TRABALHO DE GRADUAÇÃO 2

DETECÇÃO DE INSTRUMENTOS CIRÚRGICOS EM IMAGENS DE LAPAROSCOPIA UTILIZANDO MACHINE LEARNING.

Matheus Escovedo da Costa

Brasília, Outubro de 2021

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO 2

DETECÇÃO DE INSTRUMENTOS CIRÚRGICOS EM IMAGENS DE LAPAROSCOPIA UTILIZANDO MACHINE LEARNING.

Matheus Escovedo da Costa

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Mariana Costa Bernardes, ENE/UnB _____
Orientador

Prof. Cláudia Patrícia Ochoa Diaz, FGA/UnB _____
Examinadora Interna

Prof. Roberto de Souza Baptista, FGA/UnB _____
Examinador Interno

Brasília, Outubro de 2021

FICHA CATALOGRÁFICA

MATHEUS, DA COSTA
Detecção de instrumentos cirúrgicos em imagens de laparoscopia utilizando machine learning,

[Distrito Federal] 2021.

x, 62p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação 1 – Universidade de Brasília.Faculdade de Tecnologia.

1. Machine Learning 2.Laparoscopia
3.CNN 4.Instrumentos Cirúrgicos

I. Mecatrônica/FT/UnB II. Detecção de instrumentos cirúrgicos em imagens de laparoscopia utilizando machine learning.

REFERÊNCIA BIBLIOGRÁFICA

COSTA, MATHEUS ESCOVEDO DA, (2021). Detecção de instrumentos cirúrgicos em imagens de laparoscopia utilizando machine learning. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 06, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 62p.

CESSÃO DE DIREITOS

AUTOR: Matheus Escovedo da Costa

TÍTULO DO TRABALHO DE GRADUAÇÃO: Detecção de instrumentos cirúrgicos em imagens de laparoscopia utilizando machine learning.

GRAU: Engenheiro ANO: 2021

Matheus Escovedo da Costa

Qsc 17, nº 21, Taguatinga Sul.

72016-170 Brasília – DF – Brasil.

RESUMO

A laparoscopia é definida como a visualização telescópica da cavidade abdominal, por meio da técnica operatória videocirurgia (SILVA, 2006). Contribuiu de forma significativa para os avanços tecnológicos relacionados a intervenções cirúrgicas abdominais, como a apendicectomia (retirada total ou parcial do apêndice), colecistectomia (retirada total ou parcial da vesícula biliar), cirurgias ginecológicas, a hernioplastia, entre outros (NETO, 2003). Apresenta diversas vantagens sobre as cirurgias abertas como redução do tempo de internação, da morbidade pós-operatória, das complicações da ferida cirúrgica, pequenas incisões, além de permitir recuperação e retorno precoce às atividades, evidenciando a menor agressividade desse método cirúrgico (NETO, 2003). Geralmente é necessário o mínimo de dois médicos para a ação da laparoscopia, um médico é responsável pelos instrumentos cirúrgicos e o outro fica manipulando a câmera. Uma área de pesquisa que vêm crescendo é a de automação de robôs cirúrgicos que possam manipular a câmera e até mesmo realizar alguns tipos de operações, como por exemplo (CHOI,2017; CHENG,2020), toda via , devido a limitação em espaço, visão e feedback, estes robôs são muito propensos a erros, do qual, em uma cirurgia poderia causar lesões ao paciente, prejudicar os equipamentos, dentre outros. Uma proposta de solução para este problema é a aplicação de algoritmos de machine learning para que o robô possa ter um feedback mais consistente da posição dos equipamentos cirúrgicos e das dimensões internas, de modo que o seu desempenho seja elevado suficientemente para a aplicação destes robôs em cirurgias de laparoscopia. Portanto, este trabalho tem o objetivo de gerar um algoritmo com o auxilio de machine learning capaz de identificar instrumentos cirúrgicos em imagens de laparoscopia, para isso, foi utilizado redes neurais convolucionais como modelo de machine learning. Os trabalhos de (PEREZ,2020) e (RONNEBERGER,2015) são as principais fontes de inspiração, o banco de imagens disponibilizado por (PEREZ,2020) permitiu um treinamento com imagens simuladas pela plataforma unity 3D, de modo que, foi obtido como resultado dois modelos de redes neurais: Uma treinada com imagens mais simples, do qual, não foi capaz de identificar os aparelhos cirurgicos em imagens reais. O segundo modelo, treinado com imagens mais complexas, é capaz de destacar o aparelho cirurgico do restanto da imagem.

Palavras Chave: Machine Learning, CNN, Instrumentos Cirúrgicos, Laparoscopia

ABSTRACT

Laparoscopy is defined as the telescopic visualization of the abdominal cavity, through the surgical technique of videosurgery (SILVA, 2006). It has significantly contributed to technological advances related to abdominal surgical treatments, such as appendectomy (total or partial removal of the appendix), cholecystectomy (total or partial removal of the gallbladder), gynecological surgeries, hernioplasty, among others (NETO, 2003) . Several advantages over open surgery, such as reduced hospital stay, postoperative morbidity, surgical wound complications, small incisions, in addition to allowing early recovery and return to activities, evidencing the less aggressiveness of this surgical method (NETO, 2003) A minimum of two doctors is required for the laparoscopy action, one doctor is responsible for the surgical instruments and the other is handling the camera. A growing area of research is the automation of surgical robots that can manipulate a camera and even perform some types of operations, such as (CHOI,2017; CHENG,2020), however, due to space limitations , vision and feedback, these robots are very prone to errors, which, in a surgery, could cause injuries to the patient, damage the equipment, among others. A proposed solution to this problem is an application of machine learning algorithms so that the robot can have more consistent feedback on the position of surgical equipment and internal dimensions, so that its performance is sufficiently high for their application. robots laparoscopy surgeries. Therefore, this work aims to generate an algorithm with the aid of machine learning, capable of identifying surgical instruments in laparoscopy images. It was used a convolutional neural network as a machine learning model. The works by (PEREZ, 2020) and (RONNEBERGER, 2015) are the main sources of inspiration, the image bank provided by (PEREZ,2020) allowed a training with images, which was simulated by the unity 3D platform. The result was two models of neural networks: One trained with simpler images, which was not able to identify the surgical devices in real images. The second model, trained with more complex images, is able to detach the surgical apparatus from the rest of the image.

Keywords: Machine Learning, CNN, Laparoscopy, Surgical Instruments

SUMÁRIO

Capítulo 1

Introdução

1.1 Contextualização

A laparoscopia é definida como a visualização telescópica da cavidade abdominal, por meio da técnica operatória videocirurgia (SILVA, 2006). Contribuiu de forma significativa para os avanços tecnológicos relacionados a intervenções cirúrgicas abdominais, como a apendicectomia (retirada total ou parcial do apêndice), colecistectomia (retirada total ou parcial da vesícula biliar), cirurgias ginecológicas, a hernioplastia, entre outros (NETO, 2003). Apresenta diversas vantagens sobre as cirurgias abertas como redução do tempo de internação, da morbidade pós-operatória, das complicações da ferida cirúrgica, pequenas incisões, além de permitir recuperação e retorno precoce às atividades, evidenciando a menor agressividade desse método cirúrgico (NETO, 2003).

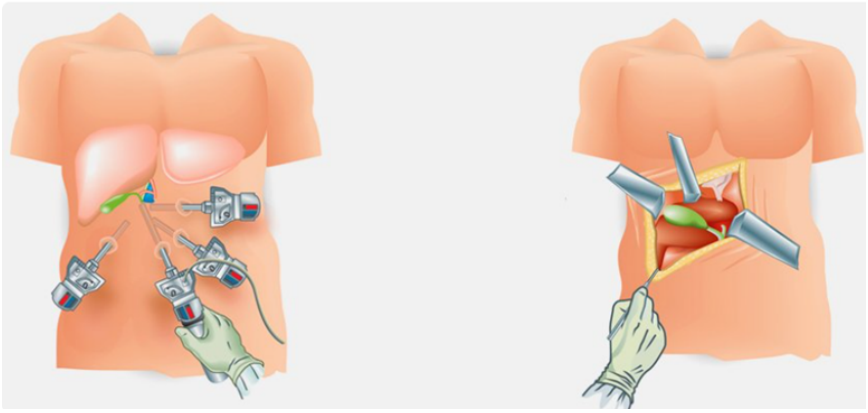

Figura 1.1: Diferenças entre ato cirurgico de laparoscopia com o método convencional. Fonte: RSAUDE. (2018)

1.2 Definição do problema

Geralmente é necessário o mínimo de dois médicos para a ação da laparoscopia, um médico é responsável pelos instrumentos cirúrgicos e o outro fica manipulando a câmera. Uma área de pesquisa que vêm crescendo é a de automação de robôs cirúrgicos que possam manipular a câmera e até mesmo realizar alguns tipos de operações, como por exemplo (CHOI,2017; CHENG,2020). Com a atuação do robô é descartada a necessidade de dois médicos, sendo necessário apenas um, dessa forma, os principais beneficios são:

Baratear o custo, uma vez que o custo do médico-hora (Mh) no Brasil tem o piso de 80 R$ (FENAM,2021), ja o custo médio de kilowatts-hora (kWh) em 2021 é de 0,90 R$ (ANEEL,2021), portanto, o custo por hora tem o potencial de redução de até 88 vezes. Além da redução do custo, uma vez automatizado e garantido a robustez do sistema, o robô será muito menos propicio a erro e estará sempre a disposição, de forma que, independente do horário de chegada do paciente o robô sempre estara disponivel para atuar na operação, desde que já não esteja em uso.

Todavia, desenvolver um robô desta proporção com a garantia total de seu funcionamento não é trivial, devido a limitação em espaço, visão e feedback, estes robôs são muito propensos a erros, do qual, em uma cirurgia poderia causar lesões ao paciente, prejudicar os equipamentos, dentre outros. É fundamental que todo o seu processo de desenvolvimento garanta alta confiabilidade em seu funcionamento.

Uma proposta de solução para este problema é a aplicação de algoritmos de machine learning para que o robô possa ter um feedback mais consistente da posição dos equipamentos cirúrgicos e das dimensões internas, de modo que o seu desempenho seja elevado suficientemente para a aplicação destes robôs em cirurgias de laparoscopia.

Figura 1.2: Robô cirúrgico realizando operação de forma automatizada. Fonte: SIMES. (2017)

1.3 Objetivos do projeto

1.3.1 Objetivo Geral

Este trabalho tem o objetivo de realizar um algoritmo via machine learning para a identificação de instrumentos cirúrgicos, do qual, atuará como a primeira etapa para a elaboração do feedback para a automação do robô cirurgico. O trabalho é separado em etapas para facilitar o entendimento das decisões tomadas:

1.3.2 Objetivos Específicos

- 1ª Etapa: Encontrar trabalhos publicados com objetivo de pesquisa igual ou semelhante, assim, levantando um banco de referências que facilitam o entendimento do problema.
- 2º Etapa: Com o banco de referências, definir qual tipo de algoritmo de machine learning será utilizado e qual artigo será utilizado como maior inspiração visando a replicabilidade do processo de elaboração.
- 3ª Etapa: Realização do algoritmo seguido de testes para validação de sua eficácia.

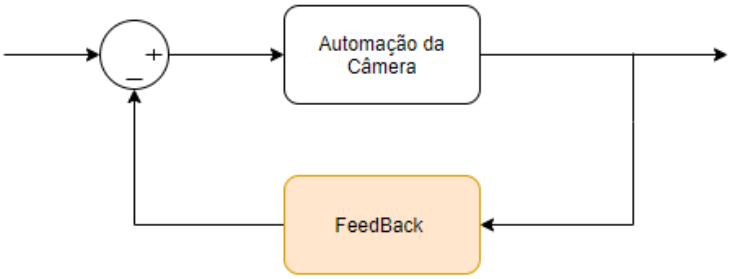

Figura 1.3: Modelo de automação com feedback. Fonte: Autor. (2021)

Capítulo 2

Fundamentos

2.1 Introdução

2.1.1 Machine Learning

Atualmente, quase tudo que se faz costuma deixar um "rastro digital", quando se compra algo, por exemplo, fica registrado o horário, o local, quanto foi gasto, nome do produto, dentre outros. Quase todos os tipos de máquinas atuais são conectados a um ou mais banco de armazenamento de dados, armazenando todas as informações do serviço realizado. (CHOI,2017)

Com os dados coletados, as pessoas elaboravam algoritmos com as informações relevantes que são aplicadas em computadores, todavia, como o número de dados aumentou, graças ao avanço no modo de armazenamento, seres humanos não são mais capazes de interpretar tamanha quantidade de informações. Assim, o método bastante abordado atualmente é entregar todos os dados ao computador e deixar que ele aprenda por si mesmo sem que haja um algoritmo que diga explicitamente o que fazer com os dados, este método é o principio do Machine Learning. (CHOI,2017)

Machine Learning (ML) é a área dentro da ciência da computação onde o computador utiliza padrões dentro dos dados processados para realizar predições, geralmente um algoritmo de ML é definido em 5 etapas, que são descritas a seguir:

1ª Etapa: Coleta e tratamento de dados. Essa etapa talvez seja a mais importante, pois determina o quão bom um modelo será em realizar predições. Muitas vezes os dados se encontram em formato inapropriado ou com informações desnecessárias, sendo responsabilidade do programador definir o que é de fato necessário e deixar a lista de dados de forma adequada para o seu modelo.

2ª Etapa: Build do Modelo. Nesta etapa é definido qual o tipo de modelo do ML será utilizado, após isso, é ajustado os seus parâmetros na busca do melhor desempenho possível, é bastante recorrente que essa etapa seja refeita diversas vezes, uma vez que não se sabe exatamente qual comportamento o ML terá, após a etapa de teste, em caso de um resultado não adequado, esta etapa será refeita.

3ª Etapa: Treino. Nesta etapa é aonde o modelo "aprende", é inserida uma quantidade de dados, os dados tratados na etapa 1ª são separados em dados de treino e dados de teste, geralmente a porcentagem para treino é bem maior que a de teste, por exemplo, 70% para treino e 30% para teste. Processando os dados de teste, o modelo vai praticando sua predição enquanto vai ajustando os seus parâmetros internos na busca da melhor porcentagem de acertos.

4ª Etapa: Validação. Nesta etapa o modelo realiza a predição de dados que ele não viu na etapa de treino, para garantir que sua assertividade não está atrelada a apenas os dados utilizados para modelar o sistema, em caso de uma boa porcentagem de acertos o modelo é aprovado, caso haja muitos erros em suas predições é uma pratica comum retornar a etapa 2 para ajustar novamente os parâmetros do modelo e continuar a partir dai.

5ª Etapa: Predição. Com o modelo tendo resultados satisfatórios em seus testes ele está pronto para realizar as predições na aplicação, do qual, foi modelado para realizar.

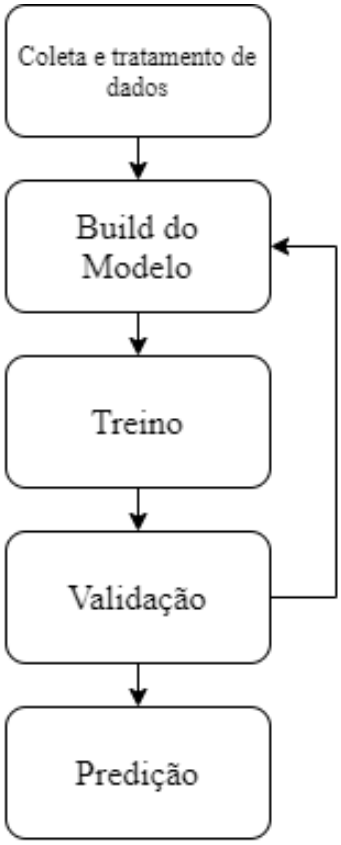

Figura 2.1: Etapas de construção de modelo genérico de Machine Learning. Fonte: Autor.

