1

Como as Máquinas Enxergam?

Um apanhado teórico e prático sobre Visão Computacional aplicada a problemas cotidianos utilizando Aprendizado de Máquina e Inteligência Artificial

Lucas Amparo Barbosa, MSc. e Leone da Silva de Jesus, MSc.
Brazilian Institute of Robotics (BIR), SENAI CIMATEC
{lucas.barbosa, leone.jesus}@fieb.org.br

Abstract

This chapter has the main objective to present the basis for image processing and manipulation, your relevant features, and how this information would be used for Machine Learning and Artificial Intelligence solutions. We explained and compared a few relevant and/or recent works in Computer Vision, consolidating the knowledge by simplified practical examples.

Resumo

Este capítulo tem como objetivos apresentar as bases do processamento e manipulação de imagens, suas características relevantes e como essas informações podem ser utilizadas para soluções de Aprendizado de Máquina e Inteligência Artificial. Apresentamos, explicamos e comparamos alguns dos trabalhos mais relevantes e/ou recentes da área de Visão Computacional, consolidando o conhecimento com atividades práticas simplificadas.

1.1. Imagens: Teoria e Aquisição

Visão Computacional (VC) é a área de estudos da Computação que investiga processos capazes de permitir que as máquinas tomem decisões a partir de elementos do seu ambiente. Essa linha de pesquisa recebe esse nome porque, de forma análoga, tenta imitar o processo de visão humana. Portanto, é através dessas técnicas que podemos fazer as máquinas enxergarem. O órgão humano que possibilita a visão humana são os olhos. De forma similar, as câmeras são utilizadas para capturar informações do mundo que serão processadas pelo computador posteriormente. Veja um comparativo na Figura 1.1.

Este modelo básico de câmera recebe um nome: *pinhole*. É constituída, basicamente, por uma pequena abertura por onde a luz vai passar e ser capturada por um

Figura 1.1. Comparativo entre o olho humano e a câmera

material fotossensível, podendo ser um filme fotográfico ou, nas câmeras mais modernas, um sensor eletrônico que gera uma imagem digital.

Outras configurações podem ser relevantes nas câmeras, como a lente e a angulação delas, o tempo que a abertura fica exposta a luz, etc. Tudo isso pode modificar significativamente a imagem gerada no final. Mas, por enquanto, não iremos abordar isso. Para simplificar, vamos adotar as câmeras como o modelo pinhole básico. Assim, poderemos evoluir nossos métodos de aquisição - ou seja, captura de dados - para outros sensores relevantes.

Os seres humanos possuem dois olhos. É graças a posição frontal desses olhos que podemos ter noção de profundidade. O nosso par de olhos nos permite saber se um objeto está está próximo ou distante de nós. Em VC, também é possível simular essa característica: quando posicionamos duas câmeras em um único sistema calibrado, podemos ter informações de profundidade. Esse sistema recebe o nome de visão estéreo.

Quando falamos de um "sistema calibrado" (Figura 1.2), estamos deixando claro que as transformações entre uma câmera e outra são conhecidas. Assim, é possível criar uma correlação entre as duas imagens geradas. Dessa forma, é possível gerar informação útil para ser processada em dois domínios diferentes: A câmera simples gera imagens bidimensionais; O sistema estéreo gera informações tridimensionais.

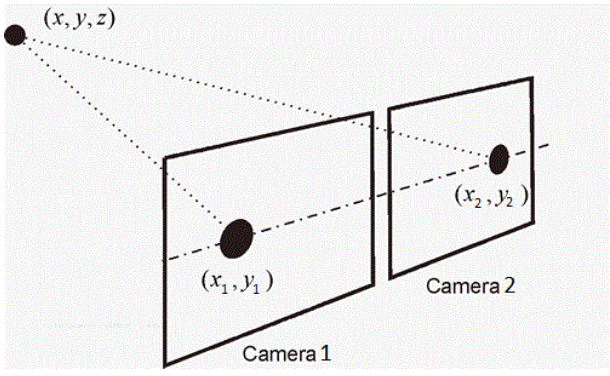

Figura 1.2. Imagens geradas em um sistema calibrado, análogo à visão estéreo. Créditos da imagem: WikiMedia.

Outros tipos de sensores podem ser utilizados para capturar dados em duas dimensões: Sensores infravermelhos podem trazer informação de calor dos objetos e suas formas; Sinais diversos, como eletricidade ou pulsos magnéticos, podem ser processados e interpretados para recuperar informação e a interação de um objeto com o ambiente.

Tabela 1.1. Breve comparativo entre alguns tipos de imagem.

Aquisição	Vantagens	Desvantagens
Coloridas	Ampla variação de cores, similar ao olho humano	Apenas informações bidimensionais
Preto e Branca	Agilidade ao analisar escala de cinza	Único canal de cor; Apenas informações bidimensionais
Infravermelha	Informação de Calor	Apenas informações bidimensionais
Disparidade	Agilidade no processamento de formas geométricas	Baixa precisão da profundidade
Profundidade	Agilidade no processamento de formas geométricas	Fácil oclusão
Nuvens de Ponto	Melhor fonte de informações tridimensionais	Alto custo de processamento

Também existem outras formas de capturar informações tridimensionais: Ondas sonoras podem recuperar posicionamento de objetos (sonares); Projeções de luz podem ser interpretadas para recuperar informação de profundidade; etc.. A vasta gama de possibilidades para aquisição de dados permite um leque igualmente grande de opções a serem investigadas em um projeto que envolva VC. Cada tipo desses possui suas peculiaridades, vantagens e desvantagens. Na Tabela 1.1 podemos ver um breve comparativo entre algumas. A Figura 1.3 apresenta múltiplas capturas com sensores diferentes executadas numa posição próxima, utilizando o sensor RGB-D (cor + profundidade) *Kinect* [1].

Figura 1.3. Imagens geradas pelo *Kinect*. A primeira é uma aquisição por câmera comum. A segunda é um mapa de profundidade, capturando informações da forma do objeto. A última é gerada pelo sensor infravermelho.

Também é possível misturar informações de mais de uma aquisição, gerando dados mais completos. Por exemplo, podemos utilizar as informações de textura da captura de cor gerada por uma câmera em conjunto com a informação de geometria capturada pelo mapa de profundidade, constituindo, no final, uma nuvem de pontos colorida. A Figura 1.4 apresenta o resultado desse processo, ainda utilizando o *Kinect*.

1.2. Filtragem

Uma imagem é uma matriz onde cada campo possui um valor, que são os píxeis. No caso de imagens coloridas, cada píxel possui um valor para cada canal, como para vermelho, azul e verde para imagens RGB, ou tonalidade (*hue*), saturação (*saturation*) e o brilho (*lightness*) para imagens HLS. Um filtro é uma operação que será executada em cada píxel

[1] https://developer.microsoft.com/pt-br/windows/kinect/

Figura 1.4. Imagens geradas pelo *Kinect*. Novamente, vemos o mapa de profundidade a esquerda e a captura da câmera a direita. Na segunda linha, temos dois pontos de vista diferentes da nuvem de pontos gerada com base nas duas imagens superiores.

da imagem. A relevância do filtro está nos cálculos feitos, que não levam em consideração apenas o píxel processado, mas também os píxeis próximos, num conceito de vizinhança. Para executar uma filtragem é necessário aplicar uma matriz (chamada de máscara) sobre a imagem original.

