua linguagem de regras EPL como meio de tratar os fluxo de dados; (3) Exemplificar um caso de uso da aplicação de CEP para tratamento de dados de aplicações IoT; (4) Apresentar um outro caso de uso de aplicação de CEP como meio de processar dados de aplicações para Smart Cities.

O terceiro capítulo é intitulado *Teoria e Prática de Microserviços Reativos: Um Estudo de Caso na Internet das Coisas*. Em especial, o minicurso tem como objetivo principal oferecer a alunos de graduação, pós graduação e pesquisadores formação para o desenvolvimento de projetos e pesquisas que adotem Microserviços reativos para a implementação das aplicações com foco em resiliência e elasticidade.

Em seguida, o capítulo quatro explora a popularização de equipamentos de captura audiovisual e de serviços para armazenamento e transmissão vídeo, com o título *Métodos baseados em Deep Learning para Análise de Vídeo*. Este cenário apresenta desafios para navegação, busca e recomendação nesse grande volume vídeos. Essas atividades requerem análise automática desse volume de forma eficiente e prática, contemplando não somente a classificação dos vídeo, mas também o entendimento de eventos complexos.

O capítulo cinco é intitulado *Recuperação de Informação Multimídia em Big Data Utilizando OPENCV Python*. Este minicurso aborda ferramentas e técnicas atuais para indexação, extração e processamento de conteúdo multimídia multimodal. As técnicas são exemplificadas em OpenCV Python sobre diferentes conteúdos (imagens, áudio, texto e vídeo), levando ao interesse de serviços como Netflix, Google e YouTube nesse assunto, despertando o interesse de pesquisadores e desenvolvedores.

Esperamos que este livro seja útil para todos aqueles interessados e praticantes da área de Sistemas Multimídia e Web.

Rio de Janeiro, outubro de 2019.

Manoel Carvalho Marques Neto (IFBA)
Glauco Fiorott Amorim (CEFET/RJ)
Coordenadores dos Minicursos do WebMedia 2019

XXV Simpósio Brasileiro de Sistemas Multimídia e Web

29 de outubro a 1 de novembro de 2019
Rio de Janeiro, Brasil

Coordenação Geral

Joel dos Santos (CEFET/RJ) – *Coordenador Geral*
Débora C. Muchaluat-Saade (UFF) – *Coordenadora Geral*
Maria da Graça Campos Pimentel (USP) – *Coordenadora do Comitê de Programa*
Alessandra Alaniz Macedo (USP-RP) – *Coordenadora do Comitê de Programa*

Organização Local

Glauco Fiorott Amorim (CEFET/RJ)
Diego Brandão (CEFET/RJ)

Coordenador de Publicação

Roberto Willrich (UFSC)

Coordenadores do I Concurso de Teses e Dissertações (WTD)

Windson Viana (UFC)
Celso Alberto Saibel Santos (UFES)

Coordenadores do XVI Workshop de Trabalhos de Iniciação Científica (WTIC)

Álan Lívio Guedes (Puc-Rio)
Kele Belloze (CEFET/RJ)

Coordenadores do XVIII Workshop de Ferramentas e Aplicações (WFA)

Carlos Ferraz (UFPE)
Glauco Amorim (CEFET/RJ)

Coordenadores de Minicursos

Manoel Carvalho Marques Neto (IFBA)
Glauco Fiorott Amorim (CEFET/RJ)

Coordenador do VI Workshop "O Futuro da Videocolaboração"

Gustavo Dias Neves (RNP)
Valter Roesler (UFRGS)

Coordenador do V Workshop Futuro da TV Digital Interativa

Marcelo Ferreira Moreno (UFJF)
Débora Christina Muchaluat Saade (UFF)
Guido Lemos de Souza Filho (UFPB)

Coordenação da Comissão Especial de Sistemas Multimídia e Web

Adriano C. Machado Pereira (UFMG) – *Coordenador*
Windson Viana (UFC) – *Vice-Coordenador*

Comitê Gestor

Carlos de Salles Soares Neto (UFMA)
Celso Alberto Saibel Santos (UFES)
Fábio de Jesus Lima Gomes (IFPI)
Guido Lemos de Souza Filho (UFPB)
José Valdeni de Lima (UFRGS)
Manoel C.M. Neto (IFBA)
Maria da Graça C. Pimentel (USP)
Roberto Willrich (UFSC)
Valter Roesler (UFRGS)

Sociedade Brasileira de Computação (SBC)

Presidência

Raimundo José de Araújo Macêdo (UFBA) – *Presidente*
André Carlos P. de L. F. de Carvalho (USP) – *Vice-Presidente*

Diretoria

Carlos André Guimarães Ferraz (UFPE)
Carlos Eduardo Ferreira (USP)
Cristiano Maciel (UFMT)
Edson Norberto Cáceres (UFMS)
Francisco Dantas de Medeiros Neto (UERN)
Itana Maria de Souza Gimenes (UEM)
José Viterbo Filho (UFF)
Marcelo Duduchi Feitosa (CEETEPS)
Priscila América Solís Mendez Barreto (UNB)
Renata Galante (UFGRS)
Rossana Maria de Castro Andrade (UFC)
Wagner Meira (UFMG)

Diretoria Extraordinária

Leila Ribeiro (UFRGS)

Contato

Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS
CNPJ: 29.532.264/0001-78
http://www.sbc.org.br

Sumário

Capítulo 1

Semântica e Multimídia: Uma Introdução à Inteligência Artificial Simbólica Aplicada à Multimídia

Guilherme Lima[1]
Rodrigo Costa[1]
Marcio Ferreira Moreno[1]
[1]IBM Research, Brazil

Guilherme.Lima@ibm.com,Rodrigo.Costa@ibm.com,mmoreno@br.ibm.com

Abstract

In this chapter, we give an introduction to symbolic artificial intelligence (AI) from first principles and discuss its relation and application to multimedia. We begin by defining what symbolic AI is, what distinguishes it from non-symbolic approaches, such as machine learning, and how it can used in the construction of advanced multimedia applications. We then introduce description logic (DL) and use it to discuss symbolic representation and reasoning. DL is the logical underpinning of OWL, the most successful family of ontology languages. After discussing DL, we present OWL and related Semantic Web technologies, such as RDF and SPARQL. We conclude the chapter by discussing a hybrid model for multimedia representation, called Hyperknowledge. Throughout the chapter, we strive to make references to technologies and extensions specifically designed to solve the kinds of problems that arise in multimedia representation.

Resumo

Neste capítulo apresentamos uma introdução à inteligência artificial (IA) simbólica a partir de seus princípios básicos e discutimos a sua relação com e aplicação à multimídia. Começamos definindo o que é IA simbólica, o que a distingue de abordagens não-simbólicas, como o aprendizado de máquina, e como ela pode ser utilizada para construir aplicações multimídia avançadas. Em seguida, apresentamos a lógica de descrição (DL) e a usamos para discutir a representação e o raciocínio simbólico. DL é a lógica que fundamenta OWL, a mais bem sucedida família de linguagens ontológicas. Após DL, apresentamos OWL e as tecnologias de Web Semântica relacionadas, como RDF e SPARQL. Concluímos o capítulo apresentando um modelo híbrido para representação de multimídia, chamado Hyperknowledge. Durante todo o capítulo, procuramos fazer referência às tecnologias e extensões especificamente projetadas para resolver os tipos de problemas que aparecem no contexto de representação multimídia.

1.1. Introdução

Um problema clássico em representação e compreensão de dados multimídia é o problema do *gap* (vão) semântico. Ele afirma que há uma grande distância de representação entre os sinais audiovisuais que compõem os objetos multimídia e os conceitos representados por esses sinais. Por exemplo, a cor dominante e a trajetória de movimento de um dado conjunto de pixels em um vídeo, ambas características de baixo nível do vídeo, em geral não fornecem informações suficientes sobre o *significado* desse conjunto de pixels—pelo menos não para os computadores. Mas avanços recentes em inteligência artificial (IA) estão mudando isso.

Métodos de aprendizado de máquina atuais, apoiados por vastos conjuntos de dados de treinamento, são capazes de extrapolar padrões complexos a partir de dados multimídia de baixo nível. Esses padrões são materializados em modelos treinados que podem ser usados para identificar pessoas e objetos rapidamente e com precisão razoável em imagens, trechos de áudio, e em menor escala trechos de vídeo. Porém, a identificação de pessoas e objetos em dados multimídia resolve apenas metade do prolema. Para emular a cognição humana e realmente entender uma cena—por exemplo, para determinar quem está fazendo o que e as consequências dessas ações—os computadores precisam de informações adicionais: eles precisam de conhecimento geral de mundo e de conhecimento de domínio, e também precisam da capacidade de inferir conhecimento novo a partir de conhecimento preexistente. E aí que entra a inteligência artificial simbólica.

A ideia básica de IA simbólica é descrever o mundo, suas entidades e seus relacionamentos através de uma linguagem formal—uma linguagem que pode ser convenientemente manipulada por computadores—e desenvolver algoritmos eficientes para consultar e deduzir coisas a partir dessas descrições formais. Para ilustrar o tipo de aplicação que a combinação entre IA simbólica e multimídia possibilita considere a Figura 1.1.

Suponha que essa figura nos foi apresentada e suponha que a única coisa que sabemos sobre ela é o que podemos inferir a partir da imagem. Podemos ver que ela retrata Jean-Paul Marat (assumindo que sabemos identificá-lo), um ferimento semelhante ao causado por uma punhalada, uma faca com manchas de sangue, e uma carta endereçada a Marat e assinada por Charlotte Corday (assumindo que podemos ler o conteúdo da carta). A analogia aqui é que extraímos essas informações—ou *fatos*—através do reconhecimento de padrões visuais presentes na imagem. Apesar desses fatos básicos nos permitirem realizar tarefas computacionais simples, como a classificação e busca de imagens indexadas por palavras-chave, eles não são suficientes para entendermos a imagem.

Para realmente *entendermos* o que a Figura 1.1 retrata precisamos ir além dos fatos básicos. Precisamos (1) de conhecimento geral de mundo, (2) de conhecimento específico sobre as pessoas mencionadas, e (3) da capacidade de combinar esses conhecimentos gerais e específicos com os fatos extraídos da imagem e a partir disso inferir novos fatos.

Suponha que (1), (2) e (3) nos sejam dados. A partir do nosso conhecimento geral de mundo, e talvez de uma nova análise mais aprofundada da imagem, podemos afirmar com confiança que Marat está segurando a carta e que ele possui um ferimento no tórax. Disso e da faca manchada de sangue que aparece abaixo de Marat, podemos concluir que é provável que a faca retratada seja o objeto que causou o ferimento. Como facas não são seres autônomos, podemos também concluir que alguém (talvez o próprio Marat)

Figura 1.1. *A Morte de Marat* (detalhe), por Jacques-Louis David, 1793. (WikiMedia)

esfaqueou-o no peito. Mas quem e por quê?

Para responder a essas perguntas vamos precisar de mais informações. Suponha que nos digam que Marat foi um jornalista e agitador político, e um dos líderes de uma facção política radical durante o período do Terror da Revolução Francesa (c. 1793). Suponha também que nos digam que Charlotte Corday, que assina a carta, foi uma inimiga declarada de Marat—ela o culpava por diversos assassinatos em Paris e outra cidades e o considerava uma grave ameaça à República Francesa. À luz desses novos fatos, podemos concluir que a Figura 1.1 parece retratar um assassinato político.

Ao combinarmos essa conclusão com o fato adicional de que é sabido que Charlotte Corday assassinou Jean-Paul Marat com uma faca enquanto ele estava na banheira segurando uma carta assinada por ela, podemos inferir com um alto grau de confiança que a Figura 1.1 se trata de uma representação gráfica desse incidente, isto é, do assassinato por motivações políticas de Jean-Paul Marat por Charlotte Corday.

A derivação desse último fato a partir dos padrões visuais da Figura 1.1 só foi possível porque tivemos acesso não só a fatos básicos extraídos da imagem, mas também a fatos gerais sobre o mundo (conhecimento de senso comum) e sobre os objetos e pessoas retratadas (conhecimento de domínio), e também porque pudemos de combinar todos esses fatos e fazer inferências.

Um dos principais objetivos de IA simbólica é permitir a representação e manipulação de pedaços de conhecimento por computadores de formas que se assemelhem ou emulem os tipos de manipulação realizados por pessoas—manipulações similares às considerações que nos levaram a determinar o verdadeiro significado da Figura 1.1. A combinação dessa capacidade com multimídia abre muitas possibilidades. O exemplo do assassinato de Marat é uma instância de aplicação de compreensão automática de imagens. Duas aplicações relacionadas são a compreensão de vídeo e de áudio, normalmente mais complexas por envolverem a extração de informação temporal.

Outras aplicações que misturam IA simbólica e multimídia incluem a recuperação, classificação, recomendação e inspeção semântica de dados multimídia—por exemplo, para detectar automaticamente atividades suspeitas em imagens de vigilância, gerar classificação indicativa de músicas e filmes, e identificar fatores de risco associados à doenças em imagens médicas. Poderíamos listar muitas outras aplicações interessantes, algumas das quais vamos discutir mais adiante, mas nosso foco aqui é outro. Estamos mais interessados em apresentar os princípios e tecnologias que possibilitam a concepção dessas aplicações em primeiro lugar.

Neste capítulo apresentamos uma introdução à IA simbólica a partir da perspectiva de multimídia. Começamos apresentando a lógica de descrição (*Description Logic*, ou DL) na Seção 1.2. O motivo de começarmos com a lógica de descrição é que ela nos permite discutir a representação simbólica num contexto abstrato, livre das complicações de uma tecnologia específica. Outro motivo é que a lógica de descrição é em si uma tecnologia prática: ela é a lógica que fundamenta a família mais expressiva de linguagens ontológicas, a família OWL.

Após apresentarmos a lógica de descrição, na seção seguinte (Seção 1.3), tratamos da principal manifestação de IA simbólica na Web, a chamada *Web Semântica*. Discutimos a visão da Web Semântica, as tecnologias por trás dessa visão (RDF, OWL, SPARQL, SWRL, etc.) e a relação entre essas tecnologias e as noções apresentadas na Seção 1.2. Em ambas as seções, sempre que possível motivamos a discussão com exemplos do domínio de multimídia. Também sempre que possível apresentamos ou mencionamos metodologias e extensões projetadas especificamente para resolver os tipos de problemas que aparecem no contexto de representação multimídia.

Concluímos o capítulo com a discussão de um modelo híbrido para representação multimídia (Seção 1.4). Esse modelo, chamado *Hyperknowledge* (HK), generaliza um modelo clássico de hipermídia com as noções de nós de conceitos e elos semânticos. Ao fazer isso, o Hyperknowledge permite a representação e o processamento integrados de conteúdo multimídia junto com a sua descrição semântica.

Sugestões de leitura são apresentadas no fim do capítulo (Seção 1.5).

1.2. Lógica(s) de Descrição

A palavra lógica possui muitos significados e nenhuma definição completamente satisfatória. Podemos entender lógica como um sistema para dedução de proposições a partir de outras proposições afirmadas anteriormente. Existem muitos tipos de lógicas. Uma das mais simples é a lógica proposicional, que trata de proposições construídas a partir dos conectivos proposicionais $\neg$ (não), $\wedge$ (e), $\vee$ (ou), $\rightarrow$ (se-então) e $\leftrightarrow$ (se e somente se). A lógica de primeira ordem, utilizada normalmente para formalizar proposições matemáticas, estende a lógica proposicional com as noções de variáveis, constantes, funções, predicados e quantificadores. Há ainda outras extensões, por exemplo, a lógica de segunda ordem, e lógicas que adotam uma abordagem ligeiramente diferente, como as lógicas modais e *fuzzy*.

O termo "lógica de descrição" (DL) não se refere a uma lógica específica mas sim a uma *família* de lógicas. Por esse motivo às vezes falamos das lógica*s* de descrição (no plural). As lógicas dessa família apresentam uma grande variedade entre si,

mas a maioria delas compartilha as mesmas características de modelagem—elas tratam de indivíduos, conceitos e papéis. Neste capítulo, vamos focar na lógica de descrição $\mathcal{SROIQ}$ [Horrocks et al. 2006]. A principais DLs atuais são sublinguagens de $\mathcal{SROIQ}$, que é também a base da linguagem de ontologia OWL 2 DL [W3C OWL WG 2012]. Vamos seguir a maneira usual de se apresentar uma linguagem formal. Começamos apresentando a forma (sintaxe) das proposições de $\mathcal{SROIQ}$ juntamente com o seu significado intuitivo. Mais tarde definiremos o significado exato (semântica) dessas proposições.

1.2.1. Sintaxe

Sejam N_I, N_C e N_R três conjuntos de nomes contendo, respectivamente, nomes de indivíduos, nomes de conceitos e nomes de papéis (*roles*). A ideia aqui é que indivíduos representam entidades do mundo real, conceitos representam características dessas entidades e papéis representam relações entre (duas) entidades.[1] Vamos assumir que os conjuntos N_I, N_C e N_R são disjuntos par a par, isto é, que nenhum nome ocorre ao mesmo tempo em mais de um deles. Esses três conjuntos de nomes constituem o *vocabulário* de $\mathcal{SROIQ}$. Eles contêm os blocos básicos que usaremos para construir as proposições da linguagem.

Vamos distinguir três tipos de proposições, também chamadas em DL de *axiomas*, as quais agruparemos em caixas (*boxes*) correspondentes: ABox, TBox e RBox. A última, RBox, está disponível apenas em lógicas de descrição bastante expressivas, como $\mathcal{SROIQ}$, mas todas as ontologias baseadas em DL possuem uma TBox e a maioria delas uma ABox. Juntas essas três caixas formam uma *base de conhecimento* (*knowledge base*).

A ABox contém o *conhecimento assertivo*: proposições fazem afirmações sobre indivíduos específicos. Por exemplo, a assertiva de conceito

$$\texttt{Diretor}(\texttt{kubrick})$$

afirma que o indivíduo cujo nome é `kubrick` é um diretor. De maneira similar, a assertiva de papel

$$\texttt{dirige}(\texttt{kubrick}, \texttt{space-odyssey})$$

afirma que o indivíduo cujo nome é `kubrick` dirige o indivíduo (isto é, o filme) chamado `space-odyssey`.

A TBox e a RBox contém o *conhecimento terminológico*: proposições que fazem afirmações que se aplicam a todos os indivíduos. A TBox contém proposições que tratam de conceitos e a RBox contém proposições que dizem respeito a papéis. Por exemplo, o axioma de TBox

$$\texttt{Diretor} \sqsubseteq \texttt{Pessoa}$$

afirma que o conceito chamado `Diretor` é subsumido pelo conceito chamado `Pessoa`, isto é, que todo indivíduo que é um diretor é também uma pessoa. Já o axioma de RBox

$$\texttt{dirige} \sqsubseteq \texttt{gosta}$$

[1] Caso o leitor esteja familiarizado com lógica de primeira ordem: nomes de indivíduos são constantes, nomes conceitos são predicados unários e nomes de papéis são predicados binários.

afirma que o papel chamado `dirige` é subsumido pelo papel chamado `gosta`, isto é, que dirigir um filme implica em gostar desse filme.[2]

Essa última afirmação é obviamente questionável. Talvez algum diretor ou diretora em algum lugar do mundo não goste de algum filme que ele ou ela tenha dirigido. No entanto, o ponto aqui é que as proposições em uma base de conhecimento, especialmente aquelas na TBox e RBox, formalizam uma *conceitualização* particular de um domínio. Essa conceitualização reflete a visão de mundo do seu projetista e pode conter simplificações que fazem sentido no seu contexto de uso.

Outra propriedade de uma conceitualização é que ela é independente de linguagem. A conceitualização é apenas um modelo conceitual que pode ser criado em uma linguagem de modelagem qualquer, como UML. O termo *ontologia* é utilizado para se referir a uma instanciação de uma conceitualização em uma linguagem de representação de conhecimento específica. Para os nossos propósitos, o termo ontologia significa o mesmo que base de conhecimento, isto é, um conjunto de axiomas particionados em ABox, TBox e RBox.

De volta à $\mathcal{SROIQ}$, vamos agora definir a sintaxe exata das proposições de cada uma das caixas.

RBox A RBox de $\mathcal{SROIQ}$ contém proposições que descrevem interdependências entre papéis e características de papéis. Um *papel* é uma das seguintes expressões: (1) um nome de papel $\mathtt{r}$, para algum $\mathtt{r}$ em N_R; (2) um nome de papel invertido $\mathtt{r}^-$, para algum $\mathtt{r}$ em N_R; ou (3) o papel universal u.

Um *axioma de inclusão de papel* (*role inclusion axiom*, ou RIA) é uma proposição da forma

$$r_1 \circ \cdots \circ r_n \sqsubseteq r,$$

em que $r_1, \ldots, r_n$ e r são papéis.

Considere o seguinte RIA:

$$\mathtt{paiDe} \sqsubseteq \mathtt{filhoDe}^- .$$

Esse RIA afirma que o papel chamado `paiDe` é subsumido pelo inverso do papel chamado `filhoDe`, isto é, que se a é pai de b então b é filho a, para quaisquer indivíduos a e b. De maneira similar, os RIAs

$$\mathtt{donoDe} \sqsubseteq \mathtt{cuidaDe} \quad \text{e} \quad \mathtt{donoDe} \circ \mathtt{parteDe} \sqsubseteq \mathtt{donoDe}$$

afirmam, respectivamente, que se a é dono de b então a cuida de b (propriedade implica em cuidado) e que se a é dono de b e b é parte de c então a é dono de c (propriedade da parte implica em propriedade do todo).[3] Um conjunto finito de RIAs é chamado de *hierarquia de papéis*.

[2]Podemos reescrever as duas últimas proposições em lógica de primeira ordem da seguinte forma: $\forall x(\mathtt{Diretor}(x) \rightarrow \mathtt{Pessoa}(x))$ e $\forall x \forall y(\mathtt{dirige}(x,y) \rightarrow \mathtt{gosta}(x,y))$. A notação de DL é inspirada na notação de teoria dos conjuntos e não faz uso de variáveis.

[3]Esses três RIAs podem ser formalizados em lógica de primeira ordem como $\forall x \forall y(\mathtt{paiDe}(x,y) \rightarrow \mathtt{filhoDe}(y,x))$, $\forall x \forall y(\mathtt{donoDe}(x,y) \rightarrow \mathtt{cuidaDe}(x,y))$, e $\forall x \forall y \forall z(\mathtt{donoDe}(x,y) \wedge \mathtt{parteDe}(y,z) \rightarrow \mathtt{donoDe}(x,z))$.

Uma *característica de papel* é uma proposição com uma das seguintes formas:

$$\mathsf{Sym}(r), \quad \mathsf{Asy}(r), \quad \mathsf{Tra}(r), \quad \mathsf{Ref}(r), \quad \mathsf{Irr}(r), \quad \mathsf{Dis}(r,s),$$

em que r e s são quaisquer papéis diferentes do papel universal u. Conforme veremos na Seção 1.2.2, papéis representam relações binárias e características de papéis afirmam que essas relações possuem determinadas propriedades—a saber, simetria (Sym), assimetria (Asy), transitividade (Tra), reflexividade (Ref), irreflexividade (Irr), e disjunção (Dis). Vamos definir o significado exato dessas expressões na próxima seção.

Uma *RBox* de $\mathcal{SROIQ}$ consiste de uma hierarquia de papéis juntamente com um conjunto finito de características de papéis. Uma RBox é dita *regular* se a sua hierarquia de papéis é regular.

Regularidade é uma propriedade desejável porque ela garante a decidibilidade da lógica resultante. Na prática, isso significa que qualquer tarefa de raciocínio terminará em tempo finito (não entrará num laço infinito). Não vamos definir o que exatamente é uma RBox regular, já que essa definição nos levaria para fora do escopo desse capítulo, mas ressaltamos que a regularidade é uma propriedade puramente sintática: ela é obtida restringindo-se a forma dos RIAs que ocorrem numa dada hierarquia de papéis. Veja [Horrocks et al. 2006] para mais detalhes.

TBox A TBox de $\mathcal{SROIQ}$ contém proposições que relacionam conceitos, ou mais precisamente, expressões de conceitos. Uma *expressão de conceito* é uma das seguintes expressões:

1. Um nome de conceito $\mathtt{C}$, para algum $\mathtt{C}$ em N_C.
2. O conceito *top* $\top$.
3. O conceito *bottom* $\bot$.
4. Um *nominal* $\{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ em que $\mathtt{a}_1, \ldots, \mathtt{a}_n$ são nomes de indivíduos em N_I.
5. Uma *negação* $\neg C$ em que C é uma expressão de conceito.
6. Uma *interseção* $C \sqcap D$ em que C e D são expressões de conceitos.
7. Uma *união* $C \sqcup D$ em que C e D são expressões de conceitos.
8. Um *existencial* $\exists r.C$ em que r é um papel e C é uma expressão de conceito.
9. Um *universal* $\forall r.C$ em que r é um papel e C é uma expressão de conceito.
10. Uma *autorrestrição* $\exists r.\mathsf{Self}$ em que r é um papel (simples).
11. Uma *restrição de cardinalidade* $\geq nr.C$ em que n é um inteiro não-negativo, r é um papel (simples), e C é uma expressão de conceito.
12. Uma *restrição de cardinalidade* $\leq nr.C$ em que n é um inteiro não-negativo, r é um papel (simples), e C é uma expressão de conceito.

Observe que uma expressão de conceito, conforme definido acima, não é uma proposição—ela não faz uma afirmação que pode ser verdadeira ou falsa. Uma expressão de conceito *constrói* um novo conceito a partir de outras expressões, incluindo outras expressões de conceitos, nomes de indivíduos ou papéis. E essa construção sempre começa com conceitos atômicos, isto é, nomes de conceitos ou os conceitos *top* $\top$ e *bottom* $\bot$ (conforme as regras 1, 2 e 3 acima).

Lembre-se que um conceito representa uma determinada característica de certos indivíduos. Podemos entender um conceito C como a coleção (ou *conjunto*) de todos os indivíduos que possuem essa determinada característica. Dessa forma, o nome de conceito `Ator` pode ser entendido como o conjunto de todos os indivíduos que são atores. Similarmente, o conceito *top* $\top$ pode ser entendido como o conjunto de *todos* os indivíduos (sem qualificação) e o conceito *bottom* $\bot$ pode ser entendido como o conjunto vazio—o conjunto contendo nenhum indivíduo.

O conceito nominal $\{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ (regra 4) representa o conjunto contendo precisamente os indivíduos chamados $\mathtt{a}_1$, ..., $\mathtt{a}_n$. Por exemplo, $\{\mathtt{kubrick}, \mathtt{scorsese}\}$ representa o conjunto cujos membros são exatamente os indivíduos chamados `kubrick` e `scorsese`.

As operações de negação, interseção e união de conceitos (regras 5, 6 e 7) nos permitem construir novos conceitos a partir de outros conceitos. Por exemplo, o conceito $\neg\mathtt{Ator}$ representa o conjunto de todos os indivíduos que não são atores. De maneira similar, os conceitos

$$\mathtt{Ator} \sqcap \mathtt{Mulher} \quad \text{e} \quad \mathtt{Ator} \sqcup \mathtt{Diretor}$$

representam, respectivamente, o conjunto dos indivíduos que são ao mesmo tempo atores e mulheres, e o conjunto dos indivíduos que são atores ou diretores (ou ambos). Observe que essas operações podem ser aplicadas a quaisquer expressões de conceitos e não apenas a nomes de conceitos. Por exemplo, o conceito complexo

$$\mathtt{Diretor} \sqcap \neg(\mathtt{Mulher} \sqcup \{\mathtt{kubrick}, \mathtt{scorsese}\})$$

representa os diretores que não são nem mulheres nem os indivíduos chamados `kubrick` ou `scorsese`.

Os quantificadores existencial e universal (regras 8 e 9) recebem um papel e um conceito e constroem um novo conceito. Por exemplo, os conceitos

$$\exists\mathtt{conhece}.\mathtt{Ator} \quad \text{e} \quad \forall\mathtt{dirige}.\{\mathtt{a\text{-}bronx\text{-}tale}, \mathtt{good\text{-}shepherd}\}$$

representam, respectivamente, o conjunto dos indivíduos que conhecem *algum* ator e o conjunto dos indivíduos que ou não dirigiram nenhum filme ou dirigiram exatamente os filmes chamados `a-bronx-tale` e `good-shepherd`. Conforme veremos na próxima seção, pela maneira como o operador $\forall$ é definido, se quisermos excluir a parte "não dirigiram nenhum filme" do significado da expressão anterior precisamos escrever

$$(\exists\mathtt{dirige}.\top) \sqcap (\forall\mathtt{dirige}.\{\mathtt{a\text{-}bronx\text{-}tale}, \mathtt{good\text{-}shepherd}\}),$$

que, a propósito, especifica o conjunto $\{\mathtt{de\text{-}niro}\}$.

O operador de autorrestrição (regra 10) recebe um papel (simples) r e constrói o conceito contendo todos os indivíduos que estão relacionados a si mesmos via r. Por

exemplo, o conceito $\exists \texttt{ama.Self}$ representa o conjunto de todos os indivíduos que amam a si mesmos. Evidentemente, esse operador só faz sentido quando aplicado a papéis que conectam o mesmo tipo de entidade, por exemplo, pessoas à pessoas, filmes a filmes, etc.

O requisito de que r é um papel *simples* na definição do operador de autorrestrição está relacionado ao problema da regularidade mencionado anteriormente. Os papéis de uma RBox $\mathcal{SROIQ}$ são classificados em simples ou não-simples dependendo da forma dos RIAs em que eles ocorrem. Informalmente, os papéis não-simples são aqueles "criados" por cadeias de papéis de comprimento maior do que um; os papéis restantes são considerados simples. Vamos assumir que os papéis r e s que ocorrem em características de papéis da forma $\mathsf{Irr}(r)$, $\mathsf{Asy}(r)$ e $\mathsf{Dis}(r,s)$ são sempre papéis simples. Veja [Horrocks et al. 2006] para mais detalhes.

Os dois construtores de conceitos $\mathcal{SROIQ}$ restantes são os operadores de restrição de cardinalidade $\geq$ e $\leq$ (regras 11 e 12). Ambos recebem um inteiro não-negativo n, um papel simples r e um conceito C, e constroem um novo conceito contendo todos os indivíduos que estão r-conectados a pelo menos/no máximo n indivíduos em C. Por exemplo, os conceitos

$$\geq 2\texttt{conhece.Ator} \quad \text{e} \quad \leq 5\texttt{dirige.}\top$$

representam, respectivamente, o conjunto dos indivíduos que conhecem dois ou mais atores e o conjunto dos indivíduos que dirigiram cinco ou menos filmes.

Agora que definimos o formato das expressões de conceito de $\mathcal{SROIQ}$ podemos definir o que é uma TBox. Uma *TBox* de $\mathcal{SROIQ}$ é um conjunto finito de *axiomas gerais de inclusão de conceitos* (*general concept inclusion axioms*, ou GCIs) em que cada GCI é uma proposição da forma

$$C \sqsubseteq D,$$

para conceitos quaisquer C e D. Assim como nos RIAs, nos GCIs o símbolo $\sqsubseteq$ representa subsunção. Por exemplo, os GCIs

$$\texttt{Ator} \sqsubseteq \texttt{Pessoa} \quad \text{e} \quad \exists\texttt{dirige.}\top \sqsubseteq \exists\texttt{conhece.Ator}$$

afirmam, respectivamente, que todo indivíduo que é um ator é também uma pessoa e que todo indivíduo que dirige algum filme conhece algum ator.[4]

ABox A ABox de $\mathcal{SROIQ}$ contém as *proposições assertivas*, isto é, as proposições que tratam de indivíduos específicos. Essas proposições possuem uma das seguintes formas:

$$C(\texttt{a}), \quad r(\texttt{a},\texttt{b}), \quad \neg r(\texttt{a},\texttt{b}), \quad \texttt{a} \approx \texttt{b}, \quad \texttt{a} \not\approx \texttt{b},$$

em que C é um conceito, r é um papel, e $\texttt{a}$ e $\texttt{b}$ são nomes de indivíduos.

Seguem alguns exemplos de proposições assertivas:

[4]Para traduzirmos um GCI para lógica de primeira ordem primeiro transformamos os conceitos que ocorrem à esquerda e à direta do símbolo $\sqsubseteq$ em fórmulas com uma única variável livre x. Em seguida, conectamos essas duas fórmulas através de uma implicação ($\rightarrow$). Por último, prefixamos à fórmula resultante o quantificador $\forall x$. Os dois últimos GCIs podem ser traduzidos como:

$$\forall x(\texttt{Ator}(x) \rightarrow \texttt{Pessoa}(x)) \quad \text{e} \quad \forall x(\exists y(\texttt{dirige}(x,y) \wedge \top(y)) \rightarrow \exists z(\texttt{conhece}(x,z) \wedge \texttt{Ator}(z))).$$

- A *assertiva de conceito* $\texttt{Ator} \sqcap \texttt{Diretor}(\texttt{de-niro})$ afirma que o indivíduo chamado `de-niro` é uma *instância* do conceito chamado $\texttt{Ator} \sqcap \texttt{Diretor}$, isto é, que esse indivíduo é ao mesmo tempo ator e diretor.
- A *assertiva de papel* $\texttt{dirige}(\texttt{kubrick}, \texttt{dr-strangelove})$ afirma que o indivíduo chamado `kubrick` está `dirige`-conectado ao indivíduo chamado `dr-strangelove`. De maneira similar, a *assertiva de papel negada* $\neg\, \texttt{dirige}(\texttt{kubrick}, \texttt{taxi-driver})$ afirma que `kubrick` não está `dirige`-conectado a `taxi-driver`.
- A *assertiva de igualdade* $\texttt{kubrick} \approx \texttt{stanley}$ e de *desigualdade* $\texttt{kubrick} \not\approx \texttt{taxi-driver}$ afirmam, respectivamente, que `kubrick` e `stanley` designam o mesmo indivíduo e que `kubrick` e `taxi-driver` não designam o mesmo indivíduo.

Base de conhecimento Uma *base de conhecimento* (*knowledge base*, ou KB) $\mathcal{SROIQ}$ consiste da união das três caixas: RBox, TBox e ABox. Uma base de conhecimento é dita regular se sua RBox é regular.

Vamos concluir esta seção sobre a sintaxe de $\mathcal{SROIQ}$ com um exemplo. A base de conhecimento abaixo contém alguns dos *fatos sobre filmes* discutidos anteriormente. Observe que a interpretação usual das proposições que ocorrem nessa base de conhecimento—aquela em que `kubrick` designa o famoso diretor—é apenas uma das muitas outras possíveis interpretações. Trataremos da noção de interpretação na próxima seção.

ABox	
$\texttt{Diretor}(\texttt{kubrick})$	"`kubrick` é um a diretor"
$\texttt{Ator} \sqcap \texttt{Diretor}(\texttt{de-niro})$	"`de-niro` é ao mesmo tempo ator e diretor"
$\geq 2\texttt{conhece}.\texttt{Ator}(\texttt{stanley})$	"`stanley` conhece ao menos dois atores"
$\texttt{dirige}(\texttt{kubrick}, \texttt{space-odyssey})$	"`kubrick` dirigiu `space-odyssey`"
$\neg\, \texttt{dirige}(\texttt{kubrick}, \texttt{taxi-driver})$	"`kubrick` não dirigiu `taxi-driver`"
$\texttt{kubrick} \approx \texttt{stanley}$	"`kubrick` e `stanley` são o mesmo indivíduo"
$\texttt{kubrick} \not\approx \texttt{de-niro}$	"`kubrick` e `de-niro` não são o mesmo indivíduo"
TBox	
$\texttt{Diretor} \sqsubseteq \exists \texttt{dirige}.\top$	"todo diretor dirige algo"
$\exists \texttt{dirige}.\top \sqsubseteq \texttt{Diretor}$	"tudo que dirige algo é um diretor"
$\texttt{Diretor} \sqsubseteq \texttt{Pessoa}$	"todo diretor é uma pessoa"
$\texttt{Diretor} \sqsubseteq \geq 1\texttt{conhece}.\texttt{Ator}$	"todo diretor conhece ao menos um ator"
$(\exists \texttt{dirige}.\top) \sqcap (\forall \texttt{dirige}.\{\texttt{a-bronx-tale}, \texttt{good-shepherd}\}) \sqsubseteq \{\texttt{de-niro}\}$	"tudo que dirige exatamente $\{\texttt{a-bronx-tale}, \texttt{good-shepherd}\}$ é um $\{\texttt{de-niro}\}$"
RBox	
$\texttt{dirige} \sqsubseteq \texttt{gosta}$	"dirigir implica em gostar"
$\texttt{dirige} \circ \texttt{atuaEm}^{-} \sqsubseteq \texttt{conhece}$	"se x dirige y em que z atua, então x conhece z"

1.2.2. Semântica

Uma base de conhecimento, conforme definimos anteriormente, é basicamente um conjunto de *strings* bem-formadas as quais chamamos de proposições ou axiomas. O significado

de cada uma dessas *strings* depende, em última instância, do significado dos nomes que ocorrem em cada uma delas. Para ver isso, considere o axioma $\texttt{Diretor}(\texttt{kubrick})$. Esse axioma será verdadeiro se interpretarmos os nomes `Diretor` e `kubrick` como "a propriedade de ser um diretor de cinema" e "Stanley Kubrick", respectivamente. Porém, se interpretarmos `Diretor` como "a propriedade de ser um número par" e `kubrick` como "o número 37", então a proposição $\texttt{Diretor}(\texttt{kubrick})$ deixa de ser verdadeira. Esse exemplo é artificial mas serve para ilustrar o nosso ponto de que o significado de uma proposição deriva da interpretação dos nomes. Vamos agora definir precisamente a noção de interpretação.

Interpretação Uma *interpretação* $\mathcal{I}$ consiste de duas coisas:

1. Um conjunto não-vazio $\Delta^{\mathcal{I}}$ chamado de *domínio* ou *universo* de discurso.
2. Uma função $\square^{\mathcal{I}}$ partindo dos conjuntos de vocabulário, N_I, N_C e N_R, tal que:[5]
   (a) Se $\mathtt{a} \in \mathsf{N}_I$ então $\mathtt{a}^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.
   (b) Se $\mathtt{C} \in \mathsf{N}_C$ então $\mathtt{C}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.
   (c) Se $\mathtt{r} \in N_R$ então $\mathtt{r}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Em outras palavras, uma interpretação $\mathcal{I}$ fixa um domínio de discurso $\Delta^{\mathcal{I}}$, que pode ser qualquer conjunto não-vazio, e mapeia os nomes de invidados em elementos do domínio, os nomes de conceitos em subconjuntos do domínio, e os nomes de papéis em relações binárias no domínio.

Vamos reservar o termo *indivíduo* para nos referirmos à contrapartida semântica de um nome de indivíduo, isto é, a um elemento do domínio. E vamos reservar os termos *extensão de conceito* e *extensão de papel* para nos referirmos, respectivamente, às contrapartidas semânticas de conceitos e papéis, isto é, às relações unárias e binárias no domínio.

Vamos agora estender a definição da função interpretação $\square^{\mathcal{I}}$ a papéis e conceitos:

$$u^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \tag{1}$$
$$(r^{-})^{\mathcal{I}} = \{\langle \delta', \delta \rangle \mid \langle \delta, \delta' \rangle \in r^{\mathcal{I}}\} \tag{2}$$
$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \tag{3}$$
$$\bot^{\mathcal{I}} = \varnothing \tag{4}$$
$$\{\mathtt{a}_1, \ldots, \mathtt{a}_n\}^{\mathcal{I}} = \{\mathtt{a}_1^{\mathcal{I}}, \ldots, \mathtt{a}_n^{\mathcal{I}}\} \tag{5}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C \tag{6}$$
$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \tag{7}$$
$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}} \tag{8}$$
$$(\exists r.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \text{ e } \delta' \in C^{\mathcal{I}}, \text{ para algum } \delta' \in \Delta^{\mathcal{I}}\} \tag{9}$$

[5] Observe que $x^{\mathcal{I}}$ é apenas uma forma mais compacta de escrever $\mathcal{I}(x)$ que é a notação usual de aplicação de função.

$$(\forall r.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \text{se } \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \text{ então } \delta' \in C^{\mathcal{I}}, \text{ para todo } \delta' \in \Delta^{\mathcal{I}}\} \quad (10)$$

$$(\exists r.\mathsf{Self})^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta \rangle \in r^{\mathcal{I}}\} \quad (11)$$

$$(\geq n r.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \#\{\delta' \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \text{ e } \delta' \in C\} \geq n\} \quad (12)$$

$$(\leq n r.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \#\{\delta' \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta' \rangle \in r^{\mathcal{I}} \text{ e } \delta' \in C\} \leq n\} \quad (13)$$

Essas equações podem ser lidas da seguinte forma:

1. O *papel universal* u representa (denota) a relação que conecta quaisquer dois indivíduos do domínio.

2. O *papel invertido* r^{-} denota a mesma relação que r porém com a primeira e segunda coordenadas trocadas.

3. O *conceito top* $\top$ denota o domínio como um todo (conjunto $\Delta^{\mathcal{I}}$).

4. O *conceito bottom* $\bot$ denota o conjunto vazio ($\varnothing$).

5. O *nominal* $\{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$ denota o conjunto contendo exatamente os indivíduos denotados pelos nomes $\mathtt{a}_1$, ..., $\mathtt{a}_n$.

6. A *negação* $\neg C$ denota a diferença entre o domínio e o conjunto denotado por C, isto é, o conjunto de todos os indivíduos em $\Delta^{\mathcal{I}}$ que não estão em $C^{\mathcal{I}}$.

7. A *interseção* $C \sqcap D$ denota a interseção dos conjuntos denotados por C e D, isto é, o conjunto de todos os indivíduos que pertencem a ambos $C^{\mathcal{I}}$ e $D^{\mathcal{I}}$.

8. A *união* $C \sqcup D$ denota a união dos conjuntos denotados por C e D, isto é, o conjunto de todos os indivíduos que pertencem a ao menos um dos conjuntos $C^{\mathcal{I}}$ ou $D^{\mathcal{I}}$.

9. O *existencial* $\exists r.C$ denota o conjunto de todos os indivíduos que estão conectados via a relação $r^{\mathcal{I}}$ a algum indivíduo em $C^{\mathcal{I}}$.

10. O *universal* $\forall r.C$ denota o conjunto de todos os indivíduos a tais que se a está conectado a algum indivíduo b via $r^{\mathcal{I}}$ então b está em $C^{\mathcal{I}}$. Uma consequência do formato condicional dessa definição é que qualquer indivíduo que não esteja conectado a nenhum outro via $r^{\mathcal{I}}$ estará automaticamente no conjunto resultante (pois a implicação é vacuamente verdadeira).[6]

11. A *autorrestrição* $\exists r.\mathsf{Self}$ denota o conjunto de todos os indivíduos que estão conectados a si mesmos via a relação $r^{\mathcal{I}}$.

12. A *restrição de cardinalidade* $\geq n.rC$ denota o conjunto de todos os indivíduos que estão conectados via a relação $r^{\mathcal{I}}$ a pelo menos n indivíduos em $C^{\mathcal{I}}$.

13. A *restrição de cardinalidade* $\leq n.rC$ denota o conjunto de todos os indivíduos que estão conectados via relação $r^{\mathcal{I}}$ a no máximo n indivíduos em $C^{\mathcal{I}}$.

Vamos agora tratar do problema de definir o valor-verdade (verdadeiro ou falso) de um axioma sob uma dada interpretação.

Satisfatibilidade Uma interpretação $\mathcal{I}$ *satisfaz* (ou *é modelo de*) um axioma α, em símbolos $\mathcal{I} \vDash \alpha$, sob as seguintes condições:

1. (RIA) $\mathcal{I} \vDash r_1 \circ \cdots \circ r_n \sqsubseteq r$ sse[7] para qualquer sequência δ_0, δ_1, ..., δ_n em $\Delta^{\mathcal{I}}$:

   $$\langle \delta_0, \delta_1 \rangle \in r_1^{\mathcal{I}},\ \langle \delta_1, \delta_2 \rangle \in r_2^{\mathcal{I}},\ \ldots,\ \text{e}\ \langle \delta_{n-1}, \delta_n \rangle \in r_n^{\mathcal{I}} \quad \text{implica em} \quad \langle \delta_0, \delta_n \rangle \in r^{\mathcal{I}},$$

   ou seja, sse para qualquer caminho em $\Delta^{\mathcal{I}}$ que atravessa $r_1^{\mathcal{I}}$, ..., $r_n^{\mathcal{I}}$ (nessa ordem) há um atalho (aresta) via $r^{\mathcal{I}}$.

2. (Simetria de papel) $\mathcal{I} \vDash \mathsf{Sym}(r)$ sse $r^{\mathcal{I}}$ é uma relação simétrica, isto é, sse $\langle \delta_1, \delta_2 \rangle \in r^{\mathcal{I}}$ implica em $\langle \delta_2, \delta_1 \rangle \in r^{\mathcal{I}}$ e vice-versa, para quaisquer δ_1, δ_2 em $\Delta^{\mathcal{I}}$.

3. (Assimetria de papel) $\mathcal{I} \vDash \mathsf{Asy}(r)$ sse $r^{\mathcal{I}}$ é uma relação assimétrica, isto é, sse $\langle \delta_1, \delta_2 \rangle \in r^{\mathcal{I}}$ implica em $\langle \delta_2, \delta_1 \rangle \notin r^{\mathcal{I}}$, para quaisquer δ_1, δ_2 em $\Delta^{\mathcal{I}}$.

4. (Transitividade de papel) $\mathcal{I} \vDash \mathsf{Tra}(r)$ sse $r^{\mathcal{I}}$ é uma relação transitiva, isto é, sse $\langle \delta_1, \delta_2 \rangle \in r^{\mathcal{I}}$ e $\langle \delta_2, \delta_3 \rangle \in r^{\mathcal{I}}$ implicam em $\langle \delta_1, \delta_3 \rangle \in r^{\mathcal{I}}$, para quaisquer $\delta_1, \delta_2, \delta_3$ em $\Delta^{\mathcal{I}}$.

5. (Reflexividade de papel) $\mathcal{I} \vDash \mathsf{Ref}(r)$ sse $r^{\mathcal{I}}$ é uma relação reflexiva, isto é, sse $\langle \delta, \delta \rangle \in r^{\mathcal{I}}$, para qualquer δ em $\Delta^{\mathcal{I}}$.

6. (Irreflexividade de papel) $\mathcal{I} \vDash \mathsf{Irr}(r)$ sse $r^{\mathcal{I}}$ é uma relação irreflexiva, isto é, sse $\langle \delta, \delta \rangle \notin r^{\mathcal{I}}$, qualquer δ em $\Delta^{\mathcal{I}}$.

7. (Disjunção de papéis) $\mathcal{I} \vDash \mathsf{Dis}(r, s)$ sse $r^{\mathcal{I}} \cap s^{\mathcal{I}} = \varnothing$, isto é, sse não existem dois indivíduos ao mesmo tempo $r^{\mathcal{I}}$-conectados e $s^{\mathcal{I}}$-conectados.

8. (GCI) $\mathcal{I} \vDash C \sqsubseteq D$ sse $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, isto é, sse qualquer indivíduo de $C^{\mathcal{I}}$ é também um indivíduo de $D^{\mathcal{I}}$.

9. (Assertiva de conceito) $\mathcal{I} \vDash C(\mathtt{a})$ sse $\mathtt{a}^{\mathcal{I}} \in C^{\mathcal{I}}$, isto é, se o indivíduo denotado por $\mathtt{a}$ pertence à extensão do conceito C.

10. (Assertiva de papel) $\mathcal{I} \vDash r(\mathtt{a}, \mathtt{b})$ sse $\langle \mathtt{a}^{\mathcal{I}}, \mathtt{b}^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$, isto é, sse $\mathtt{a}^{\mathcal{I}}$ está conectado a $\mathtt{b}^{\mathcal{I}}$ via $r^{\mathcal{I}}$.

11. (Assertiva de papel negada) $\mathcal{I} \vDash \neg r(\mathtt{a}, \mathtt{b})$ sse $\langle \mathtt{a}^{\mathcal{I}}, \mathtt{b}^{\mathcal{I}} \rangle \notin r^{\mathcal{I}}$.

12. (Assertiva de igualdade) $\mathcal{I} \vDash \mathtt{a} \approx \mathtt{b}$ sse $\mathtt{a}^{\mathcal{I}} = \mathtt{b}^{\mathcal{I}}$, isto é, sse $\mathtt{a}^{\mathcal{I}}$ e $\mathtt{b}^{\mathcal{I}}$ são o mesmo elemento de $\Delta^{\mathcal{I}}$.

[6] Em lógica clássica a proposição $A \rightarrow B$ é equivalente à proposição $(\neg A) \vee B$. Dessa forma, podemos reescrever a equação (10) da seguinte maneira:

$$(\forall r.C)^{\mathcal{I}} = \{\delta \in \Delta^{\mathcal{I}} \mid \langle \delta, \delta' \rangle \notin r^{\mathcal{I}} \text{ ou } \delta' \in C^{\mathcal{I}}, \text{ para todo } \delta' \in \Delta^{\mathcal{I}}\}.$$

Evidentemente, qualquer δ que não esteja r-conectado a algum δ' satisfaz a condição acima.

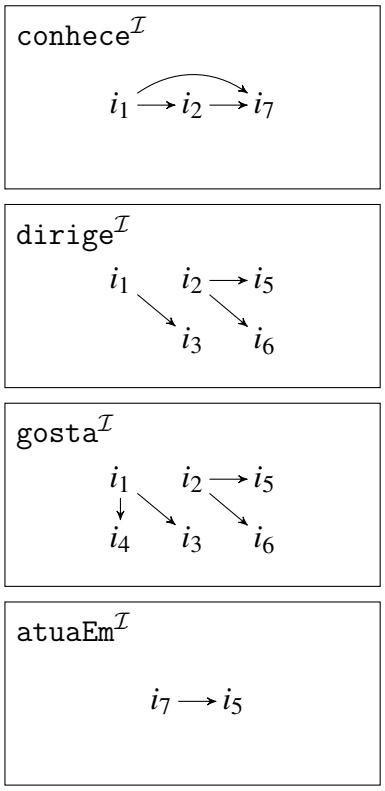
[7] Vamos usar "sse" como uma abreviação para "se e somente se".

13. (Assertiva de desigualdade) $\mathcal{I} \vDash \texttt{a} \not\approx \texttt{b}$ sse $\texttt{a}^{\mathcal{I}} \neq \texttt{b}^{\mathcal{I}}$.

Agora que definimos as condições necessárias para que uma interpretação satisfaça um dado axioma, podemos estender a noção de satisfatibilidade às bases de conhecimento (conjuntos de axiomas). Uma interpretação $\mathcal{I}$ *satisfaz* (ou *é modelo de*) uma base de conhecimento $\mathcal{KB}$, em símbolos $\mathcal{I} \vDash \mathcal{KB}$, sse $\mathcal{I} \vDash \alpha$, para qualquer axioma α em $\mathcal{KB}$. Uma base de conhecimento é dita *satisfatível* ou *consistente* se ela possui um modelo, ou seja, se existe alguma interpretação $\mathcal{I}$ que a satisfaz; caso contrário, a base de conhecimento é dita *insatisfatível* ou *inconsistente*.

Para tornar a discussão de interpretações mais concreta, considere a seguinte interpretação $\mathcal{I}$ para a KB de fatos sobre filmes apresentada no fim da seção anterior:

$$
\begin{aligned}
\Delta^{\mathcal{I}} &= \{i_1, i_2, i_3, i_4, i_5, i_6, i_7\} \\
\texttt{kubrick}^{\mathcal{I}} &= i_1 \\
\texttt{de-niro}^{\mathcal{I}} &= i_2 \\
\texttt{stanley}^{\mathcal{I}} &= i_1 \\
\texttt{space-odyssey}^{\mathcal{I}} &= i_3 \\
\texttt{taxi-driver}^{\mathcal{I}} &= i_4 \\
\texttt{a-bronx-tale}^{\mathcal{I}} &= i_5 \\
\texttt{good-shepherd}^{\mathcal{I}} &= i_6 \\
\texttt{Diretor}^{\mathcal{I}} &= \{i_1, i_2\} \\
\texttt{Ator}^{\mathcal{I}} &= \{i_2, i_7\} \\
\texttt{Pessoa}^{\mathcal{I}} &= \{i_1, i_2\} \\
\texttt{conhece}^{\mathcal{I}} &= \{\langle i_1, i_2\rangle, \langle i_1, i_7\rangle, \langle i_2, i_7\rangle\} \\
\texttt{dirige}^{\mathcal{I}} &= \{\langle i_1, i_3\rangle, \langle i_2, i_5\rangle, \langle i_2, i_6\rangle\} \\
\texttt{gosta}^{\mathcal{I}} &= \{\langle i_1, i_3\rangle, \langle i_1, i_4\rangle, \langle i_2, i_5\rangle, \langle i_2, i_6\rangle\} \\
\texttt{atuaEm}^{\mathcal{I}} &= \{\langle i_7, i_5\rangle\}
\end{aligned}
$$



O domínio $\Delta^{\mathcal{I}}$ nesse caso consiste de sete indivíduos distintos, $i_1, \ldots, i_7$, e os nomes que ocorrem na KB de fatos sobre filmes são mapeados conforme as equações acima. Observe que:

- Alguns indivíduos do domínio $\Delta^{\mathcal{I}}$ podem não possuir nome correspondente: i_7 não é denotado por nenhum nome da KB de fatos sobre filmes.
- Dois nomes distintos podem designar o mesma objeto: `kubrick` e `stanley` denotam ambos o indivíduo i_1.
- Nomes de conceitos são mapeados em subconjuntos do domínio $\Delta^{\mathcal{I}}$: `Diretor`, `Ator` e `Pessoa` denotam conjuntos de indivíduos.

- Nomes de papéis são mapeados em relações binárias no domínio $\Delta^{\mathcal{I}}$: `conhece`, `dirige`, `gosta` e `atuaEm` denotam conjuntos de pares ordenados de indivíduos. Os indivíduos i_2 e i_5 estão relacionados via a relação $\texttt{dirige}^{\mathcal{I}}$ (nessa ordem) sse o par $\langle i_2, i_5 \rangle$ pertence ao conjunto $\texttt{dirige}^{\mathcal{I}}$. Relações binárias podem ser representadas convenientemente como grafos dirigidos. Os grafos correspondentes às relações $\texttt{conhece}^{\mathcal{I}}$, $\texttt{dirige}^{\mathcal{I}}$, $\texttt{gosta}^{\mathcal{I}}$ e $\texttt{atuaEm}^{\mathcal{I}}$ são apresentados nas caixas acima.

Seja $\mathcal{KB}$ a base de conhecimento de fatos sobre filmes e seja $\mathcal{I}$ a interpretação anterior. A pergunta natural a se fazer é se $\mathcal{I} \vDash \mathcal{KB}$, isto é, se a interpretação anterior satisfaz a KB. A resposta a essa pergunta será positiva apenas se $\mathcal{I} \vDash \alpha$ para todo axioma a em $\mathcal{KB}$. Vamos verificar se essa última afirmação procede.

Começamos verificando a ABox de $\mathcal{KB}$. Evidentemente, $\mathcal{I} \vDash \texttt{Diretor}(\texttt{kubrick})$, pois $\texttt{kubrick}^{\mathcal{I}} \in \texttt{Diretor}^{\mathcal{I}}$, isto é, pois $i_1 \in \{i_1, i_2\}$, conforme demandado pela definição de satisfatibilidade para assertivas de conceito. Além disso, temos que $\mathcal{I} \vDash \neg\texttt{dirige}(\texttt{kubrick}, \texttt{taxi-driver})$, pois $\langle \texttt{kubrick}^{\mathcal{I}}, \texttt{taxi-driver}^{\mathcal{I}} \rangle \notin \texttt{dirige}^{\mathcal{I}}$. Os outros axiomas da ABox de $\mathcal{KB}$ também são satisfeitos por $\mathcal{I}$, conforme o leitor pode facilmente verificar.

Vamos agora verificar a TBox de $\mathcal{KB}$. Temos que $\mathcal{I} \vDash \texttt{Diretor} \sqsubseteq \texttt{Pessoa}$, pois todo indivíduo do conjunto $\texttt{Diretor}^{\mathcal{I}}$ é também um indivíduo do conjunto $\texttt{Pessoa}^{\mathcal{I}}$. É também o caso que

$$\mathcal{I} \vDash (\exists \texttt{dirige}.\top) \sqcap (\forall \texttt{dirige}.\{\texttt{a-bronx-tale}, \texttt{good-shepherd}\}) \sqsubseteq \{\texttt{de-niro}\}$$

já que (1) $\texttt{de-niro}^{\mathcal{I}}$ está conectado via a relação $\texttt{dirige}^{\mathcal{I}}$ a algum membro de $\top^{\mathcal{I}}$ (isto é, a algum membro do domínio como um todo), (2) tudo a que $\texttt{de-niro}^{\mathcal{I}}$ está $\texttt{dirige}^{\mathcal{I}}$-conectado é um elemento do conjunto $\{\texttt{a-bronx-tale}^{\mathcal{I}}, \texttt{good-shepherd}^{\mathcal{I}}\}$ e (3) as duas últimas afirmações valem apenas para o indivíduo $\texttt{de-niro}^{\mathcal{I}}$. O restante dos axiomas da TBox de $\mathcal{KB}$ são também satisfeitos por $\mathcal{I}$. (Novamente, deixamos a verificação desses axiomas para o leitor.)

A última coisa que precisamos verificar são os axiomas da RBox de $\mathcal{KB}$. Temos que $\mathcal{I} \vDash \texttt{dirige} \sqsubseteq \texttt{gosta}$, pois todo par da relação $\texttt{dirige}^{\mathcal{I}}$ é também um par da relação $\texttt{gosta}^{\mathcal{I}}$. Uma forma de determinar se a proposição

$$\mathcal{I} \vDash \texttt{dirige} \circ \texttt{atuaEm}^{-} \sqsubseteq \texttt{conhece}$$

vale, é procurar por três indivíduos x, y e z, não necessariamente distintos, tais que $\langle x, y \rangle \in \texttt{dirige}^{\mathcal{I}}$ e $\langle z, y \rangle \in \texttt{atuaEm}^{\mathcal{I}}$ mas $\langle x, z \rangle \notin \texttt{conhece}^{\mathcal{I}}$. Se pudermos encontrar tais indivíduos então a proposição será falsa (pois acabamos de encontrar um contraexemplo). Caso não seja possível encontrar tais indivíduos, a proposição será verdadeira. No caso específico da interpretação $\mathcal{I}$, há apenas uma escolha possível que satisfaz os dois primeiros requisitos, a saber, $x = i_2$, $y = i_5$ e $z = i_7$. Observe que essa escolha também satisfaz o terceiro requisito, já que $\langle i_2, i_7 \rangle \in \texttt{conhece}^{\mathcal{I}}$. Logo, a proposição é verdadeira, isto é, o axioma $\texttt{dirige} \circ \texttt{atuaEm}^{-} \sqsubseteq \texttt{conhece}$ é satisfeito pela interpretação $\mathcal{I}$.

Como $\mathcal{I}$ satisfaz todos os axiomas em cada uma das caixas da KB de fatos sobre filmes, concluímos que a resposta à pergunta original (se $\mathcal{I} \vDash \mathcal{KB}$) é positiva. Ou seja,

a interpretação $\mathcal{I}$ satisfaz a KB de fatos sobre filmes e, consequentemente, essa KB é consistente (satisfatível).

Duas perguntas relacionadas à essa mesma KB são as seguintes: É possível encontrar uma interpretação $\mathcal{J}$ que não satisfaz a KB de fatos sobre filmes? Como podemos modificar essa KB de forma que ela se torne inconsistente (insatisfatível)?

Uma resposta simples à primeira pergunta consiste em considerar uma interpretação $\mathcal{J}$ idêntica à interpretação $\mathcal{I}$ anterior porém em que $\texttt{kubrick}^{\mathcal{J}}$ e $\texttt{stanley}^{\mathcal{J}}$ são mapeados em indivíduos distintos do domínio Δ^{J}. Evidentemente, sob essa interpretação o axioma de ABox $\texttt{kubrick} \approx \texttt{stanley}$ é falso; logo $\mathcal{J} \not\models \mathcal{KB}$.

Com respeito à segunda pergunta, para tornar a KB de fatos sobre filmes inconsistente precisamos modificá-la de forma que seja impossível construir qualquer interpretação que a satisfaça. Uma maneira trivial de fazer isso é adicionar à KB a assertiva de desigualdade $\texttt{kubrick} \not\approx \texttt{kubrick}$. Esse axioma será falso sob qualquer interpretação pois qualquer que seja a escolha de denotação para kubrick ela sempre será igual a si mesma, isto é, $\texttt{kubrick}^{\mathcal{I}} = \texttt{kubrick}^{\mathcal{I}}$. Evidentemente, existem formas menos triviais de obtermos uma KB inconsistente. (Será que o leitor consegue achar uma que envolva um axioma de TBox?)

Isso conclui a discussão sobre satisfatibilidade. O objetivo principal de uma semântica formal é definir uma *relação de consequência* que permita determinar quando um determinado axioma decorre de uma determinada base de conhecimento. Vamos agora definir essa relação de consequência em termos da relação de satisfatibilidade.

Consequência lógica Um axioma α é *consequência lógica* de (ou *é implicado logicamente por*) uma base de conhecimento $\mathcal{KB}$, em símbolos $\mathcal{KB} \models \alpha$, sse $\mathcal{I} \models \mathcal{KB}$ implica em $\mathcal{I} \models \alpha$, para qualquer interpretação $\mathcal{I}$. Isto é, sse qualquer interpretação que é modelo para a base de conhecimento $\mathcal{KB}$ é também modelo para o axioma α.[8]

Usando essa definição de consequência lógica podemos determinar precisamente quando um α decorre de um determinado conjunto de axiomas. Conforme discutimos anteriormente, algumas KBs são inconsistentes no sentido de que não existe interpretação que satisfaça todos os seus axiomas. O problema de se ter uma base de conhecimento inconsistente é que, por definição, ela implica em qualquer coisa. Isto é, se $\mathcal{KB}$ é inconsistente então a afirmação "$\mathcal{I} \models \mathcal{KB}$ implica em $\mathcal{I} \models \alpha$ para qualquer $\mathcal{I}$" é vacuamente verdadeira pois o antecedente da implicação, a saber, $\mathcal{I} \models \mathcal{KB}$, é falso. Na prática, a inconsistência de uma KB é um indicativo de erros sérios de modelagem.

Uma tarefa importante na teoria de DL é, portanto, determinar se uma determinada KB é consistente. Vamos discutir essa e outras ditas *tarefas de raciocínio* em detalhe na próxima seção. Antes disso, porém, vamos concluir essa seção com um exemplo de consequência lógica.

[8]Observe que o símbolo $\models$ possui significados diferentes dependendo do tipo de objeto que ocorre à sua esquerda. Quando esse objeto é uma interpretação, como em $\mathcal{I} \models \alpha$, o símbolo $\models$ denota a relação de satisfatibilidade. Porém, quando esse objeto é um conjunto de axiomas, como em $\mathcal{KB} \models \alpha$, o símbolo $\models$ denota a relação de consequência lógica.

Considere a seguinte proposição sobre a $\mathcal{KB}$ de fatos sobre filmes:

$$\mathcal{KB} \models \texttt{Ator} \sqsubseteq \texttt{Pessoa}.$$

Essa proposição afirma que o axioma $\texttt{Ator} \sqsubseteq \texttt{Pessoa}$ é consequência lógica do conjunto de axiomas que compõem a $\mathcal{KB}$. Essa afirmação é verdadeira? Como podemos prová-la ou refutá-la?

Vamos tentar refutá-la. O que precisamos fazer nesse caso é encontrar uma interpretação $\mathcal{I}$ que satisfaça todos os axiomas em $\mathcal{KB}$ mas que não satisfaça $\texttt{Ator} \sqsubseteq \texttt{Pessoa}$. Por exemplo, considere a interpretação $\mathcal{I}$ para $\mathcal{KB}$ discutida anteriormente, cujo domínio consiste dos indivíduos $i_1, \ldots, i_7$. Conforme verificamos há alguns parágrafos, $\mathcal{I} \models \mathcal{KB}$. Porém, não é o caso que todo ator é uma pessoa sob essa interpretação $\mathcal{I}$. Por exemplo, $i_7 \in \texttt{Ator}^{\mathcal{I}}$ mas $i_7 \notin \texttt{Pessoa}^{\mathcal{I}}$. Logo, $\mathcal{KB} \not\models \texttt{Ator} \sqsubseteq \texttt{Pessoa}$, ou seja, a $\mathcal{KB}$ de fatos sobre filmes não implica logicamente no axioma $\texttt{Ator} \sqsubseteq \texttt{Pessoa}$.

Normalmente é mais difícil provar o contrário, isto é, provar que a base de conhecimento implica logicamente em um determinado axioma α. Provar isso significa mostrar que é impossível encontrar uma interpretação que ao mesmo tempo satisfaz a base de conhecimento e falsifica α. Por exemplo, vamos mostrar que a $\mathcal{KB}$ de fatos sobre filmes implica logicamente no axioma

$$\exists \texttt{gosta}.\top(\texttt{de-niro}).$$

Em outras palavras, vamos mostrar que se assumirmos que os axiomas da $\mathcal{KB}$ são verdadeiros, somos obrigados a concluir que de-niro gosta de algo.

Suponha que exista uma interpretação $\mathcal{J}$ que satisfaça a $\mathcal{KB}$ de fato sobre filmes, mas que não satisfaça o axioma acima. Da hipótese de que $\mathcal{J} \models \mathcal{KB}$ podemos inferir que

$$\mathcal{J} \models \texttt{Ator} \sqcap \texttt{Diretor}(\texttt{de-niro}) \tag{14}$$

$$\mathcal{J} \models \texttt{Diretor} \sqsubseteq \exists \texttt{dirige}.\top \tag{15}$$

$$\mathcal{J} \models \texttt{diretor} \sqsubseteq \texttt{gosta} \tag{16}$$

Como (14), temos que o indivíduo denotado por de-niro está no conjunto $\texttt{Diretor}^{\mathcal{J}}$. Logo, como (15), esse indivíduo está $\texttt{dirige}^{\mathcal{J}}$-conectado a alguém, isto é, $\langle \texttt{de-niro}^{\mathcal{J}}, a \rangle \in \texttt{dirige}^{\mathcal{J}}$, para algum indivíduo a. Finalmente, como (16), temos que $\langle \texttt{de-niro}, a \rangle \in \texttt{gosta}^{\mathcal{J}}$, o que contradiz a nossa hipótese original de que $\mathcal{J} \not\models \exists \texttt{gosta}.\top(\texttt{de-niro})$. Portanto, somos forçados a concluir que tal $\mathcal{J}$ não pode existir. Ou seja, qualquer interpretação que satisfaça a $\mathcal{KB}$ de fatos sobre filmes também irá satisfazer $\exists \texttt{gosta}.\top(\texttt{de-niro})$, logo, $\mathcal{KB} \models \exists \texttt{gosta}.\top(\texttt{de-niro})$.

1.2.3. Tarefas de raciocínio

Uma das principais vantagens de se formalizar um corpo de conhecimento em lógica é a possibilidade de tratá-lo como um objeto tangível ao qual certas operações podem ser aplicadas. Em DL, esse objeto é a KB (conjunto de axiomas) e as operações são *tarefas de raciocínio* que visam extrair conhecimento novo a partir da KB.

Uma característica importante das tarefas de raciocínio de DL, que faz com que elas sejam ao mesmo tempo poderosas e complexas, é que essas tarefas adotam a *hipótese de*

mundo aberto (*open-world assumption*, ou OWA). Sob essa hipótese, fatos que não podem ser deduzidos a partir da KB são considerados *desconhecidos* mas não necessariamente falsos. Isso contrasta com a *hipótese de mundo fechado* (*closed-world assumption*, ou CWA), normalmente adotada em sistemas de bancos de dados, que assume que fatos que não estão no banco são falsos. Dito isso, quando necessário, é possível emular até certo ponto a CWA em DL através dos nominals. Há também DLs especializadas (autoepistêmicas, circumscriptivas) que dão suporte à CWA.

Vamos agora descrever rapidamente os principais tipos de tarefas de raciocínio de DL. Para uma descrição mais detalhada veja [Rudolph 2011, Baader et al. 2007]. Uma descrição abrangente da complexidade computacional dessas tarefas para DLs específicas pode ser encontrada em [Zolin 2019]. E uma lista de softwares que implementam algumas dessas tarefas pode ser encontrada em [OWL@Machester 2019]

Satisfatibilidade da base de conhecimento A principal tarefa de raciocínio de DL é a que determina satisfatibilidade da base de conhecimento. O objetivo dessa tarefa é decidir se uma dada KB é satisfatível, isto é, se existe uma interpretação que satisfaz todos os seus axiomas. Outras tarefas de raciocínio importantes, como a de consequência lógica, podem ser reduzidas à tarefa de satisfatibilidade.

Existem basicamente dois tipos de abordagens para verificar se uma KB é satisfatível. As abordagens *baseadas em teoria dos modelos* tentam construir um modelo para a KB ou então mostrar que tal construção é impossível. Já as abordagens *baseadas em teoria da prova* manipulam a KB sintaticamente até derivar uma contradição ou até atingir um ponto em que é garantido que uma contradição não pode ser derivada. Exemplos de métodos do primeiro tipo são os métodos baseados em autômatos de árvore e o método do tableaux. Como exemplos de métodos do segundo tipo podemos citar os métodos baseados em provas (derivações) e aqueles baseados em resolução.

Consequência lógica O objetivo da tarefa de consequência lógica é decidir se um dado axioma α é consequência lógica de uma dada KB. Esse problema se reduz diretamente ao problema de satisfatibilidade em DLs que suportam negação (como $\mathcal{SROIQ}$). Em DLs sem suporte à negação essa redução ainda é possível mas requer a simulação da negação através de alguma noção de oposição entre axiomas.

Satisfatibilidade e classificação de conceitos O objetivo da tarefa de satisfatibilidade (consistência) de conceito é decidir se um determinado conceito C é satisfatível com respeito a uma determinada base de conhecimento $\mathcal{KB}$. Isto é, se existe uma interpretação $\mathcal{I}$ que satisfaz $\mathcal{KB}$ e tal que $C^{\mathcal{I}}$ é populado (não-vazio). Observe que alguns conceitos, como $C \sqcap \neg C$, por definição não podem ser satisfeitos.

Uma tarefa relacionada é a classificação de conceitos. O objetivo dessa tarefa é decidir se os nomes de conceitos que ocorrem em uma KB podem ser organizados hierarquicamente de acordo com os seus relacionamentos de subsunção ($\sqsubseteq$). Essa organização hierárquica é normalmente um passo preliminar executados por outras tarefas de raciocínio e também ajuda na visualização de KBs contendo muitos GCIs.

Recuperação de instâncias designadas A tarefa de recuperação de instâncias designadas recebe uma base de conhecimento $\mathcal{KB}$ e um conceito C e retorna todos os indivíduos

designados a tais que $\mathcal{KB} \models C(\mathtt{a})$. Isto é, ela determina todos os nomes de indivíduos que são instâncias do conceito C em todos os modelos da $\mathcal{KB}$. Essa tarefa pode ser estendida a papéis; nesse caso a tarefa procura por pares de nomes de indivíduos $\langle \mathtt{a}, \mathtt{b} \rangle$ tais que $\mathcal{KB} \models r(\mathtt{a}, \mathtt{b})$.

Resposta a consultas O objetivo da tarefa de reposta a consultas é encontrar a solução para uma consulta q numa dada base de conhecimento $\mathcal{KB}$. A consulta q é normalmente especificada como uma proposição parcial com partes faltantes (variáveis livres) que devem ser preenchidas com nomes de indivíduos. Uma resposta à consulta q é uma solução (preenchimento específico) que torne q consequência lógica de $\mathcal{KB}$. Ou seja, uma solução é uma sequência de nomes de indivíduos que quando substituídos pelas variáveis livres de q fazem com que $\mathcal{KB} \models q$ seja verdadeiro. Uma extensão natural do problema é encontrar *todas* as possíveis respostas que satisfazem uma determinada consulta.

