

UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
MESTRADO PROFISSIONAL

RODRIGO BIASUZ

APLICAÇÃO DA TÉCNICA MATEMÁTICA DE REDES NEURAIS
CONVOLUCIONAIS NO DESENVOLVIMENTO DE UM MODELO
COMPUTACIONAL CAPAZ DE IDENTIFICAR NÓDULOS PULMONARES

CAXIAS DO SUL
2021

RODRIGO BIASUZ

**APLICAÇÃO DA TÉCNICA MATEMÁTICA DE REDES NEURAIIS
CONVOLUCIONAIS NO DESENVOLVIMENTO DE UM MODELO
COMPUTACIONAL CAPAZ DE IDENTIFICAR NÓDULOS PULMONARES**

Dissertação apresentada como requisito obrigatório para
obtenção do título de Mestre em Engenharia Mecânica
pela Universidade de Caxias do Sul.

Área de Concentração: Inteligência Artificial

Orientador: Prof. Dr. Leandro Luís Corso

**CAXIAS DO SUL
2021**

Dados Internacionais de Catalogação na Publicação (CIP)
Universidade de Caxias do Sul
Sistema de Bibliotecas UCS - Processamento Técnico

B579a Biasuz, Rodrigo

Aplicação da técnica matemática de redes neurais convolucionais no desenvolvimento de um modelo computacional capaz de identificar nódulos pulmonares [recurso eletrônico] / Rodrigo Biasuz. – 2021.

Dados eletrônicos.

Dissertação (Mestrado) - Universidade de Caxias do Sul, Programa de Pós-Graduação em Engenharia Mecânica, 2021.

Orientação: Leandro Luís Corso.

Modo de acesso: World Wide Web

Disponível em: <https://repositorio.ucs.br>

1. Tomografia computadorizada por raios x. 2. Redes neurais (Computação). 3. Pulmões - Câncer. 4. Modelos matemáticos. 5. Inteligência artificial - Aplicações médicas. I. Corso, Leandro Luís, orient. II. Título.

CDU 2. ed.: 621.386:616.24-006

Catalogação na fonte elaborada pela(o) bibliotecária(o)
Márcia Servi Gonçalves - CRB 10/1500

RODRIGO BIASUZ

**APLICAÇÃO DA TÉCNICA MATEMÁTICA DE REDES NEURAIIS
CONVOLUCIONAIS NO DESENVOLVIMENTO DE UM MODELO
COMPUTACIONAL CAPAZ DE IDENTIFICAR NÓDULOS PULMONARES**

Dissertação apresentada como requisito obrigatório para
obtenção do título de Mestre em Engenharia Mecânica
pela Universidade de Caxias do Sul.

Aprovado em: __/__/____

Banca Examinadora

Prof. Dr. Carlos Alberto Costa
Universidade de Caxias do Sul

Prof. Dr. Guilherme Holsbach Costa
Universidade de Caxias do Sul

Prof. Dr. Luciano Selistre
Universidade de Caxias do Sul

Dedico este trabalho a todos os profissionais da saúde, que nesse momento de pandemia trabalham incansavelmente.

AGRADECIMENTOS

Agradeço esse trabalho a todas pessoas que eu tive o prazer de cruzar ao longo desta vida... Tendo em vista que, esse resultado, só foi possível graças a série de “acasos” que se somaram ao longo desses meus 25 anos e culminaram na pessoa que sou hoje.

Se apenas “o bater de asas de uma borboleta no Brasil pode ocasionar um tornado no Texas”, como se questionou Edward Lorenz¹ em 1972, uma metáfora aos sistemas caóticos, onde uma pequena alteração no presente (ou passado, olá *De Volta para o Futuro*), pode causar influências gigantescas no futuro, praticamente impossíveis de prever, e, também (excluindo a opção de volta no tempo, infelizmente) impossível de retornar ao estado inicial, o que dizer de um impacto de uma única pessoa nas nossas vidas?

Só tenho a agradecer a todos e me alegrar por termos nos encontrado.

Afinal, se você parar pra pensar no tamanho do universo e de todo o tempo que já existiu e todo o tempo que ainda existirá, qual a probabilidade de compartilharmos todas essas dimensões simultaneamente?

“Nós temos muita coisa em comum, a mesma terra, o mesmo ar, o mesmo céu. Talvez, se olhássemos o que temos em comum em vez de sempre procurar o que temos de diferente... Bem, quem sabe?” - Meowth - Pokémon, O Filme, 1998

㇏(●_●)㇏

¹ "**Predictability**: Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas?", LORENZ E. (1972) ISBN: 978 87 7066 494 3

“DON'T PANIC.”

**Guia do Mochileiro das Galáxias
Douglas Adams**

RESUMO

O câncer de pulmão é um dos mais frequentes tumores malignos, tendo, em 2018, 18,1 milhões de novos casos em todo o mundo. A detecção nos estágios iniciais é considerada o principal fator que determina a sobrevida dos pacientes. A tomografia computadorizada é o exame realizado de rotina para a detecção e estadiamento do câncer. As imagens geradas nesse processo necessitam de uma avaliação dos médicos, uma análise sujeita a falhas inerentes ao aspecto humano. O presente trabalho propõe um sistema de redes neurais convolucionais para a detecção de nódulos pulmonares para aprimorar o diagnóstico radiológico dos tumores pulmonares. Inicialmente tem-se a coleta e pré-processamento do banco de dados LIDC-IDRI, uma coletânea com 125 GB de imagens no formato DICOM, a separação em dados de treino e teste, a criação da IA, a otimização dos hiperparâmetros, a avaliação das métricas de classificação para a rede e, por fim, o desenvolvimento de uma aplicação WEB para uso prático do modelo matemático construído. O sistema criado apresentou uma acuracidade de 89,77%, um índice de *recall* de 96,37%, uma precisão de 82,66%, *f1-score* de 88,99% e área sob a curva ROC de 94,00%. O tempo de treinamento foi de aproximadamente 11 dias e o tempo médio para predição de uma nova imagem é de 24 ms.

Palavras-chave: Redes Convolucionais, CNN, Redes Neurais Artificiais, Aprendizagem Profunda, Tomografia Computadorizada, Nódulo Pulmonar, Inteligência Artificial, Modelo Matemático, Otimização de hiperparâmetros, Desenvolvimento WEB

ABSTRACT

Lung cancer is one of the most frequent malignant tumors, having, in 2018, 18.1 million new cases worldwide. Detection in the early stages is considered the main factor that determines patient survival. Computed tomography is the routine examination performed for the detection and staging of cancer. The images generated in this process require an analysis by doctors, a process subject to flaws inherent to the human aspect. The present work proposes a system of convolutional neural networks for the detection of pulmonary nodules to improve the radiological diagnosis of lung tumors. Initially we have the collection and pre-processing of the LIDC-IDRI database, a collection with 125 GB of images in DICOM format, the separation in training and test data, the creation of the AI model, the optimization of the hyperparameters, the evaluation of the classification metrics for the network and, finally, the development of a WEB application for practical use of the built mathematical model. The machine learning system created showed an accuracy of 89.77%, a recall rate of 96.37%, a precision of 82.66%, an F1-score of 88.99% and an area under the ROC curve of 94.00%. The training time was approximately 11 days and the average time for predicting a new image is 24 ms.

Key words: Convolutional Networks, CNN, Artificial Neural Networks, Deep Learning, Computed Tomography, Lung Node, Artificial Intelligence, Mathematical Model, Hyperparameter Optimization, WEB Development

LISTA DE FIGURAS

| | |
|--|----|
| Figura 1 - Exemplificação da construção da imagem a partir da CT | 20 |
| Figura 2 - Representação de um neurônio artificial | 21 |
| Figura 3 - Funções de ativação comumente usadas..... | 22 |
| Figura 4 - Arquitetura de uma rede 9x9x9, 3 camadas com 9 neurônios | 23 |
| Figura 5 - Representação de um Otimizador SGD | 26 |
| Figura 6 - Rede neural convolucional..... | 27 |
| Figura 7 - Representação de uma Etapa de Convolução em uma imagem..... | 28 |
| Figura 8 - Aplicação de um Max Pooling 2x2 em uma imagem..... | 29 |
| Figura 9 - Aplicação de Dropout em uma estrutura de redes | 29 |
| Figura 10 - Matriz de Confusão..... | 34 |
| Figura 11 - Matriz de Confusão..... | 34 |
| Figura 12 - Cross Validation | 36 |
| Figura 13 - Cruzamento de ponto de corte único | 39 |
| Figura 14 - Cruzamento de ponto de corte múltiplo..... | 39 |
| Figura 15 -Função “HelloWorld” em Python e em Java. | 43 |
| Figura 16 - Crescimento de Python dentro do StackOverflow..... | 44 |
| Figura 17 - Metodologia do trabalho proposto..... | 46 |
| Figura 18 - Etapa de validação e otimização | 51 |
| Figura 19 - Estrutura do Sistema Desenvolvido..... | 53 |
| Figura 20 - Imagem recém carregada no formato JPEG | 56 |
| Figura 21 - Pseudo-algoritmo que descreve a etapa de pré-processamento | 57 |
| Figura 22 - Separação dos dados em treino e teste..... | 58 |
| Figura 23 - Modelo de CNN inicial construído..... | 59 |
| Figura 24 - Evolução do modelo durante o treinamento | 60 |
| Figura 25 - Matriz de Confusão do Modelo Inicial | 61 |
| Figura 26 - Evolução do modelo B durante o treinamento..... | 64 |
| Figura 27 - Evolução do modelo C durante o treinamento..... | 65 |
| Figura 28 - Evolução do modelo D durante o treinamento | 65 |
| Figura 29 - Matriz de confusão do modelo B na etapa de teste..... | 66 |
| Figura 30 - Matriz de confusão do modelo C na etapa de teste..... | 67 |
| Figura 31 - Matriz de confusão do modelo D na etapa de teste | 68 |
| Figura 32 - Curva ROC do modelo escolhido | 70 |

| | |
|--|----|
| Figura 33 - Exemplos de dados que o modelo previu corretamente | 72 |
| Figura 34 - Exemplos de dados que o modelo previu erroneamente..... | 73 |
| Figura 35 - Estrutura de diretórios da aplicação..... | 74 |
| Figura 36 - Página de entrada | 75 |
| Figura 37 - Predição sem nódulo..... | 76 |
| Figura 38 - Predição com nódulo | 77 |

LISTA DE QUADROS

| | |
|---|----|
| Quadro 1 - Resumo das anotações do banco de dados LIDC-IDRI | 55 |
| Quadro 2 - Espaço de Busca..... | 62 |
| Quadro 3 - Hiperparâmetros de cada modelo | 63 |
| Quadro 4 - Resumo dos resultados | 69 |

LISTA DE ABREVIATURAS E SIGLAS

| | |
|-----------|---|
| AG | Algoritmo Genético |
| API | <i>Application Programming Interface</i> |
| CAD | <i>Computer-aided diagnostic</i> |
| CNN | <i>Convolutional Neural Networks</i> |
| FDA | <i>Food and Drug Administration</i> |
| FN | Falsos negativos |
| FNIH | <i>Foundation for the National Institutes of Health</i> |
| FP | Falsos positivos |
| HTML | <i>Hypertext Markup Language</i> |
| IA | Inteligência Artificial |
| LIDC-IDRI | <i>Lung Image Database Consortium image collection</i> |
| MAE | <i>Mean Absolute Error</i> |
| ML | <i>Machine Learning</i> |
| MLP | <i>Multilayer perceptron</i> |
| MSE | <i>Mean Squared Error</i> |
| NCI | <i>National Cancer Institute</i> |
| NPS | Nódulos Pulmonares Solitários |
| REST | <i>Representational State Transfer</i> |
| RMSE | <i>Root Mean Squared Error</i> |
| RNA | Rede Neural Artificial |
| ROC | <i>Receiver Operating Characteristic</i> |
| SGD | <i>Stochastic Gradient Descent</i> |
| TC | Tomografia Computadorizada |
| TN | Verdadeiros negativos |
| TP | Verdadeiros positivos |

SUMÁRIO

| | |
|---|-----------|
| 1 INTRODUÇÃO | 16 |
| 1.1 CONTEXTO DO TRABALHO PROPOSTO | 16 |
| 1.2 JUSTIFICATIVA | 17 |
| 1.3 OBJETIVOS | 18 |
| 1.3.1 Objetivo Geral | 18 |
| 1.3.2 Objetivos Específicos | 18 |
| 2 REFERENCIAL TEÓRICO | 19 |
| 2.1 NÓDULOS PULMONARES | 19 |
| 2.2 TOMOGRAFIA COMPUTADORIZADA | 19 |
| 2.3 REDES NEURAIS ARTIFICIAIS | 20 |
| 2.3.1 <i>Backpropagation</i> - Fazendo o modelo aprender | 24 |
| 2.4 CONVOLUTIONAL NEURAL NETWORK..... | 26 |
| 2.4.1 Convoluções | 27 |
| 2.4.2 <i>Pooling</i> | 28 |
| 2.4.3 <i>Dropout</i> | 29 |
| 2.4.4 Aplicações de IA e CNNs área da saúde | 30 |
| 2.5 AVALIAÇÃO DE MODELOS DE CNNS | 32 |
| 2.5.1 <i>Cross-Validation</i> | 35 |
| 2.6 OTIMIZAÇÃO DE HIPERPARÂMETROS | 36 |
| 2.7 DESENVOLVIMENTO WEB | 40 |
| 2.7.1 Python | 42 |
| 3 MÉTODO PROPOSTO | 46 |
| 3.1 CARACTERIZAÇÃO DA PESQUISA | 46 |
| 3.2 MÉTODO PROPOSTO PARA O TRABALHO..... | 46 |

| | |
|--|-----------|
| 3.2.1 Etapa 0 - Coletar os dados..... | 47 |
| 3.2.2 Etapa 01 – Pré-Processar os dados coletados | 47 |
| 3.2.3 Etapa 02 – Separar os dados em Teste e Treino..... | 48 |
| 3.2.4 Etapa 03 – Criar o modelo de CNN..... | 49 |
| 3.2.5 Etapa 04 – Validar o modelo | 50 |
| 3.2.6 Etapa 05 - Ajuste de hiperparâmetros | 51 |
| 3.2.7 Etapa 06 – Analisar os resultados obtidos | 52 |
| 3.3 CONSTRUÇÃO DE UMA PLATAFORMA COM OS RESULTADOS OBTIDOS ... | 52 |
| 4 APLICAÇÃO DO MÉTODO..... | 54 |
| 4.1 ENTENDENDO OS DADOS COLETADOS..... | 54 |
| 4.2 PRÉ-PROCESSAMENTO DOS DADOS COLETADOS..... | 55 |
| 4.3 SEPARAÇÃO EM TREINO E TESTE..... | 57 |
| 4.4 CRIAÇÃO DA CNN | 58 |
| 4.4 TESTE DO MODELO..... | 60 |
| 4.5 AJUSTE DE HIPERPARÂMETROS | 62 |
| 4.7 TESTE DOS MODELOS COM OS AJUSTES DE HIPERPARÂMETROS | 66 |
| 4.8 ANÁLISE DOS RESULTADOS OBTIDOS | 68 |
| 4.9 CONSTRUÇÃO DA APLICAÇÃO WEB | 74 |
| 5 CONCLUSÃO..... | 79 |
| 5.1 CONSIDERAÇÕES FINAIS | 79 |
| 5.2 SUGESTÕES DE TRABALHOS FUTUROS | 81 |
| 5.2.1 Utilização de AG para otimização de hiperparâmetros | 82 |
| 5.2.2 Agrupamento de diferentes modelos matemáticos..... | 82 |
| 5.2.3 Desenvolvimento de um App Mobile..... | 82 |
| 5.2.4 Sistema de rankeamento de TCs..... | 83 |
| 5.2.5 Aplicação de Transfer Learning para outros hospitais. | 83 |

| | |
|---|------------|
| REFERÊNCIAS | 85 |
| APÊNDICE A - ALGORITMOS DO SISTEMA CRIADO | 93 |
| APÊNDICE B - EXEMPLOS DE IMAGENS E SEUS RESULTADOS | 99 |
| APÊNDICE C - ALGORITMOS DA MODELAGEM..... | 106 |

1 INTRODUÇÃO

1.1 CONTEXTO DO TRABALHO PROPOSTO

Com o aumento no poder de processamento dos computadores e do volume de informações disponíveis no mundo, a ciência de dados tem crescido exponencialmente. Essa ciência tem aplicações em diversas áreas, como as ciências sociais (BRADY, 2019), as engenharias (KAPLIŃSKI; KOŁELEVA; ROPAITÈ, 2016), a biologia (GHARAJEH, 2018), a financeira (LÓPEZ, 2019), entre outras.

A área da saúde tem sido uma das principais impactadas por esses avanços, e conta com medidas vigorosas para converter o potencial dessa tecnologia em melhores serviços e com redução dos custos (DASH et al., 2019). Com uma forte integração de dados biomédicos, com resultados de exames de pacientes, imagens de Raio-X, Tomografia Computadorizada (TC), entre diversos outros, as organizações modernas de área podem revolucionar as terapias médicas e a medicina personalizada, por meio de diagnósticos cada vez mais assertivos e rápidos (KLANG., 2018).

Dos inúmeros desafios que a área da saúde possui, o manejo do câncer, nome dado a centenas de patologias que, em comum, possuem o crescimento desordenado de células anormais que se dividem incontrolavelmente, invadem e destroem tecidos e órgãos do corpo humano (INCA, 2019), tem ganhado destaque. Em 2016, o então presidente dos Estados Unidos, Barack Obama, lançou o programa *Cancer Moonshot*, para acelerar as pesquisas científicas sobre o assunto, aumentando o volume de dados disponíveis e, também, atraindo a atenção dos pesquisadores de ciência de dados para construir soluções na área (NARAYANAN, 2016).

O câncer é um dos maiores desafios do século, essa doença isoladamente é responsável por um quarto de todos os óbitos registrados e, estima-se que, até 2030, será a maior causa de mortes em todo o planeta. Apenas em 2018, 18,1 milhões de casos e 9,6 milhões de falecimentos ocorreram pela enfermidade pelo mundo (IARC, 2019).

Os sistemas de saúde, privado e público, têm gastos elevados em decorrência do câncer. O SUS, por exemplo, gastou aproximadamente 250 milhões de reais no ano de 2019 com o diagnóstico da doença (IARC, 2019). Ao mesmo tempo que os problemas de saúde relacionados aos diferentes tipos de câncer crescem ano após ano, os desenvolvimentos e investimentos em soluções que utilizam técnicas de Inteligência Artificial (IA) crescem

proporcionalmente no mundo todo, atingindo a marca de 1,7 bilhões de reais em 2017, com projeções de até 8,8 bilhões até 2022 (BBC, 2018).

Nesse cenário, técnicas que utilizam algoritmos de IA e auxiliam na detecção da doença em seus estágios iniciais por meio de imagens, de forma rápida, sem o tempo consumido para interpretação de um radiologista, agilizando o processamento e, consequentemente, facilitando a detecção do problema, são de grande valia. Os modelos desenvolvidos de IA atuam não apenas reduzindo custos operacionais para hospitais e governos, mas também aumentando a chance de sobrevivência para os pacientes que sofrem com a comorbidade.

Um dos modelos de IA que têm se destacado para aplicações no processamento de imagens são as Redes Neurais Convolucionais (*CNN*, do inglês *Convolutional Neural Network*), sendo amplamente utilizado em imagens radiológicas (YASAKA e ABE, 2018). A *CNN* é considerada um modelo matemático que busca reconhecer e encontrar pequenas variações em padrões de imagens. No caso da pesquisa deste trabalho, variações em TCs, por meio de uma rede que simula o comportamento do cérebro humano. Desse modo, este trabalho propõe a implementação da técnica matemática de *CNNs* no desenvolvimento de um modelo computacional capaz de identificar nódulos pulmonares em imagens de TC.

1.2 JUSTIFICATIVA

Dos mais de 100 tipos de câncer existentes, o câncer de pulmão é o mais comum a nível mundial (11,6% dos casos) e um dos com maior mortalidade, sendo responsável por 18,4% dos óbitos pela doença (IARC, 2019). Estudos demonstram que a TC é o exame mais indicado para a detecção precoce de nódulos pulmonares, fornecendo imagens de alta qualidade de detalhamento e informações precisas para especialistas responsáveis pela análise (SLUIMER et al., 2006).

Todavia, imagens de TC exigem mão de obra especializada para sua avaliação, tornando a detecção precoce de nódulos uma tarefa desafiadora e custosa (KAMIYA et al., 2011). Outro agravante são as altas taxas de falsos positivos e falsos negativos, que aumentam os desafios da avaliação adequada das TCs (ARDILA et al, 2019).

Ao mesmo tempo, a área de *Machine Learning* (ML), ou, em português, como vem sendo chamada, “Aprendizado de Máquina”, cresce a cada ano, com grandes empresas, como Google, Facebook, Amazon que tem investido em pesquisas e estruturas cada vez mais otimizadas e robustas para tarefas de ML, publicando seus resultados e incentivando novas pesquisas na área.

São exemplos disso o *BERT*, sistema dedicado ao processamento de linguagem natural da Google, (DEVLIN et al., 2018), o modelo de segmentação genética do Facebook (ZHOU et al., 2020) e o *Kaggle* (<https://www.kaggle.com>), uma plataforma colaborativa que oferece prêmios para soluções de ML. É comum as equipes que participam dessas competições interagirem entre si, agregando melhorias em suas estruturas e modelos utilizados, como se pode observar no trabalho de Sutton et al. (2018). Este trabalho apresentou como foi solucionado um problema de Ciência dos Materiais usando IA e *Crowdsourcing*.

Com isso, verifica-se que existe oportunidade de otimizar o processo de análise das tomografias com a assistência de modelos de decisão embasados em algoritmos de IA. Assim, pode-se ganhar velocidade de processamento, reduzir os custos associados ao processo e aumentar a confiabilidade no diagnóstico (ARDILA et al., 2019). Nesse sentido, a pergunta de motivação da pesquisa é: o quanto a modelagem matemática pode auxiliar na detecção de nódulos pulmonares em imagens de TC?

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Implementar a técnica matemática de redes neurais convolucionais, no desenvolvimento de um modelo computacional capaz de identificar nódulos pulmonares em imagens de tomografia computadorizada.

1.3.2 Objetivos Específicos

- a) Compreender a transformação das TCs em resultados de exame.
- b) Construir um modelo de CNN e treiná-lo para a tarefa de detecção de nódulos.
- c) Aplicar o modelo para prever os resultados de um conjunto de imagens.
- d) Analisar os resultados por meio de métricas de algoritmos de classificação.
- e) Desenvolver uma plataforma WEB para avaliar TCs.

2 REFERENCIAL TEÓRICO

2.1 NÓDULOS PULMONARES

Nódulos pulmonares, ou Nódulos Pulmonares Solitários (NPS), são, conforme Silva, Blagio e Gazzana (2009), uma opacidade pulmonar com menos de três centímetros de diâmetro e, sua identificação é importante, já que há probabilidade de 35% dos casos serem cânceres.

Os nódulos geralmente são identificados em pacientes que realizam exames de Raio-X ou Tomografia Computadorizada do Tórax (OST et al. 2003), porém sua detecção é atrapalhada por falsos positivos que podem ocorrer por diversas ocasiões, como citado por Silva, Blagio e Gazzana (2009):

Pseudo-nódulos causados por eletrodos ou pelos mamilos podem simular lesões intrapulmonares (...) Lesões cutâneas como verrugas, hemangiomas, neurofibromas e lipomas, que formam uma protuberância na pele do paciente e que são circundadas por ar atmosférico, podem parecer intrapulmonares em uma única projeção radiográfica. Outras lesões que podem mimetizar um NPS incluem lesões ósseas escleróticas, fraturas de costelas e osteófitos. Da mesma forma, lesões mediastinais e pleurais que são pedunculadas e que se projetam dentro do pulmão (p. ex: placas pleurais) podem parecer NPS quando vistas de frente. Muitas opacidades nodulares focais vistas radiograficamente são na verdade estruturas vasculares tortuosas. Ocasionalmente, uma mucocoele dentro de um brônquio dilatado pode simular um NPS.

Segundo Clavero (2015), os nódulos encontrados devem então ser avaliados por equipes multidisciplinares com experiência em diferentes procedimentos, de forma a detectar o início de um possível tumor a tempo de tratar o mesmo, o que eleva o custo e estende o tempo necessário para avaliação.

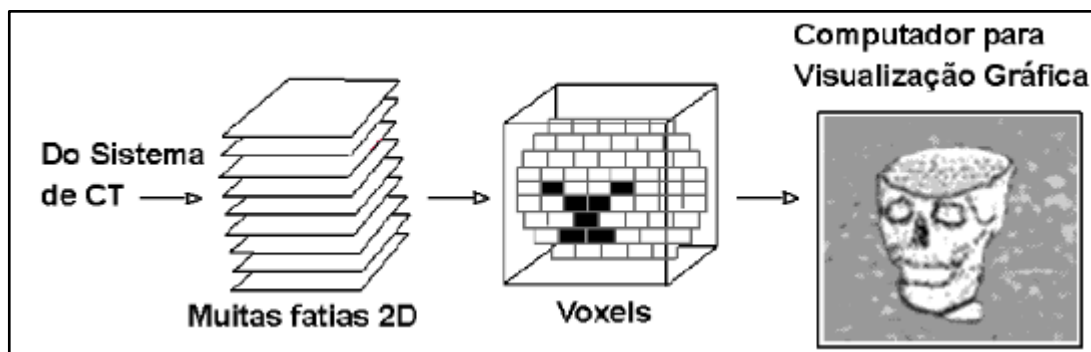
É com esse cenário que, nos últimos anos, a aplicação de modelos matemáticos e de ML para detecção de nódulos aumentou significativamente. Um dos trabalhos mais relevantes da área, conduzido por Ardila et al. em 2019, alcançou o até então estado da arte em performance, com 0,94 de área sobre a curva ROC, usando um modelo de *Deep Learning* nas imagens geradas a partir das TCs.

2.2 TOMOGRAFIA COMPUTADORIZADA

A TC revolucionou o diagnóstico de diversas doenças como um método não invasivo, rápido e fidedigno de representar visualmente as patologias, (ARELLANO, 2001). O exame consiste em um feixe de raios-x que gira ao redor do paciente, capturando camadas transversais

do mesmo, formando pequenos volumes chamados de voxels, constituídos de um conjunto de imagens bidimensionais, conforme a Figura 1.

Figura 1 - Exemplificação da construção da imagem a partir da CT



Fonte: Adaptado pelo autor de Ferraz e Conci (2003)

Cada pixel da imagem representa, em uma escala de cinza, a atenuação linear dos fótons transmitidos pelos feixes pelos tecidos encontrados pelo caminho. Considerando dessa forma o número de fótons enviados por uma amostra de tecido e o número de fótons que atravessou esse tecido e continuou sua trajetória (FERRAZ e CONCI, 2003).

Os especialistas podem utilizar tanto as imagens bidimensionais geradas quanto uma reconstrução tridimensional construída a partir dos voxels utilizando computação gráfica para avaliar se existe alguma patologia no paciente. Neste trabalho, utilizou-se apenas das imagens bidimensionais (as “fatias”) em escala de cinza geradas pelas TCs como entrada para o modelo de CNN pois os voxels 3D são apenas interpolações (ou reconstruções) das imagens 2D, os seja, não contêm informações adicionais que possam ajudar o modelo matemático em diferenciar as amostras.

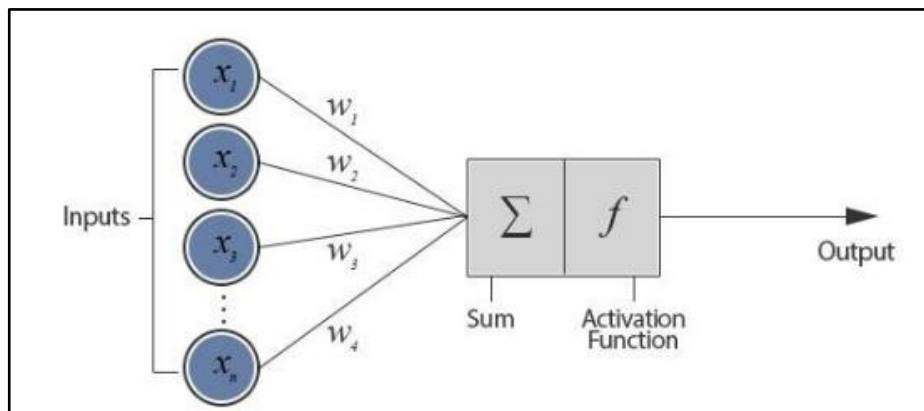
2.3 REDES NEURAIS ARTIFICIAIS

Uma Rede Neural Artificial (RNA), que aqui é trazida inicialmente por ser a base das CNNs utilizadas no projeto, consiste em conjunto de conexões entre unidades de processamento chamadas de neurônios, já que sua idealização inicial parte da ideia de “reproduzir” o cérebro humano virtualmente. As RNAs apresentam como habilidade fundamental a capacidade de aprender com a repetição, extrair informações de generalizações em um conjunto de valores nunca antes visto e tomar decisões com base nisso, como a nossa habilidade de interpretar as

palavras que outra pessoa escreveu em um papel, mesmo sem nunca antes ter visto aquela caligrafia (KUKREJA et al., 2016).

Estes neurônios recebem uma série de entradas com diferentes pesos, chamadas de *features*, que correspondem a atributos iniciais de algum sistema, como pixels de uma imagem ou as cotações diárias de uma ação, os valores numéricos de uma onda sonora ou mesmo um conjunto de palavras de um artigo, e processam esses dados enviando um novo sinal para uma camada posterior (MACKAY, 2005), como representado pela Figura 2.

Figura 2 - Representação de um neurônio artificial



Fonte: Adaptado pelo autor de Pokharna (2016)

Análogo a isso, pode-se comparar as entradas com os cinco sentidos humanos como a visão, que recebe sinais externos e envia esses sinais para os neurônios que, dependendo da intensidade do sinal, vão ser ativados ou não para enviar um novo sinal para uma camada de neurônios posterior e assim sucessivamente, fazendo o processamento desses “inputs”.

Cada entrada x_n (onde n é um número natural 1, 2, 3...) é multiplicada por um peso w_n (iniciado com um valor aleatório, a ser corrigido posteriormente, na etapa de treino), então é realizado um somatório desses produtos, passando esse resultado para uma função de ativação.

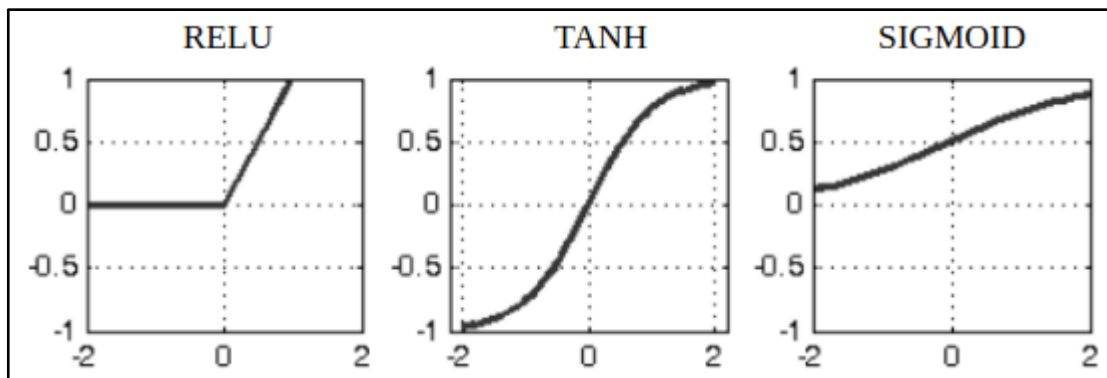
Essa função pode ser desde um simples comparativo, com um limiar (comumente chamado de *threshold*) enviando esse resultado para a próxima camada de neurônios (chamamos a saída de *output*) exemplificado na Equação 1, ou algo mais elaborado, como uma função sigmoidal (POKHARNA, 2016).

$$\text{saída} = \begin{cases} 0 & \text{SE } \sum x_j w_j \leq 0.5 \\ \text{SENÃO} & 1 \end{cases} \quad (1)$$

Nesse exemplo, o *threshold* é igual a 0,5, e o neurônio só poderá enviar para as camadas subsequentes os valores binários 0 ou 1, entretanto, existem diversas funções de ativação possíveis.

Não existe uma regra geral sobre qual função de ativação apresenta uma performance melhor em diferentes modelos, dessa forma, é usual testar mesmas estruturas de rede usando diferentes funções para fins de comparação. São exemplos de função de ativação comumente utilizadas em ML, conforme Michael (2015) e Jain (2019), as ilustradas na Figura 3.