Aplicado em quase todas as áreas, ML se tornou uma ferramenta bastante conveniente graças a sua acessibilidade e fácil uso em geral, por exemplo, existem algoritmos com o uso de ML capazes de identificar a atenção do aluno na aula, assim, o professor poderá ter um maior feedback sobre quais técnicas utilizar para abordar em cada assunto. (LITJENS,2017). Como o processamento de dados pode exigir demais de computadores de uso comum, existem plataformas como o google colab que disponibilizam unidades de processamento gráfico (GPU) para o público em geral,

além de diversos cursos orientando a como utilizar ML, assim, incentivando a pesquisa em ML e inteligência artificial. (COLAB,2021).

Através das bases de dados científicos **Scopus** e **Periódico Capes**, foi realizado um levantamento sobre a produção cientifica mundial envolvendo ML. A busca em ambas as bases foi feita com a string: "Machine Learning", no Scopus foi encontrado 295766 publicações entre os anos de 1973 e 2021, a figura 2.2 evidencia como a pesquisa em ML vem crescendo de maneira exponencial, chegando a 68225 publicações no ano de 2020. A figura 2.3 mostra a porcentagem de publicações por área de pesquisa, as duas maiores porcentagens pertencem a ciência da computação e a engenharia, todavia, vale ressaltar que a medicina está em 3ª com 5,5%, trabalhos como o (RAVAUT, 2021), do qual, realiza a predição de resultados adversos oriundos de complicações da diabetes via ML mostram como a área da saúde em geral vem se aproveitando bastante dos avanços do ML.

A figura 2.4 relata os países com maior número de publicações, sendo o Estados Unidos o primeiro lugar, o Brasil estando apenas em 13ª, mostra como a pesquisa no país ainda está muito atrás das grandes potências na área. A figura 2.5 é o resultado da pesquisa no periódico Capes novamente evidenciando o crescimento exponencial na área, com 717712 publicações encontradas entre os anos de 2003 a 2020, sendo 138217 somente no ano de 2020.

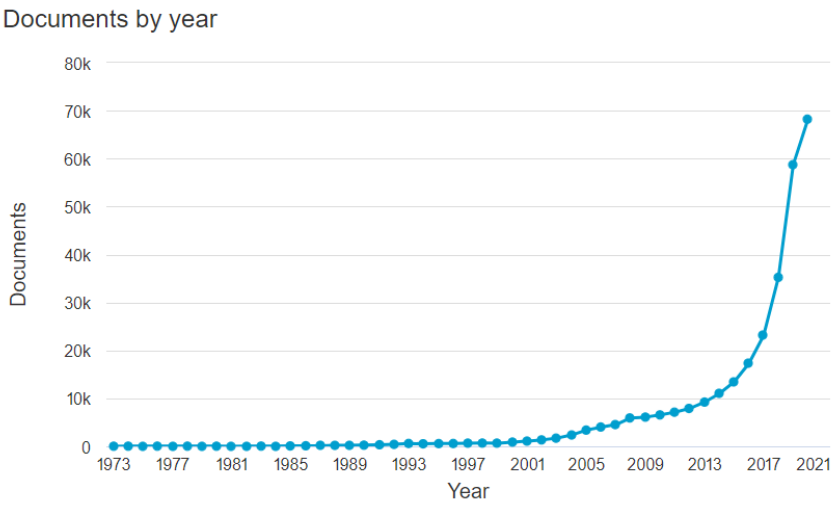

Figura 2.2: Número de publicações por ano. Fonte: Scopus (2021)

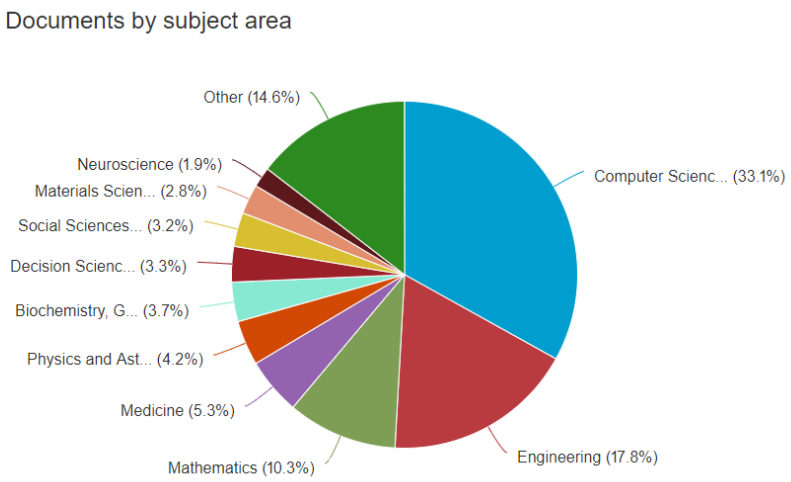

Figura 2.3: Porcentagem de publicações por área de pesquisa. Fonte: Scopus (2021).

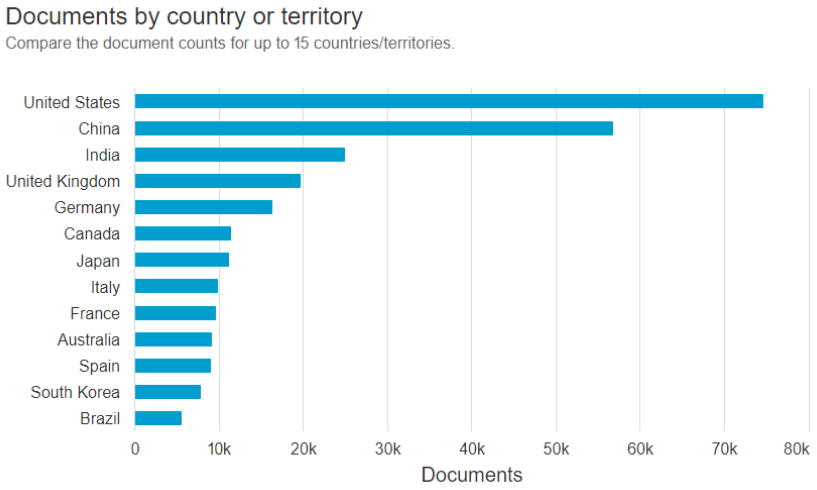

Figura 2.4: Número de publicações por País. Fonte: Scopus (2021).

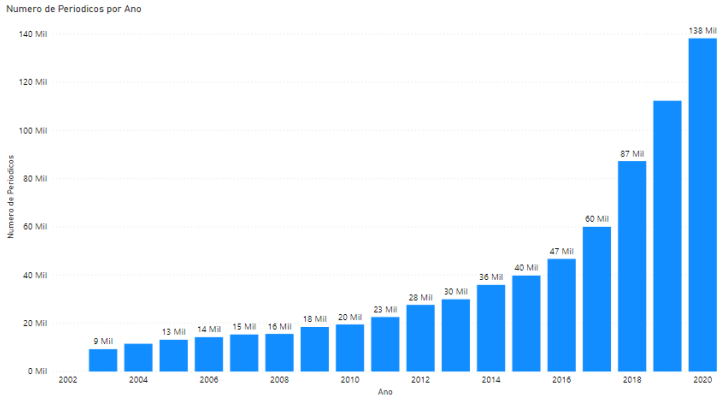

Figura 2.5: Número de publicações por ano. Fonte: Periódico Capes (2021).

2.2 Fundamentação

2.2.1 Machine Learning e Laparoscopia

Para levantar um banco de referências foi utilizado 3 bases de dados disponibilizadas pela UnB, as bases são: Periódicos Capes, IEE Explore e Scopus. As três bases podem ser acessadas pelo método de comunidade acadêmica federada (CAFE), onde o portal periódico capes em parceria com a UnB permite que os alunos da instituição acessem remotamente base de dados conceituadas, sendo o próprio periódico capes uma grande base. Em cada base foi realizado 3 buscas, em cada uma, o autor deste trabalho tomou a decisão de descartar qualquer trabalho em que o título não deixasse claro o envolvimento com o tema, após isso, os trabalhos foram selecionados por seu resumo, com o mesmo critério para descarte, e por fim foi feito a leitura completa dos artigos restantes deixando na tabela 1, em anexo, apenas os que de fato poderiam agregar ao trabalho.

As strings de busca foram:

- **LAPAROSCOPY AND (MACHINE LEARNING).**
- **LAPAROSCOPY AND (ARTIFICIAL INTELLIGENCE).**
- **LAPAROSCOPY AND (MACHINE LEARNING) AND (TRACK*).**

Critérios de Seleção:

- **Título: Coerente com o tema.**
- **Resumo: Claro envolvimento com o tema.**
- **Leitura Completa:**

2.2.2 Escolha do Método

A tabela 1, em anexo, evidencia como esta área de pesquisa é recente, 78% dos trabalhos são de 2019 a 2021. Dentre os trabalhos selecionados o método de machine learning aplicado é quase unânime para Rede Neurais Convolucionais (CNN) ou alguma de suas variações, como a Rede Neural Convolucional baseada em região (R-CNN), portanto, o método aplicado neste trabalho será CNN uma vez que diversos relatos comprovam a sua eficácia e grande possibilidade de aplicação.

2.2.3 CNN e Saúde

Realizando uma busca no SCOPUS, com a seguinte string de busca, a seguir:

- **(((convolutional AND neural AND network) OR cnn) AND (health OR medicine))**

Foi encontrado o resultado de 4534 de trabalhos publicados, a figura 2.8 deixa claro como desde 2012, grande estopim da CNN a nível mundial, as pesquisas utilizando as CNN's para soluções na área de saúde vem crescendo exponencialmente. A figura 2.9 mostra um resultado alarmante de como o Brasil está atrasado nesta área, se comparado com a China com 997 publicações, 1ª lugar, o Brasil publicou apenas 76, mais de 13 vezes menos. Portanto, trabalhos como este são fundamentais para promover a pesquisa do país.

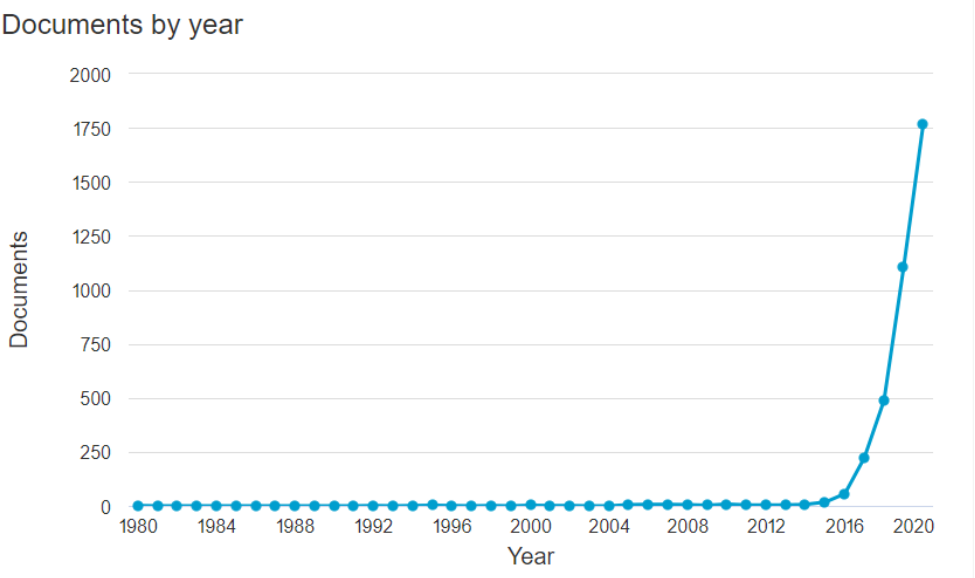

Figura 2.6: Número de publicações por ano. Fonte: Scopus. (2021).

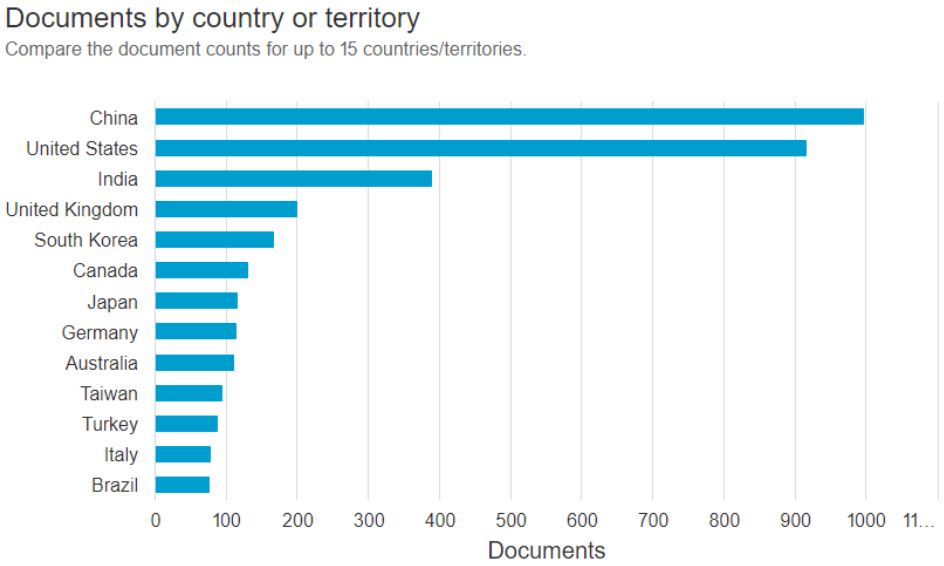

Figura 2.7: Número de publicações por país. Fonte: Scopus. (2021).

2.2.4 CNN

Uma rede neural convolucional (CNN) é uma classe de redes neurais profundas, do qual, costuma ser aplicado em análise de imagens. CNNs são formadas por diversas unidades de processamento, comumente chamadas de neurônios, são uma versão de perceptrons multicamadas com nova ação de regularização, essas redes possuem uma espécie de conexão total, ou seja, todos os neurônios de uma camada estão ligados a todos os neurônios da camada seguinte, porém, diferente dos perceptrons, as CNNs utilizam os padrões hierárquicos de dados para montar padrões mais complexos com o uso de padrões mais simples. Uma grande vantagem das CNNs é o pouco pre-processamento necessário, os filtros não precisam ser inseridos manualmente, a rede aprende sozinho os filtros, logo, o esforço humano é bem menor.

2.2.4.1 Funcionamento dos Neurônios

Os neurônios são unidades de processamento que visam reproduzir o funcionamento de um neurônio biológico, ou seja, após receber uma sequência de estímulos (entradas) ele produz um sinal de resposta (saída). Para o contexto da CNNs, os neurônios são responsáveis por realizar previsões de acordo com suas entradas. Para cada entrada, do qual, costuma ser a ligação entre um neurônio da camada atual com a camada anterior, existe um peso de ponderação, no final o neurônio soma o valor de todas suas entradas junto a um valor de "bias"e, caso o valor seja suficiente para ativar sua função de ativação, o seu valor de saída é reproduzido para os neurônios da camada seguinte.

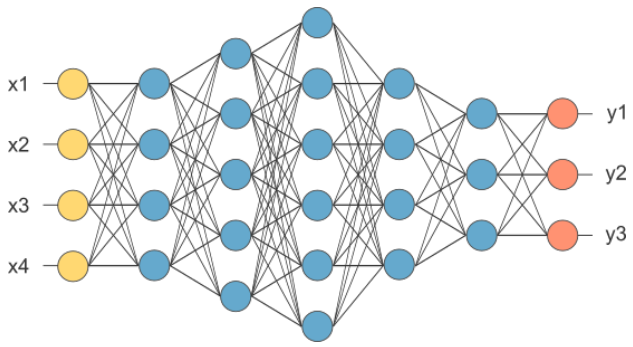

Figura 2.8: Exemplo de rede neural artificial. Fonte: DECOM. (2021).

