UNIFBV | WYDEN
CENTRO UNIVERSITÁRIO UNIFBV
COORDENAÇÃO DE ENGENHARIA
CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Matheus de Lima Santana

Matheus Trajano Antonino

DETECÇÃO E CLASSIFICAÇÃO DE DOENÇAS RESPIRATÓRIAS INFECCIOSAS POR IMAGENS DE RAIO X UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Recife
2022

Matheus de Lima Santana
Matheus Trajano Antonino

DETECÇÃO E CLASSIFICAÇÃO DE DOENÇAS RESPIRATÓRIAS INFECCIOSAS POR IMAGENS DE RAIO X UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de conclusão apresentado ao curso de graduação, como parte dos requisitos necessários à obtenção do título de Bacharel em Ciências da Computação.

Recife
2022

Matheus de Lima Santana
Matheus Trajano Antonino

DETECÇÃO E CLASSIFICAÇÃO DE DOENÇAS RESPIRATÓRIAS INFECCIOSAS POR IMAGENS DE RAIO X UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Trabalho de conclusão apresentado ao curso de graduação, como parte dos requisitos necessários à obtenção do título de Bacharel em Ciências da Computação.

Aprovado em: ___/___/____

Fausto José Feitosa Barbosa Gominho
Orientador(a)

Professor 1
Avaliador(a)

Professor 2
Avaliador(a)

Recife
2022

Do aluno Matheus Trajano Antonino, este trabalho é todo dedicado aos meus pais, pois é graças ao esforço deles que hoje posso concluir o meu curso.

Do aluno Matheus de Lima Santana, dedico este trabalho ao meu pai e minha mãe, que nunca mediram esforços perante a minha educação, a minha família que é base, a minha namorada pelo companheirismo durante esta jornada e a todos os amigos e principalmente minha dupla.

AGRADECIMENTOS

Do aluno Matheus Trajano Antonino, agradeço a meu pai e minha mãe por sempre estarem presentes e me apoiarem durante toda a minha caminhada nesse curso, sem eles com certeza essa trajetória teria sido muito mais árdua. Agradeço a minha noiva que sempre me deu apoio, carinho e incentivo. A meu amigo e dupla, que fez parte de toda minha formação e continuará presente em minha vida com certeza. A todos que direta ou indiretamente fizeram parte de minha formação, o meu muito obrigado.

Do aluno Matheus de Lima Santana, agradeço primeiramente a minha mãe por sempre ter me dado forças e me direcionado durante toda minha vida acadêmica, sendo um exemplo. Ao meu pai por garantir meus estudos e ter se dedicado tanto a cuidar desse núcleo familiar. Além disso, a toda minha família que sempre me acolheu e teve parte integral no meu caráter atualmente. Ao meu primo Igor por ser a porta de entrada no universo do curso de ciências da computação e que me ajudou em diversas outras vezes como mentor, amigo e um dos melhores desenvolvedores que conheço e admiro.

Ao meu amigo Heitor que foi conselheiro e um ombro amigo nos momentos difíceis na faculdade, sempre com conversas certeiras e importantes que abriram meus olhos diversas vezes. A todos os meus amigos que fiz na faculdade e que me ensinaram tanto quanto as aulas, pelas risadas e pelo companheirismo. Também aos amigos que não foram encontrados na faculdade, mas que acompanharam um pouco que seja a minha trajetória.

A minha namorada Bruna Oliveira de Sousa, a maior companheira e com quem compartilhei as dores durante a escrita deste trabalho.

Principalmente a minha dupla, meu xará Matheus, parceiro não só na conclusão do curso, mas durante todo ele, com certeza tudo isso fez com que nos déssemos muito bem para terminar este trabalho.

E por fim, a todas as pessoas que fizeram, fazem e ainda vão fazer parte da minha história.

“In A.I., the holy grail was how do you generate internal representations.”
Geoffrey Hinton

RESUMO

Devido à pandemia causada pelo novo coronavírus, COVID-19, a busca pela rápida identificação e controle da doença tornou-se ponto crítico na área da saúde em contexto mundial. Os métodos de diagnósticos foram se desenvolvendo pouco a pouco e, apesar de ser um complemento para identificação, o uso de radiografias pulmonares foi adotado. Além disso, um grupo de doenças pulmonares também podem passar pela identificação a partir de radiografias: pneumonia e tuberculose. O processamento de imagens é um tema já bastante estudado na literatura de inteligência artificial e, mais especificamente, na área da visão computacional. Neste trabalho o objeto de estudo principal é utilizar a arquitetura de redes neurais convolucionais para auxiliar na classificação e possível detecção desse grupo de doenças observáveis por raios X. A pesquisa foi realizada baseada na análise de três modelos neurais e da comparação dos seus resultados aplicando mudanças na arquitetura e técnicas de *deep learning*.

Palavras-chave: COVID-19; redes neurais convolucionais; radiografia; visão computacional.

ABSTRACT

Due to the pandemic brought by the new coronavirus, COVID-19, the search for rapid identification and control of the disease has become a critical issue in health care worldwide. The methods of diagnosis have been developed little by little, and although it is a complement for identification, the use of lungs radiographs has been adopted. In addition, a group of lung diseases can also be identified from radiographs, these are pneumonia and tuberculosis. Image processing is a topic already well studied in the artificial intelligence literature and, more specifically, in the computer vision area. In this work the main object of study is to use convolutional neural network architecture to assist in the classification and possible detection of this group of diseases observable by X-rays. The research was conducted by analyzing three neural models and comparing their results by applying changes in the architecture and deep learning techniques.

Keywords: COVID-19; convolutional neural networks; radiography; computer vision.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

SARS-CoV-2	*Severe acute respiratory syndrome-associated coronavirus 2*
OMS	Organização Mundial de Saúde
COVID-19	Coronavirus disease 19
SARS	*Severe acute respiratory syndrome*
PCR-RT	*Real-time polymerase chain reaction*
CNN	*Convolutional neural networks*
ANN	*Artificial neural networks*
MLP	*Multilayer perceptron*
SRM	*Structural risk minimization*

SUMÁRIO

1 INTRODUÇÃO

O novo coronavírus da síndrome respiratória aguda grave 2 (SARS-CoV-2), do inglês *severe acute respiratory syndrome-associated coronavirus 2*, assola o globo com seus primeiros casos sendo datados, segundo a Organização Mundial de Saúde (OMS), no final de dezembro de 2019, em Wuhan, na China. Por ser caracterizado como um vírus de infecção respiratória possui uma disseminação simplificada, alcançando assim escalas globais em 2020 e transformando completamente o estilo de vida de todos os habitantes ao redor do planeta (BRITO *et al.*, 2020).

O SARS-CoV-2 é responsável pela doença causada pelo novo coronavírus 2019, do inglês *coronavirus disease 19* (COVID-19), que provoca consequentemente a síndrome respiratória aguda grave, do inglês *severe acute respiratory syndrome* (SARS). Dentre os pacientes acometidos pela COVID-19 cerca de 80% apresentaram forma leve a moderada da patologia, e nas formas mais graves é possível se observar complicações mais aparentes no sistema respiratório (LIMA, 2020).

O principal método de diagnóstico da COVID-19 é o teste molecular (PCR-RT), do inglês *real-time polymerase chain reaction*, que possui maior acurácia e precisão na constatação dos resultados. Entretanto, a radiografia na região torácica, ganhou um importante papel na avaliação dos pacientes acometidos, mesmo não sendo recomendada a sua utilização como determinante no reconhecimento da doença. Através da análise das imagens é possível identificar sinais da patologia na região pulmonar de maneira mais rápida em relação ao PCR-RT (MEIRELLES, 2020).

Contudo, a COVID-19, por ser um problema com sintomas semelhantes às outras infecções pulmonares, apresenta aos profissionais a dificuldade na classificação por meio dessas imagens de raio X (SILVA MÜLLER; L. MÜLLER, 2020). Devido a essas similaridades e a notável baixa qualidade das imagens, a área da tecnologia, especificamente a inteligência artificial, consegue prestar suporte para a melhor precisão do diagnóstico.

Sendo assim, tratando-se da inteligência artificial, a análise de exames radiográficos por imagens é alocada na literatura da visão computacional. Nestes

estudos é possível propor sistemas capazes de detectar padrões e gerar novas informações a partir dos elementos que compõem a imagem (MARTINS *et al.*, 2020).

Na visão computacional, as Redes Neurais Convolucionais (*CNN*), do inglês *Convolutional Neural Networks*, são consideradas um modelo matemático de classificação inspirado no sistema biológico de processamento de dados visuais (SOUZA *et al.*, 2017 *apud* GU *et al.*, 2015; JASSMANN *et al.*, 2015).

As *CNNs* apresentam como diferença às redes neurais artificiais a possibilidade da aplicação de filtros nos dados de entrada, às imagens em um primeiro momento, e durante o processamento das camadas, seus dados aprofundados. Esses filtros constroem uma hierarquia de características capaz de descrever os resultados com maior acurácia (KRIZHEVSKY *et al.*, 2012; CIRESAN *et al.*, 2011).

Dessa forma, uma *CNN* aplicada às análises de imagens radiográficas com foco não somente na classificação da COVID-19, mas de outras possíveis infecções pulmonares mais conhecidas, como a pneumonia e tuberculose, pode trazer grande suporte no diagnóstico para os radiologistas e até outros profissionais da área de saúde que necessitam dessa análise mais rápida por meio desse exame.

1.1 Contextualização do problema da pesquisa

As redes neurais convolucionais se destacam na área da visão computacional como um dos principais modelos para o processamento de imagens. Sua utilização poder ser estendida a diversas categorias deste segmento e principalmente na classificação dessas imagens com diminuição da complexidade algorítmica e de *hardware*.

Na área da radiologia, a análise das imagens geradas é usualmente utilizada visando a busca por lesões e padrões que auxiliem na classificação de duas ou mais classes possíveis em um modelo. Por exemplo, a classificação de nódulos, benignos ou malignos, nos pulmões a partir de tomografias (YAMASHITA *et al.*, 2018).

Logo, este trabalho tem como finalidade a construção de uma série de modelos convolucionais comparados para análise dos resultados de classificação, com acurácia satisfatória sobre um determinado grupo de doenças respiratórias infecciosas que possuem padrões particulares observáveis durante uma análise

clínica dos exames realizados, sendo elas: pneumonia, tuberculose e COVID-19 por imagens de radiografias do tórax.

1.2 Objetivos

1.2.1 Objetivo geral

Construir um classificador de pneumonia, tuberculose e COVID-19, a partir de imagens de raio X utilizando modelos neurais convolutivos para o processamento dessas imagens e obtendo um nível de acurácia satisfatório.

1.2.2 Objetivos específicos

- Construir uma base de dados contendo as imagens de cada categoria de doença utilizada na pesquisa, além de um conjunto de imagens de pulmões classificados como saudáveis (normais);
- Desenvolver um conjunto de redes neurais convolucionais aplicando diferentes técnicas para processamento das imagens de raio X torácico;
- Analisar e comparar os resultados entre os modelos convolucionais desenvolvidos de modo a avaliar seu desempenho.

1.3 Justificativa

Devido à pandemia causada pelo vírus da COVID-19, a procura por métodos de controle e detecção da doença se tornaram pauta obrigatória dada a proporção do tema.

Na área da inteligência artificial e mais especificamente de *deep learning*, o processamento de imagens já tem uma longa base e pode ser de grande auxílio para áreas que lidam com esse tipo de dado. O desenvolvimento de modelos neurais para classificação de imagens radiográficas já era uma realidade antes e, com a pandemia, houve um esforço nesse tópico em busca de alternativas para agilizar a detecção da COVID-19.

Este trabalho tem como fundamento o intuito de alcançar um modelo com boa otimização, considerando sua acurácia e aumento no nível de segurança de um possível diagnóstico ou método que complemente o mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

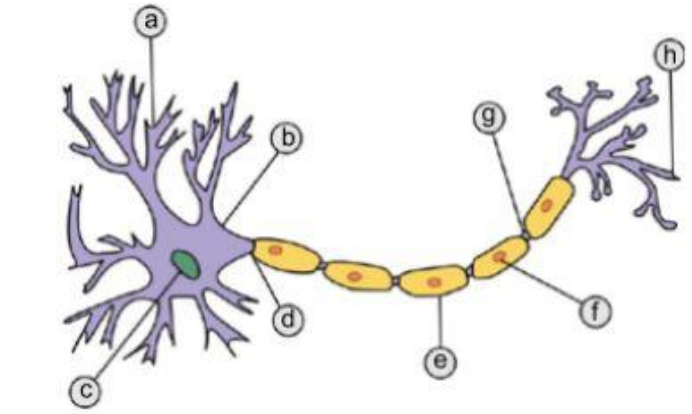
Nesta parte do trabalho são apresentados alguns conceitos relacionados ao tema. Devido a grande utilização de terminologias em inglês, nas publicações e artigos acadêmicos relacionados ao tema, optamos por manter esse padrão no trabalho.

2.1 Redes Neurais Artificiais

Há mais de 80 anos, o neurocientista Warren Mcculloch e o profissional de lógica computacional Walter Pitts, se juntaram para o desenvolvimento do primeiro modelo neural palpável, baseado no funcionamento do cérebro humano composto pelos neurônios e uma rede unificada de transmissões entre eles (MCCULLOCH; PITTS, 1990).

A parte do neurônio responsável pela transmissão de um sinal, ou impulso, para outros na rede é chamada axônio. Devido a essa função, o axônio consegue controlar, mediante a informação recebida, sua ativação.

