Universidade Estadual de Londrina

GUSTAVO CHICHANOSKI

ALGORITMOS PARA AUXÍLIO NO DIAGNÓSTICO DE DOENÇAS PULMONARES UTILIZANDO INTELIGÊNCIA ARTIFICIAL

Londrina
2022

GUSTAVO CHICHANOSKI

ALGORITMOS PARA AUXÍLIO NO DIAGNÓSTICO DE DOENÇAS PULMONARES UTILIZANDO INTELIGÊNCIA ARTIFICIAL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Estadual de Londrina como parte dos requisitos necessários para a obtenção do Título de Mestre em Engenharia Elétrica.

Orientador: Profa. Dra. Maria Bernadete de Morais França

Londrina
2022

Ficha Catalográfica

Gustavo Chichanoski

ALGORITMOS PARA AUXÍLIO NO DIAGNÓSTICO DE DOENÇAS PULMONARES UTILIZANDO INTELIGÊNCIA ARTIFICIAL - Londrina - PR, 2022 - 128 p., 30 cm.

Orientador: Profa. Dra. Maria Bernadete de Morais França

1. Aprendizado de Máquina. 2.Redes Neurais. 3.Deep Learning. 4. Covid-19. 5.Grad Cam.

I. Universidade Estadual de Londrina. Curso de Engenharia Elétrica. II. ALGORITMOS PARA AUXÍLIO NO DIAGNÓSTICO DE DOENÇAS PULMONARES UTILIZANDO INTELIGÊNCIA ARTIFICIAL.

GUSTAVO CHICHANOSKI

ALGORITMOS PARA AUXÍLIO NO DIAGNÓSTICO DE DOENÇAS PULMONARES UTILIZANDO INTELIGÊNCIA ARTIFICIAL

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Estadual de Londrina como parte dos requisitos necessários para a obtenção do Título de Mestre em Engenharia Elétrica.

BANCA EXAMINADORA

Orientador: Profa. Dra. Maria Bernadete de Morais França
Universidade Estadual de Londrina – UEL

Prof. Dr. Leonimer Flávio de Melo
Universidade Estadual de Londrina – UEL

Prof. Dr. Flávio José de Oliveira Morais
Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP (Campus de Tupã)

Londrina, 29 de setembro de 2022

Dedico este trabalho a todos aqueles
que acreditaram em meu potencial.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Sinceros agradecimentos à equipe de professores e colegas do Laboratório de Automação e Instrumentação do Departamento de Engenharia Elétrica da Universidade Estadual de Londrina (UEL) e demais setores envolvidos.

“Quando as pessoas querem o impossível,
apenas os mentirosos podem satisfaze-las“
(Thomas Sowell)

CHICHANOSKI, GUSTAVO. **ALGORITMOS PARA AUXÍLIO NO DIAGNÓSTICO DE DOENÇAS PULMONARES UTILIZANDO INTELIGÊNCIA ARTIFICIAL**. 2022. 128 p. Dissertação do Programa de Mestrado em Engenharia Elétrica - Universidade Estadual de Londrina, Londrina - PR.

Resumo

O mundo vem sofrendo com o corona vírus desde 2019, quando o primeiro caso foi relatado, desde então foi disseminado por todo o mundo, causando perdas econômicas e humanas. O presente trabalho visa desenvolver um software para o diagnósticos visual de doenças pulmonares utilizando imagens de raio-x como base para o treinamento de redes neurais, visando ser mais esclarecedor da gravidade da doença. Utilizando *datasets* fornecidos gratuitamente pelos governos Americano e Chinês, foi realizado a segmentação do pulmão da imagem de raio-x, realizando o treinamento com recortes das imagens para o treinamento local. Após essa etapa, a rede realiza diversos recortes na imagem passando todas eles pela rede, gerando o vetor de probabilidade da imagem original. Esse processo foi repetido para cada rede. Assim a análise do entendimento da rede foi feita através do Grad-CAM gerado pelos modelos de redes pré treinadas através da transferência de aprendizado ResNet50V2, DenseNet121, InceptionResnetV2 e VGG-19, obtendo desempenho nos parâmetros de precisão e especificidade melhores que a literatura de referência.

Palavras-Chave: 1. Aprendizado de Máquina. 2.Redes Neurais. 3.Deep Learning. 4. Covid-19. 5.Grad Cam.

CHICHANOSKI, GUSTAVO. **SYSTEM TO ASSIST LUNG DISEASES DIAGNOSIS**. 2022. 128 p. Dissertation on a Master's Degree in Electrical Engineering - Londrina State University, Londrina - PR.

Abstract

The world has been suffering from the corona virus since 2019, when the first case was reported, since then it has spread all over the world, causing economic losses and human. The present work aims to develop software for visual diagnostics of lung diseases using x-ray imaging as a basis for training neural networks, aiming to clarify the severity of the disease. Using datasets provided free of charge by the American and Chinese governments, segmentation was performed of the lung of the x-ray image, performing the training with clippings of the images to the local training. After this step, the network performs several clippings in the image, passing all of them through the network, generating the probability vector of the original image. This process was repeated for each network. Thus, the analysis of the understanding of the network is done through of the Grad-CAM generated by the models of ResNet50V2, DenseNet121, InceptionResnetV2 and VGG-19. Achieving performance in the parameters of precision and specificity better than the reference literature.

Key-words: 1. Machine Learning. 2.Neural Network. 3. Deep Learning. 4. COVID-19. 5. Grad Cam.

Lista de Figuras

Lista de Siglas e Abreviaturas

ANN	Redes Neurais Artificias
CNN	Redes Neurais Convolucionais
LSTM	Long Short Term Memory
RRN	Redes Neurais Recorrentes
OMS	*World Health Organization* - Organização Mundial da Saúde

Sumário

1 Introdução

Nos últimos anos, termos ligados a Inteligência Artificial e aprendizado de máquina ganharam notoriedade, do qual o uso não está restrito a ambientes de pesquisa, mas se faz presente em dispositivos e ferramentas do cotidiano das pessoas, sendo amplamente aplicados no setor financeiro, como robôs de compra e venda de ações em corretoras, a classificadores de créditos em bancos.

Nesse sentido a Inteligência Artificial vem sendo aplicado cada vez mais para auxiliar médicos no diagnósticos de doenças, exames médicos e entre outros, sendo uma das aplicações a segmentação de imagens de tomografias computadorizadas, como visto em (CHO; JANG; TAN, 2021) e (YIN et al., 2020).

Porém em dezembro de 2019, o mundo foi surpreendido por uma série de casos de pneumonia severa na cidade de Wuhan na província chinesa de Hubei, causado por um vírus da família dos coronavírus, chamado SARS-CoV2, popularmente conhecido como coronavírus.

Para ajudar no combate ao coronavírus, inúmeras ferramentas estão sendo desenvolvidas, algumas usando a inteligência artificial tanto para prever o número de casos da doença em determinadas regiões, como em (RUSTAM et al., 2020), ou a classificação entre contaminados e não contaminados, como realizado em (RUSTAM et al., 2020).

Na área de classificação, a visão computacional vem fornecendo diversas ferramentas, utilizando de imagens médicas fornecidas por raios-x do tórax e tomografias computadorizadas, estão sendo desenvolvidos *software* de auxílio no diagnóstico de doenças, esses softwares podem realizar a segmentação da doença, separação as partes infectadas do corpo do restante da imagem, e classificação, que determina se o paciente está ou não com a doença do pulmão do paciente.

Diversos artigos foram publicados nesse sentido, como por exemplo os artigos (OH; PARK; YE, 2020) e (TABIK et al., 2020), que serviram de base para este trabalho de dissertação, o artigo (OH; PARK; YE, 2020) descreve um método mais explicativo de como a rede determinou a classe de uma imagem, gerando um mapa de calor do grau de importância de cada pixel para a classe de saída, como por exemplo, se um raio-x é de um paciente com coronavírus ou se de um paciente normal. Enquanto o artigo (TABIK et

al., 2020), descreve um método de classificação da gravidade da doença a categorizando em normal, suave, moderado e severo, através do índice RALE.

Determinar a severidade do Covid-19 é importante pois os pacientes possuem um pequeno intervalo de tempo para procurar ajuda médica em casos graves, visto isso, para determinar o grau de gravidade na Espanha era utilizado o índice RALE, que normalmente é realizado por médicos, cujo valor varia de 0 a 8. O conceito desse método é dividir cada pulmão em 4 partes iguais, se algumas dessas partes conter uma consolidação ou vidro fosco aquela parte adquire o valor de 1, depois somam-se cada parte, se o valor for 0 é normal, 1 a 2 é suave, 3 a 5 é moderado e 5 a 8 é grave.

Este trabalho vem no sentido de criar uma intersecção em ambos os métodos, afim de obter um modelo mais acurada do método descrito em (TABIK et al., 2020), utilizando o método em (OH; PARK; YE, 2020) para gerar as imagens contendo as áreas demarcadas do pulmão a serem mapeadas, assim definindo com maior precisão o índex RALE, comparando o resultando utilizando diferentes tipos de redes pré treinadas através da transferência de aprendizado, como a Resnet, Inception-Resnet, DenseNet e VGG, de modo a obter a melhor acurácia possível.

1.1 Objetivos

O principal objetivo deste trabalho é desenvolver um sistema capaz de ajudar no diagnóstico de doenças pulmonares através de exames de raio-x. Para tanto, foram estabelecidos os seguintes objetivos específicos:

- Testar diferentes redes de inteligência artificial (ResNet50V2, InceptionResnetV2, DenseNet121 e VGG-19);
- Segmentar a imagem original retirando o pulmão;
- Classificar os estados de saúde do paciente como normal, pneumonia ou Covid-19.

1.2 Organização do trabalho

Os capítulos a seguir estão organizados em revisão, capítulo 2, onde será feita uma revisão da situação atual dos métodos utilizados para a segmentação e classificação, apre-

sentando os resultados obtidos na literatura. A fundamentação teórica, capítulo 3, apresenta a fundamentação teórica necessária para o entendimento do trabalho, explicando o funcionamento de redes neurais e a utilidade da segmentação. O capítulo de metodologia, capítulo 4, apresenta a arquitetura da rede neural utilizada e os métodos para a avaliação da rede. Os resultados obtidos foram apresentados no capítulo 5, mostrando os valores obtidos pelas diversas redes e a comparação com a literatura, e por fim o capítulo 6 que conclui o presente trabalho apresentando as conclusões do autor.

2 Revisão Bibliográfica

A seguir será feita uma revisão da atual situação dos métodos utilizados para a segmentação e classificação, apresentando os resultados obtidos na literatura.

2.1 Inteligência Artificial em Imagens Médicas

Os termos relacionados à inteligência artificial ganharam evidência na mídia devido à sua presença em dispositivos como *smartphones*, assistentes virtuais e carros autônomos, onde como conceito geral, os pesquisadores buscam automatizar decisões executadas por humanos. Assim como nas ferramentas de uso cotidiano, a medicina e mais especificamente à área de radiologia, vem passando por grandes mudanças tecnológicas nos últimos anos.

A inserção de sistemas computadorizados e digitalização do ambiente radiológico, no final da década de 1990, possibilitaram o surgimento de sistemas de armazenamento e acesso a internet, tornando possível a disponibilização destas imagens para que outros especialistas ou sistemas computacionais possam analisá-las. Os avanços dessas técnicas de diagnósticos aumentaram o número de informações e consequentemente a complexidade das análises, impossibilitando que um único especialista consiga lidar com todas as possibilidades de diagnóstico.

Desta forma, a implementação das técnicas de Inteligência Artificial na radiologia contribuiu para o aumento da precisão e acurácia dos exames médicos, diminuindo o tempo de espera por resultados, integrando os dados clínicos em grande banco de dados, dando origem às ferramentas de auxílios em diagnóstico.

2.1.1 COVID-19 e a Inteligência Artificial

O COVID-19 é uma doença causada por uma síndrome respiratória aguda grave, transmitida pelo vírus nomeado em (ORGANIZATION, 2020a), como *Severe Acute Respiratory Syndrome CoronaVirus 2* (SAR-CoV-2), que emergiu em Dezembro de 2019 e rapidamente evoluiu para uma pandemia (ORGANIZATION, 2020b). O COVID-19 se apresenta como uma síndrome do trato respiratório e é altamente infeccioso.

Como descrito em (De Oliveira Lima, 2020) e (DONG et al., 2021), os sintomas típicos do COVID-19 são caracterizados pela febre, tosse, falta de ar, dores musculares, confusão mental, dores de cabeça, dor de garganta, rinorréia, dor no peito, diarréia, náuseas, vômitos, pneumonia, diminuição da contagem de células brancas ou a diminuição de linfócitos.

Segundo (SHI et al., 2021), existem duas formas para realizar o testes clínicos para a detecção do COVID-19.

1. **Testes moleculares**: recomendados pela Organização Mundial da Saúde, é o teste mais comum para a detecção de uma infecção de SARS-CoV-2. O exame consiste na análise de uma amostra retirada da nasofaringe, para verificar a presença dos genes do vírus. Apesar do diagnóstico ter uma alta sensibilidade, o mesmo não detecta se houve a infecção recentemente, podendo resultar em falsos negativos.

2. **Testes Serológicos**: diferentes de testes moleculares, estes não detectam a presença do vírus, mas a existência dos anticorpos formados pelo corpo para combater a doença, esses testes são usados para detectar pessoas que foram contaminados recentemente do COVID-19, mas não detecta a presença do vírus.

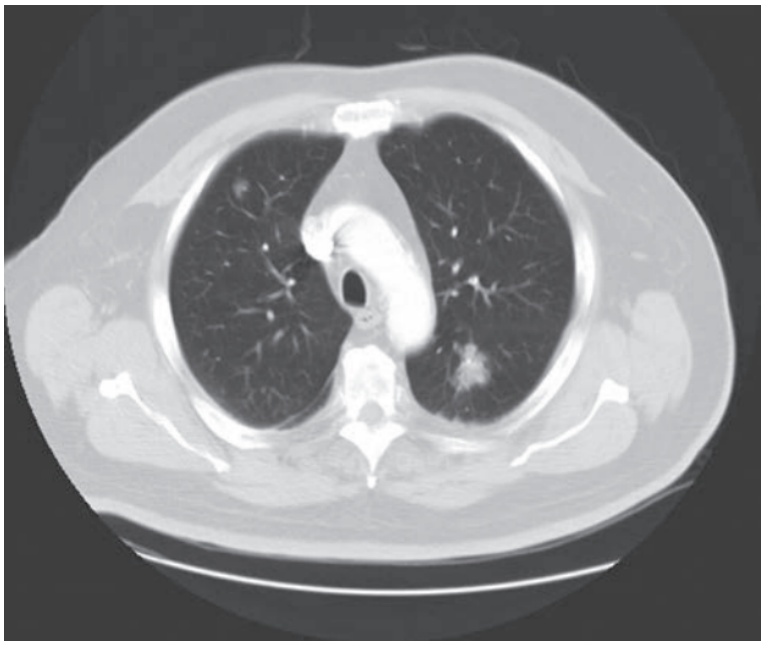
Caso confirmado o COVID-19, os pacientes podem ser classificados em quatro categorias, leve, moderado, grave e crítico.

- **Leve**: sintomas leves e sem sinais de pneumonia na radiologia do tórax.
- **Moderado**: com febre e sintomas respiratórios, e com sinais de pneumonia na radiologia do tórax.
- **Grave**: dificuldade respiratórias, saturação de oxigênio e pressão arterial parcial do oxigênio, e um adulto que apresente uma imagem do tórax que mostre uma obvia progressão lesão $> 50\%$ em 24 a 48 horas.
- **Crítico**: falha respiratória, requerendo ventilação mecânica.

O diagnóstico da COVID-19 também pode ser realizado a partir de análises de imagens do pulmão do paciente, descrito por (SHI et al., 2021), pois pulmões com a doença podem apresentar lesões do tipo vidro fosco, ou *Ground-Glass Opacities* (GGO), consolidação do pulmão, sombreamento bilateral irregular, fibrose pulmonar, múltiplas lesões e padrões

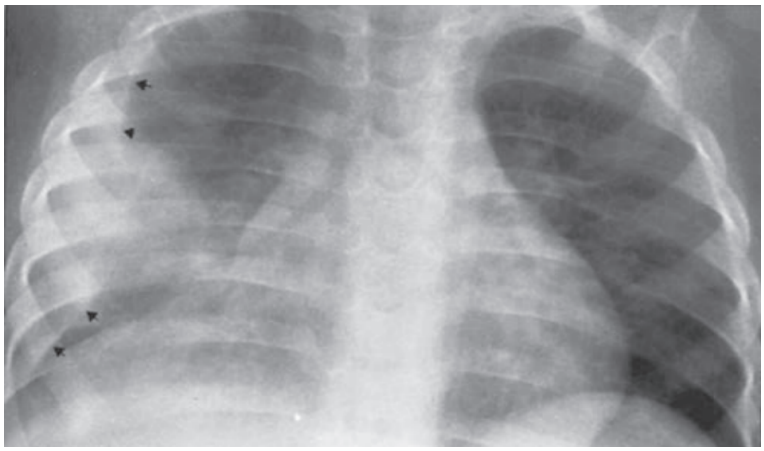
de *crazy-paving*. Estes tipos de lesões são mostradas nas figuras 1, 2, 3 e 4, apresentadas por (EISENBERG, 2010).

Figura 1 – Imagem de uma tomografia computadorizada com GGO.



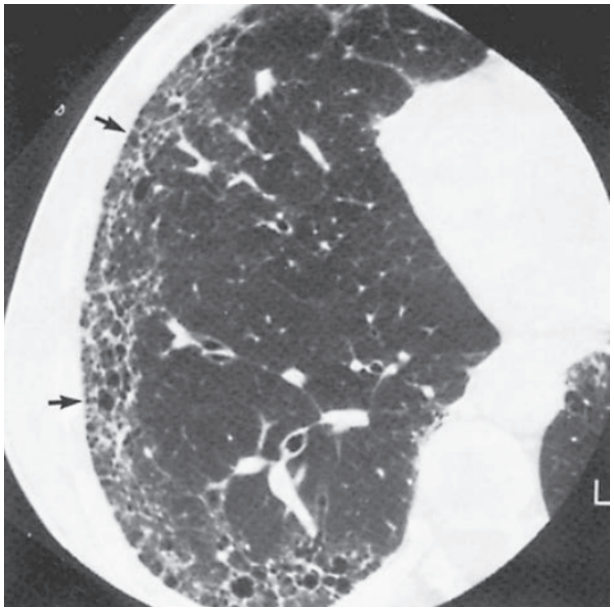
Fonte: (EISENBERG, 2010).

Figura 2 – Imagem de raio-x com consolidação do pulmão.



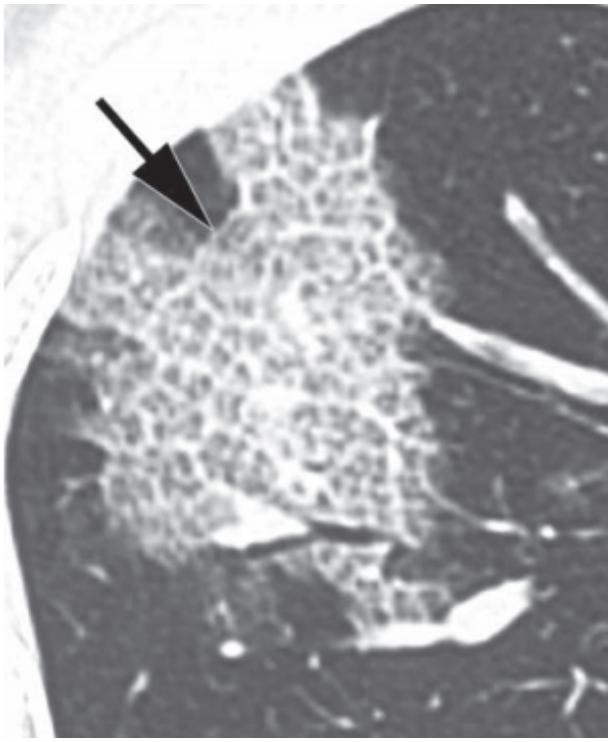
Fonte: (EISENBERG, 2010).

Figura 3 – Imagem de uma tomografia computadorizada com fibrose pulmonar.



Fonte: (EISENBERG, 2010).

Figura 4 – Imagem de uma tomografia computadorizada com *crazy-paving*.



Fonte: (EISENBERG, 2010).

Esses sintomas se apresentam em três estágios, inicial, progressão e forte. No estágio inicial da doença, os pulmões apresentam múltiplas pequenas sombras irregulares e alterações intersticiais, mais aparentes em zonas periféricas dos pulmões. No estágio de

progressão apresentam múltiplos vidro foscos opacos e infiltrações em ambos os pulmões e no estágio forte, há a consolidação dos pulmões.

As imagens do pulmão do paciente podem ser obtidas por 4 diferentes exames, como dito por (BALAS; MISHRA; KUMAR, 2021) e listadas a seguir:

- **Radiografia do tórax**: exame de raio-x que cria uma imagem das estruturas internas do corpo, utilizando uma pequena quantidade de radiação. Tende a expor mais os ossos do que o pulmão.
- **Tomográfica computadorizada**: utiliza a radiação X para visualizar pequenas fatias de regiões do corpo, por meio de rotação do tubo emissor de Raio X ao redor do paciente.
- **Ressonância Magnética**: utiliza ondas eletromagnéticas para a formação das imagens. A ressonância magnética é imprecisa quando usada para uma imagem de estruturas que estão em movimento, como os pulmões,
- **Tomografia por emissão de pósitrons**: emissão de pósitrons para medir a variação nos processos bioquímicos. Muitas vezes é combinado utilizando em conjunto com a tomografia computadorizada, o que permite uma imagem mais detalhadas da área na tomografia computadorizada.

Devido a esses fatores, o diagnósticos do COVID-19 via imagens se divergiu em dois tipos: radiografias do tórax, como feito em (NARIN; KAYA; PAMUK, 2020), e tomografias computadorizadas, *Computed Tomography* ou CT, como complementos de exames em diagnósticos e classificação dos pacientes entre moderados e graves, utilizado em (SHI et al., 2021).

2.2 Redes Neurais

2.2.1 Trabalhos Relacionados

Dentre os trabalhos realizados sobre o diagnóstico de Covid-19 utilizando imagens CT do tórax, se destaca o trabalho (WANG; LIN; WONG, 2020), cujo o objetivo foi produzir uma design de redes neurais convolucionais profundas denominado COVID-Net, obtendo uma acurácia de 93,3%, utilizando 11,75 milhões de parâmetros e com uma sensibilidade

ao COVID-19 de 91%, citado por (OH; PARK; YE, 2020) como um dos métodos mais bem sucedido no diagnóstico. Já (WANG; LIN; WONG, 2020) utilizou o *dataset* COVIDx, dividido entre três categorias, Normal (sem infecções), Pneumonia e COVID-19, o trabalho foi disponibilizado em <https://github.com/lindawangg/COVID-Net>.

Outro artigo a propor uma nova abordagem de diagnósticos foi o (OH; PARK; YE, 2020), utilizando de uma (FC)-DenseNet103 para realizar a segmentação do pulmão e a rede Resnet-18 para realizar a classificação, disponibilizado em <https://github.com/jongcye/Deep-Learning-COVID-19-on-CXR-using-Limited-Training-Data-Sets>.

A proposta do artigo, é realizar o treinamento dos pesos da rede ResNet-18 utilizando apenas um pacote da imagem real randômica selecionado da imagem, sendo que esse pacote não pode ser totalmente zero ou preto. Enquanto o teste é feito retirando K recortes da imagem, também não pode ser totalmente zero, passando cada pacote pela imagem rede treinada e definindo o vencedor baseado no maior número de votos de cada classe. Obtendo como resultado uma sensibilidade de 100% para COVID-19 e uma precisão de 76,9%.

2.2.2 Segmentação da imagem do pulmão

Para reduzir a complexidade da inteligência artificial a ser utilizada, a segmentação de uma imagem é uma etapa essencial. Ela delimita as regiões de interesse, *Regions Of Interest* ou (ROIs), da sua imagem, como os pulmões, lóbulos, bronco pulmonares e regiões infectadas ou lesionadas, diminuindo o números de dados de desinteresse para a inteligência artificial. A segmentação das ROI pode ser feita manualmente ou através do auxílio de outros algoritmos, descritos em diversos artigos, como em (ZHENG et al., 2020).

Existem diversos algoritmos para realizar a segmentação do pulmão, porém os mais usados se baseiam em redes neurais para segmentar uma imagem. Para a criação de uma rede neural existem diversos modelos a ser escolhido, e entre eles o modelo do UNet foi criado especificamente para a segmentação de imagens médicas. No artigo (RONNEBERGER; FISCHER; BROX, 2015) é apresentado o modelo sugerido pelo autor, o qual é utilizado em diversos artigos de identificação médica como mostrado em (SHI et al., 2021).

Como mostrado em (RONNEBERGER; FISCHER; BROX, 2015) ou (CAO et al.,

2020), esses modelos geram máscara de valor de saída binário da imagem de entrada da rede neural, sendo necessária uma multiplicação entre as imagens para isolar o pulmão do resto da imagem, como mostrado na Figura 5.

Figura 5 – Exemplo de uma imagem de pulmão. (a) Imagem do pulmão, (b) máscara do pulmão e (c) imagem segmentada do pulmão.

(a) **(b)** **(c)**

Fonte : (GORDIENKO et al., 2019)

Após a segmentação dos pulmões, é necessário escolher um método de inteligência artificial para realizar o treinamento e por fim o processamento da nova imagem a ser analisada. Geralmente para esses casos é escolhido uma rede neural, que como mostrado por (RONNEBERGER; FISCHER; BROX, 2015), conseguem obter uma acurácia acima dos 90%.

2.2.3 Classificação da doença

Segundo (ZHAO et al., 2020), nas 101 imagens de raios-x coletadas em 4 instituições em Hunan, na China, de pacientes entre 21 a 50 anos, divididas em não emergenciais (87), que corresponde as classificadas entre suaves ou comuns, e emergenciais (14), que são os casos severas ou fatais.

A maioria dos pacientes possuía GGO, (87 dos casos totais que corresponde a 86,1%), ou uma mistura de GGO com consolidação do pulmão, (65 dos casos totais que corresponde a 64,4%). Todavia quando se olhava exclusivamente para pacientes emergenciais (14 casos), o número de pacientes que apresentavam GGO chegava a 14 (100%), a consolidação subia de 41,4% em casos não emergenciais, para 57,1% em casos emergenciais, enquanto a mistura de ambos se manteve constante em 64,4% ambos os tipos de casos.

O estudo ainda relata que pacientes com Covid confirmado, produzem imagens com características típicas, que podem ser úteis no rastreamento e na avaliação da severidade da doença. Já o artigo (De Oliveira Lima, 2020), relata que dos 121 pacientes analisados, 94 (78%) já apresentava GGO nos raios-x realizados nos estágios iniciais da doença, dando ainda mais respaldo na utilização desse tipo de análise.

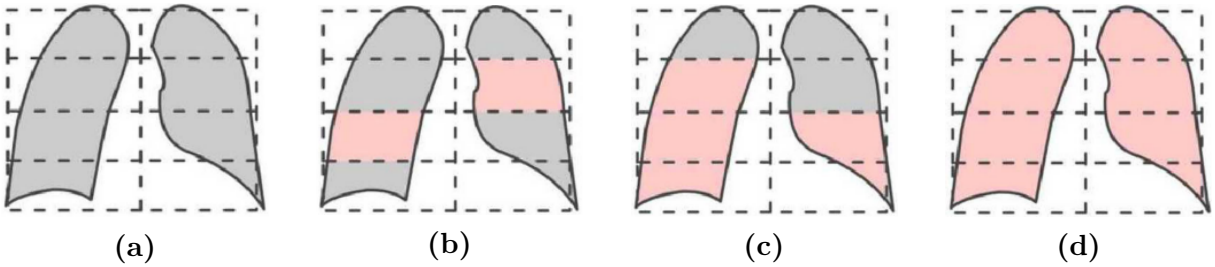
2.2.4 Classificação da Severidade

A classificação da severidade da doença tem como objetivo auxiliar a equipe médica a tomar a melhor decisão no tratamento do paciente em questão. Com esse objetivo diversos artigos propuseram novos métodos de classificação utilizando desde métodos já conhecidos anteriormente na literatura, como o índice RALE apresentado na seção seguinte, a novos métodos desenvolvidos especificamente para redes neurais, como o *Brixia Score*.

No artigo (TABIK et al., 2020), ele sugere outra maneira de classificar o nível da doença, utilizando-se do índice RALE. Segundo (ZIMATORE et al., 2021), o índice RALE, avaliação radiográfica de edema pulmonar ou *Radiographic Assessment of Lung Edema*, é um método proposto para avaliar a extensão e densidade da opacidade alveolar em radiografias do tórax de pacientes com síndrome de sofrimento respiratório agudo (*Acute Respiratory Distress Syndrome* ou ARDS), estudo o qual tinha como objetivo determinar a acurácia do método para diagnóstico e prognóstico de ARDS, sugerindo que o método possui excelente acurácia no seu diagnóstico de ARDS.

Nesse método dividi-se cada pulmão em quatro partes, atribuindo a cada pulmão uma nota de 0 a 4, dependendo da extensão visual de características como consolidação e GGO. Baseado na soma dessas pontuações identifica-se o grau da doença, sendo 0 normal, Figura 6(a), 1 a 2 suave, Figura 6(b), 3 a 5 moderado, Figura 6(c), e de 6 a 8 severo, Figura 6(d). Esse método era normalmente aplicado por radiologistas especializados, porém (TABIK et al., 2020) sugere um novo método.

Figura 6 – Exemplo de como o índice RALE é calculado. (a) Índice RALE: 0, Normal. (b) Índice RALE: 2, Suave. (c) Índice RALE: 5, Moderado. (d) Índice RALE: 8, Severo.

