A utilização de redes neurais convolucionais para a identificação de doenças respiratórias em radiografias de tórax

Stheffany da Câmara Guilhermino[1], Paulo César Linhares da Silva[2], Suene Campos Duarte[3]

Resumo: Este trabalho propõe estudar o conceito de redes neurais convolucionais (RNC) como uma ferramenta metodológica para o reconhecimento de doenças respiratórias em radiografias de tórax. As redes neurais convolucionais (RNC) são um tipo especializado de modelo de aprendizado elaborado, planejadas para processar dados em forma de grade, como imagens ou vídeos. São especialmente eficientes em tarefas relacionadas à visão computacional. As imagens médicas desempenham um papel crucial nesse treinamento, fornecendo dados visuais que proporcionam à RNC aprender padrões e características especificas relacionadas a diferentes condições pulmonares. A alta complexidade e alterabilidade das radiografias de tórax necessitam de algoritmos elaborados para a detecção precisa de anomalias. As RNCs, sobretudo projetadas para processar imagens, são bem ajustadas para este fim. Ao qual conseguem constatar características tênues em várias regiões da imagem, como sombras, texturas e opacidades, que são capazes de ser indicativas de várias doenças respiratórias. Ao treinar uma RNC com um grande conjunto de dados de imagens médicas rotuladas, o modelo aprende a relacionar padrões específicos com diagnósticos determinantes. Isso permite que a rede neural desenvolva e identifique padrões semelhantes em novas imagens que não foram parte do conjunto de treinamento. As redes neurais pertencem ao campo de estudo da ciência da computação chamado inteligência artificial, campo de estudo este que cresce a cada dia devido a sua grande versatilidade em aplicações práticas das mais diversas áreas do conhecimento científico e tecnológico. A modelagem e funcionamento da rede neural utilizada aqui tem como base a linguagem de programação *Python*. Esta linguagem foi utilizada por apresentar diversas bibliotecas nativas de inteligência artificial o que ajuda bastante no treinamento da rede neural. O *Python* atualmente é uma das linguagens de programação mais utilizadas para estudo em inteligência artificial, aprendizado de máquina, ciência de dados e dentre outras.

Palavras-chave: Redes neurais convolucionais; Redes neurais; Doenças respiratórias; Reconhecimento; Imagens médicas; Python.

1. INTRODUÇÃO

Atualmente vive-se a era da inteligência das máquinas. Conhecida como a quinta revolução industrial. A Inteligência artificial (IA) é uma tecnologia ou conjunto de tecnologias computacionais que utiliza redes neurais artificiais, algoritmos e sistemas de aprendizado de máquina com ou sem reforço, com o objetivo de simular as capacidades mentais humanas, como por exemplo, raciocínio, percepção de ambiente, reconhecimento de padrões e capacidade de tomada de decisão.

As redes neurais artificiais têm uma extensa história de desenvolvimento, ao qual remota aos anos 1940 e 1950 [1]. A concepção de se fabricar um mecanismo autônomo ou máquina, que seja favorecido de inteligência, se estabelece de um sonho antigo de pesquisadores da área. Naquele tempo o matemático estatístico Warren McCulooch e o neurofisiologista Wlater Pitts apresentaram um modelo de neurônio artificial, inspirado no funcionamento dos neurônios biológicos [2].

O cérebro humano é uma rede complexa de neurônios interconectados, formando uma “máquina” poderosa e complexa, ao qual trabalham juntos para processar informações sensoriais, realizar tarefas cognitivas e controlar o comportamento. O desenvolvimento do cérebro acontece durante os primeiros dois anos de vida, mas se encaminha por toda a vida [3].

A célula elementar do sistema nervoso cerebral é o neurônio, o qual sua unidade básica é o de transmitir informações visto que entre suas propriedades destacam-se a condução de mensagens nervosas e a excitabilidade [3]. O papel do neurônio biológico é o de receber sinais de entrada de outros neurônios ou do ambiente externo, processar esses sinais e produzir uma saída semelhante, que é transmitida também para outros neurônios. A rede neural do cérebro é capaz de aprender e ajustar-se a partir de experiências, ajustando as conexões entre os neurônios e os pesos das sinapses, para aperfeiçoar a precisão e a eficiência do processamento de informação [4]. As redes neurais artificias são programadas para imitar esse processo de aprendizado, empregando modelos matemáticos para simular o funcionamento dos neurônios biológicos e a rede neural do cérebro.

O progresso das redes neurais profundas, conhecida como redes neurais convolucionais, foi um amplo avanço na utilização das redes neurais em tarefas de visão computacional. As redes neurais profundas são aptas a aprender automaticamente os atributos relevantes das imagens, levando a em eficazes tarefas de reconhecimento de segmentação de imagens, objetos e outras tarefas complicadas [5].

As imagens médicas utilizadas neste trabalho, radiografias de tórax servem, em trabalhos que envolvem redes neurais artificiais convolucionais (RNCs), para o reconhecimento de doenças respiratórias. Essas imagens representam com detalhes estruturas torácicas e pulmonares, reunindo informações essenciais sobre o estado dos pulmões e estruturas relacionadas. Além do mais, as radiografias de tórax englobam uma ampla série de condições respiratórias, desde infecções respiratórias comuns até condições mais difíceis, como pneumonias. Estas características são fundamentais para treinar uma rede neural convolucional de forma ampla, possibilitando que o modelo aprenda a reconhecer uma variedade de padrões relacionados a diferentes doenças.

No contexto do reconhecimento de doenças respiratórias, as RNC atuam em um papel crucial, visto que são projetadas para extrair características complexas de imagens. Constatando padrões simples, como texturas e bordas, e em camadas subsequentes, determinam essas características para identificas padrões mais complexos como opacidades e contornos pulmonares.

O treinamento de uma RNC abrange o uso de grandes conjuntos de dados de radiografias de tórax anteriormente rotuladas. Cada imagem está relacionada a um diagnóstico conhecido. Ela aprende a relacionar padrões visuais específicos com diagnósticos determinados, possibilitando a detecção de doenças respiratórias. Uma vez treinada, ela pode detectar padrões parecidos em novas imagens que não foram parte do conjunto de treinamento. Isso quer dizer que ela pode diagnosticar novos casos com exatidão.

Desde os anos 2000 as redes neurais artificiais foram combinadas com outras técnicas de inteligência artificial, como aprendizagem por reforço, para criar sistemas de inteligência artificial mais avançados. Na atualidade as redes são grandemente utilizadas em diversas áreas da inteligência artificial como o reconhecimento de voz, análise de dados, visão computacional, análise de imagens, diagnósticos médicos, entre outros [1]. Desta forma, esse desenvolvimento tem sido conduzido justamente pelo aumento do poder de processamento dos computadores, tornando-se uma ferramenta necessária na área da inteligência artificial.

Neste trabalho, é utilizado um modelo de rede neural convolucional para classificar se uma imagem de radiografia de tórax possui ou não doenças ou algum padrão que caracterize uma doença respiratória. O diagnóstico precoce de doenças respiratórias é de imensa importância para a saúde pública, e as redes neurais desempenham um papel categórico nesse contexto. Dessa forma, segue a seguinte organização: a seção 1 com a introdução, seção 2 com o desenvolvimento, onde é apresentado o algoritmo de aprendizado responsável pelo treinamento da rede neural e detalhes de seu processo de desenvolvimento, seção 3 com os resultados e discussões onde destacamos as métricas de desempenho, para avaliar a eficácia do modelo, e a seção 4 com as considerações finais do trabalho.

1.1 O neurônio biológico e o neurônio artificial

Os neurônios biológicos e artificias são constituintes necessários das redes neurais, sistemas de processamento de informação fundamentais para diversas aplicações em aprendizado de máquina e inteligência artificial. Os neurônios biológicos, presentes nos sistemas nervosos central e periférico de organismos vivos, desempenham um papel crucial na transmissão de informações. Responsáveis por receber, processar e transmitir sinais elétricos e químicos, esses neurônios estabelecem conexões complexas com outras células nervosas e diferentes tecidos.

Cada neurônio biológico é estruturado com um corpo celular, um axônio tubular e ramificações arbóreas conhecidas como dendritos. Os dendritos funcionam como ramificações receptoras, capturando sinais de outras células nervosas. Em contraste, os axônios atuam como linhas de transmissão, se dividindo em ramos que se conectam a pequenos bulbos, formando sinapses com os dendritos de outros neurônios [6]. Esses espaços sinápticos são essenciais para a memorização de informações [7].

Tanto os neurônios biológicos quanto os artificiais são organizados em redes neurais, nas quais os neurônios estão interconectados por meio de sinapses. O fluxo de informação ocorre quando os neurônios enviam sinais uns aos outros, e a força desses sinais é modulada pelas sinapses [4]. Ambos os tipos de neurônios são capazes de processar informações e aprender com dados de entrada, possuindo estruturas semelhantes, como dendritos, corpo celular, axônio e terminais nervosos.

