UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Carlos Eduardo Angelucci Marchiori

Aplicação de uma Rede Neural de Convolução para Posicionamento de Seccionadora em Linha de Produção

Blumenau
2023

Carlos Eduardo Angelucci Marchiori

Aplicação de uma Rede Neural de Convolução para Posicionamento de Seccionadora em Linha de Produção

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.
Orientador: Prof. Orientador, Dr. Mauri Ferradin

Blumenau
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Angelucci Marchiori, Carlos Eduardo
Aplicação de uma rede neural de convolução para posicionamento de seccionadora em linha de produção / Carlos Eduardo Angelucci Marchiori ; orientador, Mauri Ferrandin, 2023.
68 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Aprendizado de Máquina. 3. Visão Computacional. 4. Posicionamento de Peça. I. Ferrandin, Mauri. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Carlos Eduardo Angelucci Marchiori

Aplicação de uma Rede Neural de Convolução para Posicionamento de Seccionadora em Linha de Produção

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Engenheiro de Controle e Automação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 29 de Junho de 2023.

Banca Examinadora:

Documento assinado digitalmente
Mauri Ferrandin
Data: 12/07/2023 14:30:28-0300
CPF: ***.580.869-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Mauri Ferrandin
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Adao Boava
Data: 12/07/2023 14:42:51-0300
CPF: ***.640.969-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Adao Boava
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Marcos Vinicius Matsuo
Data: 12/07/2023 15:30:35-0300
CPF: ***.580.029-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Marcos Vinícius Matsuo
Universidade Federal de Santa Catarina

Este trabalho é dedicado a minha família e aos meus amigos

AGRADECIMENTOS

Gostaria de agradecer à minha família, em especial à minha mãe Lucimar e ao meu irmão Pedro, por terem norteado os meus planos no caminho dos estudos. A minha mãe que abriu mão da sua juventude para cuidar de nós e garantir que nada nos faltasse, espero poder retribuir e prover um futuro com tudo que ela deveria ter vivido durante esses anos de sacrifícios por nós.

Aos meus grandiosos amigos Daniel Rodrigues Pacheco Coutinho e Igor Semke Chagas, por terem proporcionado uma trajetória inesquecível, pelos ombros estendidos durante os momentos de incerteza e pela parceria nos momentos de responsabilidade, diversão, dificuldades e, principalmente, nos momentos de alegria.

Agradeço ao professor Dr. Mauri Ferrandin pela orientação. Foi um privilégio assistir às aulas de um professor extremamente capacitado, não só em termos de conhecimento, mas também em termos de capacidade de ministrar as aulas com uma didática inclusiva.

E, por último, mas não menos importante, gostaria de agradecer a mim por todas as noites mal dormidas, pela resiliência e dedicação.

"Conheça todas as teorias, domine todas as técnicas, mas ao tocar uma alma humana seja apenas outra alma humana"(JUNG, Carl)

RESUMO

Este trabalho tem como objetivo aplicar uma rede neural de convolução para auxiliar no posicionamento de uma seccionadora no local de teste da mesma. Com o avanço da inteligência artificial, tem-se observado o surgimento de novas aplicações para esta área de conhecimento. O intuito da solução proposta é centralizar a eficiência do processo de posicionamento da seccionadora em uma solução de engenharia. Para isso, foi realizada uma revisão bibliográfica sobre o tema, bem como o estudo de aplicações que se assemelham. No desenvolvimento da solução proposta, foram estabelecidas quatro estruturas diferentes de rede neural de convolução e comparado o desempenho entre os quatro modelos, assim como a adição de características físicas no ambiente de teste da seccionadora a fim de auxiliar no aprendizado dos modelos durante as etapas de convolução. Os resultados obtidos demonstraram que um modelo de CNN pode obter um desempenho mais satisfatório do que um operador em termos de inferência sobre o posicionamento da peça, aumentando não só a eficiência do processo, mas também a segurança do mesmo. Conclui-se que a utilização de redes neurais de convolução é uma ferramenta capaz de auxiliar os operadores, garantindo a eficiência do processo e permitindo que os esforços humanos sejam direcionados para processos mais pertinentes.

Palavras-chave: Aprendizado de Máquina; Visão Computacional; Posicionamento de Peça.

ABSTRACT

This work aims to apply a convolutional neural network to assist in the positioning of a disconnector switch in its testing site. With the advancement of artificial intelligence, new applications for this field of knowledge have been observed. The purpose of the proposed solution is to centralize the efficiency of the positioning process of the piece into an engineering solution. To achieve this, a literature review on the topic was conducted, as well as a study of similar applications. In the development of the proposed solution, four different convolutional neural network structures were established and compared for performance, as well as the addition of physical characteristics in the disconnector switch testing environment to assist in the learning of the models during the convolution stages. The results showed that a CNN model can achieve more satisfactory performance than an operator in terms of inference on the positioning of the piece, increasing not only the efficiency of the process but also its safety. It is concluded that the use of convolutional neural networks is a tool capable of assisting operators, ensuring the efficiency of the process and allowing human efforts to be directed towards more pertinent processes.

Keywords: Machine Learning; Computer Vision; Piece Positioning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AI	*Artifical Intelligence*
CNN	*Convolutional Neural Network*
DL	*Deep Learning*
ETO	*Engineering to Order*
GAN	*Generative Adversarial Network*
GPU	*Graphic Processing Unit*
ML	*Machine Learning*
ReLU	*Rectified Linear Unit*
RNN	*Recurrent Neural Networks*

SUMÁRIO

1 INTRODUÇÃO

Uma questão recorrente na indústria reside na exigência de operadores capacitados em processos que poderiam ser monitorados de forma automatizada. A falta destes operadores significa maiores chances de falha e ineficiência do processo, afetando diretamente a produtividade da fábrica. Devido ao citado, é justificável a busca por soluções de engenharia que descentralizem este tipo de conhecimento do operador, e centralizem em sistemas que são confiáveis, constantes e menos suscetíveis a interferências externas.

Dentre as soluções de engenharia, o aprendizado de máquina é uma opção para minimizar este tipo de problema devido a sua vasta gama de aplicação. O aprendizado de máquina é uma subárea da inteligência artificial, e aprendizado profundo de máquina por sua vez é uma subárea do aprendizado de máquina, também conhecido como *deep learning*.

Deep learning é uma técnica que se baseia em redes neurais profundas e tem sido aplicado em uma variedade de tarefas, desde classificação de imagens e processamento de linguagem natural até detecção de anomalias. Devido ao seu desempenho satisfatório e o avanço nas pesquisas, o *deep learning* tem sido amplamente utilizado em diversas áreas e tem o potencial de reestruturar a maneira como utilizamos os dados, principalmente os não estruturados, que não possuem um formato definido e devido a isso, possuem um nível de complexidade maior no processamento e análise se comparado com dados estruturados.

O avanço do *deep learning* se deve ao fato da capacidade de processar esses tipos de dados não estruturados que vão desde imagens e sons até textos e vibrações. Portanto, aplicar soluções de *deep learning* em processos que são fontes de dados deste tipo pode garantir não só a eficiência do processo, mas também a segurança e escalabilidade do mesmo.

1.1 JUSTIFICATIVA

A empresa na qual a solução foi desenvolvida possui um programa de padronização de processos dividido por categorias, sendo uma delas relacionada a aplicação de inteligência artificial. Em um determinado período, a empresa é auditada e é atribuída uma pontuação que define a classificação da mesma, podendo ser básico, padrão, avançado ou experiente. O processo de inteligência artificial se enquadra em unidades que possuem processos considerados em nível avançado e experiente. Isso posto, havia a necessidade de desenvolver a primeira aplicação de inteligência artificial na fábrica visando garantir a padronização dos processos para alcançar a classificação avançada ou experiente. A maior parte da fábrica consiste em linhas do tipo ETO (*Engineering to Order*), ou seja, os produtos fabricados são customizados conforme a necessidade do cliente, em via disso, as etapas de produção podem variar, o que dificulta o desenvolvimento de uma solução de *Artifical Intellingence* (AI).

A ausência de parâmetro para auxiliar o posicionamento da seccionadora no local de teste foi reportado por um dos operadores da linha, e este processo é padronizado independentemente do nível de customização do painel elétrico, dito isso, foi proposto desenvolver uma solução através de um modelo de DL (*Deep Learning*) que fosse capaz de solucionar o problema e atender os requisitos do programa de padronização de processos da empresa.

1.2 ESTRUTURA DO LOCAL DE TESTE DA SECCIONADORA

É fundamental conhecer a estrutura do local de teste da seccionadora para compreender o desenvolvimento deste trabalho. O ambiente de teste consiste em uma esteira do tipo rolete na qual as seccionadoras são apoiadas, e em um lugar específico desta esteira é o local de teste conforme representado na Figura 1. Logo abaixo deste local de teste, há dois pistões pneumáticos que ao serem acionados mudam para a posição de avanço e devem encaixar nos contatos da seccionadora, além disso, acima do local de teste há um suporte para a fiação elétrica e para a câmera utilizada para a visão computacional.

1.2.1 Problemas com o teste

O problema principal que pode acontecer na realização do teste é o não fechamento do contato devido ao mau posicionamento da seccionadora. Como dito na seção anterior, a peça deve ficar posicionada logo acima dos pistões pneumáticos cujo a finalidade é subir e descer o contato a fim de energizar a peça para realizar o teste e desenergizar para manuseio da mesma. O posicionamento incorreto implica no não fechamento dos contatos, e, portanto, a impossibilidade de realizar o teste, impactando diretamente a eficiência da linha de produção.

1.3 OBJETIVO

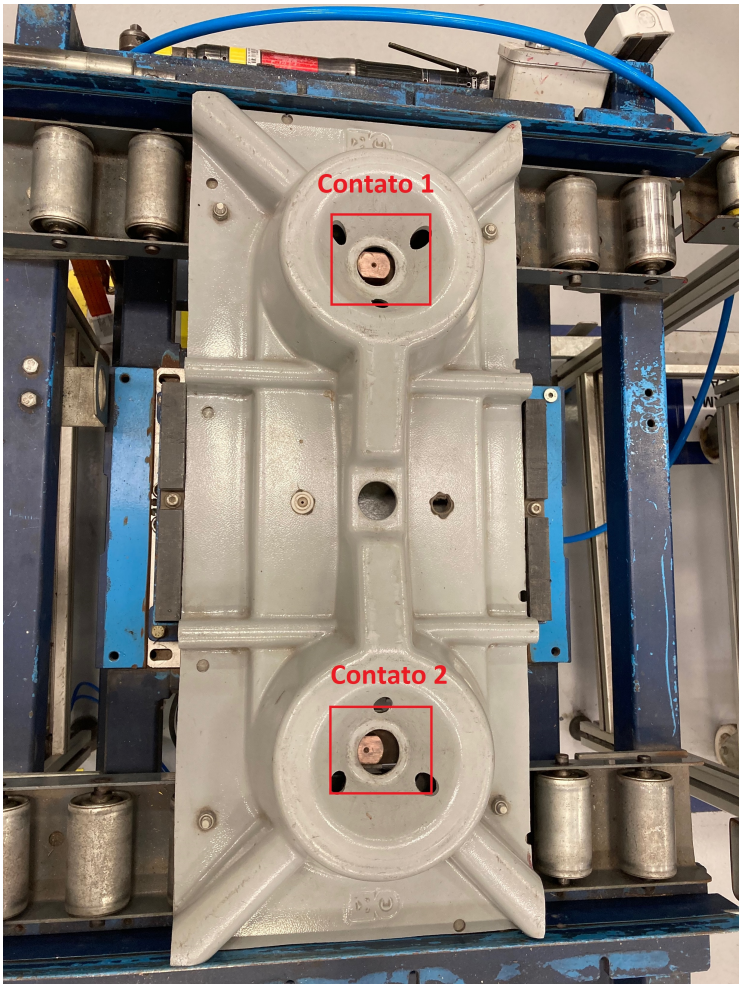
O objetivo deste trabalho é desenvolver uma solução que verifique de forma automática se a seccionadora foi posicionada de forma correta (ou não) no local de teste. Esta solução por sua vez, consiste em um modelo de *deep learning*, através de uma rede neural de convolução, capaz de classificar as imagens obtidas através da visão computacional entre posição correta e posições incorretas.

A fim de criar uma solução completa, foi desenvolvido um *endpoint* local através do micro *framework* Flask que recebe as imagens através de uma câmera e retorna as imagens com a classificação fornecida pelo modelo em tempo real, permitindo que o operador utilize a solução para garantir que a peça esteja posicionada no local correto.

Em suma, os objetivos específicos para o desenvolvimento desta solução são:

- Obter imagens da seccionadora em ambas as posições de interesse (posição correta e posição incorreta).

Figura 1 – Local de teste da seccionadora.



Fonte: O autor

- Treinar o modelo de DL (*Deep Learning*) para classificar as imagens.
- Avaliar os resultados obtidos em termos de acuracidade e eficiência.
- Integrar o modelo com o servidor local através do *framework* Flask.
- Utilizar a solução em um ambiente de produção.

O restante deste documento está estruturado da seguinte maneira: o Capítulo 2 apresenta a fundamentação teórica para compreensão do presente trabalho; o Capítulo 3 o desenvolvimento da solução proposta; o Capítulo 4 descreve os resultados obtidos da solução desenvolvida; e o Capítulo 5 a conclusão sobre o trabalho. Ao final, são apresentadas as referências bibliográficas utilizadas durante o desenvolvimento.

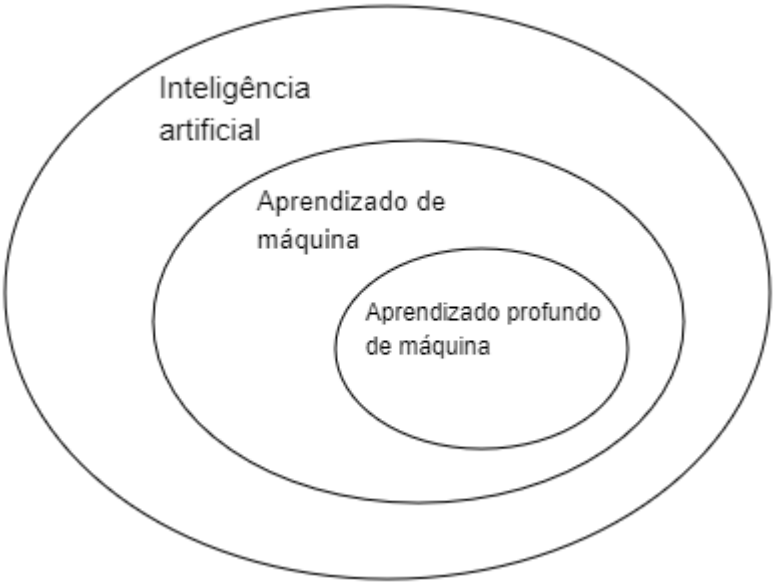
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo introduzir de maneira sucinta os conceitos que são fundamentais para a compreensão do presente trabalho. Faz-se uma breve introdução sobre inteligência artificial e as respectivas subáreas com ênfase em DL e, também, no micro framework Flask. Os conceitos de *deep learning* são apresentados com maior destaque, pois a solução desenvolvida foi baseada nas teorias da mesma.

2.1 INTELIGÊNCIA ARTIFICIAL

A inteligência artificial, do inglês *Artifical Intelligence*, pode ser definida como o esforço para automatizar tarefas intelectuais normalmente realizadas por humanos (FRANÇOIS CHOLLET, 2021), através desta definição geral, se torna justificável o fato das subáreas como ML (*Machine Learning*) e DL estarem contempladas nos tópicos relacionados a AI (*Artifical Intelligence*). A omissão do termo ”aprendizado” se deve ao fato

Figura 2 – Áreas da inteligência artificial.



Fonte: O autor.

de que em algumas aplicações de inteligência artificial consistem de um conglomerado de regras escritas por desenvolvedores de tal modo a prever todos ou quase todos os cenários possíveis que o programa em si possa se encontrar. Portanto, o programa não possui a capacidade de aprender com estes cenários, somente respondê-los baseado nas regras previamente escritas por seres humanos.

2.2 APRENDIZADO DE MÁQUINA

O aprendizado de máquina, do inglês *machine learning*, por sua vez, possui a capacidade de aprender com os cenários propostos, ou seja, diferente dos modelos de inteligência artificial que ao receber algum tipo de entrada retorna uma saída baseada nas regras previamente escritas, ele analisa as entradas e as respectivas saídas, e então, retorna as regras que devem existir baseados em modelos estatísticos. Com os dados certos, um modelo de ML pode analisar problemas altamente dimensionais para descobrir a função ideal que pode prever um resultado com uma determinada entrada. Os modelos de ML geralmente podem proporcionar segurança estatística sobre previsões e também sobre seu desempenho geral (AWS, 2023). Pode-se diferenciar as duas abordagens da seguinte maneira

$$dados + regras = respostas \tag{1}$$

e

$$dados + respostas = regras \tag{2}$$

Sendo a Equação (1) referente a programação clássica de um modelo de inteligência artificial e a Equação (2) referente ao modelo de *machine learning*.

2.3 APRENDIZADO DE MÁQUINA SUPERVISIONADO

O aprendizado de máquina supervisionado é um tipo de abordagem utilizada em casos como classificação ou regressão, ou seja, na relação que há entre variáveis dependentes e variáveis independentes. No aprendizado supervisionado os dados de treino são fornecidos para o algoritmo com os resultados desejados, chamados de categorias (AURÉLIEN GÉRON, 2017), também conhecido como *labels* ou rótulos. De maneira sucinta, o algoritmo de aprendizado tem como dados de treinamento pares de entrada e saída, sendo x a entrada e y a saída também conhecida como categoria, e ao associar as características dos valores de entrada x com as saídas y, o algoritmo se torna capaz de prever ou classificar a saída y ao receber um valor aleatório de x.

Existem diversos métodos para classificação, os quais são aplicáveis em diferentes contextos, como classificadores lineares, árvores de decisão, floresta aleatória entre outros que são utilizados para situações que possuem um número finito de possibilidade de resposta. Das aplicações do tipo de regressão as mais comuns são regressão linear, regressão logística e regressão polinomial e são utilizadas em situações em que a resposta pode ser um número qualquer não estando limitado a categorias previamente descritas.

