Capítulo 3

Explicando as decisões com IAs: Demonstrando sua aplicação em imagens médicas

Elineide Silva dos Santos, Justino Duarte Santos, Luis Henrique Silva Vogado, Leonardo Pereira de Sousa, Hélcio de Abreu Soares and Rodrigo de Melo Souza Veras

Abstract

It is already part of everyday being surrounded by technologies directed by Artificial Intelligences, from personal assistants to facial recognition. In the context of medicine is no different. It is remarkable the increasing insertion of technologies to support the diagnosis of diseases. However, in this context it is of paramount importance to take a step further: the explanation of the decisions taken by the AI models, especially those based on models of Deep Neural Networks (DNN). Achieving confidence and finding justification in the prediction of a DNN can hardly be achieved if the user does not have access to a satisfactory explanation for the process that led to its exit. In this short course, we will address theoretical aspects about explicability, its importance in the medical context, and we will know several methods of explanation of DNN models.

Resumo

Já faz parte do cotidiano sermos rodeados de tecnologias dirigidas por Inteligências Artificiais, desde assistentes pessoais até reconhecimentos faciais. No contexto da medicina não é diferente. É notável a crescente inserção de tecnologias de apoio ao diagnóstico de doenças. Entretanto, neste contexto é de suma importância dar um passo além: a explicação das decisões tomadas pelos modelos de IA, em especial as baseadas em modelos de Redes Neurais Profundas (DNN). Alcançar a confiança e encontrar justificativa na predição de uma DNN dificilmente pode ser alcançado se o usuário não tiver acesso a uma explicação satisfatória para o processo que levou à sua saída. Neste minicurso, abordaremos aspectos teóricos sobre explicabilidade, sua importância no contexto médico, e conheceremos vários métodos de explicação de modelos DNN.

3.1. Introdução

Ao processar uma imagem é necessário seguir um conjunto de etapas, pois processamento digital de imagens não é uma tarefa trivial. A etapa inicial consiste na captura da imagem por meio de um sistema de aquisição, o qual captura e digitaliza uma cena para formato específico. A imagem resultante de um processo de digitalização, precisa ser representada de uma forma apropriada para uso computacional. Elas podem ser representadas em duas ou mais dimensões [Acharya and Ray 2005].

Atualmente, a Inteligência Artificial (IA) tenta resolver tarefas difíceis de serem descritas formalmente, por exemplo reconhecimento de fala, padrões e imagens, ou seja, ligados a intuição humana. Trata-se de um grande desafio para um computador a habilidade de entender conhecimentos passados e predizer/resolver problemas futuros. Nesse contexto, surgiu uma subárea da IA, o Aprendizado de Máquina (*Machine Learning* - ML) capaz de utilizar métodos computacionais para tratar esse desafio.

Em ML, a aprendizagem é feita por meio de treinamentos em banco de dados, que representam eventos e experiências passadas, possibilitando a construção de sistemas capazes de aprender de forma automática [Dundas and Chik 2011]. Uma área de aplicação da ML é a classificação de imagens, no qual quanto maior a quantidade de exemplos, maior a capacidade de predição dos classificadores gerados. Porém, em cenários reais, dispor de bases de dados com uma grande quantidade de exemplos rotulados nem sempre é fácil, na maioria das vezes, custosa.

Desse modo, podemos utilizar abordagens de *Data Augmentation*, em síntese compreendem técnicas computacionais com o objetivo de aumentar a quantidade de exemplos rotulados em um conjunto de dados e assim, melhorar os resultados obtidos [Taylor and Nitschke 2018]. O uso de *Data Augmentation* é evidente em algumas aplicações da classificação de imagens, como aplicações com imagens aéreas e médicas cujo os dados são limitados.

Uma das técnicas de aprendizado de máquina amplamente utilizada é o *Deep Learning*. Desenvolvida a partir de redes neurais artificiais que são capazes de reconhecer e classificar padrões. Atualmente, as redes neurais profundas (*Deep Neural Networks* - DNNs) auxiliam nos avanços de visão computacional, tem sua estrutura baseada em redes neurais artificiais, porém, as DNNs são modelos extremamente complexos, com milhões de parâmetros, e são conhecidos pela alta capacidade de ocorrer *overfitting*, devido à demanda por enormes quantidades de dados durante o treinamento e pela dificuldade em oferecer interpretabilidade dos modelos.

Modelos como regressões lineares, regressões logísticas, árvores de decisões e *KNearest Neighbors* (KNN) são de fácil entendimento, ainda que simples. Por outro lado, modelos baseados em redes neurais artificiais e, em particular, redes convolucionais, não têm explicação natural para seus resultados. Tais modelos são compostos por uma enorme quantidade de parâmetros que se relacionam de várias maneiras.

Apesar de redes convolucionais serem chamadas de modelos caixas pretas no contexto de interpretabilidade, isso não quer dizer que não há nenhuma maneira de investigar os mecanismos internos destes modelos. Pode-se empregar técnicas de interpretabilidade que são *model-agnostic* (ou que não dependem do modelo), como, por exemplo,

LIME, que utiliza a imagem de entrada para identificar regiões importantes para o modelo. Outras técnicas foram desenvolvidas especialmente para lidar com redes convolucionais, como CAM, Grad-CAM, *Guided Backpropagation*, dentre outras.

3.1.1. Representação Computacional de Imagens

Segundo Gomes e Velho [Gomes and Velho 1997], um modelo matemático é necessário para representação e manipulação de imagens no computador. Considerando que as imagens são captadas por meio da luz emitida pelos objetos, pode-se estabelecer um modelo abstrato de imagens visando a representação discreta e futura codificação para uma linguagem compreendida pelo computador.

No contexto matemático, uma imagem é uma função bidimensional $f(a,b)$ onde a e b são coordenadas planas, e a amplitude de f em qualquer par de coordenadas (a,b) é chamada de intensidade ou nível de cinza da imagem para um determinado ponto. Nesse contexto, quando tanto os valores a, b quanto o valor da amplitude f compreendem um conjunto finito, denota uma imagem digital [Gonzalez et al. 2002].

De modo geral, uma imagem digital pode ser descrita como uma matriz $m \times n$, onde cada parte da matriz equivale a intensidade (brilho) ou nível de cinza $f(a,b)$, essas partes são denominadas os *pixels* de uma imagem. Em imagens binárias os *pixels* tem valores 0 e 1, enquanto em imagens em tons de cinza, seus valores podem variar entre 0 e 255. Já uma imagem colorida, esse *pixel* é representado por três valores que variam de 0 a 255.

Em imagens coloridas que possuem múltiplas bandas de frequências, existe uma função de intensidade de brilho associada a cada uma das banda de frequências. A Figura 3.1 mostra um exemplo das bandas presentes no padrão RGB (Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*), o mais popular utilizado.

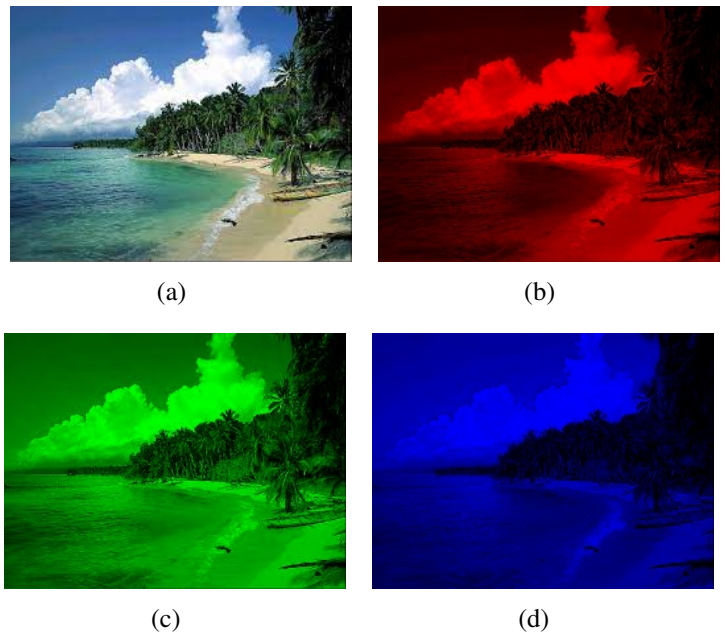


Figura 3.1. Imagens coloridas no padrão RGB.(a) RGB, (b) Vermelho, (c) Verde, (d) Azul.

Dessa maneira, na Figura 3.1 é possível notar que uma imagem colorida consiste em três imagens monocromáticas e a sobreposição dessas imagens compõem a imagem colorida, inclusive a mistura de cada uma dessas cores obtém-se as demais cores.

3.1.2. Técnicas de *Data Augmentation*

Dados limitados são um grande obstáculo na aplicação de modelos de aprendizagem profunda, como redes neurais convolucionais. Frequentemente, classes desequilibradas podem ser um obstáculo adicional, embora possa haver dados suficientes para algumas classes, igualmente importantes, mas classes abaixo da amostra sofrerão com baixa precisão específica da classe. Esse fenômeno é intuitivo. Se o modelo aprender com alguns exemplos de uma determinada classe, é menos provável prever a invalidação da classe e os aplicativos de teste [Shorten and Khoshgoftaar 2019].

As primeiras demonstrações que mostram a eficácia das técnicas de *Data augmentation* vêm de transformações simples, como inversão horizontal, aumento de espaço de cores e corte aleatório. Essas transformações codificam muitas das invariâncias discutidas anteriormente que apresentam desafios para as tarefas de reconhecimento de imagem [Shorten and Khoshgoftaar 2019].

3.1.2.1. Transformações Geométricas

Esta seção descreve técnicas capazes de modificar a forma, tamanho e posição dos componentes presentes dentro da imagem. Todos esses métodos são fundamentados em transformações geométricas e têm como característica principal a fácil implementação.

Flipping: Essa técnica faz uma inversão na imagem original, tal inversão pode ser na horizontal ou na vertical. Em geral, inverter o eixo horizontal é mais comum do que inverter o eixo vertical. Esse aumento é um dos mais fáceis de implementar e provou ser útil em conjuntos de dados como MNIST, ImageNet e CIFAR-10.

Rotação: É uma técnica que realiza os aumentos girando a imagem para a direita ou esquerda em um eixo entre 1° e 359°. O parâmetro do grau de rotação é responsável por garantir a segurança dos aumentos realizados pela rotação. Um detalhe significativo consiste em medida que o grau de rotação aumenta, os rótulos dos dados não são mantidos após a transformação [Shorten and Khoshgoftaar 2019].

Translação: Essa técnica pode mudar as imagens para a esquerda, direita, para cima ou para baixo. Essa característica pode ser vantajoso em conjuntos com distorções posicionais. Por exemplo, em um determinado conjunto de imagens centralizadas, o modelo seria mais eficiente se também fosse testado em imagens devidamente centralizadas. Nesse caso, a translação preserva as dimensões espaciais por meio do preenchimento com um valor constante (0 ou 255) ou um ruído (aleatório, gaussiano), o espaço restante proveniente da aplicação da técnica sobre a imagem.

Zoom / escala: Um zoom aleatório é obtido pelo argumento *zoom_range*. Um zoom menor que 1.0 amplia a imagem, enquanto um zoom maior que 1.0 diminui o zoom da imagem.

***Shear* / cisalhamento:** Essa transformação inclina a forma da imagem. O con-

traste entre rotação e cisalhamento refere-se fixação de um eixo, durante a aplicação da transformação de cisalhamento. Em seguida a imagem é esticada até um determinado ângulo conhecido como ângulo de cisalhamento, então, o resultado alcançado é uma espécie de "alongamento" dessa imagem, característica ausente quando trata-se da rotação.

A Figura 3.2 mostra exemplos da aplicação do aumento de dados sobre a imagem de um gato.

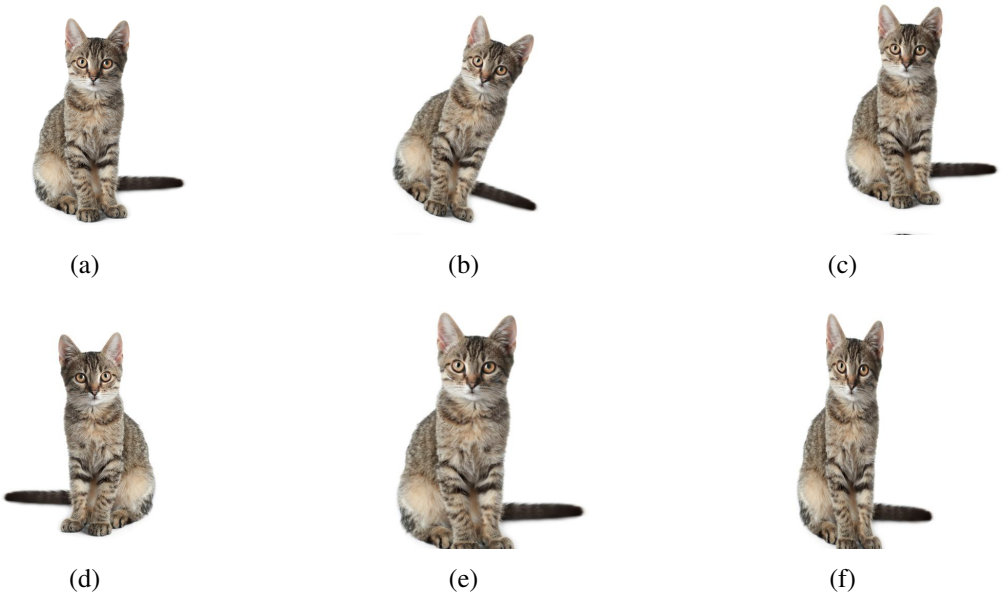


Figura 3.2. Aplicação de aumento de dados. (a) Imagem Original, (b) Rotação de 40°, (c) Translação de 20%, (d)Flipping Horizontal, (e) Zoom de 20% e (f) Cisalhamento de 20°.