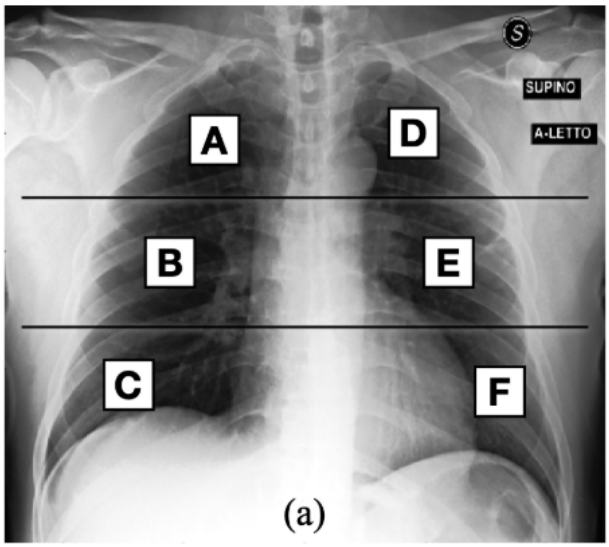


Fonte: (TABIK et al., 2020).

Neste novo método proposto utiliza redes neurais para realizar o treinamento da rede para a classificação do pulmão como Normal, Pneumonia e Covid-19. Então, utilizariam o método Grad-CAM para gerar o mapa de calor do Covid-19 na imagem, que mostraria os locais que mais auxiliaram na identificação da doença, e assim poderem utilizar a imagem para definir onde o Covid-19 se encontra e calcular o índice RALE.

O artigo (SIGNORONI et al., 2021), sugere a criação de uma rede neural afim de analisar a gravidade da doença, para isso ele utiliza do método *Brixia Score*. Similar ao índice de RALE, *Brixia Score* separa o pulmão em 6 partes, A, B, C, D, E e F, como mostrado na Figura 7, porém a análise não é binária.

Figura 7 – Separacão do pulmão pelo método de *Brixia Score*.



Fonte: (SIGNORONI et al., 2021).

Os radiologistas determinam o valor de cada região, que podem assumir valores inteiros entre 0 e 3, sendo 0 para um pulmão normal, 1 para infiltrações intersticial, 2 para

infiltrações intersticial e alvéolos, intersticial dominante, e 3 para infiltrações intersticial e nos alvéolos e alvéolos dominantes. A soma de todas as partes constituem a Pontuação Global, *Global Score*, com valores entre 0 e 18.

3 Fundamentação Teórica

Neste capítulo será apresentado a teoria necessária para a compressão do presente trabalho, apresentando noções sobre inteligência artificial, visão computacional e aprendizado profundo, e uma breve explicação do funcionamento de redes neurais artificiais (Arificial Neural Network ou ANN) e redes neurais convolucionais (Convolutional Neural Network ou CNN) utilizadas.

3.1 Inteligência Artificial

Segundo (RUSSEL, 1995), Inteligência Artificial, *Artificial Intelligence* ou IA, é uma área nova. Ela foi formalmente iniciada em 1956, quando o nome foi cunhado, atualmente a IA abrange diversos subcampos, com propostas bem diversas.

Sistemas IA podem ser definidos em quatro categorias, sistemas que pensam como humanos, sistemas que pensam racionalmente, sistemas que agem como humanos e sistemas que agem racionalmente. As definições a esquerda medem o sucesso em relação a performance humana, enquanto as definições a direita medem o sucesso em comparação um conceito ideal de inteligência. Um sistema é racional, quando a máquina realiza a coisa certa. As duas últimas linhas sintetizam o pensamento da época.

Sendo que a abordagem centrada no homem deve ser uma ciência empírica, envolvendo confirmações hipotéticas e experimentais, enquanto a abordagem racional envolve uma combinação de matemática e engenharia.

3.2 Visão Computacional

Visão computacional é uma das diversas áreas de inteligência artificial. Segundo (HUANG, 1997), visão computacional possui duplo objetivo. Do ponto de vista da ciência biológica, a visão computacional visa criar modelos computacionais da visão humana. Do ponto de vista da engenharia, a visão computacional visa construir sistemas autônomos que possam performar tarefas que precisem da visão humana.

Seu campo de atuação varia desde o processamento de imagem, reduzindo a comple-

xidade da imagem crua para outro sistema, até a fotogrametria, que gera modelos em 3D precisos através de câmeras.

Com o advento dos processadores cada vez mais potentes, pode-se utilizar outras áreas da inteligência artificial, como *Deep Learning*, que permitiu o desenvolvimentos de diversas novas aplicações, como direção autônoma, reconhecimento de produtos e mais recentemente a detecção da utilização de máscaras.

3.3 Aprendizado Profundo

Aprendizado profundo, ou *Deep Learning*, é uma subárea de inteligência artificial que foi originalmente projetado para modelar a visão biológica e o processamento do cérebro.

Apesar de não existir uma definição exata de quantas camadas ocultas torna um modelo uma *Deep Learning*, é dito que acima de duas camadas ocultas já são classificadas em *Deep Learning*. Essas camadas podem ser formadas de ANN, CNN, LSTM, RRN, Hopfield e entre outras.

A primeira arquitetura de rede a resolver o *dataset* ImageNet2012 de aprendizado profundo é relatado em (GONZALEZ, 2007), quando Alex Krizhevsky desenvolveu o modelo de redes neurais profundas aplicado no reconhecimento de imagens, denominado AlexNet.

AlexNet continha 8 camadas, as cinco primeiras camadas eram formadas por CNN seguidas por uma *max pooling*, e três camadas densas conectadas, podendo classificar mais de 1.000 tipos de classes, com uma imagem de entrada de apenas 224x224 pixeis.

AlexNet abriu a possibilidade de se utilizar modelos de aprendizado de máquinas na identificação de bancos de dados enormes, com inúmeras categorias diferentes, mostrando o poder dessa tecnologia.

3.4 Redes Neurais Artificiais

Redes Neurais Artificias, *Artificial Neural Network* ou ANN, foram primeiramente definidas em (ROSENBLATT, 1958), seu funcionamento foi baseado no cérebro humano, desenvolvendo o primeiro modelo de *perceptron*, que tentava explicar o funcionamento da visão humana.

Uma rede neural artificial é composta por neurônios, esses neurônios são organizados em camadas de entrada, ocultas ou intermediárias e de saída. Os valores de entrada dos neurônios passam pela operação da camada e resultam em um valor de saída.

Caso todas as saídas de todos os neurônios da camada anterior sejam valores de entrada da próxima camada é dito que essa é uma camada Densa. Para que a camada possa resultar em valores lineares é adicionado mais um neurônio com entrada de 1, esse neurônio adicional é chamado de *bias*.

3.5 Redes Neurais Convolucionais

Uma rede neural convolucional, *Convolutional Neural Network* ou CNN, se comporta de maneira semelhante as ANN, os neurônios executam a operação, que no caso de CNN é uma convolução, em relação a entrada, os pesos são substituídos por filtros. Suas aplicações variam entre processamento de imagens até processamento de voz.

Segundo (ALBAWI; MOHAMMED; AL-ZAWI, 2018), o maior benefício de uma CNN em relação a uma ANN é a redução do número de parâmetros necessários no modelo, permitindo desenvolver modelos para resolver tarefas mais complexas.

Visto que para resolver uma imagem com dimensões de 28x28x1 na ANN, será necessários 784 pesos, todavia caso a entrada se expanda para 56x56x1, seriam necessários 3.136 pesos, enquanto na mesma situação os filtros da CNN permaneceriam com o mesmo tamanho.

3.5.1 Convolução

A convolução é a representação matemática de como um sistema linear opera sobre um sinal. A convolução já era muita aplicada em processamento de sinais e imagens, desde filtros passa-baixa ao filtro gaussiano, sendo usada para remoção ou aplicação de bordas, redução de brilho e outras tarefas, que auxiliavam no tratamento de imagens.

Assim uma interpretação que pode ser realizada sobre o funcionamento dos neurônios em uma CNN, seria que cada neurônio aplica um filtro diferente na sua entrada, levando muitos autores a dizerem que não é necessário a aplicação de uma filtragem em imagens inserida na CNN visto que a própria CNN definiria o melhor filtro.

Logo, o que uma CNN faz é aplicar diversos filtros na imagem, calculados através de um processo automatizado, a fim de obter o resultado semelhante as imagens de treinamento. A equação matemática da convolução pode ser apresentada em duas formas, sua forma contínua mostrada em:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau) \tag{3.1}$$

Em sua forma discreta mostrada em:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[k-n] \tag{3.2}$$

A convolução em computadores é calculada através de sua forma discreta, devido ao fato de computadores fazerem amostragem entre intervalos de tempos. Todavia, convoluções em duas dimensões podem ser realizadas de maneira mais simples. Suponha que se deseja fazer a convolução de um filtro $K \in \mathbb{R}^{FxF}$, mostrado na equação:

$$K = \begin{bmatrix} 1 & 4 & 8 \\ 4 & 11 & 15 \\ 1 & 8 & 17 \end{bmatrix} \tag{3.3}$$

Sobre uma imagem $I \in \mathbb{R}^{NxN}$ mostrado na equação:

$$I = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 3 & 4 & 0 \\ 1 & 1 & 3 & 2 & 0 \\ 0 & 0 & 4 & 2 & 6 \\ 1 & 0 & 7 & 8 & 0 \end{bmatrix} \tag{3.4}$$

A dimensão de saída é definida por:

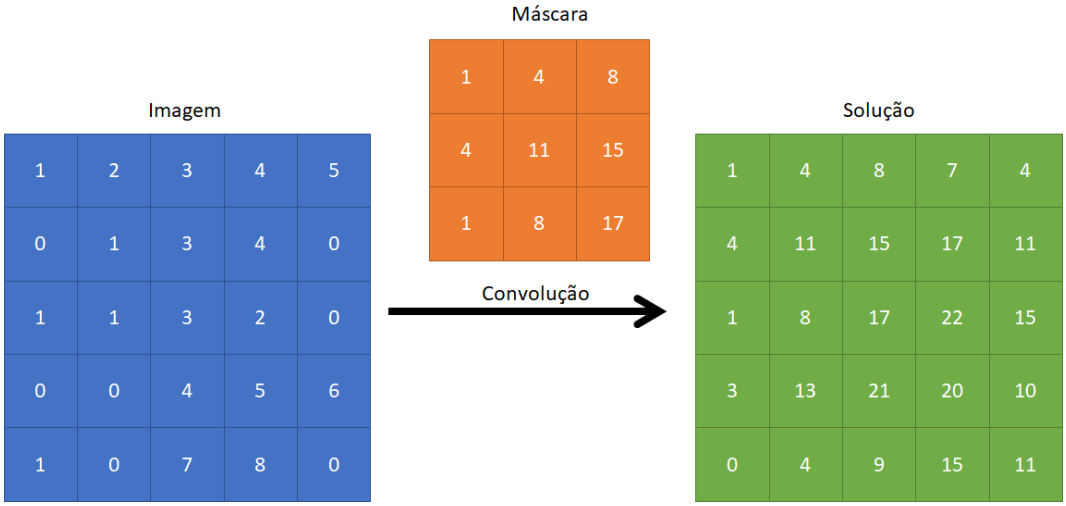
$$G = \frac{N-F}{S} + 1 \tag{3.5}$$

Onde G é o valor de saída da imagem, N é o tamanho da imagem, S é o passo da convolução e F é a dimensão do filtro.

Para realizar a convolução, bastaria realizar o produto Hadamart entre o filtro e a imagem, e somar os valores da matriz resultante, assim teríamos o resultado da primeira

posição da matriz de convolução, 1, agora bastaria realizar o cálculo para as demais posições e obtém-se a matriz de saída desejada.

Figura 8 – Convolução entre a imagem, o filtro ao centro e a matriz resultante.



Fonte: Autor.

Nota-se que ocorreu uma redução na ordem da matriz de saída se comparada a entrada, pois o filtro deve multiplicar apenas áreas validas da matriz de entrada, o que não ocorreria nos cantos da matriz. Para corrigir esse problema pode-se adicionar zeros a matriz de entrada para que as dimensões da matriz de saída e entrada sejam as mesmas, a fim de evitar alterações nas dimensões de saída e entrada, esse método é chamado de *padding*.

Para calcular quantos zeros devem ser inseridos nas bordas da matriz de entrada usa-se a equação:

$$P = \frac{F - 1}{2} \tag{3.6}$$

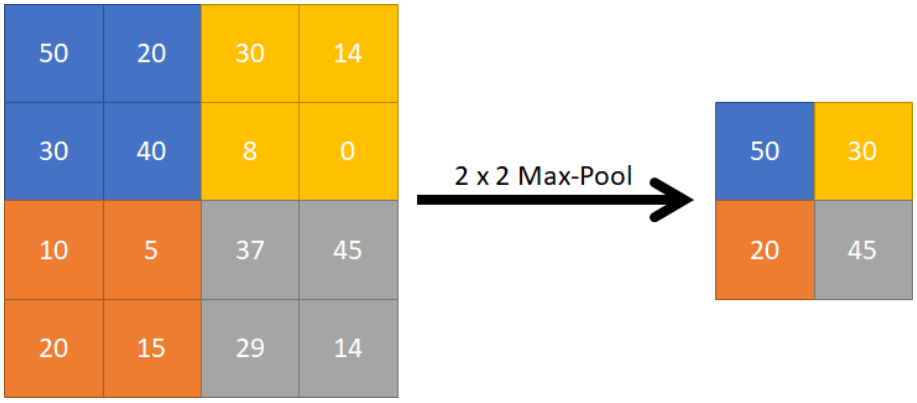
Onde P é o número de zeros a ser inseridos, *pad* ou P, e F é a ordem do filtro da convolução.

Assim a dimensão da matriz de saída será dada pela equação:

$$G = \frac{N + 2P - F}{S} + 1 \tag{3.7}$$

Onde G é a ordem da matriz de saída, N é a ordem da matriz de entrada, P é a ordem do *pad*, k é a ordem do filtro e S é o passo da convolução.

Figura 9 – Exemplo da camada *max pooling*, com janela 2x2 e um passo de 1.



Fonte: Autor.

3.6 Camadas Pooling

Pooling é o processo de realizar a redução de amostragem, *downsampling*, da imagem das suas dimensões espaciais (altura e largura), para isso uma janela de tamanho P passará pela entrada, com um passo s através de cada dimensão. A janela poderá executar dois tipos de operações.

- MaxPooling: quando apenas o maior valor na janela de entrada permanece na saída.
- AveragePooling: quando apenas a média dos valores de entrada permanece na saída.

O cálculo do tamanho da saída pode ser feito por:

$$o = int\left(\frac{i-p}{s}\right) + 1 \tag{3.8}$$

Onde o é o tamanho de saída, i é o valor de entrada, p é o tamanho da janela e s é o passo da janela, e int é o operador de arredondamento para baixo da equação.

Um exemplo de *pooling* pode ser visto na Figura 9, tendo uma matriz de entrada com tamanho 5x5, uma janela de tamanho 2x2 e um passo de 1. Se realizado a equação (3.8) verifica-se que a saída deveria ser de uma matriz 4x4, como calculado em:

$$o = int\left(\frac{5-2}{1}\right) + 1 = 4 \tag{3.9}$$

3.7 Funções de Ativações

Funções de ativações é a transformada linear ou não linear realizada no sinal de entrada. Quando não se tem uma função de ativação, é dito que a função de ativação é uma função Linear, pois a saída é igual a entrada.

Funções de ativações se tornaram importantes para resolver o problema da XOR. Redes neurais lineares, conseguem com resolver problemas de AND e OR, porém não conseguem implementar uma função não-linear como XOR, necessitando adicionar funções não lineares a rede para resolver esse tipo de problema, como a sigmoide, *softmax*, ReLU e entre outros.

3.7.1 Sigmoide

Sigmoide, ou *Logistic*, é a função de ativação que implementa a sigmoide com uma entrada x, cuja equação pode ser vista a seguir:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.10}$$

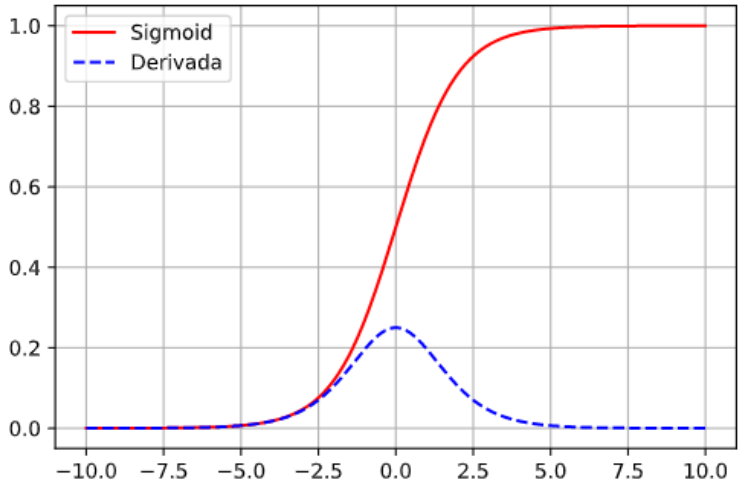
E sua derivada é dada pela equação:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{3.11}$$

Seus valores variam entre 0 e 1. Segundo (GÉRON, 2017) relata, normalmente para a classificação assume-se que para valores após a sigmoide maiores que 0,5 seu resultado é 1, e para valores menores que 0,5 seu resultado é 0.

A Figura 10 apresenta a curva característica da função *sigmoid*, a linha vermelha sólida representa a curva direta, enquanto a curva em azul tracejado representa a curva da derivada da sigmoide.

Figura 10 – Curva característica para a função sigmoide. O eixo x representa a entrada da camada e o eixo y representa a saída da função.



Fonte: Autor.

3.7.2 ReLU

ReLu, *Rectified Linear Units* ou ReLU, foi proposta no artigo (AGARAP, 2018), seu funcionamento baseia-se em verificar se a entrada é maior que 0, se for verdade ela é copiada para saída, caso contrário, a saída será 0, ou seja, $f(x) = max(0, x_i)$, onde i representa o neurônio a ser analisado em ANN, ou *feature* em CNN. Este funcionamento é semelhante a um diodo ideal e a equação representada por:

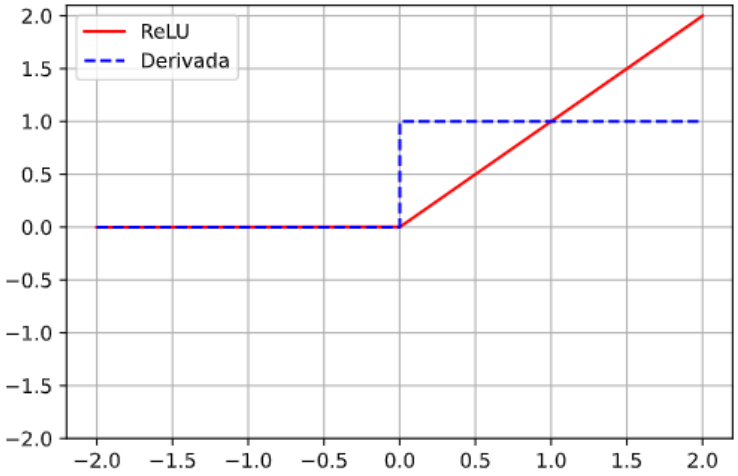
$$\sigma(x) = max(0, x) \tag{3.12}$$

O cálculo do gradiente necessita da derivada da função, que semelhante a função direta possui equacionamento extremamente simples, onde para valores menores que 0 a saída é 0 e para valores maiores de 0 a saída é o próprio valor, porém a derivada em zero não está definida em 0, mas considera-se o valor de 1, como mostrado em:

$$\sigma'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & c.c \end{cases}. \tag{3.13}$$

Logo sua curva características pode ser vista na Figura 11, sendo em vermelho sólido é a sua forma direta e em azul tracejado a sua forma derivada.

Figura 11 – Curva característica da função de ativação ReLU. O eixo x representa a entrada da função e o eixo y representa a saída da função.



Fonte: Autor

Apesar de sua simplicidade, ela vem sendo aplicada em diversas arquiteturas, por adicionar uma complexidade não-linear na rede e ainda não exigir muito processamento em seu cálculo, entretanto como as saídas tendem a irem a zero, ela acaba por impedir o treinamento de alguns neurônios.

3.7.3 Softmax

A *Softmax*, função de normalização exponencial, é uma generalização da função sigmoide. Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), a função *softmax* é uma das mais estáveis contra erros de *overflow* e *underflow*, normalmente utilizada na camada de predição, sua equação pode ser vista a seguir:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{3.14}$$

Onde x representa o vetor de entrada da função enquanto i representa o valor atual do vetor de entrada. Essa função é normalmente utilizada na classificação, quando só um resultado de saída é válido.

Se todas as entradas forem iguais a uma constante c, a saída será $\frac{1}{n}$, onde n é o número de termos de x. Porém caso c seja um valor de alta magnitude negativa, o valor da *softmax* tenderá a infinito ($\pm\infty$), no outro extremo oposto, caso o valor de c seja de alta magnitude positiva a função tenderá a mais infinito. Ambos os casos podem ser resolvidos aplicando uma substituição de x por $z = x - max_i x_i$, assim quando for o maior

de x, ele será subtraído por si mesmo resultando em 0, resultando em pelo um termo no denominador sendo 1. Porém essa causa mais um problema, caso os valores de x sejam muito pequenos pode resultar em uma saída de 0. Esse problema pode ser corrigido adicionando um logaritmo após o cálculo da *softmax*, como sugere (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.8 Funções de Custo

Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016), funções cujo o objetivo seja maximizar ou minimizar são chamadas funções objetiva ou funções de critério. Quando seu objetivo é minimizar, ela é definida como uma função custo, função erro ou função de perda.

O objetivo de uma função de custo em redes neurais é determinar a distância entre os valores estimados e o valor real. Podem ser divididos em dois grupos, funções de regressões, onde a saída assume qualquer valor real, e de classificações, onde saída assume valores entre 0 e 1, neste trabalho será abordado apenas funções de classificações.

3.8.1 *Dice Loss*

O coeficiente Dice é amplamente usado como métrica na visão computacional para calcular a similaridade entre duas imagens, devido a sua eficiência ele foi adaptado em (SUDRE et al., 2017) para poder ser usado como uma função de custo. Sua equação é descrita por:

$$DL(y, \hat{p}) = 1 - \frac{2y\hat{p} + 1}{y + \hat{p} + 1} \tag{3.15}$$

Onde $\hat{p}$ representa os valores preditos pelo modelo e y os valores reais da saída.

A adição dos valores 1 no denominador tem como objetivo evitar situações em que $\hat{p}$ e y sejam 0, fazendo a função ser indefinida, o que é compensado pela adição do 1 no numerador acima para os demais valores.

3.8.2 Log-Cosh *Dice Loss*

Segundo (JADON, 2020), apesar da grande utilidade do *Dice Loss*, descrito na secção 3.8.1, ele se comporta como uma função não convexa, o que pode prejudicar no treina-

mento para encontrar os resultados ótimos, para isso ele sugere uma correção a equação (3.15) adicionando a extensão de Lovsz. Essa adição é representada na equação a seguir:

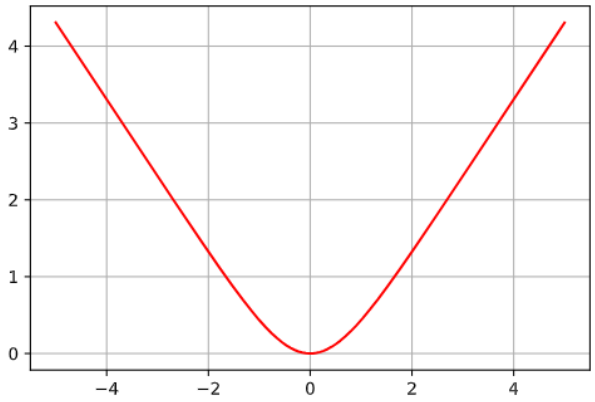
$$\sigma(x) = \log\left(\frac{e^x + e^{-x}}{2}\right) \tag{3.16}$$

Onde a entrada x representa o valor de saída da função *Dice Loss*, e a curva característica dessa extensão é mostrada na Figura 12.

Assim a equação total é obtida por:

$$DL = \log\left(\cosh\left(1 - \frac{2y\hat{p} + 1}{y + \hat{p} + 1}\right)\right) \tag{3.17}$$

Figura 12 – Curva características de Log-Cosh.



Fonte: Autor.

3.9 Otimizadores

Segundo (KINGMA; BA, 2015), otimização de gradiente estocástico é uma pratica muito importante em muitas áreas da ciência e engenharia, onde se precisa realizar a minimização ou maximização de determinados parâmetros, como por exemplo uma função custo, onde se precisa minimizar os pesos de uma rede.

3.9.1 Adam

Descrito em (KINGMA; BA, 2015), o otimizador Adam, estimativa de momento adaptativo ou *Adaptive Moment Estimation*, é um método do gradiente descendente estocástico

baseado em estimadores de momento de primeira ou segunda ordem. Sua equação pode ser vista a seguir:

$$v_t = (1 - \beta_2^p) \sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p \tag{3.18}$$

Onde t é o passo e β_2 é constante igual a 0,999.

3.9.2 AdaMax

Descrito em (KINGMA; BA, 2015), o AdaMax é uma variação do algoritmo do Adam, baseado na norma do infinito. No Adam, a regra de atualização para pesos individuais é para escalar seus gradientes inversamente proporcional a seus gradientes atuais e passados na norma L^2. Generalizando a regra de atualização de L^2 para L^p, onde $p \to \infty$, surge um algoritmo mais simples e estável que o Adam.

Assim, fazendo $u_t = \lim_{p\to\infty}(v_t)^{1/p}$, obtém-se a equação:

$$u_t = max\left(\beta_2 \cdot u_{t-1}, |g_t|\right) \tag{3.19}$$

Onde v_t é a equação (3.18). A equação do AdaMax é mostrada em (3.19), onde t é o passo, g é o gradiente e β_2 é uma constante igual a 0,999, cujo valor inicial é $u_0 = 0$.

3.10 Modelos de redes Pré Treinadas

3.10.1 ResNet50V2

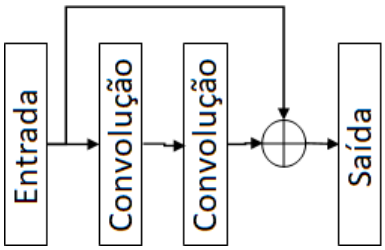
A ResNet50 apresentado em (HE et al., 2016), desenvolvida pela Microsoft, ela foi treinada utilizando as imagens da ImageNet2012, que consistia em 1.000 classes de imagens diferentes e 1,43 milhões de imagens, sendo divididas em 1.28 milhões de imagens para treinamento, 50 mil imagens para validação e 100 mil para teste, tendo respectivamente uma proporção aproximada de 89,5%, 7% e 3,5%, fornecido gratuitamente pela Google.

Utilizando-se de transferência de aprendizado para aproveitar dos benefícios proporcionados pela rede já treinada para acelerar o aprendizado, foi trocada a camada de saída, de 1.000 categorias, por outra camada de saída contendo apenas 3 categorias de saída.

As ResNet são redes neurais residuais, que são semelhantes a redes neurais planas, como a VGG, exceto que é adicionado uma rede concatenando a entrada com a última

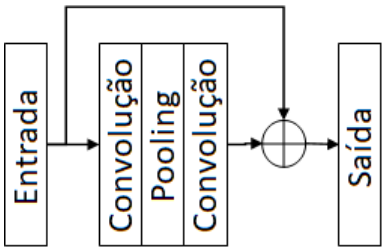
convolução, a cada n convoluções, onde n pode variar conforme a necessidade, mas comumente são utilizado duas convoluções como mostrado na Figura 13. Sendo que a imagem de entrada do bloco e de saída são de mesmo tamanho. Porém outro método que pode ser utilizado é um bloco com adição de uma camada de *pooling* após a primeira convolução. O nome da rede varia conforme o número de camadas residuais, por exemplo, uma ResNet-18 apresenta 18 camadas residuais, enquanto uma ResNet-50 apresentará 50 camadas residuais em sequência.

Figura 13 – Exemplo de bloco de convolução residual.



Fonte: Adaptado de (HE et al., 2016).

Figura 14 – Exemplo de bloco de convolução residual com adição de uma camada de *pooling*.



Fonte: Adaptado de (HE et al., 2016).

A equação do bloco ResNet pode ser visto a seguir:

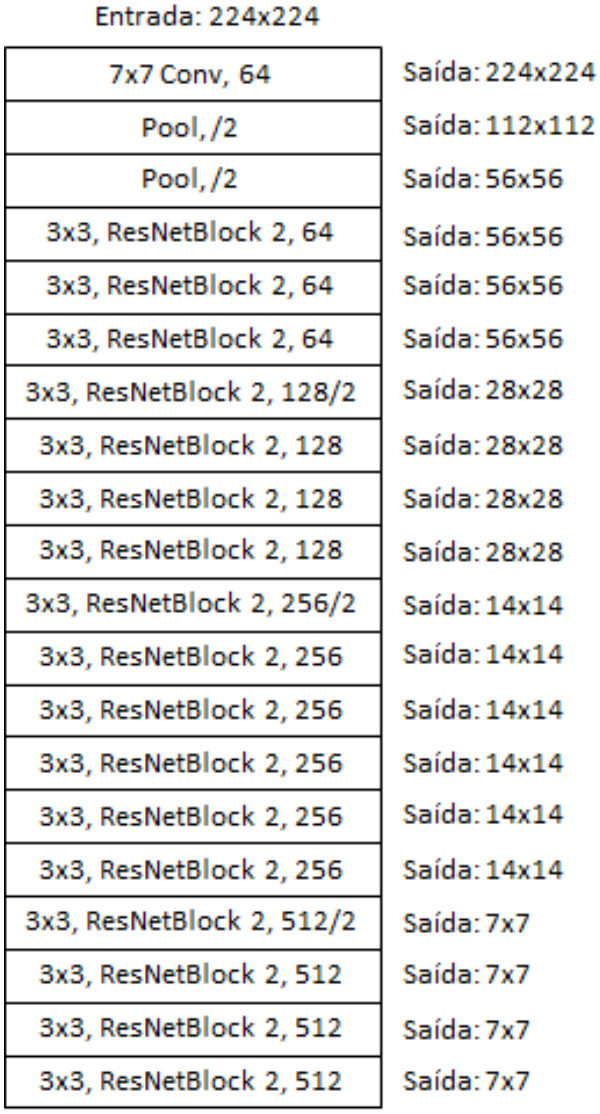
$$x_l = H_l(x_{l-1}) + x_{l-1} \tag{3.20}$$

Onde x_l representa a saída da camada, l a camada atual e $H_l(\cdot)$ representa a função executada nas camadas, como por exemplo no caso de uma camada mostrada na Figura 13, H_l seria, uma convolução (Conv) com filtros 3x3, seguida de outra camada de convolução igual, e por fim uma camada de retificação (RELU), Relu-Conv(3x3)-Conv(3x3).