2.4 APRENDIZADO DE MÁQUINA NÃO SUPERVISIONADO

O aprendizado não supervisionado elimina a necessidade de dados rotulados e da etapa de *feature engineering*, permitindo métodos de ML mais flexíveis e automatizados (DRIDI, 2021). Diferente do aprendizado de máquina supervisionado, o aprendizado não supervisionado tem como dados de treinamento somente os valores de entrada x, cujo objetivo do algoritmo de aprendizado é encontrar padrões nesta base de dados, de tal forma a estabelecer critérios de classificação para os valores de entrada.

Dentre os tipos de aprendizado não supervisionado de máquina, há a principal abordagem do tipo *Clustering* que agrupa dados similares, detecção de anomalias cujo objetivo é encontrar dados que não fazem parte do padrão determinado pelo algoritmo e por último redução de dimensionalidade que permite comprimir uma base da dados grande em uma base menor de tal forma a manter as informações pertinentes à base de dados em questão.

2.5 APRENDIZADO PROFUNDO DE MÁQUINA

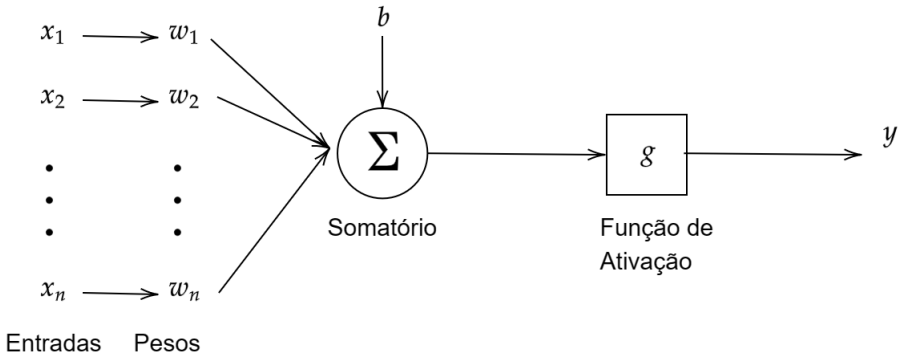
Deep learning utiliza redes neurais artificiais para identificar padrões complexos em dados não estruturados, devido a sua capacidade em identificar estes padrões, esta subárea da inteligência artificial possui extensa aplicabilidade em tarefas como reconhecimento de fala, processamento visual de dados e processamento de linguagem natural. *Deep learning* é a área que permite tornar as máquinas capazes de terem uma percepção do mundo assim como os seres humanos, ou seja, assim como somos capazes de identificar e classificar objetos e ações ao visualizar um cenário real, as máquinas também possuem essa capacidade através de modelos de DL, assim como entender uma sequência de palavras e sugerir as próximas de maneira satisfatória, ou até mesmo traduções simultâneas através do processamento de linguagem natural. Existem diversos tipos de modelos de DL como redes neurais de convolução, redes neurais recorrentes, redes neurais recursivas entre outras. Em síntese os modelos de CNN (*Convolutional Neural Network*) são utilizados em aplicações de visão computacional, como identificação de objetos em imagens. Já RNN (*Recurrent Neural Networks*) é utilizado em aplicações como tradução automática e análise de sentimento de uma sequência de palavras. GAN (*Generative Adversarial Network*) é aplicado em tarefas como geração de imagens e vídeos.

As redes neurais artificiais são modeladas a partir da estrutura do cérebro humano. Os nós presentes nas camadas do modelo são equivalentes aos neurônios, cuja função é processar uma informação e gerar uma saída (NWADIUGWU, 2021), assim como um neurônio real produz um impulso elétrico. Embora os conceitos sejam baseados no funcionamento do cérebro humano, não há nenhuma comprovação de que ambos funcionam da mesma maneira, já que pouco se sabe sobre o funcionamento do cérebro humano. Uma das principais diferenças é a capacidade de processar diversos tipos de informação de maneira

simultânea, enquanto o cérebro humano é capaz de se adaptar e sintetizar múltiplos tipos de informação de tal forma a compreender o ambiente que o cerca, as redes artificiais são capazes apenas de processar as informações de maneira sequencial. O princípio de funcionamento de um neurônio no contexto de redes neurais representado na Figura 3 consiste em receber um ou mais valores de entrada $x = (x_1, ..., x_n)$ a depender do tipo de dado sendo utilizado, e um vetor w cujos valores representam os pesos associados a cada valor de entrada. O resultado da soma do produto escalar entre x e w com o acréscimo de um valor b conhecido como *bias* conforme Equação (3) é então introduzido como entrada no neurônio, que por sua vez será calculado em uma função de ativação g gerando a saída y. O *bias*, assim como os pesos associados com as entradas, são valores inicialmente arbitrários, e são otimizados conforme o algoritmo melhora o seu desempenho.

$$n = \sum_{i=1}^{n} w_i x_i + b \tag{3}$$

Figura 3 – Neurônio no contexto de redes neurais.



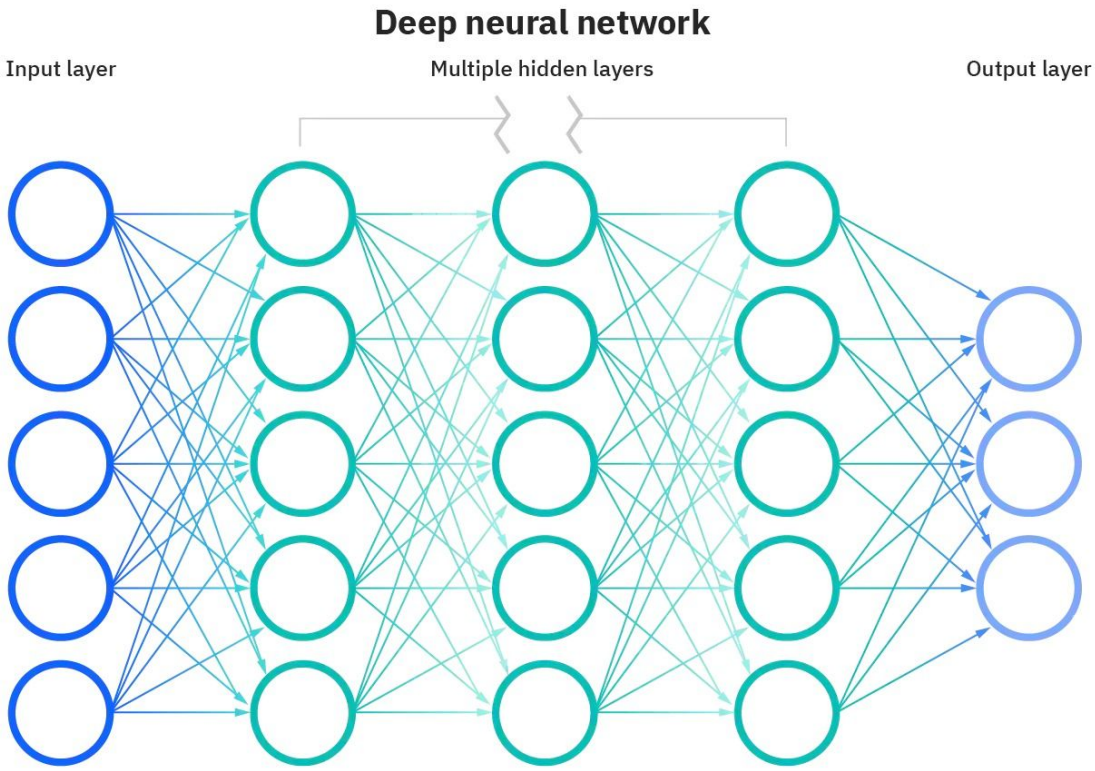
Fonte: O autor.

Um dos fatores responsáveis pela ascensão do aprendizado profundo de máquina é a utilização das GPU (*Graphic Processing Unit*) nos procedimentos padrões no desenvolvimento de um modelo de *deep learning* como treinamento e validação. Através da GPU é possível acelerar este processo através da sua grande capacidade de processamento paralelo dos dados (PANDEY *et al.*, 2022). O fato dos algoritmos de DL serem paralelizáveis permite que as GPUs processem múltiplas operações simultaneamente, dessa forma, é possível treinar modelos com maior velocidade, portanto, modelos que antes eram inviáveis, atualmente podem ser treinados. Como um dos fatores determinantes na performance do modelo é o tamanho da base de dados utilizada para treiná-lo, o uso das GPUs possibilitou o desenvolvimento de modelos mais precisos e aplicáveis em situações

reais, incentivando o estudo e entusiasmo para o desenvolvimento contínuo da área de aprendizado profundo de máquina.

A arquitetura do modelo de DL apesar de ser variável conforme a necessidade do problema, possui camadas características e em sua essência consiste em uma camada de entrada, uma ou mais camadas ocultas e por último uma camada de saída conforme representado na Figura 4.

Figura 4 – Camadas de um modelo de DL.



Fonte: (IBM, 2023)

2.5.1 Função de ativação

Funções de ativação são utilizadas em redes neurais para transformar um sinal de entrada em um sinal de saída, que por sua vez é fornecido como entrada para a camada seguinte do modelo. O cálculo da saída do neurônio é feito através da soma dos produtos de entradas e seus respectivos pesos, e por fim, aplica-se a função de ativação para obter a saída (SHARMA, Siddharth; SHARMA, Simone, 2020).

As funções de ativação desempenham um papel fundamental ao capacitar a rede neural a aprender padrões não lineares nos dados de entrada. No contexto de DL, as não

linearidades são amplamente prevalentes, tornando a presença de uma função de ativação essencial. Existem diversas funções como:

- Sigmoid
- ReLU (*Rectified Linear Unit*)
- Softmax

2.5.1.1 SIGMOID

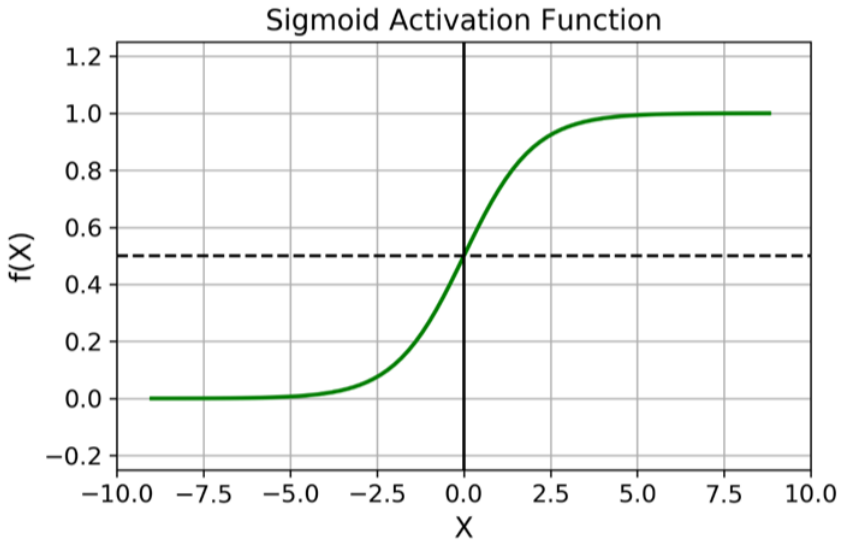
A função de ativação *sigmoid* retorna valores no intervalo de 0 e 1 e é utilizada para dados de treino que também variam entre 0 e 1 (P.SIBI, 2013)

$$0 \leqslant y \leqslant 1 \tag{4}$$

A sua função é representada na Equação (5) e seu gráfico na Figura 5

$$f(x) = \frac{1}{1+e^{-x}} \tag{5}$$

Figura 5 – Gráfico da função *sigmoid*.



Fonte: (SHANU, 2023)

2.5.1.2 ReLU

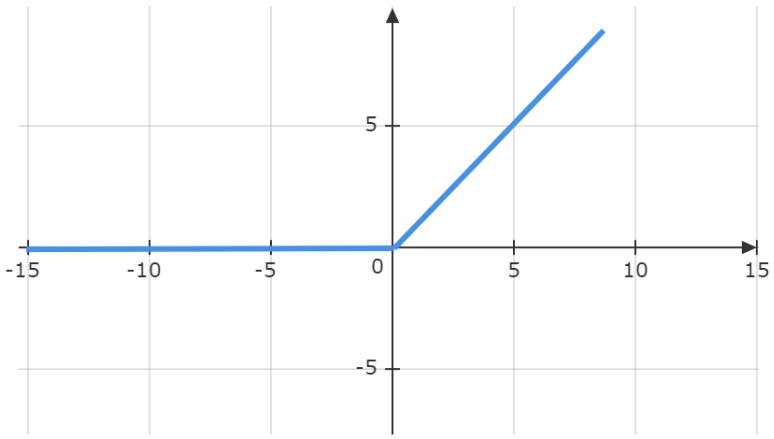
ReLU do inglês *rectified linear unit* é uma função de ativação não linear, e é uma das mais utilizadas (P.SIBI, 2013) devido a sua vantagem no custo computacional e na sua performance em questões de precisão. A eficácia desta ativação consiste no fato de que os nós nas camadas da rede neural não são todos calculados de maneira simultânea, e

isso se deve ao fato de que para valores menores que zero, a saída da função de ativação será zero, e, portanto, não implicará em novos cálculos.

$$\begin{aligned} f(x) &= x,\ x \geqslant 0 \\ f(x) &= 0,\ x < 0 \end{aligned} \tag{6}$$

Um dos fatores responsáveis por tornar a função de ativação ReLU a mais utilizada em redes neurais (IDE; KURITA, 2017) é o fato dela evitar o problema de dissipação do gradiente, já que o seu gradiente é constante para valores positivos. O gráfico da função está representado na Figura 6 e o seu gradiente na Figura 7.

Figura 6 – Gráfico da função *ReLU*.



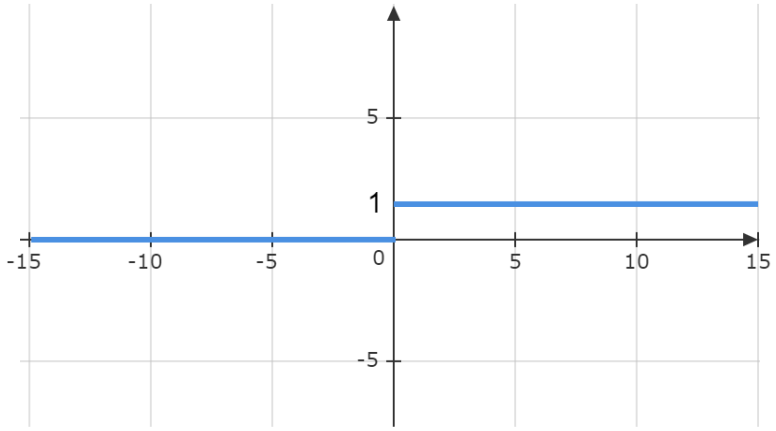
Fonte: O autor

2.5.1.3 Softmax

A função de ativação *softmax* é comumente utilizada para casos em que há mais de uma classificação possível, na qual ela retorna valores entre zero e um para cada entrada, que por sua vez representa a probabilidade daquela entrada representar determinado valor na saída. Se o modelo for composto por x possíveis classificações, então a camada de saída haverá x nós, um para cada possível resultado, e, portanto, por representarem a distribuição probabilística, a soma das respectivas saídas totaliza um, sendo que a saída com o maior número representa a classificação atribuída pelo modelo para aquela entrada específica.

$$f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{7}$$

Figura 7 – Gradiente da função *ReLU*.



Fonte: O autor

2.5.2 Redes Neurais Convolucionais

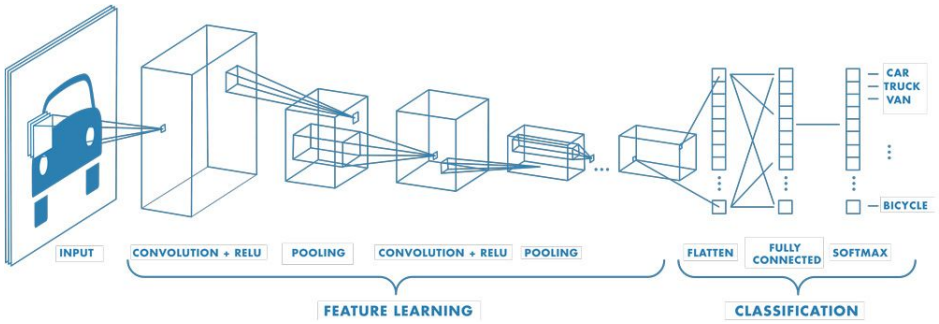
Os modelos mais utilizados na área de DL, são os modelos de redes neurais de convolução (TONG MAO ZHONGJIE SHI, 2021). O processo de extração das informações dos dados, também conhecido como *feature engineering* se torna demasiadamente moroso quando se trata de uma quantidade significativa de dados, o que eleva o nível de complexidade na execução do projeto, muitas vezes o tornando inviável. Devido a isso, os modelos de DL acabam por se destacar em diversas aplicações se comparados com modelos de *machine learning* ou inteligência artificial.

O termo convolucional indica que a rede neural aplica uma operação matemática conhecida por convolução. Redes convolucionais são simplesmente redes neurais que utilizam convolução ao invés de multiplicação matricial em pelo menos uma de suas camadas (GOODFELLOW; BENGIO; COURVILLE, 2016). A estrutura básica de uma rede neural de convolução pode ser visualizada na Figura 8.

2.5.3 Convolução

Convolução no contexto de inteligência artificial é a operação responsável, com suporte das outras, em aprender as características das imagens. A camada de convolução aprende características de baixo nível como segmentos de reta, curvas, diagonais nas camadas iniciais, e posteriormente as utilizam para reconhecer características de nível maior nas camadas seguintes como, por exemplo, um conjunto de retas formando um quadrado com uma cruz no meio representando uma janela e outros padrões que através do aprendizado supervisionado o modelo possa aprender.

Figura 8 – Camadas de uma CNN.