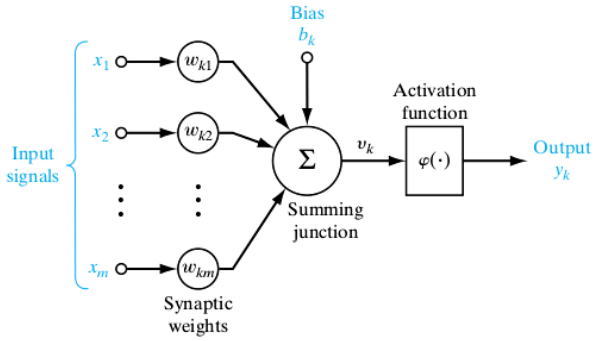

Figura 2.9: Esquema genérico para representar um neurônio em CNN's. Fonte: DECOM. (2021).

2.2.4.2 Arquitetura

A arquitetura de uma CNN costuma ser como mostra a figura 2.10, de forma geral, é aplicado uma imagem como entrada, após isso, é aplicado uma série de "convolution", seguido de "pooling" e, por fim, uma série de "deconvolution"quando se deseja que a saída também seja uma imagem.

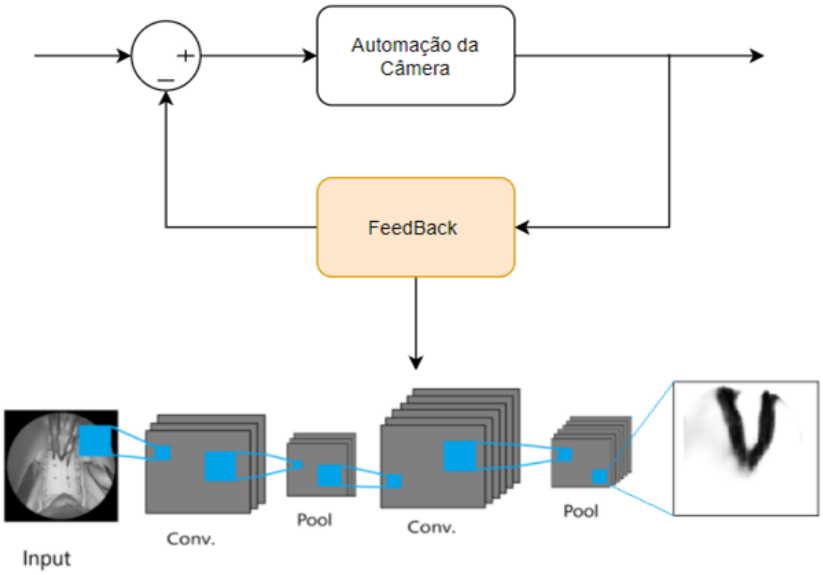

Figura 2.10: Modelo padrão de CNN para classificação de imagens. Fonte: Autor. (2021).

2.2.4.3 Convolution

O bloco principal de uma CNN é a camada convolucional, o método de convolução é uma maneira de mesclar dois conjuntos em apenas um, essa técnica utiliza um filtro conhecido por kernel que é aplicado na entrada para produzir um "feature map", como a entrada de uma imagem RGB é 3D, o kernel também é 3D. O kernel realiza uma multiplicação de matriz com a entrada, após isso, é feito uma soma de todos os resultados e o somatório é adicionado no feature map como mostra a figura 2.11.

Caso a convolução seja realizada com a técnica padding, o feature map produzido terá as mesmas dimensões de largura e altura, porém com profundidade reduzida, por exemplo, sem em uma entrada 32x32x3 é aplicado o filtro 5x5x3 o resultado será um feature map de 32x32x1 como mostra a figura 2.12. Após a criação do feature map é utilizado uma função de ativação, para o caso de reconhecimento de imagens é utilizado o "relu", por fim, é comum após a ação de convolução fazer o "dropout" de certa porcentagem de neurônios, para que a próxima camada não sofra overfitting.

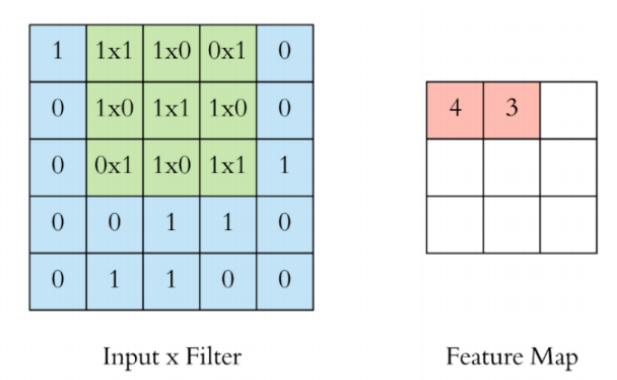

Figura 2.11: Ação do filtro na entrada gerando o feature map. Fonte: (DERTAT,2017).

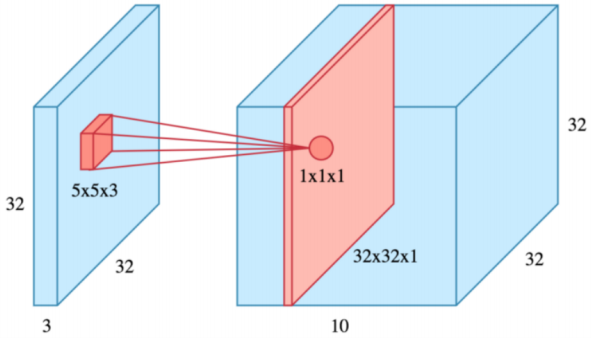

Figura 2.12: Saída do conjunto de feature maps formados. Fonte: (DERTAT,2017).

2.2.4.4 Conv2DTranspose

Conhecido também como deconvolução é uma classe de camada de modelos disponibilizada pelo Keras. Esta classe surge pela necessidade de refazer as reduções de tamanho ocasionadas pela implementação de maxpooling, quando se deseja que a saída tenha um tamanho especifico, como por exemplo, um modelo com imagens de entrada de tamanho 256 x 256 que vai retornar a mesma imagem com um novo filtro, para isso, é necessário que a saída do modelo também seja de 256 x 256. Assim, para não limitar o uso de maxpooling que poderia prejudicar a qualidade do modelo a deconvolução foi implementada. As entradas obrigatórias são :

filtro:

Inteiro indicando a quantidade de filtros na saída da deconvolução.

Kernel Size:

Inteiro ou Tupla com dois inteiros, com o objetivo de especificar a altura e largura da "janela de convolução" (convolution window).

Strides:

Inteiro ou Tupla de dois inteiros que defini o tamanho do passo na altura e largura na convolução.

Activation:

É o parâmetro que defini como será realizado a ativação do neurônio, as versões utilizadas é o "relu" e o "sigmoid". O relu retorna o valor máximo entre zero e a entrada do tensor, ou seja, é uma função linear retificada. Sigmoid é a função de ativação que ativa a formula:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Com este modelo de ativação, a saida sempre estará entre 0 e 1, onde para valores de entrada menores que -5 o valor estará muito proximo de zero e para valores maiores que 5 o valor estará muito próximo de 1.

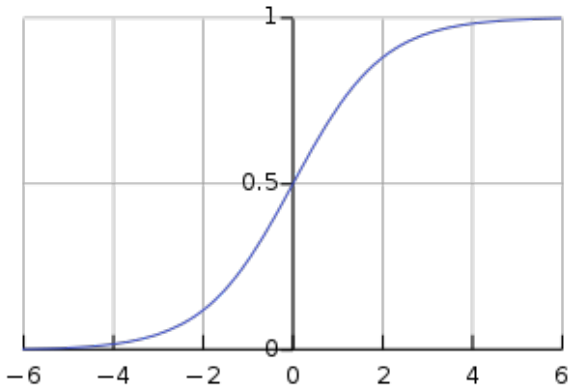

Figura 2.13: Função de Ativação Sigmoid. Fonte: (WIKIPEDIA, 2021).

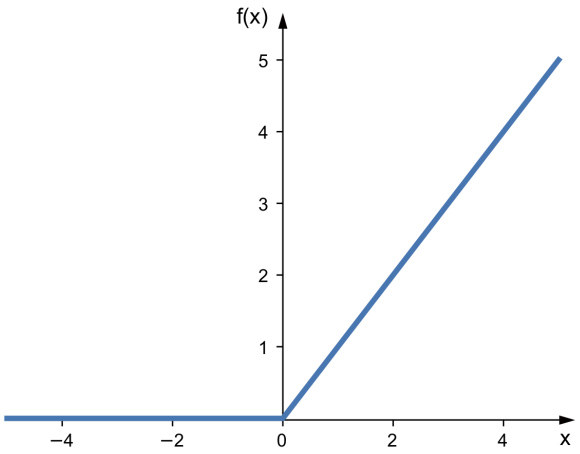

Figura 2.14: Função de Ativação Relu. Fonte: (SEBASTIAN, 2021).

Padding:

É o parâmetro que controla a quantidade de preenchimento que será adicionado na entrada para que o filtro do kernel possa passar por toda a imagem sem ultrapassar suas dimensões máximas.Quando o Padding é o "same" será adicionado o preenchimento na entrada para que o seu tamanho espacial seja igual ao da saída. Quando o Padding é o "valid", então não será adicionado preenchimento e a saída ficara menor que a entrada. (KERAS.IO,2021)

2.2.4.5 Concatenate

É uma classe de camada de modelos disponibilizada pelo Keras que concatena uma lista de entradas, recebe uma lista de tensores, onde é necessário que estes possuam as mesmas dimensões com exceção do eixo que será concatenado, retornando um único tensor resultado da concatenação das entradas. Os parâmetros obrigatórios são a lista de tensores de entradas e o eixo que será realizado a concatenação, como é possível ver no exemplo a seguir, disponibilizado pelo tensorflow: (KERAS.IO,2021)

x = [0 1 2 3 4] # shape(1,5)

y = [5 6 7 8 9] # shape (1,5)

z = tf.keras.layers.concatenate([x, y], axis=0)

z = [[0 1 2 3 4] , [5 6 7 8 9]] # shape (2,5)

2.2.4.6 Pooling

O pooling é uma camada geralmente adicionada após cada convolução, o pooling é uma camada que reduz a dimensão da camada criada pela convolução, este método ajuda a reduzir o número de parâmetros para treinamento, além de ser benéfica contra o overfitting. A técnica mais comumente

utilizada é o “maxpooling”, desta maneira, com um window, que geralmente é 2x2, é verificado qual o maior valor dentre os parâmetros e apenas ele é transmitido para a nova camada, como mostra a figura 2.15. A figura 2.16 mostra como o pooling é feito em cada feature map individualmente, adicionando todos em uma única nova camada de dimensões reduzidas.

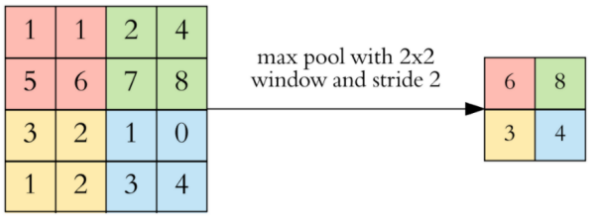

Figura 2.15: Ação do maxpooling em uma camada do feature map. Fonte: (DERTAT,2017).

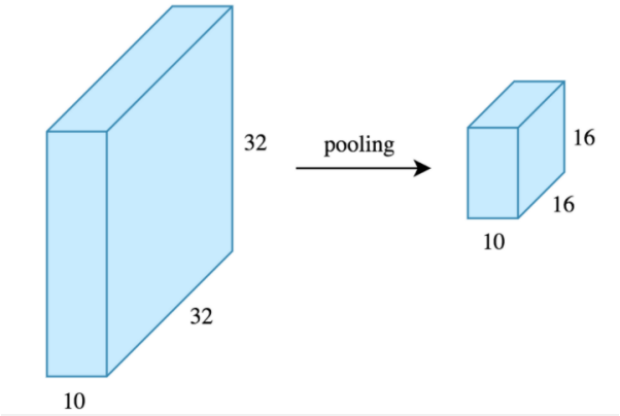

Figura 2.16: Visão geral da criação da camada via pooling. Fonte: (DERTAT,2017).

2.2.5 Escolha do Artigo inspiração

Dentre os artigos da tabela 1, em anexo, o artigo (PEREZ,2020) foi selecionado como alvo de replicação de trabalho. O artigo selecionado tem como objetivo gerar modelo capaz de identificar objetos cirúrgicos em imagens de procedimentos reais, para isso, foi elaborado uma grande base de imagens simuladas via software Unity3D para alimentar o modelo. As imagens criadas via simulação são separadas em três grandes grupos, onde cada grupo é designado dado a complexidade da imagem.

O primeiro grupo é de imagens básicas e de baixa complexidade, a diferença do objeto para o restante da imagem é dado apenas pela variação de uma única cor. O segundo grupo é o conjunto de imagens de complexidade intermediária, possui uma paleta de cores maior, suficiente para gerar formas e impressão de profundidade. O terceiro grupo são de imagens complexas, bem próximas do que se encontra em imagens reais, a sua paleta de cores é a maior dentre os três grupos, apresenta formas, profundidade e brilho, todavia, o tamanho deste tipo de imagem é consideravelmente maior.

O artigo (LEE,2019) explicita um grande problema para modelos que almejam realizar identificação precisa de objetos, a necessidade de uma grande quantidade de imagens para treino do modelo, sem isso a rede não consegue ter alta eficiência. Por motivos éticos, infelizmente não existem muitas imagens públicas de laparoscopias, portanto, o banco de imagens simuladas feita pelo artigo (PEREZ,2021) é a melhor solução para garantir uma quantidade mínima de entrada de dados no algoritmo, uma vez que para cada grupo de imagens existem 10.375 exemplares.

O artigo de (RONNEBERGER,2015) é utilizado como grande inspiração para o modelo de rede neural artificial, neste trabalho é apresentado um modelo geral que apresenta excelentes resultados para segmentação de imagens médicas, portanto, foi utilizado como base sofrendo apenas algumas modificações em camadas específicas, ver capítulo 3.1.1.1, com o objetivo de se obter novos resultados se comparados ao (PEREZ,2020).

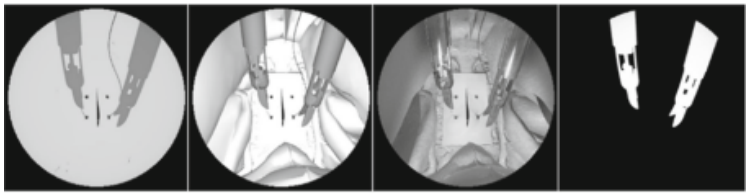

Figura 2.17: Grupo de imagens obtidas por simulação, da esqueda para a direita, imagem básica, intermediária, complexa e a label em comum. Fonte: (PEREZ,2020). (2021).

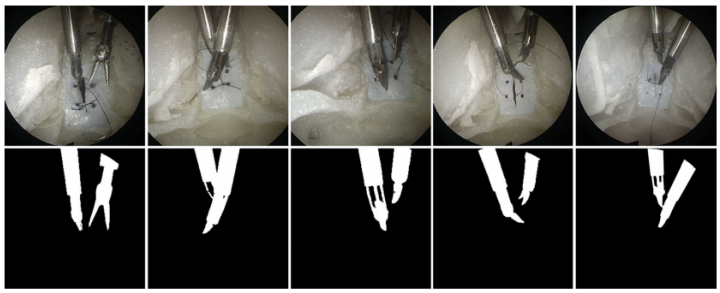

Figura 2.18: Imagens reais com a sua respectiva label. Fonte: (PEREZ,2020). (2021).

2.3 Plataforma

Devido ao alto poder computacional exigido, foi utilizado a plataforma do Google Colab. O Colab fornece integração a GPUs de alta perfomance, além de as principais bibliotecas relacionadas a inteligência artifical já estarem implementadas na plataforma. Foi utilizado a versão PRO do Colab, versão paga que fornece GPUs ainda mais robustas e mais memória RAM para acesso. (COLAB,2021)

2.4 Bibliotecas

2.4.1 OpenCV

O Open Source Computer Vision Library (Open CV) é uma biblioteca multiplataforma de uso livre voltada para a visão computacional, com módulos para processamento de imagens e vídeos, além de diversos algoritmos de visão computacional como, por exemplo, gerador de filtros de imagens e reconhecimento de objetos. O open CV é uma ferramenta que pode ser aplicada na etapa 1 da elaboração do modelo de ML, ver figura 2.1, quando o objetivo é realizar a predição baseado em imagens, realizando então o tratamento dos dados de entrada. Alguns exemplos desta aplicação vão de casos simples como determinar se a imagem possui um gato ou um cachorro, até casos mais complexos como a detecção de objetos em tempo real por veiculo aéreo não tripulado (UAV). (ALPISTE,2019).