A arquitetura pode ser vista na Figura 15, mostrando a arquitetura da rede ResNet, o nome de cada camada representa a configuração do bloco, a primeira característica representa o tamanho do filtro de cada convolução, o segundo o tipo da camada, podendo

ser seguido pelo o número de convoluções, e por último o número de neurônios da camada, que podem conter o "/2"no nome, representando a adição de uma camada de *pooling* após a convolução. Exemplo disso pode ser visto em "3x3, ResNetBlock 2, 64", que pode ser lido como um bloco ResNet, com duas convoluções de 64 neurônios, com filtros de tamanho 3x3.

Figura 15 – Arquitetura da rede ResNet-18, contendo apenas as camadas convolucionais, com a imagem entrando na parte superior, com saída na parte de baixo.



Fonte: adaptado de (HE et al., 2016).

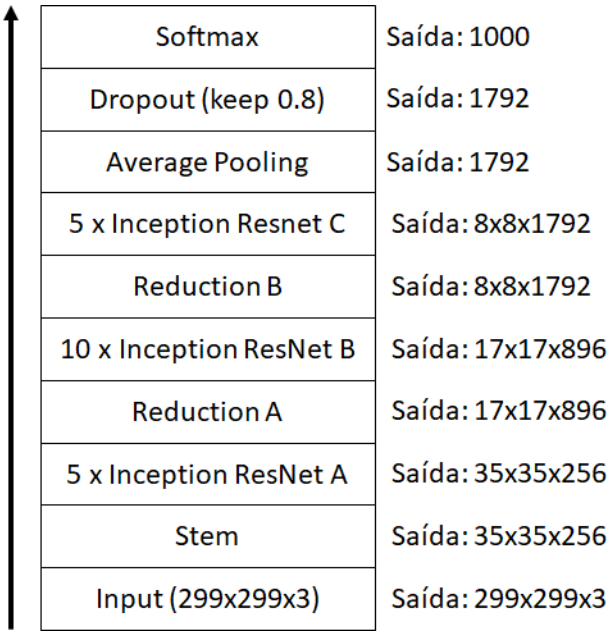
3.10.2 InceptionResnetV2

Descrita em (SZEGEDY et al., 2017), desenvolvida pela Google, com a ideia de misturar as arquiteturas Inception e ResNet. Arquiteturas Inception costumavam ser treinadas

de forma particionada, onde cada replica era particionado em múltiplas sub redes de modo a ser possível treinar todo o modelo em memória.

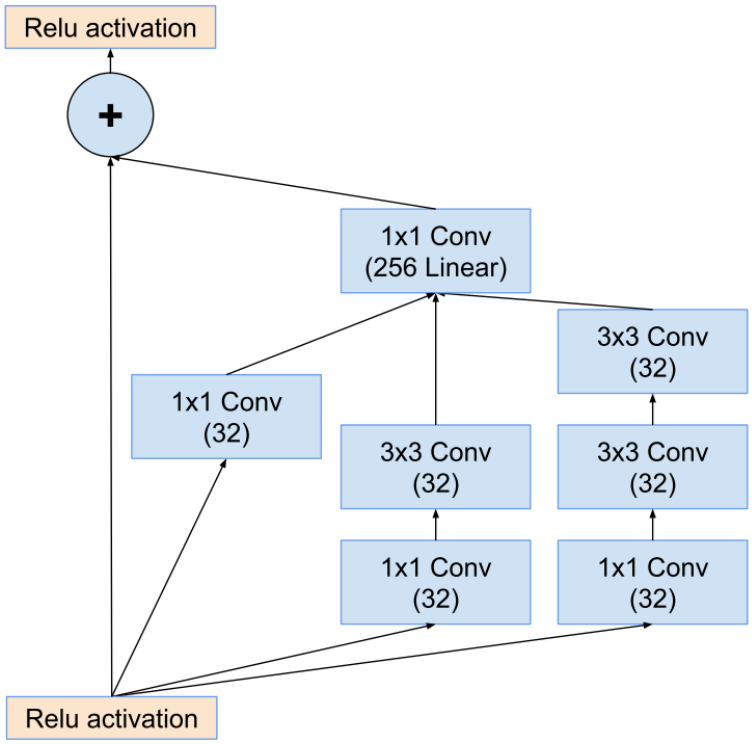
Todavia com as otimizações de códigos realizadas pelas versões mais recentes de *frameworks* , foi possível inserir mais complexidade a rede. Uma rede Inception ResNet é composta por 6 tipos de camadas, A mostrado na Figura 17, B mostrado na Figura 19, e C mostrado na Figura 21, e as reduções de A mostrado na Figura 18, B mostrado na Figura 20, e Stem mostrado na Figura 22. Estes blocos são unidos na arquitetura Inception ResNet, mostrado na Figura 16.

Figura 16 – Arquitetura Inception ResNet, com entrada na parte de baixo e saída na parte de cima.



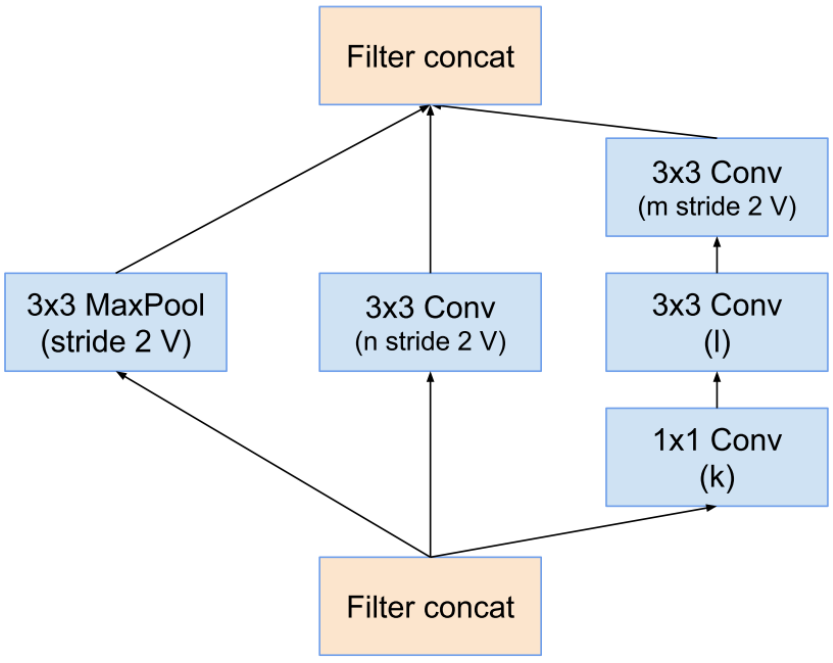
Fonte: (SZEGEDY et al., 2017).

Figura 17 – Bloco A de uma rede Inception.



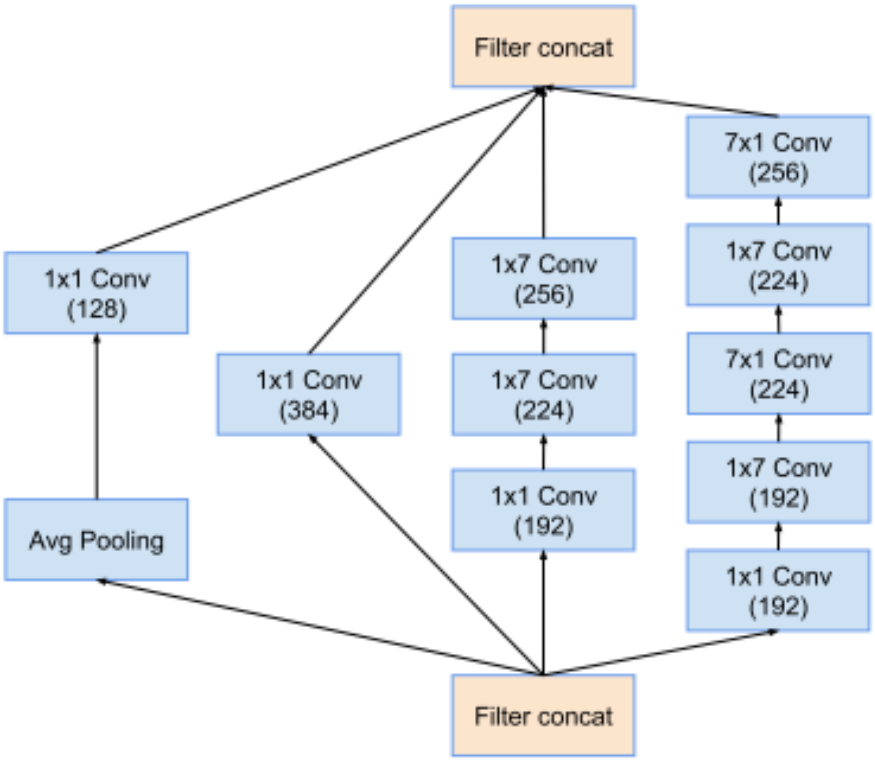
Fonte: (SZEGEDY et al., 2017).

Figura 18 – Bloco A de redução de uma rede Inception.



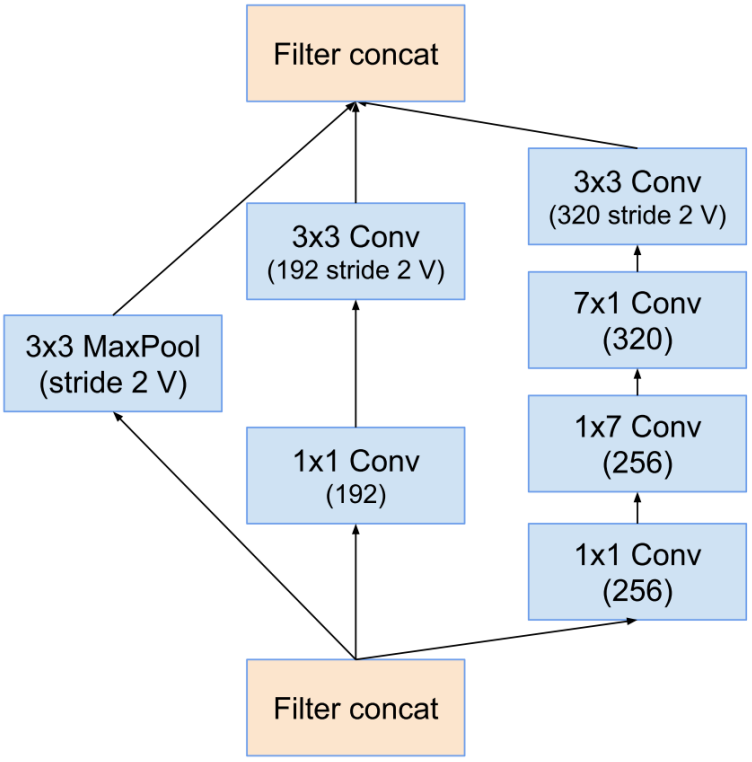
Fonte: (SZEGEDY et al., 2017).

Figura 19 – Bloco B de uma rede Inception.



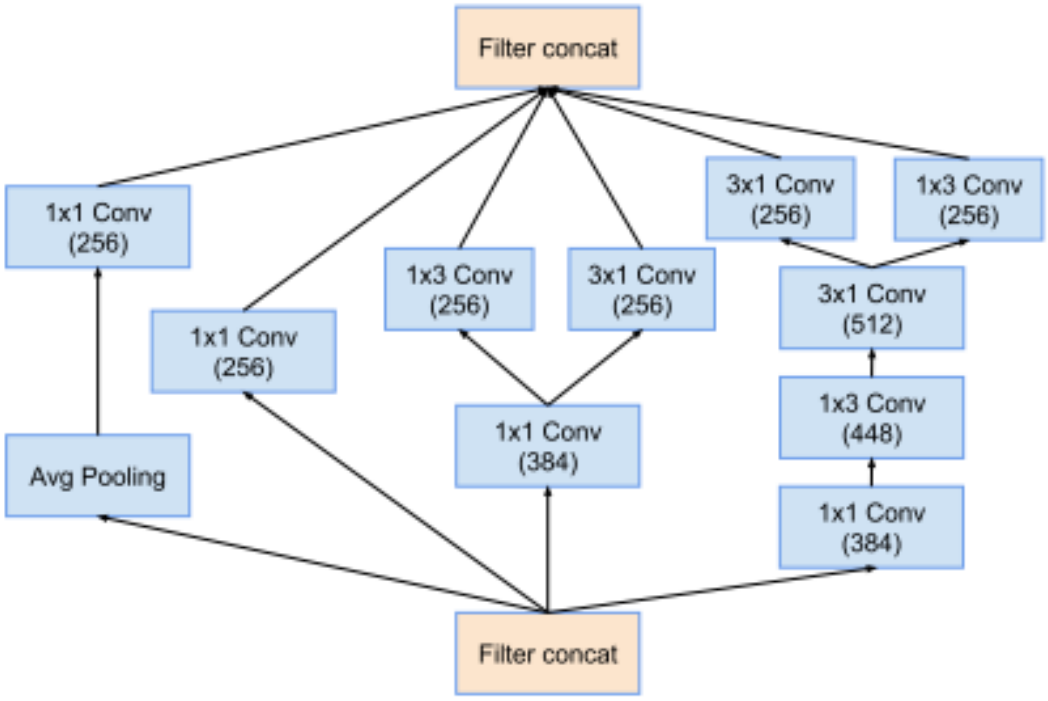
Fonte: (SZEGEDY et al., 2017).

Figura 20 – Bloco B de redução de uma rede Inception.



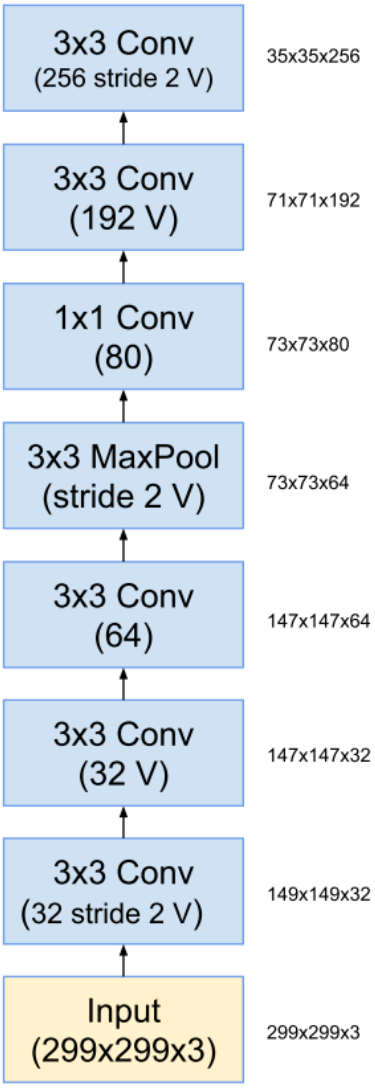
Fonte: (SZEGEDY et al., 2017).

Figura 21 – Bloco C de uma rede Inception.



Fonte: (SZEGEDY et al., 2017).

Figura 22 – Camada Stem da arquitetura Inception ResNet.



Fonte: (SZEGEDY et al., 2017).

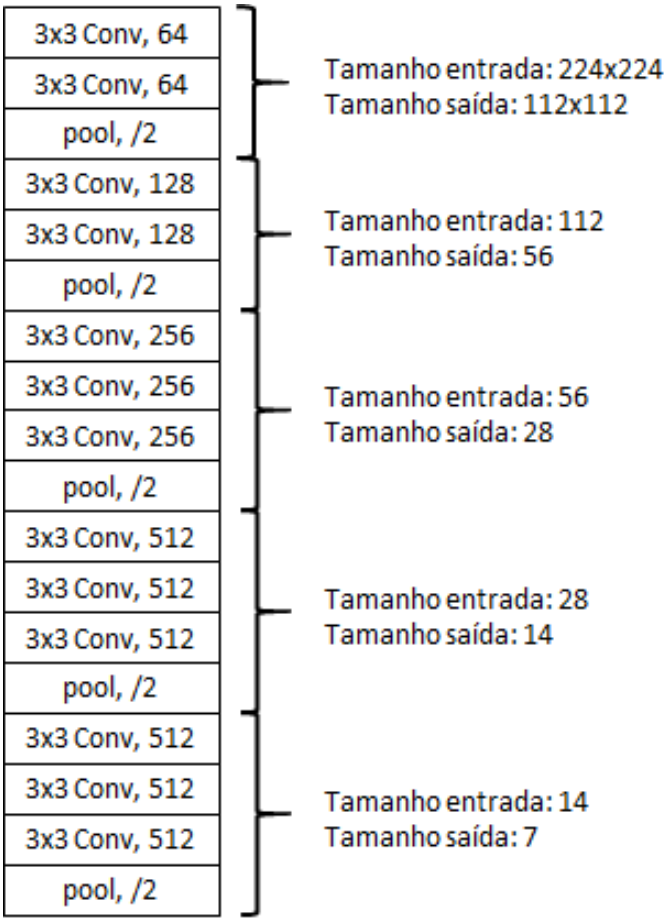
3.10.3 VGG-19

A VGG-19, apresentada no artigo (SIMONYAN; ZISSERMAN, 2015), foi desenvolvida na Universidade de Oxford, sua arquitetura se baseia no empilhamento de redes convolucionais seguidas por uma camada de *max pooling*, com uma janela de 2x2 pixel e um passo de 2. Ela foi treinada utilizando as imagens da ImageNet2012, que consistia em 1.000 classes de imagens diferentes e 1,43 milhões de imagens, sendo divididas em 1,28 milhões de imagens para treinamento, 50 mil imagens para validação e 100 mil para teste, tendo respectivamente uma proporção aproximada de 89,5%, 7% e 3,5%, seus pesos foram fornecido gratuitamente pela Google.

A Figura 23, apresenta a arquitetura da rede VGG contendo apenas as camadas con-

volucionais. Normalmente a rede é seguida por 2 camadas densas, com 4.096 neurônios e uma última camada densa de 1.000 neurônios, todavia como esse trabalho só utilizou as camadas convolucionais da arquitetura, foi preferido não apresenta-las.

Figura 23 – Arquitetura da rede VGG-19, contendo apenas as camadas convolucionais.



Fonte: (HE et al., 2016).

3.10.4 DenseNet121

A arquitetura *DenseNet*, descrita em (HUANG et al., 2017), apresenta um novo bloco de convoluções chamados *Dense*. Entre um bloco *Dense* e outro é colocado uma camada de transição, composta por uma convolução e um *pooling*.

Em um bloco de *ResNet* tradicional sua entrada está ligada diretamente na entrada do próximo bloco. Todavia em um bloco *Dense*, a saída está ligada diretamente nas entradas dos próximos blocos, assim a camada l^n recebe todas as entradas das camadas anteriores

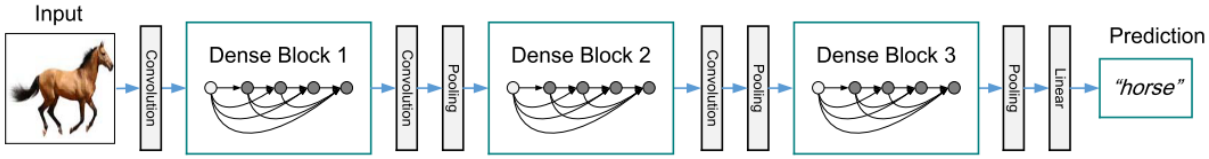
como entrada, $x_0, \ldots, x_{l-1}$, como descrito em:

$$x_l = H_l\left([x_0, x_1, \ldots, x_{l-1}]\right) \tag{3.21}$$

Onde $H_l(\cdot)$ é composto por três operações, uma normalização de pacote (BN), seguida de uma unidade de retificação linear (ReLU) e uma convolução (Conv) com filtros de tamanho 3x3, BN-Relu-Conv(3x3), ou pode ser uma camada de Bottleneck, composta por BN-ReLU-Conv(1x1)-BN-ReLU-Conv(3x3), utilizada para reduzir o número de *feature-maps* e assim melhorar a eficiência computacional.

A arquitetura da rede é mostrada na Figura 24. Como pode ser visto, cada bloco *Dense* é separado por blocos de transições formados por convoluções e *pooling*. A adição de mais blocos define o nome da rede, logo uma rede DenseNet101 possui 101 blocos Dense, enquanto uma rede DenseNet203, possui 203 blocos *Dense*. A última camada é composta de uma rede artificial densa com uma ativação linear.

Figura 24 – Arquitetura de uma DenseNet.



Fonte: (HUANG et al., 2017).

3.11 Algoritmos de visualização

Nessa sessão será explicados o funcionamento de diversos métodos de visualizações dos valores de importância da CNN. Esses métodos foram criados com o intuito de segmentar a imagem de forma automática, mas seus usos vêm crescendo em outra áreas por poderem visualizar a imagem como a rede em treinamento a visualiza. Entre esses métodos o Grad-CAM é o mais utilizado por poder ser aplicado em qualquer rede sem muita alteração da mesma. Todavia foi utilizado um método mais recente de Grad-CAM, com o intuito de obter resultados melhores se comparado com os trabalhos utilizando exclusivamente o método Grad-CAM, como o trabalho (TABIK et al., 2020).

3.11.1 CAM

Proposta em (ZHOU et al., 2016), a *Class Activation Map* ou **CAM**, gera uma imagem para uma determinada categoria c indicando a região da imagem utilizada pela CNN para identificar a categoria. Para uma dada imagem, onde $f_k(x, y)$ representa a ativação da unidade k na última camada convolucional em localização espacial (x, y). Então para uma unidade k, o resultado para um camada pool, F^k é $\sigma_{x,y} f_k(x, y)$.

Dado uma classe c, a entrada para a *sotfmax*, S_c, é $\sum_k w_k^c F^k$ onde w_k^c é o peso correspondente a classe c para a unidade k. Essencialmente, w_k^c indica a importância de F^k para a classe c. Substituindo os valores nas equações anteriores tem-se a equação:

$$S_c = \sum_{x,y} \sum_k w_k^c f_k(x, y) \tag{3.22}$$

Após o cálculo realiza-se o *upsampling* da imagem gerada para o tamanho original e terá o CAM para a sua imagem em determinada categoria.

A Figura 25 é um exemplo de uma saída do método CAM, a Figura 25(a) apresenta o mapa de calor gerado pelo CAM, nota-se que a área em vermelho seria o local de maior importância da imagem para a rede para classificar a imagem, a região em vermelho nesse caso se apresenta justamente sobre as escovas de dentes, sendo assim a rede da mais importância a escova de dente na escolha da classe. O mesmo pode ser visto em 25(b), nele a região de maior importância em vermelho se mostra sobre a imagem do homem e da motosserra para classificar a imagem.

Figura 25 – Exemplo da imagem gerada pelo método CAM. (a) Lavando os dentes. (b) Cortando Árvores.

(a) **(b)**

Fonte: (ZHOU et al., 2016).

3.11.2 Grad-CAM

Proposta em (SELVARAJU et al., 2020), o *Gradient-weighted Class Activation Mapping* (Grad-CAM) é uma generalização da CAM para uma maior variedade de arquiteturas de redes convolucionais, permitindo aplicar o método CAM sem a modificação da arquitetura da rede ou as entradas utilizadas, tornando o CAM um caso especial do Grad-CAM, o que o transformou num dos métodos mais utilizadas para visualização dos valores de importância de cada neurônio para uma decisão de interesse, dado por:

$$L^c_{Grad-Cam} = ReLU\left(\sum_k \alpha^c_k A^k\right) \tag{3.23}$$

Onde a ReLU é utilizada pois os pesos negativos não são interessantes, α^c_k representa a importância dos pesos do neurônio e A^k o *feature map*, gerando um mapa de calor, *heatmap*, de mesmo tamanho da camada convolucional de interesse, que no caso deste trabalho é a última camada convolucional de tamanho uxv.

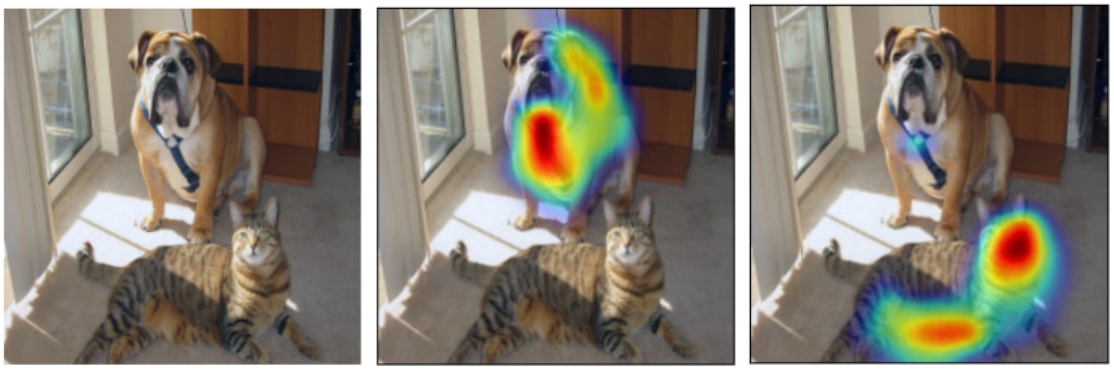
O peso α^c_k representa uma linearização parcial da rede profunda atrás de A e mostra a importância da *feature map* k da classe alvo c, é calculado segundo a equação:

$$\alpha^c_k = \frac{1}{Z}\sum_i^u \sum_j^v \frac{\partial y^c}{\partial A^k_{ij}} \tag{3.24}$$

Onde y^c é a probabilidade da classe c, e Z é o número de pixeis da *feature map*, isto é, $Z = u \cdot v$.

A Figura 26 apresenta um exemplo de imagem de saída gerada pelo método Grad-CAM, tendo como entrada do método a Figura 26(a), semelhante a Figura 25 a região em vermelho também define a região de maior importância na classificação da imagem, como a Grad-CAM é uma generalização do método CAM, ambos apresentam saída semelhante, como na Figura 26(b) onde a região de maior importância se encontra sobre o cachorro, já a na Figura 26(c) se encontra sobre o gato.

Figura 26 – Exemplo do mapa de calor para cada classe gerada pelo método Grad-CAM para uma imagem de entrada. (a) Entrada. (b) Classe Cachorro. (c) Classe Gato.



(a) **(b)** **(c)**

Fonte: (SELVARAJU et al., 2020).

3.11.3 Grad-CAM Probabilístico

Todavia, para conseguir um melhor entendimento de como a rede tinha predito os recortes das imagens originais (OH; PARK; YE, 2020) sugere uma modificação a equação (3.23), mostrado na equação (3.26), a dividindo em duas partes, local e global.

A primeira, parte referente a predição do pacote, é chamado de escopo local, nela calcula-se o Grad-CAM do pacote com o objetivo de se obter seu *heatmap*, através de:

$$l^c(x) = Up\left(ReLU\left(\sum_k \alpha_k^c f^k(x)\right)\right) \in \mathbb{R}^{mxn} \tag{3.25}$$

Onde $x \in \mathbb{R}^{pxq}$ é um pacote da imagem de entrada, a função $f(x) \in \mathbb{R}^{uxv}$ gera o *feature map* do pacote de x_k e $Up(\cdot)$ é o operador de *upsampling* do *feature map*, aumentando o *heatmap* de uxv para o tamanho do pacote de pxq.

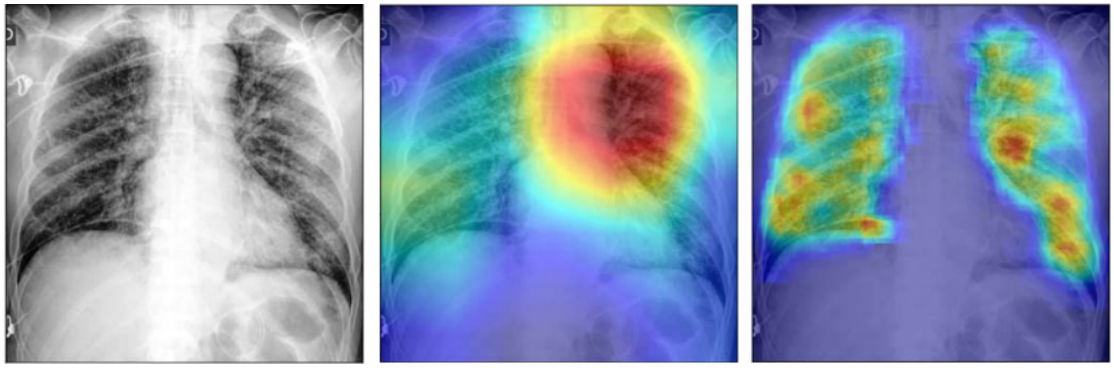
Para a aproximação global utilizou-se a equação a seguir:

$$[l^c_{prob}(x)]_i = \frac{1}{K_i}\left[\sum_{k=1}^{K} r^c(x_k)\mathcal{Q}_k(l^c(x_k))\right]_i \tag{3.26}$$

Seu objetivo é gerar o *heatmap* da imagem inteira, onde $x \in \mathbb{R}^{mxn}$ é a imagem de entrada, $x_k \in \mathbb{R}^{pxq}$ representa o pacote k da entrada, $\mathcal{Q}_k : \mathbb{R}^{pxq} \mapsto \mathbb{R}^{mxn}$ é o operador que copia para a posição adequada do novo *heatmap*, $r^c(x_k)$ a probabilidade da classe c calculada antes da camada de *softmax* para o pacote k e K_i denota a frequência que um determinado pixel i foi utilizado.