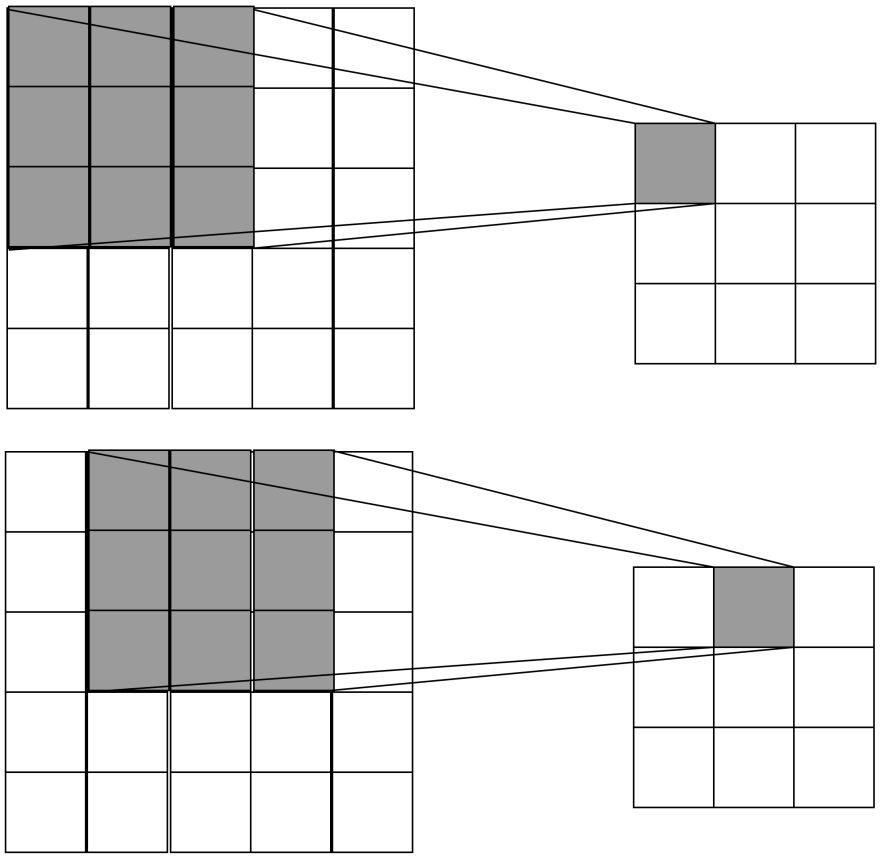
Fonte: (MATLAB, 2023a)

O princípio de funcionamento desta operação é através de uma matriz chamada filtro ou *kernel*, sendo que, no mapa de características gerado após a convolução entre a imagem original e o filtro, é possível saber em que região da imagem se encontra o padrão do filtro aplicado na operação. Existem diversas técnicas que podem ser aplicadas através dos filtros como detecção de bordas, remoção de ruídos entre outros.

Um modelo de DL aplica múltiplos filtros nas imagens de entrada, o que o torna capaz de detectar múltiplas características simultaneamente (AURÉLIEN GÉRON, 2017) e essa operação múltipla retorna algo chamado de mapa de características, além do fato do posicionamento da característica em questão não afetar na capacidade do modelo de reconhece-lá (FRANÇOIS CHOLLET, 2021), ou seja, se um padrão detectado no canto superior direito aparecer no canto inferior esquerdo no *frame* ou imagem seguinte, o modelo ainda assim será capaz de dizer que aquela imagem possui aquela determinada característica aprendida anteriormente. Essa característica é crucial para o desempenho dos modelos de *deep learning* tendo em vista que no mundo real, esses padrões podem estar se locomovendo e não estar necessariamente no mesmo local que estava no *frame* anterior.

A convolução na prática consiste em aplicar o filtro em diversas regiões da imagem e extrair os valores de cada uma delas para formar o mapa de características, a maneira na qual o filtro é aplicado na imagem pode ser controlada através de alguns parâmetros, como o passo, também conhecido como *stride*, este parâmetro define quantos pixeis o filtro irá mover em relação a região anterior para realizar a próxima convolução, este procedimento pode ser visualizado na Figura 9 com o *stride* equivalente a 1 e o mapa de característica sendo formado na matriz à direita conforme o *kernel* é realocado nas regiões da imagem de entrada. Na Figura 9 é possível visualizar que o mapa de características, ou seja, a matriz resultante após a operação de convolução entre a matriz de entrada e o *kernel*, possui uma dimensionalidade menor se comparada com a matriz de entrada, isso acontece pois ao considerar por exemplo uma matriz de entrada 5x5 e um *kernel* 3x3, só há 9 possíveis posições em que o filtro pode ser aplicado na matriz de entrada, e após o produto escalar,

Figura 9 – Convolução.



Fonte: O autor

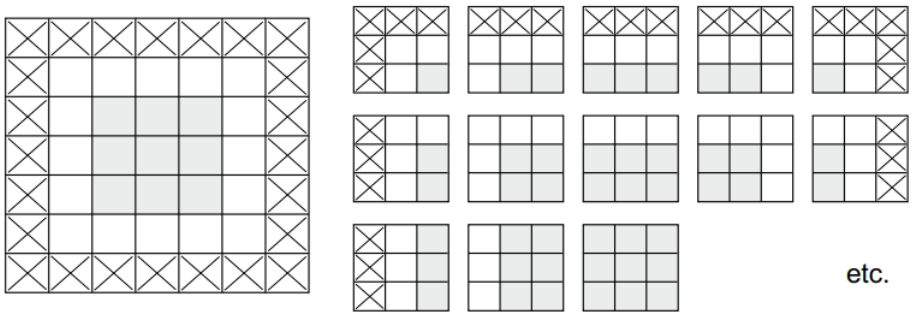
a matriz de entrada com 25 pixeis, é transformada em uma matriz com apenas 9 pixeis. A dimensão do mapa de característica será equivalente a equação (8), sendo m a dimensão da matriz de entrada, k a dimensão do *kernel* e f a dimensão do mapa de característica.

$$f = m - k + 1 \tag{8}$$

Caso a perda de dimensionalidade não seja algo desejável, há a possibilidade de utilizar a técnica de preenchimento (*padding*). O intuito dessa técnica é justamente evitar a perda desta dimensionalidade adicionando pixeis (linhas e colunas) conforme Figura 10, geralmente são adicionados pixeis nulos ao redor da matriz de entrada, de tal forma que o mapa de característica possua a mesma dimensão que a matriz de entrada após a operação de convolução.

Considerando que após aplicar o *padding* na matriz de entrada, a dimensão m passou a ser 7x7 e que a dimensão n do *kernel* é 3x3, ao substituir estes valores na equação (8), f será igual a 5x5, mesma dimensão da matriz de entrada antes da adição do *padding*, evitando assim a perda de dimensionalidade.

Figura 10 – Padding de uma matriz de entrada 5x5.



Fonte: (FRANÇOIS CHOLLET, 2021)

O cálculo dos valores de cada pixel do mapa de características é feito através da equação (9).

$$f_{ij} = \sum_{i=1}^{k} m_{ij}k_{ij} \; + \; b \tag{9}$$

Onde:

- m: elemento da matriz
- k: elemento do kernel
- f: elemento do mapa de características

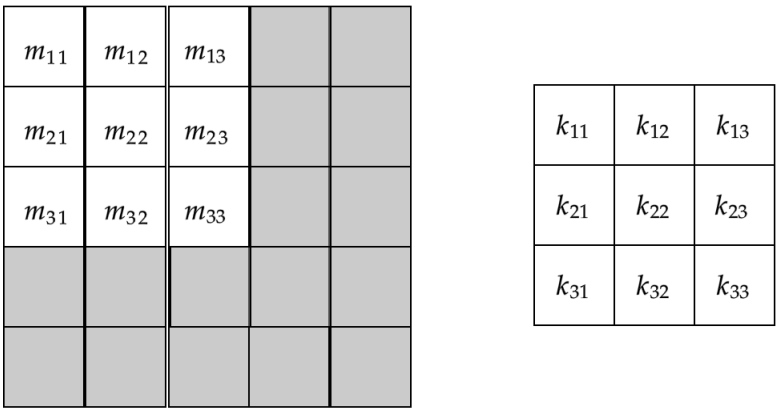
Considerando a matriz de entrada e o *kernel* da Figura 11, o valor de f_{11} seria equivalente a Equação (10) sendo b conhecido como *bias*.

$$\begin{aligned} f_{11} = (&m_{11}k_{11} + m_{12}k_{12} + m_{13}k_{13} \\ &+m_{12}k_{21} + m_{22}k_{22} + m_{23}k_{23} \\ &+m_{31}k_{31} + m_{32}k_{32} + m_{33}k_{33}) + b \end{aligned} \tag{10}$$

2.5.4 Gradiente Descendente

O gradiente descendente é um dos algoritmos mais utilizados para otimização principalmente no contexto de redes neurais (RUDER, 2017). Este algoritmo é uma das principais ferramentas para treinamento do modelo de *deep learning*, através dele é possível saber se os pesos de cada nó da rede neural devem ser acrescidos ou diminuídos de tal modo que o erro do modelo diminua. Entende-se por peso o valor associado aos nós da rede neural, este valor por sua vez é responsável por transformar o valor de entrada com origem na camada anterior em um novo valor através da função de ativação. Este valor calculado passa para a camada seguinte até que chegue na última, conhecida como camada de saída, e espera-se que a saída obtida seja o mais próximo possível do valor real.

Figura 11 – Matriz de entrada e *kernel*.



Fonte: O autor

Em suma, o gradiente descendente tem por objetivo encontrar a melhor combinação de pesos de tal modo que o erro seja o menor possível.

$$erro = Y_{pred} - Y_{real} \tag{11}$$

Esta reestruturação é feita através do cálculo diferencial da função em questão. Para tanto, é necessário utilizar uma função de perda, como por exemplo o erro quadrático médio que pode ser definido através da equação:

$$J(p) \ = \ \frac{1}{m} \sum_{i \ = \ 1}^{m} (Ypred \ - \ Yreal)^2 \tag{12}$$

Sendo m equivalente ao número de exemplos e p a matriz de pesos do modelo. Posteriormente calcula-se o gradiente desta função de perda em relação aos pesos:

$$\nabla J(p) = \frac{\partial J}{\partial p} \tag{13}$$

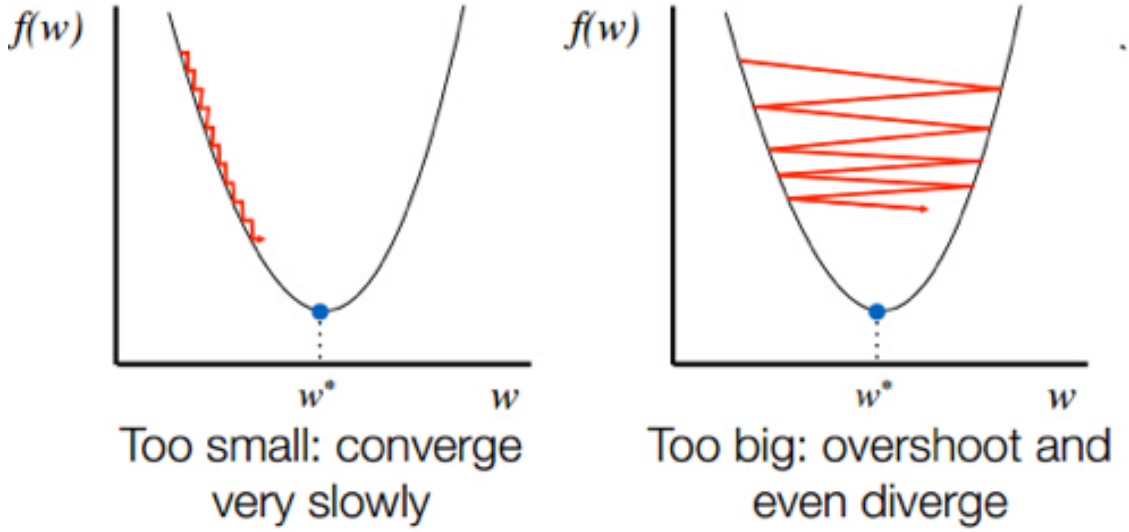
O cálculo deste gradiente é equivalente a inclinação da curva da função de perda para um determinado ponto, de tal modo que ao saber a direção da função, sabe-se se o erro irá aumentar ou diminuir. No contexto de DL, este cálculo é realizado computacionalmente de forma iterativa até que se encontre um valor que satisfaça o objetivo da aplicação, sendo o valor mínimo global o ideal a ser calculado. Portanto, pode-se otimizar os pesos do modelo na direção oposta ao gradiente, através de uma taxa de aprendizado adequada η:

$$p \ = \ p \ - \ \eta \nabla J(p) \tag{14}$$

A taxa de aprendizagem permite controlar a variação de cada reestruturação de peso do modelo, e influencia diretamente na eficácia do mesmo, tanto em termos de

resultados quanto em custos computacionais. Ao optar por uma taxa de aprendizado mais conservadora, garante-se o cálculo de mais pontos, logo, menor a probabilidade de ultrapassar o ponto mínimo da função e maior o custo computacional devido ao acréscimo no número de cálculos realizados, ao utilizar um valor maior, o custo computacional diminui e a probabilidade de ultrapassar o mínimo aumenta caracterizando um *overshooting* conforme representado na Figura 12.

Figura 12 – Efeito da taxa de aprendizagem em ambos os casos.



Fonte: (RUFAI, 2023)

2.5.5 Retro propagação

A retro propagação, do inglês *backpropagation* é o processo que de fato atualiza os pesos dos nós que constituem a rede neural, através dos cálculos explicitados nas seções anteriores. O termo retro propagação ou propagação reversa se deve ao fato de que este cálculo é inicializado na camada de saída e segue rumo a camada de entrada, ou seja, de trás para frente. O algoritmo de propagação reversa segue um fluxo padrão no qual o erro (1) também conhecido como valor de custo, é propagado de volta na rede neural e os pesos das camadas subsequentes são recalculados. Esta reestruturação dos pesos é feita através do seguinte procedimento:

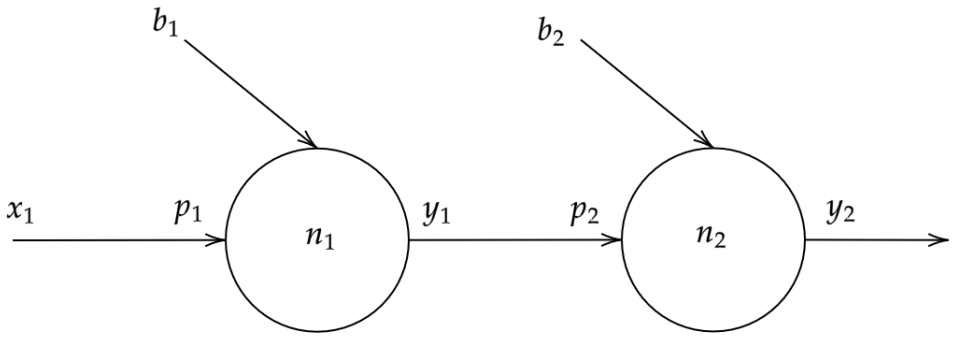
- Calcular a saída apenas com a propagação direta;
- Calcular o erro (1);
- Atualizar os pesos;
- Repetir as etapas anteriores até que seja atingido um valor de custo satisfatório para a aplicação.

Ao considerar uma rede neural com apenas duas camadas e cada uma delas com um neurônio conforme Figura 13 e uma função de perda do tipo:

$$E = \frac{1}{2}(Y - y_2)^2 \tag{15}$$

sendo Y o valor real.

Figura 13 – Rede neural com duas camadas e dois neurônios.



Fonte: O autor.

Podemos escrever o peso recalculado em termos do gradiente da função de perda e da taxa de aprendizagem η.

$$p_i = p_i - \eta\frac{\partial E}{\partial p_i} \tag{16}$$

Sabendo que:

$$n_1 = x_1 p_1 + b_1 \tag{17}$$

e

$$n_2 = y_1 p_2 + b_2 \tag{18}$$

pode-se escrever p_1 e p_2 em termos de y_1 e y_2 e o gradiente em relação a p_2 passa a ser

$$\frac{\partial E}{\partial p_2} = \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial n_2}\frac{\partial n_2}{\partial p_2} \tag{19}$$

Ao substituir a equação (19) em (16) chega-se na expressão final responsável por atualizar o peso p_2 de tal forma a diminuir o erro do modelo

$$p_2 = p_2 - \eta(\frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial n_2}\frac{\partial n_2}{\partial p_2}) \tag{20}$$

e através da regra da cadeia chega-se nas expressões referentes aos pesos das camadas anteriores.

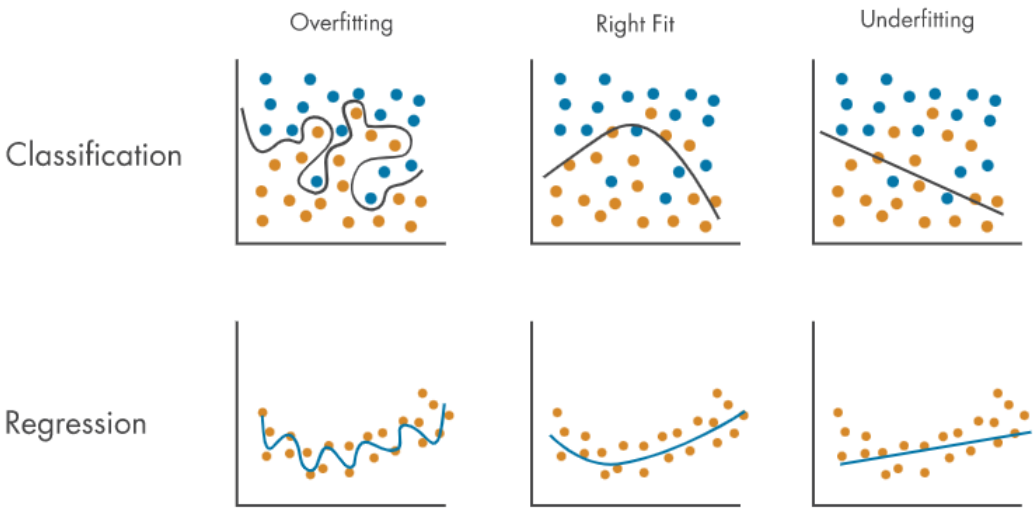
2.5.6 Overfitting e Underfitting

Overfitting é um problema comum em *deep learning*, que impossibilita o modelo de generalizar de maneira satisfatória as características dos dados utilizados no treino do mesmo, de tal forma que, ao aplicar o modelo em dados não utilizados anteriormente, ele não é capaz de atingir a mesma performance se comparado com a performance durante o treinamento. (YING, 2019).