2.4.2 Tensorflow

O tensorflow (TF) é uma plataforma de código aberto voltado para ML, do qual, por meio de suas bibliotecas é possível desenvolver, treinar e implantar modelos. Atualmente possui versões para Python, C++ e JavaScript. O tensorflow possui parceria com a ferramenta Google Colab, assim, é possível desenvolver modelos pelas APIS do TensorFlow e executa-los remotamente nas máquinas disponibilizadas pelo Google. O TF já disponibiliza uma versão conhecida como TensorFlow GPU, onde, caso exista alguma placa gráfica compatível, é possível, de maneira simples, integrar ao tensorflow para que o processamento seja feito pela GPU, uma vez que uma rede neural basicamente se resume a grndes operações entre matrizes, o uso da GPU pode reduzir drasticamente o tempo de execução. O google Colab disponibiliza de forma gratuita a integração com suas GPUs mais simples, para execuções mais densas existe a possibilidade de aderir a um plano para ter acesso a GPUs melhores, logo, menos tempo de execução. (TENSORFLOW,2021)

2.4.3 Keras

O Keras é uma API de alto nível desenvolvida em cima da plataforma TensorFlow, seu objetivo é tornar o processo entre a ideia e o resultado o mais rápido e simples possível. Reduzindo ao máximo o numero de ações necessárias para montar seu modelo de maneira completa, permitindo a integração a GPUs e TPUs que promovem grande velocidade ao desenvolvimento do modelo. Na plataforma de competição Kaggle (KAGGLE,2021), conhecida por disponibilizar competições financiadas por grandes empresas em buscas de modelos inovadores em Machine Learning. Dentre as top 5 equipes por competição o framework mais utilizado é o **Keras**, justamente por seu simples manejo e alta velocidade (KERAS.IO,2021)

2.5 Funções utilizadas com o keras

2.5.1 Compile

Uma vez montado o modelo da rede, o keras fornece um método chamado compile, que configura automaticamente o modelo para ser treinado. As variáveis de entrada mais relevantes são:

2.5.1.1 Optmizer

É uma classe, do qual, possui o método que o modelo será treinado de fato. Um bom optmizer é fundamental par se obter velocidade e perfomance altas. Atualmente existem diversos optmizers já implementados, o utilizado neste trabalho é o SGD. (KERAS.IO,2021)

SGD:

Otimizador de descida gradiente com momentum (SGD em inglês) vem por padrão com uma taxa de aprendizado de 0,01 e momentum 0. Quando o momentum é 0, a regra para atualização do peso w é:

$$W = W - LEARNING\_RATE * G \tag{2.2}$$

2.5.1.2 LOSS

É a classe responsável por dizer ao otimizador o quanto está se aproximando ou afastando do resultado desejado. Para o projeto foi utilizado o método "Bynary Cross Entropy", do qual, utiliza a fórmula 2.3 onde y é o label e p(y) é a probabilidade de aquele ponto ser da classe designada pela label, ou seja, o "Bynary Cross Entropy" para o caso do projeto vai mostrar para cada ponto da imagem a probabilidade daquele ponto ser um instrumento cirúrgico.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(p(y_i)) + (1 - y_i) * log(1 - p(y_i)) \tag{2.3}$$

2.5.1.3 METRICS

As métricas são funções utilizadas para avaliar a performance do modelo, assim como o loss, a unica diferença é que as métricas são apenas para análise, ou seja, elas não influenciam no treinamento do modelo. Para o Projeto existem duas métricas:

BynaryAccuracy:

Mostra a frequência em que as predições acertam, onde, a comparação é feita binário a binário entre a predição e a label. (KERAS,2021)

MeanIOU:

É uma métrica extremamente utilizada em segmentações de imagem, onde é calculado o IOU para cada classe e, após isso, é feito a media dos IOUs resultants. Lembrando que o IOU é definido como:(KERAS,2021)

$$IOU = \frac{verdadeiro\_positivo}{verdadeiro\_positivo + falso\_positivo + falso\_negativo} \tag{2.4}$$

2.5.2 Fit

Quando compilado o modelo, resta apenas treinar o modelo e, para isto, o keras possui o método Fit. Inserindo os argumentos de entrada: número de épocas, número de passos por época, dados de treino e de validação; O keras faz todo o resto, treina o modelo mostrando a cada época o resultado para cada uma das métricas utilizadas, além da função loss. (KERAS,2021)

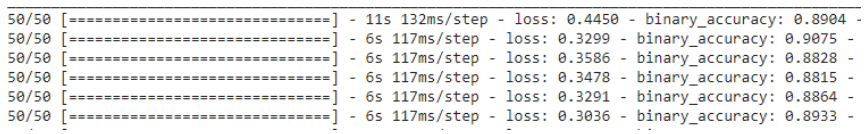

Figura 2.19: Exemplo de rede neural em treinamento. Fonte .Autor (2021).

2.5.3 Predict

Quando treinado, o método desenvolvido pelo keras PREDICT permite que o modelo realize predições além das de seu treino, muito util para validar de fato o resultado e para aplicar o modelo de fato, o unico cuidado necessário com este método é sempre garantir que a entrada esteja com o formato designado pelo modelo, assim, o modelo não terá dificuldades em mostrar o seu desempenho.(KERAS,2021)

Capítulo 3

Desenvolvimento

3.1 Algoritmo

O Algoritmo desenvolvido pode ser separado em 3 blocos, figura 3.2, que representam todas as classes e funções implementadas. As classes e funções estão anexadas no fim do documento. O funcionamento, embora contenha diversas etapas, é simples:

Imagens para treino e validação são armazenadas em pequenos lotes por vez, estas imagens são alteradas por filtros, após isso, são utilizadas para o treino onde ocorre o ajuste do peso dos parâmetros, este processo se repete até que o treino esteja finalizado, por fim, o algoritmo retorna o modelo treinado pronto para ser testado.

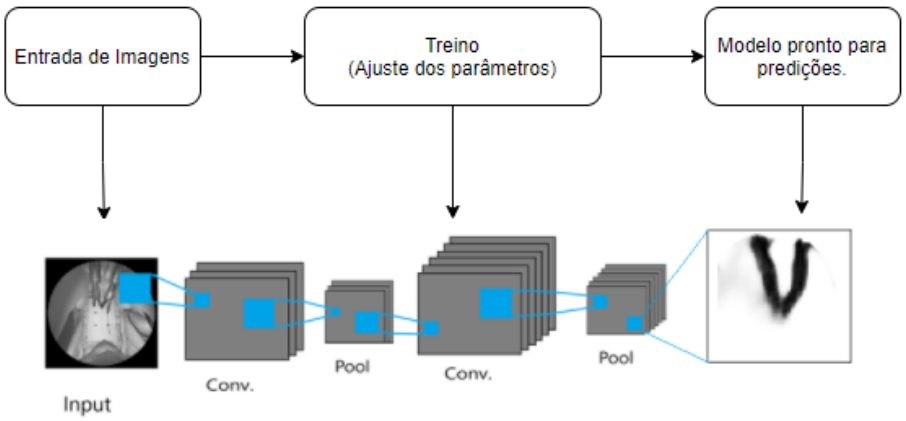

Figura 3.1: Blocos de funcionamento do algoritmo. Fonte .Autor (2021).

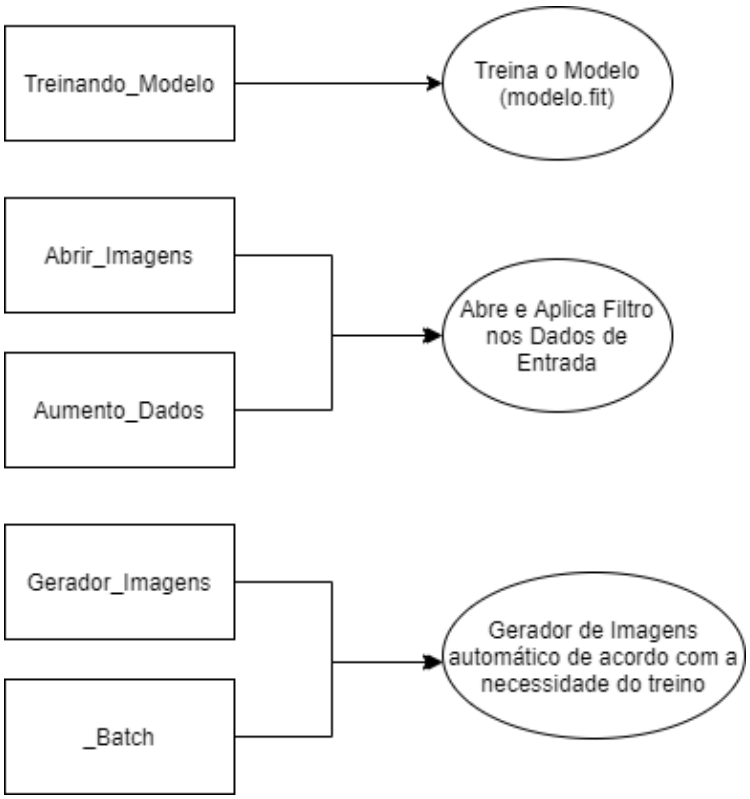

Figura 3.2: Blocos de funções implementadas. Fonte .Autor (2021).

3.1.1 Funções Implementadas

3.1.1.1 _Modelo

O primeiro bloco, responsável por montar a rede, foi implementado com apenas uma classe chamada **_modelo**, do qual retorna o resultado mostrado na figura 3.3. Este modelo é uma variação do criado pela referência (RONNEBERGER,2015), figura 3.4, e reproduzido por (PEREZ,2020). As modificações feitas foram a adição de duas camadas de convolução/deconvolução com números de filtros igual a 32 e a adição de mais uma camada de convolução/deconvolução com número de filtros igual a 512. O terceiro bloco foi implementado através das classes e funções: **hora_da_verdade** e **array_para_imagem**. Durante o treinamento do modelo, após a execução de cada época, é gerado predições do resultado e armazenadas para validação do modelo.

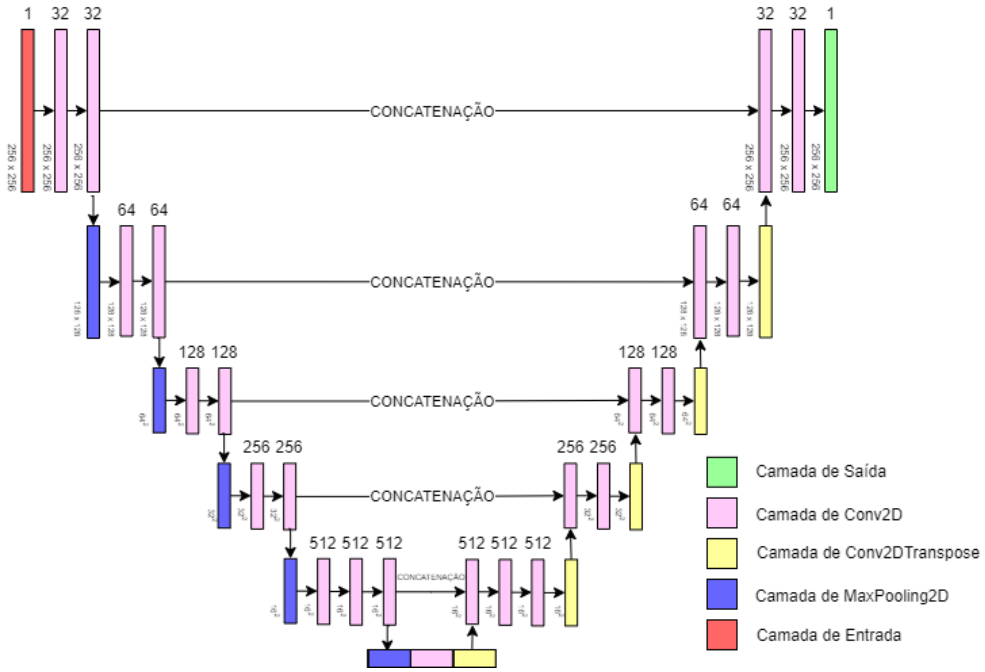

Figura 3.3: Modelo de camadas da rede neural implementada. Fonte .Autor (2021).

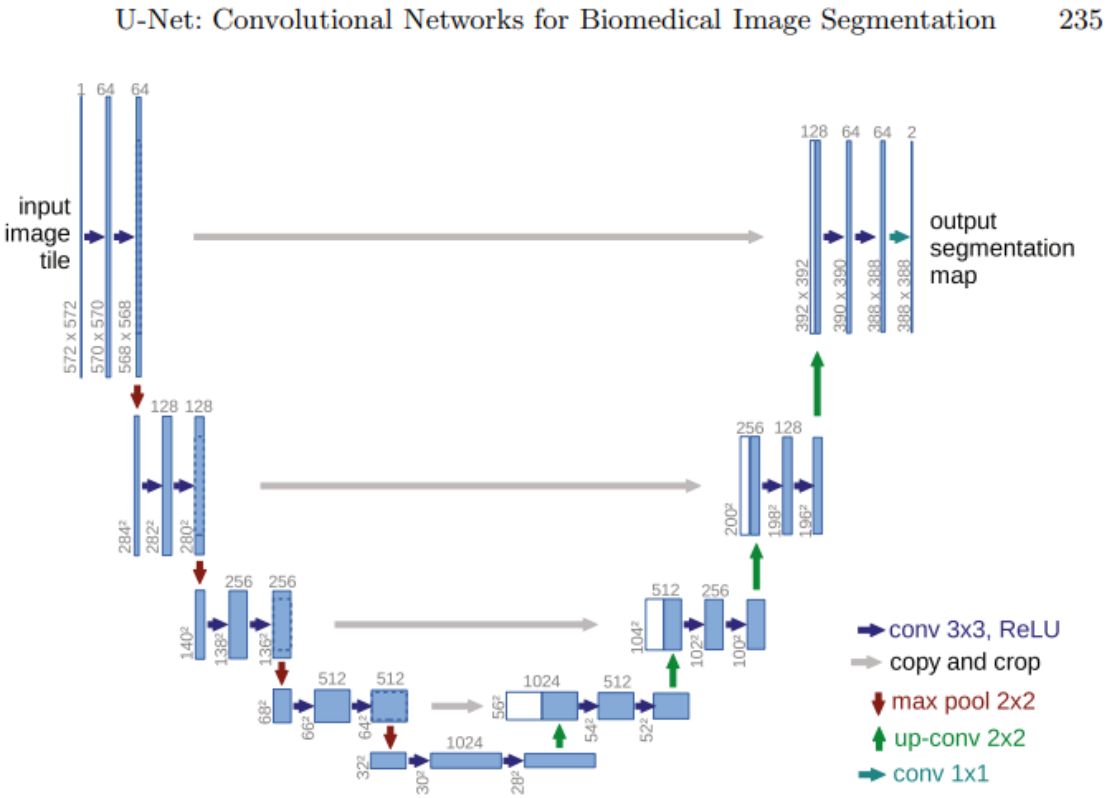

Figura 3.4: Modelo de camadas da rede neural implementada por (RONNEBERGER,2015) . Fonte .RONNEBERGER (2015).

