UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e de Computação

JIMI TOGNI

Avaliação de Modelos de Classificação de Imagens de Raios-X para Detecção de COVID-19

Campinas

2021

JIMI TOGNI

Avaliação de Modelos de Classificação de Imagens de Raios-X para Detecção de COVID-19

Dissertação apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica, na Área de Engenharia de Computação.

Orientador: Prof. Dr. Romis Ribeiro de Faissol Attux

Este trabalho corresponde à versão final da dissertação/tese defendida pelo aluno JIMI TOGNI, e orientada pelo Prof. Dr. Romis Ribeiro de Faissol Attux.

Campinas

2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

T572a

Togni, Jimi, 1987-
Avaliação de modelos para classificação de imagens de raios-X para detecção de Covid-19 / Jimi Togni. – Campinas, SP : [s.n.], 2021.

Orientador: Romis Ribeiro de Faissol Attux.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Aprendizado de máquina. 2. Aprendizado por transferência. 3. Covid-19. 4. Redes neurais convolucionais. I. Attux, Romis Ribeiro de Faissol, 1978-. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Comparative analysis of X-ray image classification models for COVID-19 detection
Palavras-chave em inglês:
Machine learning
Transfer learning
Covid-19
Convolutional neural network
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Romis Ribeiro de Faissol Attux [Orientador]
Rafael Ferrari
Ricardo Suyama
Data de defesa: 17-12-2021
Programa de Pós-Graduação: Engenharia Elétrica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-2331-0699
- Currículo Lattes do autor: http://lattes.cnpq.br/6673135275840128

COMISSÃO JULGADORA - DISSERTAÇÃO DE MESTRADO

Candidato(a): Jimi Togni RA: 226359
Data de defesa: 17 de dezembro de 2021
Título da Dissertação: "Avaliação de Modelos de Classificação de Imagens de Raios-X para Detecção de COVID-19"

Prof. Dr. Romis Ribeiro de Faissol Attux (Presidente)
Prof. Dr. Rafael Ferrari (FEEC/UNICAMP)
Prof. Dr. Ricardo Suyama (CECS/UFABC)

A Ata de Defesa, com as respectivas assinaturas dos membros da Comissão Julgadora, encontra-se no SIGA (Sistema de Fluxo de Dissertação/Tese) e na Secretaria de Pós-Graduação da Faculdade de Engenharia Elétrica e de Computação.

Dedico esta dissertação a meu pai José Carlos Togni, mãe Maria Cristina Bozza Togni e irmã Janina Togni.

Agradecimentos

Externo, minha eterna gratidão ao meu orientador Professor Romis Ribeiro de Faissol Attux, que é uma das melhores, provavelmente a melhor, pessoa que já tive a oportunidade de conhecer na vida.

Ao CNPq pelo apoio financeiro.

Aos meus pais, José Carlos Togni e Maria Cristina Bozza Togni, e minha irmã Janina Togni.

Aos Professores Palazzo, Rafael, Levy, por todo o conhecimento compartilhado, paciência e humildade de todos.

Aos colegas de laboratório DSPCOM.

Aos funcionários do SAPPE, CECOM e FEEC.

Graças a estes, mas principalmente, ao meu orientador, estou tendo a oportunidade de realizar meu sonho. Muito obrigado a todos!

O presente trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico – Brasil (CNPq) - Processo número 157956/2019-9.

"Aquele que começou já está na metade da obra: ouse saber, comece." [1]

(Horácio)

[1] "Dimidium facti, qui coepit, habet; Sapere aude, incipe." Primeiro Livro de Cartas, Carta II.

Resumo

Este trabalho tem como objetivo apresentar e analisar comparativamente os resultados obtidos por um conjunto de modelos de redes neurais na tentativa de detectar infecção por COVID-19 em imagens de raios-X. As imagens foram divididas em três categorias: normal, pneumonia e covid. Para a realização dos experimentos, utilizou-se a técnica de transferência de aprendizagem junto a oito redes convolucionais profundas pré-treinadas: SqueezeNet, DenseNet, ResNet, AlexNet, VGG, GoogleNet, ShuffleNet e MobileNet. Optou-se por utilizar estas redes por serem amplamente empregadas e citadas em trabalhos acadêmicos recentes de visão computacional, detecção e classificação de imagens. Como resultado, observou-se que a rede DenseNet obteve a melhor precisão, com 98,17%, com o uso do otimizador ADAM na abordagem multiclasse. A abordagem binária, por sua vez, obteve o melhor resultado de acurácia, de 99,98%, nas redes VGG, ResNet e MobileNet. São apresentados mapas de calor para que se possam avaliar os motivos subjacentes aos diferentes desempenhos.

Palavras-chave: Aprendizado de Máquina, Transferência de Aprendizado, COVID-19, Redes Neurais Convolucionais Profundas.

Abstract

This work has the objective of presenting and analyzing the results obtained with a representative group of neural models trained to detect COVID-19 infection from lung X-ray images. The images were divided into three categories: normal, pneumoni and covid. The experiments were performed using transfer learning over eight pre-trained deep convolutional networks: SqueezeNet, DenseNet, ResNet, AlexNet, VGG, GoogleNet, ShuffleNet and MobileNet. These networks were chosen in view of their widespread use in the literature on computer vision and image classification. The DenseNet obtained the best precision: 98.17% when the ADAM optimizer is employed within a multiclass formulation. The binary approach, on the other hand, led to the best accuracy: 99.98% for the VGG, ResNet and MobileNet. Heat maps are presented to allow the assessment of the reasons underlying the verified performances.

Keywords: Machine Learning, Transfer Learning, COVID-19 Detection, Deep Convolutional Neural Networks.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AG Attention Gates.

ARM Advanced RISC Machine.

CAM Class Activation Mapping.

CNN Convolutional Neural Networks.

COCO Common Objects in Context.

DCNN Deep Convolutional Neural Networks.

EQM Erro Quadrático Médio.

FCN Full Convolutional Networks.

FLOP FLoating-point Operations Per Second.

GAP Global Average Pooling.

GB Giga Byte.

GPU Graphics Processing Unit.

IgG Imunoglobulinas de classe G.

IgM imunoglobulinas de classe M.

ILSVRC Large Scale Visual Recognition Challenge.

MFLOP Mega FLoating-point Operations Per Second.

MSE Mean Squared Error.

RAM Random Access Memory.

RELU Rectified Linear Units.

RGB Red Blue Green.

RMSProp Root Mean Squared Propagation.

RT-PCR Real Time - Polymerase Chain Reaction.

SARS Severe Acute Respiratory Syndrome.

SARS-CoV-2 Severe Acute Respiratory Syndrome Coronavirus 2.

SGD Stochastic Gradient Descent.

SVHN Street View House Numbers.

TL Transfer Learning.

YOLO You Only Look Once.

Sumário

1 Introdução

1.1 Introdução

No ano de 2020, o mundo foi acometido por uma pandemia sanitária provocada por um vírus denominado Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2) (geralmente chamado de 'coronavírus', segundo o trabalho de (ZHANG et al., 2020)). Anteriormente, esse vírus era encontrado apenas em morcegos, mas passou a contaminar seres humanos, não raro com risco de desenvolvimento de quadros mais graves (Secretaria de Saúde de Minas Gerais, 2021). Após um rápido crescimento no número de casos na China, a doença se espalhou pelo planeta. No início de dezembro de 2021, haviam sido confirmados mais de 265 milhões de casos e cerca de 5,2 milhões de mortes no mundo todo de acordo com (Johns Hopkins University and Medicine. COVID-19 Map, 2021).

As duas formas de detecção mais utilizadas são, segundo o Ministério da Saúde do Brasil, os testes realizados por meio das técnicas de Real Time - Polymerase Chain Reaction (RT-PCR) em tempo real e sequenciamento parcial ou total do genoma viral. Outra técnica é o Teste Rápido, que é utilizado para detecção anticorpos imunoglobulinas de classe M (IgM) e Imunoglobulinas de classe G (IgG) contra o coronavírus na pessoa infectada, e tem por base a metodologia de cromatografia de fluxo lateral. Para isso, as amostras humanas que podem ser utilizadas para a realização dos testes são: Soro, Plasma e Sangue, por coleta venosa ou punção digital (PCR-RT., Ministério da Saude Brasil, 2021). Quando é utilizada a técnica de RT-PCR, o resultado pode levar horas ou até mesmo dias, gerando um tempo significativo de espera pelo resultado. Ressalta-se também o custo elevado envolvido na utilização destas técnicas para a detecção (Ministério da Saude Brasil, 2021).

Dessa forma, o objetivo deste trabalho é propor uma análise comparativa de modelos de redes neurais convolucionais profundas construídos, a partir de arquiteturas já existentes, através da técnica de transferência de aprendizado (do inglês Transfer Learning (TL)). A meta é demostrar a capacidade de cada um deles de auxiliar a detecção de um quadro causado pelo vírus SARS-CoV-2 a partir de imagens de raios-X do tórax. As imagens, neste trabalho, são classificadas como: "normal" ou seja, sem presença de características associadas a doenças, "pneumonia" quando características ligadas a um quadro de pneumonia estão presentes nas imagens e "covid" quando são encontrados padrões de contaminação pelo SARS-CoV-2 na imagem de raios-X. Para isso, as redes convolucionais profundas são treinadas com imagens pré-classificadas por um especialista, possibilitando, assim, que o modelo aprenda os padrões e, após seu treinamento, seja capaz de identificar os mesmos padrões em novas imagens. Quando infectado(a), o(a)

paciente tem seus pulmões afetados (ZHANG et al., 2020), ocasionando pneumonia grave, edema pulmonar, síndrome respiratória aguda grave (também conhecida por Severe Acute Respiratory Syndrome (SARS)) ou falência de múltiplos órgãos (ZHANG et al., 2020), gerando um padrão peculiar de infecção nos pulmões quando se analisam as imagens de raios-X. Os padrões nas imagens de cada uma das classificações citadas podem ser identificados com a ajuda de redes neurais profundas pré-treinadas em um conjunto de dados chamado Imagenet, que possui 14 milhões de imagens divididas em mais de 20 mil classes diferentes, ou seja, 20 mil categorias de imagens dos mais variados tipos, imagens estas que vão de cães e gatos a comidas e bebidas. (ImageNet Project, 2021).

As Redes Neurais Convolucionais Profundas (conhecidas como Deep Convolutional Neural Networks (DCNN)) são redes amplamente utilizadas no reconhecimento de imagens (GOODFELLOW; BENGIO; COURVILLE, 2016). Elas têm por base o emprego da operação de convolução, que suscita um comportamento com elementos análogos aos da resposta do córtex visual humano (HASEGAWA et al., 1998). Neurônios individuais do córtex respondem a estímulos apenas em uma região específica do campo visual, conhecida como campo receptivo. Os campos receptivos de diferentes neurônios se sobrepõem parcialmente, cobrindo todo o campo visual.

1.2 Proposta

Neste trabalho, dois tipos diferentes de estratégias foram exploradas para detectar imagens de raios-X associadas à COVID-19. Primeiramente, utilizou-se uma estratégia que aborda o problema de classificação que contém mais de duas classes, comumente chamado, na literatura, de abordagem multiclasse. A segunda estratégia, por sua vez, utiliza a abordagem binária de classificação, que consiste em abordar o problema com apenas dois tipos de classificação possíveis. Para o treinamento de todas as redes neurais utilizadas neste trabalho, utilizou-se da técnica de transferência de aprendizado.

Foram utilizadas as duas abordagens mais comuns na literatura atual para geração, divisão e aumento artificial do tamanho do *dataset* (do inglês *data augmentation*). A utilização do aumento artificial foi necessária neste trabalho, tendo em vista que possuía-se um número relativamente pequeno de imagens para cada classe: assim, evita-se um possível viés no modelo (GOODFELLOW; BENGIO; COURVILLE, 2016). Um primeiro teste foi feito com a criação do *dataset* utilizando a técnica de aumento dos dados de forma virtual online. Nesta abordagem, o *dataset* físico não sofre alteração e as transformações são feitas a cada incremento durante as passagens das épocas de cada modelo, ou seja, cada vez que o modelo executa uma época de treinamento, as imagens são transformadas, de acordo com as especificações definidas para a função que faz essas transformações, tanto para testes com aumento de dados online, quanto offline (GOODFELLOW; BENGIO;

COURVILLE, 2016). Um segundo teste foi feito utilizando a técnica de aumento de dados de forma artificial offline. Neste caso, as imagens sofrem as transformações que foram especificadas e depois ficam salvas nas pastas de suas respectivas classes. Para a execução deste trabalho, optou-se por detalhar apenas os resultados obtidos com a geração do *dataset* de forma online, uma vez que os melhores resultados foram obtidos para essa abordagem (GOODFELLOW; BENGIO; COURVILLE, 2016).

1.3 Organização do Trabalho

O Capítulo 2 apresenta ao leitor o tema abordado neste trabalho, em conjunto com uma revisão de trabalhos pertinentes presentes na literatura.

No Capítulo 3, são discutidos os fundamentos das metodologias de aprendizado de máquina que serão de particular interesse para este trabalho.

No Capitulo 4, disserta-se sobre cada um dos modelos de redes neurais convolucionais utilizados para fazer a transferência de aprendizado.

No Capitulo 5, são detalhados os experimentos que possibilitaram alcançar o objetivo deste trabalho, nele, estão contidos todas as variações, parâmetros, técnicas de aprendizado de maquina utilizados, entre outros detalhes técnicos julgados importantes para a realização deste trabalho.

O Capitulo 6, apresenta os resultados e a análise comparativa envolvendo os modelos utilizados neste trabalho. Consideram-se diferentes abordagens: multiclasse e binária.

Finalmente, a conclusão é apresentada no Capitulo 7.

2 Breve Revisão da Literatura

Neste trabalho, busca-se analisar comparativamente o uso de diversas arquiteturas de redes neurais profundas, em configuração de transferência de aprendizado, no problema de detecção de COVID-19. Uma vez que a pandemia começou a ter maior impacto mundial em 2020, a revisão bibliográfica e a pesquisa se desenvolveram em paralelo ao desenvolvimento da própria área de pesquisa. Foi uma situação bastante nova para nós, e realizar uma revisão bibliográfica típica de uma dissertação de mestrado se tornou tarefa bastante difícil. A opção que fizemos foi discutir trabalhos que foram particularmente relevantes para o desenvolvimento da pesquisa. Faremos isso neste capítulo, e, nos capítulos seguintes, passaremos à discussão dos aspectos específicos de implementação, teste e análise das técnicas empregadas.

Primeiramente, consideramos o trabalho de Basu et al. (BASU; MITRA; SAHA, 2020). Os autores utilizaram a técnica de transferência de aprendizado para classificação de imagens de raios-X, e estipularam 4 classes para treinar os modelos: 'normal', 'pneumonia', 'covid19' e 'outras doenças'. A acurácia geral foi avaliada em 90,13%, e mapas de calor foram usados para interpretar a operação do modelo. Esses mapas demonstraram correlação entre a atenção das redes neurais e elementos de diagnósticos clínicos validados pelos especialistas. Após a divisão do conjunto de dados em duas classes, "A" e "B", os autores treinaram modelos de Convolutional Neural Networks (CNN) a partir do zero, num conjunto de dados "A", para aprender a classificar entre "infectado" e "normal" nas imagens de raios-X. Em seguida, eles substituíram a última camada totalmente conectada da rede pré-treinada por uma nova camada totalmente conectada, tendo o número de neurônios igual aos das quantidades de classes que, no caso deste trabalho, foram quatro: "normal", "outra doença", "pneumonia" e "covid". Os autores utilizaram três modelos populares de redes neurais convolucionais, com o número crescentes de camadas: Alexnet com 8 camadas, VGG com 16 camadas e ResNet com 50 camadas. Inicialmente, todos os modelos foram treinados do zero usando a base de dados "A", para que pudessem aprender a distinguir imagens de raios-X normais e infectados. Após isso, os modelos passavam por um ajuste fino na base de dados "B", usando a estratégia de transferência de aprendizado para aprender a distinguir entre as quatro classes.

No trabalho (JAIN et al., 2021), os autores realizaram uma investigação avaliando três métodos: Inception V3, Xception e ResNeXt. O trabalho lança mão do dataset Kaggle, com 6432 imagens, sendo 5467 separadas para treinamento e 965 para validação. Para permitir o treinamento das redes profundas, realizou-se um processo de aumentação de dados baseado em três operadores: rotação, zoom e compartilhamento de imagens. Os resultados mostraram que o modelo Xception levou ao melhor desempenho, com 97,97%

de acurácia. Esse trabalho é conceitualmente próximo ao nosso. Alguns pontos em comum foram as classes utilizadas para a classificação — "normal", "covid" e "pneumonia" — e a base de dados.

No trabalho (ABBAS; ABDELSAMEA; GABER, 2021), os autores aplicam uma metodologia de transferência de aprendizado para lidar com o problema de classificação de imagens de raios-X. A base é um modelo de construção de rede convolucional denominado DeTraC (Decompose, Transfer and Compose), que busca lidar com irregularidades nas fronteiras das classes do *dataset* por meio de um mecanismo específico. A base para transferência de aprendizado é formada pelos modelos AlexNet, VGG19, GoogleNet, ResNet e SqueezeNet. Foram reportadas acurácias atingidas da ordem de 93,1%, com 100% de sensibilidade. Esse trabalho possui pontos em comum com a proposta desta pesquisa especialmente na escolha de arquiteturas de redes neurais.

Em (JANGAM; BARRETO; ANNAVARAPU, 2021), os autores propõem uma metodologia flexível para uso tanto em imagens de raios-X quanto em tomografias de tórax. Utilizam-se quatro modelos pré-treinados de redes: VGG 19, ResNet 101, DenseNet 169 e WideResNet e cinco bases de dados diferentes. Emprega-se aumentação de dados com quatro operadores, e se realizam diversos testes variando arquiteturas e hiperparâmetros. Reporta-se, para todas as bases, desempenhos máximos bastante satisfatórios, com acurácias sempre acima de 90%.

Em (GAÁL; MAGA; LUKÁCS, 2020), os autores fizeram um trabalho que teve por objetivo produzir máscaras de segmentação de órgãos em radiografias de tórax, ou seja, para entrada de imagens que necessitam de previsões densas em termos de pixels, sobre se o pixel fornecido faz parte do pulmão esquerdo, do pulmão direito, do coração ou nenhum deles. Para este propósito, o grupo utilizou redes totalmente convolucionais (conhecidas também como Full Convolutional Networks (FCN)), mais especificamente uma arquitetura U-Net, permitindo assim calcularem com eficiência a máscara de segmentação na mesma resolução das imagens de entrada. A arquitetura totalmente convolucional também permite o uso de imagens de diferentes resoluções, uma vez que ao contrário de redes convolucionais padrão, as FCNs não contêm camadas dependentes do tamanho da entrada. O grupo demonstrou que, para tarefas de análise de imagens médicas, a integração da proposta de Attention Gates (AG) melhorou a precisão dos modelos de segmentação, preservando a eficiência computacional. A arquitetura proposta recebeu o nome de Attention U-Net. Segundo os autores, sem o uso de AGs, é prática comum usar CNNs em cascata, selecionando uma região de interesse com outra CNN onde o órgão-alvo provavelmente está contido. Com o uso de AGs, segundo relato dos autores, elimina-se a necessidade da rede de pré-seleção, em vez disso, o modelo criado pelo grupo aprende a se concentrar nas características locais mais importantes, e ignorar as menos relevantes. O grupo notou que focar a atenção em locais menos importantes também resulta em taxas

de falsos positivos menores.

Em (OZTURK et al., 2020), os autores propõem um novo modelo para detecção automática de COVID-19 usando radiografia de tórax, o qual se adapta aos casos de classificação binária — classes 'covid' e 'não encontrado' — e ternária — classes 'covid' e 'não encontrado' e 'pneumonia'. O modelo proposto pelos autores produziu uma precisão de classificação de 98,08% para os casos de classificação binária e, 87,02% para classificação multiclasse. Esse modelo, que recebeu o nome de DarkNet, foi usado no estudo como um classificador para o sistema de detecção de objetos em tempo real You Only Look Once (YOLO)).

Por fim, gostaríamos de mencionar também o trabalho de (BASSI, 2021), no qual se apresenta uma contribuição baseada em duas camadas de transferência de aprendizado — com uma rede treinada junto à base ImageNet e com uma rede treinada para um dataset de imagens de raios-X (para outras doenças). Foi, ademais, investigado o impacto da segmentação das áreas dos pulmões no desempenho do classificador, de modo a verificar a consistência do desempenho obtido.

3 Fundamentos de Aprendizado de Máquina

Em 1950, Alan Mathison Turing escreveu um artigo com o título "Computing Machinery and Intelligence" (TURING, 1950), no qual tratou da questão da capacidade de os computadores conseguirem realmente pensar. Nas primeiras linhas de seu trabalho, Turing propõe a questão "Máquinas podem pensar?", mas, devido às complicações decorrentes dessa indagação, opta por tratar como equivalente a questão "Existe um computador digital capaz de realizar uma imitação sólida do ser humano?". E, assim, descreve um teste, que chamou de "jogo da imitação", o qual é base para o hoje denominado teste de Turing. Consideramos esse trabalho o marco inicial da área de inteligência artificial, à qual se vincula o campo conhecido como aprendizado de máquina (em inglês *machine learning*).

Uma definição de aprendizado de máquina foi apresentada por Arthur Samuel em 1959, em seu artigo (SAMUEL, 1959). Segundo ele, que atuava fortemente no estudo de jogos de computador, aprendizado de máquina diz respeito a dar aos computadores a capacidade de aprender sem serem explicitamente programados. Em 1997, Tom M. Mitchell (MITCHELL; MCGRAW-HILL, 1997) propôs uma definição de que uma máquina aprende com relação a uma determinada tarefa "T" (do inglês task), utilizando uma métrica de desempenho denominada "P" (do inglês performance), em um tipo de experiência "E" (do inglês experience), se o sistema melhora de forma confiável seu desempenho "P" na tarefa "T", seguindo a experiência "E".

Essas definições mostram que a noção de aprendizado de máquina traz a perspectiva de que sistemas computacionais sejam dinamicamente configurados, a partir de informação presente em dados, para resolver problemas. Esse *modus operandi* contrasta com a perspectiva clássica de que a solução de um problema computacional deve advir de uma análise algorítmica feita, em alto nível, por um especialista humano. Analisaremos agora, de maneira mais detalhada, qual é a base teórica que suporta um paradigma desse tipo.

3.1 Aprendizado Supervisionado e Não-Supervisionado

De certa forma, pode-se afirmar que o aprendizado de máquina tem uma base eminentemente estatística. Em (BISHOP, 1995), o autor declara que aprendizado de máquina diz respeito a um conjunto de métodos que podem detectar padrões nos dados automaticamente, e, então, usar esses padrões detectados para predizer dados futuros ou até mesmo para realizar decisões onde existe incerteza no problema a ser resolvido. No aprendizado de máquina, essa incerteza decorre do caráter informativo dos dados, da finitude dos dados disponíveis e da eventual presença de fatores destrutivos como o ruído.

É comum dividir as estratégias de aprendizado em dois tipos principais: aprendizado supervisionado, também conhecido como preditivo, e aprendizado não-supervisionado ou descritivo (RUSSELL; NORVIG, 2021). Na abordagem supervisionada, o objetivo é, em termos simples, construir um mapeamento (ou um sistema dinâmico) que associe entradas X a saídas Y, tendo por base um conjunto de dados pré-rotulados na forma de pares de entrada e saída. Para cada elemento do conjunto de treinamento, X_i é um vetor de D dimensões contendo atributos ou características. Já a saída Y_i é uma variável que pode ser de natureza discreta / categórica ou contínua. Quando Y_i é uma variável discreta / categórica, o problema é entendido como sendo de classificação ou de reconhecimento de padrões; já quando Y_i possui um valor pertencente ao conjunto dos números reais, o problema é interpretado como uma tarefa de regressão (MURPHY, 2012).

Na abordagem não-supervisionada, também conhecida, de acordo com (BURKOV, 2019), como descritiva, tem-se acesso exclusivamente aos dados de entrada X_i. A perspectiva, então, é de buscar padrões presentes nesses dados, num processo que pode corresponder a uma descoberta de conhecimento. Perceba que, neste caso, não é possível saber a priori as classes ou grupos a que os dados eventualmente pertencem: o que se busca neste tipo de abordagem é identificar padrões no conjunto de imagens e, em certos casos, agrupar os dados segundo esses padrões.

Em complemento à divisão proposta, pode-se mencionar a aprendizagem por reforço, em que o algoritmo busca executar um comportamento específico e recebe uma recompensa do observador quando executa o comportamento que é esperado ou uma punição quando o comportamento é contrário ao que é esperado pelo observador (MITCHELL, 2006). Naturalmente, o processo de tradução de recompensa / punição numa variação efetiva dos parâmetros do modelo é o cerne do processo, e uma discussão dos métodos que permitem que isso ocorra transcende o escopo deste trabalho. Um exemplo a que se pode recorrer é o adestramento canino: quando o cão que está sendo adestrado se comporta de acordo com o comando dado pelo adestrador, recebe um petisco - como recompensa positiva - e, geralmente, uma advertência verbal quando não faz o que lhe foi comandado.