Figura 1 - Representação biológica de um neurônio



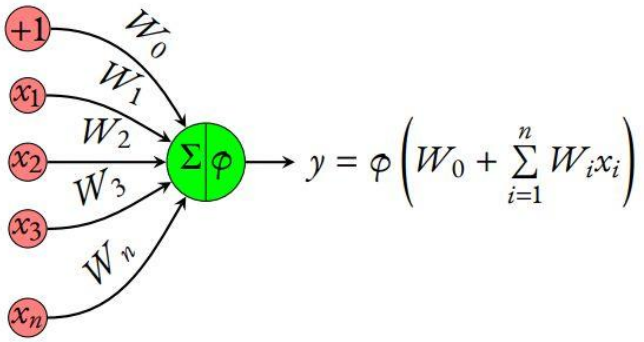
a. Dendrite b. Soma c. Núcleo d. Axónio e. Bainha de mielina
f. Célula de Schwann g. Nódulo de Ranvier h. Axónio terminal

Fonte: (MOREIRA, 2013)

As Redes Neurais Artificiais, do inglês *Artificial Neural Networks* (*ANNs*), também são baseadas em neurônios, denominados *perceptrons* caracterizados como modelos segmentados matematicamente que também ativam a determinados

estímulos conforme as informações analisadas (SAZLI, 2006). Um exemplo de representação ao modelo do *perceptron,* ou neurônio artificial, pode ser observado na figura abaixo:

Figura 2 - Representação de um *perceptron*



Fonte: (VICENTE, 2017)

A propagação de estímulos a partir de um perceptron segue um modelo bem definido chamado *feed-forward*. Nele ocorre o recebimento de uma ou mais entradas, que representam dados quantificados, e a resolução da função que decide se o resultado de saída é passível de ativação (VICENTE, 2017).

Relacionado a cada entrada x_j existe um multiplicador denominado peso, que é um elemento otimizador da rede. O somatório do produto de x_j com seu peso, w_j, correspondente, somado a um viés, no inglês denominado bias, *b* com resultado posteriormente adicionado a função de ativação do perceptron resulta em uma saída de valor binário. A representação matemática do perceptron é definida como (CERRI, 2020):

$$f(b + \sum_{j} w_j . x_j)$$

Devido à saída binária, os *perceptrons* atuam com o aprendizado de funções lineares. Caso o problema ou análise a ser feita seja baseada em até dois conjuntos, possivelmente não serão encontradas limitações estruturais, porém, essa característica pode ser um fator restritivo quando se tratam múltiplas classes de

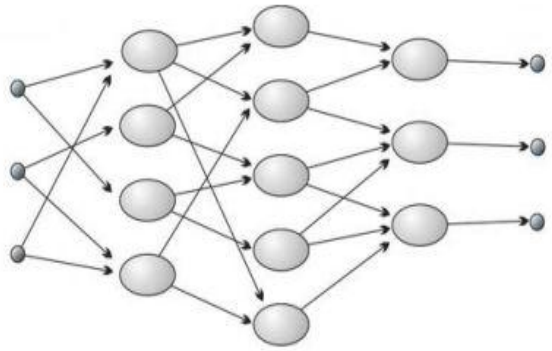
dados, visto que uma reta conta com a divisão em duas áreas bem definidas no plano cartesiano.

Em contrapartida, o psicólogo cognitivo e cientista da computação Geoffrey Hinton em 1986 apresentou uma reformulação na forma de aprendizado das redes utilizando arquitetura de múltiplas camadas de *perceptrons*, do inglês *multilayer perceptron (MLP)*. A arquitetura acompanhada do algoritmo de *backpropagation*, que teve contribuição do cientista, ajusta os pesos das entradas criando uma rede capaz de resolver problemas utilizando funções não lineares (HINTON, 1986).

2.2 Processo de aprendizado das redes neurais

As *MLPs* são categorizadas em dois grandes tipos baseados na conexão entre os neurônios: redes neurais *feed-forward* e redes neurais recorrentes. Caso não exista um *feedback*, ou seja, um *loop* de resposta de um neurônio a sua entrada ou entradas de outros, a rede é classificada como *feed-forward*, do contrário, é uma rede neural recorrente. Nas redes neurais *feed-forward,* foco de estudo nesta sessão, a propagação segue o fluxo das camadas de entrada até as de saída, onde os resultados de saída de cada camada não são aplicados novamente nessa mesma camada. Essa categoria de *ANN* é recomendada para o reconhecimento de padrões, principalmente problemas de classificação (SAZLI, 2006).

Figura 3 - Rede neural *feed-forward* camadas ocultas



Fonte: Adaptado de (FERNEDA, 2006)

Redes neurais são uma representação matemática do modelo neural orgânico humano, assim sendo, também possuem sua forma particular de treino e consequentemente aprendizado através de algoritmos, parâmetros e funções.

O processo de aprendizado é realizado em ciclos de treinamento denominados épocas. A rede usa dados de treino para adquirir conhecimento e é por meio das épocas que fica definida a quantidade de vezes que esse conjunto vai ser repassado, obedecendo um critério de parada baseado, por exemplo, na taxa de acurácia ou taxa de perda.

Classificação e regressão são os principais problemas resolvidos por *ANNs*. Os algoritmos desenvolvidos buscam com base em um conjunto de dados de entrada $\{x_n\}$ e um vetor y de resultados y= (y1;y2;...;yn) a predição de um valor *y* dado um novo valor de *x* qualquer. A diferença entre classificação e regressão fica no retorno de y*,* no primeiro caso o valor é discreto, no segundo, progressivo (NIE; HU; LI, 2018).

2.2.1 *Forward Propagation*

Forward Propagation é o movimento de uma rede caracterizado pela propagação dos valores da esquerda para direita, das camadas iniciais para as de saída. Esse movimento é diretamente associado ao modelo *feed-forward* dos *perceptrons*, unidades básicas da rede. Nessa etapa, os pesos, parâmetros multiplicados com o respectivo valor de entrada em um *perceptron*, determinam a importância de um segmento da rede frente a outros para melhor resolutividade (SAZLI, 2006).

2.2.2 Função de perda

O desempenho dos algoritmos de *machine learning* são avaliados a partir da sua função de resolver o problema de otimização da rede, *structural risk minimization (SRM).* Sua fórmula é representada abaixo:

$$\min_f \frac{1}{N} \sum_{i=1}^{N} L_\theta(f(x_i)) + \lambda R(f)$$

Sendo *N* o conjunto de dados de treinamento, *L* a função de perda que compõe parte do primeiro termo, representando o risco empírico, e o segundo termo,

onde *R(f)* configura a complexidade do modelo e λ o balanceador entre o risco empírico e a complexidade do modelo (WANG *et al.*, 2020).

Como observado, a função de perda, ou função de custo, é um dos componentes principais do risco estrutural. É responsável por calcular o quanto o valor de saída da rede está próximo ao valor real esperado, determinando um sinal de erro. A escolha de uma função de perda é alinhada ao objetivo proposto, ou seja, para melhorar a efetividade do algoritmo é necessária uma escolha assertiva da mesma.

2.2.3 *Backpropagation* e Gradiente descendente

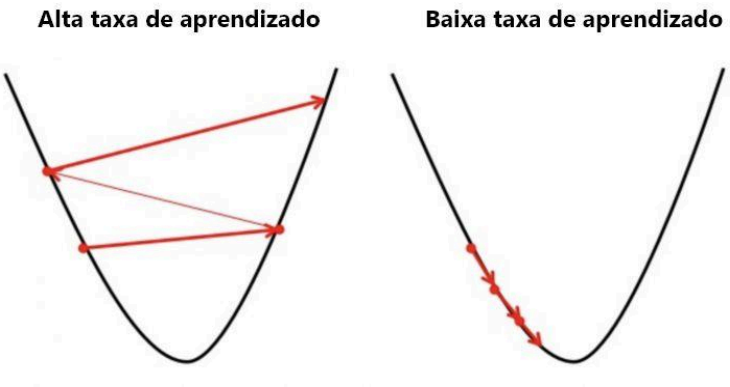
Conforme citado, em 1986 o cientista Geoffrey Hinton revolucionou o estudo das redes neurais a partir da utilização da arquitetura *MLP* combinada com algoritmo de aprendizagem *backpropagation* para tornar a rede capaz de lidar com problemas de forma não linear.

O algoritmo é empregado na etapa final de um ciclo de treinamento onde ocorre a atualização dos pesos em toda extensão da rede a partir da transmissão do erro previamente calculado em direção oposta, da camada de saída para as camadas intermediárias, com o intuito de equiparar ao máximo o valor de saída com o valor desejado ou até um limiar médio (BARCA; SILVEIRA; MAGINI, 2005).

Para buscar o menor valor de perda na função de custo e consequentemente otimizar a rede, o algoritmo de *backpropagation* utiliza uma técnica de minimização conhecida como gradiente descendente (PAPAGELIS; KIM, 2022). Sua base é a busca pelo mínimo global, ou seja, o menor valor que a função de perda pode proporcionar para aquele modelo.

O modelo neural conta com um parâmetro denominado taxa de aprendizado, que determina a velocidade em que a rede alcança uma adequada otimização dos pesos. O gradiente descendente se relaciona com a taxa de aprendizagem de modo a aumentar ou diminuir o tempo de busca pelo mínimo global. Quanto menor a taxa de aprendizado, o gradiente descendente consequentemente levará mais tempo para alcançar o mínimo; quanto maior a taxa, o gradiente atinge um mínimo rapidamente, porém considerando uma confiabilidade inferior (BROWNLEE, 2019).

Figura 4 - Associação entre a taxa de aprendizado e o mínimo global



Fonte: Adaptado de (DONGES, 2021)

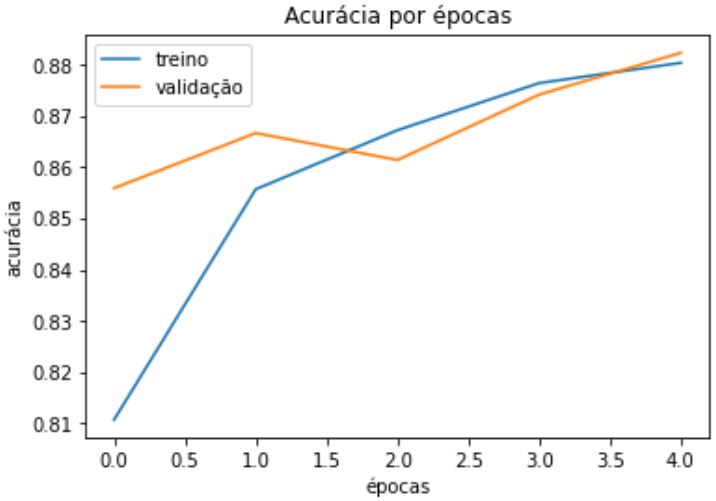
Outro parâmetro do modelo neural utilizado para otimização é o *momentum*, também associado ao gradiente descendente, trabalha em conjunto com a taxa de aprendizado considerando o foco do algoritmo de gradiente pela busca do mínimo global, o *momentum* auxilia a rede a evitar mínimos locais, aumentando a velocidade do aprendizado (SMITH, 2018).

2.2.4 *Overfitting* e *underfitting*

Ao final do treinamento de uma *ANN* é necessário validar o quão bom se tornou o modelo na resolução do problema proposto por meio da sua capacidade de generalização, isto é, a taxa de acerto nas previsões. Uma das principais métricas é verificar a acurácia ou a taxa de acerto da rede, sendo essa a mais básica e insuficiente em determinados casos, que serão apresentados posteriormente.

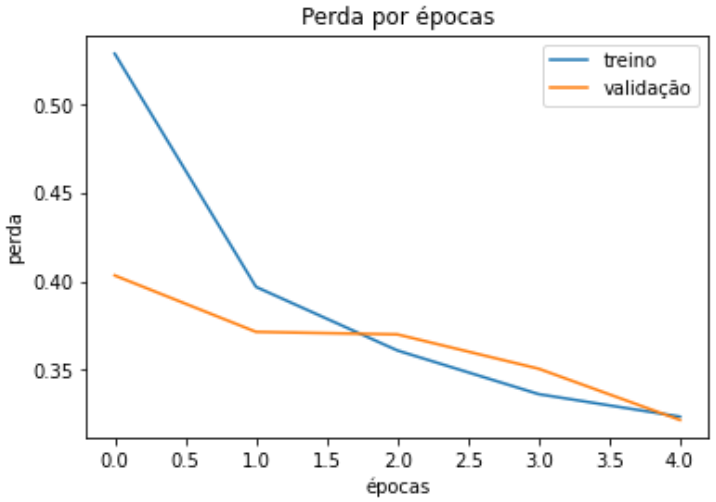
Os processos de criação e desempenho da rede são compreendidos entre os conjuntos de dados de treino e teste respectivamente. Para metrificar a qualidade da mesma, existe a possibilidade da criação de um terceiro conjunto, geralmente extraído dos dados já disponíveis, chamado validação. Dispondo de uma imparcialidade em relação ao treinamento, o conjunto de validação é avaliado de forma semelhante ao conjunto de treino, contendo medidas de perda e acurácia. O comparativo entre essas duas medidas nos dois conjuntos durante as épocas, serve como métrica de validação do modelo geral (BROWNLEE, 2019).