Olhando a figura 3.4, é possível perceber que o modelo segue a lógica: em seu lado “esquerdo”, a cada duas camadas de convolução, é armazenado a ultima camada e feito um Pooling, ao chegar ao centro, geralmente com o número de filtros igual a 1024, é iniciado as etapas de deconvolução no lado “direito”, após cada deconvolução é concatenado o número de filtros com a camada que possui a mesma quantidade, porém do lado “esquerdo”, resultando no final em um filtro com o tamanho da imagem de entrada (256 x 256). A rede possui um total de 26.375.329 milhões de parâmetros capazes de serem treinados.

```
Total params: 26,375,329
Trainable params: 26,375,329
Non-trainable params: 0
```

Figura 3.5: Quantidade total de parâmetros treináveis. Fonte .Autor (2021).

3.1.1.2 Treinando_Modelo

O **treinando_modelo** é aonde o ajuste dos pesos de entrada e saida dos neurônios (instrução *Fit* do keras e tensorflow) é executado, ou seja, onde o treino de fato acontece. Os parâmetros de entrada são o número de épocas, a quantidade de passos por época , informações sobre o diretório e dimensões da imagem, o retorno da função é o modelo treinado. Realiza um loop onde para cada época, aciona o **gerador_imagens** que retorna as imagens necessárias para o treino e validação, ao final de cada ciclo no loop uma época de treino é finalizada, assim, as métricas do momento são geradas. O funcionamento do ajuste dos parâmetros, considerando a chamada das funções pode ser visto na figura 3.6.

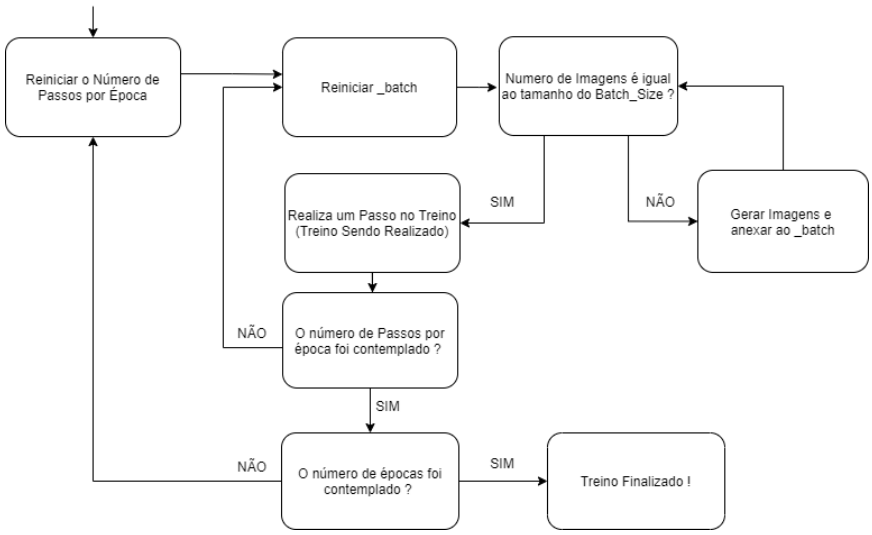

Figura 3.6: Sequência de ações executadas durante o treinamento. Fonte .Autor (2021).

3.1.1.3 Gerador_Imagens

O **Gerador_Imagens** é uma espécie de ponto de controle, executa um loop responsável por acionar o **_batch** sempre que necessário, recebe as imagens concatenadas no tamanho exato para o *batch* e as entrega para o *Fit* realizar o treino. Com este modelo de aplicação, é fornecido para o ajuste de parâmetros apenas a quantidade necessária para a execução do *batch*, ao invés de todas imagens de uma só vez, assim, mesmo com um grande número de imagens para serem processadas a memória necessária é relativamente baixa.

3.1.1.4 _Batch

Responsável por concatenar as imagens geradas pelos filtros implementados em **Aumento_Dados**. Executa um loop onde recebe imagens em format de array do **Aumento_Dados**, executa um *threshold* em 0,5 forçando os valores do array para 1 ou 0, após isso, é concatenado as imagens de treino e validação para fornecer no tamanho exato exigido pelo *batch*.

3.1.1.5 Aumento_Dados

Nesta classe são aplicados dois filtros com proposito geral de modificar as imagens de maneira que, se comparadas com a original, existe distinção suficiente para categorizar como uma nova imagem, assim, mesmo com o número fixo de imagens do *database*, os filtros possibilitam a criação de novas imagens para serem embutidas no modelo. Os dois filtros implementados são: Rotação , Fliping.

Rotação:

Rotaciona o imagem de forma aleatória respeitando apenas o *range* delimitado, para a aplicação o *range* definido foi de -30 a 30 graus.

Fliping:

Inverte o sentido da imagem.

3.1.1.6 Abrir_Imagens

Responsável por abrir a imagem, através do diretorio fornecido, e entrega-la em formato de array e dimensões adequadas. Este processo é bem simples graças a biblioteca OpenCV.

3.1.1.7 Hora_da_verdade

Responsável por armazenar o resultado da predição no formato adequado em tom de cinza. Junto com o resultado e armazenado a entrada e a label para comparação.

3.1.1.8 Array_para_imagem

Transforma um array em imagem no tom de cinza.

3.2 Parâmetros

Os parâmetros utilizados são todos pertencentes a função de ajuste de parâmetros (*Fit*) fornecido pela API do keras, o significado e ajuste de cada um será descrito a seguir:

- **Número de épocas:** Uma época como descrito na figura 3.6 é definido como uma etapa completa no treinamento, ou seja, quando todas as imagens designadas para aquela etapa foram utilizadas para o ajuste no sentido de ida e volta dos neurônios da rede. O valor utilizado para o algoritmo foi de **150**, este valor foi designado após uma sequência de testes com diversos valores, como é descrito no capítulo 4.1, a limitação de memória RAM impede que o programa treine muitas imagens de uma só vez, portanto, a melhor solução encontrada foi diminui-las, redução do número de passos por época, e aumentar o número de etapas do treinamento.
- **Passos por época:** Os passos por época são definido como a quantidade de vezes que conjuntos de imagens de treino serão utilizadas para treino a cada etapa do ajuste dos parâmetros. O valor utilizado é de **50**. Como descrito no tópico anterior, devido a baixa quantidade de memória RAM disponivel, foi determinado utilizar poucas imagens por treino, que acarreta em poucos passos por treino e em uma grande quantidade de etapas.
- **Tamanho do Lote:** O tamanho do lote, *Batch Size* é a quantidade de imagens que serão processadas a cada passo de uma época. O valor definido é de **10**.
- **Taxa de Aprendizado:** A taxa de aprendizado, *Learning Rate*, é considerado por muitos o principal parâmetro a ser definido, é o responsável por ditar a velocidade do treinamento da rede. Uma vez que o modelo está sempre na busca de um valor ótimo de seus parâmetros que sejam capaz de entregar a melhor predição possível, a taxa de aprendizado vai determinar a velocidade em que o treinamento vai de encontro a um valor ótimo de parâmetros. Uma taxa de aprendizado baixa vai fazer com que o treino leve muito tempo até encontrar um valor ótimo, uma taxa alta vai fazer com que o modelo tenha grandes oscilações em seus resultados. Portanto, é fundamental encontrar um intermédio entre o que é uma taxa alta e baixa para o modelo, neste trabalho a taxa que aoresentou melhores resultados foi de **0,02**, não é tão lenta e ainda é capaz de esquivar de ótimos locais incapazes de entregar predições consideradas satisfatórias.

3.3 Métricas:

- **Perda:** Como explicado no capítulo 2.5.1.2; A perda (*Loss*) é a classe que diz ao otimizador o quão próximo ou afastado a saída do modelo se encontra do valor desejado, evidenciado

pelas *labels*. A métrica escolhida foi a ***Bynary Cross Entropy***, esta métrica é a mais recomendada para classificadores binários, a sua saída indica a probailidade do valor de entrada ser verdadeiro ou falso, no caso deste trabalho, vai indicar a probabilidade de cada bit ser parte de um instrumento cirúrgico.

- **Acurácia Binária:** Mostra a porcentagem de acerto da predição do modelo em comparação a label. Como o intuito deste trabalho é demarcar apenas o instrumento cirúrgico, esta métrica auxilia na compreensão do resultado durante o processo de treino, uma vez que para cada época o resultado das métricas são armazenados, com isso, o autor teve a possibilidade de entender, realizando testes, que algumas modificações ao longo do tempo pioravam a acurácia binária, assim, o ajuste dos parâmetros era refeito.

3.4 Execução

3.4.1 Ambiente de execução

O ambiente de execução utilizado foi o Google Colab versão PRO, ver no capítulo 2.3. Com esta plataforma é possível ter acesso a GPUs de alta perfomance, infelizmente o computador do autor não foi capaz de processar a grande quantidade de imagens, a expectativa de horas do treinamento era de **150 horas**, o que torna o treino do modelo inviável, a melhor alternativa encontrada foi o Colab, sendo necessário apenas realizar algumas atualizações em bibliotecas para que suas funcionalidades correspondessem as expectativas.

3.4.2 Treino

3.4.2.1 Banco de Imagens

Para a execução foram utilizado dois bancos de imagens fornecido por (PEREZ,2020), um menos realista, logo, mais simples e outro mais renderizado, bem mais próximo da realidade.

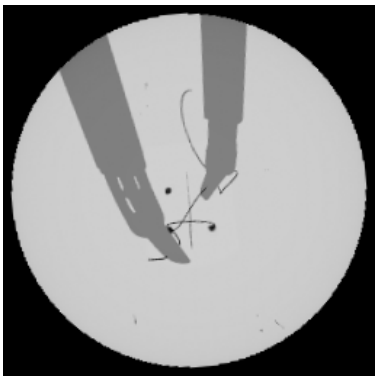

Figura 3.7: Exemplo de imagem de modelagem simples. Fonte .(PEREZ,2020).

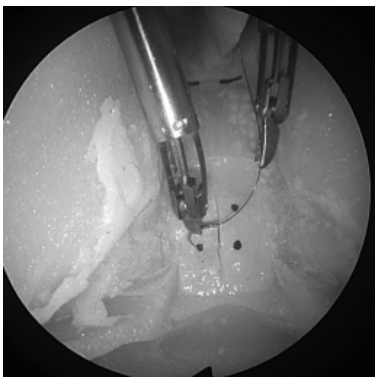

Figura 3.8: Exemplo de imagem de modelagem complexa. Fonte .(PEREZ,2020).

3.4.2.2 Treinamento

Este trabalho resultou em dois modelos de rede neural, logo, é possivel separar o treinamento em dois. O primeiro treino, fonte do primeiro modelo, foi utilizado apenas as imagens simples do banco, ou seja, no ajuste dos pesos dos parâmetros apenas as imagens básicas foram fornecidas, o intuito é mostrar se mesmo com imagens pobre em detalhes o modelo seria capaz de realizar boas predições em imagens reais. O segundo treino utilizou apenas imagens complexas do banco, embora necessite de mais processamento, é relevante ver o quão bom o modelo pode ser quando no ajuste dos parâmetros é utilizado imagens próximas da realidade, assim, é possível analisar a influência que a abstração da imagem terá nas predições.

Capítulo 4

Resultados

4.1 Saída dos Modelos

Ao treinar o modelo, o resultado das predições vem no formato de array, todavia, a biblioteca do *OpenCV* é capaz de fazer a conversão do bloco de array para imagem automaticamente. Devido a gama de números permitidos para a saída, a predição sai com o resultado no tom de azul e amarelo, como é possível ver pela figura 4.1 e 4.2, logo, para manter a mesma tonalidade da label (preto e branco) é aplicado uma conversão de tonalidade ao salvar a imagem por meio da função ***cmap***, fornecida pela biblioteca *matplotlib.pyplot*. Com a alteração de tonalidade ao salvar as imagens, o resultado obtido respeita a mesma tonalidade que a label, todavia, as cores estão invertidas, na label o instrumento é branco e o restante preto, já nas predições o instrumento é preto e o restante branco, embora fosse possível converter mais uma vez as cores, o autor não achou necessário, uma vez que já é possível realizar uma boa interpretação dos resultados.

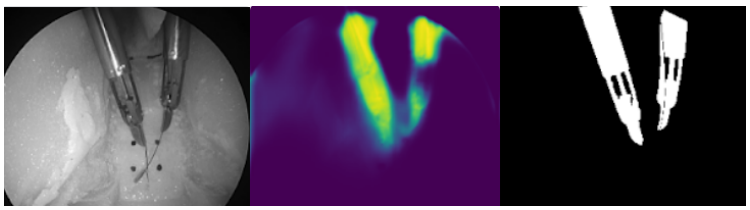

Figura 4.1: Imagem de entrada, resultado da saída e label da etapa de validação, respectivamente. Fonte. Autor (2021).

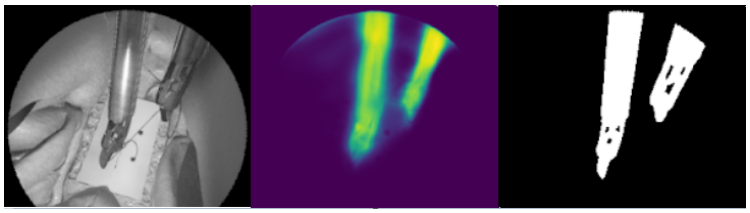

Figura 4.2: Imagem de entrada, resultado da saída e label da etapa de validação, respectivamente. Fonte. Autor (2021).

4.2 Execução Primeiro Modelo (Imagens Simples)

Com o fim do treinamento, o modelo salvo gerou o seguinte resultado, figura algo e figura outro, o treino foi satisfatório para o modelo conseguir identificar nas imagens mais simples a posição exata do instrumento (figura 4.3), todavia, para imagens reais, aonde o dimensionamento de formatos e a paletização de cores é bem mais complexa, ele não foi capaz de identificar os intrumentos (figura 4.4), ou seja, este modelo foi efetivo apenas no treino, sendo ineficaz na validação e predição. Na realidade ele demarcou de forma geral a área aonde o instrumento não está, acredita-se que isso se deve principalmente devido a pouca complexidade entre os tons de cinza da imagem de entrada. A execução do modelo levou aproximadamente 3 horas.

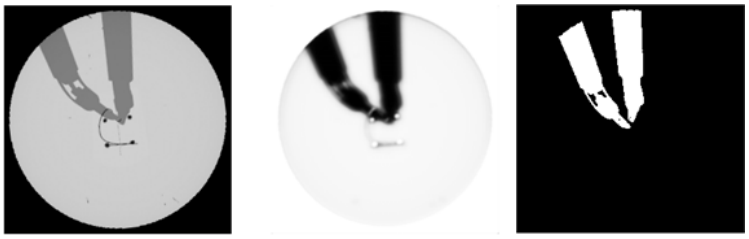

Figura 4.3: Imagem de entrada, resultado da saída e label da etapa de treino, respectivamente. Fonte. Autor (2021).

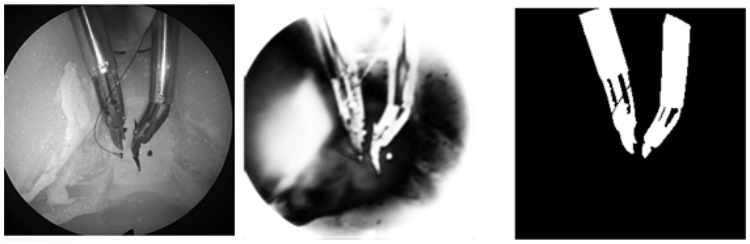

Figura 4.4: Imagem de entrada, resultado da saída e label da etapa de validação, respectivamente. Fonte. Autor (2021).

O resultado das métricas utilizadas no aprendizado podem ser visualizadas nas figuras 4.5 e 4.6. Para a métrica de **Perda** fica claro que próximo da época de número 20 a perda do teste divergiu totalmente, mostrando como o modelo não era capaz de identificar imagens mais complexas, ou seja, o modelo entrou em *overfitting*, o modelo funciona muito bem no treino mas não nas predições, isso ocorre por que o modelo apenas "decorou"a forma de acertar as predições do treino e acaba replicando este método na validação sem ponderar de forma adequada a diferença entre ambos. Assim como para a métrica de perda, a **acurácia binária** próximo da epóca de número 20 passou a divergir e o modelo encontrou muita dificuldade em acertar as imagens de teste, mesmo com o treino tendo bom resultado. Portanto, a diferença da imagem simples para a imagem de teste/validação é muito grande para o modelo ser capaz de compreender as duas.