3.2. Aprendizado Profundo

Deep Learning ou Aprendizagem Profunda é uma técnica de aprendizado de máquina desenvolvida a partir das Redes Neurais Artificiais (RNAs). As RNAs são modelos matemáticos que tentam simular algumas das estruturas neurais biológicas, possuindo capacidade computacional adquirida através do aprendizado e generalização [Haykin 2007]. Pode-se dizer então que as RNAs são capazes de reconhecer e classificar padrões e posteriormente generalizar o conhecimento adquirido. Na Figura 3.3 observamos um exemplo de RNA genérica com duas camadas densas.

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e, com isso, melhorar seu desempenho. O aprendizado em RNAs está normalmente associado à capacidade de as mesmas adaptarem os seus parâmetros como consequência da sua interação com o meio externo. O processo de aprendizado é interativo e por meio dele a RNA deve melhorar o seu desempenho gradativamente à medida que interage com o meio externo [Rezende 2003]. Pode-se denominar o algoritmo de aprendizado como um conjunto de regras bem definidas para a solução de um problema de aprendizado [Haykin 2007]. Existem muitos tipos de algoritmos de aprendizado específicos para determinados modelos de redes neurais. Estes algoritmos diferem entre si,

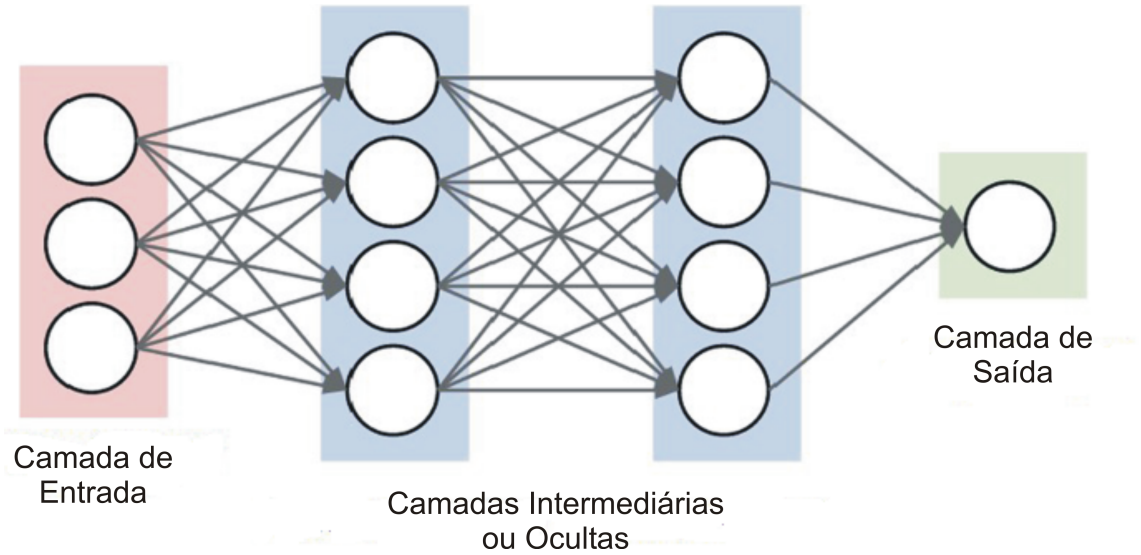


Figura 3.3. Exemplo de rede neural com com duas camadas densas. Fonte: Sophiya and Jothilakshmi 2018

principalmente, pelo modo como os pesos são modificados.

O *Deep Learning* passa por grandes evoluções e vem se destacando no seu paradigma de "habilitar o computador a aprender a partir da observação dos dados". O Aprendizado Profundo passou a ser aplicado em diversas áreas, em particular, mas não exclusivamente, as áreas de Visão Computacional, Processamento de Imagens, Computação Gráfica, entre outras mais [Ronneberger et al. 2015, Bejnordi et al. 2018]. As Redes neurais convolucionais começaram a aparecer como base para métodos do estado da arte em diversas aplicações. A competição *ImageNet* [Deng et al. 2009] teve grande impacto nesse processo, começando uma corrida para encontrar o modelo que seria capaz de superar o atual campeão nesse desafio de classificação de imagens, além de segmentação de imagens, reconhecimento de objetos, entre outras tarefas.

Uma das vantagens dos algoritmos de *Deep Learning* é sua a capacidade de aprendizagem em grandes quantidades de dados de uma forma não-supervisionada, sendo assim uma ferramenta valiosa para *Big Data Analytics* onde a maioria dos dados são desta natureza, também designados por dados não-estruturados.

3.2.1. Aplicação de Deep Learning

O aprendizado de máquina e o *Deep Learning* são amplamente utilizados em muitos domínios, tais como: na medicina, em documentos, nos bancos, no processamento de linguagem natural, na recuperação de imagens, entre outros domínios.

Na medicina pode ser utilizada na detecção de células cancerígenas, restauração de imagem de ressonância magnética cerebral, impressão de genes, entre outras. Nos documentos, essas áreas poderão auxiliar na resolução de imagens de documentos históricos e segmentação de texto em imagens de documentos. Para o bancos poderá ser realizada uma previsão de ações e decisões financeiras [Pacheco and Pereira 2018].

O Processamento de Linguagem Natural são sistemas de recomendação como por

exemplo: Netflix que utiliza um sistema de recomendação para sugerir filmes aos usuários com base em seu interesse, análise de sentimentos e marcação de fotos. Para a recuperação de Informações poderão ser aplicadas os mecanismos de busca, pesquisa de texto e pesquisa de imagens como as utilizadas pelo Google, Amazon, Facebook e Linkedin.

Deep Learning pode ser usada para:

⇒ Pré-processamento: que é uma maneira de realizar um ajuste ou melhoramento nos dados;

⇒ Extração de recursos:que é um processo para reconhecer algum padrão entre os dados e tem como intuito tornar o processo de decisão mais fácil durante a classificação;

⇒ Classificação: que é uma tarefa de predição e até uma classificação de várias classes.

Existem outras funcionalidades do *Deep Learning*, tais como regressão, reconhecimento de objetos, previsões, entre outras.

3.2.2. Redes Neurais Convolucionais

As CNNs fazem parte do conjunto de técnicas de *Deep Learning*. Essas redes convolucionais são uma classe de RNAs que modelam abstrações em alto nível através de imagens e camadas convolucionais dispostas de forma sequencial ou não. O conceito de CNN foi apresentado por Yann LeCun [Lecun et al. 1998] e Fukushima [Fukushima 1988] na década de 90. No entanto, apenas no século XXI essa tecnologia foi desenvolvida com eficácia.

Atualmente as CNNs são empregadas nos mais diversos problemas que envolvem a classificação, segmentação e detecção em imagens. Uma das principais vantagens na utilização de CNNs é a alta capacidade de aprender os mais diversos padrões que não são perceptíveis para outras técnicas tradicionais que utilizam descritores. No entanto, essa vantagem demanda um alto custo computacional e a necessidade de grandes bases de dados para o treinamento.

Camadas Convolucionais: Como as demais redes neurais artificiais, as CNNs apresentam estruturas no formato de camadas que auxiliam na extração de características, redução e classificação. A principal é a que denomina esse tipo de rede. As camadas convolucionais são compostas por uma quantidade c de filtros com tamanho $d \times d$ que irão extrair mapas de características ao serem convoluídos com as imagens de entrada ou com saídas de outras camadas.

A saída da convolução é denominada mapa de características ou *features map*. Esses mapas geralmente representam características gerais extraídas da imagem, como cor, borda, textura e forma. Um exemplo da operação de convolução é apresentada na Figura 3.4.

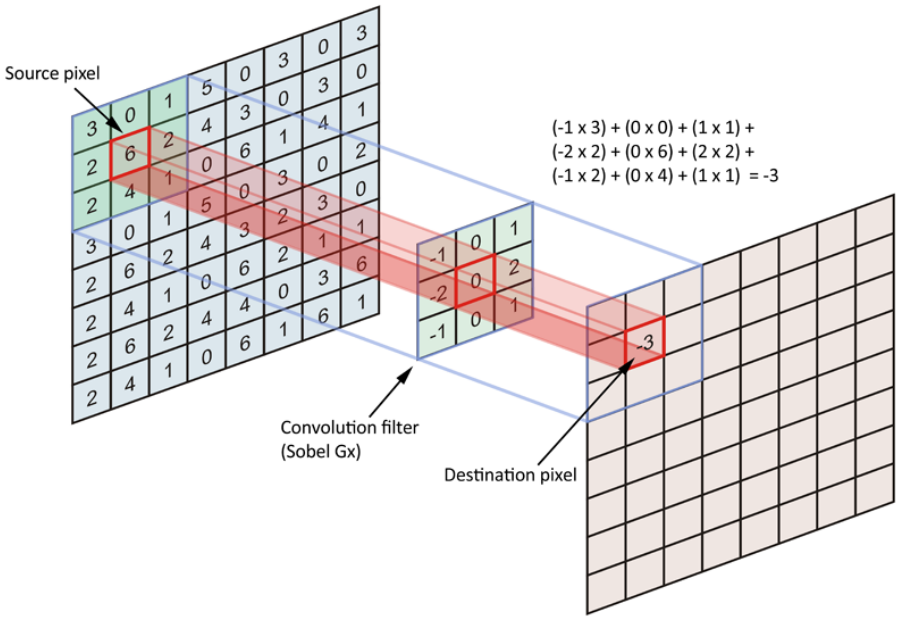


Figura 3.4. Exemplo de operação de convolução com tamanho de filtro 3x3. Fonte: Hachilif et al. 2019

Camadas de Pooling: Após as camadas convolucionais, geralmente são empregadas as de *pooling*. Esse tipo de camada reduz a dimensão espacial dos mapas gerados por camadas anteriores, consequentemente reduzindo o custo computacional. Dentre os diversos tipos, de *pooling* existe o *maxpooling*. Essa operação utiliza uma janela deslizante de tamanho $n \times n$ no mapa de características e para cada passo realizado, o valor máximo daquela janela é retirado. Na Figura 3.5 essa operação é ilustrada com um mapa aleatório.

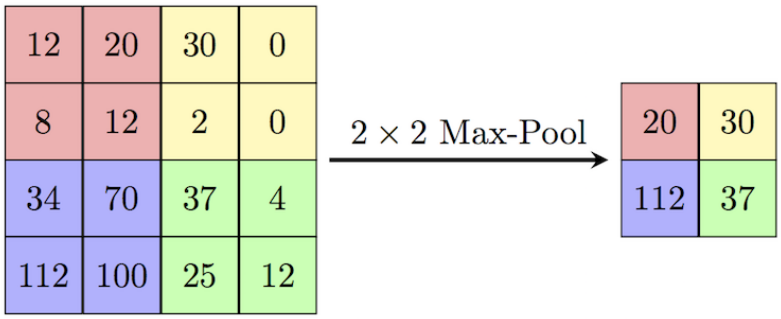


Figura 3.5. Exemplo de operação de maxpooling. Fonte: Gupta et al. 2018

Camadas Totalmente Conectadas: As camadas densas ou totalmente conectadas (*Fully-Connected layers* - FCs) foram apresentadas inicialmente nas RNAs. São constituídas de neurônios que representam pesos e guardam o aprendizado da rede neural. Nas CNN's elas apresentam a mesma função e geralmente aparecem ao final da arquitetura, após inúmeras camadas convolucionais. A operação que transforma os mapas de características em um vetor de neurônios é realizada pela camada *flatten*. A camada de classificação

presente nas CNNs é do tipo densa, no entanto, apresenta uma função de ativação do tipo *softmax*, enquanto as FCs contam com ativação por meio da ReLu.