Figura 5 - Exemplo comparativo da taxa de acurácia nos conjuntos de treino e validação



Fonte: Elaborada pelos autores

Figura 6 - Exemplo comparativo da taxa de perda nos conjuntos de treino e validação



Fonte: Elaborada pelos autores

Dos problemas comuns na análise de *ANNs* após o treinamento, observáveis facilmente na comparação entre as taxas nos conjuntos de treino e validação, destacasse o *overfitting* e o *underfitting*, sendo o primeiro mais recorrente. No *overfitting*, a rede neural a partir de um período não evolui a capacidade de resolver o problema, ao invés disso, passa a decorar padrões de treinamento, criando um vício nos resultados. No underfitting, o modelo se torna incapaz de aprender corretamente sobre a variedade dos dados de treino (JABBAR; KHAN, 2014)

Para evitar o *overfitting* a rede precisa ser regularizada, suavizando as previsões para uma melhor generalização, evitando complexidade no modelo. Um dos métodos disponíveis é o *early stopping*, onde o treinamento é interrompido no momento em que o erro de validação sinaliza uma ascensão (SARLE, 1995).

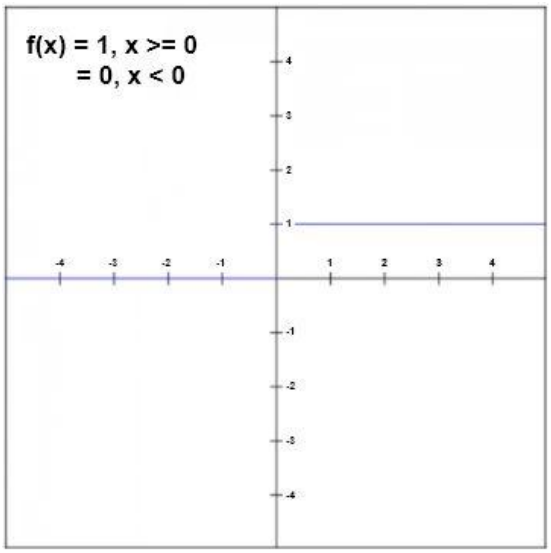
2.3 Função de ativação

Função de ativação é a parte final de um *perceptron*. A partir dela um sinal de entrada vai ser calculado em um sinal de saída para a próxima camada da rede e assim consecutivamente. Todo o cálculo é feito conforme a soma dos produtos das entradas e seus respectivos pesos aplicados a uma função de ativação. A acurácia e, principalmente o aumento no poder de resolução de problemas complexos pela rede, estão ligadas a escolha e uso dessa função. (SHARMA SIDDHARTH; SHARMA SIMONE, ATHAIYA, 2020).

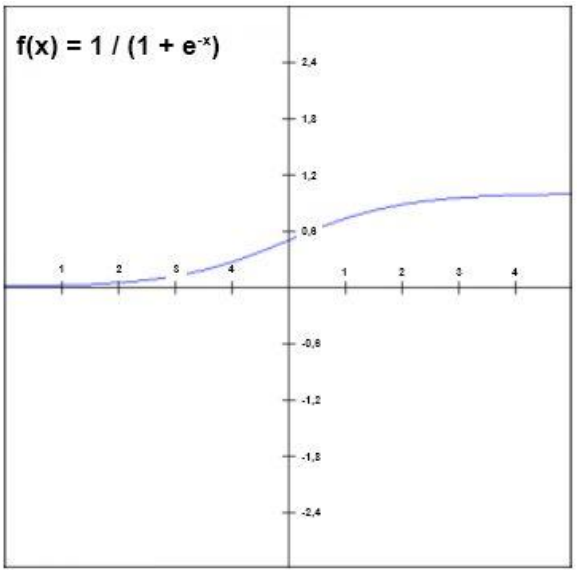
Esse cálculo realizado no *perceptron* sem uma função de ativação é capaz de gerar um valor de saída simples, representado graficamente por uma função linear, que é um polinômio de grau um. Nesse ponto existe a relação entre simplicidade e complexidade, a função linear é simples, mas se limita a resolver problemas de complexidade baixa. Para tarefas com dados na ordem de imagens, vídeos, áudio, etc. a aplicabilidade da função de ativação não linear, pois existem funções de ativação lineares, se mostra necessária (SHARMA SIDDHARTH; SHARMA SIMONE, ATHAIYA, 2020).

Figura 7 - Funções de ativação comumente utilizadas

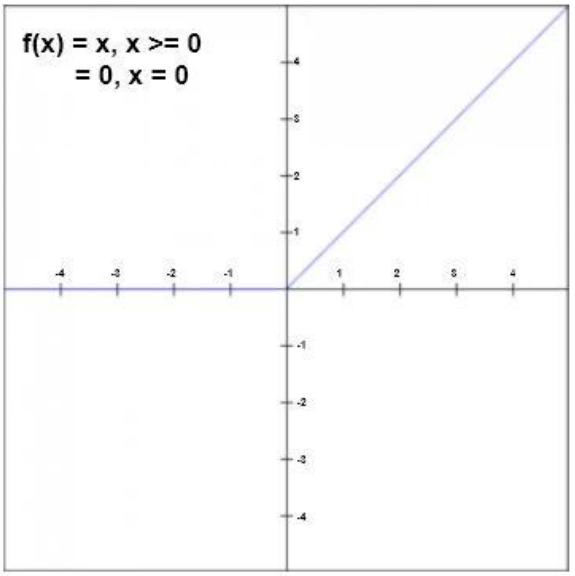
(a) Função de Etapa Binária



(b) Função Sigmóide



(c) ReLu



fonte: Adaptado de (SHARMA SIDDHARTH; SHARMA SIMONE, ATHAIYA, 2020)

2.4 Parâmetros e hiperparâmetros

Utilizados durante a construção e treino do modelo, os parâmetros são ferramentas dinâmicas atuantes em diversos processos de otimização. Os parâmetros internos da rede trabalham com os dados durante o treinamento, por exemplo, os pesos, e não sofrem interferência manual por estarem ligados a esse processo. Os hiperparâmetros diferem na área de configuração, pois são definidos

antes da etapa de treino, sendo usados como um conjunto de valores ligados à performance dos dados nessa etapa (WU *et al.*, 2019).

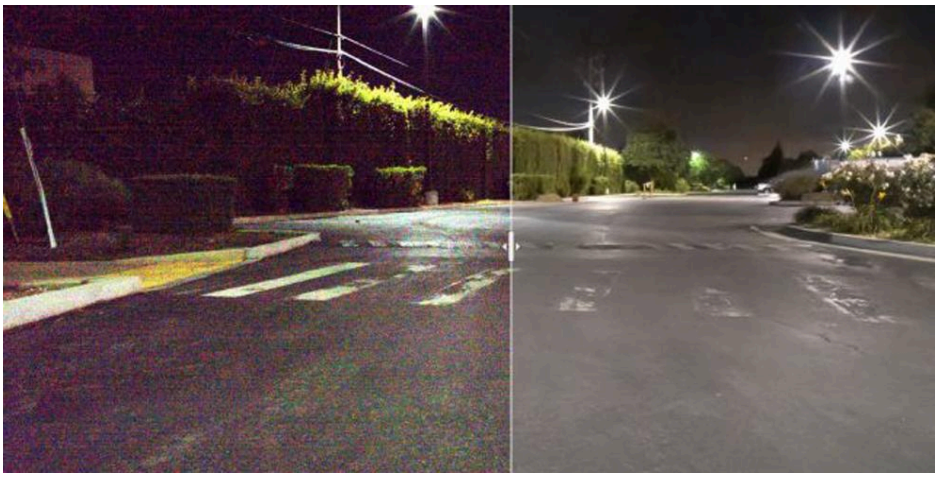
2.5 *Deep Learning*

Devido ao grande volume de dados gerados atualmente, conhecido como a era *Big Data*, os problemas relacionados à análise e interpretação desse conjunto massivo de informação produziu novos desafios às áreas de inteligência artificial e *machine learning*. O *deep learning* foi criado com o intuito de resolver esse problema utilizando técnicas de *machine learning* aliadas a diversas camadas de processamento (EMMERT-STREIB, 2020 *apud* HINTON *et al.*, 2006).

Uma das principais áreas de aplicabilidade da arquitetura de *deep learning* é a visão computacional, a emulação da visão humana destinada à busca de informações no todo ou parcialmente em uma imagem passa por interpretação. Os processos nesse campo de estudo podem ser divididos em 3 partes (MARENGONI; STRINGHINI, 2009):

- Processos de baixo-nível que lidam com melhorias estéticas aplicadas nas imagens.

Figura 8 - Imagem com melhorias de ruído



fonte: (BELIN, 2018)

- Processos de médio-nível operando em segmentação (divisão das imagens em regiões menores observáveis) ou classificação de imagens (reconhecimento de objetos na imagem).

Figura 9 - Processo de classificação de imagem

fonte: Elaborada pelos autores

- Processos de alto-nível são associados a tarefas de cognição relacionados à visão humana.

Redes neurais convolucionais são redes *feed-forward* aliadas ao *deep learning*, amplamente usadas em tarefas de reconhecimento como apuração de documentos, escrita e principalmente classificação de imagens. A arquitetura das *CNNs* trabalha com a técnica de convolução que acrescenta duas novas representações de camadas e diminuição da parametrização da rede (LIU *et al.*, 2018).

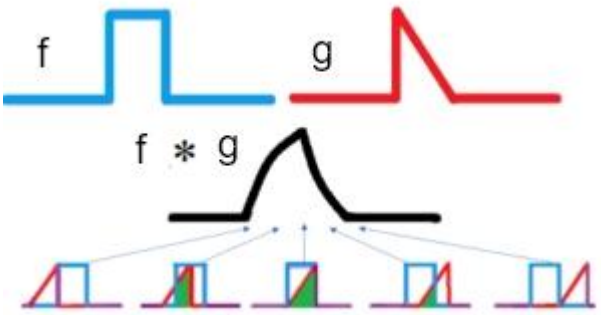
2.6 Convolução

Matematicamente, a convolução é um operador linear que realiza o somatório entre duas funções, *f(x)* e *g(x)*, resultando em uma terceira, *h(x)*, que expressa o formato de *f(x)* após sofrer a aplicação de *g(x)*, a fórmula da convolução é representada abaixo como:

$$h(x) = (f * g)(x) = \sum_{u=0}^{U} f(u)g(x-u)$$

Desse modo, *g(x)* é deslocada pela área de *f(x)* de forma invertida, nessa sobreposição é criada a região de resultado compreendida a *h(x)*. Pode-se entender a primeira função como um dado e a segunda, um filtro, conhecido como *kernel*. No *deep learning* o resultado da convolução é um mapa de características do dado analisado.

Figura 10 - Representação gráfica da convolução

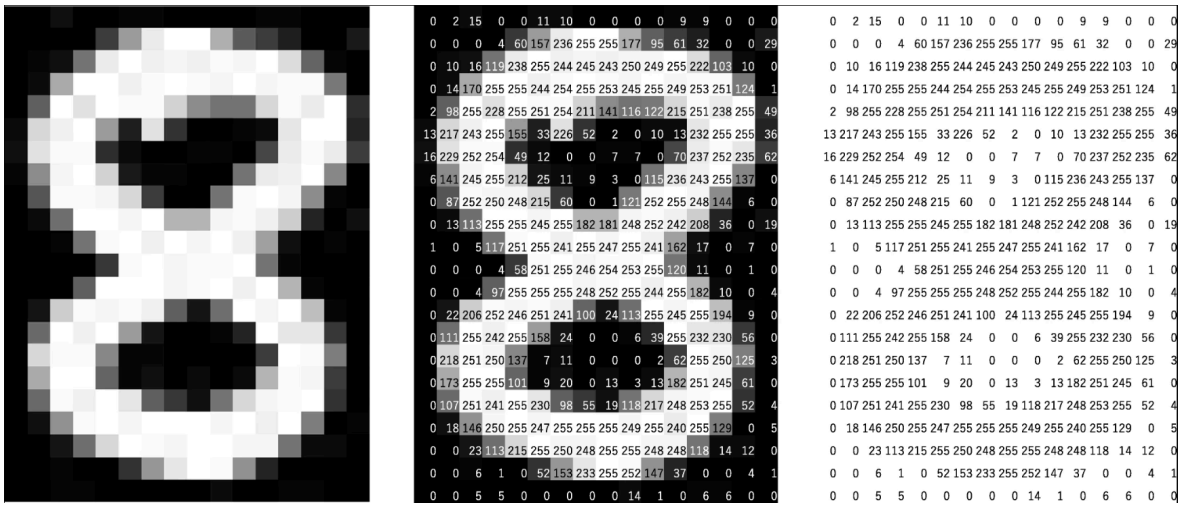


fonte: Adaptado de (GLEN, 2020)

2.7 Redes Neurais Convolucionais

O algoritmo mais usado na área de *deep learning* é o modelo de rede neural convolucional (*CNN*), dominante nas tarefas de visão computacional e processamento de linguagem natural. Redes convolucionais são uma variação das redes *MLP* capazes de lidar com grandes volumes de dados, as *CNNs* se diferem pelo uso de técnicas aplicadas em camadas especiais, sobretudo a convolução.

Figura 11 - Imagem representada matricialmente



fonte: (YAMASHITA *et al.*, 2018)

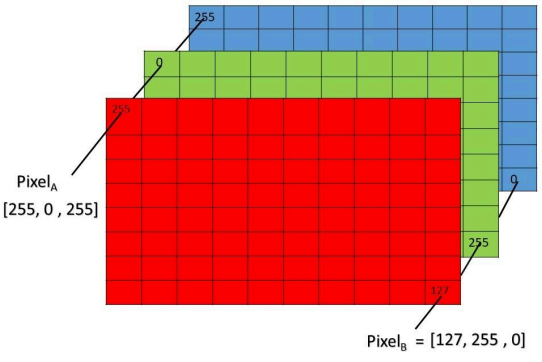
CNNs processam dados no formato de matriz, aprendendo e se adaptando, transformando-os em características divididas hierarquicamente em níveis. Os níveis inferiores são definidos por características que contemplam informações difusas sobre o objeto, o prosseguimento para os níveis intermediários e consequentemente

os finais, especificam cada vez mais esses atributos durante o processo melhorando a identificação de padrões pela rede (YAMASHITA *et al.*, 2018).