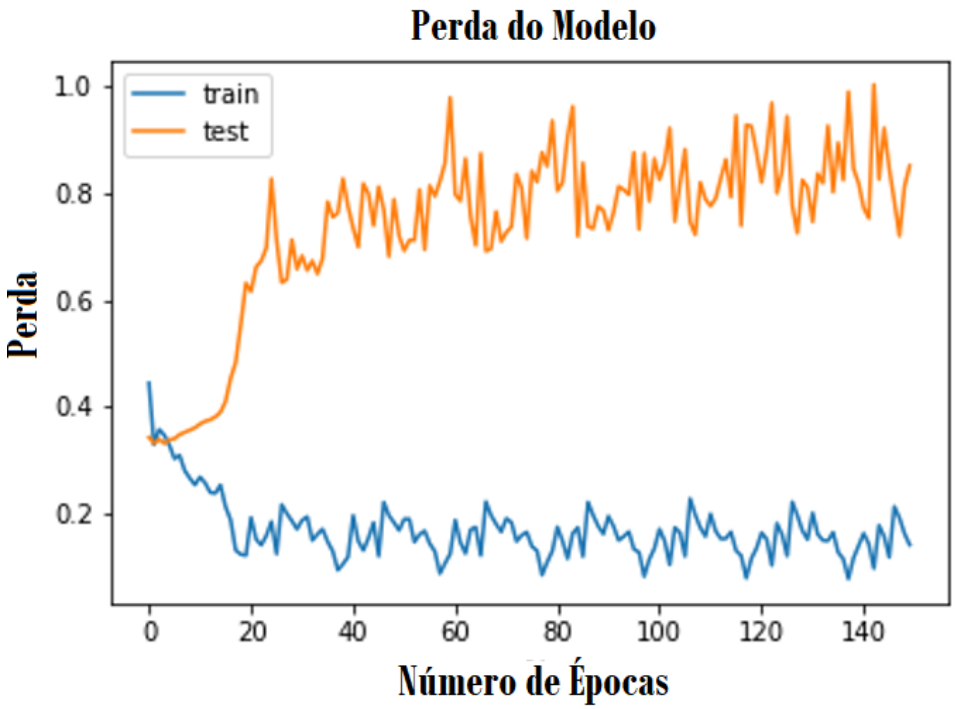

Figura 4.5: Valor de Perda ao longo das épocas. Fonte: (PEREZ,2020). (2021).

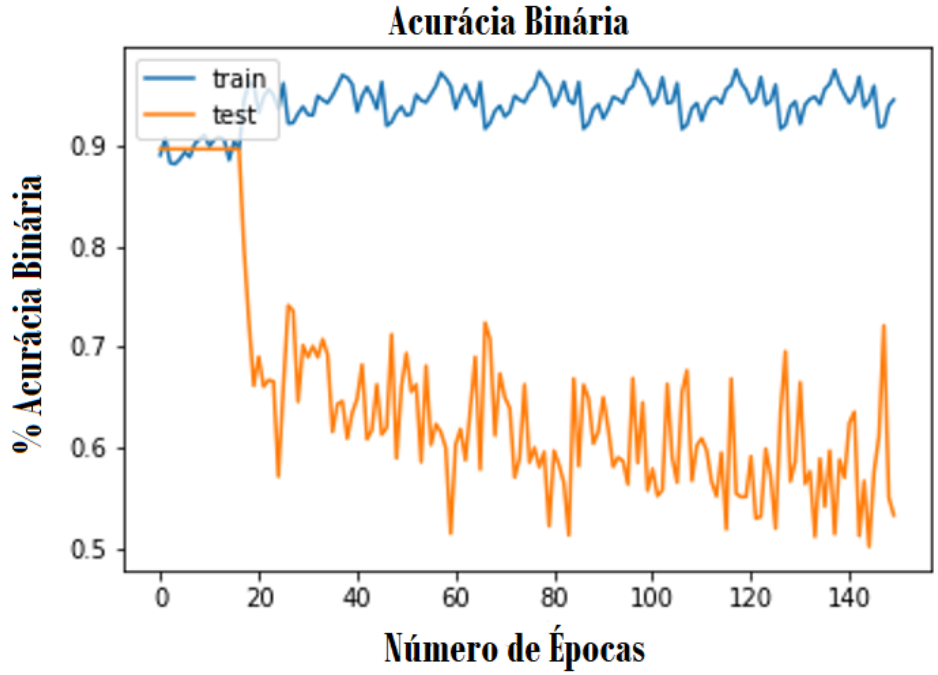

Figura 4.6: Valor de acurácia binária ao longo das épocas. Fonte: (PEREZ,2020). (2021).

4.3 Execução Segundo Modelo (Imagens Complexas)

Neste caso as imagens de entrada para o treino são bem mais complexas, se aproximando muito ao aspecto da imagem de validação. Inicialmente o modelo foi treinado de forma direta, porém como visto pela figura 4.7, o modelo chega a um ponto ótimo e possui muita dificuldade de sair desta região, ou seja, mesmo com o ajuste de parâmetros o modelo não foi capaz de ultrapassar este resultado. A solução encontrada para este problema foi utilizar o modelo inicialmente treinado com as imagens simples e aplicar um novo treino em cima com as imagens complexas, para isso, foi necessário apenas importar o modelo já treinado pela biblioteca do *keras* e executar novamente o treino, só que dessa vez com as imagens mais complexas. Esta abordagem gerou os resultados mostrados nas figuras 4.8 e 4.9, é possível ver que o modelo apresenta um bom resultado de forma geral em identificar os intrumentos mesmo nas imagens mais complexas de validação. A execução do modelo levou aproximadamente 6 horas, sendo considerado as 3 horas para rodar o primeiro modelo e 3 horas para execução do segundo.

Figura 4.7: Resultado da saída e label da etapa de validação. Modelo por treino direto. Fonte: Autor. (2021).

Figura 4.8: Imagem de entrada, resultado da saída e label da etapa de treino, respectivamente. Modelo gerado a partir de duas etapas de treino. Fonte. Autor (2021).

Figura 4.9: Imagem de entrada, resultado da saída e label da etapa de validação, respectivamente. Modelo gerado a partir de duas etapas de treino. Fonte. Autor (2021).

As métricas utilizadas no aprendizado podem ser visualizadas nas figuras 4.11 e 4.10. Para a perda, em comparação com o primeiro modelo de imagens simples, ver figura 4.5, ao longo das épocas a curva de teste não diverge, indicativo de *overfitting*. Existem alguns picos de queda esporádicos, todavia, a tendência permanence em 0,15. A acurácia binária a partir da época de número 80, atingiu a tendência de aproximadamente 0,95 no teste e 0,93 no treino, no caso da acurácia os pontos de divergência são poucos e representam uma queda de aproximadamente 10% na acurácia.

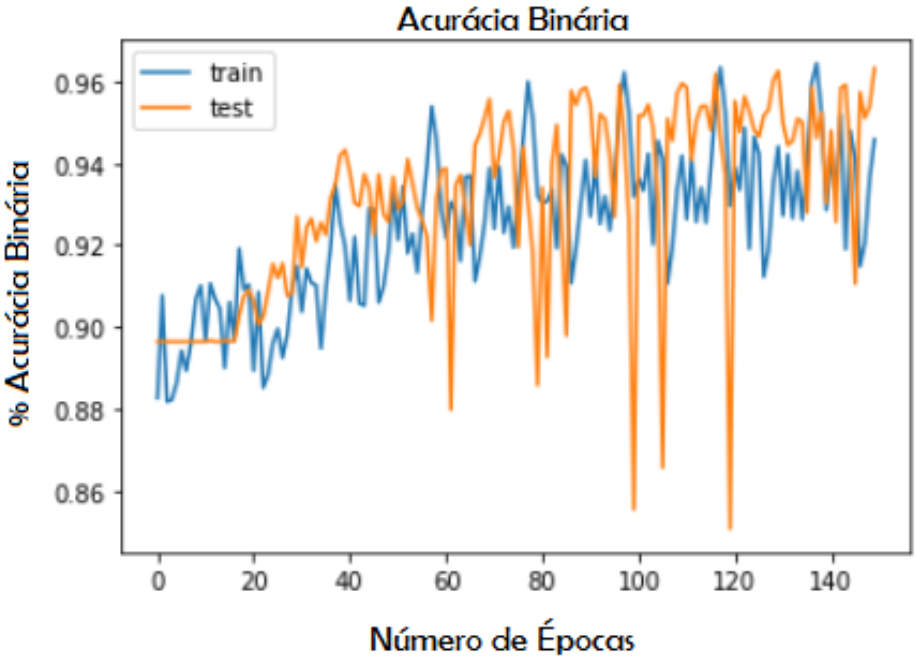

Figura 4.10: Imagem de entrada, resultado da saída e label da etapa de validação, respectivamente. Modelo gerado a partir de duas etapas de treino. Fonte. Autor (2021).

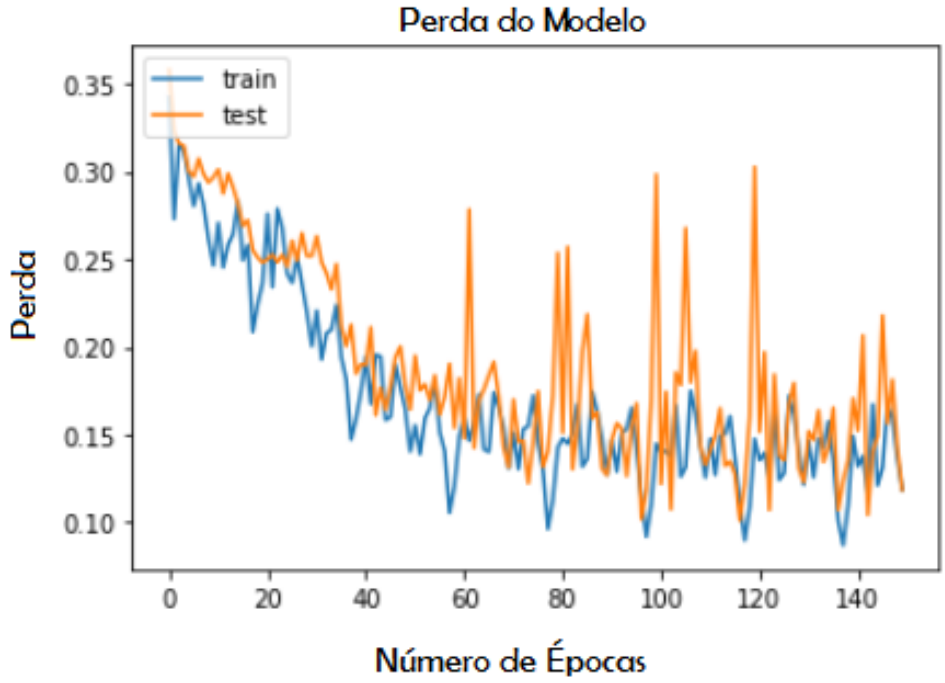

Figura 4.11: Imagem de entrada, resultado da saída e label da etapa de validação, respectivamente. Modelo gerado a partir de duas etapas de treino. Fonte. Autor (2021).

A partir da figura 4.12, é evidente como o modelo dependendo da posição do instrumento, possui dificuldade de encontrar o limite do aparelho do lado esquerdo, isso ocorre devido a proximidade no tom de cores e formatos caracteristico da região, assim, o modelo estima que naquela região está a borda e, com isso, acaba **manchando** a região na busca de melhorar a sua exatidão. Outro problema encontrado é quando os aparelhos estã muito próximos, novamente o modelo apresenta dificuldades para encontrar as bordas que definem aonde começa um e aonde termina o outro, figura 4.13, com isso, acaba demarcando toda a região entre eles, passando a **falsa** percepção de ser apenas um objeto.

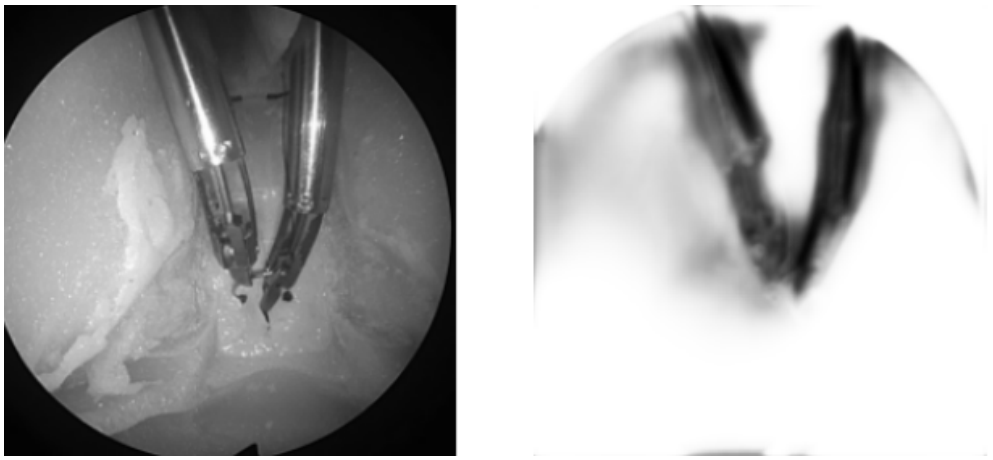

Figura 4.12: Imagem de entrada, resultado da saída da etapa de validação, respectivamente. Modelo gerado a partir de duas etapas de treino. Fonte. Autor (2021).

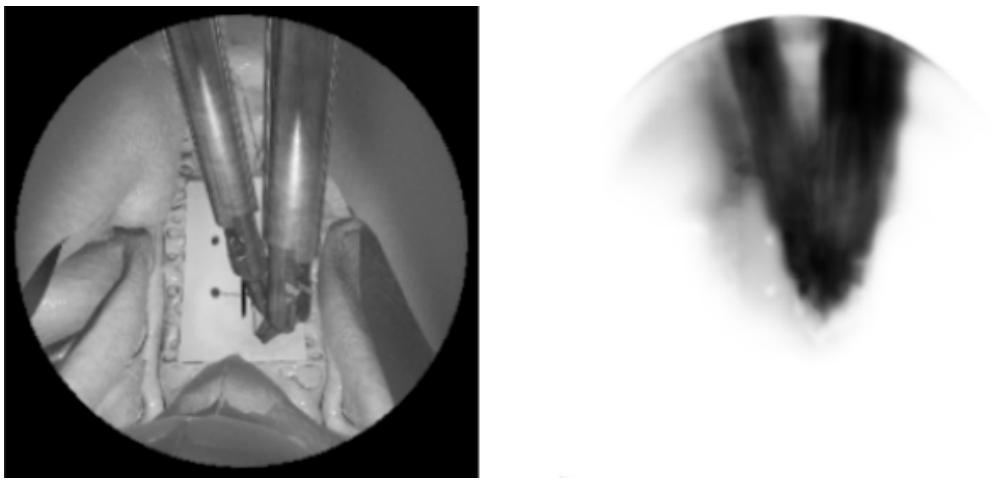

Figura 4.13: Imagem de entrada, resultado da saída da etapa de validação, respectivamente. Modelo gerado a partir de duas etapas de treino. Fonte. Autor (2021).

Capítulo 5

Conclusões

5.1 Comparação com (PEREZ,2020)

Comparando o resultado do primeiro modelo com o resultado de (PEREZ,2020), figura 5.1, ambos são ineficazes em realizar predições, levando a conclusão que o treino apenas com imagens simples não é suficiente para evitar o *overfitting*, ou seja, mesmo com bons resultados na fase de treino, a predição é inconclusiva. O resultado do segundo modelo, figura 5.2, apresenta uma predição satisfatória e próxima do resultado encontrado por (PEREZ,2020), logo, os dois modelos acabaram por encontrar um valor ótimo de predição em comum, este resultado era esperado já que as imagens de alimentação do modelo são as mesmas. Embora o segundo modelo apresente resultados satisfatórios, ainda existem dificuldades como as relatadas no capítulo 4.3, onde com o objeto em certas posições faz com que a rede não seja capaz de predizer com alto grau de confiabilidade, uma proposta de solução é o uso de imagens reais na fase de treino. Se compararmos a gama de posições e ângulos em que o objeto se encontra na simulação e no ato cirúrgico de fato, conclui-se que o total de possibilidades contamplada pela simulação ainda é pequena, com isso, o modelo não é capaz de estar preparado para todas as possibilidades.

