Universidade de Brasília
Faculdade de Tecnologia

Análise automática de imagens de soldagem GMAW utilizando técnicas de inteligência artificial

Lucas Henrique Alves Rosa

PROJETO FINAL DE CURSO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2023

Universidade de Brasília
Faculdade de Tecnologia

Análise automática de imagens de soldagem GMAW utilizando técnicas de inteligência artificial

Lucas Henrique Alves Rosa

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Dr. Jones Yudi Mori Alves da Silva

Brasília
2023

Rosa, Lucas Henrique Alves.
R789a Análise automática de imagens de soldagem GMAW utilizando técnicas de inteligência artificial / Lucas Henrique Alves Rosa; orientador Jones Yudi Mori Alves da Silva . -- Brasília, 2023.
132 p.

Projeto Final de Curso (Engenharia de Controle e Automação) -- Universidade de Brasília, 2023.

1. Soldagem GMAW. 2. Inteligência Artificial. 3. Redes Neurais Convolucionais. 4. Identificação de Parâmetros. I. , Jones Yudi Mori Alves da Silva, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Análise automática de imagens de soldagem GMAW utilizando técnicas de inteligência artificial

Lucas Henrique Alves Rosa

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 5 de outubro de 2023:

Prof. Dr. Jones Yudi Mori Alves da Silva, UnB/FT/ENM
Orientador

Prof. Dr. José Maurício Santos Torres da Motta , UnB/FT/ENM
Examinador interno

Prof. Dr. Alysson Martins Almeida Silva , UnB/FT/ENM
Examinador interno

Brasília
2023

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Gostaria de aproveitar este momento para expressar minha imensa gratidão e reconhecimento a todas as pessoas que estiveram ao meu lado durante a jornada de realização do meu TCC. Em especial, quero agradecer ao meu orientador, Professor Jones Yudi, por sua orientação valiosa ao longo deste processo.

Também quero expressar meu profundo agradecimento à minha família, que sempre esteve ao meu lado, fornecendo apoio emocional e encorajamento inabalável. Seu amor e apoio foram fundamentais para me manter motivado e determinado a superar todos os desafios que encontrei pelo caminho.

Agradeço também à minha amada, Ana Clara Cotrim, por sua compreensão, incentivo e por sempre acreditar em mim. Sua presença constante e apoio incondicional foram verdadeiros pilares para o meu sucesso.

Além disso, gostaria de estender meu agradecimento aos amigos do curso, que estiveram ao meu lado durante essa jornada acadêmica. Compartilhamos momentos de estudo, discussões e desafios, e essas experiências enriqueceram meu aprendizado. Seu apoio mútuo e camaradagem foram essenciais para superarmos obstáculos juntos.

Não posso deixar de agradecer também a todos os professores que fizeram parte da minha formação acadêmica. Suas aulas inspiradoras, conhecimento transmitido e dedicação à educação foram fundamentais para a minha evolução como estudante e pesquisador.

Por fim, agradeço a todos aqueles que, de alguma forma, contribuíram para a conclusão bem-sucedida do meu trabalho. Sou profundamente grato pela oportunidade de aprender e crescer durante esse processo, e levo comigo lições valiosas que serão fundamentais para minha vida profissional e pessoal.

“O sucesso é a soma de pequenos esforços
repetidos dia após dia”
(Robert Collier)

Resumo

O objetivo deste trabalho é desenvolver uma inteligência artificial capaz de identificar, a partir do processo de soldagem GMAW (*Gas Metal Arc Welding*) convencional, a posição e largura da poça de soldagem, assim como a posição e altura do arame, com base em imagens do próprio processo. A soldagem GMAW é amplamente utilizada na indústria, e a capacidade de identificar parâmetros durante o processo pode auxiliar na otimização e controle da qualidade da solda.

Para alcançar esse objetivo, foram criadas redes neurais convolucionais que realizam a identificação de parâmetros relacionados ao momento da soldagem, poça de fusão e o arame, também conhecido como eletrodo. A arquitetura das redes neurais convolucionais foi baseada em modelos conhecidos, como LeNet, AlexNet e VGG, com ajustes em quase todos os parâmetros para melhorar o desempenho. Inicialmente, a rede não atingiu as expectativas para problemas de regressão, mas mostrou resultados promissores para problemas de classificação.

Como alternativa, adotou-se uma segunda abordagem utilizando a arquitetura YOLO (You Only Look Once), reconhecida por sua eficiência e precisão na detecção de objetos em imagens. A rede neural convolucional baseada na arquitetura YOLO foi treinada com um conjunto de dados adequado para o processo de soldagem GMAW convencional. Os resultados obtidos foram promissores, com a capacidade de identificar com precisão os parâmetros relacionados ao processo de soldagem a partir das imagens analisadas.

O projeto envolveu uma pesquisa abrangente sobre arquiteturas e técnicas usadas em redes neurais, além de experimentação e ajuste fino para obter os melhores resultados. As descobertas deste estudo contribuem para o campo da identificação automatizada de parâmetros em processos de soldagem, oferecendo potencial para maior eficiência, controle de qualidade e otimização em aplicações de soldagem industrial.

Palavras-chave: Soldagem GMAW. Inteligência Artificial. Redes Neurais Convolucionais. Identificação de Parâmetros.

Abstract

This work aims to develop an artificial intelligence capable of identifying welding parameters, such as welding pool position and width, as well as wire position and height, from images of the conventional Gas Metal Arc Welding (GMAW) process. GMAW welding is widely used in the industry, and the ability to identify parameters during the process can assist in optimizing and controlling the welding quality.

To achieve this goal, convolutional neural networks were created to perform the identification of welding parameters, including the welding moment, fusion pool, and wire (electrode) aspects. The architecture of the convolutional neural networks was based on well-known models, such as LeNet, AlexNet, and VGG, with adjustments to enhance their performance. Initially, the network did not meet expectations for regression problems, but it showed promising results for classification tasks.

As an alternative, a second approach was adopted using the YOLO (You Only Look Once) architecture, renowned for its efficiency and accuracy in object detection in images. The YOLO-based convolutional neural network was trained with a suitable dataset for the conventional GMAW welding process. The obtained results were promising, as it accurately identified welding parameters from the analyzed images.

The project involved comprehensive research on neural network architectures and techniques, along with experimentation and fine-tuning to achieve the best results. The findings of this study contribute to the field of automated identification of welding parameters, offering potential for increased efficiency, quality control, and optimization in industrial welding applications.