A vantagem no uso de uma *CNN* é a redução da complexidade computacional diante de *ANNs* tradicionais. Usando o processamento de imagem como exemplo (durante os próximos tópicos o foco vai ser no trabalho com imagens), a entrada é formada a partir das 3 dimensões da imagem, sendo elas largura, altura e o canal de cor. Uma imagem de 64 x 64 x 3 (canal de cor colorido) faz com que a primeira camada da rede tenha 12.288 pesos, parâmetros internos do modelo. O *overfitting* pode ocorrer devido à quantidade desses parâmetros de treino afetando a otimização da rede caso a arquitetura tradicional seja usada, além dos recursos computacionais e de tempo gerados pela quantidade de operações com valores dessa escala (O'SHEA; NASH, 2015).

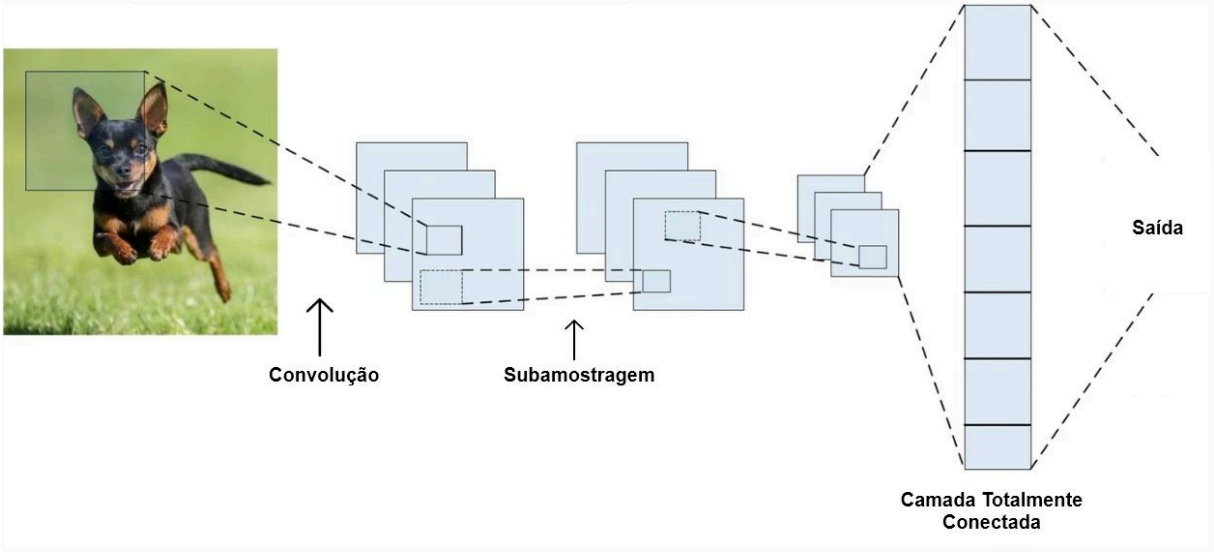
Imagens utilizadas nos dados de entrada de uma rede neural possuem a particularidade da presença ou ausência de cor. Como discutido, a imagem é uma matriz tridimensional onde o último elemento identifica se a imagem possui coloração: o canal de cor. Valores *m* x *n* x 1 representam imagens em preto e branco, com 1 canal de ativação para cor: preto ou branco. Para *m* x *n* x 3 o espectro *RGB*, do inglês *red, green, blue color model*, determina as cores de um *pixel* a partir das cores primárias em um intervalo [0,255].

Figura 12 - Formação de *pixel* em uma imagem *RGB*



fonte: Adaptado de (IHRITIK, 2018)

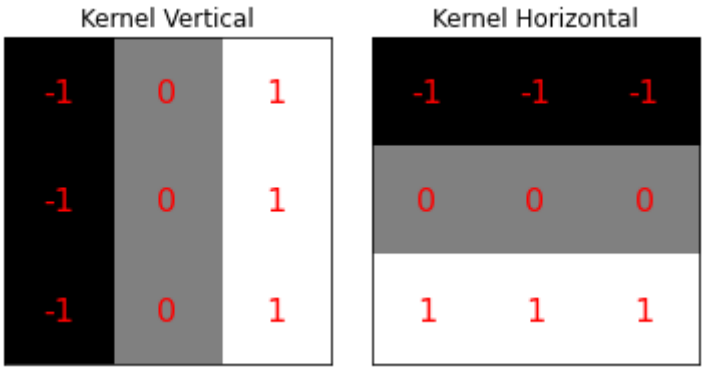
Figura 13 - Arquitetura de uma *CNN*



fonte: Adaptado de (ALZUBAIDI *et al.*, 2021)

Redes neurais convolutivas são semelhantes às *MLPs*, divididas em camadas. As *CNNs* apresentam camadas iniciais que realizam a elaboração do mapa de características e fazem o processo de subamostragem: camadas convolutivas e de *pooling*, até as finais: *fully connected*. A distribuição dessas camadas consiste na sequência de convoluções e *poolings* seguidas por *n* camadas conectadas (ALZUBAIDI *et al.*, 2021). Um exemplo dessa arquitetura é demonstrado na figura 13.

Figura 14 - Exemplos de *kernels*

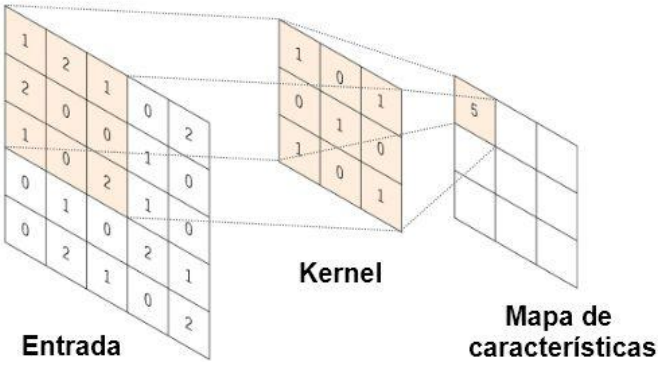


fonte: Elaborada pelos autores

A camada de convolução é onde atuam os já citados *kernels*, filtros convolucionais que formam o mapa de características. O dado de entrada sofre a operação de convolução a partir da aplicação de um ou mais *kernels* invertidos.

Esses filtros têm um padrão de *grid* com medidas pré-definidas ajustadas conforme o tamanho de encaixe na entrada (ALZUBAIDI *et al.*, 2021).

Figura 15 - Operação de convolução

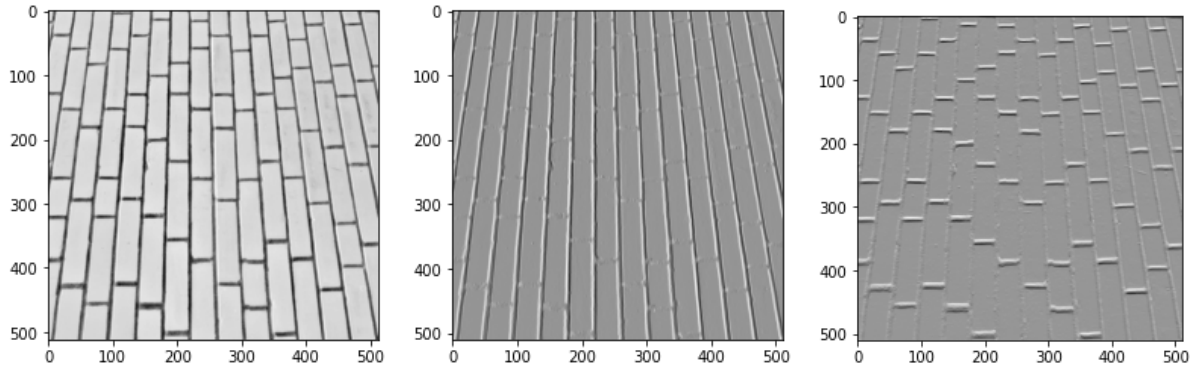


fonte: Adaptado de (YAMASHITA *et al.*, 2018)

O mapa de características gerado após a operação de convolução é menor que o dado de entrada, possuindo pontos que demonstram o nível de ativação para um determinado conjunto de *pixels*.

O *kernel*, ou um conjunto deles, pode ser especializado em buscar características específicas da imagem, como, por exemplo, bordas horizontais, verticais, ou formas complexas da imagem.

Figura 16 - Exemplos de mapas de características conforme a especialidade do *kernel* (vertical e horizontal)



fonte: Elaborada pelo autor

A camada de *pooling* realiza a subamostragem, redução da representação dimensional espacial da imagem com a manutenção de informações relevantes. Esse processo auxilia na melhor generalização da rede já que reduz o número de parâmetros internos e consequentemente a complexidade do modelo. O cálculo

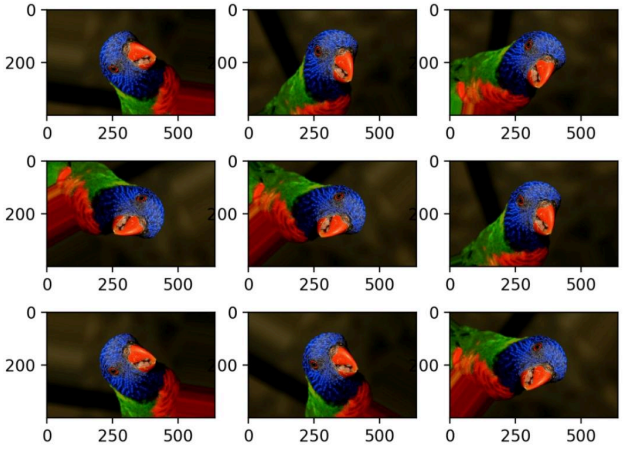
dessa camada é realizado na simples escolha do maior *pixel* de ativação advindo do mapa de características (O'SHEA; NASH, 2015).

O mapa de características da última camada de *pooling* da rede é um dado de entrada no formato esperado (*array* de uma dimensão) pela última camada geral da rede, a camada totalmente conectada (YAMASHITA *et al.*, 2018). Nela acontecem os procedimentos comuns das redes neurais e por fim a classificação esperada.

2.8 *Data augmentation*

Para a construção de bons modelos de *deep learning*, a otimização da relação entre o erro de validação e treino deve ser bem balanceada. Esse ponto tem ligação direta com a quantidade e qualidade do conjunto de dados usado no processo de treino da rede e evita o *overfitting*. Para alcançar bons níveis de generalização, a rede precisa consumir um número expressivo de dados, o suficiente para aprender bem. Uma das técnicas para auxiliar nesse problema é o *data augmentation*, utilizado para criar dados transformados a partir dos existentes no *dataset* (MIKOŁAJCZYK; GROCHOWSKI, 2018).

Figura 17 - Exemplo de *image data augmentation*



fonte: (BROWNLEE, 2019)

No caso de imagens a transformação acontece no posicionamento, sentido, coloração, tamanho etc. A eficácia da técnica pode ser observada em simples modificações como inversão horizontal, aumento de espaço de cores e corte aleatório (CLARO *et al.*, 2020).

3 METODOLOGIA

O trabalho apresentado segue a metodologia qualitativa, que, de modo geral na área da computação, se caracteriza como o estudo de um sistema no ambiente que ele está sendo utilizado. Nesse caso, de forma ativa (pesquisa-ação), procurasse, por meio do desenvolvimento e implementação de um sistema, a futura análise e avaliação dos resultados do que foi posto em prática (WAINER, J., 2007).

Competem aos resultados desse trabalho a apresentação das análises realizadas após o desenvolvimento, treino e otimização das *CNNs*. Para a obtenção dos mesmos, foram utilizadas as técnicas e métodos descritos a seguir.

3.1 Ferramentas utilizadas

Para a construção das redes neurais convolucionais e os materiais necessários que as compõem, foram utilizadas neste trabalho ferramentas públicas gratuitas e bibliotecas *open-source*.

- **Kaggle**: É uma plataforma voltada para *machine learning*, contém *datasets* públicos criados pela comunidade e vários outros recursos como fóruns de discussão, competições de *machine learning* etc (KAGGLE, 2022).
- **Keras**: Segundo a própria documentação, o Keras é uma biblioteca *open-source* que auxilia na criação de modelos e seus recursos, focando na velocidade de implementação. É escrita na linguagem Python e utiliza como base a plataforma de *machine learning* Tensorflow (KERAS, 2022).
- **Google Colab**: Plataforma da Google que interpreta e executa Python no navegador, além de permitir a visualização do código entre os usuários do *notebook*. Possui uma vasta gama de bibliotecas de *machine learning* e, para o processamento de modelos, retira a responsabilidade do *hardware* da máquina do usuário para os próprios recursos disponibilizados pela plataforma (COLAB, 2022).
- **Sklearn e Matplotlib**: Bibliotecas tradicionais utilizadas para criação da matriz de confusão e gráficos, respectivamente. O Sklearn neste trabalho foi utilizado somente na criação da matriz de confusão (SKLEARN, 2022; MATPLOTLIB, 2022).