3.2.2.1. Arquiteturas e ImageNet

A popularidade das CNNs cresceu durante o *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [Russakovsky et al. 2015]. Essa competição ocorreu a partir de 2010 por 7 anos ininterruptos e teve como principais desafios a classificação de imagens e detecção de objetos em larga escala. O *dataset* popularmente conhecido como ImageNet possui mais de 1 milhão de imagens e cerca de mil classes.

Durante essa competição foram propostas arquiteturas que atualmente compõem o estado da arte, dentre elas temos a AlexNet [Krizhevsky et al. 2012], Inception ou GoogLeNet [Szegedy et al. 2015], VGGNet [Simonyan and Zisserman 2014] e ResNet [He et al. 2016]. Essas arquiteturas foram propostas por diversas companhias do ramo de tecnologia e cada uma possui características específicas que acabaram influenciando o desenvolvimento das posteriores. Abaixo elencaremos as mais relevantes nos respectivos anos de desafios e sua contribuição na literatura.

AlexNet: Em 2012 [Krizhevsky et al. 2012] propuseram a AlexNet, uma CNN com cinco camadas convolucionais seguidas de *maxpooling*, duas camadas totalmente conectadas e uma de classificação. Essa CNN obteve excelentes resultados no ILSVRC 2012 quando comparados com os anos anteriores. Em 2010 a taxa de erro era de 28,2%, em 2011 a evolução não foi como esperado e o erro baixou para 25,8%. Em 2012 o erro baixou para 16,4% com a utilização da AlexNet.

Nessa competição observamos a efetividade das CNNs quando comparadas com outras abordagens. Entretanto, os autores constataram que o desempenho foi obtido devido a profundidade da arquitetura e que isso representa um alto custo computacional. Ao todo, são 62,3 milhões de parâmetros, sendo assim necessário a utilização de GPUs para processar todos os dados em tempo hábil. Na Figura 3.6 observamos o *design* da AlexNet.

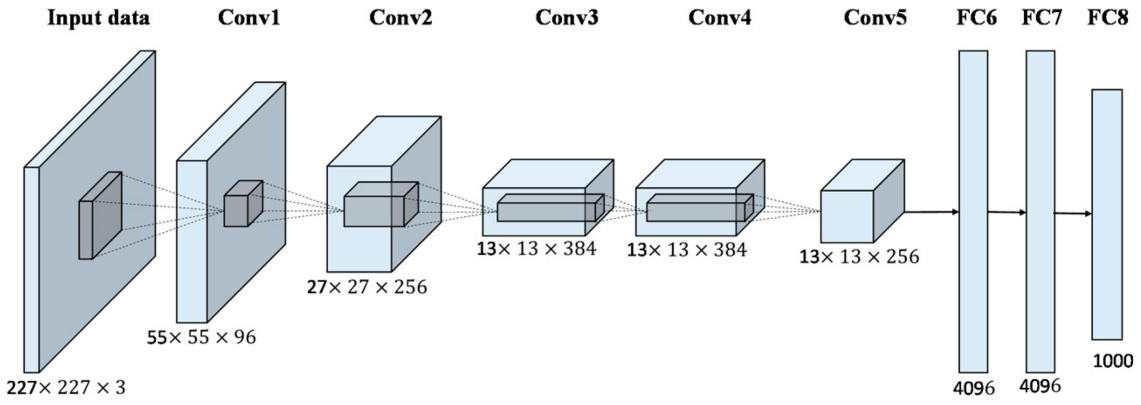


Figura 3.6. Arquitetura da AlexNet. Fonte: Shaees et al. 2020

GoogLeNet: Com o aumento popularidade do evento proporcionada pela Alex-

Net, as grandes companhias da tecnologia como a Google e a Microsoft montaram times para participar da competição. Em 2014, a Google e o seu time obtiveram os melhores resultados na competição com a GoogLeNet [Szegedy et al. 2015]. Essa CNN faz uso de módulos denominados *inception*. Eles são capazes de armazenar múltiplas camadas convolucionais em paralelo, mudando o paradigma e a rede deixa de ser sequencial. Utilizando essa estratégia, a GoogLeNet alcançou uma taxa de erro de 6,7% para a classificação de imagens. No entanto, para a localização de objetos, a erro foi de 26,4%, maior que o erro obtido pela VGG, sua principal concorrente na competição.

A estrutura da GoogLeNet conta com topologia de 22 camadas com cerca de 4 milhões de parâmetros. O *design* dessa CNN foi inspirado na LeNet [Lecun et al. 1998] com os módulos *inception*. Na Figura 3.7 observamos a arquitetura da CNN e a sua profundidade quando comparada com os vencedores anteriores.

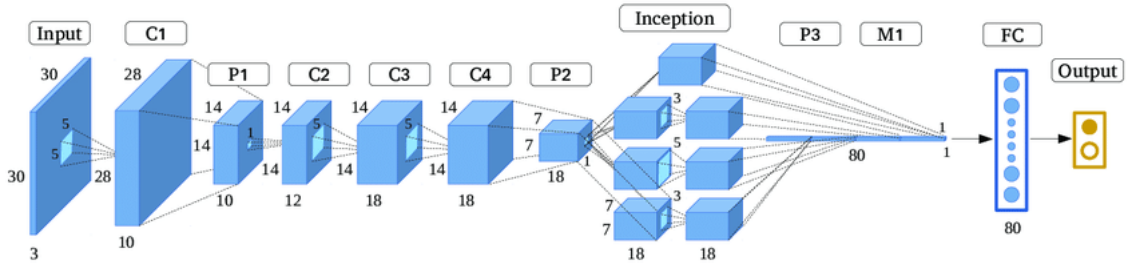


Figura 3.7. Arquitetura da GoogleNet. Fonte: Guo et al. 2017

VGGNet: A principal concorrente da GoogLeNet no ILSVRC 2014 foi a VGGNet proposta por Simonyan and Zisserman [Simonyan and Zisserman 2014] com 16 camadas convolucionais dispostas de forma sequencial. No ILSVRC 2014, a VGGNet alcançou a segunda melhor taxa de erro para classificação de imagens com 7,3% e o melhor resultado para localização de objetos, com 25,3%.

A estratégia utilizada no desenvolvimento dessa CNN é a uniformidade no tamanho dos filtros convolucionais, sendo todos 3×3. No entanto, a quantidade de filtros dobra em quase todos os blocos convolucionais, onde apenas o último mantém a mesma quantidade. A arquitetura dispõe de duas camadas totalmente conectadas com 4096 elementos.

Os autores justificaram a utilização de filtros 3×3 devido a quantidade de operações realizadas durante o convolução. Eles constataram que duas camadas com 3×3 executam uma quantidade próxima de operações quando comparadas com camadas de tamanho 5×5.

ResNet: Proposta em 2015 [He et al. 2016], a Rede Neural Residual foi desenvolvida pelo time da Microsoft para solucionar o problema do desaparecimento do gradiente ou *vanishing gradient* que ocorre quando são adicionadas muitas camadas em um modelo sequencial. Com o excesso de camadas, ao ser propagado de volta na topologia, o excesso de operações decresce o valor do gradiente. Como consequência, o aprendizado se torna mais lento decorrente da saturação no desempenho da CNN, degradando com o tempo.

A ResNet venceu a competição ILSVRC-2015 com taxas de erro de apenas 3,56%, considerada uma taxa menor que a dos seres humanos que alcançam entre 5% a 10%.

Os autores propuseram a ResNet utilizando um estado de ablação que resultou em cinco arquiteturas com diferentes profundidades, sendo elas com 18, 34, 50, 101 e 152 camadas. Na Figura 3.8 apresentamos uma ilustração da CNN ResNet.

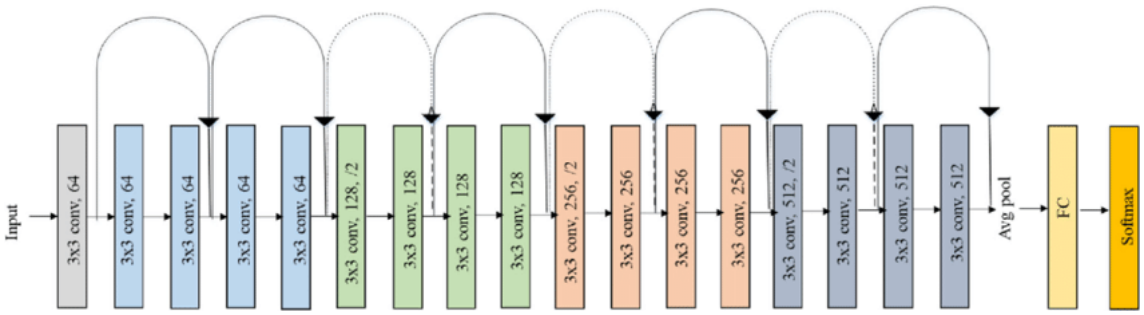


Figura 3.8. Arquitetura da ResNet com 18 camadas. Fonte: Ramzan et al. 2019

Em 2016, o vencedor da competição de classificação do ILSVRC foi a metodologia Trimps-Soushen. A taxa de erro obtida ao final da competição foi 2,99%. Nessa edição do evento os vencedores não apresentaram nenhuma inovação relacionada a estrutura das CNNs como nos anos anteriores. Diante disso, essa abordagem não se tornou conhecida como as demais CNNs.

Já em 2017, a *Squeeze-and-Excitation Network* (SENet) [Hu et al. 2018] foi a vencedora com 2,25% de taxa de erro. A denominação utilizada nesse tipo de rede neural vem dos blocos convolucionais *Squeeze-and-Excitation*. Esses blocos recalibram adaptativamente as características de saída de cada canal, modelando explicitamente as interdependências entre os canais. Diante disso, as redes neurais convolucionais consegue generalizar efetivamente diferentes bases de dados com a mesma estrutura.

Outras arquiteturas famosas não foram inicialmente propostas para o concurso, dentre elas temos a CaffeNet [Jia et al. 2014], InceptionV3 [Szegedy et al. 2016], Xception [Chollet 2017] e DenseNet [Huang et al. 2017]. Além das CNNs utilizadas para classificar imagens, foram propostas algumas com o objetivo de segmentar regiões.

U-Net: Sendo desenvolvida para segmentar imagens biomédicas, a U-Net é uma CNN que realiza a segmentação de uma determinada região tendo como base a sua marcação real [Ronneberger et al. 2015]. A ideia desenvolvida para essa CNN é de utilizar os mapas de características para contrair e expandir o vetor da imagem segmentada. Para realizar essa operação, a arquitetura é dividida em três partes, a contração, o gargalo e a expansão. A etapa de contração é realizada por blocos com camadas convolucionais com filtros 3×3 seguidos por *maxpooling* com janelas 2×2.

A parte de expansão é considerada a mais importante, uma vez que possui blocos com camadas convolucionais e de *upsampling* que possuem o mesmo tamanho das camadas da fase de contração. Além dos blocos, são concatenados mapas de características da etapa de contração que permitem um melhor aprendizado durante a reconstrução da segmentação. O efeito proporcionado pela expansão mantém a simetria original da imagem de entrada, fazendo com que a saída seja de mesmo tamanho. Na Figura 3.9, apresentamos uma ilustração da arquitetura onde observamos as etapas descritas previamente.

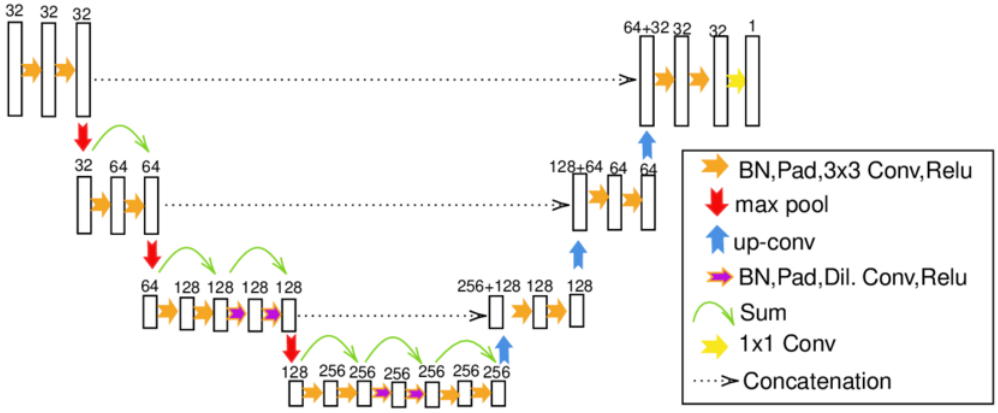


Figura 3.9. Arquitetura U-Net. Fonte: Furat et al. 2023

3.2.2.2. Arquitetura Genérica

Para exemplificar a construção de uma CNN e a definição das camadas convolucionais, totalmente conectadas e seus tamanhos, apresentamos uma arquitetura genérica desenvolvida para solucionar um problema simples de classificação. Neste caso, utilizamos o *dataset* MNIST. Essa base contém imagens com dimensão 24×24 e está dividida em 10 classes que representam os números de 0 a 9. Cada imagem possui um número escrito a mão livre e o objetivo é classificar os números de acordo com a sua respectiva classe.