3.2 Redes Neurais de Camada Única

Quando se aborda o problema de classificação, é possível compreender a operação do modelo empregado a partir da definição de funções discriminantes, que são responsáveis, efetivamente, pela separação dos dados em grupos distintos. Em geral, a escolha mais simples de função discriminante consiste de uma combinação linear das variáveis de entrada — sendo os coeficientes da combinação linear parâmetros do modelo. Trabalhar com estruturas lineares é interessante do ponto de vista de simplicidade analítica / computacional, desde que a limitação de capacidade de aproximação associada não seja

um problema incontornável.

O perceptron de Rosenblatt (BISHOP, 1995), uma das bases da teoria de redes neurais, insere-se no contexto de discriminação linear na configuração clássica de camada única, mas também aponta um caminho para a inclusão de não-linearidade caso ocorra uma extensão para uma arquitetura de múltiplas camadas. Veremos com cuidado de que forma essa extensão se faz possível.

3.2.1 Discriminantes Lineares

Funções discriminantes que são ótimas, podem ser determinadas a partir de densidades condicionais de classe por meio do teorema de Bayes (BISHOP, 1995). Em vez de realizar estimativa de densidade, no entanto, podem-se admitir formas funcionais parametrizadas específicas para as funções discriminantes e usar o conjunto de dados de treinamento para determinar valores ideais para os parâmetros. Nesta seção, vamos discorrer sobre algumas formas de discriminante linear e suas propriedades.

3.2.1.1 Duas Classes

Consideremos o problema da classificação em duas categorias / classes (BISHOP, 1995). Introduzimos o conceito de uma função discriminante $y(X)$ de modo que o vetor X seja atribuído à classe C_1 se $y(X) > 0$ e à classe C_2 se $y(X) < 0$. Uma função discriminante linear pode ser expressa da seguinte forma:

$$y(X) = W^T X + w_0. \tag{3.1}$$

onde o vetor W é chamado de vetor de pesos e o parâmetro w_0 é geralmente denominado *bias*. Por vezes, $-w_0$ é chamado de limite (*threshold*). Para densidades condicionais de classe com distribuições normais e matrizes de covariância iguais, um discriminante linear da forma (3.1) é ótimo, de acordo com (BISHOP, 1995).

A expressão em (3.1) tem uma interpretação geométrica simples (DUDA; HART et al., 1973). Nota-se primeiro que a fronteira de decisão $y(X) = 0$ corresponde a um hiperplano no espaço de entrada. Para o caso de um espaço de entrada bidimensional $d = 2$, a fronteira de decisão é uma linha reta, conforme mostrado na Figura 3.1. Se X^A e X^B são dois pontos no hiperplano, então $y(X^A) = 0 = y(X^B)$ dessa forma, usando (3.1), temos $W^T(X^B - X^A) = 0$. Assim, W é perpendicular a qualquer vetor no hiperplano, e, assim, observa-se que W determina a orientação da fronteira de decisão. Se x for um ponto no hiperplano, a distância normal da origem ao hiperplano é dada por:

$$l = \frac{W^T X}{||W||} = -\frac{w_0}{||W||} \tag{3.2}$$

Onde $y(X) = 0$ junto com (3.1) (BISHOP, 1995). Assim, o viés w_0 determina a posição do hiperplano, conforme indicado na Figura 3.1. Existe outra notação diferente que pode-se adotar e, que frequentemente se mostra conveniente se definidos novos vetores $\tilde{W} = (w_0, W)$ e $\tilde{X} = (1, X)$, então pode-se reescrever (3.1) na forma:

$$y(X) = \tilde{W}^T \tilde{X} \tag{3.3}$$

Com esta notação, interpreta-se a fronteira de decisão $y(X) = 0$ como um hiperplano que passa pela origem do espaço $\tilde{X}$ com dimensão (d + 1). Representando a função discriminante linear em (3.1) ou (3.3) em termos de um diagrama de rede, como é feito na Figura 3.2, as entradas $x_1, ..., x_n$ são mostradas como círculos, que são conectados pelos pesos $w_1, ..., w_n$ à integração da saída $y(X)$, e o viés w_{b1} é representado como um peso de uma entrada extra n_1 que é fixo.

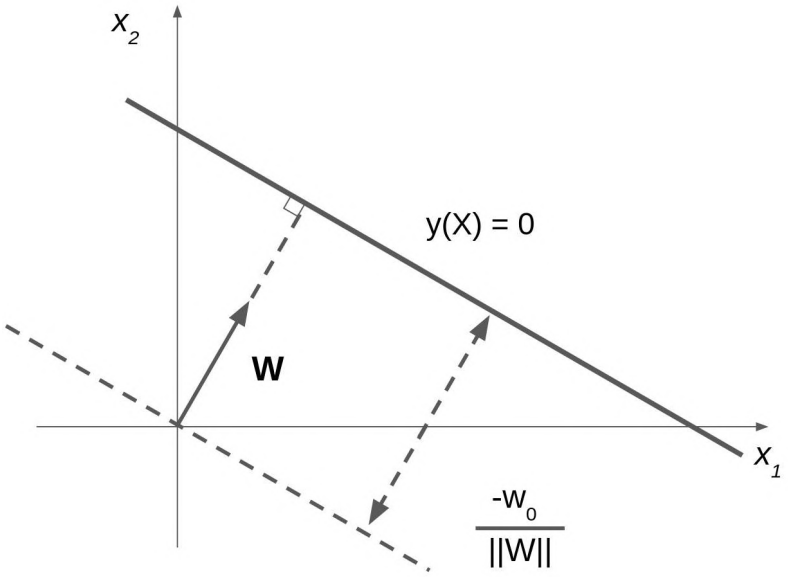


Figura 3.1 – Uma região de decisão linear, correspondendo a $y(X) = 0$, em um espaço de entrada bidimensional x_1, x_2. O vetor de pesos X, que pode ser representado como um vetor no espaço X, define a orientação do plano de decisão, enquanto a polarização também define a posição do plano com respeito à origem. Figura adaptada de (BISHOP, 1995).

3.2.1.2 Múltiplas Classes

De acordo com (BISHOP, 1995), os discriminantes lineares podem ser facilmente estendidos para o caso de c classes, seguindo as ideias apresentadas e usando uma função discriminante $y_k(x)$ para cada classe C_k da forma:

$$y_k(C) = W_k^T X + w_{k0} \tag{3.4}$$

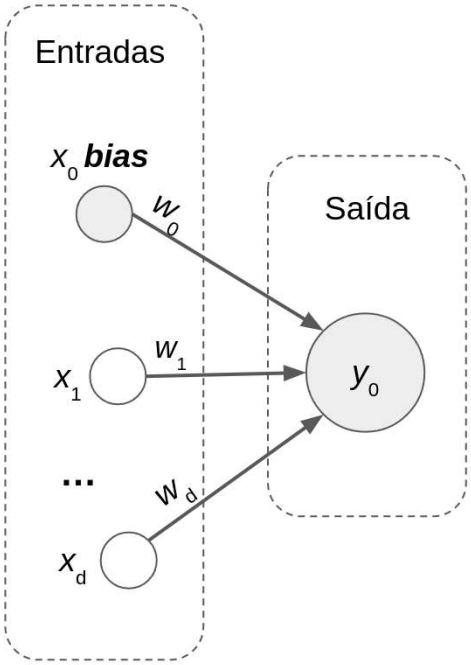


Figura 3.2 – Representação de uma função discriminante linear como um diagrama típico de uma rede neural. Cada componente do diagrama corresponde a uma variável na expressão. O *bias* x_0 pode ser considerado como um parâmetro de peso de uma entrada extra cujo valor está definido como +1. Figura adaptada de (BISHOP, 1995).

Um novo ponto x é então atribuído à classe C_k se $y_k(X) > y_j(X)$ para todo $k \neq j$. As fronteiras de decisão que separam a classe C_k da classe C_j são dadas por $y_k(X) = y_j(X)$ sendo que, para discriminantes lineares, correspondem hiperplanos no formato:

$$(W_k - W_j)^T X + (w_{k0} - w_{j0}) = 0 \tag{3.5}$$

Por analogia com os resultados anteriores obtidos para o caso de discriminante único (3.1), observa-se que a norma para a fronteira de decisão é dada pela diferença entre os dois vetores de peso, e que a distância perpendicular da fronteira de decisão da origem é dada por:

$$l = \frac{(w_{k0} - w_{j0})}{||W_k - W_j||} \tag{3.6}$$

A função de discriminante linear para múltiplas classes (3.4) pode ser expressa em termos de um diagrama de rede neural, como demonstrado na Figura 3.3 (BISHOP, 1995). Os círculos no direita do diagrama, correspondendo às funções $y_k(W)$ em (3.4), que, por vezes são chamados de unidades de processamento, e a avaliação das funções

discriminantes podem ser vistas como um fluxo de informação das entradas para as saídas. Cada saída $y_k(X)$ está associada a um vetor de peso W_k e um *bias* w_{k0}. Expressando as saídas da rede em termos dos componentes dos vetores W_k, tem-se:

$$y_k(X) = \sum_{i=1}^{d} w_{ki}x_i + w_{k0} \tag{3.7}$$

Então, a conexão entre uma entrada i e uma saída k corresponde a um parâmetro de peso w_{ki}. Considerando os termos de *bias* como pesos de uma entrada extra $x_0 = 1$, obtém-se:

$$y_k(X) = \sum_{i=1}^{d} w_{ki}x_i \tag{3.8}$$

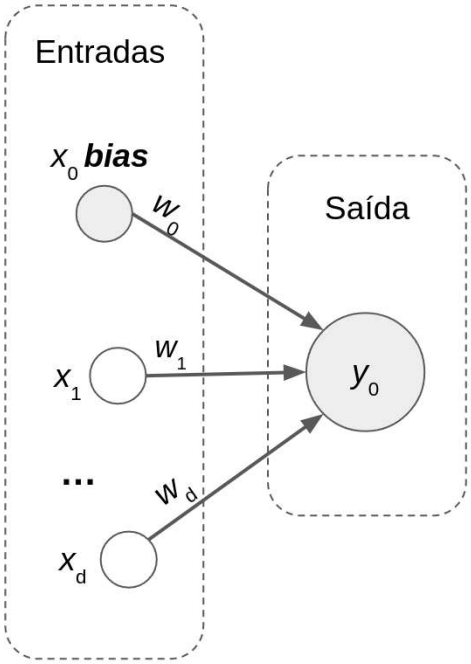


Figura 3.3 – Representação de múltiplas funções discriminantes lineares $y_k(X)$ como um diagrama de rede neural tendo c unidades de saída. Os *bias* são representados como pesos de uma entrada extra, onde $x_0 = 1$. Figura adaptada de (BISHOP, 1995)

Uma vez que a rede é treinada, um novo vetor é classificado aplicando-o às entradas da rede, computando as ativações das unidades de saída e atribuindo o vetor à classe cuja unidade de saída tem a maior ativação. Isso leva a um conjunto de regiões de decisão que são conectadas e convexas. Para isso, consideremos dois pontos X^A e X^B,

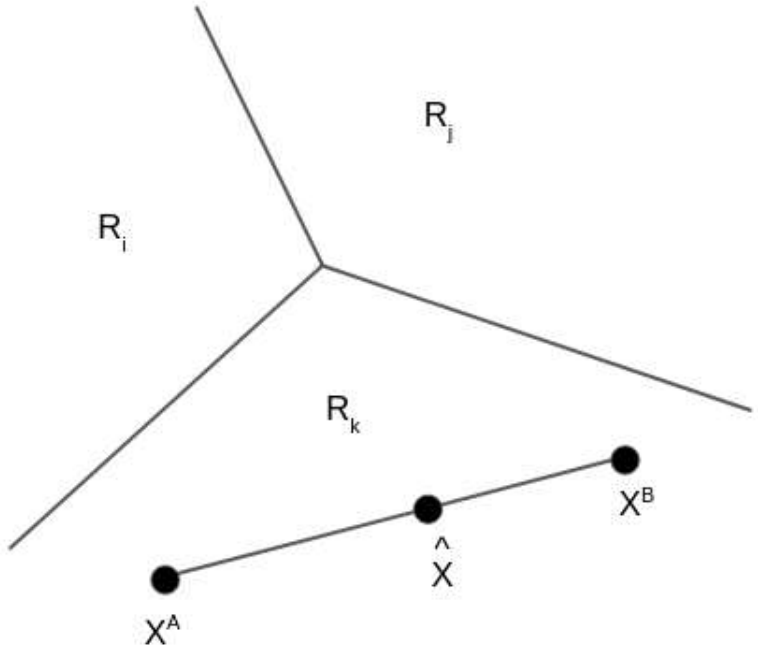


Figura 3.4 – Exemplo das fronteiras de decisão produzidos por uma função de discriminante linear com múltiplas classes. Se dois pontos X^A e X^B estão, ambos na região de decisão R_k, então cada ponto $\hat{X}$ na linha que os conecta também deve estar na região R_k. Portanto, as regiões de decisão devem ser simplesmente conectadas e convexas. Figura adaptada de (BISHOP, 1995)

ambos na região R_k, conforme mostrado na Figura 3.4. Qualquer ponto $\hat{x}$ que se encontra na linha que une x^A e x^B como

$$\hat{X} = aX^A + (1 - a)X^B \tag{3.9}$$

onde $0 \leq a \leq 1$. Como X^A e X^B estão em R_k, eles devem satisfazer $y_k(X^A) > y_j(X^A)$ e $y_k(X^B) > y_j(X^B)$ para todo $j \neq k$. Usando 3.4 e 3.9, segue que $y_k(\hat{X}) = ay_k(X^A) + (1 - a)y_k(X^B)$ e, portanto, $y_k(\hat{X}) > y_j(\hat{X})$ para todo $j \neq k$. Assim, todos os pontos na linha que conecta X^A e X^B também estão em R_k: portanto, a região R_k deve ser simplesmente conectada e convexa.

3.2.2 O Perceptron

Redes de camada única com funções de ativação de limiar foram propostas por Frank Rosenblatt (ROSENBLATT, 1958) e denominadas perceptrons. Essas redes foram aplicadas, à época, a problemas de classificação, em que as entradas eram geralmente imagens de caracteres ou formas simples (BLOCK, 1962). Ao mesmo tempo em que Rosenblatt estava desenvolvendo o perceptron, Widrow e outros trabalhavam em ideia semelhante, usando sistemas conhecidos como adalines (WIDROW; LEHR, 1990). O

termo adaline vem de *ADAptive LINear Element*, e se refere a uma única unidade de processamento com limiar de não-linearidade (WIDROW; HOFF, 1960) considerado *a posteriori* (BISHOP, 1995).

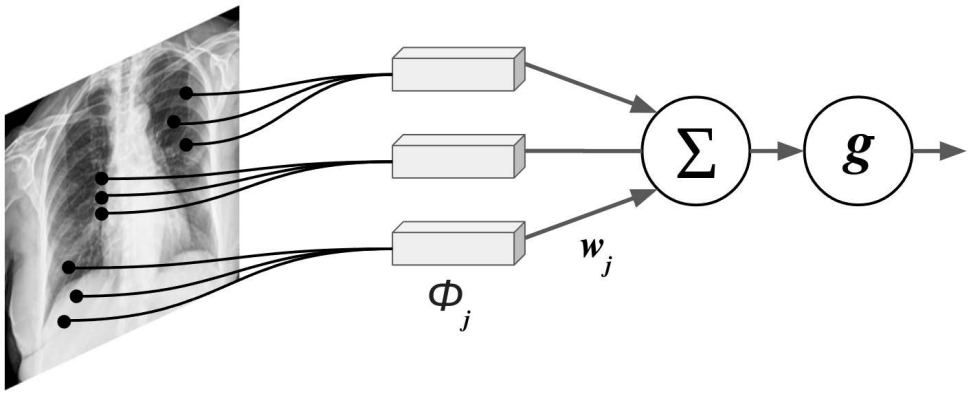


Figura 3.5 – Rede perceptron. Figura adaptada de (BISHOP, 1995)

Na Figura 3.5, mostra-se o uso de um conjunto fixo de elementos de processamento, denotados ϕ_j, seguidos por uma camada de pesos adaptativos w_j e uma função de ativação de limiar $g(\cdot)$. Os elementos de processamento ϕ_j também tinham funções de ativação por limiar e recebiam entradas de um subconjunto de pixels da imagem de entrada.

Como discutido anteriormente, uma rede com uma única camada de pesos tem recursos muito limitados (BISHOP, 1995). Para melhorar o desempenho do perceptron, Rosenblatt usou uma camada de elementos de processamento fixos para transformar os dados de entrada brutos (ROSENBLATT, 1958). Esses elementos de processamento podem ser considerados as funções básicas de um discriminante linear generalizado. Eles, normalmente, tomavam a forma de pesos fixos conectados a um subconjunto aleatório de pixels de entrada, com uma função de ativação de limiar da forma demonstrada na Figura 3.5. Deve-se utilizar a convenção introduzida anteriormente e definir uma função de base extra ϕ_0, cuja ativação é fixada em $+1$, juntamente com um parâmetro de polarização correspondente w_0-. A saída do perceptron é, portanto, dada por

$$y = g\left(\sum_{j=0}^{M} w_j \phi_j(X)\right) = g(W^T \phi) \tag{3.10}$$

onde ϕ denota o vetor formado a partir das ativações ϕ_0, ..., ϕ_m a função de ativação da unidade é mais convenientemente escolhida para ser uma versão anti-simétrica da função de ativação de limite da forma $g(a) = -1$ quando $a < 0$ e $+1$ quando $a \geq 0$.

Sendo o objetivo produzir um sistema de classificação eficaz, seria natural definir a função de erro em termos do número total de classificações erradas no conjunto

de treinamento. De forma geral, pode-se introduzir uma matriz de perda e considerar a perda total incorrida como resultado de uma classificação particular do conjunto de dados. Essas medidas de erro são difíceis de utilizar na prática, pois mudanças suaves nos valores dos pesos e *bias* fazem com que os limites de decisão se movam entre os *data points* do conjunto de dados, resultando em mudanças descontínuas (BISHOP, 1995). A função de erro é constante por partes, e, portanto, procedimentos semelhantes à descida de gradiente não podem ser aplicados. Portanto, houve interesse em outras funções de erro que pudessem ser minimizadas mais facilmente.

Consideremos uma função de erro contínua e linear por partes, chamada de critério do perceptron. À medida que cada vetor de entrada X^n é apresentado às entradas da rede, ele gera um vetor correspondente de ativações ϕ^n nos elementos de processamento da primeira camada. Suponha que associemos a cada vetor de entrada X^n um valor alvo correspondente t^n, de modo que a saída desejada da rede seja $t^n = +1$ se o vetor de entrada pertencer à classe C_1, e $t^n = -1$ se o vetor pertencer à classe C_2. Na Equação 3.10, queremos $W^T\phi^n > 0$ para vetores da classe C_1, e $W^T\phi^n < 0$ para vetores da classe C_2. Portanto, para todos os vetores que queremos ter $W^T(\phi^n t^n) > 0$. Isso sugere que tentamos minimizar a seguinte função de erro, conhecida como critério perceptron:

$$E^{perc}(W) = \sum_{\phi^n \in M} W^T(\phi^n t^n) \tag{3.11}$$

onde M é o conjunto de vetores ϕ_n que são classificados incorretamente pelo vetor de peso atual W. A função de erro $E^{perc}(W)$ é a soma de vários termos positivos e é igual a zero se todos os pontos de dados estiverem classificados corretamente. A partir de funções de discriminante linear, vemos que $E^{perc}(W)$ é proporcional à soma, sobre todos os padrões de entrada que foram classificados incorretamente, das distâncias absolutas até a fronteira de decisão, durante o treinamento, a fronteira de decisão se moverá e alguns pontos que foram classificados incorretamente serão classificados corretamente e vice-versa, de modo que o conjunto de padrões que contribuem para a soma em 3.11 será alterado. O critério do perceptron é, portanto, contínuo e linear por partes com descontinuidades em seu gradiente (BISHOP, 1995).

Quando os perceptrons estavam sendo estudados experimentalmente na década de 1960, descobriu-se que eles podiam resolver muitos problemas rapidamente, enquanto outros problemas, que superficialmente pareciam não ser difíceis, se mostraram impossíveis de resolver. Uma avaliação crítica das capacidades dessas redes, de um ponto de vista matemático formal, foi feita por Minsky e Papert (1969) em seu livro *Perceptrons* (MINSKY; PAPERT, 1969). Eles mostraram que existem muitos tipos de problemas que um perceptron não pode, em nenhum sentido prático, ser usado para resolver. Neste contexto, uma solução é considerada uma classificação correta de todos os padrões no conjunto de treinamento.

Muitos trabalhos recentes sobre redes neurais resumem a contribuição de Minsky e Papert apontando que uma rede de camada única pode apenas classificar conjuntos de dados que são linearmente separáveis, e, portanto, não pode resolver problemas como por exemplo o OR exclusivo (também conhecido como XOR), considerado anteriormente. Os argumentos de Minsky e Papert são bem mais sutis, e apontam para a natureza das redes multicamadas nas quais apenas uma das camadas de pesos é adaptativa. Considere o perceptron mostrado na Figura 3.5 (MINSKY; PAPERT, 1969). A primeira camada de unidades de processamento fixas, e não adaptativas, calcula um conjunto de funções θ_j cujos valores dependem do padrão de entrada (MINSKY; PAPERT, 1969). Mesmo que o conjunto de dados de padrões de entrada possa não ser linearmente separável no espaço das variáveis de entrada originais, o mesmo conjunto de padrões pode se tornar linearmente separável quando transformado no espaço de ϕ_j. Assim, um perceptron pode resolver um problema linearmente não-separável, desde que se tenha um conjunto apropriado de elementos de processamento (BISHOP, 1995).

A dificuldade real com o perceptron surge do fato de que esses elementos de processamento são fixados com antecedência e não podem ser adaptados ao problema específico, ou ao conjunto de dados que está sendo considerado. Como consequência disso, verifica-se que o número, ou complexidade, de tais unidades deve crescer muito rapidamente, geralmente exponencialmente, com a dimensionalidade do problema se o perceptron deve permanecer capaz, em geral, de fornecer uma solução. Portanto, é necessário limitar o número ou a complexidade das unidades da primeira camada (MINSKY; PAPERT, 1969). Minsky e Papert discutem uma gama de diferentes formas de perceptron, dependendo da forma das funções ϕ_j, e para cada uma delas eles fornecem exemplos de problemas que não podem ser resolvidos (BISHOP, 1995).

A solução para a questão do treinamento de redes com múltiplas camadas veio apenas nas décadas de 1970 e 1980, com a ideia de retropropagação de erro, que discutiremos na seção 3.6.

3.3 Redes Neurais de com Múltiplas Camadas

Redes de camada única têm limitações importantes em relação à capacidade de representar funções. Para permitir mapeamentos mais gerais, podem-se considerar transformações sucessivas correspondentes a camadas de redes com pesos adaptativos. É possível observar que redes com apenas duas camadas de pesos são capazes de aproximar qualquer função contínua. De maneira mais geral, pode-se considerar diagramas de rede arbitrários, não necessariamente com uma estrutura em camadas simples, uma vez que qualquer diagrama de rede pode ser convertido em sua função de mapeamento correspondente. A única restrição é que o diagrama deve ser *feed-forward*, de modo que não

contenha laços de realimentação: isso garante que as saídas da rede possam ser calculadas como funções explícitas das entradas e dos pesos (BISHOP, 1995).

Serão revisadas, inicialmente, as capacidades representacionais de redes multicamadas com funções de ativação de limiar ou sigmoidais. Essas redes são geralmente chamadas de perceptrons multicamadas. Para redes com funções de ativação diferenciáveis, existe um método poderoso e computacionalmente eficiente, denominado retropropagação de erro ou *backpropagation*, para encontrar as derivadas parciais de uma função de erro com relação aos pesos e *bias* na rede. Esta é uma característica importante destas redes, uma vez que essas derivadas desempenham um papel central na maioria dos algoritmos de treinamento para redes multicamadas e, portanto, este trabalho trará a retropropagação em certa extensão na Seção 3.6.

3.3.1 Redes Feed-forward

Nesta seção, discutiremos restrições fundamentais às capacidades das redes, e, portanto, suporemos que redes arbitrariamente grandes podem ser construídas, se necessário.

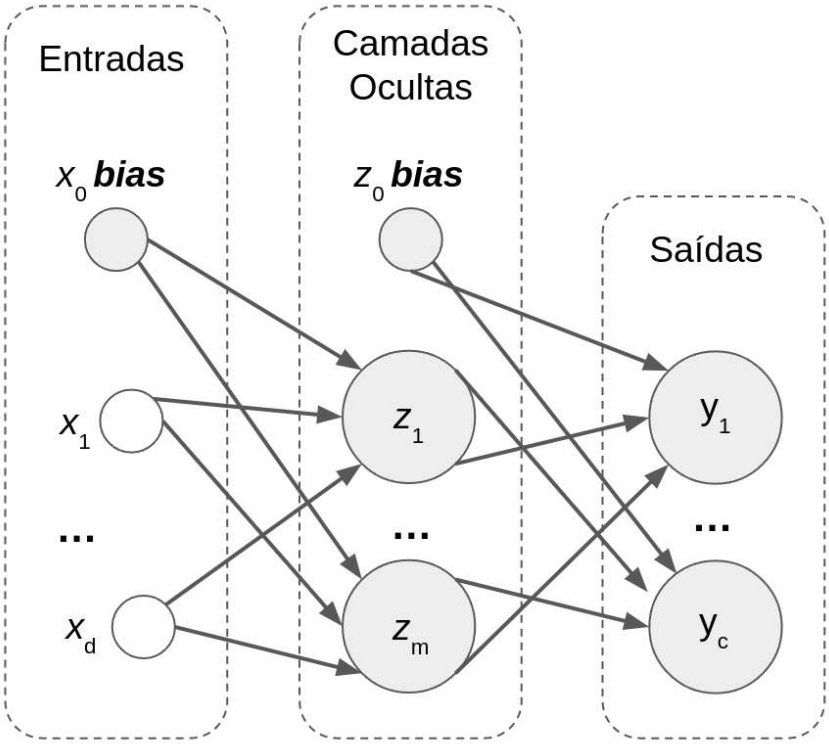


Figura 3.6 – Exemplo de uma rede *feed-forward* que possui duas camadas de pesos adaptativos, o *bias* na primeira camada mostra os pesos de uma entrada extra que tem valor fixo de $x_0 = 1$. De forma similar, a segunda camada, chamada de camada oculta, tem uma unidade extra com peso fixo de $z_0 = 1$. Figura adaptada de (BISHOP, 1995).