Figura 3 - Funções de ativação comumente usadas



Fonte: Adaptado pelo autor de Hamdan (2018)

O principal propósito das diferentes funções de ativação é englobar não linearidade nas saídas dos neurônios (devido ao fato que os dados disponíveis no mundo geralmente estão em formatos não lineares e deseja-se que o modelo seja capaz de lidar com essa não-linearidade). As equações 2, 3 e 4 representam os equivalentes algébricos das imagens apresentadas na Figura 3.

$$[\text{RELU}] \quad f(x) = \max(0, x) \quad (2)$$

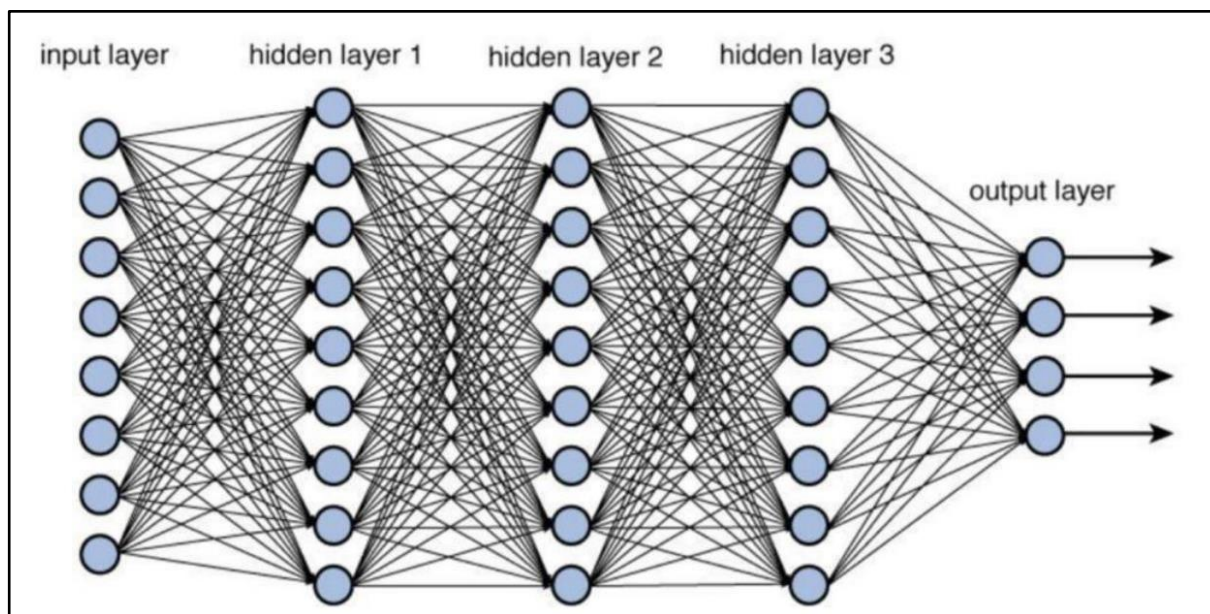
$$[\text{TANH}] \quad \tanh(x) = 2\sigma(2x) - 1 \quad (3)$$

$$[\text{SIGMOID}] \quad \sigma(x) = 1 / (1 + \exp(-x)) \quad (4)$$

A forma como cada neurônio se conecta com os demais é chamada de arquitetura da rede, conforme Souza (2008) e Parmar (2018), os mesmos são distribuídos em camadas, sendo a primeira chamada de “*input layer*”, a última camada é chamada de “*output layer*” e as

intermediárias chamadas de “*hidden layers*”. A Figura 4 exemplifica uma arquitetura de 3 *hidden layers* 9x9x9 (ou seja, três camadas, cada uma com 9 neurônios dentro).

Figura 4 - Arquitetura de uma rede 9x9x9, 3 camadas com 9 neurônios



Fonte: Adaptado pelo autor de Parmar (2018)

A rede é descrita usando apenas as camadas ocultas devido ao fato de que as demais têm um número determinado previamente de neurônios. A camada de entrada é a única que recebe os dados reais, vindo de fora do modelo, e contém o exato número de *features* do sistema (ex.: uma foto de 28x28 pixels teria 784 *features*, uma para cada pixel, e esse seria o tamanho do *input layer*).

A camada de saída, por sua vez, tem como número de neurônios o mesmo número de saídas que o número de classes a serem preditas pelo modelo, por exemplo, em uma tarefa binária (e. g. gato ou cachorro) se teria 2 neurônios e em uma tarefa multiclasse, como identificar os dígitos de 0 até 9, a rede apresentaria 10 neurônios na última camada.

É na última camada que o processo decisório final acontece, dando o resultado da predição. No caso de uma tarefa classificatória, o neurônio “mais ativo”, ou seja, com o maior valor resultante da função de ativação, é a classe resultante.

Quanto às camadas ocultas, cabe ao arquiteto da rede determinar o *trade off* ótimo entre o número das mesmas e seus respectivos números de neurônios com a performance e o tempo de processamento final do modelo, considerando que mais camadas ocultas tendem a

melhorar a capacidade do sistema (mas não de forma linear, podendo levar ao *overfitting*² e piorar o resultado) na contrapartida de requerer mais poder de processamento e, conseqüentemente, mais tempo para treinamento e predições (PARMAR, 2018).

2.3.1 *Backpropagation* - Fazendo o modelo aprender

Provavelmente, um modelo de uma classificação binária com amostras balanceadas vai começar com uma taxa de acerto de aproximadamente 0,5 (o mesmo que jogar cara ou coroa para tomar uma decisão), isso pelo fato de que todos os pesos são iniciados aleatoriamente, e é na etapa de treinamento em que se aumenta essa taxa, corrigindo cada peso de forma iterativa.

De acordo com Zhang e Sabuncu (2018) e Nar (2019), o primeiro passo para corrigir os pesos de uma rede é utilizar uma equação para avaliar o erro, chamada de *loss function* (em português função de perda), que define a rede é penalizada por cada divergência entre a sua saída e o resultado esperado, durante o período de treino.

Assim como diversas funções de ativação, existem diversas funções de perda. As mais comuns em *ML*, conforme Parmar (2018), são:

- a) *Mean Squared Error* (MSE) - a média das diferenças quadráticas entre as predições e os resultados esperados, conforme a Equação 5.
- b) *Root Mean Squared Error* (RMSE) - mesmo do MSE, porém do resultado é extraída a raiz quadrada, conforme Equação 6.
- c) *Mean Absolute Error* (MAE) - a média das diferenças absolutas entre as predições e os resultados esperados, conforme a Equação 7.
- d) *Cross Entropy Loss* - a mais comum para tarefas de classificação, pois penaliza de forma mais intensa valores confiantes errados do que valores incertos errados (de forma prática, os neurônios geram valores de 0 até 1, e caso um neurônio que devesse apresentar a saída 1 apresentasse a saída zero, ele seria mais penalizado do que um neurônio que também devesse apresentar a saída 1, porém apresentou um resultado de 0,5), conforme a Equação 8.

² *Overfitting* ocorre quando nosso modelo se ajusta perfeitamente aos dados de treino, mas não é capaz de generalizar para casos nunca antes vistos, tendo assim uma excelente performance no conjunto de treino, mas péssima no conjunto de teste. Sua contrapartida é o *Underfitting*, onde a generalização é tão grande que a performance é baixa até nos dados de treino (Ghojogh e Crowley, 2019) **The Theory Behind Overfitting, Cross Validation** - <https://arxiv.org/pdf/1905.12787>. Acessado em 4 de Mar. 2020.

$$MSE = (\sum_{i=1}^n (y_i - \hat{y}_i)^2) / n \quad (5)$$

$$RMSE = \sqrt{MSE} \quad (6)$$

$$MAE = (\sum_{i=1}^n |y_i - \hat{y}_i|) / n \quad (7)$$

$$CrossEntropyLoss = -(\hat{y}_i \log(\hat{y}_i) + (1 - \hat{y}_i) \log(1 - \hat{y}_i)) \quad (8)$$

Em todas as equações, y_i corresponde ao valor real esperado, \hat{y}_i ao valor predito pelo sistema e n corresponde ao número de amostras.

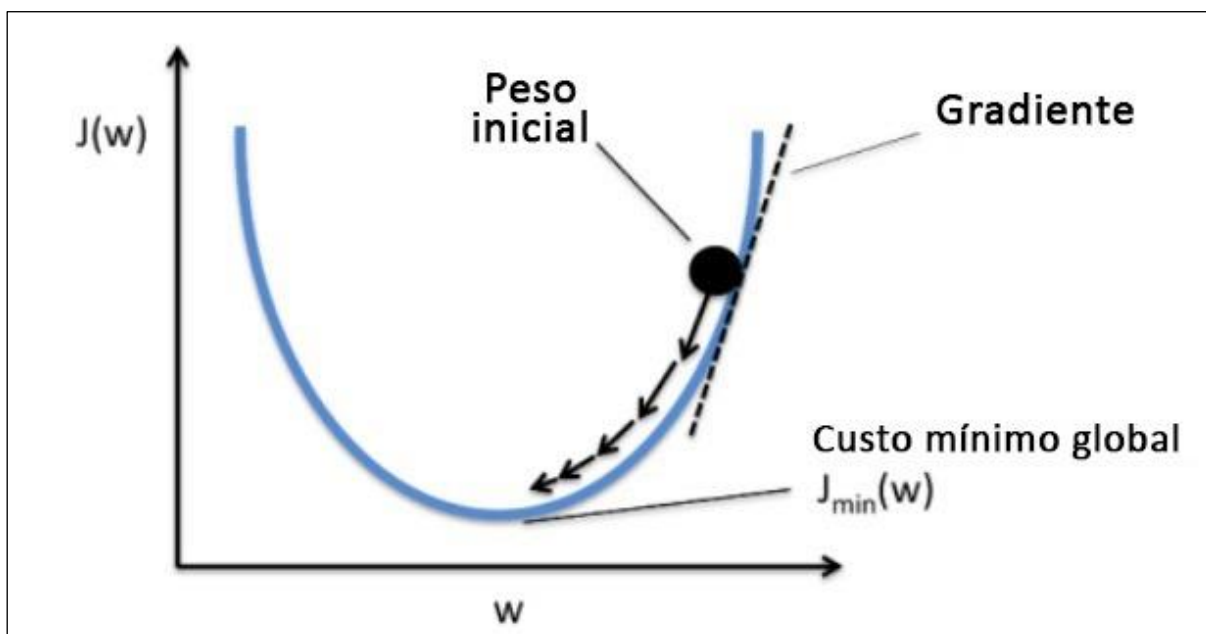
A função de *Cross Entropy Loss* também é recomendada por Nasr, Badr e Joun (2002) que obtiveram resultados superiores em termos de velocidade de convergência do algoritmo para o ponto ótimo. Todas as funções tendem a zero, à medida que a diferença entre y_i e \hat{y}_i (valor real e valor predito) se aproximam um do outro.

Após o cálculo do erro para cada amostra em cada “geração” do treinamento (cada uma das amostras de treino deve passar um número determinado de vezes pela rede, valor a ser definido pelo arquiteto da rede, chama-se esse número de geração ou época, em inglês o termo usado é “*epoch*”), é necessário uma terceira função, essa com a tarefa de otimizar os pesos de forma progressiva e na direção da última camada para a primeira, por isso o nome de *Backpropagation* (KUKREJA et al., 2016).

Outra forma de treinar a rede em relação ao número de épocas é, no lugar de se definir um número fixo, colocar alguma condição de parada, como, por exemplo, a rede atingir um determinado resultado, ou, após um número de gerações que a rede não apresenta uma melhora significativa nos resultados. Essas técnicas, por sua vez, devem ser usadas com cautela, visto que, na primeira, caso a condição de parada não seja encontrada, o sistema entrará em um *loop* infinito e, na segunda, é possível parar em um ponto de mínimo local, sem estar no mínimo absoluto possível.

Como função de otimização, utiliza-se comumente, de acordo com Song et al. (2016) e Bottou (2010), a *Stochastic Gradient Descent* (SGD), uma função convexa que pode ser descrita como um método iterativo para encontrar os valores dos parâmetros (nesse caso, cada peso da rede) de forma a minimizar o custo de uma outra função (nesse caso, a *Cross Entropy Loss*), conforme ilustrado na Figura 5.

Figura 5 - Representação de um Otimizador SGD



Fonte: Suryansh (2018)

A velocidade com que o valor do peso inicial vai se deslocar pela linha do gradiente depende de um parâmetro a ser definido pelo arquiteto da rede chamado de Taxa de Aprendizado (do Inglês, “*Learning Rate*”). Uma taxa alta pode nunca convergir para um ponto ótimo, enquanto uma taxa baixa pode levar muito tempo para convergir e, consequentemente, ter um custo de processamento muito alto associado. O objetivo é que a cada iteração o peso se aproxime do valor do custo mínimo, e o gradiente é obtido pela derivação da função custo na posição atual do peso.

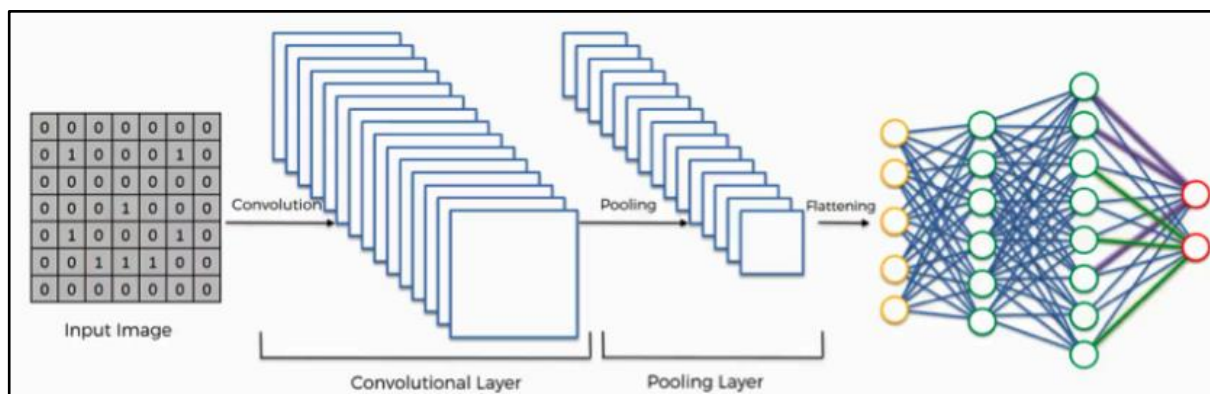
2.4 CONVOLUTIONAL NEURAL NETWORK

As CNNs podem ser descritas como extratoras de características das imagens, que são posteriormente entradas de uma rede neural. Uma CNN pode usar efetivamente informações da vizinhança de cada um dos pixels das imagens por meio de filtragens lineares, que são implementadas pela convolução entre a imagem e um filtro (daí o nome Convolucional), que destacam diferentes características marcantes, seguida de uma camada preditiva, usualmente uma RNA comum (WU, 2017).

Nessas redes, cada camada gera sucessivamente uma abstração de maior nível dos dados de entrada por meio de sequências de convoluções, criando *features* específicas para cada imagem e possibilitando que uma rede neural simples, no final dessa CNN, seja capaz de utilizar

essas novas *features* geradas como novos *inputs* em uma tarefa de reconhecimento (SZE et al., 2017), como ilustrado na Figura 6.

Figura 6 - Rede neural convolucional



Fonte: Adaptado pelo autor de Ogunyale (2019)

Como representado na figura, uma imagem de entrada pode passar por camadas de convoluções e *pooling*, que correspondem a etapa de extração de novas *features*, para só depois entrarem em uma estrutura de RNA. O número de camadas, tanto da parte de extração de *features* quanto da RNA, são definidos pelo arquiteto da rede.

Outro componente das CNNs consiste na ligação entre os *inputs* gerados pelas camadas de convolução e *pooling* com a estrutura de rede final, chamada de *flattening* (em português, achatamento). Esse achatamento é responsável apenas por deixar os dados no formato necessário para uma RNA, ou seja, ele transforma as matrizes em vetores unidimensionais (JIN, DUNDAR e CULURCIELLO, 2015).

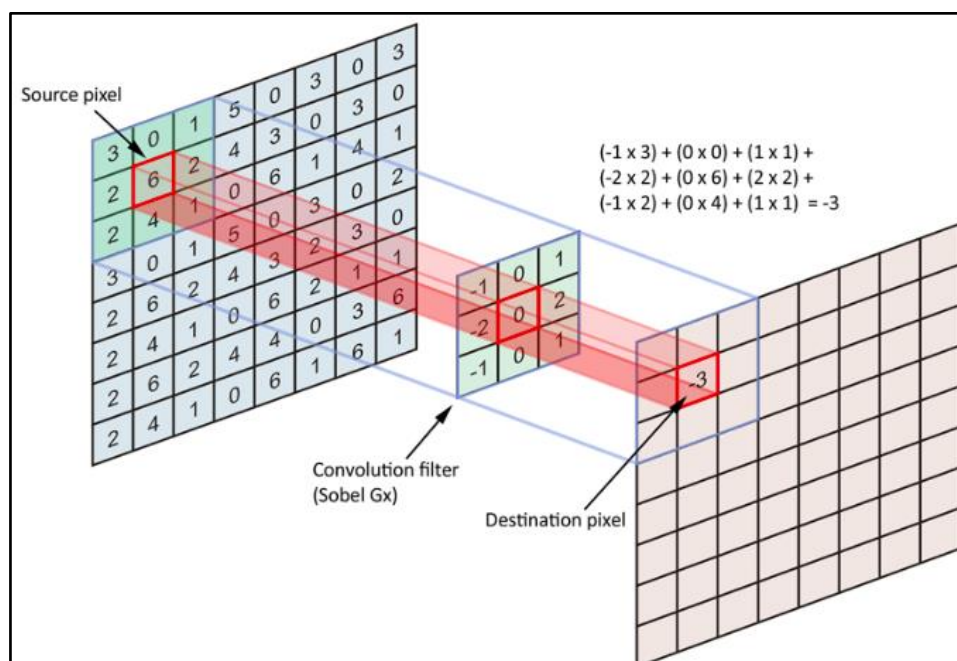
2.4.1 Convoluções

Na matemática, convolução é a operação de somatório do produto entre duas funções, ao longo da região em que as mesmas se sobrepõem, em razão ao deslocamento existente entre elas. Essa técnica é amplamente utilizada para se criar filtros digitais em processamento de sinais e, mais recentemente, em ML para processamento de imagens.

Nesse contexto, as convoluções dentro de um modelo de IA são utilizadas como filtros com o objetivo de capturar traços marcantes de imagens, que facilitem a identificação da mesma. De forma prática, cada convolução enxerga um conjunto de pixels de tamanho específico, a ser determinado pelo arquiteto da rede, e aplica um filtro (inicialmente aleatório,

ajustado em cada etapa do treino da rede). Esse filtro vai então varrer toda a imagem (ALVES, 2018), conforme representado na Figura 7.

Figura 7 - Representação de uma Etapa de Convolução em uma imagem



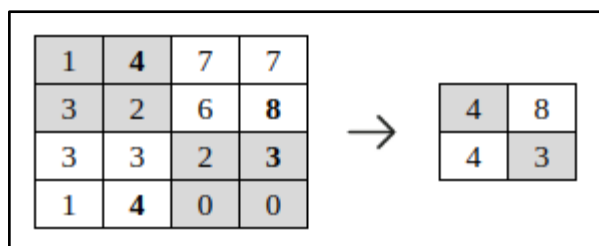
Fonte: Adaptado pelo autor de Dertrat (2017)

Cada conjunto de pixel (3x3, no caso da Figura 7) submetido à convolução, é multiplicado matricialmente pelos coeficientes do filtro, resultando em um novo valor de pixel. Ao final, quando o filtro percorre todos os pixels da imagem, se obtém o resultado da convolução, uma nova imagem, que serve como entrada para a próxima camada da rede (DERTAT, 2017).

2.4.2 Pooling

Outra camada existente em uma CNN é a chamada de *pooling* (ou *max pooling*), que consiste na redução dimensional de cada saída gerada pelas convoluções. O *max pooling*, por sua vez, não tem parâmetros a serem ajustados ao longo da etapa de treino, ele simplesmente retorna o maior valor de um conjunto de pixels, conforme a Figura 8.

Figura 8 - Aplicação de um Max Pooling 2x2 em uma imagem



Fonte: Elaborado pelo autor (2020)

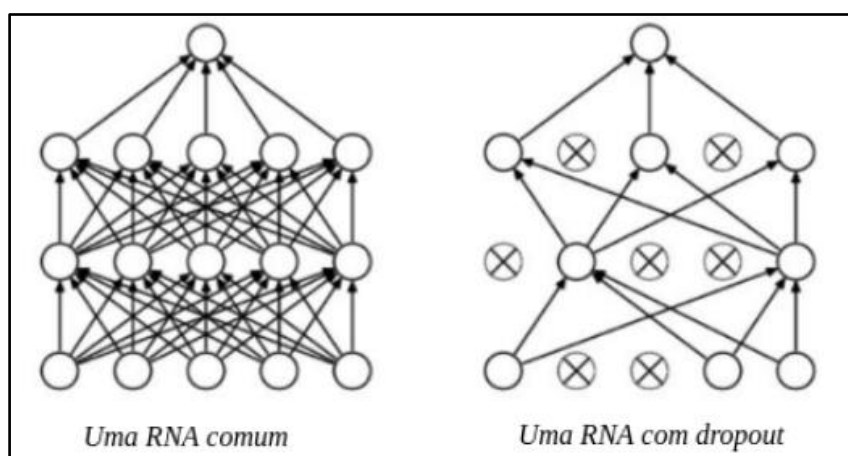
O objetivo dessa etapa é simples: reduzir o número de parâmetros da rede, agilizando e simplificando o processo de treinamento da rede e reduzindo também um possível *overfitting* em imagens do conjunto inicial de treino (CORNELISSE, 2018). O conjunto de pixels a passar pelo *max pooling* também é definido pelo arquiteto da rede.

2.4.3 Dropout

Outra técnica que busca reduzir o *overfitting* durante a etapa de treinamento e é incorporada nas arquiteturas de CNN se chama *dropout*. Ela consiste em remover aleatoriamente um percentual (definido na construção da rede) de neurônios a cada imagem analisada em cada geração (BUDHIRAJA, 2016).

Quando um neurônio é escolhido para ficar inativo durante uma etapa de treino, ele simplesmente é ignorado das etapas de cálculo, conforme a Figura 9 ilustra.

Figura 9 - Aplicação de *dropout* em uma estrutura de redes



Fonte: Adaptado pelo autor de Budhiraja (2016)

A adição de *dropout* nas camadas da CNN tende a aumentar o tempo de treino até o modelo convergir, sendo necessário equilibrar seu uso com a manipulação da taxa de aprendizado. Quando o modelo é posto em produção (após a etapa de treinamento), não se aplica *dropout*.

2.4.4 Aplicações de IA e CNNs área da saúde

Aplicações de IA têm sido amplamente estudadas na área da saúde. Um fator crucial que impulsiona isso é o crescimento exponencial dos dados disponíveis (KLANG, 2018), que torna humanamente impossível a tarefa de analisar todos os fatores envolvidos de forma rápida e eficiente sem o uso de algoritmos para auxiliar no processo.

Exemplos de aplicações recentes são a implementação de máquinas de suporte de vetor por Gu et al. (2019) na detecção de nódulos pulmonares, que apresentou um *recall* de 87,81%, o uso de uma rede neural para diagnóstico de pneumonia em imagens de Raios-X por Rajpurkar et al. (2017) com um *f1-score* de 43,50% (nesse mesmo estudo, compararam a performance do modelo com 3 radiologistas, que tiveram um resultado médio de 38,70% de *f1-score*) e o modelo de aprendizado profundo de conjunto (*Deep Ensemble*) para detectar melanoma em imagens de lesões cutâneas construído por Phillips et al. (2019) com uma área sobre a curva ROC de 90,10%.

Na área da saúde, a análise de imagens tende a exigir elevados esforços, muitas vezes em trabalhos repetitivos e tediosos, sendo suscetível ao erro humano e a divergência entre diferentes especialistas (RAZZAK, NAZ e ZAIB, 2017). Uma alternativa para isso são os avanços da área de visão computacional, porém a tradicional ML (modelos clássicos como regressão logística, árvore de decisão e RNAs de poucas camadas) tende a não lidar bem com a complexidade elevada desses problemas.

Nesse cenário, as CNNs se destacam na tarefa de análise de imagens, citando como exemplo o trabalho de Kaldera, Ramesh e Disssanayake (2019), no qual aplicaram a rede para classificação de tumores cerebrais com um índice de confiabilidade de 94,6%. Outras aplicações, como a detecção de retinopatia diabética, com uma acuracidade de 89,5% (BROOKS, 2018) e a tarefa de reconstrução de TCs (GUPTA, 2018), também são destaques.

Devido à alta performance de modelos de CNN em tarefas de reconhecimento de imagem, a aplicação dessa tecnologia em imagens radiológicas vêm aumentando ao longo dos anos (YASAKA e ABE, 2018). Entretanto, alguns fatores limitam seu uso de forma ampla na área: a baixa interpretabilidade dos modelos, quanto mais profunda a rede (maior número de

camadas), mais difícil é explicar e prever seus resultados e a intransponibilidade de modelos treinados entre diferentes hospitais, como demonstrado por Zech et al. (2018), é necessário prestar atenção em como as diferenças entre locais e populações de pacientes afetam o desempenho dos modelos.

Em suma, é esperado que a IA mude de forma gradual a forma como a prática clínica de análise de TCs trabalhe, com resultados cada vez melhores em termos de performance, ajudando os radiologistas em suas análises. Entretanto, uma possível alteração de escopo de trabalho desses especialistas pode surgir com o advento do uso de ML, com tarefas como auxiliar na preparação e anotação de conjuntos de dados para treinamento e teste de modelos de ciência de dados (YASAKA e ABE, 2018),.

Outros estudos, como o de Panch, Szlovits e Atun (2018) propõe que, com a evolução da ciência de dados, novos processos na área da saúde possivelmente serão executados de forma colaborativa por seres humanos e modelos de IA, aproveitando tanto das habilidades humanas de levantar hipóteses e analisar novos casos quanto da capacidade de computadores em analisar volumes gigantescos de dados de forma rápida. Os autores ainda exemplificam uma possível aplicação, para auxiliar no diagnóstico de pneumonia, onde um modelo processaria inicialmente as imagens dos exames e destacaria qual região o médico deve focar sua avaliação, com o intuito de auxiliar o fluxo de trabalho, e reduzir a chance de erro do especialista.

Apesar das vantagens e desenvolvimentos promissores que surgem cada vez mais a cerca de aplicações de ML na área da saúde, uma questão que se tem levantado sobre a aplicação prática de modelos matemáticos, como o deste trabalho, nas instituições e empresas, é a ética envolvida. Ao longo dessa pesquisa, optou-se por não entrar nesse mérito, focando apenas na aplicação de um modelo matemático, na avaliação dos resultados e na construção de uma aplicação WEB para o mesmo. Entretanto, com o intenso crescimento na utilização de *Machine Learning*, nos mais diversos ramos, algumas perguntas começam a surgir no âmbito ético, como “de quem é a responsabilidade por um erro de um modelo matemático caso um paciente sofra algum problema por causa de um diagnóstico errado?”. O artigo “*Ethical principles in machine learning and artificial intelligence*” (princípios éticos em aprendizado de máquina e inteligência artificial) da revista *Nature* traz uma série de cenários em que o uso de IA deve ser avaliado eticamente, como no uso de veículos autônomos ou no meio jurídico (LOPIANO, 2020).

Outra questão para a aplicação de modelos de ML na área da saúde é a disponibilidade de mão-de-obra especializada para a construção dos mesmos. Este ponto vem sendo facilitado ao longo dos anos, apesar de ser um tipo de especialização ainda escassa no mercado, a

tecnologia é cada vez mais acessível. Com bibliotecas *open-source* como o *TensorFlow*, da Google, sistemas que facilitam a implementação do ponto de vista de infraestrutura, como o *AWS Machine Learning* e diversos treinamentos e capacitações disponíveis no mercado, as empresas podem capacitar seus funcionários ou mesmo contratar serviços terceirizados para auxiliar na construção desses modelos.

2.5 AVALIAÇÃO DE MODELOS DE CNNs

Apesar da facilidade de interpretação, apenas avaliar a acurácia (número de acertos sobre o total de amostras testadas) de um modelo de IA tende a levar a interpretações errôneas, principalmente em problemas desbalanceados (prevalência de uma categoria em detrimento de outras), sendo necessário utilizar outras métricas complementares (DANJUMA, 2015).

Por exemplo, um classificador de *spam* que simplesmente recebe um e-mail e considera como não *spam*, considerando um cenário em que a cada 1000 emails recebidos, apenas 1 é *spam*, esse classificador teria uma acurácia de 99,9%, mesmo nunca acertando casos que realmente são *spam*.

Dessa forma, é importante encontrar métricas adequadas que avaliem o desempenho de um classificador independentemente do cenário de distribuição dos dados. Em um classificador binário, conforme Galdi e Tagliaferri (2018), são comuns os seguintes parâmetros (considera-se um cenário de avaliação de existência ou não de um tumor, por exemplo):

- a) Verdadeiros positivos (TP), avaliações corretas de presença de tumor.
- b) Verdadeiros negativos (TN), avaliações corretas de ausência de tumor.
- c) Falsos positivos (FP), avaliações incorretas de presença de tumor.
- d) Falsos negativos (FN), avaliações incorretas de ausência de tumor.

Dados os parâmetros apresentados, é possível calcular métricas como *f1-score*, precisão, *recall* ou mesmo a acuracidade do modelo. É importante calcular todas essas métricas a cada alteração na arquitetura do modelo ou de algum parâmetro específico do mesmo, para identificar quais são as melhores combinações possíveis para uma determinada tarefa. As Equações 9, 10, 11 e 12 representam as métricas de acuracidade, precisão, *recall* e *f1-score*, respectivamente.

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \quad (9)$$

$$\text{Precisão} = \frac{TP}{TP + FP} \quad (10)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (11)$$

$$F1\text{-score} = \frac{2 \cdot \text{precisão} \cdot \text{recall}}{\text{precisão} + \text{recall}} \quad (12)$$

Enquanto a acurácia atua como uma média simples do modelo (pode ser resumida a todos os acertos sobre o total de amostras), a precisão descreve dentre todas as predições de positivo quantas estão corretas, o *recall* mostra dentre todos os casos reais positivos, qual o percentual de acerto e o *f1-score* é uma média harmônica entre precisão e *recall* (DANJUMA, 2015).

Ou seja, quando FP são considerados mais prejudiciais que FN (na área de investimentos, por exemplo, é comum a preferência por um modelo que deixe um bom investimento passar do que um que aponte um investimento ruim como sendo bom) deve-se atentar mais para a métrica de precisão. No caso contrário, quando FN são mais prejudiciais do que FP (na área da saúde isso é comum, visto que é pior não detectar uma doença em um paciente doente do que detectar uma doença em alguém saudável) o *recall* é prioritário (RODRIGUES, 2019).

Quando ambos os erros são problemáticos igualmente, *f1-score* é recomendado, tendo em vista que, por ser uma média harmônica, será sempre mais próximo do menor valor entre o *recall* e a precisão. No caso de amostras balanceadas (números similares de amostras de cada classe), a acurácia também pode ser utilizada.

Outro artifício para representar a performance do modelo comumente utilizado é a matriz de confusão, que dispõe os valores de TP, TN, FP e FN em um formato de matriz, dando ao leitor uma forma mais intuitiva de entender como a RNA se comporta com os dados ao qual foi apresentada (LEVER, KRZYWINSKI e ALTMAN, 2016). A Figura 10 ilustra uma matriz de confusão.

Figura 10 - Matriz de Confusão

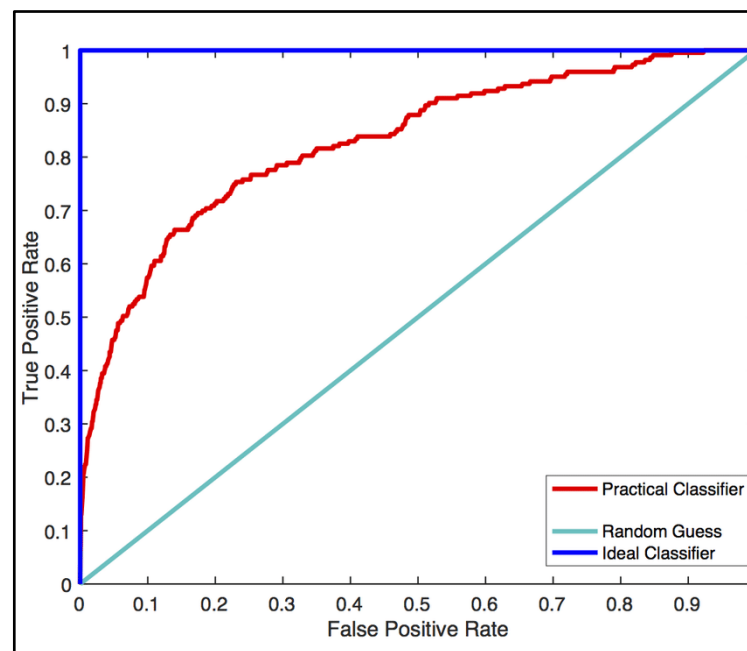
| | | Classe predita | |
|-------------|-------------|----------------|----|
| Classe real | Classe real | TP | FN |
| | Classe real | FP | TN |

Fonte: Elaborado pelo autor (2020)

Na matriz de confusão, valores elevados na diagonal principal representam uma boa performance, enquanto valores altos na diagonal secundária representam erros do modelo de ML. Outra forma comum de representar a matriz é normalizando a mesma, apresentando cada linha em um formato de percentual, onde a soma da mesma corresponde 100% dos casos da classe que corresponde.