Diante do problema apresentado, propomos uma CNN com duas camadas convolucionais, duas de *maxpooling* e uma camada totalmente conectada. Na Tabela 3.1, apresentamos a estrutura da CNN com o tamanho dos filtros, entrada e saída.

Tabela 3.1. Estrutura de uma CNN genérica.

Tipo de camada	Entrada	Qtd. de filtros	Tam. dos filtros	Passo	Saída
conv 1	24×24×3	64	5×5×3	1	22×22×64
maxpool 1	22×22×64	-	2×2	2	11×11×64
conv 2	11×11×64	32	3×3×32	1	9×9×32
maxpool 2	9×9×32	-	2×2	2	4×4×32
flatten	4×4×32	-	-	-	1×512
fc 1	-	1×512	-	-	512
fc 2	512	-	-	-	10

3.3. Métodos de Explicabilidade

A explicabilidade basicamente ajuda na interpretação da saída de um modelo por um usuário, já que esses modelos apresentam um padrão particular para cada tomada de decisão. Uma característica importante das explicações é sua semelhança com o tipo de dado utilizado no treinamento.

A Figura 3.10 mostra diferentes explicações conforme o tipo de dados associado. Dessa forma, quando o tipo de dados é uma imagem a sua explicação pode ser um *heat-*

map, enquanto para um tipo texto a explicação pode ser palavras destacadas no texto e em casos de tabelas de atributos, a explicação é apresentada como um conjunto de regras que descrevem as combinações de atributos capazes de levar às previsões.

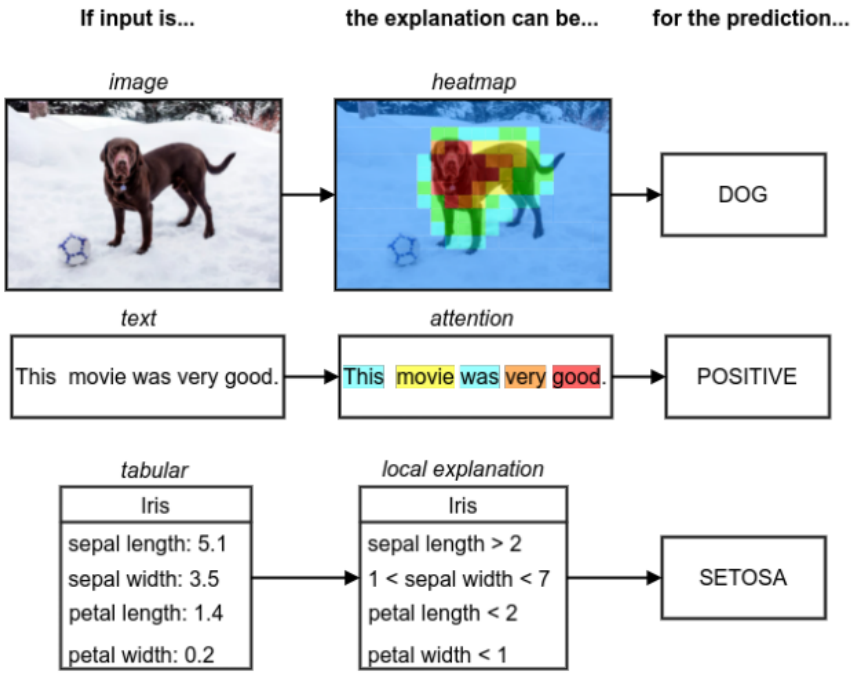


Figura 3.10. Exemplos de como as explicações podem parecer na prática. Fonte: Xie et al. 2020

Diante da enorme quantidade de parâmetros que caracterizam as CNN's, é necessário inspecionar cuidadosamente antes de ser implantado em qualquer ambiente médico, para que se tenha compreensão de seu funcionamento interno. Nesse contexto, a explicabilidade é importante para aumentar a compreensão dessas redes, bem como sua transparência, elevando a confiança dos seus usuários. Segundo Xie et al. [Xie et al. 2020] os métodos de explicabilidade podem ser categorizados em métodos de visualização, destilação de modelo e métodos intrínsecos.

As seções a seguir descrevem detalhadamente cada uma dessas categorias e na Figura 3.11 são listados alguns desses métodos.

3.3.1. Métodos de Visualização

Nos métodos de visualização a explicabilidade é demonstrada destacando as regiões que influenciam fortemente na saída da rede, os chamados *saliency maps* ou *heatmaps*. Eles podem ser subdivididos em métodos baseados em retropropagação e visualização baseada em perturbação.

Os métodos baseados em retropropagação mensuram a precisão de uma saída, dado às informações de entrada ou tal visualização pode ser feita avaliando a saída, desde que adote diferentes mapas de características. Enquanto na visualização por perturbação, avaliam a precisão realizando uma alteração na entrada (alterar ou remover recurso), em

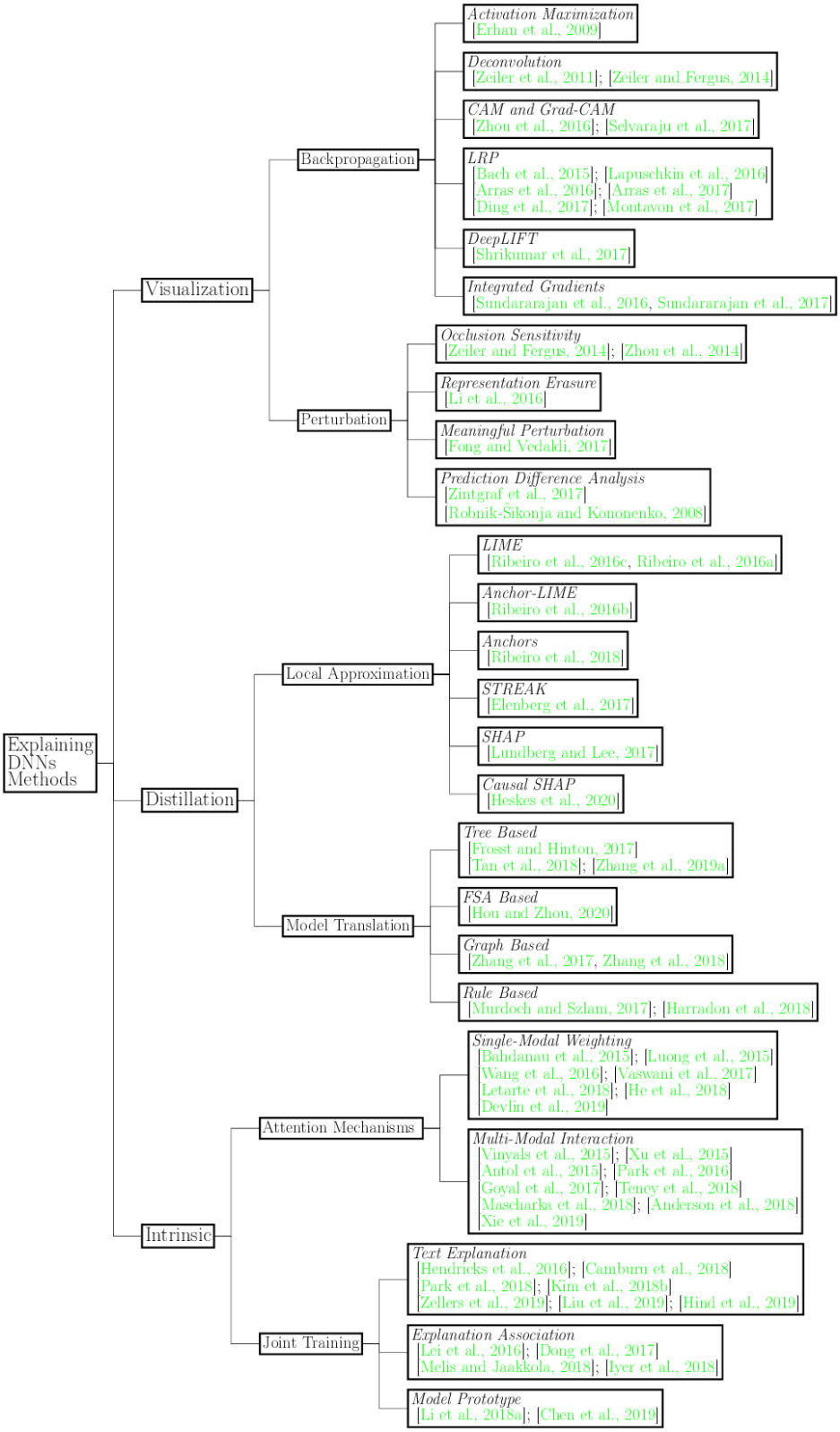


Figura 3.11. Métodos para explicar DNNs. Fonte: Xie et al. 2020

seguida, comparam a saída entre as redes alterada e original.

Alguns métodos que fazem parte desse campo são: *Activation Maximization* [Erhan et al. 2009], textitDeconvolution [Zeiler and Fergus 2014a, Zeiler et al. 2011], CAM and Grad-CAM [Zhou et al. 2016, Selvaraju et al. 2017, Lapuschkin et al. 2016], *Integrated Gradients* [Sundararajan et al. 2016, Sundararajan et al. 2017], *Occlusion Sensitivity* [Uchiyama et al. 2023, Zeiler and Fergus 2014b] e *Representation Erasure* [Li et al. 2016]. Dentre esses métodos, adotamos o Grad-CAM durante os experimentos para exemplificar a explicabilidade por método de visualização.

3.3.2. Destilação de Modelo

As abordagens utilizam um modelo de aprendizagem, no qual consiste em um modelo inerentemente explicável, popularmente conhecido como "white-bo". Nesse modelo, busca-se distinguir as regras de decisão ou características de entrada que induzem as saídas da rede.

Esse modelo tem acesso às informações da rede treinada, assim como os dados de entrada originais, então, a interpretação desse modelo pode fornecer informações pouco profundas. Porém, é possível obter informações sobre a correlação, características e regras relacionadas a rede. A destilação do modelo possui duas categorias: aproximação local e tradução de modelos.

A aproximação local descreve um modelo de aprendizado simples sob um subconjunto de entrada pequeno. Nessa categoria, a grande motivação consiste em conseguir diferenciar a variedade de dados em uma área local ao invés de levar em consideração ao conjunto completo. Enquanto a tradução de modelos treinam uma estrutura menor organizado conforme o modelo completo. Em oposição a aproximação local, essa categoria usa o conjunto de dados inteiro.

Os métodos mais usados da categoria destilação de modelo são: LIME [Ribeiro et al. 2016b, Ribeiro et al. 2016a], SHAP [Lundberg and Lee 2017], *FSA Based* [Hou and Zhou 2018] e *Rule Based* [Murdoch and Szlam 2017]. Dentre esses métodos, adotamos o LIME durante os experimentos para exemplificar a explicabilidade por destilação de modelo.

3.3.3. Métodos Intrínsecos

Os modelos apresentam nas suas saídas explicações sobre suas decisões, ou seja, as explicações devem ser intrínsecas ao processo de projeção das arquiteturas e no treinamento. Como consequência essas estruturas são facilmente explicáveis, pois eles têm a capacidade de aprender as saídas precisas por entrada e as saídas que expressam uma explicação da ação da rede.

Contudo, nessa categoria o usuário precisa ter um conhecimento amplo sobre o campo de aplicação adotado, por essa razão os métodos intrínsecos são mais difíceis de serem implementados e provavelmente exigem um maior custo computacional, já que durante o treinamento ocorre a aplicação de diversos modelos para aprender as explicações intrinsecamente.

Duas tendências são seguidas por estes modelos, a primeira é a introdução de mecanismos de atenção a uma rede, e a segunda, incluem uma explicação complementar

à tarefa original do modelo, posteriormente, realiza um teste com a combinação de ambas. Dentre esses métodos temos o *Single-Modal Weighting* [Luong et al. 2015, Devlin et al. 2018], *Multi-Modal Interaction* [Goyal et al. 2017], *Text Explanation* [Zellers et al. 2019], *Explanation Association* [Li et al. 2017], *Model Prototype* [Chen et al. 2019].

3.4. Explicabilidade na Prática

Essa seção tem o objetivo de demonstrar a aplicabilidade da explicabilidade no conjunto de imagens da base pública ISIC 2016 [Gutman et al. 2016] usando todos os conceitos apresentados até o momento.

3.4.1. Base de Imagens

No problema abordado neste capítulo, apresentamos a base ISIC 2016 [Gutman et al. 2016], uma das base de imagens dermatoscópicas encontrada em diversas literaturas. Essa base de dados possui 900 imagens, divididas em imagens de lesões benignas e malignas. Vale ressaltar que utilizarmos o conjunto de treino do Desafio (Part 3: Lesion Classification), disponível em `https://challenge.isic-archive.com/landing/2016/`. A resolução das imagens de entrada variam entre 767×576 e 3024×2016. Na Figura 3.12, apresentamos alguns exemplos da base utilizada no experimentos.