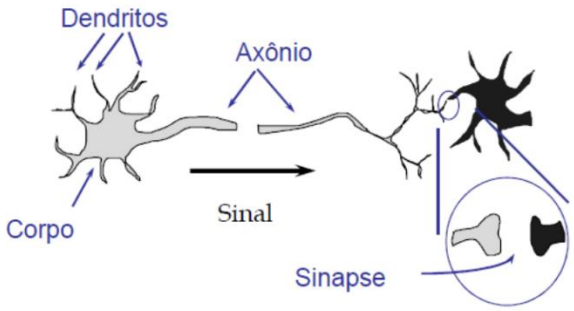

Figura 1: Modelo de um Neurônio Biológico.
Fonte: [ALENCAR, 2018]

Os neurônios artificiais são unidades de processamento de informações que são modeladas com base nas características dos neurônios biológicos [2]. Em uma rede neural, os neurônios artificiais interligam-se para construir uma complexa estrutura de processamento de informações. Essas interconexões entre os neurônios são ponderadas, ou seja, a força dessas conexões pode variar. Essas redes podem ser implementadas tanto em *hardware* quanto em *software* e apresentam diversos modelos matemáticos que definem seu comportamento, tais como o modelo de *perceptron*, o modelo de McCulloch-Pitts, entre outros [4].

É observado abaixo (Figura 2) que cada neurônio artificial tem várias entradas, que podem ser valores numéricos, caracterizando informações de entrada, como características de dados. Cada entrada é multiplicada por um peso correspondente, que define a importância daquela entrada para os dados de saída do neurônio. Os pesos são ajustados durante o treinamento do neurônio.

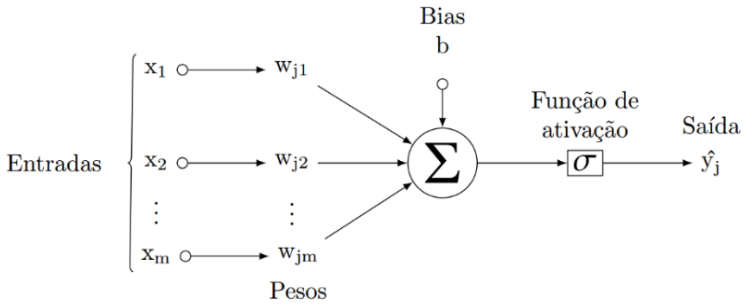

Figura 2: Modelo Artificial de um Neurônio Biológico.
Fonte: [BATISTA, 2022]

Na equação 1 é representada uma função conhecida como a função de ativação. A função de ativação introduz não linearidades na saída do neurônio ao que se refere à aplicação de uma função não linear à combinação linear das entradas ponderadas pelos pesos, mais um termo de viés. O propósito dessa não linearidade é permitir que a rede neural aprenda e represente relações complexas nos dados. A escolha da função de ativação depende do problema e pode incluir funções como a *sigmoide* que transforma qualquer valor real em um intervalo de 0 a 1 e a *ReLU* que retorna zero para valores negativos e mantém os valores positivos.

Assim, a função de ativação ($\boldsymbol{f}$) mapeia o resultado do produto escalar entre $\boldsymbol{w}$ (vetor de peso) e $\boldsymbol{x}$ (vetor de entradas), somando ao termo de viés $\boldsymbol{b}$. Em outras palavras, $\boldsymbol{f}$ opera no resultado bruto da combinação linear de entradas ponderadas pelos pesos mais o viés. Para melhoras a clareza, pode-se reescrever a equação incluindo índices para representar as entradas e pesos individualmente como é visto na equação 2, ao qual o $\boldsymbol{n}$ representa o número de entradas, $\boldsymbol{w_i}$ é o peso associado à entrada $\boldsymbol{x_i}$, e $\boldsymbol{b}$ é o termo de viés.

$$\boldsymbol{y = f\,(w \cdot x\; + b)}$$

Equação 1: Cálculo da Saída do Neurônio.

$$\boldsymbol{y = f\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right)}$$

Equação 2.

A saída do neurônio ($\boldsymbol{y}$) é o resultado da aplicação da função de ativação à soma ponderada mais o viés. Sendo assim, essa saída pode ser utilizada como entrada para outros neurônios em uma rede neural ou pode ser a saída final da rede, dependendo da arquitetura da rede e da tarefa a ser realizada. Por meio do treinamento, que inclui o ajuste dos pesos e do viés, o neurônio pode aprender a realizar tarefas específicas, resultando uma unidade flexível para modelar e processar dados em aprendizado de máquina e inteligência artificial.

Uma das principais diferenças entre esses neurônios é que os neurônios biológicos são células vivas presentes em organismos vivos, enquanto os neurônios artificias são unidades de processamento de informações que são formadas com base nas atribuições dos neurônios biológicos [4]. Além disso, os neurônios biológicos são muito mais complexos do que os neurônios artificiais, com múltiplos tipos de canais iônicos e neurotransmissores, além de terem informações com outras células e componentes do corpo. Os artificiais são mais precisos e previsíveis do que os biológicos, uma vez que os modelos matemáticos que representam seu comportamento são bem definidos e controláveis [5].

1.2 O uso da linguagem python para o estudo das redes neurais artificias

A linguagem de programação *Python* é uma das mais reconhecidas e utilizadas atualmente no campo de redes neurais convolucionais e aprendizado profundo em geral. Sua popularidade deve-se à combinação de diversas características que a tornam uma escolha ideal para essas aplicações. Criada por Guido Van Rossum (1991), uma das suas principais caraterísticas é a síntese clara e intuitiva, possibilitando que os desenvolvedores escrevam códigos de forma mais eficiente e produtiva. Na Figura 3 apresentamos um exemplo de uma rede neural simples e uma de aprendizado profundo.

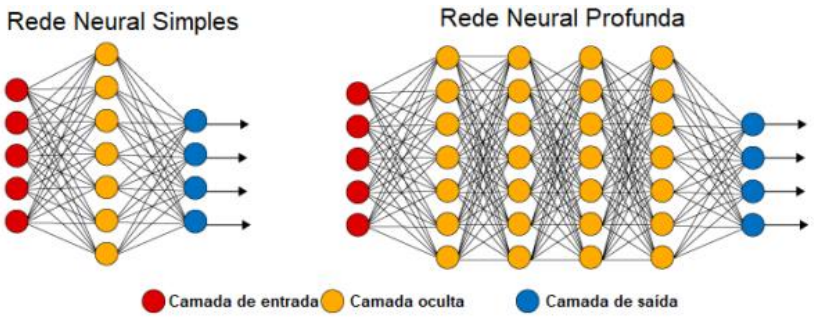

Figura 3: Exemplo de Arquiteturas de Redes Neurais.
Fonte: [Adaptado de Primo.ia, 2020]

O *Python* é uma linguagem versátil conhecida por sua facilidade de uso e legibilidade [6]. Que pode ser empregada para uma extensa gama de aplicações e tem se tornado uma escolha popular para o estudo e implementação de redes neurais artificiais (RNAs) [8], incluindo redes neurais convolucionais (RNCs) para tarefas de visão computacional. A linguagem detém uma comunidade ativa e diversas bibliotecas avançadas que são fundamentais para o desenvolvimento de RNCs e outras aplicações de aprendizado profundo.

Uma das bibliotecas mais populares é o *TensorFlow*, desenvolvido pelo Google Brain. Essa biblioteca oferece uma grande gama de ferramentas para construir, treinar e implementar redes neurais artificiais, incluindo RNCs [7]. Além disso, a biblioteca apresenta um suporte a GPUs, o que acelera significativamente o processamento das redes neurais artificiais e uma Interface de Programação de Aplicações (API) de alto nível chamada *Keras*, que simplifica ainda mais o desenvolvimento de modelos. O API é um conjunto de regras e definições que permitem que diferentes softwares se comuniquem entre si, definindo os métodos e formatos de mensagens que os desenvolvedores podem usar ao construir aplicativos que simplifica ainda mais o desenvolvimento de modelos. Já o *Keras* foi desenvolvido para auxiliar o uso do *TensorFlow*, seus principais componentes e etapas são: camadas, modelos, compilação e treinamento [7], e foi desenvolvido sobre o *Theano* que foi uma biblioteca de código aberto para computação numérica eficiente em *Python*, especialmente adequada para treinamento de redes neurais, ao qual permitia que os usuários definissem, otimizassem e avaliassem expressões matemáticas envolvendo *arrays* multidimensionais eficientemente.