A Figura 27 apresenta um exemplo da imagem gerada pelo método, a imagem 27(a) é servida de entrada para a rede, enquanto a Figura 27(b) a região de maior importância se concentra apenas na região da lesão principal, se tornando difícil a diferenciação do lesões multi focadas, GGO dispersos e consolidações, a Figura 27(c) gerada pelo Grad-CAM probabilístico, diferente dos métodos CAM e Grad-CAM, apresenta uma região de maior importância mais espalhada por toda a Figura apresentando mais regiões de importância a serem analisadas do que seus concorrentes. Essa característica é mais desejável em diagnósticos, por mostrar com mais clareza as regiões que possuem lesões, não somente a região de maior lesão.

Figura 27 – Exemplo dos mapas de calor gerado pelo método Grad-CAM probabilístico. (a) Imagem do pulmão de entrada. (b) Grad-CAM convencional. (c) Grad-CAM probabilístico.



(a) **(b)** **(c)**

Fonte: (OH; PARK; YE, 2020).

4 Metodologia

A seguir será apresentado a metodologia usada para a realização deste trabalho, apresentando os equipamentos utilizados, os *datasets*, modelos e métodos.

4.1 *Hardware* e ambiente de programação

Para a realização deste trabalho foi utilizado um notebook ASUS X555LF, serie V, com processador Intel Core i7 5500U, memória RAM de 10 GB DDR3L 1600MHz, GPU NVIDIA GeForce 930M com 2 GB de memória compartilhada, e armazenamento de 256GB SSD e 1TB no disco rígido, do próprio autor. Porém o treinamento foi realizado em nuvem através do computador virtual fornecido gratuitamente pelo Google atráves da plataforma Kaggle. Todo o projeto foi feito utilizando o Python 3.7.9, com o auxilio do *framework* Keras, fornecido gratuitamente também pela Google.

Criado por François Chllet, um engenheiro da Google, Keras é uma API de escrita de aprendizado profundo em Python, seu código é aberto rodando por cima da plataforma de aprendizado de máquina TensorFlow. Apesar de ser poderoso, o TensorFlow possui um complicado método de criação de modelos, com isso o objetivo do Keras foi facilitar a criação desses modelos de redes.

4.2 Segmentação do pulmão

4.2.1 *Datasets* utilizados

A seguir serão apresentado o conjunto de dados utilizados para o treinamento e teste da rede neural, os *datasets* de segmentação classificação, a fim de analisar o desempenho da rede. A escolha desses *datasets* se deu pela disponibilidade pública do conjunto e a utilização dos mesmos na literatura.

Para o *dataset* da parte de segmentação do pulmão foram utilizados os dois *dataset* fornecidos em (JAEGER et al., 2014), ambos disponibilizados de maneira pública pelo Departamento de Saúde e Serviços Humanos de Montogomery County (MC), em Maryland, Estados Unidos, e o Hospital das Pessoas de Shenzhen 3 da Universidade Médica

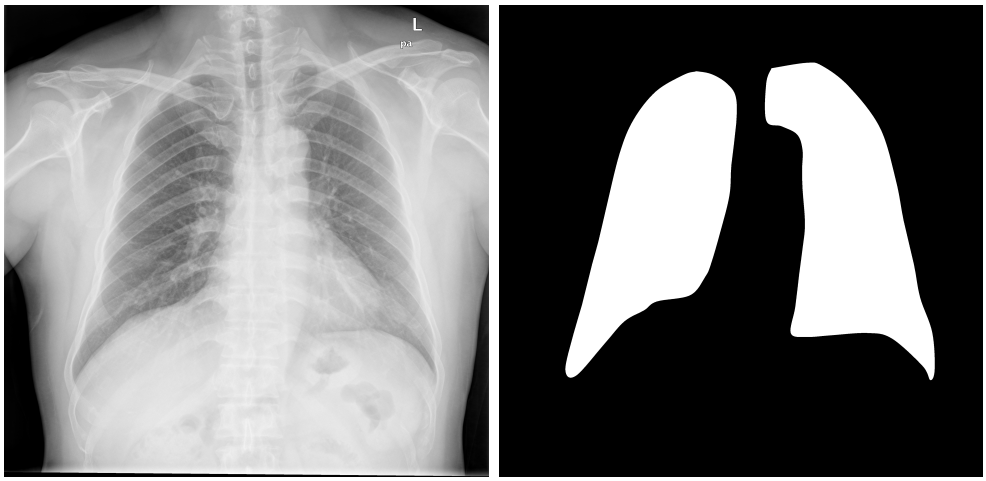
Guangdong em Shenzhen (Shenzhen), China. Neles contem imagens de raio-X do tórax no formato PNG de 12bit em níveis de cinza, com tamanhos de de 4020x4892 ou 4892x4020, separados por tipo, normal e tuberculose mostrado na Tabela 1, as imagens são mostrados em Figura 28, porém a separação não foi considerado para a segmentação, disponível em (MADER, 2017).

Tabela 1 – Dispersão do *dataset* de segmentação entre Normal e Tuberculose.

Dataset	**Normal**	**Tuberculose**	**Total**
MC	80	58	**138**
Shenzhen	326	336	**662**
Total	**406**	**394**	800

Fonte: Autor

Figura 28 – Imagens fornecidas pelos *datasets*. (a) Imagem Original. (b) Máscara do pulmão.



(a) **(b)**

Fonte: (JAEGER et al., 2014).

Porém as máscaras foram fornecidas por (STIRENKO et al., 2018), baseadas nas imagens do *dataset* acima, o mesmo cita o *dataset*, Tabela 1. O *dataset* é considerado pequeno, formado por menos de 10^3 imagens, o que pode produzir más predições e com baixa acurácia.

O *dataset* foi divido em três partes: treinamento, validação e teste, sendo as imagens escolhidas aleatoriamente em uma proporção de 72%, 18% e 10% respectivamente, como mostrado na Tabela 2, tendo como regra a não utilização de uma imagem em mais de uma parte, visando diminuir o enviesamento da rede.

Tabela 2 – Dispersão do *dataset* entre treino, validação e teste.

Dataset	**Total Imagens**	**Porcentagem [%]**
Treino	576	72
Validação	144	18
Teste	80	10
Total	800	100

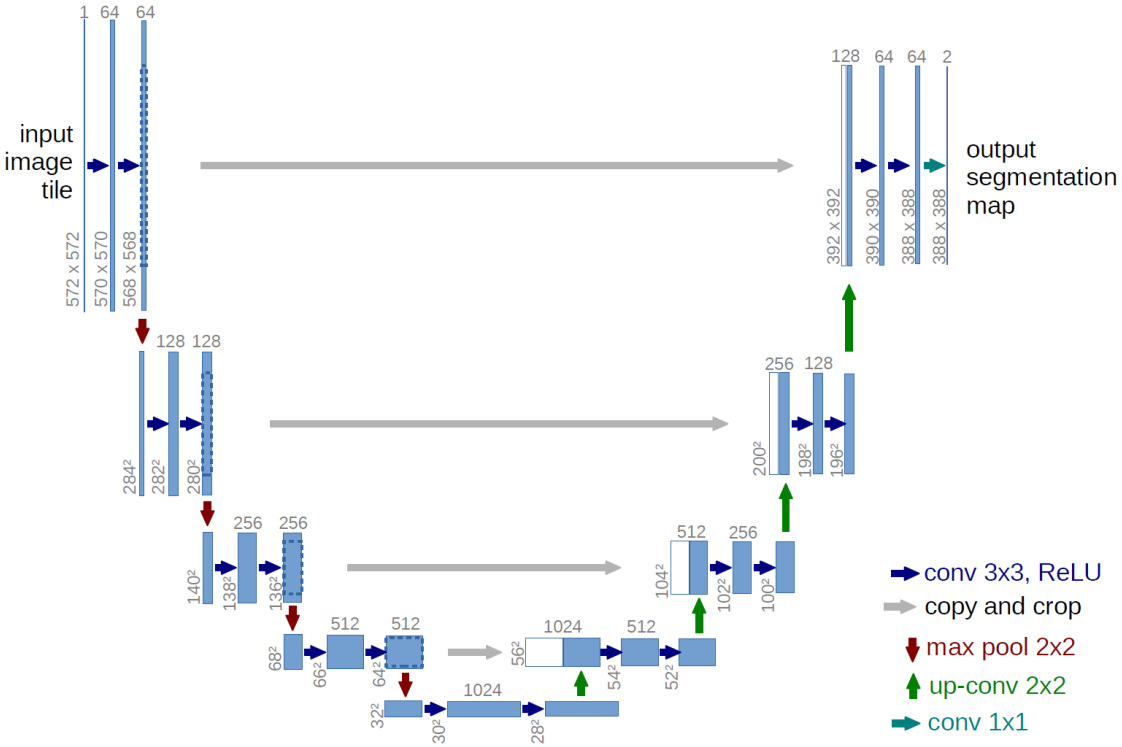
Fonte: Autor

A etapa de treinamento é a primeira etapa realizada servindo para calcular os pesos da rede, enquanto a validação é a verificação se o resultados do treinamento realmente são verdadeiros, servindo de métrica afim de evitar o efeito de *Overfitting*. Este efeito surge quando a rede se especializa nas imagens do treinamento mas não segmenta com a mesma acurácia imagens desconhecidas. Isso tende a ocorrer em maior escala em *datasets* pequenos. Como última etapa, o teste é a validação final da arquitetura proposta, utilizando imagens inéditas, não utilizadas nas etapas anteriores.

4.2.2 Modelos das Redes

O modelo sugerido para a segmentação do pulmão foi o baseado no clássico U-Net, descrito em (RONNEBERGER; FISCHER; BROX, 2015) e mostrado na Figura 29, com os valores da dimensão alterados para uma imagem de 256x256 pixeis e adicionando uma camada de *Dropout* antes da última camada de convolução para diminuir o efeito de *overfitting*. Este modelo é relativamente simples em comparação a modelos mais complexos como a ResNet, que utiliza 1.962.337 parâmetros da rede, sendo todos treináveis.

Figura 29 – Arquitetura U-Net. Cada caixa azul corresponde a um mapa de recursos multicanais. As caixas brancas representam a copias dos mapas de recursos. As setas denotam as diferentes operações realizadas.

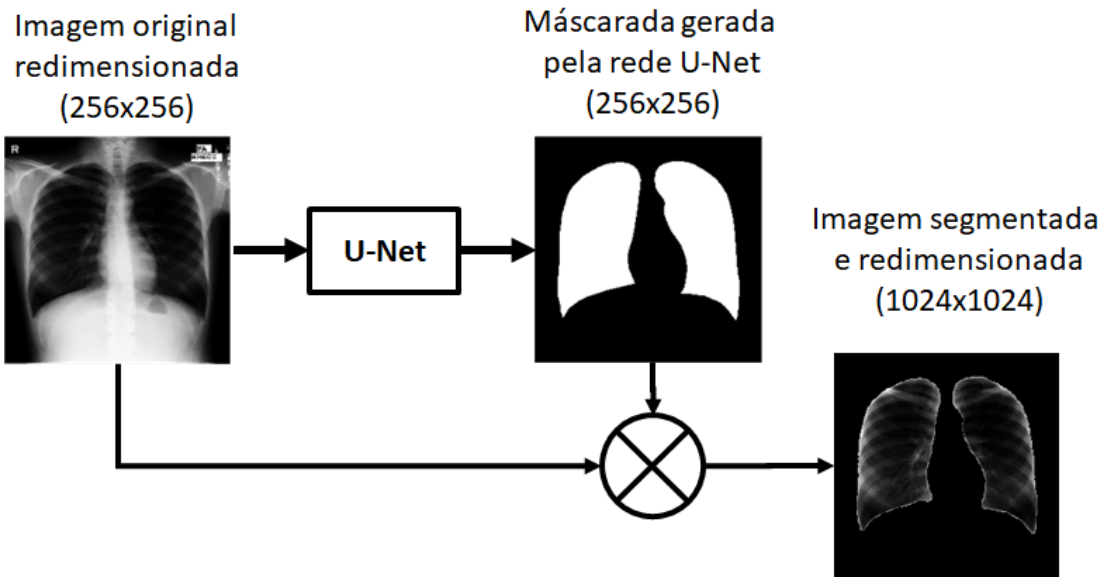


Fonte: (RONNEBERGER; FISCHER; BROX, 2015)

Como explicado em (RONNEBERGER; FISCHER; BROX, 2015), o modelo consiste em duas partes, contração e expansão: a contração da imagem consiste em repetidamente aplicar duas convoluções com filtros de tamanho 3x3, sem preenchimento, cada uma seguida por uma camada de ativação de retificação linear (ReLU) e uma operação contração através da camada *max pooling*, com passo de 2x2, sendo que a cada passo dobra o número de neurônios da camada anterior. A expansão realiza o trabalho inverso da contração, expandindo o tamanho da imagem através da camada de *upsampling*, com um passo de 2x2, e concatenando as saídas da parte de contração com a parte de expansão, seguidas por duas camadas de convolução. Na camada final do modelo é usada uma convolução com filtros de tamanho 1x1.

A obtenção da imagem de segmentação pode ser vista na Figura 30, o modelo receberá a imagem do pulmão redimensionada, gerando a máscara de segmentação e por fim multiplica a imagem original pela máscara gerada pelo modelo, a redimensionando par a servir de entrada para a rede de classificação.

Figura 30 – Exemplo do funcionamento da segmentação do pulmão.



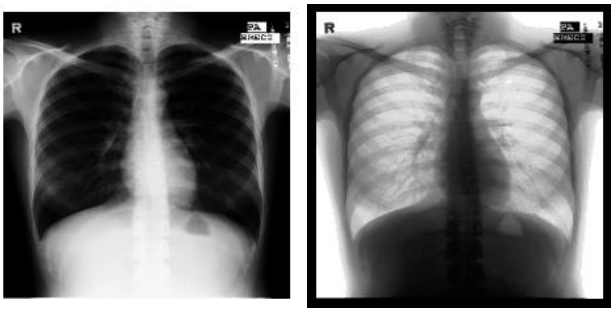
Fonte: Autor.

4.2.2.1 Pré-processamento

Para melhorar o desempenho do modelo de segmentação, o número de imagens foram artificialmente aumentadas utilizando o processo de *data augmentation*, utilizando a biblioteca disponibilizada gratuitamente em (BUSLAEV et al., 2020), que visa modificar as imagens originais do *dataset* para gerar novas imagens para aumentar seu tamanho.

Para isso primeiramente todas as imagens do *dataset* passaram por duas etapas, inversão das cores e equalização do histograma, um exemplo desse processo pode ser visto na Figura 31.

Figura 31 – Exemplo do pré processamento de todas as imagens do dataset. (a) Entrada. (b) Saída

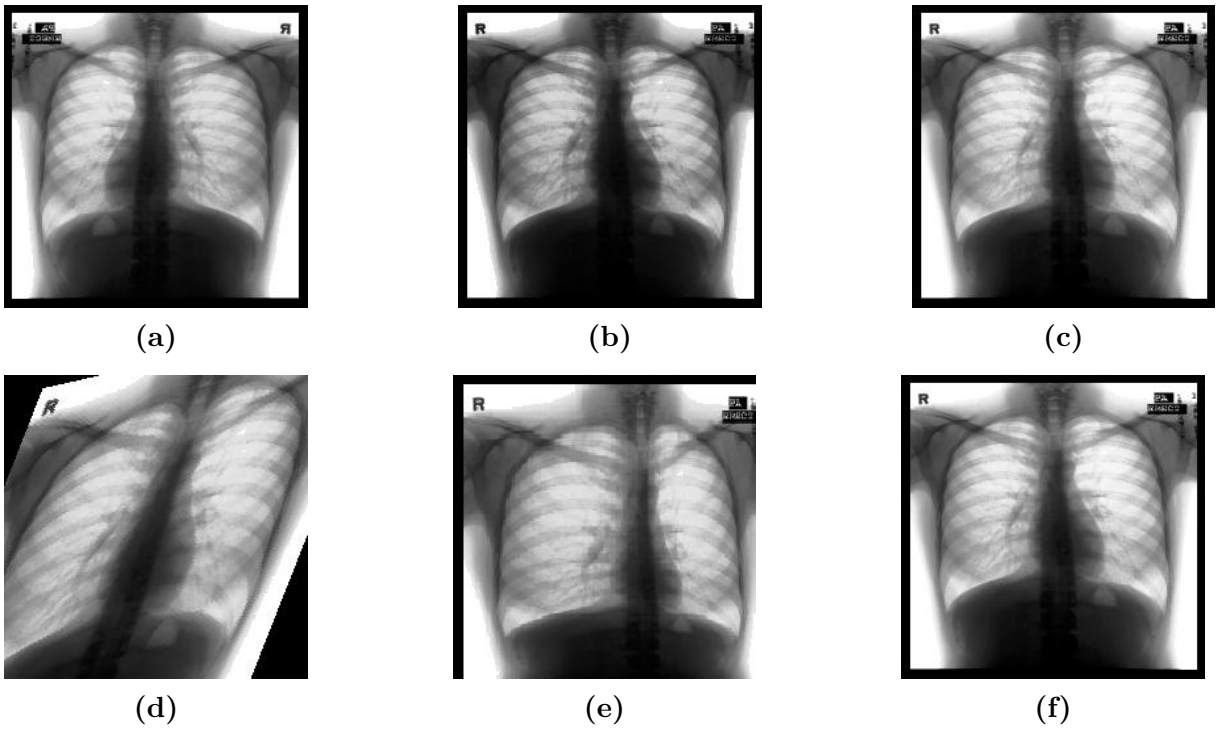


(a) **(b)**

Após esse procedimento é realizados as operações de *data augmentation* no *dataset*, porém nessa etapa as operações tem uma probabilidade, e ainda parâmetros randômicos

em alguns processos, de serem aplicadas associado a cada etapa. Com isso é possível que uma o Essas etapas são, inversão horizontal, correção gamma, ruido gaussiano, transformação elástica e alteração do brilho e contraste. Essas modificações pode ser vistas na Figura 32 e seu código pode ser visto no Algoritmo 1.

Figura 32 – Exemplo do pré processamento de todas as imagens do *dataset*. (a) Inversão horizontal. (b) Correção gamma. (c) Ruído gaussiano. (d) Transformação elástica. (e) Distorção da rede. (f) Alteração do contraste e brilho.



(a) **(b)** **(c)**

(d) **(e)** **(f)**

```
"Processo de data augmentation do dataset de segmentacao"""
import albumentations as A

transform = A.Compose([
  A.InvertImg(p=1.0),
  A.Equalize(p=1.0),
  A.HorizontalFlip(p=0.5),
  A.RandomGamma(always_apply=False,
                p=1.0,
                gamma_limit=(23, 81),
                eps=1e-07),
  A.RandomBrightnessContrast(always_apply=False,
                             p=1.0,
```

```
14                              brightness_limit=(-0.2, 0.2),
15                              contrast_limit=(-0.2, 0.2),
16                              brightness_by_max=True),
17   A.GaussNoise(var_limit=(200,300), p=0.5),
18   A.ElasticTransform(always_apply=False,
19                      p=0.5, alpha=3.0,
20                      sigma=50.0,
21                      alpha_affine=50.0,
22                      interpolation=0,
23                      border_mode=0,
24                      value=(0, 0, 0),
25                      mask_value=None,
26                      approximate=False),
27   A.GridDistortion(always_apply=False,
28                    p=1.0,
29                    num_steps=5,
30                    distort_limit=(-0.3, 0.3),
31                    interpolation=0,
32                    border_mode=0,
33                    value=(0, 0, 0),
34                    mask_value=None)
35 ])
```

Algoritmo 1 – Código para *Data Augmentation* do modelo de segmentação.

A linha 2 é referente a importação da biblioteca para a *data augmentation* das imagens de raio-x. Após a importação, é criado o *compose* onde se insere as transformações a serem aplicadas no *dataset*. A partir da linha 7 a linha 34, é adicionado as classes de transformações.

Com esse processo foi possível gerar 9.200 imagens, aumentando de 800 imagens originais do *dataset* para 10.000 imagens no total.

4.2.2.2 Hiper Parâmetros

A seguir será apresentado os hiper parâmetros utilizados durante a etapa de treinamento para todas as redes pré treinadas, contendo a função de erro utilizada, o otimizador do erro e a taxa de aprendizado do treino.

- *Loss*: utilizou-se de uma variação da função *Dice Loss*, descrita em (SUDRE et al., 2017), denominada *Log Cosh Dice Loss* proposta em (JADON, 2020), descrita pela equação 3.17, onde y são os valores reais e $\hat{p}$ são os valores preditos.
- Otimizador: o otimizador utilizado foi o *Adamax* proposto em (KINGMA; BA, 2015).
- Taxa de aprendizado: utilizou-se uma taxa de aprendizado de 10^{-5}.

Foram utilizados callbacks para gerenciar o treinamento do modelo automaticamente. O código para a geração podem ser visto em Algoritmo 2. As linhas 2 a 8 contém as configurações para a criação do checkpoint do modelo, monitorando o parâmetro da validação do erro salvando apenas os pesos da época do menor valor, e por fim a linha 9 cria o objeto da classe de checkpoint.

As linhas 10 a 18 apresentam as configurações do plano de redução da taxa de aprendizado, que reduzirá com um fator de 0,5, caso o erro de validação do modelo não reduza por três épocas consecutivas, com um redução do erro mínima de 10^{-3}, valor mínimo de taxa de aprendizado de 10^{-8}, valendo a partir da 2 época.

```
1  def get_callbacks() -> List[Callback]:
2      check_params = {
3          "monitor": "val_loss",
4          "verbose": 1,
5          "mode": "min",
6          "save_best_only": True,
7          "save_weights_only": True,
8      }
9      checkpoint = ModelCheckpoint(".\\.model\\best.weights.hdf5", **check_params)
10     reduce_params = {
11         "factor": 0.5,
12         "patience": 3,
13         "verbose": 1,
14         "mode": "max",
15         "min_delta": 1e-3,
16         "cooldown": 2,
17         "min_lr": 1e-8,
18     }
19     reduce_lr = ReduceLROnPlateau(monitor="val_loss", **reduce_params)
```

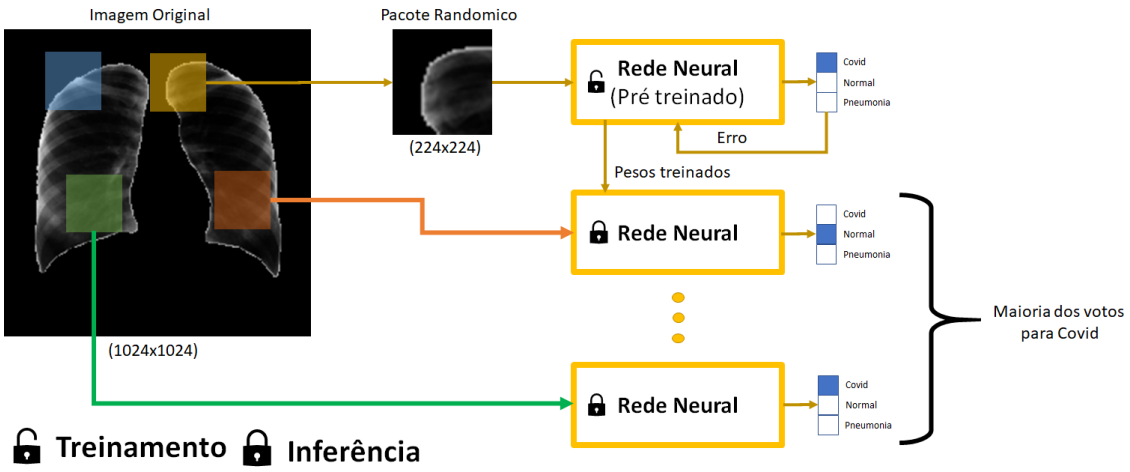
```
    terminate = TerminateOnNaN()
    callbacks = [checkpoint, reduce_lr, terminate]
    return callbacks
```

Algoritmo 2 – Criação dos *callbacks* para o modelo de segmentação.

4.3 Classificação da doença

A classificação é divida em duas etapas: escopo global e local. No escopo local é determinado a classe de um recorte da imagem do pulmão, enquanto no escopo global é definido a classe da imagem do pulmão, porém o escopo local pode ser de uma classe diferente do escopo global. Essa etapa tem como objetivo gerar o mapa de calor Grad-CAM probabilístico que será utilizado na classificação da severidade do Covid-19. O processo de classificação é mostrado na Figura 33.

Figura 33 – Exemplo do funcionamento do modelo de classificação.



Fonte: Autor.

O Algoritmo 3 apresenta o algoritmo em python para a criação do modelo de classificação.

```
def classification_model(
    dim: int = 256,
    channels: int = 1,
    classes: int = 3,
    final_activation: str = "softmax",
    activation: str = "relu",
```

```
7     drop_rate: float = 0.2,
8     model_name: str = "ResNet50V2",
9 ) -> Model:
10    input_shape = (dim, dim, channels)
11    inputs = Input(shape=input_shape)
12    layers = BatchNormalization()(inputs)
13    layers = Conv2D(filters=3, kernel_size=(3, 3), padding="same")(layers)
14    layers = Activation(activation=activation)(layers)
15    layers = Dropout(rate=drop_rate)(layers)
16    layers = base(model_name=model_name, split_dim=dim)(layers)
17    layers = Flatten()(layers)
18    layers = Dense(units=classes)(layers)
19    layers = Activation(activation=final_activation)(layers)
20    outputs = Dropout(rate=drop_rate)(layers)
21    model = Model(inputs=inputs, outputs=outputs)
22    return model
```

Algoritmo 3 – Criação do modelo de classificação.

As linhas 1 a 9 definem a função de criação do modelo, que retornará um objeto da classe *Model* da biblioteca Keras. Os parâmetros da função são a dimensão da imagem de entrada (*dim*), o número de canais da imagem de entrada (*channels*), o número de classes de saída (classes), a função de ativação da saída do modelo (*final_activation*), as funções de ativação entre as camadas (*activation*), o taxa de drop de neurônios entre as camadas (*drop_rate*) e o nome da arquitetura a ser utilizada (*model_name*). As linhas 10 a 22 a apresentam a lógica para a criação do modelo de classificação, onde a linha 1 é a camada de entrada para o modelo, de 12 a 15 tem a criação do bloco de convolução, a linha 16 cria o modelo base de classificação entre os modelos ResNet50V2, DenseNet121, InceptionResnetV2 e VGG-19, por fim a linha 21 cria o objeto da classe *Model* do Keras.

Para tal foi realizado testes com diversas modelos de redes neurais afim de definir qual possui os melhores resultados com a menor quantidade de parâmetros, para tal foram testados quatro modelos de redes, ResNet50V2, DenseNet121, InceptionResnetV2 e VGG-19, treinadas utilizando os mesmos *datasets*, hiper parâmetros e processos. Esses modelos de redes foram escolhidos por serem muito utilizados na literatura. A ResNet50V2 sendo uma versão com mais camada da versão de (OH; PARK; YE, 2020), a DenseNet121 por obter melhores resultados em (HUANG et al., 2017), VGG-19 por estar ser desenvolvida

com o objetivo de auxiliar no diagnósticos de doenças e a InceptionResnetV2 por possuir mais parâmetros que as demais. O Algoritmo 4 apresenta o código para a definição do modelo de rede neural usada.

```
1  def base(model_name: str = "ResNet50V2", split_dim: int = 224) -> Model:
2  """
3  Retorna a função intermediaria para a rede utilizada.
4
5  Args:
6      model_name (str, optional): Nome do modelo intermediario. Defaults to
   ↪  "ResNet50V2".
7      split_dim (int, optional): tamaho da imagem de entrada. Defaults to 224.
8
9  Returns:
10     Model: Modelo intermediario de aprendizado.
11 """
12     base = None
13     shape = (split_dim, split_dim, 3)
14     params = {
15         "include_top": False,
16         "weights": "imagenet",
17         "pooling": "avg",
18         "input_shape": shape,
19     }
20     if model_name == "VGG19":
21         base = VGG19(**params)
22     elif model_name == "VGG16":
23         base = VGG16(**params)
24     elif model_name == "InceptionResNetV2":
25         base = InceptionResNetV2(**params)
26     elif model_name == "ResNet50V2":
27         base = ResNet50V2(**params)
28     elif model_name == "ResNet101V2":
29         base = ResNet101V2(**params)
30     elif model_name == "ResNet152V2":
31         base = ResNet152V2(**params)
32     elif model_name == "DenseNet121":
33         base = DenseNet121(**params)
34     elif model_name == "DenseNet169":
```

```
35          base = DenseNet169(**params)
36      else:
37          base = DenseNet201(**params)
38      return base
```

Algoritmo 4 – Criação do modelo base para a classificação.

4.3.1 Datasets utilizados

Para a classificação da doença foi utilizado um *dataset* contendo de 7.103 imagens, sendo 853 de COVID-19, 1.887 Normais e 4.363 de Pneumonia, que foram organizadas em três partes, como mostrado na Tabela 3 e um exemplo de cada tipo de imagem pode ser vista em Figura 34.