A definição da matriz vai depender do objetivo do filtro e da intensidade a ser aplicada. Por exemplo: Para suavizar uma imagem, ou seja, diminuir as divergências entre os píxeis, eu posso aplicar um filtro de média. Esse efeito é comumente chamado de desfoque (*blur*). Basta simplesmente somar todos os valores da vizinhança e dividir pela quantidade total de píxeis (uma média simples), produzindo uma imagem mais suavizada. Podemos ver um exemplo de borrão na Figura 1.5. Já na Figura 1.6, podemos ver o comparativo entre três métodos de borragem. Além do filtro de média, utilizamos um filtro gaussiano, onde o peso do píxel vai diminuindo conforme ele se afasta do centro da janela, e um filtro bilateral [20], que é ótimo para suavizar imagens sem que as informações de contraste sejam perdidas.

Outros efeitos para filtragem que também podem ser interessantes:

- *Motion Blur*: Simula o efeito de movimento na captura;
- *Sharpening*: Enfatiza divergências na imagem;

Figura 1.5. Imagem original à esquerda. À direita, imagem com aplicação do filtro de média com janela de tamanho 5.

- *Embossing*: Simula ou enfatiza iluminação;
- *Erosion*: Diminui regiões uniformes;
- *Dilation*: Aumenta regiões uniformes.

Figura 1.6. Comparativo de métodos de borragem. A primeira imagem é a original. A segunda, filtrada pela média. A terceira, utilizando um filtro gaussiano. A última, utilizando filtro bilateral.

Esses filtros podem ser utilizados com os mais diferentes objetivos. Por exemplo: observe na Figura 1.7 uma pequena planta. Esta planta possui detalhes em suas folhas que não são simples de serem vistas em imagens comuns. Para tentar enfatizar esses detalhes, você pode aplicar o filtro de *sharpening*, enfatizando as divergências entre píxeis vizinhos, fazendo assim os detalhes da imagem mais visíveis.

Figura 1.7. Aplicação do filtro de enfatização. A esquerda, imagem original. A direita, imagem filtrada.

Outro processo de filtragem muito comum e útil está no par *erosion/dilation*. Com eles é possível preencher buracos na imagem ou unir/separar objetos que estejam muito próximos. Um exemplo pode ser visto na Figura 1.8.

Figura 1.8. Aplicação do processo de dilatação e erosão. Imagem original, imagem dilatada e imagem erodida, respectivamente.

Os filtros não são úteis apenas para recuperar ou enfatizar o que já existe na imagem. Também é possível criar novas informações derivadas da imagem original, como é o caso dos algoritmos de borda. Utilizando filtros, podemos computar o gradiente da imagem, ou seja, a variação de intensidade em píxeis vizinhos. A partir desse gradiente, as bordas da imagem ficam evidentes, podendo ser utilizadas como uma característica relevante da mesma. Várias técnicas podem ser utilizadas para a detecção das bordas. Uma das mais eficientes é a Diferença de Gaussianas, onde você irá filtrar a imagem usando dois filtros gaussianos de tamanho diferente e subtrair os resultados. Um comparativo breve desses métodos podem ser vistos na Figura 1.9.

De forma genérica, quanto mais informação relevante for enfatizada ou gerada na imagem, mais útil ela será para os processos de aprendizado de máquina. Os algoritmos utilizam essas características (em inglês *features*) para executar seus objetivos. Pense novamente no problema da folha, apresentada na Figura 1.7. Caso um especialista precise diferenciar duas plantas, a forma e os detalhes da folha são importantes para a atividade. O mesmo vale para os algoritmos inteligentes: precisamos passar para eles imagens que contenham *features* relevantes para executar a tarefa.

Figura 1.9. Vários métodos para detecção de bordas. No superior, temos a imagem original e o filtro de *Sobel*. No inferior, à esquerda temos o filtro de *Canny* e à direita a Diferença de Gaussianas.

1.3. Pontos Chave

O que identifica um objeto específico? Existem certos objetos que basta ver uma parte dele que já conseguimos dizer do que se trata. Algumas partes dos objetos são extremamente descritivas. Sabendo disso, seria possível levar essa nossa percepção para as máquinas e auxiliá-las nas atividades de reconhecimento de imagens? Sim. Atualmente, isso nem chega a ser um grande desafio. Assim como nós humanos reconhecemos determinados padrões e armazenamos essa informação em nossos cérebros, as máquinas também executam tarefas parecidas.

Esses padrões são compostos por pontos relevantes da imagem: os *keypoints* (em português, pontos-chave). Existem os mais diversos tipos deles: variação de contraste, bordas, repetição de padrão em uma região, etc. Lembra do capítulo anterior? Nele foi possível entender os processos de filtragem e como eles podem modificar as imagens. Muitos desses filtros são utilizados para criar ou enfatizar *keypoints*. Basta lembrar do filtro de *sharpening*, onde muitos detalhes da folha (Figura 1.7) foram evidenciados depois do processo de filtragem.

Entre os algoritmos de detecção de *keypoints* mais famosos, podemos destacar o SIFT, o BRISK e o ORB. O SIFT (*Scale-Invariant Feature Transform*) [14] é um dos mais eficientes processos de detecção de *keypoints*. Contudo, esse algoritmo é patenteado, impossibilitando seu uso em produtos de VC. Como potenciais soluções, podemos utilizar BRISK (*Binary Robust Invariant Scalable Keypoints*) [13], que possui uma qual-

idade semelhante ao SIFT, ou o ORB (*Oriented Fast and Rotated Brief*) [18], que é um dos algoritmos mais rápidos disponíveis. Na Figura 1.10 podemos ver o comparativo do resultado de cada um dos detectores citados.

Existem vários outros, com vantagens e desvantagens, podendo ser aplicados aos mais diversos tipos de imagens. A escolha do detector de pontos-chave vai estar muito ligado ao objetivo final do seu projeto: Vai rodar em tempo real? Tem tolerância a falhas na detecção? Tudo isso é pertinente no momento de escolher.

Figura 1.10. Vários métodos para detecção de *keypoints*. No superior, à esquerda temos a imagem original e à direita, a detecção usando SIFT. No inferior, à esquerda o resultado do BRISK e à direita, o ORB.

1.4. Descrição

No processo de utilizar imagens como dados em algoritmos de inteligência, nós já aprendemos a: melhorar o conteúdo que as imagens contém, através dos filtros; reconhecer pontos relevantes dessas imagens, os *keypoints*. Agora, precisamos padronizar as informações contidas em cada uma dessas figuras para que, assim, possamos comparar umas com as outras.

Esse processo é chamado de descrição. É bem complicado comparar duas imagens com uma numeração de *keypoints* divergente. Assim, os processos de descrição, executados pelos algoritmos descritores, conseguem receber como entrada uma coleção de tamanho irrestrito de pontos e devolver um vetor padronizado com os valores que representem aquele conjunto de informação.

O exemplo mais simples e útil para entender esse processo é feito utilizando o *His-*

togram of Oriented Gradients (HOG) [2]. Esse descritor utiliza a informação de gradiente e a orientação dos píxeis de uma imagem. Por se tratar de um histograma, é possível configurar quantos *bins* (espaços) o seu descritor vai ter. Por fim, basta distribuir os valores nesses espaços.