Outra biblioteca bastante usada é o *PyTorch*, que é conhecida por sua dinâmica para construção de modelos, permitindo flexibilidade e facilidade de depuração. Assim como o *TensorFlow*, o *PyTorch* também oferece suporte a GPUs para acelerar o treinamento e a inferência de RNAs [9]. Uma característica importante dessa biblioteca é sua dinâmica para a construção de modelos.

Conforme as necessidades especificas, pode-se escolher a biblioteca que se quer trabalhar no *Python*. Assim tem-se vários tipos de redes neurais artificiais como redes neurais convolucionais que são ideais para tarefas de visão computacional, enquanto redes neurais recorrentes são mais adequadas para lidar com sequência de dados ou redes neurais generativas que são utilizadas para criação de conteúdo. Neste trabalho, a rede neural abordada é a convolucional, e o *Python* foi o escolhido devido ao seu ambiente poderoso e flexível para o estudo e implementação de redes neurais artificiais.

2. DESENVOLVIMENTO

Nesta seção é abordado as etapas de construção do algoritmo utilizado para o reconhecimento dos padrões nas imagens de radiografias do pulmão, aonde o treinamento do modelo de rede neural convolucional e testes feitos com ele foram realizados no *Spyder* (anaconda3). O *Spyder* é um ambiente de execução gratuito, prático e fácil de instalar graças ao gerenciador de pacote *Python*.

Basicamente, o algoritmo a ser apresentado consta de dez etapas. A primeira etapa refere-se a coleta de dados no banco de imagens chamado *Grape Disease Dataset (Original)* disponível na plataforma *Kaggle*. Nas etapas dois, três e quatro é feito o tratamento das imagens para que a rede neural não seja sobrecarregada com a realização das diversas operações envolvendo o grande volume de pixels que formam as imagens. A quinta etapa consta da criação da rede neural densa que receberá os dados após o tratamento das etapas anteriores. As etapas seis, sete e oito trata do processo de compilação da rede neural e as etapas nove e dez mostram os resultados previstos pela rede neural.

2.1 Banco de Imagem

O banco de imagens usado para a execução deste trabalho foi *Chest X-Ray Images (Pneumonia)* do autor *Paul Mooney*, que é um banco de dados de imagens públicas desde que siga os termos listados na licença (CC BY 4.0). Assim, seguindo os termos impostos pela plataforma online *Kaggle* que oferece uma variedade de recursos para cientistas de dados, pesquisadores e entusiastas de aprendizado de máquina, fundada em 2010 e adquirida pela *Google* em 2017. Foi permitido o compartilhamento e adaptação das imagens. Tal plataforma contém cerca de 5.863 imagens de radiografias de tórax separadas em 2 categorias: Pneumonia e Normal [16].

A radiografia do tórax ou raio x é um exame que serve para auxiliar o médico no diagnóstico ou avaliação da resposta aos tratamentos em várias patologias, como por exemplo, a pneumonia. A radiografia é uma técnica utilizada na medicina há muitos anos, tornando-se um método de diagnóstico de primeira linha em termos de avaliação da anatomia humana [10].

A radiografia de tórax normal mostra uma imagem clara e nítida dos pulmões e das estruturas circundantes. Os pulmões aparentam transparência, com uma trama vascular fina se ampliando pela área. As áreas do tórax análogos aos pulmões têm de ter uma aparência escura na radiografia, mostrando que a maior parte da radiografia passou através dos pulmões sem ser absorvida. Já na radiografia de tórax que apresenta pneumonia, que é uma infecção inflamatória dos pulmões, normalmente causada por bactérias, fungos, vírus ou outros agentes infecciosos, é possível observar algumas características diferentes como a opacidades, consolidação, bordas definidas e aumento das tramas vasculares. Todos esses fatores podem variar de acordo com o estágio da infecção, do agente causador e de outras causas (Figura 4).

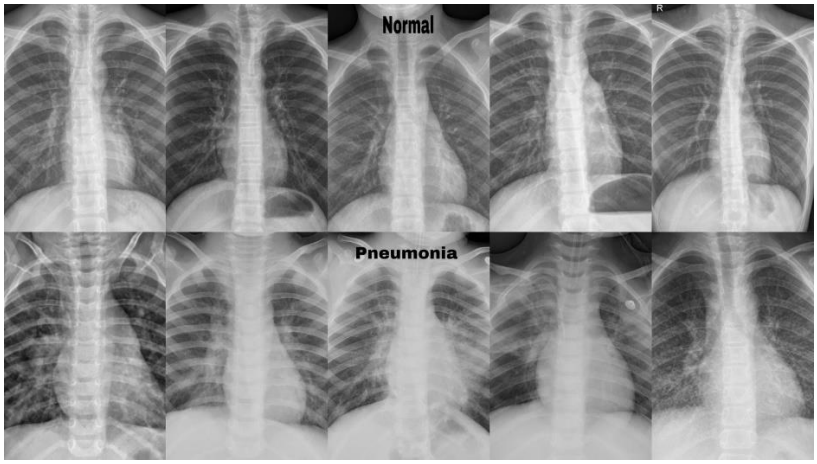

Figura 4: Imagens das Radiografias Normal e Pneumonia.
Fonte: [Adaptado de Mooney, 2023]

As imagens utilizadas são divididas em categorias para a preparação do *dataset* que é uma coleção de dados organizados e estruturados de maneira que seja possível realizar análises, processamento ou treinamento de modelos, e assim serem usadas na aplicação. Foi feita uma análise das imagens pelo próprio do autor *Paul*, onde em um primeiro momento foram examinadas considerando a qualidade das mesmas, excluindo todos os exames de baixa qualidade ou ilegíveis. Os diagnósticos das radiografias foram avaliados por dois médicos especializados para somente assim, ocorrer a liberação. Logo, foi feita uma divisão das imagens em uma arquitetura de pastas, tendo uma de teste e uma de treinamento, cada uma delas possui 2 pastas em seu interior denominadas de normal e pneumonia. A pasta de teste tem 234 imagens de radiografia normal e 390 imagens de

radiografia com pneumonia. Na pasta de treinamento, tem-se 1.349 imagens de radiografia normal e 3.884 imagens de radiografia com pneumonia. Essa divisão de quantidade de imagens em cada categoria é essencial para que a rede não favoreça a interferência de uma categoria sobre outra.

Imagens médicas como as descritas acima compõem uma categoria importante de dados usada em trabalhos de redes neurais convolucionais (RNC). Tais trabalhos podem resultar em diagnósticos mais precisos, tratamentos mais eficazes e melhor compreensão das doenças.

Para treinar a RNC é essencial preparar e representar os dados de forma adequada. Essa rede opera em imagens representadas como matrizes de *pixels*, as quais a representação varia de acordo com a natureza da imagem. Como exemplo, considere as imagens em escala de cinza, onde cada *pixel* é representado por um único valor de intensidade que varia de 0 (preto) a 255 (branco) conforme apresentado na Figura 5. A imagem é uma matriz bidimensional, onde cada entrada corresponde a um *pixel*. Por outro lado, as imagens coloridas, RGB, são representadas como matrizes tridimensionais. Cada *pixel* é descrito por três valores equivalentes às intensidades das cores vermelho, verde e azul, como mostrado na Figura 6. Logo, a imagem é representada como uma matriz 3D, normalmente com dimensões (altura, largura, 3) onde 3 representa os três canais de cor.

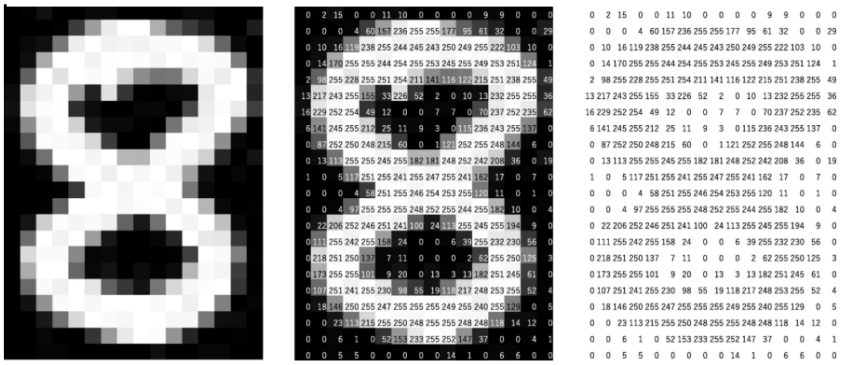

Figura 5: Imagem como Matriz de Pixels.
Fonte: [UNAL, 2019]

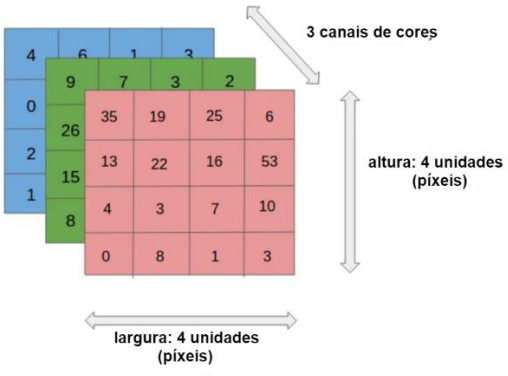

Figura 6: Imagem Utilizando Três Canais.
Fonte: [Adaptado de MEDIUM, 2018]

Antes de fornecer as imagens à RNC, é comum realizar o pré-processamento delas para garantir que estejam em um formato apropriado para o modelo. Nesse sentido, frequentemente deve ser feito um redimensionamento para um tamanho fixo, caso as imagens tenham tamanhos diferentes, visto que a maioria das redes neurais convolucionais requerem entradas de tamanho consistente. Normalizar os valores dos *pixels* para um intervalo específico, como [0,1] ou [-1,0], ajuda na convergência do treinamento e no melhor desempenho da rede e principalmente quando se trabalha com valores RGB. Também é necessário aumentar a variedade de dados de treinamento e tornar o modelo mais desenvolvido, procedimentos como rotação, deslocamento e espelhamento podem ser aplicadas às imagens.