Por fim, outra forma relevante de avaliar desempenho de *CNNs* é a área sob a curva (integral da curva) *ROC* (do inglês, “*receiver operator characteristic curve*”). A curva ROC é construída cruzando em um gráfico a taxa de TP, no eixo y, com a taxa de FP, no eixo x, de forma que a linha gerada, quanto mais próxima do canto superior esquerdo, melhor, conforme ilustrado na Figura 11.

Figura 11 - Matriz de Confusão



Fonte: Adaptado pelo autor de Chen (2017)

Conforme a imagem, o cálculo da área formada pela linha do classificador com a base do gráfico indica quão bem nosso modelo trabalha na classificação de novas imagens. Uma área de 0,5 seria equivalente ao jogar de uma moeda (representada pela linha azul claro) enquanto uma área próxima ao valor de 1 (representada pela linha azul escuro) seria um classificador ideal (YANG e BERDINE, 2017). Na prática, os resultados tendem a ficar em um meio termo (representado pela linha vermelha), sendo que quanto mais próximo da linha azul claro pior e quanto mais próximo da linha azul escuro melhor. O resultado da área é expresso pela Equação 13.

$$\text{Área sob a Curva ROC} = \int \text{recall} \cdot (1 - \text{precisão}) dt \quad (13)$$

Conforme observado, a área sob a curva *ROC* é baseada diretamente nas métricas de precisão e *recall*, dessa forma, seu resultado serve como uma forma de resumir ambas em um único valor, sendo especialmente significativa quando FP e FN são igualmente ruins para o sistema, nesses casos, a métrica é comumente utilizada em conjunto com *f1-score*.

2.5.1 Cross-Validation

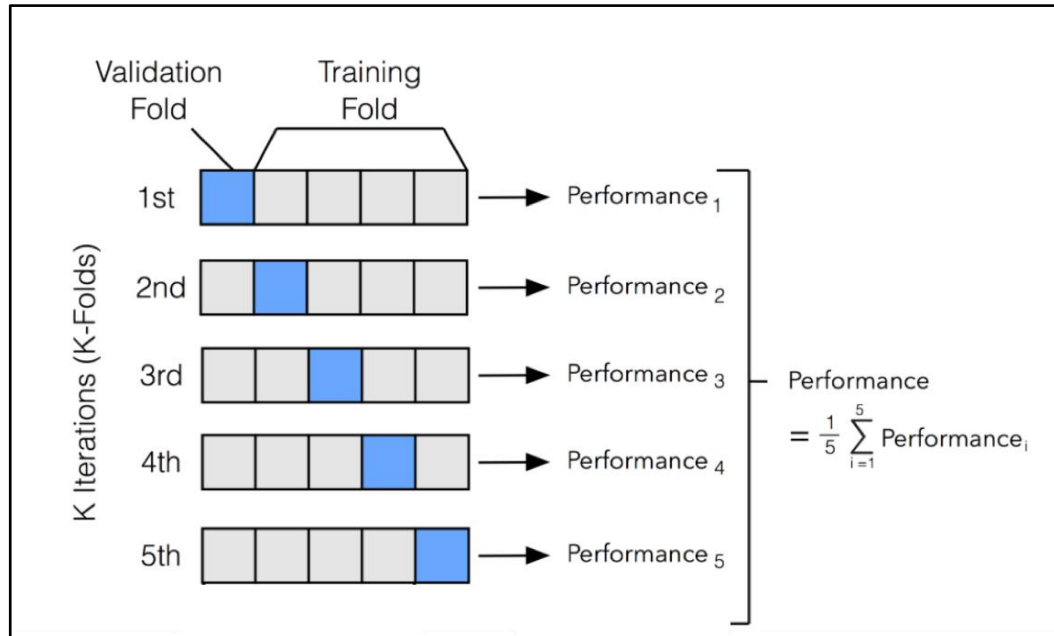
Cross-validation é uma técnica utilizada para estimar o erro dos modelos de ML durante o treinamento, utilizando um conjunto de validação, o que permite acompanhar a evolução do mesmo durante cada “época” do treinamento, antes da etapa de teste (BERRAR, 2019). O principal intuito desse método é evitar o corriqueiro problema de *overfitting*. A técnica consiste em rodar simulações utilizando diferentes combinações aleatórias, pertencendo a família de técnicas de Monte Carlo³.

Inicialmente, os dados disponíveis para treinamento do modelo são separados aleatoriamente em N conjuntos (usualmente três ou cinco). Desses conjuntos, 1 é separado para validação, enquanto os outros ficam para treino. Os resultados são armazenados e o modelo é resetado, sendo escolhido outro conjunto para ser a validação, o conjunto escolhido anteriormente entra no grupo de treino e os resultados são novamente computados (GHOJOGH e CROWLEY, 2019). Esse ciclo é repetido até que todos os N conjuntos separados inicialmente tenham sido usados como conjunto de validação por uma vez, dessa forma, o resultado final

³ Métodos de Monte Carlo são técnicas computacionais que se baseiam em amostragens aleatórias massivas para obter resultados numéricos significativos por meio de repetições de simulações, usando probabilidade e estatística. (Yoriyaz, 2009).

consiste da média aritmética de cada um dos ciclos de treino/teste. A Figura 12 ilustra o processo.

Figura 12 – Processo de *Cross-Validation*



Fonte: Adaptada pelo autor de Asiri (2018)

Com essa técnica busca-se reduzir significativamente o viés, pois usa-se a maioria dos dados para validação, e também reduz significativamente a variação, pois a maioria dos dados também está sendo usada no conjunto de treino (GUPTA, 2017). Espera-se também que a utilização da mesma faça com que o resultado encontrado na etapa de treino seja parecido com o resultado da etapa de teste, evitando a discrepância usual quando o modelo apresenta o problema de *overfitting*, onde os dados de treino têm uma precisão alta, enquanto a performance do teste é baixa.

2.6 OTIMIZAÇÃO DE HIPERPARÂMETROS

Uma rede neural apresenta uma série de parâmetros que são otimizados durante o treinamento da mesma. Um exemplo disso é o valor de cada um de seus neurônios, que podem variar de menos infinito até mais infinito. Para cada um desses parâmetros, inicializa-se com valores pseudo-aleatórios e o ajuste dos mesmos ocorre por meio de algoritmos como *backpropagation* associados a funções de perda, até encontrar os melhores valores para cada um.

Um grupo de valores, no entanto, não se altera durante o treinamento da rede, mesmo tendo influência direta no resultado, sendo fixo desde a concepção da rede. Esses valores são chamados de hiperparâmetros e são definidos, em geral, pelo arquiteto da rede, como, por exemplo, o número de camadas da rede, ou o número de neurônios em cada camada (Bergstra et al., 2013),

O grande desafio relacionado a busca pelo melhor conjunto possível de hiperparâmetros é o alto custo para um grande número de testes. Cada avaliação requer o cálculo do desempenho do modelo pós-treinamento com todos os dados disponíveis e, tecnicamente, o número de diferentes combinações possíveis de hiperparâmetros é infinito (Bergstra e Bengio, 2012). Além disso, os hiperparâmetros podem pertencer a 3 distintas categorias:

- a) contínuos (taxa de aprendizado);
- b) discretos (número de neurônios e de camadas);
- c) categóricos (tipo de função de ativação).

Dependendo dos recursos disponíveis em termos de *hardware* e do tamanho dos conjuntos de dados, cada treino com diferentes hiperparâmetros pode ter elevados custos computacionais, por isso, uma análise cuidadosa é fundamental (Krizhevsky et al., 2012). Modelos de *deep learning* são característicos por necessitar de grandes conjuntos de dados e, conseqüentemente, a tarefa de otimizar os hiperparâmetros tende a ser mais custosa do que os ganhos proeminentes de pequenas melhorias na acuracidade do modelo.

Os principais métodos de otimização dos hiperparâmetros são a “busca manual”, no qual o arquiteto faz testes com valores escolhidos manualmente, baseado em experimentos similares que tenha conhecimento ou literaturas, a “busca de grade”, onde se delimita um espaço de escolha particular, no qual acredita-se que seja um potencial local para o melhor resultado, e inicia-se uma varredura por esse espaço e, por fim, a “busca aleatória” (Bergstra e Bengio, 2012).

A busca aleatória é uma alternativa que busca reduzir o esforço da busca de grade e ainda possibilitar o aumento do espaço de escolha, tendo em vista que uma busca de grade completa faz com que o número de testes a serem feitos cresça de forma rápida (por exemplo, para um RNA simples, se delimitarmos apenas dois hiperparâmetros, número de camadas entre 1 ou 3 e número de neurônios de 10 a 12, já temos um espaço de busca de 39 combinações diferentes).

Por fim, o método que tende a demonstrar os melhores resultados em termos de otimização de hiperparâmetro para redução de erro, mas, ao mesmo tempo, a pior performance em termos de custo computacional (e por isso acaba sendo pouco utilizado em cenários práticos) é o de otimização de hiperparâmetros em *CNNs* por meio de Algoritmos Genéticos (AGs), conforme Xiao et al. (2020).

Os AGs, conforme o nome sugere, são algoritmos de busca baseados nos mecanismos da seleção natural e da genética (LIU e WANG, 1999), o que significa que ele trabalha com uma população de indivíduos (que na tarefa em questão são os diferentes grupos de hiperparâmetros) e apenas os “mais fortes” vão sobreviver e gerar “descendentes”, ou seja, os conjuntos que tiverem os menores erros são priorizados.

As etapas básicas de um AG simples são, conforme Schuler (2007):

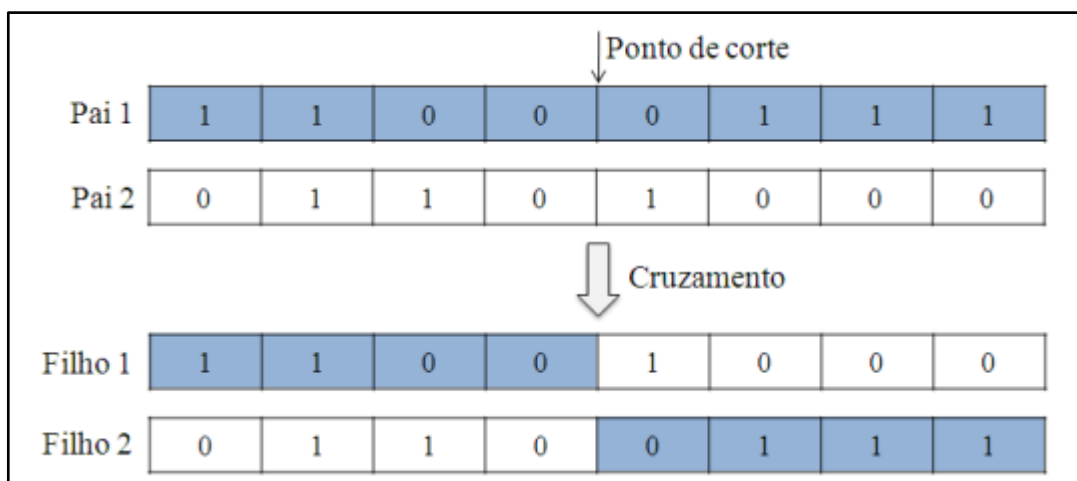
- a) Formação da população Inicial.
- b) Avaliação da população pela função objetivo.
- c) Seleção da população que gerará os descendentes.
- d) Cruzamento.
- e) Elitismo.
- f) Mutação.
- g) Substituição da população original.
- h) Retorna ao passo B caso o critério de parada não seja atingido.

De acordo com os passos listados, logo após a formação da população inicial, tem-se a etapa com o custo computacional mais alto - avaliar a população pela função objetivo - a tarefa escala rapidamente quando se fala em melhorar o desempenho de uma *CNN*. Nesse caso, essa etapa B depende de treinar a rede com cada um dos indivíduos da etapa A, fazer previsões com um conjunto de teste e avaliar os resultados de acordo com a função objetivo escolhida (precisão, *recall*, acuracidade, *f1-score*, etc).

Um percentual dos indivíduos que tiveram o pior desempenho em relação a métrica escolhida (em geral, os 50% piores) são então descartados e os restantes passam pelo processo de “cruzamento”, que nada mais é que a geração de um novo grupo de indivíduos para a próxima iteração do AG. Para Linden (2008), o cruzamento é o operador mais importante dos AGs, pois é o responsável pelo fato do método não poder ser comparado a uma busca aleatória, fazendo com que o sistema identifique as áreas mais promissoras de busca e combiná-las de forma a tentar gerar soluções de maior qualidade para o problema em questão.

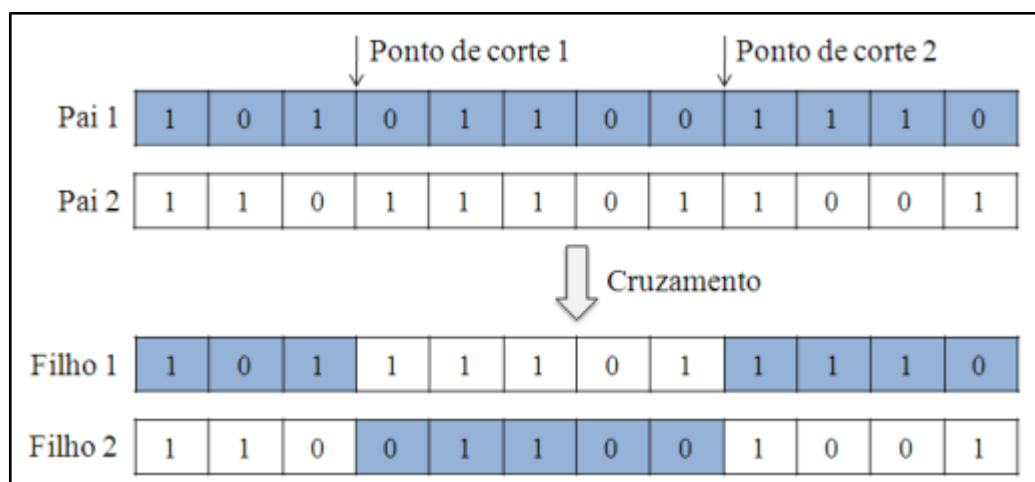
Existem diversas formas de realizar o cruzamento, as mais comuns são ponto de corte único e ponto de corte múltiplo (BOECHEL, 2003) são ilustradas nas Figuras 13 e 14.

Figura 13 - Cruzamento de ponto de corte único



Fonte: Adaptada pelo autor Boechel (2003)

Figura 14 - Cruzamento de ponto de corte múltiplo



Fonte: Adaptada pelo autor Boechel (2003)

Em ambas as metodologias, a escolha dos “pais” é aleatória (dentro dos indivíduos que foram elegíveis para o cruzamento, ou seja, os de melhor desempenho) e a localização dos pontos de corte também é aleatória.

Após todos os “filhos gerados”, os “pais” que tiveram o melhor resultado na etapa B são mantidos para a próxima geração, junto com o novo grupo de indivíduos, essa etapa se chama de elitismo e tem apenas o intuito de preservar os melhores resultados já obtidos para evitar uma possível perda de performance de uma iteração do método para outra. Em geral, de

1 a 10% dos melhores indivíduos são “protegidos” pelo elitismo, sendo também imunes aos efeitos da próxima etapa, a mutação (CASTRO, 2001).

Dependendo da população inicial escolhida, pode não haver variedade suficiente de indivíduos para assegurar que o AG cubra um espaço de busca suficientemente grande para atingir o ponto ótimo do problema. Além de que, devido ao cruzamento sempre dos melhores indivíduos e do elitismo, o algoritmo pode parar em um dos chamados ótimos locais (uma região que apresenta resultados melhores que seus vizinhos, porém não é a melhor região do sistema). Aderindo uma solução para esse problema, existe a etapa de mutação (CUS e BALIC, 2003).

A mutação é realizada em um percentual baixo dos indivíduos, entre 1 a 15%, conforme Deb e Dixit (2008). Para evitar um problema de convergência, a mutação troca algum valor do indivíduo aleatoriamente. Por exemplo, um indivíduo que tem uma camada com 100 neurônios pode “sofrer uma mutação” e passar a ter uma camada com 500 neurônios.

Após esse processo, a nova população está completa e pronta para substituir a população anterior. Esse processo é então repetido por um número N de vezes, ou até uma condição de parada ser encontrada (por exemplo, o *recall* do modelo com algum dos conjuntos de hiperparâmetros ser maior que 95%).

Com exceção da etapa B, o restante do AG exige pouco processamento do computador e é de fácil execução. Uma das maneiras que empresas encontram para aplicar essa forma de otimização de hiperparâmetros é, de acordo com Xiao et al. (2020), paralelizar a etapa de treinamento dos indivíduos gerados em diferentes CPUs ou, preferencialmente, GPUs (Unidades Centrais de Processamento ou Unidades de Processamento Gráfico), visto que cada amostra pode ser treinada individualmente e não depende das outras para isso.

2.7 DESENVOLVIMENTO WEB

A necessidade de integração entre sistemas, serviços e produtos é cada vez maior em indústrias dos mais diversos segmentos. Na área de ciência de dados não é diferente, com um número cada vez maior de algoritmos de IA e um volume de dados gigantesco para ser analisado, encontrar uma forma de colocar isso em “produção”, seja na forma de um serviço ou de um produto, é fundamental para a o aproveitamento máximo das pesquisas.

Inicialmente, as aplicações necessitavam que fosse instalado um programa específico no computador do cliente para utilização como interface gráfica. Esse tipo de solução fazia com que a distribuição e manutenção das aplicações fosse muito dispendiosa. Dessa forma, o

desenvolvimento WEB (*World Wide Web*) surgiu como uma alternativa, utilizando HTML (*Hypertext Markup Language* - documentos criados para serem exibidos na internet) como forma para renderizar as informações necessárias para os clientes sem a necessidade de nenhuma alteração nos equipamentos dos mesmos (GOMES, 2008).

Existem muitos protocolos e padrões designados para o desenvolvimento WEB. Alguns exemplos são: *WS-Notification*, *WS-Security*, *Web Services Description Language* (WSDL) e *Simple Object Access Protocol* (SOAP), esses padrões têm decaído em uso devido a grande complexidade para implementá-los (RICHARDSON e RUBY, 2007), dando espaço para o REST (*Representational State Transfer*), uma arquitetura simples e leve para o desenvolvimento de sistemas WEB, que busca simplicidade na disponibilização de APIs (*Application Programming Interface*). APIs, por sua vez, são um conjunto de interfaces de comunicação entre diferentes sistemas por meio de requisições e respostas (FIELDING, 2000).

A arquitetura REST é uma coleção de princípios e restrições arquitetônicas para o desenvolvimento de aplicações distribuídas na internet e não uma linguagem de programação em si. Ele foca na relação cliente-servidor, onde as requisições partem inicialmente do cliente e são respondidas pelo servidor (XAVIER, 2011), e possui 4 formas de manipulação de dados utilizando o protocolo HTTP (*HyperText Transfer Protocol*), uma para cada uma das seguintes operações:

- a) GET - retorna o estado atual do dado.
- b) PUT - inserir ou alterar algum dado.
- c) DELETE - deleta algum dado.
- d) POST - enviar dado para algum processamento.

Pode-se encontrar usos de uma arquitetura API *Restful* (forma de programar que segue o padrão REST para a criação de APIs) por toda a internet. Nela, um navegador (*software* utilizado para navegar na internet, como o *Google Chrome* ou o *Internet Explorer*) executam operações HTTP GET para solicitar dados, a fim de renderizar uma página, por exemplo, ou usuários preenchem dados que são enviados via HTTP POST para realizar um cadastro em algum serviço, entre outros.

Outro ponto do desenvolvimento de um sistema é a construção do *backend* (a “parte de trás”) e o *frontend* (a “parte da frente”). Essa distinção existe, pois, em geral, o usuário final não está interessado em todos os cálculos e processamentos que o *backend*, a parte lógica,

realiza, então, o *frontend* existe como uma abstração do *backend* de forma amigável, como um formulário para preenchimento ou uma página da internet.

Optando pela arquitetura *Restful* e com a distinção entre *backend* e *frontend*, ainda é necessário tomar mais uma decisão para o desenvolvimento WEB, a escolha da linguagem de programação. Uma linguagem de programação consiste em um conjunto de regras definidas, sintáticas e semânticas, para a implementação de um código fonte, uma forma de passar instruções de processamento para um computador (FISCHER, FRANCES, 1993).

Existem diversas linguagens de programação, C, Java, Go, Swift, PHP, R, Ruby, Assembly, Pearl e Fortran para citar algumas, cada uma delas criada para um propósito e com uma série de vantagens e desvantagens. Algumas delas são mais focadas em performance de execução (o computador compila e executa posteriormente elas mais rápido), outras priorizam a produtividade dos programadores que as usam. Neste trabalho, a linguagem escolhida para executar os procedimentos necessários é Python.

2.7.1 Python

Python é uma linguagem muito versátil, criada durante os anos 80 por Guido van Rossum, com sua primeira versão lançada em 1991 (JOHN, 2016), seu nome é um tributo ao grupo britânico de comédia, *Monty Python*. O principal objetivo da linguagem é ser simples, mas ao mesmo tempo funcional, sendo capaz de desenvolver e gerenciar até mesmo grandes sistemas.

Um dos principais diferenciais de Python é sua legibilidade. Isso ocorre, pois, seu código não exige o uso de marcadores frequentes (como ponto e vírgula ao final de cada sentença, colchetes e chaves), a Figura 15 compara uma simples função (escrever “*HelloWorld*”) em Python e Java. Além disso, é um software livre para uso disponível para diversas plataformas. (BEAZLEY e JONES, 2013).

Figura 15 -Função “HelloWorld” em Python e em Java.

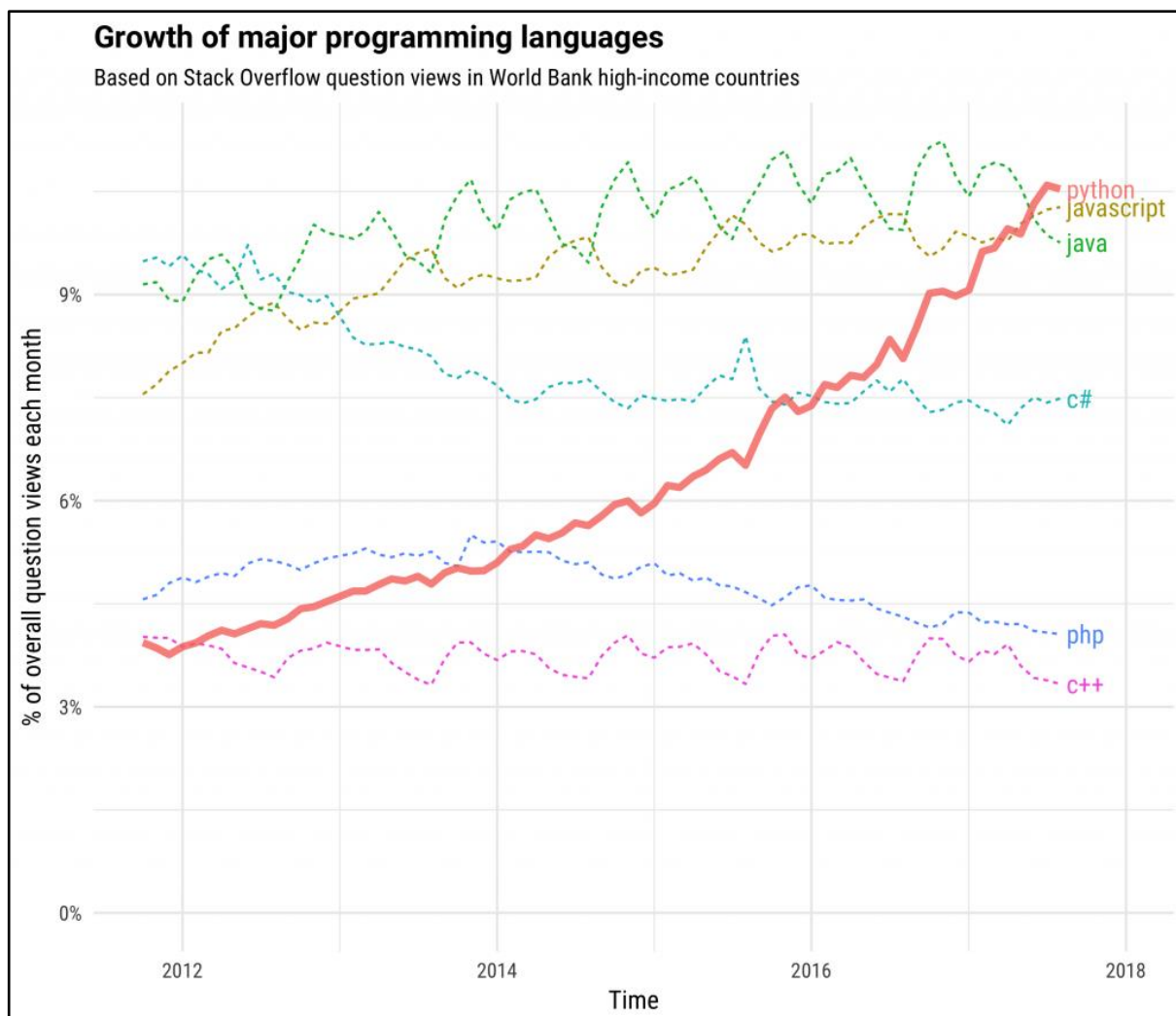
```
3
4
5
6 Em Python:
7 print('HelloWorld')
8
9
10
11 Em Java:
12 public class HelloWorld {
13     public static void main(String[] args) {
14         System.out.println("HelloWorld");
15     }
16 }
17
18
```

Fonte: Elaborado pelo autor (2020)

Apesar de ser uma linguagem relativamente antiga, Python teve uma ascensão muito grande nos últimos anos, sendo a que mais cresceu em percentual de uso. Esse crescimento é em parte resultado de suas aplicações em ciência de dados e inteligência artificial, motivo também que a concedeu o prêmio de “linguagem de programação do ano 2020” pela TIOBE, uma métrica que calcula a popularidade de cada linguagem de programação baseada em mecanismos de busca. O crescimento em 2020 de Python foi de 2,01%, seguido de C, com 1,66% e R com 1,10% (TIOBE, 2021).

Não apenas dentro da pesquisa TIOBE que Python tem tido destaque. Recentemente, o *StackOverflow*, site de perguntas e respostas referente a programação e áreas relacionadas, também conduziu uma pesquisa que aponta Python como a linguagem que mais cresceu nos últimos anos, a Figura 16 mostra o resultado do estudo, que foi feito em 2018 (ROBINSON, 2018).

Figura 16 - Crescimento de Python dentro do StackOverflow



Fonte: Robinson (2018)

Dentro das engenharias, o uso de Python também tem crescido aceleradamente. É possível automatizar uma série de processos utilizando a ferramenta, bem como fazer simulações complexas, cálculos matemáticos e estatísticos, controle de planilhas e dados, manutenção preditiva, aprendizado de máquina, gerenciamento de tarefas, aplicações WEB, conexão a bancos de dados relacionais e não relacionais, gestão logística, análise financeira, entre muitas outras tarefas que podem surgir, tudo isso com o auxílio de Python e suas diversas bibliotecas.

Dessa forma, engenheiros e cientistas têm aproveitado dessa linguagem de programação para resolver os problemas e desafios de seu dia a dia nos mais diversos ramos de atuação, como pode-se observar no trabalho de Millman e Aivazis (2011) intitulado “Python

para Engenheiros e ciência” e de Fangohr (2015), “Python para engenharia e ciência da computação”.

3 MÉTODO PROPOSTO

3.1 CARACTERIZAÇÃO DA PESQUISA

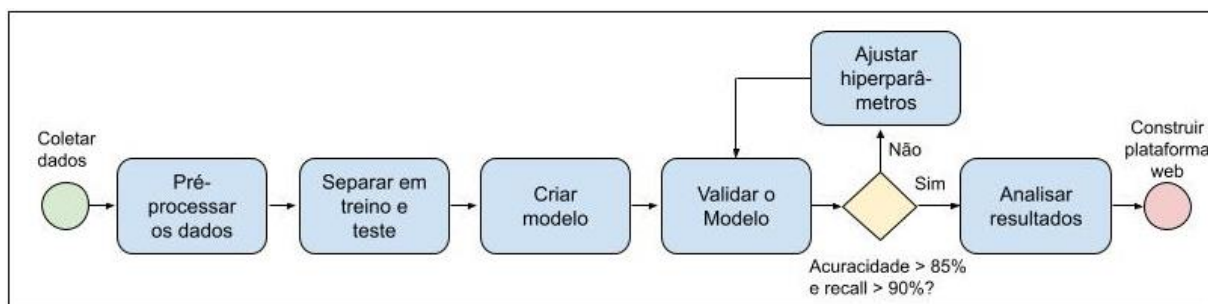
A pesquisa desenvolvida aqui é de natureza aplicada, ou seja, existem interesses locais com a solução proposta, com uma abordagem qualitativa, considerando que se busca identificar padrões em dados de característica descritiva e de caráter dedutivo. A partir da análise dos dados existentes, tenta-se generalizar um modelo capaz de explicar o todo, com base na lógica (PRODANOV, 2013).

Quanto a caracterização de objetivo, a pesquisa é exploratória, de forma que a finalidade é descrever os fatos e comportamentos dos dados sem interferir nos mesmos, utilizando para isso o método experimental. Este tipo de pesquisa visa avaliar diferentes hipóteses e validar (ou invalidar) as mesmas com utilização de simulações e testes, buscando encontrar uma relação entre as variáveis envolvidas (PRODANOV, 2013).

3.2 MÉTODO PROPOSTO PARA O TRABALHO

Para o andamento e execução do trabalho foi proposta uma metodologia de trabalho, ilustrada na Figura 17. Aqui se podem ver as principais etapas desenvolvidas, onde cada uma é detalhada a seguir.

Figura 17 - Metodologia do trabalho proposto



Fonte: Elaborado pelo autor (2020)

As principais macro atividades são listadas e descritas nos subitens seguintes.

3.2.1 Etapa 0 - Coletar os dados

O banco de dados coletado, chamado de LIDC-IDRI (*Lung Image Database Consortium image collection*), consiste em um trabalho em conjunto, realizado por entidades públicas e privadas, a NCI (National Cancer Institute), FNIH (*Foundation for the National Institutes of Health*), FDA (*Food and Drug Administration*) e mais 7 centros acadêmicos e 8 companhias médicas (ARMATO et al., 2011). Os dados são disponíveis publicamente no endereço <https://wiki.cancerimagingarchive.net/display/Public/LIDC-IDRI>, e consistem de 125 GB de imagens no formato DICOM (do inglês, *digital imaging and communications in Medicine*, o formato mais comum para arquivos gerados por TCs) e mais arquivos em XLS e CSV com informações sobre as imagens.

O objetivo desse levantamento, que contém 1018 casos, cada caso com aproximadamente 50 imagens, é prover recurso suficiente para que pesquisadores possam desenvolver sistemas CAD (*Computer-aided diagnostic*) para auxiliar na detecção de nódulos pulmonares, além de validar e disseminar práticas clínicas. Todas as TCs são casos reais, entretanto, todas passaram por um processo de anonimização, onde informações sensíveis, como o nome do paciente, foram removidas.

Este conjunto de dados têm sido bastante utilizado em pesquisas acadêmicas. O artigo de descrição do mesmo (ARMATO et al., 2011) já tem, atualmente (janeiro de 2021), 1174 citações acumuladas. Alguns exemplos de pesquisas recentes realizadas com os dados: segmentação de TCs utilizando clusterização *K-means* (SAHU et al., 2020), validação de aprendizagem para TCs por desequilíbrio de classe multivariável (ANH; HOANG, 2019) e segmentação de nódulos usando campos aleatórios de *Markov* e *Bayesian probability* (ZHANG et al., 2020).

3.2.2 Etapa 01 – Pré-Processar os dados coletados

Com os dados adquiridos, a primeira etapa consistiu em analisar e pré-processar as imagens coletadas, com o intuito de padronizar e converter as mesmas em matrizes numéricas que servem de dados de entrada para o modelo a ser construído.

A primeira atividade dessa etapa é converter o formato das imagens de tomografia, que são geradas em DICOM para o formato JPEG e padronizar as dimensões das mesmas. Neste trabalho se optou por reduzir a resolução de todas para a ordem 100x100 por meio de uma

interpolação simples (as dimensões originais eram de 512x512 e essa redução dimensional tem o objetivo de acelerar o processo de treinamento).

Após isso, ordená-las em relação a ordem da altura que cada uma corresponde no corpo do paciente (essa ordenação se faz necessária pois os arquivos CSV e XLS disponíveis do banco de dados referenciam as informações de cada imagem baseados nessa posição) e criando posteriormente um índice binário para cada imagem, em que 0 representa uma tomografia sem nódulo e 1 uma tomografia que apresenta nódulo pulmonar, de acordo com a classificação inicial dos especialistas. A definição do valor 0 para quando uma imagem não apresenta nódulo e 1 para quando apresenta é apenas convenção, pois o modelo matemático utilizado só aceita valores inteiros na sua camada de saída.

Esse vetor gerado com esta aplicação, consistente do par imagem - valor binário (0 quando a imagem não apresenta nódulo e 1 quando apresenta) é importante pois é utilizado tanto para treino do sistema construído quanto para a avaliação do mesmo. Entretanto, em utilizações futuras (com o modelo já treinado e avaliado) apenas a imagem servirá de entrada para o *software*, nesse caso, o valor binário será apenas a saída do sistema, indicando a presença ou não de nódulo dentro das taxas de acerto levantadas na etapa de avaliação do modelo.