Keywords: GMAW Welding. Artificial Intelligence. Convolutional Neural Networks. Parameter Identification.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CNN	Convolutional Neural Network 49
DIB	*Device Independent Bitmap* 67
ELU	Exponential Linear Unit 33
FPGA	*field-programmable gate array* 56
GMAW	Gas Metal Arc Welding 17
IA	Inteligência Artificial 18
ILSVRC	*ImageNet Large Scale Visual Recognition Competition* 58
LMS	Least mean squares 42
LREN	*liquid rocket engine nozzle* 62
MAE	Mean Absolute Error 93
MAG	metal active gas 23
MIG	metal inert gas 23
MLP	multilayers perceptron 40
MSE	Mean Squared Error 80
MSE	*Mean Squared Error* 93
PANet	*Path Aggregation Network* 61
PMC	Perceptron de múltiplas camadas 40
R2	R-squared 93
ReLU	Rectified Linear Unit 32
RNA	Rede Neural Artificial 27
RNC	Rede neural convolucional 49
SELU	Scaled Exponential Linear Unit 33
SPP	*Spatial Pyramid Pooling* 61
UnB	Universidade de Brasília 56
VGG	*Visual Geometry Group* 58
VLSI	Very Large-Scale Integration 27
WFS	wire feed speed 64
YOLO	*You Only Look Once* 60

Lista de símbolos

Símbolos romanos

A	Altura do volume de entrada de uma camada convolucional	52
A_{mp}	Altura do volume resultante de uma camada convolucional	52
$d^{(k)}$	k-ésima amostra do treinamento	38
F	Tamanho do filtro usado em uma camada convolucional	52
fan_{in}	Número de unidades de entrada no tensor	88
fan_{out}	Número de unidades de saída do tensor	88
g	Função de ativação	30
k	k-ésima amostra do treinamento	38
L	Largura do volume de entrada de uma camada convolucional	52
L_{mp}	largura do volume resultante de uma camada convolucional	52
$loss$	*Loss function*	82
P	Tamanho do *zero-padding* usado em uma camada convolucional	52
S	Passo do filtro usado em uma camada convolucional	52
u	Potencial de ativação	30
var	variância	88
w_i	Pesos sinápticos	30
x_i	Sinais de entrada	29
y	Sinal de saída	30
y_{pred}	Valor de saída da rede	82
y_{true}	Valor de saída real ou esperado	82

Símbolos gregos

η	Taxa de aprendizagem da rede	38
σ	Combinador linear	30
θ	Limiar de ativação	30

Sumário

1 Introdução

A soldagem é um processo de união de metais que tem sido amplamente utilizado em diversas indústrias, como a automotiva, aeronáutica, construção civil, naval, entre outras. A arte de unir dois ou mais materiais metálicos já é conhecida pelo homem desde a pré-história na união de ligas de ouro e cobre, mas foi no século XIX com a chegada da revolução industrial que a soldagem começou a ter papel importante em diversas aplicações relacionadas a construção, impermeabilização, reparo e manutenção, (WEMAN, 2012). A grande vantagem da soldagem sobre os processos de união de metais se dá pela simplicidade e economia, uma vez a técnica exige pequenas quantidades de material. A técnica envolve a fusão das peças que serão unidas, geralmente com a ajuda de um material de adição, como o metal de solda, formando uma ligação permanente entre elas. A importância da soldagem se dá pelo fato de que ela permite a união de materiais que, de outra forma, não poderiam ser unidos, além de possibilitar a criação de peças mais complexas e resistentes. No entanto, é importante ressaltar que a soldagem requer habilidade e conhecimento técnico para garantir a qualidade e a segurança das uniões realizadas.

Existem diversos tipos de processos de soldagem, o processo que analisaremos nesse trabalho é a soldagem do tipo GMAW (Gas Metal Arc Welding). É um processo de união de metais que utiliza um arco elétrico para fundir o metal base e o metal de adição, que é fornecido através de um arame contínuo alimentado automaticamente na solda. A soldagem GMAW é um processo versátil e amplamente utilizado na indústria devido à sua alta produtividade e qualidade das juntas produzidas. Ele pode ser utilizado em uma ampla gama de espessuras de materiais, desde chapas finas até peças mais espessas, e em uma variedade de materiais, incluindo aços carbono, aços inoxidáveis, alumínio, ligas de níquel e outros metais.

A soldagem em si não constitui o objetivo principal de sua aplicação, entretanto como afeta diretamente a segurança e a economia da aplicação, seu estudo torna-se cada vez mais importante sendo considerado um dos itens principais em processos que a envolvem. Por isso além do projeto adequado da junta soldada é necessário seguir uma sequência de operações que inclui a qualificação dos procedimentos, bem como a seleção dos métodos de inspeção para garantir a estrutura as características funcionais segundo as quais foi projetada, (OKUNMURA; TANIGUCHI, 1982).

As indústrias tendem a substituir o trabalho manual por sistemas automatizados pela falta de mão de obra especializada que é escassa e por consequência, de alto custo, esses sistemas variam de simples automações de deslocamento até soldagens completamente robotizadas. Porém sistemas automatizados dificilmente detectam a qualidade do cordão

depositado, que por outro lado, um operador qualificado faz com facilidade e atua sobre o processo corrigindo os parâmetros envolvidos no processo. É possível observar que trata-se de um sistema de malha fechada de alta complexidade e que mesmo com sistemas automatizados se torna necessário a presença de um operador para lidar com ajustes no processo.

No monitoramento dos parâmetros de soldagem é comum observar o monitoramento da tensão e corrente de soldagem, que são parâmetros facilmente observados pelo baixo custo do sistema de monitoramento, porém alguns aspectos do processo também podem ser monitorados. Parâmetros como a geometria da poça, altura do arame, posição do arame e luminosidade podem ser monitorados com visão computacional. Avaliações de parâmetros visuais podem ser sujeitas a erros e inconsistências, além disso analisar grandes quantidades de imagens manualmente não é a solução mais eficiente. Aqui entra o desenvolvimento desse projeto, em que consiste em utilizar um aproximador universal para identificar parâmetros que envolvem o processo de soldagem.

Na década de 90 havia estudos sobre o uso da visão computacional para inspeção pós soldagem. Esses estudos eram aplicados pós soldagem devido ao fato da capacidade de processamento da época não ser capaz de realizar a inspeção em tempo real. A intenção na época era ter um processo que tivesse uma menor exposição do operador nos ambientes de soldagem, um monitoramento completo e ininterrupto da produção. Hoje com o desenvolvimento da tecnologia é possível não só reduzir o tempo de exposição do operador como também substituir sua participação por uma inteligência artificial, que além de monitorar todos os parâmetros é capaz de informar com precisão e rapidez as variáveis que envolvem o processo.

(WINSTON, 1977) define a inteligência artificial (IA) da seguinte forma “IA é o estudo das ideias que permitem habilitar os computadores a fazerem coisas que tornam as pessoas inteligentes”. É uma área da ciência da computação que se dedica a criar sistemas capazes de realizar tarefas que normalmente requerem inteligência humana, como reconhecimento de fala, visão computacional, tomada de decisão e aprendizado de máquina. Uma das técnicas mais importantes dentro da IA é a utilização de redes neurais, que são modelos computacionais inspirados na estrutura e funcionamento do cérebro humano. Isso significa que esses modelos possuem a capacidade de aquisição e manutenção do conhecimento e podem ser definidas como um conjunto de unidades de processamento.

As redes neurais são compostas por camadas de neurônios artificiais interconectados, que processam informações de entrada e geram uma saída ou resposta. Essas redes são treinadas com um conjunto de dados para aprender a reconhecer padrões e realizar tarefas específicas. Durante o treinamento, a rede ajusta seus pesos e conexões para maximizar sua capacidade de acertar as respostas corretas. Elas são especialmente úteis em tarefas que envolvem grande quantidade de dados e complexidade, nas quais é difícil para os humanos

identificar padrões e relações.

Uma das vantagens das redes neurais é a sua capacidade de generalizar a partir do aprendizado. Isso significa que uma rede neural treinada com um conjunto de dados pode ser utilizada para realizar tarefas semelhantes em outras situações. Além disso, as redes neurais podem ser combinadas com outras técnicas de IA, como algoritmos genéticos e lógica fuzzy, para melhorar sua eficácia e precisão.