Finalmente, as imagens pré-processadas são convertidas em *arrays NumPy*, ao qual é uma biblioteca *Python* que ajuda o trabalho com vetores e matrizes. As imagens são transformadas para que possam ser manipuladas ativamente pela RNC. As imagens que são em escala de cinza têm suas matrizes bidimensionais convertidas em um *array NumPy 2D* e as imagens que são coloridas (RGB) tem suas matrizes convertidas em um *array NumPy 3D*, mantendo as dimensões (altura, largura, 3).

Essa etapa de preparação de dados é fundamental para capacitar a rede neural convolucional (RNC) a aprender padrões distintos nas imagens e consequentemente realizar com sucesso tarefas complexas de visão computacional.

2.2 Passos para a criação do algoritmo para uso da rede neural

Nesta seção, são discutidas as etapas envolvidas na criação do algoritmo utilizado para identificar padrões. Para a elaboração e treinamento do modelo de aprendizado profundo, utiliza-se a biblioteca *Keras*, que é uma API de nível superior integrada ao *TensorFlow*, bem como o *ImageDataGenator* que é uma classe da biblioteca *Keras* particularmente útil para o pré-processamento das imagens. Abaixo na Tabela 1, tem-se as etapas de criação e utilização do algoritmo.

Tabela 1: Etapas de criação e utilização do algoritmo da rede neural.

1ª etapa: Coleta e Tratamento dos Dados
2 ª etapa: Realização do Processo de Convolução
3 ª etapa: Realização do Processo de Pooling
4ª etapa: Realização do processo de Flattening
5ª etapa: Criação da Rede Neural Densa
6ª etapa: Compilação do Modelo
7ª etapa: Processo de Augmentation
8ª etapa: Treinamento da Rede Neural
9ª etapa: Previsão de uma Imagem de Teste
10 ª etapa: Resultado da Previsão

Tabela 2: Etapas na construção de uma rede neural convolucional

A seguir é detalhado cada uma das dez etapas apresentadas na Tabela 1 de um algoritmo de redes neurais convolucionais (RNC) para processar imagens de radiografias de tórax.

1ª Etapa: Coleta e Tratamento dos dados

Nessa etapa, inicialmente, é realizada uma pesquisa para encontrar as imagens que serão processadas pelo algoritmo. O banco de dados selecionado para essa finalidade foi encontrado no site *Kaggle* com o seguinte título *"Chest X-Ray Images (Pneumonia)"* e está disponível sob a licença (CC BY 4.0). No total, são utilizadas 5.863 imagens nesse banco de dados. Essas imagens são organizadas e separados em três pastas distintas *(train, test, val)*, cada uma destinada a um propósito específico no processo de treinamento e validação do modelo. Além disso, dentro de cada uma dessas pastas, são criadas subpastas para categorizar as imagens em relação à presença ou ausência de pneumonia. Para garantir a qualidade dos dados usados na análise, todas as imagens passam pelo controle de qualidade removendo todas as varreduras de baixa qualidade ou ilegível, o que contribui significativamente para evitar potenciais erros de classificação durante o treinamento do modelo.