Indução e abdução O objetivo da tarefa de indução é, a partir dos axiomas assertivos da ABox de uma KB, gerar axiomas terminológicos de TBox e RBox correspondentes. Já o objetivo da tarefa de abdução é determinar, dadas uma base de conhecimento $\mathcal{KB}$ e um axioma α não implicado logicamente por $\mathcal{KB}$, quais axiomas (com determinadas características) precisam ser adicionados à $\mathcal{KB}$ para que $\mathcal{KB} \models \alpha$ se torne verdade. Note que diferentemente das tarefas anteriores, ambas indução e abdução são tarefas que não preservam verdade: os axiomas que elas podem ser falsificados.

1.2.4. Regras

Às vezes as tarefas de raciocínio discutidas acima não são poderosas o suficiente. Algumas aplicações precisam representar relações ou permitir derivações que vão além do que pode ser expresso e obtido em DL pura. Por exemplo, relações e derivações que envolvam computações arbitrárias, como àquelas que envolvem aritmética, ou a derivação de fatos apenas quando determinados outros fatos não podem ser derivados (inferências não-monotônicas).

Uma abordagem comum para aumentar a expressividade de uma DL é associar os axiomas da KB a um conjunto *regras* expressas em alguma linguagem de especificação de regras. Uma escolha popular de linguagem de regra são as cláusulas de Horn—um fragmento de lógica de primeira ordem que é também base para linguagens de programação lógicas, como Prolog e Datalog. Por exemplo, a linguagem SWRL (*Semantic Web Rule Language*) é essencialmente a combinação de Datalog com a linguagem de ontologia OWL. Outras combinações de DL com regras que aumentam a expressividade da DL (mas que nesse caso mantém a decidibilidade) são as linguagens DLP (*Description Logic Programs*) [Grosof et al. 2003] e DLR (*Description Logic Rules*) [Krötzsch et al. 2008]. Para mais detalhes sobre linguagens de regra veja [Hitzler et al. 2010].

1.2.5. Outras DLs

A família de DLs adota uma convenção que nos permite determinar as características de uma lógica a partir do seu nome. Essa convenção é dada pelo seguinte esquema:

$$((\mathcal{ALC} \mid \mathcal{S})\,[\mathcal{H}] \mid \mathcal{SR})\,[\mathcal{O}]\,[\mathcal{I}]\,[\mathcal{F} \mid \mathcal{N} \mid \mathcal{Q}],$$

Tabela 1.2. Características associadas às letras do nome de uma DL.

Símbolo	Características presentes/ausentes	exemplo
$\mathcal{ALC}$	Ausentes: axiomas de RBox, papel universal, papéis inversos, restrições de cardinalidade, nominals, autorrestrição.	$\neg(\texttt{Musica} \sqcup \texttt{Filme})$
$\mathcal{S}$	$\mathcal{ALC}$ com RIAs da forma $\texttt{r} \circ \texttt{r} \sqsubseteq \texttt{r}$, para r em N_R.	$\texttt{parteDe} \circ \texttt{parteDe} \sqsubseteq \texttt{parteDe}$
$\mathcal{H}$	$\mathcal{ALC}$ com RIAs da forma $r \sqsubseteq s$.	$\texttt{refilmagemDe} \sqsubseteq \texttt{baseadoEm}$
$\mathcal{SR}$	$\mathcal{ALC}$ com autorrestrição e todos os tipos de axiomas de RBox.	$\texttt{filhoDe} \circ \texttt{irmaoDe} \sqsubseteq \texttt{sobrinhoDe}$
$\mathcal{O}$	Presente: nominals.	$\{\texttt{universal}, \texttt{disney}\} \sqsubseteq \texttt{Estudio}$
$\mathcal{I}$	Presente: papéis inversos.	$\texttt{filhoDe} \sqsubseteq \texttt{paiDe}^{-}$
$\mathcal{F}$	Presente: axiomas de funcionalidade de papel (expressos como $\top \sqsubseteq \leq 1 r.\top$).	$\top \sqsubseteq \leq 1 \texttt{websiteOficial}.\top$
$\mathcal{N}$	Presente: restrições de cardinalidade da forma $\geq nr.\top$ e $\leq nr.\top$.	$\texttt{Serie} \sqsubseteq \geq 2.\texttt{possuiEpisodio}.\top$
$\mathcal{Q}$	Presente: todos os tipos de restrições de cardinalidade.	$\texttt{Serie} \sqsubseteq \geq 2.\texttt{possuiEpisodio}.\texttt{Episodio}$

em que os parênteses denotam grupos, a barra vertical representa uma escolha, e os colchetes delimitam os componentes opcionais. O significado de cada uma das letras acima é apresentado na Tabela 1.2.

Além de $\mathcal{SROIQ}$ outras lógicas de descrição populares são $\mathcal{ALC}$ e $\mathcal{SHOIQ}$. $\mathcal{ALC}$ [Schmidt-Schauß and Smolka 1991] é a menor DL com fechamento Booleano, isto é, em que os operadores Booleanos podem ser aplicados a conceitos de maneira irrestrita. $\mathcal{SHOIQ}$ [Baader et al. 2007] é uma lógica relacionada à linguagem de ontologia OWL DL (versão 1). Note que ambas $\mathcal{ALC}$ e $\mathcal{SHOIQ}$ são sublinguagens de $\mathcal{SROIQ}$.

Extensões espaciais e temporais de DL Em representação multimídia, é comum precisarmos descrever relações espaciais e temporais entre objetos. Por exemplo, durante o processo de anotar uma cena de um filme pode ser preciso especificar a posição relativa dos personagens na cena ou a sequência temporal das suas ações e diálogos.

Um formalismo simples porém expressivo para a especificação de relações espaciais 2D é o *Cálculo de Conexão de Regiões* (*Region Connection Calculus*, ou RCC8). O RCC8 [Randell et al. 1992] descreve oito tipos básicos de relações que podem se estabelecer entre duas regiões 2D: desconectadas (DC), externamente conectadas (EC), iguais (EQ), parcialmente sobrepostas (PO), parte própria tangencial (TPP), parte própria inversa (TPPi), parte própria não-tangencial (NTPP) e parte própria não-tangencial inversa (NTPPi). Essas oito relações são apresentadas na Figura 1.2. A família de lógicas de descrição espaciais $\mathcal{ALCI}_{\mathrm{RCC}}$ [Wessel 2003] contém lógicas especificamente projetadas para suportar as

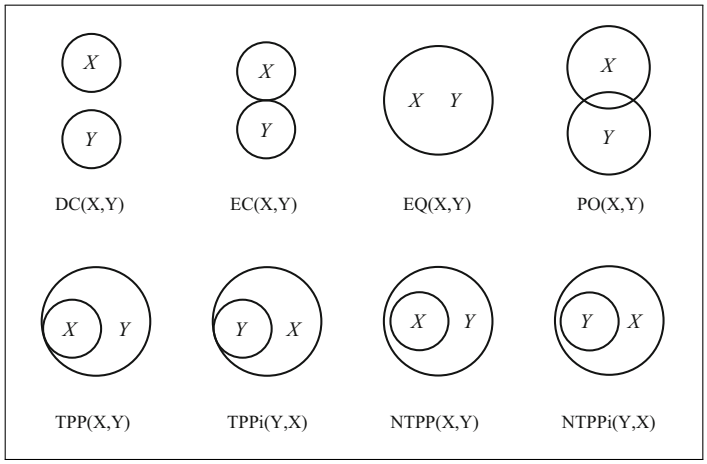


Figura 1.2. As relações do RCC8 [Sikos 2017].

relações do RCC8. Outras DLs que podem ser utilizadas para raciocínio espacial são as lógicas $\mathcal{ALC}(\mathrm{F})$ [Hudelot et al. 2015] e $\mathcal{ALC}(\mathrm{CDC})$ [Cristiani and Gabrielli 2011]. Essas últimas são propostas mais recentes que não se baseiam no RCC.

Para descrever fatos no tempo podemos utilizar uma das várias lógicas temporais disponíveis. A maioria dessas lógicas são extensões que adicionam à DL os operadores usuais de lógica temporal, isto é, operadores como $\Diamond$ (possivelmente no futuro) e $\Box$ (sempre no futuro). A lógica de descrição $\mathcal{ALC}$-LTL [Baader and Lippmann 2014] combina $\mathcal{ALC}$ com Lógica Temporal Linear (*Linear Temporal Logic*, ou LTL), que adota uma noção de tempo linear (sem ramificações) e é frequentemente utilizada na especificação de sistemas dinâmicos. Outra DL temporal é DL-CTL [Wang et al. 2014] que combina DL com *Computational Tree Logic* (CTL), uma lógica temporal com suporte a tempo não-linear (com ramificações). Há também uma extensão da lógica de descrição $\mathcal{SHIN}$ que quando usada em conjunto com OWL é chamada de tOWL [Milea et al. 2012]. tOWL permite a representação de pontos no tempo e de relações entre pontos e intervalos.

Extensões fuzzy de DL As informações extraídas de dados multimídia são frequentemente ambíguas e imprecisas. No entanto, as DLs tradicionais, por serem fragmentos de lógica de primeira ordem clássica, não cedem lugar a ambiguidade ou imprecisão—a lógica é Booleana: um dado fato é verdadeiro ou é falso. Isso significa que quando usamos as DLs tradicionais para representar dados multimídia é possível que alguma informação importante sobre incerteza seja perdida. Ou pior, podemos induzir um falso senso de certeza quando tal certeza não existe. Uma forma de amenizar esse problema é representar a incerteza na lógica.

As lógicas fuzzy são um tipo particular de lógica multi-valorada (não-Booleana) cuja semântica envolve a noção de *graus de verdade*. Os graus de verdade da lógica funcionam como possíveis "tons de cinza" entre os Booleanos verdadeiro e falso, e podem ser usados para formalizar indefinição ou imprecisão. Existem diversas extensões fuzzy de DL. Um bom ponto de partida para estudá-las é [Borgwardt and Peñaloza 2017].

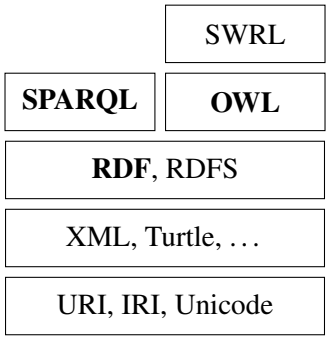


Figura 1.3. As camadas da Web Semântica.

1.3. Web Semântica

O termo *Web Semântica* diz respeito a um esforço iniciado no começo dos anos 2000 para estender a Web tradicional, então uma biblioteca global de documentos HTML interligados, para permitir a descrição e processamento semânticos desses documentos. Duas décadas depois, a Web se tornou um ambiente bem mais diverso e complexo, e o sucesso e viabilidade da visão original da Web Semântica é de certa forma contestável. O que não é contestável, porém, é o sucesso dos padrões e tecnologias desenvolvidas nesse período. Atualmente, o termo Web Semântica é mais utilizado para se referir à essas tecnologias e padrões.

Nesta seção, vamos descrever as principais tecnologias da Web Semântica. Essas tecnologias podem ser organizadas em camadas, conforme apresenta a Figura 1.3. As tecnologias que aparecem nas camadas superiores são definidas sobre aquelas que aparecem nas camadas inferiores. Vamos discutir praticamente todas as tecnologias que aparecem na figura, mas nosso foco será naquelas que aparecem em negrito, a saber, RDF, SPARQL e OWL, com ênfase em especial na última.

1.3.1. RDF

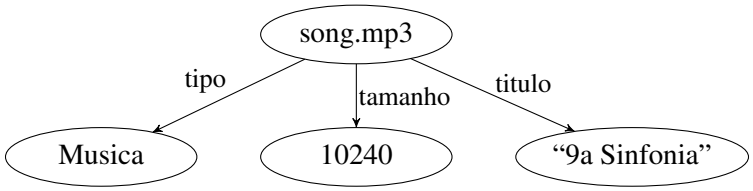
Um dos principais padrões da Web Semântica é o *Resource Description Framework* (RDF) [Wood et al. 2014]. O RDF é uma linguagem de representação baseada em grafos. Ele provê um vocabulário e construtores para a descrição de *conjuntos de triplas* que representam grafos dirigidos. Essas triplas triples chamadas de *triplas s-p-o* possuem o seguinte formato:

$$\langle sujeito, predicado, objeto \rangle.$$

As três triplas s-p-o abaixo descrevem, respectivamente, o tipo, tamanho em bytes e título associados ao arquivo de áudio “song.mp3”:

$$\langle \text{song.mp3}, \text{tipo}, \text{Musica} \rangle,\ \langle \text{song.mp3}, \text{tamanho}, 10240 \rangle,\ \langle \text{song.mp3}, \text{titulo}, \text{“9a Sinfonia”} \rangle.$$

Aqui está o grafo correspondente à essas triplas s-p-o:



A ideia é que cada tripla representa uma aresta dirigida no grafo. O sujeito da tripla determina o nó de origem da aresta, o objeto determina o nó de destino e o predicado determina o rótulo da aresta.

Evidentemente, o exemplo anterior é apenas um exemplo abstrato. Na prática, os componentes das triplas de um grafo RDF, isto é, os rótulos dos nós e arestas do grafo, devem seguir uma *sintaxe* específica. Uma sintaxe popular para RDF é a Turtle (*Terse RDF Triple Language*) [Carothers and Prudhommeaux 2014]. A sintaxe Turtle é uma alternativa à amplamente utilizada porém complicada sintaxe RDF/XML. Todos os nossos exemplos serão escritos em Turtle.

Há três tipos de nó em um grafo RDF: nós IRI, nós literais e nós em branco. Um *nó IRI* é um nó cujo rótulo é uma IRI (*Internationalized Resource Identifier*); uma IRI é uma generalização de URI com suporte à caracteres Unicode. Por exemplo, o endereço abaixo é uma IRI:

```
https://pt.wiktionary.org/wiki/maçã
```

Note os caracteres não-ASCII “ç” e “ã” acima. Em RDF, IRIs são usadas para se referir a *recursos* arbitrários, como objetos físicos, documentos, imagens, conceitos abstratos, números, *strings*, etc.

O segundo tipo de nó que ocorre num grafo RDF é o nó literal. Um *nó literal* é um nó rotulado por um literal, isto é, uma *string* que denota um valor. Na sintaxe Turtle, literais são escritos entre aspas duplas e opcionalmente acompanhados de *tags* de linguagem ou de tipo. Por exemplo, aqui estão três literais RDF (na sintaxe Turtle):

```
"Rock", "maçã"@pt, "10240"^^xsd:integer.
```

O primeiro é um literal simples, o segundo é um literal com *tag* de linguagem (Português no caso) e o terceiro é um literal com *tag* de tipo, também chamado de literal tipado. O prefixo `xsd:` que ocorre no terceiro literal denota um *namespace* apropriado que define a *tag* de tipo `integer`. Teremos mais a dizer sobre *namespaces* em instantes.

O terceiro tipo de nó que pode aparecer num grafo RDF é o nó em branco. Um *nó em branco* é um nó sem denotação específica; ele é um nó não-rotulado que não é nem nó IRI nem nó literal. Nós em branco são comumente utilizados como nós postiços (*dummies*) cujo único propósito é facilitar a codificação de estruturas complexas em grafos.

Numa tripla RDF $\langle s, p, o \rangle$ o sujeito s é necessariamente uma IRI ou nó em branco, o predicado p é necessariamente uma IRI e o objeto o é necessariamente uma IRI, literal ou nó em branco. Quando afirmamos uma determinada tripla RDF, estamos afirmando informalmente que “é válido o relacionamento indicado pelo predicado entre o recurso denotado pelo sujeito e o recurso ou valor denotado pelo objeto”.

A Figura 1.4 apresenta uma possível versão Turtle do grafo que descreve o arquivo de áudio "song.mp3" apresentado há alguns parágrafos.[9] Cada aresta do grafo (isto é, tripla do conjunto) corresponde a uma tripla no documento Turtle (linhas 5–7). Em Turtle, as partes do sujeito, predicado e objeto são separadas por espaços e cada tripla é terminada por um ponto (`.`). Observe que as IRIs aparecem entre os caracteres `<...>`.

```
1 @base <http://example.org/data/> .
2 @prefix ex: <http://example.org/#> .
3 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
4 <song.mp3> ex:tipo ex:Musica .
5 <song.mp3> ex:tamanho "10240"^^xsd:integer .
6 <song.mp3> ex:titulo "9a Sinfonia" .
```

Figura 1.4. Representação em Turtle das triplas RDF que descrevem "song.mp3".

As diretivas que ocorrem nas linhas 1–3 da Figura 1.4 definem os espaços de nomes (*namespaces*) do documento. Isto é, elas especificam como IRIs relativas ou prefixadas devem ser interpretadas. A diretiva `@base` define a IRI base de todo o documento Turtle. Qualquer IRI relativa que ocorra no documento será interpretada como relativa à essa IRI base. Logo, por exemplo, as ocorrências de `<song.mp3>` nas linhas 4–6 da Figura 1.4 serão interpretadas como a IRI absoluta:

```
http://example.org/data/song.mp3
```

A diretiva `@prefix` introduz um novo rótulo de prefixo que pode ser utilizado para abreviar IRIs. Por exemplo, na Figura 1.4, os rótulos `ex:` e `xsd:` correspondem aos seguintes prefixos de IRIs:

```
http://example.org/#
http://www.w3.org/2001/XMLSchema#
```

Observe que IRIs prefixadas não são escritas entre `<...>`.

A sintaxe Turtle possui muitas outras facilidades. Algumas dessas são úteis para evitar repetições dentro do documento. Por exemplo, a Figura 1.5 utiliza algumas dessas facilidades adicionais para reescrever em menos linhas o documento da Figura 1.4. Observe os caracteres ponto e vírgula "`;`" nas linhas 3 e 4 da Figura 1.5. Esses caracteres são utilizados para separar pares predicado-objeto que se referem a um mesmo sujeito; nesse caso, à IRI `http://example.org/data/song.mp3`. Esse tipo de construção é chamada de *lista de predicados*. Observe também o literal tipado 10240 na linha 4. Alguns tipos comuns de literais, como inteiros e números em ponto flutuante, podem ser escritos diretamente em suas sintaxes tradicionais em Turtle.

[9] É comum se usar nós retangulares para representar literais em grafos RDF. Por se tratar de uma representação abstrata, o grafo anterior não segue essa convenção.

```
@prefix ex: <http://example.org/#> .
<http://example.org/data/song.mp3>
  ex:tipo ex:Musica ;
  ex:tamanho 10240 ;
  ex:titulo "9a Sinfonia" .
```

Figura 1.5. Exemplo da Figura 1.4 reescrito usando-se uma lista de predicados.

É importante perceber que um grafo RDF é apenas uma estrutura de dados, e que Turtle, RDF/XML e outras sintaxes para RDF são apenas formas de se codificar essa estrutura de dado em uma *string*. Ainda mais importante é perceber que os significados dos rótulos que ocorrem em um grafo RDF não são definidos pelo padrão RDF. Esses rótulos podem indicar alguma coisa para pessoa que os lê, mas para computador que entende apenas RDF esses rótulos não possuem significado—eles servem apenas para identificar e distinguir componentes específicos do grafo.

Existe um outro padrão, chamado *RDF Schema* (RDFS) [Hayes and Patel-Schneider 2014], definido sobre o RDF, que especifica um vocabulário básico (conjunto predefinido de termos) e significados associados. Esse vocabulário está disponível no seguinte *namespace*:

`http://www.w3.org/2000/01/rdf-schema#`

O RDFS provê um conjunto básico de termos que pode ser utilizado para definir termos mais específicos cujo significado é dado através da descrição de suas inter-relações com outros termos. (Note que usamos uma abordagem similar na Seção 1.2 quando introduzimos *strings* representando conceitos e papéis, como `Diretor` e `dirige`, e em seguida definimos os seus significados através de GCIs e RIAs que os relacionam com outros termos.)

O RDFS é às vezes chamado de *linguagem de ontologia leve* por permitir a definição ontologias (especificações de conceitualizações com semântica formal) através de um vocabulário pequeno cuja semântica é relativamente simples. Essas características podem ser suficientes para certas aplicações mas, em geral, aplicações mais sofisticadas demandam linguagens ontológicas mais poderosas. A escolha natural nesse caso é OWL, também uma tecnologia da Web Semântica. Vamos discutir OWL na Seção 1.3.3. Antes disso, porém, vamos apresentar SPARQL, a linguagem de consulta de RDF.

1.3.2. SPARQL

Na seção anterior mostramos um exemplo de grafo RDF contendo de quatro nós e três arestas. Quando estamos trabalhando com grafos relativamente pequenos como esse não precisamos nos preocupar com eficiência. Mas, na prática, grafos RDF raramente são pequenos. Não é incomum encontrarmos aplicações que lidam com grafos contendo milhões de nós e arestas. Para tais aplicações, ferramentas simples não são o bastante. Elas precisam de ferramentas especializadas, em particular, bancos de dados especializados, os chamados *bancos de dados de grafos* ou *bases de triplas* (*triple stores*), projetados para armazenar e processar de forma eficiente grandes volumes de dados RDF.

SPARQL (*SPARQL Protocol and RDF Query Language*) está para os bancos de dados de grafos e bases de triplas assim como SQL está para os bancos de dados

relacionais. SPARQL é uma interface declarativa para consultar e manipular o conteúdo da base/grafo [Harris and Seaborne 2013]. Há dois tipos principais de consultas SPARQL: reconhecimento de padrões em grafos e navegação em grafos. Vamos discutir ambos os tipos a seguir.

Reconhecimento de padrões em grafos Uma consulta de reconhecimento de padrão é uma consulta que busca por um dado padrão estrutural (limitado) no grafo. Em SPARQL, tais padrões estruturais são expressados como *padrões de triplas*, isto é, triplas RDF em que o sujeito, predicado e objeto podem ser variáveis (nomes começando com "`?`"). Um *padrão básico de grafo* (*Basic Graph Pattern*, ou BGP) consiste da combinação de um ou mais padrões de triplas.

Considere o grafo RDF apresentado na Figura 1.6 que descreve o filme *Taxi Driver*. Suponha que queiramos listar todos os pares de pessoas distintas que atuaram nesse filme. Podemos especificar essa consulta em SPARQL da seguinte forma:

```
1 PREFIX: <http://example.org/#>
2 SELECT ?x1 ?x2
3 WHERE {
4   ?x1 :atuaEm ?x3 . ?x1 :tipo :Pessoa .
5   ?x2 :atuaEm ?x3 . ?x1 :tipo :Pessoa .
6   ?x3 :titulo "Taxi Driver" . ?x3 :tipo :Filme .
7   FILTER (?x1 != ?x2)
8 }
```

Essa consulta expressa um BGP contendo seis padrões de triplas. Ela seleciona todos os pares de rótulos x_1 e x_2 (linha 2) com $x_1 \neq x_2$ (linha 7) tais que, para algum rótulo x_3, cada uma das seguintes triplas ocorre no grafo:

$$\langle x_1, \texttt{:atuaEm}, x_3 \rangle \quad \text{e} \quad \langle x_1, \texttt{:tipo}, \texttt{:Pessoa} \rangle \qquad \text{(linha 4)}$$
$$\langle x_2, \texttt{:atuaEm}, x_3 \rangle \quad \text{e} \quad \langle x_2, \texttt{:tipo}, \texttt{:Pessoa} \rangle \qquad \text{(linha 5)}$$
$$\langle x_3, \texttt{:titulo}, \texttt{"Taxi Driver"} \rangle \quad \text{e} \quad \langle x_3, \texttt{:tipo}, \texttt{:Filme} \rangle \qquad \text{(linha 6)}$$

Se aplicarmos essa consulta ao grafo da Figura 1.6 vamos obter o seguinte resultado:

```
1 ?x1          ?x2
2 ---------------------
3 :scorsese    :de-niro
4 :de-niro     :scorsese
```

Ou seja, o nosso *conjunto resposta* contém exatamente dois pares de rótulos: um em que x_1 é `:scorsese` e x_2 é `:de-niro` (linha 3 do resultado), e outro em que x_1 é `:de-niro` e x_2 é `:scorsese` (linha 4 do resultado).

Navegação em grafos O outro tipo comum de consulta suportado por SPARQL são as consultas de navegação em grafos. Essas são consultas que navegam na topologia do grafo percorrendo caminhos de tamanhos potencialmente arbitrários. Uma *consulta de caminho*

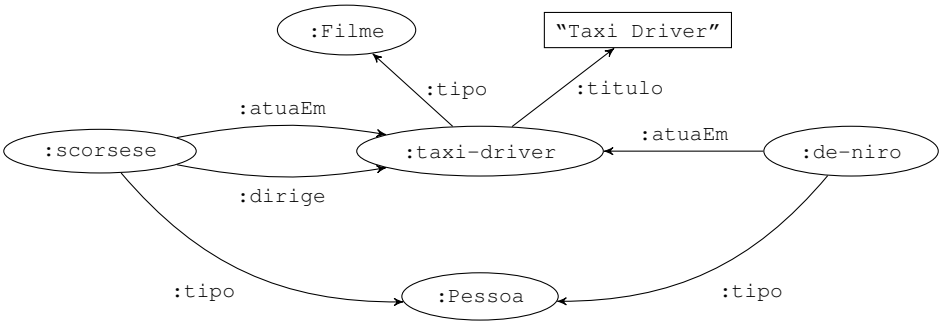


Figura 1.6. Grafo RDF descrevendo *Taxi Driver*.

é um tipo básico de consulta de navegação da forma $x \xrightarrow{\alpha} y$ que obtém todos os pares $\langle x, y \rangle$ tais que existe um caminho no grafo de x para y que satisfaz uma dada condição α.

Por exemplo, as consultas de caminho

$$x \xrightarrow{\texttt{:dirige}} y \quad \text{e} \quad x \xrightarrow{\texttt{:atuaEm} \circ \texttt{:tipo}} y$$

selecionam, respectivamente, todos os nós x e y tais que existe uma aresta com rótulo `:dirige` de x para y, e todos os nós x e y tais que existe um caminho de tamanho dois de x para y em que a primeira aresta do caminho possui rótulo `:atuaEm` e a segunda aresta possui rótulo `:tipo`. (Lembre-se de que um *caminho* em um grafo é uma sequência de arestas que conectam uma sequência de nós.)

Podemos especificar essa última consulta em SPARQL da seguinte forma:

```
SELECT ?x ?y
WHERE  ?x (:atuaEm/:tipo) ?y
```

Aqui o símbolo "`/`" denota encadeamento de rótulos de arestas e corresponde ao "$\circ$" da notação abstrata. Se aplicarmos essa consulta SPARQL ao grafo da Figura 1.6 vamos obter a resposta:

```
?x           ?y
------------------
:scorsese    :Movie
:de-niro     :Movie
```

Para um exemplo mais interessante, considere a consulta de caminho

$$x \xrightarrow{(\texttt{:atuaEm} \circ \texttt{:atuaEm}^{-})^{+}} y.$$

Essa consulta busca todos os atores que possuem uma distância de colaboração finita[10], isto é, atores x e y tais que x e y contracenam no mesmo filme (distância 0), ou x e y contracenam com o mesmo z em algum filme (distância 1), ou x contracena com algum z_1 em algum filme e y contracena com algum z_2 em algum filme e z_1 e z_2 contracenam no mesmo filme (distância 2), e assim por diante. O código SPARQL abaixo implementa essa consulta:

```
SELECT ?x ?y
WHERE  ?x (:atuaEm/^:atuaEm)+ ?y
```

Nesse caso, o símbolo "^" denota o inverso de um rótulo de aresta (ele corresponde ao símbolo "−" da notação abstrata) e o símbolo "+" denota uma ou mais repetições do padrão que o precede.

Suponha que adicionemos as seguintes triplas ao grafo da Figura 1.6:

$\langle$:de-niro, :atuaEm, :cassino$\rangle$ e $\langle$:sheron-stone, :atuaEm, :cassino$\rangle$.

Se rodarmos o SPARQL anterior sobre esse grafo estendido vamos obter o seguinte resultado:

```
?x              ?y
-----------------------------
:scorsese       :de-niro
:de-niro        :scorsese
:de-niro        :sharon-stone
:sharon-stone   :de-niro
:scorsese       :sharon-stone
:sharon-stone   :scorsese
```

SPARQL possui muitas outras funcionalidades, como grupos, opções, alternativas, filtros, etc., que podem ser combinadas com BGPs e operadores de navegação para especificar consultas complexas. Nosso objetivo aqui foi apresentar uma visão geral de SPARQL. Para uma introdução mais detalhada à linguagem sugerimos [Hitzler et al. 2010]. Além de SPARQL, há outras linguagens para consultas em grafos. Veja [Angles 2012] para uma discussão geral dos princípios por trás dessas linguagens. Para uma introdução prática aos bancos de dados de grafos recomendamos [Robinson et al. 2013].

1.3.3. OWL

Web Ontology Language (OWL) é uma família de linguagens ontológicas cuja sintaxe e semântica (formal) é padronizada [W3C OWL WG 2012]. Uma *ontologia* é um conjunto de proposições descritivas sobre um determinado domínio—proposições como àquelas de lógica de descrição. OWL é baseado em lógica de descrição mas possui muitas outras facilidades que vão além de DL pura, como por exemplo tipos de dados e funcionalidades para versionamento e anotação de ontologias. Nesta seção, vamos focar na versão 2 de OWL (chamada de OWL 2). Essa é a versão mais recente da linguagem e é compatível com a versão anterior.

Existem três sublinguagens de OWL, também chamadas de *espécies*: OWL Full, OWL DL e OWL Lite. OWL Full contém ambas OWL DL e OWL Lite, e OWL DL contém OWL Lite. O que distingue essas três sublinguagens é o seu grau de expressividade e, consequentemente, a complexidade computacional das tarefas de raciocínio associadas a cada uma delas.

[10]Quando x é o ator Kevin Bacon essa distância é conhecida como a *distância Bacon* ou *grau Bacon*. Veja: https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon

Tabela 1.3. Diferenças terminológicas entre OWL e DL.

OWL	DL
nome de classe	nome de conceito
classe	conceito
nome de propriedade de objeto	nome de papel
propriedade de objeto	papel
ontologia	base de conhecimento
axioma	axioma
vocabulário	vocabulário/assinatura

Como qualquer linguagem, para apresentar OWL vamos ter que escolher uma determinada sintaxe. No caso, vamos utilizar a sintaxe RDF/Turtle. Um documento OWL escrito nessa linguagem é também um documento RDF válido. Além de RDF, outras sintaxes comuns de OWL são a sintaxe em estilo funcional e a sintaxe Manchester. No entanto, a única sintaxe cujo suporte é mandatório para todas as ferramentas OWL 2 é a sintaxe RDF/XML [Hitzler et al. 2014].

A seguir, vamos apresentar os termos e a semântica de OWL a partir de traduções de DL. (Uma ontologia OWL 2 DL é essencialmente uma base de conhecimento $\mathcal{SROIQ}$.) Antes de fazer isso, porém, precisamos esclarecer alguns conflitos terminológicos. Por razões históricas, OWL e DL às vezes usam termos distintos para se referir ao mesmo objeto. Essas diferenças terminológicas são apresentadas na Tabela 1.3.

De $\mathcal{SROIQ}$ para OWL 2 DL A tradução de uma base de conhecimento $\mathcal{SROIQ}$ para um documento OWL consiste de três passos.[11] Primeiro, precisamos prefixar ao documento as seguintes declarações de *namespace*:

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

Os prefixos `rdfs:`, `rdf:` e `xsd:` são necessários pois OWL reusa os termos desses *namespaces*.

A segunda coisa que precisamos fazer é adicionar ao documento OWL declarações de tipos para os nomes de conceitos e nomes de papéis que ocorrem na KB. Ou seja, para cada nome de conceito `C` que ocorre na KB, adicionamos ao documento a tripla:

```
C rdf:type owl:Class .
```

E para cada nome de papel `r` que ocorre na KB, adicionamos a tripla:

```
r rdf:type owl:ObjectProperty .
```

[11] As traduções que apresentamos aqui foram adaptadas de [Rudolph 2011].

A Figura 1.7 apresenta o documento OWL correspondente à KB de fatos sobre filmes apresentada no final da Seção 1.2.1. As declarações de tipo nas linhas 7–13 foram geradas pelas traduções acima. Observe que por causa da diretiva `@prefix` na linha 5, IRIs com prefixo vazio, como `:Ator`, são interpretadas como sendo prefixadas pela IRI `http://example.org/movie-facts#`.

O terceiro passo da tradução de uma KB $\mathcal{SROIQ}$ para OWL é o mais complicado dos três: precisamos traduzir os axiomas que ocorrem em cada uma das caixas da KB. A complexidade, nesse caso, decorre da estrutura sintática de GCIs e RIAs que não pode ser codificada diretamente em RDF. Para reescrevermos GCIs e RIAs em RDF vamos precisar utilizar nós em branco e técnicas de codificação de listas em grafos.

A tradução exata de axiomas $\mathcal{SROIQ}$ para OWL é dada pela função $[\![\square]\!]$ abaixo:

$$
\begin{aligned}
[\![r_1 \circ \cdots \circ r_n \sqsubseteq r]\!] &= [\![r]\!]_{\mathbf{C}}\ \texttt{owl:propertyChainAxiom}\ (\ [\![r_1]\!]_{\mathbf{R}} \ldots [\![r_n]\!]_{\mathbf{R}}\ )\ . \\
[\![\mathsf{Sym}(r)]\!] &= [\![r]\!]_{\mathbf{R}}\ \texttt{owl:type owl:SymmetricProperty}\ . \\
[\![\mathsf{Asy}(r)]\!] &= [\![r]\!]_{\mathbf{R}}\ \texttt{owl:type owl:AsymmetricProperty}\ . \\
[\![\mathsf{Tra}(r)]\!] &= [\![r]\!]_{\mathbf{R}}\ \texttt{owl:type owl:TransitiveProperty}\ . \\
[\![\mathsf{Ref}(r)]\!] &= [\![r]\!]_{\mathbf{R}}\ \texttt{owl:type owl:ReflexiveProperty}\ . \\
[\![\mathsf{Irr}(r)]\!] &= [\![r]\!]_{\mathbf{R}}\ \texttt{owl:type owl:IrreflexiveProperty}\ . \\
[\![\mathsf{Dis}(r,s)]\!] &= [\![r]\!]_{\mathbf{R}}\ \texttt{owl:propertyDisjointWith}\ [\![s]\!]_{\mathbf{R}}\ . \\
[\![C \sqsubseteq D]\!] &= [\![C]\!]_{\mathbf{C}}\ \texttt{rdfs:subClassOf}\ [\![D]\!]_{\mathbf{C}}\ . \\
[\![C(\mathtt{a})]\!] &= \mathtt{a}\ \texttt{rdf:type}\ [\![C]\!]_{\mathbf{C}}\ . \\
[\![\mathtt{r}(\mathtt{a},\mathtt{b})]\!] &= \mathtt{a}\ \mathtt{r}\ \mathtt{b}\ . \\
[\![\mathtt{r}^-(\mathtt{a},\mathtt{b})]\!] &= \mathtt{b}\ \mathtt{r}\ \mathtt{a}\ . \\
[\![\neg r(\mathtt{a},\mathtt{b})]\!] &= \texttt{[] rdf:type owl:NegativePropertyAssertion ;} \\
&\qquad \texttt{owl:assertionProperty}\ [\![r]\!]_{\mathbf{R}}\ \texttt{;} \\
&\qquad \texttt{owl:sourceIndividual}\ \mathtt{a}\ \texttt{;} \\
&\qquad \texttt{owl:targetIndividual}\ \mathtt{b}\ . \\
[\![\mathtt{a} \approx \mathtt{b}]\!] &= \mathtt{a}\ \texttt{owl:sameAs}\ \mathtt{b}\ . \\
[\![\mathtt{a} \not\approx \mathtt{b}]\!] &= \mathtt{a}\ \texttt{owl:differentFrom}\ \mathtt{b}\ .
\end{aligned}
$$

em que as funções auxiliares $[\![\square]\!]_{\mathbf{R}}$ e $[\![\square]\!]_{\mathbf{C}}$ são usadas para traduzir papéis e expressões de conceitos. Essas funções auxiliares são definidas da seguinte forma:

$$
\begin{aligned}
[\![u]\!]_{\mathbf{R}} &= \texttt{owl:topObjectProperty} \\
[\![\mathtt{r}]\!]_{\mathbf{R}} &= \mathtt{r} \\
[\![\mathtt{r}^-]\!]_{\mathbf{R}} &= \texttt{[owl:inverseOf :r]}
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathtt{C}]\!]_{\mathbf{C}} &= \mathtt{C} \\
[\![\top]\!]_{\mathbf{C}} &= \texttt{owl:Thing} \\
[\![\bot]\!]_{\mathbf{C}} &= \texttt{owl:Nothing} \\
[\![\{\mathtt{a}_1,\ldots,\mathtt{a}_n\}]\!]_{\mathbf{C}} &= \texttt{[rdf:type owl:Class ; owl:oneOf(}\ :\mathtt{a}_1\ \ldots\ :\mathtt{a}_n\ \texttt{)]} \\
[\![\neg C]\!]_{\mathbf{C}} &= \texttt{[rdf:type owl:Class ; owl:complementnOf}\ [\![C]\!]_{\mathbf{C}}\ \texttt{]}
\end{aligned}
$$

$$[\![C_1 \sqcap \cdots \sqcap C_n]\!]_{\mathbf{C}} = \texttt{[rdf:type owl:Class ; owl:intersectionOf (} [\![C_1]\!]_{\mathbf{C}} \ldots [\![C_n]\!]_{\mathbf{C}} \texttt{)]}$$

$$[\![C_1 \sqcup \cdots \sqcup C_n]\!]_{\mathbf{C}} = \texttt{[rdf:type owl:Class ; owl:unionOf (} [\![C_1]\!]_{\mathbf{C}} \ldots [\![C_n]\!]_{\mathbf{C}} \texttt{)]}$$

$$[\![\exists r.C]\!]_{\mathbf{C}} = \texttt{[rdf:type owl:Restriction ;}$$
$$\texttt{owl:onProperty} [\![r]\!]_{\mathbf{R}} \texttt{ ; owl:someValuesFrom} [\![C]\!]_{\mathbf{C}} \texttt{]}$$

$$[\![\forall r.C]\!]_{\mathbf{C}} = \texttt{[rdf:type owl:Restriction ;}$$
$$\texttt{owl:onProperty} [\![r]\!]_{\mathbf{R}} \texttt{ ; owl:allValuesFrom} [\![C]\!]_{\mathbf{C}} \texttt{]}$$

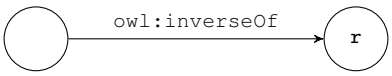
$$[\![\exists r.\mathsf{Self}]\!]_{\mathbf{C}} = \texttt{[rdf:type owl:Restriction ;}$$
$$\texttt{owl:onProperty} [\![r]\!]_{\mathbf{R}} \texttt{ ; owl:hasSelf true]}$$

$$[\![\geq n r.C]\!]_{\mathbf{C}} = \texttt{[rdf:type owl:Restriction ;}$$
$$\texttt{owl:minQualifiedCardinality} n \texttt{ ;}$$
$$\texttt{owl:onProperty} [\![r]\!]_{\mathbf{R}} \texttt{ ; owl:onClass} [\![C]\!]_{\mathbf{C}} \texttt{]}$$

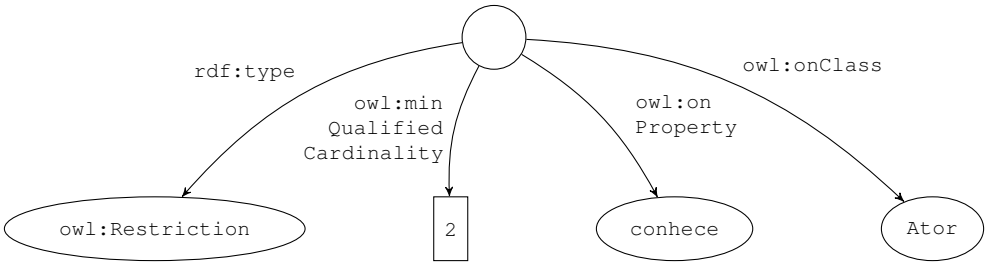
$$[\![\leq n r.C]\!]_{\mathbf{C}} = \texttt{[rdf:type owl:Restriction ;}$$
$$\texttt{owl:maxQualifiedCardinality} n \texttt{ ;}$$
$$\texttt{owl:onProperty} [\![r]\!]_{\mathbf{R}} \texttt{ ; owl:onClass} [\![C]\!]_{\mathbf{C}} \texttt{]}$$

Dois comentários sobre a tradução acima se fazem necessários. Primeiro, observe que nomes de indivíduos, nomes de conceitos e nomes de papéis são mantidos sem alteração pelas funções $[\![\square]\!]$, $[\![\square]\!]_{\mathbf{R}}$ e $[\![\square]\!]_{\mathbf{C}}$. Para simplificar, vamos assumir que esses nomes estão definidos no *namespace* de prefixo vazio.

Segundo, observe que papéis e expressões de conceitos complexas são traduzidas pelas funções $[\![\square]\!]_{\mathbf{R}}$ e $[\![\square]\!]_{\mathbf{C}}$ como subgrafos identificados por nós em branco. Na sintaxe Turtle, esses nós em branco são introduzidos por colchetes. Por exemplo, o papel $\texttt{r}^-$ é traduzido para o subgrafo



em que o nó em branco à esquerda representa o inverso de `r`. De maneira similar, o conceito complexo $\geq$2`conhece.Ator` é traduzido para o subgrafo



Novamente, o nó em branco acima representa o conceito complexo como um todo.

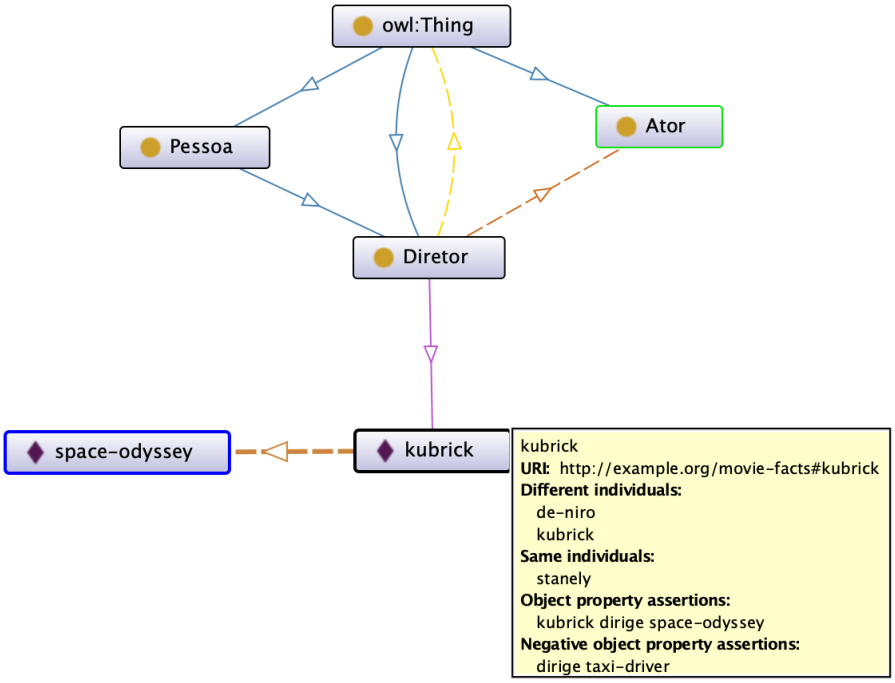
O resultado da aplicação da função $[\![\square]\!]$ aos axiomas da KB fatos sobre filmes é apresentado na Figura 1.7. Os axiomas de ABox resultantes aparecem nas linhas 16–28, os de TBox nas linhas 31–53 e os de RBox nas linhas 55–56.

```
@prefix owl:    <http://www.w3.org/2002/07/owl#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix :       <http://example.org/movie-facts#> .

:Ator       rdf:type    owl:Class .
:Diretor    rdf:type    owl:Class .
:Pessoa     rdf:type    owl:Class .
:atuaEm     rdf:type    owl:ObjectProperty .
:dirige     rdf:type    owl:ObjectProperty .
:conhece    rdf:type    owl:ObjectProperty .
:gosta      rdf:type    owl:ObjectProperty .

# Abox
:kubrick    rdf:type    :Diretor .
:de-niro    rdf:type    [rdf:type owl:Class ;
                         owl:intersectionOf (:Ator :Diretor)] .
:stanley    rdf:type    [rdf:type owl:Restriction ;
                         owl:minQualifiedCardinality 2 ;
                         owl:onProperty :conhece ; owl:onClass :Ator ] .
:kubrick    :dirige    :space-odyssey .
[]          rdf:type    owl:NegativePropertyAssertion ;
                        owl:assertionProperty :dirige ;
                        owl:sourceIndividual :kubrick ;
                        owl:targetIndividual :taxi-driver .
:kubrick    owl:sameAs  :stanely .
:kubrick    owl:differentFrom   :de-niro .

# TBox
:Diretor   rdfs:subClassOf     [rdf:type owl:Restriction ;
                                owl:onProperty :dirige ;
                                owl:someValuesFrom owl:Thing] .
[rdf:type owl:Restriction ;
 owl:onProperty :dirige ;
 owl:someValuesFrom owl:Thing]
           rdfs:subClassOf     :Diretor .
:Diretor   rdfs:subClassOf     :Pessoa .
:Diretor   rdfs:subClassOf     [rdf:type owl:Restriction ;
                                owl:minQualifiedCardinality 1 ;
                                owl:onProperty :conhece ;
                                owl:onClass :Ator] .
[rdf:type owl:Class ;
 owl:intersectionOf  ([rdf:type owl:Restriction ;
                       owl:onProperty :dirige ;
                       owl:someValuesFrom owl:Thing]
                       [rdf:type owl:Restriction ;
                        owl:onProperty :dirige ;
                        owl:allValuesFrom [rdf:type owl:Class ;
                                           owl:oneOf (:a-bronx-tale
                                                      :good-shepherd)]])]
           rdfs:subClassOf     [rdf:type owl:Class ;
                                owl:oneOf (:de-niro)] .
# RBox
:gosta      owl:propertyChainAxiom  (:dirige) .
:conhece    owl:propertyChainAxiom  (:dirige [owl:inverseOf :atuaEm]) .
```

Figura 1.7. OWL correspondente à KB de fatos sobre filmes da Seção 1.2.1.

Simulando os tipos adicionais de axiomas de OWL em $\mathcal{SROIQ}$ As funcionalidades de OWL utilizadas pelo algoritmo de tradução anterior são suficientes para representar qualquer axioma $\mathcal{SROIQ}$. No entanto, OWL possui muitas outras funcionalidades, incluindo tipos adicionais de axiomas lógicos que não possuem contrapartida direta em $\mathcal{SROIQ}$. Esses tipos adicionais de axiomas de OWL são úteis na prática, mas eles não adicionam expressividade extra do ponto de vista lógico. Ou seja, esses axiomas adicionais são apenas açúcares sintáticos que podem ser sempre reescritos em termos dos tipos de axiomas disponíveis em $\mathcal{SROIQ}$. Veja [Rudolph 2011] para mais detalhes.

Ferramentas para manipulação de ontologias Se carregarmos o arquivo Turtle da Figura 1.7 na ferramenta Protégé[12] e solicitarmos um grafo da ontologia vamos obter o seguinte:



Aqui os nós contendo um círculo amarelo representam conceitos (classes), os nós contendo losangos roxos representam indivíduos e as setas coloridas representam relacionamentos específicos entre nós. Por exemplo, a seta azul de `Pessoa` para `Diretor` representa o relacionamento "possui subclasse". O retângulo amarelo, contendo informações adicionais sobre `kubrick`, é o *tool-tip* apresentado pelo Protégé quando movemos o cursor do mouse sobre um nó ou aresta do grafo.

O Protégé é o editor de ontologias mais utilizado; ele suporta diversos *plugins* para visualização de ontologias, depuração e raciocínio. Além do Protégé, há muitas outras ferramentas para projetar e implementar ontologias. Para uma lista abrangente, veja [Hitzler et al. 2010] e [W3C Semantic Web Wiki 2019]

***Linked data* e grafos de conhecimento** Ao longo dos anos, o foco da visão da Web Semântica migrou da descrição de documentos para a descrição de dados em geral. O

[12]https://protege.stanford.edu/

	Vocabulário/Ontologia	Linguagem
Derivadas de descritores	Content Ontology for the TV-Anytime Content	OWL
	Content Ontology for the TV-Anytime Format	OWL
	Core Ontology for Multimedia (COMM)	OWL
	Visual Descriptor Ontology (VDO)	OWL
De propósito geral	Dublin Core	RDFS
	FOAF	OWL
	Schema.org	OWL
	SUMO	SUO-KIF
	WordNet 3.x	OWL
Especializadas	3D Modeling Language (3DMO)	OWL 2
	Audio Effects Ontology (AUFX-O)	OWL
	Linked Movie Database (LMD)	RDFS
	Multimedia Metadata Ontology (M3O)	OWL
	Ontology for Media Resources	OWL
	Video Ontology (VidOnt)	OWL

Tabela 1.4. Alguns vocabulários e ontologias para multimídia.

termo *Linked Open Data* (LOD) tem sido utilizado para se referir às bases abertas de dados relacionados (*linked data*), tais como DBpedia[13] (derivada da Wikipedia), Wikidata[14] (curada pela Fundação Wikimedia), etc. A maioria dessas bases adota as tecnologias discutidas nessa seção, porém muitas delas não são *ontologias* propriamente ditas.

Outro termo comumente associado a *linked data* é o termo *grafo de conhecimento*. Esse último é um termo geral que faz alusão ao fato de que o sistema ou banco de dados ao qual o termo se aplica utiliza grafos para representar conhecimento. Nesse sentido, um documento OWL codificado como um RDF é um grafo de conhecimento. Mas o fato de ser um grafo de conhecimento não implica em ser uma ontologia. Por exemplo, o grafo apresentado na Figura 1.6 é um grafo de conhecimento, mas certamente não é uma ontologia.

Ontologias para multimídia Há diversas ontologias para descrição de dados multimídia. Podemos classificá-las basicamente em três grupos: as derivadas de descritores, as de propósito geral e as ontologias especializadas. As ontologias do primeiro grupo são aquelas derivadas dos padrões de metadados clássicos, tais como MPEG-7, MPEG-21 e TV-Anytime. Essas ontologias são complexas, limitadas em termos de expressividade e possuem falhas de projeto que dificultam o seu uso (elas normalmente não seguem boas práticas de projeto de ontologias). As ontologias do segundo grupo tendem a ser mais maduras e bem definidas. Já as ontologias do terceiro grupo, assim como as do primeiro, também são problemáticas. Elas normalmente possuem problemas de conceitualização, por exemplo, escopo incerto ou mal definido, e costumam não utilizar termos de ontologias mais gerais (ontologias *upper level*) o que é considerado uma prática ruim.

[13]https://wiki.dbpedia.org/
[14]https://www.wikidata.org/

A Tabela 1.4 apresenta algumas ontologias que fazem parte desses três grupos. Para mais detalhes sobre essas ontologias, incluindo referências, veja [Sikos 2017].

O propósito de se utilizar uma das ontologias anteriores é, evidentemente, promover o reúso de termos e definições. Por exemplo, usando os ontologias Schema.org and Dublin Core podemos reescrever o RDF que descreve o arquivo de áudio "song.mp3" (Figura 1.5) da seguinte forma:

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://schema.org/> .
<http://example.org/data/song.mp3> rdf:type schema:AudioObject ;
                                   schema:contentSize "10240" ;
                                   dc:title "9a Sinfonia" .
```

Observe que podemos usar *URI fragments* [Pfeiffer et al. 2012] fazer referência a partes específicas do recurso que está sendo descrito. Por exemplo, se quisermos indicar que o tom de um segmento específico de "song.mp3", digamos dos 10s aos 20s, é C Maior, podemos escrever:

```
@prefix keys: <http://purl.org/NET/c4dm/keys.owl#>
@prefix mo:   <http://purl.org/ontology/mo/>
<http://example.org/data/song.mp3#t=10,20> mo:key keys:CMajor .
```

em que `mo:key` e `keys:CMajor` são termos definidos na Music Ontology[15]. De maneira similar, podemos usar *URI fragments* fazer referencias a fragmentos espaciais de recursos especificados via `#xywh=`x`,`y`,`w`,`h. Falaremos mais sobre objetos de mídia e suas partes na seção seguinte.

1.4. Uma Abordagem Híbrida: Hyperknowledge

O *Hyperknowledge* [Moreno et al. 2017] é um modelo híbrido para representação de conhecimento que permite especificar e interligar objetos multimídia e conceitos. O Hyperknowledge estende o *Nested Context Model* (NCM) [Soares and Rodrigues 2005], um modelo hipermídia clássico, com construções que combinam funcionalidades dos domínios de hipermídia e representação de conhecimento. Para melhor entendermos a origem desse modelo, vamos discutir rapidamente as preocupações principais de ambas essas comunidades.

Nas últimas décadas a comunidade de hipermídia tem se dedicado a definição de modelos e linguagens para expressar relacionamentos entre objetos de mídia (textos, imagens, vídeos, etc.). Em linguagens como HTML, NCL, SMIL e SVG, podemos especificar como diferentes objetos interagem uns com os outros e com os usuários, mas não há como relacionar esses objetos aos conceitos que eles representam—pelo menos não diretamente. Também não podemos descrever nessas linguagens ontologias como as que discutimos nas seções anteriores. Padrões de metadados (tais como MPEG-7, MPEG-21, SMPTE, EXIF) tem sido propostos como uma solução para combinar conteúdo de mídia

[15]http://musicontology.com/

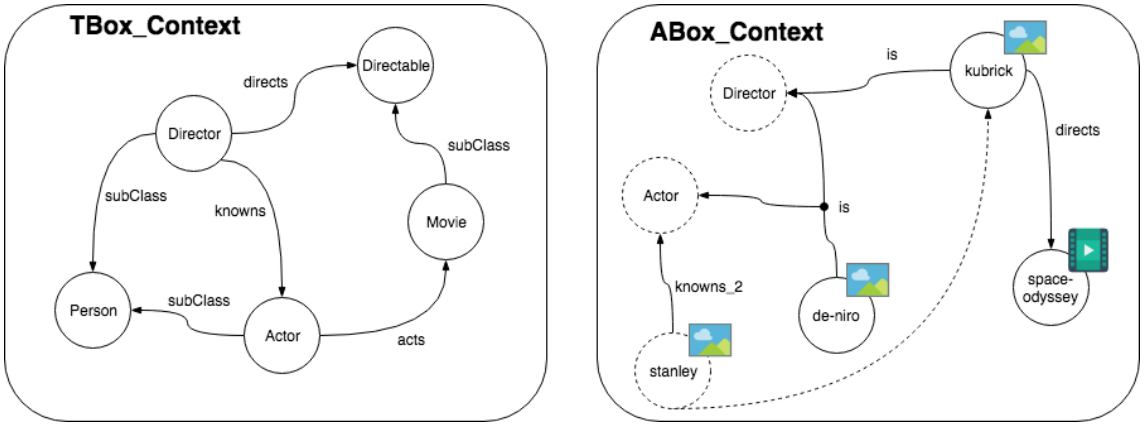


Figura 1.8. Uma base Hyperknowledge correspondente à KB de fatos sobre filmes.

com descrição semântica, mas esses padrões tendem a focar apenas em características de baixo nível do conteúdo multimídia, como codecs, taxas de bits, espaços de cores, etc.

A comunidade de representação de conhecimento, por outro lado, tem se dedicado principalmente ao desenvolvimento de modelos para representação de fatos do mundo real e de métodos para consulta e inferência de fatos a partir desses modelos. As aplicações do domínio de representação de conhecimento tendem a assumir que as bases de conhecimento são compostas apenas por fatos e normalmente desconsideram dados multimídia relacionados a esses fatos. Por exemplo, podemos usar RDF e OWL para codificar fatos em uma base de conhecimento, mas os conteúdos multimídia descritos por esses fatos são normalmente mantido em bases separadas, manipuladas via ferramentas e sistemas distintos.

O objetivo do Hyperknowledge é prover um *framework* que unifique hipermídia e representação de conhecimento e que, com isso, torne possível o desenvolvimento de aplicações multimídia *semantic-aware* avançadas e bases de conhecimento *multimedia-aware*.

1.4.1. Hyperknowledge

As principais entidades do modelo Hyperknowledge, herdadas do NCM, são os nós atômicos, composições, elos e conectores. Vamos utilizar um exemplo para introduzir essas entidades. A Figura 1.8 apresenta uma possível representação em Hyperknowledge de uma parte da KB de fatos sobre filmes apresentada no fim da Seção 1.2. Para simplificar, vamos adotar a hipótese de mundo fechado, isto é, vamos assumir que os fatos não representados na figura são falsos. (Discutimos essa hipótese e a alternativa mais geral, hipótese de mundo aberto, na Seção 1.2.3.)

Um *contexto* Hyperknowledge é um elemento contêiner utilizado para agrupar elementos relacionados (possivelmente outros contextos). A caracterização do que são elementos relacionados e a decisão de agrupá-los em contextos é uma decisão de modelagem que depende da aplicação. Na Figura 1.8, os contextos são representados pelos retângulos de bordas arredondadas. Há dois contextos na figura. O primeiro, `TBox_Context`,

contém os elementos e relacionamentos que correspondem à TBox da KB de fatos sobre filmes. O outro contexto, `ABox_Context`, contém os elementos e relacionamentos que correspondem à ABox.

Os círculos rotulados na Figura 1.8 denotam nós atômicos. Esses podem representar tanto conceitos quando conteúdo multimídia. Aqui estamos usando o termo "conceito" no mesmo sentido utilizado em DL. Isto é, um *conceito* (ou classe na terminologia OWL) representa uma característica abstrata de certos indivíduos. Na Figura 1.8, todos os nós da composição `TBox_Context`, a saber, `Pessoa`, `Diretor`, `Ator`, `Filme` e `Dirigivel`, são nós de conceito.

O outro tipo de nó atômico que ocorre na Figura 1.8 são os nós de mídia. Um *nó de mídia* é um nó contendo algum conteúdo multimídia, por exemplo, uma imagem, áudio, vídeo, texto, etc. Na figura, os nós de mídia são os nós `de-niro`, `kubrick` e `space-odyssey` que ocorrem no contexto `ABox_Context`. Os ícones à direita desses nós indicam que `de-niro` e `kubrick` são imagens e que `space-odyssey` é um vídeo.

Os nós de borda tracejada na Figura 1.8 não são nem nós de conceito nem nós de mídia; eles representam *nós de referência*. Isto é, os nós tracejados são nós que se referem a outros nós, permitindo que esses últimos possam ser reusados em outra situações (possivelmente, outros contextos). Por exemplo, os nós de conceito `Ator` e `Diretor`, definidos no contexto TBox, são reusados no contexto ABox. De maneira similar, o nó `stanley` do contexto ABox é uma referência para o nó `kubrick` do mesmo contexto. A ideia aqui é que qualquer coisa que opere sobre `stanley` estará operando na verdade sobre `kubrick`.

Esse tipo de indireção é útil em casos em que a mesma entidade está associada a diferentes nomes em contextos possivelmente diferentes. Por exemplo, o cantor e compositor Bob Dylan gravou algumas músicas sob o pseudônimo Blind Boy Grunt. Qualquer referência a Blind Boy Grunt é essencialmente uma referência a Bob Dylan. Mas a distinção entre esses dois pode ser importante em algumas aplicações.

De volta a Figura 1.8, as setas denotam elos que representam relacionamentos entre nós. Cada elo está associado a um determinado conector (não representado na figura) que especifica o tipo do elo. Um conector define uma relação abstrata e associa a essa relação um rótulo e um conjunto de restrições. Um elo podem ser visto como uma instância de seu conector, isto é, como uma realização concreta da relação abstrata definida pelo seu conector.

Na figura, o elo do contexto TBox entre `Ator` e `Pessoa` com rótulo "subClass" expressa o fato de que toda instância de `Ator` é também uma instância de `Pessoa`. Em outras palavras, esse elo codifica o axioma de TBox $\texttt{Ator} \sqsubseteq \texttt{Pessoa}$. De maneira similar, o elo "dirige" entre `Diretor` e `Dirigivel` codifica o axioma de TBox $\texttt{Diretor} \sqsubseteq \exists \texttt{dirige}.\top$.

Diferentemente de arestas de grafos RDF, que são necessariamente binárias, os elos de Hyperknowledge podem conectar mais de duas entidades. Por exemplo, na Figura 1.8 o elo "is" no contexto ABox conecta `de-niro` a ambos `Ator` e `Diretor`, e dessa forma codifica o axioma de ABox $\texttt{Ator} \sqcap \texttt{Diretor}(\texttt{de-niro})$.

O modelo Hyperknowledge possui muitas outras funcionalidades que facilitam

a manipulação de descrições semânticas em conjunto com conteúdo multimídia. Por exemplo, o modelo suporta a noção de âncoras que delimitam partes de nós de mídia [Fiorini et al. 2019]. Âncoras podem ser tanto origem quanto destino de elos e também pode ser designadas de forma independente por nós de referência. Usando a noção de âncoras, por exemplo, dada uma base Hyperknowledge (HKBase) é possível obter fragmentos específicos de nós de mídia que satisfaçam uma dada consulta semântica. Tais consultas podem ser definidas numa linguagem de consulta declarativa chamada HyQL (*Hyperknowledge Query Language*) que possui operadores nativos para comparação espacial [Moreno et al. 2018] e temporal [Moreno et al. 2018] de fragmentos.

Vamos agora discutir as ferramentas que permitem utilizar o modelo Hyperknowledge na prática.

1.4.2. Hyperknowledge na prática

Existem duas ferramentas principais para se desenvolver aplicações baseadas em Hyperknowledge: *Hyperknowledge Base* (HKBase) e *Knowledge Explorer System* (KES).

HKBase A *HKBase* é um banco de dados híbrido baseado em Hyperknowledge. Por híbrido, queremos dizer que a base armazena tanto objetos de mídia quanto fatos de um determinado domínio. Além disso, a HKBase foi especificamente projetada para utilizar o modelo Hyperknowledge como modelo de dados interno e API externa. Por razões de compatibilidade, a HKBase também suporta a importação e exportação de RDF e OWL. No caso da importação, os fatos são convertidos para Hyperknowledge antes de serem armazenados no banco de dados subjacente. A Figura 1.9 apresenta a arquitetura geral da HKBase.

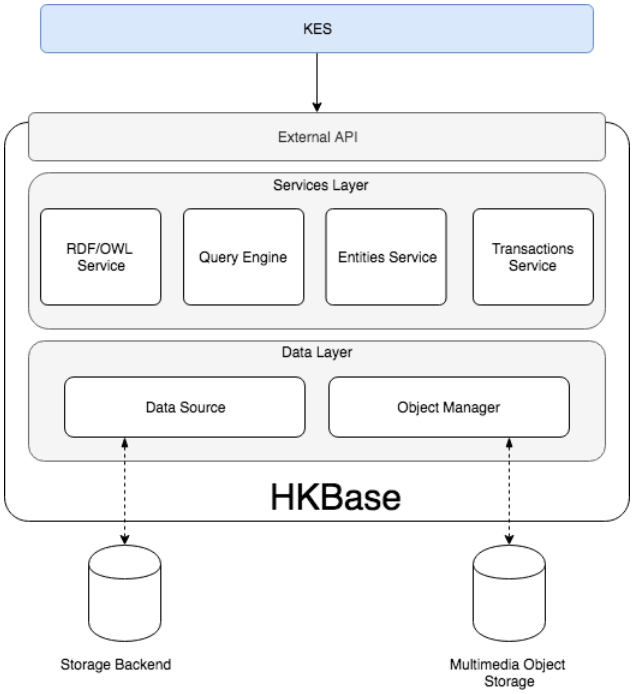


Figura 1.9. Arquitetura geral da HKBase.

A arquitetura da HKBase consiste de três camadas. A primeira delas é a API externa que expõe uma API REST para aplicações. A camada seguinte é a camada de serviços que implementa as funcionalidades principais da HKBase. Nessa camada, está o serviço de RDF/OWL que trata da conversão desses formatos de e para Hyperknowledge. O serviço de *query* trata do processamento de consultas HyQL. O serviço de entidades trata da criação, recuperação, atualização e remoção de entidades Hyperknowledge. E o serviço de transação gerencia requisições que devem executar de forma transacional.

A terceira e última camada da HKBase é a camada de dados que gerencia o acesso de baixo nível às bases de dados subjacentes. O componente *data source* é responsável pela persistência de fatos estruturados. E o componente *object manager* trata da persistência de objetos multimídia.

KES O *KES* (*Knowledge Explorer System*) [Moreno et al. 2018] é uma aplicação construída sobre a HKBase. Usando o KES, os usuários conseguem visualizar e editar o conteúdo armazenado numa base Hyperknowledge. O KES é um sistema colaborativo, isto é, múltiplos usuários podem editar simultaneamente a mesma base. Além disso, o KES possui funcionalidades para importação e manipulação de arquivos RDF e OWL.

A Figura 1.10 apresenta uma captura de tela do KES. O *canvas* principal mostra uma representação baseada em grafos do conteúdo de uma base Hyperknowledge. Além das entidades Hyperknowledge, o KES permite que o conteúdo multimídia seja manipulado diretamente. Por exemplo, a Figura 1.10 mostra que o usuário executou a consulta: "Select Goal where Goal by Neymar". (A base Hyperknowledge, nesse caso, contém fatos sobre o domínio de futebol.) Essa consulta retorna todos os gols marcados pelo jogador chamado "Neymar". Observe que o resultado apresentado pelo KES contém todas as entidades Hyperknowledge que satisfazem a consulta, isto é, todos os nós de mídia que são instâncias de `Goal` e que possuem ao menos um elo que expressa que o gol foi marcado por `Neymar`. O conteúdo desses objetos de mídia é apresentado à direta da tela.

1.5. Sugestões de Leitura

Neste capítulo, apresentamos uma visão geral de IA simbólica e discutimos a sua aplicação à multimídia. Como qualquer introdução, esta é necessariamente incompleta, especialmente por conta da abrangência dos tópicos abordados. Concluímos o capítulo com algumas sugestões de leitura que complementam aquelas apresentadas ao longo do texto.

Logica Uma introdução clássica à lógica matemática em geral é [Enderton 2001]. Já uma referência primária para lógica de descrição é [Baader et al. 2007]. Uma introdução breve porém excelente a DL a partir de $\mathcal{SROIQ}$ é apresentada em [Rudolph 2011]. Para lógicas modais, em especial lógicas temporal, um bom ponto de partida é [Goldblatt 1992]. Para uma discussão de DL e multimídia, recomendamos [Sikos 2017].

Web Semântica Uma discussão seminal da visão da Web Semântica pode ser encontrada em [Berners-Lee et al. 2001]. Uma visão geral dos seus princípios e tecnologias é apresentada em [Hitzler et al. 2010]. Especificamente sobre OWL, um bom ponto de partida é [Hitzler et al. 2014]. E para um tratamento abrangente de ontologias, veja [Staab and Studer 2009].

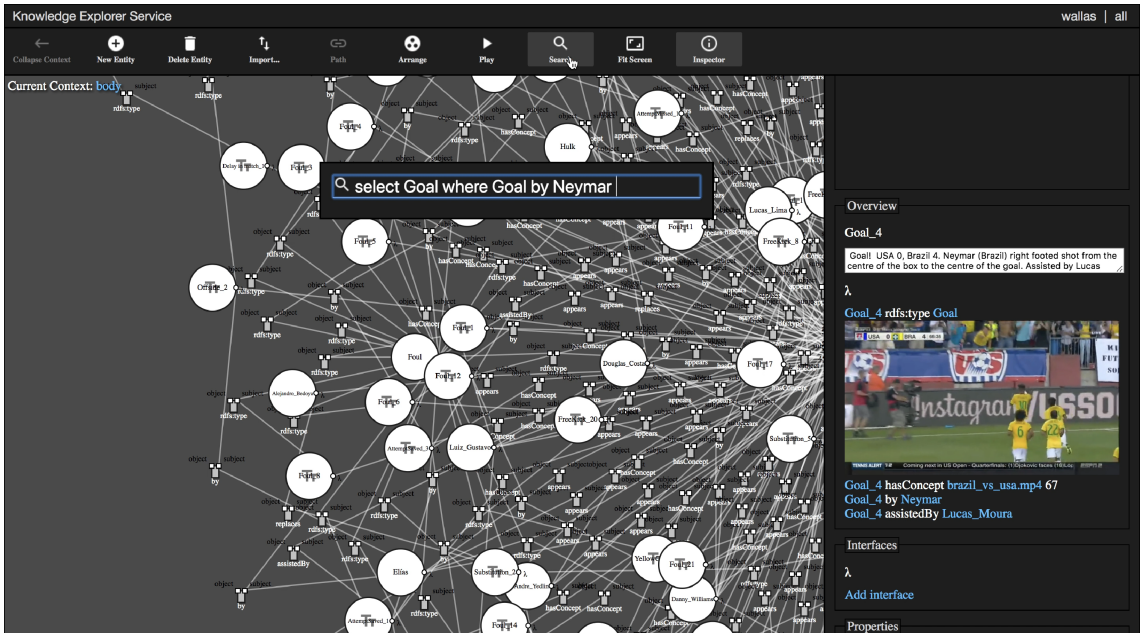


Figura 1.10. Captura de tela do KES.

Hyperknowledge A referência principal sobre Hyperknowledge é [Moreno et al. 2017]. Para uma discussão de aplicações de Hyperknowledge envolvendo interpretação de documentos e raciocínio temporal veja [Moreno et al. 2018] e [Moreno et al. 2018].

Referências

[Angles 2012] Angles, R. (2012). A comparison of current graph database models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177. IEEE.

[Baader et al. 2007] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2007). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition.

[Baader and Lippmann 2014] Baader, F. and Lippmann, M. (2014). Runtime verification using the temporal description logic $\mathcal{ALC}$-LTL revisited. *J. Appl. Log.*, 12(4):584–613.

[Berners-Lee et al. 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, pages 96—101.

[Borgwardt and Peñaloza 2017] Borgwardt, S. and Peñaloza, R. (2017). Fuzzy description logics – a survey. In *Scalable Uncertainty Management*, pages 31–45. Springer.

[Carothers and Prudhommeaux 2014] Carothers, G. and Prudhommeaux, E. (2014). RDF 1.1 Turtle. Recommendation, W3C. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[Cristiani and Gabrielli 2011] Cristiani, M. and Gabrielli, N. (2011). Practical issues of description logics for spatial reasoning. In *AAAI Spring Symposium, Stanford University, Stanford, 21–23 March, 2011*.

[Enderton 2001] Enderton, H. B. (2001). *A Mathematical Introduction to Logic*. Academic Press, 2nd edition.

[Fiorini et al. 2019] Fiorini, S. R., dos Santos, W. S., Mesquita, R. C., Lima, G. F., and Moreno, M. F. (2019). General fragment model for information artifacts.

[Goldblatt 1992] Goldblatt, R. (1992). *Logics of Time and Computation*. CSLI, 2nd edition.

[Grosof et al. 2003] Grosof, B. N., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: Combining logic programs with description logic. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 48–57. ACM.

[Harris and Seaborne 2013] Harris, S. and Seaborne, A. (2013). SPARQL 1.1 query language. Recommendation, W3C. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[Hayes and Patel-Schneider 2014] Hayes, P. and Patel-Schneider, P. (2014). RDF 1.1 semantics. Recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/.

[Hitzler et al. 2014] Hitzler, P., Krötzsch, M., adn Peter F. Patel-Schneider, B. P., and Rudolph, S. (2014). OWL 2 Web Ontology Language primer (second edition). Recommendation, W3C. http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[Hitzler et al. 2010] Hitzler, P., Krötzsch, M., and Rudolph, S. (2010). *Foundations of Semantic Web Technologies*. Chapman & Hall.

[Horrocks et al. 2006] Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible $\mathcal{SROIQ}$. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'06, pages 57–67. AAAI Press.

[Hudelot et al. 2015] Hudelot, C., , Atif, J., and Bloch, I. (2015). $\mathcal{ALC}$(F): A new description logic for spatial reasoning in images. In Agapito, L., Bronstein, M. M., and Rother, C., editors, *Computer Vision - ECCV 2014 Workshops*, pages 370–384. Springer.

[Krötzsch et al. 2008] Krötzsch, M., Rudolph, S., and Hitzler, P. (2008). Description logic rules. In *Proceedings of the 18th European Conference on Artificial Intelligence, Patras, February 2008*, pages 80–84. IOS Press.

[Milea et al. 2012] Milea, V., Frasincar, F., and Kaymak, U. (2012). tOWL: A temporal web ontology language. *IEEE Trans. Syst; Man, and Cybern., Part B (Cybernetics)*, 42(1):268–281.

[Moreno et al. 2018] Moreno, M., Santos, R., Mozart, R., Santos, W., and Cerqueira, R. (2018). Assisting seismic image interpretations with hyperknowledge. In *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*, pages 48–51.

[Moreno et al. 2018] Moreno, M., Santos, R., Santos, W., Brandão, R., Carrion, P., and Cerqueira, R. (2018). Handling hyperknowledge representations through an interactive visual approach. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 139–146.

[Moreno et al. 2018] Moreno, M., Santos, R., Santos, W., Silva, R., and Cerqueira, R. (2018). Knowledge bases enrichment with temporal reasoning using hyperknowledge. In *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 125–128.

[Moreno et al. 2018] Moreno, M., Schirmer, L., Bayser, M., Brandão, R., and Cerqueira, R. (2018). Understanding documents with hyperknowledge specifications. In *Proceedings of the ACM Symposium on Document Engineering 2018*, DocEng '18, pages 41:1–41:4. ACM.

[Moreno et al. 2017] Moreno, M. F., Brandao, R., and Cerqueira, R. (2017). Extending hypermedia conceptual models to support hyperknowledge specifications. *Int. J. Semant. Comput.*, 11(1):43–64.

[OWL@Machester 2019] OWL@Machester (2019). List of reasoners. http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/.

[Pfeiffer et al. 2012] Pfeiffer, S., Mannens, E., Troncy, R., and Deursen, D. V. (2012). Media Fragments URI 1.0 (basic). W3C recommendation, W3C. http://www.w3.org/TR/2012/REC-media-frags-20120925/.

[Randell et al. 1992] Randell, D. A., Cui, Z., and Cohn, A. G. (1992). A spatial logic based on regions and connection. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, KR '92, pages 165–176. Morgan Kaufmann.

[Robinson et al. 2013] Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases*. O'Reilly.

[Rudolph 2011] Rudolph, S. (2011). *Foundations of Description Logics*, pages 76–136. Springer.

[Schmidt-Schauß and Smolka 1991] Schmidt-Schauß, M. and Smolka, G. (1991). Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26.

[Sikos 2017] Sikos, L. F. (2017). *Description Logics in Multimedia Reasoning*. Springer.

[Soares and Rodrigues 2005] Soares, L. F. G. and Rodrigues, R. F. (2005). Nested Context Model 3.0 part 1: NCM core. Monographs in computer science, Informatics Department, PUC-Rio, Rio de Janeiro, Brazil.

[Staab and Studer 2009] Staab, S. and Studer, R., editors (2009). *Handbook on Ontologies*. Springer-Verlag, 2nd edition.

[W3C OWL WG 2012] W3C OWL WG (2012). OWL 2 Web Ontology Language document overview (second edition). Recommendation, W3C. http://www.w3.org/TR/2012/REC-owl2-overview-20121211/.

[W3C Semantic Web Wiki 2019] W3C Semantic Web Wiki (2019). Semantic web development tools. https://www.w3.org/2001/sw/wiki/Tools.

[Wang et al. 2014] Wang, Y., Chang, L., Li, F., and Gu, T. (2014). Verification of branch-time property based on dynamic description logic. In *Intelligent Information Processing VII*, pages 161–170. Springer.

[Wessel 2003] Wessel, M. (2003). Qualitative spatial reasoning with the $\mathcal{ALCI}_{\mathrm{RCC}}$ family: First results and unanswered questions. Memo 324/03, University of Hamburg. https://kogs-www.informatik.uni-hamburg.de/publikationen/pub-wessel/report7.pdf.

[Wood et al. 2014] Wood, D., Lanthaler, M., and Cyganiak, R. (2014). RDF 1.1 concepts and abstract syntax. Recommendation, W3C. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[Zolin 2019] Zolin, E. (2019). Complexity of reasoning in description logics. http://www.cs.man.ac.uk/~ezolin/dl/.

Capítulo 2

Introdução ao processamento de fluxo de dados: uma abordagem orientada a eventos complexos

Marcos Roriz Junior[1], Álan L. Vasconcelos[2], Fernando B. V. Magalhães[2], Sérgio Colcher[2], Markus Endler[2]

[1]Faculdade de Ciências e Tecnologia – Engenharia de Transportes, UFG

[2]Departamento de Informática, PUC-Rio

marcosroriz@ufg.br, alan@telemidia.puc-rio.br, fmagalhaes@inf.puc-rio.br

colcher@inf.puc-rio.br, endler@inf.puc-rio.br

Abstract

Most of information technologies courses are focused on exposing the traditional static data (e.g., SGBD) processing model and lack on presenting the data stream model. Such model can be applied to fields such as Internet of Things, busyness and finances, logistics and smart cities) of the industry. Complex Event Processing consists of a programming approach to handle Data Stream Processing. It provide primitives to process and detect the occurrence of patterns in data streams. This chapter has the objective of presenting the complex event processing (CEP) programming model as a means of dealing with the specificities of data flows.

Resumo

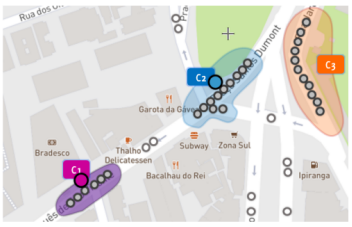
A maioria dos cursos de tecnologias da informação está focada em expor o modelo de processamento tradicional de dados estáticos (e.g., SGBD) e a falta de apresentação do modelo de fluxo de dados. Esse modelo pode ser aplicado a áreas como Internet das Coisas, ocupação e finanças, logística e cidades inteligentes) da indústria. O processamento complexo de eventos consiste em uma abordagem de programação para lidar com o processamento de fluxo de dados. Ele fornece primitivas para processar e detectar a ocorrência de padrões nos fluxos de dados. Este capítulo tem o objetivo de apresentar o modelo de programação de processamento de eventos complexos (CEP) como um meio de lidar com as especificidades dos fluxos de dados.

2.1. Introdução

Avanços técnicos na arquitetura de computadores e computação móvel têm possibilitado não somente a popularização de dispositivos portáteis e embarcados, com conectividade com a Internet e capacidade de sensoriamento e atuação, mas também a utilização dos mesmos para a construção de aplicações de Cidades Inteligentes (*Smart City*) e Internet das Coisas (*Internet of Things*) [van der Zee and Scholten 2013].

Coletivamente, esses dispositivos podem produzir grandes fluxos de dados, que ao serem interconectados possibilitem gerar uma inteligência competitiva ou detectar situações de interesse [Zheng et al. 2014, McAfee and Brynjolfsson 2012]. Por exemplo, pode-se utilizar sensores para controlar plantas na indústria (Figura 2.1a), detectar engarrafamento de veículos a partir de dados de localização (Figura 2.1b) e monitorar a reputação de uma empesa em redes sociais a partir de mensagens dos usuários. Desta forma, cada vez mais é importante explorar técnicas que possibilitem não somente a extração de informação, mas também a reação a anomalias em fluxo de dados.

(a) Sensor de temperatura fixo[1]



(b) Engarrafamento [Roriz Junior 2017]

Figura 2.1: Cenários de uso de aplicações orientadas a fluxo de dados.

Para tais tarefas existem várias tecnologias focadas na análise de fluxos de dados, as quais vão desde bancos de dados ativos até o processamento de eventos complexos. O processamento de eventos complexos [Luckham 2001, Etzion and Niblett 2010], também conhecido como CEP (Complex Event Processing), consiste em um modelo de programação que fornece primitivas para processar e derivar eventos (informações) mais complexas a partir de fluxos de dados.

Em uma indústria, por exemplo, o evento de incêndio pode ser gerado a partir de eventos de aumento de temperatura e detecção de Fumaça. Em *Smart Cities*, por exemplo, o evento de engarrafamento pode ser detectado a partir de eventos mais simples que indicam que vários veículos não estão se movendo em um determinado período de tempo.

Neste contexto, este capítulo tem o objetivo apresentar e exemplificar o modelo de programação de processamento de eventos complexos (CEP) como meio de lidar com as especificidades de fluxos de dados. O capítulo está organizado da seguinte maneira. A Seção 2.2 apresenta os conceitos e problemas fundamentais de processamento de fluxo de dados e o modelo CEP. Já a Seção 2.3 apresenta a tecnologia Esper e a sua linguagem de regras EPL como meio de tratar os fluxo de dados. A Seção 2.4 exemplifica um caso de uso da aplicação de CEP para tratamento de dados de aplicações IoT, enquanto que a Seção 2.5 apresenta um outro caso de uso de aplicação de CEP como meio de processar

[1] `https://www.electronicdesign.com/iot/wireless-sensor-networking-industrial-iot`

dados de aplicações para *Smart Cities*. Por fim, a Seção 2.6 apresenta nossas considerações finais.

2.2. Fundamentação teórica e introdução a CEP

O Processamento de Eventos Complexos (*Complex Event Processing* - CEP) é um paradigma de programação voltado a tratar, analisar e reagir a um fluxo de dados (encapsulados como eventos) em aproximadamente tempo real, isto é, em poucos segundos [Luckham 2001, Kudyba 2014, Flouris et al. 2016]. Por exemplo, utilizando os conceitos de CEP é possível detectar, em aproximadamente tempo real (poucos segundos), engarrafamentos e acidentes no trânsito [Dunkel et al. 2011, Roriz Junior et al. 2019], padrões em redes sociais [Cameron et al. 2012, Yadranjiaghdam et al. 2017] e anomalias em fluxos de áudios de usuários [Maison et al. 2013].

Para possibilitar a detecção em tempo real, ao contrário dos sistemas gerenciadores de banco de dados, nos quais os dados são primeiramente armazenados e depois consultados posteriormente, o CEP inverte esta lógica armazenando as consultas continuamente e executando os dados diretamente nelas. Precisamente, em vez de armazenar os dados, o CEP implementa as consultas em um estado contínuo, sempre funcionando, com intuito de analisar e reagir aos eventos ao passo que os mesmos apareçam no fluxo de dados. Cada consulta implementa continuamente uma ou mais primitivas de processamento de fluxo de evento em tempo real, como *filtro*, *sequência* e *negação* [Cugola and Margara 2012, Suhothayan et al. 2011].

Os fluxos de eventos são as principais fontes de entrada de uma consulta contínua do CEP. No CEP, os eventos são criados por meio de produtores, que são entidades (*e.g.*, sensores, usuários) que encapsulam os dados do domínio da aplicação, como localização, imagem, texto e *clicks*. Estes eventos são denominados de eventos simples (*raw events*), visto que ainda não foram processados. Os eventos são caracterizados por um tipo, a hora que foi gerado (*timestamp*) e uma carga de dados (*payload*) [Luckham 2001]. O tipo do evento define o esquema do *payload*, isto é, o nome e o domínio dos atributos correspondentes que representam a ocorrência do evento em questão. Por exemplo, podemos definir o tipo de evento `LocationUpdate` para representar a atualização da posição de um objeto em movimento usando o seguinte esquema de carga útil: *id*, latitude, longitude e velocidade. Já o esquema de eventos originados em uma rede social pode conter o nome do usuário, o texto da mensagem transmitida, imagens anexadas, localização e tempo de envio.

Um fluxo de dados de eventos é a sequência resultante dos eventos criados e transmitidos pelos produtores [Luckham 2001, Etzion and Niblett 2010]. Neste sentido, os eventos em um fluxo de dados seguem o mesmo tipo e sua ordem é baseada no tempo de origem (*timestamp*) do mesmo. Desta maneira, uma consulta contínua pode utilizar as primitivas de processamento em tempo real, como *filtro*, *divisão* e *sequência*, para reagir e processar o fluxo de eventos à medida que este chegam. Por exemplo, podemos utilizar a primitiva de *filtro* para reter o fluxo de evento `LocationUpdate` com intuito de descobrir os veículos que se encontram próximos de um determinado ponto de interesse. Semelhantemente, uma empresa área pode utilizar a primitiva *dividir* para continuamente extrair e analisar as palavras-chave de mensagens de uma rede social que contenham o

nome da companhia.

As consultas contínuas podem usar várias primitivas em tempo real para reagir, processar e derivar outros eventos (complexos) de nível superior a partir de fluxos de eventos puros. Os eventos de saída das consultas contínuas são dito complexos, pois representam eventos processos e contém informações derivadas [Luckham 2001]. Ambos, eventos simples e complexos, podem ser usados como parte da definição de um outro evento complexo. Precisamente, é possível criar hierarquias de eventos, nas quais eventos intermediários podem ser usados para definir outros eventos complexos de nível superior. Por exemplo, um evento complexo `TrafficJam` pode ser criado pela combinação de vários eventos `LocationUpdate` na mesma área e período. Outro exemplo, podemos construir um evento complexo `PossibleDelay` a partir da composição de múltiplas mensagens de uma rede social que contenham palavras-chave associadas ao atraso de vôos de uma companhia em um pequeno espaço de tempo.

Cada consulta contínua é executada por um Agente de Processamento de Eventos do CEP, também conhecido por *Event Processing Agent* (EPA), que podem ser interligados para processarem os eventos [Suhothayan et al. 2011, Cugola and Margara 2012]. Especificamente, um estágio EPA continuamente faz as seguintes tarefas: reage aos eventos recebidos; analisa e manipula-os; e gera eventos complexos (derivados) para consumidores de eventos, podendo ser outros estágios EPAs ou aplicações finais. Ao interconectar os EPAs é possível construir uma rede de processamento de eventos (*Event Processing Network* - EPN), um *workflow* topológico que analisa o fluxo de eventos de entrada à medida que passa. A estrutura topológica da EPN, um gráfico direcionado, facilita a distribuição do processamento de EPAs para diferentes máquinas. Cada EPA nessa rede é responsável por receber eventos, processar a consulta contínua e, se houver uma etapa derivativa, enviar os eventos complexos aos próximos estágios.

Para exemplificar esses conceitos considere a EPN ilustrada na Figura 2.2 que visa detectar anomalias no trânsito a partir da localização dos veículos e de uma rede social. No cenário descrito, os EPAs de borda da EPN recebem continuamente a localização de cada ônibus da cidade, junto com mensagens dos usuários da rede social. Se os eventos

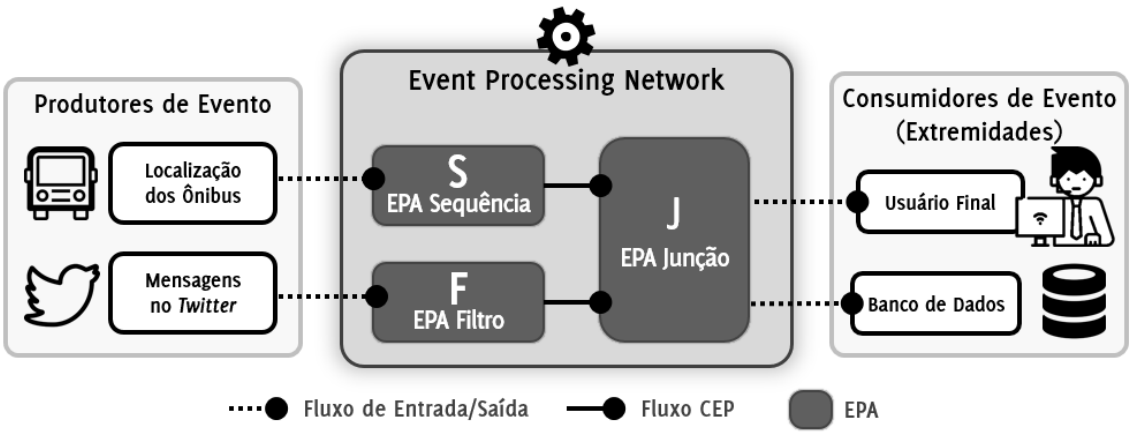


Figura 2.2: Exemplo de EPN para detecção de anomalias no trânsito.

analisados pelos EPAs satisfizerem a lógica da primitiva de processamento, o EPA emitirá um evento complexo como saída. Tais eventos são entregues a EPAs interessados (conectados) para serem processados em sequência.

Por exemplo, a unidade de processamento `S` identifica se os ônibus estão parados (ou andando devagar) verificando se a distância percorrida pelo mesmo, na sequência de eventos de localização enviados, for menor que um limiar (*e.g.*, 250 metros). Caso positivo, o EPA deriva um evento complexo agregando o identificador do veículo e a sequência de posições. Paralelamente, o EPA `F` filtra as mensagens da rede social que contenham palavras-chave relacionados a anomalias no trânsito, tais como acidente e engarrafamento, que por sua vez são adicionalmente processadas se possuírem um valor maior que um determinado limiar dentro de um período (*e.g.*, 50 mensagens nos últimos 5 minutos). Finalmente, `J` correlaciona os dados de ônibus que não se moveram com o grupo de mensagens com palavras-chave que ocorreram na mesma região espacial. Como saída, o EPA gera um evento complexo de possível anomalia de trânsito, que consequentemente, é repassado para entidades interessadas na borda da EPN, como o usuário final ou banco de dados.

2.2.1. Motor de Inferência CEP e Linguagens de Consultas Contínuas

Um motor de inferência (*engine*) CEP permite instanciar os conceitos apresentados. Ele fornece operações para definir os tipos de eventos (esquema e carga útil) e primitivas em tempo real para expressar as consultas contínuas (EPAs) e para interconectá-las (EPN). Há várias implementações de moto de inferência CEP, por exemplo, Esper [EsperTech 2019], Siddhi [Suhothayan et al. 2011], STREAM [Arasu et al. 2003], Microsoft StreamInsight [Ali et al. 2011], Sase+ [Yanlei Diao Neil Immerman and Gyllstrom 2007], Apache Flink [Carbone et al. 2015] e Red Hat Drools Fusion [Bali 2009].

As principais diferenças entre as *engines* são os construtores e semântica das linguagens fornecidas aos desenvolvedores para definir os eventos e primitivas de processamento em tempo real. Em geral, as semânticas das linguagens CEP podem ser divididas em duas categorias: orientada a fluxo e orientado a regras [Etzion and Niblett 2010, Cugola and Margara 2012, Kudyba 2014, Zhao et al. 2017].

As linguagens de processamento de eventos complexos orientadas a fluxos tipicamente são baseadas em um design formal da linguagem de consulta contínua, também denominada *Continuous Query Language* (CQL) [Arasu et al. 2005]. A CQL é uma extensão da linguagem *Structured Query Language* (SQL) com operadores e primitivas específicas para tratamento e análise de fluxo de dados.

Para ilustrar a expressividade do modelo de linguagem orientada a fluxo do CEP, considere a consulta contínua escrita na linguagem EPL (*Event Processing Language*) do motor de inferência CEP Esper [EsperTech 2019] no Código 2.1. A EPL descreve um EPA que detecta um evento complexo `AlertMessage` quando existir mais de 50 mensagens em uma janela de tempo de 60 segundos contendo as palavras-chave `acidente` ou `engarrafamento` e que estejam dentro do intervalo delimitado (*e.g.*, limites do município).

Para implementar esse EPA, o motor de inferência CEP realiza os seguintes passos. Ao receber um evento do fluxo `TwitterMessages`, a consulta utiliza o *timestamp* do mesmo para deslizar a janela de tempo e recuperar todos os eventos recebidos nos últimos

60 segundos. O resultado desta operação, um subconjunto do fluxo `TwitterMessages`, é transformado em uma relação temporária. Utilizando essa relação, a consulta contínua filtra os eventos cujos valores de carga útil possuem pelo menos uma das palavras-chave pré-determinada e estão dentro dos intervalos de latitude e longitude especificado. Em seguida, a consulta contínua conta o número de tuplas filtradas. Caso este número seja superior a 50 mensagens, será gerado um novo evento complexo `AlertMessages` contendo a projeção (seleção) das palavras-chave e localização dos eventos resultantes do filtro.