3.2.3 Etapa 02 – Separar os dados em Teste e Treino

Com os dados padronizados na estrutura imagem - binário, estes foram separados em dois conjuntos distintos: um para treino da rede, o qual foi utilizado para ajustar os pesos de cada parâmetro do modelo criado, correspondendo a 80% do total de imagens, e outro para teste, com os 20% restantes. Esse conjunto de imagens para teste fica reservado com dados que não são vistos pela rede em nenhum momento durante a construção da solução/treinamento do sistema, dessa forma, ele é apto para validar de forma prática o desempenho do *software*.

Devido às amostras não estarem balanceadas, ou seja, existem mais imagens sem nódulo, em uma proporção de aproximadamente 1000 para 1, nem todas amostras da categoria dominante são utilizadas para o treinamento. Isso é feito para proporcionar um balanceamento maior dos dados. Esse procedimento busca evitar que o modelo fique tendencioso para a classe majoritária, o que aconteceria caso ele recebesse, na etapa de treinamento, amostras nesta proporção de imagens.

Realiza-se a separação nos conjuntos de treino e teste de forma pseudoaleatória, usando uma semente de geração de números aleatórios fixa para reprodutibilidade posterior. Ao fim, tem-se os seguintes conjuntos de amostras:

- a) 1800 imagens para treino (80% do total);
- b) 450 imagens para teste (20% do total).

É importante lembrar que, das 1800 imagens para treino, a cada rodada do laço de treinamento, apenas 1440 são utilizadas, as outras 360 fazem parte do pacote de validação. Estas imagens são utilizadas para avaliar o desempenho do modelo ainda durante a etapa de treinamento, pelo processo de *cross-validation*, verificando se a cada época o modelo, o mesmo apresenta alguma melhora de desempenho.

3.2.4 Etapa 03 – Criar o modelo de CNN

Para construir o modelo de CNN, utilizou-se as bibliotecas *Tensorflow* e *Keras*, ambas são projetos *opensource* para Python criados por equipes da *Google*. O *Tensorflow* teve sua primeira versão lançada em 2015, e, conforme seus desenvolvedores (Abadi et al. 2015), esta é uma ferramenta que permite a programadores e cientistas experimentarem diferentes algoritmos de inteligência artificial com diversos conjuntos de parâmetros, enquanto o *Keras* é uma API focada em agilizar experimentos práticos utilizando esses diversos conjuntos de parâmetros.

Os autores ainda citam que diversos serviços da *Google* utilizam as bibliotecas em produção e a mesma tem sido amplamente utilizada para pesquisas na área de ciência de dados, devido a sua flexibilidade para pesquisa e consistência para implantação. Nas palavras dos desenvolvedores: “Ser capaz de ir de ideias aos resultados o mais rápido possível é a chave para fazer uma boa pesquisa” e “*Keras/Tensorflow* é para *Deep Learning* o que *Ubuntu* é para sistemas operacionais” (traduzido de Abadi et al. 2015).

Com o modelo arquitetado com suas respectivas camadas de filtragem, agregação, *max pooling*, *dropout*, função de ativação e camadas ocultas de neurônios, utilizou-se as imagens do conjunto de treino para otimizar os parâmetros. Durante a etapa de treinamento, uma parte dos dados ($\frac{1}{5}$ do amostral, 360 imagens) foram separadas aleatoriamente para o processo de *Cross-validation*, os outros $\frac{4}{5}$ foram utilizados para otimizar os parâmetros da rede.

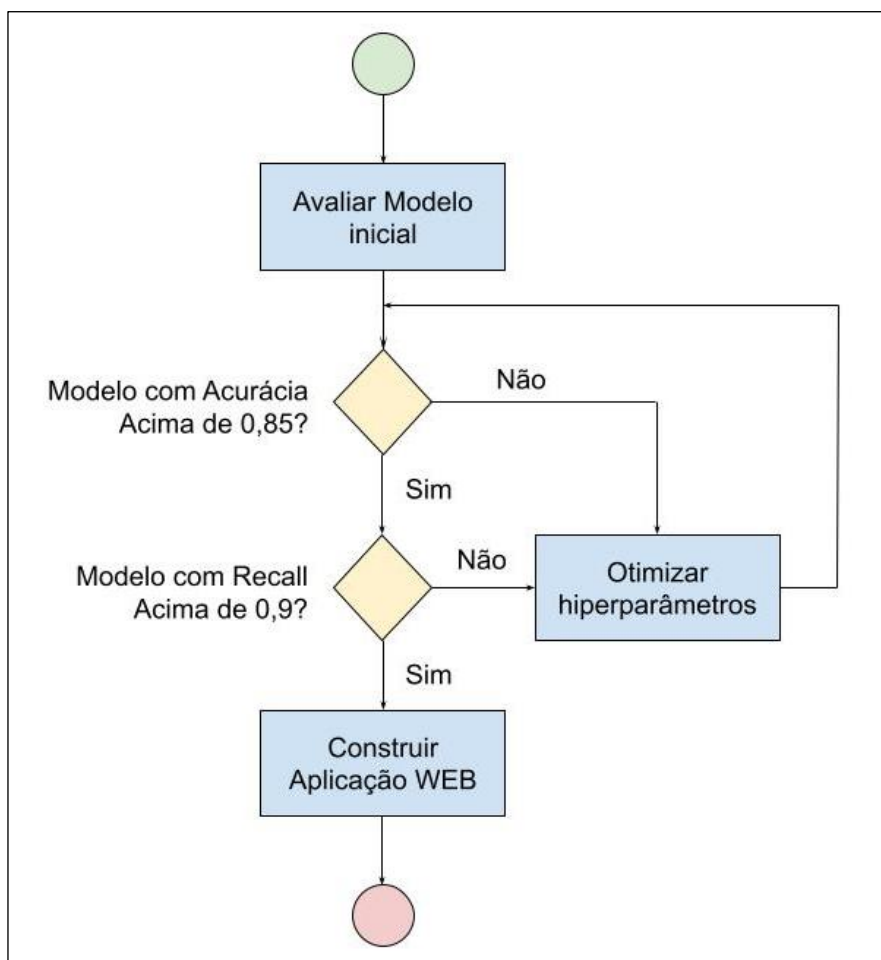
3.2.5 Etapa 04 – Validar o modelo

Nessa etapa, foi utilizado o conjunto de dados reservados para teste com o intuito de validar o modelo previamente criado. A técnica utilizada para isso foi a simples aplicação prática do modelo, simulando um caso real. Para isso, realizou-se a predição para cada imagem de teste com o sistema e se comparou os resultados encontrados com o real esperado, que são os valores definidos pelos radiologistas.

As métricas escolhidas para avaliação do desempenho do modelo foram o *recall*, com um objetivo de no mínimo 0,90 e a acuracidade, com um objetivo de no mínimo 0,85. A opção pela métrica de *recall* (além da métrica geral de acuracidade) como indicador para tomada de decisão, referente a avançar ou não nas etapas do projeto se deu pela característica da pesquisa. É considerado mais grave um falso negativo do que um falso positivo, ou seja, é mais grave que um paciente que apresenta a presença de nódulo em sua tomografia pense que não tenha do que um paciente que não tenha nenhum nódulo pense que tenha. Isso pois, em geral, no caso do resultado de “presença de nódulo”, um acompanhamento maior vai ser realizado, o que ocasionalmente vai incidir na descoberta do erro, enquanto o erro de falso negativo pode atrasar o início do tratamento e diminuir a chance de sobrevivência do paciente.

Caso a validação tenha um índice de *recall* menor de 90% ou um índice de acerto geral menor de 85%, constroem-se novas versões da rede com diferentes conjuntos de hiperparâmetros por meio de técnicas de otimização dos mesmos (a otimização será abordada em mais detalhes nos capítulos 3.2.6 e 4.5). As novas redes também são avaliadas com as mesmas métricas, o intuito é, após a otimização, encontrar uma configuração que atenda os critérios estipulados inicialmente para que o modelo possa avançar para a última parte do trabalho, a construção de uma plataforma WEB para uso prático do modelo (esse desenvolvimento será abordado nos capítulos 3.3 e 4.9). A Figura 18 ilustra esse processo.

Figura 18 - Etapa de validação e otimização



Fonte: Elaborado pelo autor (2020)

Um ponto importante do processo ilustrado é que o mesmo pode entrar em um laço infinito, onde nenhum conjunto de hiperparâmetros testados resulte no atingimento dos valores de métricas estipulados. Para evitar isso, é imposto um tempo limite para que esse laço de otimização - avaliação - otimização aconteça, de 15 dias. Após esse limite de tempo (definido de maneira arbitrária, apenas com o intuito de prosseguir com a pesquisa), é dado continuidade com o melhor resultado encontrado até então.

3.2.6 Etapa 05 - Ajuste de hiperparâmetros

Na possibilidade de o sistema não superar o objetivo das métricas estipuladas na sua primeira versão, um novo sistema é construído, com um novo conjunto de hiperparâmetros. A metodologia utilizada nesse caso é a “busca aleatória”, onde o próprio arquiteto da rede estipula

um espaço amostral para os hiperparâmetros, e a cada rodada um conjunto aleatório dentro desse range de possibilidades é escolhido de forma pseudoaleatória.

A escolha da “busca aleatória” se dá por limitações de processamento computacional. Nesta pesquisa foi utilizado um computador com processador Intel Core I7, 16gb de RAM com o sistema operacional Ubuntu 20.04, e, para o volume de dados, uma RNA clássica de 3 camadas com 1000 neurônios cada levou aproximadamente 40 minutos para ser treinada, enquanto uma *CNN* de 5201762 parâmetros (valor obtido pela própria biblioteca de Python utilizada, o *Tensorflow*) levou aproximadamente 20 horas. Ou seja, para treinar cada modelo seria necessário quase 1 dia, nesses cenários, a “busca aleatória” consegue, em menos tempo, passar por um espaço de busca maior que o de uma busca em grade.

Outra opção possível, que tende a encontrar resultados melhores, seria a otimização de hiperparâmetros utilizando algoritmos de busca evolutiva, como, por exemplo, algoritmos genéticos, conforme visto no capítulo 2.6. Entretanto, esse formato de otimização para redes consideravelmente complexas também exige um tempo elevado de treinamento, o que, em alguns casos, com modelos que necessitam de atualizações constantes, se torna inviável.

3.2.7 Etapa 06 – Analisar os resultados obtidos

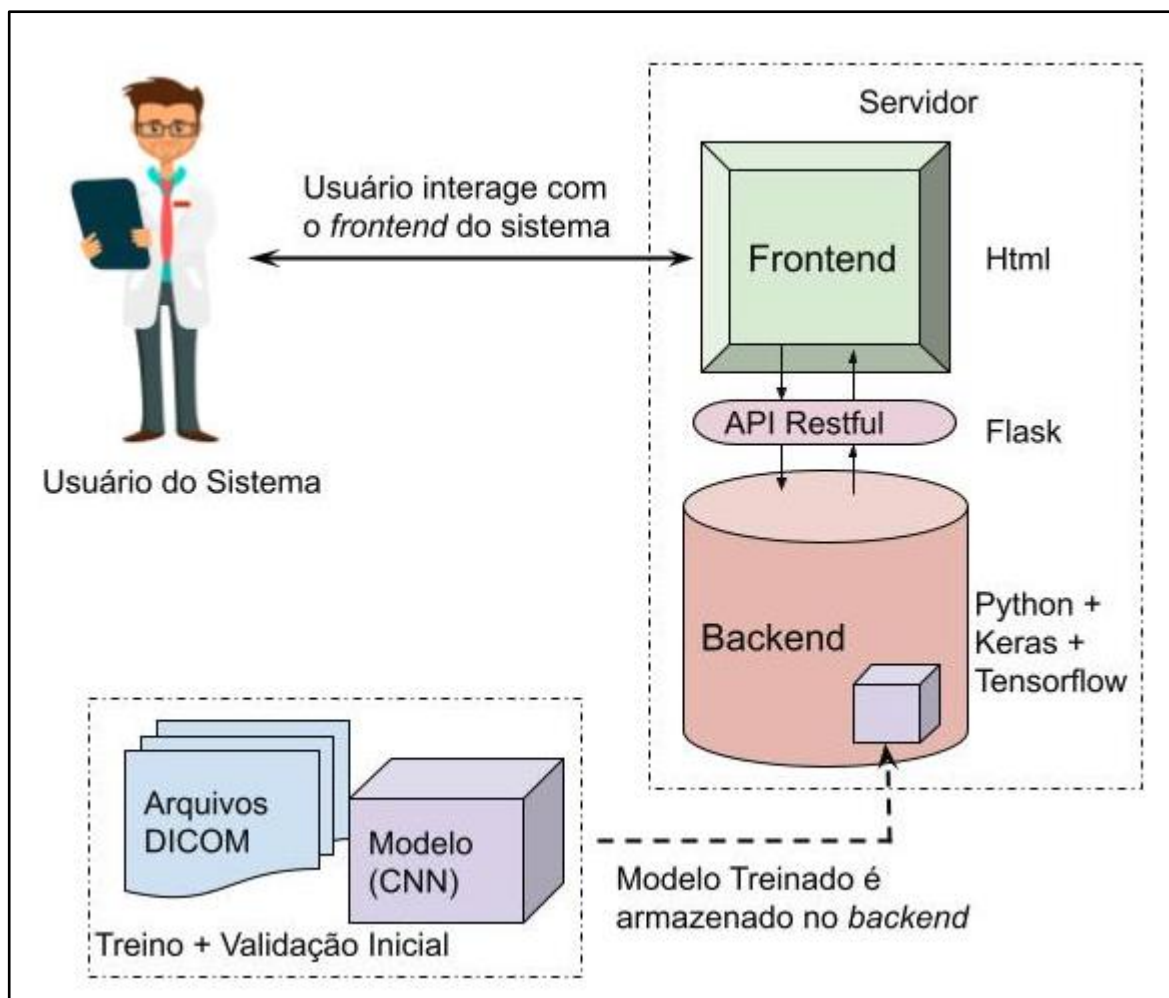
Por fim, são analisados os resultados obtidos em termos de métricas de classificação (acurácia, precisão, *recall* e *f-score*) e tempo de execução, tanto do treinamento do modelo quanto para predizer novos resultados com novas imagens.

O objetivo dessa análise é definir qual a taxa de assertividade que uma rede convolucional tem na tarefa predizer nódulos pulmonares e também a viabilidade de executar o sistema em um ambiente de produção.

3.3 CONSTRUÇÃO DE UMA PLATAFORMA COM OS RESULTADOS OBTIDOS

Com a conclusão de todas as etapas do método, é construída uma aplicação WEB utilizando a arquitetura *Restful* para utilização do modelo de forma prática. A linguagem de programação escolhida para desenvolvimento é *Python*, devido a sua simplicidade. As APIs utilizadas, por sua vez, são elaboradas com *Flask* (um *framework* de desenvolvimento Web para *Python*) que utiliza HTML para renderização das páginas e, para fins demonstrativos, rodando em um servidor local. A Figura 19 ilustra a estrutura do sistema desenvolvido.

Figura 19 - Estrutura do Sistema Desenvolvido



Fonte: Elaborado pelo autor (2020)

O modelo, já treinado e validado, é alocado então no *backend* de uma aplicação que utiliza as mesmas tecnologias utilizadas para as etapas anteriores (*Python*, *Keras* e *Tensorflow*). A aplicação, por sua vez, inicia com algum usuário interagindo com o *frontend* da mesma (que consiste em um arquivo HTML renderizado em uma página *web*). O usuário pode enviar um arquivo DICOM para ser avaliado inserindo o mesmo na página, nesse momento, a *API Restful* (construída com *Flask*) vai fazer uma ponte de comunicação do *frontend* com o *backend*, enviando a solicitação do usuário, bem como os dados do arquivo enviado por meio de um HTTP POST.

Com o arquivo, o *backend* é responsável pelo pré-processamento da imagem e pela execução do modelo de IA, que retornará um resultado (com ou sem nódulo). Para o usuário receber esse resultado, novamente a *API Restful* faz uma ponte entre o *backend* e o *frontend*, dessa vez usando um método de requisição HTTP GET, que vai renderizar um HTML contendo o resultado final. Os códigos desta aplicação estão disponíveis nos Apêndices A do trabalho.

4 APLICAÇÃO DO MÉTODO

Neste capítulo, encontram-se os resultados das aplicações do método apresentado no capítulo anterior, sendo demonstradas as saídas de cada etapa, bem como considerações e conclusões acerca das mesmas.

4.1 ENTENDENDO OS DADOS COLETADOS

O banco de dados utilizado, conhecido por LIDC-IDRI, tem sido bastante utilizado por pesquisadores para o desenvolvimento de modelos matemáticos, conforme visto no capítulo 3.2.1. Entretanto, o entendimento de como os dados foram gerados e, no caso de trabalhos de inteligência artificial, como os dados foram anotados⁴, se faz essencial para o bom desenvolvimento da pesquisa.

Nesta aplicação, dos 1018 casos incluídos nos dados originais, existem 1010 pacientes (8 pacientes tiveram TCs de diferentes datas incluídas no conjunto de dados). Para a anotação de todas as diversas imagens de cada um dos 1018 casos, foi desenvolvido um processo de duas etapas (ARMATO et al., 2011). Inicialmente 4 especialistas (radiologistas torácicos) recebiam os conjuntos de dados para categorizar, de forma anônima, cada imagem em 3 categorias:

- a) Sem Nódulo
- b) Nódulo de até 3mm
- c) Nódulo com 3 ou mais milímetros.

Após essa etapa, cada um dos radiologistas recebia as informações das avaliações de outros especialistas (ainda de forma anônima) para revisar novamente as suas anotações, dessa vez comparando com as dos outros, com as opções de manter o que já havia anotado ou fazer alguma alteração. Entretanto, apenas as imagens com nódulo eram obrigatoriamente revisadas, as demais (da categoria “sem nódulo”) não tinham essa necessidade, ficando a cargo do especialista.

⁴ A anotação de dados para ML consiste na tarefa inicial, muitas vezes realizada de forma manual, de pegar um grupo de amostras disponível e as categorizar de acordo com categorias definidas inicialmente, para em uma etapa posterior treinar e testar um modelo matemático. Tanto os dados iniciais quanto os metadados obtidos por meio das anotações podem estar em diversos formatos diferentes, como conjuntos textuais, imagens, vídeos, áudios. (MORIKAWA, 2020)

A saída da segunda etapa do processo foi convertida em um arquivo *XLS* e outro em *CSV*, como anotação oficial do banco de dados, devido ao alto uso desses dois formatos em serviços WEB. Por fim, o Quadro 1 mostra os resultados finais da etapa de anotação do banco.

Quadro 1 - Resumo das anotações do banco de dados LIDC-IDRI

| Descrição | Número de Marcações |
|--|---------------------|
| Pelo menos 1 especialista marcou a presença de algum nódulo | 7371 |
| Pelo menos 1 especialista marcou um nódulo com mais de 3mm | 2669 |
| Todos os 4 especialistas marcaram um nódulo com mais de 3mm | 928 |
| Todos os 4 especialistas marcaram a presença de algum nódulo | 1940 |

Fonte: Adaptado e traduzido pelo autor de Armatto et al., (2011).

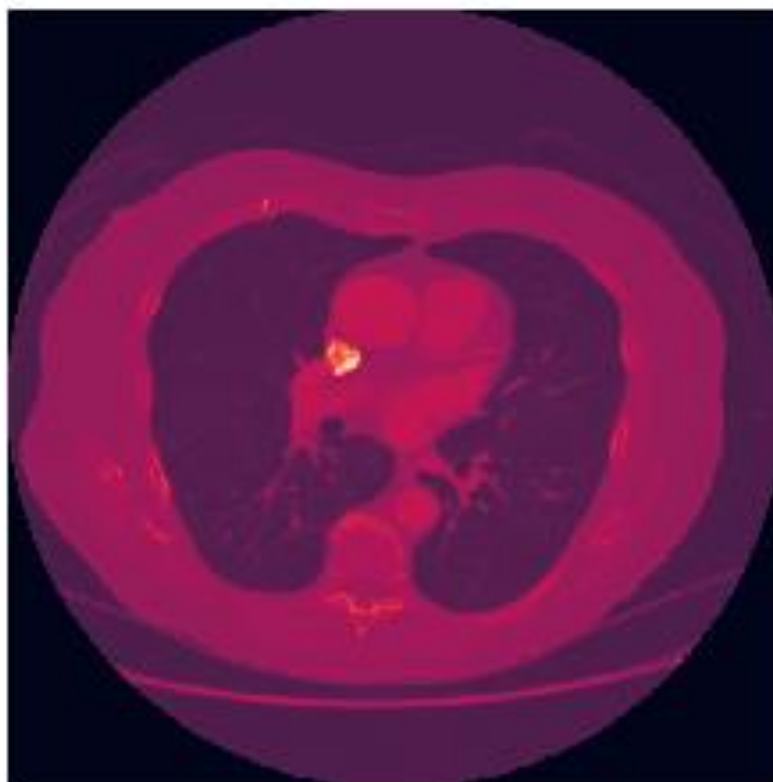
Como se pode observar, o processo de anotação apresentou inúmeras discordâncias entre os especialistas. Uma mesma imagem, de acordo com Armatto et al., (2011), foi classificada em cada uma das 3 categorias possíveis por diferentes radiologistas. Apenas 1940 imagens, que correspondem a 26,3% das 7371 que foram apontadas por pelo menos 1 responsável como presença de nódulo, tiveram o consenso de todos.

Dessa forma, para prosseguir com o pré-processamento dos dados e as demais etapas propostas para esse trabalho, inclui-se apenas imagens em que os 4 especialistas concordaram na presença de algum nódulo. Reduzindo assim o espaço amostral para 1940 imagens com nódulo.

4.2 PRÉ-PROCESSAMENTO DOS DADOS COLETADOS

Para leitura das imagens foi utilizado a biblioteca *openCV*, um repositório de código aberto e colaborativo, capaz de ler a imagem originalmente no formato DICOM e converter a mesma para o formato JPEG. A Figura 20 representa uma das imagens convertidas por esse processo (é utilizado uma paleta de cores apenas para facilitar a visualização, tendo em vista que as imagens são em escala de cinza).

Figura 20 - Imagem recém carregada no formato JPEG



Fonte: Elaborado pelo autor (2020)

Cada imagem também é interpolada para a resolução de 100x100 pixels (a resolução original é de 512 x 512). Para isso também é utilizada a biblioteca *openCV*, e, no mesmo laço de repetição onde cada imagem é pré-processada, já se constrói o vetor binário que serve para identificação das imagens que contém ou não o nódulo. Utiliza-se, por definição arbitrária, o valor 0 para casos sem nódulo e o valor 1 para casos com nódulo.

A sequência de passos dessa etapa é descrita pelo pseudo-algoritmo na Figura 21.

Figura 21 - Pseudo-algoritmo que descreve a etapa de pré-processamento

```

Início
casos ← carrega planilha de avaliação dos especialistas
lista_de_imagens = []
para cada i paciente faça:
    para cada j imagem de i faça:
        img ← carrega dicom
        info ← carrega metadados de img
        id ← i + info[slice]
        img ← converte img para matrix numérica
        img ← interpola img para dimensões 100x100
        se id estão em casos com nódulos:
            vetor_final ← [ img, 1]
        senão:
            vetor_final ← [img, 0]
        adiciona vetor_final em lista_de_imagens
    fim para
fim para
retorna lista_de_imagens

```

Fonte: Elaborado pelo autor (2020)

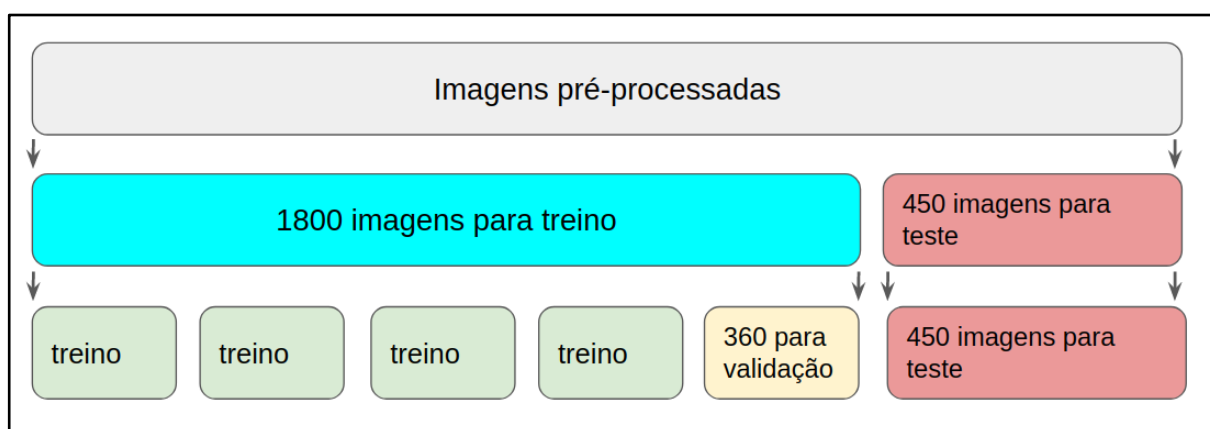
4.3 SEPARAÇÃO EM TREINO E TESTE

Com a lista de imagens já preparada (dados pré-processados e indicação da presença ou não de nódulos), chega-se à etapa de separação dos dados de teste e dos dados de treino. Para concluir essa etapa, implementou-se o método disponível na biblioteca do *SciKit* (<https://github.com/scikit-learn/>), que usa uma semente para gerar valores pseudoaleatórios.

Entretanto, antes da separação, utilizou-se da linguagem de programação Python para “sortear” um número menor de imagens que não apresentaram nódulo, devido essa categoria ser muito maior do que a categoria com nódulos, em uma proporção de aproximadamente 1000 para 1, pois, conforme citado no capítulo 3, existe o risco de o modelo acabar tendencioso para a classe majoritária em bancos de dado desbalanceados. Além disso, um número menor de imagens diminui o tempo de treinamento dos modelos.

O tamanho de amostra final para cada conjunto ainda é o estipulado no capítulo 3, consistente de 1800 imagens para treino e 450 para teste, ou seja, 80% dos dados treinam o modelo enquanto 20% dos dados testam os resultados. Outro ponto é que das 1800 amostras da etapa de treino, 360 delas são separadas a cada iteração para o processo de *cross-validation*. A Figura 22 ilustra essa etapa de separação de dados:

Figura 22 - Separação dos dados em treino e teste



Fonte: Elaborado pelo autor (2020)

Em linhas gerais, as imagens coletadas e pré-processadas são separadas em dois conjuntos, 1800 para treino e 450 para teste, e, do conjunto de treino, usando o método de *cross-validation* separa em 5 subconjuntos em que 1 deles (com 360 imagens) é utilizado para validação (e acompanhamento dos resultados) durante as épocas de treinamento do modelo.

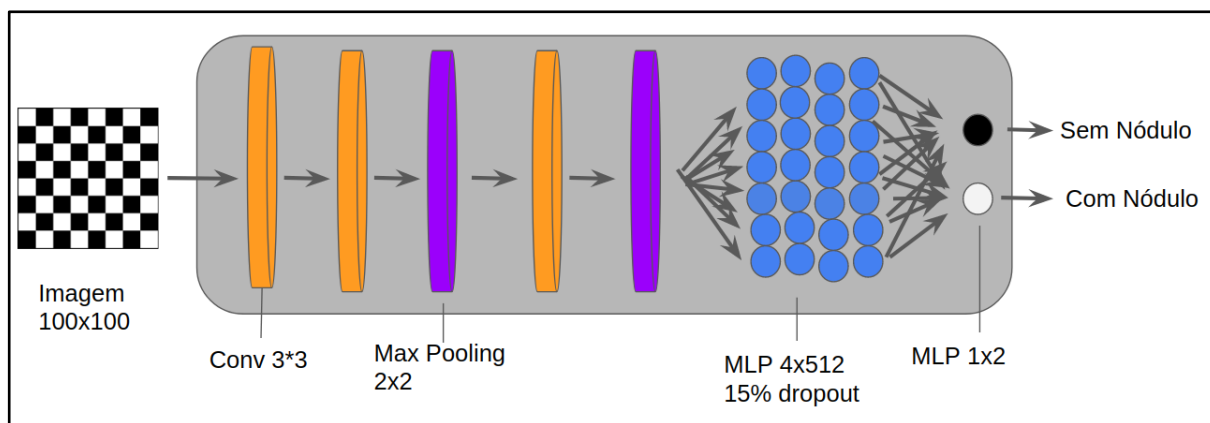
4.4 CRIAÇÃO DA CNN

Conforme citado no capítulo 3.2.3, utilizou-se as bibliotecas *TensorFlow* e *Keras* para desenvolver a rede a ser treinada, o modelo inicial construído foi baseado em experiências práticas do autor do projeto em aplicações similares, levando em consideração o tamanho do banco de dados disponível e a característica da tarefa de classificação de imagens em escala de cinza entre duas categorias possíveis. O modelo é constituído, em ordem, pelas seguintes camadas (representadas pela figura 23):

- a) Convolução 3x3;
- b) Convolução 3x3;
- c) MaxPooling 2x2;

- d) Convolução 3x3 ;
- e) MaxPooling 2x2;
- f) 4 Camada de 512 Neurônios em sequência, com *dropout* de 15% em cada;
- g) 1 Camada final de 2 Neurônios.

Figura 23 - Modelo de CNN inicial construído



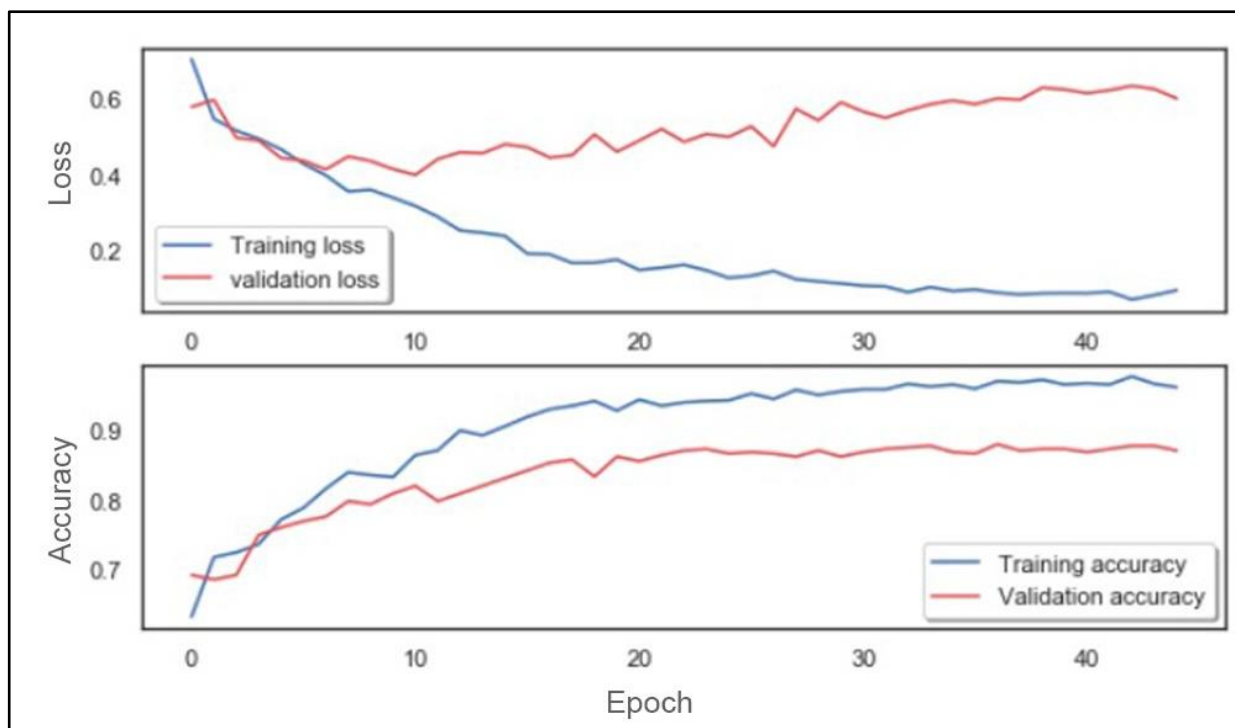
Fonte: Elaborado pelo autor (2020)

Na Figura acima, as etapas representadas pela cor laranja correspondem às convoluções 3x3, as “etapas roxas” ao MaxPooling 2x2. Os MLPs (as circunferências) são as redes neurais clássicas existentes no final das CNNs (MLP é de *Multilayer perceptron*).

Outras informações sobre o modelo construído: taxa de aprendizado adaptativa, iniciando em 0.001 e terminando em 0.00001, função de ativação retilínea uniforme, 50 épocas de treinamento, 5 “*folds*” para *cross-validation* durante o treinamento, *Cross Entropy Loss* como função de perda.

A Figura 24 representa a média da função de perda e da acuracidade do modelo (eixo y de cada gráfico) durante cada uma das épocas de treinamento (eixo x de cada gráfico). Esse gráfico é gerado automaticamente pela biblioteca utilizada, *Keras*, e a função de perda, que corresponde ao eixo y do gráfico superior é representada pela palavra “*loss*”, que quanto menor, melhor.