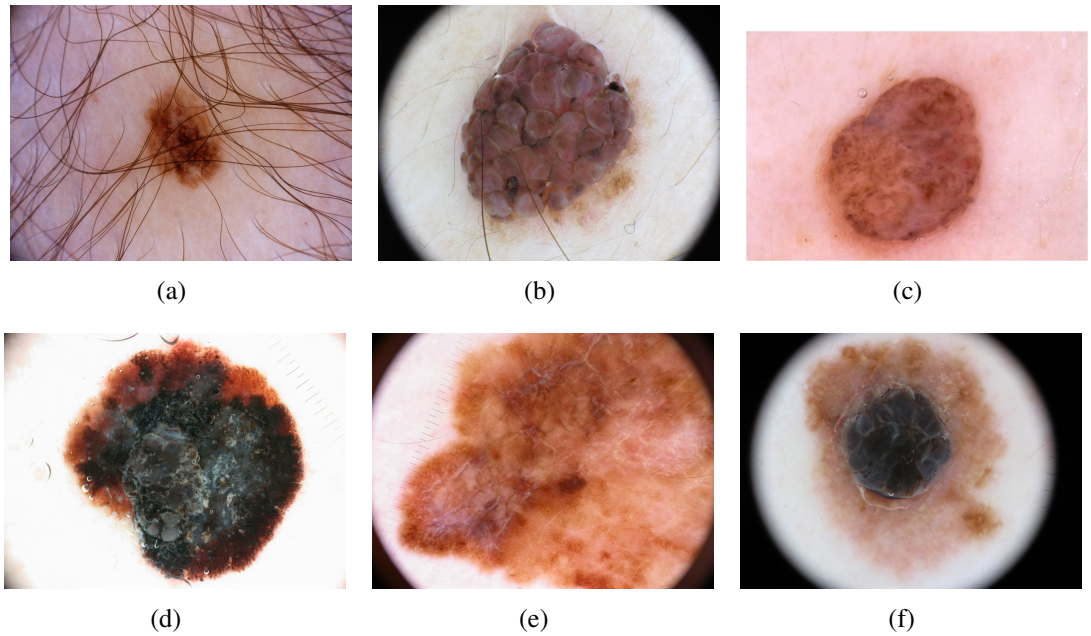


Figura 3.12. Exemplos de imagens da base ISIC 2016 (a, b, c) são benignas e (d, e, f) são lesões malignas.

- Pré-Processamento da base de imagens

  Para realizar o experimento, precisamos realizar um pré-processamento das imagens, com a intenção de dividir a base disponibilizada em treino e teste. A separação é feita utilizando o código abaixo:

```
1 #Conectar com o Drive
```

```
from google.colab import drive
drive.mount('/content/drive')

#Importar bibliotecas
import pandas as pd
import os
import csv
import cv2
from sklearn.model_selection import train_test_split

tabela=pd.read_csv('/content/drive/MyDrive/Minicurso SBCAS/
    ISBI2016_ISIC_Part3_Training_GroundTruth.csv',sep=",") #
    Leitura de arquivo csv
#display(tabela) #Exibe na tela o arquivo lido

#Cria pastas
dir = '/content/drive/MyDrive/Minicurso SBCAS/' #Local das pastas
os.mkdir(dir+'base/')
os.mkdir(dir+'base/original/')
os.mkdir(dir+'base/train/')
os.mkdir(dir+'base/test/')
os.mkdir(dir+'base/original/malignant/')
os.mkdir(dir+'base/train/malignant/')
os.mkdir(dir+'base/test/malignant/')
os.mkdir(dir+'base/original/benign/')
os.mkdir(dir+'base/train/benign/')
os.mkdir(dir+'base/test/benign/')

mal=[] #armazena nomes imagens malignant
ben=[] #armazena nomes imagens benign

with open(dir+'ISBI2016_ISIC_Part3_Training_GroundTruth.csv','r')
    as csvfile:
     reader_obj = csv.reader(csvfile)
     for row in reader_obj:
         if str(row[1]) == 'malignant':
             mal.append(str(row[0]))
         if str(row[1]) == 'benign':
             ben.append(str(row[0]))

caminho_base_descompactada = '/content/drive/MyDrive/Minicurso
    SBCAS/ISBI2016_ISIC_Part3_Training_Data/
    ISBI2016_ISIC_Part3_Training_Data/'

for i in range(0,len(mal)):
     data = cv2.imread(caminho_base_descompactada+mal[i]+'.jpg')
     cv2.imwrite(dir+'base/original/malignant/' + mal[i]+'.jpg',
    data)

for i in range(0,len(ben)):
     data = cv2.imread(caminho_base_descompactada+ben[i]+'.jpg')
     cv2.imwrite(dir+'base/original/benign/' + ben[i]+'.jpg',data)

#Dividir base em conjunto de treino 80% e teste (20%)
trainMal, testMal = train_test_split(mal, train_size=0.8)
```

```
trainBen, testBen = train_test_split(ben, train_size=0.8)

# Salva imagens em suas respectivas pastas treino malignas
for i in range(0,len(trainMal)):
    data = cv2.imread(dir+'base//original/malignant'+trainMal[i]+'.jpg')
    cv2.imwrite(dir+'base/train/malignant/' + trainMal[i]+'.jpg', data)

# Salva imagens em suas respectivas pastas teste malignas
for i in range(0,len(testMal)):
    data = cv2.imread(dir+'base/original/malignant/'+testMal[i]+'.jpg')
    cv2.imwrite(dir+'base/test/malignant/' + testMal[i]+'.jpg', data)

# Salva imagens em suas respectivas pastas treino benignas
for i in range(0,len(trainBen)):
    data = cv2.imread(dir+'base/original/benign/'+trainBen[i]+'.jpg')
    cv2.imwrite(dir+'base/train/benign/' + trainBen[i]+'.jpg',data)

# Salva imagens em suas respectivas pastas teste benignas
for i in range(0,len(testBen)):
    data = cv2.imread(dir+'base/original/benign/'+testBen[i]+'.jpg')
    cv2.imwrite(dir+'base/test/benign/' + testBen[i]+'.jpg',data)
```

3.4.2. Aprendizado profundo

- Definição de Parâmetros Globais: Define a rede a ser utilizada no teste, pode ser utilizada tanto a VGG-16 quanto a Resnet50. Ainda, deve-se se definir por meio da variável “rede” se a rede testada possui arquitetura original ou houve alguma alteração. A quantidade de épocas (qt.depocas) é utilizada para limitar o treinamento da rede e “size_global”define o tamanho da entrada da rede.

```
#rede_utilizada = 'VGG-16'
rede_utilizada = 'Resnet50'

rede = 'original'
#rede = 'modificada'

qtdepocas=200

if rede == 'original':
    size_global = 224 # usado para as redes originais

elif rede == 'modificada':
    size_global = 112 # usado para as redes modificada
```

- Criação das Redes Neurais: Após a definição de parâmetros, pode-se criar a estrutura das redes neurais a serem aplicadas nos experimentos. Elas são definidas conforme o código a seguir:

```
from tensorflow.keras import layers
from keras.layers import Dense, GlobalAveragePooling2D,
    BatchNormalization
from tensorflow.keras.applications import VGG16,ResNet50
import tensorflow as tf
from tensorflow.keras import Model

def cria_minha_rede_VGG16():
    if rede == 'original':
        rede_model = VGG16(include_top=True, weights='imagenet')

    elif rede == 'modificada':
        rede_model = VGG16(include_top=False, weights='imagenet')

    input = rede_model.input

    # bloco 1
    conv = rede_model.layers[1](input)
    conv = rede_model.layers[2](conv)
    bn = BatchNormalization()(conv)
    max = rede_model.layers[3](bn)

    # bloco 2
    conv = rede_model.layers[4](max)
    conv = rede_model.layers[5](conv)
    bn = BatchNormalization()(conv)
    max = rede_model.layers[6](bn)

    # bloco 3
    conv = rede_model.layers[7](max)
    conv = rede_model.layers[8](conv)
    conv = rede_model.layers[9](conv)
    bn = BatchNormalization()(conv)
    max = rede_model.layers[10](bn)


    # bloco 4
    conv = rede_model.layers[11](max)
    conv = rede_model.layers[12](conv)
    conv = rede_model.layers[13](conv)
    bn = BatchNormalization()(conv)
    max = rede_model.layers[14](bn)

    # bloco 5
    conv = rede_model.layers[15](max)
    conv = rede_model.layers[16](conv)
    conv = rede_model.layers[17](conv)
    bn = BatchNormalization()(conv)
    max = rede_model.layers[18](bn)

    if rede == 'original':
        x = rede_model.output

    elif rede == 'modificada':
        x = GlobalAveragePooling2D()(max) # (x)
```

```
    x = Dense(512, activation='relu')(x)
    predictions = Dense(4, activation='softmax')(x)

    meu_modelo = Model(inputs=rede_model.input, outputs=
    predictions)
    meu_modelo.compile(optimizer='adam', loss=tf.losses.
    SparseCategoricalCrossentropy(),metrics=['accuracy'])

    print("VGG-16 512 e input_size = ", size_global)
    meu_modelo.summary() #exibe o modelo criado

    return meu_modelo

def cria_minha_rede_Resnet50():
    if rede == 'original':
        meu_modelo = ResNet50(include_top=True, weights='imagenet'
    )
        x = meu_modelo.output

    elif rede == 'modificada':
        meu_modelo = ResNet50(include_top=False, weights='imagenet
    ')
        x = meu_modelo.output
        x = GlobalAveragePooling2D()(x)

    x = Dense(512, activation='relu')(x)

    predictions = Dense(4, activation='softmax')(x)
    meu_modelo = Model(inputs=meu_modelo.input, outputs=
    predictions)
    meu_modelo.compile(optimizer='adam', loss=tf.losses.
    SparseCategoricalCrossentropy(),metrics=['accuracy'])

    meu_modelo.summary()

    print("ResNet50 512 e input_size = ", size_global)

    return meu_modelo
```

- Aumento de Dados: Nessa seção, defini-se os aumentos de dados aplicados na base ISIC 2016, nesse caso, optou-se pelo uso de rotação, translação, cisalhamento, zoom e flipping. A biblioteca utilizada foi "ImageDataGenerator" que permite a geração de imagens apenas no momento da execução do programa, sem a necessidade de alocar imagens na memória do computador/Drive. A definição dessas técnicas é descrita no código a seguir.

```
!pip install Keras-Preprocessing
```

```
from keras_preprocessing.image import ImageDataGenerator



def aumentodedadosTrain():
    #Imagens aumentadas apenas no momento do processamento
```

```
    train_datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1. / 255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='reflect'
    )

    test_datagen = ImageDataGenerator(rescale=1./255)


    train_generator = train_datagen.flow_from_directory(
            traindata,  # pasta treino
            target_size=(size_global, size_global),  #redimensiona
    imagens
            color_mode='rgb',
            batch_size=32,
            class_mode='binary')
    return train_generator


def aumentodedadosTest():
    #Imagens aumentadas apenas no momento do processamento
    train_datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1. / 255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='reflect'
    )


    test_datagen = ImageDataGenerator(rescale=1./255)


    test_generator = test_datagen.flow_from_directory(
            testdata, # pasta teste
            target_size=(size_global, size_global),  #redimensiona
    imagens
            color_mode='rgb',
            batch_size=32,
            class_mode='binary')
    return test_generator
```

A Figura 3.13 apresenta alguns exemplos de imagens resultantes do aumento de dados na base ISIC 2016.