Outro descritor relevante é o *Local Binary Pattern Histogram* (LBPH). Semelhante ao HOG, ele também é um histograma. Contudo, dessa vez, o valor máximo vai estar relacionado ao tamanho da janela a ser utilizada no processo de binarização da imagem. Visualize exemplos de descrição nas Figuras 1.11 e 1.12.

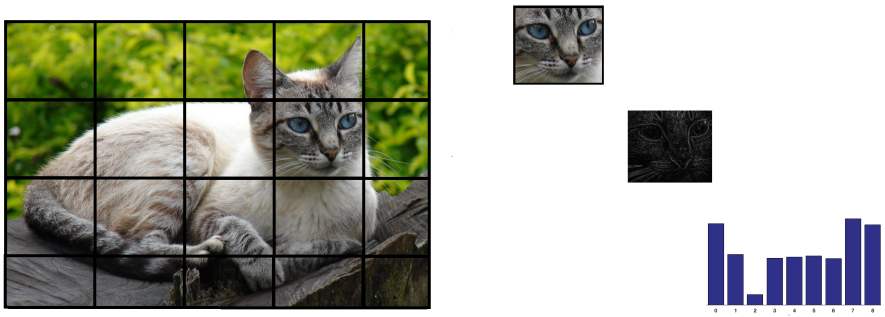

Figura 1.11. Um gatinho sendo descrito através do HOG. Cada janela irá gerar um vetor com base no gradiente de seus pixeis. Os vetores de todas janelas serão concatenados, resultando no vetor de descrição.

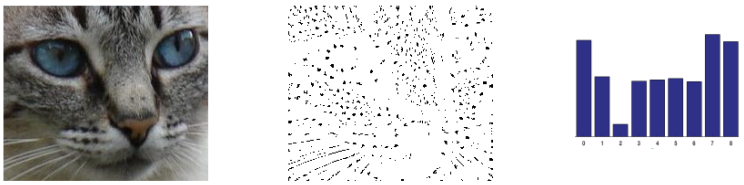

Figura 1.12. Uma das janelas da imagem do gatinho sendo descrita através do LBPH.

Outros descritores relevantes podem ser obtidos através das redes neurais. A técnica mais básica para criar um descritor desta forma é treinar um classificador básico e, durante o processo de descrição, utilizar os valores obtidos na penúltima camada da rede, aquela imediatamente anterior a camada de resultado da classificação. Um exemplo do uso desse tipo de redes será demonstrado mais adiante.

Agora, já que temos um processo padronizado para representar uma imagem, fica bem mais simples de encontrar as que são semelhantes, as que são diferentes e, até mesmo, o quanto diferente uma pode ser da outra. Distâncias bem conhecidas no ensino médio- como Euclidiana e Manhattan, ou mais refinadas, como Mahalanobis- são amplamente utilizadas para isso. No mais, descritores são ferramentas importantes para algoritmos de aprendizado de máquina, principalmente de duas formas:

- Para reduzir a dimensionalidade de um objeto durante o treinamento de um modelo de inferência, pois é bem menos custoso treinar um vetor de 100 valores do que uma imagem inteira de 200×200 píxeis;

- Em processos de validação, como uma classificação ou um agrupamento por similaridade.

1.5. Aprendizado de Máquina e Inteligência Artificial

De forma bem genérica, Inteligência Artificial (IA) são tecnologias que permitem às máquinas reproduzirem inteligência semelhante à humana. Dentro da IA, temos o Aprendizado de Máquina (*Machine Learning*, ML), que é a capacidade de criar modelos matemáticos capazes de detectar padrões em fontes de dados diversas.

Existem duas formas básicas de se treinar esses modelos de inferência: Ou passamos um conjunto de dados rotulados para o modelo aprender, num processo supervisionado de aprendizagem; ou passamos um conjunto desconhecido para que os algoritmos consigam construir os rótulos do dado apresentado, num processo não-supervisionado de aprendizado. Um modelo híbrido é chamado de semi-supervisionado, onde características dos dois podem ser utilizados. A Figura 1.13 ilustra os dois modelos básicos de treinamento.

No treinamento supervisionado, os rótulos informados são utilizados para corrigir o processo de aprendizagem. Assim, quando um novo dado for inserido, o modelo matemático construído anteriormente vai ser capaz de atribuir um rótulo para ele. Já no treinamento não-supervisionado, devido à ausência de rótulos, o modelo precisa extrair informação do que tem "em mãos". Por exemplo, imagine um modelo que seja capaz de classificar um veículo em carro, moto ou ônibus. Caso os rótulos do treinamento não sejam informados, precisamos criá-los. Motos tem 2 rodas, carros tem 4 e ônibus tem, pelo menos, 6. Isso já pode servir de base para criar as bases do classificador. Contudo, como o modelo não tem como deduzir os nomes "moto", "carro" e "ônibus", irá atribuir classe "1", "2" e "3", exigindo do programador que ele, explicitamente, faça a conversão das classes para algo compreensível para humanos (caso seja necessário).

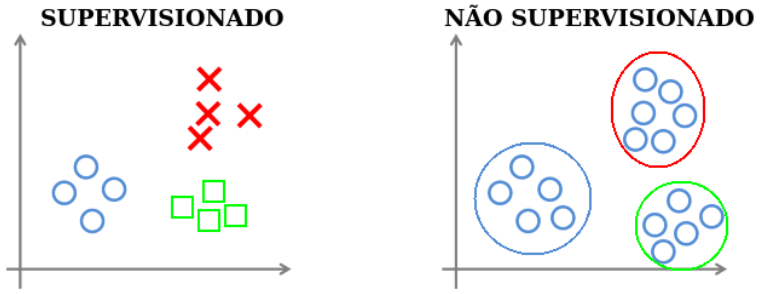

Figura 1.13. Imaginando um problema qualquer de aprendizado. Podemos ter dados já rotulados, como apresentado no gráfico à esquerda, ou podemos ter dados onde não se tem informação, como os apresentados na direita. Uma nova instância de dados tentará resolver o problema usando a informação derivada dos dados inseridos.

Problemas mais simples, que são facilmente detectáveis, como variações de cor ou formato dos objetos, podem ser resolvidos com técnicas menos refinadas, como SVM (*Support Vector Machine*) ou Árvores de Decisão. Quando encontramos problemas mais complexos, que envolvem análise de vizinhança ou coesão temporal, por exemplo, torna-se necessário utilizar técnicas mais refinadas, como as redes neurais e, até mesmo, as

redes profundas.

A dificuldade de um treinamento utilizando redes profundas é que, além de exigir um maior custo computacional para criar o modelo matemático, também é necessário que se tenha um conjunto relevante de dados a ser utilizado no treino. Relevância essa que varia de caso em caso.

Imagine o seguinte problema: um conjunto de imagens que contenham carros e barcos. São dois objetos muito diferentes, em seu formato principalmente. Não há necessidade de utilizar uma técnica muito avançada para resolver esse caso. Agora, imagine que o seu conjunto seja apenas de carros e seu desafio seja classificar o modelo de cada um. Bem mais complexo, pois carros do mesmo estilo tendem a ser parecidos. Por conta disso, precisamos ter um modelo matemático mais robusto e muito mais imagens para treinar. A rede irá aprender a extrair características relevantes de cada modelo e, assim, conseguir acertar a classe de cada um.