No que diz respeito à importação das bibliotecas, o código inicia importando todas as bibliotecas necessárias para a execução do projeto. Isso inclui, entre outras, os módulos do *Keras*, uma API de alto nível amplamente utilizada para a construção e treinamento de redes neurais. Além disso, também é importado o módulo "*ImageDataGenerator*" do *Keras*, que desempenha um papel claro no pré-processamento das imagens, preparando-as para serem fornecidas ao modelo de aprendizado de máquina. Essa etapa de pré-processamento é essencial para garantir que as imagens estejam em um formato adequado para o treinamento eficaz do modelo. Na Figura 8 que está abaixo, são apresentadas as bibliotecas utilizadas no algoritmo.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten,Dense,Dropout
from keras.layers.normalization.batch_normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
#from keras.preprocessing import image
from tensorflow.keras.preprocessing import image
```

Figura 8: Bibliotecas Utilizadas.

2ª Etapa: Realização do Processo de Convolução

A arquitetura da Rede Neural Convolucional (CNN) é criada utilizando o modelo Sequencial do *Keras*, o qual permite empilhar camadas, uma após a outra, de maneira linear e assim facilitando a construção de modelos complexos. A primeira camada adicionada é uma camada de convolução (Conv2D), que é essencial para extração de características importantes das imagens. Essa camada possui 32 filtros de tamanho 3x3. Os filtros

agem deslizando sobre a imagem e calculando produtos de convolução em pequenas regiões, o que possibilita detectar padrões e características visuais como texturas, formas e bordas. A função de ativação *ReLu* é aplicada a cada saída da convolução (activation='relu'), e é utilizada para introduzir não linearidade, tornando a rede capaz de aprender relações complexas nos dados. Após a camada de convolução, é inserida uma camada de normalização em *batch* que se refere a uma coleção de dados ou amostras que são processadas simultaneamente durante uma operação em aprendizado de máquina ou treinamento de redes neurais (BatchNormalization ()). Essa camada tem o objetivo de normalizar as ativações dos neurônios da camada anterior. Isso quer dizer que ela ajusta as médias e desvios padrão das ativações durante o treinamento, o que ajuda a estabelecer e acelerar o treinamento da rede. Além do mais, a normalização em lote ajuda a diminuir problemas como o desvanecimento de gradientes, melhorando a capacidade da rede de aprender representações importantes dos dados.

Na Figura 8 é observado uma imagem de entrada 6x6 e um filtro 3x3. Para cada campo receptivo, é realizada a soma dos produtos alcançados. Esse número caracteriza o quanto aquela região foi ativada pelo filtro. Assim esse processo vai se repetindo até que toda a imagem tenha sido percorrida, dando origem a um mapa de ativação como é mostrado na Figura 9.

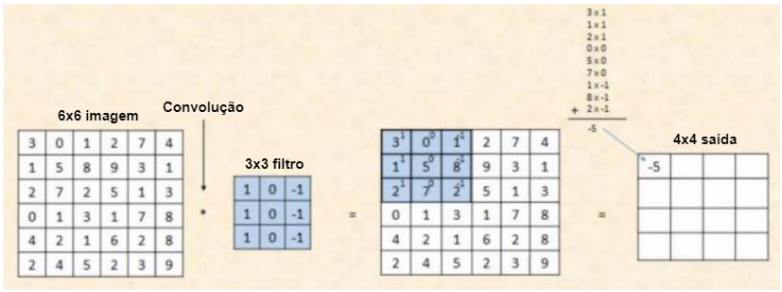

Figura 8: Convolução entre imagem de entrada 6x6 e filtro 3x3.
Fonte: [Adaptado de RIZWAN, 2018]

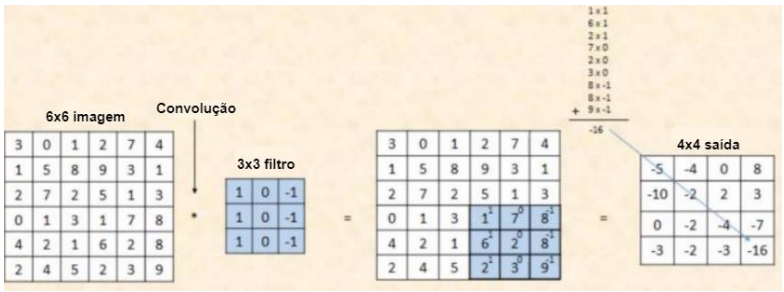

Figura 9: Mapa de ativação com dimensões 4x4 gerado.
Fonte: [Adaptado de RIZWAN, 2018]

No exemplo acima, a imagem de entrada com dimensões 6x6 e um filtro de tamanho 3x3 que gerou um mapa de ativação de tamanho 4x4. Em alguns casos, esse encolhimento pode ser indesejável, especialmente quando se deseja manter as dimensões originais. Para resolver isso, uma técnica chamada "*zero padding*" (preenchimento com zeros) é aplicada, onde uma borda de zeros é adicionada à imagem original antes da convolução como é mostrado na Figura 10.

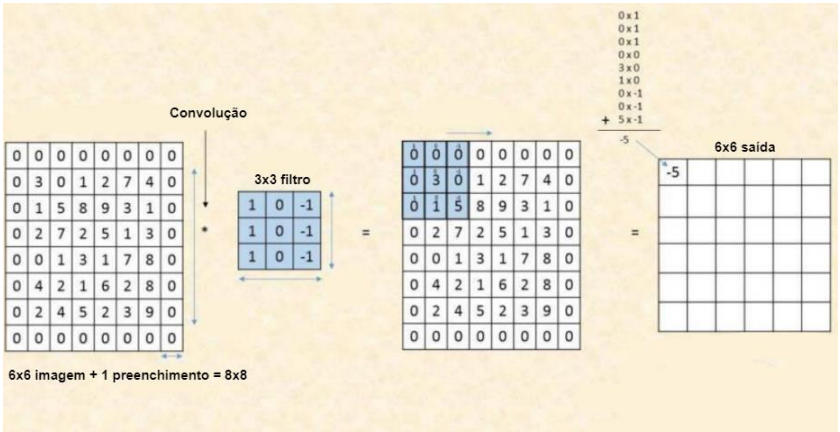

Figura 10: Borda de zeros detém que os dados encolham na convolução.
Fonte: [Adaptado de RIZWAN, 2018]

3ª Etapa: Realização do Processo de *Pooling*

Esse processo é uma etapa importante após as camadas de convolução. Seu principal objetivo é diminuir a dimensionalidade das representações de características, o que traz uma série de vantagens. Primeiramente, isso resulta na rede mais eficiente computacionalmente, uma vez que diminui o volume de dados que precisa ser processado nas camadas subsequentes. Além disso, a diminuição de dimensionalidade contribui para diminuir a quantidade de parâmetros que precisam ser treinados, o que ajuda a evitar o *overfitting* que ocorre quando um modelo de aprendizado de máquina se ajusta excessivamente aos dados de treinamento, capturando não apenas os padrões relevantes, mas também o ruído e detalhes específicos desses dados, um problema comum em redes profundas. Outro benefício significativo do *Pooling* é sua habilidade de tornar a rede mais desenvolvida em relação a pequenas variações na posição das características nas imagens. Isso quer dizer que, mesmo que um objeto de relevância se desloque um pouco dentro de uma imagem, a rede ainda será capaz de identificá-lo de forma eficiente.

O tipo mais comum de *pooling* que é utilizado no trabalho é o *MaxPooling*, com tamanho de *pool* (2,2). O *MaxPooling2D* atua em regiões pequenas da imagem e apenas seleciona o valor máximo dentro de cada região. Assim ajuda a preservar as características mais relevantes na imagem, enquanto descarta informações menos relevantes. Geralmente, esse processo é repetido várias vezes em uma RNC, aumentando gradativamente a complexidade e a abstração das características retiradas. Isso ocorre por conta que as primeiras camadas capturam detalhes de baixo nível, como texturas e bordas, enquanto as camadas decorrentes focam em características de alto nível, como objetos e formas. Na Figura 8 é abordado um exemplo que mostra o *MaxPooling* retângulo 2x2 e *stride* 2, que o valor máximo no interior do retângulo é retornado para cada região percorrida.

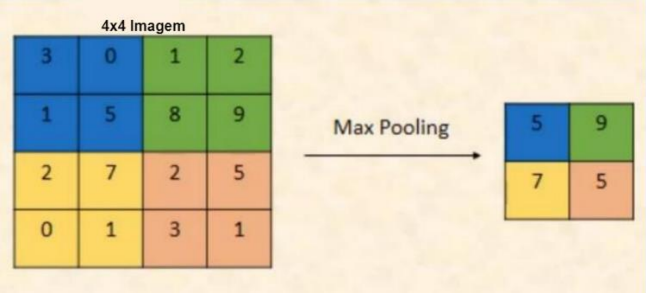

Figura 8: MaxPooling retângulo 2x2 e sride 2.
Fonte: [Adaptado de RIZWAN, 2018]

4ª Etapa: Realização do Processo de *Flattening*

Agora, uma etapa determinante é a introdução de uma camada de achatamento (*Flatten*), que tem a função de transformar a saída atual, que está em formato de matriz ou tensor, em um vetor unidimensional. Essa transformação é fundamental para a subsequente conexão com camadas densas (totalmente conectadas), desempenhando um papel claro no processo de aprendizado da rede neural. A camada de achatamento simplifica a representação das características extraídas das camadas anteriores, convertendo-a em um formato adequado para que as camadas densas processem as informações de maneira eficaz. Isso é essencial para que a rede neural seja capaz de identificar e compreender padrões e relações mais intrincados e complexos entre essas características, permitindo-lhe realizar tarefas de aprendizado mais sofisticadas. Na ilustração apresentada na Figura 9, conforme destacado na apresentação de Andrew Ng (2017), ao conduzir uma operação de convolução com 400 filtros de dimensões 5x5x16 em um bloco de dados de dimensões 5x5x16, resulta em um vetor de dados 1x1x400. Este procedimento pode levar à perda de características especiais, visto que os filtros têm o mesmo tamanho que os mapas de ativação. Isso evidencia o processo de achatamento, ou *flattening*, no qual a representação tridimensional é reduzida a um vetor unidimensional de modo a preservar as características mais essenciais do conjunto de dados.

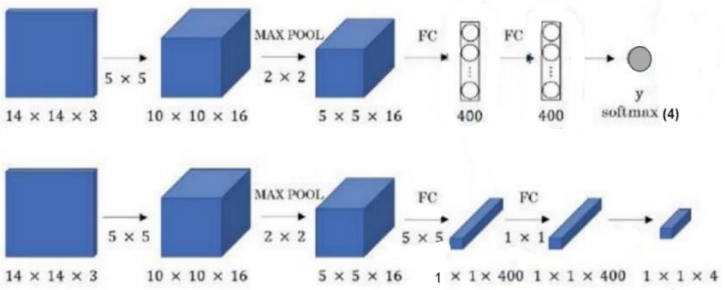

Figura 9: Exergando RNCs como FCNs.
Fonte: [NG, 2017]

5ª Etapa: Criação da Rede Neural Densa