```
INSERT INTO AlertMessages
SELECT keyword, lat, lng
  FROM TwitterMessages#TIME(60 s)
 WHERE keyword IN ("acidente", "engarrafamento")
   AND (lat > −21 AND lat < −23 AND lng > −42 AND lng < −43)
HAVING COUNT(*) > 50;
```

Código 2.1: Exemplo de EPA escrito na linguagem EPL da *engine* Esper.

Observe que as consultas contínuas de uma EPA, escritas em uma linguagem orientada a fluxo, podem implementar uma ou mais primitivas em tempo real. No caso ilustrado, a consulta EPL implementa as primitivas *filtro*, *agregação* e *projeção*. Além disso, como dito, observe que a sintaxe da linguagem EPL (baseada em CQL) é semelhante a uma consulta SQL. Uma das diferenças é que a saída de uma consulta contínua produz um fluxo de eventos que, consequentemente, pode ser consumido e processado por outras consultas contínuas. Nesse caso, o EPA produz continuamente eventos `AlertMessages` que podem ser analisados por outras consultas. Outra diferença frente ao SQL padrão é a capacidade de usar janelas de tempo para analisar e correlacionar eventos recebidos em subconjuntos do fluxo. Abordaremos o conceito de janelas de tempo em uma subseção posterior deste capítulo.

Ao contrário das linguagens semelhantes a CQL, *engines* CEP que empregam linguagens orientadas a regras são baseados em cláusulas de inferência evento-condição-ação (ECA) [Wu et al. 2006, Anicic et al. 2010, Zhao et al. 2017]. Regras ECA separam a manipulação de eventos, a verificação de condições e a ação em diferentes cláusulas. Primeiro, a consulta contínua da EPA é acionada quando o evento especificado acontece. Em seguida, uma restrição ou condição é verificada. Finalmente, se a condição for atendida, as regras executam a cláusula de ação.

Para exemplificar uma linguagem orientada a regra, considere a definição do EPA ilustrado no Código 2.2. É uma consulta contínua semelhante ao exemplo anterior, descrito em EPL, mas agora escrita na linguagem de regras da *engine* CEP da plataforma Red Hat Drools Fusion [Bali 2009]. A regra é acionada ao chegar um novo evento do fluxo `TwitterMessages` que possua pelo menos uma das palavras-chave mencionadas e esteja dentro da área de interesse. Em seguida, a *engine* retorna o acúmulo de mensagens do fluxo que situam no mesmo intervalo de localização e contenham as palavras-chave indicadas. Após essa etapa, é feito a contagem de mensagens (`$numMsgs`) que passaram

pelo respectivo filtro. Caso possua mais de 50 eventos a regra prossegue para a etapa de consequência (`THEN`). Em seguida realizamos a projeção dos atributos `keyword`, `lat` e `lng` dos eventos para criar a lista resultante `$listMsgs`. Por fim, utilizamos a lista de eventos filtrada na etapa anterior para construir e enviar o evento complexo `AlertMessages`.

```
RULE "Detectar possível anomalia no trânsito em uma dada região"
  WHEN
    TwitterMessages (keyword IN ("acidente", "engarrafamento")
                     AND (lat > -21 AND lat < -23)
                     AND (lng > -42 AND lng < -43))
    ACCUMULATE (
      $msgs = TwitterMessages (keyword IN ("acidente", "engarrafamento")
                               AND (lat > -21 AND lat < -23)
                               AND (lng > -42 AND lng < -43)
                               OVER WINDOW:TIME(60 s),
                               $numMsgs = COUNT($msgs);
                               $numMsgs > 50)
    )
  THEN
    $listMsgs = collectList($msgs.keyword, $msgs.lat, $msgs.lng);
    INSERT(AlertMessages($listMsgs))
END
```

Código 2.2: Exemplo de EPA escrito na linguagem EPL da *engine* Red Hat Drools.

Como dito, a principal diferença entre as semântica das linguagens CEP é a sintaxe fornecida para criar consultas contínuas. Precisamente, aquelas orientadas a fluxo são baseadas na CQL (uma extensão da SQL), enquanto as orientadas a regras usam cláusulas evento-condição-ação (ECA) para processar os eventos recebidos. Note que os dois modelos *podem* implementar as mesmas primitivas CEP. Por exemplo, tanto o Código 2.1, quanto o Código 2.2, implementam as primitivas de *filtro*, *agregação* e *projeção*.

No entanto, na prática, as *engines* CEP suportam um conjunto diferente de primitivas de processamento [Cugola and Margara 2012]. Em geral, os motores CEP que utilizam linguagens orientadas a fluxos se concentram no suporte às primitivas de *transformação*, tais como *filtrar*, *dividir* e *combinar*, enquanto as orientadas a regras são focadas em primitivas de detecção de padrão, como *sequência* e *negação* de padrões de eventos.

A Figura 2.3 ilustra uma classificação taxonômica das principais primitivas CEP, adaptadas de [Etzion and Niblett 2010, Cugola and Margara 2012], que são tipicamente suportadas em cada modelo semântico do CEP. Entretanto, ressaltamos que alguns motores de inferência CEP, tais como Esper [EsperTech 2019], Microsoft StreamInsight [Microsoft 2015] e Apache Flink [Carbone et al. 2015] oferecem suporte aos dois tipos semânticos para construção das consultas contínuas. Na subseção a seguir, definiremos as classes de primitivas e apresentaremos concisamente a função de cada uma delas.

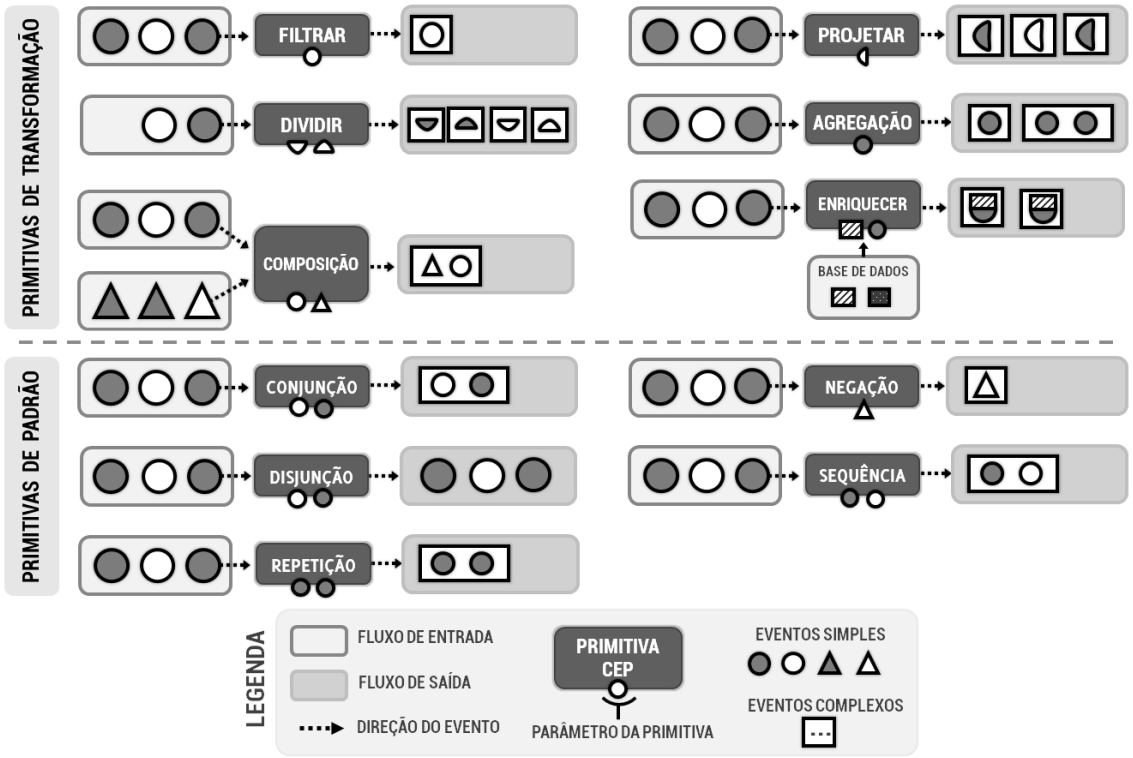


Figura 2.3: Taxonomia das primitivas CEP.

2.2.2. Primitivas CEP

As primitivas de transformação CEP são aquelas que filtram, modificam ou correlacionam os eventos do fluxo de dados [Luckham 2001, Anicic et al. 2010, Cugola and Margara 2012]. Elas podem converter o evento de entrada em um complexo usando os atributos do mesmo ou uma fonte de dados externa. Como visto ao longo do texto, a primitiva de *filtrar* possibilita reter ou passar um conjunto de eventos conforme um ou mais predicados. Já a primitiva de *dividir* possibilita transformar um evento em vários. Por exemplo, podemos utilizar a primitiva para separar uma mensagem de uma rede social em múltiplos eventos complexos com base nas palavras-chave da mesma.

Como na álgebra relacional, a primitiva *projetar* cria um evento complexo usando um subconjunto de seus atributos, enquanto a primitiva *enriquecer* utiliza uma fonte de dados externa (*e.g.*, tabela na memória ou em um banco de dados) para criar um evento derivado contendo novas informações ou modificações de atributos originais [Luckham 2001]. Por exemplo, um evento de ocorrência de engarrafamento pode ser enriquecido com pontos de interesses de um banco de dados para verificar se os mesmos se encontram próximo (*e.g.*, acidente próximo ao aeroporto).

Já as primitivas de agregação permitem combinar eventos de entrada em um único de saída, por exemplo, múltiplas mensagens de reclamação podem ser combinada em um evento de alerta. Além disso, essa primitiva pode empregar as funções clássicas de agregação do SQL, tais como média (*avg*), mínimo (*min*) e máximo (*max*). A primitiva de composição possibilita realizar a junção entre dois ou mais fluxos de eventos de entrada,

utilizando um critério, para derivar eventos complexos oriundos dessa correspondência. Como previamente exemplificado, podemos combinar os fluxos de eventos de localização de veículo com as mensagens de acidentes em redes sociais.

Por outro lado, as primitivas de detecção de padrões são baseadas em modelo de regras (*templates*) de evento [Wu et al. 2006, Cugola and Margara 2012]. Tais *templates* usam operadores lógicos – como *conjunção*, *disjunção*, *repetição*, *negação* e *sequência* – para definir o padrão que será utilizado pela consulta contínua [Anicic et al. 2010].

Por exemplo, a primitiva de *repetição* define um padrão de evento que detecta quando um determinado tipo de evento se repete pelo menos n vezes em uma determinada janela de tempo. Por exemplo, considere a primitiva de repetição com o seguinte padrão de evento E [n] [3 min]. Essa primitiva consome e processa continuamente eventos até encontrar pelo menos n eventos do tipo E dentro no período especificado. Podemos utilizar tal primitiva para detectar surtos de mensagens contendo palavras-chave semelhante em um pequeno espaço de tempo em uma rede social.

Em seguida, destacamos a primitiva *conjunção* que possibilita definir um padrão de evento que devem ocorrer em conjunto. Para tal, devemos utilizar o seguinte *template*: E_1 `AND` E_2 `AND` ... `AND` E_n. Esta primitiva continuamente analisa o fluxo a procura do padrão, sendo ativada quando encontrar os eventos E_1, E_2, ..., e E_n, no mesmo. Note que a definição não impõe uma ordem, isto é, a regra não depende da ordem de chegada da aparição dos eventos. Para ilustrar essa primitiva considere o seguinte *template*: *Atraso* [50] [30 min] `AND` *ProblemaAeroporto (nome = "Congonhas")*. A consulta contínua detectará o padrão quando observar pelo menos 50 eventos do tipo *Atraso* em meia hora e um evento que indica um problema no aeroporto de Congonhas. Como resultado, as primitivas de padrão criam um evento complexo contendo os elementos que dispararam a mesma.

Semelhante à primitiva de *conjunção*, a *disjunção* define um padrão de evento usando o operador lógico OU (`OR`), por exemplo, E_1 `OR` E_2 `OR` ... `OR` E_n. A *disjunção* é satisfeita quando a consulta contínua detecta pelo menos um dos eventos especificados no *template*.

O CEP também provê outros operadores de relações de padrão, como a *negação* e *sequência*. A primitiva de *negação* define um padrão que detecta a ausência de um determinado evento. Por exemplo, o padrão $\neg E$ [10 min] significa que a consulta contínua deve ser acionada se nenhum evento E for detectado dentro de 10 minutos. Para exemplificar o funcionamento dessa primitiva, considere o seguinte padrão: $\neg$*ProblemaAeroporto (nome = "Congonhas")* [30 min]. Essa consulta contínua será acionada se não receber mais nenhum evento *ProblemaAeroporto*, cuja carga útil se refere ao aeroporto de Congonhas, em um período de 30 minutos.

A primitiva de *sequência* define um *template* para capturar o relacionamento entre os eventos, especificamente a ordem de chegada e tipo dos mesmos. Por exemplo, uma consulta contínua contendo o padrão $E_1 \rightarrow E_2 \rightarrow E_1 \rightarrow E_3$ é acionada quando a mesma recebe os seguintes eventos em ordem E_1, E_2, outro E_1 e E_3. Para ilustrar essa primitiva, considere um exemplo onde pretendemos detectar o surgimento e término de engarrafamentos no trânsito a partir de mensagens de uma rede social. O padrão *Acidente* $\rightarrow$ *MsgEngarrafamento* [50] $\rightarrow$ *Engarrafamento* $\rightarrow \neg$*Engarrafamento* [60 min], inicia a de-

tecção a partir de um evento de acidente, seguido de 50 mensagens de congestionamento. Após isso, o mesmo verifica pela detecção de um evento do tipo *Engarrafamento*. Depois disso, o padrão procura a ausência de um evento de congestionamento por 60 minutos. Essa situação pode capturar a indicação de que o engarramento gerado por um dado acidente.

2.2.3. Contexto e Janelas de Tempo

Várias primitivas apresentadas requerem o conceito de uma janela, seja de tempo ou de tamanho, para processar o fluxo de eventos recebidos. Por exemplo, a primitiva *agregação* permite combina eventos correlatos que ocorrem dentro de um subconjunto do fluxo em um único evento complexo. Para tal, o CEP fornece meios de agrupar eventos relacionados em um *contexto* (janela) para permitir que eles sejam processados de maneira relacionada. Um contexto CEP subdivide o fluxo de eventos em uma ou mais partições [Luckham 2001, Etzion and Niblett 2010, Cugola and Margara 2012] usando predicados lógicos e/ou temporais. Desta maneira, cada partição de contexto representa um subconjunto do fluxo de eventos recebidos.

Por exemplo, considere a declaração da partição de contexto *MesmoVeículo* ilustrada na Figura 2.4. Esse contexto subdivide o fluxo de eventos *PosiçãoVeículo* de acordo com o identificador de cada veículo (utilizando o atributo `id`). A partição de contexto resultante gera vários subfluxos, de modo que todos os eventos localizados em uma determinada partição contenham o mesmo *id*, ou seja, contenham apenas eventos emitidos pelo mesmo veículo. Como a partição de contexto é um fluxo de eventos (um subconjunto), todas as primitivas de CEP funcionam normalmente nelas. Por exemplo, podemos utilizar o seguinte padrão de sequência, $MesmoVeículo \rightarrow \neg MesmoVeículo$ [2 min], para detectar a situação onde um dado veículo não transmite sua posição em um intervalo de 2 minutos.

Semelhantemente, podemos utilizar o conceito de contexto para particionar o fluxo de eventos de mensagens de uma rede social utilizando as palavras-chave da mesma. Nesse contexto, cada partição irá conter apenas mensagens que possuem a palavra-chave em questão.

As janelas de tempo são partições de contexto. Precisamente, uma janela de tempo é um contexto temporal que subdivide o fluxo de evento em intervalos de tempo utilizando o atributo *timestamp* de cada evento [Luckham 2001, Cugola and Margara 2012]. Desta maneira, a janela irá particionar o fluxo para incluir apenas os eventos cuja chegada esteja no intervalo especificado, *e.g.*, $(t - \Delta, t)$ onde t é a hora atual [Matysiak 2012, Amini et al. 2014]. Por exemplo, a declaração da janela de tempo *PosiçãoVeículo* [*Deslizar* 30 seg] cria automaticamente uma partição de contexto temporal que retém apenas os eventos deste fluxo recebidos nos últimos 30 segundos. Quando uma EPA processa um evento com tempo de chegada t nessa janela, as primitivas irão processar apenas os eventos que estão no intervalo $(t - 30, t)$. O conceito de janela é útil, pois, além de permitir tratar e lidar com o possível crescimento infinito dos fluxos de dados, também possibilita que as primitivas considerem apenas os eventos mais atuais na etapa de processamento.

Os dois principais tipos de janelas de tempo no CEP são lotes (*landmark/batch*) e deslizantes (*sliding*) [Matysiak 2012, Arasu et al. 2005] como ilustrado na Figura 2.5. As

janelas *landmark* provê a capacidade de processar o fluxo de eventos em lotes. Para tal, a janela armazena temporariamente (em um *buffer*) todos os eventos recebidos durante um intervalo de tempo e aplica as consultas contínuas considerando todos os eventos do lote. Neste caso, haverá um atraso entre a hora de chegada do evento e o tempo de processamento. Por exemplo, na Figura 2.5 (a), enquanto os eventos E_1 e E_2 chegaram no tempo $t = 1$, eles serão processados somente quando o período do lote terminar ($t = 2$).

Ao armazenar temporariamente os eventos em um *buffer*, é possível usar os predicados da primitiva de agregação, como *min* e *max*, para sintetizar todo o conteúdo do lote em um evento complexo. No entanto, se os eventos que devem ser processados juntos, por exemplo, por uma primitiva de sequência, forem colocados em lotes adjacentes, a mesma falhará em detectar a correlação entre os eventos, uma vez que a mesma considera apenas o conteúdo em análise. Uma maneira de atenuar esse problema é aumentar o período do lote, mas isso causa um atraso adicional para processar os eventos.

As janelas deslizantes tratam desse problema ao mover a janela de tempo junto com os eventos recebidos. Portanto, em vez de ter períodos de lote predefinidos, a borda da janela deslizam conforme o evento em análise. Mais especificamente, podemos definir uma janela deslizante como uma janela de lote em movimento que contém os eventos dos últimos Δ unidade de tempo. Para ilustrar esse conceito, considere o fluxo de eventos com uma janela de tempo deslizante de 1 segundo na Figura 2.5 (b). Por exemplo, quando $t = 3$, a janela de tempo inclui os eventos do segundo passado ($t = 2$) até o horário atual.

O movimento da janela, que desliza automaticamente os limites da janela, atenua o problema de correlacionar eventos situados próximos, mas em diferentes lotes. Por exemplo, os eventos E_3 e E_4 são colocados em lotes diferentes mesmo estando próximos um do outro. Este problema seria mitigado ao utilizar uma janela deslizante. Precisamente, a movimentação da janela possibilita incluir os eventos adjacentes e, ao mesmo tempo, fornece um processamento e reação ao fluxo em aproximadamente tempo real.

É importante destacar a existência de outros tipos de janela, como as de decaimento e de pulo [Ali et al. 2011, Cugola and Margara 2012, Amini et al. 2014]. A janela

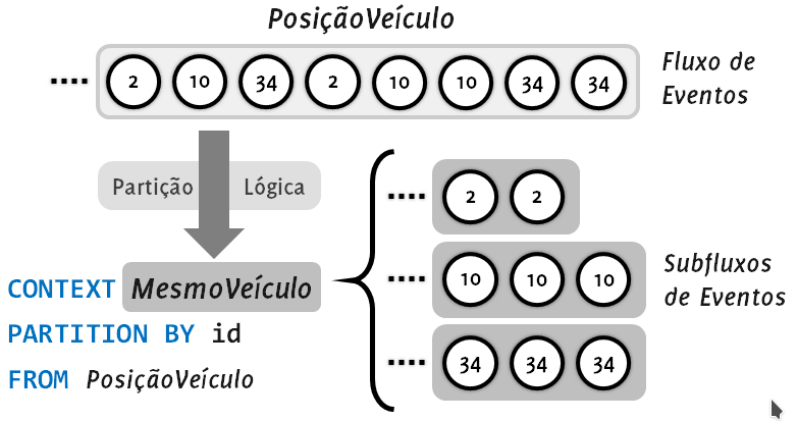


Figura 2.4: Exemplo de uma partição de contexto (*MesmoVeículo*) que subdivide o fluxo de eventos *PosiçãoVeículo* usando o valor *id* de cada evento.

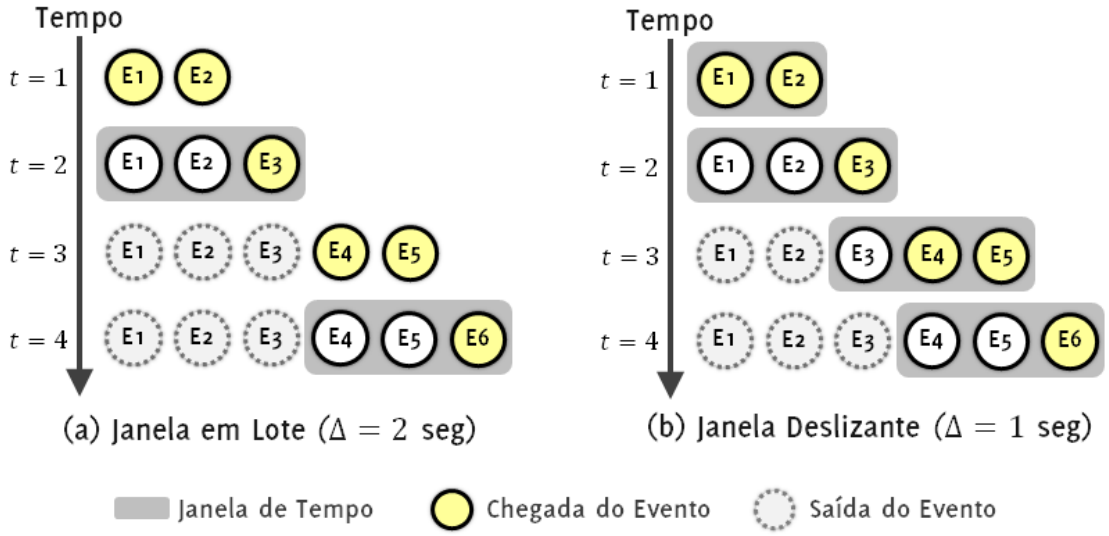


Figura 2.5: Exemplo de Janela em Lote e Deslizante.

de decaimento é uma variação de uma janela deslizante em que se aplica um fator de esquecimento λ aos eventos de acordo com a idade dos mesmos. Isto é feito para diferenciar a importância e a atualidade dos eventos na etapa de processamento, ou seja, eventos mais recentes têm maior importância que os eventos mais antigos. Para tal, é necessário que o usuário informe o peso λ e uma função de esquecimento f que determina como aplicar o peso aos eventos. No entanto, destacamos que o cálculo contínuo dos pesos de cada evento usando o fator de decaimento é muito caro para o processamento em tempo real dos dos fluxos [Amini et al. 2014].

Já a janela de pulo (*hopping*) mistura os conceitos de lote e de deslizamento [Ali et al. 2011, Cugola and Margara 2012]. Precisamente, a janela de pulo recebe dois atributos, b e h, sendo b o tamanho da janela em lote, h o tempo a ser deslizado e $h < b$. A idéia é que a janela de lote não deva pular diretamente para o próximo lote e sim deslizar em h unidades de tempo. Como $h < b$ a janela de lote irá avançar para um novo lote, mas irá levar consigo parte dos eventos acumulados anteriormente. Para exemplificar esta janela considere que a janela atualmente contenha os eventos no intervalo $(t-b,\ t)$. A próxima janela irá conter os eventos no intervalo $(t-b+h,\ t+h)$. Como $h < b$ a janela conterá eventos do lote anterior (especificamente do intervalo $(t-b+h,\ t)$.

Podemos ver que as janelas de tempo são um conceito central no processamento das primitivas do CEP. Entretanto, ressaltamos que nem todas as *engines* CEP fornecem suporte aos tipos de janelas vistas aqui. Na próxima seção será apresentado o motor de inferência Esper [EsperTech 2019], que suporta um bom conjunto das primitivas e janelas descritas.

2.3. A *engine* Esper e sua linguagem EPL

Esper é uma *engine* CEP de código aberto disponibilizada como um conjunto de bibliotecas Java. Isso significa que Esper pode ser embutida desde em aplicações para desktop

quanto em aplicações para servidores, essa característica dá flexibilidade a *engine* e permite por exemplo um prototipagem e testes locais anteriores ao *deployment*. Além de uma versão para .NET chamada Nesper, Esper possui um conjunto de adaptadores para entrada e saída de dados chamado EsperIO. Estes permitem acoplar a *engine* Esper diretamente a canais de comunicação como o Apache Kafka ou o protocolo HTTP. Nessa sessão apresentaremos de forma geral e daremos alguns exemplos de uso das bibliotecas que compõem a Esper e a EsperIO, adotamos a versão 8.2 da *engine*.

Os exemplos de código discutidos nesse capítulo estão disponíveis no repositório deste capítulo: https://github.com/fmagalhaes-inf-puc-rio/2019-CEP-Webmedia.

2.3.1. Arquitetura Geral

A *engine* Esper, desde a versão 8.0, consiste em uma linguagem (Esper EPL), um compilador (Esper Compiler) e um ambiente de execução (Esper Runtime).

- A Esper EPL é uma linguagem declarativa para descrever as regras CEP que definem o processamento de eventos. Ela é compatível com o padrão SQL-97 porém o estende com uma série de operadores focados no processamento de fluxos de eventos e detecção de padrões.
- O Esper Compiler converte regras EPL em bytecode java o que permite que as regras sejam processadas mais rapidamente no ambiente de execução.
- O Esper Runtime é executado sobre a máquina virtual java (JVM) tendo como entrada os eventos e como saída o resultado do seu processamento. A API java do Esper oferece métodos e interfaces para entrada de eventos assim como o consumo dos resultados porém também é possível utilizar a EsperIO que permite acoplar middleware diretamente a entrada e ou saída do Esper Runtime.

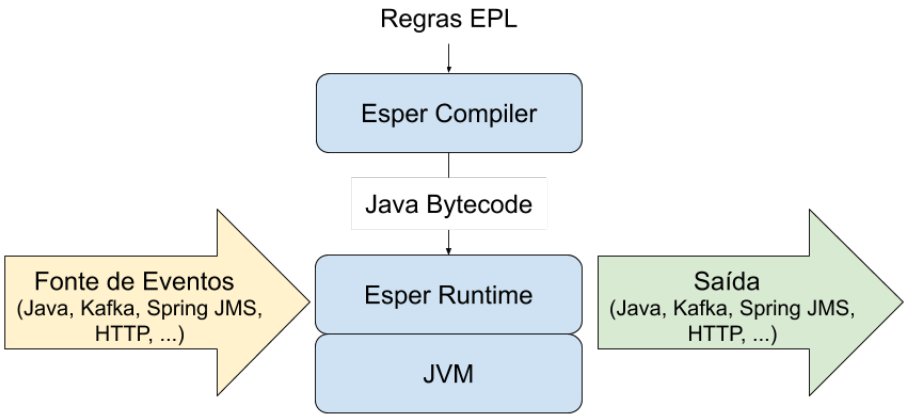


Figura 2.6: Arquitetura Geral da *engine* Esper.

2.3.2. Configuração do Motor CEP

A configuração permite especificar os seguintes parâmetros antes de iniciar o processamentos de eventos:

- Acesso a um Banco de Dados, permitindo que as regras EPL acessem dados contidos no banco e, ou insiram dados no banco;

- Importar classes e pacotes para o ambiente Esper permitindo que métodos declarados nessas classes sejam acessíveis dentro das regras CEP.

- Adição de tipos de eventos, o que também pode ser feito em tempo de execução a partir de regras EPL;

Essas configurações são feitas a partir da classe **Configuration**. É possível configurar a **engine** de forma programática no próprio código java como apresentado no código 2.3 ou a partir de ou carregar um aquivo de configuração em XML, como apresentado nos códigos 2.4 e 2.5.

```
Configuration c = new Configuration()
c.getCommon().addEventType(MyEvent.class);
c.getCommon().addImport("mypackage.MyClass");
Properties props = new Properties();
props.put("username", "myusername"); props.put("password", "mypassword");
props.put("driverClassName", "com.mysql.jdbc.Driver");
ConfigurationCommonDBRef cDB = new ConfigurationCommonDBRef();
cDB.setDataSourceFactory(props, BasicDataSourceFactory.class.getName());
c.getCommon().addDatabaseReference("mydb", cDB);
```

Código 2.3: Exemplo de configuração da *engine* ESPER como código Java.

```
Configuration c = new Configuration()
c.configure(myfile.xml);
```

Código 2.4: Exemplo de configuração da *engine* ESPER a partir de um arquivo XML.

```
<?xml version="1.0" encoding="UTF-8"?>
  <esper-configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.espertech.com/schema/esper"
    xsi:schemaLocation=
          "http://www.espertech.com/schema/esper/esper-configuration-8-0.xsd">
  <common>
    <event-type name="MyEvent" class="mypackage.MyEvent"/>
  </common>
</esper-configuration>
```

Código 2.5: Exemplo de arquivo de configuração XML.

2.3.3. Definição de regras em Esper EPL

A linguagem Esper EPL extende o padrão SQL-97, portanto operações padrões como `SELECT, FROM, WHERE, JOIN, CREATE, DELETE, ...` são suportadas. Por exemplo a regra a seguir é válida em Esper EPL e seleciona todo novo evento do tipo LeituraSensor que for recebido no Esper Runtime.

```
1 SELECT * FROM MeuEvento
```

Assumimos que o(a) leitor(a) já possui alguma familiaridade com a definição de consultas em SQL e apresentaremos os pontos onde a EPL Esper estende os padrões do SQL incluindo operadores idealizados para o processamento de fluxos de eventos.

2.3.3.1. Janelas de Eventos

Como os fluxos de eventos são potencialmente infinitos, é necessário fornecer um método para delimitar o escopo de uma regra a um faixa do fluxo, em Esper EPL isso é feito com janelas de eventos. Essas janelas podem ser delimitadas por tempo ou por número de eventos, além de processadas de forma deslizante ou em lotes.

Na regra a seguir, o símbolo # indica o início da definição de uma janela e o termo `LENGHT` denota que a janela é deslizante e delimitada por número de eventos. Nesse caso a cada novo evento do tipo `MeuEvento` recebido, a regra será ativada e retornará os valores dos atributos do novo evento além disso a regra retornará como valor de `avg1` a média do valor do `atributo1` nos últimos 10 eventos, ou seja o novo evento e os 9 eventos imediatamente anteriores se houverem.

```
1 SELECT *, AVG(MeuEvento.atributo1) AS avg1 FROM MeuEvento#LENGHT(10)
```

Para utilizar uma janela deslizante delimitada por tempo, bastaria trocar o termo `LENGHT` por `TIME` e incluir um unidade de tempo logo após o numeral 10. Se nenhuma unidade for especificada, o compilador de regras assume que o tempo está em segundos. Por exemplo, se substituíssemos o numeral `10` por `10 min` a consulta continuaria produzindo saídas a cada novo evento porém `avg1` passaria a conter a média do do valor do `atributo1` nos eventos recebidos no últimos 10 minutos. Ao utilizar janelas de processamento em lotes, as consultas passam a retornar resultados em lote ao invés de a cada evento recebido, ou seja cada vez que o lote é preenchido a regra retorna um resultado com todos os eventos do lote. As janelas de lote também podem ser delimitadas por tempo ou por número de eventos e para definir indicar uma janela de lote, basta adicionar o termo `BATCH` como na regra a seguir.

```
1 SELECT * FROM MeuEvento#TIME_BATCH(10)
```

Também é possível criar uma janela de eventos nomeada. Essa opção é especialmente útil quando múltiplas regra consultam a mesma janela de eventos. Para definir uma

janela de eventos basta usar a cláusula `CREATE WINDOW`, é necessário especificar o tamanho da janela, que assim como as janelas não nomeadas pode ser delimitado por tempo ou por número de eventos. A seguir um exemplo de regra que define uma janela nomeada que armazena 10 segundos de eventos.

```
1 CREATE WINDOW MeuEvento15sTIME(10) AS MeuEvento
```

O segundo passo é popular a janela isso é feito a partir de uma regra CEP que utiliza a cláusula `INSERT INTO`. Dessa maneira é possível filtrar os eventos que iram preencher a janela. Por exemplo, se janela deve conter apenas os eventos com `atributo1` maior que 5, pode-se fazer isso com a seguinte regra

```
1 INSERT INTO MeuEvento15s SELECT * FROM MeuEvento WHERE atributo1 > 5
```

Então é possível definir regras que consultam a janela nomeada. A regra abaixo por exemplo consulta os eventos dos últimos 15 segundos com atributo1 superior a 5 e atributo2 superior a > 20.

```
1 SELECT * FROM MeuEvento15s WHERE atributo2 > 20
```

2.3.3.2. Filtros

Filtros funcionam de forma semelhante a cláusula `WHERE`, permitindo especificar valores (ou faixas de valores) para as propriedades de um evento nas consultas. Porém enquanto um filtro restringe os eventos que entraram na janela, as cláusulas `WHERE` especificam dentre os eventos da janela quais serão retornados pela consulta. Por exemplo, as duas regras a seguir selecionam eventos com temperatura acima de 100:

```
1 SELECT temperatura FROM LeituraSensor(temperatura>100)#LENGHT_BATCH(10)
```

Código 2.6: Seleciona lotes de 10 eventos `LeituraSensor` com temperatura maior que 100, sempre retorna 10 eventos.

```
1 SELECT temperatura FROM LeituraSensor(temperatura>100)#LENGHT_BATCH(10)
```

Código 2.7: Seleciona dentre lotes de 10 eventos `LeituraSensor` aqueles com temperatura superior a 100, retorna 10 ou menos eventos.

2.3.3.3. Definições de padrões utilizando `PATTERN` e `MATCH_RECOGNIZE`

A EPL Esper permite descrever padrões de sequências de eventos utilizando cláusulas `PATTERN` ou `MATCH_RECOGNIZE`, elas possuem sintaxe e um conjunto de operadores diferentes. Como exemplo de regra de uso de `PATTERN` temos a regra abaixo, ela dispara quando um evento de `FornalhaLigada` não é seguida por um evento de `FornalhaDesligada` correspondente após passadas 4 horas.

```
SELECT a.*
FROM PATTERN
[
 EVERY a=FornalhaLigada -> (TIMER:INTERVAL(4 hours)
 AND NOT FornalhaDesligada(idFornalha=a.idFornalha))
]
```

As cláusulas `PATTERN` possuem 3 tipos de operadores:

1. Operadores de repetição:
   (a) `EVERY` seleciona todas as sequências de eventos que se encaixam em um padrão.
   (b) `EVERY-DISTINCT` análogo ao operador `EVERY` porém elimina resultados duplicados.
   (c) `[n]` seleciona n sequências de eventos que se encaixam em um padrão.
   (d) `UNTIL` especifica o fim de um padrão.
2. Operadores de sequência:
   (a) `->` (seguido por) define um padrão caracterizado pela ocorrência de uma sequencia de eventos em uma ordem especifica.
   (b) `OR` define um padrão caracterizado pela ocorrência de um ou outro evento.
   (c) `AND` define a ocorrência de uma sequência de eventos em qualquer ordem. Isso significa que `(EventoA AND EventoB)` é equivalente a `(EventoA->EventoB) OR (EventoB->EventoA)`.
3. Operadores de controle:
   (a) `TIMER:WITHIN(tempo)` serve para definir um padrão que deve acontecer dentro de um espaço de tempo.
   (b) `TIMER:INTERVAL(tempo)` define um intervalo de tempo que fará parte do padrão.
   (c) `WHILE` define que um padrão é valido apenas enquanto uma condição for verdadeira.

Já a especificação de padrões utilizando `MATCH_RECOGNIZE` é baseada em expressões regulares. Possuindo os operadores de:

- Agrupamento: `()`
- Quantificadores: `*, +, ?, {min, max}`
- Concatenação: `AB`
- Alternância: `|`

Como exemplo de regra de uso de `MATCH_RECOGNIZE` temos a regra a seguir, ela dispara quando um evento de `LeituraSensor` com temperatura superior a 50 é imediatamente seguido por um evento com temperatura ainda maior ou por um evento com radiação superior a 4.

```
SELECT * FROM LeituraSensor
MATCH_RECOGNIZE
(
   MEASURES A.temperatura AS aTemp, B.temperatura AS bTemp, C.radiacao AS cRad
    PATTERN (A (B | C))
      DEFINE A AS A.temperatura >= 50, B AS B.temperatura > A.temperatura,
             C AS C.radiacao >= 4
)
```

Outros exemplos de regras com `PATTERN` e `MATCH_RECOGNIZE` estão disponíveis no repositório deste capítulo.

2.3.4. Usando a biblioteca EsperIO

A biblioteca EsperIO provê um conjunto de adaptadores de entrada e saída pra o EsperRuntime, permitindo consumir ou exportar eventos diretamente de ou para arquivos, páginas HTML, Apache Kafka, ou outros. Nessa subseção apresentaremos um exemplo de uso dessa biblioteca para ler eventos de entrada de um arquivo e exportar eventos complexos gerados para arquivos de registro.

O objetivo desse exemplo é processar eventos do tipo `AtualizacaoSensor` com as propriedades `temperatura, Umidade, idSala, data_hora`. E produzir dois tipos de eventos complexos que representam estados de alerta: `BaixaUmidade` e `AltaTemperatura` que herdam as propriedades do evento simples.

Assumindo que uma leitura de umidade inferior a 35% é considerada baixa e uma temperatura acima de 35°C é considerada alta, utilizamos as seguintes regras para produzir o eventos complexos a partir de eventos de `AtualizacaoSensor`:

```
INSERT INTO BaixaUmidade -- Gerar eventos de BaixaUmidade
SELECT s.temperatura AS temperatura, s.umidade AS umidade,
       s.idSala AS idSala, s.data_hora AS data_hora
FROM PATTERN
[
 EVERY-DISTINCT(s.data_hora) s=SensorUpdate(humidade<0.35)
]
```

```
8
9  INSERT INTO AltaTemperatura -- Gerar eventos de AltaTemperatura
10 SELECT s.temperatura AS temperatura, s.umidade AS umidade,
11        s.idSala AS idSala, s.data_hora AS data_hora
12 FROM PATTERN
13 [
14    EVERY-DISTINCT(s.data_hora) s=SensorUpdate(temperatura>35)
15 ]
```

Agora o código EPL que define a leitura dos eventos de `AtualizacaoSensor` de um arquivo:

```
1  CREATE DATAFLOW SensorCSVEntrada
2     -- A regra define a geracao de um fluxo de evento a partir de um arquivo
3     FileSource -> sensorstream<SensorUpdate> {
4        --- O arquivo que sera lido
5        file: 'input.csv',
6        -- A ordem em que cada propriedade aparece em cada linha do arquivo csv
7        propertyNames: ['temperature','humidity','roomId', 'data_hora']
8     }
9  --Faz com que o EsperRuntime consuma o fluxo de eventos
10 EventBusSink(sensorstream){}
```

Por fim definimos o código que faz o registro dos eventos complexos:

```
1  CREATE DATAFLOW BaixaUmidadeCSVSaida
2     -- Coleta o fluxo de eventos de BaixaUmidade
3     EventBusSource -> outstream<BaixaUmidade> {}
4        -- Define uma saida em arquivo
5        FileSink(outstream) {
6        -- O arquivo onde sera salvo o registro
7        file: 'RegistroBaixaUmidade.csv'
8        -- Se o arquivo ja existir adiciona os valores novos no final
9        append: true
10    }
11 -- Fazemos o mesmo para os eventos de AltaTemperatura
12 CREATE DATAFLOW AltaTemperaturaCSVSaida
13    EventBusSource -> outstream<AltaTemperatura> {}
14       FileSink(outstream) {
15       file: 'RegistroAltaTemperatura.csv'
16       append: true
17    }
```

O código completo desse exemplo, incluindo a configuração necessária para usar a EsperIO está no diretório *EsperIODemo* do repositório online.

2.4. Estudo de Caso IoT: Monitoramento e Controle de temperatura

Apresentaremos aqui um exemplo onde CEP foi usado para monitorar e controlar a temperatura em um pequeno reator químico. Durante uma reação química exotérmica (que libera calor) para a produção de algum composto, é necessário um controle de resfriamento. A temperatura no recipiente deve ser mantida o mais próximo possível da temperatura ideal para a reação. Além disso é interessante um sistema de monitoramento que emita notificações caso essa temperatura sofra mudanças bruscas ou se distancie muito da ideal. O exemplo apresentado aqui foi testado em um modelo físico em um reator onde implementamos sistema que usa CEP para monitorar a temperatura e controlar o resfriamento de um reator. O resfriamento foi feito usando uma bomba que mantinha um fluxo de água gelada em uma camisa ao redor do reator.

Foi utilizado um sensor de temperatura no reator e outro no tanque de água gelada, os dados gerados por esses sensores foram encapsulados em eventos do tipo `EventoTemperatura` com as propriedades `temperaturaReator`, `temperaturaTanque` e `data_hora`. Como os sensores produzem dados com ruído, a primeira regra criada gera eventos com a média das últimas 5 leituras de modo a dissolver o ruído:

```
INSERT INTO TemperaturaMedia
SELECT AVG(temperaturaReator) AS temperaturaReator,
       AVG(temperaturaTanque) AS temperaturaTanque,
       CAST(AVG(data_hora), long) AS data_hora
FROM EventoTemperatura#LENGTH(5)
```

A bomba possui um potenciômetro para controlar a intensidade do fluxo de água. O ângulo dele é modificado sempre que a *engine* CEP produz eventos do tipo `MudarAngulo`. Esse evento foi definido usando o operador `CREATE SCHEMA` na regra a seguir:

```
CREATE SCHEMA MudarAngulo AS (angulo double, acao string, fonte string)
```

Nos eventos `MudarAngulo`, `acao` pode conter o valor “abrir” ou “fechar”, respectivamente representando um aumento ou diminuição do fluxo de água gelada. Existem dois tipos de regras controlam o sistema gerando esses eventos:

1. **Regras de incidente:** São disparadas caso de incidentes onde a temperatura chega a um nível de alerta ou crítico.

2. **Regras de ajuste fino:** São disparadas quando a temperatura está fora dos níveis de alerta ou crítico e buscam estabilizar a temperatura próximo de seu valor ideal.

2.4.1. Regras de incidente

Nesse caso de uso, o ângulo mínimo suportado pela bomba é 50° e o máximo é 170° e dada a temperatura ideal de 30°C considerou-se estado de alerta valores abaixo de 22.5°C ou acima de 37.5°C e estado crítico valores abaixo de 19.5°C ou acima de 40.5°C. As regras de estado critico buscam agir o mais rápido possível utilizando sempre o ân-

gulo máximo ou o angulo mínimo da bomba. Já as regras de alerta aumentam ou diminuem o ângulo em 5 graus sempre que uma delas é disparada, elas utilizam a variável `var_AnguloBomba` que armazena o valor atual do ângulo da bomba. Posteriormente explicaremos como o valor dessa variável é modificado automaticamente. As regras de incidente estão listada a seguir:

```
1
2  -- Regra para estado critico de temperatura baixa
3  INSERT INTO MudarAngulo
4  SELECT 50 AS angulo, "fechar" AS acao, "incidente" AS fonte
5  FROM TemperaturaMedia(temperaturaReator < 19.5)
6
7  -- Regra para estado critico de temperatura alta
8  INSERT INTO MudarAngulo
9  SELECT 170 AS angulo, "abrir" AS acao, "incidente" AS fonte
10 FROM TemperaturaMedia(temperaturaReator > 40.5)
11
12 -- Regra para estado de alerta de temperatura baixa.
13 INSERT INTO MudarAngulo
14 SELECT max(50, (var_AnguloBomba-5)) AS angulo, "fechar" AS acao,
15       "incidente" AS fonte
16 FROM TemperaturaMedia(temperaturaReator < 22.5 AND temperaturaReator >= 19.5)
17
18 -- Regra para estado de alerta de temperatura alta.
19 INSERT INTO MudarAngulo
20 SELECT min(170, (var_AnguloBomba+5)) AS angulo, "abrir" AS acao,
21       "incidente" AS fonte
22 FROM TemperaturaMedia(temperaturaReator > 37.5 AND temperaturaReator =< 40.5)
```

Além da variável `var_AnguloBomba`, temos a variável `var_UltimaAbertura` para armazenar se o último `MudarAngulo` com `acao` "abrir" foi gerado a partir uma regra de incidente ou de ajuste e a variável `var_UltimoFechamento` que armazena o mesmo para `acao` "fechar". As regras abaixo modificam o valor dessas variáveis sempre que há um novo evento `MudarAngulo` utilizando o comando `SET` dentro de uma cláusula `ON <EVENTO>`.

```
1  -- Modifica o valor de var_AnguloBomba e de var_UltimoFechamento
2  ON MudarAngulo(acao = "fechar") AS ma
3  SET var_AnguloBomba = ma.angulo, var_UltimoFechamento = ma.fonte
4
5  -- Modifica o valor de var_AnguloBomba e de var_UltimaAbertura
6  ON MudarAngulo(acao = "abrir") AS ma
7  SET var_AnguloBomba = ma.angulo, var_UltimaAbertura = ma.fonte
```

2.4.2. Regras de ajuste fino

As regras de ajuste fino tentam estabilizar a temperatura quando ela está subindo ou descendo além do valor ideal. Primeiramente, elas calculam a diferença entre o valor atual

da temperatura e o valor ideal. Então o ajuste feito no ângulo da bomba é determinado multiplicando essa diferença por quatro.

```
-- Regra para corrigir aquecimentos
INSERT INTO MudarAngulo
-- O angulo nao deve ultrapassar 170
SELECT min(170, var_AnguloBomba+((e2.temperaturaReator-30)*4)) AS angulo,
      "abrir" AS acao, "ajuste" AS fonte
-- A temperatura deve estar superior ao valor ideal, mas fora da faixa de alerta
FROM PATTERN [
  EVERY-DISTINCT(e2.data_hora) e1=TemperaturaMedia
  ->
  e2=TemperaturaMedia(temperaturaReator in (30.1, 37.5)
]
-- A temperatura deve estar subindo
WHERE e2.temperaturaReator > e1.temperaturaReator

-- Regra para corrigir resfriamentos
INSERT INTO MudarAngulo
-- O angulo nao deve pode ser inferior a 50
SELECT max(50, var_AnguloBomba-((30-e2.temperaturaReator)*4)) AS angulo,
      "fechar" AS acao, "ajuste" AS fonte
-- A temperatura deve estar inferior ao valor idel, mas fora da faixa de alerta
FROM PATTERN [
  EVERY-DISTINCT(e2.data_hora) e1=TemperaturaMedia
  ->
  e2=TemperaturaMedia(temperaturaReator in (22.5, 29.9)
]
-- A temperatura deve estar descendo
WHERE e2.temperaturaReator<e1.temperaturaReator
```

2.4.3. Regras de monitoramento

Além das regras de controle discutidas anteriormente também foram definidas regras de monitoramento, que geram notificações caso a temperatura irregularidades. Algumas das regras são muito semelhantes as regras de incidente, detectando quando a temperatura está em níveis críticos ou de alerta. Porém ao invés de gerar eventos `MudarAngulo` elas retornam os atributos do evento `TemperaturaMedia` que disparou a regra. Por exemplo a regra para níveis de alerta de temperatura baixa é a seguinte:

```
SELECT *
FROM TemperaturaMedia(temperaturaReator < 22.5 AND temperaturaReator >= 19.5)
```

Também foram definidas regras para detectar mudanças repentinas na temperatura que representem saltos para longe da temperatura ideal. Eles analisam os quatro últimos eventos de temperatura média e são acionadas se os eventos em sequência se afastam

cada vez mais do valor ideal e a temperatura do último evento difere em mais de 5% do primeiro.

```
-- Detecta aumento repentino
SELECT * FROM TemperaturaMedia
MATCH_RECOGNIZE
(
 MEASURES
    A.temperaturaReator AS temperaturaInicial,
    D.temperaturaReator AS temperaturaFinal,
    ((D.temperaturaReator-A.temperaturaReator)/A.temperaturaReator) AS aumento
 PATTERN (A B C D)
 DEFINE
    A AS A.temperaturaReator > 30
    B AS B.temperaturaReator > A.temperaturaReator
    C AS C.temperaturaReator > B.temperaturaReator
    D AS D.temperaturaReator > C.temperaturaReator
 AND D.temperaturaReator > (A.temperaturaReator * 1.05)
)
-- Detecta queda repentina
SELECT * FROM TemperaturaMedia
MATCH_RECOGNIZE
(
 MEASURES
    A.temperaturaReator AS temperaturaInicial,
    D.temperaturaReator AS temperaturaFinal,
    ((A.temperaturaReator-D.temperaturaReator)/A.temperaturaReator) AS queda
 PATTERN (A B C D)
 DEFINE
    A AS A.temperaturaReator < 30
    B AS B.temperaturaReator < A.temperaturaReator
    C AS C.temperaturaReator < B.temperaturaReator
    D AS D.temperaturaReator < C.temperaturaReator
 AND D.temperaturaReator < (A.temperaturaReator * 0.95)
)
```

2.4.3.1. Resultados obtidos

Nesse estudo de caso, as regras CEP conseguiram manter a temperatura no reator em uma faixa de 3°C ao redor do valor ideal. A figura 2.7 demonstra esses resultados. Nos testes iniciamos o controlador CEP com a temperatura do reator 18°C graus acima do valor ideal para verificar se o sistema conseguiria estabilizar a temperatura.

2.5. Estudo de Caso de Smart City (Cidades Inteligentes)

Com intuito de ilustrar os conceitos de CEP e da *engine* Esper descritos esta seção apresenta um estudo de caso de cidade inteligente que utiliza o fluxo de dados da posição dos

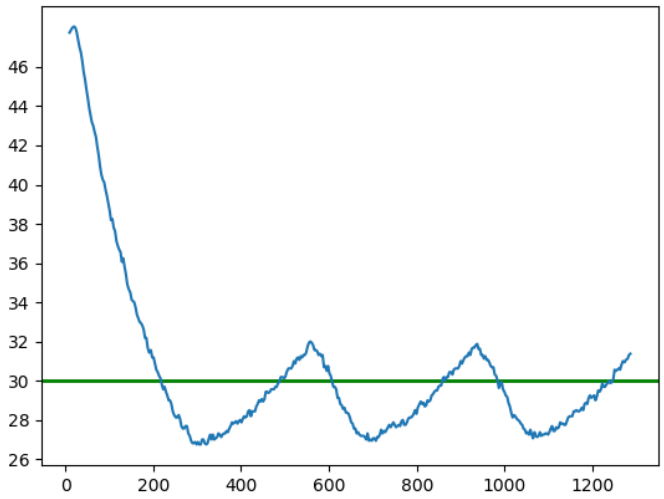


Figura 2.7: Temperatura ao longo do tempo usando as regras CEP para controle

ônibus da cidade do Rio de Janeiro para detectar possíveis locais de congestionamentos em tempo real. A arquitetura da aplicação, ilustrada pela Figura 2.8, descreve uma visão geral da interação entre as diferentes entidades necessárias para construir tal sistema, como produtor de evento, recepção de fluxo de entrada e EPAs utilizados pela máquina de inferência CEP.

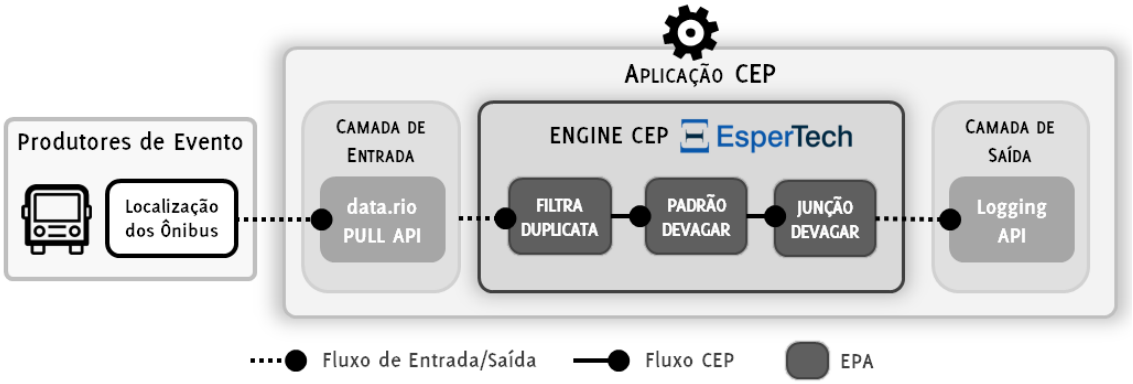


Figura 2.8: Arquitetura da aplicação CEP de cidades inteligentes para detecção de possíveis congestionamentos no trânsito.

O primeiro passo para a construção da aplicação é a definição da fonte do fluxo de eventos. Neste caso, será utilizado a plataforma de dados abertos `data.rio`[1] para obtenção da localização de todos os ônibus da cidade. A Tabela 2.1 ilustra um subconjunto

[1]O `data.rio` é uma plataforma de dados abertos que divulga em tempo real diversas informações da cidade do Rio de Janeiro. A posição de todos os ônibus pode ser obtida na seguinte URL: http://dadosabertos.rio.rj.gov.br/apiTransporte/apresentacao/rest/index.cfm/obterTodasPosicoes

deste fluxo de dados. Cada dado contém informações referentes ao horário de envio (`DATAHORA`), identificador (`ORDEM`), posição (`LATITUDE` e `LONGITUDE`), além da velocidade do veículo, em metros por segundo, durante a medição (`VELOCIDADE`). Note que como o serviço retorna a última posição de todos os ônibus, pode ocorrer de ter dados com datas anteriores à atual, visto que os mesmos podem não ter atualizados suas posições ou terem parados de operar. Utilizando os campos descritos na fonte de dados podemos definir o primeiro evento da aplicação: `LocalizacaoOnibusAPI`. Adota-se o termo API para ilustrar que o evento não foi pré-processado e reflete diretamente o dado da fonte do fluxo.

Tabela 2.1: Exemplo do formato de dados de ônibus da API `data.rio`

DATAHORA	ORDEM (ID)	LINHA	LATITUDE	LONGITUDE	VELOCIDADE
09-14-2019 18:04:44	A72062	14	-22.921499	-43.185966	0.56
09-14-2019 18:04:45	A72160	105	-22.982031	-43.240974	2.5
09-14-2019 18:04:47	B10124	323	-22.80504	-43.206684	0

Os dados podem ser obtidos em tempo real consultando a *Application Programing Interface* (API) do serviço `data.rio`, isto é, aplicações interessadas podem puxar (*pull*) os dados da API para obter um reflexo do estado da posição dos veículos. Isto pode ser realizado periodicamente para reconstruir o fluxo de eventos. Precisamente, no estudo de caso buscamos os dados a cada 30 segundos. Entretanto, como a API retorna as últimas posições de todos os ônibus, pode ocorrer de que leituras subsequentes da API contenham dados duplicados.

Para reter eventos duplicados, o primeira EPA utiliza a primitiva de *filtrar* e *sequência*, como descrito no Código 2.8. O EPA funciona da seguinte maneira. Ao receber um evento do tipo `LocalizacaoOnibusAPI` a consulta contínua executa as seguintes etapas. Primeiro, verifica-se se o evento é diferente dos eventos prévios que tenha recebido. Para cada evento `b1` distinto (`EVERY-DISTINCT`) inicia-se a busca por um evento subsequente do mesmo veículo, mas com data diferente. Caso o evento seja repetido, descarta-se o mesmo. Além disso, verifica-se se o evento distinto recebido casa com alguma sequência prévia, isto é, de um evento passado do mesmo veículo. Nesta situação, gera-se o evento do tipo `LocalizacaoOnibus`, que tem estrutura equivalente ao `LocalizacaoOnibusAPI`, mas que representa um evento não duplicado.