No entanto, as RNAs também têm algumas limitações. Elas podem ser computacionalmente intensivas e requerer grande quantidade de dados para treinamento, além de serem propensas a erros devido o sobreajuste ou subajuste aos dados de treinamento. Além disso, a interpretabilidade das decisões tomadas pelas redes neurais pode ser difícil de ser compreendida pelos humanos, o que pode gerar preocupações éticas e de segurança.

1.1 Objetivo

O objetivo deste trabalho consiste em desenvolver uma inteligência artificial capaz de identificar, a partir de imagens de soldagem GMAW convencional, parâmetros como a posição e largura da poça de soldagem, a posição e altura do arame de solda e momentos de curto-circuito. Para alcançar esse propósito, os seguintes objetivos específicos devem ser alcançados:

- Rotulação de 6000 imagens de soldagem, assim como a criação de um banco de dados para armazenar a os parâmetros obtidos.
- Explorar técnicas de processamento de imagens, aprendizado de máquina e redes neurais convolucionais.
- Desenvolver uma solução de inteligência artificial para análise automática de imagens de soldagem GMAW
- Avaliar a eficácia da solução desenvolvida.

A rede neural será treinada utilizando um conjunto de imagens previamente rotuladas, que correspondem aos diferentes estados e configurações da soldagem GMAW, incluindo informações sobre a geometria e posição do arame de soldagem, a poça de solda e a identificação dos momentos de curto-circuito do processo.

Durante a fase de treinamento, a rede neural aprenderá a reconhecer padrões nas imagens e associá-los aos rótulos corretos. Essa capacidade permitirá que a rede, posteriormente, analise novas imagens e realize a identificação automática dos parâmetros desejados. Essa abordagem visa reduzir a necessidade de intervenção humana e agilizar o processo de análise.

O objetivo é proporcionar benefícios significativos para a área de soldagem GMAW no modo curto-circuito. A automação do processo de análise e medição possibilitará maior precisão e rapidez na obtenção dos parâmetros relevantes, o que pode resultar em melhorias na qualidade das soldas, no controle do processo e na redução de defeitos.

1.2 Organização do texto

A organização do trabalho segue uma estrutura clássica, composta pelas seguintes seções: Introdução, Fundamentação Teórica, Referências Bibliográficas, Metodologia, Avaliação de Resultados e Conclusão. Cada seção desempenha um papel fundamental na apresentação e desenvolvimento do estudo.

A seção de Introdução tem como objetivo contextualizar o tema do trabalho, fornecendo uma visão geral do problema abordado e justificando a relevância do estudo. Nessa seção, são apresentados o objetivo geral, os objetivos específicos. Além disso, são destacados os principais desafios e a importância do desenvolvimento de uma inteligência artificial para identificar parâmetros no processo de soldagem GMAW convencional.

A seção de Fundamentação Teórica é destinada a revisão dos principais conceitos e teorias relacionados ao tema do trabalho. Nessa seção, são apresentados os fundamentos da soldagem GMAW convencional, discutindo os principais parâmetros envolvidos no processo. Além disso, são abordados os conceitos de inteligência artificial e redes neurais convolucionais.

As Referências Bibliográficas inclui trabalhos anteriores que serão continuados no presente relatório, assim como a apresentação de arquiteturas e trabalhos relevantes na área de criação de redes neurais convolucionais.

A seção de Metodologia descreve detalhadamente o processo utilizado para desenvolver a inteligência artificial proposta. Nessa seção, são apresentados os passos seguidos para a criação da rede neural convolucional personalizada e também para o treinamento da rede baseada na arquitetura YOLO. São discutidos aspectos como a escolha dos conjuntos de dados, os parâmetros de treinamento, as ferramentas e técnicas utilizadas, visando garantir a replicabilidade e a validade dos resultados obtidos.

A seção de Avaliação de Resultados apresenta os resultados obtidos após a aplicação da metodologia proposta. São analisados e discutidos os desempenhos das duas redes neurais convolucionais desenvolvidas, comparando os resultados obtidos pela rede personalizada e pela YOLO.

Por fim, a seção de Conclusão reúne os principais pontos abordados ao longo do trabalho. Reúne as principais descobertas, destaca as contribuições do estudo e discute suas limitações. Também são apresentadas possíveis direções futuras de pesquisa, abrindo espaço

para o desenvolvimento e aprimoramento da inteligência artificial proposta.

2 Fundamentação teórica

Esse capitulo fornece a base conceitual necessária para a compreensão do contexto e os princípios que envolvem o tema em questão. Será explorado os estudos mais relevantes que fundamentam o projeto, buscando uma compreensão mais profunda dos conceitos essenciais e das abordagens existentes.

2.1 Soldagem GMAW

A soldagem é um processo essencial na indústria moderna, permitindo a união de materiais de forma eficiente e confiável. Dentre os diversos métodos disponíveis, destaca-se a soldagem GMAW (Gas Metal Arc Welding). Para chegar no processo em questão é importante explicar as camadas de classificação dentro dos tipos de processo de soldagem, para facilitar o entendimento do processo. Todas as classificação citadas e as demais existentes podem ser encontradas no seguinte livro (OKUNMURA; TANIGUCHI, 1982).

O processo de soldagem GMAW é realizado por fusão, ou seja, o processo consiste em unir materiais através da fusão do material base e, em alguns casos, de um material de adição. Nesse processo, o calor é controladamente aplicado nas peças a serem unidas, formando uma poça de fusão. Esse método de soldagem permite a realização de junções complexas e a união de diferentes tipos de materiais, incluindo metais ferrosos e não ferrosos, com alta resistência e integridade estrutural.

Na hierarquia de classificação dos processos de soldagem, a soldagem por fusão está no mesmo nível das soldagens por pressão e brasagem. No entanto, a soldagem por fusão oferece a vantagem de possibilitar a realização de junções complexas e a união de diferentes tipos de materiais.

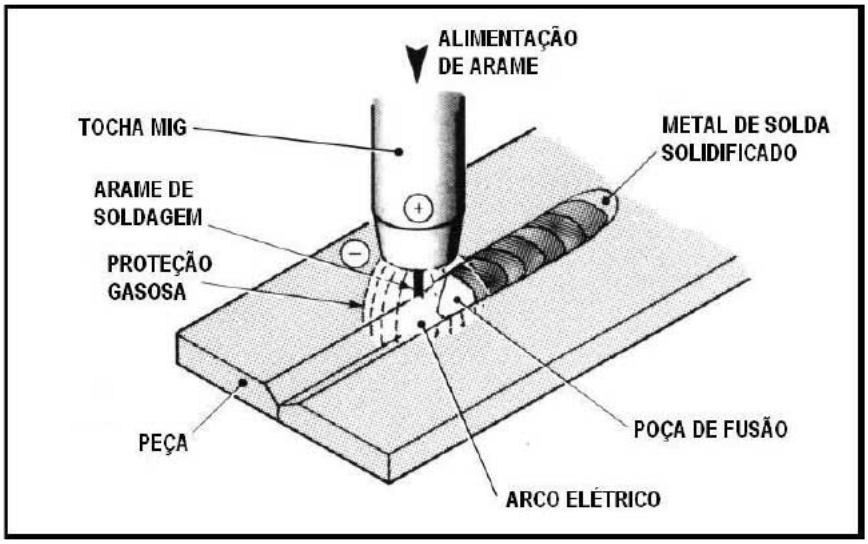
Mais especificamente, a soldagem GMAW é um processo de união por arco elétrico, no qual o calor é gerado por meio de um arco elétrico estabelecido entre um eletrodo metálico e as peças a serem soldadas. A corrente elétrica flui através do eletrodo, criando um arco elétrico concentrado e intenso, que provoca o aquecimento do material a ser soldado. Durante o processo, o metal de adição é adicionado usando um eletrodo revestido ou um arame consumível. O metal de adição é depositado no material base, formando uma poça de fusão. À medida que o arco é movido ao longo da junta de solda, a poça de fusão resulta no metal de solda solidificado.