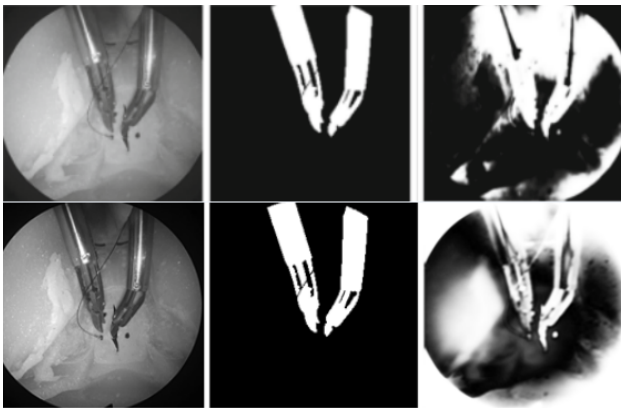

Figura 5.1: Imagem de entrada, label e resultado da saída da etapa de validação, respectivamente. Primeira linha representa o trabalho de (PEREZ,2020), segunda linha representa o resultado do **primeiro modelo** do autor . Fonte. Autor. (2021) e (PEREZ,2020)

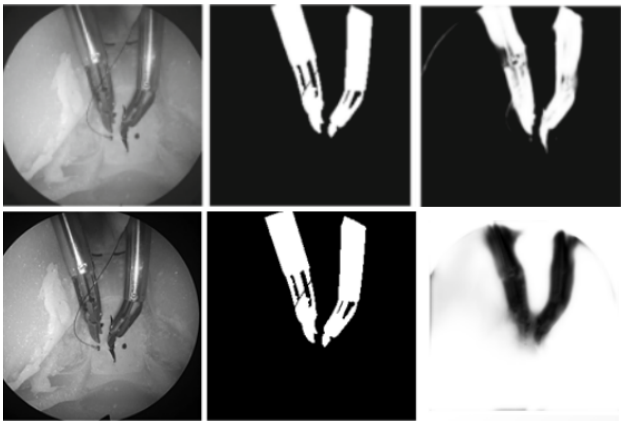

Figura 5.2: Imagem de entrada, label e resultado da saída da etapa de validação, respectivamente. Primeira linha representa o trabalho de (PEREZ,2020), segunda linha representa o resultado do **segundo modelo** do autor . Fonte. Autor. (2021) e (PEREZ,2020)

5.2 Perspectivas Futuras

É evidente que as imagens simuladas como entrada possuem um limite de precisão para o modelo, assim, com o uso de imagens reais em grandes quantidades o modelo será capaz de se tornar mais preciso e preparado para situações que ocorrem na realidade, uma vez que a simulação não contempla posições inesperadas que acabam por ocorrer e, dado a criticidade do problema que se deseja resolver (automação de camera em operações de laparoscopia), é necessário que o modelo tenha uma assertividade muito alta e que tenha um controle para momentos onde o modelo não tenha certeza da posição do aparelho.

Uma possibilidade de treino não implementada é a mesclagem das imagens de nível básico com as imagens complexas, dessa forma, o modelo pode vir a levar menos tempo para treinar e ainda ser capaz de realizar boas predições. O tempo limitado para elaboração deste projeto não permitiu que este autor realize determinado teste, todavia, aos que desejarem melhorar a eficácia do modelo, este é um ótimo ponto de partida.

Embora a etapa de treino leve um tempo consideravel, aproximadamente 6 horas, quando finalizado o modelo pode ser exportado e inserido nas mais diversas aplicações, ou seja, para seguir adiante com a pesquisa é necessário apenas exportar o modelo para o hardware de preferência, do qual, mesmo que não possua uma gpu, este será capaz de realizar as predições em tempo real. Caso novas etapas de treino sejam realizadas, por exemplo treino com imagens reais , a melhor solução para a redução do tempo de treinamento é a adequação da máquina onde será executado. O uso de gpus e alta quantidade de memória RAM são primordiais, portanto, caso não se deseje o uso do Google Colab é primordial uma máquina de alta capacidade de processamento de imagens.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ANEEL. *Agência Nacional de Energia Elétrica*. [S.l.]: Disponível em: `https://www.aneel.gov.br/`, keywords = Acessado: 01-11-2021.

[2] ANTEBY, R. et al. Deep learning visual analysis in laparoscopic surgery: a systematic review and diagnostic test accuracy meta-analysis. *Surgical Endoscopy*, v. 35, p. 1521–1533, jan. 2021.

[3] ALPISTE, I. M. et al. Benchmarking machine-learning-based object detection on a uav and mobile platform. *IEEE Wireless Communications and Networking Conference (WCNC)*, jan. 2019.

[4] CHENG, C.-H. A real-time robot-arm surgical guiding system development by image-tracking. *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, p. 133–133, jan. 2020.

[5] CHOI, B. et al. Surgical-tools detection based on convolutional neural network in laparoscopic robot-assisted surgery. *IEEE Engineering in Medicine and Biology Society, EMBS*, p. 1756–1759, jan. 2017.

[6] DERTAT, A. *TOWARDS DATA SCIENCE. Applied Deep Learning.* [S.l.]: Disponível em: `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`, keywords = Acessado: 13-10-2020.

[7] DECOM. *Fundamentos de Redes Neurais*. [S.l.]: Disponível em: `http://www2.decom.ufop.br/imobilis/fundamentos-de-redes-neurais/`, keywords = Acessado: 12-11-2021.

[8] FENAM. *Federação Nacional dos Médicos*. [S.l.]: Disponível em: `http://www.fenam.org.br/`, keywords = Acessado: 13-11-2021.

[9] GODOY, D. *TOWARDS DATA SCIENCE. understanding binary cross entropy.* [S.l.]: Disponível em: `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`, keywords = Acessado: 27-10-2021.

[10] GOOGLE.COM. *GOOGLE COLAB.* [S.l.]: Disponível em: `https://colab.research.google.com/`, keywords = Acessado entre: 01-09-2021 e 25-10-2021.

[11] GRIGOROIU, A.; YOON, J.; BOHNDIEK, S. E. Deep learning applied to hyperspectral endoscopy for online spectral classifcation. *Scientific Reports*, v. 10, p. 1–10, jan. 2020.

[12] KAGGLE. *Kaggle*. [S.l.]: Disponível em: `https://www.kaggle.com/`, keywords = Acessado em: 27-10-2021 .

[13] KLETZ, S. et al. Identifying surgical instruments in laparoscopy using deep learning instance segmentation. *Proceedings - International Workshop on Content-Based Multimedia Indexing*, jan. 2019.

[14] KERAS.IO. *Layers actvations*. [S.l.]: Disponível em: `https://keras.io/`, keywords = Acessado em: 27-10-2021 .

[15] LITJENS, G. et al. A survey on deep learning in medical image analysis. *Medical Image Analysis*, v. 42, p. 60–88, jan. 2017.

[16] LEE, E.-J. et al. Weakly supervised segmentation for real-time surgical tool tracking. *Healthcare Technology Letters*, v. 6, p. 231–236, jan. 2019.

[17] MISHRA, K.; SATHISH, R.; SHEET, D. Learning latent temporal connectionism of deep residual visual abstractions for identifying surgical tools in laparoscopy procedures. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, p. 2233–2240, jul. 2017.

[18] MATSUNAGA, Y.; NAKAMURA, R. Analysis of hemostasis procedures through machine learning of endoscopic images towards automatic surgery. *Sensors and Materials*, v. 32, p. 947–958, jan. 2020.

[19] NIEVES, D. et al. Low-level event detection system for minimally-invasive surgery training. *Proceedings of the 4th international Workshop on Sensor-based Activity Recognition and Interaction*, v. 5, p. 1–6, jan. 2017.

[20] NETO, J. de A. P.; LACOMBE, D. Cirurgia laparoscópica vídeo-assistida com acesso manual combinado: Estudo randomizado comparativo com laparotomia. *UFRJ - Rio de Janeiro*, jan. 2003.

[21] PEREZ, S. A. H. et al. The effects of different levels of realism on the training of cnns with only synthetic images for the semantic segmentation of robotic instruments in a head phantom. *International Journal of Computer Assisted Radiology and Surgery*, v. 15, p. 1257–1265, jan. 2020.

[22] POZDEEV, A. A.; OBUKHOVA, N. A.; MOTYKO, A. A. Anatomical landmarks detection for laparoscopic surgery based on deep learning technology. *Proceedings of the 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering*, p. 1668–1672, jan. 2021.

[23] PENG, K. S.; HONG, M.; ROZENBLIT, J. Single shot state detection in simulation-based laparoscopy training. *Society for Modeling Simulation International (SCS)*, v. 51, jan. 2019.

[24] RAVAUT, M. et al. Predicting adverse outcomes due to diabetes complications with machine learning using administrative health data. *Digital Medicine*, p. 4–24, jan. 2021.

[25] RSAUDE. *Laparoscopia o que é ?* [S.l.]: Disponível em: `https://rsaude.com.br/apucarana/materia/laparoscopia-o-que-e/15684.`, keywords = Acessado: 01-11-2021.

[26] RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. *Springer International Publishing Switzerland 2015*, jan. 2015.

[27] SILVA, R. S. D. *PRINCÍPIOS EM VIDEOCIRURGIA*. [S.l.]: ArtMed, 2006. (Silva, De Carli Cols.).

[28] SIMES. *Grandes invenções da Medicina: você sabe quando a cirurgia robótica foi inventada?* [S.l.]: Disponível em: `https://www.simes.org.br/novosite/noticias_exibir.php?noticia=438`, keywords = Acessado: 01-11-2021.

[29] SEBASTIAN. *Why the RELU function not differentiable im x= 0?* [S.l.]: Disponível em: `https://sebastianraschka.com/faq/docs/relu-derivative.html`, keywords = Acessado em: 12-11-2021 .

[30] TOKUYASU, T. et al. Development of an artifcial intelligence system using deep learning to indicate anatomical landmarks during laparoscopic cholecystectomy. *Surgical Endoscopy*, v. 35, p. 1651–1658, jan. 2021.

[31] TENSORFLOW. *TensorFlow*. [S.l.]: Disponível em: `https://www.tensorflow.org/?hl=pt-br`, keywords = Acessado em: 27-10-2021 .

[32] WIKIPEDIA. *Sigmoid Function.* [S.l.]: Disponível em: `https://en.wikipedia.org/wiki/Sigmoid_function`, keywords = Acessado: 12-11-2021.

[33] ZADEH, S. M. et al. Surgai: deep learning for computerized laparoscopic image understanding in gynaecology. *Surgical Endoscopy*, v. 34, p. 5377–5383, jan. 2020.

Anexo

Tabela de artigos

Artigo Nome:	Método Utilizado:	Ano:	Referência:	Resumo:
Surgical-tools Detection based on Convolutional Neural Network in Laparoscopic Robot-assisted Surgery	Rede Neural Convolucional (CNN do inglês)	2017	(CHOI,2017)	É proposto modelos em tempo real capazes de realizar a detecção de até 7 instrumentos em uma cirurgia de laparoscopia via inteligência artificial com uso da CNN.
Anatomical Landmarks Detection for Laparoscopic Surgery Based on Deep Learning Technology	Rede Neural Convolucional (CNN do inglês)	2021	(POZDEEV,2021)	É proposto o método de detecção de pontos de referência para laparoscopia com o uso de imagens fluorescentes via CNN.
A survey on deep learning in medical image analysis	Rede Neural Convolucional (CNN do inglês)	2017	(LITJENS,2017)	Analise dos conceitos mais relevantes de CNN's para uso em análise de imagens médicas, trabalho leva em conta mais de 300 trabalhos publicados.
Deep learning applied to hyperspectral endoscopy for online spectral classifcation	Rede Neural Convolucional (CNN do inglês)	2020	(GRIGOROIU,2020)	Rede CNN é utilizada para classificação online de dados obtidos durante a endoscopia com imagem hiperespectral.
Development of an artifcial intelligence system using deep learning to indicate anatomical landmarks during laparoscopic cholecystectomy	YOLOv3	2021	(TOKUYASU,2021)	Desenvolvimento de um sistema que realiza o delineamento de 4 pontos de referência, relevantes para evitar a ocorrência de lesão do ducto biliar quando realizando colecistectomia laparoscópica

Learning Latent Temporal Connectionism of Deep Residual Visual Abstractions for Identifying Surgical Tools in Laparoscopy Procedures	Rede Neural Convolucional (CNN do inglês)	2017	(MISHRA,2017)	Método que aprende a detectar via CNN, a presença de objetos em vídeos de laparoscopia, aprendendo as relações de longa e curta ordem das características visuais espaciais extraídas do vídeo cirúrgico.
single shot state detection in simulation-based laparoscopy training	Rede Neural Convolucional (CNN do inglês)	2019	(PENG,2019)	A contribuição desta pesquisa é apresentar um modelo colaborando com um detector de objetos de aprendizagem profunda, que pode ser aplicado ao simulador de treinamento cirúrgico, bem como outros sistemas de detecção visual e automação.
Identifying Surgical Instruments in Laparoscopy Using Deep Learning Instance Segmentation	Rede Neural Convolucional (CNN do inglês)	2019	(KLETZ,2019)	Investigação e reconhecimento de instrumentos cirúrgicos em vídeos gravados de ginecologia laparoscópica.
Analysis of Hemostasis Procedures through Machine Learning of Endoscopic Images towards Automatic Surgery	SVM Linear	2019	(LEE,2019)	Analise de procedimentos de hemostasia por detecção de região por meio de machine learning (SVM).
Weakly supervised segmentation for real-time surgical tool tracking	Rede Neural Convolucional (CNN do inglês)	2019	(LEE,2019)	Proposta de método fracamente supervisionado para segmentação e rastreamento de ferramentas cirúrgicas com base em sistemas de sensores híbridos.

Low-level Event Detection System for Minimally-Invasive Surgery Training	Rede Neural Convolucional (CNN do inglês)	2017	(NIEVES,2017)	Sistema de detecção de eventos em uma cirurgia laparoscópica, como parte de um projeto mais ambicioso de supervisão por observação.
Deep learning visual analysis in laparoscopic surgery: a systematic review and diagnostic test accuracy meta analysis	Rede Neural Convolucional (CNN do inglês)	2021	(ANTEBY,2021)	Avaliação se redes de aprendizagem profunda são capazes de analisar com precisão vídeos de procedimentos laparoscópicos.
SurgAI: deep learning for computerized laparoscopic image understanding in gynaecology	Rede Neural Convolucional (CNN do inglês)	2020	(ZADEH,2020)	Elaboração e aplicação de conjunto de dados dedicado à detecção baseada em imagens de ginecologia.
A Real-Time Robot-Arm Surgical Guiding System Development by Image-Tracking	YOLO	2020	(CHENG,2020)	Aplicação de técnicas de identificação e localização de instrumentos cirúrgicos para treinamento médico.
The effects of different levels of realism on the training of CNNs with only synthetic images for the semantic segmentation of robotic instruments in a head phantom	Rede Neural Convolucional (CNN do inglês)	2020	(PEREZ,2020)	Investigação de diferentes níveis de realismo no treinamento de redes neurais profundas para segmentação de instrumentos robóticos.

Tabela 1: Tabela com todos os artigos selecionados.