De acordo com (BISHOP, 1995), devem-se ver as redes neurais *feed-forward* como fornecendo uma estrutura geral para representar mapeamentos funcionais não-lineares entre um conjunto de variáveis de entrada e um conjunto de variáveis de saída. Isso é obtido representando a função não-linear de muitas variáveis como uma composições de funções não lineares de uma única variável, chamadas funções de ativação. Cada função multivariada pode ser representada na forma de um diagrama de rede, de forma que haja uma correspondência um a um entre os componentes da função e os elementos do diagrama. Da mesma forma, qualquer topologia de diagrama de rede, desde que seja *feed-forward*, pode ser traduzida na função de mapeamento correspondente.

Nesta seção, inicia-se examinando as redes que consistem em múltiplas camadas de pesos adaptativos. Como visto, as redes de camada única são baseadas em uma combinação linear das variáveis de entrada que são transformadas por uma função de ativação não linear. Podem-se construir funções mais gerais considerando redes com múltiplas camadas sucessivas de neurônios com conexões que vão de cada neurônio em uma camada para todos os neurônios na camada seguinte, mas sem outras conexões permitidas. Essas redes em camadas são mais fáceis de analisar teoricamente do que as topologias mais gerais e muitas vezes podem ser implementadas de forma mais eficiente em uma simulação de software, segundo (BISHOP, 1995). Um exemplo de rede em camadas é mostrado na Figura 3.6. As unidades que não são tratadas como neurônios de saída correspondem a unidades intermediárias ou ocultas. Nesta rede, existem d entradas, m unidades ocultas e c unidades de saída. Pode-se escrever a função analítica correspondente à Figura 3.6. A saída da j-ésima unidade oculta é obtida formando primeiro uma combinação linear ponderada dos d valores de entrada:

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} xi + w_{j0}^{(1)}. \tag{3.12}$$

Ainda de acordo com (BISHOP, 1995), aqui $w_{ji}^{(1)}$ denota um peso na primeira camada, indo da entrada i para a camada oculta j, e $w_{j0}^{(1)}$ denota o *bias* para a camada oculta j. Tal como acontece com as redes de camada única, as entradas relativas ao termo de *bias* estão explícitos na Figura 3.6, ocorrendo a inclusão de uma variável de entrada extra x_0 cujo valor é permanentemente definido em $x_0 = 1$. Isso pode ser representado analiticamente reescrevendo a Equação 3.12 na forma:

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} xi. \tag{3.13}$$

A ativação da unidade oculta j é então obtida transformando a soma linear na

Equação 3.13 usando uma função de ativação $g(.)$, então obtém-se (BISHOP, 1995):

$$z_j = g(a_j). \tag{3.14}$$

Consideraremos duas formas principais de função de ativação, dadas respectivamente pela função degrau e por funções sigmoidais contínuas. As saídas da rede são obtidas transformando as ativações dos neurônios das camadas ocultas usando uma segunda camada de neurônios de processamento. Assim, de acordo com (BISHOP, 1995), para cada saída k, construímos uma combinação linear das saídas da camadas oculta imediatamente anterior desta forma:

$$a_k = \sum_{i=1}^{M} w_{kj}^{(2)} zj + w_{k0}^{(2)}. \tag{3.15}$$

Mais uma vez, de acordo com (BISHOP, 1995), pode-se absorver o *bias* nos pesos para obter:

$$a_k = \sum_{i=1}^{M} w_{kj}^{(2)} zj, \tag{3.16}$$

que pode ser representado em diagrama, incluindo uma unidade extra oculta com ativação $z_0 = 1$, como mostrado na Figura 3.6. A ativação da unidade de saída kth é então obtida transformando esta combinação linear usando uma função de ativação não-linear, o que leva a:

$$y_k = \tilde{g}(a_k). \tag{3.17}$$

Usamos a notação $\tilde{g}(.)$ para a função de ativação das camadas de saída para enfatizar que ela não precisa ser a mesma função usada para os neurônios das camadas ocultas. Ainda segundo (BISHOP, 1995), se combinarmos as Equações 3.13, 3.14, 3.16 e 3.17, obtém-se uma expressão explícita para a função completa representada pelo diagrama de rede na Figura 3.6 na forma:

$$y_k = \tilde{g}\left(\sum_{i=1}^{M} w_{kj}^{(2)} g\left(\sum_{i=1}^{d} w_{ji}^{(1)} xi\right)\right). \tag{3.18}$$

Observa-se que, se as funções de ativação para as unidades de saída são consideradas lineares, de modo que $\tilde{g}(a) = a$, esta forma funcional torna-se um caso especial da função discriminante linear generalizada, na qual as funções de base são dados pelas funções particulares z_j definidas nas Equações 3.13 e 3.14. Segundo (BISHOP, 1995), a diferença crucial é que aqui deve-se considerar os parâmetros de peso que aparecem na

primeira camada da rede, bem como os da segunda camada, como sendo adaptativos, para que seus valores possam ser alterados durante o processo de treinamento da rede.

A rede demonstrada na Figura 3.6 corresponde a uma transformação das variáveis de entrada por duas redes sucessivas de camada única. Pode-se estender essa classe de redes considerando outras transformações sucessivas do mesmo tipo geral, correspondendo a camadas extras de pesos. Quando se utiliza o termo "rede com L camadas", referimo-nos a uma rede com L camadas de pesos adaptativos. Portanto, pode-se denominar a rede da Figura 3.6 rede de duas camadas, enquanto as redes da Seção 3.2 são chamadas de redes de camada única. Deve-se notar, entretanto, que uma convenção alternativa as vezes também é encontrada na literatura, em que se contam camadas de unidades em vez de camadas de pesos e as entradas são consideradas unidades separadas. Neste trabalho, não adotaremos esta convenção, uma vez que são as camadas de pesos adaptativos aquelas que são cruciais para determinar as propriedades da rede. Além disso, os círculos que representam as entradas em um diagrama de rede não são verdadeiras unidades de processamento, pois seu único propósito é representar os valores das variáveis de entrada (BISHOP, 1995).

3.4 Aprendizado Baseado no Gradiente

Projetar e treinar uma rede neural não é muito diferente de treinar qualquer outro modelo de aprendizado de máquina. Uma grande diferença entre, por exemplo, modelos lineares e redes neurais jaz no fato de que a não-linearidade de uma rede neural leva a funções de perda tipicamente não-convexas (GOODFELLOW; BENGIO; COURVILLE, 2016). A complexidade trazida pelo caráter não-linear da rede também faz com que seu treinamento tenha de ocorrer ao longo de várias iterações. Classicamente, os otimizadores são baseados no gradiente e na propriedade de que passos na direção contrária do mesmo são minimizantes.

O gradiente descendente estocástico aplicado para funções não convexas de perda não tem convergência garantida e é sensível aos valores que são configurados nos parâmetros iniciais. Ainda de acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), para redes *feed-forward*, é importante inicializar todos os pesos com valores baixos aleatórios. Os *bias* podem ser inicializados com 0 ou com valores positivos baixos. Os modelos de redes neurais com iterações baseadas em algoritmos de gradiente para sua otimização são utilizados na maioria dos modelos de redes neurais profundas.

3.5 Função Custo

De acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), um aspecto fundamental no projeto de redes neurais profundas é a escolha da função custo.

Talvez o critério clássico em aprendizado de máquina seja o de erro quadrático médio (ou Erro Quadrático Médio (EQM)) ou, visto com frequência na literatura como Mean Squared Error (MSE), sendo a função custo associada a uma medida direta da energia da diferença entre a saída do modelo e o sinal de referência. Considerando que $\hat{\boldsymbol{y}}^{(test)}$ representa os valores das previsões que o modelo retorna, então o MSE tendo em vista o conjunto de imagens de teste, é obtido por:

$$MSE_{test} = \frac{1}{m} \sum_{i} \left(\hat{\boldsymbol{y}}_i^{(test)} - \boldsymbol{y}_i^{(test)} \right)^2 . \tag{3.19}$$

O MSE é amplamente usado em regressão, e também pode ser aplicado a uma tarefa de classificação, embora de maneira bem menos orgânica. Para classificação, um critério amplamente utilizado nos dias atuais é o da entropia cruzada, que, de maneira implícita, quantifica a divergência entre a distribuição de probabilidade do dado rotulado e das saídas da rede.

Por vezes, a função custo recebe termos adicionais de penalidade associados, por exemplo, a estratégias de regularização.

3.6 Retroprogação de Erro (Error Backpropagation)

Quando usamos uma rede neural *feed-forward* para aceitar uma entrada z e produzir um saída $\hat{y}$, a informação flui da entrada para a saída da rede. As entradas z fornecem a informação inicial que então se propaga pelos neurônios das camadas ocultas e, finalmente, produz $\hat{y}$ sendo o valor predito pelo modelo neural. Esse processo é chamado de propagação direta. Durante o treinamento, a propagação direta pode continuar até que se produza um custo escalar $J(\phi)$. O algoritmo de retropropagação, de acordo com (RUMELHART; HINTON; WILLIAMS, 1986), muitas vezes chamado simplesmente de *backpropagation* ou *backprop*, permite que as informações do custo fluam para trás através da rede, a fim de calcular o gradiente. Calcular uma expressão analítica para o gradiente é simples, mas, avaliar numericamente tal expressão pode ser custoso do ponto de vista computacional. O algoritmo de retropropagação faz isso usando um procedimento simples e de baixo custo computacional (GOODFELLOW; BENGIO; COURVILLE, 2016).

O termo retropropagação é frequentemente mal-interpretado como significando todo algoritmo de aprendizagem para redes neurais multicamadas. Segundo (RUSSELL; NORVIG, 2021), na verdade, retropropagação refere-se apenas ao método para calcular o gradiente, enquanto outro algoritmo, como a descida do gradiente estocástico, é usado para realizar o aprendizado usando esse gradiente. Além disso, a retropropagação é muitas vezes confundida como sendo específica para redes neurais multicamadas, mas, em princípio, pode permitir o cálculo de qualquer função com uma estrutura de aninhamento.

Especificamente, iremos descrever como calcular o gradiente $\nabla_j f(a, b)$ para uma função arbitrária $f(.)$, onde a é um conjunto de variáveis cujas derivadas são desejadas e b é um conjunto adicional de variáveis que são entradas para a função, mas cujas derivadas não são necessárias. De acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), em algoritmos de aprendizagem, o gradiente que mais frequentemente requer sua computação, é o gradiente da função de custo em relação aos parâmetros $\nabla_\phi f(\phi)$. Muitas tarefas de aprendizado de máquina envolvem a computação de outras derivadas, como parte do processo de aprendizagem ou para analisar o modelo aprendido. O algoritmo de retropropagação também pode ser aplicado a essas tarefas, e não é restrito para calcular o gradiente da função de custo em relação aos parâmetros. Restringiremos nossa descrição aqui ao caso em que $f(.)$ tem uma única saída.

3.6.1 Regra da Cadeia

A regra da cadeia, estudada no âmbito do Cálculo Diferencial, é usada para calcular as derivadas de funções formadas pela composição de outras funções cujas derivadas são conhecidas (GOODFELLOW; BENGIO; COURVILLE, 2016). A retropropagação é um algoritmo que calcula a regra da cadeia com uma ordem específica de operações que é bastante eficiente. Seja x um número real, e sejam f e g ambas funções de mapeamento de um número real para outro número real. Suponha que $y = g(x)$ e $z = f(g(x)) = f(y)$, então, de acordo com a regra da cadeia:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{3.20}$$

Pode-se generalizar esse resultado para além do caso escalar. Assumindo que R representa o conjunto dos números reais, então, supondo que $x \in R^m$, $y \in R^n$, em g mapas que partem de R^m até R^n, e f mapas que vão de R^n para R. Se $y = g(x)$ e $z = f(y)$, então,

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}. \tag{3.21}$$

Em notação vetorial, a Equação 3.21 pode ser escrita da forma:

$$\nabla_x z = \left(\frac{\partial z}{\partial x}\right)^T \nabla_y z, \tag{3.22}$$

onde $\frac{\partial z}{\partial x}$ é o $n \times m$ matriz Jacobiana de g.

A partir disso, vemos que o gradiente de uma variável x pode ser obtido multiplicando uma matriz Jacobiana $\frac{\partial z}{\partial x}$ por um gradiente $\nabla_y z$. O algoritmo de retropropagação consiste de realizar tal produto para cada operação. Em redes neurais, normalmente não

aplicamos o algoritmo de retropropagação meramente a vetores, mas sim a tensores de dimensionalidade arbitrária que, de maneira simplificada / intuitiva, pode-se imaginar um tensor como uma matriz multidimensional. Conceitualmente, o processo é exatamente o mesmo que em retropropagação com vetores. A única diferença é que, como os números são organizados em uma grade para formar um tensor, pode-se imaginar o "achatamento" de cada tensor em um vetor antes de executarmos a retropropagação, calculando um gradiente de valor vetorial, e, em seguida, remodelar o gradiente de volta em um tensor (RUSSELL; NORVIG, 2021).

Para denotar o gradiente de um valor z em relação a um tensor X, escrevemos $\nabla_X z$, exatamente como se X fosse um vetor. Os índices em X agora têm várias coordenadas - por exemplo, um tensor 3D é indexado por três coordenadas. Pode-se abstrair isso usando uma única variável i para representar a tupla completa de índices. Para todos possíveis tuplas de índice i, $(\nabla_X z)i$ dado $\frac{\partial z}{\partial X_i}$. Isso é exatamente o mesmo para todos os possíveis índices inteiros i em um vetor, $(\nabla_X z)i$ dado $\frac{\partial z}{\partial x_i}$. Usando esta notação, nós podemos escrever a regra da cadeia conforme ela se aplica a tensores. Se $Y = g(X)$ e $z = f(Y)$, então,

$$\nabla_X z = \sum_j (\nabla_X Y_j) \frac{\partial z}{\partial Y_j}. \tag{3.23}$$

3.7 Redes Neurais Convolucionais

Redes convolucionais (LECUN et al., 1989), também conhecidas como redes neurais convolucionais ou CNNs (do inglês CNN), são um tipo especializado de rede neural para processamento de dados que têm uma organização sequencial no tempo e/ou no espaço. Os exemplos incluem séries temporais, que podem ser entendidas como grades 1D que coletam amostras em intervalos regulares de tempo, bem como imagens, que podem ser entendidas como uma grade multidimensional de pixels. Redes convolucionais têm sido extremamente bem-sucedidas em aplicações práticas. O epíteto "convolucional" indica que a rede emprega uma operação matemática chamada convolução. A convolução é um tipo especializado de operação linear, e redes convolucionais são, basicamente, redes neurais que usam convolução no lugar da matriz geral de multiplicação em pelo menos uma de suas camadas. Nesta seção, primeiro será descrito o que é convolução, e, em seguida, será demonstrada a motivação por trás do uso de convolução em uma rede neural. Então será exposta a operação de *pooling*, que quase todas as redes convolucionais utilizam (RUSSELL; NORVIG, 2021).

3.7.1 Operação de convolução

Consideremos duas funções $x(t)$ e $w(t)$ a convolução entre elas é dada pela seguinte integral:

$$s(t) = \int x(a)w(t-a)da. \tag{3.24}$$

Pode-se ainda denotar operação de convolução através de um asterisco:

$$s(t) = (x * w)(t). \tag{3.25}$$

Na terminologia de redes convolucionais, o primeiro argumento - neste exemplo, a função x - é considerada uma entrada, e o segundo argumento - neste exemplo, a função w - é chamado de filtro ou *kernel*. A saída é denominada mapa de características. É possível visualizar este contexto na Figura 3.7.

No exemplo, as funções foram consideradas de tempo contínuo. Se o índice temporal t puder assumir apenas valores inteiros, tem-se um caso de tempo discreto. Pode-se definir a convolução discreta como:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{3.26}$$

Em aplicações de aprendizado de máquina, a entrada geralmente é uma matriz ou um tensor de dados e o kernel é também uma matriz ou um tensor de parâmetros que são adaptados pelo algoritmo de aprendizagem. Uma vez que os elementos da entrada e do kernel são armazenados, geralmente assumimos que essas funções são zero em todos os lugares, exceto no conjunto finito de pontos para os quais armazenamos os valores.

Quando se trabalha com imagens, é preciso considerar as convoluções em mais de um eixo por vez. Por exemplo, se usarmos uma imagem I com duas dimensões como nossa entrada, usaremos um kernel K também bidimensional (GOODFELLOW; BENGIO; COURVILLE, 2016), levando à seguinte definição:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.27}$$

A convolução é comutativa, o que significa que pode-se escrever de forma equivalente:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{3.28}$$

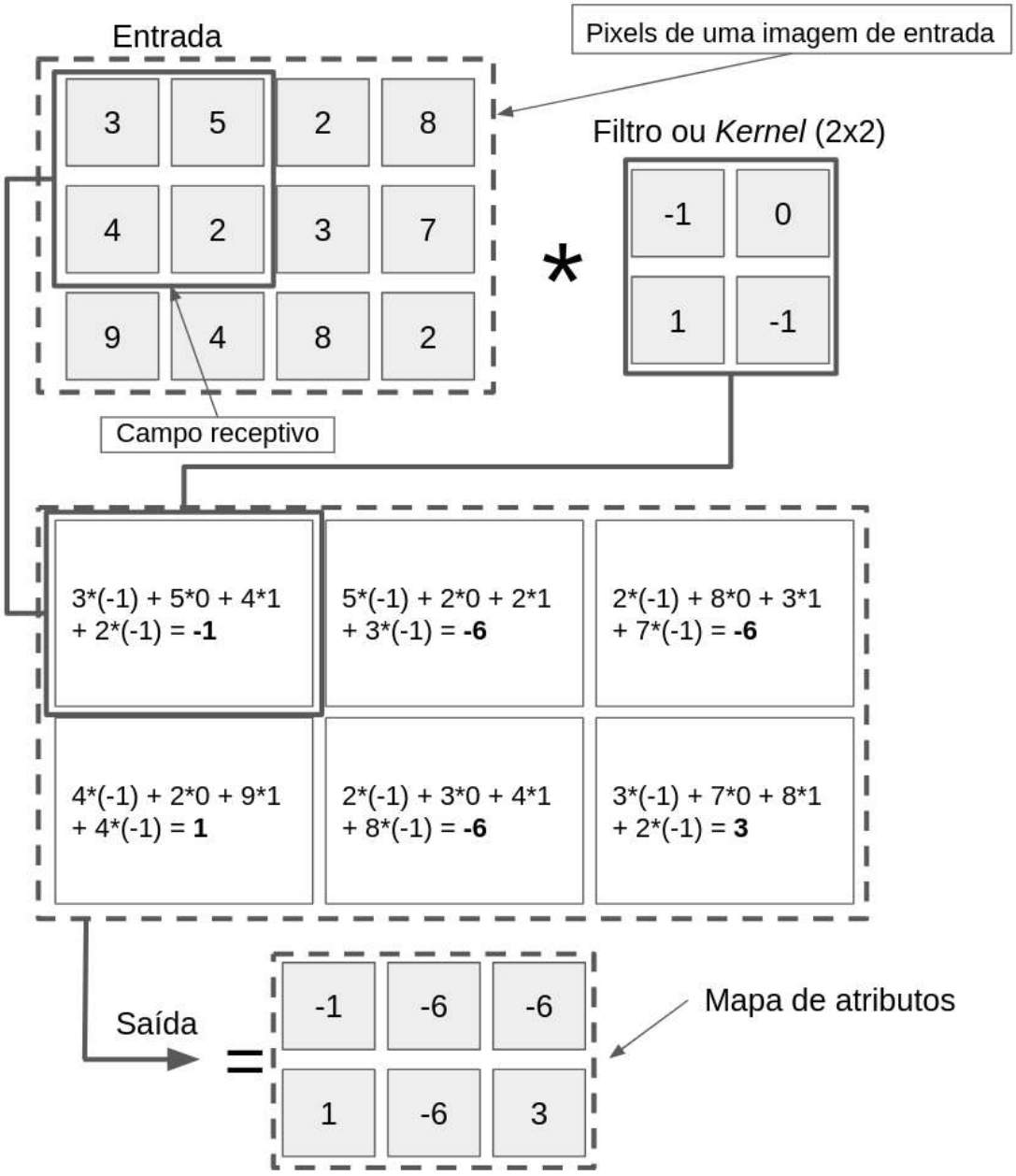


Figura 3.7 – Um exemplo de convolução 2-D sem inversão de kernel. Neste caso, restringe-se a saída a apenas posições onde o kernel está inteiramente dentro da imagem. Utilizou-se caixas com setas para indicar como o elemento superior esquerdo do tensor de saída é formado pela aplicação do kernel à região superior esquerda correspondente do tensor de entrada. Figura adaptada de (GOODFELLOW; BENGIO; COURVILLE, 2016).

Por questões numéricas ou computacionais, as redes neurais são, via de regra, implementadas não com a convolução estrita, mas com uma função chamada de correlação cruzada, na qual o cálculo do resultado é feito sem necessidade de uso de um índice de

soma negativo, ou seja, sem inversão (GOODFELLOW; BENGIO; COURVILLE, 2016):

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{3.29}$$

Muitas bibliotecas de aprendizado de máquina implementam a correlação cruzada com o nome de convolução (GOODFELLOW; BENGIO; COURVILLE, 2016). Neste trabalho, optou-se por convenção chamar de convolução ambas as operações, e especificar se queremos inverter o kernel ou não em contextos em que isso for relevante. Pode-se observar na Figura 3.7 um exemplo de convolução sem inversão de kernel, aplicada a uma matriz, ou seja, um tensor bidimensional.

3.7.2 Padding e Stride

De acordo com (BURKOV, 2019), duas propriedades importantes para a operação de convolução são *Stride* e *Padding*.

O *Padding* faz com que características importantes nas bordas das imagens não sejam perdidas. Também, quando não se deseja obter o resultado de uma convolução que tenha dimensões diferentes da imagem de origem. Nesse caso, é feito o preenchimento com pixeis nas linhas das partes superior e inferior e colunas à direta e à esquerda das imagens, para que as matrizes resultantes tenham o mesmo tamanho da matriz de origem. Segundo (MICHELUCCI, 2019) comumente, utiliza-se o preenchimento de pixeis com o valor zero, como demonstrado na Figura 3.8.

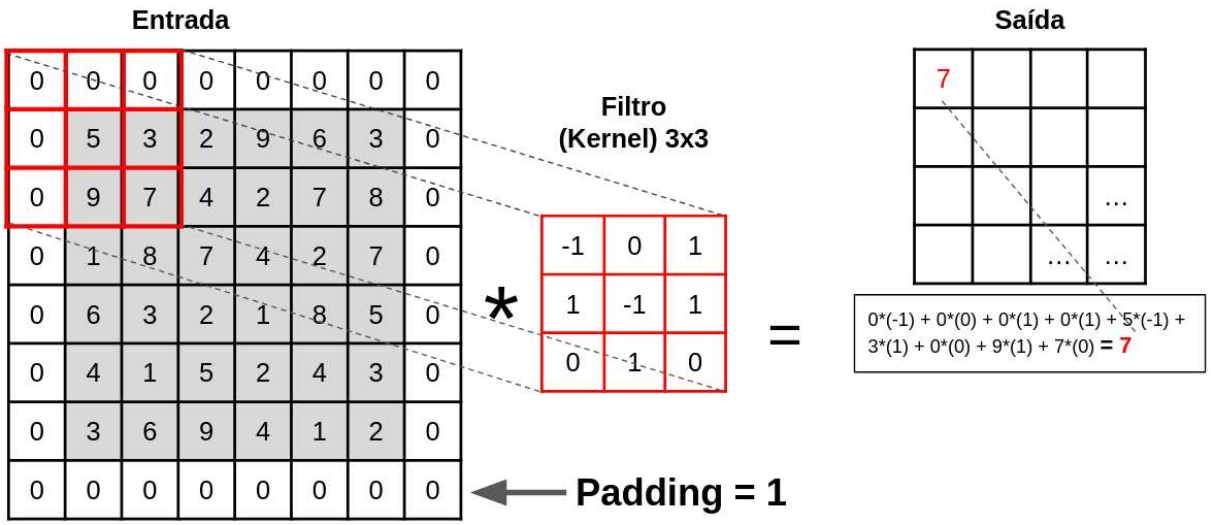


Figura 3.8 – Padding. Figura adaptada de (BURKOV, 2019).