```
1 INSERT INTO LocalizacaoOnibus
2 SELECT b2.data AS data, b2.id AS id,
3        b2.linha AS linha, b2.velocidade AS velocidade,
4        b2.latitude AS latitude, b2.longitude AS longitude
5 FROM PATTERN
6 [ EVERY-DISTINCT(b1.data, b1.id) b1 = LocalizacaoOnibusAPI
7    -> b2 = LocalizacaoOnibusAPI(id = b1.id, data != b1.data) ]
```

Código 2.8: EPA para filtrar dados de localização duplicados.

O próximo EPA consome o fluxo de eventos `LocalizacaoOnibus` para detectar quando os veículos estão se movendo lentamente (evento `OnibusDevagar`). Para isso, utiliza-se novamente a primitiva de `sequência` em conjunto com as de `filtrar` e `projetar`, como descrito no Código 2.9. A consulta contínua inicia a detecção do padrão ao receber um evento do tipo `LocalizacaoOnibus`, representado pela variável `p1`. Após isso, espera-se por um intervalo de 3 minutos. Em seguida, ao receber a próxima localização do veículo (`p2`) calcula-se a movimentação do veículo através da função `distancia`, uma função externa que computa a distância em metros entre duas posições de latitude e longitude. Caso a distância percorrida seja inferior a 750 m, isto é, menor que $\frac{750}{180} \times 3.6 = 15$ km/h, gera-se um evento `OnibusDevagar` com as localizações `p1` e `p2`, além da distância e do tempo entre os eventos. Caso negativo, o padrão é cancelado e considera-se que o veículo moveu normalmente. Note que é importante considerar um fluxo que não tenha eventos duplicados, como o utilizado, uma vez que ao considerar eventos idênticos de `LocalizacaoOnibus` chegar-se-ia a conclusão que o veículo não moveu durante o período, entretanto, isso apenas significa que o mesmo não atualizou sua posição neste intervalo.

```
INSERT INTO OnibusDevagar
SELECT p1 AS primPosicao, p2 AS ultPosicao, p1.id AS id,
       distancia(p1, p2) AS distViajada, (p2.data - p1.data) AS periodo
FROM PATTERN
[
   EVERY p1=BusLocationUpdateEvent
   ->
   timer:interval(3 min)
   ->
   p2=BusLocationUpdateEvent(id = p1.id)
]
WHERE distancia(p1, p2) <= 750
```

Código 2.9: EPA para detectar ônibus que estão se movendo lentamente.

A próxima etapa é correlacionar, isto é, agrupar, eventos de `OnibusDevagar` que ocorrem próximos um dos outros. Para tal, primeiramente se cria a janela `OnibusDevagarWindow`, como descrito no Código 2.10, que irá contemplar apenas os últimos eventos do fluxo dos últimos 5 minutos através dos operadores `#UNIQUE`(id)) e `#TIME`(5 min) respectivamente. Observe que ao passo que o veículo se move, os novos eventos de `OnibusDevagar` sobrescrevem os eventos prévios na janela de eventos nomeada.

```
CREATE WINDOW OnibusDevagarWindow#UNIQUE(id)#TIME(5 min) AS OnibusDevagar
```

Código 2.10: EPL para criar uma janela do fluxo `OnibusDevagar`.

Para encontrar a concentração de ônibus que estão se movendo lentamente e, que possivelmente representa um congestionamento, o último EPA irá correlacionar o surgimento de um novo evento do tipo `OnibusDevagar` com os restantes da janela nomeada `OnibusDevagarWindow`, como descrito no Código 2.11. Ao receber um evento `p1` do tipo `OnibusDevagar` o EPA consultará a janela `OnibusDevagarWindow`, aqui denominada `pw`. Em seguida, correlaciona `p1` com todos eventos de `pw`. . Caso exista pelo menos três eventos no mesmo período (de 5 minutos) e de ônibus diferentes localizados em até 500 metros de p1 Os eventos são ditos próximos se a distância entre os ônibus lentos for menor do que 500 metros. Caso existam pelo menos três veículos nesta situação (com pouca movimentação), em uma janela de 5 minutos, gera-se um evento complexo do tipo `PossivelCongestionamento` composto pelos eventos individuais de cada ônibus lento.

```
1 ON OnibusDevagar AS p1
2 INSERT INTO PossivelCongestionamento
3 SELECT p1, pw
4 FROM OnibusDevagarWindow AS pw
5 WHERE distancia(p1, pw) <= 500
6 HAVING COUNT(*) >= 3
```

Código 2.11: EPA para detectar aglomerados de ônibus lentos.

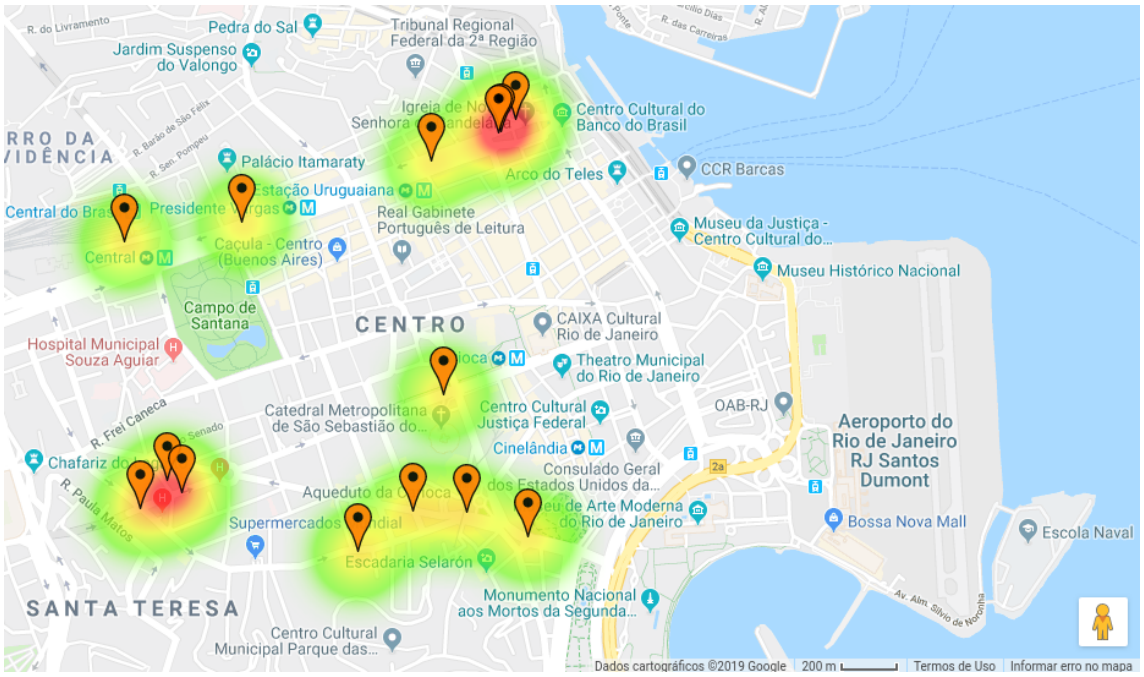


Figura 2.9: Possíveis locais de lentidão no trânsito conforme o processamento do fluxo de dados.

A Figura 2.9 ilustra o mapa de calor dos diversos eventos de `PossivelCongestionamento` detectados na região do centro do Rio de Janeiro. Cada evento desses é composto pelos

ônibus lentos próximos um dos outros, isto é, de eventos do tipo `OnibusDevargar`. As áreas e regiões de calor encontradas refletem regiões comumente associadas a lentidão e congestionamento, como Av. Presidente Vargas e Praça da Cruz Vermelha.

Além disso, ressalta-se que é possível estender essa aplicação para não somente detectar as áreas de lentidão e, possivelmente, de congestionamento no trânsito, mas também de acompanhar a evolução das mesmas. Por exemplo, pode-se monitorar os eventos `PossivelCongestionamento` para verificar se os mesmos estão aumentando ou diminuindo através da intersecção com eventos passados do mesmo tipo.

2.6. Considerações Finais

O *Complex Event Processing* (CEP – Processamento de Eventos Complexos) é um modelo de programação importante para sistemas orientados a fluxos de dados, como dados de sensores, análise de tráfego e análise de redes sociais. Este capítulo teve o objetivo de apresentar o modelo de programação CEP como meio de lidar com as especificidades de fluxos de dados.

Especificamente, o capítulo apresentar ao leitor: (1) os conceitos e problemas fundamentais de processamento de fluxo de dados e o modelo CEP; (2) o motor de inferência Esper e a sua linguagem de regras EPL como meio de tratar os fluxo de dados; (3) exemplo um caso de uso da aplicação de CEP para tratamento de dados de aplicações IoT; (4) exemplo de um caso de uso de aplicação de CEP como meio de processar dados de aplicações para *Smart Cities*.

Nesse contexto, acreditamos que a proposta do minicurso é importante para estudantes, pesquisadores e profissionais interessados em explorar as potencialidades de fluxos de dados de aplicações de Internet das Coisas e Cidades Inteligentes.

Referências

[Ali et al. 2011] Ali, M., Chandramouli, B., Goldstein, J., and Schindlauer, R. (2011). The extensibility framework in microsoft streaminsight. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 1242–1253, Washington, DC, USA. IEEE Computer Society.

[Amini et al. 2014] Amini, A., Wah, T., and Saboohi, H. (2014). On Density-Based Data Streams Clustering Algorithms: A Survey. *Journal of Computer Science and Technology*, 29(1):116–141.

[Anicic et al. 2010] Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., and Studer, R. (2010). A rule-based language for complex event processing and reasoning. In Hitzler, P. and Lukasiewicz, T., editors, *Web Reasoning and Rule Systems*, pages 42–57, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Arasu et al. 2003] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., and Widom, J. (2003). Stream: The stanford stream data manager (demonstration description). In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 665–665, New York, NY, USA. ACM.

[Arasu et al. 2005] Arasu, A., Babu, S., and Widom, J. (2005). The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142.

[Bali 2009] Bali, M. (2009). *Drools JBoss Rules 5.0 Developer's Guide*. Packt Publishing.

[Cameron et al. 2012] Cameron, M. A., Power, R., Robinson, B., and Yin, J. (2012). Emergency situation awareness from twitter for crisis management. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 695–698, New York, NY, USA. ACM.

[Carbone et al. 2015] Carbone, P., Ewen, S., Haridi, S., Katsifodimos, A., Markl, V., and Tzoumas, K. (2015). Apache Flink: Unified Stream and Batch Processing in a Single Engine. *Data Engineering*, pages 28–38.

[Cugola and Margara 2012] Cugola, G. and Margara, A. (2012). Processing Flows of Information: From Data Stream to Complex Event Processing. *ACM Computing Surveys*, 44(3):1–62.

[Dunkel et al. 2011] Dunkel, J., Fernández, A., Ortiz, R., and Ossowski, S. (2011). Event-driven architecture for decision support in traffic management systems. *Expert Systems with Applications*, 38(6):6530 – 6539.

[EsperTech 2019] EsperTech (2019). Esper - Complex Event Processing. `http://www.espertech.com/esper/`.

[Etzion and Niblett 2010] Etzion, O. and Niblett, P. (2010). *Event Processing in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

[Flouris et al. 2016] Flouris, I., Giatrakos, N., Deligiannakis, A., Garofalakis, M., Kamp, M., and Mock, M. (2016). Issues in complex event processing: Status and prospects in the Big Data era. *Journal of Systems and Software*, pages 1–20.

[Kudyba 2014] Kudyba, S. (2014). *Big Data, Mining, and Analytics*. Auerbach Publications, Boca Raton, Florida, 1st edition.

[Luckham 2001] Luckham, D. C. (2001). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Maison et al. 2013] Maison, R., Majda, E., Dobrowolski, A. P., and Zakrzewicz, M. (2013). Similarity based join over audio feeds in a multimedia data stream management system. *Bell Labs Technical Journal*, 18(1):195–212.

[Matysiak 2012] Matysiak, M. (2012). Data Stream Mining: Basic Methods and Techniques. Technical report, Rheinisch-Westfälische Technische Hochschule Aachen.

[McAfee and Brynjolfsson 2012] McAfee, A. and Brynjolfsson, E. (2012). Big data: the management revolution. *Harvard business review*, 90(10):61–68.

[Microsoft 2015] Microsoft (2015). Microsoft StreamInsight.

[Roriz Junior 2017] Roriz Junior, M. (2017). *DG2CEP: An On-line Algorithm for Real-Time Detection of Spatial Clusters from Large Data Streams Through Complex Event Processing*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.

[Roriz Junior et al. 2019] Roriz Junior, M., de Oliveira, R. P., Carvalho, F., Lifschitz, S., and Endler, M. (2019). Mensageria: A smart city framework for real-time analysis of traffic data streams. In Oliveira, J., Farias, C. M., Pacitti, E., and Fortino, G., editors, *Big Social Data and Urban Computing*, pages 59–73, Cham. Springer International Publishing.

[Suhothayan et al. 2011] Suhothayan, S., Gajasinghe, K., Loku Narangoda, I., Chaturanga, S., Perera, S., and Nanayakkara, V. (2011). Siddhi: A Second Look at Complex Event Processing Architectures. In *Proceedings of the 2011 ACM workshop on Gateway computing environments - GCE '11*, page 43, New York, New York, USA. ACM Press.

[van der Zee and Scholten 2013] van der Zee, E. and Scholten, H. (2013). Application of geographical concepts and spatial technology to the internet of things. WorkingPaper 2013-33, Faculty of Economics and Business Administration.

[Wu et al. 2006] Wu, E., Diao, Y., and Rizvi, S. (2006). High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 407–418, New York, NY, USA. ACM.

[Yadranjiaghdam et al. 2017] Yadranjiaghdam, B., Yasrobi, S., and Tabrizi, N. (2017). Developing a real-time data analytics framework for twitter streaming data. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 329–336.

[Yanlei Diao Neil Immerman and Gyllstrom 2007] Yanlei Diao Neil Immerman and Gyllstrom, D. (2007). SASE+: An Agile Language for Kleene Closure over Event Streams. Technical Report UM-CS-07-03, Department of Computer Science, University of Massachusetts Amherst.

[Zhao et al. 2017] Zhao, X., Garg, S., Queiroz, C., and Buyya, R. (2017). Chapter 11 - a taxonomy and survey of stream processing systems. In Mistrik, I., Bahsoon, R., Ali, N., Heisel, M., and Maxim, B., editors, *Software Architecture for Big Data and the Cloud*, pages 183 – 206. Morgan Kaufmann, Boston.

[Zheng et al. 2014] Zheng, Y., Capra, L., Wolfson, O., and Yang, H. (2014). Urban Computing. *ACM Transactions on Intelligent Systems and Technology*, 5(3):1–55.

BIO

Marcos Roriz. É professor adjunto da Universidade Federal de Goiás (UFG). Possui graduação em Ciência da Computação e mestrado pela UFG, e doutorado em Informática pela PUC-Rio. Suas áreas de interesse concentram-se em Sistemas Inteligentes de Transportes e Sistemas Distribuídos, com foco em algoritmos e plataformas de middleware para cidades inteligentes, inteligência artificial, computação móvel e computação ubíqua. Curriculo Lattes: `http://lattes.cnpq.br/1356289387726731`

Álan Guedes. É pesquisador de pós-doutorado no laboratório TeleMídia da PUC-Rio. Obteve graduação (2009) e mestrado(2012) pela UFPB, e doutorado em Informática (2017) pela PUC-Rio. Atuou em projetos de pesquisa em vídeo interativo, como os premiados GingaStore e Brasil4. Seus interesses de pesquisa incluem multimídia, vídeo interativo, mídia imersiva e Deep Learning para Multimídia. Curriculo Lattes: `http://lattes.cnpq.br/1481576313942910`.

Fernando B. V. Magalhães. É aluno de mestrado em Informática pela PUC-Rio e pesquisador no laboratório LAC. Possui graduação em computação (2018) pela Universidade Federal do Maranhão. Atualmente desenvolve projetos de CEP distribuído e aplicado a processos bioquímicos. Curriculo Lattes: `http://lattes.cnpq.br/9225104372311945`.

Sérgio Colcher. É professor do quadro principal da PUC-Rio desde 2001 e coordenador do laboratório TeleMídia. Obteve os títulos de Engenheiro de Computação (1991), Mestre (1993) e Doutor em Informática (1999), todos pela PUC-Rio, além do Pós-Doutorado (2003) no ISIMA (Institute Supérieur D'Informatique et de Modelisation des Applications, França). Suas áreas de interesse incluem redes de computadores, análise de desempenho de sistemas computacionais, sistemas multimídia/hipermídia e sistemas de TV digital.. Curriculo Lattes: `http://lattes.cnpq.br/1104157433492666`.

Markus Endler. Bacharel em Matemática e Mestre em Informática pela PUC-Rio (1984 e 1987), obteve o título de Dr.rer.nat. em Informatica da Technische Universitat Berlim (1992), e o título de Professor livre-docente pela Universidade de São Paulo (2001). Atualmente é professor associado da PUC-Rio. Tem experiência na área de Sistemas Distribuídos, com ênfase em: computação móvel e ubíqua, protocolos distribuídos para redes móveis, middleware, ciência de contexto e colaboração móvel. Currículo Lattes:`http://lattes.cnpq.br/6505039023842313`

Capítulo 3

Teoria e Prática de Microserviços Reativos: Um Estudo de Caso na Internet das Coisas

Cleber Santana[2,1], Leandro Andrade[1], Brenno Mello[1], José Sampaio[1], Ernando Batista[2], Cássio Prazeres[1]

[1]Departamento de Ciência da Computação (DCC), UFBA

[2]Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA)

(leandrojsa, brenno.mello, jose.sampaio prazeres)@ufba.br,

dsandrade@dcc.ufba.br, (cleberlira, enrnando.passos)@ifba.edu.br

Abstract

Microservices has recently been employed at Cloud Computing to support the construction of large-scale systems that are resilient, resilient, and better tailored to meet today's demands. In the Internet of Things (IoT) a set of intelligent applications can be built in many different scenarios and that can impact the daily routine of people's lives. The development of applications and services at IoT brings challenges such as deployment, elasticity and resilience. Microservices implemented with resilience, elasticity, and message-driven features are considered reactive microservices. Therefore, in this chapter we introduce the concept of Reactive Microservices and illustrate a case study for building an IoT application.

Resumo

Microservices recentemente tem sido empregado na Cloud Computing para suportar a construção de sistemas de larga escala que sejam resilientes, elásticas e melhor adaptados para atender as demandas atuais. Na Internet das Coisas (IoT) um conjunto de aplicações inteligentes podem ser construídas nos mais diferentes cenários e que podem impactar na rotina diária da vida das pessoas. O desenvolvimento de aplicações e serviços na IoT traz desafios como implantação, elasticidade e resiliência. Microserviços implementados com as caraterísticas de resiliência, elasticidade e dirigido a mensagens são considerados Microserviços reativos. Portanto, neste capítulo introduzimos o conceito de Microserviços reativos e ilustramos um estudo de caso para construção de uma aplicação IoT.

3.1. Introdução

Atualmente, algumas aplicações estão utilizando Microserviços em domínios como o da IoT e Mobile [Francesco et al. 2017] a fim de apoiar a construção de sistemas de grande porte que sejam mais robustos, resilientes, elásticos e melhor adaptados para atender as demandas atuais. Os Microserviços visam construir, gerenciar e projetar arquiteturas de pequenas unidades autônomas [Fowler and Lewis 2014]. As aplicações baseadas na arquitetura de Microserviços são poliglotas, ou seja, são construídas a partir da linguagem de programação da escolha do desenvolvedor e as implementações são realizadas continuamente [Humble and Farley 2011], algumas dezenas de vezes por dia [Trojak 2018] ou em alguns casos centenas de vezes [BLOOM 2013], tornando o uso dessas aplicações mais dinâmica pelos inúmeros usuários.

Os Microserviços são arquiteturas nativas da nuvem, e as aplicações baseadas nesse estilo arquitetural são projetadas para serem resilientes a incidentes e interrupções na infraestrutura, no entanto, em alguns cenários esses aplicativos podem falhar e ficar indisponíveis por horas e, em alguns casos, dias CircleCI [CI 2015]), que tem um impacto negativo na cadeia produtiva dessas organizações.

Na IoT, o número de dispositivos atualmente é de 8,3 bilhões e estima-se que até 2025 esse número chegue a 21,5 bilhões [Lasse Lueth 2018]. Esses novos dispositivos poderão interagir fornecendo novos serviços e aplicações em diferentes domínios e ambientes, gerando valor agregado para o usuário. Os dispositivos da IoT, juntamente com suas tarefas, constituem aplicações específicas de domínio (por exemplo, cidades inteligentes, casas inteligentes, transportes inteligentes, agricultura de precisão) e mercados horizontais (por exemplo, computação ubíqua e serviços analíticos), que constituem serviços independentes de domínio [Shahid and Aneja 2017]. Por exemplo, no cenário da piscicultura, o monitoramento de variáveis (por exemplo, hidrológicas, hidráulicas e de qualidade) pode melhorar o cultivo das espécies. Nesse cenário, os Microserviços podem ser distribuídos na borda da rede, permitindo elasticidade e resiliência ao sistema [de Santana et al. 2019]. Resiliência é a capacidade de resistir a falhas externas e internas; e elasticidade refere-se à capacidade de resposta do sistema de acordo com a variação da demanda [da Rosa Righi et al. 2018]. O sistema deve usar mensagens assíncronas para garantir padrões de comunicação de baixo acoplamento, isolamento e transparência de localização para [Bonér 2017]. A conformidade com essas propriedades transforma os Microserviços em Microserviços Reativos [de Santana et al. 2019].

Os Microserviços reativos podem aumentar a confiabilidade das aplicações. A confiabilidade é um requisito prioritário para aplicações IoT que estão preocupadas com qualidade de serviço (QoS) [White et al. 2017]. Confiabilidade refere-se à capacidade de um sistema executar funções sob condições especificadas por um período de tempo especificado [for Standardization/International Electrotechnical Commission et al. 2011]. Além disso, a confiabilidade leva em consideração quatro sub-características, tais como: **Maturidade**, que é a capacidade de uma aplicação evitar falhas decorrentes de defeitos na aplicação; **Disponibilidade**, a capacidade de uma aplicação estar operacional e acessível quando necessário para uso; **Tolerância a falhas**, a capacidade de uma aplicação de operar como pretendido apesar da presença de falhas de hardware ou software e **Capacidade de recuperação**, que se refere à capacidade de uma aplicação recuperar os dados direta-

mente afetados, no caso de falha, e restabelecer ao estado desejado da aplicação.

As aplicações IoT geralmente são distribuídas em dispositivos móveis e servidores implantados na Cloud Computing/Fog Computing/ Edge Computing. Nesse contexto, uma aplicação IoT pode falhar por diferentes motivos: falhas de travamento, em que um dispositivo ou servidor para e precisa de uma reinicialização; falhas de omissão, em que um dispositivo ou servidor para de enviar e receber mensagens; falhas de temporização, em que uma resposta de dispositivo ou servidor está muio lenta [White et al. 2017].

Espera-se que as aplicações IoT evoluam continuamente para lidar com novos serviços. As arquiteturas tradicionais limitariam a capacidade de um sistema IoT de evoluir, já que as mudanças exigiriam o reinício do sistema [Dragoni et al. 2017]. Portanto, isso afeta a disponibilidade de aplicações IoT.

Avaliação de desempenho para aplicações IoT ainda é uma questão aberta [Udoh and Kotonya 2018]. Para obter confiabilidade, o sistema deve atender a padrões de desempenho, uma vez que isso pode afetar a disponibilidade das aplicações IoT. Nesse contexto, consideramos que essas falhas afetam a disponibilidade das aplicações IoT, onde arquiteturas tradicionais teriam dificuldades em lidar com esses problemas. Portanto, isso leva à necessidade de projetar esses aplicações com novas abordagens arquiteturais. Em estudos recentes, [de Santana et al. 2018] afirmaram que Microserviços foi aplicado em aplicações IoT e eles podem ser implantados em containers. Sistemas de gerenciamento de containers, como o Docker[1], e sistemas de orquestração, como o Kubernetes[2], controlam aplicações e provisionam dinamicamente seus recursos, que podem ser extremamente escalonáveis, confiáveis e reativos.

Nesse contexto, este capítulo apresenta uma arquitetura baseada em Microserviços [de Santana et al. 2019] para suportar o desenvolvimento de aplicações IoT reativas. Para prover o desenvolvimento dessas aplicações, nós também apresentamos uma plataforma baseada no Vert.x[3] que pode ser implantada em dispositivos (i.e com baixo poder computacional) e servidores que estejam localizados na borda da rede, na névoa ou nuvem.

3.2. Infraestrutura para Internet das Coisas

Antes da IoT, os dados produzidos na Internet, em sua grande maioria, eram dependentes de humanos. Este fato determina algumas características estão associadas a natureza humana, como limitação de tempo, atenção e acurácia. Por sua vez, essas características possuem diferenças em dados produzidos por coisas, as quais tem seu comportamento determinado por um programa e seus componentes eletrônicos. Kevin Ashton [Ashton 2009], o pesquisador que introduziu o termo "Internet of Things" em 1999, argumentou que tecnologia de sensores e *tags* RFID podem ser capazes de rastrear e processar informações reduzindo significativamente perdas e custos. Desde então, as tecnologias para implantar coisas na Internet foram desenvolvidas, como conectividade sem fio, protocolos de comunicação, hardware para dispositivos e plataformas para prototipagem e desenvolvimento de sistemas eletrônicos, criando assim muitas possibilidades de aplicações e soluções para IoT [Gérald 2010].

[1]https://www.docker.com/get-started

[2]https://kubernetes.io/pt/

[3]https://vertx.io/

Conceitos e tecnologias inspirados na IoT foram desenvolvidos em várias aplicações do mundo real, como, por exemplo: carros autônomos, gerenciamento eficiente de energia, gerenciamento de ambientes inteligentes, gerenciamento inteligente de tráfego e monitoramento ambiental.

De acordo com *The Internet of Things – Architecture* (IoT-A) [Bauer et al. 2013], do ponto de vista da IoT existem três tipos básicos de dispositivos:

- **Sensores:** fornecer informações, conhecimentos ou dados sobre a entidade física monitorada. Por exemplo, um dispositivo mede a temperatura de uma sala ou a câmera habilitada para reconhecimento de rosto são sensores. Os dados produzidos pelos sensores funcionam como entradas para sistemas IoT e também podem ser gravados para recuperação posterior;
- ***Tags*:** são aplicados para identificar dispositivos físicos em sistemas IoT. Geralmente, as tags são fisicamente anexadas, como códigos de barras, códigos QR e RFID. Eles são usados para melhorar a automatização da identificação de dispositivos IoT pelos sistemas IoT;
- **Atuadores:** podem alterar o estado físico dos dispositivos IoT, como ligar/desligar, ações de movimento ou alterar o estado da forma. Os atuadores, em geral, agem em ambientes ou coisas físicas, e sua ação geralmente promove mudanças no local/coisa da operação. Um dispositivo IoT capaz de ligar/desligar luzes ou braço mecânico em uma fábrica de automóveis são exemplos de atuadores.

O desenvolvimento da IoT depende do projeto de novas aplicações e modelos de negócios. Em estudos recentes, [Khan et al. 2012] e [Al-Fuqaha et al. 2015] dividiram a estrutura da IoT em cinco camadas. A Figura 3.1 ilustra essas camadas, que são descritas abaixo:

- **Camada de Percepção:** contém as "coisas" da IoT, que são sensores (por exemplo, sensores de temperatura), atuadores (por exemplo, relés) e tags (por exemplo, RFID), conforme ilustrado na Figura 3.1. Esses dispositivos são a base da infraestrutura da IoT, a partir da qual os sistemas interagem com o ambiente físico;
- **Camada de Conexão:** transfere as informações da camada Percepção para a camada de Middleware. Nos sistemas de IoT, geralmente a transmissão pode ser sem fio ou com fio. Além disso, podemos usar a tecnologia 4G, Bluetooth e infravermelho, dependendo do sensor, conforme mostrado em Figura 3.1;
- **Camada de Middlware:** fornece uma interface entre a camada de percepção (através da camada de Conexão) e o restante do sistema de IoT. Essa camada é responsável pelo gerenciamento de dispositivos IoT e pelos dados coletados. O middleware também fornece interface de acesso aos dados da camada superior da IoT;
- **Camada de Aplicação:** fornece o gerenciamento global do sistema de IoT, recebendo e enviando informações para dispositivos e usuários. A Figura 3.1 mostra a camada de Aplicação através de exemplos de aplicação como saúde inteligente,

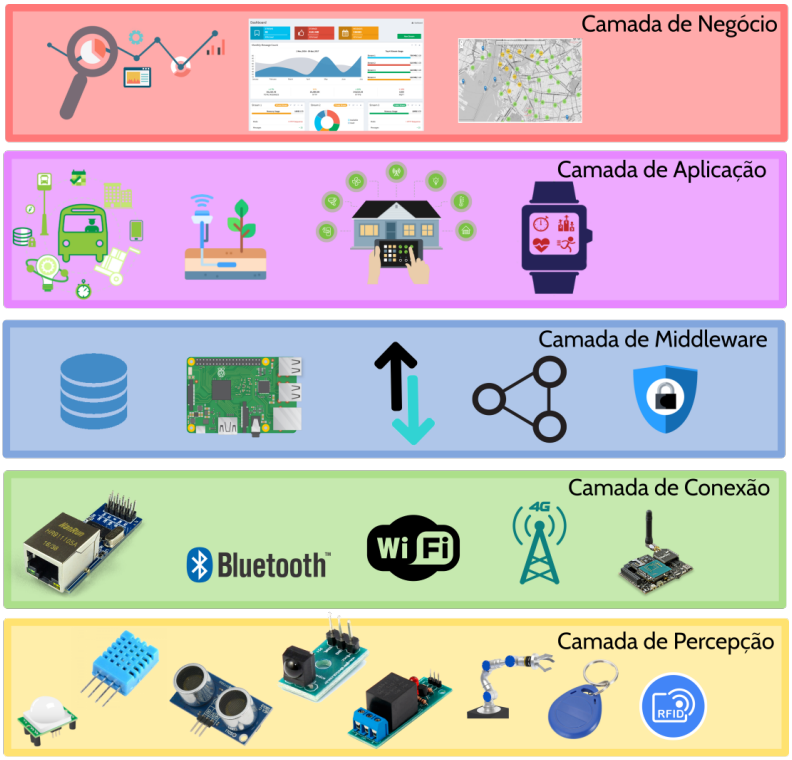


Figure 3.1. Representação de camadas da arquitetura IoT. Adaptado de [Andrade et al. 2018]

agricultura inteligente, casa inteligente, cidade inteligente, transporte inteligente, etc;

- **Camada de Negócio:** gerencia as atividades e serviços gerais do sistema IoT [Al-Fuqaha et al. 2015]. Como é ilustrado na Figura 3.1, essa camada é responsável por fornecer modelo de negócios, gráficos, fluxogramas e outras visualizações com base nos dados recebidos da camada Aplicação.

Os progressos na IoT resultaram na criação de vários ecossistemas paralelos, sem uma integração global [Sundmaeker et al. 2010]. Essa limitação dificulta o acesso a diferentes dispositivos e a criação de aplicações mais poderosos, integrando sistemas distintos. O Cluster Europeu de Pesquisa sobre a Internet das Coisas (IERC) publicou um relatório [Serrano et al. 2015a] analisando os principais desafios de pesquisa, melhores práticas, recomendações e próximos passos para proporcionar interoperabilidade na Internet das Coisas.

Além da necessidade de fornecer interoperabilidade, as soluções IoT geralmente requerem funcionalidades para registrar, anotar para gerenciar dados e dispositivos [Serrano et al. 2015b]. Além disso, os serviços IoT devem garantir a entrega e o uso desses dados para usuários, aplicações ou até outros serviços [Bassi et al. 2013]. Várias soluções baseadas em Computação em Nuvem foram propostas pela indústria e pela academia com foco na prestação de serviços e interoperabilidade. Na Seção 3.2.1 é discutido os principais aspectos das soluções de IoT baseadas na Computação em Nuvem.

A crescente demanda por mais eficiência no processamento local e mais maneiras de proteger os dados e a tecnologia IoT antes dos dados irem para a nuvem geraram novas alternativas de infraestrutura em sistemas IoT. Assim, alternativas surgiram buscando usar a capacidade de processamento, armazenamento e acesso dos dispositivos locais, na chamada Fog Computing ou Computação em Névoa. Na Seção 3.2.2, apresentamos características sobre as soluções de IoT baseadas no paradigma de Computação em Névoa. Por fim, algumas soluções IoT não utilizam conectividade externa para prover seu funcionamento, usando de modo exclusivo os dispositivos locais na chamada Computação na Borda (Edge Computing). Os detalhes sobre esse tipo de infraestrutura é apresentado na Seção 3.2.3.

3.2.1. Computação em Nuvem

Com evolução da Internet com alta demanda como segurança, armazenamento e processamento de dados, a Cloud Computing ou Computação em Nuvem transforma uma solução amplamente usada para esses problemas. O conceito de Computação em Nuvem refere-se a uma extensão da computação em grade, computação distribuída e computação paralela com um ambiente de virtualização e extensibilidade [Zhang et al. 2010].

As soluções de Computação em Nuvem fornecem um modelo que permite o acesso configurável dos recursos computacionais oferecidos como serviços [Borgia 2014]. Em geral, o gerenciamento desses recursos é controlado pelos usuários, onde eles pagam apenas o valor do uso efetivo, através do modelo *pay-as-you-go* [Cavalcante et al. 2016]. Esse tipo de solução é usado pela maioria das aplicações Web, pois fornece aspectos como alta disponibilidade, tolerância a falhas e escalabilidade de processamento e armazenamento.

Embora os recentes avanços nos dispositivos IoT, a computabilidade e o armazenamento sejam recursos limitados na maioria deles [Botta et al. 2016]. A associação do paradigma de Computação em Nuvem pode suportar esses tipos de limitação da IoT porque a nuvem dispõe de uma infraestrutura global de alta capacidade de computabilidade e armazenamento, como exemplo de processamento de uma análise de dados de grande volume de dados. Além disso, essa associação pode melhorar aspectos como privacidade, desempenho e confiabilidade de plataformas IoT.

Muitas iniciativas estão desenvolvendo arquiteturas e plataformas para a IoT usando o paradigma de Computação em Nuvem, em que o acesso e os dados gerados pelos sensores IoT são direcionados para a nuvem. Por exemplo, grandes provedores dessa tecnologia, como Amazon e Google, oferecem serviços em nuvem com suporte específico e especial aos sistemas de IoT [Cavalcante et al. 2016]. Além disso, existem várias soluções de plataformas em nuvem capazes de oferecer alta escalabilidade, armazenamento e processamento, bancos de dados distribuídos, processamento em tempo real, gerenciamento, monitoramento e implantação [Díaz et al. 2016].

A Figura 3.2 mostra uma visão geral do paradigma de Computação em Nuvem para a Internet das Coisas: Cloud of Things (CoT) [Distefano et al. 2012]. No paradigma CoT, os dados de coisas geralmente são coletados por um middleware de sensores e enviados para uma nuvem pública para processamento, armazenamento e entrega de serviços. Não há processamento de dados ou entrega de serviços na borda da rede, porque todos os

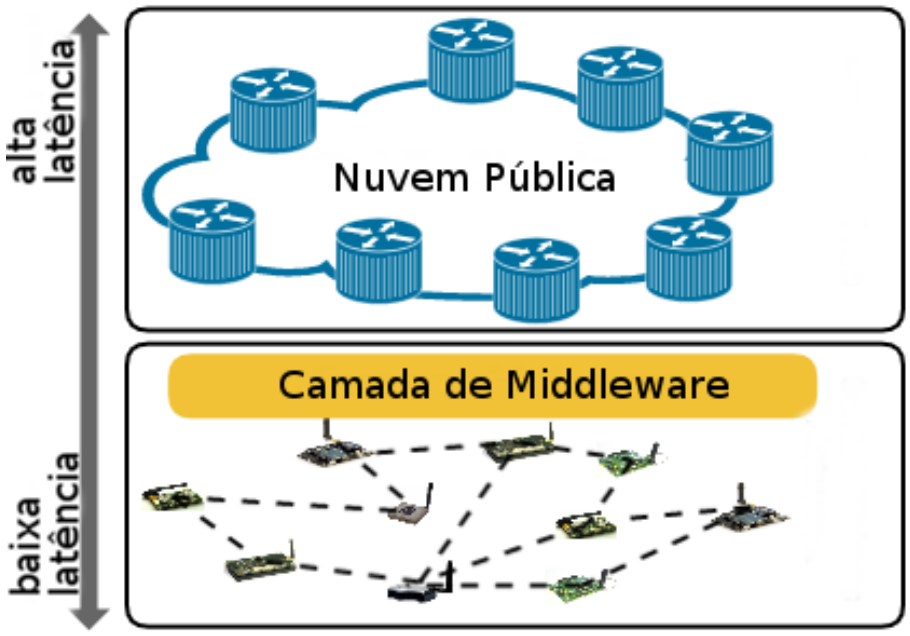


Figure 3.2. Visão Geral da Nuvem de Coisas

dados são processados por servidores remotos na nuvem.

Existem alguns desafios na integração da IoT e da computação em nuvem como a necessidade de melhorar a eficiência dos recursos da nuvem (por exemplo: rede, processamento e armazenamento), padronização de dados e serviços, segurança e privacidade de dados, preocupação com a confiabilidade (os dispositivos IoT podem ficar indisponíveis por diversas razões) e alta heterogeneidade dos ambientes IoT e de nuvem [Cavalcante et al. 2016]. De acordo com [Abdelshkour 2015] o paradigma CoT tem algumas limitações como: a conectividade com a nuvem é uma precondição, mas alguns sistemas de IoT precisam funcionar, mesmo quando a conexão está temporariamente indisponível; alta demanda por largura de banda, como resultado do envio de todos os dados pelos canais da nuvem; e, tempo de resposta lento (alta latência) e escalabilidade limitada como resultado da dependência de servidores remotos.

Nesse contexto, algumas alternativas foram propostas para evitar algumas limitações do paradigma da Computação em Nuvem. Na Seção 3.2.2, apresentamos o Fog Computing ou Computação em Névoa, que estende o paradigma de Computação em Nuvem usando a capacidade de processamento, armazenamento e comunicação dos dispositivos da rede local e tem sido adequado aos requisitos de sistemas IoT.

3.2.2. Computação em Névoa

Com o objetivo de criar alternativas para limitação da Computação em Nuvem, [Bonomi et al. 2012] propuseram o paradigma Fog Computing (ou Computação em Névoa), que, por definição, aproxima algumas operações de computação e armazenamento da borda da rede. Na Computação em Névoa, as operações são distribuídas entre dispositivos computacionais, transferindo parte da complexidade da nuvem para a borda.

Contudo, a adoção da Computação em Névoa não exclui a Computação em Nu-

vem. Esta traz características como baixa latência, reconhecimento de local, suporte à mobilidade, forte presença de aplicativos de streaming e em tempo real e escalabilidade para grande número de nós de nevoeiro [Bonomi et al. 2012]. Além disso, por motivos como requisitos de segurança ou privacidade e/ou indisponibilidade de um servidor em nuvem. A Computação em Névoa e a Computação em Nuvem são complementares em vários aspectos, pois os dispositivos de borda continuam a operar mesmo sem conectividade e, quando é possível, os dados são enviados para a nuvem. Além disso, usuários ou aplicativos locais podem acessar dispositivos e dados diretamente na rede interna e quando estes são remotos podem acessar através de servidores em nuvem, proporcionando melhor Qualidade de Experiência (QoE).

Em uma típica implementação da IoT, os dados coletados dos dispositivos são armazenados e processados em servidores na nuvem. Embora esse tipo de abordagem seja comumente usado, ele tem algumas limitações [Abdelshkour 2015]: a conectividade com a nuvem é uma pré-condição e alguns sistemas de IoT precisam funcionar, mesmo quando a conexão está temporariamente indisponível; alta demanda por largura de banda, como resultado do envio de todos os dados pelos canais da nuvem; tempo de resposta lento (alta latência) e escalabilidade limitada como resultado da dependência de servidores remotos hospedados em data centers centralizados.

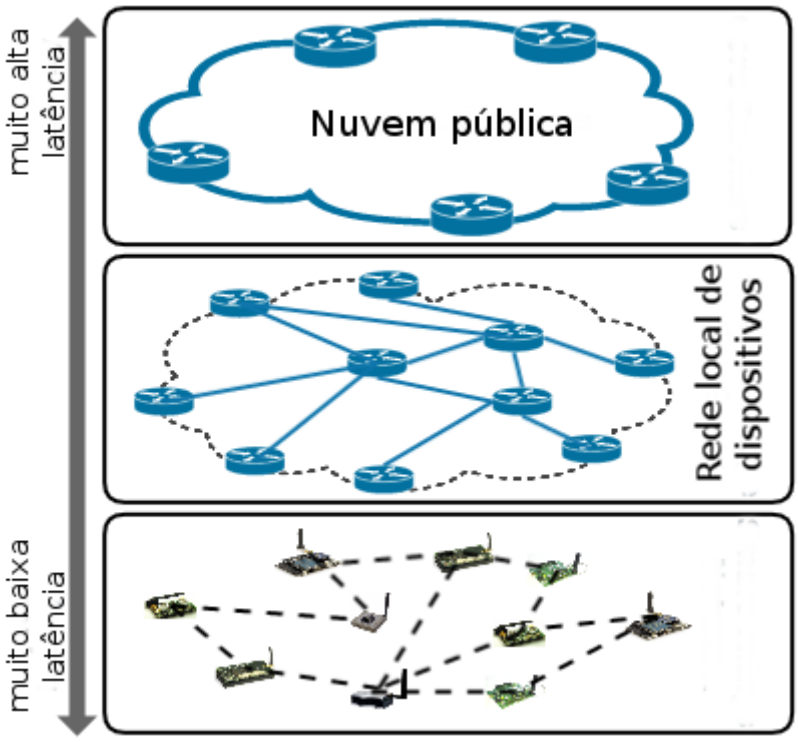


Figure 3.3. Visão geral da IoT na Computação em Névoa.

Na IoT, a Computação em Névoa visa aproveitar um pouco da complexidade da nuvem e aproximá-la de dispositivos, aplicações e/ou usuários, como uma "nuvem local e privada". A Figura 3.3 ilustra a visão geral de uma rede de equipamentos físicos (servidores, gateways, dispositivos etc.) com capacidades para processamento de dados locais e paralelos, prestação de serviços para IoT e habilitado para enviar dados resultantes desse

processamento para infraestruturas virtuais (nuvens públicas), a fim de atender aos requisitos e características da Computação em Névoa como descrito anteriormente.

Um caso que exemplifica tal associação é Névoa das Coisas (Fog of Things - FoT), na qual a Prazeres and Serrano [Prazeres and Serrano 2016] propuseram uma nova maneira de projetar e implementar plataformas IoT usando Computação em Névoa. O paradigma FoT vai além da Computação em Névoa em algumas direções, como:

- Usando toda a capacidade de processamento da borda da rede, executando o processamento de dados e a entrega de serviços em dispositivos, gateways (servidores muito pequenos) e pequenos servidores locais;
- Definindo perfis para gateways e servidores na borda da rede, a fim de definir os serviços de IoT a serem entregues;
- Distribuindo esses serviços IoT na borda da rede por meio de um middleware orientado a mensagens e serviços.

Por fim, no paradigma FoT, parte da capacidade de processamento de dados e das operações de entrega de serviços são processadas localmente em pequenos servidores combinado com operações em servidores na nuvem [Andrade et al. 2018].

3.2.3. Computação na Borda

A Computação em Borda ou Edge Computing refere-se às tecnologias que permitem que a computação e armazenamento seja executada na borda da rede, para que tais processos aconteça perto de fontes de dados [Shi and Dustdar 2016]. A adoção de tal paradigma prove benefícios como: baixa latência, mobilidade dos dispositivos, segurança e independência de conexão com a Internet.

Existem sistemas IoT que possuem sua funcionalidade e ciclo de vida envolvendo somente dispositivos locais. Esse tipo de solução é baseado na Computação na Borda restringe ao recursos para operação dos sistemas a servidores e dispositivos locais. Em alguns caso tal tipo de situação pode ocorrer de modo temporário, quando há perda de conectividade com a internet, na qual o sistema pode se adaptar e funcionar somente com seus recursos internos.

A Computação na Borda permite um grande número de aplicações, incluindo comunicação veicular, cidades inteligentes, smart grid, redes de sensores sem fio incorporadas a atuadores, monitoramento de tráfego rodoviário, monitoramento de tubulações, parques eólicos, sistema de semáforo inteligente, monitoramento ferroviário, sistemas de controle industrial e o aplicações em explorações de petróleo e gás [Bilal et al. 2018].

3.2.4. Contêiners

As aplicações IoT podem ser disponibilizadas na Nuvem/Névoa/Borda a partir da adoção de contêiners. Os ontêiners fornecem isolamento das aplicações, permitindo realizar atualizações e escalar as aplicações. As duas tecnologias de container a seguir (Kubernetes e Docker) tem sido adotadas na implantação de aplicações IoT.

3.2.4.1. Kubernetes

O Kubernetes é uma plataforma de automação para implantar, dimensionar e operar contêiners das aplicações por meio de clusters de host, e foram originalmente criados pelo Google [kubernetes 2019]. O Kubernetes trabalha com um conceito semelhante de Nós. Não há hierarquia de Nós: cada Nó um é uma instância capaz de executar tarefas em contêiners. Os Nós são controlados por um Nó controlador, responsável por monitorar os Nós e manter a lista de Nós em sincronia com as máquinas do cluster.

Para implantar as apĺicações IoT no Kubernetes é possível utilizar o Openshift. O Openshift é uma plataforma de conteinerização. O Openshift possui entidades, tais como: **configuração de build**, **configurações de deployment**, **Pods**, **Serviços e Rotas**.

A configuração de build é a parte do processo que se constrói as imagens conteinerizadas que serão usadas pelo Openshift para instanciar os diferentes contêiners que compõem a aplicação. Esta "entidade" pode trabalhar com o próprio docker, por exemplo, construindo a imagem do Dockerfile.

A configuração de Deployment é a parte do processo em que se define a instância da imagem construída pelo build, definindo qual imagem deve ser criada e quantas instâncias devem ser mantidas funcionando, quando a implementação de um container deve ser feita, quantas réplicas devem existir, atuando como um controlador de réplicas programável para ser tanto ajustável manualmente quanto para realizar o auto-scaling e também serve para realizar checagens da 'saúde' das réplicas para que não exista uma incidência de containeres inoperantes.

Os Pods são um grupo de um ou mais contêiners, porém normalmente são compostos por somente um container. A orquestração, escala e administração dos Pods é delegada ao Kubernetes.

Os serviços nos permitem comunicar com os pods sem depender de seus endereços, mas usando o endereço virtual do serviço. Um serviço atua como um proxy na frente de um grupo de pods e podem implementar uma estratégia de balanceamento de carga.

Para instalar o Openshift utiliza-se o Minishift:

```
https://github.com/minishift/minishift
```

O Minishift requer um hypervisor para executar a máquina virtual que contém o OpenShift. Dependendo do sistema operacional utilizado, pode-se escolher entre algumas alternativas de hypervisors (consulte o guia de instalação do Minishift para obter detalhes). Para instalar o Minishift, basta baixar o arquivo mais recente para o sistema operacional utilizado pelo usuário na página de downloads do Minishift:

```
https://github.com/minishift/minishift/releases
```

Descompacte o arquivo no local de preferência e adicione ao PATH. Assim que estiver instalado basta usar o comando:

```
minishift start
```

A partir daí será possível conectar à sua instância do OpenShift no endereço

```
https://192.168.64.12:8443
```

Caso seja necessário validar o SSL faça login como developer/developer

Também há necessidade de um openshift client, disponível em:

```
https://github.com/openshift/origin/releases/tag/v3.11.0
```

Descompacte na área de preferência e adicione o binário ao PATH, conecte-se ao openshift com o comando:

```
oc login https://192.168.64.12:8443 -u developer -p developer
```

Para criar um projeto use os seguintes comandos:

```
oc new-project nome-projeto

oc policy add-role-to-user admin developer -n nome-projeto

oc policy add-role-to-user view -n nome-do-projeto -z default
```

E acesse-o em:

```
open https://192.168.64.12:8443/console/project/<nome do projeto>
```

E pode-se observar a tela de projeto. Usando um plugin do maven, se criam os pods com o comando:

```
mvn fabric8:deploy -Popenshift
```

O primeiro build toma um pouco mais de tempo, mas as próximas versões serão mais rápidas pois muitos dados estarão em cache.

3.2.4.2. Docker

O Docker é um projeto de código aberto e seu objetivo é criar e manter contêiners [Bernstein 2014]. É responsável por armazenar vários serviços isoladamente do Sistema Operacional, como: servidor web, banco de dados, aplciações, entre outros. Seu back-end é baseado em Linux Contêiners.

Os containers do Docker são criados a partir de imagens. Uma imagem pode incluir apenas os fundamentos do sistema operacional ou pode consistir em uma pilha de aplicações pré-criadas. Ao construir as imagens, cada comando executado (por exemplo, apt-get install) forma uma nova camada. Os comandos podem ser executados manual ou automaticamente a partir do script Dockerfiles.

Dockerfile é um script composto por vários comandos listados sucessivamente, para que seja possível executar ações em uma imagem base para criar uma nova imagem. O Dockerfile é usado para organizar artefatos e simplificar o processo de implantação.

A opção de utilizar o Docker para implantar Microserviços vem ganhando espaço. Isso se deve à fácil escalabilidade, isolamento e facilidade de compartilhamento das imagens utilizadas pela plataforma (Se houver alguma dúvida com relação aos termos utilizados, consulte:

```
https://docs.docker.com/get-started/
```

Por quê utilizar o Docker em vez de uma VM com Open Shift? Existem diversas respostas para essa pergunta, algumas podem ser encontradas em: Mas elas revolvem basicamente em torno de três pontos: 1- Limitações de Hardware: Enquanto uma VM utiliza um sistema operacional "convidado" ou "hóspede" com acesso virtual ao sistema operacional "anfitrião" pelo Hypervisor, os containers do docker rodam nativamente no Linux, compartilhando o kernel com o sistema operacional "anfitrião" e rodando processos discretos, sem utilizar muita memória, se tornando uma opção mais leve do que as VM's que geralmente consumem bastante memória.

2- Facilidade de Cooperação: Um daemon do Docker pode ser conectado tanto no mesmo sistema quanto em outros daemons em outros lugares, permitindo assim que mais pessoas trabalhem num projeto só de seus computadores. Quado isso acontece, utiliza-se o Docker Swarm, onde existem diversos trabalhadores e alguns gerentes, e todos os membros do projetos são Docker daemons que se comunicam pela API do Docker, deixando assim o trabalho mais dinâmico e funcionando um pouco como um Github, onde se poderiam dar push e pull de várias imagens do projeto.

3- Facilidade de Remodelar/Reconstruir o projeto: Além da facilidade de comunicação e teste do projeto entre seus desenvolvedores, os containers do Docker são efêmeros e facilmente destruídos ou remodelados, então ao encontrar um erro ao invés de precisar interromper toda a maquina para consertá-lo, só precisaríamos interromper o container específico que contém a imagem problemática, consertá-la e rodar novamente outro container, atualizado.

É de suma importância que sejam instalados tanto o docker quanto o docker compose, que seja utilizado um ambiente rodando alguma distribuição do linux (neste capítulo foi utilizado o ubuntu 18.04) e possuir o git instalado.

Primeiramente é necessário que haja uma aplicação para realizar o deployment no docker.

```
https://github.com/Rck-Sanchez/microservices-in-docker
```

Para baixar o repositório basta utilizar o comando:

```
git clone https://github.com/Rck-Sanchez/microservices-in-docker.git
```

Tutorial de instalação para o docker (versão gratuita):

Primeiro, atualize sua lista atual de pacotes com o comando:

```
sudo apt update
```

Em seguida, instale alguns pacotes de pré-requisitos que permitem que o apt utilize pacotes via HTTPS:

```
sudo apt install apt-transport-https ca-certificates
curl software-properties-common
```

Então adicione a chave GPG para o repositório oficial do Docker em seu sistema:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg sudo apt-key
    add -
```

Adicione o repositório do Docker às fontes do APT:

```
sudo add-apt-repository deb arch=amd64
https://download.docker.com/linux/ubuntu bionic stable"
```

A seguir, atualize o banco de dados de pacotes com os pacotes Docker do repositório recém adicionado:

```
sudo apt update
```

Certifique-se de que a instalação se dará a partir do repositório do Docker em vez do repositório padrão do Ubuntu:

```
apt-cache policy docker-ce
```

Espera-se uma saída como esta, embora o número da versão do Docker possa estar diferente:

```
Output of apt-cache policy docker-ce
docker-ce Installed (none) Candidate 18.03.1-ce-3-0-ubuntu
Version table 18.03.1-ce-3-0-ubuntu 500 500
https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages
```

Instale o Docker:

```
sudo apt install docker-ce
```

O Docker agora deve ser instalado, o daemon iniciado e o processo ativado para iniciar na inicialização. Verifique se ele está sendo executado:

```
sudo systemctl status docker
```

A saída deve ser semelhante à seguinte, mostrando que o serviço está ativo e executando:

```
Output docker.service-Docker Application Container
Engine Loaded
loaded /lib/systemd/system/docker.service enabled
vendor preset: enabled) Active active (running)
since Thu 2018-07-05 15:08:39 UTC 2min 55s ago
Docs https://docs.docker.com Main PID: 10096 (dockerd)
Tasks: 16 CGroup: /system.slice/docker.service 10096
/usr/bin/dockerd -H fd:// 10113 docker-containerd
--config /var/run/docker/containerd/containerd.toml
```

Instalação do docker compose, necessário para rodar mais de um Microserviço em paralelo:

Primeiro checa-se a versão atual com o comando:

```
sudo curl -L https://github.com/docker/compose/releases/
download/1.21.2/
docker-compose- uname -s -uname -m -o

/usr/local/bin/docker-compose
```

Então veremos as permissões:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Verificamos a instalação:

```
docker-compose --version
```

O resultado deve ser parecido com:

```
docker-compose version 1.21.2, build a133471
```

Os Dockerfiles são os arquivos que guiam o que o docker deve fazer com a imagem para colocá-la num container, logo é uma das partes mais importantes do projeto.

O Dockerfile deve ser criado como um arquivo de texto, o que pode ser feito em qualquer editor, e cada parte do Microserviço deve possuir um Dockerfile diferente, especificando o que cada uma dessas partes deve fazer. Por exemplo, na pasta disponibilizada no github existem: Dockerfile-msm e Dockerfile-mscm. Ao abri-los podemos ver a estrutura:

```
FROM frolvlad/alpine-java
WORKDIR ./files
EXPOSE 8081
COPY ./files/hello-microservice-1.0-SNAPSHOT.jar /var/lib/docker
ADD ./files/hello-microservice-1.0-SNAPSHOT.jar apa.jar
CMD ["java", "-jar", "apa.jar"]
```

O comando FROM especifica uma imagem para o docker usar como base, pré programadas com algumas funções, nesse caso com o jdk 8. O comando WORKDIR especifica onde que estão os arquivos a serem trabalhados. O comando EXPOSE diz em qual porta o mundo externo terá acesso ao container. O comando COPY adiciona o programa ao path de execução do docker. O comando ADD nesse caso está sendo utilizado somente para reduzir o tamanho do nome do arquivo .jar, mas sua função é parecida com a do comando COPY. O comando CMD é o responsável por executar o .jar dentro do container. Depois de seguir todos os passos anteriores e após fazer os Dockerfiles, é importante que se realize o comando docker build microservices-repo para criar os containers e verificar se não há nenhum erro com os seus respectivos Dockerfiles.

Para um Microserviço funcionar, deve existir um fluxo de informações que o usuário deve ser capaz de acessar a partir dele, e esse fluxo requer comunicação entre as partes que compõem a arquitetura do Microserviço. Assim sendo, temos as partes do Microserviço funcionando, porém cada uma por si só, isoladas.

Para criarmos elos entre as partes deve-se utilizar um outro produto do Docker, o Docker compose. Essa ferramenta estabelece ligações entre as partes dos microserviços além de estabelecer uma ordem de operação, se necessário for. Na pasta disponibilizada no github, pode-se observar o arquivo docker-compose.yml, este seria o equivalente ao Dockerfile do docker compose, sendo assim imprescindível para o deployment do Microserviço. É importante salientar aqui que a identação neste documento é imprescindível para o funcionamento correto do docker compose. O documento docker-compose.yml tem diversas diferenças em relação ao Dockerfile, como se pode observar no trecho a seguir:

```
version: '2.2'
services:
    microservicemessage:
```

```
        container_name: microservicemessage
        build:
            context: .
            dockerfile: Dockerfile-msm
        image: microservicemessage:latest
        expose:
            - 8080
        ports:
            - 8080:8080

    microservicemessageconsumer:
        container_name: microservicemessageconsumer
        build:
            context: .
            dockerfile: Dockerfile-mscm
        image: microservicemessageconsumer:latest
        expose:
            - 8081
        ports:
            - 8081:8081
        links:
            - microservicemessage:microservicemessage
        depends_on:
            - microservicemessage
```

Em version, coloca-se a versão instalada do docker-compose na máquina, pode-se verificá-la com o comando:

```
docker-compose version
```

Em services, declaramos quais serviços compõem e após nomeá-los dizemos o nome do container em que a imagem se encontra. O comando build é quem propriamente cria os elos entre os microserviços, lendo o que está no contexto, ele localiza os dockerfiles selecionados e suas respectivas imagens e cria, por meio do campo links, uma comunicação entre os microserviços. Em caso de existir alguma espécie de ligação entre um programa e outro, como acontece no exemplo, é necessário que se adicione o link, que diz de qual serviço o que está sendo declarado precisa, e estabelece uma ordem de iniciação dos containers com o comando depends-on, pois se o serviço depende de outro para funcionar, logicamente o serviço só pode operar quando seus pré-requisitos estiverem funcionando.

Após cumprir com a criação de todos os arquivos necessários, basta ir até o terminal, utilizar o comando cd para entrar na pasta microservices-repo onde estão as Dockerfiles do seu microserviço e, neste diretório, utilizar o comando:

```
docker-compose build
```

Que faz o processo de building dos containers.

Após isso, basta utilizar o comando

```
docker-compose up
```

e esperar até que apareçam as mensagens de succeeded in deploying verticle. Quando isso acontecer, seus programas já estão rodando e você deve ser capaz de acessá-los em localhost:8080.

3.3. Programação na IoT

Ainda em 2014, quando o setor de desenvolvimento de sistemas apontou os Microserviços como uma arquitetura promissora para soluções para problemas de escalabilidade, disponibilidade e implantação, [Namiot and Sneps-Sneppe 2014] já considerava os Microserviços (consulte a seção 3.3.3) como uma adoção natural para o desenvolvimento de aplicações em um ecossistema IoT. De acordo com [Taveras Núñez 2017], a implementação de soluções no ambiente da IoT requer conceitos avançados de programação, como multithreading, propagação de alterações e computação elástica. Portanto, esses conceitos podem ser fornecidos pela Programação Reactiva (consulte a Seção 3.3.1) e pelo Sistema Reativo (veja a Seção 3.3.2).

3.3.1. Programação Reativa

De acordo com [Bainomugisha et al. 2013], a programação reativa é um paradigma de programação preocupado com a observação de fluxos de dados e a propagação de mudanças. [Bonér 2017] afirma que a programação reativa é fundamental para o design de Microserviços, pois possibilita a criação de serviços eficientes, responsivos e estáveis.

Na programação reativa, os estímulos são os dados que transitam no fluxo, chamados de fluxos. Com a programação reativa, os componentes individuais de um sistema podem ter melhor eficiência e desempenho por meio de execuções assíncronas.

Na IoT, [Taveras Núñez 2017] afirma que esse paradigma representa um método adequado para orquestrar cálculos com base em eventos assíncronos, como os fluxos de dados dos sensores presentes na IoT.

A programação reativa é um modelo de desenvolvimento orientado pelo fluxo e propagação de dados. Na programação reativa, o estímulos são os dados que transitam no fluxo, chamados fluxos. Existem muitas maneiras de implementar um modelo de programação reativa. Este tipo de programação é importante pois, a partir dela, criam-se passagens assíncronas de dados e, ao orquestra-las, o programador é capaz de criar aplicações que são um pouco mais resistentes a erros e a partir dessa assincronicidade são criados os sistemas reativos, uma ferramenta poderosa para criar ambientes que usam o conceito de Microserviços de maneira reativa e, justamente por isso, possuem qualidades que são bastante atrativas para os desenvolvedores.

A Listagem 1 exemplifica um trecho de código que adota programação reativa. Nesse exemplo, linha 8 a 18, Nesse trecho, o código está observando um *Observable* e é notificado quando os valores transitam no fluxo.

3.3.2. Sistema Reativos

Sistemas reativos, por sua vez, são um estilo arquitetônico utilizado para construir sistemas distribuídos. Utilizar sistemas reativos é interessante pois, a partir deste estilo, é possível se alcançar responsividade, criando sistemas que mantém-se funcional mesmo sob intensa requisição ou até mesmo, sob a existência de erros.

A Figura 3.6 ilustra os princípios para a construção de um sistema reativo:

- Responsivo, refere-se à capacidade de um sistema responder consistentemente no

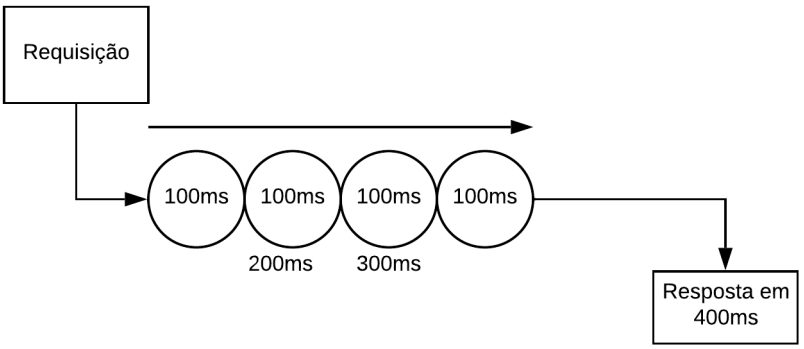


Figure 3.4. Diagrama representando o comportamento de um sistema síncrono

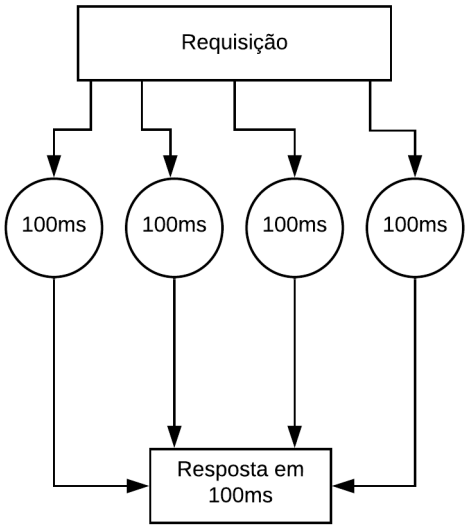


Figure 3.5. Diagrama representando o comportamento de um sistema assíncrono

menor tempo possível.

- Resiliência, esse princípio se aplica a sistemas que exigem alta disponibilidade. Um sistema resiliente responde mesmo na presença de falhas.
- Elasticidade, refere-se à capacidade de um sistema reagir adequadamente a variações de carga. Portanto, aumentar ou diminuir os recursos dependerá do número de solicitações ao sistema.

Em um sistema reativo, os componentes enviam e recebem mensagens assíncronas (consulte a Figura 3.6). Para dissociar remetentes e receptores, as mensagens são enviadas para um endereço virtual. Portanto, para que um componente receba uma mensagem, é necessário que ele esteja registrado nesse endereço virtual. Um endereço é um identificador do destino das mensagens e é representado por uma sequência ou URL. As in-

```
1  package io.vertx.book.rx;
2  import rx.Observable;
3  public class ObservableExample {
4
5  public static void main(String[] args) {
6  Observable<Integer> observable = Observable.range(0, 21);
7
8  observable.subscribe(
9      data -> {
10         System.out.println(data);
11     },
12     error -> {
13       error.printStackTrace();
14     },
15       () -> {
16             System.out.println("No more data");
17             }
18         );
19     }
20 }
```

Listing 1: Exemplo de um código contendo programação Reativa.

terações das mensagens assíncronas promovem duas propriedades: *Elasticidade*, que se refere à capacidade de escalar horizontalmente e *Resiliência*, que se refere à capacidade de permanecer responsivo em caso de falha e restauração. Além disso, melhora o tempo de resposta de um sistema, como pode ser observado nas Figuras 3.4 e 3.5.

3.3.3. Microserviços

Os Microserviços visam construir, gerenciar e projetar arquiteturas de pequenas unidades autônomas e são baseados na filosofia UNIX [Fowler and Lewis 2014]: os programas devem executar apenas uma tarefa e executar bem; os programas devem poder trabalhar juntos; os programas devem usar uma interface universal. Essas ideias levam a um design de componente reutilizável, suportando a modularização. O ponto principal é que os serviços são implantados no ambiente de produção independentemente um do outro, o que é uma das principais diferenças com a maioria das soluções tradicionais (e.g SOA) existentes.

A Figura 3.7 compara a arquitetura monolítica com a arquitetura de Microserviços. Na arquitetura monolítica, a lógica de negócios é processada em um único processo. Por esse motivo, uma atualização em qualquer parte da aplicação requer que todos as aplicações monolíticas sejam reimplementados. Em uma arquitetura de Microserviços, cada lógica de negócios é criada como um serviço independente. Assim, cada Microsserviço pode ser facilmente gerenciado em tempo de execução, incluindo várias atividades como implantação, instanciação e replicação.

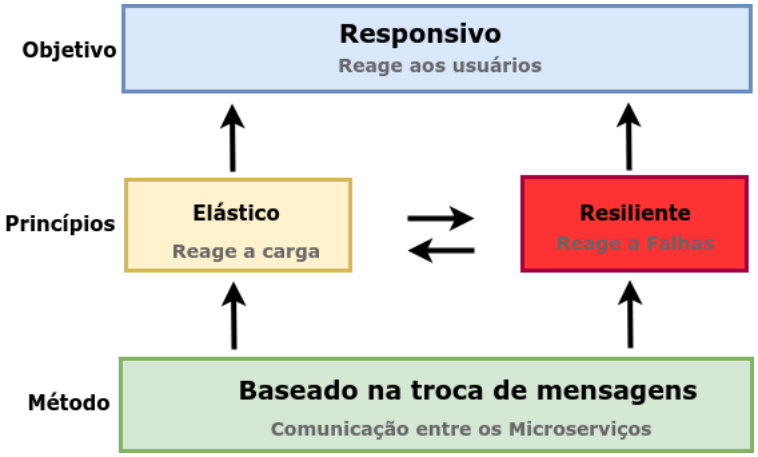
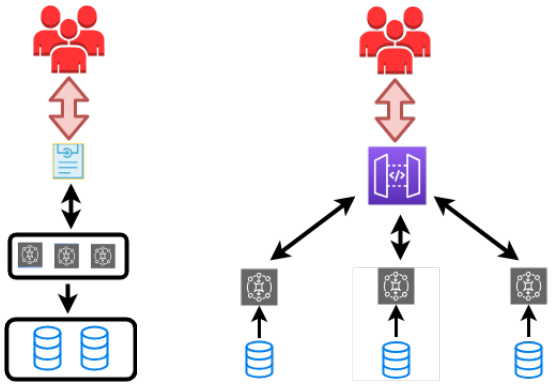


Figure 3.6. Princípios arquiteturais de um sistema reativo ([Bonér et al. 2014]).



(a) Monolito (b) Microserviços

Figure 3.7. Arquitetura Monolítica versus Arquitetura de Microserviços [Fowler and Lewis 2014].

Newman [Newman 2015] lista alguns benefícios, discutidos abaixo, ao adotar os Microserviços como uma solução no desenvolvimento de aplicações.

- **Technologia Heterogênea:** cada parte da aplicação pode ser implementada com diferentes tecnologias. Portanto, se uma parte da aplicação precisar melhorar sua qualidade de serviço, é possível decidir usar uma pilha de tecnologia diferente que seja mais adequada para atingir os níveis de QoS necessários. Por exemplo, na Figura 3.7 (lado b), cada aplicativo pode ser construído com diferentes tecnologias para atender a cada objetivo.

- **Dimensionamento:** com os Microserviços, é possível dimensionar partes da aplicação de acordo com a necessidade. Assim, é possível executar outras partes da aplicação em um dispositivo com menos poder computacional.

- **Facilidade de Implementação:** com os Microserviços, é possível alterar um único serviço e implantá-lo independentemente do restante do sistema. Se ocorrer um

problema, ele pode ser isolado rapidamente para um serviço individual, facilitando a recuperação rápida.