Este fenômeno ocorre quando o modelo aprende de maneira excessiva as características que são únicas e exclusivas do *dataset* de treinamento, fazendo com que, ao utilizar uma imagem nova como entrada para o modelo, o mesmo não seja capaz de fazer as inferências corretas pelo fato da nova entrada não possuir as características das imagens utilizadas no treinamento, impactando diretamente no desempenho final.

Underfitting é o oposto de *overfitting*, significa que o modelo não foi capaz de aprender características suficientes dos dados utilizados, e consequentemente, não sendo capaz de obter um bom desempenho em novos dados de entrada. A curva característica de ambos os casos pode ser visualizada na Figura 14.

Figura 14 – Curva característica de *Overfitting* e *Underfitting*.



Fonte: (MATLAB, 2023b)

2.5.7 Data Augmentation

Data augmentation é uma técnica que visa reduzir o *overfitting* dos modelos, aumentando a quantidade de dados de treino através das informações do próprio *dataset* de treinamento (PEREZ; WANG, 2017). O intuito de utilizar está técnica é para generalizar de maneira mais efetiva as características aprendidas durante o processo de treinamento do modelo, para que ele tenha o mesmo desempenho obtido durante o treinamento nos novos dados para inferência. O processo consiste em alterar as imagens de treino com reajustes como rotação, adição de ruído, alteração na escala RGB e outras técnicas que ao serem utilizadas podem aumentar a quantidade de dados de treinamento e influenciar de maneira positiva a generalização das características do *dataset*, em contra partida, ao transformar de maneira drástica as imagens de treino, pode-se adicionar características falsas no *dataset* fazendo com que o modelo aprenda padrões inexistentes, e, também, prejudicando o desempenho do mesmo.

2.5.8 Flatten layer

Flatten layer é uma camada comumente utilizada em redes neurais convolucionais, cujo a finalidade é transformar um vetor de dimensão nxm em um vetor de dimensão $nm \times 1$ (SURIYA; CHANDRAN; SUMITHRA, 2019) conforme representado em (21). Esta camada é normalmente, um intermédio entre a etapa de identificar e aprender quais são as características das imagens presentes no *dataset*, onde ocorrem as operações de convoluções, com a etapa de uma rede neural clássica.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \rightarrow \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{31} \\ a_{32} \\ a_{33} \end{bmatrix} \tag{21}$$

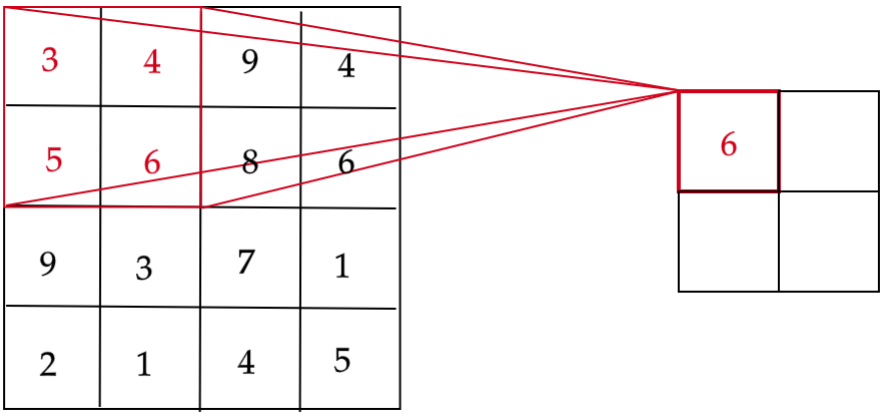
2.5.9 Pooling layer

O objetivo da camada *pooling* é criar amostras menores da imagem original para diminuir o custo computacional do modelo (AURÉLIEN GÉRON, 2017). O processo é semelhante ao da convolução, porém, por não possuir peso, o resultado obtido e a finalidade de uso é diferente.

Existem alguns tipos diferentes de camada *pooling*, porém, duas são mais utilizadas, sendo elas *max pooling layer* e *average pooling layer*. *Max pooling layer* é a mais utilizada

entre as duas, e em suma, a cada operação, é selecionado a característica dominante da região, já que apenas o maior valor é transmitido para a amostra menor da imagem que será gerada durante a camada *pooling*. Existem três parâmetros a serem definidos ao utilizar este tipo de camada, sendo o primeiro deles o *filter size*, o segundo o *stride* e por último o *padding*, porém é comum utilizar o *padding* nulo. O *filter size* é responsável por determinar o tamanho da matriz que será gerada após finalizar a operação e o *stride* a quantidade de colunas que o filtro irá se locomover para realizar a operação seguinte. Na Figura 15 pode-se visualizar um filtro 2x2 e o resultado obtido após a primeira operação da camada. A operação do tipo *average pooling* é semelhante, porém, ao invés de retornar o maior valor, retornaria a média dos quatro elementos dentro da operação do filtro.

Figura 15 – *Max pooling layer*



Fonte: O autor

Neste caso, a camada resulta em uma imagem com 4 pixeis, quatro vezes menos que a imagem da camada anterior, justificando o motivo pelo qual a camada *pooling* retorna uma amostra menor da imagem, além dos benefícios computacionais, esta operação diminui de certa forma o *overfitting* do modelo, pois após ela, o modelo aprende com uma forma mais abstrata da imagem.

2.5.10 Parâmetros de aprendizado

Em uma CNN há diversos parâmetros de aprendizado e alguns são parâmetros que são otimizados pelo próprio algoritmo como os pesos e os *bias* também conhecidos como viés. Os pesos são valores que determinam a contribuição que a conexão com o neurônio possui na rede como um todo, mais especificamente, o peso que a conexão do neurônio anterior contribui para o neurônio da camada seguinte, sendo que um valor maior representa uma maior contribuição, e um valor menor representa uma contribuição menor, já o *bias* é um valor associada a cada neurônio da rede, de tal forma a controlar a ativação deste neurônio.

Em via do citado, há um número de parâmetros de aprendizado, ou seja, parâmetros que podem ser atualizados com o objetivo de diminuir a função de custo. O número destes parâmetros em uma camada de convolução pode ser calculado conforme demonstrado na Equação (22).

$$p = m(kd) + b \tag{22}$$

Onde:

- p: número de parâmetros de aprendizado
- m: número de filtros
- k: dimensão do filtro
- d: número de canais
- b : número de *bias*

2.6 MEDIDAS DE DESEMPENHO

Medidas de desempenho no contexto de ML são métricas utilizados para avaliar o desempenho dos modelos de maneira quantitativa, através destes métodos é possível concluir se o modelo está atuando ou não dentro do esperado, e concluir se há a necessidade de alterar a estrutura do modelo, obter mais dados para treinamento ou então buscar outros parâmetros para otimizar o desempenho do mesmo.

2.6.1 Matriz de Confusão

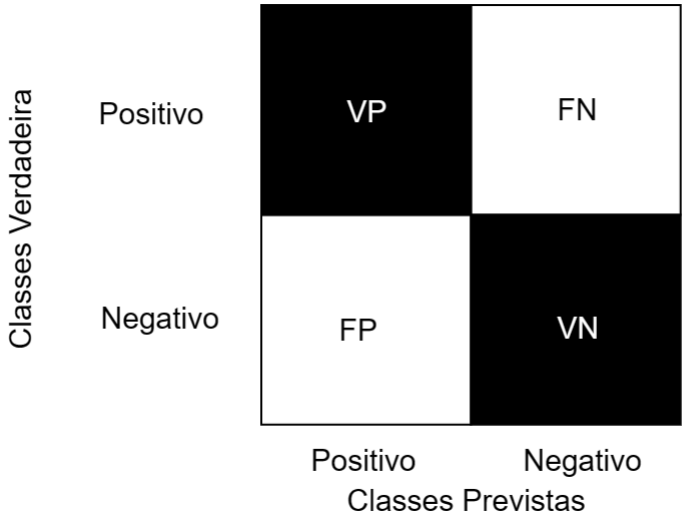
Matriz de confusão é uma matriz de dimensão $N \times N$, sendo N equivalente ao número de classes. No contexto de ML está métrica é utilizada para representar a quantidade de classificações corretas e incorretas feitas pelo modelo, a diagonal principal da matriz representa os acertos e a diagonal secundária os erros. Um exemplo desta matriz pode ser visualizada na Figura 16, sendo VP referente aos verdadeiros positivos, VN verdadeiros negativos e de forma análoga na diagonal secundária a representação dos falsos positivos e falsos negativos.

2.6.2 Acurácia

A acurácia é uma métrica que visa medir a quantidade de classificações verdadeiras realizadas pelo modelo, ou seja, a quantidade de classificações que se enquadram na diagonal principal da matriz de confusão. O valor de acurácia é obtido através da divisão da soma das classificações verdadeiras pela quantidade total de classificações feitas pelo modelo conforme Equação (23).

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN} \tag{23}$$

Figura 16 – Matriz de Confusão.



Fonte: O autor.

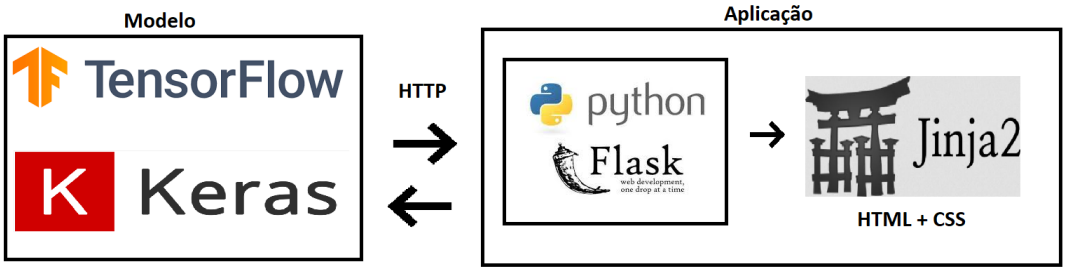
2.7 FLASK

Flask é um *framework* cuja proposta é ser um WSGI (*web server gateway interface*) leve (FLASK. . . , 2023). WSGI é uma especificação utilizada para descrever a comunicação entre servidores da Web com os aplicativos web.

O fato dele ser um micro *framework* significa que a inicialização da aplicação através dele é feita da maneira otimizada, ou seja, somente os pacotes responsáveis por possibilitar o uso da ferramenta na web sem nenhuma implementação extra, porém, isso não significa que ele não é capaz de estruturar aplicativos complexos. A frase utilizada por desenvolvedores para definir o *framework* é "uma gota de cada vez", o intuito desta frase é demonstrar que os pacotes são adicionados conforme a sua necessidade, isso otimiza a aplicação já que não há a implementação de outros pacotes cujo o desenvolvedor não julgou necessário.

O mecanismo de formatação utilizado pelo *Flask* é o *Jinja2*, através deste mecanismo é possível utilizar a linguagem de programação HTML em união com o *Python* para formatar a página web conforme a necessidade da aplicação, e através do objeto *Response* é possível trabalhar com respostas HTTP personalizadas para a aplicação, sendo possível retornar diversos objetos como texto, arquivos *JSON*, *frames* entre outros que podem ser inseridos como argumento no próprio HTML da aplicação, essa estrutura pode ser visualizada na Figura 17.

Figura 17 – Arquitetura com Flask.



Fonte: O autor

3 DESENVOLVIMENTO

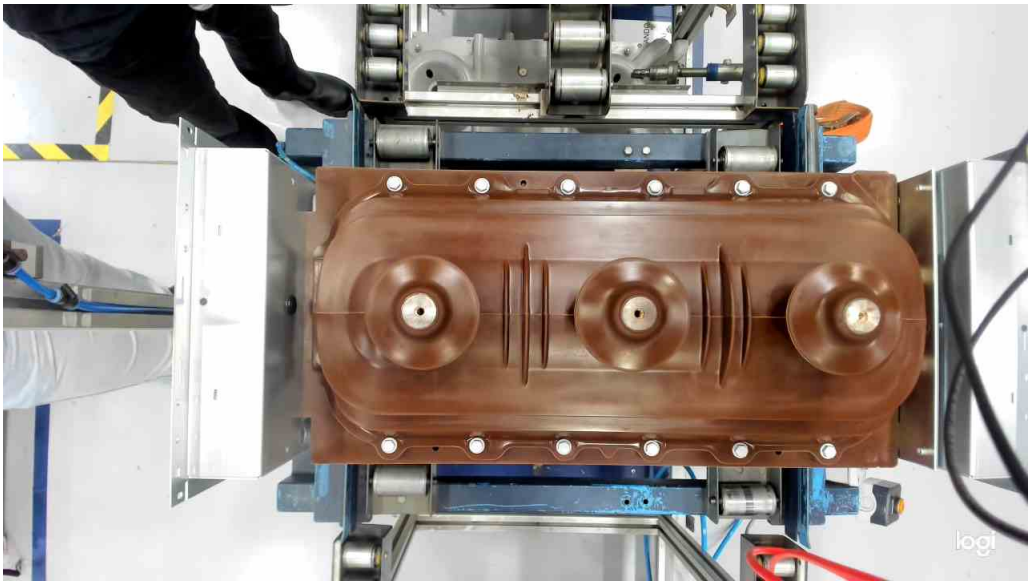
Este capítulo refere-se ao desenvolvimento da solução proposta no trabalho. Os principais pontos são abordados como aquisição dos dados, estrutura e treinamento dos modelos de *deep learning*, equipamento utilizado e estrutura do local de teste na linha de produção.

3.1 AQUISIÇÃO DOS DADOS

Os dados referentes a este projeto em questão são imagens da seccionadora, como explicitado previamente, o problema a ser resolvido pode ser definido como um problema de classificação binária, ou seja, somente duas situações podem ocorrer durante o processo, sendo a seccionadora na posição correta ou então na posição incorreta, portanto, optou-se por adquirir imagens representativas dos dois estados que serão referenciados como **PosCorreta** e **PosErrada** para facilitar a compreensão, já que ambos os estados foram definidos desta maneira durante o desenvolvimento do modelo.

Vale ressaltar que, por se tratar de um ambiente corporativo houve complicações durante as aquisições das imagens por questões logísticas e internas. Ao todo, foram capturadas 280 imagens representativas de ambos os estados, sendo 140 referentes ao posicionamento correto da seccionadora e 140 ao posicionamento incorreto conforme representado nas Figuras 18 e 19 respectivamente.

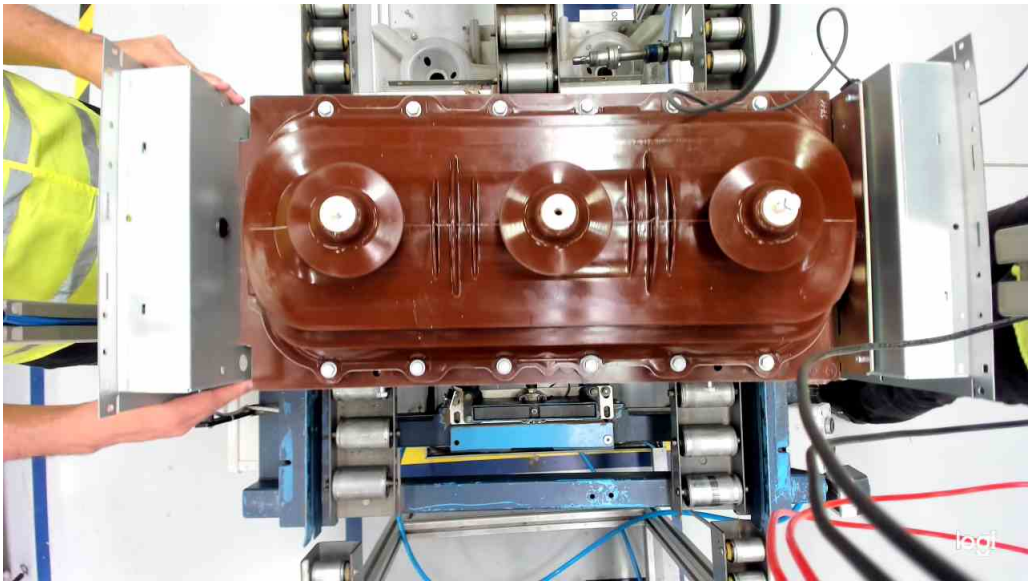
Figura 18 – Posicionamento correto da seccionadora.



Fonte: O autor.

Pode-se observar que em ambas as imagens há a presença de operadores e outros componentes como fiação e a própria esteira com roletes. Estes ruídos foram capturados

Figura 19 – Posicionamento incorreto da seccionadora.



Fonte: O autor.

de maneira intencional, isso se deve ao fato de que a solução proposta por este trabalho foi desenvolvida para atuar em um cenário real de chão de fábrica, e, tendo em vista que a locomoção desta peça é feita de forma manual pelo operador responsável por realizar o teste, o modelo deve ser capaz de classificar o posicionamento apesar das interferências externas como a mão do operador e a fiação da estrutura de teste.

A tonalidade da seccionadora pode variar conforme representado nas Figuras 18 e 19. A Figura 18 possui a tonalidade 986248 no padrão hexadecimal, e a Figura 19 a tonalidade 8E3B1D também no padrão hexadecimal conforme Figuras 20 e 21. Durante o teste dos primeiros modelos, observou-se que a tonalidade da peça interferia diretamente na performance do mesmo, tendo em vista que os primeiros modelos haviam sido treinados somente com peças da tonalidade referente a Figura 21, isso se deve ao fato de que em sua grande maioria, as seccionadoras que são utilizadas nos painéis elétricos em questão são desta tonalidade.

A fim de contornar este problema, mais dados foram obtidos com a tonalidade da Figura 20 com o objetivo de garantir a eficácia do modelo independente da tonalidade da seccionadora. Os exemplos podem ser vistos nas Figuras 18 e 22.

Os detalhes dos dados obtidos podem ser visualizados na Tabela 1.

Tabela 1 – Dados obtidos.