Dentro dos processos de soldagem por arco elétrico, o estudo em questão concentra-se no processo com eletrodo consumível. Nesse processo, um arame consumível é utilizado como fonte de material de adição e também desempenha o papel de eletrodo. A corrente

elétrica flui pelo eletrodo, criando um arco elétrico entre o eletrodo e a peça a ser soldada.

Por fim, no contexto das soldagens por eletrodo consumível, uma classificação final está relacionada à proteção do processo. É nesse ponto que entra a soldagem GMAW, que utiliza um gás de proteção para garantir a integridade da solda. Esse gás cria uma atmosfera ao redor do arco elétrico e da poça de fusão, protegendo-os da contaminação atmosférica e melhorando a qualidade da solda (NORRISH, 2006). Na figura 2.1 é possível ver o funcionamento do processo e compreender seu funcionamento.

Figura 2.1 – Soldagem GMAW



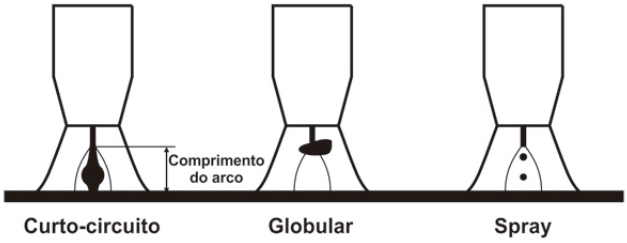
Fonte: (BALMER, 2021)

Dentro do processo de soldagem GMAW existem duas classificações referentes ao tipo de gás usado no processo. Quando os gases são inertes, como hélio e argônio, o processo é denominado como MIG (*metal inert gas*), esse processo é usado na soldagem de materiais não ferrosos, como o alumínio por exemplo. Quando os gases não são inertes, como o dióxido de carbono, o processo é denominado como MAG (*metal active gas*), esse processo é utilizado para materiais ferrosos como o aços de carbono, aços inoxidáveis, ferros e outros metais que contenham ferro (MARQUES; MODENESI; BRACARENSE, 2009).

Dentro dessa soldagem alguns parâmetros devem ser ajustados, como corrente e tensão, variando esses dois parâmetros é possível encontrar 3 tipos de transferências metálicas: Transferência por curto-circuito, Transferência por spray e transferência por gotejamento (figura 2.2). Na figura 2.3 vemos a relação que a tensão e corrente tem sobre esses 3 processos.

- **Transferência por Curto-Circuito**: caracterizado por um baixo nível de energia de soldagem. Nesse processo, a corrente elétrica é interrompida periodicamente, resultando no desprendimento do arame consumível da poça de fusão. Em seguida,

Figura 2.2 – Modos de transferência

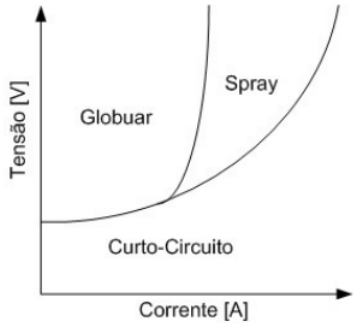


Fonte: (FORTES, 2005)

um novo curto-circuito é estabelecido e o processo se repete. O processo é também caracterizado por uma baixa taxa de deposição de metal de adição e um baixo nível de calor aplicado à peça.

- **Transferência por spray**: Na transferência por spray, a corrente elétrica é alta e o metal de adição é pulverizado em forma de pequenas gotas. Esse modo de transferência é caracterizado por alta taxa de deposição e alta energia térmica. O arco elétrico é mais estável e as gotas de metal de adição são projetadas a partir do arame para a poça de fusão.
- **Transferência por gotejamento**: Na transferência por gotejamento, a corrente elétrica é moderada e o metal de adição é transferido em forma de gotas maiores. Esse modo de transferência ocorre quando a tensão é ajustada para um valor intermediário entre os modos de spray e curto-circuito. O metal de adição goteja continuamente da extremidade do arame para a poça de fusão.

Figura 2.3 – Esquemático aproximado dos modos de transferência metálica



Fonte: Franco (2007)

Em conclusão, os processos de soldagem GMAW desempenham um papel crucial na indústria moderna, oferecendo benefícios amplamente utilizados em várias aplicações industriais. Essa soldagem é indispensável em diversos setores devido à sua capacidade de unir materiais como aços carbono, inoxidáveis, alumínio, cobre e suas ligas.

Sua eficiência e produtividade são destacáveis, permitindo altas taxas de deposição de metal de adição e reduzindo os tempos de produção. Além disso, a qualidade das juntas soldadas é notável, com alta resistência, boa aparência e baixos níveis de defeitos, graças ao controle do arco elétrico e aos parâmetros de soldagem ajustáveis.

A soldagem GMAW também oferece flexibilidade ao permitir soldas em diferentes posições, inclusive em locais de difícil acesso. Sua capacidade de soldar em ângulos e direções variadas contribui para sua versatilidade e adaptação a diversas situações de soldagem.

2.1.1 Transferência por curto-circuito

O processo de soldagem escolhido para os experimentos desse trabalho, foi a transferência por curto-circuito. Nesse processo, o arco de soldagem é estabelecido inicialmente (Figura 2.4, ponto A), com uma tensão alta, e há um equilíbrio entre as taxas de consumo e alimentação do eletrodo. Conforme o tempo passa, o eletrodo se aproxima do metal base devido à taxa de alimentação ser maior do que a taxa de consumo. Quando o eletrodo toca no material base, o arco se extingue, a tensão cai e a corrente aumenta gradualmente, limitada pela indutância do circuito de soldagem (pontos C e D).

Nesse momento, ocorre a formação de material fundido e há uma inversão no equilíbrio entre a taxa de alimentação e consumo (ponto E). Nesse período, as forças magnéticas e a tensão superficial formam um gargalo. Devido à corrente estar elevada, o gargalo é vaporizado, resultando em uma explosão que dissipa energia e restabelece o arco de soldagem, reiniciando o processo (pontos E, F e G).