Nesta etapa, foi introduzido camadas densas (*Dense*) que são totalmente conectadas, ou seja, cada neurônio em uma camada densa está interligado a todos os neurônios na camada anterior. São adicionadas duas dessas camadas densas, cada uma composta por 128 unidades. A quantidade de unidades é um hiper parâmetro ajustável, o que permite uma adaptação flexível as demandas especificas do problema em questão. A escolha da função de ativação para essas camadas é a função *ReLU* (*Rectified Linear Unit*), geralmente representada como (activation='relu'). A função *ReLU* é amplamente utilizada em redes neurais devido à sua capacidade de introduzir não linearidades nas saídas das camadas, permitindo que a rede aprenda relações complexas nos dados.

Para atenuar o risco de *Overfitting*, uma técnica geralmente empregada é a inserção de camadas de *Dropout* após cada camada densa, com uma taxa de 20%. O *Dropout* (Dropout(0.2)) é uma técnica que, durante o treinamento, aleatoriamente desativa algumas unidades (neurônios) em uma camada, o que ajuda a reduzir a dependência excessiva entre unidades e, consequentemente, aprimora a capacidade de generalização do modelo. Essa estratégia é importante quando se tem redes neurais profundas e muitos parâmetros, pois ajuda a evitar que o modelo se ajuste demais aos dados de treinamento.

A última camada densa desempenha um papel crucial na tarefa em questão. Ela possui apenas uma unidade e utiliza a função *Sigmoid* (activation='sigmoid'). A função *Sigmoid* é frequentemente empregada em problemas de classificação binaria. Ela transforma a saída da rede em uma probabilidade, indicando a probabilidade da imagem ser classificada como pulmão com pneumonia ou pulmão normal. Isto é, a saída dessa camada fornece uma estimativa da confiança da rede em sua classificação, ajudando a quantificar a incerteza associada à decisão da rede.

6ª Etapa: Compilação do Modelo

Esta etapa desempenha um papel crucial logo após a criação da arquitetura da rede neural, pois configura o modelo para o treinamento, especificando elementos vitais como o otimizador, a função de perda e as métricas de avaliação. No processo de compilação do modelo, empregou-se o *adam* (optimizer='adam') que é um otimizador que ajuda dinamicamente as taxas de aprendizado durante o treinamento, proporcionando uma convergência mais eficiente e estável, sendo uma escolha popular em algoritmos de otimização para redes neurais.

A função de perda é selecionada como *Binary Crossentropy* (loss='binary_crossentropy'), uma vez que estamos tratando de um problema de classificação binária. Neste contexto, cada amostra é categorizada como pertencendo a uma das duas classes possíveis: pulmões com pneumonia ou pulmões normais. Quanto à métrica de desempenho, foi optada por *Accuracy* (acurácia) (metrics=['accuracy']), que quantifica a proporção de previsões corretas em relação ao número total de previsões ao longo das fases de treinamento e teste. Essa métrica é amplamente utilizada para avaliar a precisão de classificação do modelo, fornecendo uma medida fundamental do seu desempenho, como é mostrada na Equação 3.

7ª Etapa: Processo de *Augmentation*

Argmentation (aumento ou aumento de dados) refere-se a uma técnica usada no treinamento de modelos de aprendizado de máquina, sendo sua principal finalidade elevar o desempenho e a capacidade de generalização dos modelos. Nesse contexto, diversas transformações são aplicadas durante o processo, com o intuito de aprimorar a qualidade e a diversidade do conjunto de treinamento.

Um elemento-chave dessa técnica é o uso do gerador de dados, especificamente o *ImageDataGenerator*, que desempenha um papel crucial no pré-processamento das imagens. No gerador de treinamento (gerador_treinamento), uma variedade de transformações é aplicada às imagens de treinamento. Isso inclui redimensionamento, rotações aleatórias, espelhamento horizontal, alterações na altura, zoom e várias outras técnicas. Todas essas transformações cumulativamente contribuem para a expansão do conjunto de treinamento, fornecendo ao modelo uma base mais sólida para aprendizado e generalização. Já no gerador de teste (gerador_teste), a abordagem é um pouco mais restrita, com apenas a etapa de redimensionamento sendo aplicada. Isso é feito para manter a coerência das imagens de teste com o formato esperado pelo modelo, garantindo que as previsões sejam feitas de maneira adequada e precisa.

Em resumo, o aumento de dados é uma prática fundamental no treinamento de modelos de aprendizado profundo, permitindo que eles se tornem mais robustos e eficazes, mesmo quando se dispõe de um conjunto de dados de treinamento limitado.

8ª Etapa: Treinamento da Rede Neural

No processo de treinamento do modelo de rede neural, várias etapas crucias são realizadas utilizando a função *fit_generator* em conjunto com diversos parâmetros fundamentais:

- **Método *fit_generator*:** Este método é essencial para treinar o modelo de rede neural, já que ele coordena o processo de treinamento, permitindo que o modelo ajuste seus pesos com base nos dados de treinamento.
- **Argumento *base_treinamento*:** O gerador de treinamento é especificando como o argumento (base_treinamento). Esse gerador contém as imagens de treinamento justamente com suas respectivas

classes. Ele é a fonte de dados que alimenta o modelo durante o treinamento.

- ***Determinação do número de épocas:*** O número de épocas é configurado usando o argumento *epochs.* As espocas representam o número de vezes que o modelo percorrera todo o conjunto de treinamento durante o treinamento. Essa iteração é fundamental para que o modelo aprenda a partir dos dados.
- **Argumento *steps_per_epoch***: Este parâmetro é calculado como a divisão do número total de amostras de treinamento pelo tamanho do lote (*batch size*). Como no trabalho que tem 5233 amostras de treinamento e um tamanho de lote de 67, então *steps_per_epoch* seria igual a 5233/67. Isso indica o número de lotes (ou mini lotes) que serão processados em cada época de treinamento.
- ***Validation_data*:** Este argumento é utilizado para fornecer dados de validação ao modelo durante o treinamento. O gerador de teste, que inclui as imagens de teste e suas classes correspondentes, é especificado como (*validation_data*). Isso permite que o modelo avalie seu desempenho em um conjunto de dados independente, o que é crucial para evitar *overfitting.*
- **Determinação de *validation_steps:*** Da mesma forma que *steps_per_epoch*, *validation_steps* é calculado como a divisão do número total de amostras de validação pelo tamanho do lote de validação. Isso indica quantos lotes de validação serão processados durante cada época de validação. Como no trabalho que tem 624 amostras de validação e um tamanho de lote de 32, *validation_steps* seria igual a 624/32.
- **Utilização do *save* para salvar o modelo:** Durante o treinamento, é comum utilizar a função *save* para salvar o modelo de rede neural treinado. Isso é especialmente útil para reutilizar o modelo posteriormente, fazer inferências ou continuar o treinamento em pontos diferentes, caso necessário.
- **Ajuste dos pesos durante o treinamento:** Ao longo do processo de treinamento, o modelo ajusta os pesos de suas camadas com o objetivo de minimizar a função de perda. Isso implica na melhoria da precisão da classificação, à medida que o modelo aprende a representar os padrões e características presentes nos dados de treinamento. O processo de treinamento visa otimizar o desempenho do modelo em relação ao problema específico que ele está sendo treinado para resolver.