Classificação	**Tonalidade 986248**	**Tonalidade 86248**
PosCorreta	110 fotos	30 fotos
PosErrada	110 fotos	30 fotos

Figura 20 – Cor 986248 padrão hexadecimal.

Fonte: O autor.

Figura 21 – Cor 8E3B1D padrão hexadecimal.

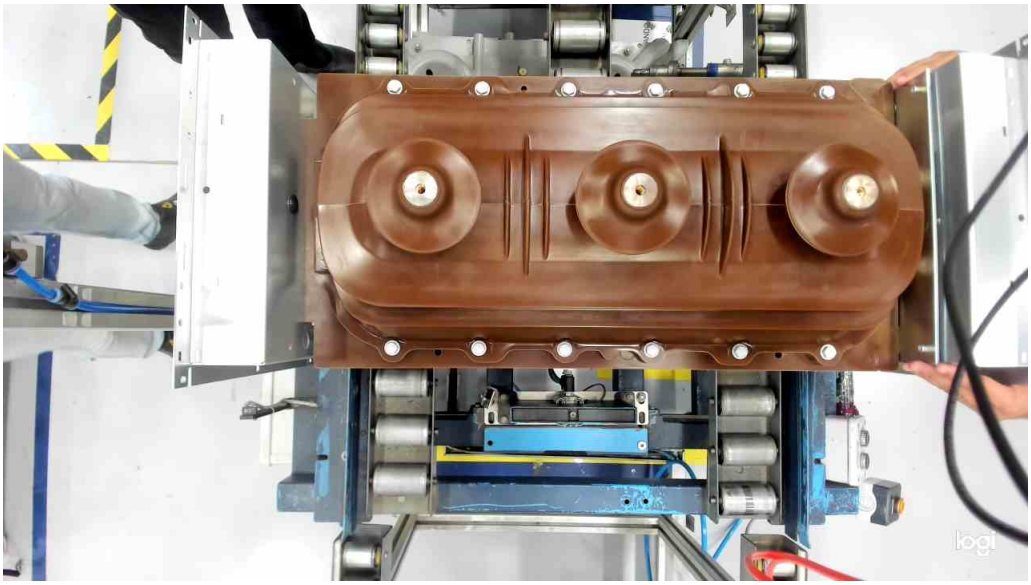
Fonte: O autor.

3.1.1 CATEGORIZAÇÃO DOS DADOS

A categorização das imagens obtidas é um dos fatores cruciais e determinantes para o desempenho do modelo de aprendizado profundo ou qualquer outro modelo de aprendizado supervisionado de máquina. Este procedimento é conhecido como categorização e consiste em atribuir um valor para cada imagem, no caso desta solução, por se tratar de uma classificação binária, somente duas classificações foram utilizadas. Este procedimento foi realizado de maneira minuciosa com suas devidas revisões, pois ao classificar de forma errônea as imagens, há a possibilidade do modelo atribuir os padrões aprendidos durante o treinamento para a classificação errada, de tal modo a prejudicar as inferências que serão feitas pelo próprio futuramente, o que não é desejável.

Para este trabalho, optou-se por classificar as imagens através do nome dos subdiretórios, e para facilitar a compreensão e a validação dos resultados obtidos, os arquivos *.jpg* foram nomeados indicando a classificação do posicionamento da peça conforme explicitado nas Figuras 23 e 24. Ao utilizar o nome dos subdiretórios, é possível fazer o uso da função $flow\_from\_directory$ do *TensorFlow*. Em suma essa função recebe o caminho de um diretório como um dos argumentos, e exige que cada subdiretório pertencente a ele

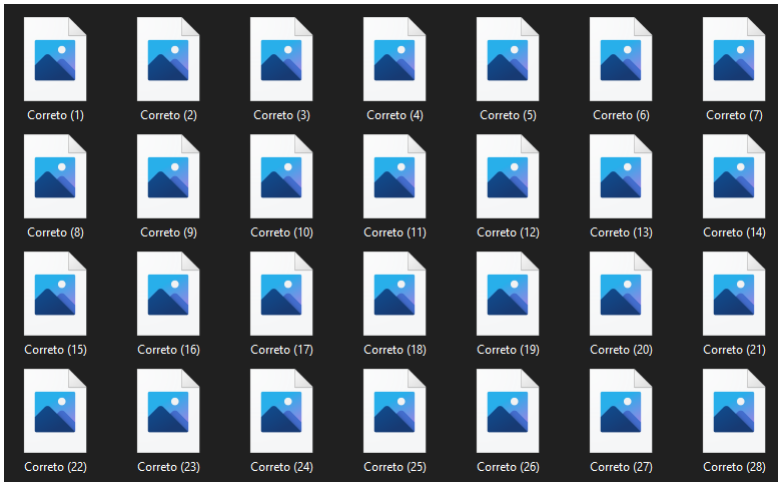
Figura 22 – Posicionamento incorreto da seccionadora com tonalidade 986248.



Fonte: O autor.

seja referente a uma das classes do modelo, de tal modo que as imagens são associadas a classificação sugerida pelo nome do subdiretório em questão, logo, as imagens pertencentes ao subdiretório **PosCorreta** são pertencentes as classificações de posições corretas, e as pertencentes ao subdiretório **PosErrada** as posições incorretas.

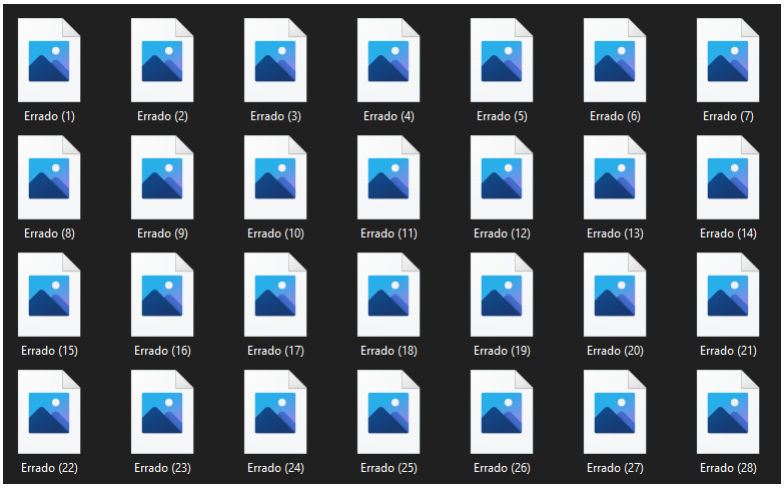
Figura 23 – Classificação das imagens - Posição Correta



Fonte: O autor.

A estrutura dos diretórios pode ser visualizada na Figura 25 sendo o diretório **Treino** referente aos dados que foram utilizados para treinar o modelo, o diretório **Validacao** referente aos dados utilizados para medir o desempenho do modelo durante o

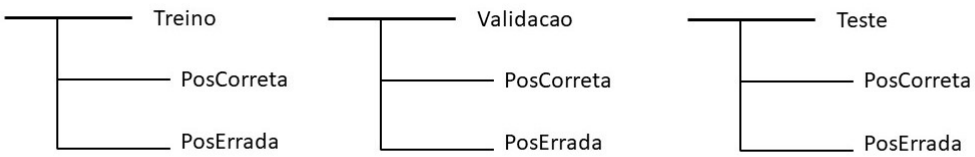
Figura 24 – Classificação das imagens - Posição Incorreta



Fonte: O autor.

treinamento e por último o diretório **Teste** referente aos dados utilizados para medir a eficácia do modelo após o seu treinamento.

Figura 25 – Diretórios para treinamento e validação do modelo.



Fonte: O autor.

3.2 CRIAÇÃO DOS MODELOS

Foram desenvolvidas quatro estruturas diferentes de modelos de CNN a fim de medir o desempenho individual de cada e comparar os resultados obtidos entre eles, e então, optar pelo modelo com melhor desempenho para ser utilizado na aplicação na linha de produção da fábrica.

Pelo fato da base de dados ser pequena, era esperado que pudesse ocorrer o fenômeno de *overfitting* durante o treinamento dos modelos, em via disso, dois dos quatro modelos

criados foram desenvolvidos com técnicas que visam evitar este fenômeno, sendo elas *data augmentation* e *dropout* conforme teoria descrita no Capítulo 2, e os outros dois apenas com as camadas clássicas de CNN como *conv2D*, *Pooling*, *Flatten* e *Dense*. Outro fator considerado durante o desenvolvimento dos modelos foi a quantidade de camadas ocultas, também conhecidas como *hidden layers*.

A descrição dos modelos e as técnicas aplicadas podem ser vistas na Tabela 2. As camadas de entrada, saída e *data augmentation* não foram consideradas na contagem das camadas. Os modelos *modelo01* e *modelo02* possuem arquiteturas semelhantes em termos de quantidade de *hidden layers*, filtros e parâmetros utilizados, porém o *modelo01* foi implementado com camadas de *data augmentation* e *dropout* visando evitar o fenômeno de *overfitting* e o *modelo02* somente com as camadas clássicas de CNN. O mesmo é válido para os modelos três e quatro, sendo que o *modelo04* utiliza *data augmentation* sem *dropout* e o *modelo03* somente as camadas clássicas de CNN assim como o *modelo02*.

Nome	Quantidade de camadas	Técnica aplicada
modelo01	12	*Data augmentation* e *Dropout*
modelo02	11	Clássica de CNN
modelo03	9	Clássica de CNN
modelo04	9	*Data augmentation*

Tabela 2 – Modelo desenvolvidos.

3.3 ESTRUTURA DOS MODELOS

A Tabela 3 e Figura 26 são referentes a estrutura do modelo *modelo01*, assim como Tabela 4 e Figura 27 referente ao modelo *modelo02*, Tabela 5 e Figura 28 ao modelo *modelo03* e por último a Tabela 6 e Figura 29 referente ao modelo *modelo04*.

Na última camada de todos os modelos, camada *dense*, foi utilizada a função de ativação *sigmoid* para obter valores entre 0 e 1 conforme especificado na Seção 2.5.1.1 já que se trata de um modelo de aprendizado supervisionado de classificação binária, precedida por uma camada do tipo *flatten*.

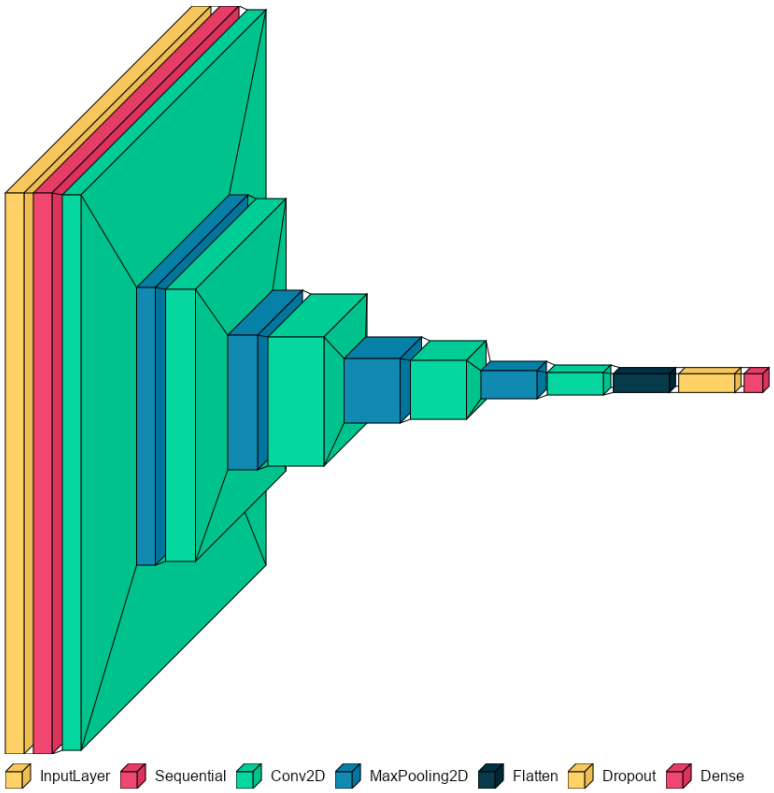
Em termos de parâmetros de aprendizado, os modelos *modelo01* e *modelo02* possuem um total de 994.881 parâmetros, valor que pode ser obtido ao somar os valores da coluna *Parâmetros de aprendizado*, e os modelos *modelo03* e *modelo04* possuem 501.313 parâmetros. A quantidade de parâmetros de aprendizado pode ser calculada através da Equação (22) especificada na Seção 2.5.10, a título de exemplo, a quantidade de 896 parâmetros da primeira camada de convolução na Tabela 5 pode ser obtida conforme explicitado na Equação (24) sendo $m = 32$, $k = 9$, e $d = 3$.

$$p = 32x(9x3) + 32 = 896 \tag{24}$$

Nome da camada	Dimensão da saída	Parâmetros de aprendizado
InputLayer	(200, 200, 3)	0
Sequential	(200, 200, 3)	0
Conv2D	(198, 198, 32)	896
MaxPooling2D	(99, 99, 32)	0
Conv2D	(97, 97, 64)	18496
MaxPooling2D	(48, 48, 64)	0
Conv2D	(46, 46, 128)	73856
MaxPooling2D	(23, 23, 128)	0
Conv2D	(21, 21, 256)	295168
MaxPooling2D	(10, 10, 256)	0
Conv2D	(8, 8, 256)	590080
Flatten	(16384)	0
Dropout	(16384)	0
Dense	(1)	16385

Tabela 3 – Estrutura modelo01.

Figura 26 – Arquitetura visual do modelo *modelo01*

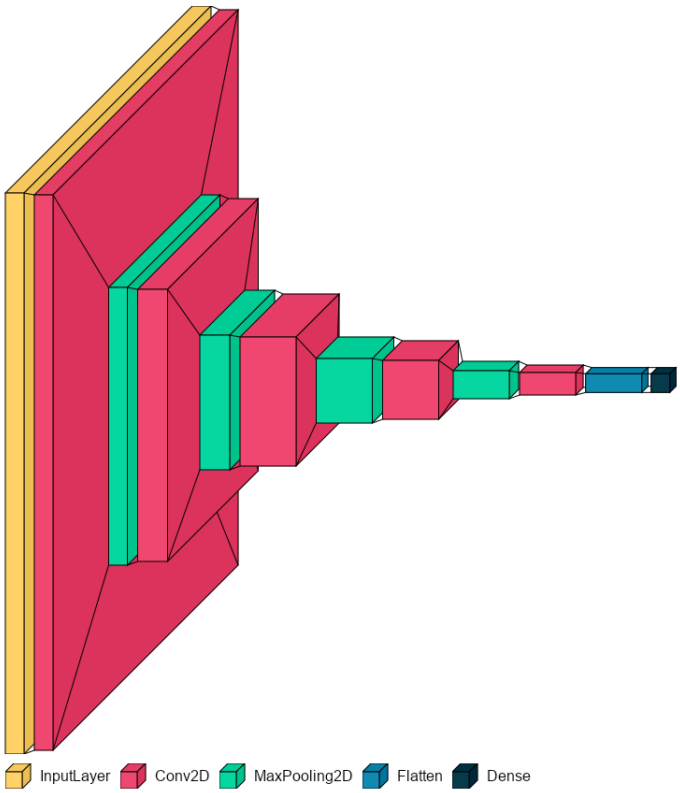


Fonte: O autor.

Nome da camada	Dimensão da saída	Parâmetros de aprendizado
InputLayer	(200, 200, 3)	0
Conv2D	(198, 198, 32)	896
MaxPooling2D	(99, 99, 32)	0
Conv2D	(97, 97, 64)	18496
MaxPooling2D	(48, 48, 64)	0
Conv2D	(46, 46, 128)	73856
MaxPooling2D	(23, 23, 128)	0
Conv2D	(21, 21, 256)	295168
MaxPooling2D	(10, 10, 256)	0
Conv2D	(8, 8, 256)	590080
Flatten	(16384)	0
Dense	(1)	16385

Tabela 4 – Estrutura modelo02.

Figura 27 – Arquitetura visual do modelo *modelo02*

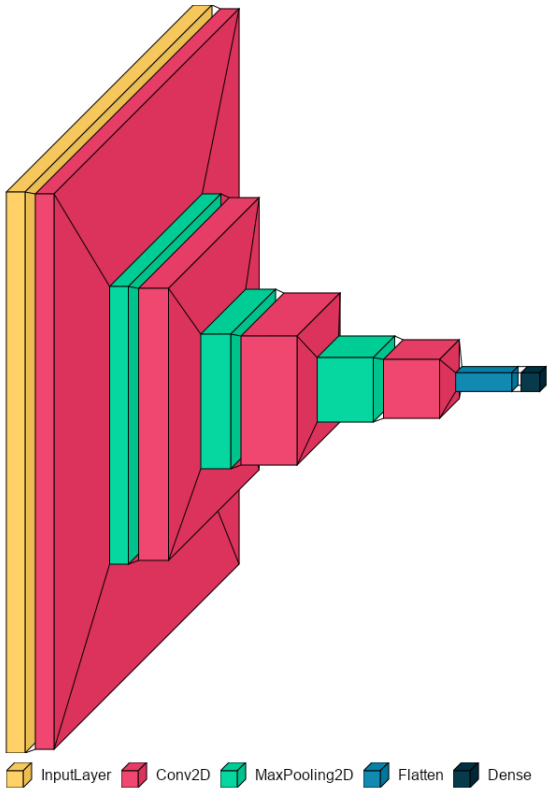


Fonte: O autor.

Nome da camada	Dimensão da saída	Parâmetros de aprendizado
InputLayer	(200, 200, 3)	0
Conv2D	(198, 198, 32)	896
MaxPooling2D	(99, 99, 32)	0
Conv2D	(97, 97, 64)	18496
MaxPooling2D	(48, 48, 64)	0
Conv2D	(46, 46, 128)	73856
MaxPooling2D	(23, 23, 128)	0
Conv2D	(21, 21, 256)	295168
Flatten	(112896)	0
Dense	(1)	112897

Tabela 5 – Estrutura modelo03.