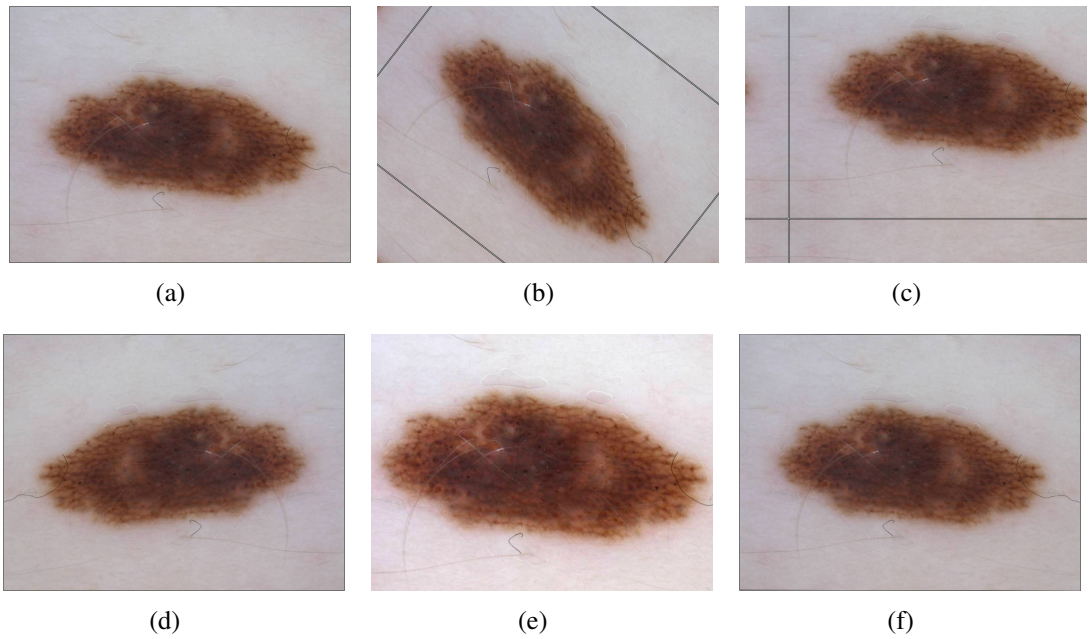


Figura 3.13. Aplicação de aumento de dados na base ISIC 2016. (a) Imagem Original, (b) Rotação de 40°, (c) Translação de 20%, (d)Flipping Horizontal, (e) Zoom de 20% e (f) Cisalhamento de 20°.

- Treinamento

```
!pip install gspread
!pip install keras
```

```
import os
from keras.callbacks import ModelCheckpoint
from keras_preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

#verificar o caminho da drive compartilhado
caminho_base='D:\\'
base_dir = caminho_base + 'base'+'\\'
traindata = os.path.join(base_dir, 'train\\')
testdata = os.path.join(base_dir, 'test\\')

#Cria a rede neural a ser utilizada
if rede_utilizada == 'VGG16':
    model = cria_minha_rede_VGG16()

elif rede_utilizada == 'Resnet50':
    model = cria_minha_rede_Resnet50()

#Realiza aumento de dados
train_generator = aumentodedadosTrain()
test_generator = aumentodedadosTest()

#treinamento da rede
model.compile(optimizer='adam', loss=tf.losses.
   SparseCategoricalCrossentropy(),metrics=['accuracy'])
```

```
save_dir = os.path.join(os.getcwd(), "save_models") #salva o modelo criado
filepath = caminho_base + rede_utilizada + rede +str(qtdepocas) + 'ep.h5'
checkpoint = ModelCheckpoint(os.path.join(save_dir, filepath), verbose=1, monitor='val_loss',save_best_only = True, mode='auto')

history = model.fit(train_generator,
                            epochs=qtdepocas,
                            callbacks = [checkpoint],
                            validation_data=test_generator,
                            validation_steps= test_generator.samples / test_generator.batch_size,
                            verbose=1
                           )

# Mostra grafico
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy","loss","Validation Loss"])
plt.show()


print('Acuracia durante treinamento \n'+ str(history.history['accuracy']) + '\n')
print('Loss durante treinamento \n'+ str(history.history['loss']) + '\n')
print('Acuracia de validacao durante treinamento \n'+ str(history.history['val_accuracy']) + '\n')
print('Loss de validacao durante treinamento \n'+ str(history.history['val_loss']) + '\n')
```

- Cálculo de Métricas de Desempenho:

  A avaliação das CNNs foi calculada com base nos valores obtidos pela matriz de confusão. Com base na matriz, obtemos quatro valores, denominados Verdadeiro Positivo (VP), Falso Positivo (FP), Falso Negativo (FN) e Verdadeiro Negativo (VN). Nesse contexto, foram calculados as métricas de Acurácia , Precisão , *Recall*, *F1-score* e o *Kappa* (Equações 1, 2, 3, 4 e 5) da classificação:

$$Acuracia = \frac{VP+VN}{VP+VN+FP+FN} \tag{1}$$

$$Precisao = \frac{VP}{VP+FP} \tag{2}$$

$$Recall = \frac{VP}{VP+FN} \tag{3}$$

$$F1-score = 2 * \frac{Precisao * Recall}{Precisao + Recall} \tag{4}$$

$$Kappa = \frac{observado - esperado}{1 - esperado} \tag{5}$$

O índice kappa considera todos os elementos da matriz de confusão e funciona como uma medida de associação usada para testar o grau de concordância na classificação. De acordo com Landis e Koch [Landis and Koch 1977], este índice assume valores entre 0 e 1, sendo o resultado qualificado em k $\leq$ 0.2: Ruim; 0.2 $<$ k $\leq$ 0.4: Moderado; 0.4 $<$ k $\leq$ 0.6: Bom; 0.6 $<$ k $\leq$ 0.8: Muito Bom and k $>$ 0.8: Excelente.

No código mostrado a seguir é possível verificar o desempenho da CNN's com base nessas métricas sobre o conjunto de teste, salienta-se que durante essa etapa não é necessário realizar o aumento de dados do conjunto, tal processo só ocorre durante o treinamento das redes.

```
import os
from keras.models import load_model
from tensorflow.keras.preprocessing.image import
    ImageDataGenerator
from sklearn.metrics import confusion_matrix,accuracy_score,
    precision_score, recall_score, roc_auc_score,
    cohen_kappa_score, log_loss, f1_score

def encontrar(elemento):
    pos_i = 0 # variavel provisoria de indice

    for i in range (len(classe)): # procurar em todas as listas
    interna
      if elemento == classe[i]: # se encontrarmos elemento
          pos_i = i # guardamos o indice i
          break
    return (pos_i)


#verificar o caminho da drive compartilhado
caminho_base='/content/drive/MyDrive/Minicurso SBCAS/'
base_dir = caminho_base + "base/"

traindata = os.path.join(base_dir, 'train/')
testdata = os.path.join(base_dir, 'test/')


train_datagen = ImageDataGenerator(rescale=1. / 255)

test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(
      traindata,  # This is the source directory for training
    images
```

```
      target_size=(size_global, size_global),  # All images will be resized to 224x224
      color_mode='rgb',
      batch_size=32,        # Since we use binary_crossentropy loss, we need binary labels
      class_mode='binary') #class_mode='categorical'

test_generator = test_datagen.flow_from_directory(
      testdata,
      target_size=(size_global, size_global),
      color_mode='rgb',
      batch_size=32,
      class_mode='binary')

batch_size = 32

datagen_pred = ImageDataGenerator(rescale=1./255)

generator_pred = datagen_pred.flow_from_directory(
      testdata,
      target_size=(size_global,size_global),
      batch_size=batch_size,
      class_mode='binary',
      shuffle=False)

datagen_pred = ImageDataGenerator(rescale=1. / 255)

generator_pred = datagen_pred.flow_from_directory(
      testdata,
      target_size=(size_global,size_global),
      batch_size=batch_size,
      class_mode='binary',
      shuffle=False)
names=generator_pred.filenames

if rede_utilizada == 'VGG-16':
    path=caminho_base + rede_utilizada + rede +str(qtdepocas) + 'ep.h5'

elif rede_utilizada == 'Resnet50':
    path=caminho_base + rede_utilizada + rede +str(qtdepocas) + 'ep.h5'

classificador=load_model(path)
predicoeskfold = classificador.predict(generator_pred, steps=generator_pred.samples / generator_pred.batch_size) # Model.predict_generator` is deprecated

list_predicoes=[]

for i in range(len(predicoeskfold)):
  classe=[]

  classe1=(predicoeskfold[i][0])
  classe2=(predicoeskfold[i][1])
```

```
  classe.append(classe1)
  classe.append(classe2)
  classemax = max(classe1,classe2)
  classe_definida=encontrar(classemax)
  list_predicoes.append(classe_definida)

predicted_classes=list_predicoes
pred_oficial = generator_pred.classes

for i in range(len(predicted_classes)):
  print('Imagem: '+ str(names[i]) + ' classe predita: '  + str(
    predicted_classes[i]) + ' classe real: ' + str(pred_oficial[i
    ]))

#Calculando
print('Rede: ' + rede_utilizada + ' - ' + rede)
print('Acuracia: ' + str(accuracy_score(pred_oficial,
    predicted_classes)))
print('Recall: ' + str(recall_score(pred_oficial,
    predicted_classes, average='weighted')))
print('Precisao: ' + str(precision_score(pred_oficial,
    predicted_classes, average='weighted')))
print('Kappa: ' + str(cohen_kappa_score(pred_oficial,
    predicted_classes)))
print('F1: ' + str(f1_score(pred_oficial, predicted_classes,
    average='weighted')))
print('Matriz: ' + str(confusion_matrix(generator_pred.labels,
    predicted_classes)))
```

```
Imagem: benign\ISIC_0000038.jpg classe predita: 0 classe real: 0
Imagem: benign\ISIC_0000039.jpg classe predita: 0 classe real: 0
Imagem: benign\ISIC_0000118.jpg classe predita: 1 classe real: 0
Imagem: benign\ISIC_0008236.jpg classe predita: 0 classe real: 0
Imagem: benign\ISIC_0011146.jpg classe predita: 0 classe real: 0
...
Imagem: malignant\ISIC_0000026.jpg classe predita: 0 classe real: 1
Imagem: malignant\ISIC_0000167.jpg classe predita: 0 classe real: 1
Imagem: malignant\ISIC_0000170.jpg classe predita: 1 classe real: 1
Imagem: malignant\ISIC_0000282.jpg classe predita: 1 classe real: 1
Imagem: malignant\ISIC_0000297.jpg classe predita: 0 classe real: 1
Imagem: malignant\ISIC_0000444.jpg classe predita: 1 classe real: 1
...
Rede: Resnet50 - original
Acurácia: 0.8287292817679558
Recall: 0.8287292817679558
Precisão: 0.8099092113340441
Kappa: 0.24633982538616517
F1: 0.7879076594764065
Matriz: [[143   3]
 [ 28   7]]
```

Figura 3.14. Resultado da classificação a partir da base ISIC 2016.

3.4.3. Aplicando a Explicabilidade

- Grad-CAM: essa seção será demonstrada a técnica Grad-CAM, esta técnica é uma das mais comuns no campo de visualização. Baseada no Class Activation Mapping (CAM), consiste em gerar mapas de calor sobre a imagem, baseados nos gradientes da rede. Assim pode-se visualizar regiões específicas da imagem que apresentaram maior contribuição para a classificação de uma determinada classe na camada de predição do modelo. Sua implementação se resume no seguinte código.

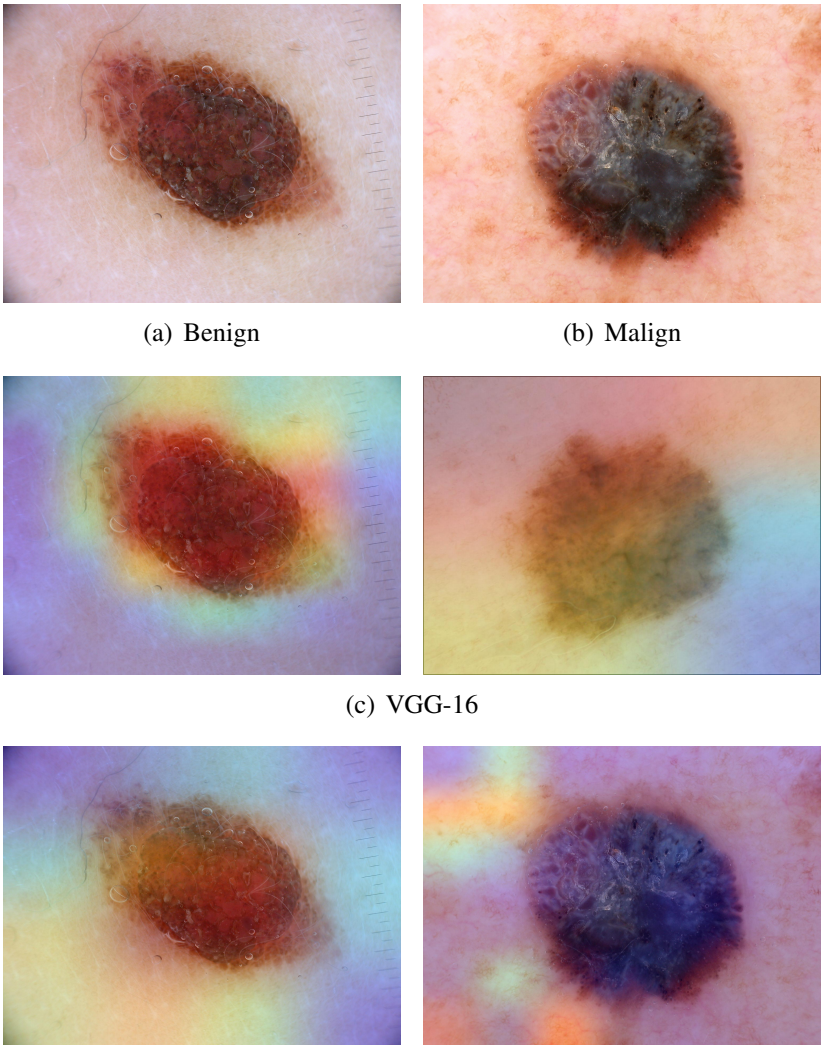
```
import numpy as np
import tensorflow as tf
from tensorflow import keras

from IPython.display import Image, display
import matplotlib.pyplot as plt
import matplotlib.cm as cm

from keras.models import load_model, Model
from keras.applications.imagenet_utils import preprocess_input



def get_img_array(img_path, size):
    # `img` is a PIL image of size 224x224
    img = keras.utils.load_img(img_path, target_size=size)
    # `array` is a float32 Numpy array of shape (224, 224, 3)
    array = keras.utils.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size (1, 224, 224, 3)
    array = np.expand_dims(array, axis=0)
    return array



def make_gradcam_heatmap(img_array, model, last_conv_layer_name,
    pred_index=None):
    # First, we create a model that maps the input image to the
    activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).
    output, model.output]
    )



    # Then, we compute the gradient of the top predicted class for
     our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