- **Alinhamento Organizacional :** com os Microserviços, é possível alinhar melhor a arquitetura da aplicação com a estrutura organizacional.

- **Composabilidade:** Um dos principais problemas nas arquiteturas orientadas a serviços e nos sistemas distribuídos é a possibilidade de melhorar a reutilização das aplicações. Com os Microserviços, é possível uma funcionalidade a ser consumida de diferentes maneiras para diferentes fins.

3.4. Arquitetura de Microserviços reativos para aplicações IoT

De acordo com o que foi discutido na Introdução, o crescente número de dispositivos na IoT permitirá o desenvolvimento de aplicações em vários domínios, como saúde, automação industrial e transporte. Essas aplicações de IoT possuirão os requisitos de QoS que devem ser concedidos em diferentes camadas da arquitetura IoT, nas quais a confiabilidade tem sido uma preocupação prioritária de sistemas IoT [White et al. 2017].

Neste contexto, a proposta de uma arquitetura de Microserviços confiáveis para o desenvolvimento de aplicações de IoT tem o objetivo de contribuir para melhorar a disponibilidade de aplicações quando comparado com soluções tradicionais em um ecossistema de IoT. A disponibilidade também pode ser definida como a capacidade do sistema de executar suas funcionalidades sob determinadas condições em um instante de tempo [Birolini 2013].

A Arquitetura de Microserviços discutida neste Capítulo foi projetada para orientar e apoiar o desenvolvimento de aplicações IoT resilientes e elásticas (ou seja, Responsivas). Esses recursos exigem um projeto de arquitetura que priorize o uso de mensagens assíncronas para garantir baixo acoplamento, transparência da localização e isolamento de comunicação entre diferentes partes da aplicação.

3.4.1. Arquitetura

Em primeiro lugar, a arquitetura define que uma aplicação IoT consiste de um conjunto de Microserviços reativos que podem ser distribuídos em dispositivos e servidores localizados na borda da rede, na névoa ou na nuvem. Os Microservices Reativos melhoram a autonomia, elasticidade e resiliência das aplicações. Em segundo lugar, nós fornecemos um broker para comunicação Máquina a Máquina. Em nossa proposta, esse broker foi projetado como um aplicação OSGI. Em terceiro lugar, adotamos containeres para aumentar o grau de elasticidade e reconfiguração das aplicações IoT. A Figura 3.8 ilustra a visão geral da proposta. Nessa arquitetura, a camada de aplicação possui duas partes principais: Microservices reativos, que guia o desenvolvimento das aplicações reativas e containers que está relacionada a infraestrutura necessária para a implementação e implantação das aplicações IoT. As características de cada componentes são discutidas a seguir.

Microserviços Reativos fornece uma arquitetura para o desenvolvimento de aplicações capazes de lidar com problemas de falha e perda de desempenho sem afetar o

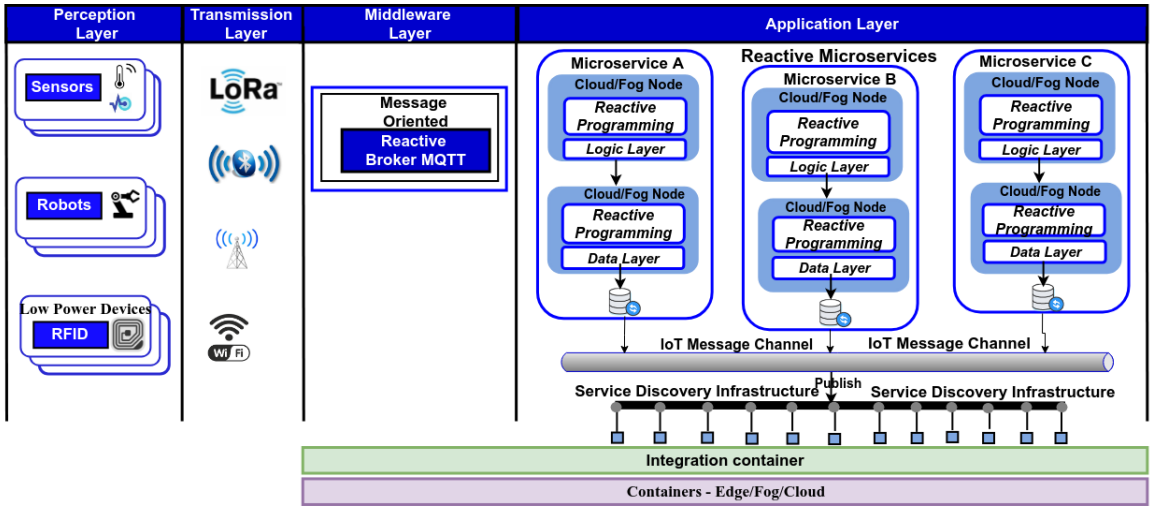


Figure 3.8. Visão Geral da Arquitetura.

comportamento de um aplicação inteira. Dividimos a camada lógica da arquitetura em duas partes: *Core* e *Utilities*. No *Core*, temos os seguintes componentes:

- **IoT Message Channel** fornece um meio de comunicação para que diferentes Microserviços possam se comunicar de maneira fracamente acoplada.
- **Service Discovery Infrastructure** fornece uma infraestrutura para a publicação e descoberta de serviços.
- **Reactive broker** fornece o ponto de comunicação com sensores e atuadores por meio de mensagens de publish/subscribe.
- **Circuit breaker** fornece um meio de evitar falhas em cascata.
- **Timeout and Bulkhead** permitir baixo acoplamento entre serviços.
- **Health check** fornece acesso ao status do Microsserviço: UP or Down.

A parte *Utilities* da arquitetura é usada em nossa proposta para fornecer suporte aos estudos de caso a serem implementados. Em Utilitários, temos os seguintes componentes:

- **Streamming** Fornece análise de dados em tempo real.
- **RESTful** Fornece acesso a dispositivos IoT.
- **Security** Fornece uma camada para autenticação e autorização para serviços.

3.4.2. Tecnologia

Para implementar cada componente dessa arquitetura (i.e. IoT Message Channel, Reactive broker, Circuit breaker,Timeout and Bulkhead, RESTful, Security) utilizamos a plataforma ilustrada na Figura 3.10. Esta plataforma é baseada no Vert.x e no ServiceMix.

3.4.2.1. Vert.x

O Vert.x[4] é um toolkit para criar sistemas reativos distribuídos no topo da Java Virtual Machine e adota um modelo de desenvolvimento assíncrono não bloqueiante. Como um toolkit, o Vert.x pode ser usado em muitos contextos: em uma aplicação standalone ou incorporado em um aplicação Spring[5]. O Vert.x e seu ecossistema são apenas arquivos jar usados como qualquer outra biblioteca: basta colocá-los em seu classpath e estará pronto para utilização. O Vert.x não fornece uma solução "all-in-one", mas fornece os blocos de construção para que desenvolvedores possa criar sua própria solução.

Uma parte do ecossistema Vert.x é mostrado na Figura 3.9. É possível selecionar qualquer um desses componentes (Figura 3.9) além do núcleo Vert.x para criar seus sistemas distribuídos. Por exemplo, para manipular conexões com clientes MQTT ou até mesmo criar um broker MQTT o componente responsável é o Vert.x MQTTT.

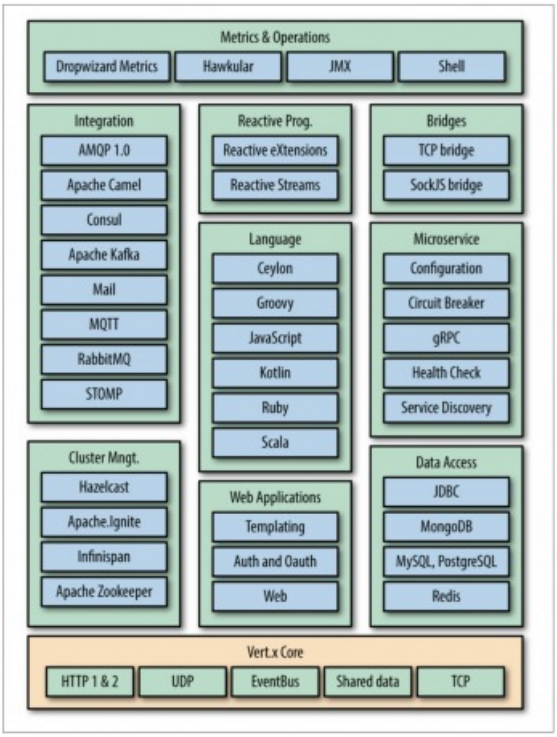


Figure 3.9. Uma parte do ecossistema Vertx. Adaptado [Bonér et al. 2014]

Uma das vantagens do Vert.x em comparação com outras tecnologias (por exemplo, Akka[6], Ratpack[7]) é o fornecimento de um conjunto de componentes capazes de projetar aplicações reativas em diferentes linguagens. Além disso, o Vert.x tem melhor desempenho do que o Akka e outros frameworks em alguns benchmarks, como em TechEmpower[8].

[4]https://vertx.io/
[5]https://spring.io/
[6]https://akka.io/
[7]https://ratpack.io/
[8]https://www.techempower.com/benchmarks/

```
1
2  //Exemplo de inclusao de dependencia no Maven
3
4  <dependency>
5     <groupId>io.vertx</groupId>
6     <artifactId>vertx-mqtt</artifactId>
7     <version>3.8.1</version>
8  </dependency>
9
10 //Exemplo de inclusao de dependencia no Gradle
11
12 compile io.vertx:vertx-mqtt:3.8.1
```

Código 3.1. Exemplos de inclusao de uma dependencia Vertx no Maven e Gradle

É possível utilizar o Vert.x de diferentes formas. Com o Maven ou Gradle é necessário apenas adicionar ao gerenciador de dependências o artefato a ser incluído no projeto. Por exemplo, na Listagem 3.1 é incluído a dependência para utilização do Vert.x MQTT Server e Vert.x MQTT Client.

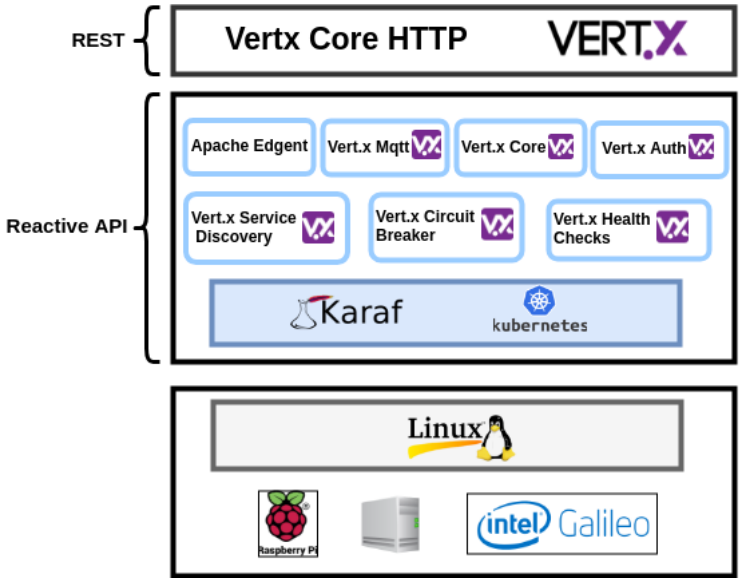


Figure 3.10. Reactive Microservices IoT Platform.

3.4.2.2. ServiceMix

O Apache ServiceMix é um aplicação de código aberto, implementada em Java, no qual serviços nativos da arquitetura ESB/OSGi oferecem toda a infraestrutura necessária para o suporte dos outros serviços da plataforma SOFT-IoT. Com o ServiceMix é possível implantar serviços (também chamados de bundles) em tempo de execução e permite que estes compartilhem dados e objetos através de relacionamentos e dependências.Figure 6.20. Componentes do Apache ServiceMixA Figura 6.20 representa o ServiceMix e os

seus principais módulos. O ServiceMix é leve e portável, tendo como requisito básico o suporte ao Java 1.7. Além disso,possui suporte ao Spring Framework e Blueprint e é compatível com o Java SE ou comum servidor de aplicativos Java EE. O Karaf é o núcleo principal do ServiceMix e ofereceo conceito de recursos, onde coleções de pacotes podem ser instalados como um grupoem um ambiente OSGi em execução. Sobe o Karaf, o ServiceMix usa o ActiveMQ parafornecer serviço de mensagens, CXF para suporte serviços RESTful, Cellar para comuni-cação e interações entre diferentes instalações do ServiceMix e Camel para integração etroca de dados através de rotas. Por fim, o ServiceMix contém módulos adicionais como H2 Database, que gerencia o sistema de banco de dados relacional baseado em arquivos.A plataforma SOFT-IoT foi baseada na versão 6.10 do ServiceMix. Para seu usoe instalação em um ambiente Linux é necessário somente baixar o pacote do programa, o qual pode ser encontrado em: `https://servicemix.apache.org/downloads/servicemix-6.1.0.html11http://servicemix.apache.org/`

Para inicializar o ServiceMix em sistemas Linux é preciso executar dentro do diretório do base o seguinte arquivo:

```
$ ./bin/servicemix
```

O ServiceMix pode ser gerenciado através de uma aplicação Web chamado webconsole. Nessa aplicação é possível gerenciar por meio de uma interface gráfica, funções como instalação, atualização, remoção e informações de bundles e também funções de debug. Para instalar o webconsole no ServiceMix basta executar o seguinte comando no terminal:

```
karaf@root()> feature:install webconsole
```

Por padrão aplicação Web inicializada pelo webconsole é configurada na porta 8181 com login e senha `karaf` e pode ser acessada pelo endereço:

`http://localhost:8181/system/console`

A Figura 3.11 mostra um recorte do webconsole. Observe que é possível através dele ter um panorama geral dos bundles em execução e também alterar seus status. O webconsole, além disso, permite instalar novos bundles e depurar o estado e eventuais erros das aplicações em execução.

3.4.3. Simulação para Internet das Coisas

Nas seções anteriores, entre outras coisas, foram abordadas as principais características sobre Internet das Coisas, Computação em Névoa e Microserviços. Com base nessas informações, apresenta-se nesta seção o processo de modelagem de ambientes que reproduzam cenários de Internet das Coisas em um ambiente de emulação/simulação. Para tal, utilizam-se ambiente para emulação de redes, tecnologias, protocolos e formatos de dados amplamente utilizados no contexto de Internet das Coisas. O objetivo é modelar cenários, realistas o suficiente, para que possibilitem a sua utilização como forma de testar e/ou validar novas soluções no âmbito de Internet das Coisas e/ou Névoa das Coisas. Na modelagem são implementados os componentes sensores, Gateway IoT e Gateway Virtualizado, todos, em conjunto com tecnologias e padrões (MQTT, ServiceMix, JSON, Plataforma Amazon EC2) já utilizados em equipamentos reais.

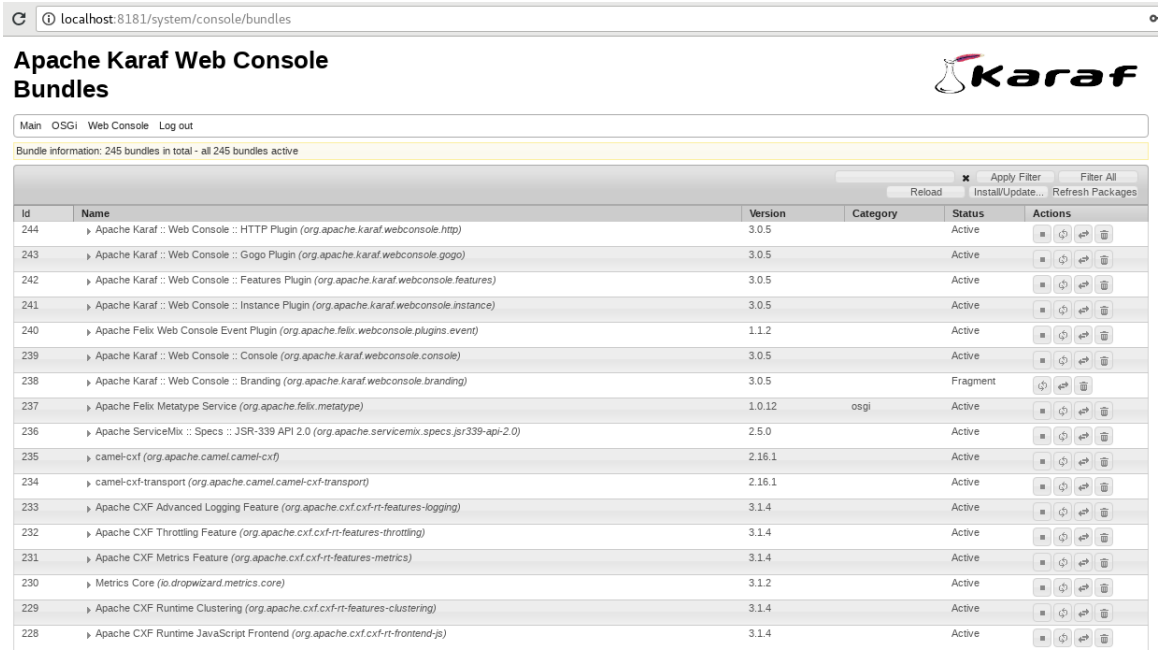


Figure 3.11. Webconsole do Apache ServiceMix (Karaf)

3.4.3.1. Emulador de Redes Mininet

O Mininet [9] é um emulador de de redes que possibilita a criação de *hosts* virtuais, *switches* e links. O emulador Mininet é um projeto [10] de código aberto e disponibilizado de forma gratuita. Essas características, associadas à flexibilidade na criação de redes, posicionam o Mininet como facilitador no desenvolvimento, prototipagem, aprendizagem, teste e depuração de novas soluções em IoT. Algumas vantagens de utilizar o Mininet: (i) possibilita o teste de infraestruturas de redes de forma simples e de baixo custo; (ii) permite testes complexos de topologia sem a necessidade de equipamentos de redes físicos; (iii) fornece API na linguagem de programação Python para criação e experimentação de infraestruturas de redes.

O Mininet utiliza a virtualização baseada em processos para a criação e execução dos elementos que constituem a rede. Com isso, todos os dispositivos de rede são dotados de recursos individuais mesmo compartilhando o mesmo *kernel* linux. O uso do mecanismo de virtualização "linux *network namespaces*" possibilita ao Mininet prover interfaces de redes e tabelas de roteamento/encaminhamento individuais, características fundamentais para a modelagem e implementação dos dispositivos de rede. As interfaces de rede no Mininet são conectadas através de links ethernets virtuais para as diversas instâncias possíveis de *switches* (ver Figura 3.12), dentre elas, o *switch* padrão do Mininet e o *Open vSwitch* [11] (*switch* de software multicamada adequado para para funcionar como virtual *switch* em ambientes de máquinas virtuais).

[9] Mininet: *An Instant Virtual Network* , disponível em: http://mininet.org/

[10] *Emulator for rapid prototyping of Software Defined Networks*, disponível em: https://github.com/mininet/mininet

[11] *What Is Open vSwitch?*, disponível em: http://docs.openvswitch.org/en/latest/intro/what-is-ovs/

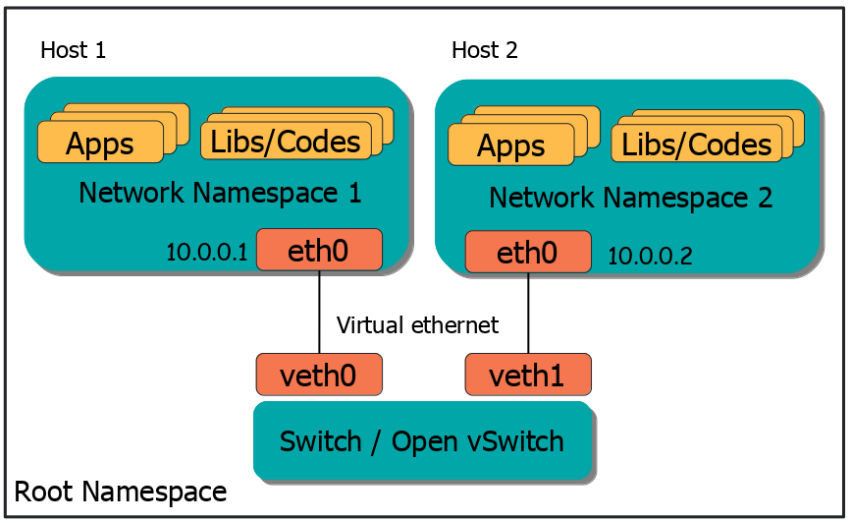


Figure 3.12. Mininet (*network namespaces*)

Com essas características, o Mininet fornece uma maneira simples para modelagem realista de ambientes que envolvem infraestrutura de redes. Os dispositivos que compõem a rede no Mininet executam código real, incluindo aplicativos e bibliotecas feitas para Linux, bem como a pilha do kernel e rede. Sendo assim, o código desenvolvido para ser executado em instâncias no Mininet (Controlador ou *hosts*) pode ser transferido para um dispositivo real com mínimas possibilidades de mudanças. Tal comportamento denota o Mininet como um emulador apropriado para implementação de um ambiente baseado no paradigma de Névoa das Coisas.

3.4.3.2. Modelagem da Internet das Coisas

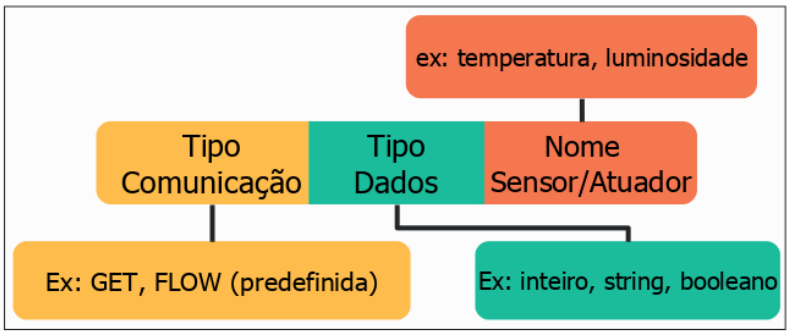
Baseado nos conceitos de Internet da Coisas, Computação em Nuvem e Névoa, descritos anteriormente, considera-se como componentes na modelagem: (i) Sensores; (ii) Gateways IoT; (iii) Nuvem / Gateways IoT virtualizados. Ressalta-se que a modelagem apresentada nesta Seção engloba os elementos básicos da Névoa das Coisas sob os aspectos de comportamento, comunicação e tecnologias utilizadas, de maneira que em um ambiente real de Névoa das Coisas (dispositivos físicos/reais) essas mesmas características tenham alto grau de similaridade com a modelagem proposta nesta Seção.

3.4.3.3. Protocolo de Comunicação TATU (*The Acessible Thing Universe Protocol*)

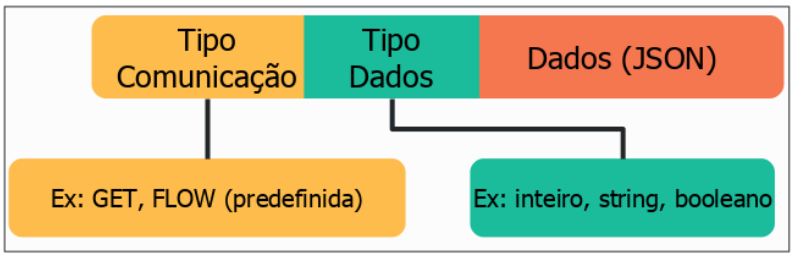
Embora o protocolo de comunicação MQTT seja leve e simples para ser implementado/executado em dispositivos IoT de capacidade limitada, existe a necessidade, dado a variedade das tarefas IoT e quantidade de informações geradas, de estabelecer padrões para a troca de mensagens em ambientes IoT. Nesse contexto, o protocolo de comunicação TATU [12], desenvolvido no contexto de IoT, Computação em Nuvem e Névoa,

[12]TATU: *The Acessible Thing Universe*, disponível em: https://github.com/WiserUFBA/TATUDevice

padroniza a troca de informações entre dispositivos IoT através de notações seguindo o formato JSON. O protocolo TATU pode ser utilizado de acordo com o tipo de comunicação/fluxo adotado nos dispositivos IoT, e atualmente estão implementados os modelos: (i) baseados em requisições GET/SET; (ii) baseados em requisições predefinidas (*flow* no TATU) [Prazeres and Serrano 2016].



(a) Requisição do tipo "pedido" no protocolo TATU



(b) Requisição do tipo "resposta" no protocolo TATU

Figure 3.13. Requisições no protocolo TATU.

A Figura 3.13 mostra os componentes de uma requisição no protocolo TATU, separadas em "pedido" e "resposta". Para realizar um "pedido", com o propósito de receber amostras com informações sobre uma grandeza (sensores) ou interagir com o ambiente (atuadores), deve-se especificar três campos (ver Figura 3.13(a)): (i) tipo de comunicação IoT; (ii) tipo de dados esperado na resposta; (iii) nome do sensor/atuador que deve atender/responder o pedido. Na requisição TATU do tipo "resposta", mostrada na Figura 3.13(b), adiciona-se um novo campo, chamado de dados ou carga útil. Esse campo contém informações no formato JSON sobre o sensor/atuador e a informação/resposta requerida pelo *FoT-Gateway* ou aplicação. O Código 3.2 mostra exemplos de mensagens construídas de acordo com o protocolo TATU. São mostradas mensagens da comunicação GET e FLOW (predifinidas) para as opções de pedido (requisição) e resposta.

Uma vez que o MQTT funciona fundamentalmente sob a abordagem de publicação/inscrição, as requisições TATU não são enviadas diretamente ao destino. Dessa forma, é necessário a organização do ambiente IoT para que os dispositivos realizem a inscrição nos endereços e tópicos corretos. A Figura 3.14 exibe o esquema necessário para realizar uma comunicação entre sensor e Gateway IoT, processo que pode ser resumido em quatro etapas (ver Figura 3.14): (1) sensor (MQTT *Client*) se inscreve no seu tópico no Gateway (MQTT *Broker*) que o gerencia; (2) Gateway IoT publica localmente no tópico do sensor desejado uma requisição TATU do tipo pedido; (3) o sensor recebe a

```
1
2  //Requisicoes baseadas em GET
3  GET INFO sensorType
4  //Resposta para Requisicoes GET
5  GET INFO RESPONSE {"code":"post","HEADER":{"requisition":"GET",
6  "name":"name_device"},"BODY":{"sensorType":"off"}
7
8  //Requisicoes predefinidas
9  FLOW INT sensorType {"collect":500, "publish":1000}
10 //Resposta p/ requisicoes predefinidas (cada 1000 milissegundos)
11 FLOW INT RESPONSE {"code":"post","HEADER":{"requisition":"FLOW",
12 "name":"device_name"},"BODY":{"sensorType":[30,31]}
```

Código 3.2. Exemplos do protocolo TATU

requisição TATU publicada pelo Gateway IoT na etapa anterior; (4) o sensor processa a informação recebida e publica a resposta no Gateway IoT.

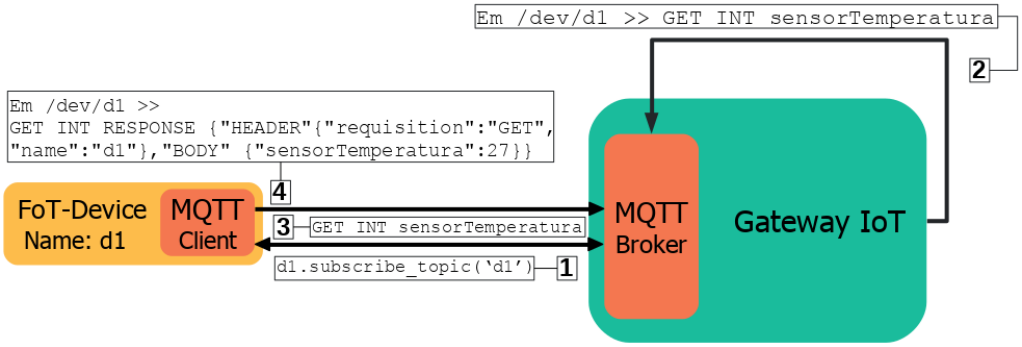


Figure 3.14. Exemplo de comunicação entre *FoT-Gateway* e *FoT-Device*

3.4.3.4. Modelagem do Sensor (virtual)

O sensor é o componente básico de um ambiente baseado no paradigma FoT. Para modelagem e, por conseguinte, simulação de um sensor virtual, alguns aspectos básicos precisam ser atendidos: (i) simulação de coleta de amostras (pseudoamostras) ou utilização de *dataset* de dados obtidos a partir de dispositivos reais e armazenamento de dados de curto período; (ii) implantação do protocolo de comunicação responsável por controlar o fluxo de informações entre sensores / Gateways / Nuvem.

O comportamento de um sensor é realizado por um serviço criado na linguagem de programação Python (ver Figura 3.15). É de sua responsabilidade a criação e armazenamento temporário de pseudoamostras. Para a identificação e construção de mensagens padronizadas de acordo com o protocolo TATU, realiza-se a importação e utilização da biblioteca que implementa as funções do protocolo TATU. A fim de realizar a comunicação, i.e. transporte de mensagens entre sensor e Gateway / Nuvem, utiliza-se o protocolo MQTT. Portanto, esse serviço executado em hosts do Mininet associado as bibliote-

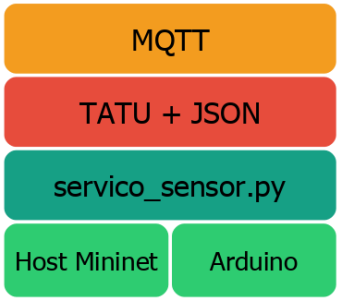


Figure 3.15. Modelo *FoT-Device*

cas e tecnologias utilizadas no processo de execução, incorporam ao host comportamentos de um sensor.

3.4.3.5. Modelagem do Gateway IoT

O Gateway IoT é o nó básico de gerência na Internet das Coisas. Gateways implementam suas funcionalidades através de serviços IoT (armazenamento, segurança, enriquecimento semântico, dentre outros) implementados seguindo especificações OSGi. Para a modelagem de um Gateway no Mininet alguns requisitos básicos devem ser atendidos: (i) implantação de framework que implementa as especificações OSGi (Apache ServiceMix); (ii) implantação dos serviços IoT no framework OSGi; (iii) implantação do protocolo de comunicação responsável por controlar o fluxo de informações entre sensores / Gateways / Nuvem.

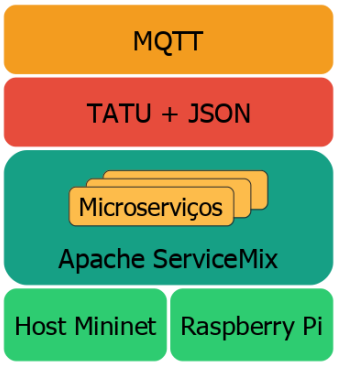


Figure 3.16. Modelo *FoT-Gateway*

A união dos componentes mostrados na Figura 3.16 caracterizam um host Mininet com o comportamento básico de um Gateway no cenário de Internet das Coisas. A gerência dos dispositivos é realizada pelos serviços IoT inclusos no Apache ServiceMix. As informações geradas pelos sensores são repassadas para os respectivos Gateways através do protocolo MQTT, já com o padrão definido pelo TATU. A partir disso, os Gateways poderão armazenar/tratar as informações recebidas.

3.4.3.6. Modelagem da Nuvem (Gateways IoT Virtualizados)

Os *FoT-Gateways* virtualizados servem como nó de gerência e realização de tarefas que exigem maior poder computacional. Podem ser disponibilizados em nível local (Névoa), através de servidores, ou em instâncias na Nuvem, conforme o proposto nesta Seção. Ao utilizar Gateways IoT disponibilizados na Nuvem, abre-se a possibilidade para a criação e desenvolvimento de estratégias que utilizem cenários híbridos (Nuvem e Névoa).

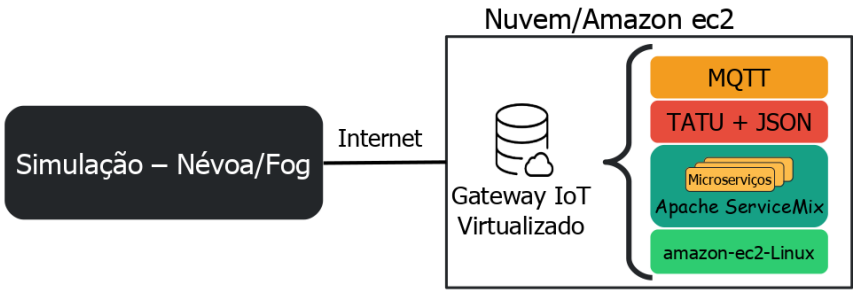


Figure 3.17. Modelo *FoT-Gateway* na Nuvem

A modelagem e implantação do Gateway IoT em um ambiente de Nuvem (ver Figura 3.17) é similar ao próprio Gateway implantado na Névoa (Mininet). No entanto, modifica-se os *hosts* do Mininet por instâncias de máquinas virtuais / Sistemas Operacionais disponibilizados na Nuvem. Para disponibilizar os Gateways em ambiente de Computação em Nuvem neste trabalho, utiliza-se a plataforma *Amazon Elastic Compute Cloud* [13] (*Amazon* EC2), que, entre outras coisas, possibilita a criação de instâncias virtuais de Sistemas Operacionais com total controle sobre instalação e gerenciamento dos seus programas.

3.5. Estudo de Caso

Para demonstrar de forma prática o desenvolvimento de aplicações IoT com Microserviços reativos um estudo de caso foi desenvolvido. Nesse estudo uma aplicação IoT foi desenvolvida para monitoramento das variáveis temperatura, umidade, som e luminosidade de um laboratório. Esse tipo de aplicação é definida na literatura como ambiente inteligente (do inglês *Smart Environment*). Um *Smart Environment* é uma sala ou espaço com sensores e/ou atuadores. Os sensores/atuadores e aplicações conectados em rede fazem a sala perceber o estado físico e as atividades internas do ambiente. Em um ambiente inteligente, as rotinas do usuário e os processos de informação podem interagir perfeitamente entre si.

Este estudo de caso foi desenvolvido tendo como base a arquitetura apresentada na Figura 3.18. Como sensores do ambiente foram utilizados o Sonoff SC apresentado na Seção 3.5.0.1. O equipamento do gateway IoT é o Raspberry Pi descrito na Seção 3.5.0.2. O estudo de caso utiliza uma arquitetura de Computação em Névoa combinada com a Computação em Nuvem. Para o ambiente de Névoa e Nuvem foram desenvolvidos Microserviços reativos e utilizadas tecnologias reativas com os seguinte objetivos:

[13] *Amazon Elastic Compute Cloud* , disponível em: https://aws.amazon.com/pt/ec2/

- **Microserviço Reativo de Análise de Dados**: Microserviço que realiza as análises de dados para tomadas de decisões no ambiente.
- **Microserviço Reativo de Coleta de Dados**: Microserviço responsável por receber e armazenar os dados produzidos pelos sensores.
- **Microserviço Reativo de Acesso a dados**: Microserviço que disponibiliza os dados coletados pelos sensores para acesso de aplicações por meio de serviços RESTFul.
- **Canal de Mensagens IoT**: Componente da arquitetura que permite a comunicação entre os Microserviços de Análise, Coleta e Acesso aos dados.
- **Broker Reativo**: Componente da arquitetura que utiliza a comunicação assíncrona pub/sub para troca de mensagens entre os Microserviços e os sensores do ambiente.

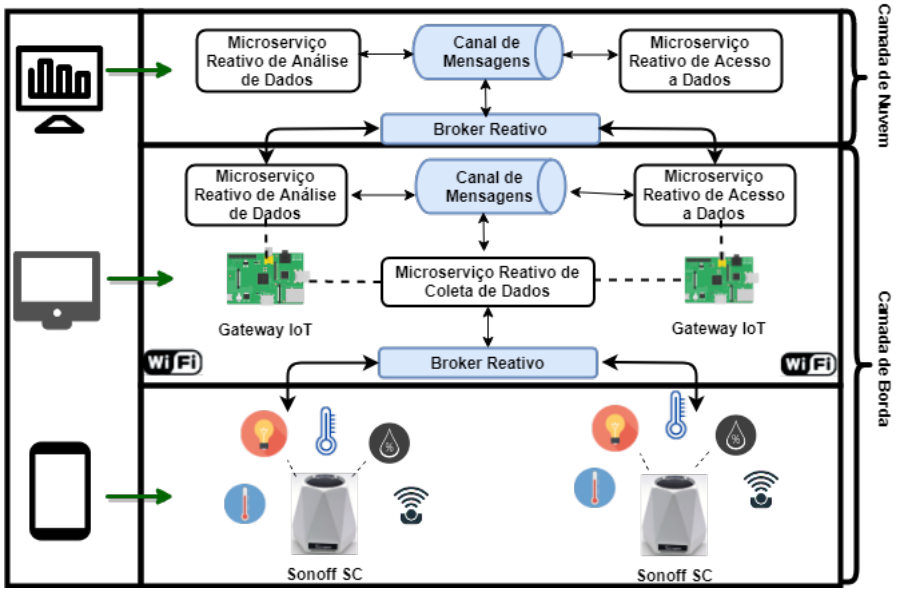


Figure 3.18. Arquitetura do Estudo de Caso.

3.5.0.1. Sonoff SC

O Sonoff SC é um nó de sensores desenvolvido pela ITEAD[14] que possui cinco sensores, são eles: temperatura, umidade, som, poeira e luminosidade. Os sensores que compõem o Sonoff SC são exibidos na Figura 3.19. A especificação dos sensores do Sonoff é a seguinte: o sensor de temperatura e umidade é o DHT11; o sensor de luminosidade é o LDR; o sensor de poeira é o Sharp GP2Y1010AU0F; para captação de som é utilizado um microfone.

O nó de sensores Sonoff SC utiliza o microcontrolador ATmega328. Esse microcontrolador do Sonoff foi desenvolvido pela Atmel Corporation e possui baixo custo.

[14]https://www.itead.cc/

Contudo, apresenta um baixo poder de processamento por causa do seu microprocessador de 8 bits. No Sonoff SC ilustrado na Figura 3.19, o ATmega328 é responsável pela conexão entre o ESP8266 e os sensores.

O ESP8266 é um módulo WiFi denominado como um SoC (System on a Chip). Esse módulo possui o protocolo TCP/IP integrado que pode dar acesso à rede WiFi para qualquer microcontrolador. O ESP8266 também possui um microprocessador de 32 bits com memória disponível. Com esse módulo WiFi é possível tanto hospedar uma aplicação quanto dar suporte às funções de rede WiFi de outro processador. Cada unidade possui uma programação com um firmware de conjunto de comandos AT. O módulo ESP8266 é uma placa com custo baixo e contínua expansão da comunidade.

Figure 3.19. Componentes Sonoff-SC.

3.5.0.2. Raspberry Pi

O Raspberry Pi, Figura 3.20, é um computador do tamanho de um cartão de crédito, desenvolvido no Reino Unido pela Fundação Raspberry Pi[15], foi criado para promover o ensino de ciência da computação nas escolas. O primeiro modelo tornou-se mais popular do que o esperado, visto que as vendas foram além do seu mercado-alvo, sendo adquiridos para usos distintos como na robótica.

Figure 3.20. Raspberry Pi Modelo A (Esquerda) e Modelo B (Direita).

[15]https://www.raspberrypi.org/

Os números mostram a aceitação do público ao dispositivo. De acordo com a Fundação Raspberry Pi, mais de 5 milhões de dispositivos foram vendidos antes de fevereiro de 2015, tornando-se o computador britânico mais comercializado. Em novembro de 2016, foram vendidos 11 milhões de unidades, atingindo 12,5 milhões em março de 2017, tornando-se o terceiro computador de propósito geral mais vendido.

O Raspberry Pi, como qualquer outro computador, usa um sistema operacional (SO). A opção Linux, chamada Raspbian, é grátis e de código aberto, reduzindo o preço da plataforma e tornando-o mais fácil de usar [Maksimović et al. 2014]. Existem outras opções de SO disponíveis [Richardson and Wallace 2012]. O fato do Raspberry Pi ser atrativo consiste em possuir uma ampla gama de aplicação.

O Raspberry Pi além de ser pequeno, possui uma série de vantagens como o baixo custo quando comparado aos demais dispositivos no mercado [Maksimović et al. 2014]. Uma redução no custo, em alguns casos, poderá resultar na capacidade de comprar mais dispositivos, tornando possível, por exemplo, implantar uma rede de maior densidade para coletar mais dados. Outro aspecto relevante está relacionado a energia. O componente principal do Raspberry Pi é a CPU que é responsável por realizar as instruções de um programa de computador através de operações matemáticas e lógicas. O processador ARMbased BCM2835, que é barato, poderoso, e não consome muita energia, é o motivo pelo qual o Raspberry Pi é capaz de operar apenas na fonte de alimentação 5V/1A fornecida pela porta micro-USB [Maksimović et al. 2014].

3.5.1. Implementação

- **Microserviço Reativo de Análise de Dados**:

  Este Microserviço habilita a análise de dados na borda da rede. O Microserviço de análise de dados pode ser verificada em:

```
https://github.com/WiserUFBA/reactsmartlab/blob/master/
   NetBeansProjects/
iot-reactive-application/src/main/java/
controller/AnaliseDadosController.java
```

- **Microserviço Reativo de Coleta de Dados**:

  Estes Microserviços capturam dados dos dispositivos conectados. Os Microserviços responsáveis pela coleta de dados podem ser verificados em:

```
https://github.com/WiserUFBA/reactsmartlab/blob/
master/NetBeansProjects/
iot-reactive-application/src/main/java/model/Sensor.java
```

```
https://github.com/WiserUFBA/reactsmartlab/
blob/master/NetBeansProjects/
iot-reactive-application/src/main/java/
controller/ReactiveController.java}
```

- **Microserviço Reativo de Acesso a dados**: Este Microserviço é responsável em expor os dados dos sensores. O Microserviço de exposição dos dados pode ser verificado em:

```
https://github.com/WiserUFBA/reactsmartlab/blob/master/
NetBeansProjects/
iot-reactive-application/src/main/java/
controller/AcessoDadosController.java
```

- **Canal de Mensagens IoT**:

  O componente Canal de Mensagens IoT permite que diferentes partes de um aplicação ou serviço se comuniquem de maneira pouco acoplada. As mensagens são enviadas aos endereços e os consumidores se registram nesses endereços para receber as mensagens. O canal de mensagens da IoT também é clusterizado, o que significa que ele pode enviar mensagens pela rede entre remetentes e consumidores distribuídos. Além disso, o Canal de Mensagens IoT envia mensagens para as instâncias disponíveis e, assim, equilibra a carga entre os diferentes nós registrados no mesmo endereço.

  A utilização do canal de mensagens IoT pode ser verificada em:

```
https://github.com/WiserUFBA/reactsmartlab/
blob/master/NetBeansProjects/
iot-reactive-application/src/main/java/
controller/ReactiveController.java}
```

  Um exemplo de utilização dessa canal de mensagens é ilustrado pelo código abaixo:

```
vertx.eventBus().send("webmedia",  hndlr.topicName() +
hndlr.payload().toString() );
```

- **Broker Reativo**:

  O Microserviço MQTT Broker fornece um servidor que é capaz de lidar com conexões, comunicação e troca de mensagens com clientes MQTT remotos. Na plataforma SOFT-IoT o broker MQTT é implemetado no módulo **soft-iot-vertx-mqtt-broker**[16] uma implementação baseada em OSGI como um *bundle* do ServiceMix.

  Este módulo é responsável por habilitar comunicações com os dispositivos conectados. Além das vantagens de usar uma arquitetura modular, o uso da API Vert.x MQTT Server torna possível escalar o agente MQTT reativo de acordo com, por exemplo, o número de núcleos do sistema e, assim, possibilitar a escalabilidade horizontal.

  Antes instalar o Microserviço soft-iot-vertx-mqtt-broker é necessário introduzir ao ServiceMix alguns módulos do Vert.x. As dependências estão localizadas no endereço:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/
tree/master/src/main/resources/dependencies
```

  As dependências são bibliotecas .jar que devem ser incluídas no ServiceMix, podendo ser introduzidas através do webconsole, conforme pode ser visto na Figura 3.11 através do botão `Install/Update`. Além da instalação das dependências é

[16]https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker

necessário também incluir arquivos referente ao certificado de segurança do soft-iot-vertx-mqtt-broker. Tais arquivos estão disponíveis em:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/tree/
    master/
src/main/resources/certificates
```

O certificado de segurança possui um arquivo de configuração disponível em:

```
https://github.com/WiserUFBA/soft-iot-vertx-mqtt-broker/
blob/master/src/main/resources/configuration/
br.ufba.dcc.wiser.soft_iot.gateway.brokers.cfg
```

Os arquivos do certificado de segurança, incluindo o arquivo de configuração devem ser armazenados em:

```
<diretorio_servicemix>/etc
```

Para instalação do soft-iot-vertx-mqtt-broker é necessário executar os seguintes comandos no terminal do ServiceMix:

```
karaf@root()> bundle:install mvn:br.ufba.dcc.wiser.soft_iot/
soft-iot-vertx-mqtt-broker/1.0.0
```

3.6. Considerações Finais

Este capítulo introduziu o conceito de Microserviços reativos, cujo objetivo é permitir a construção de aplicações na IoT que sejam performativos, resilientes e escalonáveis, o que é possível pela troca de mensagens assíncronas. Como prova de conceito, implementamos uma solução para análise de dados das temperaturas e humidade de uma laboratório de pós graduação na Universidade Federal da Bahia. Nesse experimento é possível mostrar as características de resiliência e elasticidade fornecidas nos Microserviços.

References

[Abdelshkour 2015] Abdelshkour, M. (2015). IoT, from cloud to fog computing. `http://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing`. [Online; acessado 04 de Setembro de 2019].

[Al-Fuqaha et al. 2015] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.

[Andrade et al. 2018] Andrade, L., Lira, C., Mello, B., Andrade, A., Coutinho, A., Greve, F., and Prazeres, C. (2018). Soft-iot platform in fog of things. In *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*, WebMedia '18, pages 23–27, New York, NY, USA. ACM.

[Ashton 2009] Ashton, K. (2009). That 'internet of things' thing. *RFiD Journal*, 22:97–114.

[Bainomugisha et al. 2013] Bainomugisha, E., Carreton, A. L., Cutsem, T. v., Mostinckx, S., and Meuter, W. d. (2013). A survey on reactive programming. *ACM Comput. Surv.*, 45(4):52:1–52:34.

[Bassi et al. 2013] Bassi, A., Bauer, M., Fiedler, M., Kramp, T., Kranenburg, R. v., Lange, S., and Meissner, S., editors (2013). *Enabling Things to Talk: Designing IoT Solutions with the IoT Architectural Reference Model*. Springer, Heidelberg.

[Bauer et al. 2013] Bauer, M., Boussard, M., Bui, N., Carrez, F., Jardak (SIEMENS, C., De Loof (ALUBE, J., Magerkurth (SAP, C., Meissner, S., Nettsträter (FhG IML, A., Olivereau, A., Thoma (SAP, M., Joachim, W., Stefa (CSD/SUni, J., and Salinas, A. (2013). Internet of things – architecture iot-a deliverable d1.5 – final architectural reference model for the iot v3.0.

[Bernstein 2014] Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

[Bilal et al. 2018] Bilal, K., Khalid, O., Erbad, A., and Khan, S. U. (2018). Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, 130:94 – 120.

[Birolini 2013] Birolini, A. (2013). *Reliability engineering: theory and practice*. Springer Science & Business Media.

[BLOOM 2013] BLOOM, Z. (2013). How we deploy 300 times a day.

[Bonér 2017] Bonér, J. (2017). *Reactive Microsystems The Evolution of Microservices at Scale*. O'Reilly Media, Gravenstein Highway North, Sebastopol.

[Bonér et al. 2014] Bonér, J., Farley, D., Kuhn, R., and Thompson, M. (2014). The reactive manifesto.

[Bonomi et al. 2012] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA. ACM.

[Borgia 2014] Borgia, E. (2014). The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1 – 31.

[Botta et al. 2016] Botta, A., de Donato, W., Persico, V., and Pescapé, A. (2016). Integration of cloud computing and internet of things: A survey. *Future Generation Computer Systems*, 56:684 – 700.

[Cavalcante et al. 2016] Cavalcante, E., Pereira, J., Alves, M. P., Maia, P., Moura, R., Batista, T., Delicato, F. C., and Pires, P. F. (2016). On the interplay of internet of things and cloud computing: A systematic mapping study. *Computer Communications*, 89-90:17 – 33. Internet of Things : Research challenges and Solutions.

[CI 2015] CI, C. (2015). Db performance issue incident report for circleci.

[da Rosa Righi et al. 2018] da Rosa Righi, R., Correa, E., Gomes, M. M., and da Costa, C. A. (2018). Enhancing performance of iot applications with load prediction and cloud elasticity. *Future Generation Computer Systems*, 1(1):1–13.

[Díaz et al. 2016] Díaz, M., Martín, C., and Rubio, B. (2016). State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *Journal of Network and Computer Applications*, 67:99 – 117.

[de Santana et al. 2018] de Santana, C. J. L., de Mello Alencar, B., and Prazeres, C. V. S. (2018). Microservices: A mapping study for internet of things solutions. In *2018 IEEE International Symposium on Network Computing and Applications (NCA)*, pages 1–4, Cambridge, MA, USA. IEEE.

[de Santana et al. 2019] de Santana, C. J. L., de Mello Alencar, B., and Prazeres, C. V. S. (2019). Reactive microservices for the internet of things: A case study in fog computing. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pages 1243–1251, New York, NY, USA. ACM.

[Distefano et al. 2012] Distefano, S., Merlino, G., and Puliafito, A. (2012). Enabling the cloud of things. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 858–863.

[Dragoni et al. 2017] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham.

[for Standardization/International Electrotechnical Commission et al. 2011] for Standardization/International Electrotechnical Commission, I. O. et al. (2011). Iso/iec 25010-systems and software engineering—systems and software quality requirements and evaluation (square)—system and software quality models. *Authors, Switzerland*, 1(15):1–15.

[Fowler and Lewis 2014] Fowler, M. and Lewis, J. (2014). Microservices, 2014. *URL: http://martinfowler. com/articles/microservices. html*, 1(1):1–1.

[Francesco et al. 2017] Francesco, P. D., Malavolta, I., and Lago, P. (2017). Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 21–30, Gothenburg, Sweden. IEEE.

[Gérald 2010] Gérald, S. (2010). *The Internet of Things: Between the Revolution of the Internet and the Metamorphosis of Objects*. Publications Office of the European Union.

[Humble and Farley 2011] Humble, J. and Farley, D. (2011). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Addison-Wesley Boston, EUA.

[Khan et al. 2012] Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet: The internet of things architecture, possible applications and key challenges. In *2012 10th International Conference on Frontiers of Information Technology*, pages 257–260.

[kubernetes 2019] kubernetes (2019). What is kubernetes. *https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/*.

[Lasse Lueth 2018] Lasse Lueth, K. (2018). State of the iot 2018: Number of iot devices now at 7b – market accelerating. *Press Release. Online unter: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/*.

[Maksimović et al. 2014] Maksimović, M., Vujović, V., Davidović, N., Milošević, V., and Perišić, B. (2014). Raspberry pi as internet of things hardware: performances and constraints. *design issues*, 3:8.

[Namiot and Sneps-Sneppe 2014] Namiot, D. and Sneps-Sneppe, M. (2014). On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27.

[Newman 2015] Newman, S. (2015). *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.".

[Prazeres and Serrano 2016] Prazeres, C. and Serrano, M. (2016). Soft-iot: Self-organizing fog of things. In *2016 30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 803–808.

[Richardson and Wallace 2012] Richardson, M. and Wallace, S. (2012). *Getting started with raspberry PI*. " O'Reilly Media, Inc.".

[Serrano et al. 2015a] Serrano, M., Barnaghi, P., Carrez, F., Cousin, P., Vermesan, O., and Friess, P. (2015a). Iot semantic interoperability: Research challenges, best practices, recommendations and next steps. *IERC: European Research Cluster on the Internet of Things,*.

[Serrano et al. 2015b] Serrano, M., Quoc, H. N. M., Phuoc, D. L., Hauswirth, M., Soldatos, J., Kefalakis, N., Jayaraman, P. P., and Zaslavsky, A. (2015b). Defining the stack for service delivery models and interoperability in the internet of things: A practical case with openiot-vdk. *IEEE Journal on Selected Areas in Communications*, 33(4):676–689.

[Shahid and Aneja 2017] Shahid, N. and Aneja, S. (2017). Internet of things: Vision, application areas and research challenges. In *International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pages 583–587, Palladam, India. IEEE.

[Shi and Dustdar 2016] Shi, W. and Dustdar, S. (2016). The promise of edge computing. *Computer*, 49(5):78–81.

[Sundmaeker et al. 2010] Sundmaeker, H., Guillemin, P., Friess, P., and Woelfflé, S., editors (2010). *Vision and Challenges for Realising the Internet of Things*. Publications Office of the European Union, Luxembourg.

[Taveras Núñez 2017] Taveras Núñez, P. M. (2017). *A Reactive Microservice Architectural Model with Asynchronous Programming and Observable Streams as an Approach to Developing IoT Middleware*. PhD thesis, PUCMM. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Última atualização em - 2018-05-09.

[Trojak 2018] Trojak, M. (2018). Ci/cd at scale.

[Udoh and Kotonya 2018] Udoh, I. S. and Kotonya, G. (2018). Developing iot applications: challenges and frameworks. *IET Cyber-Physical Systems: Theory Applications*, 3(2):65–72.

[White et al. 2017] White, G., Nallur, V., and Clarke, S. (2017). Quality of service approaches in iot: A systematic mapping. *Journal of Systems and Software*, 132:186 – 203.

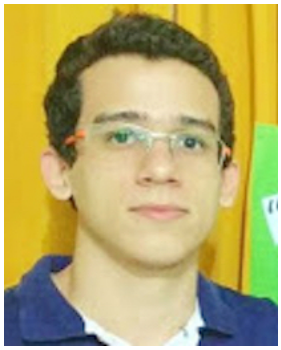
[Zhang et al. 2010] Zhang, S., Zhang, S., Chen, X., and Huo, X. (2010). Cloud computing research and development trend. In *2010 Second International Conference on Future Networks*, pages 93–97.

Biografia Resumida dos Autores

Cleber Santana, é doutorando em Ciência da Computação pela Universidade Federal da Bahia (UFBA). Obteve o título de Mestre em Sistemas e Computação (2014) e graduado em Processamento de Dados pela Faculdade Ruy Barbosa. É pesquisador do grupo WISER-UFBA/NUMAC-IFBA, atuando em projetos relacionados a Internet das Coisas, Microserviços, Web Semântica, Computação em Névoa e Educação. Cleber é membro da IEEE Communications Society, da Communications Society e possui publicações em conferências nacionais e internacionais. Desde 2013 é professor do Instituto Federal da Bahia e possui interesse em pesquisa nas áreas de Web Semântica, Serviços Web, Microserviços, Internet das Coisas, Computação em Névoa, Inteligência Artificial e Informática na Educação.

Leandro Andrade, é doutorando em Ciência da Computação pela Universidade Federal da Bahia (UFBA). Obteve o título de Mestre em Ciência da Computação (2014) e também o de Bacharel em Ciência da Computação (2012) pela UFBA. É pesquisador do grupo WISER-UFBA, atuando em projetos relacionados a Internet das Coisas, Computação em Névoa e Big Data. Realizou doutorado sanduíche no The Insight Centre for Data Analytics (NUI Galway - Irlanda). Leandro é membro da Sociedade Brasileira de Computação (SBC), da Communications Society e

possui publicações em conferências nacionais e internacionais. É professor substituto da UFBA, vinculado ao Departamento de Ciência da Computação. Possui interesse em pesquisa nas áreas de Internet das Coisas, Computação em Névoa, Serviços Web, Web Semântica, Big Data, Aprendizado de Máquina, Informática na Educação e Software Livre.



Brenno de Mello é mestrando em Ciência da Computação da Universidade Federal da Bahia (UFBA). Atualmente, é participante do grupo de pesquisa WISER, pesquisando principalmente os seguintes temas: Internet das Coisas, Mineração de Fluxo de Dados, Fog Computing, Smart Water. Mello possui graduação em Análise e Desenvolvimento de Sistemas pelo Instituto Federal da Bahia (2016). Tem experiência na área de Ciência da Computação, com ênfase em Sistemas de Informação, atuando como Analista de Sistemas.

José Sampaio, é graduando em Ciência da Computação pela Universidade Federal da Bahia (UFBA). Possui bolsa de iniciação científic apela FAPESB e faz parte do grupo WISER-UFBA, atuando em projetos relacionados a Internet das Coisas, Computação em Névoa e Microserviços Reativos. Possui interesse em pesquisa nas áreas de Internet das Coisas, Computação em Névoa, Serviços Web e Big Data.

Ernando Batista é Mestre em Ciência da Computação pela Universidade Federal da Bahia (UFBA), Engenheiro de Computação e Bacharel em Ciências Exatas pela Universidade Federal do Recôncavo da Bahia (UFRB). É pesquisador no Laboratório e Grupo de Pesquisa CNPq WISER (Web, Internet and Intelligent Systems Research Group) e Professor no Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA). Possui interesse em pesquisa nas áreas de Internet das Coisas, Computação em Névoa, Serviços Web e Redes Definidas por Software.

Cássio Prazeres é Doutor em Ciências - Área de Ciências de Computação e Matemática Computacional - pela Universidade de São Paulo (2009), é professor Associado I na Universidade Federal da Bahia (UFBA) nas áreas Internet/Web e é orientador permanente no Programa de Pós-graduação em Ciência da Computação (PGCOMP-UFBA). Prazeres é membro: da Sociedade Brasileira de Computação (SBC); do ACM SIGWEB (Special Interest Group on Hypertext the Web); do IEEE Computer Society Technical Committee on Services Computing; do IEEE Smart Cities Technical Community; do IEEE Internet of Things Technical Community; e do W3C Web of Things Community Group. É co-fundador e líder do Laboratório e Grupo de Pesquisa CNPq WISER (Web, Internet and Intelligent Systems Research Group). Tem interesse em pesquisas envolvendo tópicos de: Internet of Things, Web of Things, Web Services, Semantic Web, Microservices, Fog Computing, Fog of Things,

Web of Data. Em 2015, Prazeres realizou estágio pós-doutoral como professor visitante no DERI (Digital Enterprise Research Institute) na National University of Ireland (Galway) nas áreas de Internet das Coisas e Web Semântica.

Capítulo 4

Métodos baseados em Deep Learning para Análise de Vídeo

Gabriel N. P. dos Santos[1], Pedro V. A. de Freitas[1], Antonio José G. Busson[1], Álan L. Guedes[1], Sérgio Colcher[1] e Ruy L. Milidiú[1]

[1]Departamento de Informática, PUC-Rio

{gabrielpereira,pedropva,busson,alan}@telemidia.puc-rio.br

{colcher,milidiu}@inf.puc-rio.br

Abstract

Methods based on Deep Learning became state-of-the-art in several multimedia challenges. However, there is a gap of professionals to perform Deep Learning in the industry. This chapter focuses on presenting the fundamentals and technologies for developing such DL methods for video analyses. In particular, we seek to enable the reader to: (1) understand key DL-based models, more specifically Convolutional Neural Networks (CNN); (2) apply DL models to solve video tasks such as video classification, multi-label video classification, object detection, and pose estimation. The Python programming language is presented in conjunction with the TensorFlow library for implementing DL models.

Resumo

Os métodos baseados no Deep Learning tornaram-se state-of-the-art em vários desafios de multimídia. No entanto, existe uma lacuna de profissionais para realizar o Deep Learning na indústria. Este capítulo tem como foco apresentar os fundamentos e tecnologias para desenvolver tais métodos de DL para analise de vídeo. Em especial, buscamos capacitar o leitor a: (1) entender os principais modelos baseados em DL, mais especificamente Convolutional Neural Networks (CNN); (2) aplicar os modelos de DL para resolver tarefas de vídeo como: classificação de vídeo, classificação de multi-etiquetas de vídeo, detecção de objetos e estimação de pose. A linguagem de programação Python é apresentada em conjunto com a biblioteca TensorFlow para implementação dos modelos de DL.

4.1. Introdução

A popularização de equipamentos de captura de vídeo e serviços para seu armazenamento e transmissão, possibilitou a produção de um massivo volume de dados de vídeo. O Youtube, por exemplo, registrou em 2014[1] upload de 72 horas de vídeo por minuto. Enquanto que em 2018[2], esse número subiu para 400 horas de vídeo por minuto. Por exemplo no Brasil, serviços como o video@RNP[3] e ICD[4] constituem importantes redes de compartilhamento vídeo. Esse cenário apresenta desafio de controle do tipo de conteúdo que é carregado para esses serviços de armazenamento videos. A classificar esse tipo de conteúdo requer uma análise automática desse volume de forma eficiente e prática.

Métodos baseados em *Deep Learning* (DL) se tornaram o *estado-da-arte* em vários segmentos relacionados a análise automática de mídia. Em particular, as arquiteturas de DL baseadas em CNNs (*Convolutional neural network*), ou ConvNets, se tornaram o principal método usado para reconhecimento de padrões áudio-visuais. Tipicamente o treinamento de CNNs é feito de maneira supervisionada, e são treinadas em *datasets* que contém milhares/milhões de mídias e classes relacionadas. Durante o treinamento, as CNNs aprendem a hierarquia de *features* que são aplicadas a mídia de entrada para que seja possível realizar a classificação do seu conteúdo.

Este capítulo tem como foco avaliar e desenvolver tais métodos de DL para análise de vídeo. No decorrer dele, são implementados métodos de DL utilizando a linguagem Python. Para apresentar esse métodos, o restante deste trabalho está organizado como a segue.

- A Seção 4.2 apresenta o *framework TensorFlow*.
- A Seção 4.3 introduz os conceitos básicos de aprendizado de máquina.
- A Seção 4.4 apresenta os fundamentos de Redes Neurais
- A Seção 4.5 apresenta os fundamentos de Redes Neurais Convolucionais.
- A Seção 4.6 apresenta implementações para resolução de problemas de classificação de vídeo.
- A seção 4.7 descreve o modelo YOLO para detecção de objetos.
- A Seção 4.8 apresenta conceitos básicos de pose estimation, métricas e técnicas.
- A Seção 4.9 apresenta nossas considerações finais.

4.2. Framework TensorFlow

Nesta seção, apresentamos uma visão geral do *framework Tensorflow*. O *Tensorflow*[5] é um framework *open-source* para Python, Javascript, C/C++ e Go e tem o objetivo de

[1] https://www.domo.com/learn/data-never-sleeps-2
[2] https://www.domo.com/learn/data-never-sleeps-6
[3] http://video.rnp.br
[4] http://documentas.redclara.net/handle/10786/1104
[5] https://www.tensorflow.org/?hl=pt-br

auxiliar o processamento de dados em *machine learning* por meio de um modelo baseado em fluxo de dados. Ele foi criado em 2015 pelo time da Google responsável por pesquisas na área de Inteligência Artificial e *Deep Learning* (Google Brain). O *Tensorflow* começou como uma refatoração do antigo sistema DistBelief[6] (criado em 2011), com o intuito de melhorar seu desempenho.

Para apresentar o *Tensorflow*, o restante seção está organizado como segue. A Subseção 4.2.1 descreve o processo de instalação. Em seguida, a Subção 4.2.2 apresenta os fundamentos e estruturas básicas do *framework*. Por fim, a Subseção 4.2.3 descreve como utilizar o *framework*.

4.2.1. Instalação

Neste parte inicial, descrevermos como instalar o *Tensorflow* em sistemas Ubuntu e outras distribuições derivadas do Linux. O Python 2.7 e Pip (sistema de gerenciamento de pacotes do Python) já estão presentes no Ubuntu e na maioria das outras distribuições Linux. Porém, a versão python mais recente é o Python3. Para instalação do Python3, execute:

```
sudo apt-get install python3-pip python3-dev python-virtualenv
```

Tensorflow necessita que a versão do pip seja a 8.1 ou mais recente. Para verificar a versão atual do pip Python3 execute:

```
pip3 -V
```

Para atualizar o pip para a versão mais recente no Ubuntu:

```
sudo pip install -U pip
```

Para atualização do pip em distribuições diferentes da Ubuntu:

```
easy_install -U pip
```

A seguir, deve-se escolher a instalação com ou sem suporte para GPU. Para instalar o *Tensorflow* sem e com suporte para GPU execute respectivamente:

```
pip install -U tensorflow
```

ou

```
pip install -U tensorflow-gpu
```

Para testar a instalação inicie o terminal Python e execute:

```
import tensorflow as tf
tf.__version__
exit()
```

[6] https://ai.google/research/pubs/pub40565

4.2.2. Fundamentos e estruturas básicas do Tensorflow

O *Tensorflow* oferece facilidade para desenvolver modelos de redes neurais para uma multiplicidade de diferentes hardwares, bem como possibilita que o sistema seja executado sem ou com **GPU**s. Para utilizar o *framework*, primeiro é necessário entender os três conceitos que são descritos a seguir.

1. **Tensor**: consiste de um vetor de n dimensões. Tensores são as estruturas de dados básicas utilizadas no *TensorFlow*.

2. **Grafo de computação**: são malhas que consistem em nós conectados entre si por arestas. Cada nó possui seus *inputs* e outputs, assim como a operação que deve ser feita com os *inputs* para que o *output* seja criado. Tais arestas consistem nos valores que são passados de um nó para outro. Cada nó realiza a determinada operação assim que recebe todos os *inputs* necessários. Ao gerar seu resultado e passar para um próximo nó (ou vários) ligado a este.

3. **Sessões**: os nós do grafo de computação podem ser agrupados em sessões. Cada sessão pode ser executada separadamente em *threads* ou até mesmo em forma de computação distribuída.

4.2.3. Utilizando o Tensorflow

Após entender as estruturas básicas do *framework*, para usar o *Tensorflow*, como mostra a Listagem 4.1, basta importar o pacote para o projeto (linha 1). As linhas 3-5 mostram como criar o grafo de computação, para isso, são criados três tensores, onde o tensor **c**, consiste na multiplicação dos tensores **a** e **b**. Em seguida, as linhas 7 e 8 mostram como criar uma sessão e executar o grafo de computação. Por fim, a linha 10 mostra a saída do programa.

Listagem 4.1: Executando um grafo de computação no Tensorflow.

```
1  import tensorflow as tf
2
3  a = tf.constant([ [1.0, 2.0], [3.0, 4.0] ])
4  b = tf.constant([ [5.0, 6.0], [7.0, 8.0] ])
5  c = tf.matmul(a,b)
6
7  sess = tf.Session()
8  print(sess.run(c))
9  ---------------------------- OUTPUT -----------------------------
10 > [[19. 22.][43. 50.]]
```

Placeholders são tensores indefinidos, os quais receberão um valor posteriormente. Eles são úteis para receber as amostras de entrada e a saída que serão utilizadas no grafo de computação da rede neural. A Listagem 4.2 mostra como usar um *placeholder* no *Tensorflow*. Na linha 4 um *placeholder* do tipo *float* é criado com as dimensões 3x3. Em seguida a biblioteca Numpy é usada para gerar um tensor 3x3 com valores aleatórios. Nas linhas 9 e 10, a sessão é criada e o grafo de computação é executado. Note que desta vez o parâmetro **feed_dict** é explicitamente definido. Esse parâmetro recebe como valor

um *dict* que possui como chave o *placeholder* criado, e como valor o *array* de valores aleatório chamado *rand_array*). Por fim, na linha 13 é mostrada a saída do programa.

Listagem 4.2: Usando placeholders no Tensorflow.

```
1  import tensorflow as tf
2  import numpy as np
3
4  a = tf.placeholder(tf.float32, shape=(3,3))
5  b = tf.matmul(a,a)
6
7  rand_array = np.random.rand(3,3)
8
9  sess = tf.Session()
10 result = sess.run(b, feed_dict={a: rand_array})
11 print(result)
12 ---------------------------- OUTPUT -----------------------------
13 > [[1.9946331 1.5735985 2.126033 ][1.9040278 1.517215  2.0398355]
14 [1.7058356 1.3416395 1.8197424]]
```

4.3. Fundamentos de Aprendizado de Máquina

Aprendizagem de máquina, ou aprendizado automático é um subcampo da área de Inteligência Artificial que automatiza a construção de modelos analíticos a partir dos dados. Em 1959, o cientista da computação Artur Samuel definiu o conceito de aprendizado de máquina como "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados" [**?**]. Em outras palavras, tais algoritmos operam construindo de forma automática um modelo interno a partir das amostras de entrada e fazem previsões guiadas pelos dados ao invés de seguir instruções programadas.

O aprendizado de máquina é usado em um vasto domínio onde algoritmos tradicionais são impraticáveis. Este minicurso é focado especialmente em problemas da área de sistemas multimídia, mas existem aplicações de que vão desde problemas relacionados a robótica até problemas de sequenciamento de DNA. As tarefas de aprendizado de máquina geralmente são categorizados em três tipos: supervisionado, não-supervisionado e por reforço.

Aprendizado supervisionado: O humano fornece amostras pré-classificadas a máquina. O objetivo é aprender uma função que mapeia a entrada para um tipo de saída. Exemplos de tarefas supervisionadas são: classificação e regressão.

Aprendizado não-supervisionado: O humano fornece apenas os dados, sem classificação. O objetivo é encontrar alguma estrutura nos dados. Técnicas de agrupamento (*clustering*) são tipicamente tarefas não-supervisionadas, pois os grupos descobertos não são conhecidos previamente.

Aprendizado por reforço: A máquina recebe sinais (premiações ou punições) de um ambiente dinâmico em que se deve desempenhar um determinado objetivo.

Este minicurso é especializado em aprendizado do tipo supervisionado e aborda principalmente os algoritmos baseados em redes neurais. Por consequência, são utilizados *datasets* anotados, os quais são divididos em conjuntos distintos e são usados para avaliar as capacidades de generalização dos modelos. No método de validação cruzada

geralmente os *datasets* são divididos em três conjuntos: treino, validação e teste. O conjunto de treino é usado para realizar o treinamento do algoritmo, o conjunto de validação é usado para verificar a capacidade de generalização o algoritmo ainda em tempo de treinamento. Após o treinamento o conjunto de teste é usado avaliar o modelo. No entanto, devido ao tamanho pequeno da maioria dos *dataset* utilizados neste minicurso, eles são divididos em apenas dois conjuntos, treino e teste.

4.4. Redes Neurais

Métodos modernos de redes neurais são considerados os mais importantes modelos da área aprendizado de máquina. No entanto, até antes de 2006, não era possível treinar redes neurais para superar técnicas de aprendizado de máquina mais tradicionais (e.g. SVM, árvore de decisão), exceto em alguns domínios de problemas especializados. O que mudou em 2006 foi a descoberta de métodos que possibilitaram o advento dos modelos baseados em redes neurais profundas, também conhecido como aprendizado profundo (em inglês *Deep Learning*). A evolução das redes neurais profundas permitiu que esse tipo de arquitetura se tornasse o principal modelo para tarefas de classificação que estão relacionadas às áreas de visão computacional, reconhecimento de fala e processamento de linguagem natural.

Nesta seção, apresentamos os conceitos básicos de redes neurais. Primeiro, na Subseção 4.4.1 apresentamos os modelos Perceptron e Perceptron de Múltiplas Camadas. Em seguida, na Subseção 4.4.2 é descrito o algoritmo de Retropropagação. Na Subseção 4.4.3, apresentamos um tipo de camada chamada *Softmax*. Por fim, na Subseção é descrita uma implementação que utiliza um Perceptron de Múltiplas Camadas para classificar pontos de um *dataset* artificial.

4.4.1. Perceptron

O Perceptron [Rosenblatt 1957], inventado em 1957 por Frank Rosenblatt, é a estrutura mais básica de uma Rede Neural. A Figura 4.1 ilustra a estrutura do Perceptron, cada entrada x possui um peso w associado. Em seguida é calculado o produto escalar entre os dados de entrada e seus pesos ($z = w_1x_1 + w_2x_2 + ... + w_nx_n = w^t.x$). Então uma função de ativação é aplicada sobre o produto escalar, resultando na saída do perceptron: $a_w(x) = ativ(z) = ativ(w^t.x)$. Algumas fontes da literatura utilizam uma entrada com valor constante 1 para representar o viés (em inglês, *bias*) do neurônio. Em fontes mais recentes, o *bias* é considerado, por padrão, um dado interno do neurônio, resultando na equação: $z = w^t.x + b$.