Esse *dataset* tem sido muito utilizado em pesquisas de diagnóstico do Covid-19, que servirá de base para a argumentação posterior sobre o desempenho do método proposto neste trabalho, visto que utilizará o mesmo *dataset*.

Figura 34 – Imagens fornecidas pelos *dataset* de classificação. (a) COVID-19. (b) Normal. (c) Pneumonia.

(a) **(b)** **(c)**

Fonte: Autor.

Os valores da Tabela 3 significam o número de imagens em cada partição e o tipo, como por exemplo a segunda linha da segunda coluna, 457, nele pode-se interpretar que existem 457 imagens de COVID-19 na partição de treinamento, enquanto que a última coluna da segunda linha refere-se ao número total de imagens de COVID-19 que existem no *dataset* inteiro, já para a coluna da última linha, 4.316, é o número de imagens na partição de treinamento em relação ao *dataset*, logo pode-se concluir que existem um total de 4.316 do *dataset* na partição de treinamento.

A Tabela 3 será definidos os valores dos hiper parâmetros utilizados na segmentação dos pulmões.

Tabela 3 – Dispersão das imagens do *dataset* de classificação entre COVID-19, Normal e Pneumonia, e a separação entre treinamento, validação e teste em número de imagens.

	Treinamento	**Validação**	**Teste**	Total
COVID-19	457	114	282	853
Normal	1.069	267	551	1.887
Pneumonia	2.790	698	875	4.363
Total	4.316	1.079	1.708	7.103

Fonte: Autor

Para auxiliar no compreensão da Tabela 3, os valores foram convertidos em porcentagem e apresentados na Tabela 4, pode-se notar que a relação entre cada tipo foi mantida na partição de treino e validação, porem no teste foi a relação foi desfeita a fim de evitar enviesamento da rede.

Tabela 4 – Dispersão das imagens do *dataset* de classificação entre COVID-19, Normal e Pneumonia, e a separação entre treinamento, validação e teste em porcentagem.

	Treinamento	**Validação**	**Teste**	[%]
COVID-19	10,59	10,56	16,51	12,01
Normal	24,77	24,75	32,26	26,57
Pneumonia	64,64	64,69	51,23	61,42
Dataset	60,76	15,19	24,05	100,00

Fonte: Autor

O intuito da rede de classificação, como seu nome sugere, é classificar a imagem fornecida pela segmentação em uma das três categorias, Pneumonia, Normal e Covid-19. Para isso a rede foi treinada utilizando quatro arquiteturas de redes distintas, a ResNet50V2, InceptionResnetV2, VGG-19 e DenseNet121. Após as camadas convolucionais cada uma possuir uma camada artificial densa, essa camada será desconsiderada no trabalho, sendo sempre substituída por uma ANN densa com 3 neurônios, com uma função de ativação *Softmax*.

4.3.2 Hiper Parâmetros dos modelos de classificação

Os hiper parâmetros a seguir foram usados em todos os modelos utilizados no trabalho. A função Erro determina o tipo de função de custo será calculado pela rede, o otimizador definirá qual tipo de otimização de gradiente será usado pelo modelo, a ativação define

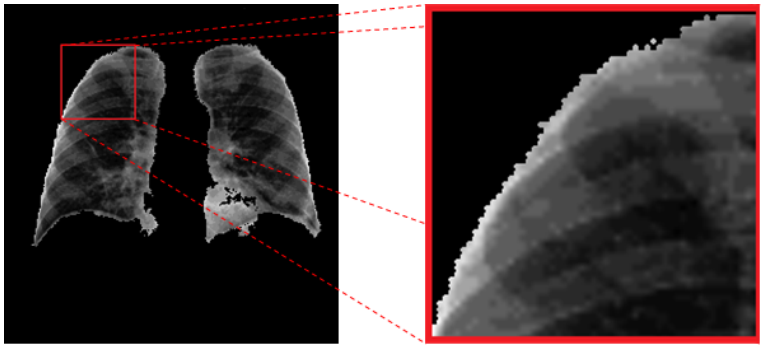
o tipo de ativação usada pela rede convolucionais de classificação, e pôr fim a taxa de aprendizado utilizado pelo otimizador do modelo.

- Erro: foi utilizada a função de perda por entropia cruzada binaria.
- Otimizador: o otimizador utilizado foi o Adamax proposto em (KINGMA; BA, 2015).
- Ativação: foram usadas ReLU em todas as camadas de convolução e densas, e Softmax na camada de saída.
- Taxa de aprendizado: utilizou-se uma taxa de aprendizado de $10e^{-}5$.
- Tamanho do lote: 1 imagem por lote.

4.3.3 Treinamento

No treinamento treinamento visa-se reduzir o erro no escopo local da classificação, onde as redes utilizam apenas um recorte de cada imagem geradas randomicamente antes de cada época. Assim, antes de cada época foi gerado um recorte 224x224 de cada imagem do *dataset* de treinamento e validação, como visto na Figura 35, totalizando 6.241 imagens que necessitam serem geradas, pré processadas e organizadas em vetores alinhados com a sua saída. A saída da rede será um vetor de c posições, onde c é o número de classes, um exemplo de saída seria $[(0,05),(0,12),(0,98)]$.

Figura 35 – Exemplo de um recorte do pulmão a ser usado no treinamento.



Fonte: Autor.

Para se garantir que o recorte gerado pelo algoritmo fossem significativos, adicionou uma condição que caso o somatórios de todos os pixeis não fossem superior a um limitador, o recorte seria eliminado e seria gerado um novo recorte, caso houvesse mais de 10

tentativas mal sucedidas, o valor de limitação diminuiria pela metade, até que atendesse as condições, o exemplo do calculo do limitador pode ser visto a seguir:

$$L = T \cdot 255 \cdot W \cdot H \tag{4.1}$$

Onde L é o limitador, T o limite inicial, variando entre 0 e 1, W o comprimento do recorte e H a altura do recorte, utilizou-se um valor inicial de 0,1 para T.

4.3.4 Definição da classe da imagem

Para determinar a classificação da imagem realiza-se o cálculo de probabilidade para os N números de recorte dessa imagem, soma-se os resultados de todos os recortes e o vencedor é o que possuir a maior probabilidade entre as classes. Supondo que o vetor de saída seja:

$$S_i = [x_i, y_i, z_i] \tag{4.2}$$

Onde S é o vetor de saída para o recorte i, x é a probabilidade da classe Covid-19 para o recorte i, y probabilidade da classe Normal para o recorte i, e z a probabilidade da para o recorte i, pode-se descrever que a classe ganhadora sendo:

$$C = arg_{max}\left(\sum_{i=1}^{K}[x_i, y_i, z_i]\right) \tag{4.3}$$

Onde C é a classe vencedora, K o número de recortes e arg_{max} uma função que retorna o índice de maior valor do vetor.

4.3.5 Avaliação dos Resultados

4.3.5.1 Avaliação dos resultados para classificação

A primeira parte descreve como a rede se comportou durante o treinamento, onde todas foram treinadas utilizando o mesmo método. Os hiper parâmetros utilizados foram os mesmos, o número de épocas máximas treinadas foram os mesmos, todavia foi configurado para que quando a rede não melhorasse o erro em validação após 3 épocas ela pararia o treinamento e os melhores pesos, aqueles que obtiveram o menor valor de erro de validação, seriam salvos. Isso acarretou em diferentes números de épocas para cada rede, visto que algumas redes tinham mais facilidade no treinamento ou eram mais complexas.

A segunda parte consiste na análise da matriz de confusão gerada para 100 imagens em cada rede. A matriz de confusão é utilizada para avaliar métodos de classificação onde as saídas são excludentes, ou seja, a saída só por ser uma entre as opções.

A matriz de confusão nesse trabalho apresentará 9 espaços, a primeira linha ficou os valores que foram preditos como Covid-19, a segunda linha os valores preditos como Normal e na terceira linha os valores preditos como Pneumonia. Enquanto as colunas da matriz de confusão representam os valores reais das imagens, a primeira coluna os resultados para Covid-19, a segunda coluna os resultados para Normal e a terceira linha os resultados para Pneumonia.

Quando ambas, colunas e linhas, possuem o mesmo índice, encontram as predições corretas da rede. Todavia olhar apenas se não há valores fora desses índices não representam um método eficaz. Por exemplo, supondo um exemplo em cada modelo de rede tem um maior valor sobre o primeiro elemento da matriz, todavia ele apresenta maiores valores nos outros elementos da coluna, como saber qual modelo se comporta da melhor forma para cada doença.

Para isso será analisado a precisão, acurácia, revocação e especifidade. A precisão determina a probabilidade de quando o modelo predizer verdadeiro a entrada ser verdadeiro, descrito em:

$$P = \frac{TP}{TP + FP} \tag{4.4}$$

Acurácia é a probabilidade do modelo predizer corretamente a partir de uma entrada, mostrado a seguir:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.5}$$

Revocação é a probabilidade do modelo predizer verdadeiro corretamente a partir de uma entrada verdadeira, cuja a equação é:

$$R = \frac{TP}{TP + FN} \tag{4.6}$$

Especificidade é probabilidade do modelo predizer negativo a partir de uma entrada negativa, cuja a equação é:

$$E = \frac{TN}{TN + FN} \tag{4.7}$$

Onde TP, *True Positive* ou verdadeiramente positivos, são os valores preditos corretamente como verdadeiros, TN, *True Negative* ou verdadeiramente negativo, são os valores preditos corretamente como negativos, FP, *False Positive* ou falsos positivos, são os valores preditos erroneamente como positivos e FN, *False Negative* ou falsos negativos, são os valores preditos erroneamente como negativos.

Na última parte será feita a comparação entre os modelos em relação aos parâmetros analisados na parte dois.

5 Resultados

Nesse capítulo será apresentado os resultados obtidos do trabalho, na primeira parte será apresentado os imagens obtidas a partir do modelo de segmentação e seguido dos valores de classificação da doença.

5.1 Segmentação

Nessa seção será apresentado os resultados obtidos pelo modelo de segmentação do pulmão, utilizando os hiper parâmetros apresentados na seção 4.2.2.2.

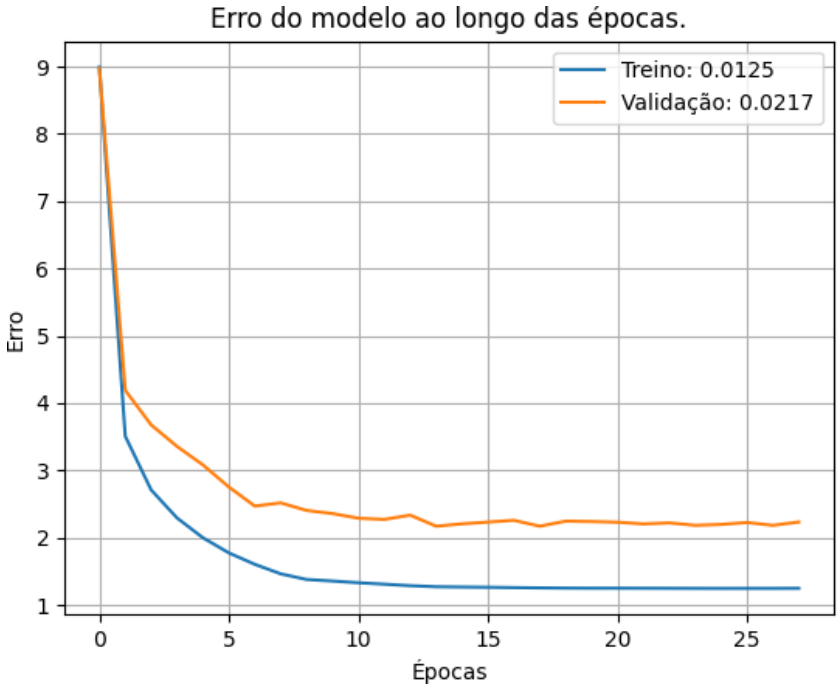
5.1.1 Parâmetros de treinamento do modelo de segmentação

Os parâmetros para verificar a validação do modelo de segmentação foram o erro, acurácia binária e valor de F1, que é a média harmônica entre precisão e revocação, calculado por:

$$F1 = 2\left(\frac{P \cdot R}{P + R}\right) \tag{5.1}$$

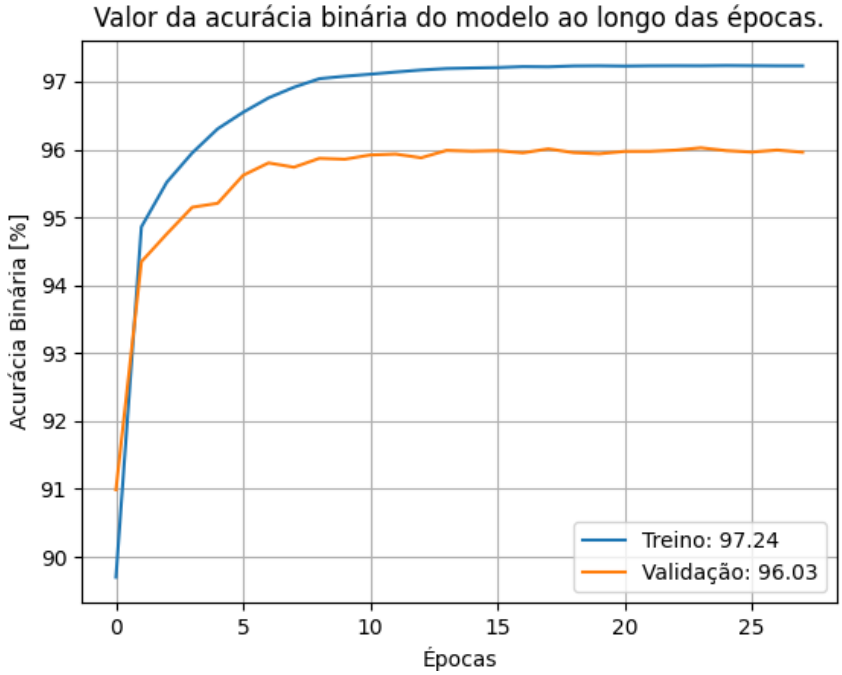
Onde P é a precisão do modelo, R a revocação do modelo e F1 é o valor de saída. Os parâmetros erro, acurácia binária e F1 são mostrados respectivamente na Figura 36, Figura 37 e Figura 38.

Figura 36 – Erro do modelo de segmentação em porcentagem.



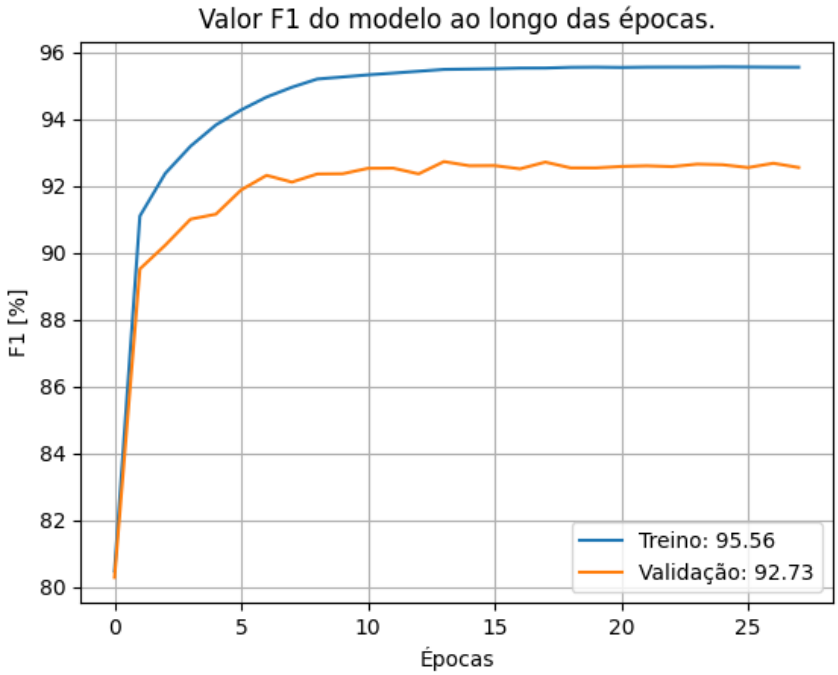
Fonte: Autor.

Figura 37 – Acurácia binária do modelo de segmentação.



Fonte: Autor.

Figura 38 – Valor F1 do modelo de segmentação.



Fonte: Autor.

Os valores foram comparados com a literatura na Tabela 5, o trabalho conseguiu resultados superiores na acurácia e F1, porém esses dados foram retirados do *dataset* de validação.

Tabela 5 – Comparação entre os modelos usado e a literatura.

Modelo	Acurácia	F1
(OH; PARK; YE, 2020)	88,90	84,40
UNet	**96,03**	**92,73**

Fonte: Autor.

5.1.2 Imagens de saída do modelo de segmentação

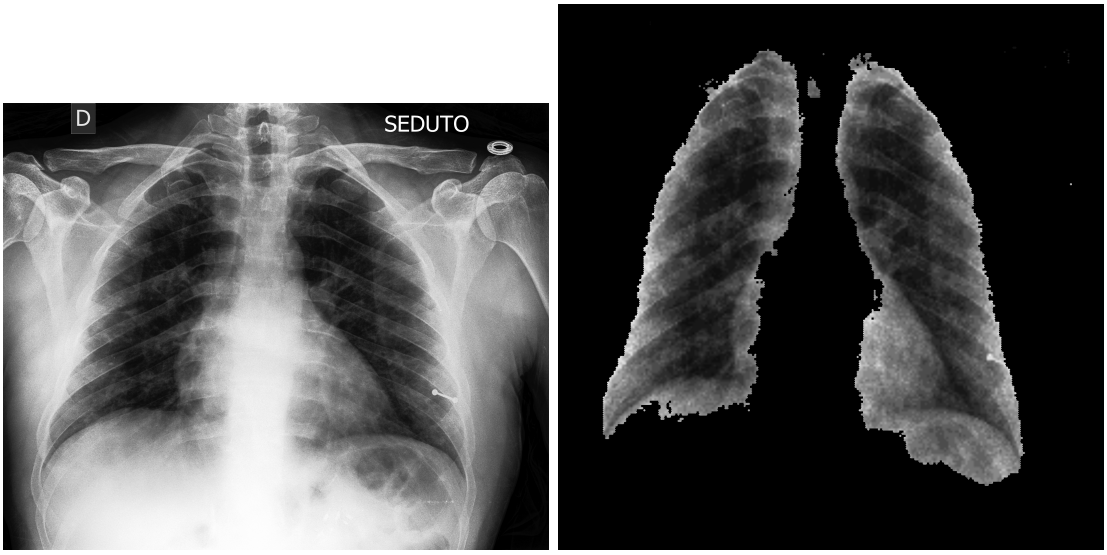
A Figura 39 apresenta os resultados obtidos do modelo de segmentação contendo a comparação entre antes e depois da segmentação. A Figura 39(a) apresenta a imagem do pulmão original com uma resolução 3840x2160 pixeis, e a imagem da Figura 39(c) contém a imagem segmentada do pulmão já redimensionadas com a resolução de 1024x1024 pixeis, que servirá de entrada para o modelo de classificação.

Figura 39 – Segmentação do pulmão com Covid-19. A imagem da esquerda é o pulmão original, no centro a máscara real do pulmão, e a direita a máscara gerada pelo modelo. (a) Imagem original. (b) Máscara original. (c) Imagem gerada.

(a) **(b)** **(c)**

Fonte: Autor.

Figura 40 – Segmentação do pulmão com Covid-19. A imagem da esquerda é o pulmão original e a direita o pulmão com segmentado. (a) Imagem original. (b) Imagem segmentada.



(a) **(b)**

Fonte: Autor.

5.2 Classificação

A seguir serão apresentados os resultados obtidos dos modelos de classificação utilizando os hiper parâmetros na seção 4.3. Todos os modelos foram carregados com os seus

pesos e veis pré treinados, porem houve um treino novamente para uma regulagem mais fina dos pesos. As seções relacionadas as redes serão divididas em quatro partes.

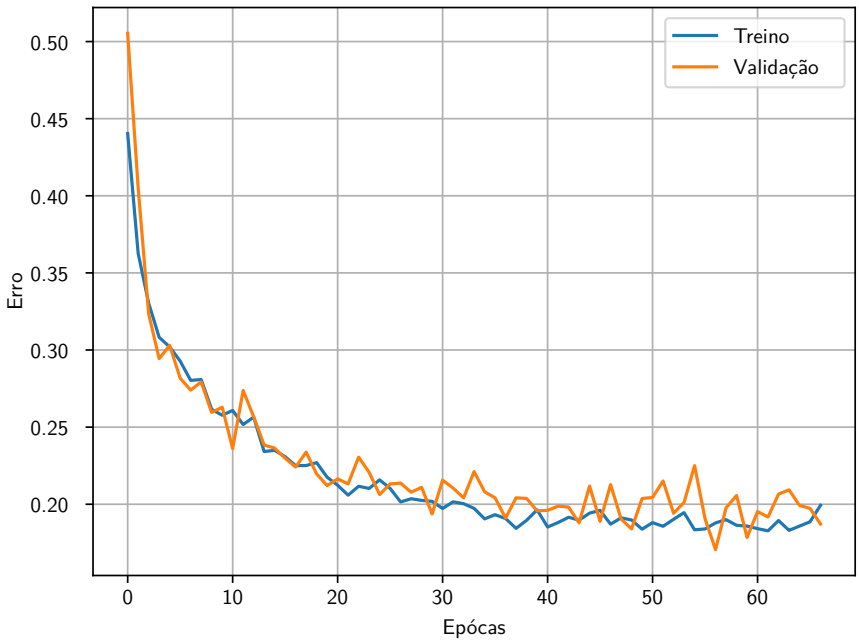
5.2.1 ResNet50V2

A seguir serão apresentados os resultado obtidos pelo modelo de rede ResNet50V2, divido em três partes, treinamento, matriz de confusão e o Grad-CAM Probabilístico do modelo.

5.2.1.1 Treinamento

A Figura 41 apresenta o valor do erro ao longo das épocas durante o treinamento da rede ResNet50V2, em azul se encontra o erro para o *dataset* de treino, e em vermelho o valor do erro para o *dataset* de validação.

Figura 41 – Gráfico do erro ao longo das épocas do treinamento.

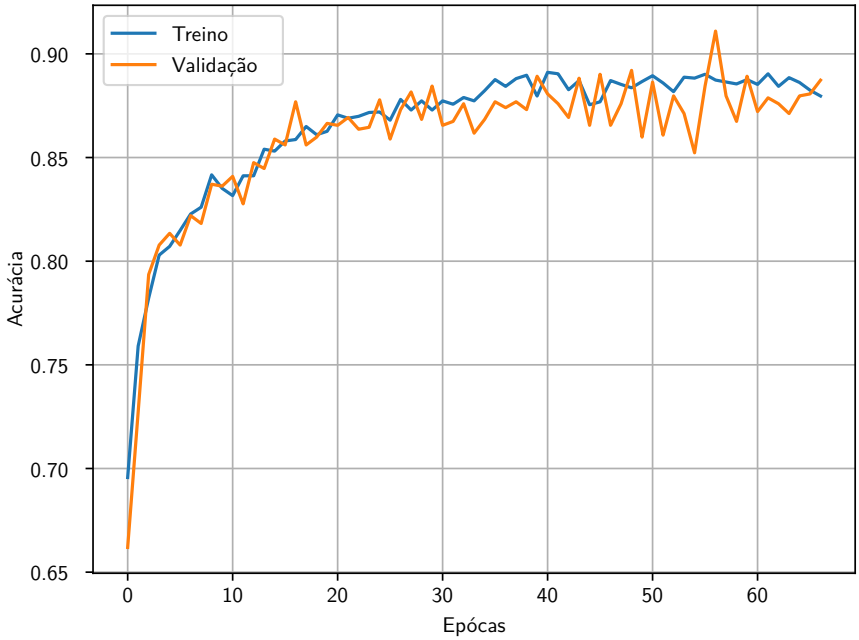


Fonte: Autor.

Tanto o erro de validação quanto o erro de treino se mostraram bem próximos em todas as época, terminando em valores abaixo de 0,20 de erro.

A Figura 42 apresenta os valores da acurácia do modelo ao longo do treinamento dos modelos. Seu comportamento se mostrou semelhante ao erro, porém as oscilações foram maiores do que o treino. Uma forma de diminuir as oscilações seria a inserção de mais imagem no modelo.

Figura 42 – Gráfico do acurácia ao longo das épocas do treinamento.

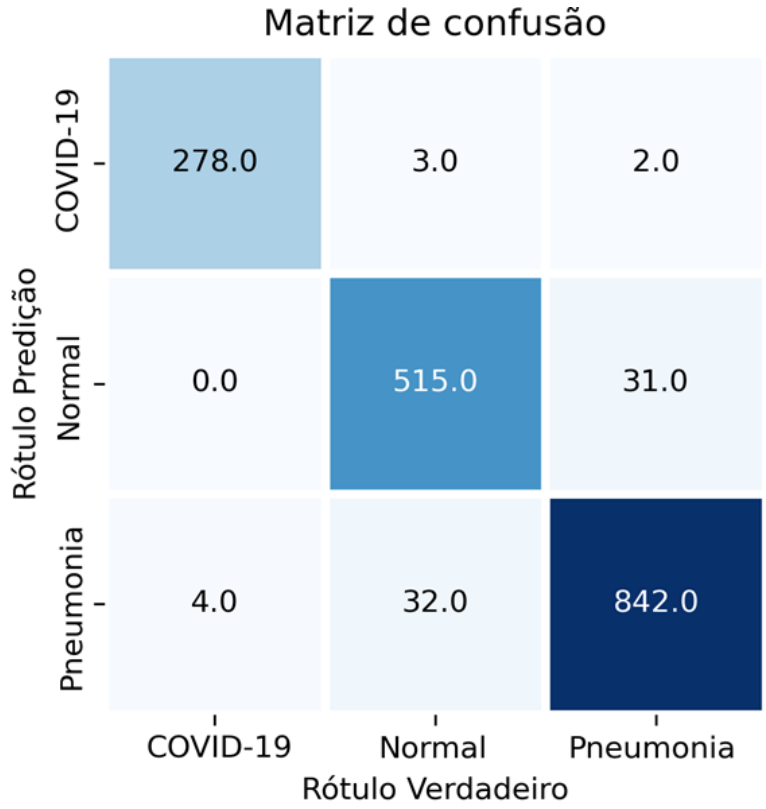


Fonte: Autor.

5.2.1.2 Matriz de Confusão

A matriz de confusão para o modelo de ResNet50V2 com 100 recortes da imagem pode ser vista na Figura 43, utilizando o *dataset* de testes contendo 1707 imagens. É possível observar que a matriz diagonal principal contém a maioria dos valores da matriz, mostrando que o modelo apresenta bons resultados.

Figura 43 – Matriz de confusão utilizando o modelo ResNet50V2 para 100 cortes da imagem.



Fonte: Autor.

Para melhor análise foi feita a Tabela 6 utilizando os valores da Figura 43. O modelo ResNet50V2 teve as melhores características para o Covid-19, apresentando todos os valores acima de 95% nos parâmetros analisados.

Tabela 6 – Avaliação do modelo ResNet50V2 utilizando 100 imagens.

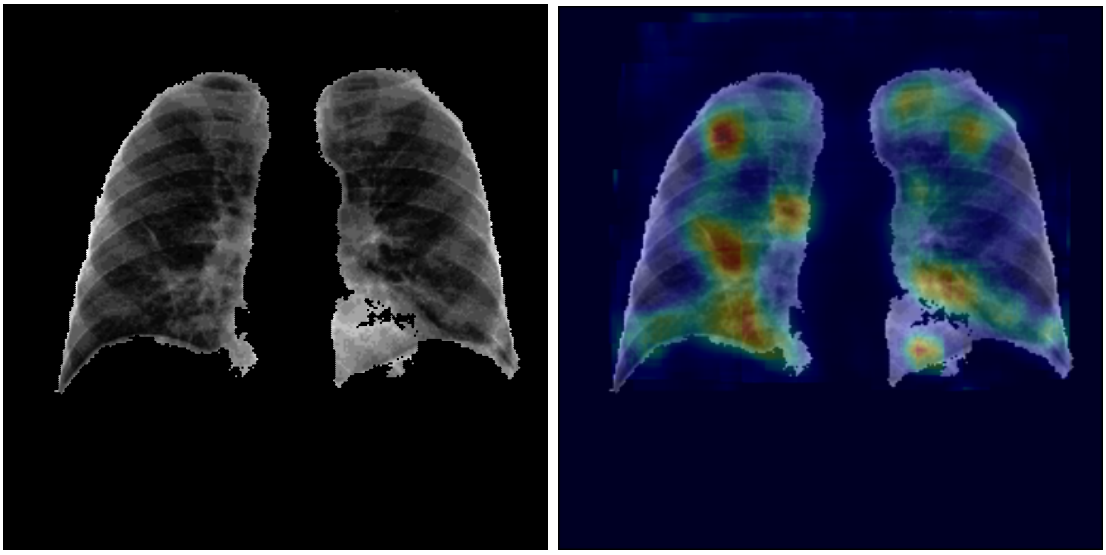
Tipo	Covid-19	Normal	Pneumonia
VP	278	515	842
VN	1420	1126	796
FP	5	31	36
FN	4	35	33
P	98,23	94,32	95,90
A	99,47	96,13	95,96
R	98,58	93,64	96,23
E	99,72	96,98	96,02

Fonte: Autor.

5.2.1.3 Grad-CAM Probabilística

A Figura 44(a) apresenta a imagem original de um pulmão com Covid-19, e a Figura 44(b) mostra o Grad-CAM Probabilístico sobreposto da imagem original, as partes em vermelhas são as regiões que mais ativaram o modelo ResNet50V2.

Figura 44 – Imagem original do pulmão. (a) Original. (b) Grad-CAM.