O *Stride* é o valor dado para especificar como será feito o deslocamento do filtro convolucional (BURKOV, 2019). Como demonstrado na Figura 3.9, pode-se observar como o deslocamento é feito, em relação as colunas e linhas da matriz, neste exemplo, o

valor do *Stride* é igual a 1, ou seja, o filtro convolucional será deslocado com o passo de uma coluna e, depois de percorrer a imagem na horizontal o filtro passa para a próxima linha, após cada operação de convolução.

Um *Stride* com valor alto reduz a representação da imagem, com isso, reduz-se também a complexidade computacional em decorrência de fazer menos operações e ajuste dos pesos, porém, quanto maior o valor, menos detalhes nas imagens são reconhecidos (MICHELUCCI, 2019).

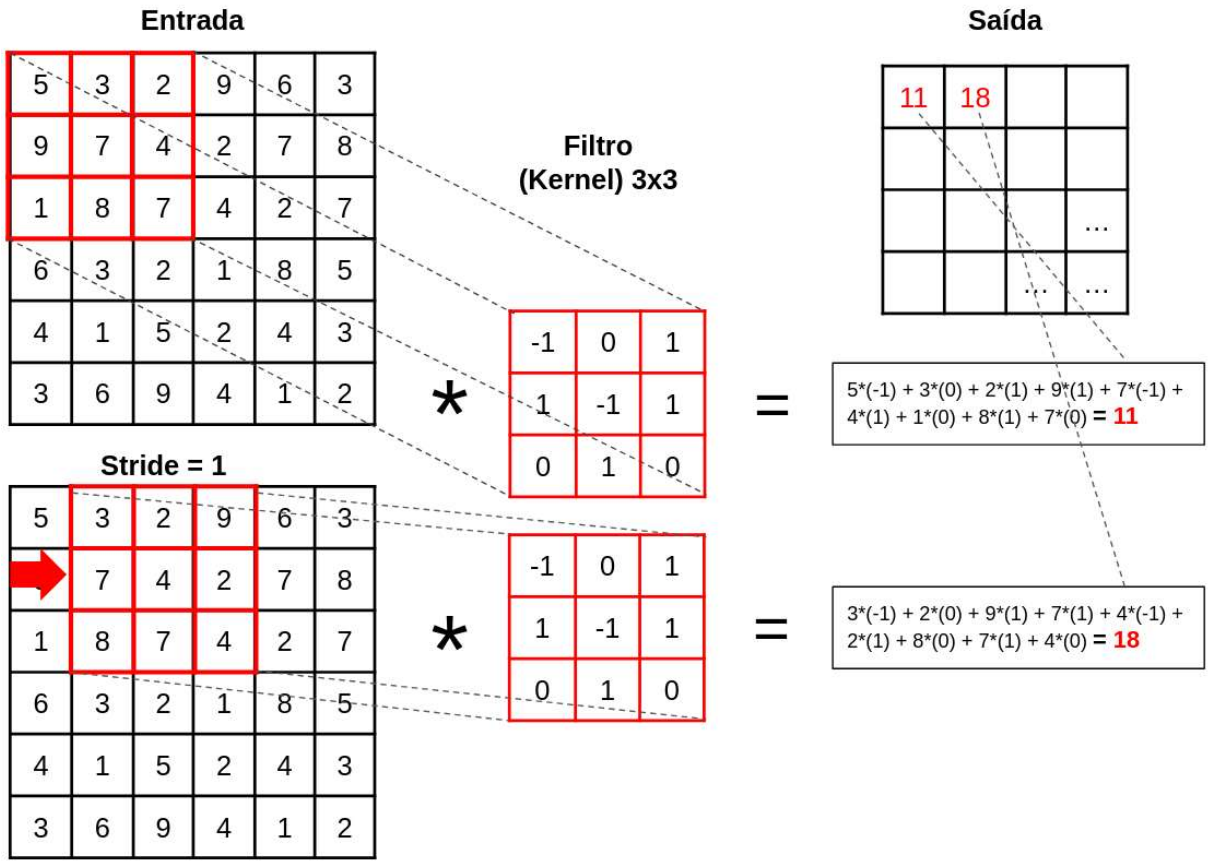


Figura 3.9 – Stride. Figura adaptada de (BURKOV, 2019).

3.7.3 Pooling

De acordo com (RUSSELL; NORVIG, 2021), uma camada típica de uma rede convolucional consiste em três estágios, como pode-se verificar na Figura 3.11. No primeiro estágio, a camada realiza várias convoluções em paralelo para produzir um conjunto de ativações lineares. No segundo estágio, a cada ativação linear, aplica-se uma função de ativação não-linear, como a função de ativação linear retificada (conhecida como Rectified Linear Units (RELU)): esse estágio às vezes é chamado de estágio de detector. Na terceira etapa, pode-se ainda utilizar uma função de agrupamento (*pooling*). Uma função desse tipo substitui a saída da rede em um determinado local por uma estatística resumida

das saídas próximas. Por exemplo, como pode-se observar na Figura 3.10, a operação de *max-pooling* retorna a entrada máxima num formato retangular na vizinhança, já a operação de average-pooling retorna a média da vizinhança que é delimitada pelo tamanho do *kernel* do *pooling* — que tem funcção diferente do *kernel* nas operações de convolução (ZHOU; CHELLAPPA, 1988).

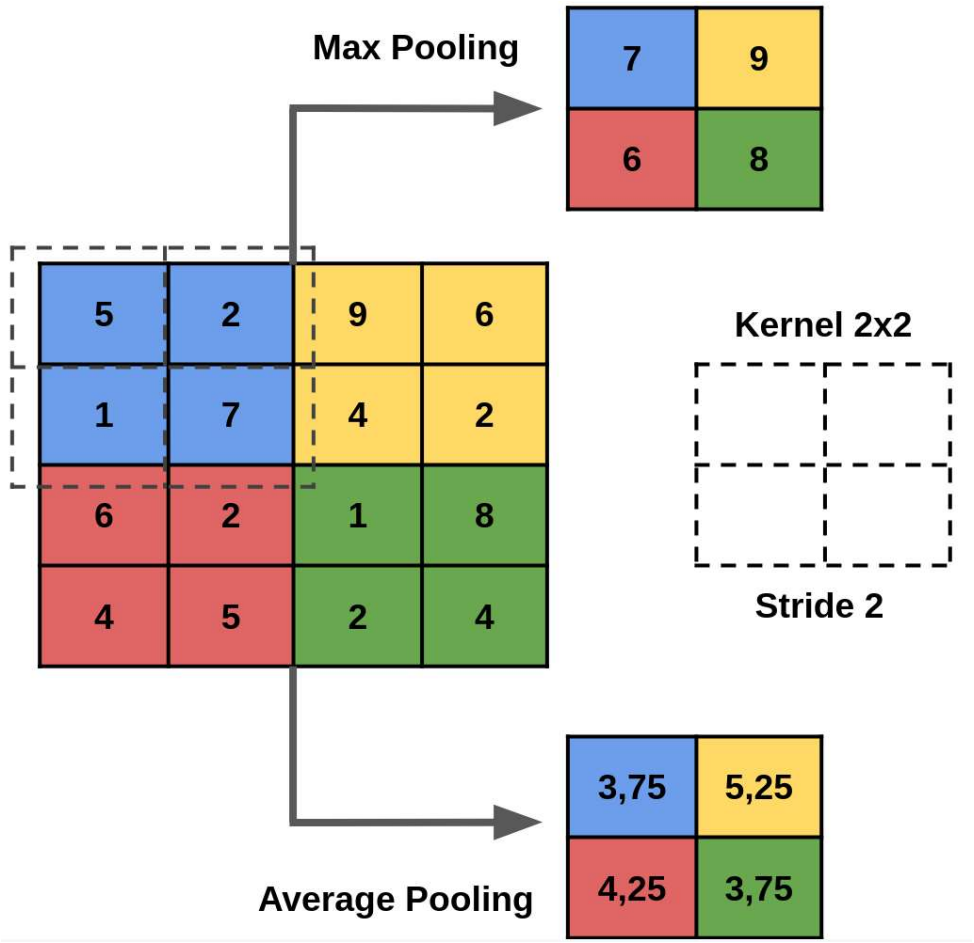


Figura 3.10 – Operações de *Max Pooling* e *Average Pooling* com *Kernel* 2x2 e *Stride* = 2. Figura adaptada de (BURKOV, 2019).

Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), outras funções de *pooling* populares incluem a média de um retângulo pela vizinhança, a norma L^2 de uma vizinhança retangular, ou uma média ponderada com base na distância do pixel central. Em todos os casos, o *pooling* ajuda a tornar a representação aproximadamente invariante a pequenas mudanças da entrada. Invariância à translação de pixels significa que, ainda de acordo com (RUSSELL; NORVIG, 2021), se mudarmos de lugar alguns pixels da imagem de entrada, os valores da maioria das saídas não mudam. Tal invariância pode ser uma propriedade muito útil se nos preocuparmos mais com a presença de um padrão que com sua localização exata.

Agrupamento sobre regiões espaciais produz invariância à translação, mas, se

agruparmos as saídas de convoluções parametrizadas separadamente, os recursos podem aprender a quais transformações se tornar invariante. Como o agrupamento resume as respostas de toda uma vizinhança, é possível usar menos unidades de *pooling* do que unidades de detecção, relatando o resumo estatístico para regiões de *pooling* com a média de k pixeis. Isso melhora a eficiência computacional da rede, porque a próxima camada tem aproximadamente k vezes menos entradas para processar. Quando o número de parâmetros na próxima camada é uma função de seu tamanho de entrada - como quando a próxima camada estiver totalmente conectada e baseada na multiplicação da matriz - isso reduz o tamanho da entrada e também pode resultar em uma eficiência estatística e requisitos de memória reduzidos para armazenar os parâmetros (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para muitas tarefas, o *pooling* é essencial para lidar com entradas de tamanhos variados. Por exemplo, segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), se quisermos classificar imagens de tamanho variável, a entrada para a camada de classificação deve ter um tamanho fixo. Isso geralmente é feito variando o tamanho de um deslocamento entre as regiões de *pool*, de modo que a camada de classificação sempre receba o mesmo número de estatísticas resumidas pelo *pooling*, independentemente do tamanho da entrada. Por exemplo, a camada final de agrupamento da rede pode ser definida para produzir quatro conjuntos de resumos estatísticos, uma para cada quadrante de uma imagem, independentemente do seu tamanho.

Alguns trabalhos na literatura auxiliam na decisão sobre quais tipos de agrupamento devem ser usados em situações variadas (BOUREAU; PONCE; LECUN, 2010). Também é possível dinamicamente agrupar recursos, por exemplo, executando um algoritmo de agrupamento em localizações de feições interessantes (BOUREAU; PONCE; LECUN, 2010). Esta abordagem produz um conjunto diferente de regiões de *pool* para cada imagem. Outra abordagem é aprender uma estrutura única de agrupamento que é, então, aplicada a todas as imagens, como em (JIA; HUANG; DARRELL, 2012). Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), existem dois conjuntos de terminologia comumente usados para descrever essas camadas. Na Figura 3.11 (esquerda), a rede convolucional é vista como um pequeno número de camadas relativamente complexas, com cada camada com muitos estágios. Nesta terminologia, há um mapeamento um a um entre tensores de kernel e camadas de rede. Já na terminologia da Figura 3.11 (Direita), a rede convolucional é vista como um número maior de camadas mais simples. Cada etapa do processamento é considerada uma camada por si só. Isso significa que nem toda camada possui parâmetros (GOODFELLOW; BENGIO; COURVILLE, 2016).

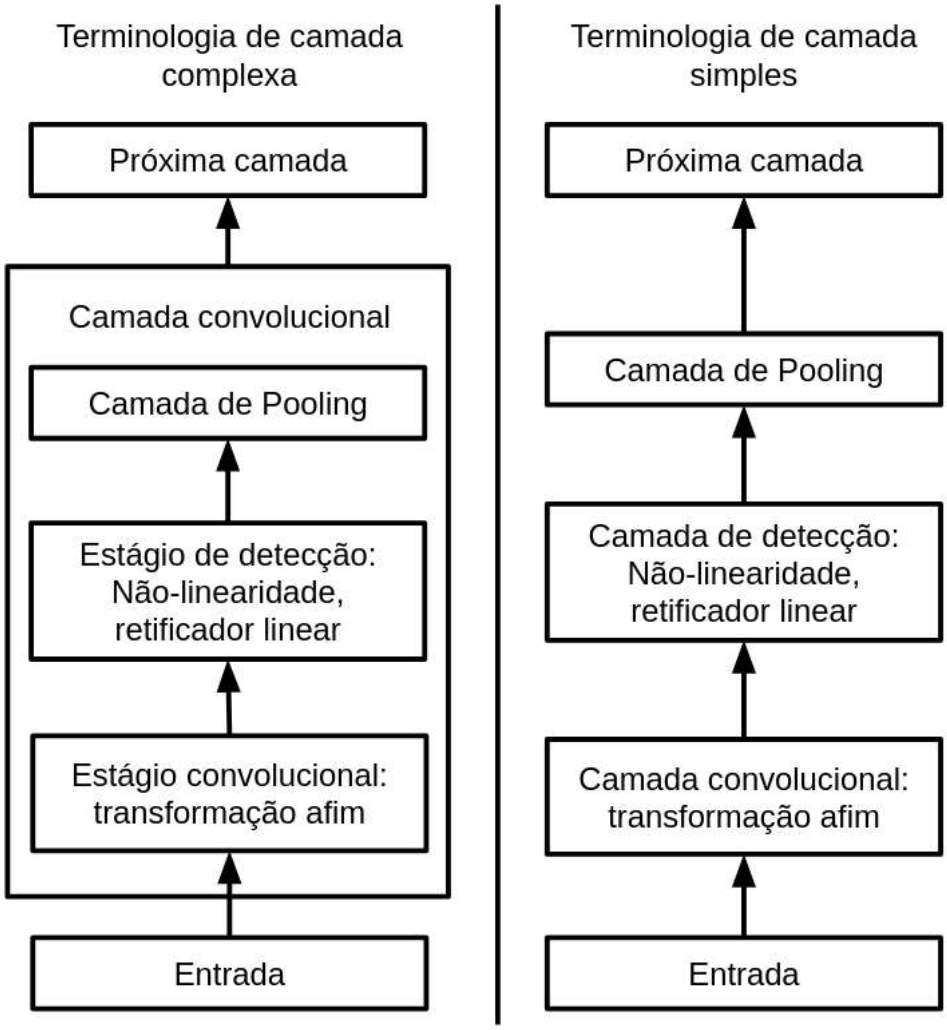


Figura 3.11 – Componentes de uma camada de rede neural convolucional padrão. Figura adaptada de (GOODFELLOW; BENGIO; COURVILLE, 2016)

3.7.4 Otimizadores

Algoritmos de aprendizado podem ser vistos, essencialmente, como processos de otimização (GOODFELLOW; BENGIO; COURVILLE, 2016). A tarefa essencial no projeto de uma rede é, via de regra o ajuste de parâmetros livres. No universo do aprendizado profundo, é típico que a quantidade de parâmetros seja da ordem de milhões ou até bilhões, o que, por vezes, faz com que o treinamento se estenda por vários dias. Frente a esse cenário, um conjunto adequado de técnicas de otimização deve ser escolhido. Esta seção apresenta técnicas de otimização para treinamento de redes neurais, incluindo uma breve visão geral de aspectos numéricos.

O processo de treinamento diz respeito a um problema específico de otimização: o de encontrar os parâmetros de uma rede neural que reduzem significativamente uma função de custo, que inclui, de alguma forma, uma medida de desempenho avaliada em todo o conjunto de treinamento, bem como termos eventuais de regularização (GOODFELLOW;

BENGIO; COURVILLE, 2016).

Algoritmos de otimização usados para treinamento de modelos de aprendizado de máquina (como as redes neurais) precisam ser analisados à luz de uma peculiaridade: embora o processo de aprendizado tenha lugar no contexto de um conjunto de treinamento, o desempenho nesse conjunto não é um fim em si mesmo. De fato, espera-se que o modelo, a partir do treinamento, venha a ter um desempenho satisfatório, primordialmente, junto a dados desconhecidos pelo modelo.

Para lidar com essa peculiaridade, um possível expediente é recorrer a técnicas de regularização, as quais permitem que se controle, de alguma forma, a flexibilidade do modelo empregado. Esses métodos podem envolver, por exemplo, termos de penalização de uma norma do vetor de pesos, parada antecipada do treinamento chamado de *early stopping* e o engenhoso esquema conhecido como *dropout*, que promove o desligamento aleatório de neurônios como mecanismo de controle (GOODFELLOW; BENGIO; COURVILLE, 2016).

Os métodos de regularização, assim como a escolha dos hiperparâmetros do método ou do processo de aprendizado, podem ter por base o uso de um conjunto de validação, como o *K-Fold* e *Holdout*. Esse conjunto é composto de uma fração dos dados de treinamento, e serve como uma medida do desempenho do método junto a dados não vistos, ou seja, junto aos dados que servirão para o teste (BISHOP, 1995). Nos testes realizados neste trabalho, foram utilizado os seguintes algorítimos de otimização: SGD, ADAM e RMSProp. Que serão discutidos no decorrer da próxima seção.

3.7.4.1 SGD

O algoritmo do gradiente estocástico ou Stochastic Gradient Descent (SGD), tem por base a metodologia clássica de "descida mais íngreme" (*steepest descent*), ou seja, de adotar a direção contrária à do gradiente como direção minimizante. A ideia geral é partir de uma solução aleatória e então usar a informação do gradiente para atualizá-la iterativamente. O parâmetro de taxa de aprendizado deve ser escolhido, e influencia na convergência do algoritmo: taxas de aprendizado muito grandes fazem com que o algoritmo tenha passos muito largos ou até instabilidade, enquanto passos menores levam a mais precisão, mas também podem gerar uma demora excessiva (RUSSELL; NORVIG, 2021).

Então, o SGD representa, a rigor, uma aproximação de um valor médio por um valor instantâneo. Assim, no lugar de selecionarmos um conjunto de *data points* para cada etapa, escolhemos aleatoriamente apenas um *data point* de todo o *dataset* a cada iteração, calculando sua derivada e assim, reduzindo muito o custo computacional.

3.7.4.2 RMSProp

O algoritmo AdaGrad individualmente adapta a taxa de aprendizado para todos os parâmetros do modelo por uma escala inversamente proporcional à raiz quadrada da soma do histórico de todos os seus valores quadráticos (DUCHI; HAZAN; SINGER, 2011). Os parâmetros com a maior derivada parcial de perda têm a sua taxa de aprendizado correspondente diminuída rapidamente, enquanto parâmetros com a menor derivada parcial têm, relativamente, uma pequena diminuição em sua taxa de aprendizado. No contexto de otimização convexa, o AdaGrad possui propriedades bastante interessantes. Porém, testes demonstraram que, para treinar modelos de redes neurais profundas, o acumulo de gradientes desde o início do treinamento pode resultar em uma diminuição prematura e alta na taxa efetiva de aprendizado (GOODFELLOW; BENGIO; COURVILLE, 2016).

Foi então proposto o algoritmo Root Mean Squared Propagation (RMSProp) por (HINTON; SRIVASTAVA; SWERSKY, 2012), que modifica o AdaGrad para um melhor desempenho em problemas não-convexos, alterando o acúmulo do gradiente em uma média móvel exponencialmente ponderada. Quando se lida com uma função não-convexa, a trajetória de aprendizagem pode passar por muitas estruturas diferentes, e, eventualmente, chegar a uma região que é uma bacia de atração de mínimo local. O AdaGrad reduz a taxa de aprendizado de acordo com toda a história do quadrado do gradiente e pode ter tornado a taxa de aprendizado muito pequena antes de chegar a estrutura convexa. O RMSProp usa uma média de descida exponencial para descartar o histórico passado muito alto, para que possa convergir rapidamente após encontrar uma área de mínimo local, como se fosse uma instância do algoritmo AdaGrad inicializado nessa área. De acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), empiricamente, o RMSProp tem demonstrado ser mais efetivo e prático para a otimização para redes neurais profundas.

3.7.4.3 ADAM

Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), o ADAM é também um algoritmo de otimização com taxa de aprendizado adaptativa. O nome ADAM deriva de *"adaptive moments"*. Tendo em vista os outros algoritmos apresentados neste trabalho, o ADAM é visto como uma variante da combinação entre RMSProp e *momentum*, com algumas diferenças importantes. Uma delas é que, no algoritmo ADAM, o *momentum* é incorporado diretamente como um estimador de primeira ordem de momento, com pesos exponenciais do gradiente. A forma mais comum ou padrão, de acordo bom (GOODFELLOW; BENGIO; COURVILLE, 2016) para se adicionar o *momentum* no RMSProp é aplicando o momento para os gradientes modificados. O ADAM inclui correções de viés em ambos os estimadores, os momentos de primeira ordem e o momento de segunda ordem descentralizados, para a sua inicialização na origem.

4 Transferência de Aprendizado: Fundamentos e Modelos

4.1 Transferência de Aprendizado

Um ponto fundamental deste trabalho é a aplicação da transferência de aprendizado (Pan; Yang, 2010). Transferência de aprendizado (TL) é uma técnica que nos possibilita usar a estrutura de uma CNN já pronta, geralmente uma rede testada em competições de reconhecimento de imagens como a Large Scale Visual Recognition Challenge (ILSVRC) ou *ImageNet Challenge* (RUSSAKOVSKY et al., 2015). Esses modelos são treinados com um grande número de imagens de diferentes categorias, como imagens de animais, carros e objetos do dia-a-dia, por exemplo, o conjunto de dados da ImageNet, que possui 15 milhões de imagens rotuladas, de 22.000 de categorias diferentes, sendo que cada categoria é representada por um tipo específico de conteúdo, ou seja, imagens de cachorro pertencem especificamente à categoria "cachorro" (uma classe por dado) (ImageNet Project, 2021), outros exemplos de base de dados que contem imagens e, em algum momento, foram utilizados no treinamento das redes pré-treinadas são: Common Objects in Context (COCO) (LIN et al., 2014), CIFAR-10 (CIFAR, 2021) e Street View House Numbers (SVHN) (SVHN, 2021).

Tanto a base quanto algumas das redes estão disponíveis para download, facilitando a criação e a avaliação de novos modelos. Note-se que é possível usar as redes já treinadas de maneira parcial e realizar treinamento sobre quaisquer partes da estrutura. Isso significa que é possível aproveitar, por exemplo, as camadas iniciais da rede - já treinadas para a tarefa geral de reconhecimento de imagens - e treinar especificamente apenas algumas camadas finais sobre o problema abordado, um exemplo de seu *modus operandi* pode ser observado na Figura 4.1 (RUSSELL; NORVIG, 2021). É dessa forma que usaremos TL no trabalho.

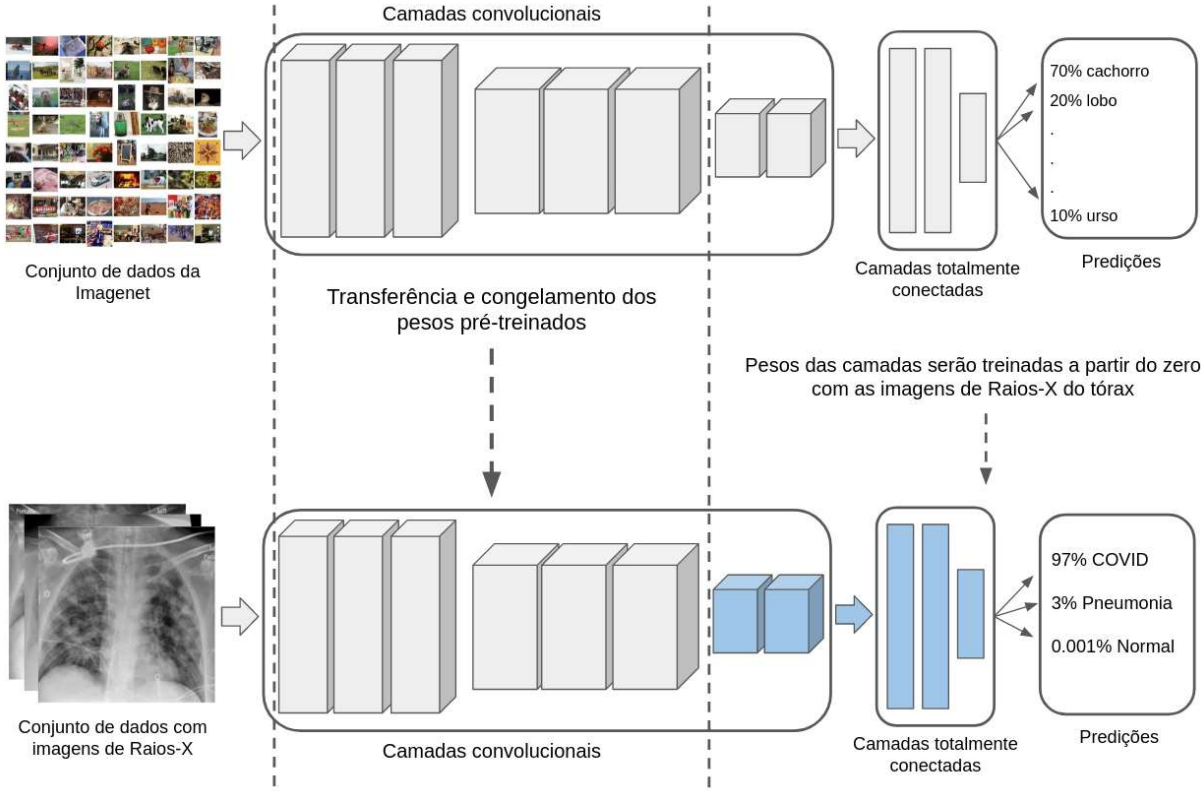


Figura 4.1 – Exemplo da aplicação da transferência de aprendizado. Figura adaptada a partir de (XU; XUE; ZHANG, 2019).