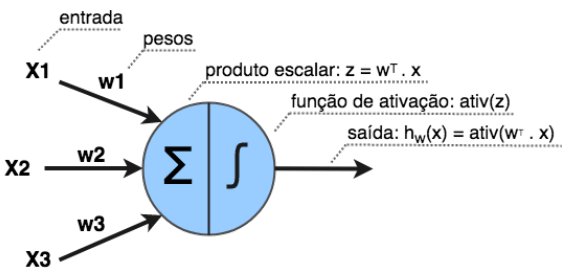


Figura 4.1: Estrutura de um Perceptron

Múltiplos Perceptrons podem ser usados para realizar tarefas de classificação múltipla. Nesse caso, como ilustra a rede A da Figura 4.2, os neurônios são organizados em paralelo e cada um fica responsável por aprender a ativar para uma classe específica. A classe predita é selecionada ao usar a função *argmax* para obter a maior ativação dentre todas as saídas dos neurônios. Esse modelo é conhecido como Perceptron Multiclasse. Adicionalmente, como ilustra a rede B da Figura 4.2, os neurônios também podem ser estruturados em múltiplas camadas, onde cada neurônio das camadas intermediárias (ou escondida) é conectado com todos os neurônios da camada anterior. Os dados da amostra de entrada são considerados os neurônios da camada de entrada, enquanto a última camada da rede é chamada de camada de saída. Nesse caso a rede aprende a aplicar uma hierarquia de transformações lineares ou não lineares (através das ativações) para gerar novas representações do dado de entrada, para que seja possível, por exemplo, realizar classificações. Esse modelo é conhecido como Perceptron de Múltiplas Camadas, ou MLP (do inglês, *Multilayer Perceptron*). Uma rede neural é considerada profunda quando ela possui mais de duas camadas escondidas.

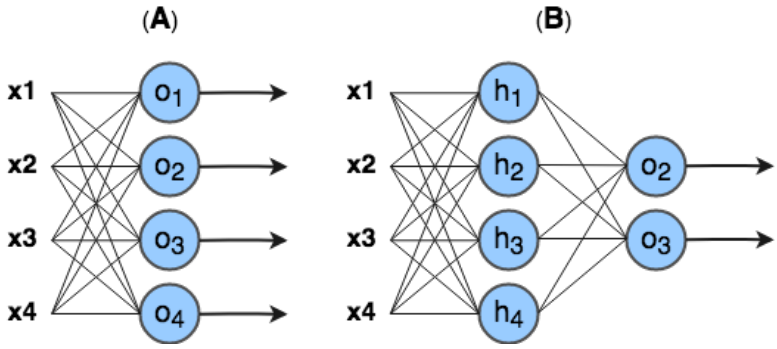


Figura 4.2: (A) Perceptron de múltiplas classes. (B) Perceptron múltiplas camadas.

A Figura 4.3[7] mostra as funções de ativação mais conhecidas. A função de ativação passo (*step*) foi utilizada na primeira versão do Perceptron. No entanto ela não oferece uma derivada útil para que possa ser usada para treinar modelos de múltiplas camadas. Funções de ativação logística (*sigmoid*) e tangente hiperbólica (*tanh*) tornaram-se populares nos anos 80 como aproximações mais suaves da função passo e permitiram a aplicação do algoritmo de retropropagação. Funções de ativação consideradas modernas como linear retificada (*ReLU*) e *maxout* são lineares por partes, computacionalmente baratas e funcionam bem na prática.

O algoritmo que permite o aprendizado da rede neural é chamado de retropropagação (em inglês, *backpropagation*). Basicamente o que se chama de "aprendizado" em redes neurais é o ajuste nos pesos (w) e *biases* (b) dos neurônios para aproximar a saída da rede de uma função $y(x)$ para toda entrada de treinamento x. Para quantificar o quão próximo a rede está do objetivo são utilizadas funções de custo (também conhecidas como funções de perda). Por exemplo, a Equação 1 descreve a função de custo quadrático, comumente utilizada em problemas de regressão. Já a equação 2, descreve a função de custo entropia cruzada, geralmente utilizada em problemas de classificação. No decorrer deste minicurso ambas as funções são utilizadas nas implementações dos projetos práticos.

[7]https://denizyuret.github.io/Knet.jl/latest/mlp.html

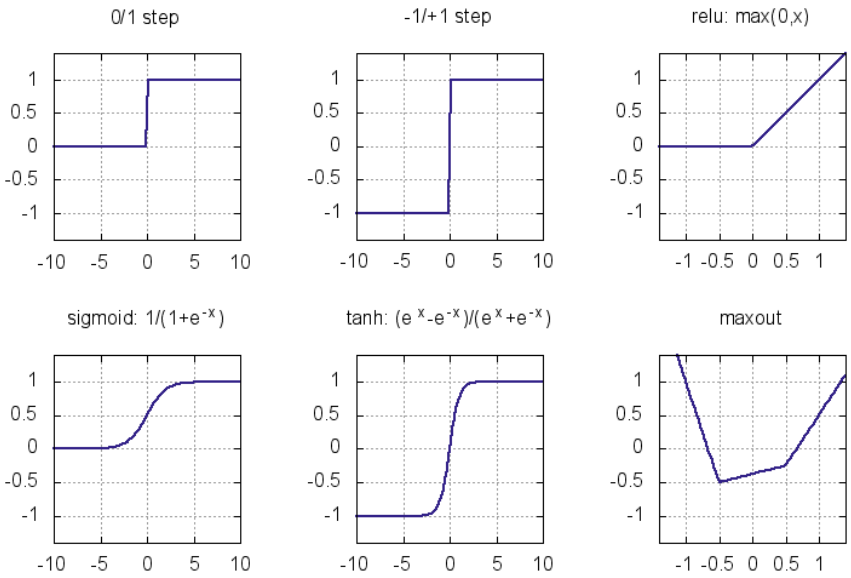


Figura 4.3: Funções de ativação.

$$\mathscr{J} = \frac{1}{2n}\sum_{x} || \, y(x) - a \, ||^2 \tag{1}$$

$$\mathscr{J} = -\frac{1}{n}\sum_{x}[y \log a + (1-y)\log(1-a)] \tag{2}$$

4.4.2. Algoritmo de Retropropagação

O *Tensorflow* encapsula o algoritmo de retropropagação, portanto não é necessário implementá-lo. No entanto, para entender redes neurais é inecessário entender com a retropropagação funciona. Basicamente, o algoritmo de retropropagação busca alterar os valores dos pesos e bias da rede para otimizar a função de custo. Este algoritmo realiza um procedimento para computar o δ_j^l (erro do j-ésimo neurônio da l-ésima camada) e então o relaciona com as derivadas parciais dos pesos e bias em relação a função de custo ($\frac{\partial C}{\partial w_{jk}^l}$ e $\frac{\partial C}{\partial b_j^l}$).

A equação do erro δ^L na camada de saída é dada por:

$$\delta_j^L = \frac{\partial C}{\delta a_j^l}\sigma'(z_j^L) \tag{3}$$

O primeiro termo $\frac{\partial C}{\delta a_j^l}$ mede quão importante é a saída de ativação do j-ésimo neurônio para a função de custo C. Se por exemplo, a saída de um neurônio não contribui para a função de custo, então δ_j^L será pequeno. De forma similar, o segundo termo, $\sigma'(z_j^L)$, mede quão importante é o produto escalar do j-ésimo neurônio para sua saída de ativação.

Ao reescrever a fórmula anterior para uma versão baseada em matriz, obtém-se:

$$\delta^L = \bigtriangledown_a C \odot \sigma'(z^L) \tag{4}$$

Onde, $\bigtriangledown_a C$ é um vetor das derivadas parciais $\frac{\partial C}{\delta a^l_j}$ e o símbolo $\odot$ denota multiplicação elementar entre vetores.

A equação do erro δ^l em relação ao erro na próxima camada (δ^{l+1}) é dada por:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{5}$$

Onde, $((w^{l+1})^T$ é a transposta da matriz de pesos e o δ^{l+1} é o erro da (l+1)-ésima camada. Essa equação permite que o erro seja retropropagado através da rede. Ao usar a equação (1) para computar o δ^L, e então usar a equação (2) em sequência para computar $\delta^{L-1}, \delta^{L-2}, \delta^{L-3}, ..., \delta^1$.

A equação para mudar o custo em relação a qualquer bias e peso são dadas respectivamente por:

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{6}$$

Isto é, o erro δ^l_j é igual a mudança $\frac{\partial C}{\partial w^l_{jk}}$.

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{7}$$

O termo $\frac{\partial C}{\partial w^l_{jk}}$ é calculado em relação ao erro δ^l_j e ativação da camada anterior a^{l-1}_k. Dessa forma, quando a ativação da camada anterior é pequena, espera-se que o termo do gradiente $\frac{\partial C}{\partial w^l_{jk}}$ tenda a ser pequeno.

Com base nas 4 equações descritas, o algoritmo de retropropagação é resumido nos cinco passos seguintes:

1. **Entrada**: Inserção do X (entrada) na rede e cálculo da ativação da camada de entrada.

2. **Propagação**: Para cada camada l = 2, 3, ..., L, calcular a ativação dos neurônios recebendo a ativação dos neurônio da camada anterior como entrada ($z^l = w^l a^{l-1}$ e $a^l = \sigma(z^l)$).

3. **Erro da saída**: Calcular o vetor de erro $\delta^L = \bigtriangledown_a C \odot \sigma'(z^L)$.

4. **Retropropagação do erro**: Para cada camada l = L-1, L-2, ..., 2, calcular o erro da camada $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Atualização dos pesos e bias**: Atualizar os pesos e bias com os respectivos gradientes: $\frac{\partial C}{\partial b^l_j} = \delta^l_j$ e $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$.

4.4.3. Camada *Softmax*

Nesta subseção é apresentada a camada *softmax*, um importante recurso usado em redes que realizam classificação. A ideia do *softmax* é definir uma nova camada saída para a rede com neurônios que usam a função de ativação softmax, a qual é descrita como:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}} \tag{8}$$

Onde o denominador da equação é a soma do produto escalar de todos os neurônios da camada *softmax*. Como resultado, o vetor de saída da camada *softmax* pode ser interpretadas como uma distribuição probabilística. Por exemplo, considerando a camada *softmax* da rede ilustrada na Figura 4.4, se o vetor do produto escalar $(z_1^L, z_2^L, z_3^L, z_4^L) = (0.1, 0.9, 0.4, 2.3)$, então o vetor de ativação $(a_1^L, a_2^L, a_3^L, a_4^L) = (0.07, 0.16, 0.09, 0.66)$. Isso significa que dada entrada X tem 66% de chance de ser da classe 4.

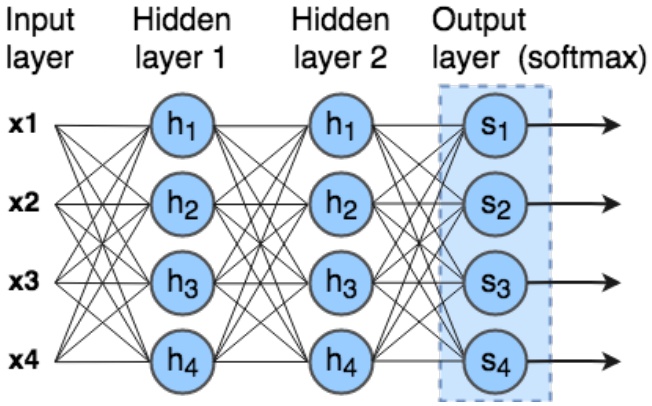


Figura 4.4: MLP Moderno com uma camada *softmax* para classificação.

4.4.4. Implementando um MLP

Nesta subseção é exemplificado o uso de um MLP para resolver um problema não linearmente separável. A Listagem 4.3 mostra o código que cria um pequeno dataset com pontos distribuídos de forma circular usando a biblioteca scikitlearn[8], que pode ser visualizado na Figura 4.5. O dataset é composto por pontos de duas dimensões (duas *features*) que pertencem a dois conjuntos de classe, vermelho (0) e azul (1).

Listagem 4.3: Criando um dataset não linearmente separável.

```
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import sklearn.datasets
5
6 from aux import draw_separator
7
8 # Setando o seed para gerar uma êsequncia  conhecida
9 tf.set_random_seed(0)
```

[8]http://scikit-learn.org/

```
10 np.random.seed(0)
11
12 # numero de classes do nosso problema
13 num_classes = 2
14
15 # gerando o dataset
16 dataset_X, dataset_Y = sklearn.datasets.make_circles(200, noise=0.05)
17 # plotando o dataset
18 plt.scatter(dataset_X[:,0], dataset_X[:,1], s=40, c=dataset_Y,
19              cmap=plt.cm.Spectral)
```

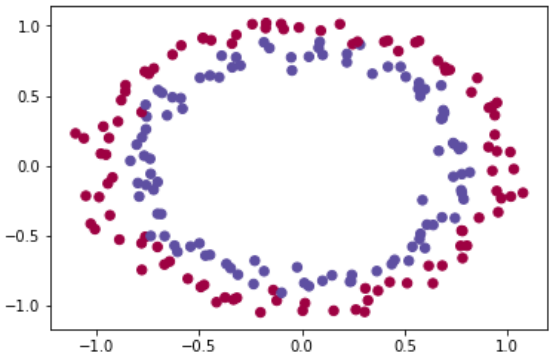


Figura 4.5: Visualização do dataset criado na Listagem 4.3

A Listagem 4.4 descreve a implementação de uma rede neural do tipo MLP. A função "build_net" recebe como parâmetro a quantidade de *features* e classes usadas no problema, neste caso, tanto a quantidade de *features* quanto a de classes são iguais a 2. Nas linhas 4 e 5 são definidos os *placeholders* do grafo de computação. O "X_placeholder" corresponde a camada de entrada da rede, enquanto o "Y_placeholder" é o vetor que contém os *labels* do dataset, e serão utilizados para comparação com a saída da rede. Na linha 8 é definida a camada escondida da rede, a qual possui cem unidades e usa a função de ativação ReLU. Na linha 11 é definida a camada de saída da rede, que neste exemplo possui apenas duas unidades. Na linha 15 o vetor de labels é convertido para o formato *OneHot* (por exemplo, a label 0 se torna o vetor [1,0] e o label 1 se torna o vetor [0,1]), dessa forma é possível a comparação com a saída da rede. Em seguida, na linha 17 é definida a função de perda, a função Entropia Cruzada é usada em conjunto com uma camada *softmax*. Na linha 20 é definido o otimizador da rede. Nas linhas 23 e 24 são definidos os tensores que classificam um exemplo de entrada da rede. Por fim, nas linhas 27 e 28, são definidos os tensores que calculam a acurácia da rede.

Listagem 4.4: Construindo um MLP.

```
1 def build_net(n_features, n_classes):
2
3     # Placeholders
4     X_placeholder = tf.placeholder(dtype=tf.float32, shape=[None,
          n_features])
5     Y_placeholder = tf.placeholder(dtype=tf.int64, shape=[None])
6
7     # camada oculta
```

```
8     layer1 = tf.layers.dense(X_placeholder, 100, activation=tf.nn.relu)
9
10    # camada de saida
11    out = tf.layers.dense(layer1, n_classes, name="output")
12
13    # adaptando o vetor Y para o modelo One-Hot Label
14    one_hot = tf.one_hot(Y_placeholder, depth=n_classes)
15
16    # funcao de perda/custo/erro
17    loss = tf.losses.softmax_cross_entropy(onehot_labels=one_hot,
          logits=out)
18
19    # otimizador
20    opt = tf.train.GradientDescentOptimizer(learning_rate=0.07).
          minimize(loss)
21
22    # classe de exemplo
23    softmax = tf.nn.softmax(out)
24    class_ = tf.argmax(softmax,1)
25
26    # acuracia
27    compare_prediction = tf.equal(class_, Y_placeholder)
28    accuracy = tf.reduce_mean(tf.cast(compare_prediction, tf.float32))
29
30    return X_placeholder, Y_placeholder, loss, opt, class_, accuracy
```

A Listagem 4.5 mostra como iniciar o TensorFlow e carregar o modelo de MLP criado. Na linha 2 é iniciada uma sessão interativa do TensorFlow. Em seguida, na linha 5 é obtida a quantidade de *features* do dataset. Na linha 8 o modelo de MLP criado na listagem anterior é carregado. Por fim, na linha 11, as variáveis do TensorFlow são inicializadas.

Listagem 4.5: Iniciando o TensorFlow e carregando o MLP.

```
1  # iniciando a sessao
2  sess = tf.InteractiveSession()
3
4  # obtendo o numero de features
5  n_features = dataset_X.shape[1]
6
7  # carregando o modelo
8  X_placeholder, Y_placeholder, loss, opt, class_,
9            accuracy = build_net(n_features,num_classes)
10 # inicializando as ávariveis
11 sess.run(tf.global_variables_initializer())
```

A Listagem 4.6 mostra o código que realiza treinamento da rede. Um laço executa o treinamento da rede mil vezes (mil épocas) usando todo o *dataset*. Vale ressaltar que devido ao tamanho pequeno do *dataset*, neste exemplo, o método de dividir o *dataset* em lotes não é usado. A cada 100 épocas o erro da rede é calculado e impresso. Ao fim do treinamento a acurácia da rede é calculada e impressa. A linha 21 exemplifica como utilizar o modelo para realizar uma classificação. Em seguida, na linha 24, a função "draw_separator" desenha o dataset separado pelo modelo, o qual pode ser visto

na Figura 4.6. Por fim, as linhas 27-37 mostram a saída do programa.

Listagem 4.6: Treinamento do MLP.

```
1  # definindo o numero de epocas
2  epochs = 1000
3  for i in range(epochs):
4
5      # treinamento (OBS: mini-batch nao usado por causa do tamanho
           pequeno do dataset)
6      sess.run(opt, feed_dict={X_placeholder: dataset_X,
7                   Y_placeholder: dataset_Y})
8
9      # a cada 100 epocas o erro e impresso
10     if  i % 100 == 0:
11         erro_train = sess.run(loss, feed_dict={X_placeholder: dataset_X
               ,
12                             Y_placeholder: dataset_Y})
13         print("O erro na epoca", i,":", erro_train)
14
15 # calculando a acuracia
16 acc = sess.run(accuracy, feed_dict={X_placeholder: dataset_X,
17                       Y_placeholder: dataset_Y})
18 print("acuracia do modelo:", acc)
19
20 cla = sess.run(class_, feed_dict={X_placeholder: dataset_X[:1]})
21 print("a classe do ponto", dataset_X[:1], "e:", cla)
22
23 # desenhando o separador
24 draw_separator(dataset_X, dataset_Y, sess, X_placeholder, class_)
25
26 ---------------------------- OUTPUT -----------------------------
27 > O erro na epoca 0 : 0.69493294
28 > O erro na epoca 100 : 0.67670804
29 > O erro na epoca 200 : 0.6603513
30 > O erro na epoca 300 : 0.643817
31 > O erro na epoca 400 : 0.6265911
32 > O erro na epoca 500 : 0.60751265
33 > O erro na epoca 600 : 0.58588564
34 > O erro na epoca 700 : 0.56282145
35 > O erro na epoca 800 : 0.5376183
36 > O erro na epoca 900 : 0.510537
37 > acuracia do modelo: 0.97
38 > a classe do ponto [[0.4013312  0.88583093]] e: [0]
```

4.5. Redes Neurais Convolucionais

Redes Neural Convolucionais (CNNs ou ConvNets) são redes especializadas no processamento de dados que são comumente organizados em topologia de grade (no caso mais comum, imagens). Esse modelo recebe este nome porque faz uso de uma operação matemática chamada convolução. As camadas de convolução possibilitam que uma CNN e encontre *features* de baixo nível nas primeiras camadas e então as compõem em *features* de mais alto nível ao decorrer da rede. A habilidade de encontrar uma estrutura hierárquica de *features* é o principal motivo pelo qual CNNs funcionam tão bem para

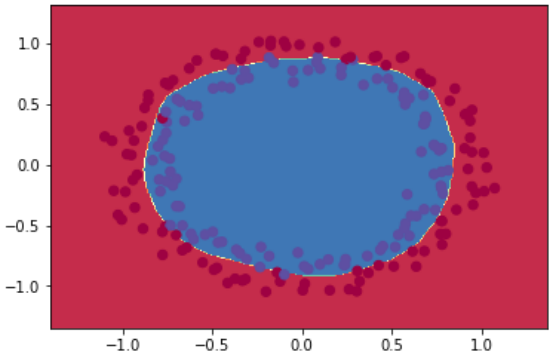


Figura 4.6: dataset separado pelo MLP.

reconhecimento de padrões em imagens.

Nesta Seção, apresentamos os fundamentos básicos e técnicas para implementação de CNNs. Na Subseção 4.5.1 a operação de convolução e *pooling* é apresentada. Em seguida, na Subseção 4.5.2 descreve a implementação de uma CNN para classificação de imagens de sinais de mão. Por fim, na Subseção 4.5.2.1 é apresentado o funcionamento e histórico de evolução da rede InceptionNet.

4.5.1. Camadas de Convolução e Pooling

A convolução consiste de um operador linear que, a partir de duas funções, resulta numa terceira que é a soma do produto dessas funções ao longo da região subentendida pela superposição delas em função do deslocamento existente entre elas. Para funções contínuas, a convolução é definida como a integral do produto de uma das funções por uma cópia deslocada e invertida da outra:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

Para funções de domínio discreto, a convolução é dada por:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Em CNNs a operação de convolução é feita em mais de uma dimensão por vez. Os pesos dos neurônios são representadas por um tensor chamado *kernel* (ou *filter*). O processo de convolução entre os neurônios e os *kernels* produzem saídas chamadas de mapas de *features*. Especificamente, baseado na equação discreta da convolução, a saída de um neurônio localizado na linha i, coluna j do mapa de *features* k em dada camada de convolução l é dada pela equação:

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k'=1}^{f'_n} x_{i',j',k'} . w_{u,v,k',k}$$

onde:

- $z_{i,j,k}$ é a saída do neurônio localizado na linha i, coluna j e no mapa de *features* k da camada convolucional l;
- $x_{i',j',k'}$ é a saída do neurônio localizado na linha i', coluna j' e no mapa de *features* k' da camada anterior (l -1);
- $w_{u,v,k',k}$ é o peso de conexão entre qualquer neurônio do mapa de *features* k da camada l e sua entrada localizada na linha u, coluna v e mapa de *features* k'.
- b_k é o bias para o mapa de *features* k na camada l
- Os parâmetros s_h e s_w representam os *strides* (deslocamentos) verticais e horizontais, f_h f_w são a altura e largura do *kernel*, e $f_{n'}$ é o número de mapa de *features* na camada anterior.

A Figura 4.7 mostra um exemplo de convolução entre dois tensores 2D. O *kernel* (em azul) tem dimensões (2,2), o tensor de entrada (i) tem dimensões (3,3) e o *stride* é igual a 1. Na primeira iteração, a saída (o) descrita pelo cálculo: $(1 \times 3) + (-1 \times 7) + (-1 \times 10) + (1 \times 8)$ = -6; Na segunda iteração, $(1 \times 7) + (-1 \times 4) + (-1 \times 8) + (1 \times 11)$ = 6; Na terceira iteração, $(1 \times 10) + (-1 \times 8) + (-1 \times 12) + (1 \times 1)$ = -9. E por fim, na quarta iteração, $(1 \times 8) + (-1 \times 11) + (-1 \times 1) + (1 \times 2)$ = -2. Note que o tensor de saída possui dimensões diferentes do tensor de entrada, isso ocorre porque ...

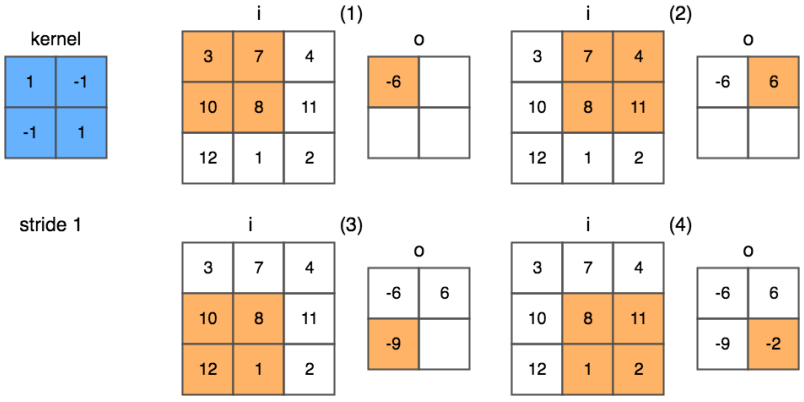


Figura 4.7: Processo de convolução.

Em CNNs a camadas de *pooling* tem a função de reduzir a dimensionalidade dos mapas de features para diminuir a carga computacional, uso de memória e número de parâmetros (dessa forma, reduzindo o risco de *overfitting*). Além disso, a redução de dimensionalidade permite que a rede tolere pequenas mudanças nos mapas de *features* (invariância de localização).

As camadas de *pooling* operam de forma semelhante as camadas de convolução, com a diferença que os *pooling kernels* não possuem pesos. Os *pooling kernels* agregam a entrada através de funções de agregação, como *max* ou *mean*. A função *max pooling*, por exemplo, retorna o maior valor dentro de uma área do tensor. Outras funções de *pooling* incluem, por exemplo, a média ou a distância L^2 entre os elementos de uma área do tensor. A Figura 4.8 ilustra um exemplo do processo de *max pooling*. Cada área colorida

representa uma etapa da operação que usa um *pooling kernel* com dimensões 2x2 e stride 2. Na área de cor laranja o maior valor é 28; Em seguida, na área de cor verde, 21; Na área azul, 27; E por fim, na área lilás, 17.

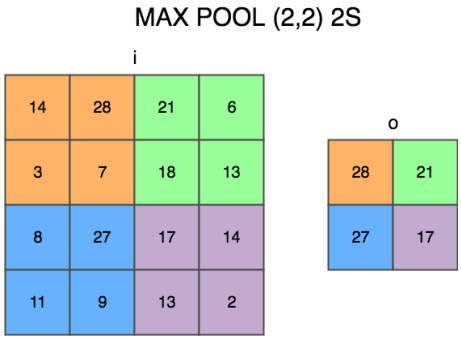


Figura 4.8: Exemplo de um processo de max *pooling* com *kernel* 2x2 e *stride* 2.

É importante também definir o conceito do que é um tensor nesse contexto. Uma rede neural é formada por um conjunto de matrizes, porém, estas podem ter mais de duas dimensões, então nos referenciaremos a elas como tensores. Operações são feitas em tensores para gerar os tensores seguintes, podendo essas reduzir ou aumentar seu numero de dimensões. Ao diminuir as dimensões em um tensor se espera representar um fator comparativo entre valores do tensor anterior para gerar o seguinte. Tal redução representa o aumento da semântica nos dados. Tal conjunto de regras registrada nos pesos, formada pela associa cão de valores, carrega informações sobre o contexto de forma que sejam uteis para diminuir o erro resultante da saída no final da rede.

A Figura 4.9 ilustra a arquitetura de uma CNN que usa camadas de convolução e *pooling*. A entrada da rede consiste de um tensor 16x16x3, correspondente a uma imagem com 16 de altura, 16 de largura e 3 canais (RGB). A primeira convolução usa 8 kernels com dimensões (4,4) e *stride* 1 seguido de uma função de ativação ReLu, resultando em 8 mapas de features com dimensões 16x16. Em seguida, é aplicada uma camada de *pooling* que usa um kernel com dimensões (4,4) e *stride* 4, resultando em mapas de *features* com dimensões reduzidas (4,4). A próxima camada de convolução usa 4 *kernels* (2,2) seguido de um ReLU, resultando em 4 mapas de features com dimensões 4x4. Por fim, uma última camada de *pooling* usa um kernel(4,4) e *stride* 1, resultando em 4 mapas de *features* 1x1. A última camada é então conectada a uma camada de saída *Fully Connected* (FC).

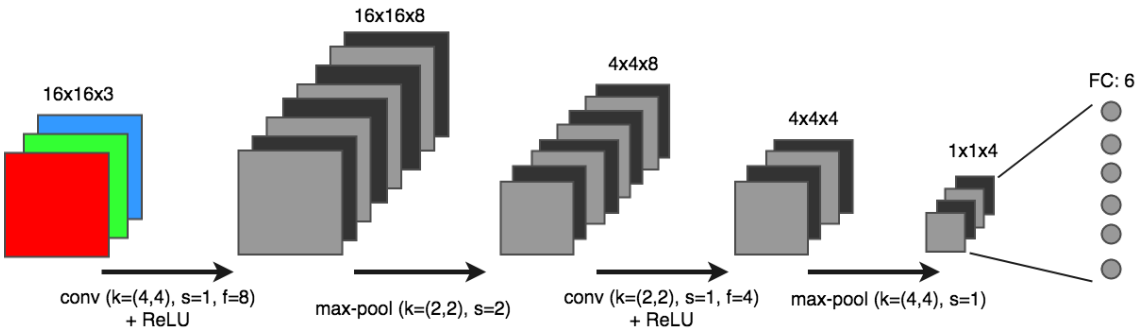


Figura 4.9: Exemplo de uma arquitetura de CNN.

4.5.2. Implementando uma Rede Neural Convolucional

Nesta subseção é exemplificada a implementação de uma CNN para reconhecer imagens de sinais de mãos. O *dataset* usado neste exemplo foi obtido no curso de especialização em Deep Learning do professor Andrew Ng. (deeplearning.ai) [9]. O *dataset* é composto por 1200 fotos de sinais de mão no formato RGB com dimensões 64x64. A Figura 4.10 ilustra os seis tipos de sinais de mãos encontrados no *dataset*, bem como os respectivos vetores de *labels* codificados no formato *OneHot*.

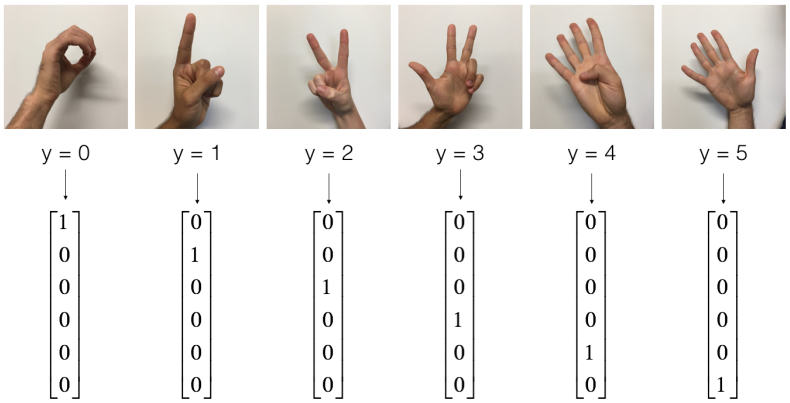


Figura 4.10: Exemplos de cada classe do *dataset* de sinais de mão e suas respectivas codificações OneHot (by Prof. Andrew Ng., deeplearning.ai).

A Listagem 4.7 mostra o código que carrega o dataset de sinais de mão. Nas linhas 1-10 são definidas as bibliotecas necessárias para implementação do exemplo. Na linha 17 o *dataset* é carregado. Em seguida, nas linhas 20-23 as dimensões dos conjuntos de treino e teste são impressas. Nas linhas 26-28 é chamada uma função que mostra uma imagem do conjunto de treino, bem como sua classe. Por fim, as linhas 30-34 mostram a saída do programa.

Listagem 4.7: Carregando o *dataset* de sinais de mão.

```
1  import math
2  import numpy as np
3  import h5py
4  import matplotlib.pyplot as plt
5  import scipy
6  from PIL import Image
7  from scipy import ndimage
8  import tensorflow as tf
9  from tensorflow.python.framework import ops
10 from cnn_utils import *
11
12 # setando o seed para gerar uma êsequncia conhecida
13 tf.set_random_seed(0)
14 np.random.seed(0)
15
16 # carregando o dataset (dividido em treino e teste)
```

[9] https://www.deeplearning.ai/

```
X_train, Y_train, X_test, Y_test, classes = load_dataset()

# imprimindo as õdimenses dos conjuntos de treino e teste do dataset
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(Y_train.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(Y_test.shape))

# exibindo um exemplo
index = 6
plt.imshow(X_train[index])
print ("y =", Y_train[index])
 ---------------------------- OUTPUT ----------------------------
> X_train shape: (1080, 64, 64, 3)
> Y_train shape: (1080,)
> X_test shape: (120, 64, 64, 3)
> Y_test shape: (120,)
> y = 2
```

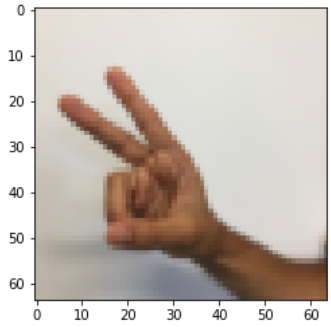


Figura 4.11: Exemplo de imagem renderizada na Listagem 4.7.

A Listagem 4.8 mostra a construção de uma simples arquitetura de CNN. A função "build_cnn" recebe como parâmetro a largura, altura e número de canais da imagem de entrada, bem como o número de classes do problema. Nas linhas 3 e 4 são definidos os *placeholders* da CNN. O "X" é o tensor que armazena as imagens de entrada da rede. Enquanto o "Y" é o vetor que contém as *labels* das imagens de entrada. Entre as linhas 11-26 estão definidas as camadas de convolução e *pooling* da rede. Todas as camadas usam *padding* SAME, ativação ReLU e são inicializadas com o método Xavier. A primeira e a segunda camadas de convolução (linha 11 e 15) tem um *kernel* de dimensões (4,4) e 32 *filters*. Em seguida, na linha 19 é definida a primeira camada de *pooling*, que possui um *kernel* de dimensões (8,8) e usa *stride* 4. A terceira camada de convolução (linha 22) tem um kernel de dimensões (2,2) e 16 *filters*. Por fim, a última camada de *pooling* possui um *kernel* de dimensões (8,8) e também usa *stride* 8. O restante do código possui as definições da função de custo, otimizador e demais tensores já explicados nos exemplos anteriores deste capítulo.

Listagem 4.8: Construindo uma CNN.

```
def build_cnn(input_width, input_height, input_channels, n_classes):

    #placeholders
```

```
4     X = tf.placeholder(tf.float32, shape=(None, input_width,
          input_height, input_channels))
5     Y = tf.placeholder(tf.int64, shape=(None))
6
7     initializer = tf.contrib.layers.xavier_initializer(seed = 0)
8
9   #camada convolucao 1
10    conv2d_1 = tf.layers.conv2d(inputs=X, filters=32, kernel_size
          =[4,4],
11    strides=1, activation=tf.nn.relu, padding = 'SAME',
          kernel_initializer=initializer)
12    #camada convolucao 2
13    conv2d_2 = tf.layers.conv2d(inputs=conv2d_1, filters=32,
          kernel_size=[4,4],
14    strides=2, activation=tf.nn.relu, padding = 'SAME',
          kernel_initializer=initializer)
15    #camada pooling 1
16    maxpool_1 = tf.layers.max_pooling2d(inputs=conv2d_2, pool_size=[8,
          8], strides=4, padding = 'SAME')
17  #camada convolucao 3
18    conv2d_3 = tf.layers.conv2d(inputs=maxpool_1, filters=16,
          kernel_size=[2,2]
19    ,strides=1, activation=tf.nn.relu, padding = 'SAME',
          kernel_initializer=initializer)
20    #camada pooling 1
21    maxpool_2 = tf.layers.max_pooling2d(inputs=conv2d_3, pool_size=[8,
          8], strides=8, padding = 'SAME')
22
23    #flatten
24    flatten = tf.contrib.layers.flatten(maxpool_2)
25
26    #output (fully_connected)
27    out = tf.contrib.layers.fully_connected(flatten, num_outputs=
          n_classes, activation_fn=None)
28
29    #adaptando o Label Y para o modelo One-Hot Label
30    one_hot = tf.one_hot(Y, depth=n_classes)
31
32    #funco de perda/custo/erro
33    loss = tf.losses.softmax_cross_entropy(onehot_labels=one_hot,
          logits=out)
34
35    #Otimizador
36    opt = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)
37
38    #Softmax
39    softmax = tf.nn.softmax(out)
40
41    #Classe
42    class_ = tf.argmax(softmax,1)
43
44    #áAcurcia
45    compare_prediction = tf.equal(class_, Y)
46    accuracy = tf.reduce_mean(tf.cast(compare_prediction, tf.float32))
47
```

```
48      return X, Y, loss, opt, softmax, class_, accuracy
```

A Listagem 4.9 mostra o código que inicializa o TensorFlow e carrega a CNN definida na listagem anterior. Nas linhas 2 e 3 os valores dos pixels das imagens são normalizados para valores entre 0 e 1. Na linha 6 é iniciada uma sessão interativa do TensorFlow. Na linha 9 o modelo de CNN é carregado. E por fim, na linha 13 as variáveis do TensorFlow são inicializadas.

Listagem 4.9: Iniciando o TensorFlow e carregando a CNN.

```
1  #normalizando os dados de entrada
2  X_train = X_train/255.
3  X_test = X_test/255.
4
5  #Iniciando
6  sess = tf.InteractiveSession()
7
8  #carregando o modelo de CNN
9  X, Y, loss, opt, softmax, class_, accuracy =
10             build_cnn(64,64,3,6)
11
12 # inicializando as variveis do tensorflow
13 sess.run(tf.global_variables_initializer())
```

A Listagem 4.10 mostra o código que realiza o treinamento da CNN com o *dataset*. Neste exemplo de implementação o modelo é treinado em 100 épocas. Em cada época, como definido nas linhas 7 e 8, uma lista de *mini-batch* é gerada. Em seguida, a rede é treinada com cada *mini-batch*. O erro do treinamento é impresso a cada 10 épocas. Ao fim do treinamento, a acurácia da CNN treinada é impressa.

Listagem 4.10: Treinando a CNN.

```
1  #definindo o numero de epocas
2  epochs = 100
3
4  seed=0
5  for i in range(epochs):
6      #gerando um mini-batch aleatorio
7      seed = seed + 1
8      minibatches = random_mini_batches(X_train, Y_train, 64, seed)
9      #treinando a rede com cada minibatch
10     for minibatch in minibatches:
11         (minibatch_X, minibatch_Y) = minibatch
12         sess.run(opt, feed_dict={X_placeholder: minibatch_X,
13                  Y_placeholder: minibatch_Y})
14
15     # imprimindo o erro a cada 100 épocas
16     if  i % 10 == 0:
17         erro_train = sess.run(loss, feed_dict={X: X_train, Y: Y_train})
18         print("erro na epoca", i,":", erro_train)
19
20
21 #calculando a acuracia da rede
22 acc = sess.run(accuracy, feed_dict={X: X_test, Y: Y_test})
```

```
print("accurcia do modelo:", acc)
---------------------------- OUTPUT -----------------------------
> erro na epoca 0 : 1.79012
> erro na epoca 10 : 1.53382
> erro na epoca 20 : 0.809482
> erro na epoca 30 : 0.586155
> erro na epoca 40 : 0.505508
> erro na epoca 50 : 0.366477
> erro na epoca 60 : 0.29243
> erro na epoca 70 : 0.240354
> erro na epoca 80 : 0.21093
> erro na epoca 90 : 0.143943
> accurcia do modelo: 0.908333
```

A Listagem 4.11 mostra como utilizar a CNN recém treinada para realizar classificações de sinais de mão. Na linha 4 é feita a predição da classe de uma imagem de entrada. Em seguida a imagem que foi utilizada é desenhada na tela (Figura 4.12). Por fim, a linha 9 mostra a saída do programa.

Listagem 4.11: Usando a CNN treinada para classificar imagens.

```
1 index = 10
2 example = X_test[index:(index+1)]
3
4 cla = sess.run(class_, feed_dict={X: example})
5 print("classe:", cla)
6
7 plt.imshow(X_test[index])
8  ---------------------------- OUTPUT -----------------------------
9 > classe: [5]
```

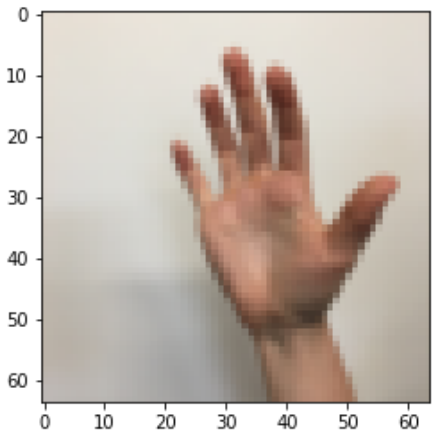


Figura 4.12: Imagem classificada como sinal 5 na Listagem 4.11.

4.5.2.1. Evolução da rede Inception

A primeira versão da rede InceptionNet (ou GoogleNet) [Szegedy et al. 2015] foi a campeã do desafio ImageNet 2014. Essa arquitetura é considerada importante porque

ataca o problema de localização da informação, pois elementos na imagem podem ter grande variedades de tamanhos. Como pode ser visto na Figura 4.13, por exemplo, a área ocupada por um cão é diferente em cada imagem. Por causa dessa variedade, escolher um tamanho de *kernel* apropriado se torna difícil. Um *kernel* largo é adequado quando a informação está distribuída globalmente, e um *kernel* curto quando a informação está distribuída mais localmente.

Figura 4.13: Esquerda: cão ocupando quase toda a imagem; Centro: cão ocupando uma parte da imagem; Direita: cão ocupando uma pequena parte da imagem.

A solução proposta pela rede InceptionNet é de usar *kernels* de diferentes tamanhos no mesmo nível, deixando a rede um pouco mais larga que profunda. Para isso, os autores da rede projetaram o modulo Inception, o qual é ilustrado na Figura 4.14. O módulo Inception realiza convoluções com três diferentes tamanhos de *kernel*, 1x1, 3x3 e 5x5. Adicionalmente é feito um *maxpooling* com um *kernel* 3x3. Para não deixar o processamento tão pesado, ele limita o número de camadas da entrada usando uma convolução 1x1 antes das convoluções 3x3 e 5x5. Apenas no *maxpooling* a convolução 1x1 é realizada posteriormente.

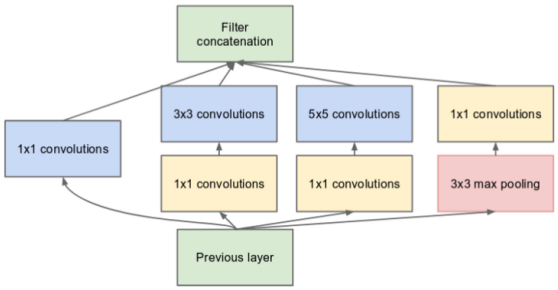


Figura 4.14: Módulo Inception da rede InceptionNet.

Como pode ser visto na Figura 4.15, a rede InceptionNet usa 9 módulos Inception em sequência, resultando em 27 camadas. Por ser uma rede profunda ela esta sujeita ao problema de desaparecimento do gradiente. Para resolver isso, os autores adicionaram duas saídas com classificadores auxiliares na saída de dois módulos Inception. O custo total da rede é dado pela soma ponderada entre o custo dado pelas três saídas da rede.

A arquitetura InceptionNet v2 [Szegedy et al. 2016] tenta reduzir o impacto de um problema conhecido como "gargalo representacional". CNNs funcionam melhor quando as convoluções não alteram a dimensão da entrada de forma drástica, pois essa redução pode causar perda de informação. Para isso, os autores criaram três novas versões do módulo Inception, que refatoraram as convoluções 5x5 em duas convoluções menores. Como

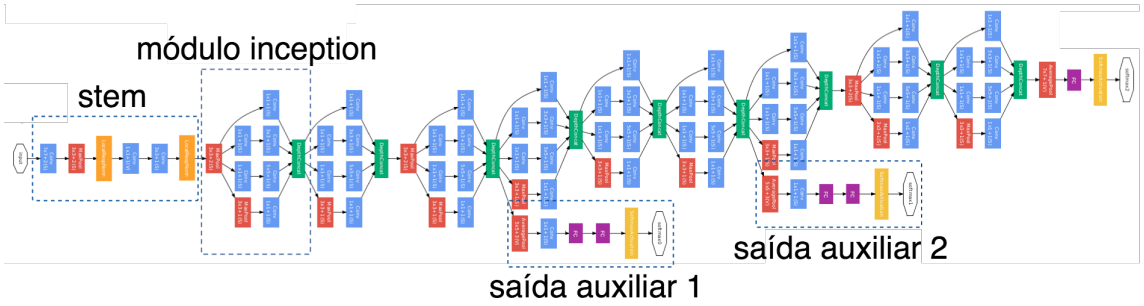


Figura 4.15: Arquitetura da rede InceptionNet (GoogleNet).

pode ser visto na Figura 4.16, no módulo A a convoluções 5x5 foi substituída por uma sequência de duas convoluções 3x3, o que implica em uma melhora de performance, já que uma convolução 5x5 é 2.78 vezes computacionalmente mais cara que uma convolução 3x3. Já no módulo B, os autores substituíram cada convolução 3x3 por uma sequência de 1xn seguida de uma nx1. Por fim, no módulo C a posição das camadas de convolução foram alteradas, de forma que ficaram mais esparsas do que profunda. Essa decisão de projeto tenta suavizar o gargalo representacional, pois se o módulo fosse mais profundo, haveria redução excessiva nas dimensões e, consequentemente, perda de informação.

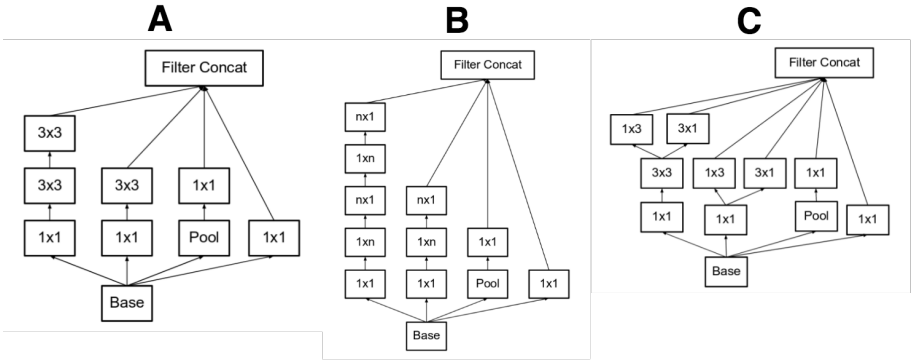


Figura 4.16: Três tipos de módulo inception da arquitetura InceptionNet v2.

A terceira versão, chamada de InceptionNet v3 [Szegedy et al. 2016] adatou o otimizador RMSProp, refatorou as convoluções 7x7 e aplicou a técnica de *Batch Normalization* (*BatchNorm*) as saídas auxiliares. Os autores da rede constataram que as saídas auxiliares não contribuíram muito até o final do processo de treinamento, quando as acurácias se aproximam da saturação. Eles argumentam que elas podem funcionar como regularizes, especialmente se eles tiverem operações *BatchNorm* ou *Dropout*.

A quarta versão, chamada InceptionNet v4 [Szegedy et al. 2017] reformulou alguns módulos da arquitetura. A Figura 4.17 ilustra a arquitetura geral da rede. A Figura 4.18 detalha cada bloco da rede. A InceptionNet v4 apresenta um bloco *Stem* modificado. Os módulo Inception A, B e C são similares aos das versões anteriores. Uma novidade proposta pelo InceptionNet v4 é a definição dos módulos de redução, que são usados para diminuir a dimensionalidade dos mapas de *features*. As versões anteriores já tinham essa funcionalidade, mas ela não estava explicitamente formalizada como um módulo da rede.

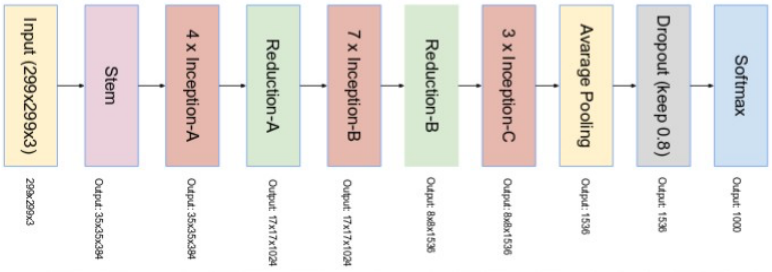


Figura 4.17: Arquitetura da rede InceptionNet v4.

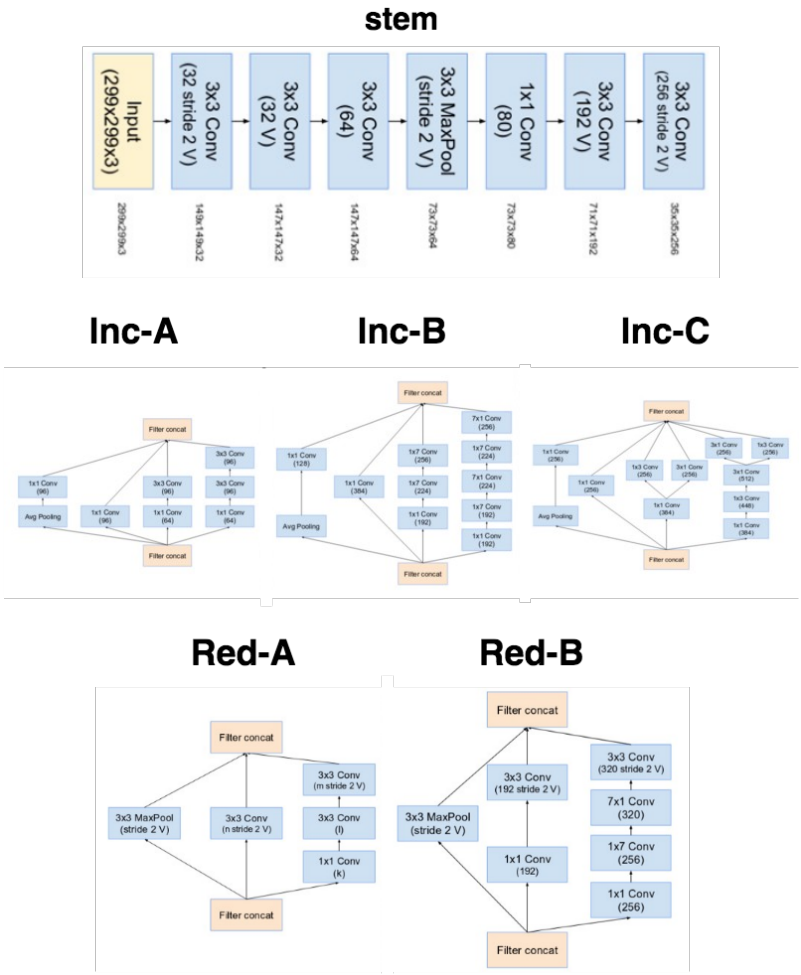


Figura 4.18: (1) Bloco *Stem* da rede InceptionNet v4. (2) IR-A, IR-B e IR-C: Três tipos de módulo Inception da rede InceptionNet v4. (3) Red-A e Red-B - Bloco de redução de 35x35 para 17x17, e 17x17 para 8x8, respectivamente.

Inspirados na performance da rede ResNet [He et al. 2016] (vencedora do desafio ImageNet 2015), os autores do InceptionNet criaram duas redes hibridas chamadas Inception-Resnet v1 e v2 [Szegedy et al. 2017]. A rede Inception-Resnet v1 tem custo computacional semelhante a rede Inception v3, enquanto a rede Inception-Resnet v2 tem custo computacional semelhante a rede Inception v4. A Figura 4.19 ilustra a arquitetura

das redes, ambas possuem a mesma estrutura para os módulos Inception-Resnet A, B e C e blocos de redução. As diferenças são os seus blocos *Stem* e hiper-parâmetros. O Inception-Resnet v1 usa o mesmo *Stem* da versão Inception v4, enquanto o Inception-Resnet v2 propõe o novo *Stem* que pode ser visto na Figura 4.20.

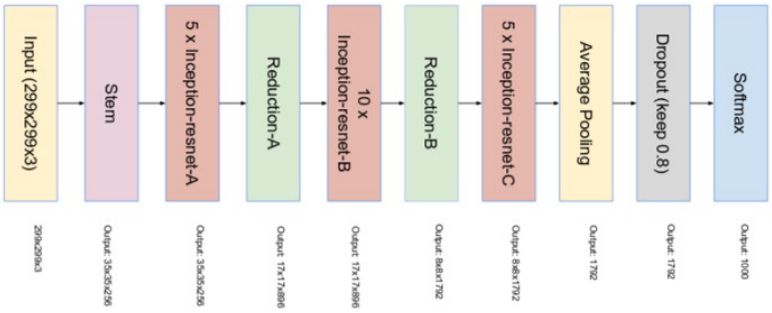


Figura 4.19: Arquitetura das redes Inception-ResNet v1 e v2.

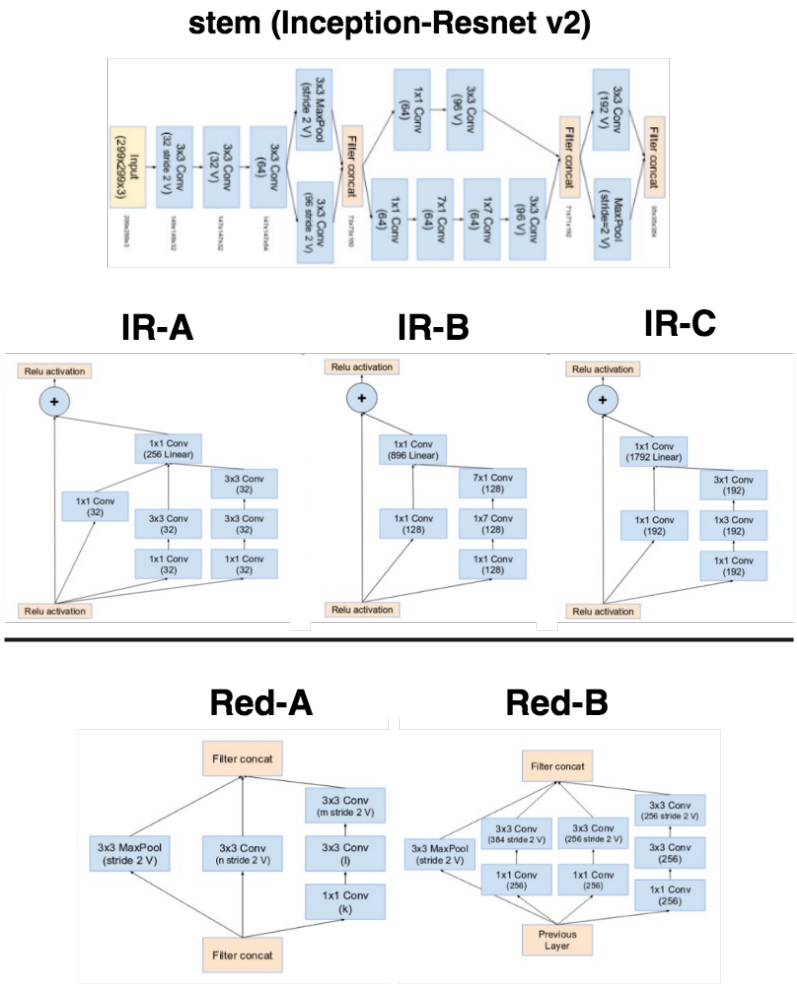


Figura 4.20: (1) Bloco stem da rede Inception-Resnet v2. (2) IR-A, IR-B e iR-C: Três tipos de módulo Inception-Resnet da arquitetura Inception-Resnet v1 e v2. (3) Red-A e Red-B - Bloco de redução de 35x35 para 17x17, e 17x17 para 8x8, respectivamente.

A principal ideia da arquitetura Inception-ResNet é a adição das conexões residuais propostas pela rede ResNet. A Figura 4.20 detalha os módulos da arquitetura Inception-Resnet. Para que a incorporação da conexão residual funcione é necessário que a entrada e a saída sejam concatenadas, e portanto que tenham a mesma dimensão. Para isso, foi adicionada uma convolução 1x1 após as convoluções tradicionais do módulo Inception para padronizar os tamanhos do mapas de *features*. A operação de *pooling* do Inception foi removido em favor da conexão residual. No entanto, essa operação ainda é presente nos blocos de redução A e B. Os autores constataram que a rede tende a instabilidade quando a rede excede mil filtros, para estabilizar a rede eles escalaram as ativações residuais por valores entre 0.1 e 0.3.

4.6. Classificação de vídeo

Nesta seção são apresentadas as técnicas para classificação de vídeo. Como ilustrado na Figura 4.21, a classificação de vídeo consiste de um processo bimodal. Primeiro as CNNs, chamadas de *backbones*, são usadas para extrair as *features* audio-visuais dos frames e áudio do vídeo.

Para extração das *features* visuais é possível utilizar uma das CNNs anteriormente citadas (e.g. Inception, ResNet) pré-treinadas no *dataset* ImageNet [Deng et al. 2009]. Já para extração de *features* de áudio, é possível utilizar uma CNN adaptada para o domínio de áudio, como AudioVGG [Hershey et al. 2017] ou WaveNet [Oord et al. 2016] pré-treinadas no *dataset* AudioSet [Gemmeke et al. 2017]. Após a extração, métodos para agregação de *features* como NetVLAD [Arandjelovic et al. 2016] e LSTM [Hochreiter and Schmidhuber 1997] são utilizados para minar as *features* audio-visuais e realizar a classificação.

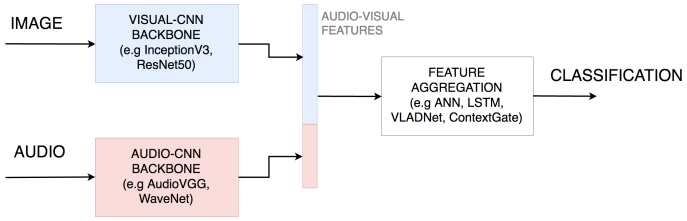


Figura 4.21: Arquitetura da ferramenta de classificação de vídeo.

Para apresentar o processo de classificação, organizamos esta seção como segue. Primeiro discutimos a extração de *features* visuais na subsection 4.6.1. Em seguida, apresentamos métodos de classificação na subsection 4.6.2.

4.6.1. Extração de features

As *features* são resultado da redução de dimensionalidade de camadas de uma rede, ou seja, suas camadas internas. Ao avançar nas camadas, a informação inserida no *input* é filtrada (ou "entrada", refere-se a primeira camada da rede), permanecendo informações que sejam úteis para a redução do erro em sua saída. Nas camadas seguintes da entrada, as informações presentes normalmente são mais simples e voltadas puramente para formas presentes na imagem, como vértices ou curvas. Em camadas mais internas a associação de tais informações podem indicar características como partes de um rosto, um carro ou

cenário; ou até mesmo informações mais complexas como o tipo de veículo presente na imagem ou se o animal presente na imagem é um gato ou não. Lembrando que nem sempre os valores das *features* representam algo discernível contextualmente, muitas vezes são relações entre inputs as quais não temos uma denominação.

As *features* concentram diversas características de um dado (no caso, imagem ou vídeo), as quais, vistas em conjunto, podem simbolizar um contexto maior ou contribuir com uma determinada inferência. Por exemplo, *features* que indicam alta iluminação, presença de água e céu, têm mais chances de estar representando uma praia do que uma apresentação musical.

Um exemplo de uso seria utilizar CNNs pré-treinadas para algum objetivo, como, classificar um animal presente na imagem. Essas redes concentram em suas camadas internas informações sobre sua entrada, logo, pode-se ignorar as camadas finais dessa rede e utiliza-las simplesmente como extratores de *features*. Estas *features* seriam utilizadas em outra rede com objetivo diferente (porém semelhante) da rede que as gerou. Tal técnica é chamada de *Transfer Learning*, quando o aprendizado é reaproveitado em um outro contexto.

4.6.2. Classificação

A normalização *softmax* transforma valores *logits*, ou seja, valores que variam entre mais e menos infinito, em valores que simbolizam probabilidades, que variam de 0 a 1; tendo sua soma igual a 1. Tal transformação é útil pois dadas várias possíveis classes simbolizadas por cada índice, tem-se a probabilidade da ocorrência de cada uma delas. Como mostra a Listagem 4.12

Listagem 4.12: Exemplo de uma função softmax.

```
import tensorflow as tf
import numpy as np

# gerando um input aleatorio
inp = [np.random.rand()*10 for i in range(5)]

# definindo a funcao softmax
softmax = tf.exp(inp)/tf.reduce_sum(tf.exp(inp),0)

with tf.Session() as sess:
    # executando o grafo
    out = sess.run(softmax)
    # imprimindo seus valores e somas
    print("\ninput: \n",inp)
    print("\nsoma input: \n",sum(inp))
    print("\nsoftmax: \n",out)
    print("\nsoma softmax: \n",sum(out))

------- OUTPUT -------
> input:
> [1.6542092596620717, 5.85058565829661, 1.807328616153372,
   9.651413386383854, 0.522533621278154]
>
> soma input:
```

```
> 19.486070541774062
>
> softmax:
> [3.28777882e-04 2.18456835e-02 3.83178820e-04 9.77336347e-01
   1.06028376e-04]
>
> soma softmax:
> 1.000000015636033
```

No código da Listagem 4.12 está a mesma rede feita na sessão 4.4.4 porém com um *dataset* diferente, sendo este, *features* extraídas de vídeos presentes em dois diretórios. Sendo cada diretório representante de uma classe, neste caso, vídeos próprios e vídeos impróprios.

Para a obtenção deste dado é utilizado um extrator de *features* baseado na *ResNet50* para extrair os *features* visuais dos vídeos.

```
from feature_extractor_image import extract_image_features
import tensorflow as tf
import numpy as np
import glob


PROPER_PATH = "./proper/*"
IMPROPER_PATH = "./improper/*"


def build_net(n_features, n_classes):
    '''
    Funcao criada na secao 4.4.4
    '''
    X_placeholder = tf.placeholder(dtype=tf.float32, shape=[None,
        n_features])
    Y_placeholder = tf.placeholder(dtype=tf.int64, shape=[None])
    layer1 = tf.layers.dense(X_placeholder, 100, activation=tf.nn.relu)
    out = tf.layers.dense(layer1, n_classes, name="output")
    one_hot = tf.one_hot(Y_placeholder, depth=n_classes)
    loss = tf.losses.softmax_cross_entropy(onehot_labels=one_hot,logits
        =out)
    opt = tf.train.GradientDescentOptimizer(learning_rate=0.07).
        minimize(loss)
    softmax = tf.nn.softmax(out)
    class_ = tf.argmax(softmax,1)
    compare_prediction = tf.equal(class_, Y_placeholder)
    accuracy = tf.reduce_mean(tf.cast(compare_prediction, tf.float32))
    return X_placeholder, Y_placeholder, loss, opt, class_, accuracy

def extract_features(filepath):
    return np.array(extract_image_features(filepath))


if __name__ == "__main__":

    # inicializando tags e datasets
    proper_tag = 1
```

```
36      improper_tag = 0
37      dataset_X, dataset_Y = [], []
38
39      # obtendo lista de arquivos por classe
40      proprios = glob.glob(PROPER_PATH)
41      improprios = glob.glob(IMPROPER_PATH)
42      print("proprios:",len(proprios),"improprios:",len(improprios))
43
44      # preenchendo dataset proprio
45      for ffile in proprios:
46          print("extraindo features de",ffile)
47          dataset_X.append(extract_features(ffile))
48          dataset_Y.append(proper_tag)
49
50      # preenchendo dataset improprio
51      for ffile in improprios:
52          print("extraindo features de",ffile)
53          dataset_X.append(extract_features(ffile))
54          dataset_Y.append(improper_tag)
55
56      #
57      dataset_X = np.array(dataset_X)
58      dataset_Y = np.array(dataset_Y)
59
60      # obtendo shape do input e output
61      n_features = dataset_X.shape[1]
62      num_classes = dataset_Y.shape[0]
63      print("n features:",n_features,"num classes:",num_classes)
64
65      # executando o modelo como feito na ãsesso 4.4.4
66      sess = tf.Session()
67
68      X_placeholder, Y_placeholder, loss, opt, class_, accuracy =
            build_net(n_features, num_classes)
69      sess.run(tf.global_variables_initializer())
70
71      epochs = 1000
72      for i in range(epochs):
73          sess.run(opt, feed_dict={X_placeholder: dataset_X,
                Y_placeholder: dataset_Y})
74          if i%30 == 0:
75              erro_train = sess.run(loss, feed_dict={X_placeholder:
                    dataset_X, Y_placeholder: dataset_Y})
76              print("erro na epoca",i,":",erro_train)
77
78      acc = sess.run(accuracy, feed_dict={X_placeholder:dataset_X,
            Y_placeholder:dataset_Y})
79      print("accuracia do modelo:",acc)
```

Neste exemplo iniciamos declarando a mesma função **build_net()** feita na seção 4.4.4, seguida da função **extract_features()** que encapsula o uso do extrator de *feature* citado anteriormente transformando sua saída em um vetor **numpy**.

A função **glob()** requisita o sistema em busca de arquivos que batem com o padrão fornecido, de acordo com as regras do sistema *Unix*, como *"./proper/*"* que corresponde

a todos os arquivos presentes no diretório chamado "proper".

Com as listas de arquivos obtidas com a função **glob()** extraimos as features visuais de cada arquivo, atribuindo as tags 0 e 1 dependendo de que diretório o arquivo foi lido. A partir da linha 65 todo o processo é exatamente igual ao que foi feito na seção 4.4.4.

4.7. Detecção de Objetos

Nesta seção descrevemos o modelo YOLO (You Only Look Once) [Redmon et al. 2016], considerado o estado-da-arte na tarefa de detecção de objetos. Sua última versão, chamada YOLOv3 [Redmon and Farhadi 2018] obteve um mAP de 57.9% no dataset COCO [Lin et al. 2014]. O YOLO é ideal para aplicações de tempo-real, visto que é o modelo de detecção de objetos baseado em CNN mais rápido da literatura, chegando a rodar próximo de 30 FPS na GPU Pascal Titan X[10].

A Subseção 4.7.1 descreve a arquitetura geral do YOLO. Em seguida, a Subseção 4.7.2 descreve a implementação de um cenário de uso que usa o YOLO para identificar objetos em imagens.

4.7.1. Arquitetura YOLO

O YOLO divide a imagem de entrada em uma grade de $S \times S$ dimensões. Cada célula pode conter *B bounding boxes* (BBs) e *scores* de confiança para cada uma. O score de confiança reflete o quão a rede tem certeza que a BB contém um objeto. Se não existe objetos na célula, então o score de confiança deve ser zero. Caso contrário, o score de confiança deve ser condicionada pela Interseção sobre União (explicada na Subseção seguinte) entre a BB predita e a BB do *ground truth*, $Pr(Object) * IOU_{pred}^{truth}$.

No YOLO, cada BB contém as seguintes informações: 1) score de confiança da célula conter um objeto; 2) coordenadas da BB (b_x, b_y, b_h, b_w), onde (b_x, b_y) representa o ponto central da BB relativa a uma célula da grade, enquanto (b_h, b_w) representa a altura e largura da BB relativa às dimensões da imagem de entrada; 3) Um vetor de probabilidades $(c_1, c_2, ..., c_n)$ para cada uma das n classes de objetos, onde cada probabilidade de classe é condicionada pela probabilidade da célula conter um objeto, $Pr(Class_i|Object)$.

O *score* da confiança de cada classe em cada BB é dada por:

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth}$$

Os *scores* informam: (1) a probabilidade de um objeto de uma classe aparecer na BB, e (2), o quão ajustado está a BB ao objeto. Como ilustra a Figura 4.22, a saída do YOLO é um tensor de dimensões $S \times S \times (B * 5 + C)$. Onde S é a dimensão do *grid* de regiões, B é o número de *bounding boxes* em cada célula, e C é a quantidade de classes no problema. A Figura 4.23 (cima) mostra três visualizações da saída do YOLO. A imagem da esquerda mostra a entrada dividida em uma grade $S \times S$ regiões. A imagem do meio mostra o mapa de probabilidade de classes para cada célula da grade. Por último, a imagem da direita mostra as BBs.

[10] https://www.nvidia.com/pt-br/geforce/products/10series/titan-x-pascal

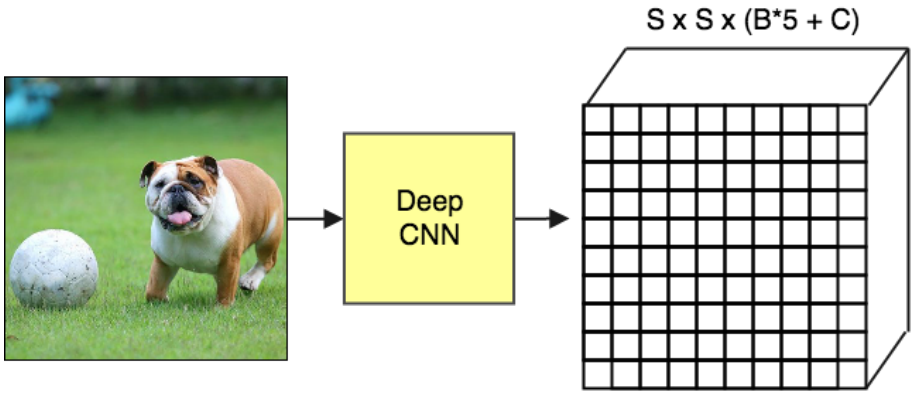


Figura 4.22: Esquema arquitetural do YOLO.

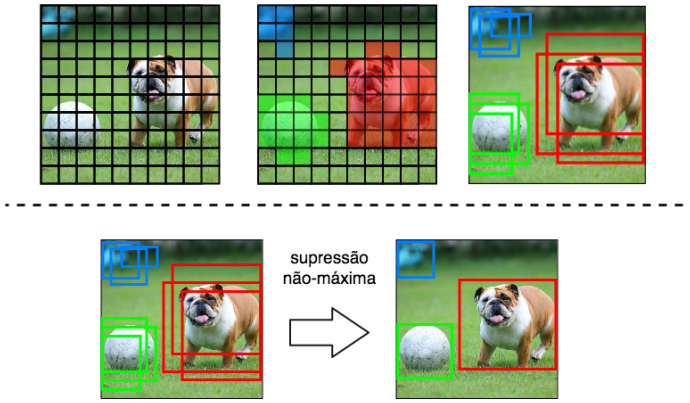


Figura 4.23: Visualização da saída do YOLO (parte superior) e remoção das BBs sobrepostas com o filtro de supressão não-máxima (parte inferior).

4.7.1.1. Supressão não-máxima

Mesmo com a filtragem pelo *score*, muitas BBs podem ficar sobrepostas uma as outras, como ilustrado na Figura 4.23 (baixo). Um segundo tipo de filtro chamado "supressão não-máxima" é necessário para remover as BBs sobrepostas. A supressão não-máxima utiliza uma técnica chamada "Interseção sobre União" (em inglês, *IoU - Intersection over Union*). Como pode ser visto na Figura 4.24, essa técnica consiste basicamente em dividir a interseção pela união de duas BBs.

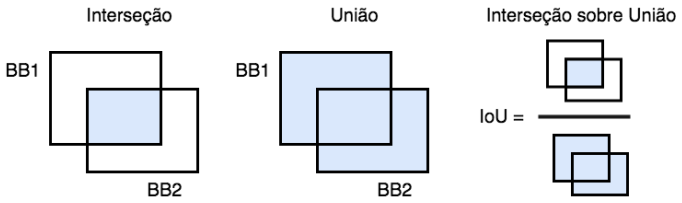


Figura 4.24: Visualização da operação de IoU.

A Listagem 4.13 mostra como implementar o IoU, nas linhas 3-7 é calculada a área de interseção entre as BBs. Em seguida, nas linhas 10-12 é calculada a área de união

entre as BBs. Por fim, na linha 16 é calculado o Iou pela divisão da área de interseção pela área de união.