Figura 28 – Arquitetura visual do modelo *modelo03*

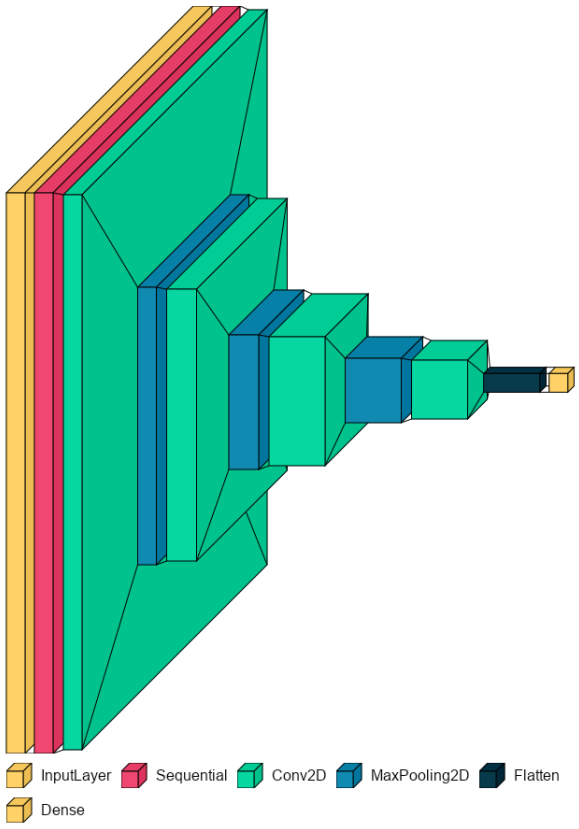


Fonte: O autor.

Nome da camada	Dimensão da saída	Parâmetros de aprendizado
InputLayer	(200, 200, 3)	0
Sequential	(200, 200, 3)	0
Conv2D	(198, 198, 32)	896
MaxPooling2D	(99, 99, 32)	0
Conv2D	(97, 97, 64)	18496
MaxPooling2D	(48, 48, 64)	0
Conv2D	(46, 46, 128)	73856
MaxPooling2D	(23, 23, 128)	0
Conv2D	(21, 21, 256)	295168
Flatten	(112896)	0
Dense	(1)	112897

Tabela 6 – Estrutura modelo04.

Figura 29 – Arquitetura visual do modelo *modelo04*



Fonte: O autor.

3.4 TREINO DOS MODELOS

Como o intuito da utilização de quatro modelos com arquiteturas diferentes era medir o desempenho em comparação com os próprios, os parâmetros utilizados durante o treinamento foram os mesmos para todos. Estes parâmetros podem ser visualizados na Tabela 7.

Parâmetro	Valor
Loss	*binary_crossentropy*
optimizer	*adam*
metrics	*accuracy*

Tabela 7 – Parâmetros de treinamento.

Os modelos que utilizaram a técnica de *data augmentation* (*modelo04* e *modelo01*) utilizaram os parâmetros da Tabela 8. E o modelo *modelo01* utilizou o parâmetro da Tabela 9 na camada de *dropout*.

Parâmetro	Valor
RandomFlip	horizontal
RandomRotation	0.1
RandomZoom	0.2

Tabela 8 – Parâmetros *data augmentation*.

Parâmetro	Valor
rate	0.5

Tabela 9 – Parâmetros camada *dropout*.

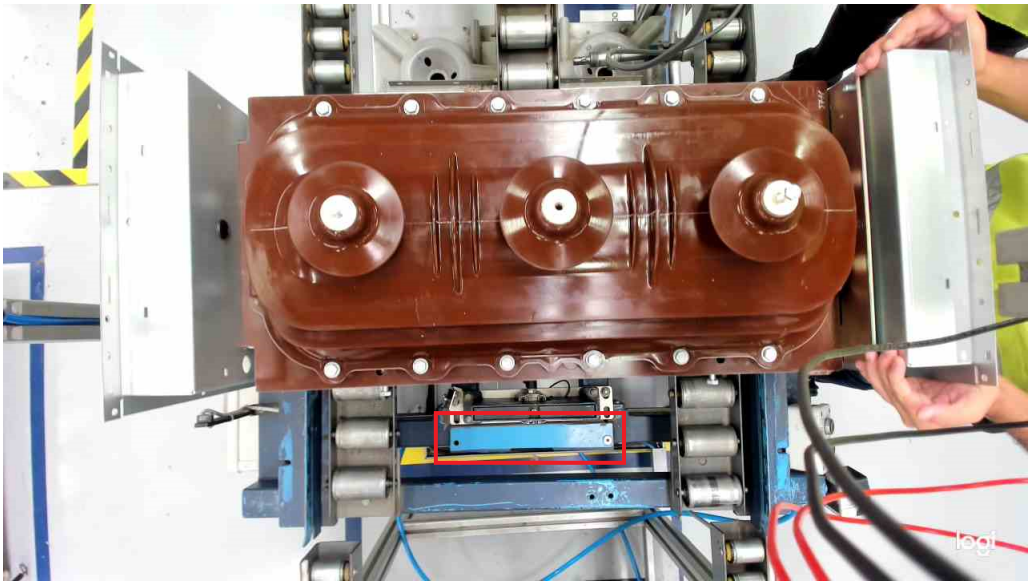
A proporção utilizada dos dados para treinamento dos modelos foi de 75%, totalizando 210 imagens, 40 imagens para validação e 31 imagens para teste dos modelos. Sendo que a mesma proporção foi adotada para selecionar as imagens de ambas as tonalidades, além de ter sido utilizado 30 *epochs* durante os treinamentos.

3.5 ESTRUTURA

Conforme explicitado no Seção 1.2 o ambiente de teste consiste em uma esteira do tipo rolete na qual as seccionadoras são apoiadas, e em um lugar específico desta esteira é o local de teste. Logo abaixo deste local, há dois pistões pneumáticos que ao serem acionados mudam para a posição de avanço e devem encaixar nos contatos da seccionadora, além disso, acima do local de teste há um suporte para a fiação elétrica e para a câmera utilizada para a visão computacional.

A fim de induzir o aprendizado do modelo de CNN, foi adicionado uma tonalidade azul conforme destacado na Figura 30 com o intuito de servir como referência de maneira estratégica, já que o fato desta faixa azul estar exposta no campo de visão da câmera implica diretamente no posicionamento incorreto da seccionadora, o contrário não é válido, o fato da faixa não estar no campo de visão da câmera não significa que a seccionadora está corretamente posicionada.

Figura 30 – Faixa azul de referência



Fonte: O autor.

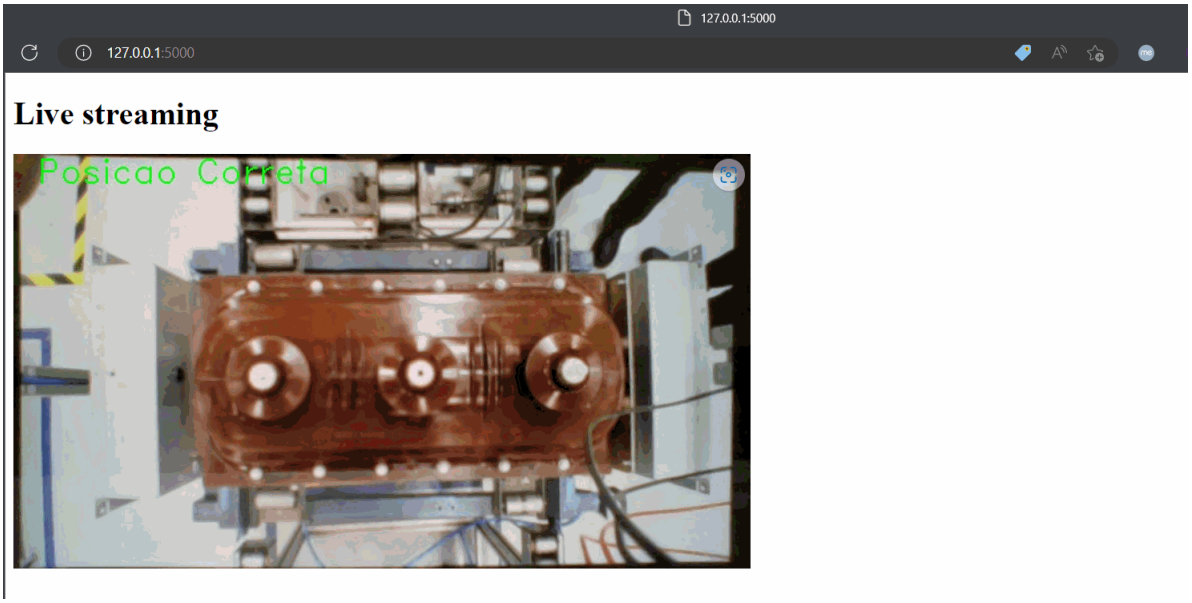
Esta referência tem como objetivo auxiliar na ativação dos filtros durante a etapa de convolução dos modelos, pois, as linhas horizontais da imagem são os parâmetros de posicionamento da peça, portanto, os filtros que detectam bordas horizontais tendem a ser ativados com maior relevância com a tonalidade azul exposta no frame, facilitando a inferência do modelo.

3.6 *DEPLOYMENT* DO MODELO VIA FLASK

Para utilizar o modelo em um ambiente de produção foi desenvolvido uma aplicação simples através do Flask com apenas duas rotas, sendo elas a página de destino padrão *@app.route('/')* e a rota criada para retornar as classificações do modelo *@app.route('/classificacao')*, além de um arquivo HTML integrado com o mecanismo *Jinja2* do Flask. Este arquivo por sua vez, foi utilizado para endereçar a rota da classificação do modelo dentro de uma das *tags* do HTML, de tal modo a retornar os frames e as classificações em tempo real conforme demonstrado na Figura 31, ou seja, ao acessar a página de destino padrão, a aplicação *Flask* renderiza o arquivo HTML especificado que

por sua vez retorna a rota *'/classificacao'* que possui a função responsável por classificar os *frames*. A classificação do posicionamento da peça pode ser visualizada no canto superior esquerdo do quadro no qual o *frame* está sendo disposto. Como o uso é local, não foi necessário hospedar a aplicação em um servidor.

Figura 31 – Aplicação web com o modelo DL

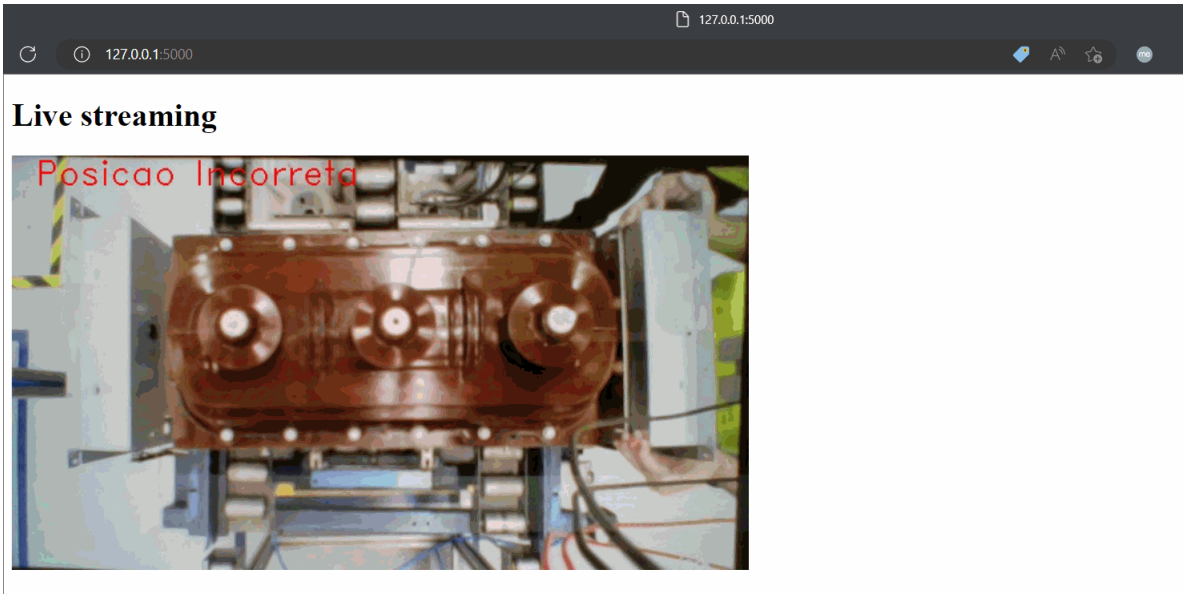


Fonte: O autor.

O texto informando o tipo de posicionamento e a cor do mesmo foram as duas indicações visuais utilizadas para auxiliar o operador, sendo a cor verde referente ao posicionamento correto e a cor vermelha o posicionamento incorreto demonstrado na Figura 32.

O modelo foi introduzido na aplicação como uma das etapas definidas na função dentro da rota *'/classificacao'* através da função *load_model* do *framework* TensorFlow, desta maneira foi possível classificar os *frames* através da função *predict* do mesmo *framework*.

Figura 32 – Exemplo da aplicação em posicionamento incorreto.



Fonte: O autor.

4 RESULTADOS OBTIDOS

Este capítulo tem o intuito de apresentar os resultados obtidos no treinamento, validação e teste dos quatro modelos de CNN, e por fim, os resultados do modelo com melhor desempenho nas etapas anteriores aplicado na linha de produção da qual a seccionadora faz parte.

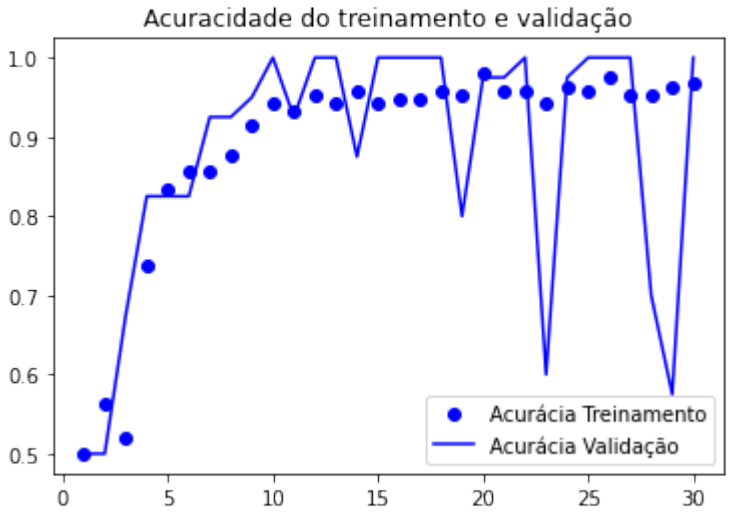
4.1 RESULTADOS DO TREINAMENTO E VALIDAÇÃO

Para o treinamento dos modelos foram utilizadas 210 imagens, sendo 105 representativas do posicionamento correto e 105 do posicionamento incorreto, garantindo a proporcionalidade de ambas as classificações. Este número representa 84% dos dados utilizados no treinamento, ou seja, do diretório *Treino* e *Validacao* conforme especificado na Seção 3.1.1.

4.1.1 Modelo 01

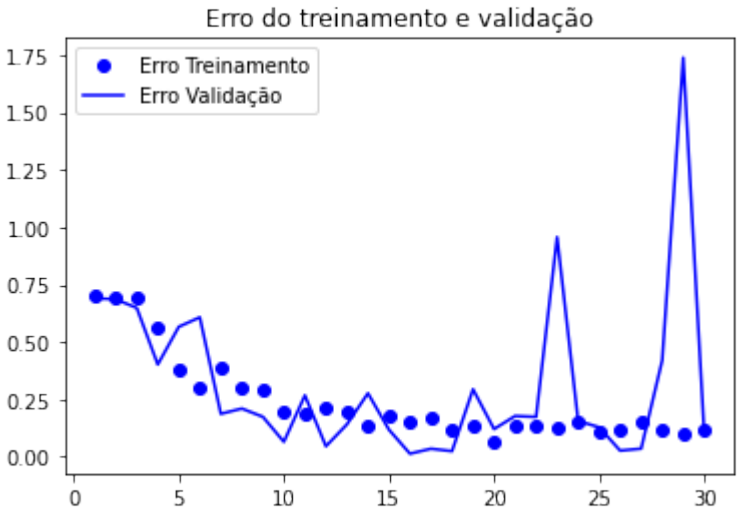
O modelo *modelo01* apresentou ruídos durante o treinamento para a classificação dos dados de validação conforme representado nas Figuras 33 e 34, oscilando sua acuracidade entre 60% e 98% para os dados de treinamento e de 50% a 100% para os dados de validação. No último *epoch* a sua acurácia foi de $96,67\%$ para os dados de treino e de 100% para os dados de validação.

Figura 33 – Acurácia do modelo01.



Fonte: O autor.

Figura 34 – Erro do modelo01.

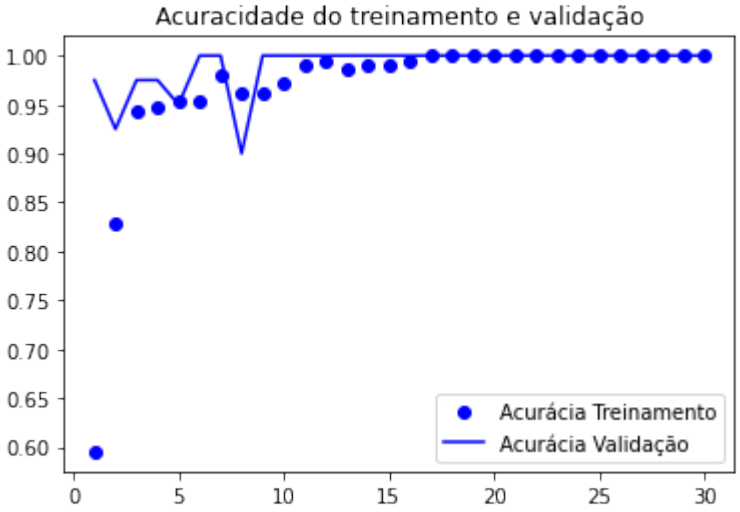


Fonte: O autor.

4.1.2 Modelo 02

O modelo *modelo02* apresentou uma curva de aprendizado mais característica de um modelo de CNN, na qual a sua acurácia aumenta conforme o avanço dos *epochs* e o seu erro diminui. Este comportamento pode ser visualizado nas Figuras 35 e 36 respectivamente. No último *epoch* a acurácia de treinamento e validação foi de 100%.