1.6. Problemas Comuns: Classificação

Um problema clássico da área de ML é a classificação. Este problema pode ser definido como "A capacidade de separar um conjunto de dados em N classes". Para entender melhor como esse problema é feito, iremos demonstrar um problema relativamente simples: Dado um conjunto de frutas, classificar corretamente cada uma delas. A Figura 1.14 apresenta como um classificador pode funcionar.

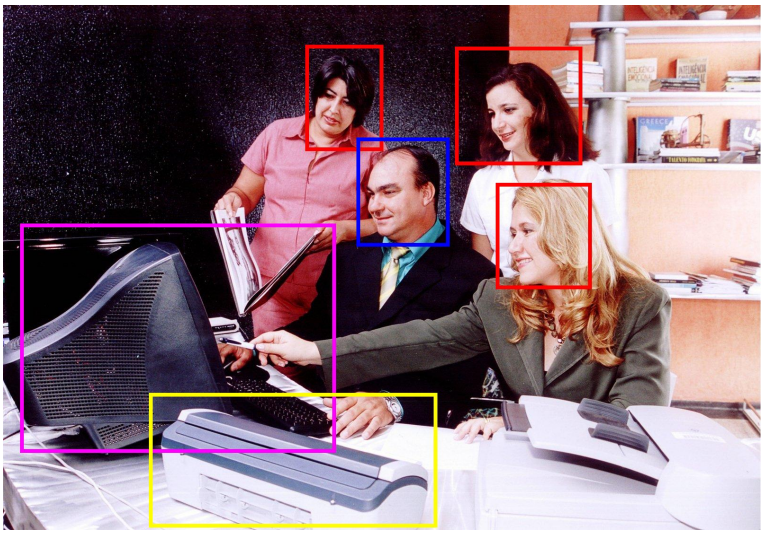

Figura 1.14. Ilustração de um processo de classificação. Em vermelho, mulher. Azul, homem. Rosa, computador. Amarelo, impressora.

O conjunto de imagens é formado por 28 frutas diferentes, como maças, abacaxis, peras, etc. (Veja Figura 1.15), e utilizamos 200 exemplos de cada classe. O objetivo da classificação é: dada uma nova imagem nunca antes vista pelo modelo matemático, atribuir uma classe correta para ele. Vale salientar que a classe da imagem deve existir no conjunto de imagens, porém aquele exemplo não pode ter sido utilizado no processo de treinamento. Uma boa forma para executar o treinamento parte do conjunto de imagens para treinamento e outra parte para a avaliação. Assim, podemos garantir que as classes de

treino e teste serão as mesmas, porém sem reutilização de membros. No nosso exemplo, utilizaremos 70% das imagens para treinamento e 30% para avaliação do modelo. Foram treinados três modelos básicos: um SVM, uma Árvore de Decisão e uma rede neural *Multilayer Perceptron* (MLP).

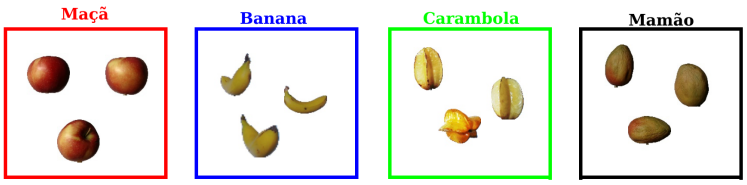

Figura 1.15. Exemplos de frutas presentes no conjunto de imagens.

Podemos então analisar mais a fundo como esse modelo pode ser treinado. Utilizamos a biblioteca *sklearn*, disponível em Python, para simplificar a implementação. Primeiro, carregamos a imagem em escala de cinza e utilizamos o HOG como descritor. Em seguida, informamos para os modelos um conjunto de dados e seus respectivos rótulos. Com esses dados, os modelos irão aprender a separar as classes. Para avaliar a precisão do mesmo, utilizamos outro conjunto de dados, que não foi usado durante o treinamento. Pegamos as predições do modelo para cada uma das imagens de teste e comparamos com a resposta correta. Assim, conseguiremos mensurar o quão bom é a técnica utilizada. A biblioteca *sklearn* já possui uma série de ferramentas para uso e validação de técnicas de aprendizado de máquina. Veja o exemplo no Algoritmo 1.1.

Algoritmo 1.1. Classificadores Simples

```
from sklearn import svm, tree
from sklearn.neural_network import MLPClassifier as mlp
from skimage.feature import hog
import numpy as np

def trainSVM(X_train, y_train):
  sig = svm.SVC(kernel='sigmoid', C=1).fit(X_train, y_train)
  return sig

def trainTree(X_train, y_train):
  sig = tree.DecisionTreeClassifier().fit(X_train, y_train)
  return sig

def trainMLP(X_train, y_train):
  sig = mlp(random_state=1).fit(X_train, y_train)
  return sig

def validModel(model, X_test, y_test):
  y_pred = model.predict(X_test)

  acc = 0.0
  for i in range(len(y_pred)):
    if y_test[i] == y_pred[i]:
      acc += 1

  return acc/len(y_test)

X_train, y_train, X_test, y_test = load_dataset("/path/to/dataset")

msvm = trainSVM(X_train, y_train)
mtree = trainTree(X_train, y_train)
mmlp = trainMLP(X_train, y_train)
```

```
acc_svm = validModel(msvm, X_test, y_test)
acc_tree = validModel(mtree, X_test, y_test)
acc_mlp = validModel(mmlp, X_test, y_test)
```

Os resultados médios são apresentados na Tabela 1.2. Observe que a precisão do SVM é muito baixa, tornando o modelo pouco útil para esse problema. Quando utilizamos a Árvore de decisão, essa métrica já começa a se tornar mais relevante, acertando cerca de 6 a cada 10 tentativas. Porém, o resultado da MLP é, sem sombra de dúvidas, o melhor, praticamente não errando a classificação. Esse comportamento é comum na área e representa o porquê das redes neurais serem tão utilizadas na maioria dos problemas de ML.

Tabela 1.2. Comparativo entre alguns tipos de imagem.

Método	Tempo (s)	Precisão Média
Classificador SVM	$\sim$20	$\sim$40%
Árvore de Decisão	$\sim$4	$\sim$68%
MLP + HOG	$\sim$12	$\sim$99%

1.7. Problemas Comuns: Detecção

Uma especialização dos classificadores é utilizar os seus rótulos gerados para poder encontrar objetos relevantes em imagens complexas. Quando os modelos são treinados, geralmente são utilizadas imagens onde apenas um objeto está inserido. Mas problemas do mundo real não são assim. As imagens do mundo real, conhecidas como capturas *in the wild*, são mais complicadas: Inúmeros objetos, da mesma classe ou não; divergências de iluminação; posicionamento diferente; objetos escondidos ou apenas parcialmente visíveis; etc.. A Figura 1.14, apresentada para exemplificar a classificação, também é útil para entender a detecção.