4.2 Modelos Utilizados para Transferência de Aprendizado

Para a realização dos testes neste trabalho, optou-se por aproveitar maximamente o conhecimento pré-adquirido por cada uma das redes, apenas adequando a primeira camada de cada um dos modelos para receber as imagens de raios-X do nosso *dataset* e retreinando a última camada com essas imagens. Intuitivamente, é como se considerássemos que as redes escolhidas são modelos capazes de reconhecer imagens gerais de maneira competente, cabendo apenas adequar esse conhecimento às imagens de raios-X de tórax com três classes: covid, normal e pneumonia. Nesta seção, discorreremos sobre cada uma das redes analisadas neste trabalho: SqueezeNet, ResNet, VGG, DenseNet, ShufleNet, MobileNet, AlexNet e GoogleNet. Tais redes foram selecionadas de modo a formar um conjunto diversificado dentre as redes disponíveis para download.

4.2.1 AlexNet

Essa rede foi proposta por Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton para a competição ImageNet (RUSSAKOVSKY et al., 2015) de 2012 (KRIZHEVSKY; SUTSKEVER; HINTON, 2017), na qual foi vencedora com um desempenho 10,8% melhor

que o do segundo colocado.

A rede contém 8 camadas no total, sendo cinco camadas convolucionais e três camadas totalmente conectadas. A rede foi treinada usando os conjuntos de dados utilizados nas competições ILSVRC-2010 e ILSVRC-2012, e seu treinamento demorou entre cinco a seis dias, utilizando duas Graphics Processing Unit (GPU) GTX 580 de 3 GB (Giga Byte (GB)) (KRIZHEVSKY; SUTSKEVER; HINTON, 2017).

Os autores tiveram uma boa ideia, que resultou em ganho de tempo e processamento, fazendo com que as GPUs se comunicassem apenas entre camadas pré definidas por eles. Assim os kernels da camada 3 recebem apenas as entradas de todos os mapas da camada 2, porém, os kernels da camada 4 recebem apenas as entradas dos mapas da camada 3 que ficam na mesma GPU (KRIZHEVSKY; SUTSKEVER; HINTON, 2017). A arquitetura pode ser visualizada na Figura 4.2.

Os autores utilizaram as técnicas de *data-augmentation* e *dropout* para evitar o *overfitting* durante o treinamento da rede. Eles descrevem em seu artigo que, em comparação com as redes neurais *feedforward* padrão com camada e tamanhos semelhantes, arquiteturas CNN possuem menos conexões e parâmetros para serem aprendidos, portanto, facilitando o treinamento da rede e, seu desempenho é provavelmente, apenas um pouco pior.

A arquitetura da rede possui, na última camada, 1000 neurônios que utilizam a função *softmax*, fazendo, assim, com que ela seja capaz de reconhecer 1000 classes diferentes. A primeira camada convolucional filtra a entrada das imagens, transformando-as no tamanho de 224 de largura por 224 de altura com 3 canais de cores, vermelho, verde e azul (do inglês Red Blue Green (RGB)), e seus 96 kernels possuem o tamanho de 11x11x3 com 4 pixels de *stride*. A segunda camada convolucional emprega 256 núcleos de tamanho 5x5x48. A terceira camada convolucional, por sua vez, possui 384 kernels com tamanho 3x3x256, enquanto a quarta possui 384 kernels de tamanho 3x3x192. A quinta e última camada convolucional possui 256 kernels de dimensão 3x3x192. Por fim, há duas camadas totalmente conectadas, com 4096 neurônios cada, e a já mencionada camada de saída (KRIZHEVSKY; SUTSKEVER; HINTON, 2017).

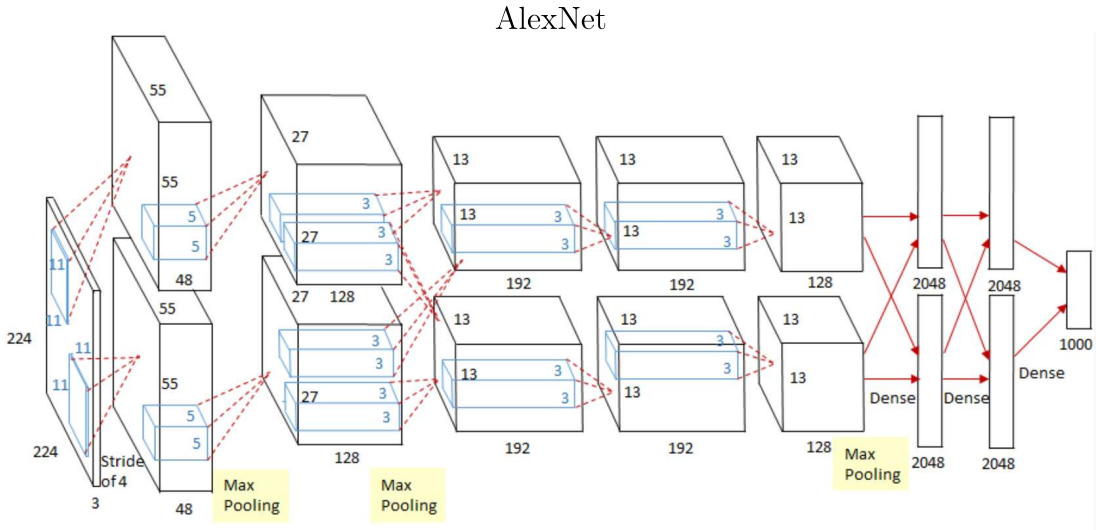


Figura 4.2 – Estrutura da AlexNet, representada com implementação em duas GPUs. Imagem adaptada de (Sik-Ho Tsang, 2018)

4.2.2 DenseNet

A DenseNet (Huang et al., 2017) propõe uma arquitetura que garante o máximo fluxo de informações entre as camadas. Uma grande vantagem da DenseNet é que todas as camadas são conectadas com seus mapas de características referentes, diretamente entre si, preservando a natureza dos dados e alimentando as próximas camadas com o mapa de características completo, como pode ser visto em Figura 4.3. Na DenseNet as camadas mais próximas da camada final do modelo recebem todos os mapas de características de todas as camadas anteriores concatenadas, diferentemente das arquiteturas tradicionais. Por possuir camadas convolucionais conectadas de forma densa, a rede recebeu o nome de DenseNet.

O modelo passou na avaliação em quatro testes de capacidade bastante competitivos na época de sua criação. Os testes foram feitos nos conjuntos de dados CIFAR-10, CIFAR-100 (CIFAR, 2021), SVHN (SVHN, 2021) e ImageNet (ImageNet Project LSVRC, 2021a). Os autores afirmam que o modelo construído por eles tendeu a exigir menos parâmetros do que os algoritmos existentes, com uma precisão comparável à do estado da arte para a época, e com desempenho superior (Huang et al., 2017).

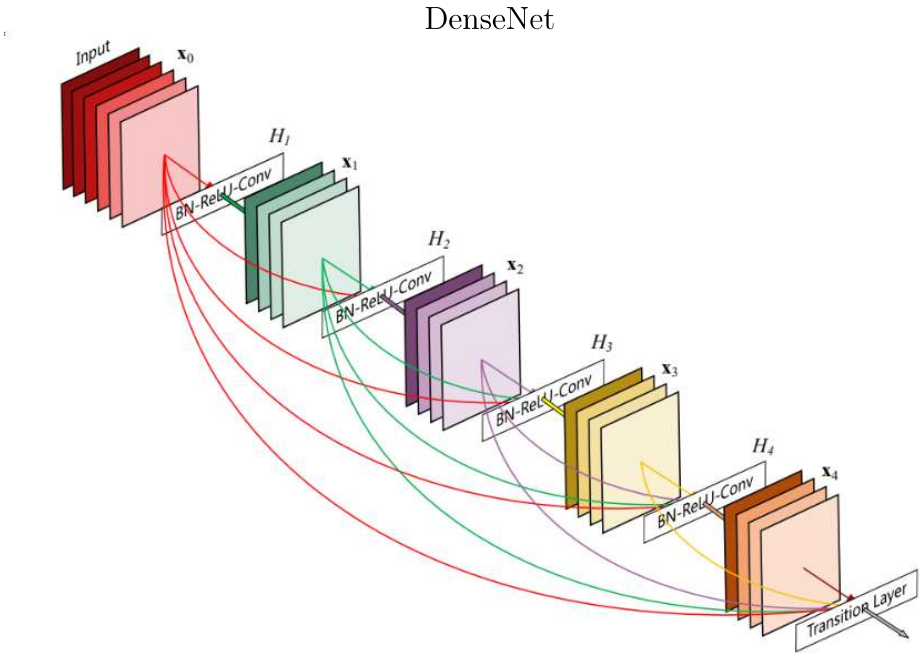


Figura 4.3 – Um bloco denso com 5 camadas, com uma taxa de crescimento de k=4, com cada camada passando todos os mapas de recursos anteriores como entrada (Huang et al., 2017).

Outro recurso aproveitado pelos autores — e que é característico das redes CNN — é o *downsampling*, que muda o tamanho dos *features maps*. Porém, para facilitar o *downsampling* no modelo por eles proposto, os autores dividiram a rede em vários blocos densos, que são, também, densamente conectados. Eles se referem às camadas entre os blocos como camadas de transição, que fazem convolução e *pooling*. As camadas de transição usadas por eles, são uma camada de *batch normalization* e uma camada convolucional 1x1 seguida por uma camada de *average-pooling* 2x2. A arquitetura pode ser visualizada em maior detalhe na Tabela 4.1.

Tabela 4.1 – Arquitetura DenseNet usada para ImageNet. A taxa de crescimento para as primeiras 3 camadas é k=32 e k=48 para DenseNet-161. Cada camada "conv" mostrada na tabela, correspondendo a sequência de BN-ReLU-Conv (Huang et al., 2017).

Camadas	Tamanho da saída	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 x 112	7 x 7 convolutional, stride 2			
Pooling	56 x 56	3 x 3 max pool, stride 2			
Dense Block (1)	56 x 56	[1 x 1 conv / 3 x 3 conv] x 6			
Transition Layer (1)	56 x 56	1 x 1 conv			
Transition Layer (1)	28 x 28	2 x 2 average pool, stride 2			
Dense Block (2)	28 x 28	[1 x 1 conv / 3 x 3 conv] x 12			
Transition Layer (2)	28 x 28	1 x 1 conv			
Transition Layer (2)	14 x 14	2 x 2 average pool, stride 2			
Dense Block (3)	14 x 14	[1x1/3x3]x24	[1x1/3x3]x32	[1x1/3x3]x48	[1x1/3x3]x64
Transition Layer (3)	14 x 14	1 x 1 conv			
Transition Layer (3)	7 x 7	2 x 2 average pool, stride 2			
Dense Block (4)	7 x 7	[1x1/3x3]x16	[1x1/3x3]x32	[1x1/3x3]x32	[1x1/3x3]x48
Classification Layer	1 x 1	7 x 7 global average pool - 1000D fully-connected, softmax			

4.2.3 GoogLeNet

Desenvolvida por C. Szegedy et al. (2015) (Szegedy et al., 2015), a arquitetura da GoogLeNet teve foco na simplicidade do desenvolvimento. Os autores afirmam que nenhuma tecnologia nova foi usada, nem melhorias em hardware foram feitas em comparação a sua versão de três anos antes.

Em seu trabalho, os autores se concentraram em um modelo com uma rede neural profunda e uma arquitetura de rede mais eficiente para visão computacional. A esta nova estratégia de desenvolvimento eles deram o nome de "Inception", que deriva do artigo (LIN; CHEN; YAN, 2013), em conjunto com o famoso "meme" da internet — relacionado ao filme de mesmo nome (e de nome "A Origem" em português) — *"we need to go deeper"* (ImageNet Project LSVRC, 2021b). Porém, no caso do trabalho por eles realizado, a palavra "profundo" é usada com dois significados diferentes: no primeiro, os autores tentam dar a entender que foi introduzido um novo nível de organização na forma de "Módulos de inicialização", e, também, no sentido de maior profundidade da rede. Os autores alegam que, em geral, pode-se ver o modelo Inception como uma combinação lógica de (LIN; CHEN; YAN, 2013), enquanto toma inspiração e orientação da teoria do trabalho de Arora et al. (ARORA et al., 2014). Estes benefícios puderam ser notados na competição ILSVRC de 2014 (RUSSAKOVSKY et al., 2015).

Na competição ILSVRC de 2014, a GoogLeNet usou 12 vezes menos parâmetros do que a rede vencedora da época. Ainda assim, a rede foi significativamente mais precisa na identificação de objetos. Segundo os autores, isso não acontecia graças a um hardware mais potente, ou uma rede "maior" ou "mais profunda". A vitória se deu, segundo os

autores, pela sinergia de arquiteturas profundas e visão computacional clássica.

Os autores fizeram questão de levar em consideração, no desenvolvimento da GoogLeNet, a eficiência da arquitetura em relação à computação móvel e embarcada. Segundo os autores, para a maioria dos experimentos, os modelos foram projetados para manter um orçamento computacional de 1,5 bilhão de operações matemáticas no momento da inferência, para que não acabasse sendo apenas uma curiosidade acadêmica, mas que pudesse ser aplicado na vida real, mesmo com um grande conjunto de dados, a um custo razoável.

GoogLeNet

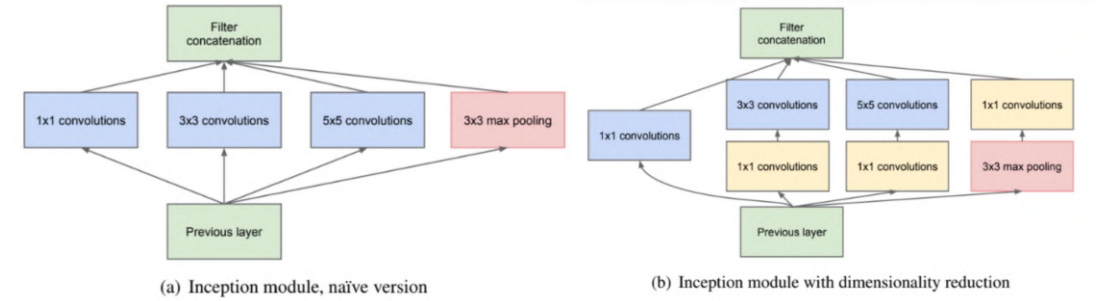


Figura 4.4 – Módulo Inception (Szegedy et al., 2015).

Ela utiliza um filtro de convolução com dimensão 1x1, com a ideia de que o vizinho mais próximo é aquele que carrega mais informação, ou está fortemente relacionado, então um filtro 3x3 e finalmente um módulo 5x5. GoogLeNet contou com três outras redes para o seu desenvolvimento, são elas LeNet-5 (LeCun et al., 1989), NiN (LIN; CHEN; YAN, 2013), R-CNN (Girshick et al., 2014). O grande motor do GoogLeNet é o bloco Inception: ela tem 9 módulos iniciais. Dentro de toda a estrutura, existem dois blocos *softmax* atuando como pré-classificadores no meio da rede, sem contar a última camada de classificação, que também é uma *softmax* como demonstrado na Figura 4.4. A arquitetura pode ser visualizada com maior detalhamento na Tabela 4.2.

Tabela 4.2 – Arquitetura GoogLeNet (Szegedy et al., 2015).

Type	Patch Size	Output size	Depth	1x1	3x3 rd	3x3	5x5 rd	5x5	Pool p	Params	Ops
Convolution	7x7/2	112x112x64	1							2.7K	34M
Max pool	3x3/2	56x56x64	0								
Convolution	3x3/1	56x56x192	2		64	192				112K	360M
Max pool	3x3/2	28x28x192	0								
Inception (3a)		28x28x256	2	64	96	128	16	32	32	159K	128M
Inception (3b)		28x28x480	2	128	128	192	32	96	64	380K	304M
Max pool	3x3/2	14x14x480	0								
Inception (4a)		14x14x512	2	192	96	208	16	48	64	364K	88M
Inception (4b)		14x14x512	2	160	112	224	24	64	64	437K	100M
Inception (4c)		14x14x512	2	128	128	256	24	64	64	437K	119M
Inception (4d)		14x14x528	2	144	144	288	32	64	64	437K	170M
Inception (4e)		14x14x832	2	160	160	320	32	128	64	840K	840K
Max pool	3x3/2	7x7x832	0								
Inception (5a)		14x14x832	2	256	160	320	32	128	64	1072K	54M
Inception (5b)		14x14x832	2	160	160	320	32	128	64	1388K	71M
Max pool	7x7/1	1x1x1024	0								
Dropout(40%)		1x1x1024	0								
Linear		1x1x1000	1							1000K	1M
Softmax		1x1x1000	0								

4.2.4 MobileNet

Em (SANDLER et al., 2018), a proposta é construir uma nova arquitetura de rede neural especificamente adaptada para celulares e recursos restritos. Segundo eles, esse trabalho oferecia o que havia de mais moderno em modelos de visão computacional adaptados para dispositivos móveis, diminuindo significativamente o número de operações e a memória necessária, e mantendo a mesma precisão de soluções com finalidade equivalente.

A principal contribuição, de acordo com os autores, é um novo módulo de camada, um módulo residual invertido com gargalo linear. Esse modelo recebe como entrada uma representação compacta de baixa dimensão que, primeiramente, é expandida para uma dimensão mais alta e filtrado com uma *'lightweight depthwise convolution'*. Depois, os recursos são novamente projetados para baixa dimensão com uma convolução linear. A implementação oficial está disponível como parte da biblioteca TensorFlow-Slim (Tensorflow, 2021).

Este módulo pode ser implementado com eficiência usando operações padrão em qualquer estrutura moderna, permitindo ganhos de desempenho em vários contextos usando *benchmarks* padrão. Além disso, este módulo convolucional é particularmente adequado a projetos móveis, porque permite reduzir significativamente a área de cobertura da memória necessária durante a sua execução, nunca fazendo uso total de grandes tensores intermediários. Isso reduz a necessidade de acesso à memória principal em muitos designs de hardware embutidos, que fornecem pequenas quantidades de memória cache controlada por software muito rapidamente.

Como uma estratégia preliminar, a MobileNet possui convoluções separadas em profundidade, o que, de acordo com os autores, corresponde a um bloco de construção chave para muitas arquiteturas eficientes de redes já existentes. A ideia básica é substituir um operador convolucional completo por uma versão fatorada, que divide a convolução em duas camadas separadas. A primeira camada é chamada de convolução em profundidade, e realiza uma filtragem leve aplicando um único filtro convolucional por entrada. A segunda camada é uma convolução 1x1, chamada de convolução pontual, que é responsável por construir novos recursos por meio do cálculo de combinações lineares dos canais de entrada. De acordo com os autores, as convoluções separáveis em profundidade são basicamente uma substituição imediata para camadas convolucionais padrão. Empiricamente, eles funcionam quase tão bem quanto as convoluções regulares, e reduzem a computação em comparação com as camadas tradicionais por um fator $k = 21$. A MobileNet usa $k = 3$ (convoluções separáveis em profundidade 3×3), de modo que o custo computacional é de 8 a 9 vezes menor do que o das convoluções padrão com uma pequena redução na precisão apenas. O bloco de construção básico é um gargalo de convolução separável em profundidade com resíduos: sua estrutura detalhada é mostrada na Tabela 4.4. A arquitetura da MobileNet contém a camada inicial de convolução completa com 32 filtros, seguida por 19 camadas de gargalo residuais descritas na Tabela 4.3.

Tabela 4.3 – Gargalo dos Blocos Residuais (SANDLER et al., 2018).

Input	Operator	Output
h x w x k	1 x 1 conv2d , ReLU6	h x w x (tk)
h x w x tk	3 x 3 dwise s=s, ReLU6	h/s x w/s x (tk)
h/s x w/s x tk	linear 1 x 1 conv2d	h/s x w/s x k'

A arquitetura do MobileNet contém a camada inicial convolucional com 32 filtros, seguida por 19 camadas de gargalo residuais descritas na Tabela 4.3. Utilizou-se ReLU como a não linearidade devido à sua robustez quando usado com computação de baixa precisão, utilizaram o tamanho do kernel 3x3 como é o padrão para redes modernas e utilizaram dropout e batch normalization durante o treinamento.

Tabela 4.4 – Arquitetura da MobileNet (SANDLER et al., 2018).

Input	Operator	t	c	n	s
224^2 x 3	conv2d	-	32	1	2
112^2 x 32	bottleneck	1	16	1	1
112^2 x 16	bottleneck	6	24	2	2
56^2 x 24	bottleneck	6	32	3	2
28^2 x 32	bottleneck	6	64	4	2
14^2 x 64	bottleneck	6	96	3	1
14^2 x 96	bottleneck	6	160	3	2
7^2 x 160	bottleneck	6	320	1	1
7^2 x 320	conv2d 1x1	-	1280	1	1
7^2 x 1280	avgpool 7x7	-	-	1	-
1 x 1 x 1280	conv2d 1x1	-	k	-	-

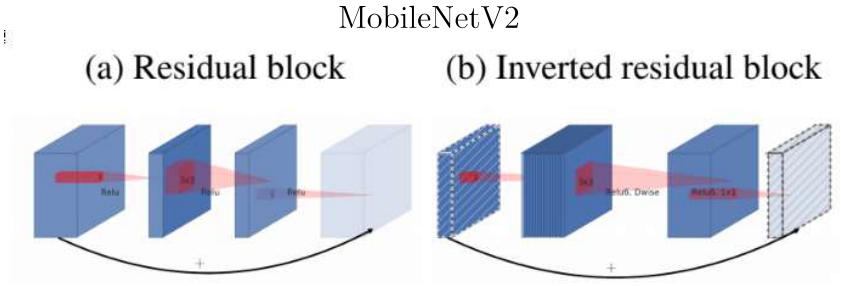


Figura 4.5 – Diferença entre os blocos residuais e os blocos residuais invertidos. (SANDLER et al., 2018).

Na Figura 4.5 pode-se observar a evolução de blocos de convolução separáveis: a textura destacada diagonalmente indica camadas que não contêm não linearidades. A última camada, levemente colorida, indica o início do próximo bloco.

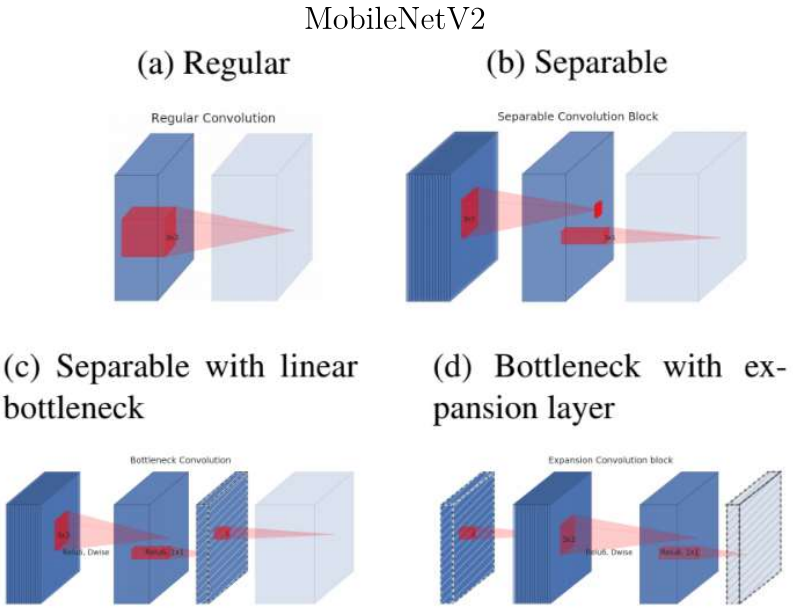


Figura 4.6 – Evolução da separação dos blocos convolucionais (SANDLER et al., 2018).

Já na Figura 4.6, pode-se observar a diferença entre o bloco residual e o residual invertido. Camadas destacadas diagonalmente não usam a não linearidade. A espessura de cada bloco é usada para indicar seu número relativo de canais. Pode-se observar como os resíduos clássicos se conectam às camadas com alto número de canais, enquanto os resíduos invertidos conectam os gargalos.

4.2.5 ResNet

Na arquitetura da rede, os autores testaram várias redes simples/residuais e observaram padrões consistentes. Como exemplo, eles usam dois casos descritos por modelos utilizados na ImageNet (He et al., 2016a):

- Rede Simples: Nas linhas de base simples (Figura 4.7, meio), a arquitetura proposta pelos autores é inspirada, principalmente, pela filosofia das redes VGG (SIMONYAN; ZISSERMAN, 2014) (Figura 4.7, esquerda). As camadas convolucionais normalmente têm filtros 3x3 e seguem duas regras de modelagem simples que são:

  1. Para o mesmo tamanho de mapa de características de saída, as camadas têm o mesmo número de filtros e;
  2. Se o tamanho do mapa de características for reduzido pela metade, o número de filtros é dobrado, de modo a preservar as características das camadas.