(a) **(b)**

Fonte: Autor.

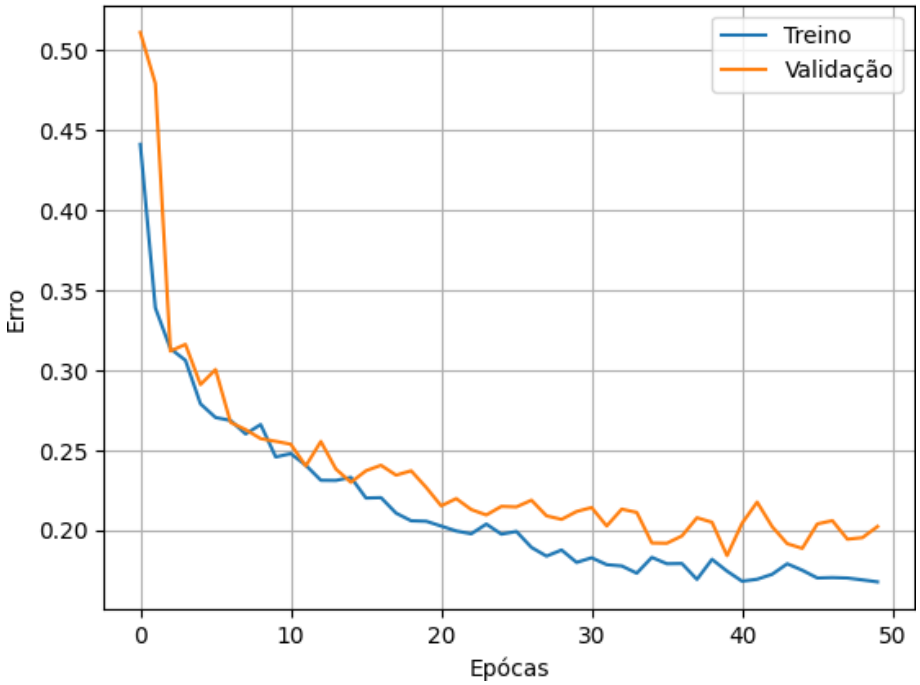
5.2.2 DenseNet121

A seguir será apresentado os resultados obtidos para a rede DenseNet121, mostrando o resultado através da Matriz de Confusão, parâmetros de avaliação ao longo do tempo e o mapa *Grad-CAM* Probabilístico gerado pelo modelo.

5.2.2.1 Treinamento

A Figura 45 apresenta o erro do modelo ao longo das épocas para o *dataset* de treino. Em azul tem-se o erro ao longo do treino e em vermelho o erro durante a validação.

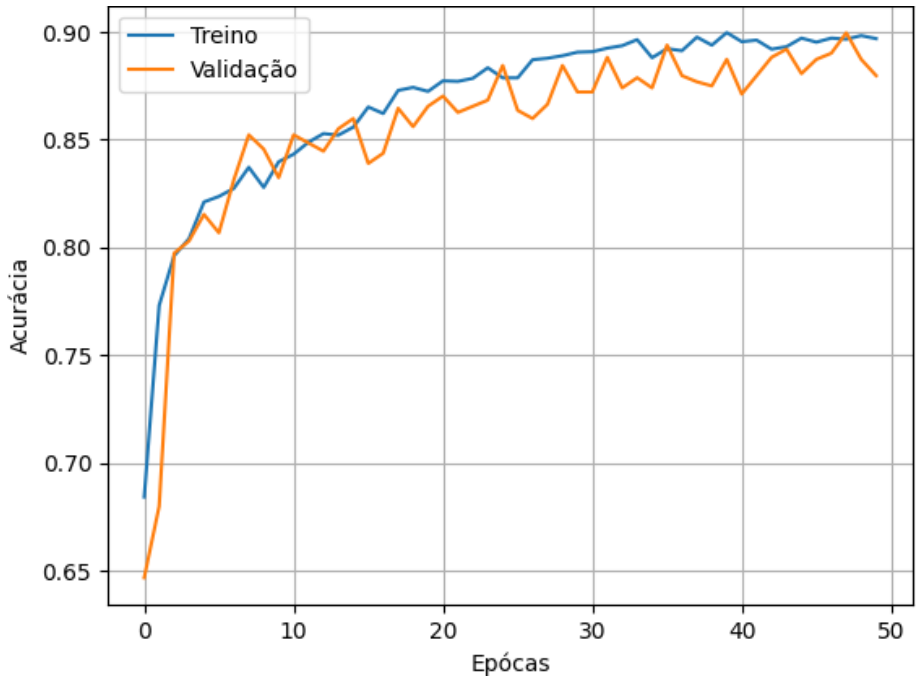
Figura 45 – Gráfico do erro ao longo das épocas do treinamento.



Fonte: Autor.

A Figura 46 mostra a acurácia do modelo ao longo das épocas durante o treinamento. Em azul os valores do erro para o *dataset* do treino, e em vermelho o erro durante o *dataset* validação.

Figura 46 – Gráfico do erro ao longo das épocas do treino.



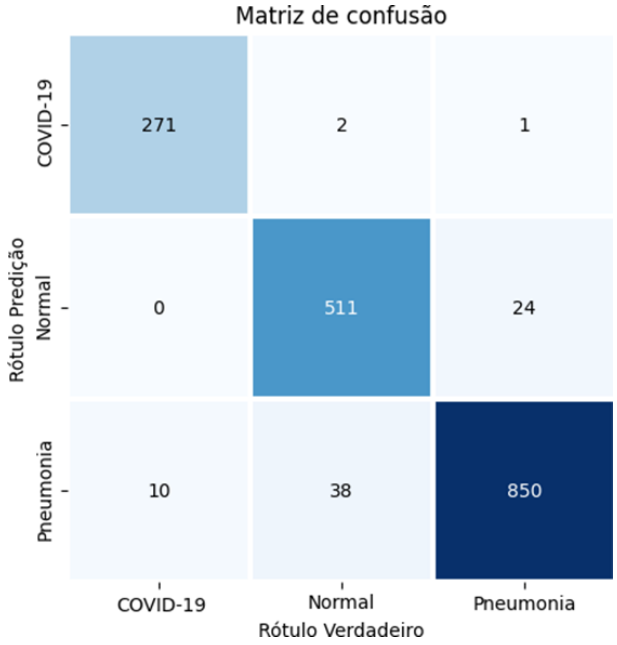
Fonte: Autor.

5.2.2.2 Matriz de Confusão

Matrizes de confusão são utilizadas para a melhor apresentação dos resultados. Ela é composta por dois eixos, no eixo horizontal se encontram os valores reais e no eixo vertical os valores preditos. Na diagonal central se encontra as predições corretas do modelo, qualquer valor fora dessa diagonal são erros do modelo.

Para melhor interpretação do modelo, é mostrado a matriz de confusão na Figura 47 obtida a partir de 100 recortes por imagem.

Figura 47 – Matriz de confusão utilizando o modelo DenseNet121.



Fonte: Autor.

Os parâmetros foram calculados na Tabela 7 para melhor avaliação do modelo. O pior desempenho do sistema foi para a revocação de um pulmão Normal, obtendo apenas 92,74%, enquanto o sistema obteve o melhor desempenho na especifidade do Covid-19, onde chegou a 99,30%.

Tabela 7 – Avaliação do modelo DenseNet121 utilizando 100 imagens.

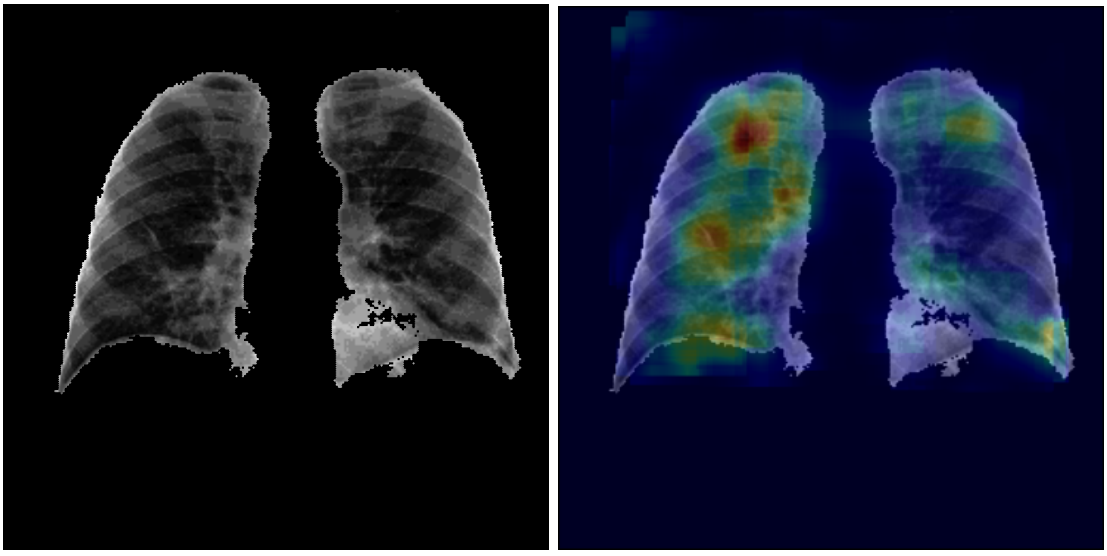
Tipo	Covid-19	Normal	Pneumonia
VP	271	511	850
VN	1423	1132	784
FP	3	24	48
FN	10	40	25
P	98,91	95,51	94,65
A	99,24	96,25	95,72
R	96,44	92,74	97,14
E	99,30	96,59	96,91

Fonte: Autor.

5.2.2.3 Grad-CAM Probabilística

A Figura 48(b) apresenta o Grad-CAM Probabilístico, gerado a partir de 400 cortes, sobreposto a imagem segmentada do pulmão, a áreas em vermelho representa a região que mais ativou o modelo, enquanto as áreas em azul as regiões de pouco interesse.

Figura 48 – Grad-CAM gerado pela rede DenseNet121. (a) Original. (b) Grad-CAM.



(a) **(b)**

Fonte: Autor.

5.2.3 InceptionResnetV2

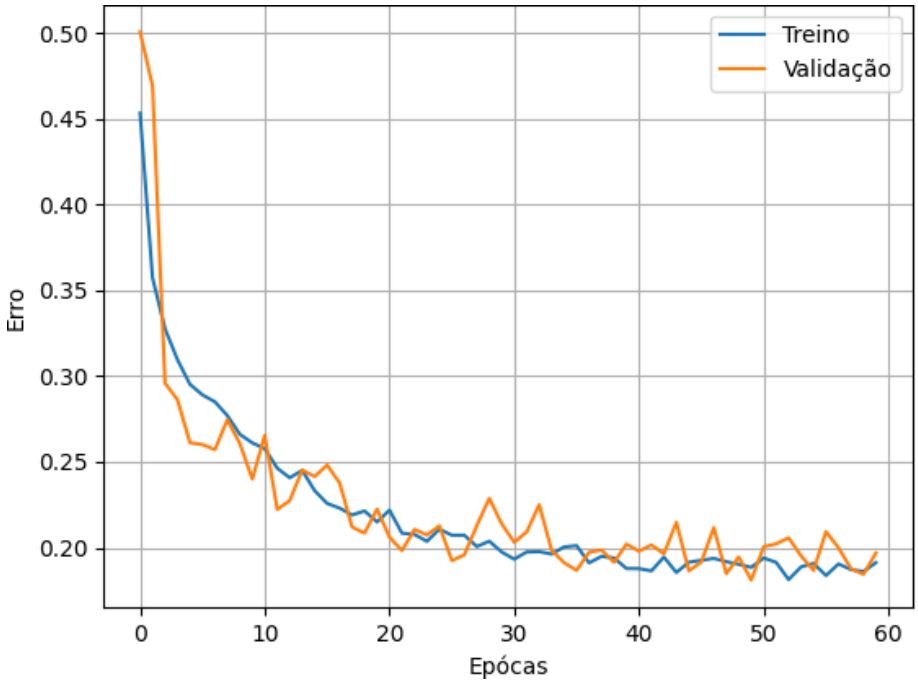
5.2.3.1 Treinamento

O treinamento da rede InceptionResnetV2 foi realizado utilizando o *dataset* de treino e validação, os resultados obtidos durante este processo foram disposto nas Figura 49,

apresentando o erro, e 50, apresentando a acurácia do treinamento.

A Figura 49 mostra que o erro tanto no treino e validação ficaram abaixo dos 0,20 no escopo local, levando quase 60 épocas para encontrar os valores para os pesos.

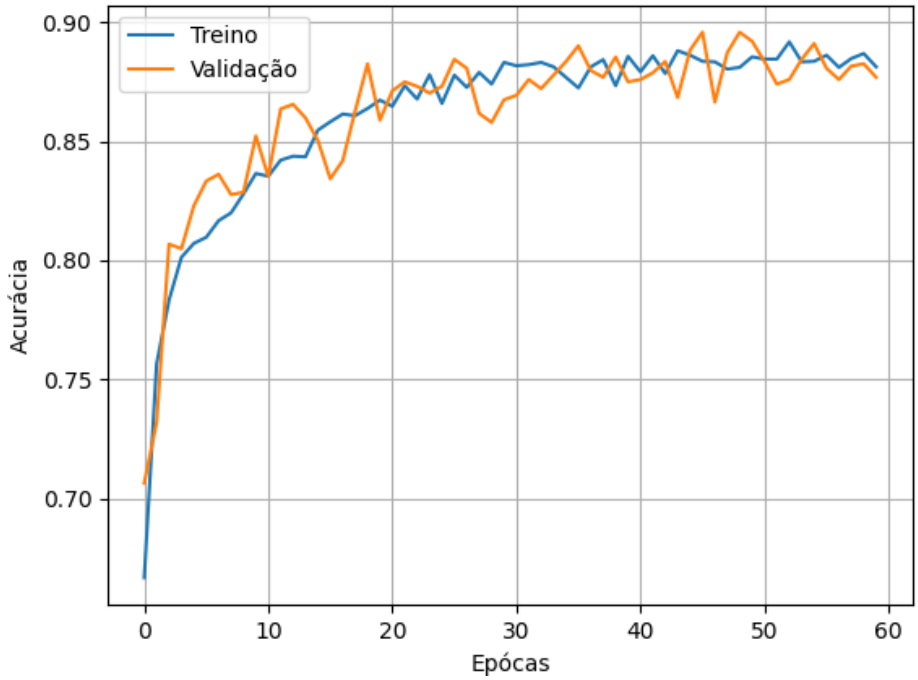
Figura 49 – Erro do modelo InceptionResnetV2 durante o treinamento.



Fonte: Autor.

A Figura 50 apresenta a acurácia do modelo no escopo local, onde o treino e a validação ficaram acima de 87%.

Figura 50 – Erro do modelo InceptionResnetV2 durante o treinamento.

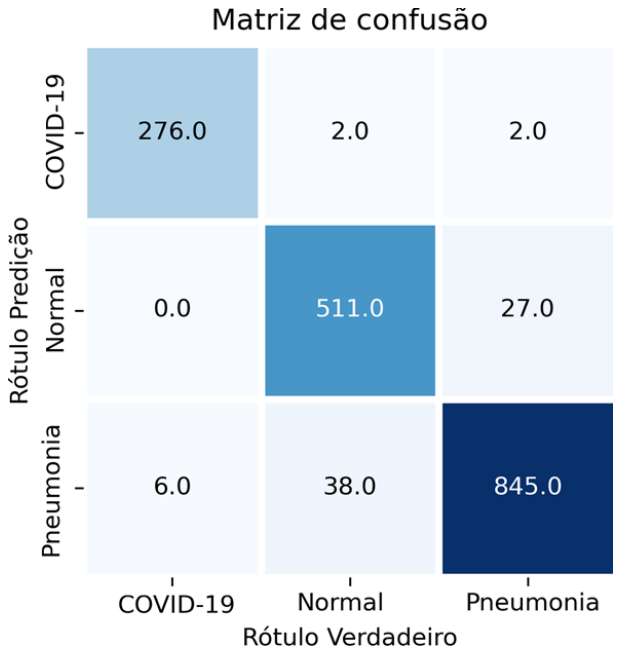


Fonte: Autor.

5.2.3.2 Matriz de Confusão

A Figura 51 apresenta a matriz de confusão para o modelo InceptionResnetV2. A matriz foi gerada utilizando 1707 imagens, realizados 100 cortes por imagem.

Figura 51 – Matriz de confusão do modelo utilizando a rede InceptionResnetV2 para 100 cortes.



Fonte: Autor.

A Tabela 8 apresenta os valores da Figura 51 organizados em verdadeiros positivos (VP), verdadeiros negativos (VN), falsos positivos (FP), falsos negativos (FN), precisão (P), acurácia (A), revocação (R) e especificidade (E). O modelo conseguiu seu melhor resultado na especificidade do Covid-19, e o pior resultado foi para a revocação de um pulmão Normal, já para a Pneumonia o modelo ficou entre 96%.

Tabela 8 – Avaliação do modelo InceptionResnetV2 utilizando 100 imagens.

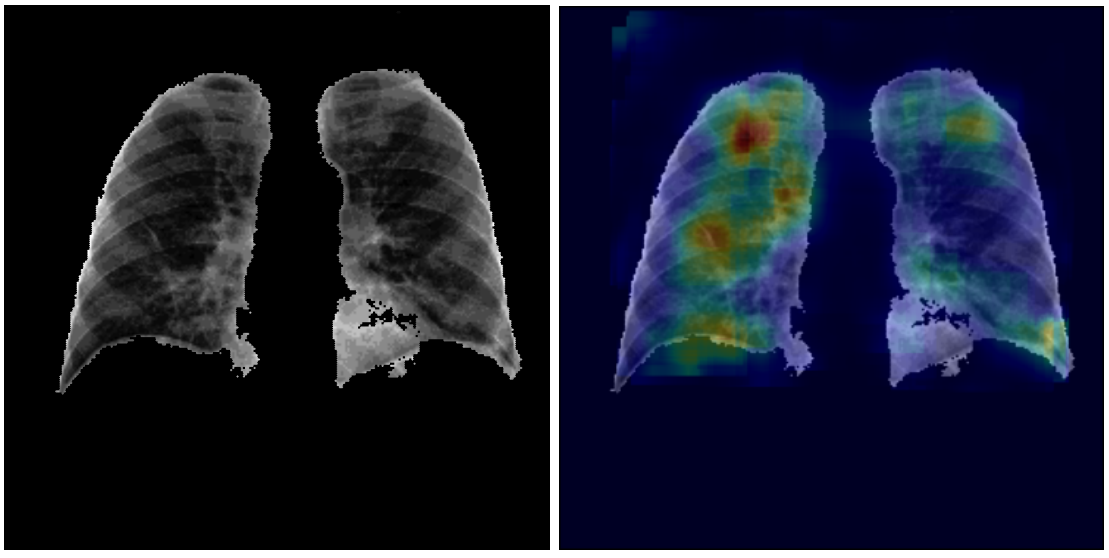
Tipo	**Covid-19**	**Normal**	**Pneumonia**
VP	276	511	845
VN	1421	1129	789
FP	4	27	44
FN	6	40	29
P	98,57	94,98	95,05
A	99,41	96,07	95,72
R	97,87	92,74	96,68
E	99,58	96,58	96,45

Fonte: Autor.

5.2.3.3 Grad-CAM Probabilístico

A Figura 52 mostra a comparação entre imagem original, Figura 52(a), e o Grad-CAM Probabilístico, Figura 52(b), obtido através do modelo InceptionResnetV2, apresentando áreas de grande interesse no pulmão esquerdo na parte superior.

Figura 52 – Grad-CAM Probabilístico gerado pela rede InceptionResnetV2 para 400 recortes. (a) Original. (b) Grad-CAM.



(a) **(b)**

Fonte: Autor.

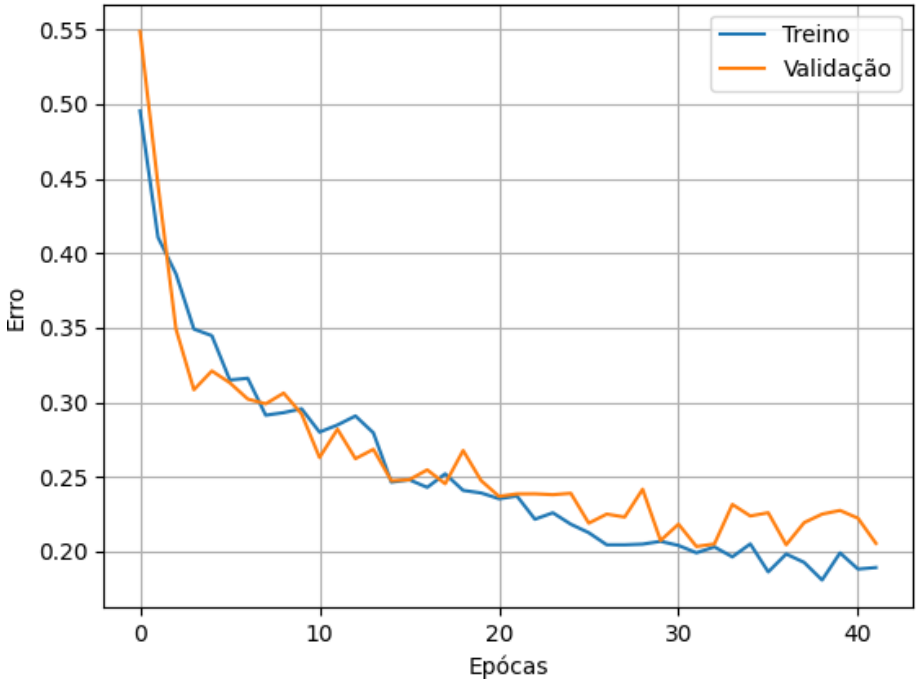
5.2.4 VGG-19

A seguir apresentará os resultados obtidos para o modelo do VGG-19 durante o treinamento, validação e teste.

5.2.4.1 Treinamento

A Figura 53 apresenta o erro do modelo VGG-19 durante o treinamento, em azul o valor do erro para o *dataset* de treino e em vermelho o *dataset* de validação.

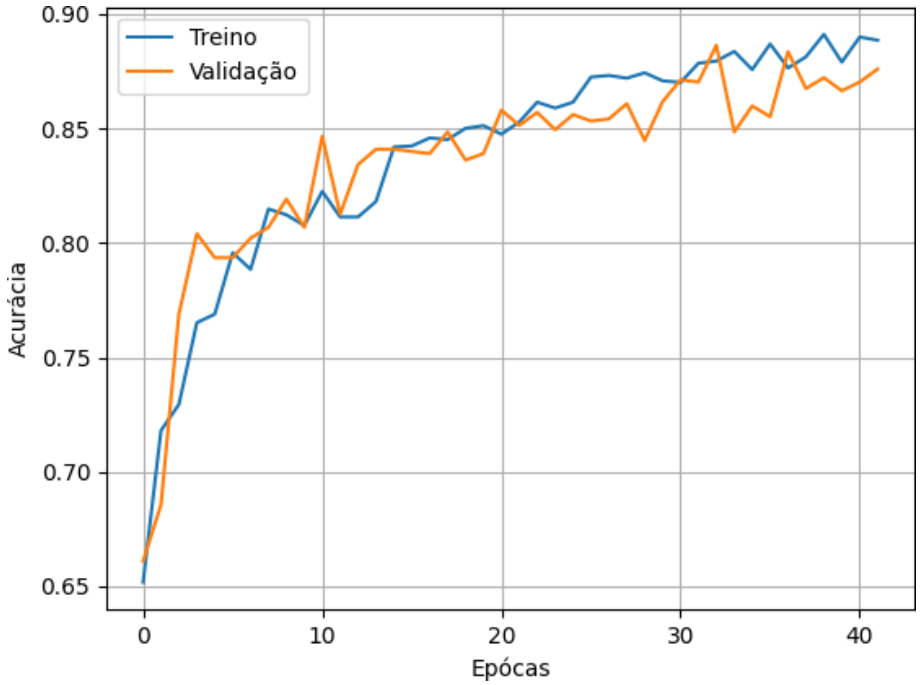
Figura 53 – Erro durante a etapa de treinamento da rede VGG-19.



Fonte: Autor.

A Figura 54 apresenta a acurácia do modelo ao longo do treinamento no escopo local, em azul tem-se os valores para o treino e em vermelho para validação. A acurácia em ambos os *datasets* ficaram abaixo dos 0,25 para o escopo local do modelo.

Figura 54 – Acurácia do modelo durante a etapa de treinamento da rede VGG-19.

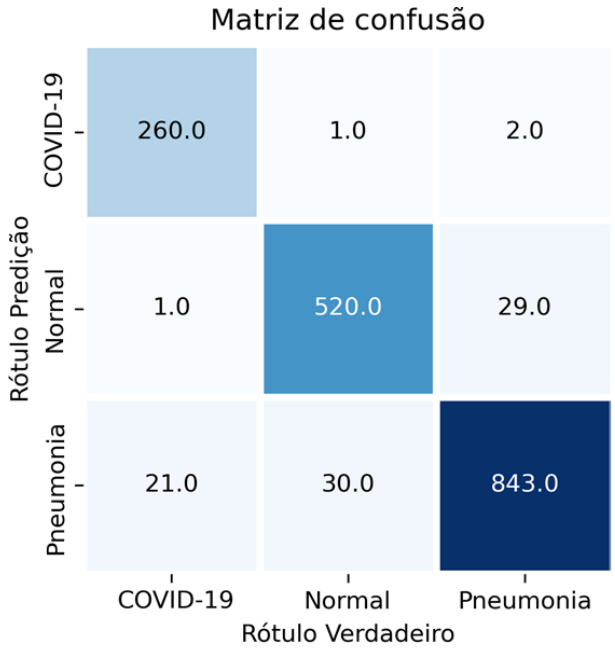


Fonte: Autor.

5.2.4.2 Matriz de Confusão

A Figura 55 apresenta a matriz de confusão do modelo VGG-19 para o *dataset* de teste utilizando 100 recortes.

Figura 55 – Matriz de confusão de rede VGG-19 para 100 recortes por imagem.



Fonte: Autor.

Para melhor avaliação do resultado do modelo VGG-19 foi elaborado a Tabela 9, contendo a precisão (P), acurácia (A), revocação (R) e especificidade (E).

Tabela 9 – Avaliação do modelo VGG-19 utilizando 100 imagens.

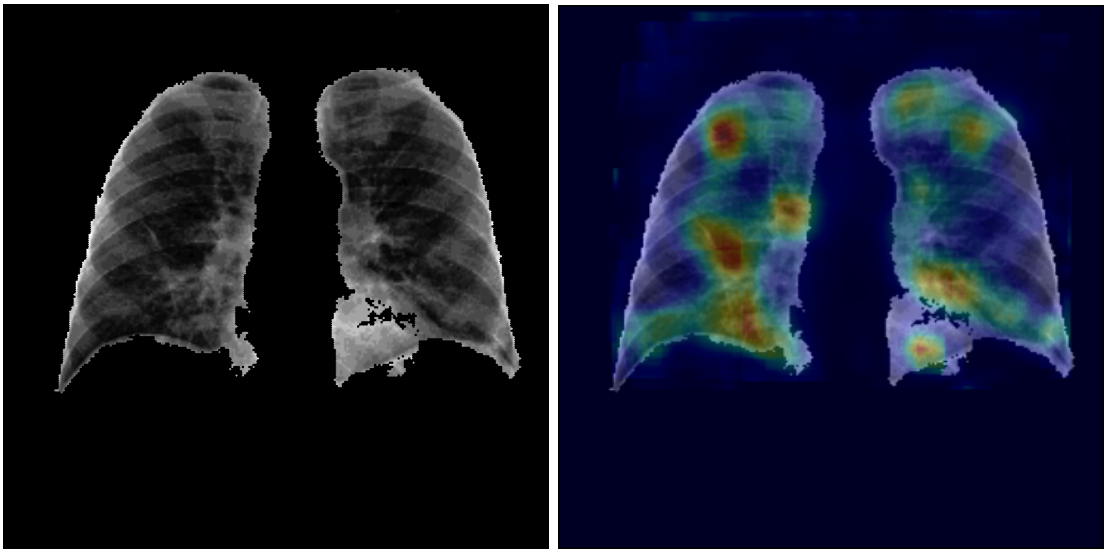
Tipo	**Covid-19**	**Normal**	**Pneumonia**
VP	260	520	843
VN	1422	1126	782
FP	3	30	51
FN	22	31	31
P	98,86	94,55	94,10
A	98,54	96,37	95,06
R	92,20	94,37	96,33
E	98,48	97,32	96,19

Fonte: Autor.

5.2.4.3 Grad-CAM Probabilístico

A Figura 56 compara a imagem original, na Figura 56(a), e o Grad-CAM Probabilístico gerado pelo modelo VGG-19 para 400 recortes, mostrado na Figura 56(b).

Figura 56 – Grad-CAM Probabilísticos para uma imagem de 400 recortes. (a) Original. (b) Grad-CAM.



(a) **(b)**

Fonte: Autor.

5.2.5 Comparação entre as Redes

Tabela 10 – Comparação das métricas de avaliação para Covid-19 usando 100 recortes.

Métricas	**ResNet50V2**	**DenseNet121**	**InceptionResnetV2**	**VGG-19**
VP	**278**	274	276	260
VN	1420	**1423**	1420	1422
FP	5	**2**	4	3
FN	**4**	8	6	22
Precisão	98,23	**99,28**	98,57	98,86
Acurácia	**99,47**	99,41	99,41	98,54
Revocação	**98,58**	97,16	97,87	92,20
Especifidade	**99,72**	99,44	99,58	98,48
Parâmetros	23.570.980	**7.040.613**	54.341.381	20.025.957

A Tabela 10 contem os resultados de cada métrica das redes testadas, VP, VN, FP, FN, precisão, acurácia, revocação, especificidade e o número de pesos. Os valores foram obtidos utilizando 100 recortes de cada imagem do *dataset* de teste. A última linha da tabela se refere ao número de pesos e parâmetros das camadas, sendo que quanto menor seu valor, menor o consumo de memória da rede.