A detecção é muito utilizada em conjunto com outros problemas pois diminui a quantidade de dados a ser processada por outros processos. Imagine o seguinte problema: Um sistema de reconhecimento facial identifica o rosto de um usuário qualquer. Sendo assim, qual a necessidade de enviar para o processo de identificação o corpo do indivíduo? Podemos utilizar um classificador de rosto humano para detectar uma ou mais faces na imagem, recortá-las e enviar apenas essas partes para o reconhecimento. A Figura 1.16 apresenta uma ilustração do uso da detecção.

1.8. Problemas Comuns: Segmentação

A segmentação é um dos desafios que surgiu na área de VC quando os algoritmos acadêmicos foram transportados do ambiente controlado de laboratório para o mundo real. O problema pode ser definido como a forma de identificar quais pixeis na imagem pertencem a uma (ou várias) classe(s) determinada(s) . Apesar de similar à ideia da detecção, a segmentação visa delimitar precisamente a área da imagem que compõe cada elemento, mesmo que seja somente a diferença entre um objeto e o plano de fundo (Figura 1.17).

Essa segmentação permite, por exemplo, uma melhor análise de elementos na imagem quando o plano de fundo ou o comportamento entre elementos da imagem pode

Figura 1.16. Exemplo da detecção aplicada a rostos humanos. A saída desse processo é muito útil em um sistema de reconhecimento facial.

Figura 1.17. Exemplo da segmentação de carros em uma imagem qualquer.

impactar nos resultados. Existem basicamente dois tipos de segmentação: a semântica (Figura 1.18), que separa pixeis de uma mesma classe na imagem, e de instância (Figura 1.19), que define os pixeis de cada instância conhecida da imagem, mesmo sendo da mesma classe. A disposição dos elementos na imagem ou a forma que eles se relacionam geram informações que não seriam possíveis de adquirir somente com a detecção e classificação (*i.e.* pessoa montada ou atrás da moto, visão de um carro obstruída por outro carro).

O Algoritmo 1.2 é um exemplo de como utilizar a segmentação semântica por meio da biblioteca *píxelLib* [1]. Essa biblioteca permite o uso de forma simples de modelos prontos para segmentação semântica e de instâncias, que estão disponíveis para baixar. O treinamento deses modelos incluiu dezenas de classes distintas, então o uso destes só permite a segmentação dessas classes. Contudo, a biblioteca também oferece meios de adicionar classes personalizadas, também de forma simples, contanto que seja informado um conjunto de imagens e suas marcações seguindo uma estrutura de diretórios pré-estabelecida.

Algoritmo 1.2. Segmentador Simples

```
import cv2
import p xellib
from p xellib.semantic import semantic_segmentation
```

[1] https://github.com/ayoolaolafenwa/píxelLib

Figura 1.18. A imagem possui diferentes classes, representadas por pixeis de diferentes cores (segmentação semântica).

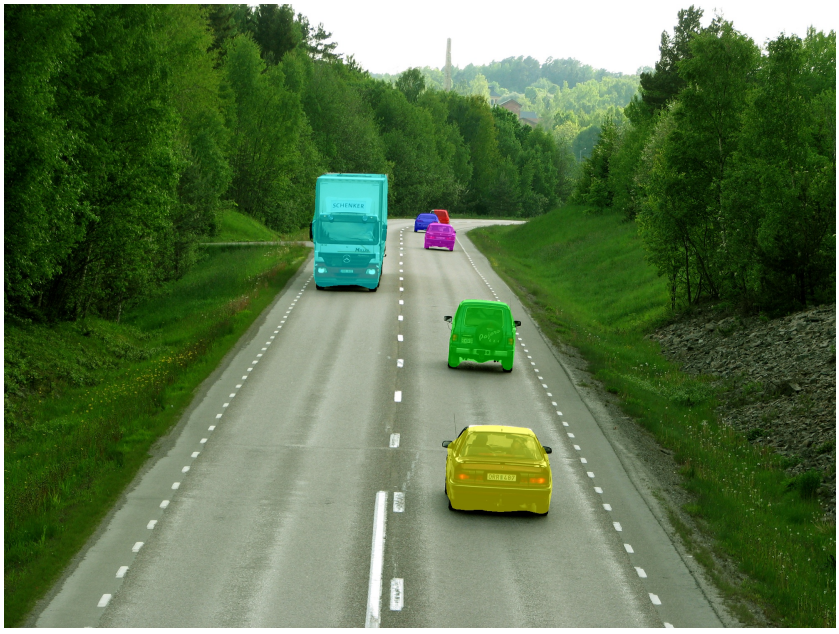

Figura 1.19. Objetos da mesma classe (veículo), mas cada um representado por pixeis de uma cor distinta (segmentação de instâncias).

```
def segmentacao_semantica(caminho_imagem):
  segmentador = semantic_segmentation()
  segmentador.load_pascalvoc_model("deeplabv3_xception_tf_dim_ordering_tf_kernels.h5")
  _, area_segmentada = segmentador.segmentAsPascalvoc(caminho_imagem)
  cv2.imwrite("area_segmentada.png", area_segmentada)
  return area_segmentada

caminho_imagem = "imagem_com_fundo.png"
imagem_original = cv2.imread(caminho_imagem)
fundo = cv2.imread('fundo.png')
area_segmentada = segmentacao_semantica(caminho_imagem)
```

```
linhas,colunas,canais = area_segmentada.shape
roi = fundo[0:linhas, 0:colunas]
cinza = cv2.cvtColor(area_segmentada, cv2.COLOR_BGR2GRAY)

ret, mascara = cv2.threshold(cinza, 10, 255, cv2.THRESH_BINARY)
mascara_inversa = cv2.bitwise_not(mascara)

recorte_fundo = cv2.bitwise_and(roi, roi, mask = mascara_inversa)
recorte_mascara = cv2.bitwise_and(imagem_original, imagem_original, mask = mascara)

montagem = cv2.add(recorte_fundo, recorte_mascara)
fundo[0:linhas, 0:colunas ] = montagem
cv2.imwrite("resultado.png", fundo)
```

Além do Algoritmo 1.2 demonstrar o fácil uso da biblioteca de segmentação, ela também evidencia que o conhecimento sobre VC é essencial para o desenvolvimento da sua própria aplicação. Com poucos comandos do OpenCV [1], conseguimos utilizar a área segmentada de um ser humano e aplicar um diferente plano de fundo. A Figura 1.20 mostra os resultados. Essa aplicação só era utilizada no passado com planos de fundos de cores únicas e específicas, mas utilizando modelos de redes neurais para segmentação, que já é um problema bem consolidado na literatura, é possível obter resultados com humanos em qualquer plano de fundo comum.