9ª Etapa: Previsão de uma Imagem de Teste

No processo de inferência de um modelo de rede neural para classificação de imagens, várias etapas são executadas utilizando funções e técnicas do *TensorFlow*:

- **Carregamento da imagem:** Inicialmente, a função *image.load_img* é empregada para carregar uma imagem teste. Esta função é parte do módulo *preprocessing* (pré-processamento) do *TensorFlow* e permite carregar a imagem a ser classificada.
- **Conversão em *array* numéricos:** Posteriormente, a imagem é convertida em uma representação numérica através da função *image.img_to_array*. Isso é fundamental, pois os modelos de rede neural operam com dados numéricos em vez de imagens diretamente.
- **Normalização da imagem:** Para garantir que os valores dos pixels estejam na escala adequada, a imagem é normalizada dividindo-se cada pixel por 255. Isso assegura que os valores estejam na faixa entre 0 e 1, o que é comum em tarefas de processamento de imagens.
- **Ajuste de formato da imagem:** A função *np.expand_dims* é aplicada para ajustar o formato da imagem. Esse passo é essencial para que a imagem seja compatível com as expectativas do modelo de rede neural. Ele adiciona uma dimensão extra à imagem, geralmente referente ao tamanho do lote (*batch size*) que é um parâmetro utilizado que especifica o número de amostras de dados que serão utilizados em uma iteração para atualizar os pesos do modelo durante o processo de treinamento, para garantir que o modelo possa processar a imagem corretamente.
- **Utilização da função *predict:*** Uma vez que a imagem tenha sido adequadamente pré-processada, a função *predict* é usada para fazer previsões ou estimativas com base na imagem. O objetivo é usar o modelo para inferir ou predizer a saída correspondente a novas entradas com base no conhecimento adquirido. Esta etapa é onde o modelo aplica esse conhecimento aprendido durante o treinamento pra classificar a imagem teste.
- **Conversão do resultado em valor *booleano*:** Para torna a saída da previsão mais interpretável, geralmente se utiliza uma comparação com 0,5. Se a previsão for maior que 0,5, é interpretada como *true*, o que indica que o modelo considera a imagem como representativa de um pulmão com pneumonia. Por outro lado, se a previsão for menor ou igual a 0,5, é interpretada como *false*, indicando que o modelo acredita que a imagem retrata um pulmão normal.

Essas etapas são essenciais no processo de inferência desse modelo permitindo que ele faça previsões confiais com base em imagens teste, o que é valioso em uma variedade de aplicações como essa de diagnóstico médico.

10ª Etapa: Resultado da Previsão

Após a execução da função *predict*, o resultado da previsão é disponibilizado e pode ser impresso no console ou em qualquer outra interface de usuário. Esse resultado é de grande importância, pois fornece uma resposta crucial quanto à natureza da imagem do raio-x analisada. O valor obtido na previsão é interpretado de forma simplificada para tornar a informação mais acessível e compreensível. Em particular, o código realiza uma

comparação desse valor com 0,5. Essa comparação determinante para a interpretação do diagnóstico:

- Se a previsão for maior que 0,5, a saída no console indicará (Pulmão com Pneumonia). Esse resultado alerta para a presença de anomalias nos pulmões que são consistentes com a pneumonia, o que pode ser crucial para o diagnóstico médico.
- Se a previsão for menor ou igual a 0,5, a mensagem no console apontará (Pulmão Normal). Nesse caso, a análise sugere que a imagem do raio-x não exibe indícios de pneumonia ou outras condições anormais nos pulmões, proporcionando um diagnóstico tranquilizador.

Abaixo a Figura 10 mostra o resultado do modelo do trabalho:

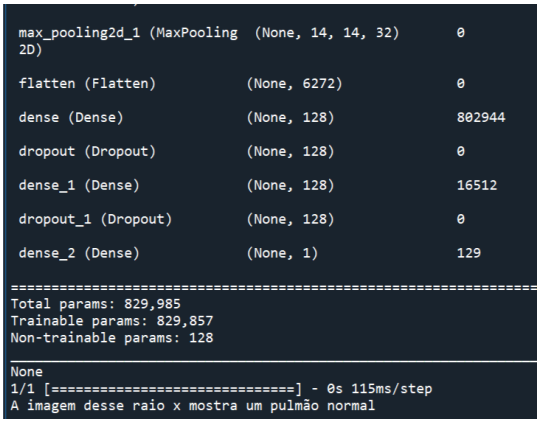

Figura 10: Resultado do Modelo.

Abaixo na Figura 7, tem-se um fluxograma detalhando das etapas para obtenção dos modelos para a classificação das imagens:

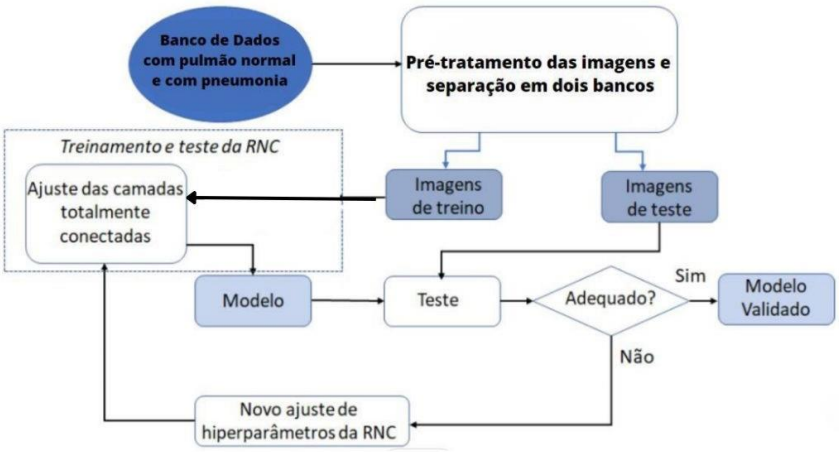

Figura 7: Etapas para obtenção das radiografias de RNCs.
Fonte: [Adaptado de Cecchetto e Botelho, 2020]

3. Resultados e Discussões

Nesta seção, é apresentado os resultados e será acompanhada uma análise da acurácia do modelo proposto para a detecção de doenças no pulmão dos indivíduos, determinando se eles estão saudáveis ou se apresentam sintomas de pneumonia. Os resultados apresentados derivam de uma avaliação rigorosa do modelo, realizada com base em um conjunto de dados de teste composto por 5.867 imagens de radiografias de tórax.

Durante a revisão da literatura relacionada às redes neurais convolucionais, foi identificada diversos desafios enfrentados em trabalhos anteriores, que estavam associados ao conjunto de dados utilizado ou à arquitetura da rede neural desenvolvida. Portanto, para aprimorar o desempenho do modelo, foi optado por adotar uma abordagem diferente em relação aos métodos previamente adotados. Essa abordagem envolveu a utilização de uma funcionalidade oferecida pela biblioteca *Keras*, chamada *ImageDataGenerator*. Essa função permitiu a configuração de várias opções para o pré-processamento das imagens, simplificando significativamente a organização e a preparação dos dados para o treinamento da rede. Para isso, bastou organizar as imagens conforme a estrutura especificada pela biblioteca. Nessa configuração, um diretório principal conteve dois subdiretórios, um destinado as imagens de treinamento e outro as imagens de teste. Por sua vez, cada um desses subdiretórios conteve subdiretórios individuais para cada classe presente na base de imagens.

A Figura 11 apresenta um resumo conciso da arquitetura desenvolvida. Ela é composta duas camadas

convolucionais, sendo a primeira (*Conv2D*) que possui 32 filtros (ou *kernels*) de tamanho 3x3. Esses filtros percorrem a entrada (uma imagem) para extrair características locais. A saída dessa camada é uma matriz tridimensional com dimensões (*None*, 62, 62, 32), onde *None* representa um lote de amostras de tamanho variável, cada uma sendo uma matriz de 62x62 pixels com 32 canais. A segunda camada (*Conv2D*) possui configurações semelhantes à primeira camada, mas com uma redução adicional nas dimensões espaciais (None, 29, 29, 32), devido à aplicação de *max-pooling*. Além disso, foi incluído duas camadas de normalização em lote (*BatchNormalization*) após as camadas convolucionais para estabilizar os valores de saída da camada de convolução, melhorando a convergência do modelo. Seguindo, foi aplicada duas camadas *MaxPooling2D* para realizar operações de *max-pooling*, reduzindo as dimensões pela metade para (31x31x32). Essa etapa destaca características importantes e reduz o número de parâmetros, tornando o modelo mais eficiente. Já a camada *Flatten* é responsável por transformar a saída tridimensional das camadas anteriores em um vetor unidimensional, preparando os dados para as camadas densas (totalmente conectadas) que seguem. As duas primeiras camadas densas possuem 128 neurônios cada, elas recebem os dados após o processo de achatamento (*flattening)* e realizam operações lineares sobre eles. A última camada, denominada *dense_2*, é a camada final, composta por um único neurônio de saída. Em problemas de classificação binária, como esse, essa camada produz a saída final da rede, representando a probabilidade da classe positiva. Duas camadas *Dropout* são inseridas antes da última camada densa para combater o *overfitting*. Durante o treinamento, elas desativam aleatoriamente um percentual dos neurônios, ajudando a rede a generalizar melhor para novos dados.