Apesar da rede ResNet50V2 se mostrar a melhor nas métricas de VP, precisão, acurácia, revocação e especificidade, ela possui um elevado número de parâmetros se comparado a rede DenseNet121. Já a rede InceptionResnetV2 obteve bons resultados em quase todos as métricas, mas não se mostrou melhor em nenhum e ainda exigiu o maior número de parâmetros. Enquanto, a VGG-19 não se mostrou a melhor em nenhuma métrica, tendo o pior resultado na maioria delas. Por fim, a rede DenseNet121 que obteve a melhor precisão entre todas as redes, mesmo possuindo o menor número de parâmetros, e obteve métricas próximas as demais redes.

Todavia, todas as redes poderiam ser utilizadas como auxílio no diagnostico de Covid-19, visto que apresentaram um valor de precisão superior a 69%, obtida pela classificação dos médicos dos exames de raio-x em (OH; PARK; YE, 2020).

Comparando ainda com o resultados obtidos em (OH; PARK; YE, 2020) na Tabela 11, que apresenta melhores valores de cada coluna em negritos, mostra que apesar de possuir uma menor especificidade para Covid-19 se comparado ao trabalho de (OH; PARK; YE, 2020), todos os outros parâmetros se mostraram superiores, mostrando que a alteração dos modelos de redes utilizados pode gerar um ganho na precisão e especifidade do sistema.

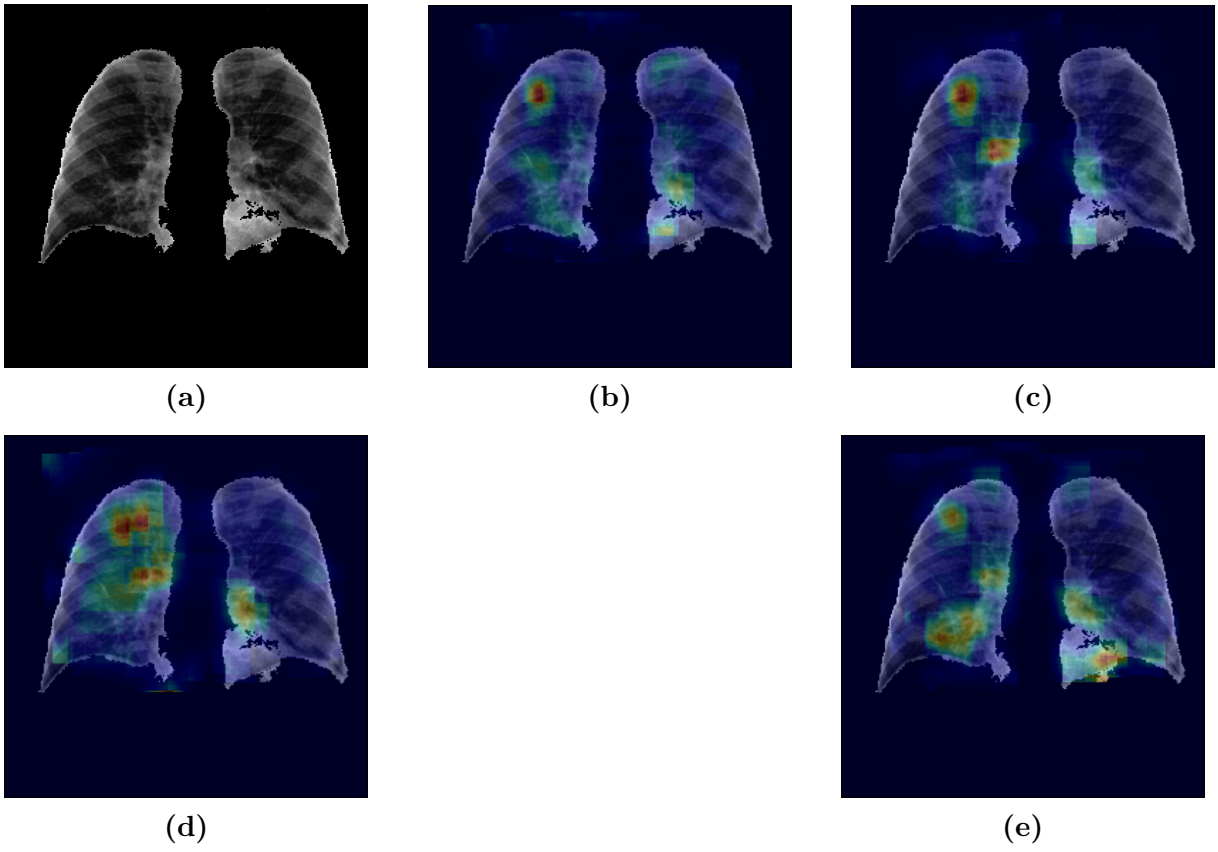
Tabela 11 – Comparação entre precisão e revocação do atual trabalho com a literatura.

	Precisão			**Especificidade**		
	Covid-19	**Normal**	**Pneumonia**	**Covid-19**	**Normal**	**Pneumonia**
(OH; PARK; YE, 2020)	90,3	95,7	90,3	**100**	90	93
ResNet50V2	98,23	94,32	**95,90**	99,72	96,98	96,02
DenseNet121	**98,91**	**95,51**	94,65	99,30	96,59	**96,91**
InceptionResnetV2	98,57	94,98	95,05	99,58	96,58	96,45
VGG-19	98,86	94,55	94,10	98,48	**97,32**	96,19

5.2.5.1 Grad-CAM Probabilísticos

As figuras a seguir apresentam a comparação entre os diversos modelos de redes utilizados na geração dos Grad-CAM Probabilísticos utilizando 100 recortes, Figura 57, e 400 recortes, Figura 58.

Figura 57 – Comparação entre os Grad-CAM Probabilísticos gerados a partir de 100 recortes por rede. (a) Original. (b) ResNet50V2. (c) DenseNet121. (d) InceptionResnetV2. (e) VGG-19.

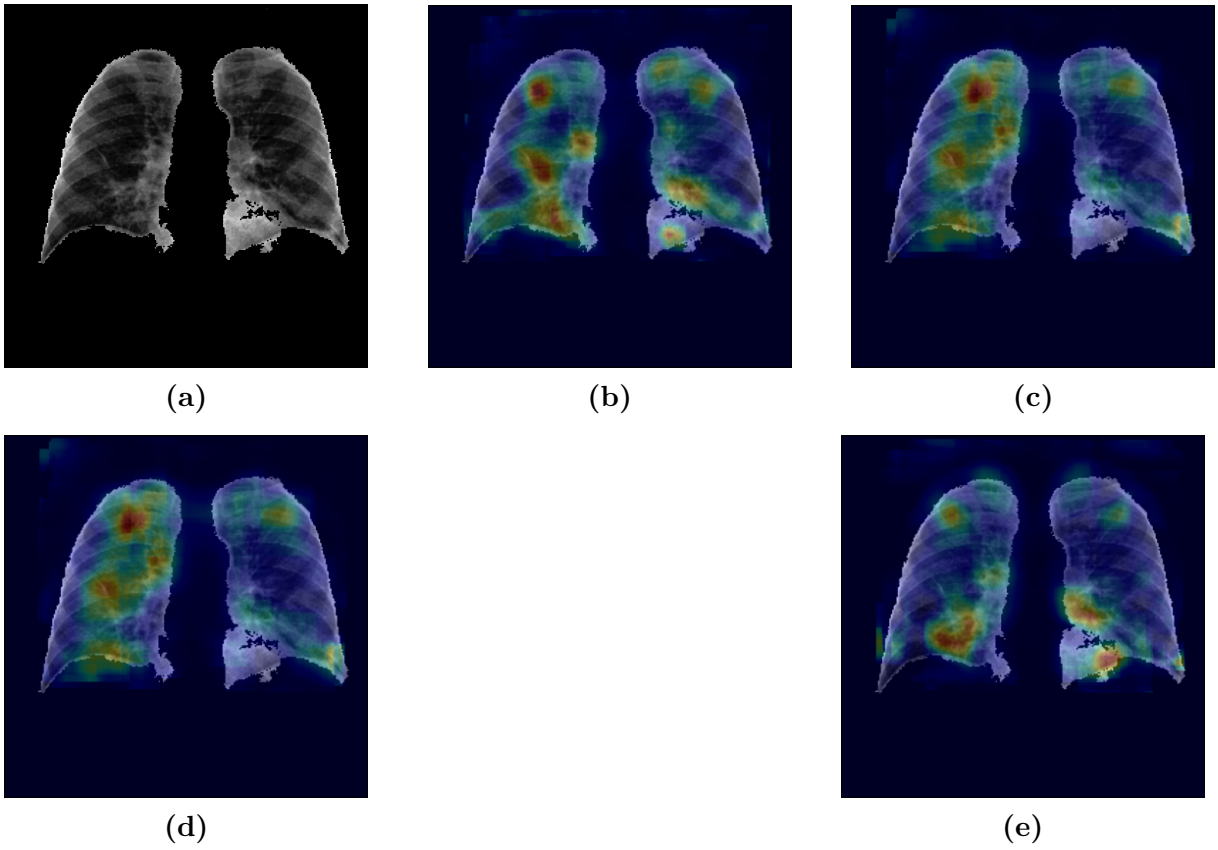


(a) **(b)** **(c)**

(d) **(e)**

Fonte: Autor.

Figura 58 – Comparação entre os Grad-CAM Probabilísticos gerados a partir de 400 recortes por cada rede. (a) Original. (b) ResNet50V2. (c) DenseNet121. (d) InceptionResnetV2. (e) VGG-19.



(a) **(b)** **(c)**

(d) **(e)**

Fonte: Autor.

As regiões em vermelho em cada mapa representam os locais que mais acionaram cada rede e as regiões em azul os locais aonde não acionaram a rede neural.

Nota-se em todas as imagens as regiões em vermelho se apresenta em maior parte no pulmão a esquerda da imagem, tendo a parte superior e inferior ressaltadas, e em locais semelhantes porem com diferentes intensidades.

As diferentes das áreas em vermelho, entre as imagens de 100 e 400 recortes, ocorre devido a aleatoriedade do sistema, a adição de mais recortes na imagem minimiza esse efeito.

6 Conclusão

O presente trabalho propôs um sistema para auxílio no diagnóstico de doença pulmonares, utilizando *dataset* públicos para o seu treinamento, com o objetivo de encontrar os modelos para a segmentação, classificação e severidade da doença.

A segmentação e classificação do pulmão, como mostrado no capítulo anterior, obtiveram resultados semelhantes a literatura, com ressalva para a classificação, que mostrou resultados promissores.

A ResNet50V2 se mostrou superior em quase todos os aspectos, porém a pouca diferença em valores dos parâmetros probabilísticos analisados, aliado a significativo número de parâmetros da ResNet50V2, fazem da DenseNet121 a melhor escolha em quase todas as situações.

As imagens de Grad-CAM Probabilísticos geradas pelos modelos apresentaram regiões de maior interesse em pontos semelhantes em quase todas as redes, exceto na rede VGG-19 que ativou regiões diferentes das demais.

Os algoritmos utilizados nesse trabalho podem ser encontrados na pagina localizada no GitHub: <https://github.com/GustavoChichanoski/COVID>.

6.1 Trabalhos Futuros

Buscando a continuidade do trabalho, são listados a seguir possíveis contribuições que visam adicionar novas fincões ou otimizar os resultados obtidos.

- Os dados obtidos durante esse trabalho podem ser usados posteriormente para o treinamento de uma rede de classificação da severidade, utilizando as imagens de Grad-CAM geradas pelo modelo de classificação como entrada.
- Geração de um dataset maior para a melhor verificação das qualidades dos resultados obtidos.
- Geração do um modelo de classificação para radiologias em 3D.

Referências

AGARAP, A. F. Deep Learning using Rectified Linear Units (ReLU). n. 1, p. 2–8, 2018. Disponível em: <http://arxiv.org/abs/1803.08375>.

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, v. 2018-Janua, n. August, p. 1–6, 2018.

BALAS, V. E.; MISHRA, B. K.; KUMAR, R. *HANDBOOK OF DEEP LEARNING IN BIOMEDICAL ENGINEERING: Techniques and Applications*. [S.l.: s.n.], 2021. 307 p. ISBN 9780128230145.

BUSLAEV, A. et al. Albumentations: Fast and flexible image augmentations. *Information*, v. 11, n. 2, 2020. ISSN 2078-2489. Disponível em: <https://www.mdpi.com/2078-2489/11/2/125>.

CAO, Y. et al. Longitudinal Assessment of COVID-19 Using a Deep Learning–based Quantitative CT Pipeline: Illustration of Two Cases. *Radiology: Cardiothoracic Imaging*, v. 2, n. 2, p. e200082, 2020.

CHO, S.; JANG, H.; TAN, J. W. DeepScribble : Interactive Pathology Image Segmentation Using Deep Neural Networks with Scribbles Korea University , College of Informatics , Department of Computer Science and Engineering , Seoul , Korea Ulsan National Institute of Science and Technology. p. 2–6, 2021.

De Oliveira Lima, C. M. A. Information about the new coronavirus disease (COVID-19). *Radiologia Brasileira*, v. 53, n. 2, p. v–vi, 2020. ISSN 01003984.

DONG, D. et al. The Role of Imaging in the Detection and Management of COVID-19: A Review. *IEEE Reviews in Biomedical Engineering*, v. 14, n. c, p. 16–29, 2021. ISSN 19411189.

EISENBERG, R. L. *Clinical Imaging: An Atlas of Differential Diagnosis*. 5th. ed. [S.l.]: LWW, 2010. (Eisenberg). ISBN 0781788609,9780781788601.

GÉRON, A. *Hands-On Machine Learing With Scikit-Learn & Tensor Flow*. O'Reilly Media, 2017. 760 p. ISSN 1662-5196. ISBN 1662-5196 (Electronic)1662-5196 (Linking). Disponível em: <http://arxiv.org/abs/1412.3919>.

GONZALEZ, T. F. ImageNet Classification with Deep Convolutional Neural Network. *Handbook of Approximation Algorithms and Metaheuristics*, p. 1–1432, 2007.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [s.n.], 2016. 1–3 p. Disponível em: <http://www.deeplearningbook.org/front_matter.pdf>.

GORDIENKO, Y. et al. Deep learning with lung segmentation and bone shadow exclusion techniques for chest x-ray analysis of lung cancer. *Advances in Intelligent Systems and Computing*, v. 754, p. 638–647, 2019. ISSN 21945357.

HE, K. et al. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, v. 2016-Decem, p. 770–778, 2016. ISSN 10636919.

HUANG, G. et al. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, v. 2017-January, p. 2261–2269, 2017.

HUANG, T. S. Computer Vision: Evolution and Promise. *Report*, 1997.

JADON, S. A survey of loss functions for semantic segmentation. *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology, CIBCB 2020*, 2020.

JAEGER, S. et al. Two public chest X-ray datasets for computer-aided screening of pulmonary diseases. *Quantitative imaging in medicine and surgery*, v. 4, n. 6, p. 475–477, 2014. ISSN 2223-4292.

KINGMA, D. P.; BA, J. L. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, p. 1–15, 2015.

MADER, K. S. *Pulmonary Chest X-Ray Abnormalities*. 2017. Acessado em 19/06/2021. Disponível em: <https://www.kaggle.com/datasets/kmader/pulmonary-chest-xray-abnormalities>.

NARIN, A.; KAYA, C.; PAMUK, Z. Automatic detection of coronavirus disease (covid-19) using x-ray images and deep convolutional neural networks. *arXiv preprint arXiv:2003.10849.*, 2020. Disponível em: <https://arxiv.org/abs/2003.10849>.

OH, Y.; PARK, S.; YE, J. C. Deep Learning COVID-19 Features on CXR Using Limited Training Data Sets. *IEEE Transactions on Medical Imaging*, v. 39, n. 8, p. 2688–2700, 2020. ISSN 1558254X.

ORGANIZATION, W. H. *Naming the coronavirus disease (COVID-19) and the virus that causes it.* 2020. Disponível em: <https://www.who.int/emergencies/diseases/novel-coronavirus-2019/technical-guidance/naming-the-coronavirus-disease-(covid-2019)-and-the-virus-that-causes-it>.

ORGANIZATION, W. H. *Statement on the second meeting of the International Health Regulations (2005) Emergency Committee regarding the outbreak of novel coronavirus (2019-nCoV).* 2020. Disponível em: <https://www.who.int/news/item/30-01-2020-statement-on-the-second-meeting-of-the-international-health-regulations-(2005)-emergency-committee-regarding-the-outbreak-of-novel-coronavirus-(2019-ncov)>.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 9351, p. 234–241, may 2015. ISSN 16113349. Disponível em: <https://arxiv.org/pdf/1505.04597.pdfhttp://link.springer.com/10.1007/978-3-319-24574-4_28http://arxiv.org/abs/1505.04597>.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958. ISSN 0033295X.

RUSSEL, S. J. *Arificial Intelligence: A modern approach*. Englewood Cliffs, New Jersey: Alan Apt, 1995. v. 74. 947 p. ISSN 17410509. ISBN 0131038052.

RUSTAM, F. et al. COVID-19 Future Forecasting Using Supervised Machine Learning Models. *IEEE Access*, v. 8, p. 101489–101499, 2020. ISSN 21693536.

SELVARAJU, R. R. et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*, v. 128, n. 2, p. 336–359, 2020. ISSN 15731405. Disponível em: <https://arxiv.org/pdf/1610.02391.pdf>.

SHI, F. et al. Review of Artificial Intelligence Techniques in Imaging Data Acquisition, Segmentation, and Diagnosis for COVID-19. *IEEE Reviews in Biomedical Engineering*, v. 14, n. c, p. 4–15, 2021. ISSN 19411189.

SIGNORONI, A. et al. Bs-net: Learning covid-19 pneumonia severity on a large chest x-ray dataset. *Medical Image Analysis*, v. 71, 2021. ISSN 13618423.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, p. 1–14, 2015.

STIRENKO, S. et al. Chest x-ray analysis of tuberculosis by deep learning with segmentation and augmentation. *arXiv*, 2018. ISSN 23318422.

SUDRE, C. H. et al. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 10553 LNCS, p. 240–248, 2017. ISSN 16113349.

SZEGEDY, C. et al. Inception-v4, inception-ResNet and the impact of residual connections on learning. *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, p. 4278–4284, 2017.

TABIK, S. et al. COVIDGR Dataset and COVID-SDNet Methodology for Predicting COVID-19 Based on Chest X-Ray Images. *IEEE Journal of Biomedical and Health Informatics*, v. 24, n. 12, p. 3595–3605, 2020. ISSN 21682208.

WANG, L.; LIN, Z. Q.; WONG, A. Covid-net: a tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. *Scientific Reports*, v. 10, n. 1, p. 19549, Nov 2020. ISSN 2045-2322. Disponível em: <https://doi.org/10.1038/s41598-020-76550-z>.

YIN, P. et al. Deep Guidance Network for Biomedical Image Segmentation. *IEEE Access*, v. 8, p. 116106–116116, 2020. ISSN 21693536.

ZHAO, W. et al. Relation between chest CT findings and clinical conditions of coronavirus disease (covid-19) pneumonia: A multicenter study. *American Journal of Roentgenology*, v. 214, n. 5, p. 1072–1077, 2020. ISSN 15463141.

ZHENG, C. et al. Deep Learning-based Detection for COVID-19 from Chest CT using Weak Label. *IEEE Transactions on Medical Imaging*, p. 1–13, 2020.

ZHOU, B. et al. Learning deep features for discriminative localization. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, v. 2016-December, p. 2921–2929, 2016. ISSN 10636919.

ZIMATORE, C. et al. Accuracy of the Radiographic Assessment of Lung Edema Score for the Diagnosis of ARDS. *Frontiers in Physiology*, v. 12, n. June 2021, 2021. ISSN 1664042X.

Anexos

Anexo I - Funções de Erro

```python
from src.models.losses.dice_loss import DiceError
from typing import Any
from tensorflow.python.keras.losses import Loss
from tensorflow.python.keras import backend as K
import tensorflow as tf
import tensorflow_addons as tfa

class LogCoshDiceError(DiceError):

    def __init__(
        self,
        regularization_factor: float = 1,
        name: str = 'log_cosh_f1'
    ) -> None:
        super().__init__(
            name=name,
            regularization_factor=regularization_factor
        )

    def call(
        self,
        y_true: tfa.types.TensorLike,
        y_pred: tfa.types.TensorLike
    ) -> tfa.types.TensorLike:
        dice_loss = super().call(y_true=y_true,y_pred=y_pred)
        total_loss = tf.math.log(tf.math.cosh(dice_loss))
        total_loss = total_loss * self.regularization_factor
        return total_loss

from typing import Any, Optional
from tensorflow.python.keras.losses import Loss
from tensorflow.python.keras.utils import losses_utils
from tensorflow.python.keras import backend as K
import tensorflow as tf
import tensorflow_addons as tfa
```

```
36
37 class DiceError(Loss):
38
39     def __init__(
40         self,
41         regularization_factor: Optional[float] = 1,
42         name: str = 'f1'
43     ) -> None:
44         if regularization_factor is not None:
45             self.regularization_factor = regularization_factor
46         else:
47             self.regularization_factor = losses_utils.ReductionV2.AUTO
48         super().__init__(name=name)
49
50     def call(
51         self,
52         y_true: tfa.types.TensorLike,
53         y_pred: tfa.types.TensorLike
54     ) -> tfa.types.TensorLike:
55
56         class_num = y_true.shape[-1] if y_true.shape[-1] is not None else 1
57         total_loss = 0
58
59         for class_now in range(class_num):
60
61             len_shape = len(y_true.shape)
62
63             if len_shape == 4 and class_num > 1:
64                 # Converte y_pred e y_true em vetores
65                 y_true = y_true[:,:,:,class_now]
66                 y_pred = y_pred[:,:,:,class_now]
67             elif len_shape == 3 and class_num > 1:
68                 # Converte y_pred e y_true em vetores
69                 y_true = y_true[:,:,class_now]
70                 y_pred = y_pred[:,:,class_now]
71             elif len_shape == 2 and class_num > 1:
72                 # Converte y_pred e y_true em vetores
73                 y_true = y_true[:,class_now]
74                 y_pred = y_pred[:,class_now]
```

```
75              y_true_f = K.flatten(y_true)
76              y_pred_f = K.flatten(y_pred)
77
78              loss = dice_loss(y_true_f, y_pred_f)
79
80              total_loss = total_loss + loss
81
82          total_loss = total_loss / class_num
83          total_loss = (1 - total_loss) * self.regularization_factor
84          return total_loss
85
86  def dice_loss(
87      y_true_f: tfa.types.TensorLike,
88      y_pred_f: tfa.types.TensorLike
89  ) -> tfa.types.TensorLike:
90      y_true_f = tf.cast(y_true_f, dtype=tf.float64)
91      y_pred_f = tf.cast(y_pred_f, dtype=tf.float64)
92      # Correção da equação de erro
93      correction_factor = tf.constant(1.0, dtype=tf.float64)
94      # Smooth - Evita que o denominador fique muito pequeno
95      smooth = K.constant(1e-6, dtype=tf.float64)
96      # Calculo o erro entre eles
97      constant = K.constant(2.0, dtype=tf.float64)
98
99      # Calcula o numero de vezes que
100     # y_true(positve) é igual y_pred(positive) (tp)
101     intersection = K.sum(y_true_f * y_pred_f)
102     # Soma o número de vezes que ambos foram positivos
103     union = K.sum(y_true_f) + K.sum(y_pred_f)
104     num = tf.math.multiply(constant,intersection) + correction_factor
105     den = union + smooth + correction_factor
106     loss = num / den
107     return loss
```

Anexo II - Métricas

```python
import tensorflow as tf
from tensorflow.python.keras.metrics import Metric
import tensorflow.python.keras.backend as K
from tensorflow.python.keras.utils import metrics_utils
from tensorflow.python.keras.utils.generic_utils import to_list
import numpy as np

class F1score(Metric):
    def __init__(
        self,
        name: str = "f1",
        dtype=None,
        thresholds: float = None,
        top_k=None,
        class_id=None,
        **kwargs
    ) -> None:
        super(F1score, self).__init__(name=name, dtype=dtype, **kwargs)
        self.init_thresholds = thresholds
        self.top_k = top_k
        self.class_id = class_id

        default_threshold = 0.5 if top_k is None else metrics_utils.NEG_INF
        self.thresholds = metrics_utils.parse_init_thresholds(
            thresholds=thresholds, default_threshold=default_threshold
        )
        self.true_positives = self.add_weight(
            "true_positives",
            shape=(len(self.thresholds),),
            initializer=tf.compat.v1.zeros_initializer,
        )
        self.false_positives = self.add_weight(
            "false_positives",
            shape=(len(self.thresholds),),
            initializer=tf.compat.v1.zeros_initializer,
        )
        self.false_negatives = self.add_weight(
```

```
38              "false_negatives",
39              shape=(len(self.thresholds),),
40              initializer=tf.compat.v1.zeros_initializer,
41          )
42          self.score = self.add_weight(name="f1", initializer="zeros")
43
44      def result(self):
45          precision = tf.math.divide_no_nan(
46              self.true_positives, self.true_positives + self.false_positives
47          )
48          precision = precision[0] if len(self.thresholds) == 1 else precision
49          recall = tf.math.divide_no_nan(
50              self.true_positives, self.true_positives + self.false_negatives
51          )
52          recall = recall[0] if len(self.thresholds) == 1 else recall
53          score = tf.math.multiply_no_nan(precision, recall)
54          score = tf.math.multiply_no_nan(tf.constant(2.0), score)
55          return tf.math.divide_no_nan(score, precision + recall)
56
57      def update_state(self, y_true, y_pred, sample_weight=None):
58          return metrics_utils.update_confusion_matrix_variables(
59              {
60                  metrics_utils.ConfusionMatrix.TRUE_POSITIVES: self.true_positives,
61                  metrics_utils.ConfusionMatrix.FALSE_POSITIVES: self.false_positives,
62                  metrics_utils.ConfusionMatrix.FALSE_NEGATIVES: self.false_negatives,
63              },
64              y_true=y_true,
65              y_pred=y_pred,
66              thresholds=self.thresholds,
67              top_k=self.top_k,
68              class_id=self.class_id,
69              sample_weight=sample_weight,
70          )
71
72      def reset_states(self):
73          num_thresholds = len(to_list(self.thresholds))
74          K.batch_set_value(
75              [
76                  (v, np.zeros((num_thresholds,)))
```

```
77                 for v in (
78                     self.true_positives,
79                     self.false_positives,
80                     self.false_negatives,
81                 )
82             ]
83         )
84 
85     def reset_state(self):
86         num_thresholds = len(to_list(self.thresholds))
87         K.batch_set_value(
88             [
89                 (v, np.zeros((num_thresholds,)))
90                 for v in (
91                     self.true_positives,
92                     self.false_positives,
93                     self.false_negatives,
94                 )
95             ]
96         )
97 
98     def get_config(self):
99         config = {
100             "thresholds": self.init_thresholds,
101             "top_k": self.top_k,
102             "class_id": self.class_id,
103         }
104         base_config = super(F1score, self).get_config()
105         return dict(list(base_config.items()) + list(config.items()))
```