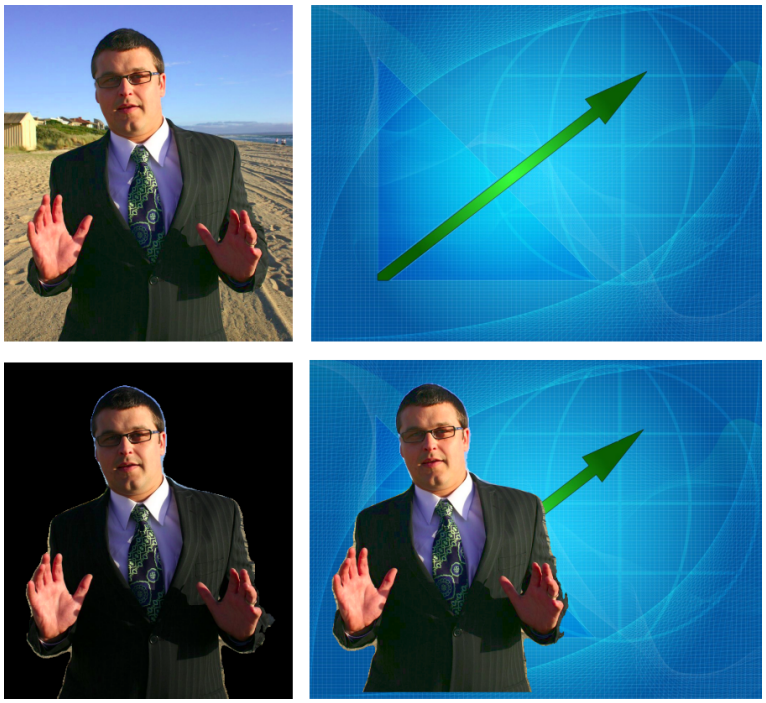

Figura 1.20. Em cima, imagens utilizadas como entrada do código. Em baixo, a imagem do humano segmentada e a montagem com um novo plano de fundo.

1.9. Um exemplo prático: Reconhecimento Biométrico

Há anos que o reconhecimento biométrico é estudado para diversos fins, especialmente para controle de acesso, seja de instalações militares ou dispositivos móveis, podendo proteger desde o acesso ao uso de recursos perigosos a informações pessoais. Diferentes

[1] https://opencv.org/

características físicas e comportamentais tem sido experimentadas na literatura, mas as mais distintivas e de mais fácil coleta (*e.g* face, impressão digital e íris) [11] foram as mais exploradas, possuindo níveis de confiança aceitáveis até mesmo para modelos já publicamente disponíveis. O reconhecimento de faces, por exemplo, apesar de ainda ser alvo de pesquisas, já superou até mesmo a capacidade humana nessa tarefa em ambientes não controlados [23], e vários modelos podem ser encontrados para o fácil uso de quem tem um mínimo conhecimento de VC. O Algoritmo 1.3 é um exemplo simples de uma aplicação que faz verificação facial.

Algoritmo 1.3. Verificação Facial

```
import cv2
import numpy
from mtcnn.mtcnn import MTCNN
from keras.models import load_model

def le_imagem(nome_arquivo):
  img = cv2.imread(nome_arquivo)
  return img

def recorta_face(imagem, detector, margem):
  ponto_x, ponto_y, largura, altura = detector['box']
  ponto_x -= margem
  ponto_y -= margem
  largura += 2*margem
  altura += 2*margem
  if ponto_x < 0:
    largura += ponto_x
    ponto_x = 0
  if ponto_y < 0:
    altura += ponto_y
    ponto_y = 0
  return imagem[ponto_y:ponto_y+altura, ponto_x:ponto_x+largura]

def detecta_recorta(modelo_deteccao, imagem):
  faces_detectadas = modelo_deteccao.detect_faces(imagem)[0]
  margem = int(0.1 * imagem.shape[0])
  detecta_recorta = recorta_face(imagem, faces_detectadas, margem)
  return detecta_recorta

def pre_processsamento(face, required_size=(160, 160)):
  ret = cv2.resize(face, required_size)
  ret = ret.astype('float32')
  mean, std = ret.mean(), ret.std()
  ret = (ret - mean) / std
  return ret

def extrai_caracteristicas(amostras):
  amostras = numpy.asarray(amostras, 'float32')
  model = load_model('facenet_keras.h5')
  yhat = model.predict(amostras)
  return yhat

def verificacao (nome_imagem, confianca, imagem):
  imagem = cv2.resize(imagem, (500, 500))
  local_texto = (50, 100)
  fonte = cv2.FONT_HERSHEY_SIMPLEX
  tamanho_fonte = 2
  largura = 5
  limiar = 10
  if confianca < limiar:
    imagem = cv2.putText(imagem, str(confianca),
        local_texto, fonte, tamanho_fonte, (0, 255, 0), largura)
  else:
    imagem = cv2.putText(imagem, str(confianca),
        local_texto, fonte, tamanho_fonte, (0, 0, 255), largura)
  cv2.imwrite(nome_imagem, imagem)
```

```
trump1 = le_imagem("Trump.jpg")
trump2 = le_imagem("Trump_2.jpg")
clinton = le_imagem("Clinton.jpg")
obama = le_imagem("Obama.jpg")

modelo_deteccao = MTCNN()
trump1 = detecta_recorta(modelo_deteccao, trump1)
trump2 = detecta_recorta(modelo_deteccao, trump2)
clinton = detecta_recorta(modelo_deteccao, clinton)
obama = detecta_recorta(modelo_deteccao, obama)

imagens = [trump1, trump2, clinton, obama]
amostras = [pre_processsamento(i) for i in imagens]

caracteristicas = extrai_caracteristicas(amostras)

confianca_trump_trump = numpy.linalg.norm(caracteristicas[0] - caracteristicas[1])
confianca_trump_clinton = numpy.linalg.norm(caracteristicas[0] - caracteristicas[2])
confianca_trump_obama = numpy.linalg.norm(caracteristicas[2] - caracteristicas[3])

verificacao("TrumpxTrump.png", confianca_trump_trump, trump2)
verificacao("TrumpxClinton.png", confianca_trump_clinton, clinton)
verificacao("TrumpxObama.png", confianca_trump_obama, obama)
```

Todos os passos para esse reconhecimento de faces foram explicados previamente, mesmo que de forma diferente. Começando pela detecção fácil, aqui feita utilizando um modelo bem conhecido para o problema chamado MTCNN [22]. A imagem da face, extraída da original, passa por um processamento para ficar com as mesmas características das imagens que foram utilizadas no treino da rede de reconhecimento facial. Tal rede vem de um modelo também bem conhecido para transformar imagens em uma representação de seus principais componentes, como explicado na Seção 1.4. Esse processo é também chamado de extração de características. A comparação entre essas características, utilizando distâncias descritas previamente, retorna um valor que representa confiança do seu método de que as características pertencem à mesma pessoa. Então a aplicação só precisa decidir se aquele valor é suficiente para validar a verificação facial. Esse processo é a base para todos os reconhecimentos biométricos, seja para verificação ou identificação, e seu fluxo está presente na Figura 1.21.

Os resultados presentes na Figura 1.22 demonstram a eficácia dos modelos utilizado tanto para detecção da face quanto para o reconhecimento. Isso é mais uma demonstração de que para aplicações já pesquisadas há muito tempo, alguém na comunidade científica já teve o esforço de treinar um modelo eficaz, mas o conhecimento mínimo de VC é essencial para você saber como adaptá-los à sua aplicação.