Figura 35 – Acurácia do modelo02.



Fonte: O autor.

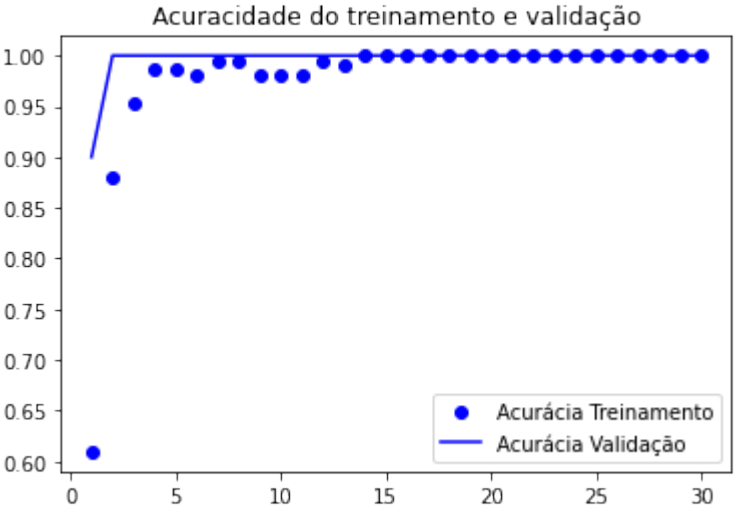
Figura 36 – Erro do modelo02.

Fonte: O autor.

4.1.3 Modelo 03

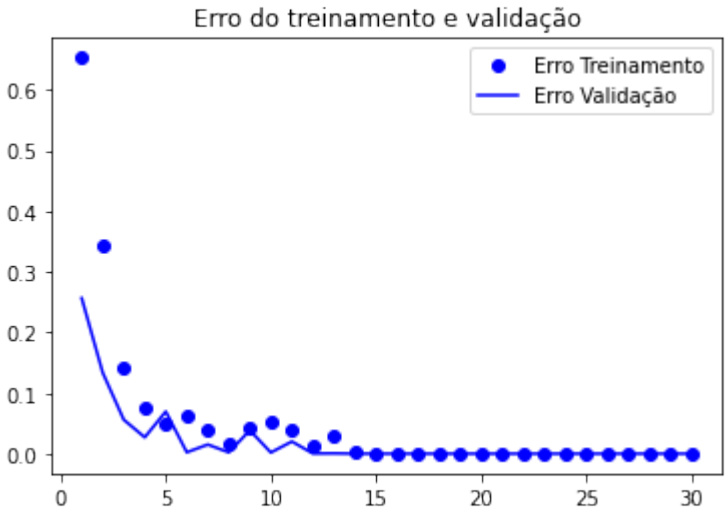
O modelo *modelo3* apresentou ainda menos ruídos do que o modelo *modelo02* tendo uma acuracidade constante do segundo *epoch* até o trigésimo, as métricas estão representadas nas Figuras 37 e 38. No último *epoch* a acurácia de treinamento e validação foi de 100%.

Figura 37 – Acurácia do modelo03.



Fonte: O autor.

Figura 38 – Erro do modelo03.

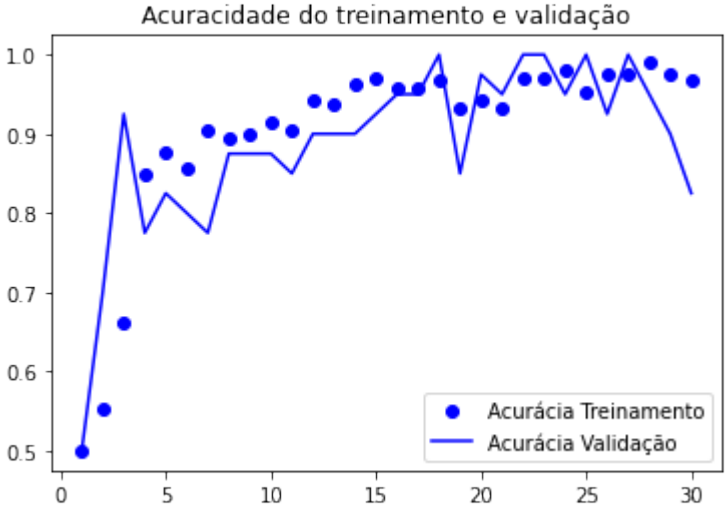


Fonte: O autor.

4.1.4 Modelo 04

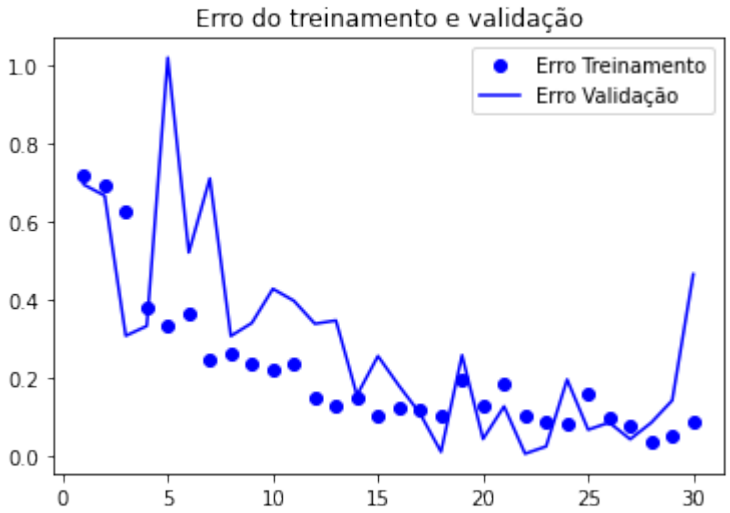
O modelo *modelo04* apresentou além de ruído durante o treinamento o pior desempenho dentre os quatro modelos conforme demonstrado nas Figuras 39 e 40. No último *epoch* a acurácia de treinamento foi de $96,67\%$ e para os dados de validação foi de $82,50\%$.

Figura 39 – Acurácia do modelo04.



Fonte: O autor.

Figura 40 – Erro do modelo04.



Fonte: O autor.

4.2 RESULTADOS DE TESTE

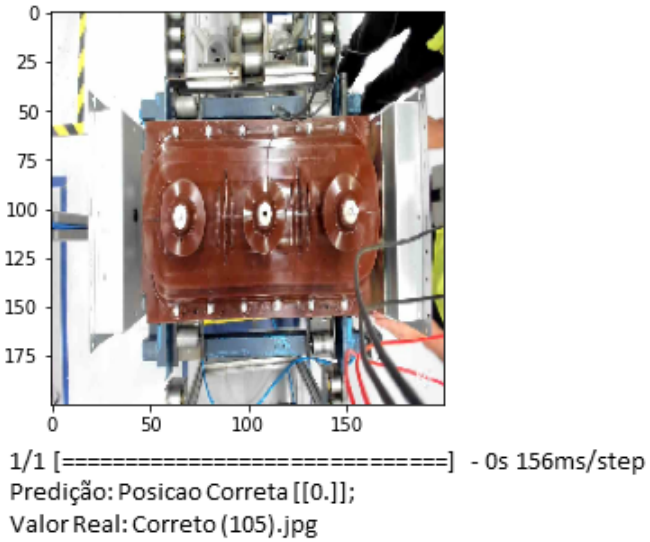
Para avaliar o desempenho dos modelos, 31 imagens que não foram aplicadas no treinamento dos mesmos foram utilizadas para inferência, sendo 15 imagens referentes ao posicionamento correto e 16 imagens para o posicionamento incorreto. Dois métodos foram utilizados para monitorar o desempenho dos modelos.

O primeiro consiste em retornar a classificação do modelo assim como o nome do arquivo que representa o tipo de posicionamento da imagem, possibilitando desta maneira, visualizar se as imagens classificadas de maneira incorreta pelo modelo representam uma situação complexa de classificar até mesmo pelo operador, ou então, se o modelo não está sendo capaz de classificar imagens que seriam facilmente classificadas pelo operador. Este primeiro método pode ser visualizado na Figura 41.

O segundo método consiste na utilização de uma matriz de confusão conforme explicitado no Seção 2.6.1. Para esta solução $N = 2$ (número de classes), ou seja, há quatro possíveis resultados na matriz de confusão, sendo verdadeiro positivo, falso positivo, verdadeiro negativo e falso negativo. O verdadeiro positivo significa que o modelo identificou o posicionamento correto da seccionadora e classificou corretamente permitindo a continuidade do teste da peça sem complicações. O falso positivo significa que o modelo não foi capaz de classificar corretamente o posicionamento da seccionadora, indicando que a peça estava no local correto sendo que o posicionamento está incorreto, e caso o teste prossiga haverá complicações, este é o resultado menos desejável dentre as quatro possibilidades, já que esse é o único que indica ao operador para prosseguir com o teste em uma situação não ideal. O verdadeiro negativo indica que o modelo identificou o posi-

cionamento incorreto da seccionadora e classificou corretamente, indicando ao operador para não prosseguir com o teste, e por último, o falso negativo significa que o modelo não foi capaz de classificar corretamente o posicionamento da seccionadora, indicando que ela está no lugar incorreto sendo que o posicionamento da mesma está correto, apesar deste resultado ser um falso resultado, não há complicações em termos de segurança já que o resultado orienta o operador a não prosseguir com o teste.

Figura 41 – Método 1 de monitoramento de inferência.



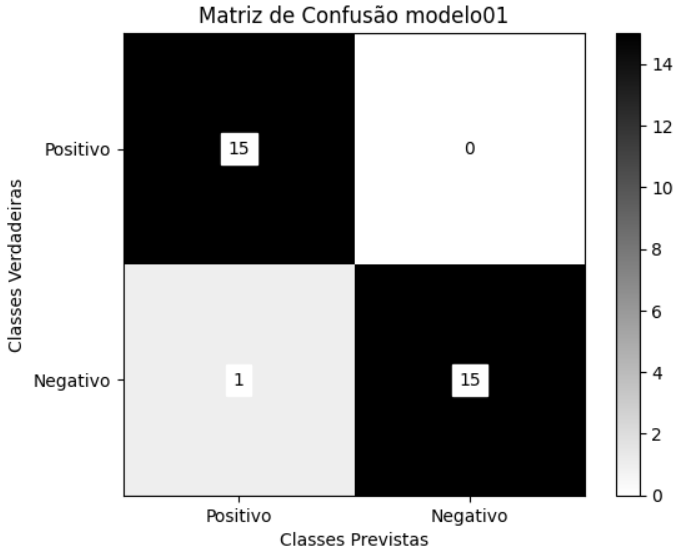
Fonte: O autor.

4.2.1 Modelo 01

O modelo *modelo01* classificou corretamente 30 das 31 imagens utilizadas durante o teste conforme representado na matriz de confusão da Figura 42. Das 30 classificações corretas, 15 foram referentes ao verdadeiro positivo, ou seja, 15 seccionadoras que estavam corretamente posicionadas no local de teste foram classificadas corretamente pelo modelo, e 15 refentes ao verdadeiro negativo, 15 seccionadoras que estavam posicionadas incorretamente foram classificas de maneira correta pelo modelo.

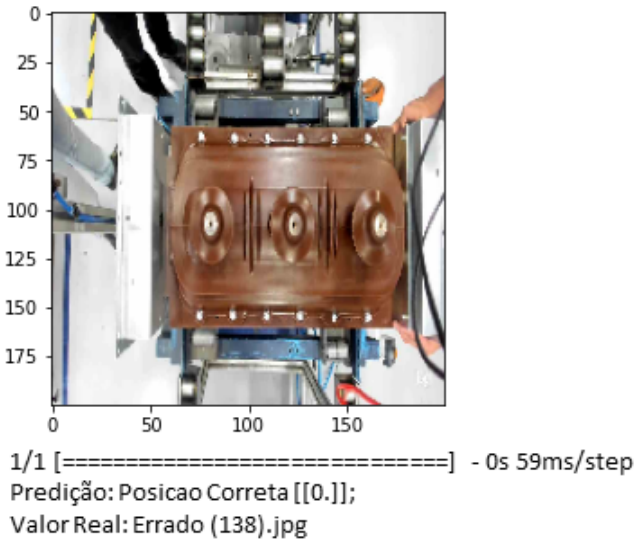
E a sua classificação incorreta foi um falso positivo representado na Figura 43. Conforme explicitado na Seção 3.5, a seccionadora na posição desta figura está consideravelmente próxima ao posicionamento correto, tendo em vista que a faixa azul de referência está coberta e que um operador teria que analisar de diversos ângulos para determinar o posicionamento da mesma.

Figura 42 – Matriz de Confusão *modelo01*.



Fonte: O autor.

Figura 43 – Falso positivo do modelo *modelo01*.



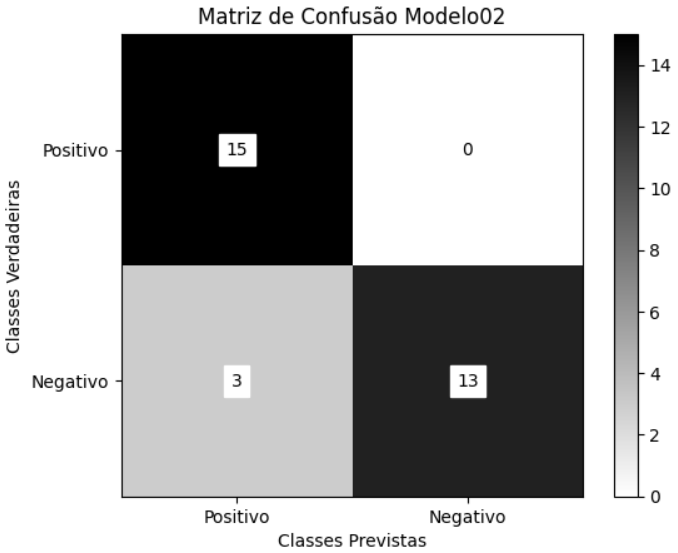
Fonte: O autor.

4.2.2 Modelo 02

O modelo *modelo02* apesar de ter obtido melhor desempenho durante o treinamento e validação, teve apenas 29 acertos das 31 imagens utilizadas durante o teste conforme demonstrado na matriz de confusão da Figura 44. Entre as 28 classificações verdadeiras, 15 foram referentes ao verdadeiro positivo, seccionadoras no local correto e corretamente classificada pelo modelo, e 13 referentes ao verdadeiro negativo, seccionadoras incorretamente

posicionadas e corretamente classificadas pelo modelo.

Figura 44 – Matriz de Confusão *modelo02*.

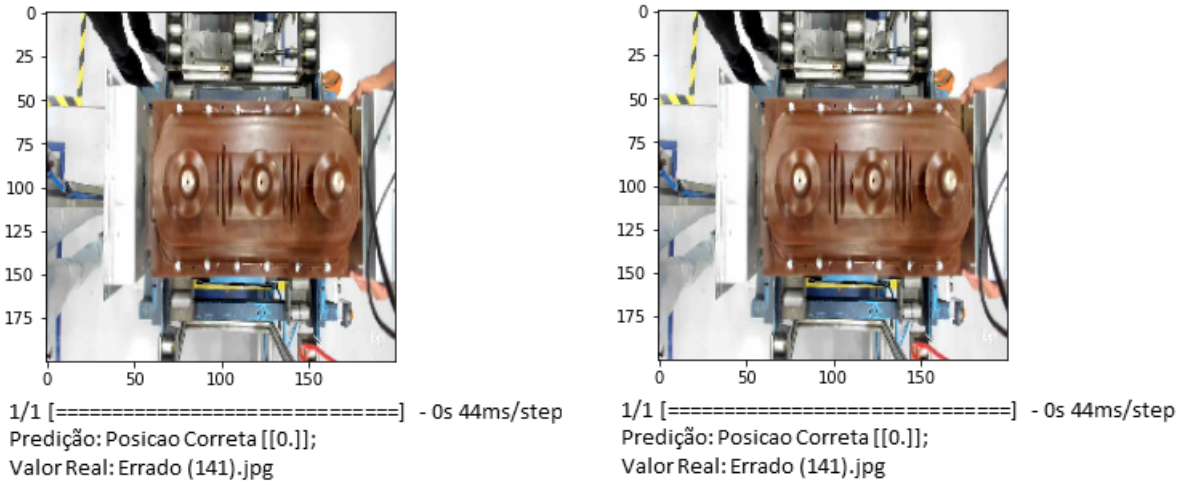


Fonte: O autor.

As três classificações incorretas foram três falso positivos e estão representadas nas Figuras 45 e 46, sendo que a Figura 46 representa a mesma imagem que também foi classificada de maneira errada pelo modelo *modelo01*. Pode-se observar que a faixa azul está visível em ambas as classificações na Figura 45, portanto, o modelo *modelo02* não foi capaz de aprender esta característica e associa-lá com o posicionamento incorreto da peça, se comparado com o desempenho de um operador, este modelo deixaria a desejar em termos de performance. já que um operador, ao visualizar a faixa azul chegaria na conclusão de que o posicionamento precisaria ser ajustado.

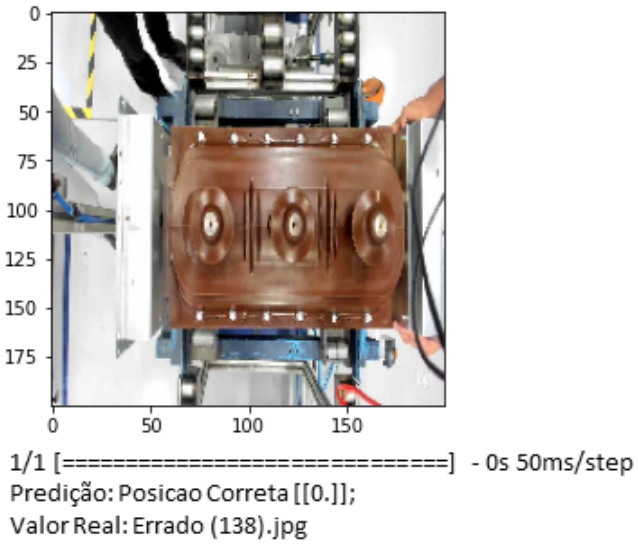
O fato das classificações falsas terem sido do tipo falso positivo, é um indicativo negativo em relação ao modelo, tendo em vista que este tipo de classificação é o único que orienta o operador da linha prosseguir com o teste em um cenário indesejado, aumentando a ineficiência da linha. Considerando este cenário no qual o modelo diz que a seccionadora está corretamente posicionada, uma afirmação falsa, o operador acionaria o botão que altera o posicionamento dos pistões pneumáticos para a posição de avanço, e os contatos responsáveis por fechar o circuito elétrico não estariam na posição correta, e ao iniciar o teste, o operador iria identificar este erro e reiniciaria todo o processo de posicionamento da peça, conferência do posicionamento e inicialização do teste, prejudicando assim, a eficiência da linha.