  Por fim, a rede termina com uma camada de *global average pooling* e uma camada totalmente conectada de 1000 neurônios com *softmax*.

- Rede Residual: Com base na rede simples, os autores inseriram atalhos que conectam as camadas, identificados por uma linha contínua que conecta as entradas e saídas das camadas intermediárias, tornando-as assim, a sua rede residual equivalente. Isso pode ser observado na Figura 4.7 (direita), porém essa conexão só ocorre quando as dimensões de entrada e saída são iguais.

De acordo com (He et al., 2016a), o modelo foi testado na (ImageNet Project, 2021) para mostrar o problema de degradação e avaliar o método. Eles conseguiram demonstrar que as redes residuais extremamente profundas são fáceis de otimizar, mas as redes lineares, em contrapartida, apresentam maior erro de treinamento quando a profundidade aumenta; ademais, as redes residuais profundas podem facilmente se aproveitar de ganhos de precisão com o aumento da profundidade, produzindo resultados substancialmente melhores do que as redes anteriores.

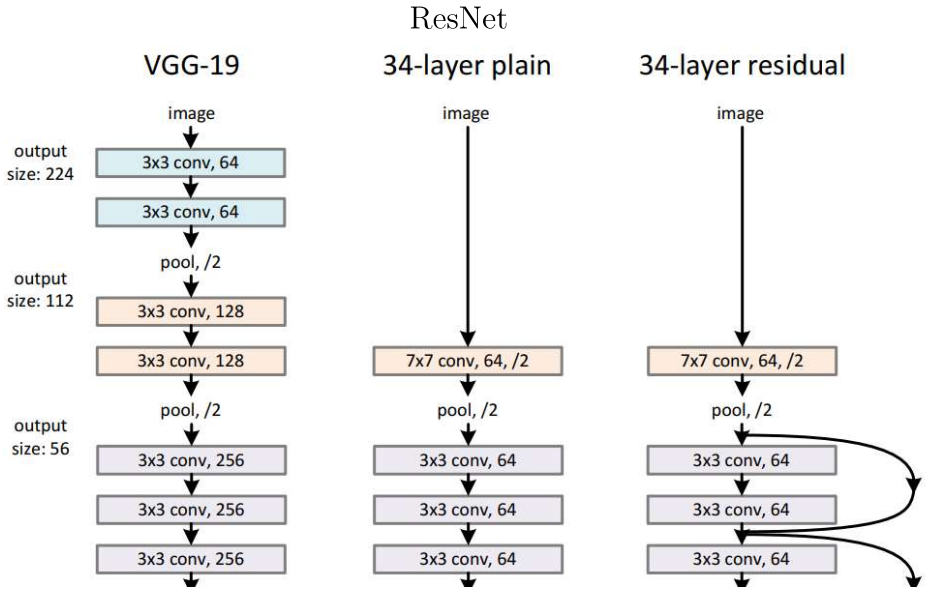


Figura 4.7 – Exemplo da arquitetura da ResNet. *(Essa é apenas uma parte da imagem, a figura completa pode ser visualizada em (He et al., 2016a) p. 773).

No conjunto de dados de classificação ImageNet (ImageNet Project LSVRC, 2021a), os autores obtiveram resultados representativos com redes residuais extremamente profundas. A rede residual de 152 camadas é a rede mais profunda já apresentada na ImageNet, embora ainda tenha uma complexidade menor do que as redes VGG (SIMONYAN; ZISSERMAN, 2014). O conjunto tem 3,57% de erro entre os 5 primeiros colocados no conjunto de testes da ImageNet e ganhou o primeiro lugar na competição de classificação ILSVRC 2015(RUSSAKOVSKY et al., 2015). Os modelos extremamente profundos também têm excelente desempenho de generalização para outras tarefas de reconhecimento, o que explica o sucesso da ResNet em detecção ImageNet, localização ImageNet, detecção COCO e segmentação COCO, nas competições ILSVRC & COCO 2015, (LIN et al., 2014).

Este ótimo resultado mostra que o princípio de aprendizagem residual é genérico, e que pode ser aplicável em outros problemas de visão computacional ou semelhantes.

É importante notar que o modelo possui menos filtros e menor complexidade do que a rede VGG (Figura 4.7, esquerda). A ResNet de 34 camadas tem 3,6 bilhões de FLoating-point Operations Per Second (FLOP), o que é apenas 18% da VGG-19 que possui 19,6 bilhões de FLOPs (He et al., 2016a).

Figura 4.7. Exemplo de arquiteturas de rede para ImageNet. Na esquerda o modelo VGG-19, com 19,6 bilhões de FLOPs como referência. No meio uma rede simples com 34 camadas de parâmetros com 3,6 bilhões de FLOPs. À direita uma rede residual com 34 camadas de parâmetros com 3,6 bilhões de FLOPs. A Tabela 4.5 mostra mais detalhes e outras variantes. (He et al., 2016a).

Tabela 4.5 – Arquitetura ResNet. Os blocos de construção estão entre colchetes, com os números dos blocos empilhados. A redução da resolução é realizada por conv3 1, conv4 1 e conv5 1 com o stride de 2 (He et al., 2016a).

Layers	Output Size	18-layer	34-layer	50-layer	101-layer	152-layer
Conv 1	112 x 112	7 x 7, 64, stride 2				
Conv 2x	56 x 56	3 x 3 max pool, stride 2				
Conv 2x	[3x3, 64]x2	[3x3, 64]x3	[3x3, 64]x3	[3x3, 64]x3	[3x3, 64]x3	[3x3, 64]x3
Conv 2x	28x28	-	-	[1x1, 256]x3	[1x1, 256]x3	[1x1, 256]x3
Conv 3x	28x28	[3x3, 128]x2	[3x3, 128]x4	[3x3, 128]x4	[3x3, 128]x4	[3x3, 128]x4
Conv 3x	28x28	[3x3, 128]x2	[3x3, 128]x4	[1x1, 512]x4	[1x1, 512]x4	[1x1, 512]x8
Conv 3x	28x28	-	-	[1x1, 512]x4	[1x1, 51]x4	[1x1, 512]x8
Conv 4x	14x14	[3x3, 256]x2	[3x3, 256]x6	[1x1, 256]x6	[1x1, 256]x23	[1x1, 256]x36
Conv 4x	14x14	[3x3, 256]x2	[3x3, 256]x6	[1x1, 256]x6	[1x1, 256]x23	[1x1, 256]x36
Conv 4x	14x14	-	-	[1x1, 1024]x6	[1x1, 1024]x23	[1x1, 1024]x36
Conv 5x	7x7	[3x3, 512]x2	[3x3, 512]x3	[3x3, 512]x3	[3x3, 512]x3	[3x3, 512]x3
Conv 5x	7x7	[3x3, 512]x2	[3x3, 512]x3	[1x1, 512]x4	[1x1, 512]x4	[1x1, 512]x8
Conv 5x	7x7	-	-	[1x1, 2048]x3	[1x1, 2048]x3	[1x1, 2048]x3
-	1x1	average pool, 1000-d fc, softmax				
Flops		$1.8x10^9$	$3.6x10^9$	$3.8x10^9$	$7.6x10^9$	$11.3x10^9$

4.2.6 ShuffleNet

Em (ZHANG et al., 2018), os autores notaram que arquiteturas básicas, como Xception (He et al., 2016b) e ResNeXt (Xie et al., 2017), são menos eficientes em redes extremamente pequenas devido às custosas e densas convoluções 1x1. Superando os efeitos colaterais advindos de convoluções de grupo, criaram uma nova operação de troca de canais para ajudar o fluxo de informações. Com base nas técnicas citadas anteriormente, construíram uma arquitetura altamente eficiente chamada ShuffleNet.

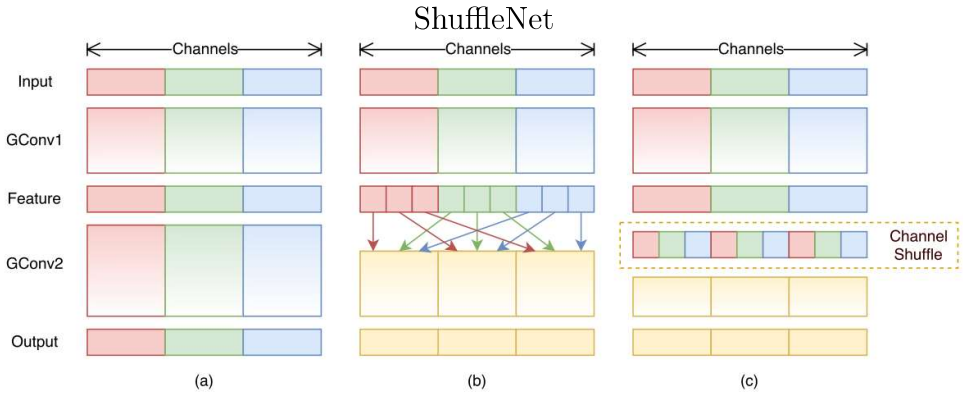


Figura 4.8 – Duas convoluções de grupo empilhadas. GConv significa convolução de grupo. (a) Possui duas camadas de convolução empilhadas, com o mesmo número de grupos. Cada canal de saída está relacionado apenas aos canais de entrada do grupo. Sem comunicação cruzada. (b) Os canais de entrada e saída são relacionados quando GConv2 obtém dados de grupos diferentes após GConv1. (c) Uma implementação equivalente à (b) porém, usando canal shuffle. (ZHANG et al., 2018).

Comparando-a com modelos populares como, ResNet (He et al., 2016a), VGG (SIMONYAN; ZISSERMAN, 2014), ResNeXt (Xie et al., 2017), para um orçamento limitado em relação a hardware, a ShuffleNet permite mais canais com mapas de atributos, o que ajuda a processar mais informações e é especialmente crítico para o desempenho de redes muito pequenas. Os modelos foram avaliados na importante competição da ImageNet (ImageNet Project LSVRC, 2021a) e em tarefas de detecção de objetos da COCO (LIN et al., 2014). Uma série de experimentos controlados mostrou a eficácia do modelo desenvolvido pelos autores em aspectos como princípios de design e melhor desempenho sobre outras estruturas, em comparação com a arquitetura de última geração na época do desafio, a MobileNet (SANDLER et al., 2018). A ShuffleNet alcançou um desempenho superior por uma margem significativa: e.g. com um erro de 7,8% absoluto em relação ao primeiro colocado na ImageNet no nível de 40 Mega FLoating-point Operations Per Second (MFLOP).

Também foram verificadas as especificidades e a forma de aplicação de aceleração em hardware real, ou seja, um núcleo de processador baseado em Advanced RISC Machine (ARM) promnto para uso. O modelo ShuffleNet atinge treze vezes o *speedup* atual (*speedup* teórico é 18x) sobre AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2017), mantendo uma precisão comparável.

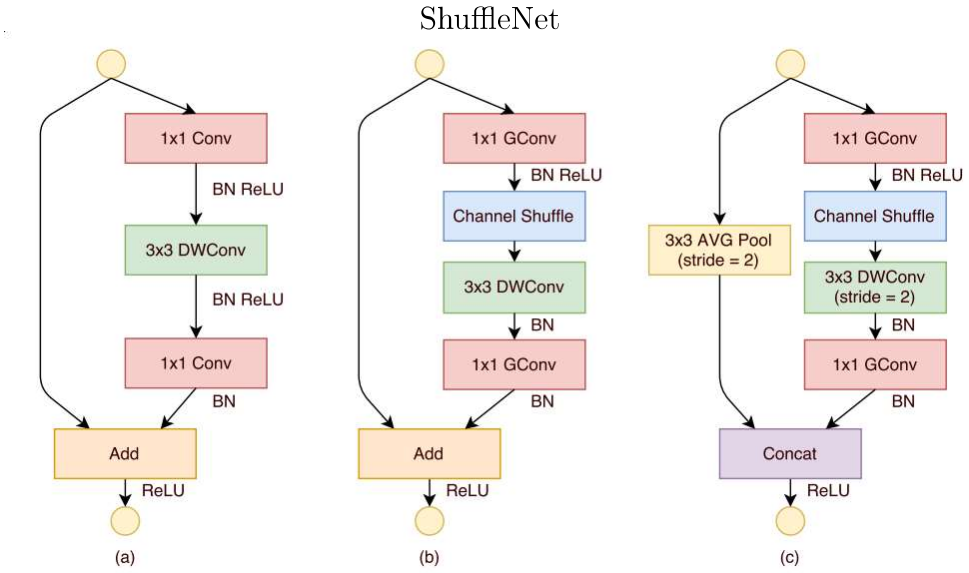


Figura 4.9 – Unidades ShuffleNet: (a) Unidade de gargalo com convolução profunda como DWConv; (b) Unidade ShuffleNet com grupo de convolução pontual (GConv) e canal shuffle. (c) Unidade ShuffleNet com *stride 2* (ZHANG et al., 2018).

Sua arquitetura foi construída em unidades ShuffleNet, como mostrado na Tabela 4.6. A rede proposta é composta, principalmente, por uma pilha de unidades ShuffleNet agrupadas em três estágios, como mostrado na Figura 4.8. O primeiro bloco de construção em cada estágio é aplicado com *stride* dois. Outros hiperparâmetros dentro de um estágio permanecem os mesmos, e, para o próximo estágio, os canais de saída são duplicados. Semelhantemente ao trabalho de (He et al., 2016a), definiram o número de canais de gargalo em 1/4 dos canais de saída para cada unidade ShuffleNet. A intenção dos autores era fornecer um design de referência o mais simples possível, embora os autores ressalvassem que um ajuste adicional de hiperparâmetros poderia gerar melhores resultados.

Em unidades ShuffleNet, Figura 4.9, o número do grupo denominado 'g' controla a dispersão de conexões de convoluções pontuais. A Tabela 4.6 explora diferentes números de grupo que foram adaptados aos canais de saída para garantir que o custo geral de processamento fosse praticamente inalterado, com aproximadamente 140 MFLOPs. De acordo com os autores, números de grupos maiores resultam em mais canais de saída, portanto, mais filtros convolucionais, para uma determinada restrição de complexidade, o que ajuda a processar mais informações.

Para personalizar a rede para uma complexidade desejada, pode-se aplicar um fator de escala denominado S no número de canais. Por exemplo, denota-se às redes na Tabela 4.6 como 'ShuffleNet 1x', então 'ShuffleNet S_x' significa, dimensionar o número de filtros em 'ShuffleNet 1x' por S vezes, portanto, a complexidade geral será aproximadamente S^2 vezes de 'ShuffleNet 1x'.

Tabela 4.6 – Arquitetura ShuffleNet. A complexidade é avaliada com FLOPs, ou seja, o número de acréscimos de multiplicação de ponto flutuante, para o Estágio 2, a convolução de grupo não foi aplicada na primeira camada de ponto, porque o número de canais de entrada é relativamente pequeno. (ZHANG et al., 2018)

Layers	Output Size	K-Size	S	R	Output channels (g groups)				
					g = 1	g = 2	g = 3	g = 4	g = 8
Image	224 x 224				3	3	3	3	3
Conv1	112 x 112	3 x 3	2	1	24	24	24	24	24
MaxPool	56 x 56	3 x 3	2						
Stage2	28 x 28		2	1	144	200	240	272	384
	28 x 28		1	3	144	200	240	272	384
Stage3	14 x 14		2	1	288	400	480	544	768
	14 x 14		1	7	288	400	480	544	768
Stage4	7 x 7		2	1	576	800	960	1088	1536
	7 x 7		1	3	576	800	960	1088	1536
GlobalPool	1 x 1	7 x 7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

4.2.7 SqueezeNet

Em (IANDOLA et al., 2016), os autores propõem um modelo que busca ser parcimonioso do ponto de vista de parâmetros, ressaltando algumas vantagens decorrentes. Dentre elas, pode-se indicar o treinamento distribuído mais eficiente da rede, uma vez que a comunicação entre os servidores não é um fator limitante, tendo em vista a baixa quantidade de parâmetros deste modelo proposto. De acordo com os autores, a sobrecarga de comunicação é diretamente proporcional ao número de parâmetros do modelo, ou seja, modelos com menos parâmetros treinam mais rápido e exigem menos comunicação entre os servidores. Isso proporciona um grande benefício para empresas que precisam fazer atualizações ou mudanças em seus modelos de aprendizado de máquina onde existe restrição de comunicação ou limitações de hardware. Os autores citam um caso específico da empresa Tesla, em que foi preciso fazer uma atualização em uma funcionalidade de direção semi-autônoma do piloto automático: a Tesla precisou fazer essa atualização em todos os carros de seus clientes e, caso estivesse utilizando uma rede neural convolucional profunda tradicional, essa atualização causaria transtornos.

Como é possível notar, na Tabela 4.7, que existem várias vantagens nas arquiteturas CNN menores: portanto, os autores se concentraram diretamente no problema de identificar uma arquitetura CNN com menos parâmetros, mas com precisão equivalente em comparação com os modelos tradicionais. Para conseguir fazer uso de todas estas vantagens,

os autores propõem uma pequena arquitetura CNN chamada SqueezeNet. A SqueezeNet alcança a precisão do nível da AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2017), no conjunto de dados da ImageNet, porém com 50 vezes menos parâmetros. Além disso, com as técnicas de compactação de modelo, pode-se compactar a SqueezeNet em menos de 0,5 MB ou seja 510 vezes menor que a AlexNet.

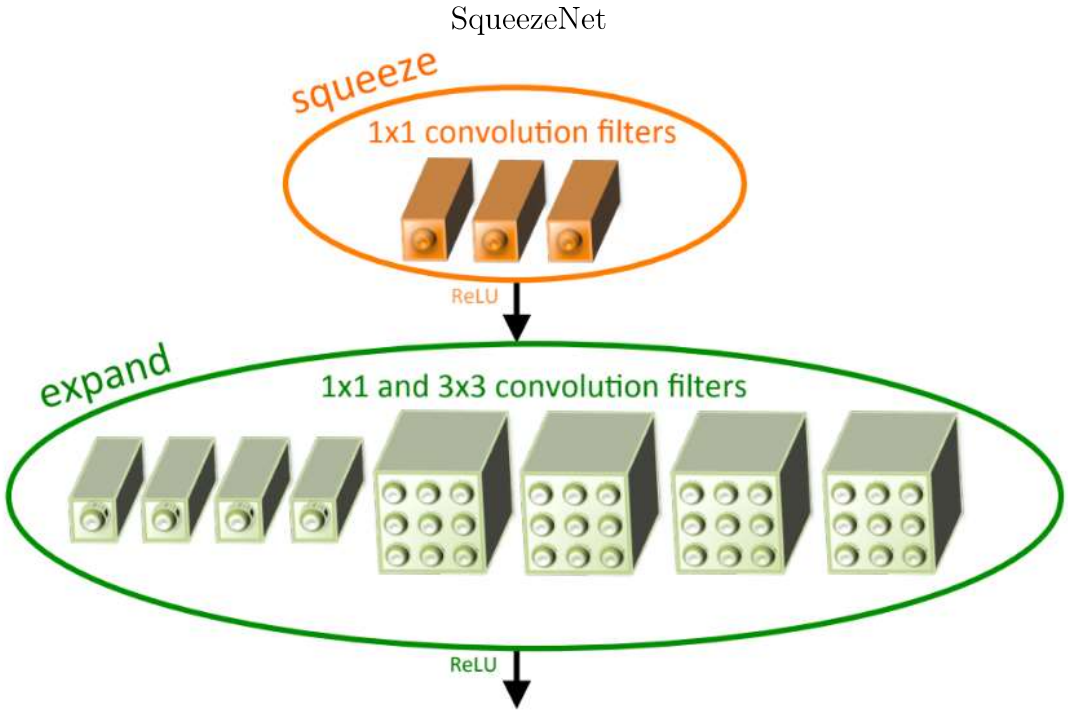


Figura 4.10 – Visão microarquitetural: Organização dos filtros de convolução no módulo Fire. Neste exemplo, s(1x1) = 3, e(1x1) = 4 e e(3x3) = 4. Ilustrando filtros de convolução. (IANDOLA et al., 2016)

O objetivo dos autores da SqueezeNet foi identificar as arquiteturas CNN que têm poucos parâmetros enquanto mantêm a precisão competitiva (IANDOLA et al., 2016). Para isso, eles empregaram três estratégias principais ao projetar a rede. A primeira estratégia consistia em substituir os filtros 3x3 por filtros 1x1: esses filtros foram escolhidos uma vez que um filtro 1x1 tem 9 vezes menos parâmetros do que um filtro 3x3. A segunda estratégia consistia em diminuir os canais de entrada para os filtros 3x3: a quantidade total de parâmetros nesta camada é o número total de canais de entrada x o número de filtros x (3x3). Portanto, para manter um pequeno número total de parâmetros no modelo, é importante não apenas diminuir o número de filtros, mas também diminuir o número de canais de entrada para os filtros 3x3. Os autores conseguiram diminuir o número de canais de entrada para os filtros 3x3 usando camadas de compressão. A terceira estratégia consiste em fazer o *downsampling* tardio na rede, para que as camadas de convolução tenham grandes mapas de ativação.

Em uma rede convolucional, cada camada de convolução produz um mapa de

ativação de saída, com uma resolução espacial de, geralmente, 1x1 e geralmente, muito maior do que 1x1 (IANDOLA et al., 2016). A altura e largura desses mapas de ativação são controladas pelo tamanho dos dados de entrada, por exemplo, imagens 256x256, e a escolha das camadas que passaram pelo *downsampling*. Mais comumente, a *downsampling* é projetada em arquiteturas CNN definidas em algumas das camadas de convolução ou agrupamento. Se os últimos 3 *layers* da rede tiverem *stride* muito grande, a maioria das camadas terá pequenos mapas de características. Por outro lado, se a maioria das camadas da rede tem um *stride* de 1 e, os *strides* maiores que 1 estão concentradas no final da rede, muitas camadas da rede terão grandes mapas de ativação. A tese dos autores era de que grandes mapas de ativação poderiam levar a uma maior precisão de classificação, devido à redução da resolução atrasada, com toda a configuração da rede mantida inalterada.

As estratégias 1 e 2 fazem o trabalho de diminuir criteriosamente a quantidade de parâmetros da CNN, enquanto tentam preservar a precisão. A estratégia 3 maximiza a precisão em um orçamento limitado de parâmetros. Na Figura 4.10 pode-se observar o módulo Fire, que é o bloco de construção para as arquiteturas CNN que nos permite empregar com sucesso as Estratégias 1, 2 e 3. A arquitetura SqueezeNet está disponível para download no repositório[1].

Tabela 4.7 – Dimensões arquitetônicas do SqueezeNet. A tabela foi inspirada no artigo da Inception2 dos autores (IOFFE; SZEGEDY, 2015) (IANDOLA et al., 2016).

Layer	Output Size	Filter Size	Depth	(1x1)	(1x1)	(3x3)	(1x1)	(1x1)	(3x3)	Parameter before pruning	Parameter after pruning
input image	224x224x3										
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			14,208	14,208
maxpool1	55x55x96	3x3/2	0								
fire2	55x55x128		2	16	64	64	100%	100%	33%	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	33%	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	33%	45,344	20,646
maxpool4	27x27x256	3x3/2	0								
fire5	27x27x256		2	32	128	128	100%	100%	33%	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	50%	33%	104,880	44,700
fire7	27x27x384		2	48	192	192	50%	100%	33%	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	50%	33%	188,992	77,581
maxpool8	13x12x512	3x3/2	0								
fire9	13x13x512		2	64	256	256	50%	100%	30%	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				20% (3x3)			14,208	14,208
avgpool10	1x1x1000	13x13/1	0								
										1,248,424 (Total)	421,098 (Total)

[1] https://github.com/DeepScale/SqueezeNet

4.2.8 VGG

Em (SIMONYAN; ZISSERMAN, 2014), os autores focaram seus esforços em aumentar constantemente a profundidade de seu modelo adicionando mais camadas convolucionais, o que é viável devido ao uso de filtros de convolução muito pequenos 3x3 em todas as camadas. Para isso, outros parâmetros foram fixados na arquitetura para que esse aumento na profundidade fosse passível de realização. Como resultado, os criadores desta rede aumentaram significativamente a precisão do seu modelo, que não só galgaram êxito nas tarefas de classificação e detecção em imagens no maior nível da competição da ImageNet. Estes resultados puderam ser notados quando a equipe participou do *ImageNet Challenge* de 2014 (ImageNet Project LSVRC, 2021a), onde garantiu primeiro e segundo lugares na competição, nas categorias de detecção e classificação de imagens, respectivamente. Os autores também relatam que, seu modelo tem a capacidade de generalização muito boa para outros conjuntos de dados de imagens, onde alcançaram bons resultados.