1.10. O que há de bom na Literatura?

Todas as técnicas apresentadas e as que, por ventura, sejam conhecidas após a leitura desse material, estão publicadas nas mais diversas fontes: IEEEXplore [1], Nature [2], Elsevier [3], etc.. Contudo, o objetivo aqui é entender como os processos de visão computacional podem ser aplicados, em conjunto com a ML e IA, para resolver problemas da área in-

[1] https://ieeexplore.ieee.org/

[2] https://www.nature.com/

[3] https://www.elsevier.com/pt-br

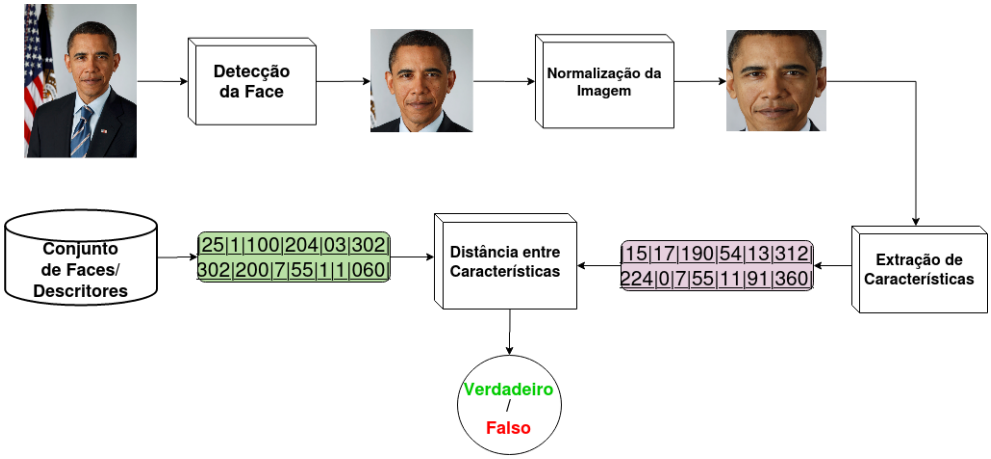

Figura 1.21. Diagrama dos passos realizados no Algoritmo 1.3. Esse é o processo básico de aplicações de biometria. Nesse caso, temos uma verificação facial. Para a identificação biométrica, o processo é similar, mas os descritores extraídos são comparadas com os descritores de todas as pessoas conhecidas pela base, considerando a de maior similaridade como a pessoa identificada. Originais retiradas de WikiMedia.

Figura 1.22. No Algoritmo 1.3 realizamos uma comparação entre uma face e três outras. A verificação diz se são pessoas distintas, colocando o resultado de sua confiança de que são a mesma pessoa em verde (se positivo) e vermelho (se negativo). Numa aplicação real, o resultado dessa verificação poderia ser utilizada facilmente em diversas aplicações em que se tivesse uma câmera. Originais retiradas de WikiMedia.

dustrial. Por isso, selecionamos alguns trabalhos relevantes e interessantes para diversas áreas de atuação: agricultura, energia, alimentação, etc..

Um sistema de inspeção visual é composto, basicamente, por uma fonte de iluminação (natural ou não), um dispositivo de captura de imagem (independente de qual tipo) e poder computacional para processar os dados, transformando a imagem em informação

relevante. Quando bem parametrizados, esses sistemas permitem uma maior confiabilidade e repetibilidade nos processos de avaliação de qualquer objeto que seja necessário.

A revisão literária apresentada por Gomes & Leta (2012) [6] demonstra como as técnicas de visão computacional são utilizadas há anos para aprimorar os processos de inspeção nas mais diversas indústrias alimentares. Existem trabalhos para controle de pestes, diminuindo custos com pesticidas e permitindo uma medida preventiva antes que uma maior parcela da produção esteja comprometida. Também foi possível executar medições de quanto do campo estava sendo utilizado com plantações úteis ou ervas daninhas e, assim, mensurar a área útil e produtiva da propriedade. Os problemas encontrados nessas abordagens esbarram em iluminações divergentes, a depender da hora do dia ou do nível de maturação da plantação.

Ainda analisando a revisão de Gomes & Leta, também é possível perceber que os processos de visão computacional podem auxiliar na detecção e classificação de defeitos nos mais diversos tipos de vegetais, como batatas ou maçãs. Análises em formato, tamanho, cor e uniformidade são feitas para identificar produtos de alta ou baixa qualidade, permitindo uma melhor separação e precificação do que será entregue aos clientes.

Não foram somente as lavouras foram beneficiadas por essas evoluções tecnológicas. Os processos industriais também podem ser otimizados por meio da computação. A revisão da literatura trouxe uma pesquisa que utilizava informações espectrais em conjunto com imagens capturadas para avaliar a qualidade de vinhos produzidos, fazendo assim dois métodos de aquisição diferentes trabalharem em conjunto.

Assim como a otimização dos processos é relevante, também é a tentativa de transformar os produtos resultantes em algo mais aprazível aos olhos humanos. É o caso de utilizar câmeras para avaliar a distribuição de ingredientes em pizzas ou treinar modelos matemáticos que auxiliem as máquinas a montarem pratos de refeições prontas que estejam mais parecidas com os pratos montados por humanos, também listados na publicação de Gomes & Leta.

Saindo um pouco da área alimentícia, o trabalho apresentado por Fathi, Dai & Lourakis (2015) [3] demonstra o uso da VC na área de Engenharia Civil. A publicação apresenta uma revisão das bases técnicas que possibilitam a reconstrução 3D de infraestruturas já existentes, possibilitando assim as mais diversas análises como a necessidade de reformas estruturais ou simular o efeito de forças adversas na construção. Também são citados no artigo os principais desafios encontrados nesse tipo de pesquisa, como o custo computacional de construção do objeto 3D ou a forte dependência que o resultado tem de características distintivas em toda a extensão do alvo da reconstrução. Esse trabalho é interessante por trazer a tona tanto as qualidades quanto os desafios do uso de tal tecnologia.

Algumas empresas de construção civil já fazem o uso de modelagem tridimensional durante a execução dos seus projetos (Modelo BIM [16], por exemplo), porém fazer a digitalização de estruturas já construídas, normalmente chamadas de *as-built*, ainda é um problema desafiador para a área. Outras abordagens úteis à área de Construção Civil está na detecção, classificação, avaliação e monitoramento de situações adversas no alvo de inspeção. Essas adversidades podem ser defeitos relevantes, como rachaduras em as-

falto [12] ou falhas estruturais em construções [4, 19].

Na área de geração e distribuição de energia elétrica, Mirallès, Pouliot & Montambault (2014) [15] apresentaram uma revisão da literatura com ênfase na gestão de linhas de transmissão: objetos muito grandes, caros e fundamentais para o funcionamento do negócio da empresa. Detecção e Inspeção desses objetos (presença de ferrugem, ausência de peças, etc.) são amplamente discutidos na publicação, demonstrando como a VC pode contribuir na manutenção de uma indústria vital para a sociedade moderna.

Trazendo nossa leitura para problemas mais "modernos", a Indústria 4.0 está intimamente ligada à automatização de processos com suporte de IA/ML e estas, intimamente ligadas com Visão Computacional. Villalba-Diez et al. (2019) [21] nos apresenta um trabalho utilizando técnicas de aprendizado profundo para monitorar e garantir a qualidade nos processos de Impressão 3D, agregando valor ao produto final produzido e reduzindo substancialmente os custos da produção.