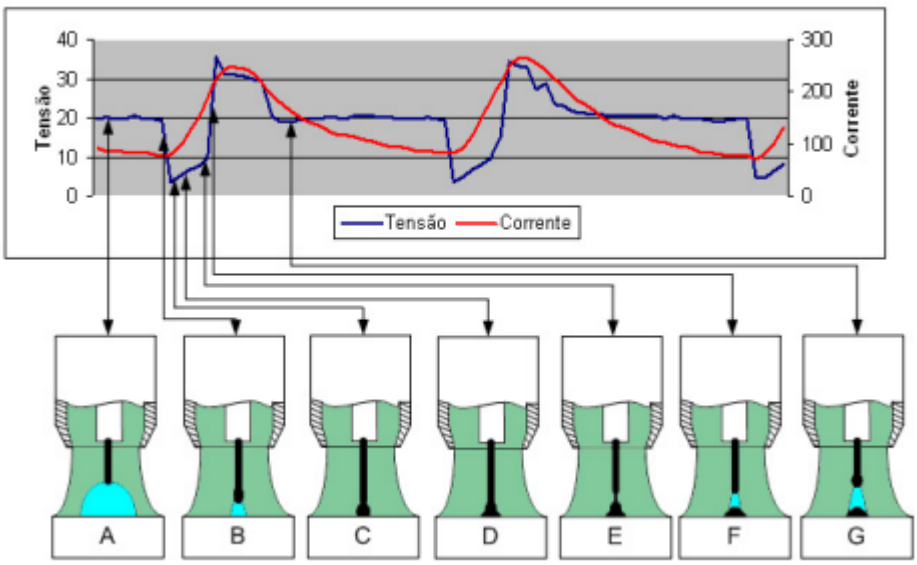
Essa descrição do processo de soldagem por curto-circuito é relevante para compreender o contexto dos experimentos realizados no estudo de Luciano et al. (2007) (FRANCO, 2007), em que foram capturadas imagens durante esse processo.

2.2 Redes neurais artificiais

Uma rede neural possui um poder computacional significativo devido à sua estrutura paralela e distribuída. Isso resulta em um efeito desejado, a generalização, que é altamente valorizado para solucionar diversos tipos de problemas. A generalização significa que a rede neural é capaz de produzir resultados satisfatórios mesmo para dados de entrada que nunca foram vistos antes.

A fim de alcançar resultados satisfatórios e desenvolver projetos bem-sucedidos

Figura 2.4 – Esquemático do modos de transferência por curto-circuito



Fonte: Franco (2007)

utilizando redes neurais, é crucial compreender as capacidades e propriedades dessa arquitetura computacional. A capacidade de processar informações de maneira paralela e distribuída permite a identificação de padrões complexos nos dados e a execução de tarefas como classificação, previsão e outros tipos de processamento de dados.

De acordo com Haykin (2001) (HAYKIN, 2001), as redes neurais possuem diversas propriedades inerentes que são importantes para entender seu funcionamento. Essas propriedades incluem:

1. **Não-linearidade**: As redes neurais são capazes de capturar relações complexas e não-lineares entre os dados de entrada e saída. Isso permite lidar com problemas mais intrincados que não podem ser solucionados por métodos lineares.

2. **Mapeamento entrada e saída**: As redes neurais têm a capacidade de mapear padrões de entrada para saídas desejadas. Isso possibilita realizar tarefas de classificação ou previsão, onde os dados de entrada são transformados em uma resposta ou uma estimativa.

3. **Adaptabilidade**: As redes neurais podem ajustar seus parâmetros internos por meio de treinamento, o que lhes permite aprender com os dados e se adaptar a diferentes tarefas e ambientes. Elas são capazes de se auto-ajustar e melhorar seu desempenho ao longo do tempo.

4. **Resposta a evidências**: As redes neurais podem processar informações de maneira distribuída e integrativa, combinando evidências de várias fontes para tomar decisões ou fazer previsões. Elas são capazes de integrar informações de diferentes partes do sistema para obter um resultado mais preciso.

5. **Informação contextual**: As redes neurais podem incorporar informações contextuais para melhorar a compreensão e o processamento dos dados. Elas podem levar em consideração o contexto em que os dados são apresentados, o que ajuda a obter uma representação mais completa e precisa dos dados.

6. **Tolerância a falhas**: As redes neurais possuem uma certa robustez em relação a falhas ou ruídos nos dados de entrada. Elas podem lidar com informações incompletas ou perturbadas, mantendo um desempenho razoável mesmo nessas condições.

7. **Implementação em VLSI:** As redes neurais podem ser implementadas em chips de VLSI (Very Large-Scale Integration), o que possibilita um processamento rápido e eficiente. Isso é especialmente relevante para aplicações que exigem um tempo de resposta rápido, como visão computacional ou processamento de sinais em tempo real.

8. **Uniformidade de análise de projeto**: As redes neurais apresentam uma metodologia uniforme de projeto e análise. Isso permite que os desenvolvedores apliquem técnicas e princípios consistentes em diferentes problemas, facilitando o desenvolvimento e a compreensão desses sistemas.

9. **Analogia neurobiológica**: As redes neurais artificiais são inspiradas em certos aspectos do funcionamento do cérebro humano. Elas buscam replicar certas propriedades e processos observados nos sistemas biológicos, o que as torna uma abordagem interessante para o estudo e a modelagem de fenômenos complexos.

Essas propriedades tornam as redes neurais uma ferramenta poderosa para resolver problemas complexos e lidar com dados não-lineares, oferecendo flexibilidade, adaptabilidade e capacidade de aprendizado.

A seguinte definição de uma rede neural artificial (RNA) foi proposta por Hecht-Nielsen em 1990:

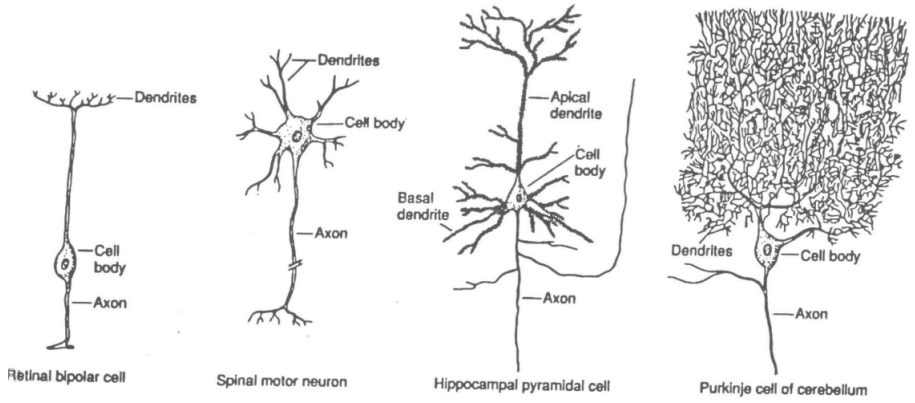
> Uma Rede neural artificial é uma estrutura que processa informações de forma paralela e distribuída e que consiste de unidades computacionais (as quais podem possuir uma memória local e podem executar operações locais) interconectadas por canais unidirecionais chamados de conexões. Cada unidade computacional possui uma única conexão de saída, que pode ser dividida em quantas conexões laterais se fizer necessário, sendo que cada uma dessas conexões transporta o mesmo sinal, o sinal de saída da unidade

> computacional. Esse sinal de saída pode ser contínuo ou discreto. O processamento executado por cada unidade computacional pode ser definido arbitrariamente, com a restrição de que ele pode ser completamente local, isto é, deve-se depender somente dos valores atuais dos sinais de entrada chegaram até a unidade computacional via as conexões de valores armazenados na memória local da unidade computacional. (HECHT-NIELSEN, 1990).