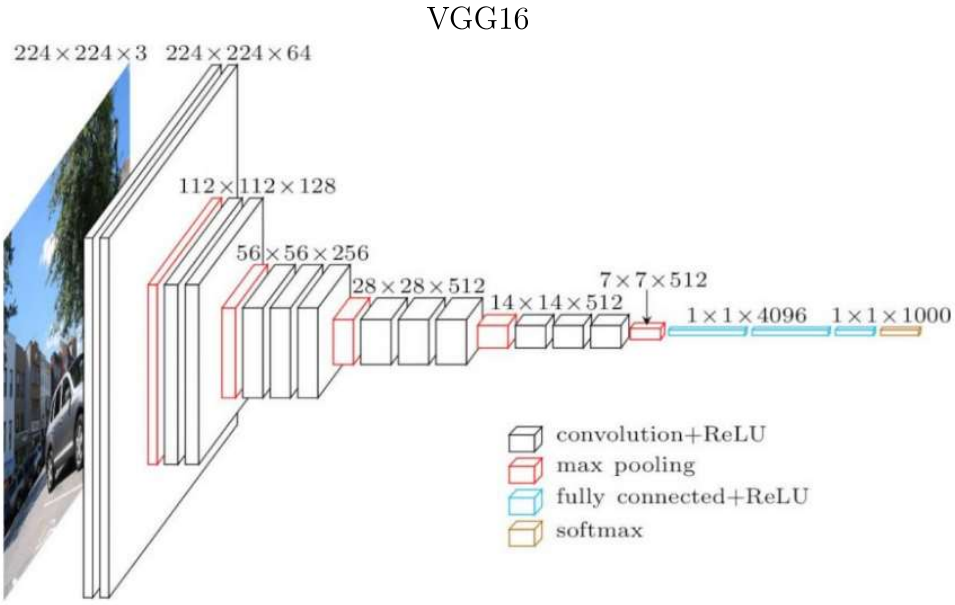


Figura 4.11 – Arquitetura VGG16. Fonte: neurohive.io. (Muneeb ul Hassan, 2021).

A arquitetura desta rede, como é mostrado na Figura 4.11, segundo os autores, passou por um treinamento que recebia imagens do tamanho fixo 224x224, e o único pré-processamento que fizeram foi subtrair o valor RGB médio, calculado no conjunto de treinamento de cada pixel. A imagem é passada por uma pilha de camadas convolucionais, onde foram utilizados filtros 3x3, em uma das configurações também foram utilizados filtros de convolução 1x1, que podem ser vistos como uma transformação linear dos canais de entrada, o *stride* é fixo em 1 pixel. Uma pilha de camadas convolucionais, que têm a profundidade diferente em arquiteturas diferentes, é seguida por três camadas totalmente conectadas, as duas primeiras têm 4096 neurônios cada uma, a terceira realiza a

classificação e contém 1000 neurônios, um para cada classe. Todas as camadas ocultas estão equipadas com função ReLU, como demonstrado na Tabela 4.8. Os autores notaram que nenhuma das suas redes, exceto uma, continham normalização nas camadas, segundo eles, a normalização não melhora o desempenho do modelo no conjunto de dados ILSVRC e leva ao maior consumo de memória e tempo de processamento (SIMONYAN; ZISSERMAN, 2014).

Tabela 4.8 – Configuração ConvNet.

A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 x 224 RGB image)					
conv3-64	conv3-64 LRN*	conv3-64 conv3-64*	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128*	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256*	conv3-128 conv3-128 conv3-256*	conv3-128 conv3-128 conv3-256 conv3-256*
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512*	conv3-512 conv3-512 conv3-512*	conv3-512 conv3-512 conv3-512 conv3-512*
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512*	conv3-512 conv3-512 conv3-512*	conv3-512 conv3-512 conv3-512 conv3-512*
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

A profundidade das configurações aumenta da esquerda (A) para a direita (E), conforme mais camadas são adicionadas (*camadas adicionadas). Os parâmetros da camada convolucional são indicados como “conv (tamanho do campo receptivo)-(número de canais)”. A função de ativação ReLU não é mostrada por questões de praticidade. (SIMONYAN; ZISSERMAN, 2014).

5 Experimentos

Neste Capítulo, discutiremos as etapas e as configurações utilizadas nos experimentos, cujos resultados serão detalhados no Capítulo 6. O objetivo dos experimentos foi encontrar uma combinação que resultasse no melhor resultado de acurácia, e, portanto, todos os 8 modelos de redes pré treinadas que foram apresentadas no Capítulo 4 foram testadas sob variação de diversos fatores: escolha da base de dados de imagens de raios-X, variação no tamanhos dos lotes que serão processados, taxa de aprendizado e otimizadores.

Os experimentos deste trabalho se dividem em duas etapas. Na primeira etapa, o problema de classificação foi abordado com três classes. Na segunda etapa, tratou-se o problema de forma binária, o que ocasionou uma melhoria em termos de acurácia.

Todos os testes foram realizados utilizando a linguagem de programação Python, juntamente com o *framework PyTorch*. As especificações de *hardware* se resumem a uma placa de vídeo GTX 1080 (GPU), processador Core i7 e 16 Gb (GB) de memória RAM (Random Access Memory (RAM)). Todo o processamento dos experimentos feitos neste trabalho foi realizado utilizando a placa de vídeo GPU.

Para todos os modelos testados neste trabalho, utilizou-se o valor máximo de 100 épocas para treinamento; caso os modelos não apresentassem melhora no valor do percentual da taxa de erro durante 20 épocas consecutivas, o treinamento era encerrado antecipadamente: essa estratégia de parada antecipada é conhecida por *Early Stopping*, que é importante para que o modelo não se sobreajuste aos dados (GOODFELLOW; BENGIO; COURVILLE, 2016).

5.1 Conjunto de Imagens

As imagens que compõem o conjunto de imagens (*dataset*) deste trabalho foram obtidas a partir dos trabalhos (CHOWDHURY et al., 2020) e (RAHMAN et al., 2021), utilizando o repositório (COVID-19 Radiography Database on Kaggle, 2021).

O conjunto de imagens foi dividido em três partes: Treinamento, Validação e Teste. Esta divisão foi feita, inicialmente, nas proporções de 80% para treinamento, 19% para validação e 1% para teste *holdout*. Optou-se por esta forma de divisão, com apenas 1% para o conjunto de teste, pois, no momento em que este trabalho estava sendo realizado, era possível adquirir, quase que diariamente, novas imagens de raios-X previamente diagnosticadas como pertencentes à classe "covid", no repositório do Github (COVID-19 image data collection, 2021), o que tornava a inclusão dessas novas imagens nos conjuntos de treinamento e validação impraticável, uma vez que teríamos que retreinar todos os

modelos. Portanto, todas as novas imagens que recebemos após a fase de treinamento das redes foram colocadas apenas no conjunto de dados de teste, como forma de garantir que os modelos estariam aptos para fazer uma boa generalização com as novas imagens obtidas após seu treinamento. Na Tabela 5.1, pode-se verificar a quantidade total de imagens.

Tabela 5.1 – Número total de imagens.

Classes	Imagens
Normal	1341
Pneumonia	1345
Covid	1170
Total	3856

A Tabela 5.2 mostra as quantidades de imagens para cada classe. Diante do exposto, na finalização deste trabalho, o conjunto de dados de teste possuía 387 imagens normalizadas, proporcionando uma melhor avaliação da capacidade de generalização dos modelos testados.

Tabela 5.2 – Número total de imagens de cada classe.

Classes	Divisão do conjunto de dados		
	Treino	Teste	Validação
Normal	1051	14	255
Pneumonia	1097	13	240
Covid	937	11	238
Total	3085	38	733

5.1.1 Aumento Artificial de Dados

De acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), um modelo de aprendizado de máquina aprenderá a generalizar melhor quando treinado em um conjunto de dados muito grande. Porém, na maioria das vezes, isso não é possível, como no caso de imagens médicas para o reconhecimento de uma enfermidade que é relativamente nova, situação que acontece neste trabalho. Podem-se realizar transformações de forma aleatória nas imagens como rotação, inversão, redimensionamento, mudança nas cores e brilho das imagens. Isso faz com que a rede seja capaz de reconhecer objetos, mesmo que alguns padrões de pixels das imagens estejam em locais diferentes em outras imagens: isso dá à rede a capacidade de ser invariante a transformações, aumentando sua capacidade de generalização. Na realização dos testes com aumento de dados online, todas as imagens sofreram as seguintes transformações: normalização do tamanho seguindo o padrão de 224

pixels de altura por 224 pixels de largura; rotação das extremidades com ponto fixo no centro das imagens de -10 a +10 graus, a cada época, e de forma aleatória, e inversão horizontal aplicada com 5% de probabilidade a cada imagem e a cada época.

Na Figura 5.1, demonstra-se como ficaram as imagens após a realização do aumento artificial do *dataset*.

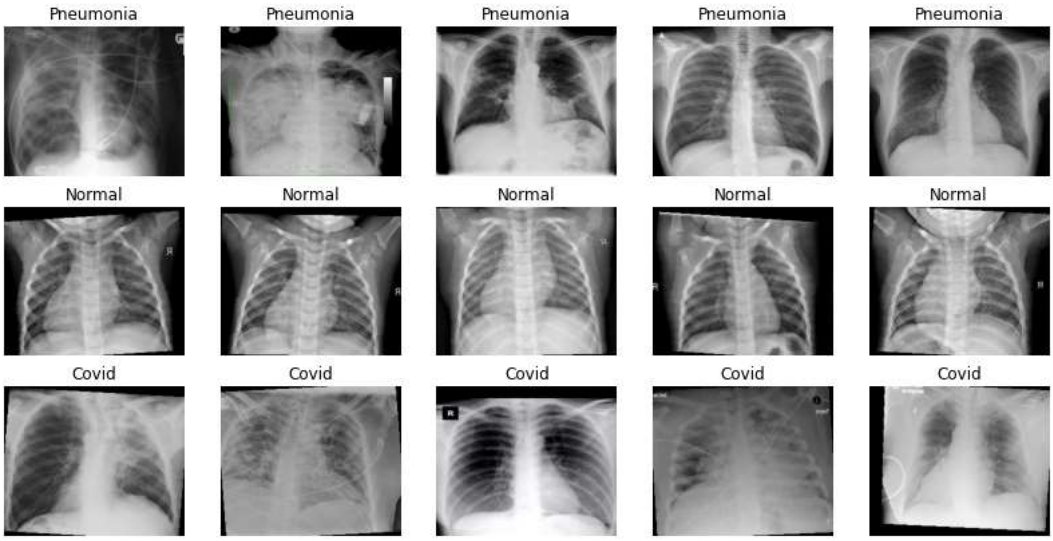


Figura 5.1 – Amostra de imagens para cada classe com as transformações aplicadas no aumento de dados.

5.2 Mini-Lotes

Os algoritmos de otimização para aprendizado de máquina normalmente calculam cada atualização dos parâmetros com base num valor esperado da função de custo, usando, para tanto, uma estimativa baseada na amostra disponível de dados (GOODFELLOW; BENGIO; COURVILLE, 2016). Não obstante, para conjuntos de dados com o numero elevado de imagens, realizar a estimativa completa da função custo ou de seu gradiente pode ser excessivamente custoso, particularmente se houver necessidade de repetir o processo diversas vezes para suporte de um método iterativo.

Uma forma de diminuir esse impacto sem a necessidade de recorrer a uma estratégia estritamente *online* é calcular uma média sobre um subconjunto aleatoriamente construído de amostras: tal subconjunto é chamado de mini-lote ou *minibatch* (RUSSELL; NORVIG, 2021).De acordo com (GOODFELLOW; BENGIO; COURVILLE, 2016), o tamanho dos mini-lotes devem ser definidos tendo em mente o seguinte compromisso: mini-lotes maiores fornecem uma estimativa mais precisa do gradiente, a um aumento de custo computacional. Alguns tipos de hardware alcançam melhor tempo de execução com tamanhos específicos de lotes: ao usar GPUs, é comum que lotes com tamanhos de

potência de 2 tenham um desempenho melhor. A potência de 2 geralmente utilizada varia entre os tamanhos de lote de 32 a 256, e 16 às vezes utilizado para modelos grandes. Uma dimensão mais modesta para os lotes pode surtir um efeito de regularização (WILSON; MARTINEZ, 2003). Neste trabalho, utilizaram-se os seguintes tamanhos de lotes: 8, 16 e 32.

5.3 Otimizadores

Na seção 3.7.4, foram apresentados fundamentos da otimização em aprendizado de máquina, bem como, os três algorítimos de otimização utilizados em todos os testes realizados neste trabalho, que foram escolhidos a partir da revisão da literatura. Os resultados de desempenho serão demonstrados e discutidos podem ser encontrados em 6.1.2.

6 Resultados e Comparações

Tanto os resultados obtidos na abordagem multiclasse quando na abordagem binária são expostos nesta seção. Os melhores resultados de cada modelo, bem como as configurações e parâmetros que levaram os modelos pré-treinados a alcançar seus melhores resultados são expostos nesta seção. Cabe ressaltar aqui que cada modelo passou por 54 testes, o que leva a um total de 432 testes, que exigiram cerca de 8 dias de processamento consecutivo com as configurações de hardware detalhadas no Capítulo 5.

6.1 Resultados Obtidos nos Testes

Na Tabela 6.1, podem-se observar os melhores resultados obtidos, no conjunto de dados de validação, para a abordagem multiclasse, e, na Tabela 6.2, são expostos os melhores resultados para a abordagem binária. Em ambos os casos, estão registrados os parâmetros utilizados para atingi-los, os quais produziram os maiores valores de acurácia.

Tabela 6.1 – Parâmetros utilizados que atingiram a melhor acurácia para a abordagem multiclasses.

Rede	Épocas	Passo	Tam. do Batch	Otimizador	Critério	Máx. Acurácia
AlexNet	100	1e-3	8	ADAM	Entropia Cruzada	94,41%
DenseNet			16			97,64%
GoogLeNet			8			96,20%
MobileNet			8			96,46%
ResNet			8			96,20%
ShuffleNet			8			95,93%
SqueezeNet			16			96,72%
VGG			8			96,55%

Através da Tabela 6.1, verificamos que a função de otimização ADAM e a taxa de aprendizado de 1e-3 foram as escolhas que levaram todos os modelos a atingirem os melhores resultados. O tamanho ótimo de minibatch foi 8, exceto para a SqueezeNet e DenseNet, que atingiram o melhor resultado com o valor 16. Outro ponto: o melhor otimizador foi sempre o ADAM. Já na abordagem binária, como mostrado na Tabela 6.2, a taxa de aprendizado e o tamanho do minibatch foram "escolhas unânime", mas não houve consenso quanto ao otimizador.

Tabela 6.2 – Parâmetros utilizados que atingiram a melhor acurácia para a abordagem binária.

Rede	Épocas	Passo	Tam. do Batch	Otimizador	Critério	Máx. Acurácia
AlexNet	100	1e-3	16	SGD	Entropia Cruzada	99,06%
DenseNet				ADAM		99,94%
GoogLeNet				ADAM		99,63%
MobileNet				ADAM		99,98%
ResNet				ADAM		99,98%
ShuffleNet				RMSprop		99,81%
SqueezeNet				SGD		99,56%
VGG				RMSprop		99,98%

6.1.1 Pontuação F1, Precisão e Sensibilidade

Antes de passarmos à análise desses resultados, revisitemos as seguintes nomenclaturas:

TP = *True Positive* (verdadeiro-positivo): resultado positivo em uma pessoa que está infectada pelo vírus;

TN = *True Negative* (verdadeiro-negativo): resultado negativo em uma pessoa não-infectada pelo vírus;

FP = *False Positive* (falso-positivo): resultado positivo em uma pessoa não infectada pelo vírus;

FN = *False Negative* (falso-negativo): resultado negativo em uma pessoa infectada pelo vírus.

A sensibilidade (taxa de verdadeiros positivos, também conhecida como *Recall*) é a capacidade de um teste diagnóstico identificar os verdadeiros positivos nos indivíduos verdadeiramente doentes (GOODFELLOW; BENGIO; COURVILLE, 2016), é obtida por:

$$Sensibilidade = \frac{TP}{TP + FN}. \tag{6.1}$$

Precisão: É uma medida para a correção de uma previsão positiva. Portanto, em termos simples, ela dá uma ideia do grau de confiança de que um resultado previsto como positivo seja efetivamente positivo (GOODFELLOW; BENGIO; COURVILLE, 2016). A definição segue a fórmula:

$$Precisão = \frac{TP}{TP + FP}. \tag{6.2}$$

Pontuação F1 (F1-Score): É uma média harmônica entre Precisão e Sensibilidade, permitindo uma combinação de fatores e trazendo uma visão complementar à da acurácia (GOODFELLOW; BENGIO; COURVILLE, 2016). É obtida através do cálculo:

$$F1 = \frac{2 * (precisão * sensibilidade)}{precisão + sensibilidade} = \frac{2 * TP}{(2 * TP) + FP + FN}. \quad (6.3)$$

Os valores de Medida F1, Precisão e Sensibilidade são expostos na Tabela 6.3 para os testes abordando o problema com múltiplas classes, e, na Tabela 6.4, para a abordagem com apenas duas classes. Reforçamos que as pontuações foram obtidas através das formulas 6.1, 6.2 e 6.3.

Tabela 6.3 – Maior precisão obtida intra-classe, de cada rede para a abordagem com múltiplas classes.

Modelos	Sensibilidade			Precisão			Pontuação F1		
Classes	Normal	Covid	Pneumonia	Normal	Covid	Pneumonia	Normal	Covid	Pneumonia
AlexNet	96%	95%	91%	98%	92%	95%	97%	93%	92%
DenseNet	97%	97%	96%	100%	96%	96%	99%	97%	96%
GoogLeNet	97%	97%	96%	100%	96%	96%	99%	97%	96%
MobileNet	96%	95%	95%	100%	95%	95%	96%	95%	95%
ResNet	95%	94%	95%	100%	95%	95%	94%	95%	95%
ShuffleNet	96%	94%	93%	99%	93%	94%	96%	94%	94%
SqueezeNet	97%	97%	95%	99%	95%	96%	98%	96%	95%
Vgg	94%	95%	95%	100%	95%	95%	96%	95%	94%

Tabela 6.4 – Maior precisão obtida intra-classe, de cada rede para a abordagem com apenas duas classes, Pneumonia e Covid.

Modelos	Sensibilidade		Precisão		Pontuação F1	
Classes	Covid	Pneumonia	Covid	Pneumonia	Covid	Pneumonia
AlexNet	99%	99%	99%	99%	99%	99%
DenseNet	99%	99%	99%	99%	99%	99%
GoogLeNet	100%	99%	99%	100%	100%	100%
MobileNet	100%	100%	100%	100%	100%	100%
ResNet	100%	100%	100%	100%	100%	100%
ShuffleNet	100%	100%	100%	100%	100%	100%
SqueezeNet	100%	100%	100%	100%	100%	100%
Vgg	99%	99%	99%	99%	99%	99%

6.1.2 Algoritmos de Otimização

Nos testes realizados neste trabalho, também foi possível avaliar o desempenho das estratégias de otimização utilizadas. Desta forma, buscou-se analisar, separadamente,

o resultado de maior acurácia para cada modelo em relação a uma função de otimização específica. A Tabela 6.5 se refere à abordagem com múltiplas classes, e Tabela 6.6 diz respeito à abordagem binária. Percebe-se, no geral, uma tendência de melhor desempenho por parte do ADAM e do RMSProp.

Tabela 6.5 – Melhor acurácia atingida por cada otimizador testado na abordagem multiclasse.

REDES	Acurácia Média		
Otimizador	SGD	ADAM	RMSProp
SqueezeNet	96, 04%	96, 72%	96, 46%
DenseNet	97, 25%	97, 64%	97, 38%
ResNet	95, 41%	96, 20%	96, 07%
VGG	95, 38%	96, 55%	96, 07%
AlexNet	94, 10%	95, 45%	94, 10%
GoogleNet	95, 54%	96, 20%	95, 07%
MobileNet	96, 20%	96, 46%	96, 20%
ShuffleNet	92, 32%	95, 93%	94, 18%

Tabela 6.6 – Melhor acurácia atingida por cada otimizador testado na abordagem binária.

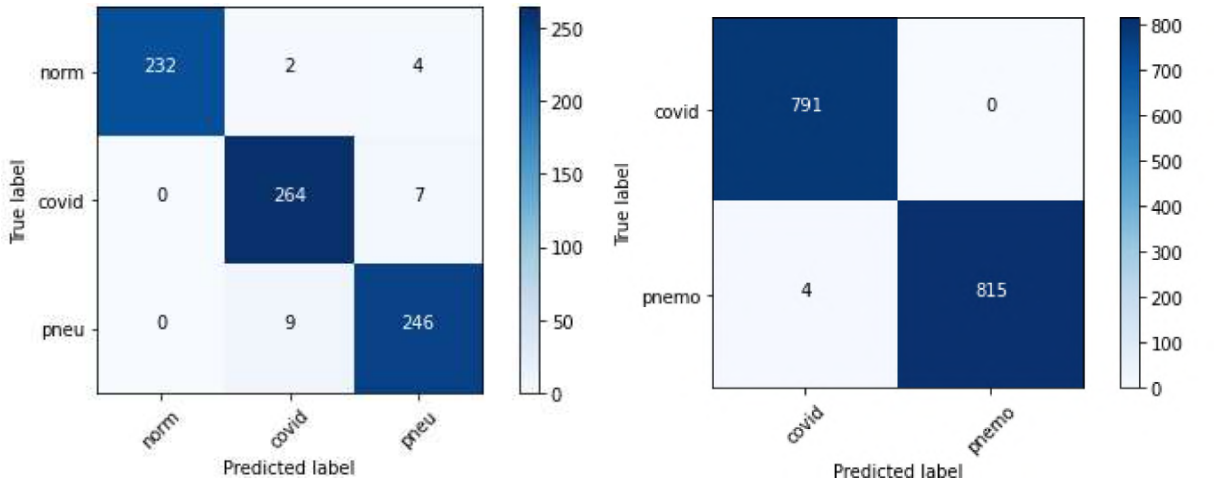
REDES	Acurácia Média		
Otimizador	SGD	ADAM	RMSProp
SqueezeNet	99, 56%	99, 08%	98, 98%
DenseNet	99, 90%	99, 94%	99, 05%
ResNet	99, 10%	99, 63%	99, 08%
VGG	99, 52%	99, 77%	99, 98%
AlexNet	99, 06%	98, 88%	98, 76%
GoogleNet	99, 63%	99, 23%	98, 87%
MobileNet	99, 83%	99, 98%	99, 87%
ShuffleNet	99, 55%	99, 53%	99, 81%

6.1.3 Melhor Resultado Obtido

O melhor resultado obtido na abordagem multiclasse foi o da rede DenseNet — como demonstrado na Tabela 6.5 — quando foi utilizado o otimizador ADAM: a acurácia foi de 97,64%. Pode-se também observar, na Tabela 6.3, sua pontuação de sensibilidade, precisão e F1 Score.

Nas Figuras 6.1a e 6.1b, é possível visualizar as matrizes de confusão resultantes dos testes da rede DenseNet, tanto no caso multiclasse (Figura 6.1a), quanto no caso binário (Figura 6.1b). Já na Figura 6.2, são apresentadas as curvas que expressam a

evolução da acurácia e da taxa de erro para o caso multiclasse, e a mesma informação pode ser encontrada na Figura 6.3 para o caso binário.



(a) Multiclasse. (b) Binário.

Figura 6.1 – Matrizes de confusão resultantes dos testes realizados com a rede DenseNet.

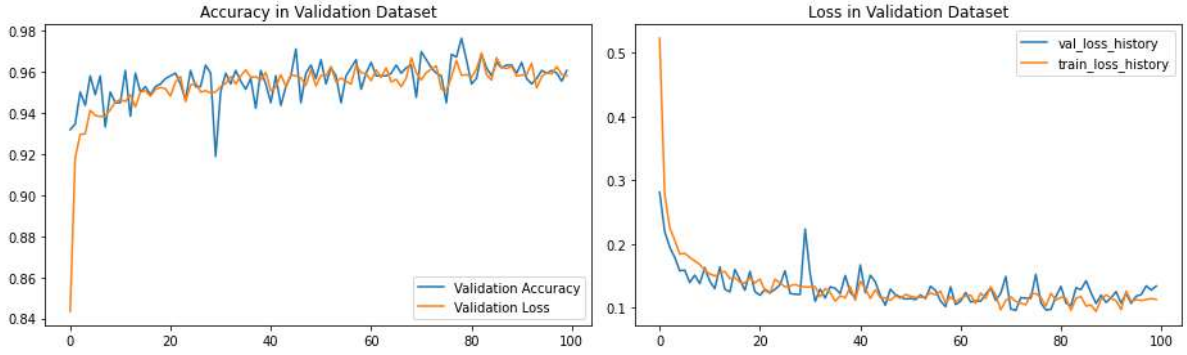


Figura 6.2 – Curvas da acurácia (esquerda) e taxa de erro (direita) ao longo de 100 épocas de treinamento, abordando o modelo com múltiplas classes.