Anexo III - Modelos

```
import tensorflow as tf

from gc import garbage
from typing import Any, List, Optional, Tuple

from tensorflow.python.keras import Model
from tensorflow.python.keras.callbacks import Callback, EarlyStopping, History,
↪  ModelCheckpoint, ReduceLROnPlateau, TerminateOnNaN
from tensorflow.python.keras.engine.base_layer import Layer
from tensorflow.python.keras.layers import Conv2D, UpSampling2D
from tensorflow.python.keras.layers import Dropout
from tensorflow.python.keras.layers import Activation
from tensorflow.python.keras.layers import Concatenate
from tensorflow.python.keras.layers import BatchNormalization
from tensorflow.python.keras.layers import MaxPooling2D
from tensorflow.python.keras.regularizers import l1_l2
from tensorflow.python.keras.regularizers import l2
from tensorflow.python.keras.optimizer_v2.adamax import Adamax
from tensorflow.python.keras.metrics import Metric
from tensorflow.python.keras.metrics import BinaryAccuracy
from tensorflow.python.keras.models import Input
from tensorflow.python.keras.utils.data_utils import Sequence

from src.models.losses.log_cosh_dice_loss import LogCoshDiceError
from src.models.metrics.f1_score import F1score
from src.models.losses.dice_loss import DiceError
from src.models.callbacks.clear_garbage import ClearGarbage

def inner_callbacks() -> List[Callback]:
    checkpoint = ModelCheckpoint(
        '.\\best.weights.hdf5', monitor='val_f1',
        verbose=1,save_best_only=True, mode='max',
        save_weights_only=True,
    )
    # Metrica para a redução do valor de LR
    reduce_lr = ReduceLROnPlateau(
        monitor='val_loss', factor=0.2, patience=3,
```

```
37          verbose=1, mode='min', min_delta=1e-2, cooldown=3,
38          min_lr=1e-8
39      )
40      # Metrica para a parada do treino
41      early = EarlyStopping(
42          monitor='val_loss', mode='min',
43          restore_best_weights=True, patience=10
44      )
45      terminate = TerminateOnNaN()
46      # Limpar o lixo do python
47      garbage = ClearGarbage()
48      # Vetor a ser passado na função fit
49      return [checkpoint, early, reduce_lr, terminate]
50
51
52  def inner_metrics() -> List[Metric]:
53      return [
54          BinaryAccuracy(name='bin_acc'),
55          F1score(name='f1')
56      ]
57
58  def unet_functional(
59      inputs: Tuple[int,int,int],
60      filter_root: int = 32,
61      depth: int = 5,
62      activation: str = 'relu',
63      final_activation: str = 'sigmoid',
64      n_class: int = 1,
65      rate: float = 0.3
66  ) -> Model:
67
68      store_layers = {}
69      first_layer = inputs
70      k = 0
71      input_layer = Input(inputs)
72      first_layer = input_layer
73      for i in range(depth):
74          filters = (2 ** i) * filter_root
75          # Cria as duas convoluções da camada
```

```
76          layer = unet_conv(
77              layer=first_layer,
78              activation=activation,
79              rate=rate,
80              k=k,
81              filters=filters
82          )
83          k += 1
84          layer = unet_conv(
85              layer=layer,
86              activation=activation,
87              rate=rate,
88              k=k,
89              filters=filters
90          )
91          k += 1
92          # Verifica se está na ultima camada
93          if i < depth - 1:
94              # Armazena a layer no dicionario
95              store_layers[str(i)] = layer
96              first_layer = MaxPooling2D(
97                  pool_size=(2,2),
98                  padding='same',
99                  name=f'max_{k}'
100             )(layer)
101         else:
102             first_layer = layer
103     for i in range(depth-2,-1,-1):
104         connection = store_layers[str(i)]
105         filters = (2 ** i) * filter_root
106         layer = UpSampling2D(name=f'up_{k}')(first_layer)
107         layer = Concatenate(axis=-1, name=f'cat_{k}')([layer, connection])
108         layer = unet_conv(
109             layer=layer,
110             activation=activation,
111             rate=rate,
112             k=k,
113             filters=filters
114         )
```

```
115         k += 1
116         layer = unet_conv(
117             layer=layer,
118             activation=activation,
119             rate=rate,
120             k=k,
121             filters=filters
122         )
123         k += 1
124         first_layer = layer
125     layer = Dropout(rate=rate,name='dropout')(layer)
126     outputs = Conv2D(
127         filters=n_class,
128         kernel_size=(1,1),
129         padding='same',
130         activation=final_activation,
131         name='output'
132     )(layer)
133     model = Model(inputs=input_layer, outputs=outputs)
134     return model
135 
136 def unet_conv(
137     layer: Layer,
138     activation: str = 'relu',
139     rate: float = 0.2,
140     filters: int = 32,
141     k: int = 0
142 ) -> Layer:
143     layer = Conv2D(
144         filters=filters,
145         kernel_size=(3,3),
146         padding='same',
147         name=f'conv_{k}'
148     )(layer)
149     layer = BatchNormalization(
150         name=f'bn_{k}'
151     )(layer)
152     layer = Activation(
153         activation,
```

```
154         name=f'act_{k}'
155     )(layer)
156     layer = Dropout(
157         rate=rate * 2,
158         name=f'drop_{k}'
159     )(layer)
160     return layer
161
162 def unet_fit(
163     model: Model,
164     x: Sequence,
165     validation_data: Sequence,
166     callbacks: Optional[List[Callback]] = None,
167     batch_size: Optional[int] = None,
168     epochs: int = 100,
169     shuffle: bool = True,
170     **params
171 ) -> History:
172     callbacks = inner_callbacks() if callbacks is None else callbacks
173     return model.fit(
174         x=x,
175         validation_data=validation_data,
176         batch_size=batch_size,
177         epochs=epochs,
178         callbacks=callbacks,
179         shuffle=shuffle,
180         **params
181     )
182
183 def unet_compile(
184     model: Model,
185     optimizer: str = 'adamax',
186     loss: str = 'log_cosh_dice',
187     metrics: Optional[List[Metric]] = None,
188     lr: float = 1e-5,
189     rf: float = 1.0,
190     **params
191 ) -> None:
192     metrics = inner_metrics() if metrics is None else metrics
```

```
193     optimizer = Adamax(learning_rate=lr) if optimizer == 'adamax' else optimizer
194     loss_function = DiceError(regularization_factor=rf) if loss == 'dice' else loss
195     loss_function = LogCoshDiceError(regularization_factor=rf) if loss ==
        ↪  'log_cosh_dice' else loss_function
196     model.compile(
197         optimizer=optimizer,
198         loss=loss_function,
199         metrics=metrics,
200         **params
201     )
202     return None
203
204 def save_weights(
205     model: Model,
206     filepath: str,
207     overwrite: bool = True,
208     **params
209 ) -> None:
210     model.save_weights(filepath, overwrite=overwrite, **params)
211     return None
212
213 from pathlib import Path
214 from typing import List, Optional, Union
215 from matplotlib import cm, pyplot as plt
216
217 import tensorflow as tf
218 from tensorflow.python.eager.monitoring import Metric
219 from tensorflow.python.keras.layers import Layer
220 from tensorflow.python.keras.layers.convolutional import Conv
221 from tensorflow.python.keras.layers import Activation
222 from tensorflow.python.keras.losses import Loss
223
224 from tensorflow_addons.utils.types import Optimizer
225
226 from src.images.process_images import split
227 from src.data.classification.cla_generator import ClassificationDatasetGenerator
228 from src.images.read_image import read_images
229 from src.models.grad_cam_split import prob_grad_cam
230 from src.output_result.folders import pandas2csv
```

```
231 from src.prints.prints import print_info
232
233 from tensorflow.python.keras import Model
234 from tensorflow.python.keras.layers import Conv2D, Activation
235 from tensorflow.python.keras import Input
236 from tensorflow.python.keras.layers.core import Dense, Dropout, Flatten
237 from tensorflow.python.keras.layers import BatchNormalization
238 from tensorflow.python.keras.callbacks import (
239     Callback,
240     History,
241     ModelCheckpoint,
242     ReduceLROnPlateau,
243     TerminateOnNaN,
244 )
245 from tensorflow.python.keras.optimizer_v2.adamax import Adamax
246 from tensorflow.python.keras.applications.vgg16 import VGG16
247 from tensorflow.python.keras.applications.vgg19 import VGG19
248 from tensorflow.python.keras.applications.inception_resnet_v2 import InceptionResNetV2
249 from tensorflow.python.keras.applications.resnet_v2 import ResNet50V2, ResNet101V2,
    ↪ ResNet152V2
250 from tensorflow.python.keras.applications.densenet import DenseNet121, DenseNet169,
    ↪ DenseNet201
251
252 import tensorflow_addons as tfa
253 import numpy as np
254 import cv2
255 from tqdm import tqdm
256
257 from tensorflow.python.keras.preprocessing.image import array_to_img
258 from tensorflow.python.keras.preprocessing.image import img_to_array
259
260 def get_callbacks() -> List[Callback]:
261     """
262     Retorna a lista callbacks do modelo
263
264     Returns:
265     --------
266         (List[Callback]): lista dos callbacks
267     """
```

```
268     # Salva os pesos dos modelo para serem carregados
269     # caso o monitor não diminua
270     check_params = {
271         "monitor": "val_loss",
272         "verbose": 1,
273         "mode": "min",
274         "save_best_only": True,
275         "save_weights_only": True,
276     }
277     checkpoint = ModelCheckpoint(".\\.model\\best.weights.hdf5", **check_params)
278 
279     # Reduz o valor de LR caso o monitor nao diminuia
280     reduce_params = {
281         "factor": 0.5,
282         "patience": 3,
283         "verbose": 1,
284         "mode": "max",
285         "min_delta": 1e-3,
286         "cooldown": 2,
287         "min_lr": 1e-8,
288     }
289     reduce_lr = ReduceLROnPlateau(monitor="val_loss", **reduce_params)
290 
291     # Termina se um peso for NaN (not a number)
292     terminate = TerminateOnNaN()
293     callbacks = [checkpoint, reduce_lr, terminate]
294     return callbacks
295 
296 def model_compile(
297     model: Model,
298     optimizer: Optional[Union[str, Optimizer]] = None,
299     loss: Union[str, Loss] = "categorical_crossentropy",
300     metrics: Optional[List[Metric]] = None,
301     lr: float = 1e-5,
302     **kwargs,
303 ) -> None:
304     """
305     Compile the model with loss and metrics define by the user.
306 
```

```
307     Args:
308         optimizer (optional | Optimizer): Optimizer of model. Defaults to None.
309         loss (str | Loss, optional): Loss of model. Defaults to
    ↪ "categorical_crossentropy".
310         metrics (List[Metric], optional): Metrics of systems. Defaults to None.
311         lr (float, optional): Learning rate of optimizer. Defaults to 1e-5.
312 
313     Returns:
314         None: compile the model with hiperparameters
315     """
316     optimizer = Adamax(learning_rate=lr) if optimizer is None else optimizer
317     metrics = ["accuracy"] if metrics is None else metrics
318     model.compile(optimizer=optimizer, loss=loss, metrics=metrics, **kwargs)
319 
320 def base(model_name: str = "ResNet50V2", split_dim: int = 224) -> Model:
321     """
322     Retorna a função intermediaria para a rede utilizada.
323 
324     Args:
325         model_name (str, optional): Nome do modelo intermediario. Defaults to
    ↪ "ResNet50V2".
326         split_dim (int, optional): tamaho da imagem de entrada. Defaults to 224.
327 
328     Returns:
329         Model: Modelo intermediario de aprendizado.
330     """
331     base = None
332     shape = (split_dim, split_dim, 3)
333     params = {
334         "include_top": False,
335         "weights": "imagenet",
336         "pooling": "avg",
337         "input_shape": shape,
338     }
339     if model_name == "VGG19":
340         base = VGG19(**params)
341     elif model_name == "VGG16":
342         base = VGG16(**params)
343     elif model_name == "InceptionResNetV2":
```

```
344         base = InceptionResNetV2(**params)
345     elif model_name == "ResNet50V2":
346         base = ResNet50V2(**params)
347     elif model_name == "ResNet101V2":
348         base = ResNet101V2(**params)
349     elif model_name == "ResNet152V2":
350         base = ResNet152V2(**params)
351     elif model_name == "DenseNet121":
352         base = DenseNet121(**params)
353     elif model_name == "DenseNet169":
354         base = DenseNet169(**params)
355     else:
356         base = DenseNet201(**params)
357     return base
358
359 def classification_model(
360     dim: int = 256,
361     channels: int = 1,
362     classes: int = 3,
363     final_activation: str = "softmax",
364     activation: str = "relu",
365     drop_rate: float = 0.2,
366     model_name: str = "ResNet50V2",
367 ) -> Model:
368     """
369     Função de criação do modelo de classificao da doença presente no pulmão.
370
371     >>> model = classification_model(256, 1, 3, 'softmax', 'relu', 0.2, 'ResNet50V2')
372
373     Args:
374         dim (int, optional): Dimensão da imagem de entrada `[0...]`. Defaults to
    ↪  `256`.
375         channels (int, optional): Numero de canais da imagem de entrada `[0...3]`.
    ↪  Defaults to `1`.
376         classes (int, optional): Numero de classes `[0...]`. Defaults to `3`.
377         final_activation (str, optional): Tipo de ativação da ultima camada Dense.
    ↪  Defaults to `'softmax'`.
378         activation (str, optional): Tipo de ativação da convolução. Defaults to
    ↪  `'relu'`.
```

```
379         drop_rate (float, optional): Taxa de Dropout. Defaults to `0.2`.
380 
381     Returns:
382         Model: Modelo de rede neural a ser treinada pelo sistema, contendo camdas
383         convolucionais e densas, cujo o meio pode alterar conforme o nomde de enetrada
384     """
385     input_shape = (dim, dim, channels)
386     inputs = Input(shape=input_shape)
387     layers = BatchNormalization()(inputs)
388     layers = Conv2D(filters=3, kernel_size=(3, 3), padding="same")(layers)
389     layers = Activation(activation=activation)(layers)
390     layers = Dropout(rate=drop_rate)(layers)
391     layers = base(model_name=model_name, split_dim=dim)(layers)
392     layers = Flatten()(layers)
393     layers = Dense(units=classes)(layers)
394     layers = Activation(activation=final_activation)(layers)
395     outputs = Dropout(rate=drop_rate)(layers)
396     model = Model(inputs=inputs, outputs=outputs)
397     return model
398 
399 
400 def confusion_matrix(
401     model: Model,
402     x: ClassificationDatasetGenerator,
403     n_splits: int = 1,
404     labels: List[str] = ["Covid", "Normal", "Pneumonia"],
405 ) -> tfa.types.TensorLike:
406     """
407     Metódo utilizado para avaliar o desempenho de uma rede de classificação.
408     A diagonal principal contem os valores preditos corretamente, enquantos os demais
409     valores são as predições incorretas realizadas pelo modelo.
410     >>> modelo.confusion_matrix(teste.x,n_splits=2)
411     Esse código gerará uma matriz de confução para as imagens teste.x usando
412     `2` recortes por imagens, explicitado em `n_splits`.
413     Args:
414         x (DataGenerator):
415             Gerador contendo os caminhos das imagens a serem preditas.
416         n_splits (int, optional):
417             Numero de recortes randomicos utilizados para gerar a predição.
```

```
418             Defaults to 1.
419     Returns:
420         (np.array):
421             [Matriz contendo os valores da matriz de confusão]
422     """
423     n_labels = len(labels)
424     matriz = np.zeros((n_labels, n_labels))
425     DIM_SPLIT = 224
426     for path in tqdm(x.x):
427         elect = make_grad_cam(
428             model=model,
429             image=path,
430             n_splits=n_splits,
431             verbose=False,
432             split_dim=DIM_SPLIT,
433         )
434         true_index = labels.index(path.parts[-2])
435         index = labels.index(elect)
436         matriz[index][true_index] += 1
437     return matriz
438
439 def last_conv_layer(model: Model) -> str:
440     """
441     Find last conv layer in the model.
442
443     Args:
444         model (Model): model to analyse.
445
446     Returns:
447         str: name of last convolution layer.
448     """
449     for layer in reversed(model.layers):
450         if isinstance(layer, Model):
451             return last_conv_layer(layer)
452         if isinstance(layer, Conv):
453             return layer.name
454     return ""
455
456
```

```python
def names_classification_layers(model: Model) -> List[str]:
    """Search in all model for find a Conv2D layer in reversed order,
    saving all the names in the track.

    Args:
        model (Model): model to analyse classification layers.

    Returns:
        List[str]: layers names of classification model.

    >>> cla_layers_names = names_classification_layer(model)
    """
    layers_names = []
    for layer in reversed(model.layers):
        if isinstance(layer, Model):
            for inner in layer.layers:
                if isinstance(inner, Conv2D):
                    return layers_names
                layers_names.insert(0, inner.name)
            layers_names.insert(0, layer.name)
        else:
            if isinstance(layer, Conv2D):
                return layers_names
            layers_names.insert(0, layer.name)
    return layers_names


def save_weights(
    model: Model,
    model_name: str,
    history: History = None,
    parent: Path = None,
    history_path: Path = None,
    overwrite: bool = True,
    metric: str = "val_f1",
    **params,
) -> None:
    """
    Save wights of model in `parent` folder with name of `model_name`,
```

```
496     if `history` is not `None` the name of save file will be add the
497     last `metric` value of model to file name. With `history_path`
498     is passed to the history is save in csv file.
499
500     Args:
501         model (Model): model of weights to save.
502         model_name (str): file name of weights.
503         history (History, optional): history of model train. Defaults to None.
504         parent (Path, optional): parent file where the files to save will be storeage.
    ↪ Defaults to None.
505         history_path (Path, optional): path to save `history` in csv. Defaults to
    ↪ None.
506         overwrite (bool, optional): `overwrite` file if already exists. Defaults to
    ↪ True.
507         metric (str, optional): metrics to analyse the weights. Defaults to "val_f1".
508     """
509     filename = model_name
510     if history is not None:
511         metric_value = history.history[metric][-1]
512         filename = f"{filename}_{metric}_{metric_value:0.2f}"
513         if history_path is not None:
514             pandas2csv(history, history_path)
515     filename = parent / filename if parent is not None else filename
516     filename = f"{filename}.hdf5"
517     print_info(f"Pesos salvos em: {filename}")
518     return model.save_weights(filename, overwrite=overwrite, **params)
519
520
521 def winner(
522     labels: List[str] = ["Covid", "Normal", "Pneumonia"],
523     votes: List[int] = [0, 0, 0]
524 ) -> str:
525     """
526     Return label of disease winner.
527     Args:
528     -----
529         labels (list): labels.
530         votes (list): predictions.
531     Returns:
```

```
532     --------
533         elect (str): winner label
534     """
535     poll = np.sum(votes, axis=0)
536     elect = labels[np.argmax(poll)]
537     return elect
538 
539 
540 def predict(
541     model: Model,
542     x: ClassificationDatasetGenerator,
543     **params
544 ) -> tfa.types.TensorLike:
545     """Predict images from generator.
546 
547     Args:
548         model (Model):
549         x (ClassificationDatasetGenerator): generator who contains image to analyse.
550 
551     Returns:
552         tfa.types.TensorLike: predictions.
553     """
554     return model.predict(x, **params)
555 
556 
557 def get_classifier_layer_names(model: Model, layer_name: str) -> List[str]:
558     """Return list string of classifier layer names
559 
560     Args:
561         model (Model): model to analyse
562 
563     Returns:
564         List[str]: list of classifier layers
565     """
566     classifier_layers_names = []
567     for layer in reversed(model.layers):
568         if isinstance(layer, Model):
569             for inner_layer in reversed(layer.layers):
570                 if inner_layer.name == layer_name:
```

```
571                     return classifier_layers_names
572                 classifier_layers_names.insert(0, inner_layer.name)
573         if layer.name == layer_name:
574             return classifier_layers_names
575         classifier_layers_names.insert(0, layer.name)
576     return classifier_layers_names
577 
578 
579 def get_classifier_layer(
580     model: Model,
581     classifier_names: List[str] = None,
582 ) -> List[str]:
583 
584     classifier_names = [] if classifier_names is None else classifier_names
585     last_activation = ''
586 
587     for layer in reversed(model.layers):
588 
589         if isinstance(layer, Model):
590             return get_classifier_layer(model, classifier_names)
591 
592         if isinstance(layer, Conv):
593             break
594 
595         if isinstance(layer, Activation):
596             last_activation = layer.name
597 
598         classifier_names.insert(0, layer.name)
599 
600     for classifier in classifier_names:
601         if classifier != last_activation:
602             classifier_names.remove(classifier)
603         else:
604             break
605 
606     return classifier_names
607 
608 def get_last_conv_layer_name(model: Model) -> Layer:
609     """Get a layer in model with submodels.
```

```
610
611     Args:
612         model (Model): model to search for layer
613
614     Returns:
615         Layer: layer wanted.
616     """
617     for layer in reversed(model.layers):
618         if isinstance(layer, Model):
619             for inner_layer in reversed(layer.layers):
620                 if isinstance(inner_layer, Conv):
621                     return inner_layer.name
622         if isinstance(layer, Conv):
623             return layer.name
624
625 def find_base(model: Model) -> Model:
626     """Get first submodel in a model.
627
628     Args:
629         model (Model): model to analyse.
630
631     Returns:
632         Model: first submodel in the model.
633     """
634     for layer in reversed(model.layers):
635         if isinstance(layer, Model):
636             return layer
637
638 def predicts_values(
639     model: Model,
640     x: Union[str, Path],
641     n_splits: int = 100,
642     threshold: float = 0.35,
643     verbose: bool = True,
644     split_dim: int = 224,
645     channels: int = 1,
646 ) -> str:
647     predicts = np.array([])
648     for image in tqdm(x.x):
```

```
649         params_splits = {
650             "verbose": verbose,
651             "dim": split_dim,
652             "channels": channels,
653             "threshold": threshold,
654             "n_splits": n_splits,
655         }
656         cuts, positions = split(image, **params_splits)
657         shape = (1, n_splits, split_dim, split_dim, channels)
658         if isinstance(image, list):
659             shape = (len(image), n_splits, split_dim, split_dim, channels)
660         cuts = cuts.reshape(shape)
661         predict_params = {"verbose": 0}
662         cuts = np.reshape(cuts, (n_splits, split_dim, split_dim, channels))
663         predict_array = predict(model, cuts, **predict_params)
664         zeros = np.zeros((3))
665         for b in predict_array:
666             zeros += b
667         zeros /= n_splits
668         predicts = np.append(zeros, predicts)
669     predicts = np.reshape(predicts, (len(x.x),3))
670     return predicts
671
672 def make_grad_cam(
673     model: Model,
674     image: Union[str, Path],
675     n_splits: int = 100,
676     threshold: float = 0.35,
677     verbose: bool = True,
678     split_dim: int = 224,
679     orig_dim: int = 1024,
680     channels: int = 1,
681     name: str = None,
682     labels: List[str] = ["Covid", "Normal", "Pneumonia"],
683 ) -> str:
684     params_splits = {
685         "verbose": verbose,
686         "dim": split_dim,
687         "channels": channels,
```

```
688         "threshold": threshold,
689         "n_splits": n_splits,
690     }
691     image_color = read_images(image, color=True)
692     images = read_images(image)
693     cuts, positions = split(images, **params_splits)
694     shape = (1, n_splits, split_dim, split_dim, channels)
695     if isinstance(image, list):
696         shape = (len(image), n_splits, split_dim, split_dim, channels)
697     cuts = cuts.reshape(shape)
698     if verbose:
699         # class_names = get_classifier_layer(model=model)
700 
701         if isinstance(image, list):
702             winner_label = labels.index(image[0].parts[-2])
703         else:
704             winner_label = labels.index(image.parts[-2])
705 
706         heatmap = prob_grad_cam(
707             cuts_images=cuts,
708             paths_start_positions=positions,
709             model=model,
710             dim_orig=orig_dim,
711             winner_pos=winner_label,
712         )
713         plot_gradcam(heatmap=heatmap, image=image_color, grad=True, name=name)
714     predict_params = {"verbose": 0}
715     cuts = np.reshape(cuts, (n_splits, split_dim, split_dim, channels))
716     votes = predict(model, cuts, **predict_params)
717     elect = winner(labels=labels, votes=votes)
718     return elect
719 
720 def superimposed_image_generate(
721     heatmap: tfa.types.TensorLike,
722     image: tfa.types.TensorLike,
723     dim: int = 1024,
724     alpha: float = 0.4,
725 ) -> tfa.types.TensorLike:
726     """Function to merge `heatmap` to `image` with original `dim` size with `alpha`
```

```
727     opacity, if `grad` is `True` then plot grad cam probabilistic, if name is not
728     `None` then save figure in the name file.
729
730     Args:
731         heatmap (tfa.types.TensorLike): heatmap to merge in image.
732         image (tfa.types.TensorLike): image of input
733         grad (bool, optional): flag to plot grad cam. Defaults to True.
734         name (str, optional): name of png output file. Defaults to None.
735         dim (int, optional): original dimension of image. Defaults to 1024.
736         alpha (float, optional): opacity value. Defaults to 0.4.
737
738     Returns:
739         str: path where save png
740     """
741     if np.max(heatmap) < 1.1:
742         heatmap = np.uint8(255 * heatmap)
743     if len(image.shape) < 3:
744         image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
745
746     jet = cm.get_cmap("jet")
747     jet_color = jet(np.arange(256))[:, :3]
748     jet_heatmap = jet_color[heatmap]
749
750     jet_heatmap0 = array_to_img(jet_heatmap)
751     jet_heatmap1 = jet_heatmap0.resize((dim, dim))
752     jet_heatmap2 = img_to_array(jet_heatmap1)
753     superimposed_image = jet_heatmap2 * alpha + image
754     superimposed_image = array_to_img(superimposed_image)
755     return superimposed_image
756
757
758 def plot_gradcam(
759     heatmap: tfa.types.TensorLike,
760     image: tfa.types.TensorLike,
761     grad: bool = False,
762     name: str = None,
763     dim: int = 1024,
764     alpha=0.4,
765 ) -> str:
```

```
766     """Function to merge `heatmap` to `image` with original `dim` size with `alpha`
767     opacity, if `grad` is `True` then plot grad cam probabilistic, if name is not
768     `None` then save figure in the name file.
769
770     Args:
771         heatmap (tfa.types.TensorLike): heatmap to merge in image.
772         image (tfa.types.TensorLike): image of input
773         grad (bool, optional): flag to plot grad cam. Defaults to True.
774         name (str, optional): name of png output file. Defaults to None.
775         dim (int, optional): original dimension of image. Defaults to 1024.
776         alpha (float, optional): opacity value. Defaults to 0.4.
777
778     Returns:
779         str: path where save png
780     """
781     superimposed_image = superimposed_image_generate(
782         heatmap=heatmap,
783         image=image,
784         dim=dim,
785         alpha=alpha
786     )
787     if grad:
788         superimposed_image = tf.cast(superimposed_image, np.uint8)
789         fig = plt.figure()
790         plt.imshow(superimposed_image, cmap="plasma")
791         # Salvar imagem
792         path = ""
793         if name is not None:
794             path = "{}.png".format(name)
795             plt.savefig(path, dpi=fig.dpi)
796         plt.show()
797     return superimposed_image
```

Anexo IV - Leitura das imagens

```
"""
    Biblioteca de funções de leitura de imagens pelos caminhos
"""

from pathlib import Path
from typing import List, Union, Any, Tuple, Optional
import cv2 as cv
import tensorflow_addons as tfa
import numpy as np
import tensorflow as tf

def read_random_image(
    paths: List[Path],
    id_start: List[int] = [0],
    **params
) -> tfa.types.TensorLike:
    """
        Lê as imagens dos ids contidos em id_start

        Args:
            paths (list): Caminhos completos das imagens a serem lidas.
            id_start (list, optional): ids das imagens a serem lidas.
                                       Defaults to [0,1].

        Returns:
            tfa.types.TensorLike: lista das imagens lidas.
    """
    images = []
    channel = 3 if params['color'] else 1
    shape = (len(id_start), params['dim'], params['dim'], channel)
    for i in id_start:
        image = read_images(images_paths=paths[i],**params)
        images.append(image)
    images = np.array(images)
    images = np.reshape(images, shape)
    return images

```

```
38
39 def read_sequencial_image(
40     paths: List[Path],
41     id_start: int = 0,
42     id_end: int = 1,
43     **params
44 ) -> tfa.types.TensorLike:
45     """
46         Lê sequencialmente as imagens
47
48         Args:
49             paths (list): Caminhos completos das imagens a serem lidas.
50             id_start (int, optional): ID inicial da imagem a ser lida.
51                                       Defaults to 0.
52             id_end (int, optional): ID final da imagem a ser lida.
53                                       Defaults to 0.
54             normalize (bool, optional): Normaliza a imagem.
55                                         Defaults to False.
56         Returns:
57             list: lista das imagens lidas
58     """
59     channels = 3 if params['color'] else 1
60     shape = (id_end - id_start, params['dim'], params['dim'], channels)
61     images = [read_images(paths[i],**params) for i in range(id_start, id_end)]
62     images = np.array(images)
63     images = np.reshape(images,shape)
64     return images
65
66 def read_images(
67     images_paths: Union[List[Path], Path],
68     id_start: Union[List[int], int] = 0,
69     id_end: int = -1,
70     color: bool = False,
71     dim: int = 1024,
72     tensor: bool = True
73 ):
74     """
75         Lê as imagens contidas na listas de caminhos
76
```

```
77          Args:
78              images_paths (str or list): Arrays contndo os caminhos das
79                                          imagens.
80              id_start (int or list,optional): ID do inicio das imagens.
81                                               Defaults to 0.
82              id_end (int, optional): ID do fim das imagens.
83                                      Defaults to -1.
84              output_dim (int, optional): Image output dimension.
85                                          Defaults to -1.
86
87          Returns:
88              (np.array or list): retorna uma lista np.array das imagens lidas
89      """
90      image = None
91      params = {'color': color, 'dim': dim}
92      if isinstance(images_paths, list):
93          if isinstance(id_start, int):
94              if id_end < id_start:
95                  id_end = len(images_paths)
96              return read_sequencial_image(images_paths, id_start, id_end, **params)
97          return read_random_image(images_paths, id_start, **params)
98      if color:
99          image = cv.imread(str(images_paths))
100     else:
101         image = cv.imread(str(images_paths), cv.IMREAD_GRAYSCALE)
102     if tensor:
103         return resize_image(image,dim,color)
104     return cv.resize(image, (dim,dim))
105
106 def resize_image(
107     image: tfa.types.TensorLike,
108     dim: int,
109     color:bool = False
110 ) -> tfa.types.TensorLike:
111     """ Resize image to (dim,dim) pass as parameter. The image need be 1 channel with
        ↪  shape:
112         - NHWC (num_images,height,width,channels)
113         - HW (height,width)
114     Args:
```

```
115         image (tfa.types.TensorLike): image with shape (dim,dim)
116         dim (int): dimension of output image
117         color (bool): flag to define if image have or not colors
118 
119     Returns:
120         tfa.types.TensorLike: image resized
121     """
122     if not color:
123         image = tf.expand_dims(image,axis=-1)
124     image = tf.image.resize(image,size=[dim,dim])
125     return image
126 
127 def adjust_gamma(
128     image: tfa.types.TensorLike,
129     gamma: float = 0.5
130 ) -> tfa.types.TensorLike:
131     # build a lookup table mapping the pixel values [0, 255]
132     # to their adjusted gamma values
133     image = tf.image.adjust_gamma(image,gamma=gamma)
134     return image
135 
136 def read_step(
137     images: tfa.types.TensorLike,
138     shape: Tuple[int,int,int,int],
139     gamma: Optional[float] = 0.5,
140     equalize: bool = True
141 ) -> tfa.types.TensorLike:
142     dim = shape[1]
143     color = shape[-1] != 1
144     image_out = np.array([])
145     for image in images:
146         image = read_images(image,color=color,dim=dim)
147         if gamma is not None:
148             image = adjust_gamma(image)
149         if equalize:
150             image = tfa.image.equalize(image)
151         image_out = np.append(image_out,image)
152     image_out = np.reshape(image_out,shape)
153     return image_out
```