2.2.1 Neurônio Biológico

As redes neurais artificiais são modelos computacionais inspirados no sistema nervoso dos seres vivos e tentam replicar seu funcionamento. O corpo humano por exemplo possui vários tipos de neurônios (figura 2.5), estima-se que o cérebro humano tenha por volta de 10^{11} neurônios, cujo comprimento total somado chega a 10^{14} metros. Cada um desses neurônios está conectado a cerca de 10^3 a 10^4 outros neurônios, ou seja, o cerébro humano teria entre 10^{14} a 10^{15} conexões (NASCIMENTO JR.; YONEYAMA, 2000)

Figura 2.5 – Tipos de Neurônios



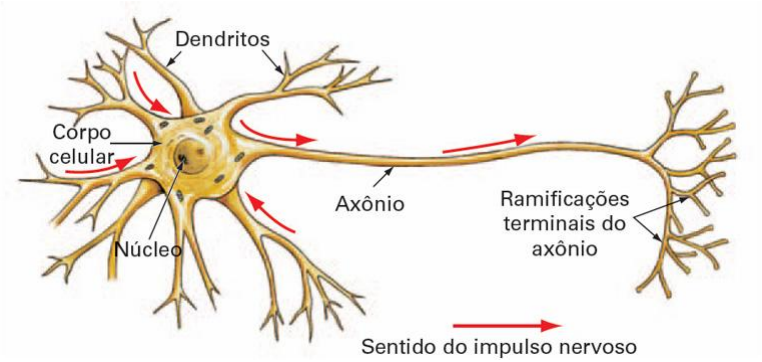
Fonte: Bauchspiess (2008)

O neurônio biológico, figura 2.2.1, é a unidade básica do sistema nervoso e é composto por três partes principais: o corpo celular, dendritos e axônio. O corpo celular contém o núcleo contém informações sobre características de hereditariedade. Os dendritos são as ramificações que recebem sinais de entrada de outros neurônios e os repassam para o corpo celular. O axônio é uma única fibra longa que transmite sinais de saída para outros neurônios através das sinapses.

Quando um neurônio recebe sinais suficientes de seus dendritos, ele dispara um impulso elétrico chamado de **potencial de ação** que se propaga ao longo do axônio. Esse impulso elétrico é transmitido para outros neurônios ou para células musculares ou glandulares por meio da **sinapse**. Dependendo do tipo do neurotransmissor predominante na sinapse, o potencial da membrana do dendrito é aumentado ou diminuído. Esses sinais recebidos pelos dendritos de vários neurônios são propagados até o corpo do neurônio onde

são aproximadamente somados. Se a soma dentro de um pequeno intervalo de tempo for acima de um determinado limite, o corpo do neurônio gera um potencial de ação que é então transmitido pelo axônio aos outros neurônios. Permitindo que o sistema nervoso controle o comportamento e a fisiologia do corpo (KANDEL; SCHWARTZ; JESSELL, 2000).

Figura 2.6 – Neurônio biológico



Fonte: Khan Academy (2015)

2.2.2 Neurônio artificial

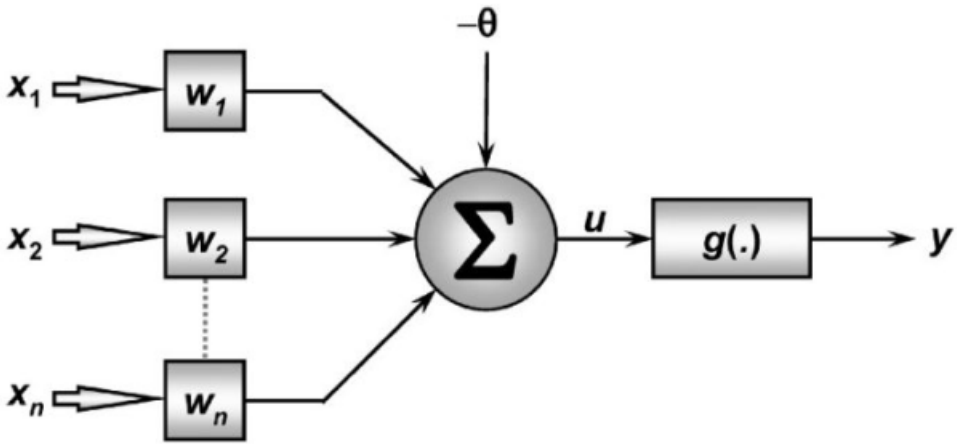
Um neurônio artificial é a unidade básica de processamento em uma rede neural artificial. É um modelo matemático que se inspira na estrutura e funcionamento dos neurônios biológicos do cérebro humano. Em (MCCULLOCH; PITTS, 1943), foi proposto um modelo que contém as características principais de um neurônio, como paralelismo e alta conectividade. Esse modelo ficou muito conhecido, sendo ainda o modelo mais utilizado nas arquiteturas de redes neurais artificiais até hoje. No livro (SILVA; SPATTI, 2016), podemos ver a representação da figura 2.2.2, onde esse modelo, tão usado, é apresentado e explicado detalhadamente.

Os diversos sinais de entrada advindos do meio externo são espelhados pelos conjuntos $\{x_1, x_2, ...,x_n\}$, que são análogos aos impulsos elétricos externos captados pelos dendritos no neurônio biológico. As ponderações exercidas pelo neurônio em relação a propagação do potencial de ação são representadas pelos pesos sinápticos $\{w_1, w_2, ...,w_n\}$. A relevância de cada uma das entradas $\{x_i\}$ do neurônio é verificada por meio da multiplicação dos respectivos pesos sinápticos $\{w_i\}$, ponderando-se todas as informações externas que chegaram ao neurônio. Assim, torna-se possível verificar que a saída do corpo celular artificial, é a soma ponderada de suas entradas.

É apresentado então o neurônio artificial que contém 7 elementos básicos, que são:

- Sinais de entrada $\{x_1, x_2, ...,x_n\}$ São sinais externos, usualmente normalizados para incrementar a eficiência computacional dos algoritmos de aprendizagem. De forma

Figura 2.7 – Neurônio Artificial



Fonte: Silva e Spatti (2016)

simples, são dados que alimentam o modelo preditivo.

- Pesos sinápticos $\{w_1, w_2, ..., w_n\}$ São os valores que servirão de ponderação para as variáveis de entrada. Esses valores são treinados durante o processo.
- Combinador linear $\{\sigma\}$ Tem como função somar todos os sinais de entrada que foram ponderados anteriormente.
- Limiar de ativação $\{\theta\}$ Especifica qual será o patamar apropriado para que o resultado produzido pelo combinador linear possa gerar um valor de disparo de ativação.
- Potencial de ativação $\{u\}$ É o resultado obtido pela diferença do valor produzido entre o combinador linear e o limiar de ativação. Se o valor for positivo, ou seja, se $u \geq 0$ então o neurônio produz um potencial excitatório, caso contrário, o potencial será inibitório.
- Função de ativação $\{g\}$ Seu objetivo é limitar a saída de um neurônio em um intervalo valores razoáveis a serem assumidos pela sua própria imagem funcional.
- Sinal de saída $\{y\}$ É o valor final de saída podendo ser usado como entrada de outros neurônios que estão sequencialmente interligados.