Todos esses trabalhos foram selecionados apenas pela ênfase aplicada a problemas industriais. As linhas de pesquisa derivadas da VC são muito grandes, com vários problemas ainda em aberto e um potencial pouco explorado, principalmente pelo mercado brasileiro. Se fossemos analisar outras pesquisas desenvolvidas e relevantes, precisaríamos de uma especialização só para isso. O importante no momento é saber da existência dessa série de trabalhos e toda a sua aplicabilidade para problemas industriais. Na próxima seção, algumas ideias serão imaginadas. Um dia elas serão desenvolvidas? Só o tempo dirá.

1.11. O que o futuro nos reserva?

O futuro da Indústria está atrelado ao termo "4.0". Esse termo foi cunhado para descrever uma possível quarta revolução industrial, onde os processos serão desenvolvidos utilizando tecnologias que permitam a fusão entre o mundo físico, digital e biológico. Como? Através da Manufatura Aditiva (Impressão 3D), Inteligência Artificial, Internet das Coisas, Biologia Sintética e Sistemas Cyber-físicos. A grande sacada dessa nova revolução está na interação entre as partes. Sendo assim, é necessário que algumas informações do mundo real sejam inseridas no mundo digital. E como isso pode ser feito? Utilizando técnicas de Visão Computacional.

Vamos pensar na linha da Manufatura Aditiva. Para construir um novo objeto que será impresso, é necessária a edição dele utilizando um software. Porém, caso já exista algo similar no mundo real, esse objeto pode ser escaneado e carregado no sistema, agilizando o trabalho do designer responsável, que poderá focar apenas nas melhorias a serem desenvolvidas. Uma vez que o objeto esteja pronto, no mundo digital, ele será impresso de fato. Durante o processo, técnicas de monitoramento podem ser aplicadas para impedir que erros de grande escala aconteçam. Métodos de inspeção podem ser utilizados no produto final para validar se o objeto impresso é realmente aquilo que foi projetado.

No contexto de Internet das Coisas (IoT), que consiste em trazer o poder de tomar algumas decisões para as "pontas" das estruturas, descentralizando o processamento de informações, as coisas ficam um pouco mais interessantes. Podemos pensar em um prob-

lema relativamente simples para aplicar: Imagine uma caldeira química, que trabalhe sempre em uma temperatura qualquer. Uma câmera térmica, instalada próxima a essa caldeira, pode auxiliar nos processos de segurança, permitindo o monitoramento da temperatura. A tomada de decisão resultante da análise dessa imagem térmica não precisa ser centralizada, pois, uma vez informado que material está sendo "fervido" na caldeira, um processador simples é capaz de perceber se o aumento de temperatura, dada uma janela de tempo, é perigosa ou não.

Dos processos da Indústria 4.0, os Sistemas Cyber-físicos são, provavelmente, os mais desenvolvidos na atualidade. Já temos, por exemplo, braços robóticos que executam as mais diversas tarefas, como montagem de equipamentos ou coleta de materiais. Porém, ainda existem muitas outras aplicabilidades. Colheita automatizada, onde um robô é capaz de analisar se um vegetal está ou não em ponto de colheita, executando o processo com alta precisão e em um intervalo de tempo muito mais curto.

Todas essas linhas podem utilizar, como base, processos de Inteligência Artificial. Quanto mais automático o processo for, menor a interação humana. E, ao diminuir essa interação, há a necessidade de trazer dados do mundo real para o computador. Se a tendência industrial é a automação, não há dúvidas de como o conhecimento em Visão Computacional vai se tornar cada vez mais relevante e aplicável nas mais diversas atividades industriais. Que processo não precisa de inspeção e monitoramento? Todos os produtos não precisam ter sua qualidade analisada?

Referências

[1] Baggio, D. L. et al. (2017) "Mastering OpenCV 3". Packt Publishing.

[2] Dalal N. e Triggs B.. (2005) "Histograms of oriented gradients for human detection," In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893.

[3] Fathi, H., Dai, F. e Lourakis, M.. (2015). "Automated as-built 3D reconstruction of civil infrastructure using computer vision: Achievements, opportunities, and challenges" In: Advanced Engineering Informatics, 29(2), p. 149–161.

[4] Feng, D. e Feng, M. Q. (2018) "Computer vision for SHM of civil infrastructure: From dynamic response measurement to damage detection – A review" In: Engineering Structures, Volume 156, p. 105-117.

[5] Garreta, R. et al. (2017) "Scikit-learn : Machine Learning Simplified: Learning Path". Packt Publishing.

[6] Gomes, J. F. S., e Leta, F. R. (2012) "Applications of computer vision techniques in the agriculture and food industry: a review". In: European Food Research and Technology 235, p. 989–1000.

[7] Gonzalez, R. C. e Woods, R. E. (2007) "Digital Image Processing". Pearson.

[8] Goodfellow, I. et al. (2015) "Deep Learning". MIT Press.

[9] Gulli, A. e Pal, S.. (2017) "Deep Learning with Keras". Packt Publishing.

[10] Howse, J. et al. (2016) "OpenCV: Computer Vision Projects with Python: Learning Path". Packt Publishing.

[11] Jain, A. K., Flynn, P. e Ross, A. A.. (2007) "Handbook of Biometrics". Springer-Verlag New York, Inc.

[12] Koch, C. et al. (2015) "A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure" In: Advanced Engineering Informatics, Volume 29, Issue 2, p. 196-210.

[13] Leutenegger, S., Chli M. e Siegwart R. Y.. (2011) "BRISK: Binary Robust invariant scalable keypoints," In: International Conference on Computer Vision, Barcelona, 2011, pp. 2548-2555.

[14] Lowe, D. G.. (2004) "Distinctive Image Features from Scale-Invariant Keypoints" In: International Journal of Computer Vision 60, p. 91–110.

[15] Miralles, F., Pouliot, N., e Montambault, S.. (2014). "State-of-the-art review of computer vision for the management of power transmission lines" In: Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry.

[16] Mohandes, S. R. e Omrany, H.. (2013). "Building Information Modeling in Construction Industry: Review Paper" In: International Conference of Seminar Kebangsaan Aplikasi Sains & Mathematic (SKASM).

[17] Rodrigues, V.. (2018) "Como Máquinas Aprendem: Fundamentos E Algoritmos de Machine Learning, Redes Neurais E Deep Learning". Independently Published.

[18] Rublee, E. et al.. (2011). "ORB: an efficient alternative to SIFT or SURF" In: Proceedings of the IEEE International Conference on Computer Vision. 2564-2571.

[19] Spencer, B. F., Hoskere, V. e Narazaki, Y.. (2019) "Advances in Computer Vision-Based Civil Infrastructure Inspection and Monitoring" In: Engineering, Volume 5, Issue 2, p. 199-222.

[20] Tomasi, C. e Manduchi, R.. (1998) "Bilateral filtering for gray and color images," In: Sixth International Conference on Computer Vision. p. 839-846.

[21] Villalba-Diez, J. et al.. (2019). "Deep Learning for Industrial Computer Vision Quality Control in the Printing Industry 4.0" In: Sensors, 19(18), 3987.

[22] Zhang, K. et al.. (2016) "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks". CoRR (ArXiv).

[23] Zheng, T., Deng, W. e Hu, J.. (2018) "Cross-Pose LFW: A Database for Studying Cross-Pose Face Recognition in Unconstrained Environments". CoRR (ArXiv).