Figura 45 – Classificações erradas 01 e 03 do modelo *modelo2*.



Fonte: O autor.

Figura 46 – Classificação errada 02 do modelo *modelo2*.



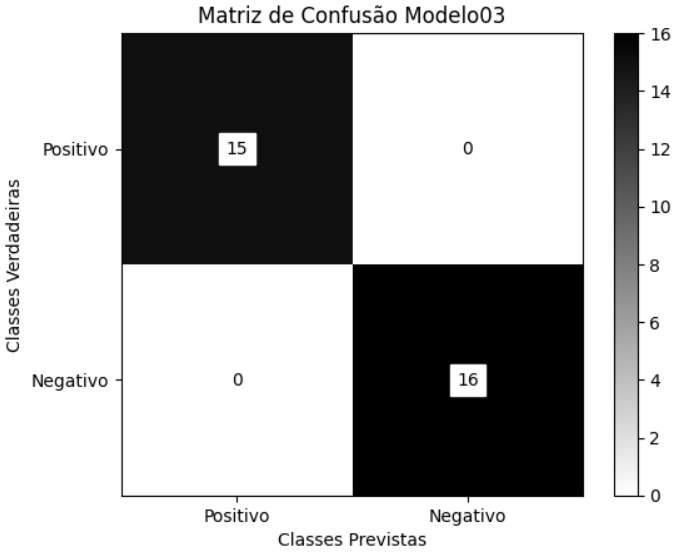
Fonte: O autor.

4.2.3 Modelo 03

O modelo *modelo03* classificou corretamente 31 das 31 imagens de teste utilizadas, conforme representado na matriz de confusão da Figura 47, isso significa que, inclusive a imagem classificada de maneira incorreta pelos modelos *modelo01* e *modelo02* foi classificada de maneira correta pelo modelo *modelo03* conforme demonstrado na Figura 48.

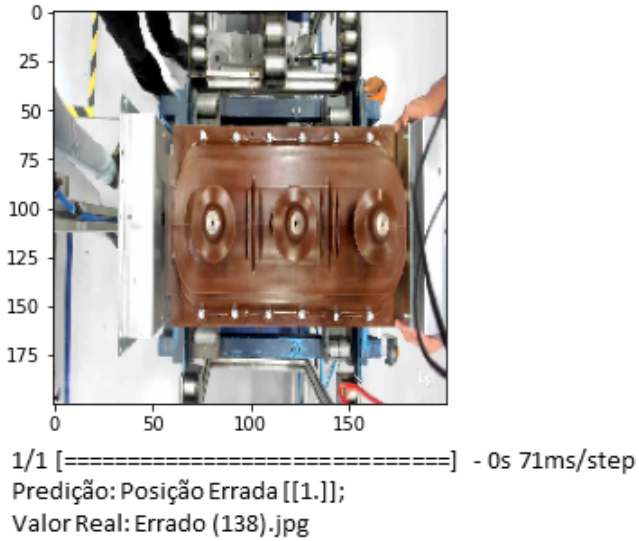
Considerando que seria necessário analisar por diversos ângulos a seccionadora

Figura 47 – Matriz de Confusão *modelo03*.



Fonte: O autor.

Figura 48 – Classificação correta da imagem "Errado (138)"pelo *modelo03*.



Fonte: O autor.

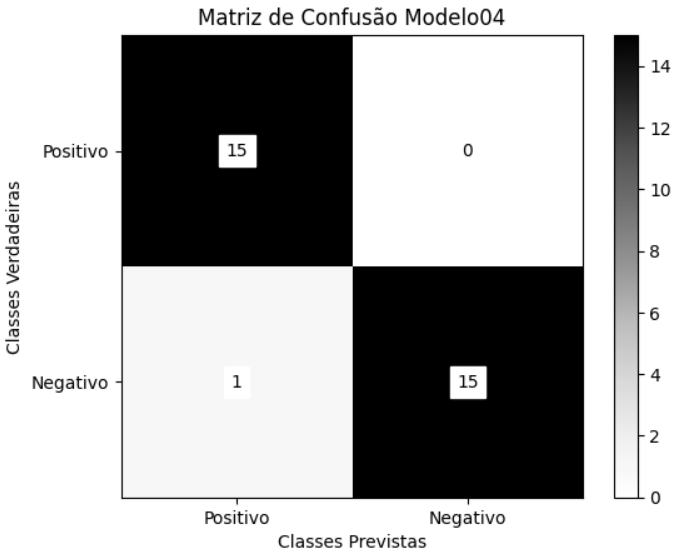
para concluir sobre o posicionamento da mesma, o modelo *modelo03* apresentou um desempenho melhor do que a de um operador da linha.

4.2.4 Modelo 04

O modelo *modelo04* apesar de ter apresentado menor acurácia e mais ruídos durante o treinamento e validação do que o modelo *modelo02*, ele classificou corretamente 30 das 31

imagens utilizadas para teste conforme demonstrado na matriz de confusão da Figura 49, sendo a sua classificação incorreta um falso positivo. A imagem que foi classificada de maneira incorreta é a mesma dos modelos *modelo01* e *modelo02*, portanto, a justificativa utilizada anteriormente também é válida para este modelo.

Figura 49 – Matriz de Confusão *modelo04*.



Fonte: O autor.

A Tabela 10 apresenta o desempenho que os modelos apresentaram durante a classificação dos dados de teste, ou seja, dados que não haviam sido apresentados durante a etapa de treinamento.

Modelo	Acertos	Erros
Modelo01	30	1
Modelo02	28	3
Modelo03	31	0
Modelo04	30	1

Tabela 10 – Desempenho nos dados de teste.

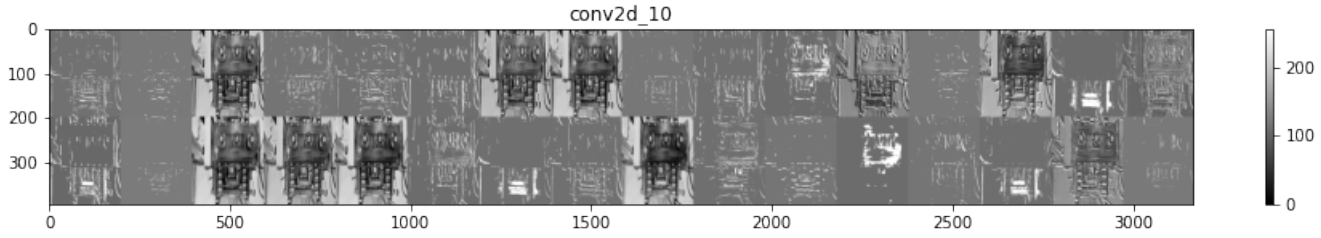
4.2.5 Considerações do modelo *modelo03*

Em via dos resultados apresentados, é possível concluir que o modelo *modelo03* apresentou melhor desempenho tanto em termos de treinamento quanto em termos de inferência em dados jamais vistos pelos modelos anteriormente, contrariando o que se esperava de que os modelos com técnicas para evitar *overfitting* teriam melhor desempenho, tendo em vista que, por possuir poucas imagens para treino, era esperado que pudesse

ocorrer este fenômeno. O fato disso ter acontecido demonstra que, apesar de possuir um número pequeno de demonstrações de ambas as classificações possíveis, foi o suficiente para o modelo *modelo03* generalizar e aprender as características necessárias para definir que tipo de posicionamento a seccionadora se encontra.

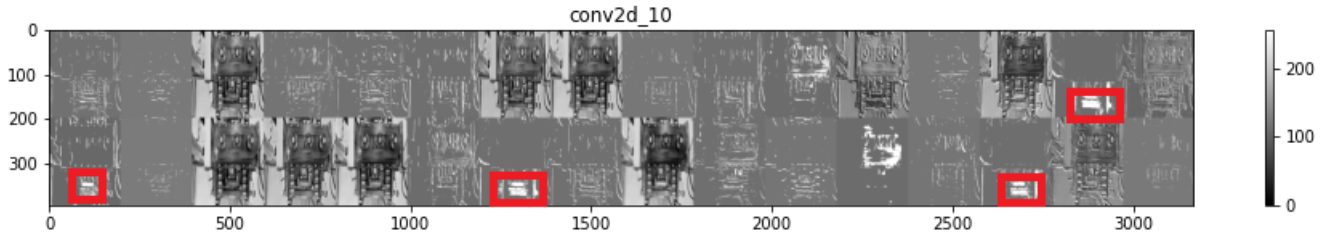
Dito isso, foi criado um modelo intermediário através da biblioteca *Model* do framework *TensorFlow* que possibilitou utilizar as camadas ocultas do modelo *modelo03* como entrada deste modelo intermediário, através deste processo foi possível adicionar um mapa de calor de ativação das camadas de convolução do modelo, de tal forma a possibilitar a visualização das regiões mais ativadas pelos filtros durante o processo de aprendizagem. Optou-se por utilizar a primeira camada de convolução do modelo denominada de *conv2d_10* que pode ser visualizada na Tabela 5

Figura 50 – Mapa de calor de ativação da primeira camada de convolução do modelo para posicionamento incorreto.



Fonte: O autor.

Figura 51 – Mapa de calor de ativação da primeira camada de convolução do modelo para posicionamento incorreto com a faixa destacada.

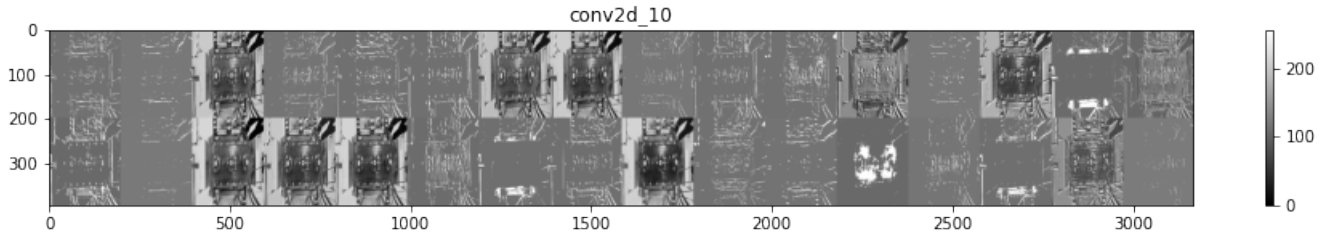


Fonte: O autor.

A imagem utilizada na Figura 50 e Figura 51 é de um posicionamento incorreto da seccionadora e é possível visualizar que alguns filtros ativam de maneira significativa a faixa azul inserida como referência conforme especificado na Seção 3.5 e demonstrado na Figura 30, demonstrando que de fato o modelo a utilizou como referência. Isto também é confirmado repetindo o processo para uma imagem do posicionamento correto da peça, conforme pode ser visualizado na Figura 52 na qual em nenhum momento a faixa azul

foi ativada pelos filtros, tendo em vista que, o posicionamento correto da seccionadora implica que a faixa azul não estará exposta para o modelo.

Figura 52 – Mapa de calor de ativação da primeira camada de convolução do modelo para posicionamento correto.



Fonte: O autor.

5 CONCLUSÃO

O objetivo deste trabalho era desenvolver uma rede neural convolucional capaz de inferir corretamente sobre o posicionamento da seccionadora no local de teste, com o intuito de descentralizar a eficiência do processo do nível de conhecimento do operador na linha de produção e atender os requisitos do programa de padronização de processos da empresa.

Durante o desenvolvimento da solução, 280 imagens representativas dos posicionamentos incorretos e corretos foram coletadas para treinamento, validação e teste do modelo desenvolvido, assim como a adição de características físicas no ambiente de teste para auxiliar no aprendizado do algoritmo. Os resultados obtidos durante o teste, ou seja, após o treinamento e validação, demonstraram que um modelo de CNN é capaz de inferir corretamente sobre o posicionamento da peça. Em via disso, é viável a aplicação de *deep learning* para processos industriais, cujo objetivo seja não só aumentar a eficiência mas também a escalabilidade do mesmo.

Os resultados do estudo realizado neste trabalho contribui para o uso de *deep learning* em indústrias, especificamente em reconhecimento de padrões relacionados ao posicionamento de peças, assim como a possibilidade de induzir o aprendizado da rede neural convolucional adicionando características físicas no ambiente a fim de aprimorar o desempenho do modelo, além de fortalecer a ideia de que a inteligência artificial é um recurso que visa auxiliar e não substituir os esforços humanos empregados no processo.

Por se tratar de um ambiente industrial com um fluxo considerável de entrada e saída de pedidos, o teste em tempo real dos modelos e do *endpoint* desenvolvido foi prejudicado devido a questões logísticas e internas da empresa, já que para realizar este procedimento seria necessário parar a linha de produção. Portanto, dentre os cinco objetivos listados no Capítulo 1, somente o objetivo de utilizar a solução em um ambiente de produção não foi concluído de maneira satisfatória devido aos motivos citados. Como trabalho futuro, sugere-se a utilização da classificação do modelo como trava de segurança, impedindo que o procedimento do teste seja iniciado no *software* caso a classificação do modelo seja negativa, além da utilização de um servidor para utilização do modelo em produção de forma mais robusta.

Em via dos resultados apresentados, este trabalho confirmou que é possível descentralizar a eficiência do processo do conhecimento do operador para uma solução de *deep learning*, e que além da capacidade dos modelos de CNN em identificar objetos e anomalias, eles também são capazes de aprender padrões e inferir sobre o posicionamento baseado neste aprendizado. Espero que este trabalho sirva de inspiração para o aprimoramento do uso de CNN em ambientes industriais.

REFERÊNCIAS

AURÉLIEN GÉRON. **Hands-On Machine Learning with Scikit-Learn and TensorFlow**. 1. ed. Estados Unidos: O'Reilly Media, Inc., 2017. P. 363.

AWS. **O que é inteligência artificial?** Machine Learning e aprendizado profundo. [*S.l.*], 2023. Disponível em: https://aws.amazon.com/pt/machine-learning/what-is-ai/. Acesso em: 30 jan. 2023.

DRIDI, Salim. Unsupervised Learning - A Systematic Literature Review, dez. 2021.

FRANÇOIS CHOLLET. **DEEP LEARNING with Python**. 4. ed. Nova Iorque: Manning Publications Co., 2021. P. 27.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [*S.l.*]: MIT Press, 2016. http://www.deeplearningbook.org.

IBM. **What is a neural network?** [*S.l.*], 2023. Disponível em: https://www.ibm.com/topics/neural-networks. Acesso em: 19 abr. 2023.

IDE, Hidenori; KURITA, Takio. Improvement of learning for CNN with ReLU activation by sparse regularization. *In*: 2017 International Joint Conference on Neural Networks (IJCNN). [*S.l.*: *s.n.*], 2017. P. 2684–2691.

MATLAB. **What Is a Convolutional Neural Network?** 3 things you need to know. [*S.l.*], 2023. Disponível em: https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html. Acesso em: 23 abr. 2023.

MATLAB. **What Is Overfitting?** Avoid overfitting machine learning models. [*S.l.*], 2023. Disponível em: https://www.mathworks.com/discovery/overfitting.html. Acesso em: 23 abr. 2023.

NWADIUGWU, Martin C. **Neural Networks, Artificial Intelligence and the Computational Brain**. [*S.l.*]: arXiv, 2021. Disponível em: https://arxiv.org/abs/2101.08635.

P.SIBI. **ANALYSIS OF DIFFERENT ACTIVATION FUNCTIONS USING BACK PROPAGATION NEURAL NETWORKS**. 47. ed. Kumbakonam: Journal of Theoretical e Applied Information Technology, 2013. P. 1266.

PANDEY, Mohit; FERNANDEZ, Michael; GENTILE, Francesco; ISAYEV, Olexander; TROPSHA, Alexander; C, Abraham; STERN; CHERKASOV, Artem. The transformational role of GPU computing and deep learning in drug discovery. **Nature Machine Intelligence**, 2022.

PEREZ, Luis; WANG, Jason. **The Effectiveness of Data Augmentation in Image Classification using Deep Learning**. [*S.l.*: *s.n.*], 2017. arXiv: 1712.04621 [cs.CV].

RUDER, Sebastian. An overview of gradient descent optimization algorithms. **Insight Centre for Data Analytics**, 2017.

RUFAI, Aminah Mardiyyah. **Linear Regression From Scratch PT2: The Gradient Descent Algorithm**. [*S.l.*], 2023. Disponível em: https://medium.com/nerd-for-tech/linear-regression-from-scratch-pt2-the-gradient-descent-algorithm-f30d42fea40c. Acesso em: 20 abr. 2023.

SHANU, Shashank. **Sigmoid Activation Function**. [*S.l.*], 2023. Disponível em: https://insideaiml.com/blog/Sigmoid-Activation-Function-1031. Acesso em: 31 jan. 2023.

SHARMA, Siddharth; SHARMA, Simone. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. **International Journal of Engineering Applied Sciences and Technology**, p. 310–316, 2020.

SURIYA, M.; CHANDRAN, V.; SUMITHRA, M.G. Enhanced deep convolutional neural network for malaria parasite classification. **International Journal of Computers and Applications**, 2019.

THE PALLETS PROJECTS. **Flask**. [*S.l.*], 2023. Disponível em: https://palletsprojects.com/p/flask/. Acesso em: 30 jan. 2023.

TONG MAO ZHONGJIE SHI, Ding-Xuan Zhou. Theory of Deep Convolutional Neural Networks III: Approximating Radial Functions. **School of Data Science**, 2021.

YING, Xue. An Overview of Overfitting and its Solutions. **Journal of Physics: Conference Series**, 2019.