As expressões seguintes resumem os resultados produzidos pela síntese das informações apresentadas

$$u = \sum_{i=1}^{n} w_i \cdot x_i - \theta \tag{2.1}$$

$$y = g(u) \tag{2.2}$$

A analogia grosseira entre neurônio artificial e neurônio biológico é que as conexões entre os nós representam os axônios e dendritos, os pesos da conexão representam as sinapses e o limiar se aproxima da atividade no soma (JAIN; MAO; MOHIUDDIN, 1996).

2.2.2.1 Funções de ativação

A função de ativação tem papel importante nas redes neurais artificiais, pois introduzem a não linearidade nas saídas dos neurônios. Basicamente a função de ativação tem o poder de decisão entre o neurônio ser ativado ou não. Elas desempenham um papel crucial na modelagem das relações complexas e na capacidade de aprendizado das redes neurais. Existem várias funções de ativação comumente utilizadas, cada uma com suas características e propriedades. Serão apresentadas as funções de ativação que já pertencem a biblioteca do *Keras* (KERAS, 2023), pois essas serão usadas e testadas no projeto.

a) **Função de ativação Linear**

A função de ativação linear, também conhecida como identidade, retorna a entrada sem qualquer alteração. Ela simplesmente propaga o valor ponderado da entrada para a saída do neurônio. Essa função não introduz não linearidade, o que limita sua capacidade de representação de padrões complexos. No entanto, pode ser útil em casos específicos, como em camadas de saída de regressão linear. Em termos matemáticos a função linear pode ser descrita da seguinte forma:

$$g(u) = u \tag{2.3}$$

b) **Função de ativação Sigmóide**:

A funções de ativação totalmente diferenciáveis são aquelas cujas derivadas de primeira ordem existem e são conhecidas em todos os pontos de domínio de definição. A função de ativação sigmóide mapeia a soma ponderada das entradas para um valor entre 0 e 1. Ela é uma função suave, diferenciável, não linear e pode ser descrita matematicamente por:

$$g(u) = \frac{1}{1 + e^{-\beta \cdot u}} \tag{2.4}$$

O formato geométrico da função de ativação logística depende do parâmetro β, caso esse parâmetro seja muito elevado, o formato da função tenderá a ser similar a uma função degrau.

c) **Função de ativação tangente hiperbólica**

A função de ativação tangente hiperbólica se assemelha a função sigmóide, mas tem como diferença o fato de assumir valores entre -1 e 1. Pode ser definida matematicamente por:

$$g(u) = \frac{1 - e^{-\beta \cdot u}}{1 + e^{-\beta \cdot u}} \tag{2.5}$$

d) **Função de ativação ReLU**

A função de ativação ReLU retorna a entrada diretamente se ela for maior que zero, caso contrário, retorna zero. Essa função é simples, computacionalmente eficiente e tem propriedades desejáveis, como não sofrer desvanecimento do gradiente. Pode ser definida matematicamente por:

$$g(u) = max(0,u) \tag{2.6}$$

e) **Função de ativação Softmax**

A função de ativação Softmax converte um vetor de valores em uma distribuição de probabilidade. Ela mapeia as saídas para uma distribuição de probabilidade, onde a soma de todas as saídas é igual a 1. Diferente da função sigmóide que lida apenas com a classificação entre das classes, a função Softmax permite que a rede neural atribua probabilidades a diferentes classes, facilitando a tomada de decisões sobre a classe mais provável. Pode ser definida matematicamente por:

$$g(u)_i = \frac{e^{u_i}}{\sum_{k=1}^{K} e^{u_k}} \quad \text{para i} = 1,...,\text{K} \tag{2.7}$$

Onde K é o número de saídas possíveis.

f) **Função de ativação SoftPlus**

A função SoftPlus é uma função suave e diferenciável que é uma aproximação suave da função ReLU. Ela retorna valores positivos para todos os valores de entrada. É útil quando se deseja uma função de ativação suave que preserve a diferenciabilidade.

$$g(u) = log(e^u + 1) \tag{2.8}$$

g) **Função de ativação Softsign**

A função softsign é outra função suave e diferenciável que mapeia os valores de entrada para o intervalo [-1, 1]. É útil quando os valores de entrada variam em uma ampla faixa.

$$g(u) = \frac{u}{(1 + |u|)} \tag{2.9}$$

h) **Função de ativação ELU**

A função ELU (*Exponential Linear Unit*) é outra função suave e diferenciável que mapeia os valores de entrada para o intervalo [-1, 1]. É útil quando os valores de entrada variam em uma ampla faixa.

$$g(u) = \begin{cases} \alpha \cdot (e^u - 1) & \text{se } u < 0, \\ u & \text{se } u \geq 0 \end{cases} \tag{2.10}$$

O parâmetro α é constante e ajustável.

i) **Função de ativação SELU**

A função ELU (*Scaled Exponential Linear Unit*) é uma variação da função ELU que tem uma propriedade especial de autonormalização. Ela mantém a média e o desvio padrão da ativação aproximadamente constantes ao longo das camadas, o que pode ajudar no treinamento de redes neurais profundas.

$$g(u) = \begin{cases} \lambda \cdot \alpha \cdot (e^u - 1) & \text{se } u < 0, \\ \lambda \cdot u & \text{se } u \geq 0 \end{cases} \tag{2.11}$$

Os parâmetros α e λ são constantes ajustáveis.

2.2.2.2 Escolhendo a função de ativação

A escolha da função de ativação depende do do problema que se deseja resolver. Nessa escolha estão envolvidos os tipos de saídas da rede, a base de dados de treinamento, o tipo de rede escolhida e por último o desempenho do treinamento. Não existe uma regra sólida para essa escolha mas as seguintes indicações tendem a convergir mais fácil e rapidamente para o resultado:

- **Problemas de classificação binária**

  Para problemas de classificação binária, a função de ativação sigmoid é uma escolha comum para a camada de saída. Ela mapeia a saída para um valor entre 0 e 1, que pode ser interpretado como uma probabilidade.

- **Problemas de classificação Multiclasse**

  Para problemas de classificação multiclasse, a função de ativação softmax é comumente usada na camada de saída. Ela gera uma distribuição de probabilidade em que a soma

das saídas é igual a 1, permitindo a interpretação das saídas como probabilidades de pertencer a cada classe.

- **Problemas de regressão**

  Para problemas de regressão, ou seja, onde é preciso prever um valor contínuo, a função de ativação linear na camada de saída pode ser apropriada. Ela permite que a rede produza valores contínuos sem restrições. Em suas camadas ocultas funções como tangente hiperbólica e ReLU podem ser usadas.

- **Redes neurais profundas**

  Para redes neurais profundas com muitas camadas ocultas, a função de ativação ReLU geralmente funciona bem. Ela ajuda a mitigar o problema de desvanecimento do gradiente, permitindo um treinamento mais estável e rápido. Nos problemas apresentados ela pode ser facilmente aplicada nas camadas ocultas das funções.

Por mais que algumas funções de ativação são destinadas a problemas específicos, as vezes, vários tipos de funções podem ser usadas e cada uma produzindo uma resultado diferente. Saber qual função de ativação utilizar em cada problema é uma pergunta que é respondida apenas após diversos testes de treinamento. Não existe uma função de ativação universalmente melhor, as vezes cada camada da rede neural pode responder melhor com uma função de ativação específica. A escolha da função de ativação é uma discussão atual, no artigo (ERTUGRUL, 2018), por exemplo, é defendido um novo tipo de funções de ativação, onde elas são treinadas para cada neurônio da rede.