Figura 24 - Evolução do modelo durante o treinamento



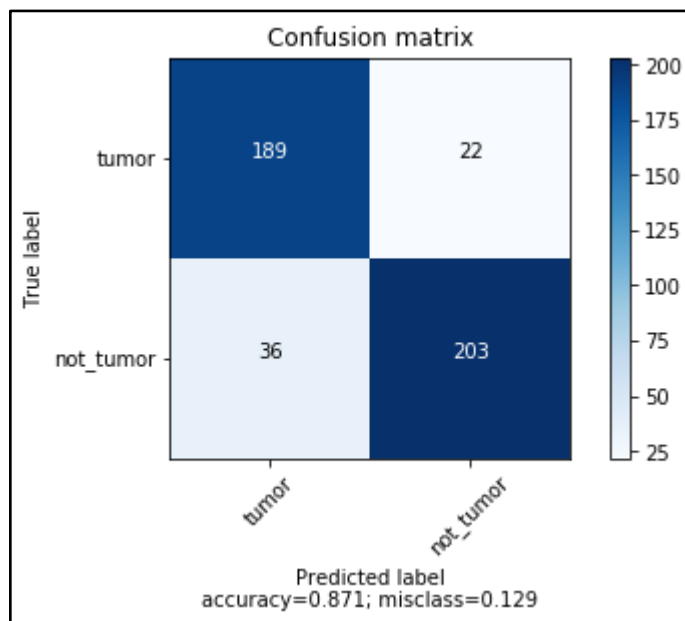
Fonte: Elaborado pelo autor (2020)

Observa-se que o modelo convergiu rapidamente, pois logo pela época 20 os resultados de acuracidade já estavam praticamente estabilizados. Entretanto, existe uma diferença acentuada entre a linha de treinamento (azul) e a linha de validação (vermelha), o que indica um certo nível de *overfitting* dos dados. Um agravante, além dessa destoante diferença visual entre a validação e o treinamento, é que a função de perda da validação (a linha vermelha na imagem superior) depois de cair durante as 10 primeiras épocas, começou a subir, o que indica fortemente a presença de *overfitting* após a décima geração do treinamento. Com isso, os dados reservados para teste são utilizados para avaliar de forma prática como o modelo se comporta.

4.4 TESTE DO MODELO

As 450 imagens, inicialmente separadas para teste, são processadas pelo modelo treinado, e os resultados de predição são comparados com as anotações realizadas pelos especialistas (consideradas como valores verdadeiros). A relação entre os valores reais e os valores preditos para o modelo inicial é exibida pela matriz de confusão representada pela Figura 25.

Figura 25 – Matriz de Confusão do Modelo Inicial



Fonte: Elaborado pelo autor (2020)

Observando a matriz de confusão, percebe-se que o número de TP e TN foi de 189 e 203, respectivamente. O modelo ainda apresentou para esse conjunto de teste 36 FP e 22 FN, o que resulta em uma acuracidade de 87,11%, superior que a meta definida para esse parâmetro de 85%. Entretanto, só a acuracidade não nos traz informações o suficiente, dessa forma calcula-se as demais métricas de classificação:

- a) *Recall*: 89,57%
- b) *Precisão*: 84,00%
- c) *F1-Score*: 86,69%

O valor do *recall* ficou menor que o da *precisão*, o que é algo esperado para o problema proposto (probabilidade de FN < que FP). O *f1-score* também ficou acima de 85%, o que é bom para o modelo, contudo, a métrica principal, de *recall*, ficou inferior à meta estipulada inicialmente de 90%. Com esse resultado, passa-se para a etapa de ajustar os hiperparâmetros, que são, no caso do modelo usado, a configuração da rede (número de camadas, número de neurônios, função de ativação, convoluções, *poolings*, etc) e realizar novos testes.

4.5 AJUSTE DE HIPERPARÂMETROS

Conforme citado no capítulo 3.2.3, foi definido a utilização do método de otimização de hiperparâmetros de busca aleatório. Para a execução do mesmo é necessário, primeiramente, definir um espaço de busca, que pode ser observado no Quadro 2.

Quadro 2 – Espaço de busca

| Hiperparâmetro | Espaço de Busca | Número de Possibilidades |
|---|---------------------------|--------------------------|
| Camadas Convolucionais | 2 até 6 | 5 |
| Kernel de Convolução | 3x3 | 1 (fixo) |
| Camadas de <i>MaxPooling</i> | 1 ou 2 | 2 |
| Dimensões de <i>MaxPooling</i> | 2x2 | 1 (fixo) |
| Camadas de RNA ocultas | 3 até 7 | 5 |
| Neurônios em cada camada | 216 até 1024 | 809 para cada camada |
| <i>Dropout</i> | 0,15 | 1 (fixo) |
| <i>Learning Rate</i> | Adaptativo | 1 (fixo) |
| Função de Ativação | ReLu | 1 (fixo) |
| <i>Folds</i> de <i>Cross-Validation</i> | 5 | 1 (fixo) |
| Função de Perda (<i>Loss</i>) | <i>Cross Entropy Loss</i> | 1 (fixo) |
| Épocas de treinamento | 50 | 1 (fixo) |

Fonte: Elaborado pelo autor (2020)

Devido ao modelo inicial já ter alcançado uma performance bem próxima ao aceitável, optou-se por alterar apenas 4 dos hiperparâmetros dentro do espaço de busca, com os demais fixos. A partir disso, iniciou-se a busca aleatória com as seguintes condições de parada: encontrar 3 modelos cujo resultado apresenta valores com o *recall* acima de 90% ou após 15 dias de execução.

O critério a ser acionado primeiro foi a segunda opção, ou seja, três modelos (a partir daqui chamados arbitrariamente de B, C e D, considerando como A o primeiro construído na etapa 4.6) apresentaram um *recall* superior a 90% em aproximadamente 11 dias de execução.

A configuração de hiperparâmetros de cada um dos modelos (incluindo novamente a do primeiro treinado) pode ser observado no Quadro 3.

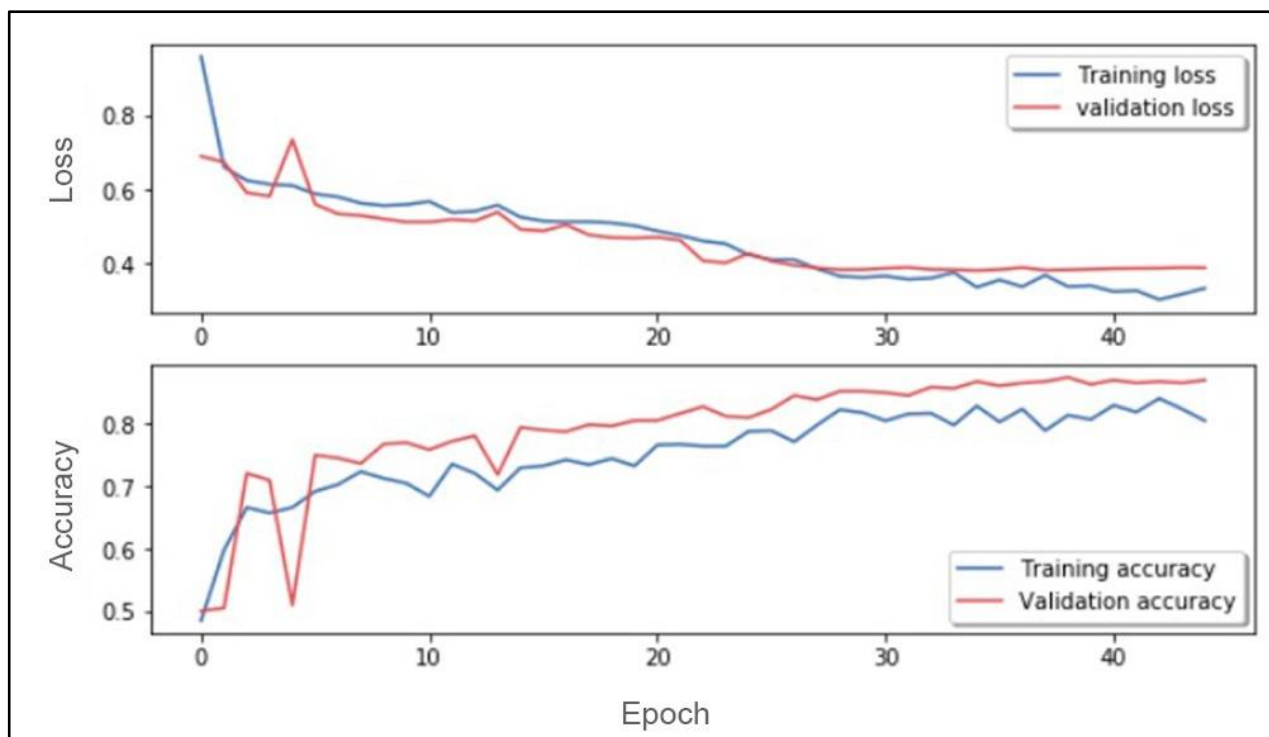
Quadro 3 – Hiperparâmetros de cada modelo

| Hiperparâmetro | Modelo A | Modelo B | Modelo C | Modelo D |
|----------------------------------|---------------------------|---------------------------|---------------------------|---------------------------|
| Camadas Convolucionais | 3 | 2 | 4 | 4 |
| Kernel de Convolução | 3x3 | 3x3 | 3x3 | 3x3 |
| Camadas de <i>MaxPooling</i> | 2 | 1 | 2 | 1 |
| Dimensões de <i>MaxPooling</i> | 2x2 | 2x2 | 2x2 | 2x2 |
| Camadas de RNA ocultas | 4 | 3 | 3 | 5 |
| Neurônios em cada camada | 512 | 311 | 677 | 497 |
| <i>Dropout</i> | 0,15 | 0,15 | 0,15 | 0,15 |
| <i>Learning Rate</i> | Adaptativo | Adaptativo | Adaptativo | Adaptativo |
| Função de Ativação | ReLu | ReLu | ReLu | ReLu |
| <i>Folds de Cross Validation</i> | 5 | 5 | 5 | 5 |
| Função de Perda (<i>Loss</i>) | <i>Cross Entropy Loss</i> | <i>Cross Entropy Loss</i> | <i>Cross Entropy Loss</i> | <i>Cross Entropy Loss</i> |
| Épocas de treinamento | 50 | 50 | 50 | 50 |

Fonte: Elaborado pelo autor (2020)

A evolução de cada um dos modelos durante os seus respectivos treinamentos pode ser observada nas Figuras 26, 27 e 28.

Figura 26 – Evolução do modelo B durante o treinamento

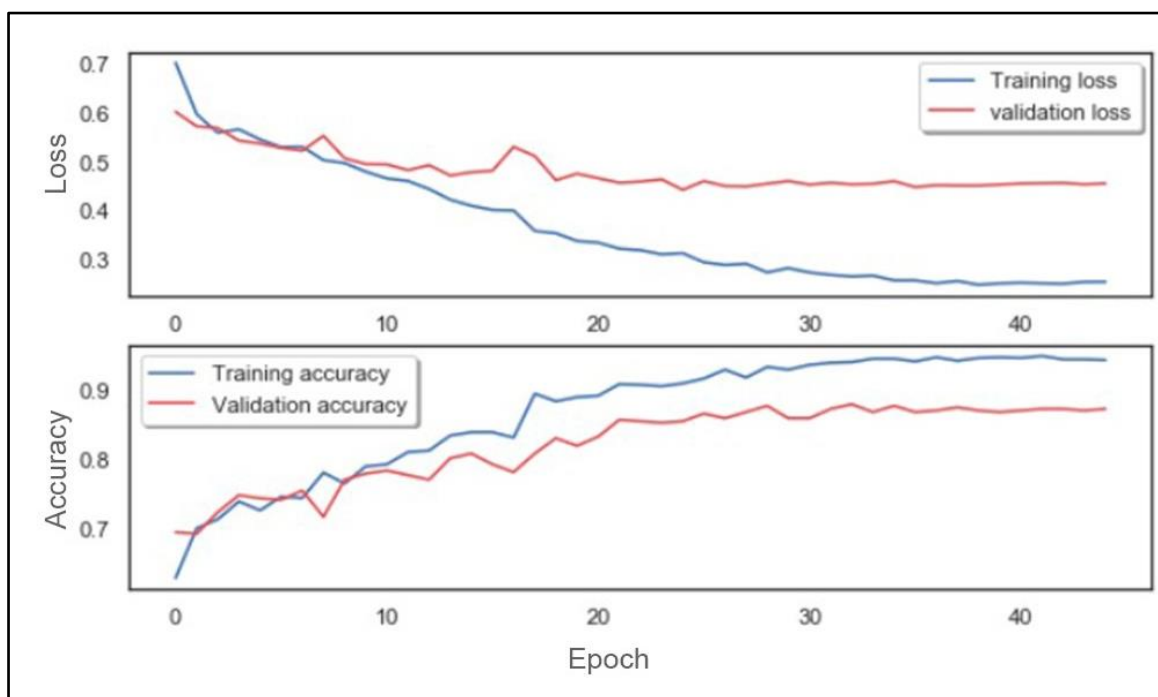


Fonte: Elaborado pelo autor (2020)

Este modelo, ao contrário do primeiro visualizado, não apresentou uma diferença considerável entre o conjunto de validação e o restante dos dados de treino (representado visualmente no gráfico pelo *gap* entre as linhas azuis e vermelhas), o que diminui a chance de um *overfitting*. Entretanto, avaliando as últimas épocas de treinamento, percebe-se que os resultados ainda não estão estáveis, o que pode indicar que ainda há espaço para melhoria do modelo, caso treinado por mais épocas.

Outro ponto que chama a atenção nesse modelo é que o começo dele apresentou um comportamento não homogêneo. Evidenciou-se que as primeiras 5 épocas apresentam desvios onde os valores da função de perda subiam e desciam de forma acentuada entre uma geração e outra, comportamento que se manteve até as últimas épocas, onde se pode notar um estilo de gráfico de serra. Essa característica pode ser um sinal de *underfitting*, e, avaliando junto ao Quadro 2, dos hiperparâmetros desse modelo, percebe-se que ele é o com menos camadas dentre os 4, o que também corrobora com o comportamento encontrado durante o treinamento dessa rede.

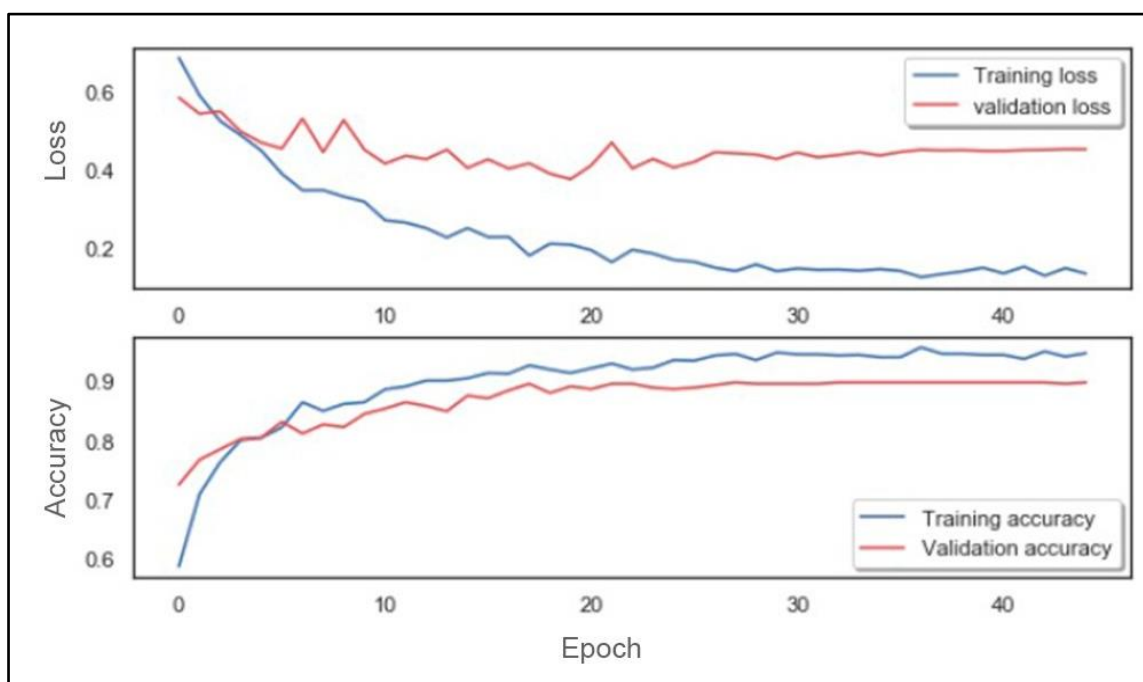
Figura 27 – Evolução do modelo C durante o treinamento



Fonte: Elaborado pelo autor (2020)

Desta vez, os resultados da *cross-validation* indicam um comportamento mais uniforme ao longo das épocas e uma estabilização dos resultados até o fim das 50 épocas de treinamento.

Figura 28 – Evolução do modelo D durante o treinamento



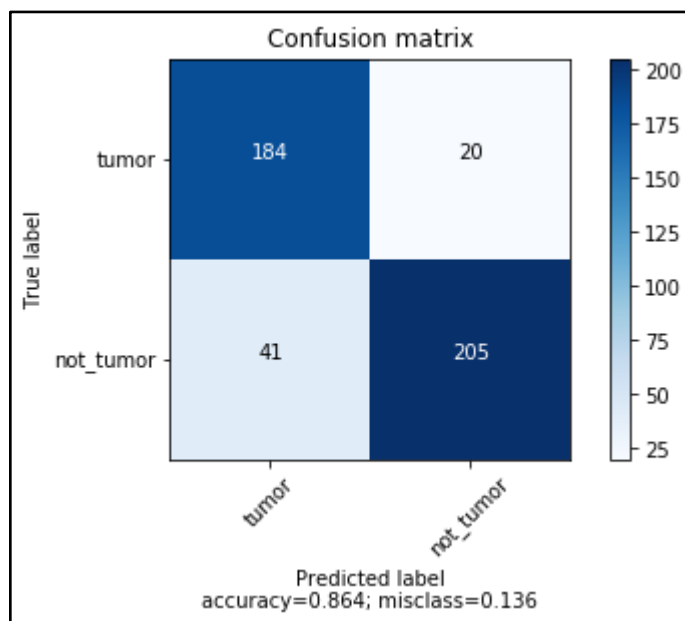
Fonte: Elaborado pelo autor (2020)

Por fim, o modelo D também teve um comportamento parecido com o anterior, sem apresentar sinais evidentes de *overfitting* e *underfitting*, além de apresentar um certo nível de estabilização durante as últimas épocas de treinamento. Com os 3 novos modelos construídos, e treinados, entra-se em uma nova avaliação, na qual se compara as métricas de *recall*, precisão, acuracidade e *f1-score*, bem como as matrizes de confusão de cada modelo em uma etapa de teste, que corresponde ao resultado final de cada um.

4.7 TESTE DOS MODELOS COM OS AJUSTES DE HIPERPARÂMETROS

As mesmas 450 imagens, separadas para teste, são então processadas por cada um dos novos modelos treinados. Os resultados são primeiramente expressos nas Figuras 29, 30 e 31, que representam cada uma das respectivas matrizes de confusão dos Modelos B, C e D.

Figura 29 – Matriz de confusão do modelo B na etapa de teste



Fonte: Elaborado pelo autor (2020)

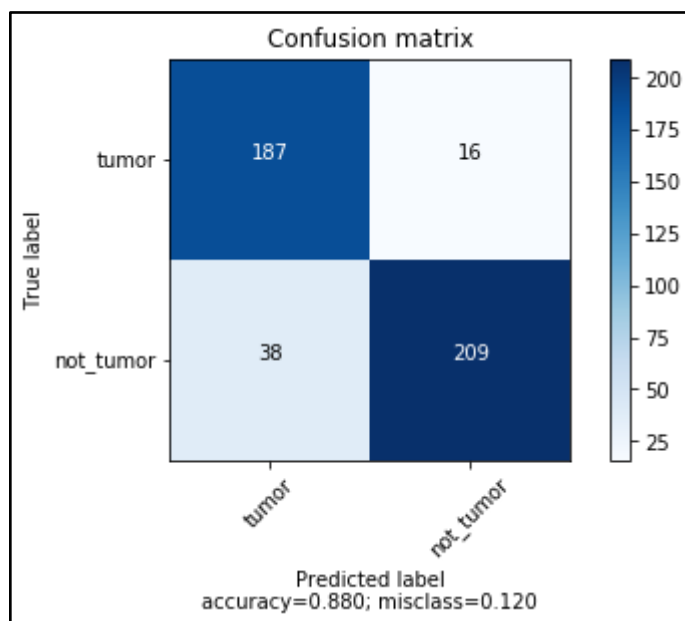
Dada a matriz de confusão, nota-se que o número de TP e TN foi de 184 e 205, respectivamente, ao passo que as predições erradas são compostas por 42 FP e 20 FN, o que resulta em uma acuracidade de 86,44%, superior que a meta definida, porém inferior ao modelo inicial, reprovado no teste por causa de seu *recall* de 87,11%. Entretanto, o número de FN

dessa vez foi menor, o que garantiu a esse modelo um *recall* superior ao inicial. Junto com as outras métricas extraídas da matriz de confusão, têm-se:

- a) *Recall*: 90,19%
- b) *Precisão*: 81,77%
- c) *F1-Score*: 85,78%

A precisão do modelo teve um desempenho pobre, resultado do elevado número de FP (41), apesar disso, o modelo B atende às especificações propostas, de mais de 85% de acuracidade e mais de 90% de *recall*.

Figura 30 – Matriz de confusão do modelo C na etapa de teste



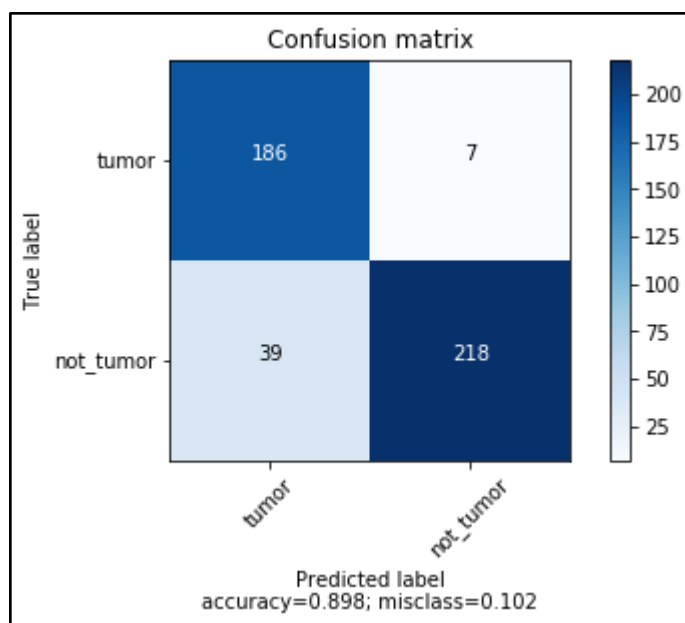
Fonte: Elaborado pelo autor (2020)

No modelo C, encontrou-se 187 para TP e 209 para TN, 38 para FP e 16 para FN, o que resulta em uma acuracidade de 88,00%, superior que os dois primeiros. Quanto as demais métricas:

- a) *Recall*: 92,11%
- b) *Precisão*: 83,11%
- c) *F1-Score*: 87,38%

Esse modelo também atende a todas as especificações propostas inicialmente, com desempenhos satisfatórios em todos os parâmetros.

Figura 31 – Matriz de confusão do modelo D na etapa de teste



Fonte: Elaborado pelo autor (2020)

No último modelo, o que se destaca diretamente olhando para a matriz de confusão é o seu baixo número de FN, com apenas 7 casos, ao mesmo tempo que mantém um número equilibrado de FP (39, apenas 1 a mais que o modelo C e 3 a mais que o modelo A). Sua Acuracidade atinge o valor de 89,77%, o restante das métricas:

- a) *Recall*: 96,37%
- b) *Precisão*: 82,66%
- c) *F1-Score*: 88,99%

Esse modelo, assim como o modelo C, também atende a todas as especificações propostas inicialmente, com desempenhos satisfatórios em todos os parâmetros.

4.8 ANÁLISE DOS RESULTADOS OBTIDOS

Para facilitar a avaliação geral dos resultados encontrados em cada modelo construído, o Quadro 4 apresenta um resumo das 4 métricas destacadas, tanto para os 3 modelos obtidos

pelo processo de otimização dos hiperparâmetros, quanto para o modelo inicialmente proposto, no capítulo 4.3.

Quadro 4 - Resumo dos resultados

| Métrica | Modelo A | Modelo B | Modelo C | Modelo D |
|-------------|----------|----------|----------|----------|
| Acuracidade | 87,11 | 86,44 | 88,00 | 89,77 |
| Recall | 89,57 | 90,19 | 92,11 | 96,37 |
| Precisão | 84,00 | 81,77 | 83,11 | 82,66 |
| F1-Score | 86,69 | 85,78 | 87,38 | 88,99 |

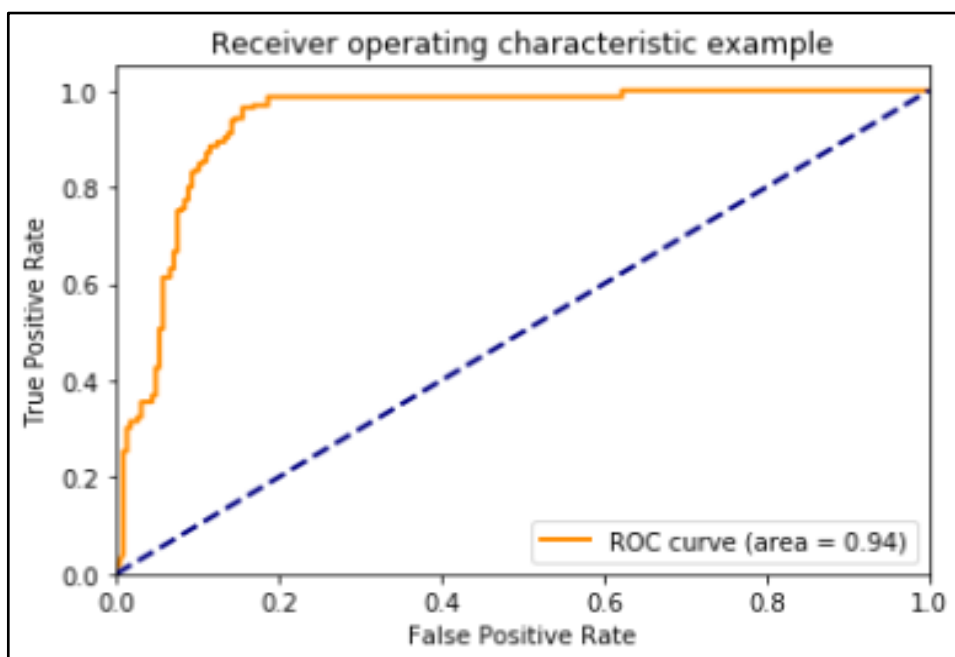
Fonte: Elaborado pelo autor (2020)

Em relação a acuracidade, o modelo D teve o melhor desempenho, seguido de perto pelo modelo C e depois pelos modelos A e B, respectivamente. Nessa métrica todos tiveram um resultado bem similar, acima da casa dos 85%, o que demonstra que a arquitetura proposta tem um bom desempenho de forma geral na tarefa.

A métrica considerada mais importante no sistema, o *recall*, também indica o último modelo como a melhor opção, com um resultado expressivamente superior aos anteriores, bem diferente do resultado de precisão, onde o modelo D teve apenas o terceiro melhor desempenho. É interessante notar que, nesse caso, o modelo A teve a melhor performance, isso pode ser explicado pelo fato que, durante a etapa de otimização dos hiperparâmetros, foi apenas considerado modelos com mais de 90% em seu *recall*. Desta forma, é provável que outros modelos com diferentes conjuntos de hiperparâmetros teriam um resultado maior de precisão em detrimento ao *recall*, porém estes foram descartados durante a otimização, por esse não ser o foco dos resultados desejados.

Por fim, o *f1-score* também aponta para o modelo D como sendo o mais indicado para a aplicação proposta, considerando os dados disponíveis, seguido pelo Modelo C, A e B, respectivamente. Dessa forma, o modelo final, a ser utilizado na etapa seguinte, de construção de uma aplicação WEB para identificar possíveis imagens de tomografia com a presença de nódulo, foi o modelo D. A curva ROC do modelo final escolhido é ilustrada na Figura 32.

Figura 32 - Curva ROC do modelo escolhido



Fonte: Elaborado pelo autor (2020)

Nota-se que a curva ROC se aproxima consideravelmente do canto superior esquerdo, o que indica uma boa performance do modelo em geral, com a área sob a curva equivalente a 94%. Para comparação, um classificador aleatório, que acerta metade dos casos, teria 50% de área sob a curva, sendo representado pela linha pontilhada no gráfico.

Outro ponto a ser analisado é a velocidade de processamento obtida, fator que impacta em cenários práticos, podendo validar (ou não) a aplicação de um modelo de IA em diferentes processos, devido ao fato desse recurso (o tempo) ser finito e, em geral, escasso, no mundo corporativo. A medida de tempo em um sistema de ML é dividida em dois principais blocos: o tempo para construção/treinamento e o tempo para predição.

O tempo para predição, após a criação do modelo, é o que vai influenciar nas novas predições a serem realizadas. Ele se dá pelo tempo em que o modelo faz os cálculos necessários e retorna o seu resultado e, em alguns casos, algum pré-processamento dos dados de entrada que seja necessário (processos como normalização, conversão, engenharia de dados do modelo, etc). No caso do modelo construído neste trabalho, o tempo de predição se dá pelo tempo de converter uma imagem DICOM para JPG, uma interpolação da mesma para a resolução 100x100 e o tempo que a CNN leva para calcular, desde a entrada da matriz da imagem, passando por cada uma de suas camadas, até a saída final (0 para imagens que ela não encontra um nódulo e 1 para as imagens que encontra).

O tempo médio para a etapa de predição, levantado utilizando uma sequência de 450 imagens, foi de 24 ms por imagem, com desvio padrão de 4 ms. Esse tempo foi obtido em um computador com processador Intel Core I7, 16gb de RAM com o sistema operacional Ubuntu 20.04, conforme citado no capítulo 3. Considerou-se esse tempo satisfatório, tendo em vista que a forma utilizada para a construção do banco de dados utilizado (inspeção visual por especialistas), apesar de não ter seu tempo citado na bibliografia, demoraria significativamente mais do que o algoritmo construído.

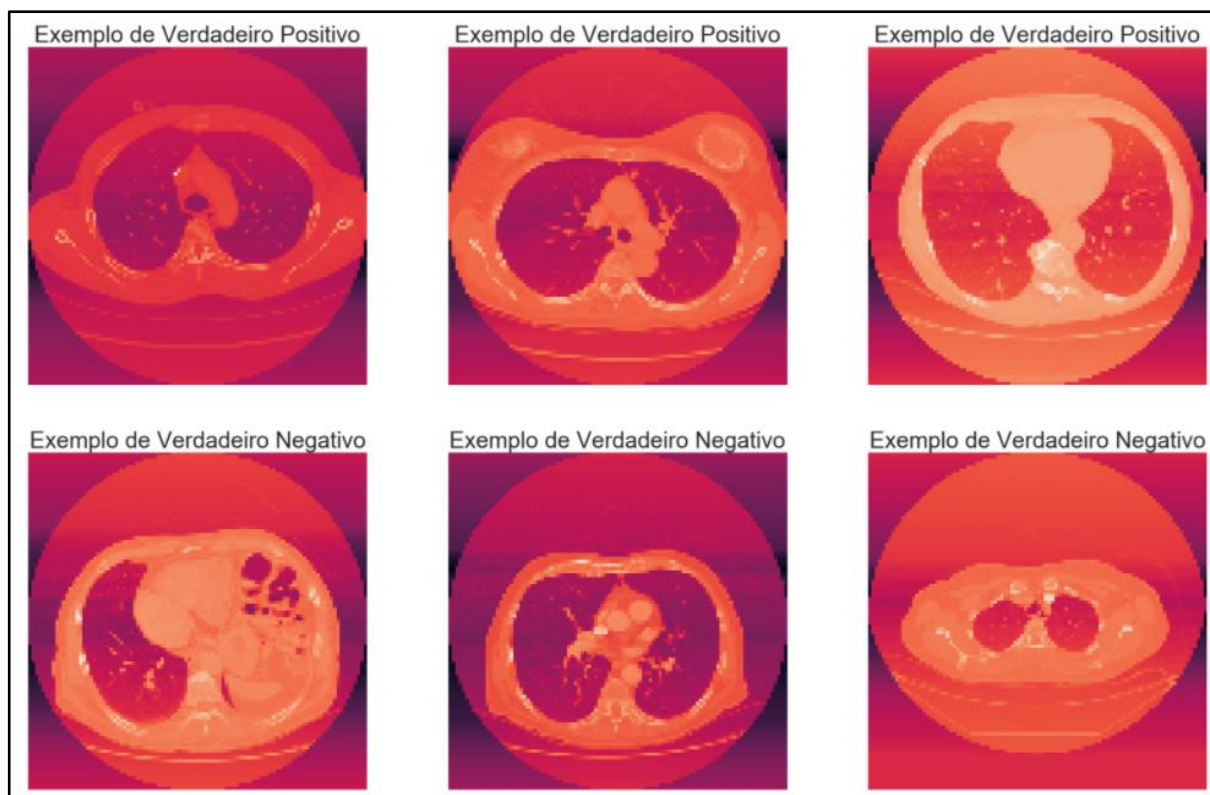
Quanto ao tempo de construção/treinamento, existem uma série de etapas que envolvem o mesmo, desde a definição inicial do problema de pesquisa, a aquisição e limpeza dos dados, que, em geral, é a etapa mais demorada de um trabalho de ML, passando pelo tempo para a criação do modelo pelos engenheiros responsáveis (na maioria das vezes, essa é a etapa mais rápida). Por fim, o tempo de treinamento do modelo propriamente dito, ou seja, o tempo que os dados coletados e pré-processados levam para treinar cada um dos parâmetros da rede construída.