3.2 Obtenção e criação do *dataset*

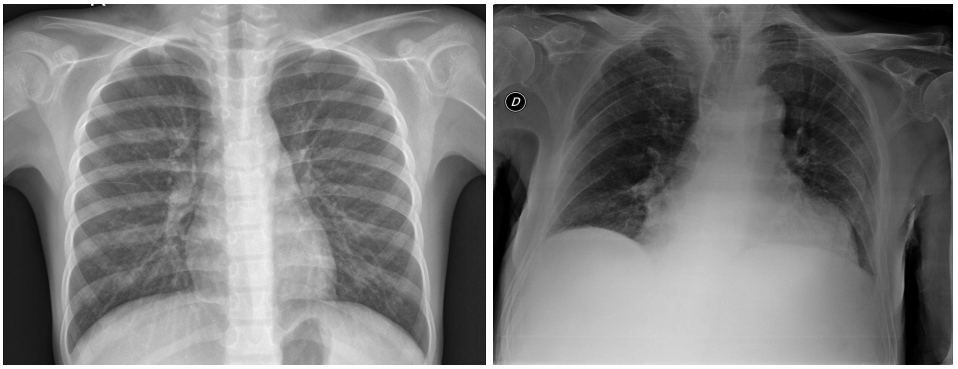
O banco de imagens de raio X foi obtido a partir de bases de dados públicas contendo os resultados de pacientes com COVID-19, pneumonia e tuberculose. Além disso, coletaram-se imagens de pulmões saudáveis que também compõem parte importante no aprendizado das *CNNs*.

O conjunto de imagens foi retirado do Kaggle utilizando o *dataset* ***Chest X-Ray (Pneumonia, Covid-19, Tuberculosis)***. No total, foram selecionadas 7.906 imagens, que estão divididas em 4 classes: *COVID19*, *NORMAL*, *PNEUMONIA*, *TURBERCULOSIS*, atualmente o *dataset* apresenta um total de 7.135 imagens.

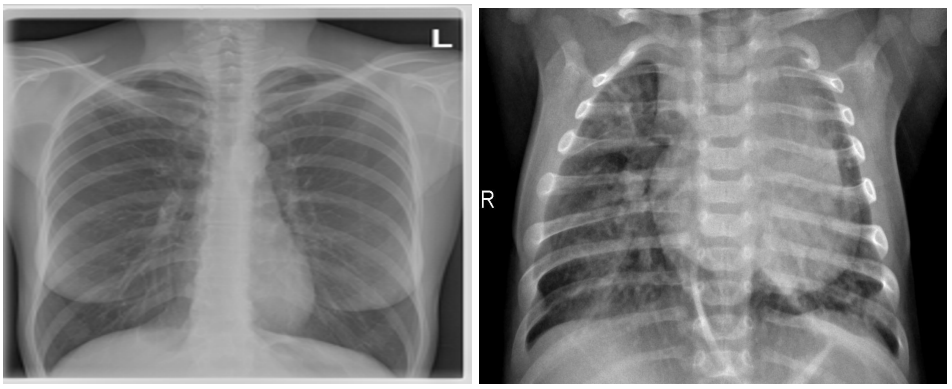
Foram selecionadas 7.097 imagens para treino, 771 para validação e 38 para teste. Representando uma porcentagem nos subgrupos em cerca de 90% para treino e 10% para auxiliar na validação.

Figura 18 - Imagens classificadas por *labels*

(a) Normal (b) COVID-19



(c) Tuberculose (d) Pneumonia



Fonte: Elaborada pelos autores

3.3 *ImageDataGenerator*

Para se trabalhar com a criação de modelos de *deep learning* é essencial uma quantidade razoável de dados de entrada para o treinamento, além dos outros conjuntos de dados utilizados durante a construção e validação do modelo. O tamanho do *database* com 7.097 imagens é considerado baixo para os padrões.

A técnica de *data augmentation* se mostra útil nesses cenários, a implementação dessa técnica foi possível através da biblioteca Keras utilizando a classe nativa *ImageDataGenerator*, que fornece uma maneira rápida e fácil de transformar imagens. A classe consegue realizar o aumento da imagem, padronização, rotação, deslocamentos, inversões, mudança de brilho etc.

O principal benefício da *ImageDataGenerator* é sua construção para fornecer aumento de dados em tempo real, ou seja, as novas imagens transformadas são geradas em tempo de execução, enquanto seu modelo ainda está em estágio de treinamento. Além disso, ela requer menor uso de memória ao carregar as novas imagens a cada época percorrida, ao contrário do que normalmente acontece, o carregamento de todas as imagens durante as épocas.

A *ImageDataGenerator* também possui a particularidade de receber a imagem original, transformá-la e retornar somente o conjunto novo transformado. Esse formato de *augmentation*, aliado ao seu processamento em tempo de treino, implica em uma variação maior de dados diferentes a cada época.

Ao utilizar o processo de *data augmentation* em imagens anatômicas, como no caso da radiografia, deve-se ter o cuidado em transformações que possam alterar a estrutura da parte analisada. No caso dos pulmões, imagens espelhadas podem prejudicar a qualidade do modelo por inverterem o pulmão direito e esquerdo, causando uma diferente visualização da estrutura completa.

Devido a isso, as imagens neste trabalho foram transformadas de maneira a evitar o espelhamento. Para as imagens de treino, os parâmetros passados a classe *ImageDataGenerator* foram os seguintes:

- ***Rescale***: Para normalização e melhora do processamento, os valores dos *pixels* em um intervalo [0,1], as imagens passam por uma refatoração na escala 1/255;

- ***Rotation_range***: Rotação em graus, valor utilizado foi 20. A leve rotação não altera a visualização completa da imagem como um espelhamento faria;
- ***Width_shift_range****:* Reposiciona a imagem no eixo horizontal;
- **Height_shift_range**: Reposiciona a imagem no eixo vertical;
- **Shear_range**: Alteração na forma angular da imagem, melhorando a visualização da mesma em pontos diferentes.

No *dataset* de validação e teste foi utilizado somente o parâmetro de *rescale*, aplicando a mesma escala de refatoração. Um dos modelos desenvolvidos neste trabalho não apresenta os métodos específicos de *data augmentation*, fazendo uso somente do *rescale* para normalização.

3.4 Definição de hiperparâmetros

Os hiperparâmetros comuns atuantes nos modelos foram escolhidos focando no melhor processamento do modelo durante o treinamento. O número de épocas definido para cada teste procura o balanceamento entre o uso de *hardware*, plataforma Google Colab, e o tempo de treino. Esse foi o único elemento utilizado como critério de parada, sem nenhum procedimento de *early stopping*.

O segundo conjunto de hiperparâmetros lida com fluxo de imagens propagados por época para cada *dataset* envolvido nessa etapa (treino e validação). A técnica realiza o cálculo entre o número de imagens totais e o *batch size* pré-definido para determinar a quantidade de *steps* tomados em cada interação, a fórmula pode ser observada a seguir:

$$N_i = \frac{D_i}{D_b}$$

Sendo N_i o valor dos *steps*, D_i o número total de imagens no *dataset* e D_b o valor do *batch_size*. Um *step* dita quantas imagens são utilizadas em um intervalo no conjunto geral, por exemplo, em um *dataset* de 2.000 imagens e um *batch size* de valor 10, o total de *steps* será de 200, ou seja, 200 imagens por época.

3.5 Construção das Redes Neurais Convolucionais

Foram construídos três modelos convolucionais diferenciados pelos parâmetros escolhidos na arquitetura. Em um deles, se optou pela não utilização da técnica de *data augmentation* para comparação dos resultados.

Foi escolhido um total igual de épocas para os três testes, com valor de 15 aliado ao já citado cálculo de *steps* nos *datasets* de treino e validação. Os treinos foram executados no Google Colab utilizando uma GPU disponibilizada pela própria plataforma, evitando assim possíveis problemas de processamento caso fossem utilizadas máquinas físicas.

Os parâmetros de compilação foram igualmente replicados para os três modelos. A função de perda escolhida foi a *categorical crossentropy* devido ao número total de classes possíveis ser acima de duas, as 3 categorias de doenças pulmonares com adição dos pulmões saudáveis (KERAS, 2022). O algoritmo de otimização *Adam* fez o papel de substituto para o gradiente descendente tradicional, sendo computacionalmente mais eficiente, consumindo menos memória e servindo bem para problemas de *deep learning* (KINGMA; LEI BA, 2015). A métrica de avaliação do modelo escolhida foi a classe do Keras *accuracy*, que mede a taxa de acerto das previsões.

A arquitetura dos modelos foi definida a partir de uma bateria de testes, tentando alcançar o melhor comparativo do que cada modelo poderia oferecer de acordo com sua configuração. As imagens foram tratadas com o mesmo *input shape* (proporções de entrada), 120 x 120 x 1.

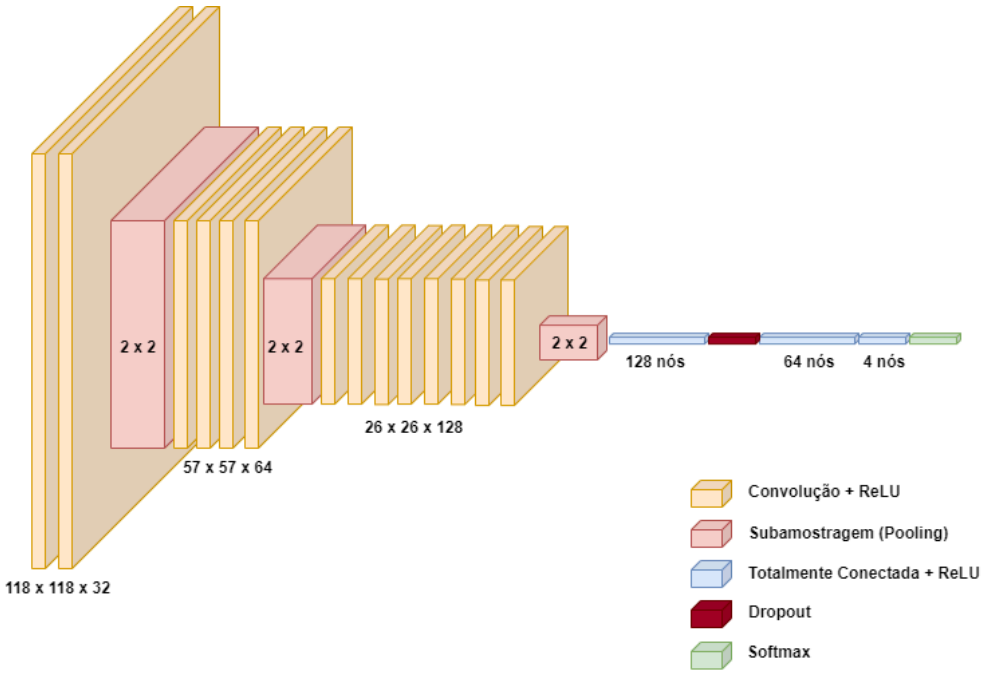
3.5.1 Modelo A

O modelo A foi o primeiro desenvolvido, escolhido como base para alterações realizadas nos outros dois. As imagens processadas por ele passaram pelo processo de *data augmentation*. Sua configuração é descrita abaixo:

- Camada de convolução com 32 filtros de dimensão 3 x 3 e função de ativação *ReLU*;
- Camada de *pooling* com 2 x 2 *pixels*;
- Camada de convolução com 64 filtros de dimensão 3 x 3 e função de ativação *ReLU*;

- Camada de *pooling* com 2 x 2 *pixels*;
- Camada de convolução com 128 filtros de dimensão 3 x 3 e função de ativação *ReLU*;
- Camada de *pooling* com 2 x 2 *pixels*;
- Camada *flatten* (redimensionamento para *array* linear)
- Camada totalmente conectada de 128 nós e função de ativação *ReLU*;
- Camada de *dropout*;
- Camada totalmente conectada de 64 nós e função de ativação *ReLU*;
- Camada totalmente conectada de 4 nós e função de ativação *Softmax*.

Figura 19 - Arquitetura da rede neural convolucional A



Fonte: Elaborada pelos autores

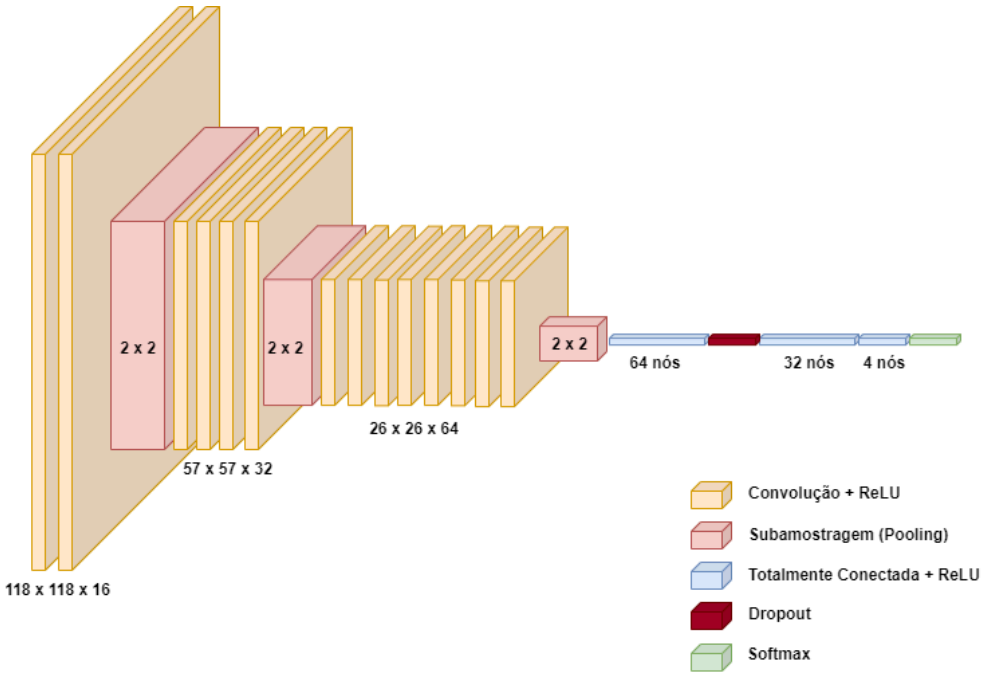
3.5.2 Modelo B

O modelo B foi derivado do A, com alterações na sua arquitetura, especificamente na composição dos filtros convolucionais. As imagens processadas por ele passaram pelo processo de *data augmentation*. Sua configuração é descrita abaixo:

- Camada de convolução com 16 filtros de dimensão 3 x 3 e função de ativação *ReLU*;

- Camada de *pooling* com 2 x 2 *pixels*;
- Camada de convolução com 32 filtros de dimensão 3 x 3 e função de ativação *ReLU*;
- Camada de *pooling* com 2 x 2 *pixels*;
- Camada de convolução com 64 filtros de dimensão 3 x 3 e função de ativação *ReLU*;
- Camada de *pooling* com 2 x 2 *pixels*;
- Camada *flatten* (redimensionamento para *array* linear)
- Camada totalmente conectada de 128 nós e função de ativação *ReLU*;
- Camada de *dropout*;
- Camada totalmente conectada de 64 nós e função de ativação *ReLU*;
- Camada totalmente conectada de 4 nós e função de ativação *Softmax*.