Algoritmo

```
import time
import tensorflow as tf
from multiprocessing import Process
import pandas as pd
import matplotlib.pyplot as plt
import yaml
import pathlib
import random
import numpy  as np
import cv2
import os
import scipy.io as sio
import PIL
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Input, Conv2DTranspose, concatenate
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.preprocessing.image import ImageDataGenerator
from google.colab.patches import cv2_imshow
```

Importando bibliotecas. Fonte: Autor. (2021).

```
[ ]  class aumento_imagens_rotacao:

       def __init__(self,angulo_min, angulo_max):
           self.angulo_min = angulo_min
           self.angulo_max = angulo_max

       def aumento_dados(self,imagem,label):
         angulo = random.randint(self.angulo_min, self.angulo_max)
         (altura, largura) = imagem.shape[:2]
         (cX, cY) = (largura // 2, altura // 2)

         Matriz = cv2.getRotationMatrix2D((cX, cY), angulo, 1.0)
         img = cv2.warpAffine(imagem, Matriz, (largura,altura))

         lbl = cv2.warpAffine(label, Matriz, (largura,altura))

         return img, lbl
```

Implementação da função aumento_dados por rotação. Fonte: Autor. (2021).

```
[ ]  class aumento_imagens_fliping:
       def aumento_dados(self,imagem,label):

           img = cv2.flip(imagem,1)
           lbl = cv2.flip(label,1)

           return img, lbl
```

Implementação da função aumento_dados por flipping. Fonte: Autor. (2021).

```
def _batch(index,
           batch_size,
            numero_amostras,
            caminhos_imagens_treino,
            caminhos_labels_treino,
            largura_imagem,
            altura_imagem,
            aumento_imgs):


    for batch_contador in range(0, batch_size):



        imagem_caminho = str(caminhos_imagens_treino[index])
        label_caminho = str(caminhos_labels_treino[index])
        index += 1
        if index >= numero_amostras:
            index = 0


        imagem = abrir_imagem(imagem_caminho,largura_imagem,altura_imagem)

        label = abrir_imagem(label_caminho,largura_imagem,altura_imagem)
```

Implementação da função _batch parte 1. Fonte: Autor. (2021).

```
[ ]
            imagem = abrir_imagem(imagem_caminho,largura_imagem,altura_imagem)

            label = abrir_imagem(label_caminho,largura_imagem,altura_imagem)


            for aumento_img in aumento_imgs:

              imagem, label = aumento_img.aumento_dados(imagem, label)

            label[label > 0.5] = 1.0
            label[label <= 0.5] = 0.0

            if batch_contador == 0:
                imagem_batch = imagem
                label_batch = label
            else:
                imagem_batch = np.concatenate((imagem_batch, imagem), axis=0)
                label_batch = np.concatenate((label_batch, label), axis=0)

        (altura, largura) = imagem.shape



        return index, \
               imagem_batch.reshape(batch_size, altura, largura, 1), \
               label_batch.reshape(batch_size, altura, largura, 1)
```

Implementação da função _batch parte 2. Fonte: Autor. (2021).

```
[ ]  def gerador_imagens(caminho_imagem,
                         caminho_label,
                         largura_imagem,
                         altura_imagem,
                         batch_size,
                         aumento_imgs):

        diretorio_imagens_treino = pathlib.Path(caminho_imagem)
        caminhos_imagens_treino = list(diretorio_imagens_treino.glob('*.png'))

        tamanho = len(caminhos_imagens_treino)

        diretorio_labels_treino = pathlib.Path(caminho_label)
        caminhos_labels_treino = list(diretorio_labels_treino.glob('*.png'))


        if len(caminhos_labels_treino) != len(caminhos_imagens_treino):
            print(len(caminhos_labels_treino))
            print(len(caminhos_imagens_treino))
            raise Exception('O numero de imagens e labels não são iguais')

        numero_amostras = len(caminhos_imagens_treino)
        dim = (largura_imagem, altura_imagem)

        index = 0
        while True:
```

Implementação da função gerador_imagens parte 1. Fonte: Autor. (2021).

```
        numero_amostras = len(caminhos_imagens_treino)
        dim = (largura_imagem, altura_imagem)

        index = 0
        while True:


                index, imagem_batch, label_batch = _batch(
                    index = index,
                    batch_size=batch_size,
                    numero_amostras=numero_amostras,
                    caminhos_imagens_treino=caminhos_imagens_treino,
                    caminhos_labels_treino=caminhos_labels_treino,
                    largura_imagem = largura_imagem,
                    altura_imagem = altura_imagem,
                    aumento_imgs=aumento_imgs)


                yield imagem_batch, label_batch
```

Implementação da função gerador_imagens parte 2. Fonte: Autor. (2021).

```
def treinando_modelo(configuracao, aumento_imgs):

    dim = (configuracao['LARGURA_IMAGEM'],configuracao['ALTURA_IMAGEM'])

    imagem_tam = (1,) + dim + (1,)

    modelo = _modelo()

    treino_diretorio_imagem = configuracao['TREINAMENTO_CAMINHO'] + 'image'

    treino_diretorio_label = configuracao['TREINAMENTO_CAMINHO'] + 'label'

    validacao_diretorio_imagem = configuracao['VALIDACAO_CAMINHO'] + 'image'

    validacao_diretorio_label = configuracao['VALIDACAO_CAMINHO'] + 'label'

    caminho_validacao = pathlib.Path(validacao_diretorio_imagem)

    caminhos_validacao = list(caminho_validacao.glob('*.png'))

    numero_amostras = len(caminhos_validacao)
```

Implementação da função treinando_modelo parte 1. Fonte: Autor. (2021).

```
numero_amostras = len(caminhos_validacao)

treino = gerador_imagens(treino_diretorio_imagem,
                         treino_diretorio_label,
                         configuracao['LARGURA_IMAGEM'],
                         configuracao['ALTURA_IMAGEM'],
                         configuracao['BATCH_SIZE'],
                         aumento_imgs)


validacao = gerador_imagens(validacao_diretorio_imagem,
                            validacao_diretorio_label,
                            configuracao['LARGURA_IMAGEM'],
                            configuracao['ALTURA_IMAGEM'],
                            1,
                            aumento_imgs=[])

lista_historico_loss = []
lista_historico_val_loss = []
lista_historico_bin_acc = []
lista_historico_mean_io = []
lista_historico_val_loss = []
lista_historico_val_bin_acc = []
lista_historico_val_mean_io = []
```

Implementação da função treinando_modelo parte 2. Fonte: Autor. (2021).

```
for i in range (0,configuracao['EPOCAS']):



    Historico = modelo.fit(treino,
                steps_per_epoch=configuracao['PASSOS_POR_EPOCA'],
                epochs=1,
                validation_data=validacao,
                validation_steps = numero_amostras)


    treino_imagem,treino_label = next(treino)

    validacao_imagem,validacao_label = next(validacao)

    resultado = modelo.predict(treino_imagem[0].reshape(imagem_tam))

    verdadeiro = treino_label[0].reshape(imagem_tam)

    entrada = treino_imagem[0].reshape(imagem_tam)

    resultado_validacao = modelo.predict(validacao_imagem[0].reshape(imagem_tam))

    verdadeiro_validacao = validacao_label[0].reshape(imagem_tam)

    entrada_validacao = validacao_imagem[0].reshape(imagem_tam)
```

Implementação da função treinando_modelo parte 3. Fonte: Autor. (2021).

```
    entrada_validacao = validacao_imagem[0].reshape(imagem_tam)

    hora_da_verdade(resultado,resultado_validacao,entrada,entrada_validacao,verdadeiro,verdadeiro_validacao,i)

    lista_historico_loss.append(Historico.history['loss'])
    lista_historico_bin_acc.append(Historico.history['binary_accuracy'])
    lista_historico_mean_io.append(Historico.history['mean_io_u'])
    lista_historico_val_loss.append(Historico.history['val_loss'])
    lista_historico_val_bin_acc.append(Historico.history['val_binary_accuracy'])
    lista_historico_val_mean_io.append(Historico.history['val_mean_io_u'])

modelo.save("modelo_1.h5")
return lista_historico_loss, lista_historico_bin_acc, lista_historico_mean_io, lista_historico_val_loss, lista_historico_val_bin_acc,lista_historico_val_mean_io
```

Implementação da função treinando_modelo parte 4. Fonte: Autor. (2021).

```
def abrir_imagem(caminho_imagem,largura_imagem, altura_imagem ):

  imagem = cv2.imread(caminho_imagem,cv2.IMREAD_UNCHANGED)

  dim = (largura_imagem, altura_imagem)

  imagem = cv2.resize(imagem,dim, interpolation = cv2.INTER_AREA)

  if len(imagem.shape) >= 3:
    raise ValueError("Imagem inadequada.")

  imagem = imagem/255

  return imagem
```

Implementação da função abrir_imagem. Fonte: Autor. (2021).

```
def array_para_imagem(imagem):

  uint = np.array(imagem*255).astype('uint8')

  imagem_cinza = cv2.cvtColor(uint,cv2.COLOR_GRAY2BGR)

  return imagem_cinza
```

Implementação da função array_para_imagem. Fonte: Autor. (2021).

```
def _modelo(tamanho_entrada =(256, 256, 1)):


    entrada = tf.keras.Input(tamanho_entrada)
    camada = Conv2D(32, 3, activation='relu', padding='same')(entrada)
    concat = Conv2D(32, 3, activation='relu', padding='same')(camada)

    camada = MaxPooling2D(pool_size=(2, 2))(concat)
    camada = Conv2D(64, 3, activation='relu', padding='same')(camada)
    concat1 = Conv2D(64, 3, activation='relu', padding='same')(camada)

    camada = MaxPooling2D(pool_size=(2, 2))(concat1)
    camada = Conv2D(128, 3, activation='relu', padding='same')(camada)
    concat2 = Conv2D(128, 3, activation='relu', padding='same')(camada)

    camada = MaxPooling2D(pool_size=(2, 2))(concat2)
    camada = Conv2D(256, 3, activation='relu', padding='same')(camada)
    concat3 = Conv2D(256, 3, activation='relu', padding='same')(camada)

    camada = MaxPooling2D(pool_size=(2, 2))(concat3)
    camada = Conv2D(512, 3, activation='relu', padding='same')(camada)
    camada = Conv2D(512, 3, activation='relu', padding='same')(camada)
    concat4 = Conv2D(512, 3, activation='relu', padding='same')(camada)
```

Implementação da função _modelo parte 1. Fonte: Autor. (2021).

```
camada = MaxPooling2D(pool_size=(2, 2))(concat3)
camada = Conv2D(512, 3, activation='relu', padding='same')(camada)
camada = Conv2D(512, 3, activation='relu', padding='same')(camada)
concat4 = Conv2D(512, 3, activation='relu', padding='same')(camada)

camada = MaxPooling2D(pool_size=(2, 2))(concat4)
camada = Conv2D(1024, 3, activation='relu', padding='same')(camada)

camada = Conv2DTranspose(512, 2, strides=(2, 2), activation='relu', padding='same')(camada)
camada = concatenate([camada, concat4], axis=3)

camada = Conv2D(512, 3, activation='relu', padding='same')(camada)
camada = Conv2D(512, 3, activation='relu', padding='same')(camada)
camada = Conv2D(512, 3, activation='relu', padding='same')(camada)

camada = Conv2DTranspose(256, 2, strides=(2, 2), activation='relu', padding='same')(camada)
camada = concatenate([camada, concat3], axis=3)

camada = Conv2D(256, 3, activation='relu', padding='same')(camada)
camada = Conv2D(256, 3, activation='relu', padding='same')(camada)

camada = Conv2DTranspose(128, 2, strides=(2, 2), activation='relu', padding='same')(camada)
camada = concatenate([camada, concat2], axis=3)
```

Implementação da função _modelo parte 2. Fonte: Autor. (2021).

```
camada = Conv2DTranspose(128, 2, strides=(2, 2), activation='relu', padding='same')(camada)
camada = concatenate([camada, concat2], axis=3)

camada = Conv2D(128, 3, activation='relu', padding='same')(camada)
camada = Conv2D(128, 3, activation='relu', padding='same')(camada)

camada = Conv2DTranspose(64, 2, strides=(2, 2), activation='relu', padding='same')(camada)
camada = concatenate([camada, concat1], axis=3)

camada = Conv2D(64, 3, activation='relu', padding='same')(camada)
camada = Conv2D(64, 3, activation='relu', padding='same')(camada)

camada = Conv2DTranspose(32, 2, strides=(2, 2), activation='relu', padding='same')(camada)
camada = concatenate([camada, concat], axis=3)

camada = Conv2D(32, 3, activation='relu', padding='same')(camada)
camada = Conv2D(32, 3, activation='relu', padding='same')(camada)

saida = Conv2D(1, 1, padding='same', activation='sigmoid')(camada)

modelo = tf.keras.Model(inputs=entrada, outputs=saida)
```

Implementação da função _modelo parte 3. Fonte: Autor. (2021).

```
modelo.compile(optimizer=tf.keras.optimizers.SGD(learning_rate= 0.02),
               loss='binary_crossentropy',
               metrics=[
                   tf.keras.metrics.BinaryAccuracy(),
                   tf.keras.metrics.MeanIoU(num_classes = 2)
               ]
               )
modelo.summary()
return modelo
```

Implementação da função _modelo parte 4. Fonte: Autor. (2021).

```
def hora_da_verdade( resultado, validacao_resultado, entrada,entrada_validacao,verdadeiro,validacao_verdadeiro, i):

  dim = (configuracao['LARGURA_IMAGEM'],configuracao['ALTURA_IMAGEM'])
  imagem_tam = (1,) + dim + (1,)
  img = array_para_imagem(verdadeiro.reshape(dim ))
  img2 = array_para_imagem(entrada.reshape(dim ))
  img3 = array_para_imagem(validacao_verdadeiro.reshape(dim ))
  img4 = array_para_imagem(entrada_validacao.reshape(dim ))
  plt.imsave(str(configuracao['RESULTADO_CAMINHO']+(str(i)+'_modelo_saida.png')), resultado.reshape(dim), cmap=plt.cm.binary)
  plt.imsave(str(configuracao['RESULTADO_CAMINHO']+(str(i)+'_verdadeiro.png')), img)
  plt.imsave(str(configuracao['RESULTADO_CAMINHO']+(str(i)+'_entrada.png')), img2, cmap=plt.cm.binary)
  plt.imsave(str(configuracao['RESULTADO_CAMINHO']+(str(i)+'_entrada_modelo.png')), img4, cmap=plt.cm.binary)
  plt.imsave(str(configuracao['RESULTADO_CAMINHO']+(str(i)+'_modelo_saida_validacao.png')), validacao_resultado.reshape(dim), cmap=plt.cm.binary)
  plt.imsave(str(configuracao['RESULTADO_CAMINHO']+(str(i)+'_modelo_verdadeiro.png')), img3)
```

Implementação da função Hora_da_verdade. Fonte: Autor. (2021).

```
configuracao_yaml = open("configuracao.yml")
configuracao = yaml.load(configuracao_yaml)


aumento_imgs = [
    #aumento_imagens_translacao(-60, 60, -30, 30),
    aumento_imagens_rotacao(-30,30),
    aumento_imagens_fliping()

    ]


lista_historico_loss,lista_historico_bin_acc,lista_historico_mean_io,lista_historico_val_loss,lista_historico_val_bin_acc,
lista_historico_val_mean_io = treinando_modelo(configuracao,aumento_imgs)
```

Implementação da main parte 1. Fonte: Autor. (2021).

```
plt.plot(lista_historico_loss)
plt.plot(lista_historico_val_loss)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('model_loss.png')
plt.show()
```

Implementação da main parte 2. Fonte: Autor. (2021).

```
plt.plot(lista_historico_bin_acc)
plt.plot(lista_historico_val_bin_acc)
plt.title('bin acc')
plt.ylabel('bin_acc')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('bin_acc.png')
plt.show()
```

Implementação da main parte 3. Fonte: Autor. (2021).

```
plt.plot(lista_historico_mean_io)
plt.plot(lista_historico_val_mean_io)
plt.title('mean io')
plt.ylabel('mean_io')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('mean_io.png')
plt.show()
```

Implementação da main parte 4. Fonte: Autor. (2021).