Esse tempo também é importante em uma aplicação prática, pois, em mercados dinâmicos e rápidos, os produtos mudam com uma frequência elevada, e, caso o modelo leve muito tempo para ser construído, corre o risco de que, logo após seu lançamento, se torne obsoleto. Em mercados mais estáveis ou em cenários que o produto/serviço não corre o risco de sofrer mudanças com tanta velocidade, esse tempo costuma ser menos relevante.

Neste trabalho, o tempo para a construção da rede não será levado em conta, juntamente ao tempo de coleta e pré-processamento dos dados, tendo em vista que ambas as etapas foram construídas ao longo do período do mestrado do autor da pesquisa (2 anos) e não representam um cenário real prático de aplicação. Dessa forma, o único tempo considerado foi o de treinamento do modelo, que levou aproximadamente 11 dias (considerando a otimização do modelo). Também se considerou um tempo aceitável para o problema proposto, considerando que o desafio de avaliar imagens de TC não passa por mudanças tão rápidas quanto esse período de tempo.

Por fim, a Figura 33 mostra exemplos de imagens que o modelo escolhido previu corretamente na etapa de teste.

Figura 33 - Exemplos de dados que o modelo previu corretamente

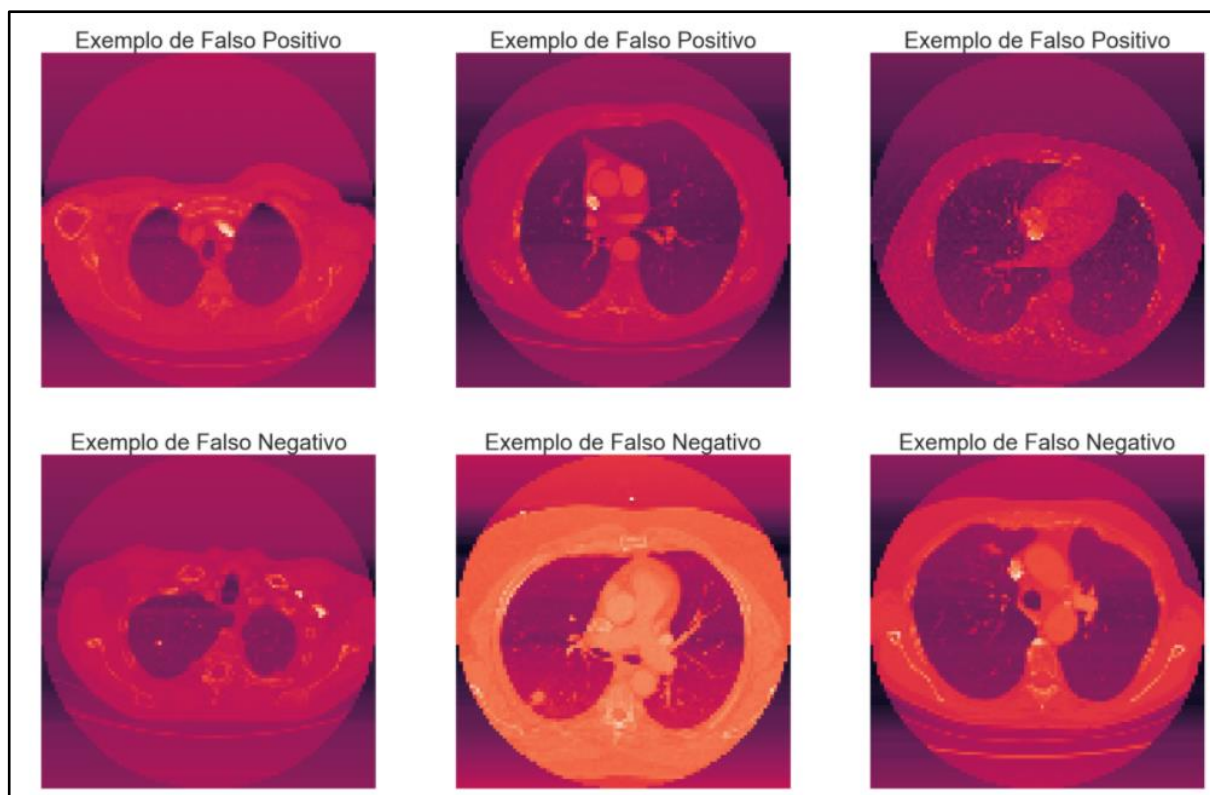


Fonte: Elaborado pelo autor (2020)

As três imagens da fileira superior correspondem a TPs, ou seja, imagens que apresentavam nódulo e o modelo corretamente apontou isso, enquanto as três imagens da fileira inferior apresentaram TNs, imagens que não apresentavam nódulos e o modelo também apontou corretamente essa característica. Essas imagens, podem ser visualizadas em um tamanho maior nos Apêndices B deste trabalho.

A Figura 34, por sua vez, mostra exemplos de imagens que o modelo não acertou em suas respostas. Tanto estas imagens quanto as anteriores, da Figura 31 são escolhidas aleatoriamente do conjunto de 450 imagens utilizadas para teste, cujo resultado foi ilustrado pela matriz de confusão da figura 31.

Figura 34 - Exemplos de dados que o modelo previu erroneamente



Fonte: Elaborado pelo autor (2020)

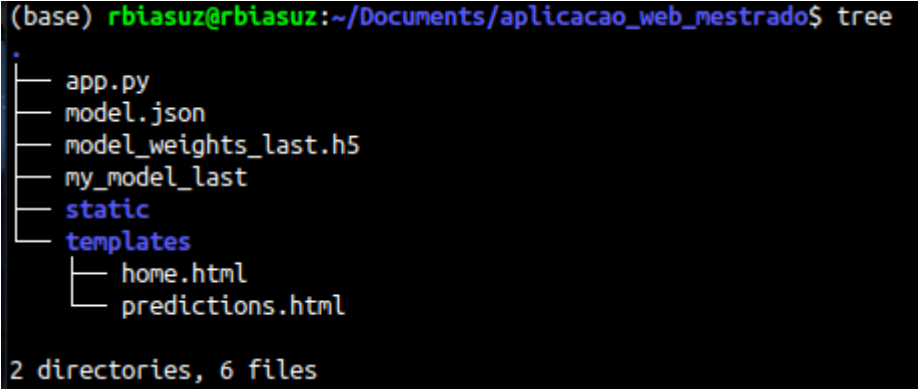
As três imagens da fileira superior correspondem a FPs, ou seja, imagens que não apresentavam nódulo, mas o modelo apontou como se existisse um nódulo nas mesmas, enquanto as três imagens da fileira inferior apresentaram FNs, imagens que na verdade apresentavam nódulos e o modelo também falhou em mostrar isso.

É interessante notar que o modelo apresentou resultados corretos (e incorretos) nas diferentes alturas possíveis das imagens de tomografia (as diferentes posições das “fatias” da TC em relação ao corpo do paciente). Além disso, imagens bem diferentes umas das outras tiveram resultados corretos, bem como errados, o que mostra a complexidade do sistema de IA criado, principalmente no que diz respeito a interpretabilidade. Um dos pontos com maior questionamento das implementações de ML é a característica “*black box*” dos resultados gerados, pois, apesar de ser possível entender cada camada de um modelo individualmente, bem como a lógica por trás de cada uma dessas camadas, é humanamente impossível entender a interação entre todas essas camadas para explicar de forma direta o motivo que leva uma imagem a ter determinada classificação dentro de um modelo.

4.9 CONSTRUÇÃO DA APLICAÇÃO WEB

Com o modelo final salvo e seus resultados avaliados, criou-se uma aplicação web utilizando o framework *Flask*, de acordo com o capítulo 3.3, com a finalidade de simular um possível cenário de uso do *software* em situações reais. Utilizou-se um sistema operacional livre (Ubuntu 20.04) e a estrutura do diretório da aplicação pode ser observada na Figura 35.

Figura 35 - Estrutura de diretórios da aplicação



```
(base) rbiasuz@rbiasuz:~/Documents/aplicacao_web_mestrado$ tree
.
├── app.py
├── model.json
├── model_weights_last.h5
├── my_model_last
├── static
└── templates
    ├── home.html
    └── predictions.html

2 directories, 6 files
```

Fonte: Elaborado pelo autor (2020)

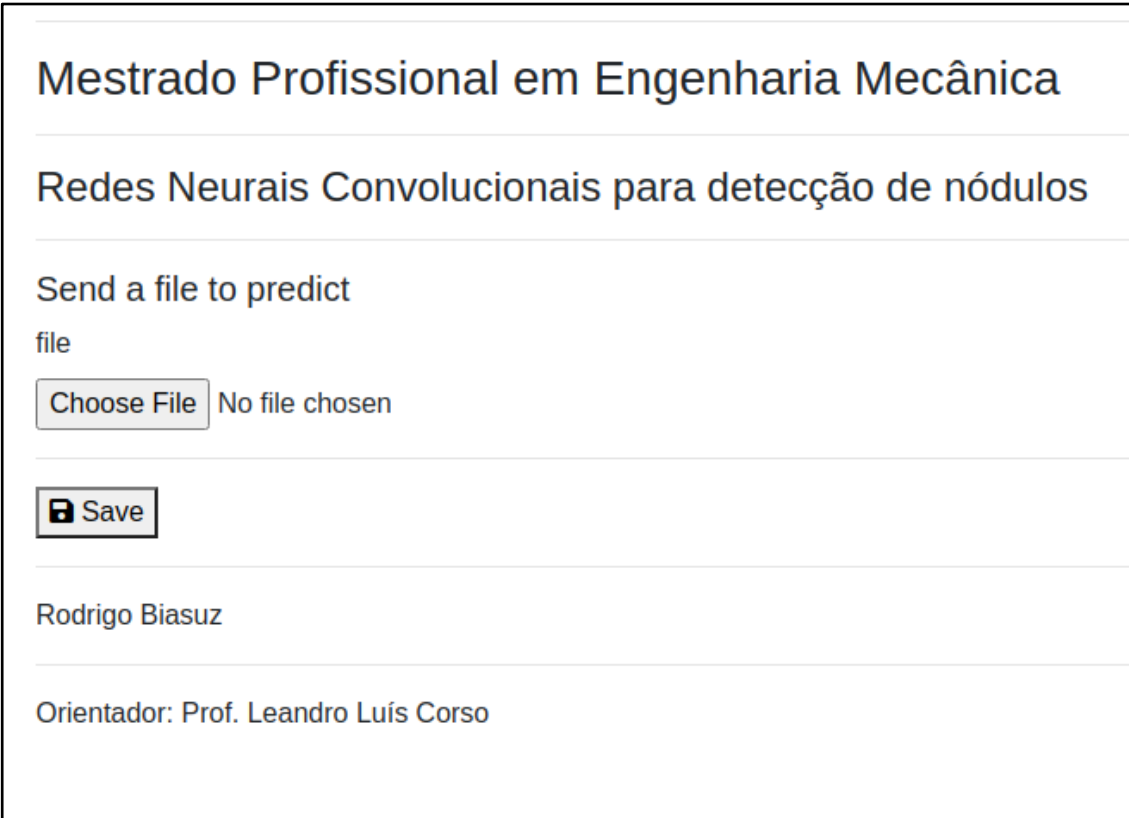
Conforme é possível se observar, a partir de um diretório base (no caso, a pasta “Documents/aplicação_web_mestrado” do computador), existem 2 pastas e inicialmente 4 arquivos. O primeiro arquivo, “app.py”, é o código python que funciona como *backend* (espécie de coração do sistema, onde as funções são executadas e os modelos fazem seus cálculos) da aplicação.

Após isso, existem 3 arquivos com “*model*” no nome, todos são referentes ao modelo de ML utilizado (o Modelo D, que teve a melhor performance). Esses arquivos correspondem à estrutura da rede (hiperparâmetros) e aos pesos já treinados de cada parâmetro da mesma. Basicamente, esses arquivos são responsáveis por carregar os dados da rede na memória de forma rápida, sem precisar realizar novamente o treinamento ou a criação do modelo.

Por fim, temos as duas pastas (em azul), a “*static*”, uma pasta vazia que é onde os arquivos estáticos ficam salvos temporariamente enquanto o sistema estiver sendo executado (no caso, as imagens DICOM carregadas para o servidor) e a pasta “*templates*”, onde existem dois arquivos em HTML que são o *frontend* da aplicação (*frontend* é a interface gráfica que algum usuário do sistema vai enxergar enquanto utiliza o mesmo, como se fosse a “casca” do *backend*).

Os códigos desenvolvidos dessas aplicações podem ser encontrados nos Apêndices C deste trabalho. As Figuras 36, 37 e 38 mostram o que um usuário enxerga nas 3 telas principais do sistema: a página de entrada, uma predição sem nódulo e uma predição com nódulo. A página de entrada consiste em uma tela onde o usuário vai enviar a imagem para ser analisada, após isso, uma das outras duas será carregada, de acordo com o resultado da análise.

Figura 36 - Página de entrada



Mestrado Profissional em Engenharia Mecânica

Redes Neurais Convolucionais para detecção de nódulos

Send a file to predict

file

Choose File No file chosen

Save

Rodrigo Biasuz

Orientador: Prof. Leandro Luís Corso

Fonte: Elaborado pelo autor (2020)

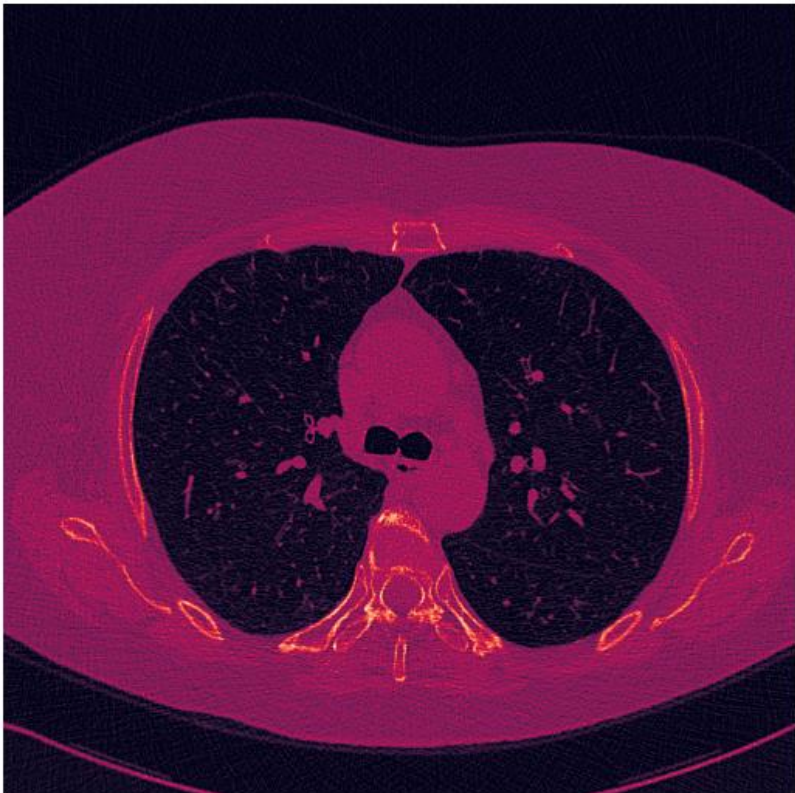
Nesta primeira tela, o usuário simplesmente tem uma caixa para selecionar “*Choose File*” (selecione o arquivo, em português), que ao ser clicada abre uma janela onde a pessoa poderá procurar pela imagem que quer avaliar em seu computador ou pode simplesmente arrastar o arquivo para dentro da janela. Após selecionar ou arrastar o arquivo, o usuário pode clicar em “*Save*”, e o modelo irá então fazer a predição sobre a presença ou não de nódulos.

Figura 37 - Predição sem nódulo

Mestrado Profissional em Engenharia Mecânica

Redes Neurais Convolucionais para detecção de nódulos

Imagem Processada:



Resultado: Não apresenta Nódulo (96,37% de confiabilidade)

Rodrigo Biasuz

Orientador: Prof. Leandro Luís Corso

Fonte: Elaborado pelo autor (2020)

Neste exemplo, o usuário enviou para o sistema uma imagem sem nódulo. O valor indicado como 96.37% de confiabilidade corresponde, nesse caso, ao índice de *recall* do modelo em uso (a chance de, quando o modelo diz que não apresenta nódulo, ser um FN). A imagem exibida é uma versão em JPG do DICOM original, com uma paleta de cores para facilitar a visualização do usuário (aumentar o contraste do formato em escala de cinza original).

Figura 38 - Predição com nódulo

Mestrado Profissional em Engenharia Mecânica

Redes Neurais Convolucionais para detecção de nódulos

Imagem Processada:



Resultado: Atenção, Apresentou Nódulo! (82,66% de confiabilidade)

Rodrigo Biasuz

Orientador: Prof. Leandro Luís Corso

Fonte: Elaborado pelo autor (2020)

Similar à tela anterior, nesse caso, a única diferença é o índice de confiabilidade indicado, de 82,66%, esse valor corresponde a precisão do modelo, e é a chance de, no caso de uma predição indicando a presença de ser nódulo, que esse resultado seja um FP.

Após as duas últimas telas, o usuário tem a opção de retornar à tela inicial ou fechar e sair da aplicação, ambos executados pela janela de comandos do próprio navegador, e isso limita as funcionalidades básicas desenvolvidas no sistema criado. Como a linguagem Python é orientada a objetos, melhorias no sistema podem ser facilmente implementadas sem alterar as

funções básicas, assim como é possível transportar as funções já criadas em outros sistemas, de forma modular, bastando, para isso, importar os arquivos que contém as informações do modelo matemático.

5 CONCLUSÃO

Este capítulo foi dividido em duas partes: as considerações finais acerca do trabalho desenvolvido e dos resultados alcançados e, por fim, sugestões de trabalhos futuros para aprimorar e expandir a pesquisa realizada com esse projeto.

5.1 CONSIDERAÇÕES FINAIS

Em um mundo onde cada vez mais a interdisciplinaridade tem papel fundamental para a inovação, impulsionando o desenvolvimento de novas formas para abordar problemas, engenharia, medicina, programação, estatística, matemática e muitas outras áreas podem se beneficiar dos avanços tecnológicos uma das outras. Este trabalho, primeiramente, ajuda a colaborar com essa ideia de que diferentes áreas em sincronia tendem a ter melhores resultados em prol de objetivos comuns, desde questões como produzir mais eficientemente até como ajudar no combate ao câncer.

O câncer, sendo um dos maiores males para o ser humano, tem mobilizado uma série de pesquisas e investimentos de organizações públicas e privadas para o seu combate, seja por meio de avanços no tratamento em uma possível cura, um possível método de prevenção ou mesmo em seu diagnóstico. Considerando que um dos principais fatores de sobrevida do paciente acometido pela doença é a sua identificação nos estágios iniciais, nesse trabalho se propôs a desenvolver um modelo matemático de CNN que possa rodar em uma aplicação WEB capaz de auxiliar nessa tarefa, mais especificamente, na detecção de nódulos em imagens de tomografia computadorizada.

Com a coleta de dados e o entendimento de como os mesmos foram construídos/anotados, foi possível pré-processar as informações das amostras e separar as mesmas em conjuntos de teste e treino, com 1800 e 450 imagens em cada um, respectivamente. Dessa forma um modelo matemático inicial foi criado, testado, otimizado e os resultados foram avaliados, possibilitando ainda a colocação do estudo em um sistema online capaz de realizar novas predições a partir de novas imagens.

Esta dissertação apresentou um modelo matemático de ML, uma das áreas multidisciplinares com maior crescimento nos últimos anos, que atingiu uma taxa de acerto de 89,77%, o que se considerou satisfatório para a tarefa proposta, tendo em vista que, no próprio banco de dados utilizado, o internacionalmente reconhecido LIDC-IDRI, os 4 especialistas envolvidos fizeram a mesma predição para uma imagem que eles consideraram como “nódulo

presente” em apenas 26,3% dos casos. Uma série de fatores podem explicar esse baixo índice de concordância entre os radiologistas, como fadiga após observar muitas imagens, dificuldade de interpretar alguma imagem pela complexidade da mesma, erro operacional, erro de anotação, entre diversas outras falhas que humanos são susceptíveis.

O resultado de acuracidade, de 89,77%, potencialmente seria maior caso a etapa de otimização tivesse focado nesse parâmetro, entretanto, optou-se por priorizar o índice de *recall* durante a pesquisa, tendo em vista o cenário prático da aplicação, onde é considerado pior um falso negativo para o paciente do que um falso positivo. Com essa priorização e enfoque no quesito de evitar ao máximo possíveis falsos negativos, o modelo final atingiu um resultado expressivo de 96,37%, o que também se considerou satisfatório para a tarefa proposta. A partir disso, esse modelo matemático foi levado para o sistema de predição WEB construído.

É sabido que um algoritmo de ML, por sua vez, também vai apresentar erros ao longo de uma série de predições. No caso deste trabalho, a classificação de amostras que contêm nódulo, durante a etapa de teste, teve uma taxa de erro de 3,62% dos casos, enquanto a classificação de amostras que não apresentam nódulos ocorre erroneamente em 17,34% das predições. Dessa forma, o uso em conjunto dos conhecimentos de especialistas com o auxílio de técnicas matemáticas se torna uma opção viável para não só aumentar a eficiência e reduzir custos operacionais de instituições, públicas ou privadas, mas também para colaborar com a assertividade dos prognósticos e das avaliações médicas.

Apesar do sistema ter apresentado resultados satisfatórios, fica evidente a necessidade de verificar alguns pontos que precisam ser checados para a replicação deste trabalho em um ambiente de produção (colocar o sistema online para uso), seja em uma empresa (com outras imagens) ou algum hospital. O primeiro é a disponibilidade de dados para o treinamento da IA. Nesse trabalho, se utilizou uma coletânea pública para pesquisas acadêmicas e desenvolvimento, porém, mesmo que para imagens similares, pequenas variações oriundas do processo de obtenção das mesmas podem ocasionar resultados diferentes. Isso se dá pelo fato de que redes profundas apresentam muitos parâmetros treinados e especializados em identificar os padrões das imagens que são dispostas inicialmente, na etapa de treino, e pequenas variações, como um outro modelo de equipamento, são suficientes para confundir uma rede. Com isso, dados disponíveis no local em que se planeja utilizar o modelo matemático são sempre a melhor opção para atingir o melhor resultado.

Com a disponibilidade dos dados, tem-se o segundo ponto a ser verificado: tempo de execução. Neste trabalho, a etapa de treinamento levou aproximadamente 11 dias para ser concluída, excluindo o tempo de processamento dos dados. Algumas instituições não dispõem

desse recurso, nesse caso, algumas soluções podem ser utilizadas, como o uso de processamento na nuvem. Serviços como a *Amazon AWS* e a *Google Cloud* permitem que o usuário alugue recursos computacionais de seus servidores. Possibilitando a realização de tarefas como o treinamento de modelos sem precisar dedicar um recurso precioso como escravo dessa atividade.

Outra dificuldade encontrada neste trabalho é a característica “*black box*” dos modelos de aprendizagem profunda. Diversas camadas e parâmetros acabam por inviabilizar a interpretabilidade de uma predição, ou seja, não se consegue explicar facilmente o motivo de uma imagem ter sido classificada em uma determinada categoria. Apesar de ser possível refazer o “caminho da imagem” ao longo do modelo, calcular cada parâmetro de entrada vezes cada um dos milhares de parâmetros existentes torna essa tarefa praticamente impossível de realizar manualmente. Dessa forma, esse conceito de interpretar cada predição acaba sendo pouco utilizado em projetos de ML, e as interpretações ficam mais no âmbito de performance do modelo, com os riscos calculados não a partir de analisar a predição de cada imagem, mas sim de uma avaliação acerca dos parâmetros de acuracidade, *recall*, precisão e *f1-score* de um grupo maior de imagens separadas inicialmente para teste.

Mesmo com os pontos levantados, fica evidente, por meio desta pesquisa, que a área de ciência de dados, de forma interdisciplinar, juntando conhecimentos de estatística, matemática, engenharia e programação, pode resultar em avanços nos mais diversos ramos em que pode ser aplicada, seja na biologia, na medicina, em investimentos, no jurídico, nos recursos humanos, na segurança, na qualidade, no ensino, em logística ou qualquer outro. Todas as áreas, quando colocadas em conjunto com a ciência de dados e o aprendizado de máquina, apresentam novos desafios e, conseqüentemente, novas oportunidades. Nesse período de pandemia e cenário econômico incerto, a IA é uma das apostas de muitas empresas, organizações e também minha, como autor desse trabalho.

5.2 SUGESTÕES DE TRABALHOS FUTUROS

Ao longo do desenvolvimento do trabalho, e ao fim do mesmo, observam-se algumas oportunidades de melhoria e de continuidade para aprimoramento da técnica e dos resultados, as sugestões são dispostas em itens e não são interdependentes entre si.

- a) Utilizar AG para otimização de hiperparâmetros;
- b) Agrupar de diferentes modelos matemáticos;

- c) Desenvolver de um App Mobile;
- d) Desenvolver um sistema de ranqueamento de TCs;
- e) Aplicar *Transfer Learning* para outros hospitais.

5.2.1 Utilizar AG para otimização de hiperparâmetros

Neste trabalho se utilizou a busca aleatória como método de otimização de hiperparâmetros, devido às limitações de tempo e recurso computacional. Como primeira sugestão de trabalhos futuros, a substituição desse formato de otimização pelo método de AGs, buscando comparar o desempenho final do sistema em termos de performance e também em tempo de velocidade de treinamento da rede.

A ideia aqui seria estabelecer um *trade-off* entre investimento no processo de treinamento, com sistemas mais complexos e, consequentemente, mais demorados e o potencial ganho em termos de acuracidade e *recall*.

5.2.2 Agrupar diferentes modelos matemáticos

Ao longo do trabalho realizado, buscou-se construir e otimizar um modelo de CNN para detecção de nódulos pulmonares. Porém, assim como o processo de anotação do banco de dados utilizado, o LIDC-IDRI, utilizou mais de uma pessoa para a tarefa (foram 4 especialistas em TCs), uma possibilidade é o uso de 4 (ou mais) modelos para a predição de uma imagem.

A ideia aqui é bem simples (apesar de trabalhosa), construir diferentes modelos, como outras CNNs, florestas aleatórias, redes neurais, algoritmos de clusterização, modelos de Markov, *Naive Bayes*, entre outros. Cada um sendo treinado e otimizado individualmente, para, por fim, passar uma imagem em cada um dos modelos e usar algum sistema de agregação desses resultados, podendo esse ser um sistema de “voto simples”, onde a categoria que mais modelos classificarem a imagem é a categoria final, ou usar alguma média baseada nas performances individuais de cada um.

5.2.3 Desenvolver um App Mobile

A terceira sugestão não está relacionada com o melhor desempenho do sistema, e sim na facilidade de uso do mesmo. Nesta pesquisa, se desenvolveu um sistema WEB como

exemplo prático de aplicação. Entretanto, atualmente, o uso de *smartphones* é parte do dia-a-dia de muitas pessoas.

Nesse cenário, a conversão do sistema para um aplicativo de dispositivos móveis pode facilitar o seu uso, ou mesmo integrar esse aplicativo a um já existente em alguma rede de hospitais, melhorando a experiência do usuário, o que pode ser um fator limitante na adesão de qualquer sistema novo.

5.2.4 Desenvolver um sistema de ranqueamento de TCs

Outra possível sugestão é a construção de, ao invés de um sistema de classificação puro, que aponta cada imagem como pertencente a uma categoria ou outra, um sistema de ranqueamento das TCs, que prioriza aquelas que têm maior probabilidade de apresentarem nódulos. É possível construir, usando a mesma lógica e metodologia usada neste trabalho, uma CNN que, no lugar de um valor 0 ou 1 na camada de saída, apresenta valores intermediários.

A ideia aqui é criar um sistema que não vai dar a resposta diretamente, mas sim vai auxiliar os especialistas a tomarem decisões mais rápidas, priorizarem análises mais críticas e, potencialmente, diminuir a taxa de erros. Um sistema assim pode ser de mais fácil implementação inicial, pelo fato de ser apenas um método de priorização, que não diz diretamente se uma imagem apresenta ou não nódulo, tendo em vista que um modelo como o proposto ao longo desse trabalho pode gerar certo desconforto para pessoas que não estão acostumadas com modelos de IA.

5.2.5 Aplicar Transfer Learning para outros hospitais.

Por fim, a última sugestão de trabalho futuro é a aplicação de *Transfer Learning* (aprendizado por transferência) em outras instituições e hospitais. A técnica consiste em pegar um modelo pré-treinado e reutilizar o mesmo para uma nova aplicação, alterando apenas as últimas camadas (ou adicionando camadas novas), e mantendo as camadas iniciais com seus pesos já treinados. Esse procedimento, bastante utilizado para processamento de imagens, têm possibilitado diversas aplicações pois, além de reduzir o tempo inicial de treinamento, pode reduzir a necessidade de bancos de dados gigantescos para apenas algumas dezenas de imagens (MEAZZINI, 2019).

A lógica por trás do *Transfer Learning* consiste na premissa de que um modelo de IA começa com abstrações mais simples e abrangentes nas camadas iniciais, como detecção de

contornos, e, à medida que avança até as últimas camadas, detecta características mais específicas. Dessa forma, é possível manter as primeiras camadas de um modelo complexo e já treinado fixas, alterando apenas as últimas para a solução de um novo problema.

A ideia aqui seria, então, se aproveitar do grande volume de imagens processadas nessa pesquisa, com os dados do banco de dados LIDC-IDRI para outros locais. O intuito é que hospitais menores, que tenham suas próprias imagens de tomografia, possam treinar um modelo de aprendizagem profunda complexo, utilizando de uma base já pré-treinada com milhares de dados e apenas ajustando as últimas camadas com os valores disponíveis localmente.

Essa aplicação tende a trazer bons resultados, pois se têm o poder de diversas imagens de um banco de dados público gigantesco e a caracterização de imagens locais (possivelmente de uma mesma máquina ou de um grupo de indivíduos com características similares devido a pertencerem à mesma região). Além disso, possibilita que instituições menores, com menos recursos e com menos dados disponíveis possam se aproveitar dos benefícios de modelos de IA em suas rotinas.

REFERÊNCIAS

ALVES, Gisely. **Entendendo Redes Convolucionais: CNNs**. 2018. Disponível em: <https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184>. Acesso em: 10 mar. 2020.

ANH, Dao Nam; HOANG, Nguyen The. Learning Validation for Lung CT Images by Multivariable Class Imbalance. *Frontiers In Intelligent Computing: Theory and Applications*, [S.L.], p. 50-60, 4 out. 2019. **Springer Singapore**. Disponível em: http://dx.doi.org/10.1007/978-981-32-9186-7_6. Acesso em: 10 jan. 2021

ARDILA, D., Kiraly, A.P., Bharadwaj, S. et al. **End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography**. *Nat Med* 25, 954–961 (2019) doi: <https://doi.org/10.1038/s41591-019-0447-x>

ARELLANO, J.C.V. **Tomografia computadorizada (TC) no diagnóstico e controle do tratamento das disfunções da articulação temporomandibular**. JBA, Curitiba, v.1, n.4, p.315-323, out./dez. 2001

ARMATO, Samuel G. et al. The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): a completed reference database of lung nodules on ct scans. *Medical Physics*, [S.L.], v. 38, n. 2, p. 915-931, 24 jan. 2011. Wiley. <http://dx.doi.org/10.1118/1.3528204>. Disponível em: <https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.3528204>. Acesso em: 10 jan. 2021.

ASIRI, Sidath. **Machine Learning Classifiers**. 2018. Disponível em: <https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623>. Acesso em: 10 mar. 2020.

BBC Research, 2018, **Machine Learning: Global Markets to 2022**, Code - IFT159A, BCC Publishing, Disponível em: <https://www.bccresearch.com/market-research/information-technology/machine-learning-global-markets.html>. Acesso em: 29 out. 2019

BEAZLEY, D. e JONES, B.K. **Python Cookbook**. Ed. Novatec, 2013.

BERGSTRA, J.; BENGIO, Y. **Random search for hyper-parameter optimization**. *Journal of Machine Learning Research*, v. 13, n. Feb, p. 281–305, 2012.

BERGSTRA, J, YAMINS, D, e COX, D. 2013. Making a science of model search: **Hyperparameter optimization in hundreds of dimensions for vision architectures**. *International Conference on Machine Learning*, páginas 115–123

BERRAR, Daniel. **Cross-Validation**. *Encyclopedia Of Bioinformatics And Computational Biology*, [s.l.], p.542-545, 2019. Elsevier. DOI: <http://dx.doi.org/10.1016/b978-0-12-809633-8.20349-x>.

BOECHEL, T. **Algoritmo de otimização: uma abordagem híbrida utilizando o algoritmo das formigas e genético**. 2003. 79 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 2003.