```

```
43
44
45      # This is the gradient of the output neuron (top predicted or
        chosen)
46      # with regard to the output feature map of the last conv layer
47      grads = tape.gradient(class_channel, last_conv_layer_output)
48
49
50
51      # This is a vector where each entry is the mean intensity of
        the gradient
52      # over a specific feature map channel
53      pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
54
55
56      # We multiply each channel in the feature map array
57      # by "how important this channel is" with regard to the top
        predicted class
58      # then sum all the channels to obtain the heatmap class
        activation
59      last_conv_layer_output = last_conv_layer_output[0]
60      heatmap = last_conv_layer_output @ pooled_grads[..., tf.
        newaxis]
61      heatmap = tf.squeeze(heatmap)
62
63
64      # For visualization purpose, we will also normalize the
        heatmap between 0 & 1
65      heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
66      return heatmap.numpy()
67
68
69
70  def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg"
        , alpha=0.4):
71      # Load the original image
72      img = keras.preprocessing.image.load_img(img_path)
73      img = keras.preprocessing.image.img_to_array(img)
74
75      # Rescale heatmap to a range 0-255
76      heatmap = np.uint8(255 * heatmap)
77
78      # Use jet colormap to colorize heatmap
79      jet = cm.get_cmap("jet")
80
81      # Use RGB values of the colormap
82      jet_colors = jet(np.arange(256))[:, :3]
83      jet_heatmap = jet_colors[heatmap]
84
85      # Create an image with RGB colorized heatmap
86      jet_heatmap = keras.preprocessing.image.array_to_img(
        jet_heatmap)
87      jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
88      jet_heatmap = keras.preprocessing.image.img_to_array(
        jet_heatmap)
```

```
    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
    display(Image(cam_path))
```

```
path=caminho_base + rede_utilizada + rede +str(qtdepocas) + 'ep.h5'

#Carrrega modelo treinado
path_model = load_model(path)

img_size = (size_global, size_global)

if rede_utilizada=='VGG-16':
 # Ao utilizar VGG-16
 last_conv_layer_name = "block5_pool"

elif rede_utilizada == 'Resnet50':
  # Ao utilizar Resnet50
  last_conv_layer_name = 'conv5_block3_2_relu'

# The local path to our target image
img_path = '/content/drive/MyDrive/Minicurso SBCAS/base/test/benign/ISIC_0000038.jpg'
img_array = preprocess_input(get_img_array(img_path, size=img_size))

# Remove last layer's softmax
path_model.layers[-1].activation = None

# Print what the top predicted class is
preds = path_model.predict(img_array)

# Generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, path_model, last_conv_layer_name)

#Display heatmap
#plt.matshow(heatmap)
#plt.show()

save_and_display_gradcam(img_path, heatmap)
```

A Figura 3.15 mostra os mapas de calor com as regiões de ativação de duas CNNs (VGG-16 e ResNet50). Nos mapas de ativação indicados na Figura 3.15, os tons vermelhos estão associados às regiões que contribuíram consideravelmente para a classificação final, enquanto as demais cores representam outras regiões contribuíram menos.



(a) Benign (b) Malign

(c) VGG-16

(d) Resnet50

Figura 3.15. Exemplos da aplicação do Grad-CAM.

- LIME: o *Local Interpretable Model-agnostic Explanations* (LIME) é uma técnica de explicabilidade de modelos de aprendizado de máquina que busca fornecer *insights* sobre como esses modelos tomam decisões.

  Essa técnica é um "modelo-agnóstico", o que significa que pode ser aplicado a qualquer tipo de modelo, desde que haja uma função de predição disponível. Ele usa uma abordagem de interpretabilidade local, que explora as decisões tomadas por um modelo em relação a instâncias específicas de dados.

  O LIME produz explicações em termos de recursos (*features*) relevantes para uma instância específica, o que ajuda os usuários a entender o motivo pelo qual o modelo tomou uma decisão. O LIME é uma técnica útil para ajudar a garantir a transparência e a responsabilidade em sistemas baseados em aprendizado de máquina. A aplicação do lime pode ser descrita pelo código abaixo.

```
pip install lime
```

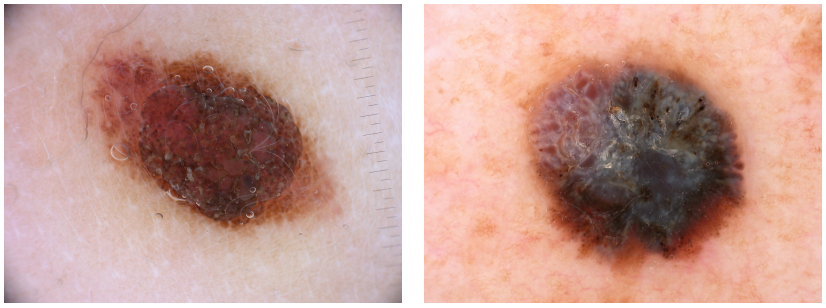
```
import lime
from lime import lime_image
from skimage.segmentation import mark_boundaries
from keras.applications.imagenet_utils import decode_predictions
import os
from skimage.io import imread
import matplotlib.pyplot as plt


%matplotlib inline


import numpy as np
from keras.applications import inception_v3 as inc_net


def transform_img_fn(path_list):
    out = []
    for img_path in path_list:
        img = keras.utils.load_img(img_path, target_size=(299,
    299))
        x = keras.utils.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        x = inc_net.preprocess_input(x)
        out.append(x)
    return np.vstack(out)


images = transform_img_fn([os.path.join(img_path)])


preds = path_model.predict(images)




%load_ext autoreload
%autoreload 2
import os,sys
try:
    import lime
except:
    sys.path.append(os.path.join('..', '..')) # add the current
    directory
    import lime
from lime import lime_image

explainer = lime_image.LimeImageExplainer()
```

```
explanation = explainer.explain_instance(images[0].astype('double'
    ), path_model.predict, top_labels=5, hide_color=0, num_samples
    =10)
```

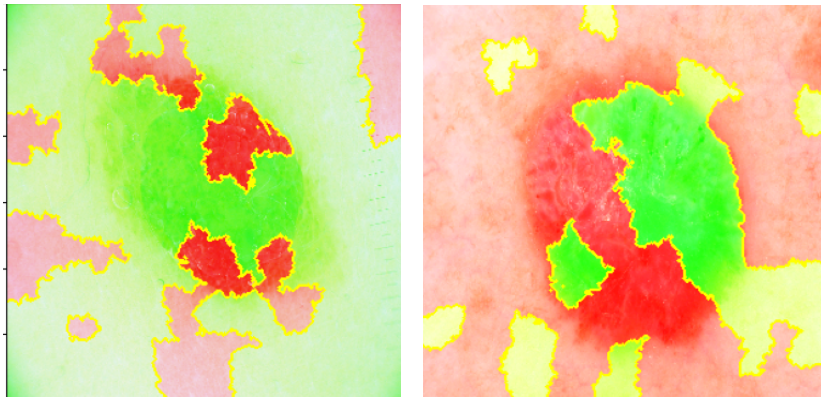
```

from skimage.segmentation import mark_boundaries
```

```
temp, mask = explanation.get_image_and_mask(explanation.top_labels
    [0], positive_only=False, num_features=100, hide_rest=False)
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
```

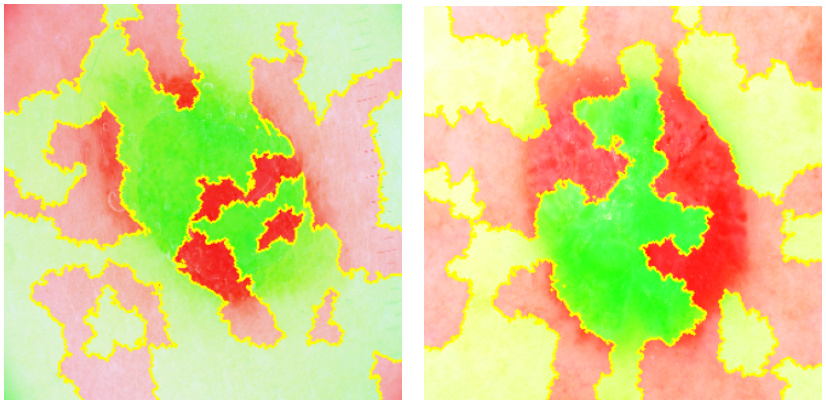
A Figura 3.16 mostra os mapas gerados pelo lime, as regiões marcadas em verde contribuíram significativamente para a classificação, enquanto em oposição as regiões em vermelho contribuem negativamente.



(a) Benign (b) Malign



(c) VGG-16



(d) Resnet50

Figura 3.16. Exemplos da aplicação do LIME.

- t-SNE: nessa seção será trabalhada a explicabilidade nos espaços de características com o algoritmo *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [van der Maaten and Hinton 2008].