Listagem 4.13: Função de cálculo do IoU.

```
1  def iou(box1, box2):
2      #Calculando a intersecao entre as BBs
3      xi1 = max(box1[0], box2[0])
4      yi1 = max(box1[1], box2[1])
5      xi2 = min(box1[2], box2[2])
6      yi2 = min(box1[3], box2[3])
7      inter_area = (xi2 - xi1)*(yi2 - yi1)
8
9      #Calculando a uniao usando a formula: Union(A,B) = A + B - Inter(A,
           B)
10     box1_area = (box1[2] - box1[0])*(box1[3] - box1[1])
11     box2_area = (box2[2] - box2[0])*(box2[3] - box2[1])
12     union_area = box1_area + box2_area - inter_area
13
14     # Calculando o IoU
15     iou = inter_area / union_area
16
17     return iou
```

4.7.1.2. Função de perda

Durante o treinamento o YOLO otimiza uma função composta por 5 partes. Cada parte é uma equação que realiza uma tarefa específica.

$$\mathscr{J} = eq1 + eq2 + eq3 + eq4$$

Para aumentar a sua estabilidade, o YOLO aumenta a perda da predições da coordenada das BBs e diminui a perda das BBs que não contém objetos. Para isso, dois parâmetros λ_{coord} e λ_{noobj} são definidos com valores 5 e 0.5, respectivamente.

A equação descrita abaixo calcula a perda relativa a posição **(x,y)** da BB. O 1_{ij}^{obj} denota se a j-ésima BB na i-ésima célula é responsável pela predição do objeto. o YOLO predita múltiplas BB por célula, durante o treinamento somente uma BB é responsável por cada objeto. Então uma BB recebe a responsabilidade de predizer baseado no maior IoU com a BB do *ground truth*.

$$eq1 = \lambda_{coord} \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

A equação descrita abaixo calcula a perda relativa a largura e altura **(w,h)**. A equação é similar a primeira, com a diferença do uso das raízes quadradas para fazer com que pequenas variações em BBs largas importem menos que em BBs pequenas.

$$eq2 = \lambda_{coord} \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

A equação abaixo calcula a perda associada ao *score* de confiança para cada BB, onde C é o score de confiança e $\hat{C}$ é o IoU entre a BB predita e a BB do *ground truth*. O termo 1_{ij}^{noobj} denota se a j-ésima BB na i-ésima célula não é responsável pelo objeto.

$$eq3 = \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=o}^{S^2} \sum_{j=o}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$

A última equação calcula a perda da classificação. Ela é similar a equação de erro de soma quadrada tradicional, mas com a adição do termo 1_i^{obj}. Este termo denota se o objeto aparece na i-ésima célula, é usado para que o erro de classificação não seja penalizado quando o não houver um objeto em uma célula.

$$eq4 = \sum_{i=o}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

4.7.2. Cenário de Uso: Sistema de Detecção de Objetos

A forma mais fácil para usar o modelo YOLO é importando o *framework* Darkflow (Tensorflow + Darknet[11]). O *framework* Darknet é escrito em C e CUDA[12] e foi usado para a implementação oficial do modelo YOLO. O Darkflow é uma re-implementação em Python do Darknet usando o Tensorflow como base.

Como descreve os comandos abaixo, para instalar o Darkflow basta realizar o download do repositório no Github[13]. E em seguida, realizar a instalação do Darknet usando o Pip. Vale ressaltar que é necessário ter o pacote Cython[14] instalado.

```
1 git clone https://github.com/thtrieu/darkflow.git
2 cd darkflow
3 pip3 install .
```

Após realizar a instalação, para utilizar o pacote basta importa-lo para o projeto, como mostra a Listagem 4.14. Na linha 6, é criado um dicionário chamado *options*, que define os atributos necessários para executar o YOLO pré-treinado no *dataset* COCO. O atributo "model" especifica o caminho do arquivo "yolo.cfg", que define a arquitetura da CNN usada. O atributo "load" define o caminho para o arquivo "yolo.weights", que são os pesos da rede pré-treinada no *dataset* COCO. Esse arquivo pode ser baixado no Drive[15] do autor da rede. O atributo "threshold" define o percentual mínimo para confidencia de detecção de objetos, nesse caso só objetos com pelo menos 10% de *score* de confidencia

[11] https://pjreddie.com/darknet
[12] https://developer.nvidia.com/cuda-zone
[13] https://github.com/thtrieu/darkflow
[14] http://cython.org
[15] https://drive.google.com/drive/folders/0B1tW_VtY7onidEwyQ2FtQVplWEU

são retornados. O atributo "gpu" define se o programa pode fazer uso da GPU do sistema. Em seguida, na linha 10, é instanciada uma rede que recebe as opções definidas. É importante ressaltar que também é necessário ter o arquivo "coco.names" na pasta "cfg", esse arquivo é encontrado no repositório do Darknet e contém os nomes das classes do *dataset* COCO.

Listagem 4.14: Configurando a aplicação com os dados pré-treinados.

```
1  from darkflow.net.build import TFNet
2  import matplotlib.pyplot as plt
3  import cv2
4  from draw_boxes import *
5
6  options = {"model": "cfg/yolo.cfg",
7             "load": "cfg/yolo.weights",
8             "threshold": 0.1,
9             "gpu": 1.0}
10 tfnet = TFNet(options)
```

Agora, desejamos criar um *boundingbox* em objetos identificadas, como ilustrado na Figura 4.25.

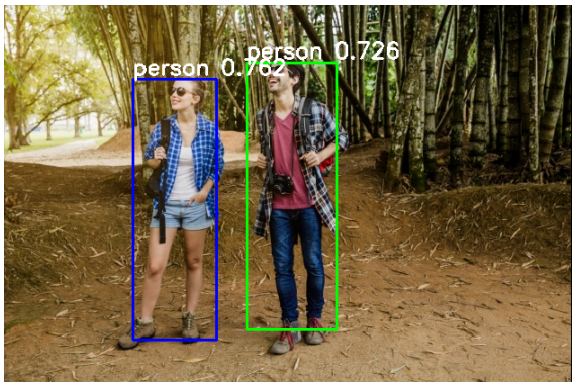


Figura 4.25: Imagem com as *bounding boxes* preditas pelo YOLO.

A Listagem 4.15 mostra como utilizar o modelo YOLO para detectar objetos. Nas linhas 1 e 2 um imagem é aberta e convertida para RGB. Em seguida, na linha 3, a imagem é usada como entrada da rede. Na linha 4 o resultado é impresso na tela. Por fim, nas linhas 6 e 7 a imagem de entrada é exibida com as BBs identificadas com pelo menos 30% de confidência (Figura 4.25). As linhas 9-10 mostram a saída do programa. Nota-se que o YOLO encontrou seis BBs, das quais apenas duas possuem score de confidência suficiente para ser desenhada.

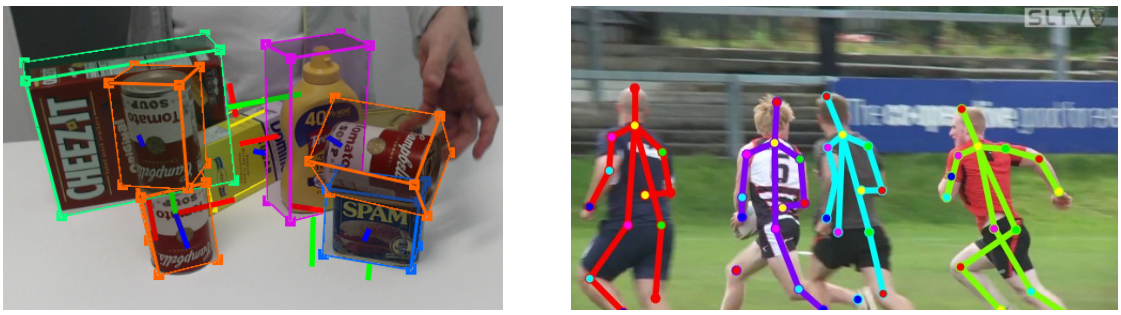
Listagem 4.15: Usando o YOLO para detectar objetos.

```
1  img = cv2.imread("images/sample1.png")
2  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
3  result = tfnet.return_predict(img)
4  print(result)
5
6  plt.imshow(boxing(img, result, 0.3))
7  plt.show()
```

```
 ---------------------------- OUTPUT -------------------------------
[{'label': 'person', 'confidence': 0.15689932, 'topleft': {'x': 314, 'y': 82},
'bottomright': {'x': 379, 'y': 253}},
{'label': 'person', 'confidence': 0.7260812, 'topleft': {'x': 268, 'y': 64},
'bottomright': {'x': 367, 'y': 358}},
{'label': 'person', 'confidence': 0.7622834, 'topleft': {'x': 143, 'y': 82},
'bottomright': {'x': 235, 'y': 369}},
{'label': 'handbag', 'confidence': 0.2231428, 'topleft': {'x': 198, 'y': 178},
'bottomright': {'x': 233, 'y': 239}},
{'label': 'skis', 'confidence': 0.12105702, 'topleft': {'x': 313, 'y': 103},
'bottomright': {'x': 375, 'y': 271}},
{'label': 'snowboard', 'confidence': 0.2203493, 'topleft': {'x': 342, 'y': 159},
'bottomright': {'x': 371, 'y': 257}}]
```

4.8. Estimação de pose

O principal objetivo da estimação de pose é, a partir de dados obtidos por meio de um sensor óptico, seja ele um sensor de profundidade ou uma câmera, é determinar a posição espacial de um objeto ou ser. Naturalmente, estimar a posição de uma pessoa apresenta um desafio maior, pois além de localizar uma pessoa, também é interessante capturar a disposição de seus membros ou articulações. Existem muitas aplicações para a estimação de pose, entre elas, animação, jogos, monitoramento e interação natural. Nesta seção, trataremos especificamente da estimação de pose de pessoas. Na Figura 4.26 pode-se observar exemplos de estimação de pose para objetos e pessoas.



(a) [Tremblay et al. 2018]

(b) [Iqbal et al. 2017]

Figura 4.26: Estimação de pose para objetos (a) e pessoas (b). Na imagem (b) os pontos de interesse (keypoints) são apresentados como círculos coloridos.

Existem muitos pontos a ser levar em consideração na criação de modelos de deep learning para estimação de pose, por exemplo, a quantidade de frames ou imagens usadas para fazer uma predição, O tipo de sensor utilizado na captura dos dados, o numero de dimensões que serão levados em conta para a estimação, etc.

Quanto à quantidade de imagens levadas em conta para obtenção do resultado fi-

nal, temos duas opções: *Singleframe* ou *Multiframe*, na qual a primeira consiste no uso de um único frame ou imagem para realização da estimação. Já na segunda opção, a técnica utilizada leva em conta um ou mais frames, buscando-se manter a trajetória de indivíduos já detectados. O uso de múltiplos frames não implica que todos eles serão necessariamente usados diretamente na predição, mas que podem ser usados em conjunto com outras técnicas para extrair informação temporal. A desvantagem do uso de múltiplos frames é o custo computacional para a estimação, que geralmente implica é maiores tempos para predição e/ou a necessidade de hardware mais robusto.

Quanto aos tipos de dados a serem utilizados os mais utilizados são imagens RBG convencionais, existem também outros trabalhos que usam dois ou mais sensores, imagens de profundidade. Sendo imagens de profundidade os dados usados pelo Kinect, um dispositivo popular que usa estimação de pose, baseado em Randomized Decision Forests [Shotton et al. 2011] anterior à popularização do deep learning. Na estimação de pose pode-se partir de dados 2D (imagens RBG monoculares ou binoculares) ou de dados em 3D (imagens RGBD ou de sensores de profundidade) e estimar coordenadas em 2D ou 3D.

Um algoritmo para estimação de pose pode ser *single-person* ou *multi-person*, apenas uma pessoa é detectada na abordagem *single-person*, a presença de mais de uma pessoa na imagem geralmente causa erros na predição. Já a abordagem *multi-person* detecta todas as pessoas em uma imagem, agrupando seus *keypoints* (pontos de interesse representativos do corpo humano) por pessoa. Apesar de ser mais lenta, a abordagem *multi-person* é mais robusta e mais confiável que a abordagem *single-person*.

Geralmente, a entrada de uma rede convolucional para estimação de pose é uma imagem, ou dados de profundidade representados como uma imagem, de tamanho fixo. Uma rede convolucional treinada tem o número de parâmetros fixo, portanto imagens maiores ou menores que o tamanho padrão para que a rede foi treinada tem que ser escaladas. Já a saída dessas redes é um conjunto de mapas de calor (*heatmaps*) para cada *keypoint*, no qual cada heatmap contém a predição da probabilidade da presença desse *keypoint* para cada pixel. A partir desses mapas de calor é possível usar diferentes ténicas para obter as coordenadas dos *keypoints* preditos, sendo simplesmente medir a coordenada de máxima ativação (ponto de maior valor) uma das técnicas mais usadas. Exemplos de *heatmaps* para alguns *keypoints* podem ser observados na Figura 4.27.

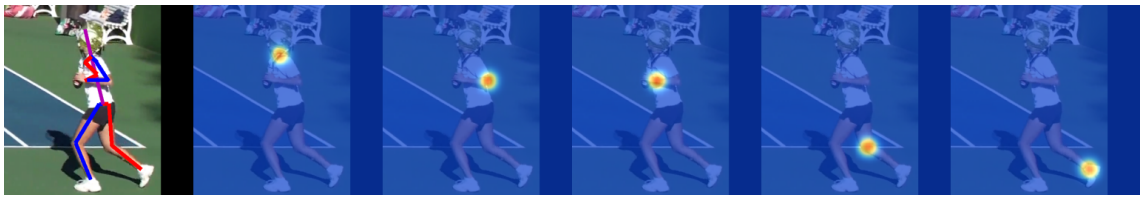


Figura 4.27: Exemplo de *heatmaps* preditos para diferentes *keypoints*. Fonte: [Newell et al. 2016]

Existem tambem duas grandes abordagens quanto ao agrupamento de *keypoints* em uma imagem. Na abordagem *top-down* primeiro se utiliza um algoritmo de detecção de pessoas na imagem original, obtendo-se assim imagens menores, recortadas, de cada pessoa detectada na imagem original. Para cada imagem gerada pelo detector de pessoas,

é executado algoritmo de estimação de pose e ao fim, se obtém todos os *keypoints* de cada pessoa mapeados na imagem original, agrupados por pessoa detectada, assim como pode ser visto pelo exemplo na Figura 4.28. A abordagem *bottom-up* por outro lado utiliza somente a imagem original. Se detecta todos os *keypoints* de todas as pessoas, sem identificação sobre qual *keypoint* pertence a qual pessoa e então se utiliza uma técnica de agrupamento para os *keypoints*, técnica essa que pode ou não ser baseada em machine learning. Por exemplo, [Cao et al. 2018] e [Insafutdinov et al. 2016] tratam o problema de atribuição de *keypoints* à pessoas distintas como um grafo.

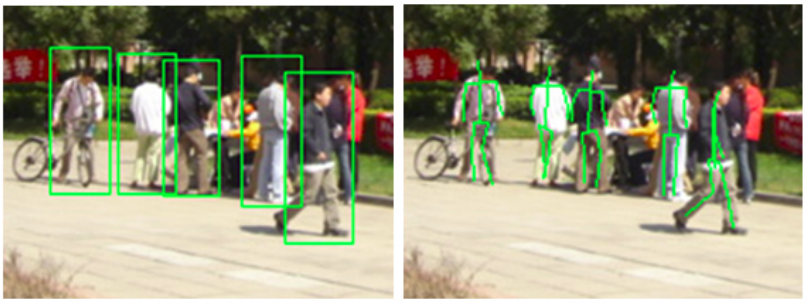


Figura 4.28: Exemplo de abordagem top-down para estimação de pose. Adaptado de [Wang et al. 2010]

Figura 4.29: Exemplo de abordagem bottom-up para estimação de pose. Fonte: [Insafutdinov et al. 2016]

4.8.1. Métricas de avaliação populares

Dentre as métricas de avaliação apresentaremos a Mean Average Precision (mAP) para estimação de pose *single-frame*, Multiple Object Tracking Accuracy (MOTA) para avaliação de estimação de pose para video e as métricas de similaridade/acerto Percentage of Correct Keypoint relative to head (PCKh) e Object Keypoint Similarity (OKS).

A métrica Mean Average Precision (mAP) consiste na media da pontuação Average Precision (AP) [Everingham et al. 2010] para todos os keypoints. Para se obter a pontuacao AP, primeiro se computa a curva precision/recall para todas as predições ordenadas. Recall é definido pela proporção de todos os Verdadeiros Positivos (TP) sobre todos os Positivos (TP + FP). Precision ou Precisão é a proporção Verdadeiros Positivos (TP) e os Verdadeiros Positivos e Falsos Negativos (TP + FN). A métrica AP resume a curva precision/recall é definida pela precisão média em 11 níveis de recall *r* igualmente espaçados [0.0,0.1,0.2,...,1.0]. Sendo AP definido por:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} P_{interp}(r)$$

Onde a precisão em cada nível de recall é interpolada pela precisão máxima correspondente a qualquer r maior que $\bar{r}$:

$$P_{interp}(r) = max_{\bar{r}:\bar{r}>r} p(\bar{r})$$

A métrica MOTA (Multiple Object Tracking Accuracy) é talvez a métrica mais utilizada para tracking [Stiefelhagen et al. 2006], pois combina três fatores de erro:

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t}$$

- FN: Falsos Negativos, são os pontos *ground-truth* em que não houve *tracking*, ou seja, em que os pontos preditos estiveram fora dos limites de distancia do ponto do *ground-truth*;
- FP: Falsos Positivos, são os pontos preditos fora dos limites de distancia do ponto *ground-truth*;
- $IDSW$: Mudanças de id, ocorrem quando há fragmentação na predição de uma trajetória. Um algoritmo de *tracking* gera um novo id sempre que encontra um objeto não rastreado em um ou mais dos últimos frames. Quando o algoritmo perde *tracking* de um objeto e o reencontra (fragmentação), e trata-o como se fosse um novo objeto, é contabilizada uma mudança de id.

Onde t é o índice do frame e GT o número de pontos *ground-truth* (pontos da trajetória real). Na Figura 4.30 observa-se exemplos de fragmentação e mudanças de id, sendo a linha tracejada o trajeto *ground-truth* (GT). As linhas vermelhas e azuis são o trajeto interpolado a partir das predições, cada cor representa o tracking de um id diferente. Os pontos preenchidos de cor azul e vermelho são as predições corretas (True Positive). Os pontos vermelhos e azuis não-preenchidos são predições False Positive (FP). Os pontos cinza são pontos False Negative (FN). Por fim, os pontos preenchidos de preto mas realçados por vermelho ou azul representam os pontos verdadeiros que foram considerados rastreados mesmo que a predição não seja exata. O sombreado cinza em torno da linha tracejada representa os limites de distancia do trajeto *ground-truth* para se considerar um ponto como rastreado ou não.

Como proposto por [Andriluka et al. 2014] a métrica PCKh (Percentage of Correct Keypoint relative to head) é uma derivação da métrica PCK (Probability of Correct Keypoint) criada por [Yang and Ramanan 2012]. A métrica PCK avalia a acurácia da localização da coordenada predita em relação à coordenada verdadeira (*ground-truth*) como um limite de distancia do ponto predito ate o ponto *ground-truth* definido por $\alpha = max(h, w)$

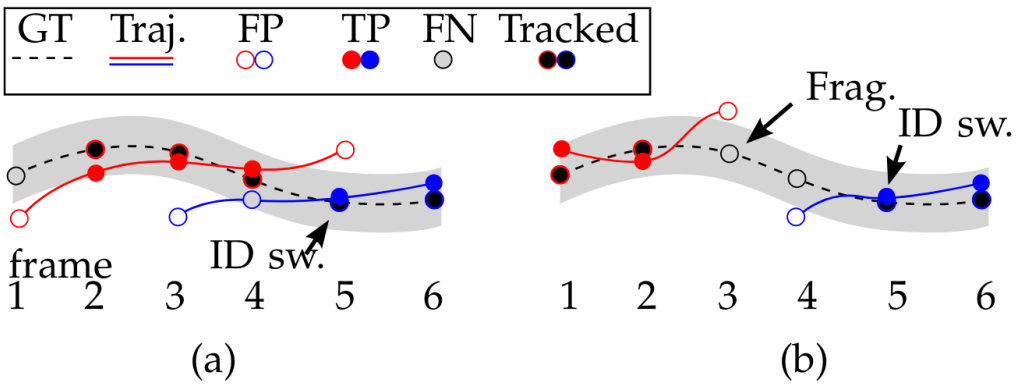


Figura 4.30: Exemplo de mudanças de id e fragmentação segundo a métrica MOTA. Imagem adaptada de [Milan et al. 2016].

onde α é uma constante percentual que controla o limite relativo para considerar um acerto e h e w são a altura e largura, respectivamente, da *bounding box* da pessoa. A métrica PCKh é definida como 50% do tamanho do segmento da cabeça, sendo o segmento da cabeça denotado pela distancia do keypoint que representa a cabeça até o keypoint que representa a clavícula/base do pescoço.

OKS ou Object Keypoint Similarity[Ruggero Ronchi and Perona 2017] é uma métrica de similaridade entre dois keypoints, é a métrica de similaridade utilizada para avaliar modelos baseado no dataset COCO (Common Objects in Context) [Lin et al. 2014] o qual abriga uma grande quantidade de imagens anotadas com keypoints de pessoas.

Simplificando, a OKS funciona como o IoU funciona na detecção de objetos. É calculada pela distância entre os pontos preditos e os pontos *ground-truth* com valores multiplicados por uma distribuição normal em torno dos pontos *ground-truth*. Essa distribuição de valores de OKS de acordo com a distância do ponto predito até o ponto real pode ser observado na Figura 4.31.

Para o cálculo do OKS são necessários os vetores preditos e *ground-truth* contendo as coordenadas de todos os keypoints, por exemplo temos um vetor de predição $x = [x_1, y_1, v_1, ..., x_k, y_k, v_k]$ onde x e y (podem ser mais dependendo da dimensão) são as coordenadas do keypoint k e v é uma flag de visibilidade do keypoint(também pode ser predita, sendo o equivalente à confiança da predição) com:

- $v = 0$: não detectado;
- $v = 1$: detectado mas não visível;
- $v = 2$: detectado e visível.

A equação que define a OKS é:

$$OKS = \frac{\sum_i \exp\left(-d_i^2/2s^2k_i^2\right)\delta\left(v_i > 0\right)}{\sum_i \delta\left(v_i > 0\right)}$$

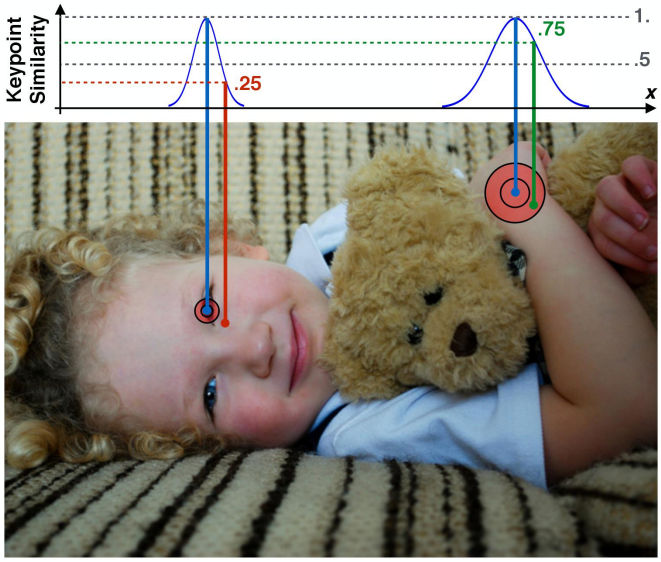


Figura 4.31: Exemplo da distribuição dos valores de OKS de acordo com a distancia entre os pontos preditos (vermelho e verde) e os pontos *ground-truth* (azuis). Fonte: [Ruggero Ronchi and Perona 2017].

Onde d_i são as distâncias Euclidianas entre cada *ground-truth* e keypoints preditos correspondentes e v_i são as flags de visibilidade do ponto *ground-truth*. As flags v_i preditas não são usadas. Para computar o OKS, passa-se d_i por uma gaussiana não normalizada com desvio padrão sk_i, onde s é a escala do objeto e k_i é uma constante por keypoint que controla a queda. Para cada keypoint é gerado um valor de similaridade entre 0 e 1. A média dessas similaridades é obtida sobre todos os keypoints cuja flag de visibilidade seja maior que zero ($v_i > 0$). Keypoints cuja flag de visibilidade seja 0 ($v_i = 0$) não influenciam na OKS. Predições perfeitas terão a $OKS = 1$ e predições nas quais todos os keypoints estão mais longe do que açguns desvios padrões sk_i terão $OKS \approx 0$. Dada a OKS, podemos calcular os valores AP assim como o IoU permite calcular essa métrica para detecção de objetos.

4.8.2. Stacked Hourglass Networks for Human Pose Estimation

Publicado em 2016, e estado-da-arte nesse ano, o artigo *Stacked Hourglass Networks for Human Pose Estimation*[Newell et al. 2016] se destaca por sua abordagem diferente em arquiteturas de redes convolucionais profundas. Através da sequenciação de vários módulos *hourglass* (Representado na Figura 4.32 (b)), os autores mostram que o principal diferencial de sua arquitetura está nas inferências sequenciadas no estilo "gargalo", e não meramente na profundidade, e portanto maior capacidade, da rede convolucional. O trabalho toma uma abordagem top-down, utilizando a rede *YOLO*, vista na Seção 4.7, para detectar e segmentar as imagens de cada pessoa em uma imagem e então executar a CNN *stacked hourglass*, para estimar os pontos de interesse de cada pessoa detectada.

O design do módulo *hourglass* foi motivado,segundo os autores, pela necessidade de capturar informação em cada escala. Enquanto evidencia local é essencial para identificar características como rostos e mãos, a predição final da pose requer um entendi-

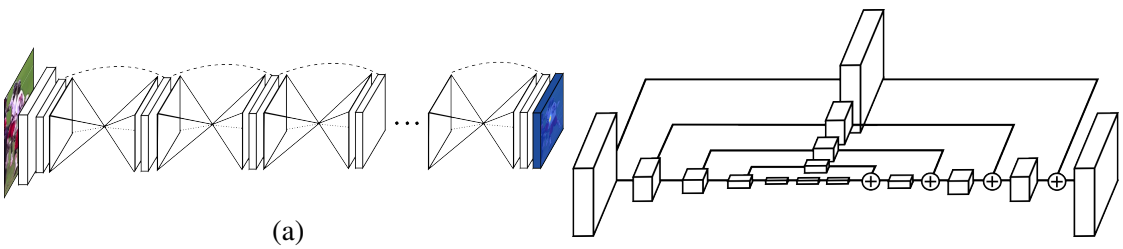
mento geral do corpo inteiro, por isso a necessidade da informação em todas as escalas. Com o objetivo de manter informação de características entre as escalas foram utilizadas *skip-layers* (Técnica de conexão entre escalas através de uma camada de convolução intermédiaria). A rede atinge a menor resolução em um tamanho de 4x4 pixels.

O módulo *hourglass* é definido da seguinte forma: Camadas convolucionais e *max pooling* são usadas para diminuir progressivamente a resolução da imagem de entrada até uma resolução muito baixa. A cada passo de *pooling* a rede ramifica e aplica mais convoluções à imagem antes do *pooling*. Após atingir a menor resolução, a rede começa uma sequencia de *upsamplings*(técnicas para aumento de resolução) e combinação de características entre escalas. Para unir informação entre duas resoluções adjacentes, os autores utilizaram o *upsampling* do vizinho mais próximo na menor resolução, como descrito em [Tompson et al. 2014], seguido de uma soma elemento-a-elemento dos dois tensores.

A topologia do modulo *hourglass* é simétrica para que cada camada de convolução e *max-pooling* tenha uma camada equivalente de *upsampling* para a combinação das características.

Ao fim do módulo *hourglass* duas convoluções 1x1 são aplicadas para se obter o conjunto de *heatmaps*. Na Figura 4.32 (b) é apresentada a organização do módulo *hourglass*. Já na Figura 4.32 (a), mostra uma visão geral de como funciona a arquitetura com os módulos hourglass empilhados. A entrada da CNN, uma imagem de dimensões 256x256, que é então reduzida para um tensor de tamanho 64x64, que é o tamanho do tensor de saída de cada módulo *hourglass*. Na saída da rede *stacked hourglass*, assim como na saída de cada módulo *hourglass*, temos um conjunto de *heatmaps*, de tamanho 64x64, um para cada *keypoint* a ser estimado. Segundo os autores, a explicação para a organização da arquitetura é a possibilidade de reconsiderar e/ou decidir quais *keypoints* ressaltar, em caso em que há, por exemplo, dois tornozelos de duas duas pessoas diferentes na mesma imagem.

A função de perda utilizada nessa arquitetura é a Mean-Squared Error (MSE), comparando o *heatmap* predito ao *heatmap ground-truth*, que consiste de uma gaussiana 2D (com desvio padrão de 1 pixel) centralizada na localização do *keypoint*.

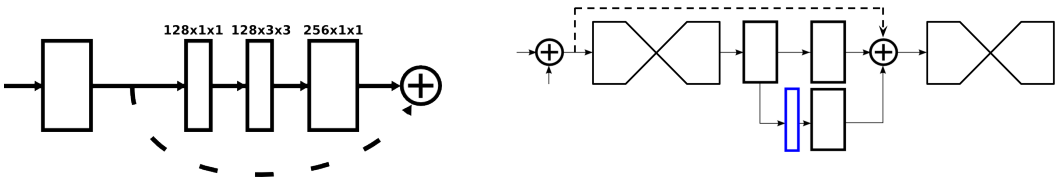


(a) (b)

Figura 4.32: (a) Organização da arquitetura *stacked hourglass*. (b) Estrutura de um módulo *hourglass*, Cada caixa corresponde a um módulo residual, como o visto na Figura 4.33 (a). Fonte: [Newell et al. 2016].

Os autores empregam também uma técnica chamada *supervisão intermediária*,

a qual consiste na geração de um grupo de *heatmaps* ao fim de cada módulo *houglass* (Destacado em azul na Figura 4.33 (b)), o que permite a aplicar uma função de *loss* entre cada conexão dos módulos *hourglass*. Esses *heatmaps* intermediários são convertidos de volta a *features* através de uma convolução 1x1 e então são somados aos *features* do módulo *hourglass* anterior e aos *features* de saída do *hourglass* atual. Uma representação gráfica pode ser consultada na Figura 4.33 (b), onde cada retângulo é um tensor, cada circulo com um simbolo de soma representa uma soma elemento-a-elemento entre tensores de mesmas dimensões, cada seta representa uma operação sobre os tensores anteriores, e por fim, cada seta tracejada representa uma conexão residual entre dois pontos. [He et al. 2016]



(a) (b)

Figura 4.33: Fonte: [Newell et al. 2016]

Na Figura 4.34 observa-se imagens geradas em diferentes etapas da predição no modelo *stacked hourglass*. Essas imagens podem ser usadas para representar os principais passos do processo de estimação do pose para qualquer abordagem top-down.

(a) (b) (c) (d)

Figura 4.34: Estágios da predição na arquitetura *stacked hourglass*. (a) Imagem original. (b) Detecção e segmentação de todas as pessoas na imagem. (c) Predição dos *heatmaps* (Aqui estão somados para aparecerem todos em uma só imagem). (d) Definição dos *keypoints* (na imagem também foram desenhados os segmentos, esses não são preditos, meramente ligam os *keypoints* preditos.) Fonte: Elaborado pelos autores.

No código[16] da Listagem 4.16 temos um exemplo para a geração das imagens da Figura 4.34, o código se baseia na classe *Inference*, que define diversas funções para a inferência com os pesos treinados da arquitetura *stacked hourglass*.

Listagem 4.16: Usando Stacked hourglass[Newell et al. 2016] para fazer estimação de pose em uma imagem.

[16]Disponível em https://github.com/pedropva/hourglasstensorlfow.git

```
import sys
sys.path.append('./') # Importar os outros arquivos na pasta
from inference import Inference
import cv2 # OpenCV, para trabalhar com imagens

# Caminho para o arquivo original
img_full_name = '0.jpg'
img_dir = './images/'
save_dir = './out/'
img_name, extension = img_full_name.split('.')
extension = '.' + extension
'''
 Instanciando um objeto da classe Inference
 Essa classe constroi o grafo de execucao dos modelos Stacked Hourglass e YOLO
 Tambem carrega os pesos treinados desses modelos.
'''
inf = Inference(config_file = 'config_tiny.cfg', model = 'hg_refined_tiny_200', yoloModel = 'YOLO_small.ckpt')

# Carregando uma imagem RGB
img = cv2.imread(img_dir+img_full_name)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Caso a imagem nao seja do tamanho 256x256, redimensionar.
if img.shape != (256,256,3):
    print('Wrong shape. Resizing.')
    img = cv2.resize(img, (256,256))

# Mostrando as dimensoes, deve ser (256X256X3) == (alturaXlarguraXnumero_de_canais_de_cor)
print('Img shape: ',img.shape)

'''
 Chamando a funcao que prediz a pose, onde os parametros sao:
 thresh : limiar de corte para o nivel de confianca para o keypoint
 pltJ : plotar keypoints (ou Joints)
 pltL : plotar segmentos (ou Limbs)
'''
new_img = inf.pltSkeleton(img, thresh = 0.5, pltJ = True, pltL = True)
new_img = cv2.cvtColor(new_img, cv2.COLOR_RGB2BGR)
cv2.imwrite(save_dir+img_name+'_prediction'+extension,new_img)

# Nessa funcao a predicao para na fase de çãdeteco de pessoas
bb = inf.pltBoundingBoxes(img)
bb = cv2.cvtColor(bb, cv2.COLOR_RGB2BGR)
cv2.imwrite(save_dir+img_name.split('.')[0]+'_bb'+extension,bb)

# Nessa funcao a predicao para quando temos os heatmaps, a funcao retorna eles
hm = inf.predictHM(img)
hm = cv2.cvtColor(hm, cv2.COLOR_RGB2BGR)
cv2.imwrite(save_dir+img_name.split('.')[0]+'_hm'+extension,hm)

print(f'Done! Check {save_dir} for the results!')
```

4.9. Considerações Finais

Este capítulo apresenta os fundamentos e tecnologias para desenvolver modelos de *Deep Learning* para analise de vídeo. O capítulo inicia com a explicação da instalação e uso básico do *framework* Tensorflow. Em seguida são apresentados os fundamentos de redes neurais e *Deep Learning*, focando principalmente em modelos do tipo CNN. Adicionalmente, o capítulo também aborda e evolução das técnicas de *Deep Learning*. Em especial, apresenta a evolução da rede InceptionNet, que é considerada um *milestone* da área e foi a vencedora do desafio ImageNet 2014. O capítulo é concluído com a apresentação de três projetos práticos, um de classificação de cenas de vídeo, reconhecimento de objetos e, por ultimo, estimação de pose

Nesse contexto, acreditamos que a proposta do capítulo é interessante para estudantes, pesquisadores e profissionais de TI. Em particular, esperamos que o leitor esteja apto para usar *Deep Learning* para desenvolver aplicações que envolvam tarefas como classificação de de cenas de vídeo, detecção de objetos e estimação de pose.

Referências

[Andriluka et al. 2014] Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. (2014). 2d human pose estimation: New benchmark and state of the art analysis. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pages 3686–3693.

[Arandjelovic et al. 2016] Arandjelovic, R., Gronat, P., Torii, A., Pajdla, T., and Sivic, J. (2016). Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5297–5307.

[Cao et al. 2018] Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., and Sheikh, Y. (2018). OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*.

[Deng et al. 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE.

[Everingham et al. 2010] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.

[Gemmeke et al. 2017] Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M., and Ritter, M. (2017). Audio set: An ontology and human-labeled dataset for audio events. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 776–780. IEEE.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hershey et al. 2017] Hershey, S., Chaudhuri, S., Ellis, D. P. W., Gemmeke, J. F., Jansen, A., Moore, C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., Slaney, M., Weiss, R., and Wilson, K. (2017). Cnn architectures for large-scale audio classification. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[Hochreiter and Schmidhuber 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8).

[Insafutdinov et al. 2016] Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M., and Schiele, B. (2016). Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In *European Conference on Computer Vision*, pages 34–50. Springer.

[Iqbal et al. 2017] Iqbal, U., Milan, A., and Gall, J. (2017). Posetrack: Joint multi-person pose estimation and tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2011–2020.

[Lin et al. 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

[Milan et al. 2016] Milan, A., Leal-Taixé, L., Reid, I., Roth, S., and Schindler, K. (2016). Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*.

[Newell et al. 2016] Newell, A., Yang, K., and Deng, J. (2016). Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*. Springer.

[Oord et al. 2016] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

[Redmon et al. 2016] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

[Redmon and Farhadi 2018] Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

[Rosenblatt 1957] Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

[Ruggero Ronchi and Perona 2017] Ruggero Ronchi, M. and Perona, P. (2017). Benchmarking and error diagnosis in multi-instance pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 369–378.

[Shotton et al. 2011] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304. Ieee.

[Stiefelhagen et al. 2006] Stiefelhagen, R., Bernardin, K., Bowers, R., Garofolo, J., Mostefa, D., and Soundararajan, P. (2006). The clear 2006 evaluation. In *International evaluation workshop on classification of events, activities and relationships*, pages 1–44. Springer.

[Szegedy et al. 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12.

[Szegedy et al. 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Szegedy et al. 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[Tompson et al. 2014] Tompson, J. J., Jain, A., LeCun, Y., and Bregler, C. (2014). Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in neural information processing systems*, pages 1799–1807.

[Tremblay et al. 2018] Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., and Birchfield, S. (2018). Deep object pose estimation for semantic robotic grasping of household objects. *arXiv preprint arXiv:1809.10790*.

[Wang et al. 2010] Wang, S., Ai, H., Yamashita, T., and Lao, S. (2010). Combined top-down/bottom-up human articulated pose estimation using adaboost learning. In *2010 20th International Conference on Pattern Recognition*, pages 3670–3673. IEEE.

[Yang and Ramanan 2012] Yang, Y. and Ramanan, D. (2012). Articulated human detection with flexible mixtures of parts. *IEEE transactions on pattern analysis and machine intelligence*, 35(12):2878–2890.

BIO

Gabriel Pereira dos Santos is an assistant researcher working under the guidance of Prof. Sergio Colcher at Pontifical Catholic University of Rio de Janeiro (PUC-Rio). He is graduating on Electrical Engineering at WYDEN Univerity on Brazil. His research interests are in multimedia systems and artificial intelligence, working mainly on the following topics: Video Classification and Economic Indicators Prediction. Currently, He is working on the VideoMR project, one of the selected projects in the Microsoft and RNP A.I. challenge, which aims to detect improper content in videos. Lattes CV: `http://lattes.cnpq.br/8088572506239597`.

Pedro Vinicius Almeida de Freitas is a MSc. candidate working under the guidance of Prof. Sergio Colcher at Pontifical Catholic University of Rio de Janeiro (PUC-Rio). He received a B.S. (2018) in Computer Science from the Federal University of Maranhão (UFMA) on Brazil. He is a assistant researcher at Telemidia Lab – PUC-Rio. His research interests are in multimedia systems and artificial intelligence, working mainly on the following topics: Video classification, Economic indicators prediction and Pose estimation. Currently, He is working on the VideoMR project, one of the selected projects in the Microsoft and RNP A.I. challenge, which aims to detect improper content in videos. Lattes CV: `http://lattes.cnpq.br/7566765486024024`.

Antonio Busson is Ph.D. candidate working under the guidance of Prof. Sergio Colcher at Pontifical Catholic University of Rio de Janeiro (PUC-Rio). He received a B.S. (2013) and M.S. (2015) in Computer Science from the Federal University of Maranhão (UFMA) on Brazil. He is researcher member at Telemidia Lab – PUC-Rio and research collaborator at the Laboratory of Advanced Web Systems (LAWS) – UFMA. His research interests are in multimedia systems, working mainly on the following topics: Coding and Processing of multimedia data; Hypermedia Document models, Pattern Recognition, and applications such as Web, iDTV and Games. Currently, He is working on the VideoMR project, one of the selected projects in the Microsoft and RNP A.I. challenge, which aims to detect improper content in videos. Lattes CV: `http://lattes.cnpq.br/1857348479447184`.

Álan Guedes holds a Ph.D. (2017) from the PUC-Rio, where he acts as post-Phd Researcher in TeleMídia Lab. He received his Bachelor (2009) and M.Sc. (2012) degrees in Computer Science from the UFPB, where he worked as researcher in Lavid Lab. In both labs, Álan worked in interactive video and TV research projects, and contribute to the Ginga and N CL specifications, which today are standards for DTV, IPTV and IBB. His research interests include Interactive Multimedia, Immersive Media, and Deep Learning for Multimedia. Lattes: `http://lattes.cnpq.br/1481576313942910`.

Ruy Luiz Milidiú received B.S. (1974) in Mathematics and M.S. (1978) in Applied Mathematics from Federal University of Rio de Janeiro. He also received a M.S. (1983) and Ph.D. (1985) in Operations Research

from University of California. Currently, he teaches in Informatics Department at Pontifical Catholic University of Rio de Janeiro (PUC-Rio). He has experience in Computer Science, focusing on Algorithmics, Machine Learning and Computational Complexity. Lattes CV: `http://lattes.cnpq.br/6918010504362643`.

Sérgio Colcher received B.S. (1991) in Computer Engineer, M.S. (1993) in Computer Science and Ph.D. (1999) in Informatics, all by PUC-Rio, in addition to the postdoctoral (2003) at ISIMA (Institute Supérieur d'Informatique et de Modelisation des Applications - Université Blaise Pascal, Clermont Ferrand, France). Currently, he teaches in Informatics Department at Pontifical Catholic University of Rio de Janeiro (PUC-Rio). His research interests include computer networks, analysis of performance of computer systems, multimedia/hypermedia systems and Digital TV. Lattes CV: `http://lattes.cnpq.br/1104157433492666`.

Capítulo 5

Recuperação de Informação Multimídia em Big Data Utilizando OpenCV Python

Rudinei Goularte[1], Tiago Henrique Trojahn[2] e Rodrigo Mitsuo Kishi[3]

[1] Universidade de São Paulo. Instituto de Ciências Matemáticas e de Computação.

[2] Instituto Federal de São Paulo

[3] Universidade Federal de Mato Grosso do Sul.

rudinei@icmc.usp.br, tiagotrojahn@ifsp.edu.br, rodrigo.kishi@ufms.br

Abstract

The popularization of systems, applications and devices to produce, view and share multimedia, saw the need to treat a large volume of data arise. In related areas (such as Multimedia Big Data, Data Science and Multimedia Information Retrieval) a key step is commonly referred as Multimedia Indexing or Multimedia Big Data Analysis, where the aim is to represent multimedia content into smaller, more manageable units, allowing the extraction of data features and information essential to the proper performance of the associated services. This mini-course discusses current tools and techniques for indexing, extracting and processing of multimodal multimedia content. The techniques are exemplified in Python OpenCV over different content (like images, audio, text and video), leading to the interest of services like Netflix, Google and YouTube on this subject, attracting the interest of researchers and developers.

Resumo

A popularização de sistemas, aplicativos e dispositivos para produzir, exibir e compartilhar conteúdo multimídia fez surgir a necessidade de tratar um grande volume de dados. Nas áreas relacionadas (como Multimedia Big Data, Ciência de Dados e Recuperação de Informação Multimídia) um pré-requisito chave é comumente conhecido como Indexação Multimídia (ou Análise Multimídia em Big Data), onde o objetivo é representar o conteúdo em unidades menores e mais gerenciáveis, permitindo a extração de features dos dados e informações essenciais para o bom funcionamento dos serviços associados. Este minicurso aborda ferramentas e técnicas atuais para indexação, extração e processamento de conteúdo multimídia multimodal. As técnicas

são exemplificadas em OpenCV Python sobre diferentes conteúdos (imagens, áudio, texto e vídeo), levando ao interesse de serviços como Netflix, Google e YouTube nesse assunto, motivando pesquisadores e desenvolvedores.

5.1. Introdução

Nos últimos anos houve um aumento na quantidade de conteúdo multimídia disponível para acesso [Lu et al., 2011; Pouyanfar et al., 2018]. Isso, em parte, pode ser explicado pela proliferação de dispositivos de baixo custo para capturá-los e codificá-los. Além disso, o avanço da tecnologia possibilitou o surgimento de uma grande variedade de meios que permitem ao usuário o uso de informações multimídia em qualquer lugar e a qualquer momento. Atualmente, as pessoas podem ter acesso a conteúdos usando diferentes tipos de dispositivos e meios (*notebooks*, celulares, *Personal Digital Assistants*, WiFi, 3G e 4G, entre outros). Toda essa evolução criou ambientes heterogêneos [Bouyakoub e Belkhir, 2008; Baraldi et al., 2017; Pouyanfar et al., 2018], surgindo com isso desafios no tratamento dos dados, já que, geralmente, quando um dispositivo acessa um conteúdo multimídia para o qual não foi projetado, a experiência do usuário é insatisfatória.

Além disso, a era digital trouxe outra importante característica: a interação do usuário com o conteúdo. Com os avanços da Web, as pessoas podem interativamente escolher diferentes caminhos de navegação, explorando variadas informações disponíveis, inclusive multimídia. Exemplos desse tipo de serviço são YouTube[1], Netflix[2] e Last.fm[3]. O avanço desses serviços fez com que, atualmente, usuários passassem a não somente acessar, mas, também, ativamente produzir conteúdo. Isso faz com que o volume de dados produzidos cresça contínua e rapidamente, agravando o problema da sobrecarga de informação: recuperar conteúdo de interesse em meio ao imenso volume de informações disponíveis [Hu et al., 2011; Toffler, 1984].

Tal problema vem sendo tratado por importantes áreas da Computação, como Recuperação de Informação [Baeza-Yates e Ribeiro-Neto, 2008], Recuperação de Informação Multimídia [Blanken et al., 2010] Recomendação e Personalização e Adaptação Multimídia [Lu et al., 2011; Pouyanfar et al., 2018] e, mais recentemente, por Ciência de Dados e *Multimedia Big Data Analisys* [Pouyanfar et al., 2018]. Um fator comum é que os sistemas desenvolvidos nessas áreas necessitam que dados (*features)* sejam extraídos para representar o conteúdo de modo mais compacto e, também, servirem de alicerce para os serviços das áreas supracitadas. Entretanto, o processo de extração, comumente chamado de Indexação Multimídia, é complexo e envolve alto custo computacional [Blanken et al., 2010].

Por exemplo, é comum que vídeos sejam descritos por meio de suas características visuais e que para obtê-las sejam utilizadas técnicas de extração de características de imagens com relativo alto custo de processamento [Iwan e Thom, 2017]. Contudo, vídeo também possui características sonoras e textuais, que, juntamente com as visuais podem representar mais fielmente a informação sendo tratada. Essa faceta inerentemente multimodal de dados multimídia faz surgir a necessidade de se desenvolver métodos também multimodais de indexação, de modo a se obter uma única

[1] www.youtube.com

[2] www.netflix.com

[3] www.lastfm.com

representação (um único vetor de características) contendo informações de todas as modalidades (visual, sonora e textual) [Xie et al., 2017].

Os métodos consagrados na literatura para indexação multimídia são majoritariamente unimodais, permitindo extrair e representar uma única característica em um vetor de características. Isso é feito por meio de extratores de características bem conhecidos (como SIFT para imagens e MFCC para áudio), utilizando medidas de dissimilaridade [Blanken et al., 2010] entre os vetores de características resultantes para aferir pertinência, desigualdade e outras operações sobre os objetos/dados. Os métodos mais recentes, multimodais, empregam conceitos mais complexos, onde tanto a dissimilaridade obtida analisando-se os dados (*features*) brutos quanto a semântica latente entre os objetos analisados contribuem para o resultado final, melhorando a eficácia das tarefas e serviços [Pouyanfar et al., 2018]. Entre tais conceitos destacam-se a Fusão de Modalidades e os Dicionários de Palavras Multimodais (*Bag of Multimodal Words*), que vêm auxiliando a obter melhores resultados em diversas áreas relacionadas a Recuperação de Informação Multimídia (RIM) [Del Fabro e Boszormenyi, 2013; Pouyanfar et al., 2018].

5.2. OpenCV e Conceitos Básicos

Nesta seção, são discutidos ambos, a biblioteca *OpenCV*, utilizada como suporte ao minicurso, assim como alguns dos principais conceitos relacionados a recuperação de informação multimídia. A Seção 5.2.1 descreve a biblioteca OpenCV, composta de diferentes funcionalidades que auxiliam o desenvolvimento de aplicações voltadas a visão computacional e também utilizadas neste trabalho. Já a Seção 5.2.2 apresenta os principais conceitos relacionados.

5.2.1. A Biblioteca OpenCV

A biblioteca OpenCV[4] tem como objetivo incentivar o desenvolvimento de aplicações complexas voltadas para visão computacional em geral. Foi desenvolvida inicialmente pela Intel® em meados do ano 2000, sendo mantida atualmente pela *OpenCV Foundation*.

OpenCV é de código-aberto (licença BSD) com suporte a linguagens como C, C++, Java e Python. Além do suporte oficial, diversas outras bibliotecas, desenvolvidas por terceiros, podem ser utilizadas para estender a OpenCV a outras linguagens, como Ruby[5] e C#[6]. Além da variedade de linguagens de programação, a biblioteca também pode ser utilizada em diversas plataformas e sistemas operacionais, tais como Microsoft® Windows™, FreeBSD, OpenBSD, Linux, OS X™, Android™ e iOS™.

Um dos grandes destaques da biblioteca é a extensa documentação oficial, principalmente para as linguagens Python e C/C++, com a apresentação detalhadas das funcionalidades, tutoriais para diversas tarefas comuns e exemplos distribuídos juntamente com a biblioteca. Isso, aliado ao grande número de desenvolvedores e amplo suporte da comunidade tornam a OpenCV uma ferramenta bastante útil em diversos trabalhos na área.

[4] https://opencv.org/
[5] https://github.com/ruby-opencv/ruby-opencv
[6] http://www.emgu.com/wiki/index.php/Main_Page

Nesse sentido, a documentação da biblioteca conta com a descrição detalhada das classes e funções dos diferentes módulos que compõem a OpenCV. Além disso, está disponível um amplo conjunto de tutoriais detalhados que apresentam diversos recursos da biblioteca, como a extração de histogramas, erosão e dilatação, derivadas de Sobel, *template matching*, entre outros.

Neste texto utiliza-se a versão estável 3.4 da OpenCV. Para sua instalação recomenda-se instalar antes o utilitário pip do Python. Exemplo: em ambiente Mac OS X:

>> easy_install pip

>> pip install –r setup.txt

Onde *setup.txt* é um arquivo texto contendo os nomes das bibliotecas a serem instaladas (uma por linha e sem vírgula), incluindo a OpenCV: numpy, scipy, opencv-contrib-python==3.4.0.12, python_speech_features, moviepy.

A Listagem 5.1 apresenta um exemplo simples de uso da biblioteca OpenCV via um script Python. Na linha 1 tem-se a importação da biblioteca.

```
1  import cv2
2
3  # Abre e carrega a imagem de entrada
4  imagem = cv2.imread("imagem.jpg")
5  # Converte a imagem para a escala de cinza
6  escala_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
7  # Espelha a imagem (rotaciona) verticalmente
8  rotacionada = cv2.flip(escala_cinza, +1)
9  # Salva a imagem rotacionada como 'saida.jpg'
10 cv2.imwrite("saida.jpg", rotacionada)
```

Listagem 5.1. Exemplo de código Python utilizando OpenCV. Arquivo exemplo1.py

Na Listagem 5.1, os métodos *imread*, *cvtColor*, *flip* e *imwrite* pertencem à biblioteca OpenCV e realizam, pela ordem, a leitura de uma imagem para a memória, a conversão da imagem para escala de cinzas, o espelhamento da imagem e a escrita da imagem processada em arquivo. A documentação dos métodos assim como tutoriais da OpenCV podem ser encontrados em https://opencv-python-tutroals.readthedocs.io/en/latest/**.** Para executar o código da Listagem 5.1 basta salvar o arquivo texto correspondente (exemplo1.py) em um diretório e, via linha de comando/shell, digitar:

>> python exemplo1.py

O resultado será um arquivo, chamado *saída.jpg*. Um exemplo de imagem de entrada e correspondente saída para o código da Listagem 5.1 pode ser visualizado na Figura 5.1.

Figura 5.1. Imagem original (esquerda) e processada (direita).

5.2.2. Conceitos Relacionados

O presente minicurso se insere na área conhecida como Sistemas Multimídia, ou simplesmente Multimídia. O termo **Multimídia**, em Computação, pode ser definido como "a utilização simultânea de dois ou mais tipos diferentes de mídia, integradas e entregues por computador" [Havaldar e Medioni, 2009]. **Mídia**, nesse caso, pode ser entendida como "o meio pelo qual a informação é representada, apresentada, transmitida ou armazenada por sistemas computacionais" [Mandal, 2002].

A informação apresentada pelo computador é percebida pelos seres humanos quando estes utilizam seus canais sensoriais. Assim, uma informação percebida via o canal auditivo é dita ser da modalidade aural, do mesmo modo que uma informação percebida pelo canal visual é dita ser da modalidade visual. Logo, **modalidade** se refere à relação entre a mídia utilizada para comunicar a informação e o canal sensorial adequado para receber tal informação. **Multimodalidade**, por consequência, é a utilização conjunta de duas ou mais modalidades diferentes que, em Computação, tem como objetivo melhorar o desempenho em tarefas computacionais relacionadas ao processamento, representação ou recuperação de informações.

Uma área recente da Computação que faz uso intenso de multimodalidade é a **Recuperação de Informação Multimídia**. O objetivo desta área é recuperar dados textuais, sonoros, de imagens e de vídeo relacionados aos interesses do usuário e classificá-los de acordo com o grau de relevância relacionado à consulta do usuário [Blanken et al., 2010]. Atualmente o problema fundamental está em como habilitar ou melhorar a recuperação multimídia utilizando métodos baseados em conteúdo (não baseados em texto). Métodos baseados em conteúdo são necessários quando anotações textuais são inexistentes ou incompletas. Além disso, métodos baseados em conteúdo potencialmente podem melhorar a acurácia da recuperação quando anotações textuais estão presentes fornecendo entendimento adicional sobre as coleções de mídias [Blanken et al., 2010; Pouyanfar et al., 2018].

5.3. Extração de características

Um sistema de recuperação de informação multimídia deve ser capaz de indexar dados de sua coleção pelos seus respectivos conteúdos. Graças ao grande volume de dados e à complexidade das informações, a indexação é realizada através de alguma propriedade discriminatória, chamada **característica** [Atrey *et al.*, 2010], termo advindo da língua

inglesa - ***feature***. As características comumente se dividem três categorias, de acordo com seu nível de abstração [Martinet e Sayad, 2012]: baixo, médio e alto nível semântico. Características de baixo nível semântico são leituras numéricas computadas diretamente dos dados brutos, sem que haja o emprego de conhecimento externo, aprendizado de máquina ou análise estatística de outros documentos. Exemplos de características de baixo nível semântico são informações de cor de uma imagem ou de intensidade de uma amostra de áudio.

Segundo Martinet e Sayad (2012), as características de médio nível semântico são definidas como leituras numéricas ou simbólicas produzidas por uma etapa de refinamento semântico em características de baixo nível semântico. Um exemplo de característica de médio nível semântico é o *Bag-of-Features* (BoF), que é uma extensão do modelo *Bag-of-Words* (BoW) da área de Processamento de Linguagem Natural. Já as características de alto nível semântico são informações sobre o significado do conteúdo, diretamente interpretáveis por humanos. Elas se apresentam, costumeiramente, através de texto descrevendo o conteúdo. Uma imagem, por exemplo, pode ter a ela associados os rótulos "carro" e "moto", indicando que tal imagem ilustra um carro e uma moto. Estes rótulos podem também representar conceitos não materiais como, por exemplo, "raiva" ou "tranquilidade". As características de alto nível semântico podem ser geradas por anotação manual ou automaticamente por análise computacional do conteúdo (a partir de características de baixo/médio nível semântico), sujeita à lacuna semântica [Smeulders et al., 2000].

As características de baixo e de médio nível semântico extraídas de conteúdo multimídia são codificadas em vetores numéricos n-dimensionais, chamados **vetores de características** [Martinet e Sayad, 2012]. Os vetores de características, representações obtidas através de processamento e análise computacional, servem para realizar comparação entre vídeos e se caracterizam como representações compactas e relevantes do conteúdo. O processo de indexar automaticamente uma base de vídeos por suas características é chamado **extração de características** [Blanken et al., 2010]. A uma ferramenta que computa um vetor de características de um canal de informação é dado o nome de detector ou extrator de características [Tuytelaars e Mikolajczyk, 2008].

A extração de características de alto nível semântico está fora do escopo deste texto. Assume-se que tais características, na forma de textos descritivos de conteúdo ou rótulos de conteúdo, podem estar presentes e podem ser utilizados como entrada para um sistema de RIM.

5.3.1. Características de Baixo Nível Semântico

Características Visuais

Características visuais de baixo nível semântico são extraídas de imagens ou de quadros de vídeo, que em última análise são também imagens. Características de imagens estáticas se dividem em globais e locais, de acordo com seu escopo de representação. As características globais se referem à imagem inteira e normalmente são compactas. Uma das mais populares características globais de comparação de conteúdo visual é o histograma global de cor [Stockman e Shapiro, 2001].

Um histograma de cor de uma imagem é uma contagem dos pixels, por cor, de todos os pixels dessa imagem. Ele pode ser implementado como um vetor

unidimensional onde cada índice corresponde a um intervalo de cores e o conteúdo de cada célula corresponde ao total de pixels da imagem que se encaixam no intervalo relacionado àquele índice. A obtenção de histogramas de cor tem baixo custo computacional e eles são relativamente invariantes à diferenças de translação, rotação centralizada no eixo de captura, pequenas rotações não centralizadas no eixo de captura, mudanças de escala e oclusão parcial. Como os histogramas de cor desconsideram informação espacial dos pixels, eles falham quando precisam diferenciar imagens muito diferentes com cores semelhantes. Além da indiferença à informação espacial, histogramas de cor falham ao comparar duas imagens com conteúdo parecido, mas que foram capturadas em condições de iluminação muito diferentes [Lu et al. 2001].

Uma característica local, diferentemente de uma global, se refere a um subconjunto de uma imagem. Este subconjunto pode ser um ponto, região ou mesmo um segmento de aresta contendo um padrão que o diferencia de sua vizinhança [Tuytelaars e Mikolajczyk, 2008]. Espera-se, com a extração de características locais, representar estruturas distintivas de uma imagem. De acordo com Grauman e Leibe (2011), um bom extrator de características locais deve garantir repetibilidade e precisão, extraindo sempre as mesmas características de duas imagens distintas que exibem o mesmo objeto e, ao mesmo tempo, oferecer distintividade, a habilidade de distinguir dois objetos diferentes. Geralmente tais métodos operam seguindo uma série de passos: detectando regiões distintivas na imagem, chamadas de **pontos de interesse**, normalmente relativas a arestas, cantos e junções T; realizando operações com os valores ao redor de tal ponto para torná-lo robusto a variações na escala, rotação e iluminação; transformando esses valores em um vetor de características.

Existem hoje diversos extratores locais populares, como SURF (*Speeded up Robust Features*), SIFT (*Scale-Invariant Feature Transform*), FAST (*Features from Accelerated Segment Test*), BRIEF (*Binary Robust Independent Elementary Features*) e ORB (*Oriented FAST and Rotated BRIEF*). Os dois primeiros estão patenteados, sendo seu uso livre para propósitos acadêmicos. O último é uma alternativa aberta suportada pela biblioteca OpenCV. Detalhes sobre esses e outros métodos podem ser encontrados no trabalho de Leng et al. (2019). Neste texto, a título de ilustração, iremos abordar em um pouco mais de detalhes o método SIFT.

O método de extração de características locais SIFT, proposto por Lowe (2004), é um dos mais conhecidos atualmente. O método SIFT foi apresentado como uma combinação do detector de regiões de interesse por diferenças de gaussianas, do inglês *Difference-of-Gaussians* (DoG), e de uma técnica para produzir um vetor de características. As características extraídas pelo método SIFT são invariantes à escala e à rotação.

O detector de regiões de interesse baseia-se no fato de que pontos de máximo de laplacianas de gaussianas, do inglês *Laplacian-of-Gaussians* (LoG), são equivalentes à arestas e/ou cantos de uma imagem. O cálculo de LoG em uma imagem é computacionalmente custoso e, por isso, o SIFT usa a DoG como uma aproximação para a LoG. A DoG é basicamente uma subtração envolvendo uma imagem e uma versão desfocada dela mesma. Para cada imagem, o SIFT computa três cópias redimensionadas sucessivamente, onde cada cópia tem dimensões correspondentes à um quarto da cópia anterior. As diferentes escalas são chamadas oitavas e sua obtenção compõe a estratégia responsável pela invariância à escala. Para cada oitava são

computadas quatro cópias suas com desfoque gaussiano, aumentado sequencialmente em cada cópia. A DoG é, então, computada entre as imagens de cada oitava e a imagem seguinte com maior desfoque da mesma oitava. Os pontos aproximados (pontos-chave) de máximo e mínimo são encontrados, realizando uma busca em vizinhanças 3×3×3 na matriz tridimensional resultante da combinação das matrizes bidimensionais DoG.

A quantidade de pontos chave produzida é grande e muitos deles não correspondem a regiões interessantes. Por este motivo, são realizadas duas etapas de filtragem, rejeitando pontos com baixo contraste e depois aqueles com alta resposta de aresta em uma única direção usando determinante do hessiano, do inglês *Determinant of Hessian* (DoH). Depois das etapas de filtragem, inicia-se a construção do vetor de características, ao se determinar a orientação mais proeminente dos gradientes de direção em torno de cada ponto-chave. Essa orientação é atribuída ao ponto chave e todo cálculo posterior será relativo a ele, permitindo a invariância à rotação. A orientação é obtida através de um histograma de 36 posições, onde cada posição cobre 10 graus e tem como valor a soma das magnitudes dos gradientes. O maior pico, assim como os outros picos com valor de pelo menos 80% dele, são transformados em pontos chave individuais onde a orientação é dada pelo valor da posição daquele pico no histograma.

O último passo do método SIFT consiste em gerar uma assinatura para cada ponto chave, que é o próprio vetor de características. O vetor de características é obtido pela inspeção de uma janela 16×16 em volta do ponto chave. Essa janela é dividida em dezesseis janelas 4×4 e cada janela produz um histograma de 8 posições de magnitudes de gradientes, onde cada posição do histograma cobre 45 graus. O conteúdo adicionado a cada posição do histograma também é proporcional à distancia do ponto chave. Fazendo isso para cada uma das dezesseis regiões, um vetor de características de 16×8 = 128 bytes é produzido. Esse vetor de características é normalizado, dele é subtraída a orientação do ponto chave e por fim é limiarizado para 0:2. A subtração da orientação do ponto chave serve para prover independência à rotação. Já a limiarização provê independência à diferenças de iluminação. Um exemplo de extração de pontos chave pode ser visto na Figura 5.2. A imagem à esquerda corresponde à imagem original. A imagem à direita corresponde à imagem processada, indicando-se por meio de círculos coloridos os pontos-chave (*keypoints*) detectados. A literatura da área refere-se a esse processo como detecção de pontos-chave. Embora a documentação da OpenCV refira-se ao processo como extração de pontos-chave, são o mesmo processo.

Figura 5.2. Exemplo de Extração de Pontos-chave.

```
1  import numpy as np
2  import cv2
3
4  # Carrega a imagem de entrada
5  imagem = cv2.imread('pisa.jpg')
6
7  # Converte a imagem para a escala de cinza
8  escala_cinza = cv2.cvtColor(imagem, cv2.COLOR_BGR2GRAY)
9
10 # Cria o extrator de caracteristicas SIFT
11 sift = cv2.xfeatures2d.SIFT_create()
12
13 # Detecta os pontos-chave da imagem
14 keypoints, descritores = sift.detectAndCompute(escala_cinza, None)
15
16 # Desenha os pontos-chave como circulos na imagem 'escala_cinza'
17  imagem_saida = cv2.drawKeypoints(escala_cinza, keypoints, imagem)
18  # Salva a imagem com os pontos-chave como 'pontos_chave_SIFT.jpg
19   cv2.imwrite('pontos_chave_SIFT.jpg', imagem_saida)
```

Listagem 5.2. Exemplo de código para extração de pontos-chave.

Um exemplo de código em OpenCV Python para detecção/extração de pontos-chave pode ser encontrado na Listagem 5.2. Nesse exemplo utilizou-se o detector fornecido pela implementação OpenCV do extrator de características SIFT. Percebe-se, na linha 14, que o método *detectAndCompute* devolve tanto os pontos-chave (*keypoints*) quanto os vetores de características SIFT (*descritores*).

Características Aurais

As características aurais (sonoras), de maneira análoga às características visuais, representam propriedades específicas do sinal de áudio em um formato compacto, possibilitando a comparação com outros sinais de áudio. Características como momentos de silêncio, amplitude do sinal e frequência são bastante comuns e triviais de serem obtidas (com o auxílio de ferramentas como SoundForge[7] e Audacity[8]) e armazenadas em um vetor de características. Apesar de úteis, tais características isoladamente limitam a realização de tarefas importantes como detecção e reconhecimento de fala e similares com maior grau de complexidade. Para tais tarefas, características mais robustas tem sido empregadas, como *Linear Predictive Coding* (LPC), *Linear Predictive Cepstral Coefficients* (LPCC), *Linear Frequency Cepstral Coefficients* (LFCCs) and *Exponential Frequency Cepstral Coefficients* (EFCCs) [Vestman et al., 2018].

Um dos mais famosos extratores de características aurais, o *Mel Frequency Cepstral Coefficients* (MFCC), foi introduzido por Davis e Mermelstein (1980) e era destinado à tarefa de reconhecimento de palavras monossilábicas. Mais tarde, o MFCC se mostrou aplicável em tarefas como o reconhecimento de locutores, integrando o

[7] https://www.magix.com/br/musica/sound-forge/

[8] https://www.audacityteam.org/

estado da arte em análise de áudio atual [Yang et al., 2019]. O MFCC foi desenvolvido para mimetizar o comportamento da percepção de fala humana.

A frequência percebida por um humano é diferente do que é emitido pela fonte. A mimetização é realizada pelo uso da Escala Mel [Stevens et al., 1973], uma função da frequência para valores mel, que são unidades obtidas pela percepção por ouvintes de frequências consideradas como iguais, em distância, uma da outra. A extração de características MFCC é iniciada pela divisão do sinal de áudio em quadros. O sinal nestes quadros é convertido para o domínio de frequências através de uma transformada discreta de Fourier e um espectro de frequências mel é computado sobre estas frequências. O espectro de frequências mel é submetido a uma função logarítmica, para imitar a percepção humana de volume, também logarítmica. O próximo passo consiste na remoção de características dependentes do locutor por meio do cálculo de coeficientes cepstrais. Este passo se baseia no fato de que a fala humana pode ser modelada por um modelo fonte-filtro que descreve a produção de fala como um sinal e um filtro. O cepstro tem como objetivo suprimir o sinal fonte, mantendo apenas o sinal de filtro. Ele é obtido pela aplicação de uma transformada discreta de cosseno no sinal, mantendo os coeficientes de ordem baixa, que apresentam a resposta de frequência do trato vocal. Os coeficientes de ordem alta que apresentam a resposta de frequência do sinal de origem são descartados. Os vetores MFCC típicos mantém 13 coeficientes cepstrais.

```
import numpy as np
from python_speech_features import mfcc
import scipy.io.wavfile as wav
# Abre o audio de entrada 'audio.wav', retornando seu sinal e o valor da taxa de frequencia em Hz
(sinal, taxa_frequencia) = wav.wave("audio.wav")
# Extraem descritores mfcc do audio de entrada.
# Cada descritor será gerado sobre 0.1s de áudio, usando uma janela deslizante com deslocamento de 0.05s. Os 13 primeiros ceptrais serão retornados
caracteristicas = mfcc(sinal, taxa_frequencia, winlen=0.1, winstep=0.05, numcep=13)

# Imprime as características encontradas no áudio de entrada
for c in caracteristicas:
   print(c)
```

Listagem 5.4. Exemplo de código para extração de característica aurais MFCC.

Apesar de amplamente utilizado, o MFCC possui limitações e é passível de melhora (Sharma e Ali, 2015). Uma das mais problemáticas é a falta de significado de seus valores, já que apenas os dois primeiros coeficientes possuem significado determinado. O primeiro é a média da energia de todas as bandas de frequência e o segundo é o balanço de componentes de baixa e alta frequência do quadro de sinal. Os outros 11 coeficientes contém detalhes do espectro. O MFCC também é bastante sensível à presença de ruído no sinal, enfrentando uma degradação de performance severa nesse tipo de situação. Outro problema é a presunção de que os coeficientes

cepstrais da frequência fundamental são mais baixos do que dos componentes de frequência da mensagem linguística, que para alguns locutores do sexo feminino é falsa.

A listagem 5.4 traz um exemplo de código para a extração de características MFCC. Nesse exemplo, do áudio de entrada (*audio.wav*) gera-se um sinal (método *wave*) e desse sinal serão extraídas as características MFCC (método *mfcc*).

5.3.2. Características de Médio Nível Semântico

Bag of Words

O *Bag of Words* (BoW) é um modelo que gera uma representação simplificada de um texto [Aggarwal e Zhai, 2012], representado por um vetor que contém a contagem das ocorrências de suas palavras de acordo com um dicionário. Com isso, dois documentos podem ser comparados por seus vetores BoW gerados com base em um dicionário comum, usando alguma medida de distância entre vetores. Formalmente, o modelo consiste em obter uma representação de um documento $D = (t_1,t_2,...,t_p)$ no qual p é o tamanho do vocabulário e ti é a relevância do termo ti no documento D [Salton e Buckley, 1988]. Cada termo no modelo BoW pode se referir a uma única palavra, a duas palavras (bigrama) ou até mesmo uma frase completa. Já a relevância do termo pode ser calculada de diversas formas, incluindo a frequência do termo (do inglês *term frequency* - *tf*) ou a frequência inversa do termo (do inglês *inverse term frequency* - *idf*) [Ko, 2015].

Considerando $f_{t,d}$ como a frequência do termo t no documento d, a frequência do termo (*tf*) pode ser obtida de diversas formas [Aggarwal e Zhai, 2012], como a definida a seguir:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{5.1}$$

Ao invés de contar o número de ocorrências de cada termo no documento, a frequência inversa do termo (*idf*) busca medir o número de documentos de um *corpus* em que o termo ocorre, separando palavras comuns e que aparecem na maioria dos documentos ou palavras raras que aparecem em poucos documentos. Considerando D o conjunto de documentos do corpus textual e $|D|$ o número de documentos em D e $|d \in D: t \in d|$ o número de documentos que possuem o termo t, a frequência inversa do termo $idf(t;D)$ pode ser calculada como:

$$idf(t,D) = \log \frac{|D|}{1+|d \in D: t \in d|} \tag{5.2}$$

Além do cálculo da frequência de um termo, um importante aspecto na aquisição do conhecimento por informações textuais não-estruturadas é a etapa de pré-processamento [Rezende et al., 2011], no qual os termos de entrada são tratados e padronizados, preservando as principais características do texto de entrada. Exemplos de abordagens de pré-processamento amplamente utilizadas são a remoção de palavras muito comuns, chamadas de *stop words*, padronização do texto para minúsculas, radiciação (do inglês *stemming*) e lematização (do inglês *lemmatization*), entre outros.

Nesse sentido, Uysal e Gunal (2014) reportam uma avaliação de diferentes técnicas de pré-processamento e seu impacto em diversos *corpus* textuais diferentes.

Bag of Features

Os primeiros trabalhos utilizando o método *Bag of Features* (BoF), análogo do BoW aplicado em outros tipos de informação, são da área de Visão Computacional e datam da década de 2000. Neles, os textos foram substituídos por imagens e as palavras por *textons*, vetores que representam texturas [Leung e Malik, 2001]. Mais tarde, Csurka et al. (2004) propuseram o *Bag-of-Keypoints*, também destinado à comparação de imagens, que utilizava centroides de grupos de pontos chave SIFT no lugar dos *textons*. Muitos autores utilizam o termo *Bag-of-Visual-Words* (BoVW) quando referenciam um BoF orientado a conteúdo visual, já que no método, os padrões distintivos representados pelos centroides são análogos às palavras de um texto. O modelo BoF também foi estendido para outras naturezas de dados como, por exemplo, dados aurais, formando um *Bag-of-Aural-Words* (BoAW) [Carletti et al., 2013].