BOTTOU, Leon. **Large-Scale Machine Learning with Stochastic Gradient Descent**. NEC Lab America, Princeton NJ 08542, USA. Disponível em: <https://leon.bottou.org/publications/pdf/compstat-2010.pdf>. Acessado em: 21 jun. 2019.

BRADY, Henry E.. The Challenge of Big Data and Data Science. **Annual Review Of Political Science**, [s.l.], v. 22, n. 1, p. 297-323, 11 maio 2019. Annual Reviews. DOI: <http://dx.doi.org/10.1146/annurev-polisci-090216-023229>.

BROOKS, Megan. **FDA AI Device to Detect Diabetic Retinopathy**. 2018. Disponível em: <https://www.webmd.com/diabetes/news/20180412/fda-oks-ai-device-to-detece-diabetic-retinopathy>. Acesso em: 10 mar. 2020.

BUDHIRAJA, Amar. **Dropout in (Deep) Machine learning**. 2016. Disponível em: <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>. Acessado em: 25 jun. 2019.

CASTRO, R. E. **Otimização de estruturas com multiobjetivos via Algoritmos Genéticos**. 2001. 226 f. Tese (Doutorado em Engenharia Civil) – COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2001.

CHEN, Hongge. (2017). **Novel machine learning approaches for modeling variations in semiconductor manufacturing**. Disponível em: https://www.researchgate.net/publication/320487716_Novel_machine_learning_approaches_for_modeling_variations_in_semiconductor_manufacturing. Acessado em: 16 mar. 2020

CLAVERO, José Miguel. NÓDULOS PULMONARES. **Revista Médica Clínica Las Condes**, [s.l.], v. 26, n. 3, p.302-312, maio 2015. Elsevier BV. DOI: <http://dx.doi.org/10.1016/j.rmclc.2015.06.007>.

CORNELISSE, Daphne. **An intuitive guide to Convolutional Neural Networks**. 2018. Disponível em: <https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/>. Acessado em: 21 jun 2019.

CUS, F.; BALIC, J. Optimization of cutting process by GA approach. **Robotics and Computer Integrated Manufacturing**, v. 19, p. 113-121, 2003.

DANJUMA, Kwetishe Joro. **Performance Evaluation of Machine Learning Algorithms in Post-operative Life Expectancy in the Lung Cancer Patients**. **Ijcsi International Journal Of Computer Science Issues**, Volume 12, Issue 2, Adamawa., abr. 2015. Disponível em: <https://arxiv.org/pdf/1504.04646.pdf>. Acesso em: 10 mar. 2020.

DASH, Sabyasachi et al. Big data in healthcare: management, analysis and future prospects. : management, analysis and future prospects. **Journal Of Big Data**, [s.l.], v. 6, n. 1, 19 jun. 2019. Springer Science and Business Media LLC. DOI: <http://dx.doi.org/10.1186/s40537-019-0217-0>.

DEB, S. ; DIXIT, U. S. Intelligent machining: Computational methods and optimization. In: DAVIM, J. P. (Org.). **Machining: fundamentals and recent advances**. London: Springer, 2008. cap. 12.

DEVLIN, Jacob et al. **BERT**: Pre-training of Deep Bidirectional Transformers for Language Understanding. Arxiv:1810.04805v2 [cs.cl], New York,, 24 maio 2019. Disponível em: <https://arxiv.org/abs/1810.04805>. Acesso em: 10 mar. 2020.

DETTAT, Arden. **Applied Deep Learning**: convolutional neural networks. Convolutional Neural Networks. 2017. Disponível em: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>. Acesso em: 10 mar. 2020.

FANGOHR, H. **Python for Computational Science and Engineering**, Faculty of Engineering and the Environment University of Southampton, Southampton 2015.

FERRAZ, Inhaúma e CONCI, Aura. (2003). **Processamento de imagens na tomografia computadorizada**. INCON 2003 2º (SBMAC), São José dos Campos. Disponível em: https://www.researchgate.net/publication/256793882_Processamento_de_imagens_na_tomografia_computadorizada. Acesso em: 10 mar. 2020.

PHILLIPS, Michael; MARSDEN, Helen; JAFFE, Wayne; MATIN, Rubeta N.; WALI, Gorav N.; GREENHALGH, Jack; MCGRATH, Emily; JAMES, Rob; LADOYANNI, Evmorfia; BEWLEY, Anthony. Assessment of Accuracy of an Artificial Intelligence Algorithm to Detect Melanoma in Images of Skin Lesions. **Jama Network Open**, [S.L.], v. 2, n. 10, 16 out. 2019. American Medical Association (AMA). DOI: <http://dx.doi.org/10.1001/jamanetworkopen.2019.13436>.

FISCHER E. Alice. FRANCES E. Grodzinsky. **The Anatomy of Programming Languages**. Prentice-Hall, 1993 Englewood Cliffs, New Jersey: Prentice Hall. p. 3. 557 páginas. ISBN 0-13-035155-5

FIELDING, R. T. **REST Architectural Styles and the Design of Network-based Software Architectures**. University of California, Irvine, 2000.

GHARAJEH, Mohammad Samadi. Biological Big Data Analytics. **Advances In Computers**, [s.l.], p. 321-355, 2018. Elsevier. DOI: <http://dx.doi.org/10.1016/bs.adcom.2017.08.002>.

GOMES, Guilherme Gustavo Ramos. **Tábula**: uma framework para o desenvolvimento de aplicações rest. 2008. 189 f. Dissertação (Mestrado) - Curso de Engenharia Informática, Universidade da Madeira, Funchal - Portugal, 2008. Disponível em: <https://core.ac.uk/download/pdf/62477494.pdf>. Acesso em: 20 jan. 2021.

GU Y, LU X, ZHANG B, ZHAO Y, YU D, GAO L, et al. (2019) **Automatic lung nodule detection using multi-scale dot nodule-enhancement filter and weighted support vector machines in chest computed tomography**. DOI: <https://doi.org/10.1371/journal.pone.0210551>

GUPTA, Harshit; JIN, Kyong Hwan; NGUYEN, Ha Q.; MCCANN, Michael T.; UNSER, Michael. CNN-Based Projected Gradient Descent for Consistent CT Image Reconstruction. **Ieee Transactions On Medical Imaging**, [s.l.], v. 37, n. 6, p.1440-1453, jun. 2018. Institute of Electrical and Electronics Engineers (IEEE). DOI: <http://dx.doi.org/10.1109/tmi.2018.2832656>.

GUPTA, Prashant. **Cross-Validation in Machine Learning**. 2017. Disponível em: <https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f>. Acesso em: 10 mar. 2020.

HAMDAN, Muhammad. **VHDL: auto-generation tool for optimized hardware acceleration of convolutional neural networks on FPGA (VGT)**, 2018, Disponível em: https://www.researchgate.net/figure/Activation-Functions-ReLU-Tanh-Sigmoid_fig4_327435257, Acesso em: 29 out. 2019

IARC - **INTERNATIONAL AGENCY FOR RESEARCH ON CANCER** - WORLD HEALTH ORGANIZATION, 2019, Disponível em: <https://www.iarc.fr>, Acesso em: 29 out. 2019

INCA - **Instituto Nacional de Câncer**, 2019. Disponível em: <http://www2.inca.gov.br/wps/wcm/connect/inca/portal/home>. Acesso em: 29 out. de 2019.

JAIN, Pawan. **Complete Guide of Activation Functions**. 2019. Disponível em: <https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044>. Acesso em: 10 mar. 2020.

JIN, Jonghoon; DUNDAR, Aysegul; CULURCIELLO, Eugenio. **Flattened Convolutional Neural Networks for Feedforward Acceleration**. **Arxiv:1412.5474 [cs.ne]**, New York, 20 nov. 2015. Disponível em: <https://arxiv.org/abs/1412.5474>. Acesso em: 10 mar. 2020.

JOHN, Gutttag, V. 2016. **Introduction to Computation and Programming Using Python: With Application to Understanding Data**. MIT Press. ISBN 978-0-262-52962-4.

KALDERA, Nikalal, RAMESH, Shanaka & DISSANAYAKE, Maheshi. (2019). **Brain tumor Classification and Segmentation using Faster R-CNN**. 10.1109/ICASET.2019.8714263.

KAMIYA, H. et al. **Pulmonary nodules: A quantitative method of diagnosis by evaluating nodule perimeter difference to approximate oval using three-dimensional CT images**. *Clinical Imaging*, v. 35, n. 2, p. 123–126, 2011.

KAPLIŃSKI, Oleg; KOŁELEVA, Natalija; ROPAITÈ, Guoda. **BIG DATA IN CIVIL ENGINEERING: a state-of-the-art survey. : A STATE-OF-THE-ART SURVEY. Engineering Structures And Technologies**, [s.l.], v. 8, n. 4, p. 165-175, 16 dez. 2016. Vilnius Gediminas Technical University. DOI: <http://dx.doi.org/10.3846/2029882x.2016.1257373>.

KLANG, Eyal. **Deep learning and medical imaging**. **Journal Of Thoracic Disease**, [s.l.], v. 10, n. 3, p.1325-1328, mar. 2018. AME Publishing Company. DOI: <http://dx.doi.org/10.21037/jtd.2018.02.76>.

Krizhevsky, A., Sutskever, I., e Hinton, G. E. (2012). **Imagenet classification with deep convolutional neural networks**. In *Advances in neural information processing systems*, páginas 1097–1105.

KUKREJA, Harsh, N. Bharath, C. Siddesh, S. Kuldeep Shiruru, (2016). **AN INTRODUCTION TO ARTIFICIAL NEURAL NETWORK**. *International Journal of Advance Research and Innovative Ideas in Education*. 1. 27-30.

LEVER, Jake; KRZYWINSKI, Martin; ALTMAN, Naomi. **Classification evaluation**. **Nature Methods**, [s.l.], v. 13, n. 8, p.603-604, 28 jul. 2016. Springer Science and Business Media LLC. <http://dx.doi.org/10.1038/nmeth.3945>.

LIU, Y.; WANG, C. A modified genetic algorithm based optimization of milling parameters. **International Journal of Advanced Manufacturing Technology**, v. 15, p. 796-799, 1999.

LINDEN, R. **Algoritmos Genéticos: uma importante ferramenta da inteligência computacional**. 2. ed. Rio de Janeiro: Brasport, 2008.

LÓPEZ de Prado, Marcos, **Advances in Financial Machine Learning**: Numerai's Tournament (Presentation Slides) (November 1, 2019). Disponível em: <https://ssrn.com/abstract=3478927> DOI: <http://dx.doi.org/10.2139/ssrn.3478927> Acessado em: 19 de abr. 2020.

LOPIANO, Samuele. **Ethical principles in machine learning and artificial intelligence**: cases from the field and possible ways forward. Humanities And Social Sciences Communications, [S.L.], v. 7, n. 1, p. 17 jun. 2020. Springer Science and Business Media LLC. DOI: <http://dx.doi.org/10.1057/s41599-020-0501-9>.

MACKAY, D. J. C. **Information Theory, Inference and Learning Algorithms**. Cambridge: Cambridge University Press, 2005.

ABADI, Martín, Ashish AGARWAL, Paul BARHAM, Eugene BREVDO, Zhifeng CHEN, Craig CITRO, Greg S. CORRADO, Andy DAVIS, Jeffrey DEAN, Matthieu DEVIN, Sanjay GHEMAWAT, Ian GOODFELLOW, Andrew HARP, Geoffrey IRVING, Michael ISARD, Rafal JOZEFOWICZ, Yangqing JIA, Lukasz KAISER, Manjunath KUDLUR, Josh LEVENBERG, Dan MANÉ, Mike SCHUSTER, Rajat MONGA, Sherry MOORE, Derek MURRAY, Chris OLAH, Jonathon SHLENS, Benoit STEINER, Ilya SUTSKEVER, Kunal TALWAR, Paul TUCKER, Vincent VANHOUCKE, Vijay VASUDEVAN, Fernanda VIÉGAS, Oriol VINYALS, Pete WARDEN, Martin WATTENBERG, Martin WICKE, Yuan YU, and Xiaoqiang ZHENG. **TensorFlow: Large-scale machine learning on heterogeneous systems**, 2015. Disponível em: www.tensorflow.org. Acessado em: 21 jun. 2020.

MEAZZINI, Luis. **Transfer Learning aplicado no reconhecimento de flores**. 2019. Disponível em: <https://medium.com/ensina-ai/tutorial-transfer-learning-3972cac5e9b5>. Acesso em: 15 jan. 2021.

MILLMAN, K. Jarrod; AIVAZIS, Michael. **Python for Scientists and Engineers**. Computing In Science & Engineering, [S.L.], v. 13, n. 2, p. 9-12, mar. 2011. Institute of Electrical and Electronics Engineers (IEEE). DOI: <http://dx.doi.org/10.1109/mcse.2011.36>.

MORIKAWA, Rei. **What is Data Annotation and How is it Used in Machine Learning?** 2020. Disponível em: <https://lionbridge.ai/articles/data-annotation-machine-learning/>. Acesso em: 10 jan. 2021

NAR, Kamil et al. **Cross-Entropy Loss and Low-Rank Features Have Responsibility for Adversarial Examples**. 2019. Disponível em: <https://arxiv.org/pdf/1901.08360.pdf>. Acessado em: 21 de set. 2019.

NARAYANAN, Ramaswamy. Big Data Analytics and Cancer. **Moj Proteomics & Bioinformatics**, [s.l.], v. 4, n. 2, 13 out. 2016. MedCrave Group, LLC. DOI: <http://dx.doi.org/10.15406/mojpb.2016.04.00115>.

NASR, G.e.; BADR, E.a.; JOUN, C.. **Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand**. 2002. Disponível em: <https://pdfs.semanticscholar.org/f287/51818854d9dbb27e60706c4161b40a326d81.pdf>>. Acessado em: 21 jun. 2019.

Nielsen, Michael A. **Neural Networks and Deep Learning**, Determination Press, 2015. <http://static.latexstudio.net/article/2018/0912/neuralnetworksanddeeplearning.pdf>. Acessado em 20 de fev. de 2020

OGUNYALE, Kehinde. **Convolutional Neural Network on Nigerian Foods**. Disponível em: <https://blog.usejournal.com/convolutional-neural-network-on-nigerian-foods-565493fcdd0e>. Acesso em: 10 mar. 2020.

OST D, Fein AM, Feinsilver SH et al. Clinical practice. **The solitary pulmonary nodule**. N Engl J Med 2003; 348:2535-42.

PANCH, Trishan; SZOLOVITS, Peter; ATUN, Rifat. Artificial intelligence, machine learning and health systems. **Journal Of Global Health**, [S.L.], v. 8, n. 2, 21 out. 2018. International Global Health Society. DOI: <http://dx.doi.org/10.7189/jogh.08.020303>.

PARMAR, Ravindra. **Common Loss functions in machine learning**. 2018. Disponível em: <https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23>. Acesso em: 10 mar. 2020.

POKHARNA, Harsh. **The Introduction to Neural Networks we all need**. 2016. Disponível em: <https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-c50f6012d5eb>. Acessado em: 12 nov. de 2019.

PRODANOV, C. C, FREITAS, E. C. (2013). **Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico** (2ª ed). Brasil: Universidade Feevale

RAJPURKAR P, IRVIN J, ZHU K, YANG B, MEHTA H, DUAN T, et al. (2017) **CheXNet: radiologist-level pneumonia detection on chest x-rays with deep learning**, arXiv:1711.05225v3

RAZZAK, Muhammad Imran; NAZ, Saeeda; ZAIB, Ahmad. **Deep Learning for Medical Image Processing: Overview, Challenges and Future**. Arxiv:1704.06825 [cs.cv], New York, v. 1, n. 1, p.1-1, abr. 2017. Disponível em: <https://arxiv.org/pdf/1704.06825.pdf>. Acesso em: 10 mar. 2020.

RICHARDSON, Leonard e RUBY, Sam. **RESTful Web Services**. O'Reilly, 2007.

ROBINSON, David. **The Incredible Growth of Python**. 2018. Disponível em: <https://stackoverflow.blog/2018/09/06/incredible-growth-python/>. Acesso em: 20 nov. 2020.

RODRIGUES, Vitor. **Métricas de Avaliação: acurácia, precisão, recall**. acurácia, precisão, recall. 2019. Disponível em: <https://medium.com/vitorborbarodrigues/m%C3%A9tricas-de-avalia%C3%A7%C3%A3o-acur%C3%A1cia-precis%C3%A3o-recall-quais-as-diferen%C3%A7as-c8f05e0a513c>. Acesso em: 10 mar. 2020.

SAHU, Satya Prakash *et al.* Segmentation of Lungs in Thoracic CTs Using K-means Clustering and Morphological Operations. **Advances In Biomedical Engineering And Technology**, [S.L.], p. 331-343, 29 set. 2020. Springer Singapore. DOI: http://dx.doi.org/10.1007/978-981-15-6329-4_28.

SCHULTER, M. **Aplicação de Algoritmos Genéticos na modelagem de transformadores a partir de ensaios**. 2007. 99 f. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, 2007.

SILVA, Denise Rossato; BAGLIO, Pierângelo Tadeu; GAZZANA, Marcelo Basso. Nódulo pulmonar solitário. **Rev Bras Clin Med**, Porto Alegre, v. 1, n. 7, p.132-139, 2009. Disponível em: <http://files.bvs.br/upload/S/1679-1010/2009/v7n2/a010.pdf>. Acesso em: 10 mar. 2020.

SLUIMER, I. et al. **Computer analysis of computed tomography scans of the lung: A survey**. IEEE Transactions on Medical Imaging, v. 25, n. 4, p. 385–405, 2006.

SONG, Shuang & CHAUDHURI, Kamalika & D. Sarwate, Anand. (2013). **Stochastic gradient descent with differentially private updates**. 2013 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2013 - Proceedings. 245-248. 10.1109/GlobalSIP.2013.6736861.

SUTTON, Christopher et al. NOMAD 2018 Kaggle Competition: Solving Materials Science Challenges Through Crowd Sourcing. **Arxiv:1812.00085 [cond-mat.mtrl-sci]**, New York, v. 1, n. 1, p.1-1, 30 nov. 2018. Disponível em: <https://arxiv.org/abs/1812.00085>. Acesso em: 10 mar. 2020.

SZE, Vivienne et al. **Efficient Processing of Deep Neural Networks: A Tutorial and Survey**. 2017. Available at: <<https://arxiv.org/pdf/1703.09039.pdf>>. Access in: 21 jun. 2019.

TIOBE. **Python is TIOBE's Programming Language of 2020!** 2021. Disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em: 30 jan. 2021.

WU, Jianxin. **Introduction to Convolutional Neural Networks**. 2017. National Key Lab for Novel Software Technology Nanjing University, China. Available at: <<https://pdfs.semanticscholar.org/450c/a19932fcef1ca6d0442cbf52fec38fb9d1e5.pdf>>. Access in: 21 jun. 2019.

XAVIER, Otávio Calaça. **Serviços Web Semânticos Baseados em RESTful: um estudo de caso em redes sociais online**. Goiânia, set. 2011. Disponível em: https://www.researchgate.net/publication/301561960_Servicos_Web_Semanticos_Baseados_em_RESTful_Um_Estudo_de_Caso_em_Redes_Sociais_Online. Acesso em: 20 jan. 2021.

XIAO, Xueli et al. **Efficient Hyperparameter Optimization in Deep Learning Using a Variable Length Genetic Algorithm**. 2020. Disponível em: <https://arxiv.org/pdf/2006.12703.pdf>. Acesso em: 26 jan. 2021.

YANG, Shengping; BERDINE, Gilbert. **The receiver operating characteristic (ROC) curve. The Southwest Respiratory And Critical Care Chronicles**, [s.l.], v. 5, n. 19, 5 maio 2017. The Southwest Respiratory and Critical Care Chronicles (SWRCCC). DOI: <http://dx.doi.org/10.12746/swrccc.v5i19.391>.

YASAKA K, ABE O (2018) **Deep learning and artificial intelligence in radiology**: Current applications and future directions. PLoS Med 15(11): e1002707. DOI: <https://doi.org/10.1371/journal.pmed.1002707>

YORIYAZ, Hélio. **Método de Monte Carlo**: princípios e aplicações em Física Médica. **Revista Brasileira de Física Médica**. 2009;3(1):141-9., São Paulo, jan. 2009. Disponível em: <https://www.ipen.br/biblioteca/2008/14528.pdf>. Acesso em: 10 mar. 2020.

ZECH JR, BADGELEY MA, LIU M, COSTA AB, TITANO JJ, OERMANN EK. **Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs**: A cross-sectional study. PLoS Med. 2018;15(11):e1002683. DOI: <https://doi.org/10.1371/journal.pmed.1002683>

ZHANG, Shaorong *et al.* Segmentation of small ground glass opacity pulmonary nodules based on Markov random field energy and Bayesian probability difference. **Biomedical Engineering Online**, [S.L.], v. 19, n. 1, p. 99-999, 17 jun. 2020. Springer Science and Business Media LLC. DOI: <http://dx.doi.org/10.1186/s12938-020-00793-0>.

ZHANG, Zhilu; SABUNCU, Mert R. **Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels**. 2018. Disponível em: <https://papers.nips.cc/paper/8094-generalized-cross-entropy-loss-for-training-deep-neural-networks-with-noisy-labels.pdf>>. Acessado em 21 jun 2019.

ZHOU, Peng et al. **Generate, Segment and Refine**: Towards Generic Manipulation Segmentation. Conference On Artificial Intelligence (aa), São Francisco, fev. 2020. Disponível em: <https://ai.facebook.com/research/publications/generate-segment-and-refine-towards-generic-manipulation-segmentation/>. Acesso em: 10 mar. 2020.

APÊNDICE A - ALGORITMOS DO SISTEMA CRIADO

```
# App.py (backend)

# -*- coding: utf-8 -*-

"""

@author: rodrigo.biasuz

"""

import tensorflow as tf

import keras

sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph())

tf.compat.v1.keras.backend.set_session(sess)

import flask

from flask import Flask, jsonify, request, render_template, redirect, g, flash, url_for,
make_response, abort, Response, session

from jinja2 import Environment, FileSystemLoader

from urllib.parse import urljoin, urlparse, urldefrag, unquote

import json

import os

import requests

import cv2

import imageio as im

import numpy as np

from sklearn.preprocessing import StandardScaler, normalize

from keras.models import Sequential, load_model

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D

from matplotlib import pyplot as plt

import seaborn as sns

from keras.models import model_from_json
```

```

from keras.models import load_model

sns.set(style='white', context='notebook', palette='deep')

print("\n libs carregadas \n")

global graph

keras.backend.clear_session()

#graph = tf.get_default_graph()

json_file = open('model.json', 'r')

loaded_model_json = json_file.read()

json_file.close()

model = load_model('my_model_last')

print("\n modelo criado \n")

model.load_weights('model_weights_last.h5')

print("\n pesos carregados \n")

app = Flask(__name__)

UPLOAD_FOLDER = './static'

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

print("\n servidor online carregados \n")

def make_prediction(imagem):

    img = cv2.resize(imagem, (100, 100))

    list_image = [normalize(img)]

    X = np.array(list_image)

    X = X.reshape(-1,100,100,1)

    predictions = model.predict_classes(X)

    print('predições')

    print(predictions)

    if predictions[0] == 0:

        return "Não apresenta Nódulo (96,37% de confiabilidade)"

```

```

else:

    return "Atenção, Apresentou Nódulo! (82,66% de confiabilidade)"

def dicon_as_jpg(imagem, path):

    plt.imsave(path+'.png', imagem)

    return path+'.png'

@app.route('/predict', methods=['GET', 'POST'])
def predict():

    if request.method == 'GET':

        return render_template('home.html')

    if request.files.get('file'):

        file = request.files['file']

        path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)

        file.save(path)

        print(path)

        imagem = im.imread(path).astype(np.float64)

        print('imagem lida')

        resultado = make_prediction(imagem)

        path2 = dicon_as_jpg(imagem, path)

        return render_template('predictions.html', imagem=path2, resultado = resultado)

    elif request.form.get('paste'):

        return Response( f"<h2> Execute </h2>")

if __name__ == '__main__':

    app.run()

```

home.html (frontend)

```

<!DOCTYPE html>

<html lang="en">

```



```

<head>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-
fNmOCqBTIWIj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">

<title>home</title>

</head>

<body>

<div class="container">

<hr>

<h3>Mestrado Profissional em Engenharia Mecânica</h3>

<hr>

<h4>Redes Neurais Convolucionais para detecção de nódulos</h4>

<hr>

<h5>Send a file to predict</h5>

<form action="/predict" method="POST" enctype="multipart/form-data">

<div class="form-group">

<label for="file">file</label>

<input type="file" class="form-control-file" id="file" name="file">

</div>

<hr>

```

```

        <button type="submit" class="btn btn-primary"> <i class="fas fa-save"></i> Save
    </button>

    </form>

    <hr>

    <h7> Rodrigo Biasuz </h7>

    <hr>

    <h7> Orientador: Prof. Leandro Luís Corso </h7>

    </div>

</body>

</html>

```

Arquivo predictions.html (frontend)

```

<!DOCTYPE html>

<html lang="en">

<head>

    <!-- Required meta tags -->

    <meta charset="utf-8">

    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->

    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-
fNmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">

    <title>Predictions</title>

</head>

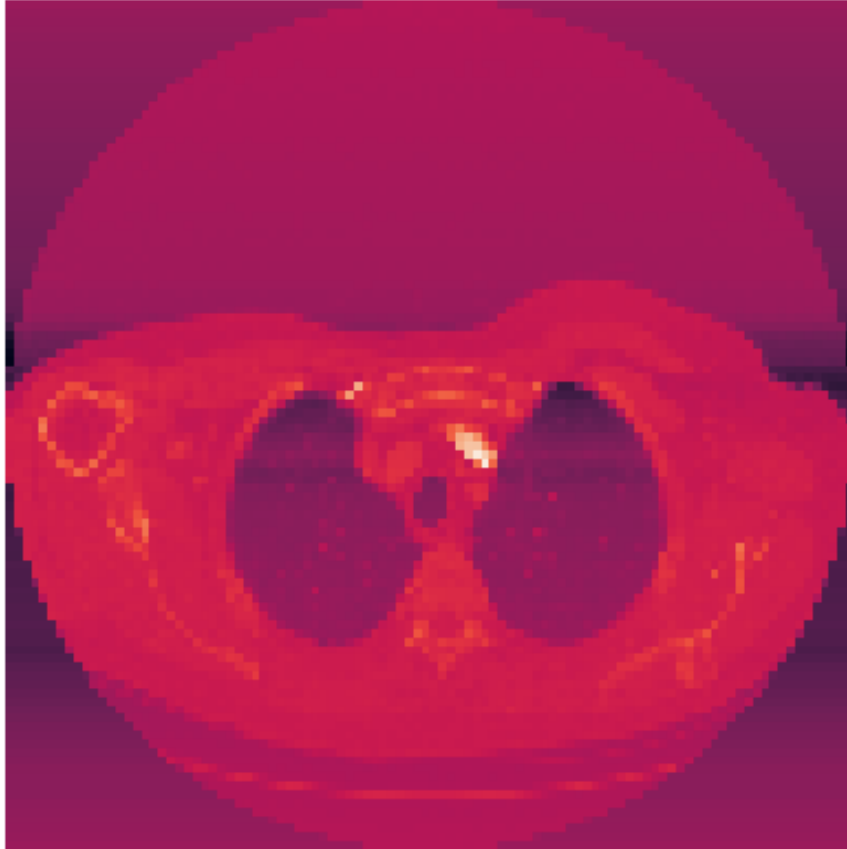
<body>

```

```
<div class="container">  
  
<hr>  
  
  <h3>Mestrado Profissional em Engenharia Mecânica</h3>  
  
<hr>  
  
<h4>Redes Neurais Convolucionais para detecção de nódulos</h4>  
  
<hr>  
  
  <h5>Imagem Processada:</h5>  
  
    
  
<hr>  
  
  <h4>Resultado: {{ resultado }}</h4>  
  
<hr>  
  
<h7> Rodrigo Biasuz </h7>  
  
<hr>  
  
  <h7> Orientador: Prof. Leandro Luís Corso </h7>  
  
</div>  
  
</body>  
  
</html>
```