O t-SNE é um algoritmo que permite a redução e visualização de dados de altas dimensões. Por meio dessa técnica, é possível representar características extraídas por CNNs em alto nível, permitindo criar novas representações que justifiquem como o aprendizado ocorreu e quais características são cruciais para a separação das classes utilizando um classificador linear. Sua implementação é apresentada pelo seguinte código.

```
from __future__ import print_function
import time
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D

import keras
from keras.applications.vgg16 import preprocess_input
from keras_preprocessing import image
from keras.models import Model, Sequential
from keras.layers import Input, Dense, Dropout, Flatten,
    BatchNormalization

import os
import glob

import sklearn
from sklearn.svm import SVC
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier as rdf
from sklearn import metrics
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import precision_score, f1_score,
    recall_score, cohen_kappa_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate,
    cross_val_predict
from sklearn.manifold import TSNE

vgg16_model = keras.applications.vgg16.VGG16(include_top=True,
    weights='imagenet')
model = Sequential()

for layer in vgg16_model.layers[:-1]:
    model.add(layer)

def extracao_vetor_CNNs(files, classe, model):
    print('Iniciando extracao...')
    modelo = 1
    # Selecionando a VGG-16 como descritor
    # Definindo a camada a ser extraida, a "fc2" e a ultima camada
    totalmente conectada antes da ativacao.
    layer_name = 'fc2'
```

```
    tamanho = (224, 224)
    print('Modelo VGG-16 escolhido...')
    # model.summary()
    # Definindo a matriz que ira conter as caracteristicas extraidas
    feature_list = []
    # Para todas as imagens definidas no diretorio, faca:
    for f1 in files:
        print(f1)
        # Carregando imagem original e redimensionando para 224 x224
        img = image.load_img(f1, target_size=tamanho)

        # Transformando a imagem um array
        img_data = image.img_to_array(img)
        img_data = np.expand_dims(img_data, axis=0)
        img_data = preprocess_input(img_data)

        # A partir do modelo, definimos que a saida da CNN e a camada objetivo (fc2) utilizando model.get_layer.output
        #modelo_camada_intermediaria = Model(inputs=model.input,
         #                                   outputs=model.get_layer('fc1').output)

        # Passando a imagem de entrada atraves do modelo pre-treinado
        saida_atual = model.predict(img_data)
        print(saida_atual.shape)

        # Concatenando a classe ao vetor extraido
        saida_atual = np.append(saida_atual, classe)

        # Concatenando o vetor da imagem i extraida na matriz de caracteristicas final
        feature_list.append(saida_atual.flatten())

    print('Extracao concluida!! A matriz de caracteristicas foi construida!!')
    return feature_list


def construcacao_matriz(dir_doente, dir_saudavel, model):
    # Carregando os diretorios das imagens doentes
    data_path_doente = os.path.join(dir_doente, '*')
    files_doente = glob.glob(data_path_doente)
    # Definindo a classe doente como 1
    classe_doente = 1
    # Extraindo a matriz de caracteristicas
    matriz_doente = np.array(extracao_vetor_CNNs(files_doente, classe_doente, model))

    # Carregando os diretorios das imagens saudaveis
    data_path_saudavel = os.path.join(dir_saudavel, '*')
    files_saudavel = glob.glob(data_path_saudavel)
    # Definindo a classe saudavel como 0
```

```
    classe_saudavel = 0
    # Extraindo a matriz de caracteristicas
    matriz_saudavel = np.array(extracao_vetor_CNNs(files_saudavel,
     classe_saudavel, model))

    # Concatenando as matrizes doente e saudavel.
    matriz_final = np.concatenate((matriz_doente, matriz_saudavel)
    , axis=0)

    return matriz_final

classe_0_teste = 'path\\'
classe_1_teste = 'path\\'

vetor_teste = construcacao_matriz(classe_0_teste,classe_1_teste,
    model)

classe_0_treino = 'path\\'
classe_1_treino = 'path\\'

vetor_treino = construcacao_matriz(classe_0_treino,classe_1_treino
    ,model)

## T-SNE

# Carregando as 1000 primeiras caracteristicas de cada imagem
pca = PCA(n_components=100)

data_X = vetor_treino[:,0:4096]
#data_X = pca.fit_transform(data_X)

data_Y = vetor_teste[:,0:4096]


# Carregando as labels de cada imagem
y = vetor_teste[:,4096]
x = vetor_treino[:,4096]

# Executando o tsne
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=0, perplexity=50.0)

tsne_obj= tsne.fit_transform(data_Y)

a = numpy.asarray(tsne_obj)
numpy.savetxt("VGG-16_TSNE.csv", a, delimiter=",")

tsne_df = pd.DataFrame({'X':tsne_obj[:,0],
                        'Y':tsne_obj[:,1],
                        'Classificacao':y})
tsne_df.head()

#classificador_SVM = svm.SVC(kernel='linear', gamma=0.01, C = 1)
classificador_SVM = rdf()
classificador_SVM.fit(data_X, x)
```

```
142
143 teste_A = classificador_SVM.predict(data_Y)
144 print(teste_A.shape)
145 print(y.shape)
146 #predito_SVM = cross_validation.cross_val_predict(
        classificador_SVM, data_X, y, cv=10)
147 print('Experimento A :------------------------------')
148 acuracia_SVM = metrics.accuracy_score(y, teste_A)
149 print('Acc:', acuracia_SVM)
150 precisao_SVM = precision_score(y, teste_A)
151 print('Precisao weighted:', precisao_SVM)
152 kappa_SVM = cohen_kappa_score(y, teste_A)
153 print('Kappa:', kappa_SVM)
154 recall_SVM = recall_score(y, teste_A)
155 print('Recall weighted:',recall_SVM)
156 cm = confusion_matrix(y, teste_A)
157 print(cm)
158
159 print('Experimento B:--------------------------------')
160 SVM_B = svm.SVC(kernel='linear', gamma=0.01, C = 1)
161 teste_B = cross_val_predict(SVM_B, data_Y, y, cv=10)
162 print('---------SVM----------')
163 acuracia_SVM = metrics.accuracy_score(y, teste_B)
164 print('Acc:', acuracia_SVM)
165 precisao_SVM = precision_score(y, teste_B)
166 print('Precisao weighted:', precisao_SVM)
167 kappa_SVM = cohen_kappa_score(y, teste_B)
168 print('Kappa:', kappa_SVM)
169 recall_SVM = recall_score(y, teste_B)
170 print('Recall weighted:',recall_SVM)
171 cm = confusion_matrix(y, teste_B)
172 print(cm)
173
174
175 from mlxtend.plotting import plot_decision_regions
176 import mlxtend
177 import numpy as np
178 import matplotlib.pyplot as plt
179 from sklearn import datasets, svm
180
181 y_t = np.int64(y)
182
183 clf = svm.SVC(kernel = 'linear', gamma=0.01, C = 1,
        decision_function_shape='ovo')
184 clf.fit(tsne_obj, y_t)
185
186 # Plot Decision Region using mlxtend's awesome plotting function
187 plot_decision_regions(X=tsne_obj,
188                       y=y_t,
189                       clf=clf,
190                       legend=2)
191
192 # Update plot object with X/Y axis labels and Figure Title
193 plt.xlabel('Analise de caracteristicas', size=14)
194 #plt.ylabel(X.columns[1], size=14)
```

```
plt.title('SVM Decision Region Boundary', size=16)
plt.savefig('TSNE_VGG_SAMPLE.png')
```

3.5. Conclusão

O trabalho demonstrou a definição da base de imagens, bem como seu pré-processamento. Além disso, abordou a organização e criação de duas CNN a partir do conjunto ImageNet. Essas CNN's foram treinadas e em seguidas testadas com a finalidade de verificar a quantidade de predições realizadas corretamente. Após obter as predições, aplica-se a explicabilidade nas imagens preditas corretamente.

Conforme a saída da técnica de explicabilidade adotada, pode-se entender quais características influenciaram no processo de classificação das CNN's, ou seja, é possível justificar a tomada de decisão. Logo, a utilização da explicabilidade traz um grande benefício para o meio científico e consequentemente uma maior aceitação em trabalhos clínicos, pois aumenta a distância do padrão "caixa-preta", no qual o porquê da saída da CNN é desconhecido.

Ainda, um direcionamento futuro para trabalhar com explicabilidade seria buscar otimizar o desempenho das CNNs por meio de mudanças de parâmetros, utilização de pré-processamento das imagens, técnicas de aumento de dados com transformações de espaço de cores ou transformações baseadas em *Deep Learning*. Outro ponto relevante seria a aplicação de métodos intrínsecos sobre as imagens para uma explicação mais transparente.

Referências

[Acharya and Ray 2005] Acharya, T. and Ray, A. K. (2005). *Image processing: principles and applications*. John Wiley & Sons.

[Bejnordi et al. 2018] Bejnordi, B. E., Mullooly, M., Pfeiffer, R. M., Fan, S., Vacek, P. M., Weaver, D. L., Herschorn, S., Brinton, L. A., van Ginneken, B., Karssemeijer, N., Beck, A. H., Gierach, G. L., van der Laak, J. A. W. M., and Sherman, M. E. (2018). Using deep convolutional neural networks to identify and classify tumor-associated stroma in diagnostic breast biopsies. *Modern Pathology*, 31:1502–1512.

[Chen et al. 2019] Chen, C., Li, O., Tao, D., Barnett, A., Rudin, C., and Su, J. K. (2019). This looks like that: deep learning for interpretable image recognition. *Advances in neural information processing systems*, 32.

[Chollet 2017] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1800–1807.

[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 *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Devlin et al. 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Dundas and Chik 2011] Dundas, J. and Chik, D. (2011). Implementing human-like intuition mechanism in artificial intelligence. *arXiv preprint arXiv:1106.5917*.

[Erhan et al. 2009] Erhan, D., Bengio, Y., Courville, A., and Vincent, P. (2009). Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1.

[Fukushima 1988] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119 – 130.

[Furat et al. 2023] Furat, O., Kirstein, T., Leißner, T., Bachmann, K., Gutzmer, J., Peuker, U. A., and Schmidt, V. (2023). Multidimensional characterization of particle morphology and mineralogical composition using ct data and r-vine copulas.

[Gomes and Velho 1997] Gomes, J. and Velho, L. (1997). *Image processing for computer graphics*. Springer Science & Business Media.

[Gonzalez et al. 2002] Gonzalez, R. C., Woods, R. E., et al. (2002). Digital image processing.

[Goyal et al. 2017] Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. (2017). Making the v in vqa matter: Elevating the role of image understanding in visual question answering.

[Guo et al. 2017] Guo, Z., Chen, Q., Wu, G., Xu, Y., Shibasaki, R., and Shao, X. (2017). Village building identification based on ensemble convolutional neural networks. *Sensors*, 17:2487.

[Gupta et al. 2018] Gupta, A., Harrison, P., Wieslander, H., Pielawski, N., Kartasalo, K., Partel, G., Solorzano, L., Suveer, A., Klemm, A., Spjuth, O., Sintorn, I.-M., and Wählby, C. (2018). Deep learning in image cytometry: A review. *Cytometry Part A*, 95.

[Gutman et al. 2016] Gutman, D., Codella, N. C. F., Celebi, M. E., Helba, B., Marchetti, M. A., Mishra, N. K., and Halpern, A. (2016). Skin lesion analysis toward melanoma detection: A challenge at the international symposium on biomedical imaging (ISBI) 2016, hosted by the international skin imaging collaboration (ISIC). *arXiv*.

[Hachilif et al. 2019] Hachilif, R., Baghdadi, R., and Benhamida, F. (2019). *Graduation Thesis Implementing and Optimizing Neural Networks using Tiramisu*. PhD thesis.

[Haykin 2007] Haykin, S. (2007). *Redes neurais: princípios e prática*. Bookman Editora.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE Computer Society.

[Hou and Zhou 2018] Hou, B.-J. and Zhou, Z.-H. (2018). Learning with interpretable structure from rnn. *arXiv preprint arXiv:1810.10708*.

[Hu et al. 2018] Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141.

[Huang et al. 2017] Huang, G., Liu, Z., v. d. Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.

[Jia et al. 2014] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678.

[Krizhevsky et al. 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Landis and Koch 1977] Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174.

[Lapuschkin et al. 2016] Lapuschkin, S., Binder, A., Montavon, G., Muller, K.-R., and Samek, W. (2016). Analyzing classifiers: Fisher vectors and deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2912–2920.

[Lecun et al. 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lecun et al. 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

[Li et al. 2016] Li, J., Monroe, W., and Jurafsky, D. (2016). Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*.

[Li et al. 2017] Li, J., Monroe, W., and Jurafsky, D. (2017). Understanding neural networks through representation erasure.

[Lundberg and Lee 2017] Lundberg, S. and Lee, S.-I. (2017). A unified approach to interpreting model predictions.

[Luong et al. 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

[Murdoch and Szlam 2017] Murdoch, W. and Szlam, A. (2017). Automatic rule extraction from long short term memory networks,[in:] international conference on learning representations. *Toulon, France, April*, pages 23–26.

[Pacheco and Pereira 2018] Pacheco, C. A. R. and Pereira, N. S. (2018). Deep learning conceitos e utilização nas diversas áreas do conhecimento. *Revista Ada Lovelace*, 2:34–49.

[Ramzan et al. 2019] Ramzan, F., Khan, M. U., Rehmat, A., Iqbal, S., Saba, T., Rehman, A., and Mehmood, Z. (2019). A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using resting-state fmri and residual neural networks. *Journal of Medical Systems*, 44.

[Rezende 2003] Rezende, S. O. (2003). *Sistemas inteligentes: fundamentos e aplicações*. Editora Manole Ltda.

[Ribeiro et al. 2016a] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016a). "why should i trust you?"explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

[Ribeiro et al. 2016b] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016b). Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*.

[Ronneberger et al. 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597.

[Russakovsky et al. 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Selvaraju et al. 2017] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626.

[Shaees et al. 2020] Shaees, S., Naeem, M. R., Naeem, H., Syed, H., Arslan, M., and Aldabbas, H. (2020). Facial emotion recognition using transfer learning.

[Shorten and Khoshgoftaar 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60.

[Simonyan and Zisserman 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

[Sophiya and Jothilakshmi 2018] Sophiya, E. and Jothilakshmi, S. (2018). Large scale data based audio scene classification. *International Journal of Speech Technology*, 21:825–836.

[Sundararajan et al. 2016] Sundararajan, M., Taly, A., and Yan, Q. (2016). Gradients of counterfactuals. *arXiv preprint arXiv:1611.02639*.

[Sundararajan et al. 2017] Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

[Szegedy et al. 2016] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826.

[Szegedy et al. 2015] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

[Taylor and Nitschke 2018] Taylor, L. and Nitschke, G. (2018). Improving deep learning with generic data augmentation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1542–1547. IEEE.

[Uchiyama et al. 2023] Uchiyama, T., Sogi, N., Niinuma, K., and Fukui, K. (2023). Visually explaining 3d-cnn predictions for video classification with an adaptive occlusion sensitivity analysis. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1513–1522.

[van der Maaten and Hinton 2008] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

[Xie et al. 2020] Xie, N., Ras, G., van Gerven, M., and Doran, D. (2020). Explainable deep learning: A field guide for the uninitiated. *J. Artif. Intell. Res.*, 73:329–396.

[Zeiler and Fergus 2014a] Zeiler, M. D. and Fergus, R. (2014a). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.

[Zeiler and Fergus 2014b] Zeiler, M. D. and Fergus, R. (2014b). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer.

[Zeiler et al. 2011] Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *2011 international conference on computer vision*, pages 2018–2025. IEEE.

[Zellers et al. 2019] Zellers, R., Bisk, Y., Farhadi, A., and Choi, Y. (2019). From recognition to cognition: Visual commonsense reasoning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6720–6731.

[Zhou et al. 2016] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929.