Figura 20 - Arquitetura da rede neural convolucional B



Fonte: Elaborada pelos autores

3.5.3 Modelo C

O modelo C possui a mesma arquitetura do modelo base A, a diferença entre os dois fica no processamento das imagens de entrada. Nesse modelo as imagens não passaram por processos de *data augmentation*, somente a normalização com *rescale*.

3.6 Funções de ativação

As funções de ativação escolhidas para os modelos foram: *ReLU*, do inglês *Rectified Linear Unit*, e *Softmax*. Sendo a *ReLU* utilizada nas camadas convolucionais, de entrada e ocultas da camada totalmente conectada, enquanto a *Softmax* foi usada na camada de saída.

Um neurônio com a função de ativação *ReLU* aceita qualquer valor real de entrada, porém sua ativação só ocorre nos maiores que 0. Valores negativos são convertidos para 0 (CHIMA, 2020). Segue abaixo a representação matemática:

$$R(z) = max(0, z)$$

A *Softmax* é tradicionalmente utilizada para problemas de classificação com múltiplas classes. Ela define probabilidades de um determinado dado ao final pertencer a cada classe esperada, no intervalo de 0 a 1. A partir deste valor final é que ocorre a classificação (CHIMA, 2020).

4 ANÁLISE E RESULTADO DOS DADOS

Esta seção apresenta e discute os resultados apurados pelas *CNNs* implementadas neste trabalho. Dos testes realizados, dois deles foram feitos com o *dataset* aumentado e apenas um sem nenhuma transformação significativa nos dados. Após a finalização, os resultados foram comparados utilizando critérios descritos abaixo. Estes resultados são apresentados nas matrizes de confusão e métricas de classificação.

4.1 Matriz de confusão

Um dos principais métodos de análise de desempenho para algoritmos de classificação é a matriz de confusão. Ela é definida como uma matriz que agrupa as previsões e as classificações reais para determinado conjunto de dados (MARKOULIDAKIS *et al.*, 2021).

A diagonal principal da matriz representa o número de exemplos que receberam a classificação correta, as demais colunas de uma linha representam valores classificados erroneamente e suas respectivas classes.

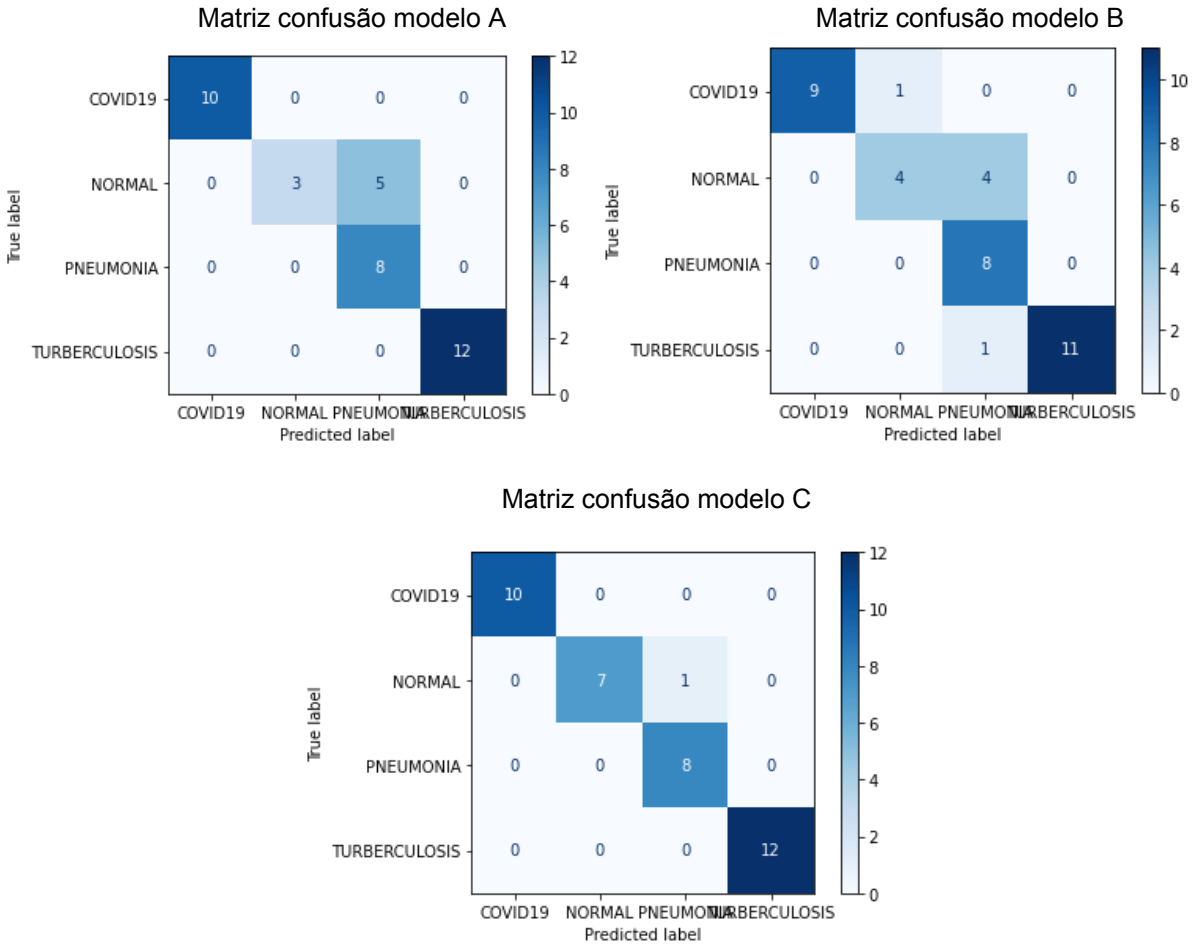
Analisando a matriz de confusão gerada pelo teste do **Modelo A** (figura 21), pode-se perceber que as 10 radiografias da classe *COVID19* utilizadas no teste foram classificadas corretamente. Das radiografias pertencentes a classe *NORMAL,* 3 foram classificadas corretamente e 5 foram erroneamente como *PNEUMONIA.* Já nas radiografias da classe *PNEUMONIA* todas as 8 foram classificadas corretamente e por fim, das radiografias da classe *TURBERCULOSIS* todos os 12 exemplos também foram corretamente classificados.

Analisando a matriz de confusão gerada pelo teste do **Modelo B** (figura 21), pode-se perceber que 9 exemplos da classe *COVID19* utilizadas no teste foram classificados corretamente, enquanto 1 foi classificado como *NORMAL*. Na classe *NORMAL,* 4 foram classificados corretamente e 4 erroneamente, como *PNEUMONIA.* Nesse modelo todos os exemplos da classe *PNEUMONIA* foram classificadas corretamente e por fim, na classe *TURBERCULOSIS,* 11 exemplos foram corretamente classificados, enquanto 1 foi classificado erroneamente, como *PNEUMONIA*.

Por fim, analisando a matriz de confusão do **Teste C** (figura 21), gerada a partir do mesmo modelo do **Teste A**, porém sem a aumentação de dados. É possível ver que das 10 amostras da classe *COVID19* ele acertou todas. Já as amostras da classe *NORMAL*, foram classificadas corretamente 7 delas, e apenas 1 classificada erroneamente como *PNEUMONIA.* Em sequência temos as classes de *PNEUMONIA* e *TURBERCULOSIS,* que tiveram todos as suas amostras classificadas corretamente, com um total respectivamente de 8 e 12 amostras.

Analisando o resultado geral, é possível perceber que na classe *PNEUMONIA* todas as amostras foram classificadas corretamente, enquanto a classe *NORMAL* foi onde houve a maior quantidade de erros. O principal motivo disso é a representatividade de amostras no *dataset* das classes citadas, enquanto a *NORMAL* tem um défice de amostras a *PNEUMONIA* é a classe com a maior quantidade de exemplares.

Figura 21 – Comparativo das matrizes de confusão

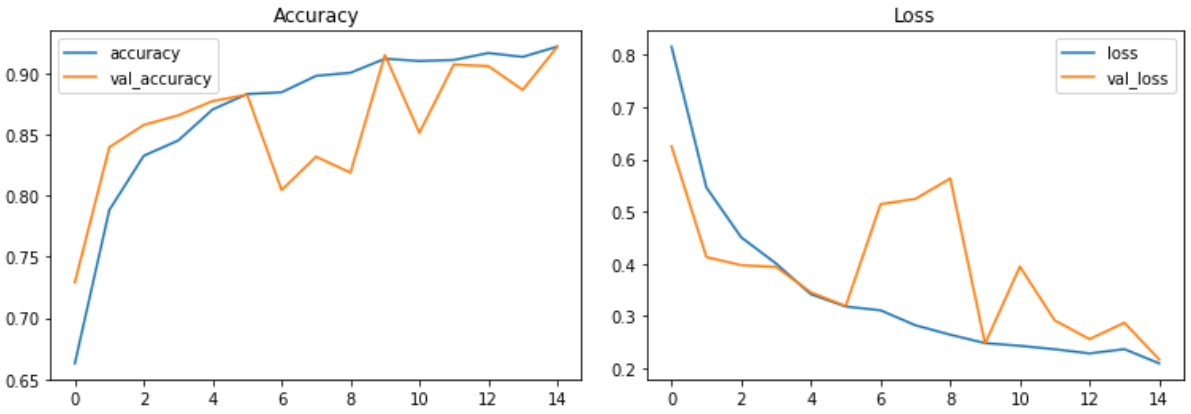


Fonte: Elaborada pelos autores

4.2 Gráficos de acurácia e perda

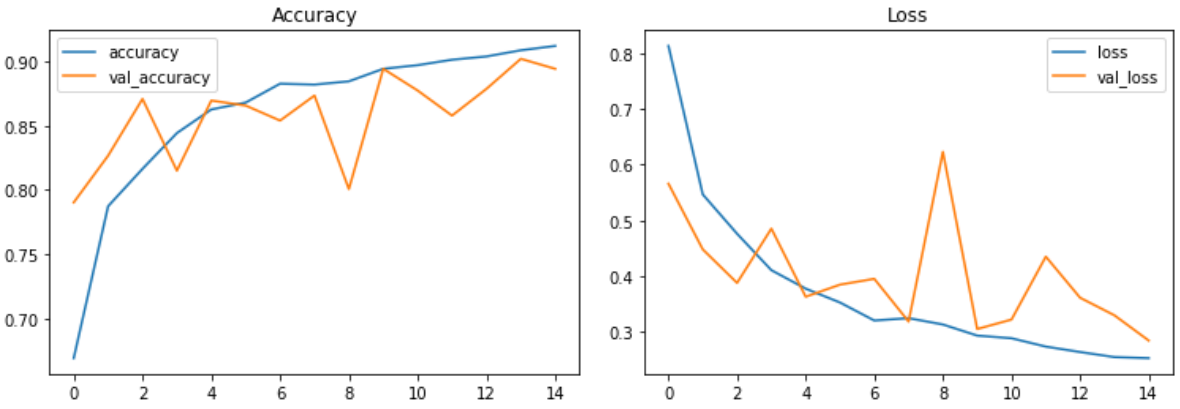
Para os três modelos foi observado um problema na relação entre os *datasets* de treino e validação causado pela inexpressividade dos dados de validação. Segundo Baeldung (2020) e como observado, as curvas possuem a mesma direção, mas o *dataset* de validação apresenta um ruído durante toda a progressão de treino devido à menor quantidade de imagens, a curva referente demonstra que esses dados apresentam baixo nível de informação para confirmar a generalização da rede.