```
conv2d (Conv2D)              (None, 62, 62, 32)        896

batch_normalization (BatchN  (None, 62, 62, 32)        128
ormalization)

max_pooling2d (MaxPooling2D  (None, 31, 31, 32)        0
)

conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248

batch_normalization_1 (Batc  (None, 29, 29, 32)        128
hNormalization)

max_pooling2d_1 (MaxPooling  (None, 14, 14, 32)        0
2D)

flatten (Flatten)            (None, 6272)              0

dense (Dense)                (None, 128)               802944

dropout (Dropout)            (None, 128)               0

dense_1 (Dense)              (None, 128)               16512

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 1)                 129

=================================================================
Total params: 829,985
Trainable params: 829,857
Non-trainable params: 128
```

Figura 11: Sumário das camadas e saídas.

A rede foi executada durante um total de 55 épocas, que durou cerca de 1:30 horas, cerca de 78 segundos por época, no gráfico abaixo é observado a acurácia do modelo no decorrer das épocas:

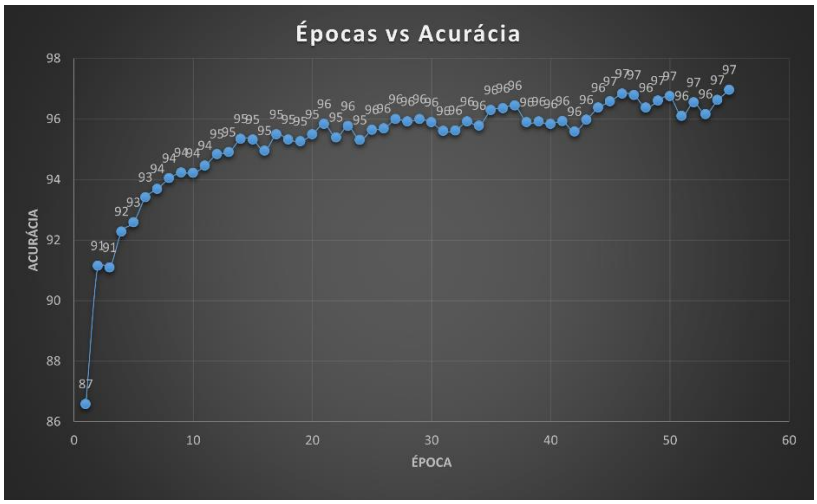

Figura 12: Gráfico da execução da rede convolucional das 55 épocas.

É observado com frequência que, nas primeiras épocas, a acurácia de treinamento (*accuracy*) supera a acurácia de teste (*val_accuracy*). Esse fenômeno ocorre devido ao fato de que os modelos iniciais ainda estão se adaptando aos dados de treinamento aos quais foram submetidos. Os resultados do modelo evidenciam um

aumento contínuo na acurácia de treinamento a cada época, o que está alinhado com as expectativas, uma vez que o modelo aprende a se ajudar de forma mais precisa aos dados de treinamento. Por outro lado, a acurácia de teste, após um início promissor, tende a estabilizar ou apresentar uma diminuição gradual.

Conforme observado nas etapas anteriores, torna-se evidente que a quantidade de imagens disponibilizadas para o treinamento do modelo desempenha um papel fundamental na classificação. A precisão do modelo foi avaliada utilizando a métrica de acurácia, a qual é uma medida amplamente empregada para avaliar o desempenho de algoritmos de classificação e modelos de aprendizado de máquina. Essa métrica quantifica a proporção de predições corretas realizadas pelo modelo em relação ao total de predições. Neste contexto, o modelo demonstrou uma notável acurácia de 0,9696, conforme ilustra na figura 12.

Abaixo está apresentada a fórmula que calcula a acuraria, medida que avalia a proporção de predições corretas feitas pelo modelo em relação ao total de predições realizadas.

$$\boldsymbol{Acurácia} = \frac{\boldsymbol{Número\ de\ previsões\ corretas}}{\boldsymbol{Total\ de\ previsões}}\ \boldsymbol{x}\ \mathbf{100}\%$$

Equação 3.

O valor da acurácia do modelo, cerca de 96,96%, indica que ele classificou corretamente aproximadamente 96,96% das amostras nos conjuntos de dados de teste durante a última época de treinamento. Isso representa uma taxa de precisão notável, sinalizando que o modelo está fazendo previsões precisas para a grande maioria das amostras. Em termos práticos, uma acurácia de 96,96% significa que, em um conjunto de teste com 100 amostras, o modelo classifica corretamente cerca de 96 delas.

Esses resultados enfatizam o potencial do modelo proposto para discernir com precisão entre pulmões normais e casos de pneumonia, como é observado na Figura 13 que apresenta o resultado do modelo. Esta contribuição é significativa para o avanço dos diagnósticos na área médica. Contudo, é importante ressaltar a presença de limitações que devem ser abordados para aprimorar ainda mais a acurácia do modelo.

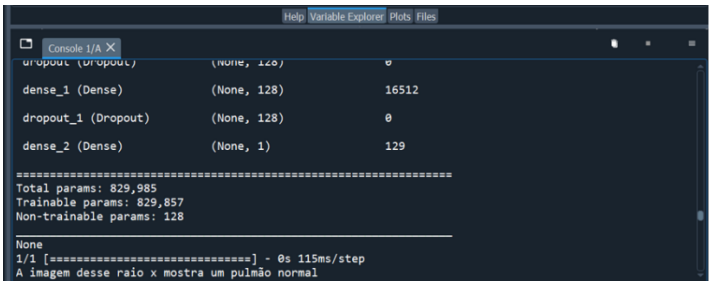

Figura 13: Resultado do Modelo.

4. CONCLUSÕES

A perspectiva de contar com um método eficaz para distinguir entre um pulmão saudável e um afetado por pneumonia, aproveitando as capacidades do aprendizado de máquina e visão computacional, suscita grande interesse no campo da medicina. O modelo desenvolvido neste estudo visa endereçar questões nesse domínio, empregando abordagens voltadas para aprimorar a precisão na identificação e detecção de doenças.

Neste estudo, foram aplicadas técnicas populares de arquiteturas de redes neurais convolucionais (RNCs), incluindo ferramentas como *Keras, ReLU e TensorFlorw*. Essas arquiteturas foram utilizadas para classificar imagens radiográficas em duas categorias: pulmões normais e pulmões com pneumonia. O conjunto de dados constituiu de 5.867 imagens, sendo 1.349 de indivíduos com pulmões normais e 3.884 de indivíduos com pneumonia. Essas imagens foram divididas em conjuntos de treinamento/validação e teste. Para treinar a rede, optou-se pela técnica de transferência de aprendizado, com foco na adaptação das camadas totalmente conectadas das redes. O modelo foi treinado ao longo de 55 épocas.

Os resultados obtidos demonstraram que o modelo conseguiu efetivamente aprender as características distintivas relacionadas a cada categoria durante o processo de treinamento. Isso resultou em uma inteligência artificial capaz de determinar se uma imagem de raio-x representa um pulmão normal ou com pneumonia, alcançando uma acurácia de 96,96%, um resultado notável. Esses achados são encorajadores, indicando que a aplicação de RNCs em imagens de radiografias para diagnostico apresenta avanços promissores e é passível de ser objeto de estudos futuros mais aprofundados. No entanto, é importante ressaltar que os resultados ainda não atingiram um patamar que justifique a substituição dos métodos de diagnostico convencionais.

Para futuras pesquisas, seria interessante propor novas arquiteturas de redes neurais direcionadas para a mesma tarefa de classificação. Essa abordagem permitiria a comparação dos resultados obtidos pelo novo modelo com os alcançados pelas redes pré-treinadas utilizadas neste trabalho. Ademais, o uso de uma base de dados mais ampla poderia aprimorar ainda mais o processo de treinamento da rede. Além disso, seria valioso aprofundar a investigação por meio de análises mais detalhadas das características presentes nas imagens de raio-

x, com o intuito de compreender de forma mais precisa como esses traços impactam o desempenho das redes neurais.

5. REFERÊNCIAS BIBLIOGRÁFICAS

[1] ALENCAR, F. (2023). O que são redes neurais artificiais? Hardware. Disponível em: <https://www.hardware.com.br/artigos/o-que-sao-redes-neurais-artificiais/>. Acesso em: 06 março 2023.

[2] BATISTA, M; MARTINS, A. (2022). Redes Neurais no Ensino Básico. Revista Eletrônica da Sociedade Brasileira de Matemática, 10(4), 454-481. DOI: 10.21711/2319023x2022/pmo1032.

[3] DAS, E. (2022). Capítulo 2: Uma Breve História das Redes Neurais Artificiais. Deeplearningbook. Disponível em: <https://www.deeplearningbook.com.br/uma-breve-historia-das-redes-neurais-artificiais/>. Acesso em: 06 março 2023.

[4] DAS, E. (2022). Capítulo 4: O Neurônio, Biológico e Matemático. Deeplearningbook. Disponível em: < https://www.deeplearningbook.com.br/o-neuronio-biologico-e-matematico/>. Acesso em: 06 março 2023.

[5] SILVA, I; SPATTI, D; FLAUZINO, R. (2010). Redes Neurais Artificiais para Engenharia e Ciências Aplicadas. São Paulo, SP: Editora Artliber.

[6] ALENCAR, G; CAMPELLO, R; FAUSTO, A; VASCONCELOS, G; ADEODATO, P. (2018). Redes Neurais. Pernambuco, PE. Disponível em: Reconhecimento de Padrões (ufu.br).

[7] SILVA, Josenildo C; VIEIRA, Raimundo Osvaldo. (2022). Introdução às Redes Neurais Profundas com Python. Capítulo 4, p. 79-98. DOI: 10.5753/sbc.11014.1.

[8] FLECK, Leandro; TAVARES, Maria Hermínia Ferreira; EYNG, Eduardo; HELMANN, Andrieli Cristina; ANDRADE, Minéia Aparecida de Moares. (2016). Redes Neurais Artificiais: Princípios Básicos. Revista Eletrônica Científica Inovação e Tecnologia, 1(13), 47-57. ISSN 2175-1846.

[9] LARANJEIRA, Camila. (2022). Redes Neurais: Deep Learning com PyTorch. Alura. Disponível em: https://www.alura.com.br/conteudo/pln-deep-learning. Acesso em: 25 junho 2023.

[10] OLIVEIRA, Catarina (2020). Radiografia do Tórax. Disponível em: https://www.saudebemestar.pt/pt/exame/imagiologia/radiografia-de-torax/. Acesso em: 31 agosto 2023.

[11] NG, Andrew. Object Detection. 27 nov. 2017. Apresentação de slides. Disponível em: https://x-wei.github.io/notes/Ng_DLMooc_c4wk3.html Acesso em: 16 setembro 2023.

[12] RIZWAN, Muhammad. Convolutional Neural Networks – In a Nut Shell. engMRK. 17 set. 2018. Disponível em: https://engmrk.com/convolutional-neural-network-3/ Acesso em: 16 setembro 2023.

[13] DESHPANDE, Adit. A Beginer's Guide to Understanding Convolutional Neural Networks. Adit Deshpande Blog. 2016. Disponível em: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-ToUnderstanding-Convolutional-Neural-Networks/. Acesso em: 17 setembro 2023.

[14] CECCHETTO, Bernardo; BOTELHO, Viviane Rodrigues. Classificação de nódulos de tireoide em imagens de ultrassonografia utilizando técnicas de machine learning. In: CONGRESSO BRASILEIRO DE ENGENHARIA BIOMÉDICA, 27., 2020, Vitória. Anais [...]. Vitória: Sbeb, 2020. p. 488-493. Disponível em: http://www.sbeb.org.br/site/cbeb/. Acesso em: 15 setembro 2023.

[15] PRIMO.IA. Deep Learning. 2020. Online. Disponível em: http://primo.ai/index.php?title=Deep_Learning. Acesso em: 02 setembro 2023.

[16] MOONEY, Paul. Chest X-Ray Images (Pneumonia). 2018. Disponível em: https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia. Acesso em: 12 julho 2023.