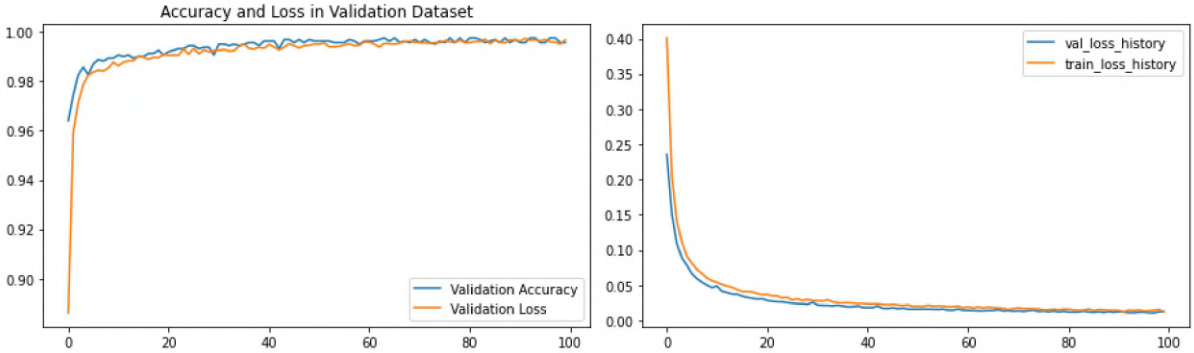


Figura 6.3 – Curvas da acurácia (esquerda) e taxa de erro (direita) ao longo de 100 épocas de treinamento, abordando o modelo com apenas duas classes: 'Covid' e 'Pneumonia'.

6.1.4 Mapas de Calor

O Mapeamento de ativação das classes, também conhecido como Class Activation Mapping (CAM) ou mapa de calor (*Heat Map*), tem por objetivo mostrar quais partes da imagem estão mais ativas, do ponto de vista da rede, no momento da classificação. A cor vermelha indica forte ativação na região, a cor amarela indica uma ativação pouco menor e as cores azul claro e azul escuro representam baixa ou nenhuma ativação no local.

O embasamento matemático e metodológicos foram implementados de acordo com (ZHOU et al., 2016), que, segundo os autores, antes da camada final do modelo, é utilizado o Global Average Pooling (GAP) nos mapas de características das camadas convolucionais anteriores à classificação, essas características são passadas para a camada totalmente conectada que produz a classificação. Então pode-se identificar a importância de uma região na imagem, resgatando os pesos da camada de saída nos mapas de atributos das camadas convolucionais. O GAP produz uma média do mapa de características para cada peso da última camada convolucional, e então, a soma ponderada desses valores é usada para gerar a saída final para cada um dos mapas de características. Onde, finalmente, calcula-se a soma ponderada dos mapas de características para obter o mapa de ativação final, onde, os *pixels* da imagem com os maiores pesos indicam maior atenção da rede, assim, dando a cor avermelhada para locais com maior atenção da rede, e cores próximas às azuis para regiões com menor atenção da rede, este procedimento pode ser observado na Figura 6.4.

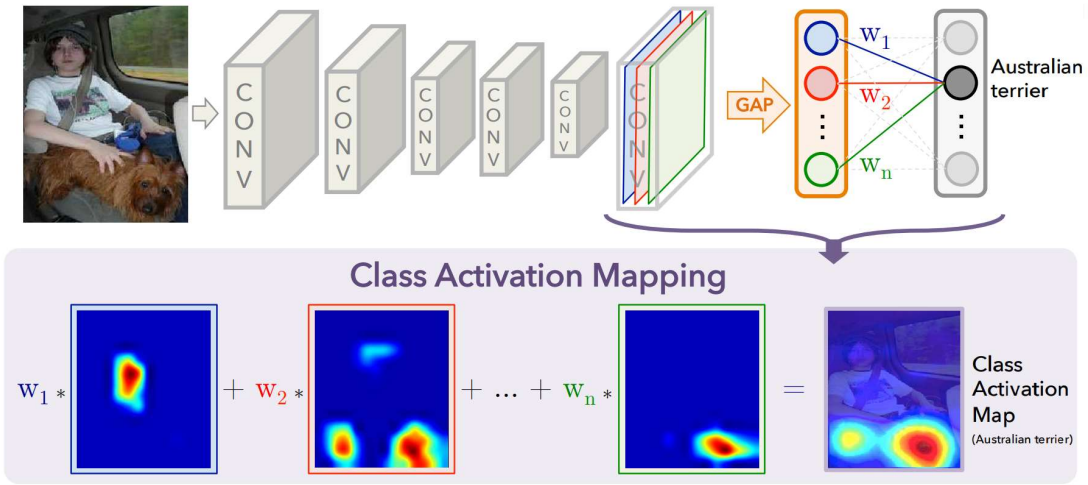


Figura 6.4 – Mapeamento de ativação de classe: a pontuação de classe predita pelo modelo neural é mapeada de volta para a camada convolucional anterior para gerar os mapas de ativação de classe (CAMs), que destaca as regiões que descrevem de forma especifica que a imagem pertence a uma determinada classe (ZHOU et al., 2016).

Nota-se que na Figura 6.5 é possível observar que, do ponto de vista dos classificadores das redes neurais convolucionais utilizadas neste trabalho, as classes 'covid'

e ‘pneumonia’ são bastante distintas 6.6, sendo que as imagens da classe ‘pneumonia’ têm a maior concentração de calor (em vermelho) do lado esquerdo da figura. Já as imagens classificadas como ‘covid’ têm a maior concentração de calor nos lados direto e esquerdo superior. Julgamos que isso explique por que o teste binário alcançou uma acurácia bastante alta em todos os testes.

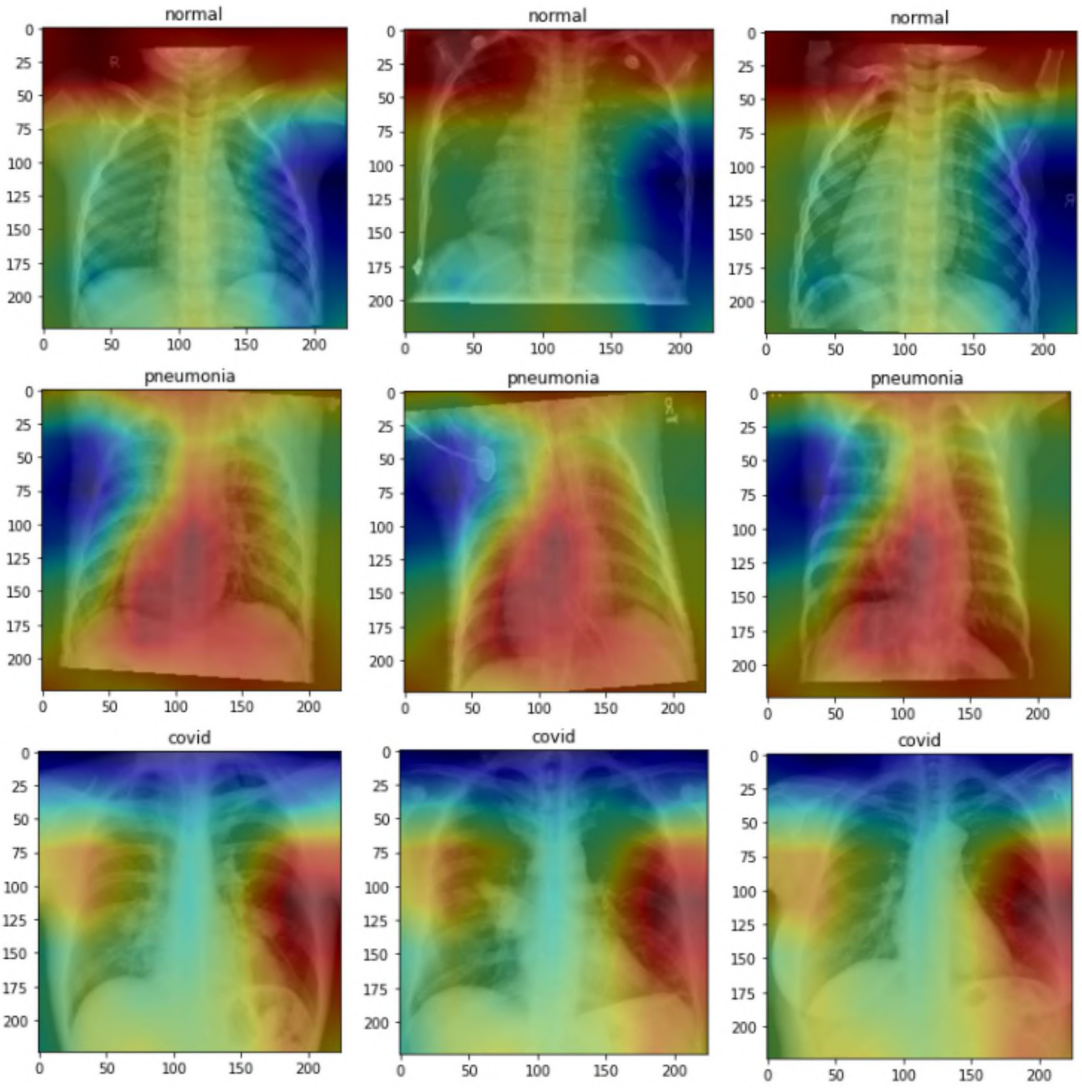


Figura 6.5 – Mapa de Calor para classificação Multiclasse.

Os resultados mostram, em primeiro lugar, que as metodologias de redes profundas são bastante promissoras para formar a base de uma ferramenta para auxílio ao diagnóstico da COVID-19. Para que se chegue a uma ferramenta ainda mais funcional, além da continuidade de investigações como a realizada neste trabalho, será importante que bases de dados maiores e mais sistematicamente organizadas sejam construídas.

Em termos mais específicos, julgamos que houve uma tendência de melhor desempenho por parte do otimizador ADAM. Para a abordagem multiclasse, a rede DenseNet teve desempenho ligeiramente superior ao das demais, e, na abordagem com

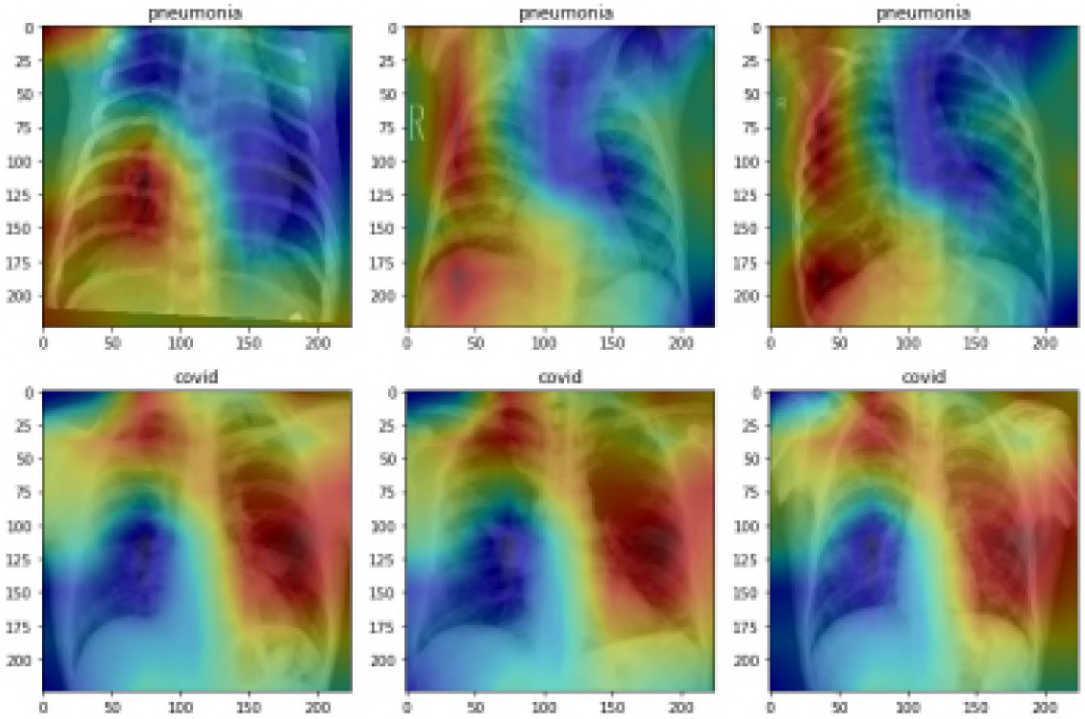


Figura 6.6 – Mapa de Calor para classificação Binária.

duas classes, os desempenhos foram muito similares. Isso indica um caminho inicial para pesquisa, mas, a nosso ver, a opção final por elementos de um paradigma específico requer um estudo com mais dados e, preferencialmente, sob condições reais, com apoio de profissionais da área médica.

Os mapas de calor foram muito importantes porque indicaram que, efetivamente, as redes buscarem informação aparentemente relevante junto à imagem. Sabe-se que, por seu grande poder de aproximação, as redes profundas têm o potencial de sofrer sobreajuste, sendo importante buscar formas de evitar resultados falsamente encorajadores, que terminarão por se mostrar enganosos numa implementação prática subsequente.

Caberia, agora, realizar uma investigação mais ampla sobre seu desempenho tendo em vista a perspectiva de uma aplicação mais próxima da realidade do diagnóstico clínico. Para tanto, num trabalho futuro, seria importante cooperar com os profissionais da área médica, buscando formar uma base ampla de imagens e aproximando as condições de projeto da estrutura presente daquelas vividas por quem atua na linha de frente de combate à pandemia.

7 Conclusão

Neste trabalho, realizou-se uma análise comparativa de diferentes abordagens de redes neurais na solução do problema de classificação de imagens de raios-X de tórax. A meta era permitir a detecção de padrões correspondentes àqueles de pacientes com COVID-19, fornecendo subsídios para a pesquisa nessa área, que tem grande relevância no contexto atual, bem como testar um grupo representativo de modelos de redes pré-treinadas, utilizando a técnica de transferência de aprendizado, a fim de possibilitar uma análise comparativa significativa. Ressalta-se que houve uma investigação do desempenho dos modelos em condições variadas de hiperparâmetros e otimizadores.

Como esperado, o problema na forma binária levou a um melhor desempenho frente ao caso multiclasse - não obstante, convém buscar alguma forma de análise de ambos os casos do prisma da teoria da informação. Foi possível concluir que diferentes estruturas convolucionais foram capazes de abordar, de maneira competente, o problema de classificação em esquema de transferência de aprendizado. A DenseNet se mostra uma opção bem interessante, assim como o otimizador ADAM.

Como perspectiva de trabalho futuro, apontamos a continuidade da investigação de técnicas de aprendizado profundo, com a preocupação de que o estudo realizado seja rigoroso para evitar resultados que não se confirmem na prática. Julgamos que seria muito interessante buscar a cooperação de profissionais da área médica, talvez no âmbito de um projeto formal, para buscar construir uma base de dados em condições uniformes e controladas, bem como para realizar testes em situações reais de trabalho.

Referências

ABBAS, A.; ABDELSAMEA, M. M.; GABER, M. M. Classification of covid-19 in chest x-ray images using detrac deep convolutional neural network. *Applied Intelligence*, Springer, v. 51, n. 2, p. 854–864, 2021. Citado na página 22.

ARORA, S. et al. Provable bounds for learning some deep representations. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2014. p. 584–592. Citado na página 55.

BASSI, P. R. A. S. *A Study of Deep Neural Networks for Image Recognition in BCIs and COVID-19 Detection*. [S.l.]: Dissertação de Mestrado, FEEC/UNICAMP, 2021. Citado na página 23.

BASU, S.; MITRA, S.; SAHA, N. Deep learning for screening covid-19 using chest x-ray images. In: IEEE. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. [S.l.], 2020. p. 2521–2527. Citado na página 21.

BISHOP, C. M. *Neural networks for pattern recognition*. [S.l.]: Oxford university press, 1995. Citado 15 vezes nas páginas 9, 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 e 48.

BLOCK, H.-D. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, APS, v. 34, n. 1, p. 123, 1962. Citado na página 30.

BOUREAU, Y.-L.; PONCE, J.; LECUN, Y. A theoretical analysis of feature pooling in visual recognition. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. [S.l.: s.n.], 2010. p. 111–118. Citado na página 46.

BURKOV, A. *The hundred-page machine learning book*. [S.l.]: Andriy Burkov Canada, 2019. v. 1. Citado 5 vezes nas páginas 9, 25, 43, 44 e 45.

CHOWDHURY, M. E. H. et al. Can ai help in screening viral and covid-19 pneumonia? *IEEE Access*, v. 8, p. 132665–132676, 2020. Citado na página 70.

CIFAR. *PCR-RT*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://www.cs.toronto.edu/~kriz/cifar.html>. Citado 2 vezes nas páginas 50 e 53.

COVID-19 image data collection. *COVID-19 image data collection*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://github.com/ieee8023/covid-chestxray-dataset>. Citado na página 70.

COVID-19 Radiography Database on Kaggle, T. *COVID-19 Radiography Database on Kaggle*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://www.kaggle.com/tawsifurrahman/covid19-radiography-database>. Citado na página 70.

DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, v. 12, n. 7, 2011. Citado na página 49.

DUDA, R. O.; HART, P. E. et al. *Pattern classification and scene analysis*. [S.l.]: Wiley New York, 1973. v. 3. Citado na página 26.

GAÁL, G.; MAGA, B.; LUKÁCS, A. Attention u-net based adversarial architectures for chest x-ray lung segmentation. *arXiv preprint arXiv:2003.10304*, 2020. Citado na página 22.

Girshick, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 580–587. Citado na página 56.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado 20 vezes nas páginas 9, 10, 19, 20, 37, 38, 39, 41, 42, 43, 45, 46, 47, 48, 49, 70, 71, 72, 75 e 76.

HASEGAWA, A. et al. A shift-invariant neural network for the lung field segmentation in chest radiography. *VLSI Signal Processing*, v. 18, p. 241–250, 04 1998. Citado na página 19.

He, K. et al. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 770–778. Citado 7 vezes nas páginas 10, 13, 60, 61, 62, 63 e 64.

He, K. et al. Deep residual learning for image recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 770–778. Citado na página 62.

HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, v. 14, n. 8, p. 2, 2012. Citado na página 49.

Huang, G. et al. Densely connected convolutional networks. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2017. p. 2261–2269. Citado 5 vezes nas páginas 10, 13, 53, 54 e 55.

IANDOLA, F. N. et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. Citado 5 vezes nas páginas 10, 13, 65, 66 e 67.

ImageNet Project. *ImageNet Overview*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <http://www.image-net.org/>. Citado 3 vezes nas páginas 19, 50 e 61.

ImageNet Project LSVRC. *The ImageNet Large Scale Visual Recognition Challenge (LSVRC)*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <http://www.image-net.org/challenges/LSVRC/>. Citado 4 vezes nas páginas 53, 61, 63 e 68.

ImageNet Project LSVRC. *Now your meme: We need to go deeper*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <http://knowyourmeme.com/memes/we-need-to-go-deeper>. Citado na página 55.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 02 2015. Citado 2 vezes nas páginas 13 e 67.

JAIN, R. et al. Deep learning based detection and analysis of covid-19 on chest x-ray images. *Applied Intelligence*, Springer, v. 51, n. 3, p. 1690–1700, 2021. Citado na página 21.

JANGAM, E.; BARRETO, A. A. D.; ANNAVARAPU, C. S. R. Automatic detection of covid-19 from chest ct scan and chest x-rays images using deep learning, transfer learning and stacking. *Applied Intelligence*, Springer, p. 1–17, 2021. Citado na página 22.

JIA, Y.; HUANG, C.; DARRELL, T. Beyond spatial pyramids: Receptive field learning for pooled image features. In: IEEE. *2012 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.], 2012. p. 3370–3377. Citado na página 46.

Johns Hopkins University and Medicine. COVID-19 Map. *Johns Hopkins University and Medicine. COVID-19 Map - Johns Hopkins Coronavirus Resource Center*. [S.l.], 2021. Acessado em: 2021-11-09. Disponível em: <https://coronavirus.saude.mg.gov.br/blog/84-grupos-de-risco-para-covid-19>. Citado na página 18.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, ACM New York, NY, USA, v. 60, n. 6, p. 84–90, 2017. Citado 4 vezes nas páginas 51, 52, 63 e 66.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989. Citado na página 40.

LeCun, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, v. 1, n. 4, p. 541–551, 1989. Citado na página 56.

LIN, M.; CHEN, Q.; YAN, S. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. Citado 2 vezes nas páginas 55 e 56.

LIN, T.-Y. et al. Microsoft coco: Common objects in context. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 740–755. Citado 3 vezes nas páginas 50, 61 e 63.

MICHELUCCI, U. *Advanced applied deep learning: convolutional neural networks and object detection*. [S.l.]: Springer, 2019. Citado 2 vezes nas páginas 43 e 44.

Ministério da Saude Brasil. *Diagnóstico Clínico e Laboratorial*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://coronavirus.saude.gov.br/diagnostico-clinico-e-laboratorial>. Citado na página 18.

MINSKY, M.; PAPERT, S. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969. Citado 2 vezes nas páginas 32 e 33.

MITCHELL, T.; MCGRAW-HILL, M. L. *Edition*. [S.l.]: New York: McGraw-Hill, Inc, 1997. Citado na página 24.

MITCHELL, T. M. *The discipline of machine learning*. [S.l.]: Carnegie Mellon University, School of Computer Science, Machine Learning . . . , 2006. v. 9. Citado na página 25.

Muneeb ul Hassan, V. *VGG16 - Convolutional Network for Classification and Detection*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://neurohive.io/en/popular-networks/vgg16/>. Citado 2 vezes nas páginas 10 e 68.

MURPHY, K. P. *Machine learning: a probabilistic perspective*. [S.l.]: MIT press, 2012. Citado na página 25.

OZTURK, T. et al. Automated detection of covid-19 cases using deep neural networks with x-ray images. *Computers in biology and medicine*, Elsevier, v. 121, p. 103792, 2020. Citado na página 23.

Pan, S. J.; Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, v. 22, n. 10, p. 1345–1359, 2010. Citado na página 50.

PCR-RT., Ministério da Saude Brasil. *PCR-RT*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://coronavirus.saude.mg.gov.br/blog/70-pcr-rt-para-coronavirus>. Citado na página 18.

RAHMAN, T. et al. Exploring the effect of image enhancement techniques on covid-19 detection using chest x-ray images. *Computers in Biology and Medicine*, v. 132, p. 104319, 2021. ISSN 0010-4825. Disponível em: <https://www.sciencedirect.com/science/article/pii/S001048252100113X>. Citado na página 70.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado 2 vezes nas páginas 30 e 31.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986. Citado na página 38.

RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, Springer, v. 115, n. 3, p. 211–252, 2015. Citado 4 vezes nas páginas 50, 51, 55 e 61.

RUSSELL, S.; NORVIG, P. *Artificial intelligence: a modern approach - Fourth Edition - Global Edition*. [S.l.]: Pearson Education, 2021. Citado 8 vezes nas páginas 25, 38, 40, 44, 45, 48, 50 e 72.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, IBM, v. 3, n. 3, p. 210–229, 1959. Citado na página 24.

SANDLER, M. et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 4510–4520. Citado 7 vezes nas páginas 10, 13, 57, 58, 59, 60 e 63.

Secretaria de Saúde de Minas Gerais. *PCR-RT*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://coronavirus.saude.mg.gov.br/blog/84-grupos-de-risco-para-covid-19>. Citado na página 18.

Sik-Ho Tsang. *Review: AlexNet, CaffeNet — Winner of ILSVRC 2012 (Image Classification)*. [S.l.], 2018. Acessado em: 2022-02-03. Disponível em: <https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160>. Citado 2 vezes nas páginas 10 e 53.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014. Citado 5 vezes nas páginas 60, 61, 63, 68 e 69.

SVHN. *PCR-RT*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <http://ufldl.stanford.edu/housenumbers/>. Citado 2 vezes nas páginas 50 e 53.

Szegedy, C. et al. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. p. 1–9. Citado 5 vezes nas páginas 10, 13, 55, 56 e 57.

Tensorflow. *Mobilenetv2 source code*. [S.l.], 2021. Acessado em: 2021-12-06. Disponível em: <https://github.com/tensorflow/models/tree/master/research/slim/netsr>. Citado na página 57.

TURING, A. M. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX, n. 236, p. 433–460, 10 1950. ISSN 0026-4423. Disponível em: <https://doi.org/10.1093/mind/LIX.236.433>. Citado na página 24.

WIDROW, B.; HOFF, M. E. *Adaptive switching circuits*. [S.l.], 1960. Citado na página 31.

WIDROW, B.; LEHR, M. A. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, IEEE, v. 78, n. 9, p. 1415–1442, 1990. Citado na página 30.

WILSON, D. R.; MARTINEZ, T. R. The general inefficiency of batch training for gradient descent learning. *Neural networks*, Elsevier, v. 16, n. 10, p. 1429–1451, 2003. Citado na página 73.

Xie, S. et al. Aggregated residual transformations for deep neural networks. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2017. p. 5987–5995. Citado 2 vezes nas páginas 62 e 63.

XU, J.; XUE, K.; ZHANG, K. Current status and future trends of clinical diagnoses via image-based deep learning. *Theranostics*, v. 9, p. 7556–7565, 10 2019. Citado 2 vezes nas páginas 10 e 51.

ZHANG, X. et al. Viral and host factors related to the clinical outcome of covid-19. *Nature*, Nature Publishing Group, v. 583, n. 7816, p. 437–440, 2020. Citado 2 vezes nas páginas 18 e 19.

ZHANG, X. et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2018. p. 6848–6856. Citado 6 vezes nas páginas 10, 13, 62, 63, 64 e 65.

ZHOU, B. et al. Learning deep features for discriminative localization. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 2921–2929. Citado 2 vezes nas páginas 11 e 79.

ZHOU, Y.-T.; CHELLAPPA, R. Computation of optical flow using a neural network. In: *ICNN*. [S.l.: s.n.], 1988. p. 71–78. Citado na página 45.