Figura 22 – Gráficos de acurácia e perda do modelo A



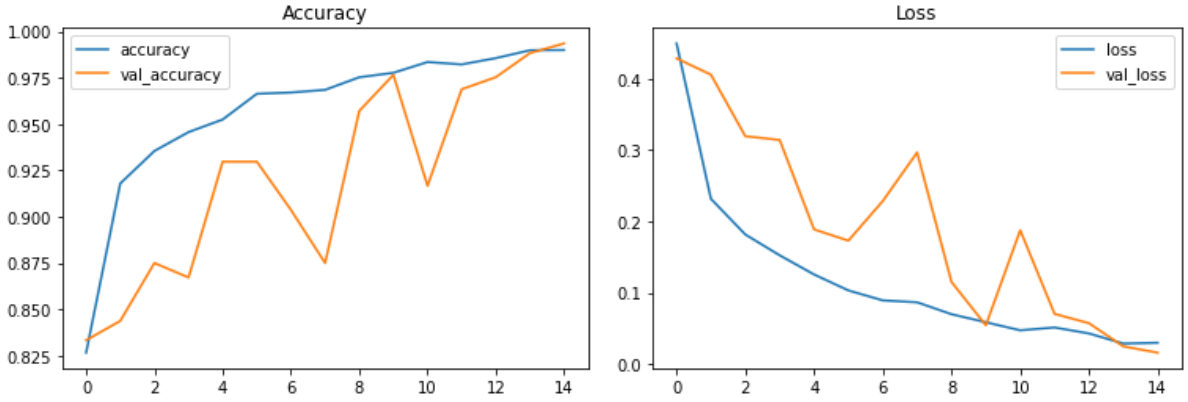
Fonte: Elaborada pelos autores

Figura 23 – Gráficos de acurácia e perda do modelo B



Fonte: Elaborada pelo autor

Figura 24 – Gráficos de acurácia e perda do modelo C



Fonte: Elaborada pelos autores

Nos gráficos apresentados a faixa azul representa o *dataset* de treino, enquanto a amarela representa o *dataset* de validação.

O modelo C, figura 24, tem uma diferença maior em relação ao ruído de progressão por não utilizar o processo de *data augmentation*, somente a normalização das imagens.

4.3 Relatório de classificação

Além da matriz de confusão, um relatório contendo métricas para avaliação do modelo foi gerado, validando os resultados obtidos. A função *classification_report* da biblioteca Sklearn disponibiliza esse relatório com as seguintes métricas: precisão, *recall* e *F1 score*, além da acurácia que também foi avaliada individualmente na sessão seguinte sendo descartada na análise do relatório. Os resultados obtidos para os três modelos podem ser observados nas tabelas 1, 2 e 3 após a explicação de cada métrica segundo Hossin e Sulaiman (2015):

- **Precisão:** Medida que calcula a proporção de classificações positivas que são realmente positivas, por exemplo, utilizando este trabalho, a quantidade de radiografias classificadas a uma classe e fazem parte da mesma.
- ***Recall*:** Medida que calcula a proporção de exemplos positivos de uma determinada classe classificados corretamente, novamente utilizando o exemplo desse trabalho, a quantidade de radiografias da classe *NORMAL* atribuídas a ela.

- ***F1 score*:** Média harmônica entre as medidas de precisão e *recall*.

O relatório também contém uma coluna de suporte que demonstra a quantidade de itens analisados no *dataset* de teste para cada classe.

Tabela 1 - Relatório de métricas do modelo A

Classe	Precisão	Recall	F1 Score	Suporte
COVID19	1.00	1.00	1.00	10
NORMAL	1.00	0.38	0.55	8
PNEUMONIA	0.62	1.00	0.76	8
TUBERCULOSIS	1.00	1.00	1.00	12

Fonte: Elaborada pelos autores

Tabela 2 - Relatório de métricas do modelo B

Classe	Precisão	Recall	F1 Score	Suporte
COVID19	1.00	0.90	0.95	10
NORMAL	0.80	0.5	0.62	8
PNEUMONIA	0.62	1.00	0.76	8
TUBERCULOSIS	1.00	0.92	0.96	12

Fonte: Elaborada pelos autores

Tabela 3 - Relatório de métricas do modelo C

Classe	Precisão	Recall	F1 Score	Suporte
COVID19	1.00	0.90	0.95	10
NORMAL	1.00	0.88	0.93	8
PNEUMONIA	0.89	1.00	0.94	8
TUBERCULOSIS	1.00	1.00	1.00	12

Fonte: Elaborada pelos autores

Analisando a medida de *F1 Score* nas três tabelas, por ela ser uma média das duas outras optou-se pela sua análise para comparação, é possível inferir que o modelo C aparenta ser o mais balanceado.

Os modelos demonstraram boas medidas nas classes de COVID19 e TUBERCULOSIS. Já as classes NORMAL e PNEUMONIA não apresentaram

medidas constantes, mesmo no modelo C. Esses valores podem ter sido influenciados pela menor quantidade de exemplos dessas classes no *dataset* de teste.

4.4 Acurácia dos testes

A acurácia é basicamente a medida que determina a taxa de acerto, ou previsões corretas, de um modelo diante do número total de exemplos avaliados (HOSSIN; SULAIMAN, 2015). Essa métrica reúne todos os valores que podem ser observados na matriz de confusão: verdadeiro positivo, falso positivo, verdadeiro negativo e falso negativo.

Tabela 4 - Acurácia entre os modelos

Modelo	Acurácia (%)
Modelo A	86.84
Modelo B	84.21
Modelo C	97.37

Fonte: Elaborada pelos autores

Por fim, realizando uma análise nas acurácias dos três modelos, é possível notar que o modelo que obteve melhores resultados foi o C.

Os valores de acurácia dos modelos A e B foram bem semelhantes, ambos utilizaram da mesma abordagem de aumentação de dados, o que difere entre eles são os valores dos filtros convolucionais. O modelo B possui os valores de filtros convolucionais menores que o modelo A, sendo esse o motivo da pequena diferença no resultado da acurácia entre eles.

Já o modelo C, que obteve uma taxa de assertividade de 97,37%, não utiliza da abordagem de aumentação de dados e possui os mesmos valores de filtros convolucionais do modelo A.

O que pode-se analisar desses resultados é que o uso da aumentação de dados pode ter criado uma diferença de representatividade nas classes, fazendo com que o modelo gerado, tenha mais sucesso em prever a classe mais representada.

5 CONSIDERAÇÕES FINAIS

A pandemia causada pela COVID-19 exigiu a união de diversas áreas do conhecimento em prol do bem-estar da sociedade, até a data atual de finalização desse trabalho, o ano de 2022, a recuperação é gradativa e cautelosa. A tecnologia, e mais especificamente o segmento dos modelos de inteligência artificial, tem um histórico familiar com a principal frente de combate ao vírus, a área da saúde. Pela dificuldade de diagnóstico rápida e a falta de alternativas para o mesmo durante o primeiro momento da pandemia, o estudo produzido aqui avalia a utilização de uma série de modelos convolucionais para a classificação a partir de radiografias pulmonares.

Além da COVID-19, o trabalho incluiu outras duas doenças que possuem semelhança no que diz a possibilidade de diagnóstico por meio do raio-X, trazendo maior diversificação de exemplos para os modelos. A pneumonia e a tuberculose foram escolhidas por serem doenças conhecidas e com tratamentos já bem definidos, mas que podem receber a contribuição da classificação.

A base de imagens utilizadas recebeu o tratamento manual de segmentação para cada classe e divisão entre treino, validação e teste nos modelos. As principais técnicas utilizadas durante a construção dos três modelos de análise foram focadas na mudança dos filtros nas camadas de convolução, que não apresentaram grande diferença nas métricas, e na utilização de *data augmentation* para aumentar o número de imagens nos *datasets* que faziam parte do treino. Os resultados obtidos para o modelo que não usou imagens transformadas pelo *augmentation* se mostraram melhores diante dos outros dois, as imagens com transformação apenas no tamanho para normalização resultaram em 97% de acurácia na rede.

A quantidade de imagens usadas, mesmo com a técnica de *data augmentation* para validação, é um ponto de destaque que influenciou nos gráficos de acurácia e perda desse *dataset* durante o treinamento. Uma nova quantidade significativa dessas imagens pode ajudar em pesquisas futuras no comportamento dos modelos que utilizaram a técnica.

Diante do exposto, as redes convolucionais se mostraram viáveis para classificação das radiografias com uma boa taxa de acerto, servindo de possível método auxiliar para o diagnóstico das doenças analisadas.

5.1 Trabalhos futuros

Pesquisas futuras relacionadas a esse trabalho podem tratar de alguns pontos já citados além de novas abordagens:

- Novos modelos com variação na arquitetura convolucional;
- Utilização de uma quantidade maior de imagens para validação;
- Extração de lesões ou outras características dos pulmões acometidos nas imagens resultantes da classificação;
- Utilizar outras funções de ativação na camada totalmente conectada, como *maxout* ou *network-in-network*;
- Selecionar novas métricas de qualidade para os modelos, além das apresentadas;
- Verificar até que ponto a técnica de *data augmentation* se faz importante em determinados casos.

REFERÊNCIAS

ALZUBAIDI, Laith et al. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. **Journal of big Data**, v. 8, n. 1, p. 1-74, 2021.

BAELDUNG. **Learning Curves in Machine Learning | Baeldung on Computer Science**. Disponível em: <https://www.baeldung.com/cs/learning-curve-ml>. Acesso em: 18 maio 2022.

BARCA, Maria Carolina Stockler; SILVEIRA, Tiago Redondo de Siqueira; MAGINI, Marcio. Treinamento de redes neurais artificiais: o algoritmo Backpropagation. **IX Encontro Latino Americano de Iniciação Científica, V Encontro Latino Americano de Pós-Graduação–Universidade do Vale do Paraíba, Anais. Jacareí**, p. 13, 2005.

BRITO, Sávio Breno Pires et al. Pandemia da COVID-19: o maior desafio do século XXI. **Vigilância Sanitária em Debate: Sociedade, Ciência & Tecnologia**, v. 8, n. 2, p. 54-63, 2020.

BROWNLEE, Jason. **How to Configure Image Data Augmentation in Keras**. Disponível em: <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>. Acesso em: 30 abril 2022.

BROWNLEE, Jason. **Understand the impact of learning rate on neural network performance**.Disponível em: <https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/>. Acesso em: 07 abril 2022.

CERRI, Ricardo. Introdução às Redes Neurais Artificiais com Implementações em R. In: **Anais da I Escola Regional de Aprendizado de Máquina e Inteligência Artificial de São Paulo**. SBC, 2020. p. 47-50.

CHIMA, P. **Activation Functions: ReLU & Softmax**. Disponível em: <https://medium.com/@preshchima/activation-functions-relu-softmax-87145bf39288>. Acesso em: 15 maio 2022.

CIRESAN, Dan Claudiu et al. Flexible, high performance convolutional neural networks for image classification. In: **Twenty-second international joint conference on artificial intelligence**. 2011.

CLARO, Maíla et al. Utilização de Técnicas de Data Augmentation em Imagens: Teoria e Prática. **Sociedade Brasileira da Computação-SBC. Disponível em:< https://sol. sbc. org. br/livros/index. php/sbc/catalog/download/48/217/455-1>**, 2020.

DONGES, Niklas. **Gradient Descent: An Introduction to One of Machine Learning's Most Popular Algorithms.** Disponível em: <https://builtin.com/data-science/gradient-descent>. Acesso em: 05 abril 2022.

EMMERT-STREIB, Frank et al. An introductory review of deep learning for prediction models with big data. **Frontiers in Artificial Intelligence**, v. 3, p. 4, 2020.

FERNEDA, Edberto. Neural networks and its application in information retrieval systems. **Ciência da Informação**, v. 35, p. 25-30, 2006.

GOOGLE. Google Colaboratory. Disponível em: <https://colab.research.google.com/>.

HINTON, Geoffrey E. et al. Learning distributed representations of concepts. In: **Proceedings of the eighth annual conference of the cognitive science society**. 1986. p. 12.

HOSSIN, Mohammad; SULAIMAN, Md Nasir. A review on evaluation metrics for data classification evaluations. **International journal of data mining & knowledge management process**, v. 5, n. 2, p. 1, 2015.

JABBAR, H.; KHAN, Rafiqul Zaman. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). **Computer Science, Communication and Instrumentation Devices**, v. 70, 2015.

KAGGLE. Kaggle: Your Home for Data Science. Disponível em: <https://www.kaggle.com/>.

KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. **Advances in neural information processing systems**, v. 25, 2012.

LIMA, Claudio Márcio Amaral de Oliveira. Information about the new coronavirus disease (COVID-19). **Radiologia brasileira**, v. 53, p. V-VI, 2020.

LIU, Jin et al. Applications of deep learning to MRI images: A survey. **Big Data Mining and Analytics**, v. 1, n. 1, p. 1-18, 2018.

Machine Learning melhora consideravelmente fotos em ambientes escuros. Disponível em: <https://www.tecmundo.com.br/software/130330-machine-learning-melhora-considera velmente-fotos-ambientes-escuros.htm>. Acesso em: 19 abril 2022.

MARENGONI, Maurício; STRINGHINI, Stringhini. Tutorial: Introdução à visão computacional usando opencv. **Revista de Informática Teórica e Aplicada**, v. 16, n. 1, p. 125-160, 2009.

MARKOULIDAKIS, Ioannis et al. Multiclass Confusion Matrix Reduction Method and Its Application on Net Promoter Score Classification Problem. **Technologies**, v. 9, n. 4, p. 81, 2021.

MARTINS, João Victor Gomes et al. Classificação da COVID-19 em Radiografias do Tórax Utilizando Redes Neurais Profundas e Padrões Binários Locais. **Journal of Health Informatics**, v. 12, 2021.

MATLAB | RGB image representation. Disponível em: <https://www.geeksforgeeks.org/matlab-rgb-image-representation/>. Acesso em: 26 abril 2022.

MATPLOTLIB. Matplotlib: Python plotting — Matplotlib 3.1.1 documentation. Disponível em: <https://matplotlib.org/>.

MCCULLOCH, Warren S.; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, p. 115-133, 1943.

MEIRELLES, Gustavo de Souza Portes. COVID-19: a brief update for radiologists. **Radiologia Brasileira**, v. 53, p. 320-328, 2020.

MIKOŁAJCZYK, Agnieszka; GROCHOWSKI, Michał. Data augmentation for improving deep learning in image classification problem. In: **2018 international interdisciplinary PhD workshop (IIPhDW)**. IEEE, 2018. p. 117-122.