APÊNDICE B - EXEMPLOS DE IMAGENS E SEUS RESULTADOS

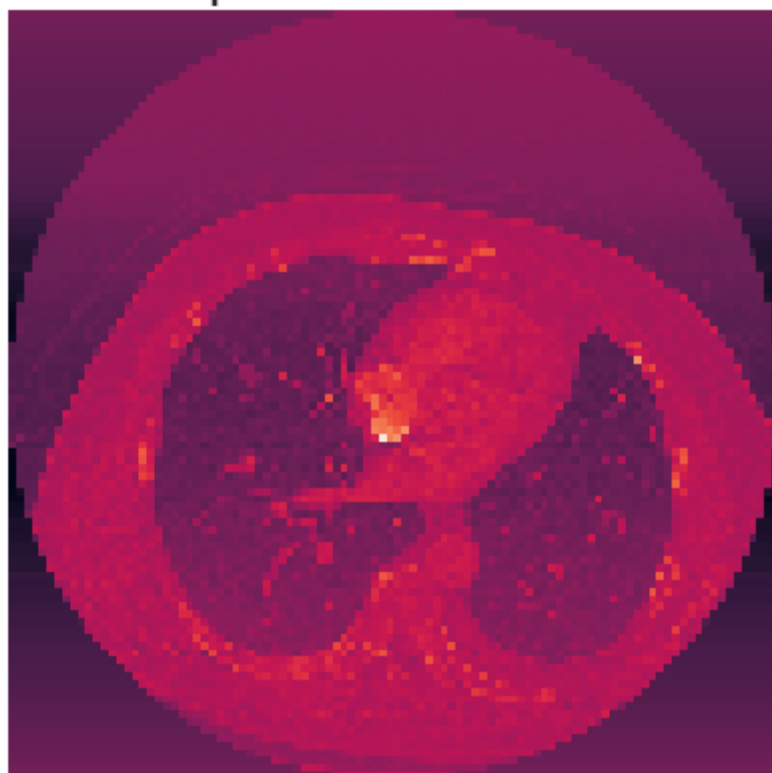
Exemplo de Falso Positivo



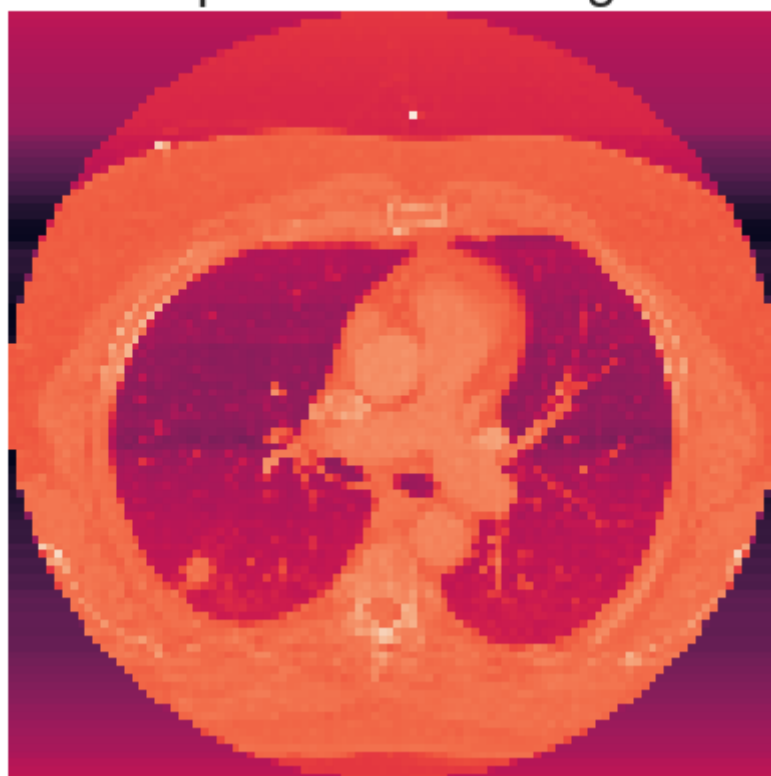
Exemplo de Falso Positivo



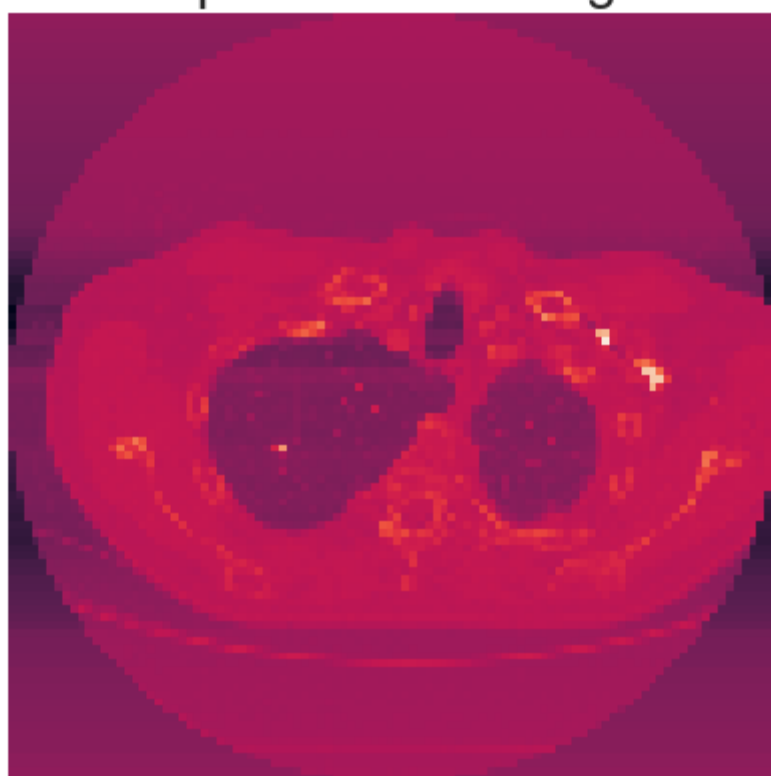
Exemplo de Falso Positivo



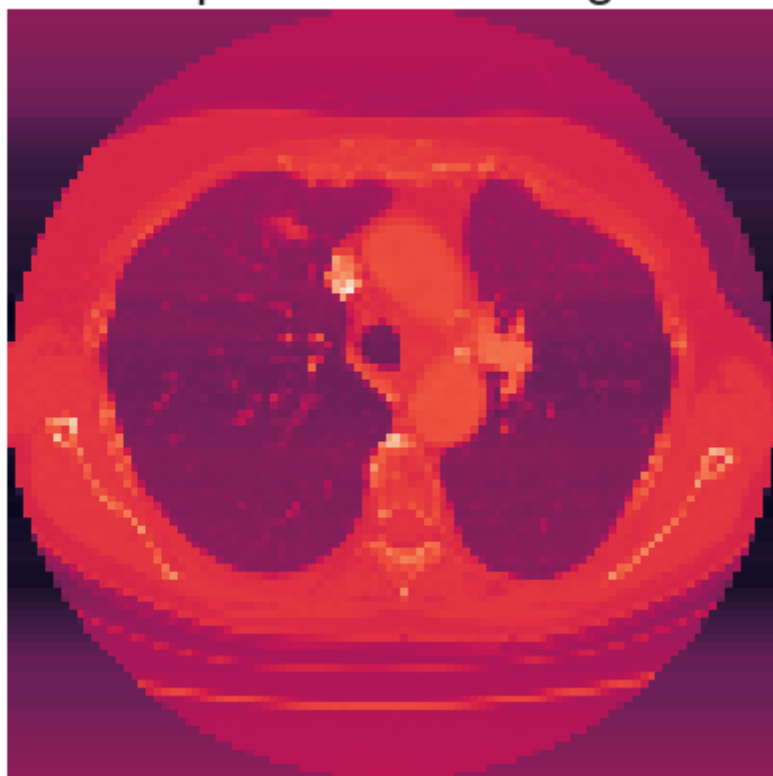
Exemplo de Falso Negativo



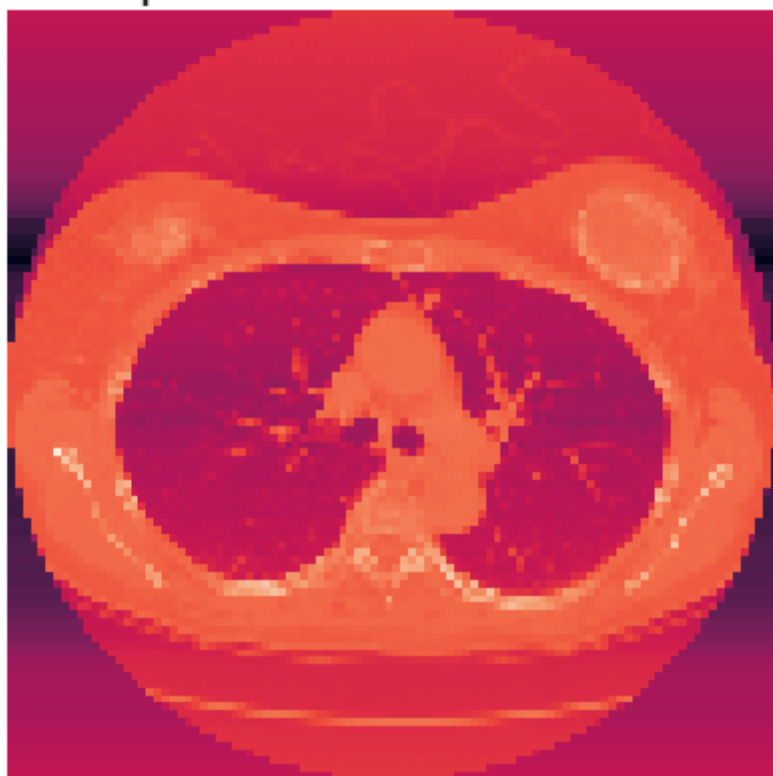
Exemplo de Falso Negativo



Exemplo de Falso Negativo



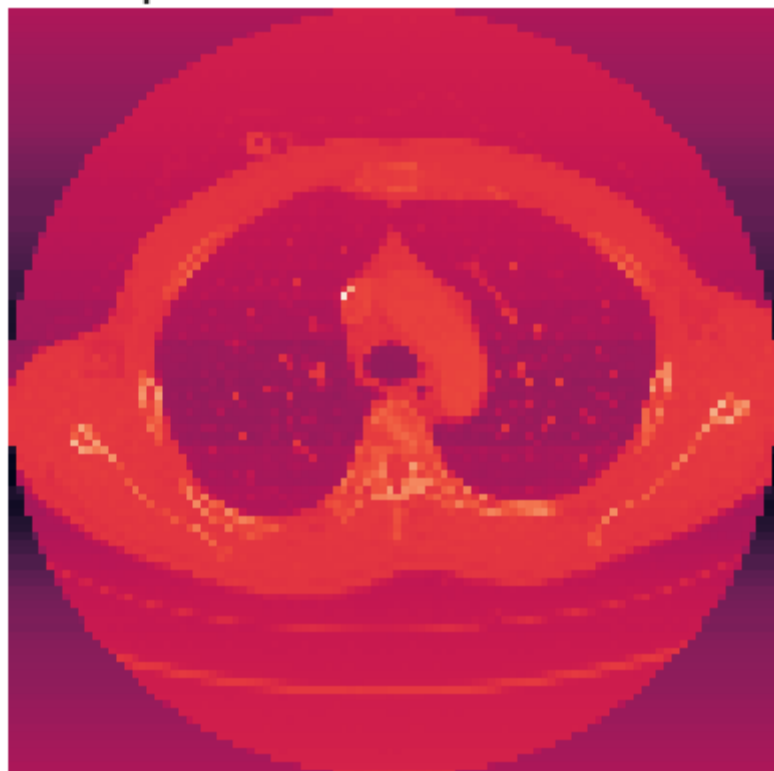
Exemplo de Verdadeiro Positivo



Exemplo de Verdadeiro Positivo



Exemplo de Verdadeiro Positivo



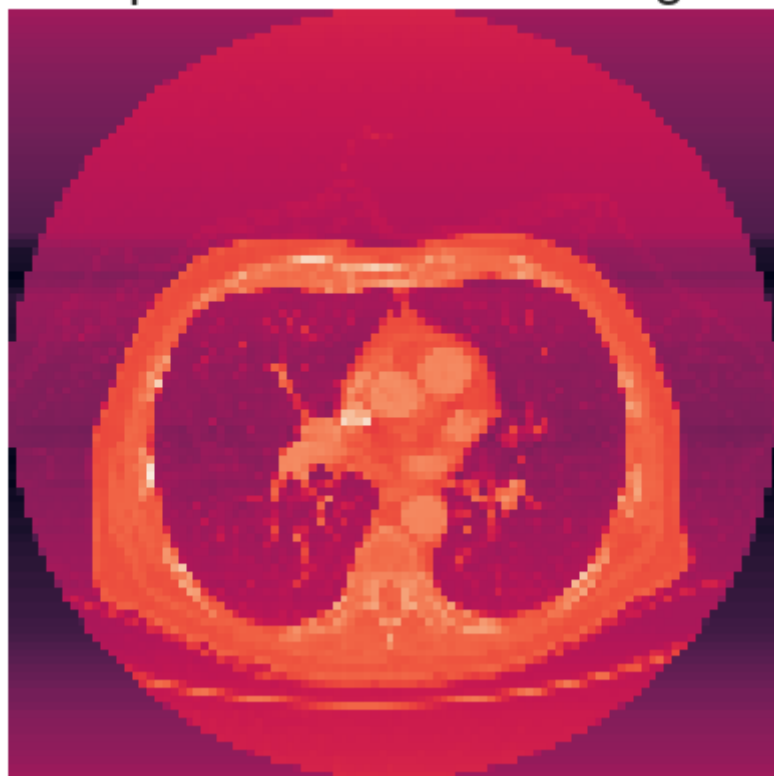
Exemplo de Verdadeiro Negativo



Exemplo de Verdadeiro Negativo



Exemplo de Verdadeiro Negativo



APÊNDICE C - ALGORITMOS DA MODELAGEM

Pré-processamento, construção inicial das redes, treinamento e geração das imagens.

```
import keras

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import imageio as im

import os

import time

%matplotlib inline

import matplotlib.image as mpimg

import seaborn as sns

import cv2

sns.set(style='white', context='notebook', palette='deep')

from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D

from keras.optimizers import RMSprop

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau

# import Sequential from the keras models module

from keras.models import Sequential

# import Dense, Dropout, Flatten, Conv2D, MaxPooling2D from the keras layers module

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
```

```

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, roc_auc_score, accuracy_score, roc_curve,
classification_report, recall_score, precision_score, f1_score

import itertools

import random

from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

from sklearn.preprocessing import StandardScaler, normalize

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

cases = pd.read_csv(r'lista.csv')

cases.head(10)

loaded_images = pd.DataFrame(columns=['scan', 'slice'])

list_images = []

loaded_images

def leitura_update(path):

    url_base = path

    global loaded_images

    global list_images

    start = time.time()

    for file_name in os.listdir(url_base):

        start = time.time()

        for file2 in os.listdir(url_base+'/'+file_name):

            for file3 in os.listdir(url_base+'/'+file_name+'/'+file2):

                for file4 in os.listdir(url_base+'/'+file_name+'/'+file2+'/'+file3):

                    if file4.endswith('.dcm'):

                        img = im.imread(os.path.join(url_base, file_name, file2, file3,
file4)).astype(np.float64)

```

```

img_dic = img.meta

scan = img_dic['SeriesNumber']

slice_n = img_dic['InstanceNumber']

loaded_images.loc[len(loaded_images)] = [scan, slice_n]

img = cv2.resize(img, (100, 100)) # redução dimensional

list_images.append(img)

end = time.time()-start

print('Levou', round(end,2), 'segundos para executar na pasta', file_name)

leitura_update(url_base1)

# Salvando as fotos carregadas

import pickle

with open('list_images_2020', 'wb') as f:

    pickle.dump(list_images, f)

with open('loaded_images_2020', 'wb') as f:

    pickle.dump(loaded_images, f)

# para carregar

import pickle

with open(r'list_images_2020', 'rb') as f:

    list_images = pickle.load(f)

with open(r'loaded_images_2020', 'rb') as f:

    loaded_images = pickle.load(f)

normalized_list = []

for i in list_images:

    normalized_list.append(normalize(i))

X = np.array(normalized_list)

X.shape

X = X.reshape(-1,100,100,1)

```

```

X.shape

loaded_images.head()

to_add = cases[['scan', 'slice']].copy()

to_add['has_tumor'] = 1

to_add.head()

to_add = to_add.drop_duplicates(subset=['scan', 'slice'], keep='last')

loaded_images_grouped = pd.merge(loaded_images,
                                  to_add,
                                  on=['scan','slice'],
                                  how = 'left',
                                  validate = 'm:1')

loaded_images_grouped.fillna(0, inplace=True)

loaded_images_grouped.head()

to_add.head()

loaded_images_grouped.info()

loaded_images_grouped['has_tumor'].value_counts().plot(kind='pie');

loaded_images_grouped['has_tumor'].value_counts()

loaded_images_grouped['ID'] = np.arange(len(loaded_images_grouped))

len(loaded_images_grouped)

tumor = loaded_images_grouped[loaded_images_grouped.has_tumor==1]

not_tumor = loaded_images_grouped[loaded_images_grouped.has_tumor==0]

tumor.head()

tumor.shape

not_tumor.shape

from sklearn.utils import resample

tumor_downsampled = resample(tumor,
                              replace=False, # sample with replacement

```

```

        n_samples=750, # match number in majority class

        random_state=12) # reproducible results

not_tumor_downsampled = resample(not_tumor,

        replace=False, # sample with replacement

        n_samples=750, # match number in majority class

        random_state=12) # reproducible results

resampled = pd.concat([tumor_downsampled , not_tumor_downsampled])

resampled.shape

X_resampled = []

for row in resampled.ID:

    X_resampled.append(X[row])

X_resampled = np.array(X_resampled)

X_resampled.shape

y = resampled['has_tumor'].values

y = np.array(y)

y = to_categorical(y, num_classes = 2) # tem nodulo= [0,1], sem nodulo= [1,0]

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y, test_size = 0.3,
random_state=12, stratify=y)

print(X_train.shape, 'train samples')

print(X_test.shape, 'test samples')

print(y_train.shape, 'respostas do treino')

print(y_test.shape, 'respostas do teste')

class RepeatVector(Layer):

    def __init__(self, n, **kwargs):

        super(RepeatVector, self).__init__(**kwargs)

        self.n = n

        self.input_spec = InputSpec(ndim=2)

    def compute_output_shape(self, input_shape):

```

```

        return (input_shape[0], self.n, input_shape[1])

    def call(self, inputs):

        return K.repeat(inputs, self.n)

    def get_config(self):

        config = {'n': self.n}

        base_config = super(RepeatVector, self).get_config()

        return dict(list(base_config.items()) + list(config.items()))

class Lambda(Layer):

    @interfaces.legacy_lambda_support

    def __init__(self, function, output_shape=None,

                  mask=None, arguments=None, **kwargs):

        super(Lambda, self).__init__(**kwargs)

        self.function = function

        self._input_dtypes = None

        self.arguments = arguments if arguments else {}

        if mask is not None:

            self.supports_masking = True

        self.mask = mask

        if output_shape is None:

            self._output_shape = None

        elif isinstance(output_shape, (tuple, list)):

            self._output_shape = tuple(output_shape)

        else:

            if not callable(output_shape):

                raise TypeError('In Lambda, `output_shape` '

                                'must be a list, a tuple, or a function.')

            self._output_shape = output_shape

```



```

def compute_output_shape(self, input_shape):
    if self._output_shape is None:
        if K.backend() in ('tensorflow', 'cntk'):
            if isinstance(input_shape, list):
                xs = [K.placeholder(shape=shape, dtype=dtype)
                       for shape, dtype in zip(input_shape, self._input_dtypes)]
                x = self.call(xs)
            else:
                x = K.placeholder(shape=input_shape, dtype=self._input_dtypes)
                x = self.call(x)
            if isinstance(x, list):
                return [K.int_shape(x_elem) for x_elem in x]
            else:
                return K.int_shape(x)
        warnings.warn("`output_shape` argument not specified for layer {} '
                        'and cannot be automatically inferred '
                        'with the Theano backend. '
                        'Defaulting to output shape `{}` '
                        '(same as input shape). '
                        'If the expected output shape is different, '
                        'specify it via the `output_shape` argument.'
                        .format(self.name, input_shape))
    return input_shape

elif isinstance(self._output_shape, (tuple, list)):
    if isinstance(input_shape, list):
        num_samples = input_shape[0][0]
    else:

```

```

        num_samples = input_shape[0] if input_shape else None
    return (num_samples,) + tuple(self._output_shape)

else:
    shape = self._output_shape(input_shape)
    if not isinstance(shape, (list, tuple)):
        raise ValueError("`output_shape` function must return a tuple or '
                           'a list of tuples.')
    if isinstance(shape, list):
        if isinstance(shape[0], int) or shape[0] is None:
            shape = tuple(shape)
    return shape

def call(self, inputs, mask=None):
    arguments = self.arguments
    if has_arg(self.function, 'mask'):
        arguments['mask'] = mask
    if isinstance(inputs, list):
        self._input_dtypes = [K.dtype(x) for x in inputs]
    else:
        self._input_dtypes = K.dtype(inputs)
    return self.function(inputs, **arguments)

def compute_mask(self, inputs, mask=None):
    if callable(self.mask):
        return self.mask(inputs, mask)
    return self.mask

def get_config(self):
    if isinstance(self.function, python_types.LambdaType):
        function = func_dump(self.function)

```

```

        function_type = 'lambda'
    else:
        function = self.function.__name__
        function_type = 'function'
    if isinstance(self._output_shape, python_types.LambdaType):
        output_shape = func_dump(self._output_shape)
        output_shape_type = 'lambda'
    elif callable(self._output_shape):
        output_shape = self._output_shape.__name__
        output_shape_type = 'function'
    else:
        output_shape = self._output_shape
        output_shape_type = 'raw'
    config = {'function': function,
             'function_type': function_type,
             'output_shape': output_shape,
             'output_shape_type': output_shape_type,
             'arguments': self.arguments}

    base_config = super(Lambda, self).get_config()

    return dict(list(base_config.items()) + list(config.items()))

    @classmethod
    def from_config(cls, config, custom_objects=None):
        config = config.copy()

        globs = globals()

        if custom_objects:
            globs = dict(list(globs.items()) + list(custom_objects.items()))

        function_type = config.pop('function_type')

```

```

if function_type == 'function':
    function = deserialize_keras_object(
        config['function'],
        custom_objects=custom_objects,
        printable_module_name='function in Lambda layer')
elif function_type == 'lambda':
    function = func_load(config['function'], globs=globs)
else:
    raise TypeError('Unknown function type:', function_type)
output_shape_type = config.pop('output_shape_type')
if output_shape_type == 'function':
    output_shape = deserialize_keras_object(
        config['output_shape'],
        custom_objects=custom_objects,
        printable_module_name='output_shape function in Lambda layer')
elif output_shape_type == 'lambda':
    output_shape = func_load(config['output_shape'], globs=globs)
else:
    output_shape = config['output_shape']
if 'arguments' in config:
    for key in config['arguments']:
        if isinstance(config['arguments'][key], dict):
            arg_dict = config['arguments'][key]
            if 'type' in arg_dict and arg_dict['type'] == 'ndarray':
                config['arguments'][key] = np.array(arg_dict['value'])
config['function'] = function
config['output_shape'] = output_shape

```

```

        return cls(**config)

class Dense(Layer):
    @interfaces.legacy_dense_support
    def __init__(self, units,
                  activation=None,
                  use_bias=True,
                  kernel_initializer='glorot_uniform',
                  bias_initializer='zeros',
                  kernel_regularizer=None,
                  bias_regularizer=None,
                  activity_regularizer=None,
                  kernel_constraint=None,
                  bias_constraint=None,
                  **kwargs):
        if 'input_shape' not in kwargs and 'input_dim' in kwargs:
            kwargs['input_shape'] = (kwargs.pop('input_dim'),)
        super(Dense, self).__init__(**kwargs)
        self.units = units
        self.activation = activations.get(activation)
        self.use_bias = use_bias
        self.kernel_initializer = initializers.get(kernel_initializer)
        self.bias_initializer = initializers.get(bias_initializer)
        self.kernel_regularizer = regularizers.get(kernel_regularizer)
        self.bias_regularizer = regularizers.get(bias_regularizer)
        self.activity_regularizer = regularizers.get(activity_regularizer)
        self.kernel_constraint = constraints.get(kernel_constraint)
        self.bias_constraint = constraints.get(bias_constraint)

```

```

self.input_spec = InputSpec(min_ndim=2)

self.supports_masking = True

def build(self, input_shape):
    assert len(input_shape) >= 2

    input_dim = input_shape[-1]

    self.kernel = self.add_weight(shape=(input_dim, self.units),
                                   initializer=self.kernel_initializer,
                                   name='kernel',
                                   regularizer=self.kernel_regularizer,
                                   constraint=self.kernel_constraint)

    if self.use_bias:
        self.bias = self.add_weight(shape=(self.units,),
                                     initializer=self.bias_initializer,
                                     name='bias',
                                     regularizer=self.bias_regularizer,
                                     constraint=self.bias_constraint)

    else:
        self.bias = None

    self.input_spec = InputSpec(min_ndim=2, axes={-1: input_dim})

    self.built = True

def call(self, inputs):
    output = K.dot(inputs, self.kernel)

    if self.use_bias:
        output = K.bias_add(output, self.bias, data_format='channels_last')

    if self.activation is not None:
        output = self.activation(output)

    return output

```

```

def compute_output_shape(self, input_shape):
    assert input_shape and len(input_shape) >= 2
    assert input_shape[-1]
    output_shape = list(input_shape)
    output_shape[-1] = self.units
    return tuple(output_shape)

def get_config(self):
    config = {
        'units': self.units,
        'activation': activations.serialize(self.activation),
        'use_bias': self.use_bias,
        'kernel_initializer': initializers.serialize(self.kernel_initializer),
        'bias_initializer': initializers.serialize(self.bias_initializer),
        'kernel_regularizer': regularizers.serialize(self.kernel_regularizer),
        'bias_regularizer': regularizers.serialize(self.bias_regularizer),
        'activity_regularizer':
            regularizers.serialize(self.activity_regularizer),
        'kernel_constraint': constraints.serialize(self.kernel_constraint),
        'bias_constraint': constraints.serialize(self.bias_constraint)
    }
    base_config = super(Dense, self).get_config()
    return dict(list(base_config.items()) + list(config.items()))

class ActivityRegularization(Layer):
    def __init__(self, l1=0., l2=0., **kwargs):
        super(ActivityRegularization, self).__init__(**kwargs)
        self.supports_masking = True
        self.l1 = l1

```

```

self.l2 = l2

self.activity_regularizer = regularizers.L1L2(l1=l1, l2=l2)

def get_config(self):
    config = {'l1': self.l1,
              'l2': self.l2}

    base_config = super(ActivityRegularization, self).get_config()

    return dict(list(base_config.items()) + list(config.items()))

def compute_output_shape(self, input_shape):
    return input_shape

model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same',
                  activation = 'relu', input_shape = (100,100,1)))

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                  activation = 'relu'))

model.add(MaxPool2D(pool_size=(2,2)))

model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                  activation = 'relu'))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                  activation = 'relu'))

model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation = "relu"))

model.add(Dropout(0.5))

model.add(Dense(2, activation = "softmax"))

model.summary()

```



```

optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

# Compile the model

model.compile(optimizer = optimizer

               , loss = "categorical_crossentropy"

               , metrics=["accuracy"])

# Set a learning rate

learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',

                                             patience=3,

                                             verbose=0,

                                             factor=0.5,

                                             min_lr=0.00001)

epochs = 50

batch_size = 40

datagen = ImageDataGenerator(

    featurewise_center=False, # set input mean to 0 over the dataset

    samplewise_center=False, # set each sample mean to 0

    featurewise_std_normalization=False, # divide inputs by std of the dataset

    samplewise_std_normalization=False, # divide each input by its std

    zca_whitening=False, # apply ZCA whitening

    rotation_range=3, # randomly rotate images in the range (degrees, 0 to 180)

    zoom_range = 0.05, # Randomly zoom image

    width_shift_range=False, # randomly shift images horizontally (fraction of total width)

    height_shift_range=False, # randomly shift images vertically (fraction of total height)

    horizontal_flip=False, # randomly flip images

    vertical_flip=False) # randomly flip images

datagen.fit(X_train)

history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batch_size),

```

```

        epochs = epochs,

        validation_data = (X_test,y_test),

        verbose = 0,

        steps_per_epoch=X_train.shape[0] // batch_size

        , callbacks=[learning_rate_reduction])

fig, ax = plt.subplots(2,1, figsize=(9, 5))

ax[0].plot(history.history['loss'], color='b', label="Training loss")

ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])

legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['acc'], color='b', label="Training accuracy")

ax[1].plot(history.history['val_acc'], color='r',label="Validation accuracy")

legend = ax[1].legend(loc='best', shadow=True)

predictions = model.predict_classes(X_test)

print('Acuracidade do modelo: ',round(accuracy_score(predictions,np.argmax(y_test, axis =
1))*100,2), '%')

print('Demais Indicadores: \n')

print(' recall_score: ',round(recall_score(predictions,np.argmax(y_test, axis = 1)),2))

print(' precision_score: ',round(precision_score(predictions,np.argmax(y_test, axis = 1)),2))

print(' f1_score: ',round(f1_score(predictions,np.argmax(y_test, axis = 1)),2))

print(' roc: ',round(roc_auc_score(predictions,np.argmax(y_test, axis = 1)),2))

print(classification_report(predictions,np.argmax(y_test, axis = 1)))

def plot_confusion_matrix(cm, target_names, title='Confusion matrix'):

    accuracy = np.trace(cm) / float(np.sum(cm))

    misclass = 1 - accuracy

    cmap = plt.get_cmap('Blues')

    plt.rcParams.update(plt.rcParamsDefault)

    %matplotlib inline

    #plt.figure(figsize=(8, 6))

```

```

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

if target_names is not None:

    tick_marks = np.arange(len(target_names))

    plt.xticks(tick_marks, target_names, rotation=45)

    plt.yticks(tick_marks, target_names)

thresh = cm.max() / 2

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

    plt.text(j, i, "{:,}".format(cm[i, j]),

             horizontalalignment="center",

             color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label\naccuracy={:0.3f}; misclass={:0.3f}'.format(accuracy, misclass))

plt.show()

plot_confusion_matrix(confusion_matrix(predictions,np.argmax(y_test, axis =
1)),target_names=['tumor','not_tumor'])

# predict probabilities

probs = model.predict(X_test)

# keep probabilities for the positive outcome only

probs = probs[:, 1]

# calculate AUC

auc = roc_auc_score(np.argmax(y_test, axis = 1), probs)

print('AUC: %.3f' % auc)

# calculate roc curve

fpr, tpr, thresholds = roc_curve(np.argmax(y_test, axis = 1), probs)

# plot

```

```

plt.figure()

lw = 2

plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve (area = %0.2f)' % auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()

# Salvando os pesos do modelo com bom desempenho
model.save_weights('model_weights_last.h5')

# Salvando o modelo em si
model.save('my_model_last')

# save as JSON
import json

json_string = model.to_json()

with open('model.json','w') as f:
    json.dump(json_string, f)

# para carregar
'''

model.load_weights('my_model_weights.h5')

from keras.models import load_model

new_model = load_model('my_model')

''';

```

```

# Display some error results

# Errors are difference between predicted labels and true labels
errors = (predictions - np.argmax(y_test, axis=1) != 0)

print("De um total de",len(predictions), "errou", sum(errors), "vezes")

# Dados de entrada dos previstos errados
X_test[errors].shape

# Ajustando dimensões para uma imagem
X_test[errors][37].reshape(100,100).shape

# Acessando os originais
np.argmax(y_test[errors], axis=1).shape

# Acessando a resposta original de alguma previsão
np.argmax(y_test[errors], axis=1)[1]

final_table = pd.DataFrame({'previsao': predictions, 'real':np.argmax(y_test, axis=1)})

print('falso positivo')

final_table[final_table['previsao'] > final_table['real']]

print('falso negatvo')

final_table[final_table['previsao'] < final_table['real']]

import matplotlib.image as mpimg

import seaborn as sns

import cv2

sns.set(style='white', context='notebook', palette='deep')

print('Imagens que o sistema previu errado, com sua resposta original \n')

fig, ax = plt.subplots(2,3, figsize=(28, 18), sharex=True,sharey=True)

ax[0,0].imshow(X_test[14].reshape(100,100))

ax[0,0].set_title("Exemplo de Falso Positivo", fontsize=30)

ax[0,1].imshow(X_test[449].reshape(100,100))

ax[0,1].set_title("Exemplo de Falso Positivo", fontsize=30)

```

```

ax[0,2].imshow(X_test[434].reshape(100,100))
ax[0,2].set_title("Exemplo de Falso Positivo", fontsize=30)
ax[1,1].imshow(X_test[51].reshape(100,100))
ax[1,1].set_title("Exemplo de Falso Negativo", fontsize=30)
ax[1,0].imshow(X_test[321].reshape(100,100))
ax[1,0].set_title("Exemplo de Falso Negativo", fontsize=30)
ax[1,2].imshow(X_test[417].reshape(100,100))
ax[1,2].set_title("Exemplo de Falso Negativo", fontsize=30)
ax[0,0].axis('off')
ax[1,0].axis('off')
ax[1,2].axis('off')
ax[0,1].axis('off')
ax[1,1].axis('off')
ax[0,2].axis('off')
plt.show()

# Display some correct results
acertos = (predictions - np.argmax(y_test, axis=1) == 0)
print("De um total de",len(predictions), "acertou", sum(acertos), "vezes")
final_table = pd.DataFrame({'previsao': predictions, 'real':np.argmax(y_test, axis=1)})
print('trues')
final_table[final_table['previsao'] == final_table['real']]

import matplotlib.image as mpimg
import seaborn as sns
import cv2

sns.set(style='white', context='notebook', palette='deep')

print('Imagens que o sistema previu zero, com sua resposta original \n')
fig, ax = plt.subplots(2,3, figsize=(28, 18), sharex=True,sharey=True)

```

```

ax[0,0].imshow(X_test[0].reshape(100,100))
ax[0,0].set_title("Exemplo de Verdadeiro Positivo", fontsize=30)
ax[0,1].imshow(X_test[2].reshape(100,100))
ax[0,1].set_title("Exemplo de Verdadeiro Positivo", fontsize=30)
ax[0,2].imshow(X_test[4].reshape(100,100))
ax[0,2].set_title("Exemplo de Verdadeiro Positivo", fontsize=30)
ax[1,1].imshow(X_test[17].reshape(100,100))
ax[1,1].set_title("Exemplo de Verdadeiro Negativo", fontsize=30)
ax[1,0].imshow(X_test[440].reshape(100,100))
ax[1,0].set_title("Exemplo de Verdadeiro Negativo", fontsize=30)
ax[1,2].imshow(X_test[429].reshape(100,100))
ax[1,2].set_title("Exemplo de Verdadeiro Negativo", fontsize=30)
ax[0,0].axis('off')
ax[1,0].axis('off')
ax[1,2].axis('off')
ax[0,1].axis('off')
ax[1,1].axis('off')
ax[0,2].axis('off')
plt.show()

to_iterate = final_table[final_table['previsao'] == final_table['real']]
for i in to_iterate.index:
    print(i)

import shutil
import random

def leitura_paste(path):
    cont_nodulo = 0
    cont_nonolo = 0

```

```

url_base = path
mylist = np.arange(50)
for file_name in os.listdir(url_base):
    for file2 in os.listdir(url_base+'/'+file_name):
        for file3 in os.listdir(url_base+'/'+file_name+'/'+file2):
            for file4 in os.listdir(url_base+'/'+file_name+'/'+file2+'/'+file3):
                if file4.endswith('.dcm'):
                    if random.choices(mylist) == [7]:
                        img = im.imread(os.path.join(url_base, file_name, file2, file3,
file4)).astype(np.float64)
                        img_dic = img.meta
                        scan = str(img_dic['SeriesNumber'])
                        slice_n = str(img_dic['InstanceNumber'])
                        referencia = scan+slice_n
                        if to_add[to_add['referencia'] == referencia]['has_tumor'].any():
                            pass
                            #cont_nodulo +=1
                            #if cont_nodulo < 20:
                                #shutil.copy2(os.path.join(url_base, file_name, file2, file3, file4),
'nodulo')
                        else:
                            cont_nonolo +=1
                            if cont_nonolo < 25:
                                shutil.copy2(os.path.join(url_base, file_name, file2, file3, file4), 'sem
nodulo')
mylist = np.arange(50)
if to_add[to_add['referencia'] == '300062277']['has_tumor'].any():
    print(True)

```



```

min_lr=0.00001)

epochs = 50

history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batch_size),

                               epochs = epochs,

                               validation_data = (X_test,y_test),

                               verbose = 0,

                               steps_per_epoch=X_train.shape[0] // batch_size

                               , callbacks=[learning_rate_reduction])

fig, ax = plt.subplots(2,1, figsize=(9, 5))

ax[0].plot(history.history['loss'], color='b', label="Training loss")

ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])

legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['acc'], color='b', label="Training accuracy")

ax[1].plot(history.history['val_acc'], color='r',label="Validation accuracy")

legend = ax[1].legend(loc='best', shadow=True)

predictions = model.predict_classes(X_test)

print('Acuracidade do modelo: ',round(accuracy_score(predictions,np.argmax(y_test,
axis = 1))*100,2), '%')

print('Demais Indicadores: \n')

print(' recall_score: ',round(recall_score(predictions,np.argmax(y_test, axis = 1)),2))

print(' precision_score: ',round(precision_score(predictions,np.argmax(y_test, axis =
1)),2))

print(' f1_score: ',round(f1_score(predictions,np.argmax(y_test, axis = 1)),2))

print(' roc: ',round(roc_auc_score(predictions,np.argmax(y_test, axis = 1)),2))

return (round(accuracy_score(predictions,np.argmax(y_test, axis = 1))*100,2), '%'),
round(recall_score(predictions,np.argmax(y_test, axis = 1)),2)), predictions )

lista_final = []

while True:

```

```
modelo_string = one_search()
acc, rec, results = loop_master(modelo_string)
if acc > 0.85:
    if rec > 0.90:
        lista_final.append((acc,rec,results))
if len(lista_final) == 3:
    break
```