Apesar de existirem variações em determinadas aplicações, é possível enquadrar os métodos BoF num arcabouço genérico para comparar dois ou mais documentos[9] de qualquer natureza. Uma descrição em alto nível deste arcabouço para comparação de dois documentos é apresentada no Algoritmo 1.

Algoritmo 1 – Arcabouço da técnica de comparação de documentos por BoF

COMPARA_BOF(a,b,k,d): recebe um par de documentos a e b, um tamanho de dicionário k e uma distância de similaridade entre vetores como entrada. Devolve um valor d_{bof} que corresponde à similaridade entre a e b.

1: Extraia múltiplos vetores de características representativos de a e de b, e armazene-os em conjuntos de vetores de características v_a e v_b, respectivamente;
2: $D \leftarrow v_a \cup v_b$
3: Agrupe os elementos do conjunto D em k grupos, obtendo um conjunto de grupos W;
4: Calcule, para cada grupo $w \in W$, um vetor de características que melhor represente os elementos de w. O conjunto resultante destes vetores é o dicionário *Dict*;
5: Calcule os histogramas h_a e h_b, calculando as ocorrências de v_a e de v_b, respectivamente, em *Dict*;
6: **devolva** $\delta(h_a,h_b)$.

As Linhas 1 e 2 do Algoritmo 1, descrevem a extração de características dos dois documentos e sua união em um conjunto único. A justificativa desta etapa é a criação de um conjunto que possua todas as características presentes em qualquer um dos documentos, para permitir a identificação de padrões comuns em seguida. O passo descrito na Linha 3 do algoritmo serve para encontrar padrões entre todos os vetores de características de a e de b. Espera-se que os vetores de características de documentos diferentes, mas que representam um padrão semelhante em ambos os documentos, sejam atribuídos a um mesmo grupo. A instrução presente na Linha 4 do algoritmo é

[9] "Documento" como sinônimo de objeto multimídia, que pode inclusive ser um documento textual, mas também pode ser uma imagem ou vídeo.

responsável por definir uma representação única para cada um dos padrões identificados no passo anterior. A determinação do vetor de características que melhor representa um elemento de um grupo normalmente se dá através do cálculo de um centróide ou de um medóide. O conjunto de representações únicas corresponde a um dicionário de padrões encontrados nos documentos. Na etapa descrita na Linha 5 é realizada uma contagem de quantos vetores de características, do total de um documento, são mais próximos a cada um dos elementos de *Dict*. O resultado dessa contagem é o histograma de ocorrência de padrões. Por fim, a Linha 6 contém o cálculo de distância entre os vetores de ocorrência de padrões. É esperado que, nesta etapa, documentos com padrões semelhantes sejam considerados semelhantes.

O Algoritmo 1 pode ser facilmente adaptado para aplicação em sistemas RIM, onde o objetivo é recuperar os documentos similares a um documento dado como entrada. Para isso, basta gerar o dicionário de uma base de documentos, por exemplo, uma base de imagens. Em seguida, deve-se gerar os histogramas representativos de cada documento da base. Esses histogramas formam então a base de dados para comparação. Ao receber um documento de entrada, o sistema usa o dicionário para gerar o histograma representativo deste documento, comparando o mesmo com todos os histogramas da base. Os histogramas considerados similares – acima de um determinado limiar para alguma medida de similaridade – são selecionados para indicar os documentos/imagens a serem devolvidosb3t4schin

A Seção 5.5 apresenta uma discussão e exemplos de como gerar *Bags-of-features* e dicionários de modalidades diferentes, fundindo as características para que sejam utilizadas em uma tarefa específica de RIM. A Fusão de Modalidades é discutida na Seção 5.4.

5.4. Fusão de Modalidades

O emprego de multimodalidade em um sistema exige a definição de algum tipo de estratégia para combinar a informação proveniente de diferentes fontes. A combinação da informação de diferentes modalidades é chamada fusão multimodal. De acordo com Atrey *et al.* (2010), a fusão multimodal é realizada, geralmente, em dois níveis. Um destes níveis é a fusão tardia, também conhecida como fusão no nível de decisão. O arcabouço de fusão tardia é apresentado na Figura 5.3, inspirada em Atrey *et al.* (2010).

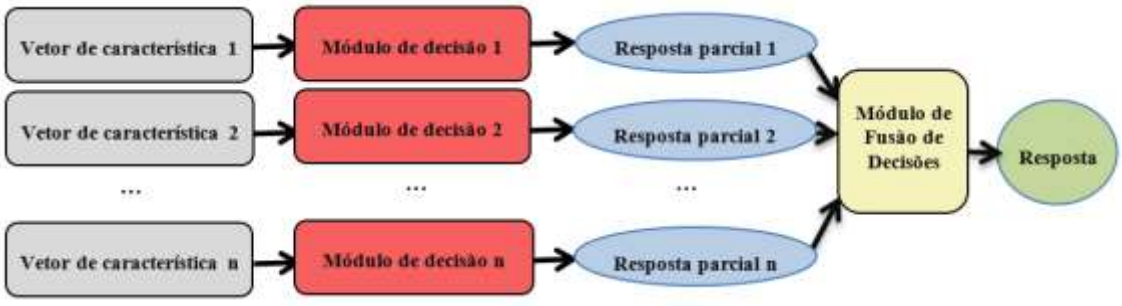


Figura 5.3. Fusão Tardia

Na fusão tardia, há uma etapa de decisão para cada um dos *n* vetores de características. Estas *n* decisões individuais são realizadas por *n* módulos de decisão, representados por retângulos vermelhos na Figura 5.3. As *n* etapas de decisão produzem *n* respostas parciais à tarefa, ilustradas na Figura 5.3 pelas elipses azuis. As respostas

parciais são então combinadas em uma resposta consensual por um módulo de fusão de decisões, representado na Figura 5.3 pelo retângulo amarelo.

Uma das vantagens da fusão tardia é a facilidade em combinar informações heterogêneas já que cada diferente representação proveniente de uma diferente modalidade dará origem a uma resposta parcial. Combinar as respostas parciais é relativamente simples considerando que todas as respostas têm a mesma representação. A fusão tardia também oferece relativa facilidade de inclusão/remoção de modalidades no processo pelo mesmo motivo. Outra vantagem da fusão tardia é a possibilidade de escolha de diferentes métodos de decisão para as diferentes modalidades. É possível, não somente escolher o método mais adequado para cada modalidade, como também parametrizá-lo para otimizar o aproveitamento da informação unimodal. A fusão tardia, no entanto, é incapaz de utilizar qualquer correlação existente entre as diferentes modalidades antes de uma etapa de decisão, podendo levar à perda de informação importante para esta etapa. Além disso, por requerer múltiplas etapas de decisão individuais, torna-se computacionalmente cara.

O segundo nível de fusão multimodal existente é a fusão prévia, também conhecida como fusão no nível de características. A fusão prévia é ilustrada na Figura 5.4, inspirada em Atrey et al. (2010).

Figura 5.4. Fusão Prévia

Em uma abordagem fusão prévia, há uma combinação de *n* vetores de características, produzindo um novo vetor de características que contém uma combinação da informação dos vetores de características de origem. Este vetor com informação combinada, chamado vetor de características fundidas, é ilustrado na Figura 5.4 por um retângulo azul. A ferramenta que efetua esta combinação é chamada módulo de fusão e é representada pelo retângulo de cor amarela da Figura 5.4. O vetor de características fundidas é então fornecido ao módulo de decisão, que produz a resposta esperada de acordo com a tarefa de vídeo. Em um sistema de segmentação temporal de vídeo, por exemplo, o módulo de decisão utiliza o vetor de características fundidas para encontrar as fronteiras entre os segmentos de vídeo. O módulo de decisão e a resposta produzida por ele são representados, respectivamente, pelo retângulo vermelho e pela elipse de cor verde na Figura 5.4.

A fusão prévia possibilita a identificação e o uso de correlação entre as diferentes características em uma etapa inicial, viabilizando a extração de melhores recursos para a execução da tarefa [Atrey *et al.*, 2010]. Outra vantagem da fusão prévia é a necessidade de uma única etapa de decisão que utiliza o vetor de características fundidas como entrada, dispensando o processamento individual dos múltiplos vetores unimodais. Isso se traduz em um ganho de eficiência, caso o vetor de características

fundidas seja pequeno em relação ao tamanho de todos os vetores de características unimodais juntos. Por fim, a complexidade em parametrizar e ajustar um único método de decisão é consideravelmente menor do que fazer o mesmo para *n* modalidades.

Apesar de suas vantagens, a fusão prévia é difícil de ser implantada. Um dos motivos é a dificuldade de identificação de correlação entre características. Isso acontece porque as características de diferentes modalidades tem diferentes representações. Um vetor de características SIFT, por exemplo, carrega magnitudes de gradientes em suas células enquanto um vetor de características MFCC carrega coeficientes que caracterizam um espectro sonoro. Outro motivo é a dificuldade de identificação de correlação entre características intermodais não sincronizadas temporalmente. Um padrão visual, por exemplo, pode ser complementar a um padrão aural que ocorre em um instante de tempo diferente. Por fim, ainda não se conhece um modelo adequado para representação única de informação combinada proveniente de múltiplas modalidades que seja compacto e, ao mesmo tempo, capaz de representar adequadamente o conteúdo. Um exemplo de método de fusão prévia é discutido na Seção 5.5.

5.5. Dicionários Multimodais e Representação de Fusões

Este seção discute um método eficaz para fusão prévia multimodal. Fusão prévia implica na obtenção de representações únicas (um único vetor de características) advindas de múltiplas representações de informações de uma coleção, de tal forma que estas representações únicas contenham informações relevantes provenientes das múltiplas representações originais. Estas representações únicas são ditas fundidas. O método discutido aqui é independente de domínio de aplicação e é flexível em relação à quantidade de modalidades a serem fundidas. Além disso, diferente da maioria dos trabalhos similares atualmente encontrados na literatura, é capaz de produzir representações que contém, simultaneamente, padrões unimodais e multimodais.

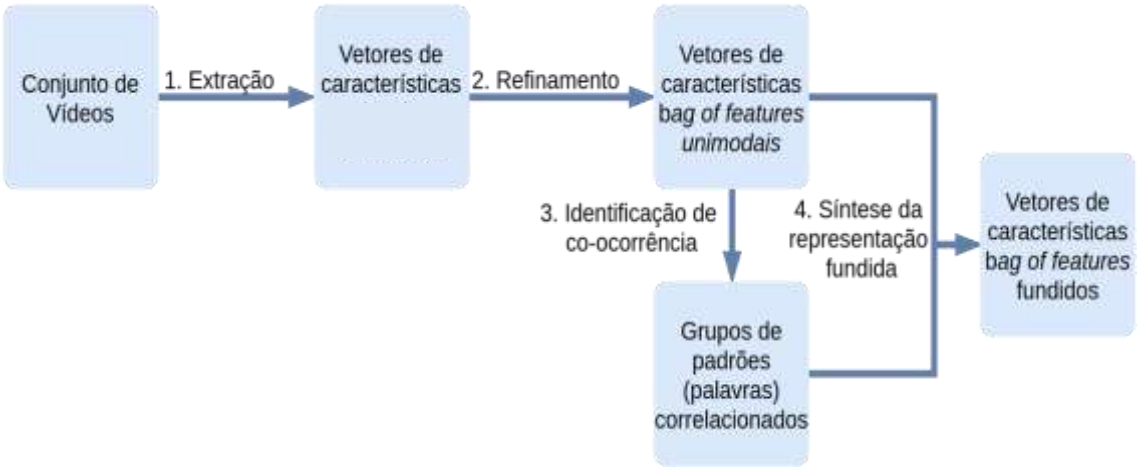


Figura 5.5. Visão geral do método de fusão prévia

A Figura 5.5 apresenta uma visão em alto nível dos passos do método, o qual é composto por quatro etapas. A primeira delas (Extração) determina múltiplos processos de produção de vetores de características, um por modalidade. A segunda etapa (Refinamento) se dá pela produção, à partir das características unimodais brutas, de representações comuns por meio de características de médio nível semântico (BoF). Na terceira etapa (Identificação de co-ocorrência) é realizada uma análise das

características unimodais BoF em busca de correlação inter/intra modalidades. Por fim, na última etapa (Síntese da representação fundida), uma representação única é produzida, a partir das as representações de médio nível semântico das correlações descobertas entre elas.

Usaremos vídeo como fonte de informação por conveniência. Explica-se: vídeo é multimodal por natureza, possuindo tanto fontes de informações visuais (imagens e texto) quanto aurais (áudio), sendo propício para exemplificar fusão multimodal assim como para aplicar o método a alguma tarefa de análise de dados (de vídeo no caso). Deve-se, contudo, estar claro que o método é independente dos tipos de fontes de informação.

Iremos tomar como exemplo a tarefa de segmentação temporal de vídeo em cenas, uma tarefa ainda em aberto na área [Kishi, R., Trojahn, T. e Goularte, R., 2019]. A segmentação temporal de vídeo em cenas consiste em definir onde, temporalmente, estão as fronteiras entre as cenas deste vídeo. Uma cena é definida como um conjunto de tomadas adjacentes semanticamente relacionadas [Saraceno e Leonardi, 1997] onde, tomadas, por sua vez, são sequências de quadros capturados ininterruptamente por uma única câmera [Koprinska e Carrato, 2001]. Considerando que a maioria das técnicas de segmentação de vídeo em cenas é baseada no agrupamento de tomadas adjacentes por sua semântica [Del Fabro e Boszormenyi, 2013], a extração da semântica das tomadas de um vídeo caracteriza-se como etapa fundamental neste processo, o que pode ser alcançado de modo latente por métodos de dicionários multimodais.

5.5.1. Extração de Características

A etapa 1 do processo ilustrado na Figura 5.5 corresponde à extração de características. Para o caso do vídeo a extração ocorre nas modalidades visual e aural, extraindo-se *features* de baixo nível das imagens (SIFT) e do áudio (MFCC). Esta etapa ocorre conforme discutido na Seção 5.3.1.

Como o volume de informações em vídeo é muito alto, o número de vetores de características extraídas também é alto. Comumente limita-se a quantidade de vetores escolhendo-se um subconjunto representativo das informações antes da extração. Quando se trata da extração de características da modalidade visual, é realizada a seleção de um ou mais quadros chave para representar o conteúdo de cada trecho de um determinado vídeo ou até mesmo de todo o vídeo, dependendo da tarefa em questão. Um quadro chave é um dos quadros que compõem a imagem em movimento do vídeo, escolhido para representar da melhor forma possível o conteúdo de todos os outros quadros restantes daquele segmento.

```
1  def sift(frame):
2    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
3    sift = cv2.xfeatures2d.SIFT_create()
4    kp, des = sift.detectAndCompute(gray, None)
5    return des
6
7  def extrairSIFT(video, keyframes):
8    cap = cv2.VideoCapture(video)
9    frameNumber = 0
10   featuresVisuais = []
11
12   while(cap.isOpened()):
13     ret, frame = cap.read()
14     if not ret:
15       break
16
17     if frameNumber in keyframes:
18       print("Extracting SIFT from keyframe %d" % frameNumber)
19       featuresVisuais.append(sift(frame))
20
21     frameNumber = frameNumber+1
22   cap.release()
23   return featuresVisuais
```

Listagem 5.5. Método geral para extração de características SIFT de uma tomada de vídeo.

A Listagem 5.5 apresenta um exemplo de código Python para extração de características visuais SIFT de um vídeo. O método *extrairSIFT* (linha 7) recebe como entrada o vídeo a ser processado (*video*) e uma lista contendo os números sequenciais dos quadros de vídeo que são quadros-chave (*keyframes*). Esta lista é obtida selecionando o quadro mediano de cada tomada do vídeo. Por exemplo, supondo que a tomada 1 comece no quadro 0 do vídeo e termine no quadro 100, o quadro-chave representativo da tomada 1 será o quadro 50. Percorre-se então o vídeo extraindo-se seus quadros (linhas 12 e 13) um a um, armazenando a imagem resultante (*frame*, linha 13) e buscando-se seu número sequencial na lista de quadros-chave, caso o quadro atual esteja presente na lista de quadros-chave (linha 17), a imagem correspondente é processada para se extrair os vetores de características SIFT. Para isso chama-se o método *sift* (linha 19) passando como argumento a imagem (*frame*). O resultado para cada quadro/imagem, isto é, os vetores de características, é armazenado na lista *featuresVisuais* (linha 19). O método *sift* (linha 1), por sua vez, transforma a imagem para tons de cinza (linha 2), cria o extrator de características (linha 3) e computa os pontos-chave e os vetores de características (*kp* e *des*, linha 4), retornando os vetores (*des*, linha 5).

A mesma ideia se aplica, de maneira análoga, à modalidade aural de vídeos. É escolhido um conjunto de amostras que melhor representam o conteúdo aural de todo o vídeo/segmento de vídeo e que seja compacto o suficiente para possibilitar e/ou agilizar o processamento.

```
1  def extrairMFCC(video, segTomadas):
2      audio = VideoFileClip(video).audio
3      fps = VideoFileClip(video).fps
4      features = []
5
6      for tomada in segTomadas:
7          inicio = math.ceil(tomada[0]/fps)
8          fim = math.floor(tomada[1]/fps)
9          audio_tomada = audio.subclip(inicio, fim).to_soundarray()
10         mfcc_tomada = mfcc(audio_tomada, 44100)
11         features.append(mfcc_tomada)
12
13     return features
```

Listagem 5.6. Método geral para extração de características MFCC de um segmento de vídeo.

A Listagem 5.6 apresenta um método/função em Python que recebe como entrada um vídeo (*video*) e um arquivo contendo as transições de tomadas do vídeo (*segTomadas*) no formato *Comma Separated Values* (CSV), onde o primeiro valor de cada linha representa o quadro inicial da tomada e o segundo valor representa o quadro final da tomada. Por exemplo,

0, 123

124, 357

significa um vídeo com duas tomadas, onde a primeira tomada termina no quadro 123 e a segunda tomada começa no quadro 124. O segmento de vídeo, então, para o qual se irá gerar uma representação MFCC é uma tomada. Como as características aurais são extraídas do áudio em um intervalo de tempo, o intervalo de cada tomada foi calculado para servir de parâmetro para o método de extração. O cálculo do intervalo para cada tomada (linha 6) divide o número do quadro inicial e final da tomada pela taxa de quadros do vídeo (*fps*). Assim, tem-se que as variáveis *inicio* e *fim* determinam o intervalo de tempo da tomada (linhas 7 e 8). Por exemplo, para uma taxa de quadros igual a 25fps e sendo a tomada 1 compreendida entre os quadros 0 e 123, temos que o intervalo de tempo da tomada 1 inicia no segundo 0 do vídeo e que termina no segundo 4,92. O método OpenCV *audio.subclip* (linha 9) determina o segmento de áudio, enquanto o método *to_soundarray* retorna as informações sobre o segmento (veja Listagem 5.4) necessárias para a aplicação do método *mfcc* que procederá a extração de características MFCC que irão representar a

tomada (linha 10). Cada representação de tomada é armazenada em uma lista (*features*, linha 11).

5.5.2. *Bag-of-Features* e Dicionários

A etapa seguinte do método de fusão exemplificado (etapa 2 na Figura 5.5) tem duas funções principais: realizar um refinamento semântico na semântica latente presente nos vetores de características locais em uma representação concisa e semanticamente representativa e, também, dar início ao processo transferência das informações unimodais representadas de maneira heterogênea para um espaço de representação comum. Estes dois objetivos são alcançados computando vetores de características BoF para cada modalidade, à partir dos vetores de características locais. Converter as diferentes representações unimodais para um espaço de representação comum resolve o dilema da heterogeneidade de representação de características de diferentes extratores de características, permitindo a comparação direta entre representações de diferentes modalidades. A Figura 5.6 ilustra a primeira parte do processo de computação de vetores de características BoF unimodais.

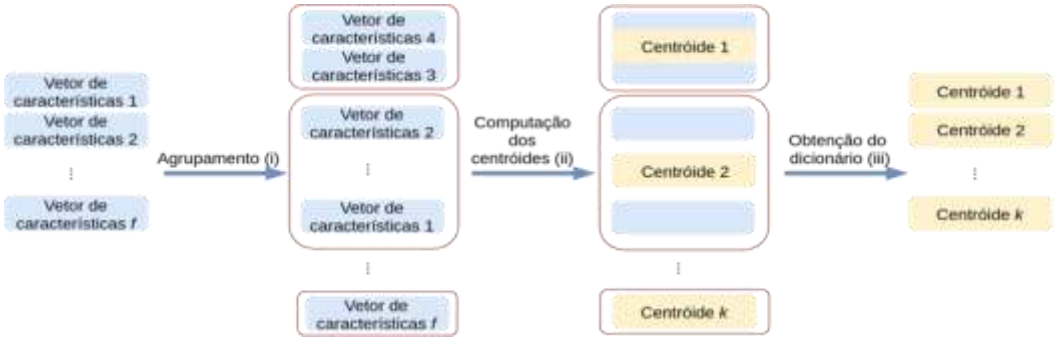


Figura 5.6. Obtenção de características BoF e do Dicionário.

O conjunto com todos os *f* vetores de características locais provenientes de um extrator de características passa, inicialmente, por um processo de agrupamento pelo método *k-means*. Os *k* grupos produzidos indicam a existência de padrões naquela modalidade. Como exemplo, espera-se que vetores de características MFCC que representam o som da voz de um locutor específico sejam mais próximos um do outro do que de vetores de características MFCC que representam a voz de um segundo locutor com voz distinta e, desta forma, sejam atribuídos a um mesmo grupo pelo processo de agrupamento.

Cada grupo/padrão é então representado por um centróide, que é um vetor de características sintetizado com as médias dos vetores de características que compõem aquele grupo. O conjunto com os *k* centróides produzidos neste processo é chamado dicionário de características e a cada um destes centroides é dado o nome de **palavra unimodal** (ou simplesmente palavra), em analogia ao modelo BoW que emprega um dicionário de palavras textuais. Além disso, costuma-se explicitar na terminologia a modalidade de origem da palavra (*e.g* palavra visual/aural).

A Listagem 5.7 apresenta um exemplo de código em OpenCV Python para a construção de dicionários de palavras unimodais. O método *criar_dicionario* recebe como parâmetro uma lista do tipo *bag-of-features* contendo as características de todas as

tomada (*f_tomadas*, linha 1), vinda, por exemplo, do método apresentado na Listagem 5.5. Nesse caso, são então *bags* de características visuais. Além disso, o método recebe também como parâmetro o tamanho do dicionário que ser pretende gerar (*numero_centroides*). Esse tamanho corresponde ao número de grupos que se pretende gerar ao se aplicar o algoritmo de agrupamento *k-means* (parâmetro k do *k-means*). Entre as linhas 3 e 5 da Listagem 5.7, para cada tomada presente na lista *f_tomadas*, são recuperados os vetores de características que, por sua vez, são adicionados a um espaço comum de características (*data*, linhas 5 e 6). Esse espaço comum será então utilizado para se gerar os k agrupamentos. Para tanto se utiliza o método OpenCV *cv2.kmeans* (linha 8), sendo que os centroides de cada grupo (palavra visual, nesse caso) são retornados e armazenados na lista *center* (linha 8). A lista *center* corresponde ao dicionário de palavras visuais de tamanho k.

```
1 def criar_dicionario(f_tomadas, numero_centroides):
2     data = []
3     for tomada in f_tomadas:
4         for feat_vet in tomada:
5             data.append(feat_vet)
6     data = np.float32(np.asarray(data))

7     crit = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
              100, 0.1)
8     ret, label, center = cv2.kmeans (data, numero_centroides, None, crit, 10,
                                       cv2.KMEANS_RANDOM_CENTERS)
9     return center
```

Listagem 5.7. Exemplo em OpenCV Python de construção de um dicionário de palavras unimodais.

A etapa seguinte, e final, da computação de vetores de características BoF é ilustrada na Figura 5.7, que ilustra a contagem de ocorrências das palavras do dicionário de características em cada segmento de informação, no exemplo cada tomada de vídeo. Isso é feito comparando cada vetor de características proveniente de cada tomada de um vídeo com todas as palavras do dicionário de características. É contabilizada, para cada um destes vetores, uma ocorrência da palavra que mais se assemelha àquele vetor de características em questão. O conjunto de contagens de ocorrências das palavras em um vídeo é armazenado em um vetor chamado histograma de ocorrência de palavras (ou vetor de características BoF) que é considerado uma representação de médio nível semântico deste vídeo. Há um mapeamento entre o índice da célula de um vetor de características BoF de uma modalidade e cada palavra daquela modalidade (por exemplo, a terceira célula de qualquer vetor de características BoF é relacionado à palavra w_3, s.p.g), permitindo a comparação entre vetores através de uma medida de similaridade/distância entre vetores unidimensionais. A intuição por trás da representação pelo modelo BoF é a de que vídeos com histogramas de ocorrência semelhantes devem apresentar padrões semelhantes naquela modalidade e, assim, portarem semântica semelhante.

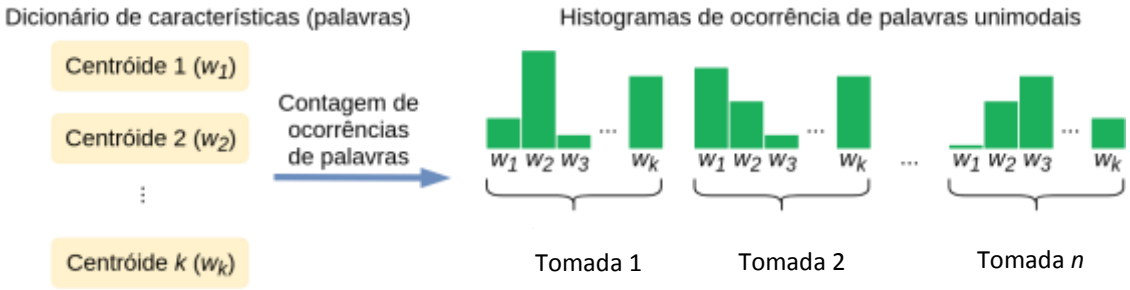


Figura 5.7. Geração dos histogramas representativos dos dados – vetores de características BoF.

```
1  def gerar_histogramas(tomadas, dicionario):
2    histograms = []
3    for tomada in tomadas:
4       histograms.append(np.zeros(len(dicionario)))

5     for i in range(len(tomadas)):
6       for feature in tomadas[i]:
7           indice = encontrar_match(feature, dicionario)
8           histograms[i][indice] = histograms[i][indice]+1
9        histograms[i] = histograms[i]/sum(histograms[i])
10     return histograms

11 def encontrar_match(feature, dicionario):
12     dist = []
13     for palavra in dicionario:
14        dist.append(distance.euclidean(feature, palavra))
15     return dist.index(min(dist))
```

Listagem 5.8. Exemplo em OpenCV Python de geração de histogramas representativos de tomadas.

A Listagem 5.8 exemplifica, em Python OpenCV, como é possível gerar os histogramas de frequências das tomadas ilustrados na Figura 5.5. O dicionário de características (Figura 5.5) equivale a *dicionário*. As características de cada tomada (*feature*) são comparadas às palavras no dicionário por meio do método *encontrar_match* (linhas 7 e 11), que usa distância euclideana (linha 14) para calcular a similaridade entre as palavras e as *features*.

Apesar dos vetores de características BoF de diferentes modalidades possuírem contagens de padrões em suas células e, possivelmente, possuírem mesmos comprimentos, os padrões representados por eles em células de mesmo índice são diferentes e provavelmente sequer relacionados. Uma comparação direta entre vetores de características BoF de diferentes modalidades não tem utilidade aparente. É necessário um processamento adicional para concluir a transferência das informações

heterogêneas unimodais para um espaço de representação comum. Isto pode ser realizado detectando-se co-ocorrência de padrões.

5.5.3. Detecção de co-ocorrência

A detecção de co-ocorrência completa o processo que permite a comparação entre informações unimodais representadas de maneira heterogênea e ainda lida com outro dilema da fusão prévia: a falta de sincronismo temporal entre informações de diferentes modalidades.

Esta etapa tem como entrada os vetores de características BoF produzidos para as diferentes modalidades na etapa descrita e gera como saída um mapeamento dos padrões correlacionados indicando quais deles, em conjunto, participam da expressão de um padrão específico presente no conjunto de vídeos. Inicialmente converte-se os histogramas de ocorrência de todas as palavras de cada modalidade em cada tomada (histograma de ocorrências por tomada) em histogramas de ocorrência de cada palavra em todas as tomadas (histograma de ocorrências por palavra).

A conversão é realizada por meio de uma transposição da matriz formada pela sobreposição dos histogramas de ocorrências de palavras em cada tomada. Em OpenCV Python isso pode ser obtido por meio do método *transpose*:

```
...
histogramasVisuais = gerar_histogramas(featuresVisuais, dicionarioVisual)
histogramasVisuais = np.asarray(histogramasVisuais).transpose()
...
```

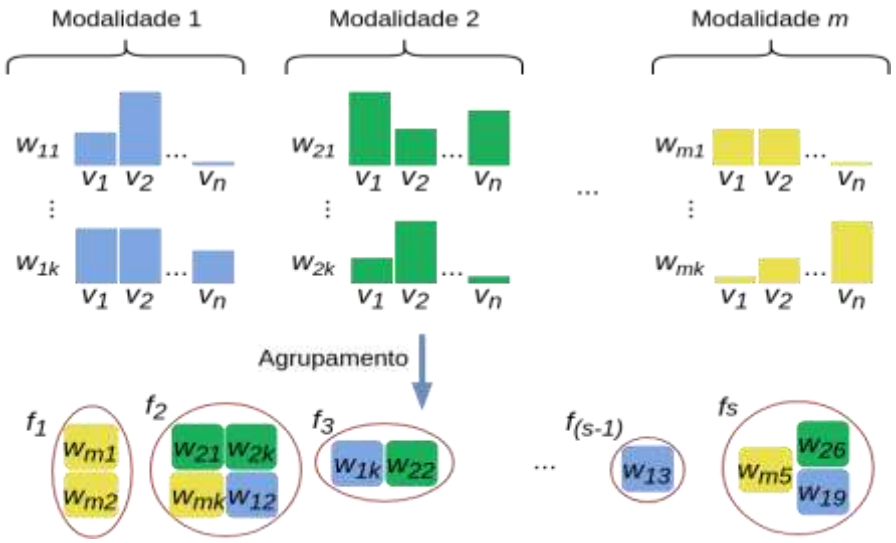


Figura 5.8. Agrupamento de palavras unimodais com base em seus padrões de ocorrência em diferentes vídeos. Cores iguais indicam que se trata de informações de uma mesma modalidade.

Os histogramas de ocorrência por palavra contêm o padrão de ocorrência daquela palavra em diferentes tomadas. Dessa forma, o cálculo de

similaridade/dissimilaridade entre histogramas de ocorrências por palavra corresponde à comparação de palavras em relação às suas ocorrências nas tomadas do conjunto. A computação dos histogramas de ocorrência por palavra de todas as modalidades permite a comparação, relativa ao padrão de ocorrência, entre palavras de diferentes modalidades. Como palavras correspondem a padrões nos vídeos/tomadas, ao comparar palavras de diferentes modalidades está se comparando, essencialmente, os padrões que as originaram. Baseando-se neste fato, o processo de detecção de co-ocorrência se completa pela etapa ilustrada na Figura 5.8.

A etapa ilustrada na Figura 5.8 é efetuada pela aplicação do método *k-means*, agrupando as palavras w_{ij} pela distância entre seus histogramas de ocorrência por palavra, onde i é o índice de uma modalidade e j é o índice de cada palavra da modalidade i, com $1 \leq i \leq m$ e $1 \leq j \leq k$. Os grupos f_o, com $1 \leq o \leq s$ produzidos neste processo contém palavras (padrões unimodais) que ocorrem em quantidades semelhantes em determinados vídeos. Ao determinar estes grupos, espera-se identificar padrões unimodais que se complementam com o objetivo de expressar um conceito semântico. Um exemplo de padrões com essa característica pode ser visto na Figura 5.9.

Figura 5.9. Exemplo de co-ocorrência de padrões aurais/visuais em diferentes vídeos.

Os dois quadros presentes na Figura 5.9, são provenientes de vídeos diferentes e contém padrões visuais semelhantes. Além disso, há padrões aurais – sons de pinguins – que também co-ocorrem nos mesmos dois vídeos. Em um processo de agrupamento como o recém descrito, estes padrões visuais e aurais provavelmente seriam atribuídos a um mesmo grupo, já que os padrões visuais presentes na representação gráfica de um pinguim são, na *BBC Dataset* (conjunto de informações utilizado), sempre acompanhados de sons de pinguins. Terminado o processo de computação de grupos de palavras unimodais correlacionadas, é necessário gerar uma representação única para cada grupo.

```
1  def deteccao_coocorrencia(hist_modalidades, numero_grupos):
2      hist_modalidades = np.float32(np.asarray(hist_modalidades))

3      crit = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER,
                100, 0.1)
4      ret, label, center = cv2.kmeans(hist_modalidades, numero_grupos, None,
                                       crit, 10, cv2.KMEANS_RANDOM_CENTERS)
5      grupos = []
6      for i in range(numero_grupos):
7          grupos.append([])

8      for i in range(len(label)):
9          grupos[label[i][0]].append(hist_modalidades [i])

10     histogramasGrupos = []
11     for grupo in grupos:
12         histogramasGrupos.append(maxPooling(grupo))

13     return histogramasGrupos

14 def maxPooling(grupo):
15     if len(grupo) == 1:
16         return grupo[0]
17     vetor = []
18     for i in range(len(grupo[0])):
19         vetor.append(max(coluna(grupo, i)))
20     return vetor
```

Listagem 5.9. Exemplo em OpenCV Python da detecção de co-ocorrência.

A Listagem 5.9 ilustra como implementar em OpenCV Python a detecção de co-ocorrência discutida nesta subseção. Os histogramas de palavras unimodais unidos em um único espaço de características (hist_*modalidades*) são entrada para o método *deteccao_coocorrencia*. As palavras unimodais são agrupadas utilizando o algoritmo kmeans gerando grupos compostos, contendo combinações de palavras visuais e aurais (linhas 3 e 4). O método *cv2.kmeans*, que implementa o algoritmo k-means, devolve, além dos centróides, uma lista de índices (*label*, linha 4). Tais índices permitem identificar em que agrupamento está cada palavra unimodal de entrada presente em *hist_modalidades*. Assim, é possível percorrer *hist_modalidades* colocando as palavras em grupos de fato (*grupos*), pois o método *cv2.kmeans* não retorna os grupos, apenas os índices. Agora, com os agrupamentos de palavras multimodais criados, pode-se completar a fusão gerando-se uma representação única, fundida. Isso é feito utilizando-se o método *maxPooling* (linhas 12 e 14), cujo detalhamento, entre outros métodos de pooling, está discutido na Seção 5.5.4.

5.5.4. Representação Única

A etapa final do método de fusão de características exemplificado lida com o dilema restante da fusão prévia: produz uma representação única para a expressão de múltiplos padrões correlacionados, mantendo o significado original das representações originais, ao mesmo passo em que descarta detalhes irrelevantes. Tal representação única permite pós processamento com o objetivo de melhor aproveitar a semântica na tarefa fim. A representação única da presença de padrões correlacionados é produzida através da técnica de *pooling*, bastante comum na área de Visão Computacional [Boureau et al., 2010]. De acordo com Boureau et al. (2010), o *pooling* produz, a partir das saídas de diferentes extratores de características, uma representação única combinada que preserva informação relevante à uma determinada tarefa, enquanto descarta detalhes irrelevantes.

A Figura 5.10 ilustra a aplicação do *pooling* na fusão de características, com uma visão detalhada dos histogramas que representam as palavras do grupo f_1.

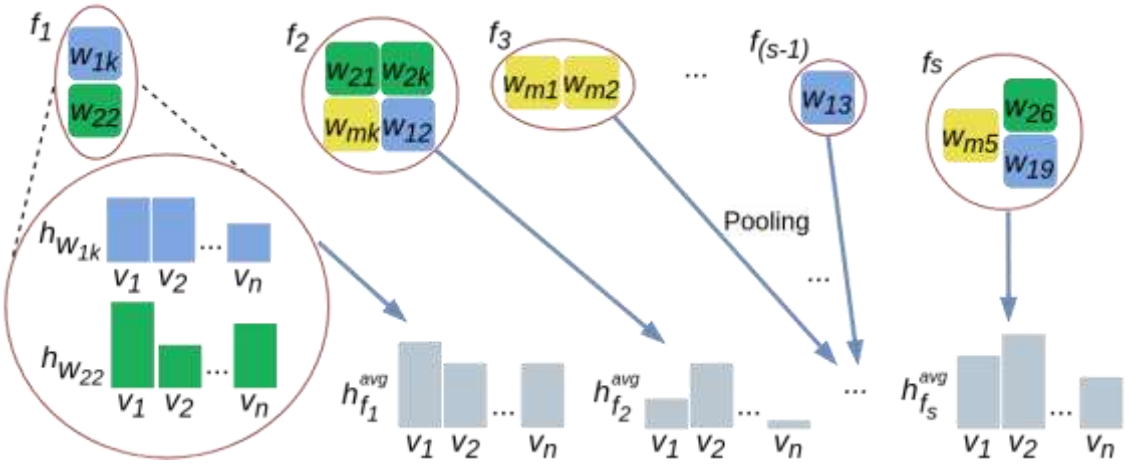


Figura 5.10. Exemplo do processo de *pooling*.

Considerando os grupos $f_1 \ldots f_s$, compostos por palavras unimodais, produzidos pela etapa descrita na Seção 5.5.3, o *pooling* ocorre através da aplicação, para cada grupo $\boldsymbol{f} \in \{\boldsymbol{f_1}, \ldots, \boldsymbol{f_s}\}$, de uma função que toma como entrada o conjunto dos histogramas de ocorrência em vídeos h_w de cada palavra unimodal w que compõe $\boldsymbol{f}$ nos vídeos $v_1,\ldots, v_n$ e produz como saída um histograma único h^{avg} que representa $\boldsymbol{f}$.

Há duas estratégias de *pooling* adequadas nesse tipo de fusão, que diferem em como as ocorrências do padrão correspondente a um determinado grupo serão contabilizadas: *average pooling* e *max pooling*. Ambas as estratégias podem, ainda, ser modificadas utilizando uma heurística de ponderação para aprimorar os histogramas de ocorrências de padrões produzidos pela fusão.

O *average pooling*, conforme sugere o nome, determina que seja computada uma média das ocorrências de cada palavra unimodal em cada vídeo. Os valores das células do histograma de ocorrências $\boldsymbol{h_f^{avg}}$, do *average pooling* do grupo $\boldsymbol{f}$ nos vídeos $v_1,\ldots, v_n$, são calculados usando a equação:

$$h_f^{avg}[v_i] = \frac{\sum_{w_j \in f} h_{w_j} v[i]}{|f|} \tag{5.3}$$

onde $\boldsymbol{h_{w_j}}$ é o histograma da palavra $\boldsymbol{w_j}$ que compõe o grupo $\boldsymbol{f}$.

O *max pooling*, também como sugere o nome, determina que sejam utilizados os máximos dentre todas as ocorrências de palavras unimodais que compõem um determinado grupo em cada vídeo. Os valores das células do histograma de ocorrências *max pooling* $\boldsymbol{h_f^{max}}$, do grupo f nos vídeos v_1,..., v_n são calculados usando a equação:

$$\boldsymbol{h_f^{max}[v_i] = \max_{w_j \in f} h_{w_j} v[i]} \tag{5.4}$$

onde $\boldsymbol{h_{w_j}}$ é o histograma da palavra $\boldsymbol{w_j}$ que compõe o grupo $\boldsymbol{f}$.

O *max pooling*, em contraste com o *average pooling*, produz um histograma de ocorrências de padrões baseado na parte mais representativa das ocorrências de palavras unimodais que compõem cada padrão. Desse modo, *o max pooling* é capaz de representar ocorrências heterogêneas com contagens semelhantes de um mesmo padrão, desde que haja uma palavra unimodal componente deste padrão sendo expressa de maneira semelhante em todas estas ocorrências.

Considerando o mesmo exemplo apresentado para o *average pooling*, diferentes expressões de explosões podem possuir conteúdo visual distinto e mesmo assim apresentarem histogramas de ocorrência *max pooling* semelhantes, desde que seu conteúdo aural seja semelhante e mais representativo do que o visual. Apesar de mais flexível, o *max pooling* é mais suscetível a erros de representação já que o valor final de contagem de ocorrências pode ser originado de qualquer uma das palavras unimodais que compõem um determinado padrão.

Ponderação

Dentre as virtudes do método de fusão proposto estão a preservação do significado original nos valores componentes dos vetores de características fundidos, bem como a rastreabilidade das origens destes valores. Graças a estas virtudes, é possível aprimorar os vetores de características fundidas identificando situações onde duas ou mais fontes de informação apresentam padrões de ocorrências semelhantes de palavras unimodais.

Duas ou mais fontes de informação indicando a existência de um padrão em um vídeo, intuitivamente, aumenta a confiança na existência daquele padrão. Pode-se então expressar a menor probabilidade de erro na identificação de um padrão ponderando sua contagem no histograma de ocorrências. O destaque destes padrões mais confiáveis aumenta o poder discriminatório das medidas de distância entre histogramas que poderão vir a ser empregadas nas ferramentas de análise. A abordagem proposta para realizar a ponderação se dá pela aplicação da seguinte equação nos histogramas de ocorrências fundidos $\boldsymbol{h_f^{fused}}$ dos grupos $\boldsymbol{f} \in \{\boldsymbol{f_1}, \ldots, \boldsymbol{f_s}\}$, produzindo histogramas de ocorrências ponderados $\boldsymbol{h_f^{weighted}}$:

$$\boldsymbol{h_f^{weighted}[v]} = \begin{cases} \boldsymbol{h_f^{fused}[v]\ x\ \partial}, & \textbf{se} \quad |\boldsymbol{f}| > \mathbf{1} \\ \boldsymbol{h_f^{fused}[v]}, & \textbf{caso contrário} \end{cases} \tag{5.5}$$

onde *v* é um dos vídeos em $\{v_1,..., v_n\}$ e $\boldsymbol{\partial}$ é o fator de ponderação. Os histogramas $\boldsymbol{h_f^{fused}}$ podem ser histogramas produzidos por *average pooling* ou *max pooling*.

Conversão em histogramas por tomada

O fim do método se dá pela conversão dos histogramas de ocorrências de cada palavra nas tomadas em histogramas de ocorrência de todas as palavras em cada tomada.

Os histogramas de ocorrência $\boldsymbol{h_v^{fused}}$, por tomada $\boldsymbol{v} \in \{\boldsymbol{v_1}, \ldots, \boldsymbol{v_n}\}$ são obtidos facilmente a partir de histogramas de ocorrência $\boldsymbol{h_f^{fused}}$ por palavra $\boldsymbol{f} \in \{\boldsymbol{f_1}, \ldots, \boldsymbol{f_n}\}$, uma vez que o histograma $\boldsymbol{h_v^{fused}}$ de tamanho *n* da tomada *i*, é formado pelos valores contidos nas posições *i* dos *n* histogramas $\boldsymbol{h_f^{fused}}$. Os histogramas $\boldsymbol{h^{fused}}$podem ser histogramas produzidos por *average pooling*, *max pooling* ou ponderação. Estes histogramas são o produto final da fusão de características e servirão de entrada para uma técnica-fim, como uma técnica de segmentação de vídeo em cenas.

5.6. Avaliação

A avaliação de sistemas de RIM normalmente implica em analisar e medir comparativamente um ou mais sistemas por meio de metodologia bem estabelecida. A metodologia compreende definir: o tipo de avaliação; uma base dados anotada (*groundtruth*) e medidas de avaliação. As avaliações podem ser realizadas com ou sem a participação direta do usuário do sistema. Neste texto estamos interessados nos métodos sem a participação do usuário. Tanto as medidas de avaliação quanto a base de dados devem ser adequadas para julgar a eficácia de um dado sistema e estão intimamente ligadas à tarefa que se deseja realizar, variando de tarefa para tarefa [Blanken et al., 2010].

O *groundtruth* utilizado para essa tarefa é composto pela a BBC *Planet Earth Dataset*[10], referenciada de agora em diante simplesmente como BBC *dataset* ou base BBC. Tal base contém 11 vídeos do documentário BBC *Planet Earth*, totalizando mais de 8 horas de vídeo, 4913 tomadas e 670 cenas. Além dos vídeos existe a anotação textual indicando os limites (cortes ou transições) entre cenas (segmentação ideal), possibilitando a comparação com segmentações geradas por técnicas automáticas.

As métricas, por sua vez, devem medir o quão distante o resultado de uma técnica de segmentação está da segmentação ideal (*groundtruth*). Nesse sentido, a literatura reporta diversas métricas diferentes, sendo as mais utilizadas pelos pesquisadores descritas a seguir nesta seção.

As métricas mais amplamente utilizadas para a tarefa em questão são a Precisão e a Abrangência. A Precisão (do inglês *precision* - *P*) e a Abrangência (do inglês *Recall*

[10]http://imagelab.ing.unimore.it/imagelab/page.asp?IdPage=5

- *R*) [Rijsergen 1979] são originárias da área de recuperação de informação. No contexto de segmentação em cenas, como cada tomada é ou não é de transição, o conjunto resposta do segmentador inclui os seguintes casos: Verdadeiro Positivo (*vp*), quando uma tomada retornada como sendo de transição corresponde de fato a uma transição no *groundtruth*; Falso Positivo (fp), quando uma tomada retornada como sendo de transição não corresponde a uma transição de fato; Verdadeiro Negativo (vn), é uma tomada não retornada pelo segmentador e que de fato não é de transição; Falso Negativo (fn), é uma tomada não predita pelo segmentador mas que, na realidade, é de transição. Assim, Precisão *P* e Abrangência *R* são definidas como:

$$P = \frac{vp}{vp + fp} \tag{5.6}$$

$$R = \frac{vp}{vp + fn} \tag{5.7}$$

A Precisão e Abrangência avaliam duas características diferentes de um segmentador qualquer. Caso um algoritmo obtenha elevada Precisão, significa dizer que a maioria das transições detectadas pelo algoritmo são transições verdadeiras, ou seja, o número de falsos positivos obtido é baixo. Mas isso não implica que todas as transições verdadeiras foram detectadas. Por outro lado, caso o algoritmo obtenha uma elevada Abrangência, significa que a maioria das transições que existem na base confiável foram detectadas, indicando um baixo número de falsos negativos. Mas não implica que somente transições verdadeiras foram detectadas.

Como tanto *P* quanto *R* medem diferentes aspectos da segmentação em si, um modo de agrupá-los em um valor único é por meio da Medida F ou F-score [Rijsergen 1979]. A Medida F é uma média harmônica entre os valores *P* e *R* que apresenta como vantagem o fato de ser mais conservadora que a média aritmética simples entre *P* e *R*, podendo inclusive dar maior prioridade tanto a *P* quanto a *R* em variantes tais como $F_{0.5}$ ou F_2. A métrica Medida F $\boldsymbol{F\beta}$ é definida como:

$$\boldsymbol{F_\beta} = (1 + \beta^2).\frac{\mathrm{P}\,.\,\mathrm{A}}{(\beta^2.\ \mathrm{P}) + \mathrm{A}} \tag{5.8}$$

Onde *b* é usado para dar prioridade à Precisão (valores abaixo de um) ou à Abrangência (valores acima de um). Caso $b = 1$ (F_1), não há prioridade qualquer, sendo a mesma doravante chamada F_{PR}, definida como:

$$\boldsymbol{F_{PR}} = 2\,.\frac{\mathrm{P}\,.\,\mathrm{A}}{\mathrm{P} + \mathrm{A}} \tag{5.9}$$

Embora amplamente utilizadas em trabalhos seminais na área [Del Fabro and Boszormenyi, 2013], as métricas de Precisão e Abrangência possuem limitações quanto a segmentação em cenas. Nesse sentido, a Figura 5.11 ilustra duas técnicas que obtém, para um *groundtruth* com duas transições (indicado como *Confiável* na Figura 5.11),

diferentes segmentações. Note que os valores de *P*, *R* e F_{PR} da **Técnica 1** (20%, 50% e 28% respectivamente), são superiores aos da **Técnica 2** (0%, 0% e 0% respectivamente), embora intuitivamente a segunda tenha obtido uma segmentação mais próxima da desejada (apenas deslocada por uma tomada).

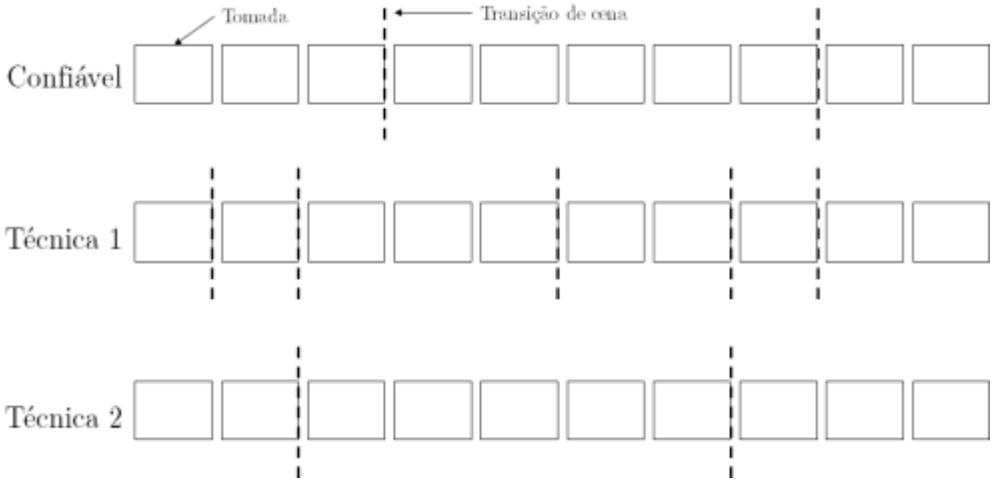


Figura 5.11. Ilustração de uma segmentação em cenas segundo o *groundtruth* (Confiável) e de duas técnicas hipotéticas para um determinado vídeo com 10 tomadas. Os retângulos representam as tomadas e a as linhas verticais denotam as transições de cenas.

Por tal motivo, diversos pesquisadores costumam adotar uma janela de tolerância, normalmente baseada em um número de quadros de vídeo, em tempo ou em número de tomadas [Baraldi et al., 2015]. Por exemplo, [Rasheed e Shah, 2003) especificam que uma transição de cena é considerada correta se estiver até dez segundos da transição confiável, enquanto Hanjalic et al. (1999) usaram três tomadas de tolerância. Tal prática, porém, causa inconvenientes tais como a dificuldade de comparar técnicas que usaram janelas de tolerância diferentes, além de ocultar pequenas diferenças que diferentes técnicas poderiam gerar. Del Fabro e Boszormenyi (2013) citam como outra possível limitação das métricas a dificuldade de medir quando uma cena é parcialmente detectada, quando o início e fim da cena não estão alinhados.

Nesse sentido, Vendrig e Worring (2002) definiram duas métricas específicas para a segmentação em cenas, conhecidas como Cobertura (do inglês *Coverage* - *C*) e Transbordamento (do inglês *Overflow* - *O*). A Cobertura procura medir quantas tomadas pertencentes à mesma cena foram corretamente agrupadas. Já o Transbordamento mede quantas tomadas, mesmo não pertencentes à mesma cena, foram incorretamente agrupados em uma mesma cena. As medidas de Cobertura (*C*) e Transbordamento (*O*) são definidas como:

$$C(x_t) = \frac{max_{j=0\ldots n}\#(y_j)}{\#(x_t)} \tag{5.10}$$

$$C(x_t) = \frac{\sum_{j=0}^{n} \#(y_j \backslash x_t).\min(1, \#(y_j \cap x_t))}{\#(x_{t-1}) + \#(x_{t+1})} \quad (5.11)$$

Sendo $\boldsymbol{x_t}$ o conjunto de tomadas que forma a cena confiável de índice *t*, y_j o conjunto de tomadas que forma a cena de índice *j* obtida pelo algoritmo segmentador, #(*x*) o operador que retorna a quantidade de tomadas de uma determinada cena *x* e #($y_j \backslash x_t$) o número de tomadas da cena detectada que não se encontram também na cena confiável (ou, em outras palavras, a operação de diferença entre y_j e x_t). É importante ressalvar que, ao contrário das métricas *P*, *R* e *C*, no qual o valor desejado é igual a 1 (100%), o valor desejado de Transbordamento (*O*) é igual a zero (0%), que corresponde ao caso de que nenhuma tomada de uma cena adjacente foi erroneamente agrupada na cena detectada.

Para obter o valor de *C* ou *O* de um dado vídeo *V*, a média ponderada é calculada de acordo com o número de tomadas do vídeo. Assim, sendo #(*Vs*) o número total de tomadas e #(*Vxj*) o número de cenas confiáveis do vídeo *V* , o cálculo da Cobertura *C*(*V*) e do Transbordamento *O*(*V*) são dados por:

$$C(\mathrm{V}) = \sum_{t=0}^{\#(Vx_j)-1} C(x_t).\frac{\#(x_t)}{\#(V_\sigma)} \quad (5.12)$$

$$O(\mathrm{V}) = \sum_{t=0}^{\#(Vx_j)-1} O(x_t).\frac{\#(x_t)}{\#(V_\sigma)} \quad (5.13)$$

Assim como no caso de Precisão e Abrangência, os valores de Cobertura e Transbordamento podem ser agrupados por meio da média harmônica Medida F, doravante representada como F_{CO}. É importante ressalvar que o valor do cálculo de Transbordamento utilizado na métrica F_{CO} é igual a 1 - *O*, visto a característica do valor desejado de *O* ser igual a zero.

Uma das principais vantagens das medidas de Cobertura/Transbordamento sobre a Precisão/Abrangência é não requerer a especificação de um parâmetro de tolerância, que pode interferir na interpretação dos resultados da técnica avaliada, facilitando inclusive a comparação com outras técnicas desenvolvidas. Outro benefício é a de que a métrica avalia o *quanto* uma cena foi detectada, sendo, portanto, mais robusta a cenas parcialmente detectadas [Del Fabro e Boszormenyi, 2013].

Contudo, a formulação da métrica de Transbordamento definida anteriormente contém uma limitação importante caso a segmentação a ser avaliada possua alto índice de subsegmentação (do inglês *undersegmentation*), quando uma cena detectada engloba

diversas cenas do *groundtruth* (que deveriam ter sido detectadas), resultando em valores inválidos. Para ilustrar tal limitação, considere um vídeo formado de 20 tomadas e 6 cenas (5 transições) e o resultado obtido por uma técnica de segmentação com 4 cenas (3 transições) sobre o mesmo vídeo. A Figura 5.12 ilustra o vídeo e as duas segmentações, confiável e detectada, mencionados.

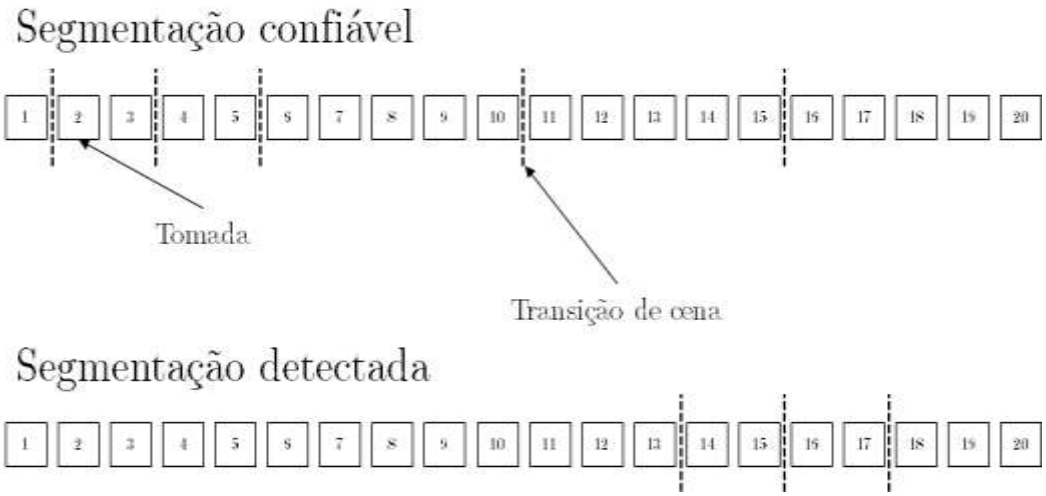


Figura 5.12. Exemplo de segmentação confiável versus segmentações geradas por técnicas hipotéticas.

Embora a segmentação detectada obtenha uma Cobertura válida de 80%, já que apenas 4 tomadas (20%) do *groundtruth* não são cobertas pela respectiva cena detectada (tomadas 14 a 17), o valor de Transbordamento é de 135%, um valor claramente inválido que excede o intervalo válido da métrica. Consequentemente, nesse caso, o valor obtido para F_{CO} (-124%) também é inválido. Para evitar tal problema, Han e Wu (2011) apresentam uma formulação alternativa para a métrica de Transbordamento, na qual se normaliza o valor para o intervalo válido [0,1]. Tal métrica, doravante chamada de *NO* (*New Overflow*), é definida como:

$$NO(x_t) = 1 - \frac{\#(x_t)}{\sum_{j, X_t \cap y_j \neq 0} \#(y_j)} \tag{5.14}$$

A principal melhoria da formulação da métrica *NO* sobre a métrica *O* é o denominador da divisão, formado pela soma do número de tomadas de todas as cenas detectadas que possuem alguma intersecção com a cena do *groundtruth* sendo analisada. Tal modificação significa que o denominador sempre será maior ou igual (melhor caso) que o numerador o que, associado com a subtração da equação, garante que o valor *NO* será um valor válido no intervalo [0,1]. Também pode-se calcular a média harmônica Medida F de maneira semelhante à métrica F_{CO}, usando o valor 1-*NO*, sendo a mesma doravante representada como F_{CNO}. Graças a tal melhoria, no exemplo hipotético ilustrado na Figura 5.12 o valor obtido de *NO* é aproximadamente 53% (0.53) e seu valor F_{CNO} é aproximadamente 59% (0.59), ambos os valores contidos no intervalo [0,1].

5.7. Considerações Finais

Este texto, parte do minicurso apresentado durante o WebMedia 2019, discutiu como realizar indexação multimídia utilizando técnicas de fusão multimodal e de detecção de co-ocorrência. Os conceitos foram apresentados e discutidos assim como exemplos de implementação em OpenCV Python. As técnicas e métodos discutidos são de interesse de uma variedade de aplicações que lidam com grandes volumes de dados não estruturados (Big Data, Recuperação de Informação Multimídia, etc.) pois, de modo eficaz, reduzem o volume de dados necessário para representar as informações mantendo representatividade. Em particular, o método apresentado de fusão prévia e de representação dos dados fundidos se mostra uma boa alternativa para tarefas relacionadas a RIM, como a segmentação de vídeo em cenas. Outras tarefas podem se beneficiar do método, como classificação de vídeos e localização de objetos.

Como limitações, a técnica de fusão apresentada utiliza apenas dois tipos de características, uma visual e outra aural, apesar de ser possível estender a técnica para utilizar também outros tipos de características, como texto. Além disso, por simplicidade, a seleção de quadros-chave (modalidade visual) apresentada no minicurso é realizada de modo ingênuo, selecionando-se o quadro mediano de cada tomada. Essa abordagem pode ser melhorada selecionando-se um conjunto de quadros-chave, o que seria mais representativo. A implementação do método de *pooling* não levou em consideração a possibilidade de se fazer ponderação, quê tem potencial para melhora de representatividade e de eficácia, consequentemente.

Currículo Resumido dos Autores

Rudinei Goularte possui graduação em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (1995). Possui mestrado (1998), doutorado (2003) e livre-docência (2011) pela Universidade de São Paulo, campus São Carlos, todos em Ciência da Computação. Atualmente é professor associado do ICMC/USP em regime de dedicação integral à docência e à pesquisa, atuando também como orientador pleno de mestrado e doutorado. Atua como consultor *ad hoc* da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). Desenvolve pesquisa em Multimídia nas linhas: codificação de vídeo digital, vídeo 3D, recuperação de informação multimídia e análise multimodal.

Tiago Henrique Trojahn possui graduação em Ciências da Computação pela Universidade Federal de Pelotas (2010), mestrado (2014) e doutorado (2019) em Ciência da Computação e Matemática Computacional pela Universidade de São Paulo, campus São Carlos. Atualmente, é professor do Instituto Federal de São Paulo, campus São Carlos, em regime de dedicação exclusiva. Possui interesse nas áreas de segmentação de vídeo digital, multimodalidade, personalização e adaptação de conteúdo e desenvolvimento para web.

Rodrigo Mitsuo Kishi possui bacharelado em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (2007) e mestrado em Ciência da Computação pela Universidade Federal de Mato Grosso do Sul (2010). É professor da

Universidade Federal de Mato Grosso do Sul desde 2011. Cursa, atualmente, doutorado em Ciência da Computação na Universidade de São Paulo (USP). Tem experiência na área de Ciência da Computação, com ênfase em Sistemas Multimídia, Bioinformática, Teoria dos Grafos e Análise de Algoritmos.

Referências

Aggarwal C.C., Zhai C. A Survey of Text Classification Algorithms. In: Aggarwal C., Zhai C. (eds) Mining Text Data. Springer, Boston, MA, 2012. p. 163–222. ISBN 978-1-4614-3223-4. Disponível em: https://doi.org/10.1007/978-1-4614-3223-4_6.

Atrey, P. K.; Hossain, M. A.; Saddik, A. E.; Kankanhalli, M. S. Multimodal fusion for multimedia analysis: a survey. Multimedia Systems, v. 16, n. 6, p. 345–379, Nov 2010. ISSN 1432-1882.

Baeza-Yates, R and Ribeiro-Neto, B., Modern Informatin Retrieval, Addison-Wesley, 2008.

Baraldi L, Grana C, Cucchiara R (2017) Recognizing and Presenting the Storytelling Video Structure with Deep Multimodal Networks. IEEE Transactions on Multimedia 19(5):955{968, DOI 10.1109/TMM.2016.2644872, URL http://ieeexplore.ieee.org/document/7797131/

Blanken, H. M., Vries, A. P., Blok, H. E. and Feng, L., Multimedia Retrieval, Springer, 2010.

Boureau, Y.; Bach, F.; Lecun, Y.; Ponce, J. Learning mid-level features for recognition. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. San Francisco, CA, USA: IEEE, 2010. p. 2559–2566. ISSN 1063-6919

Bouyakoub, F. M. and Belkhir, A. (2008) "AdaMS: Na Adaptation Multimedia System for Heterogeneous Environments", In: New Technologies, Mobility and Security (NTMS), p. 1-5.

Carletti, V.; Foggia, P.; Percannella, G.; Saggese, A.; Strisciuglio, N.; Vento, M. Audio surveillance using a bag of aural words classifier. In: 2013 10th IEEE International Conference on Advanced Video and Signal Based Surveillance. Krakow, Poland: IEEE, 2013. p. 81–86.

Chasanis, V., Kalogeratos, A. and Likas, A. (2009) "Movie segmentation into scenes and chapters using locally weighted bag of visual words", International Conference on Image and Video Retrieval, p.35:1-35:7.

Csurka, G.; Dance, C. R.; Fan, L.; Willamowski, J.; Bray, C. Visual categorization with bags of keypoints. In: In Workshop on Statistical Learning in Computer Vision, ECCV. Czech Republic: Springer, 2004. p. 1–22.

Davis, S.; Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Transactions on Acoustics, Speech, and Signal Processing, v. 28, n. 4, p. 357–366, Aug 1980. ISSN 0096-3518

Del Fabro, M., Boszormenyi, L. (2013) State-of-the-art and future challenges in video scene detection: a survey. Multimedia Syst 19(5):427–454. https://doi.org/10.1007/s00530-013-0306-4

Grauman, K.; Leibe, B. Visual Object Recognition. 1st. ed. United States: Morgan & Claypool Publishers, 2011. ISBN 1598299689, 9781598299687

Han, B.; Wu, W. Video scene segmentation using a novel boundary evaluation criterion and dynamic programming. In: IEEE International Conference on Multimedia and Expo. [s.n.], 2011. p. 1–6. ISSN 1945-7871. Disponível em: https://ieeexplore.ieee.org/document/6012001/

Hanjalic, A.; Lagendijk, R. L.; Biemond, J. Automated high-level movie segmentation for advanced video-retrieval systems. IEEE Transactions on Circuits and Systems for Video Technology, v. 9, n. 4, p. 580–588, June 1999. ISSN 1051-8215.

Havaldar, P.; Medioni, G. Multimedia Systems: Algorithms, Standards, and Industry Practices. Cengage Learning; 2009. ISBN: 1418835943.

Hu, W., Xie, N., Li, L., Zeng, X. and Maybank, S. (2011) "A survey on visual content-based video indexing and retrieval". In: IEEE Transactions on Systems, Man and Cybernatics, v. 41, p. 797-819.

Iwan, L. and Thom, J. A. 2017. Temporal video segmentation: detecting the end-of-act in circus performance videos. Multimedia Tools and Applications 76, 1 (jan 2017), 1379–1401. https://doi.org/10.1007/s11042-015-3130-3

Kishi, R. M., Trojahn, T. H., Goularte, R. 2019. Correlation based feature fusion for the temporal video scene segmentation task. Multimedia Tools and Applications 78, 1 (jun 2019), 15623–15646. https://doi.org/10.1007/s11042-018-6959-4

Ko, Y.A. A new term-weighting scheme for text classification using the odds of positive and negative class probabilities. Journal of the Association for Information Science and Technology, Wiley-Blackwell, Hoboken, NJ, USA, v. 66, n. 12, p. 2553–2565, jan. 2015. ISSN 2330-1643. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.23338.

Koprinska, I.; Carrato, S. Temporal video segmentation: A survey. Signal Processing: Image Communication, v. 16, n. 5, p. 477 – 500, 2001. ISSN 0923-5965. Disponível em: http://www.sciencedirect.com/science/article/pii/S0923596500000114.

Leung, T.; Malik, J. Representing and recognizing the visual appearance of materials using three-dimensional textons. International Journal of Computer Vision, v. 43, n. 1, p. 29–44, 2001. ISSN 1573-1405. Disponível em: http://dx.doi.org/10.1023/A:1011126920638.

Leng, C.; Zhang, H.; Li, B.; Cai, G.; Pei, Z.; He, L. "Local feature descriptor for image matching: A Survey", IEEE Access, vol. 7, p. 6424-6434, 2019.

Lowe, D. G. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, v. 60, n. 2, p. 91–110, Nov 2004. ISSN 1573-1405. Disponível em: <https://doi.org/10.1023/B:VISI.0000029664.99615.94>

Lu, C.; Drew, M. S.; Au, J. Classification of summarized videos using hidden markov models on compressed chromaticity signatures. In: Proceedings of the Ninth ACM International Conference on Multimedia. New York, NY, USA: ACM, 2001. (MULTIMEDIA '01), p. 479–482. ISBN 1-58113-394-4.

Lu, Y., Sebe, N., Hytnen, R. and Tian, Q. (2011) "Personalization in multimídia retrieval: A survey". In: Multimedia Tools and Applications, v.51, p. 247-277.

Mandal, M. K. Multimedia Signals and Systems. Kluwer Academic Publishers, 2002. ISBN: 1402072708.

Martinet, J.; Sayad, I. E. Mid-level image descriptors. In: MA, Z. (Ed.). Intelligent Multimedia Databases and Information Retrieval: Advancing Applications and Technologies. USA: IGI Global, 2012. p. 46–60.

Pouyanfar, S., Yang, Y., Chen, S., Shyu, M. and Iyengar, S. S. 2018. Multimedia Big Data Analytics: A Survey. ACM Comput. Surv. 51, 1, Article 10 (January 2018), 34 pages. DOI: https://doi.org/10.1145/3150226

Rasheed, Z.; Shah, M. Scene detection in hollywood movies and tv shows. In: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings. Madison, WI, USA: IEEE, 2003. v. 2, p. II–343. ISSN 1063-6919.

Rezende, S. O.; Marcacini, R. M.; Moura, M. F. O uso da mineração de textos para extração e organização não supervisionada de conhecimento. Revista de Sistemas de Informação da FSMA, v. 7, p. 7–21, 2011. ISSN 19835604. Disponível em: http://www.fsma.edu.br/si/7edicao.html.

Rijsbergen, C. J. V. Information Retrieval. 2nd. ed. Newton, MA, USA: ButterworthHeinemann, 1979. ISBN 0408709294.

Sakarya, U. and Telatar, Z. (2010) "Video scene detection using graph-based representations" In: Signal Processing – Image Communication, v.25, p. 774-783.

Salton, G.; Buckley, C. Term-weighting approaches in automatic text retrieval. Information Processing and Management: an International Journal, Pergamon Press, Inc., Tarrytown, NY, USA, v. 24, n. 5, p. 513–523, ago. 1988. ISSN 0306-4573. Disponível em: https://dx.doi.org/10.1016/0306-4573(88)90021-0.

Saraceno, C.; Leonardi, R. Audio as a support to scene change detection and characterization of video sequences. In: 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing. Munich, Germany: IEEE, 1997. v. 4, p. 2597–2600 vol.4. ISSN 1520-6149.

Sharma, D.; Ali, I. A modified mfcc feature extraction technique for robust speaker recognition. In: 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI). Kochi, India: IEEE, 2015. p. 1052–1057

Smeulders, A.; Worring, M.; Santini, S.; Gupta, A.; Jain, R. Content-based image retrieval at the end of the early years. IEEE Transactions on Pattern Analysis and Machine Intelligence, IEEE Computer Society, Washington DC, USA, v. 22, p. 1349-1380. ISSN 01628828. Disponível em: https://dl.acm.org/citation.cfm?id=357871.357873

Stevens, S. S.; Volkmann, J.; Newman, E. B. A scale for the measurement of the psychological magnitude pitch. The Journal of the Acoustical Society of America, v. 8, n. 3, p. 185–190, 1937. Disponível em: <http://scitation.aip.org/content/asa/journal/jasa/8/3/10.1121/1.1915893>.

Stockman, G.; Shapiro, L. G. Computer Vision. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130307963.

Tapu, R. and Zaharia, T. (2011) "High level video temporal segmentation", International Conference on Advances in Visual Computing, p. 224-235.

Toffler, A., Future Shock, Bantam, 1984.

Tuytelaars, T. and Mikolajczyk, K. (2008) "Local invariant feature detectors: a survey". In: Foundations and Trends in Computer Graphics and Vision, v. 3, p. 177-280.

Uysal, A. K.; Gunal, S. The impact of preprocessing on text classification. Information Processing and Management: an International Journal, Pergamon Press, Inc., Tarrytown, NY, USA, v. 50, n. 1, p. 104–112, jan. 2014. ISSN 0306-4573. Disponível em: <https://dx.doi.org/10.1016/j.ipm.2013.08.006.

Vendrig, J.; Worring, M. Systematic evaluation of logical story unit segmentation. IEEE Transactions on Multimedia, v. 4, n. 4, p. 492–499, Dec 2002. ISSN 1520-9210.

Vestman, V.; Gowda, D.; Sahidullah, M.; Alku, P.; Kinnunen, T. Speaker recognition from whispered speech: A tutorial survey and an application of time-varying linear prediction, Speech Communication, v. 99, 2018, p. 62-79. ISSN 0167-6393, https://doi.org/10.1016/j.specom.2018.02.009.

Xie L, Shen J, Han J, Zhu L, Shao L (2017) Dynamic multi-view hashing for online image retrieval. In: Proceedings of the 26th international joint conference on artificial intelligence, IJCAI'17, pp 3133–3139. AAAI Press. http://dl.acm.org/citation.cfm?id=3172077.317232.

Yang, L.; Wang, Y.; Dunne, D.; Sobolev, M.; Naaman, M.; Estrin, D. More than just words: Modeling non-textual characteristics of podcasts. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. New York, NY, USA: ACM, 2019. (WSDM '19), p. 276–284. ISBN 978-1-4503-5940-5. Disponível em: <http://doi.acm.org/10.1145/3289600.3290993>

Zhu, S. and Liu, Y. (2008) "Scene segmentation and semantic representation for high-level retrieval". In: IEEE Signal Processing Letters, v. 15, p. 713-716.

Índice de Autores