MOREIRA, Catarina. Neurónio. **Revista de Ciência Elementar**, v. 1, n. 1, 2013.

MÜLLER, C.; MÜLLER, Nestor L. Chest CT target sign in a couple with COVID-19 pneumonia. **Radiologia Brasileira**, v. 53, p. 252-254, 2020.

NIE, Feiping; HU, Zhanxuan; LI, Xuelong. An investigation for loss functions widely used in machine learning. **Communications in Information and Systems**, v. 18, n. 1, p. 37-52, 2018.

O'SHEA, Keiron; NASH, Ryan. An introduction to convolutional neural networks. **arXiv preprint arXiv:1511.08458**, 2015.

PAPAGELIS, Anthony J.; KIM, Dong Soo. **Mutli-Layer Perceptron - Back Propagation**. Disponível em: <https://www.cse.unsw.edu.au/~cs9417ml/MLP2/BackPropagation.html>. Acesso em: 03 abril 2022.

SARLE, Warren S. et al. Stopped training and other remedies for overfitting. **Computing science and statistics**, p. 352-360, 1996.

SAZLI, Murat H. A brief review of feed-forward neural networks. **Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering**, v. 50, n. 01, 2006.

SHARMA, Sagar; SHARMA, Simone; ATHAIYA, Anidhya. Activation functions in neural networks. **towards data science**, v. 6, n. 12, p. 310-316, 2017.

SCIKIT-LEARN. scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation. Disponível em: <https://scikit-learn.org/stable/>.

SMITH, Leslie N. A disciplined approach to neural network hyper-parameters: Part 1--learning rate, batch size, momentum, and weight decay. **arXiv preprint arXiv:1803.09820**, 2018.

SOUSA, A. de L. et al. Redes neurais convolucionais aplicadas ao processo de classificação de cultivares de guaranazeiros. In: **Embrapa Amazônia Ocidental-Artigo em anais de congresso (ALICE)**. In: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL E COMPUTACIONAL, 14., 2017, Uberlândia. **Anais...** Uberlândia: Sociedade Brasileira de Computação, 2017., 2017.

STEPHANIE. **Convolution Integral: Simple Definition**. Disponível em: <https://www.calculushowto.com/convolution-integral-simple-definition/>. Acesso em: 23 abril 2022.

VICENTE, Renato. **Perceptrons.** Disponível em: <https://www.ime.usp.br/~rvicente/MachineLearning/W2_Perceptrons.pdf>. Acesso em: 05 abril 2022.

TEAM, K. **Keras documentation: Keras API reference**. Disponível em: <https://keras.io/api/>. Acesso em: 8 maio 2022.

WAINER, Jacques et al. Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação. **Atualização em informática**, v. 1, n. 221-262, p. 32-33, 2007.

WANG, Qi et al. A comprehensive survey of loss functions in machine learning. **Annals of Data Science**, v. 9, n. 2, p. 187-212, 2022.

WU, Jia et al. Hyperparameter optimization for machine learning models based on Bayesian optimization. **Journal of Electronic Science and Technology**, v. 17, n. 1, p. 26-40, 2019.

YAMASHITA, Rikiya et al. Convolutional neural networks: an overview and application in radiology. **Insights into imaging**, v. 9, n. 4, p. 611-629, 2018.

APÊNDICE A - Código desenvolvido para o modelo convolucional A.

```
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from sklearn.metrics import classification_report, confusion_matrix,
plot_confusion_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

drive.mount("/content/gdrive")
path = '/content/gdrive/MyDrive/dataset/'

data_generator_train = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    fill_mode = "nearest"
)

data_generator_test = ImageDataGenerator(rescale = 1./255)

train_generator = data_generator_train.flow_from_directory(
    directory = path + 'train',
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = "categorical",
    batch_size = 32,
    seed = 42,
    shuffle = True
)

test_generator = data_generator_test.flow_from_directory(
    directory = path + 'test',
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = "categorical",
    batch_size = 32,
    seed = 42,
    shuffle = True
)

valid_generator = data_generator_test.flow_from_directory(
    directory = path + 'val',
```

```
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = None,
    batch_size = 1,
    seed = 42,
    shuffle = False
)

model = Sequential()

model.add(Conv2D(kernel_size=(3,3), filters=32, activation='relu', input_shape=(120,
120, 1)))
model.add(MaxPool2D(2,2))

model.add(Conv2D(kernel_size=(3,3), filters=64, activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Conv2D(kernel_size=(3,3), filters=128, activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(4, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.summary()

EPOCHS = 15
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size

model_history = model.fit(
    train_generator,
    steps_per_epoch=STEP_SIZE_TRAIN,
    validation_data=test_generator,
    validation_steps=STEP_SIZE_TEST,
    epochs=EPOCHS,
    verbose= 1
)

model.evaluate(test_generator, steps=STEP_SIZE_TEST)
model.save("XrayClassification")
```

```
pd.DataFrame(model_history.history)[['accuracy','val_accuracy']].plot()
plt.title("Accuracy")
plt.show()

pd.DataFrame(model_history.history)[['loss','val_loss']].plot()
plt.title("Loss")
plt.show()

valid_generator.reset()

pred = model.predict(valid_generator, steps=STEP_SIZE_VALID, verbose=1)

predicted_class_indices = np.argmax(pred, axis=1)

target_names = ['COVID19', 'NORMAL', 'PNEUMONIA', 'TURBERCULOSIS']

cllt = confusion_matrix(valid_generator.classes, predicted_class_indices)
disp = ConfusionMatrixDisplay(confusion_matrix=cllt, display_labels=target_names)

disp.plot(cmap=plt.cm.Blues)
plt.show()

print(classification_report(valid_generator.classes, predicted_class_indices,
target_names=target_names))

labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]

current_idx = 0
count_accurate = 0
Actual = []

for i in predictions:
    string = valid_generator.filenames[current_idx]
    substr = '/'
    actual = string[:string.find(substr)]
    Actual.append(actual)
    pred = predictions[current_idx]
    if actual == pred:
        count_accurate += 1
    current_idx += 1

acc = count_accurate/38

print(f"A acc nas imagens de validação é {round(acc*100,2)}%.")

current = [1, 2, 3, 5, 10, 15, 16, 20, 22, 25, 28, 30, 33, 35, 37]
```

```
for i in current:
    plt.imshow(plt.imread(path + 'val/' + valid_generator.filenames[i]))
    string = valid_generator.filenames[i]
    substr = '/'
    actual = string[:string.find(substr)]
    plt.title(f"True: {actual} \nPrevisto: {predictions[i]}")
    plt.show()
```

APÊNDICE B - Código desenvolvido para o modelo convolucional B.

```
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from sklearn.metrics import classification_report, confusion_matrix,
plot_confusion_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

drive.mount("/content/gdrive")
path = '/content/gdrive/MyDrive/dataset/'

data_generator_train = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    fill_mode = "nearest"
)

data_generator_test = ImageDataGenerator(rescale = 1./255)

train_generator = data_generator_train.flow_from_directory(
    directory = path + 'train',
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = "categorical",
    batch_size = 32,
    seed = 42,
    shuffle = True
)

test_generator = data_generator_test.flow_from_directory(
    directory = path + 'test',
```

```
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = "categorical",
    batch_size = 32,
    seed = 42,
    shuffle = True
)

valid_generator = data_generator_test.flow_from_directory(
    directory = path + 'val',
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = None,
    batch_size = 1,
    seed = 42,
    shuffle = False
)

model = Sequential()

model.add(Conv2D(kernel_size=(3,3), filters=16, activation='relu', input_shape=(120,
120, 1)))
model.add(MaxPool2D(2,2))

model.add(Conv2D(kernel_size=(3,3), filters=32, activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Conv2D(kernel_size=(3,3), filters=64, activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Flatten())

model.add(Dense(64, activation='relu'))
model.add(Dropout(.2))
model.add(Dense(32, activation='relu'))
model.add(Dense(4, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.summary()

EPOCHS = 15
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size

model_history = model.fit(
```

```
    train_generator,
    steps_per_epoch=STEP_SIZE_TRAIN,
    validation_data=test_generator,
    validation_steps=STEP_SIZE_TEST,
    epochs=EPOCHS,
    verbose= 1
)

model.evaluate(test_generator, steps=STEP_SIZE_TEST)
model.save("XrayClassification")

pd.DataFrame(model_history.history)[['accuracy','val_accuracy']].plot()
plt.title("Accuracy")
plt.show()

pd.DataFrame(model_history.history)[['loss','val_loss']].plot()
plt.title("Loss")
plt.show()

valid_generator.reset()

pred = model.predict(valid_generator, steps=STEP_SIZE_VALID, verbose=1)

predicted_class_indices = np.argmax(pred, axis=1)

target_names = ['COVID19', 'NORMAL', 'PNEUMONIA', 'TURBERCULOSIS']

cllt = confusion_matrix(valid_generator.classes, predicted_class_indices)
disp = ConfusionMatrixDisplay(confusion_matrix=cllt, display_labels=target_names)

disp.plot(cmap=plt.cm.Blues)
plt.show()

print(classification_report(valid_generator.classes, predicted_class_indices,
target_names=target_names))

labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]

current_idx = 0
count_accurate = 0
Actual = []

for i in predictions:
    string = valid_generator.filenames[current_idx]
    substr = '/'
    actual = string[:string.find(substr)]
    Actual.append(actual)
```

```
    pred = predictions[current_idx]
    if actual == pred:
        count_accurate += 1
    current_idx += 1

acc = count_accurate/38

print(f"A acc nas imagens de validação é {round(acc*100,2)}%.")

current = [1, 2, 3, 5, 10, 15, 16, 20, 22, 25, 28, 30, 33, 35, 37]

for i in current:
    plt.imshow(plt.imread(path + 'val/' + valid_generator.filenames[i]))
    string = valid_generator.filenames[i]
    substr = '/'
    actual = string[:string.find(substr)]
    plt.title(f"True: {actual} \nPrevisto: {predictions[i]}")
    plt.show()
```

APÊNDICE C - Código desenvolvido para o modelo convolucional C.

```
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from sklearn.metrics import classification_report, confusion_matrix,
plot_confusion_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

drive.mount("/content/gdrive")
path = '/content/gdrive/MyDrive/dataset/'

data_generator_train = ImageDataGenerator(rescale = 1./255)
data_generator_test = ImageDataGenerator(rescale = 1./255)

train_generator = data_generator_train.flow_from_directory(
    directory = path + 'train',
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = "categorical",
    batch_size = 32,
    seed = 42,
    shuffle = True
)
```

```
test_generator = data_generator_test.flow_from_directory(
    directory = path + 'test',
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = "categorical",
    batch_size = 32,
    seed = 42,
    shuffle = True
)

valid_generator = data_generator_test.flow_from_directory(
    directory = path + 'val',
    target_size = (120, 120),
    color_mode = "grayscale",
    class_mode = None,
    batch_size = 1,
    seed = 42,
    shuffle = False
)

model = Sequential()

model.add(Conv2D(kernel_size=(3,3), filters=32, activation='relu', input_shape=(120,
120, 1)))
model.add(MaxPool2D(2,2))

model.add(Conv2D(kernel_size=(3,3), filters=64, activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Conv2D(kernel_size=(3,3), filters=128, activation='relu'))
model.add(MaxPool2D(2,2))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(.2))
model.add(Dense(64, activation='relu'))
model.add(Dense(4, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

model.summary()

EPOCHS = 15
STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
```

```
STEP_SIZE_TEST = test_generator.n//test_generator.batch_size

model_history = model.fit(
    train_generator,
    steps_per_epoch=STEP_SIZE_TRAIN,
    validation_data=test_generator,
    validation_steps=STEP_SIZE_TEST,
    epochs=EPOCHS,
    verbose= 1
)

model.evaluate(test_generator, steps=STEP_SIZE_TEST)
model.save("XrayClassification")

pd.DataFrame(model_history.history)[['accuracy','val_accuracy']].plot()
plt.title("Accuracy")
plt.show()

pd.DataFrame(model_history.history)[['loss','val_loss']].plot()
plt.title("Loss")
plt.show()

valid_generator.reset()

pred = model.predict(valid_generator, steps=STEP_SIZE_VALID, verbose=1)

predicted_class_indices = np.argmax(pred, axis=1)

target_names = ['COVID19', 'NORMAL', 'PNEUMONIA', 'TURBERCULOSIS']

cllt = confusion_matrix(valid_generator.classes, predicted_class_indices)
disp = ConfusionMatrixDisplay(confusion_matrix=cllt, display_labels=target_names)

disp.plot(cmap=plt.cm.Blues)
plt.show()

print(classification_report(valid_generator.classes, predicted_class_indices,
target_names=target_names))

labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]

current_idx = 0
count_accurate = 0
Actual = []

for i in predictions:
    string = valid_generator.filenames[current_idx]
```

```
    substr = '/'
    actual = string[:string.find(substr)]
    Actual.append(actual)
    pred = predictions[current_idx]
    if actual == pred:
        count_accurate += 1
    current_idx += 1

acc = count_accurate/38

print(f"A acc nas imagens de validação é {round(acc*100,2)}%.")

current = [1, 2, 3, 5, 10, 15, 16, 20, 22, 25, 28, 30, 33, 35, 37]

for i in current:
    plt.imshow(plt.imread(path + 'val/' + valid_generator.filenames[i]))
    string = valid_generator.filenames[i]
    substr = '/'
    actual = string[:string.find(substr)]
    plt.title(f"True: {actual} \nPrevisto: {predictions[i]}")
    plt.show()
```