2.2.3 Arquitetura de redes neurais

Quando se trata de arquiteturas de redes neurais, há uma infinidade delas, e novas arquiteturas e variações estão surgindo a cada dia. No entanto, a maioria das arquiteturas utilizadas e comuns pode ser dividida em três partes distintas, conhecidas como camadas.

A primeira parte da rede é a camada de entrada, que é responsável por receber os dados, sinais, medições e informações provenientes do ambiente externo. Frequentemente, esses dados passam por um processo de normalização, a fim de obter uma melhor precisão numérica quando submetidos às operações numéricas intrínsecas da rede. A camada de entrada desempenha um papel crucial na alimentação dos dados para o restante da rede.

A segunda parte da rede é composta pelas camadas ocultas, que podem ser compostas por um ou vários neurônios. Essas camadas são responsáveis por extrair as características relevantes do problema que está sendo resolvido. Nessa etapa, os neurônios realizam operações matemáticas e transformações nos dados de entrada, permitindo que a rede aprenda padrões complexos e realize cálculos intermediários necessários para a tomada de decisões. As discussões sobre modelos e topologias de redes neurais frequentemente se concentram

nas camadas ocultas, pois é nessa parte da rede que ocorre a maior parte do processamento e da extração de informações. A escolha adequada do número de camadas, o tipo de neurônios e as conexões entre eles são fatores cruciais para o desempenho e a eficácia da rede neural.

Por fim, a terceira parte da rede é a camada de saída. Essa camada é composta por neurônios responsáveis por apresentar os resultados finais da rede neural. Dependendo da tarefa em questão, como classificação, regressão ou geração de texto, a camada de saída pode ter diferentes configurações e funções de ativação. É nessa etapa que os resultados finais da rede são disponibilizados para análise ou tomada de decisão.

Na figura A.51, encontrada no anexo A desse texto, são apresentadas boa parte das arquiteturas de redes neurais mais conhecidas junto de uma legenda que facilita o entendimento dos princípios de funcionamento dessas redes. As definições das arquiteturas apresentadas podem ser vistas em (THE ASIMOV INSTITUTE, 2016).

2.2.4 Aprendizado das redes neurais

Os processos de treinamento de redes neurais são fundamentais para o desenvolvimento e aprimoramento desses poderosos modelos de aprendizado de máquina, e significam a aplicação de passos ordenados com o objetivo de treinar uma rede que seja capaz generalizar soluções. Existem 3 paradigmas principais de aprendizagem (SILVA; SPATTI, 2016), esses são conhecidos como Treinamento supervisionado, treinamento não-supervisionado e aprendizagem por reforço.

- **Treinamento supervisionado**:

  O treinamento supervisionado é uma abordagem na qual uma rede neural é treinada usando exemplos de entrada e saída desejada. O processo é chamado de supervisionado porque durante o treinamento é como se houvesse um professor capaz de fornecer à rede neural uma resposta desejada para cada dado de treinamento. Um sinal de erro é produzido, e é definido como a diferença da resposta desejada para a resposta atual da rede. Os parâmetros são ajustados interativamente a partir do sinal de erro e dos dados de treinamento, com o objetivo de passar as informações disponíveis da forma mais completa possível. A rede está treinada quando o sinal de erro está dentro de valores aceitáveis dado o contexto da aplicação.

  O treinamento supervisionado é amplamente utilizado em tarefas de classificação e regressão, onde os dados de treinamento têm rótulos conhecidos. Exemplos incluem reconhecimento de imagem, tradução automática e diagnóstico médico. A primeira estratégia de treinamento supervisionado foi proposta em 1949 em (HEBB, 1949b).

- **Treinamento não-supervisionado**:

No treinamento não-supervisionado, a rede neural é treinada sem a supervisão explícita de pares de entrada e saída. Em vez disso, a rede busca aprender a estrutura e os padrões intrínsecos dos dados de entrada e se auto-organizar.

Existem algumas abordagens para esse tipo de treinamento, como o agrupamento, onde os dados são agrupados em *clusters* com base em sua similaridade. Essa e outras abordagens como a redução da dimensionalidade e *autoencoders* podem ser vistas em (MÜLLER; GUIDO, 2016)

- **Aprendizado por reforço**:

  O aprendizado por reforço é uma abordagem na qual uma rede neural aprende através da interação com um ambiente dinâmico. Nesse processo, a rede oma ações em um ambiente para maximizar uma recompensa numérica. O processo de treinamento e tipicamente por tentativa e erro, onde a única resposta que a rede recebe é se a saída produzida é satisfatória ou não. Esse aprendizado é comumente utilizado em jogos, robótica e otimização de sistemas. Exemplos incluem jogar xadrez, dirigir carros autônomos e controle de processos industriais. (SUTTON; BARTO, 2018).

2.2.5 Perceptron

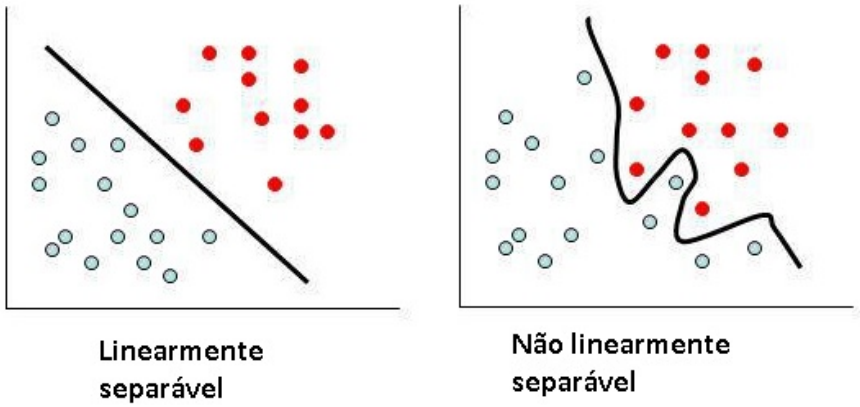
Na seção 2.2.2 foi apresentado um modelo de neurônio artificial genérico que tentava se aproximar do neurônio biológico. Esse modelo é útil para apresentar muitos modelos de redes neurais e serviu de inspiração para o modelo conhecido como o Perceptron. Idealizado por (ROSENBLATT, 1958), é a forma mais básica de configuração de uma rede neural, usada para classificação de padrões linearmente separáveis, ou seja, padrões que se encontram em lados opostos de um hiperplano, a figura 2.8 apresenta a diferença a diferença de um problema resolvível por uma rede perceptron e um problema que a rede não atende.

O perceptron consiste em um neurônio similar ao apresentado na figura 2.7, que consiste em um único neurônio com pesos sinápticos ajustáveis e bias. Os vetores de padrões usados para treinar o perceptron são retirados de duas classes linearmente separáveis, então o algoritmo do perceptron converge e posiciona a superfície de decisão entre as duas classes. O teorema da convergência proposto por Rosenblatt pode ser visto em (HAYKIN, 2001).

2.2.5.1 Funcionamento do Perceptron

A rede *perceptron* como visto em seções anteriores, pertence a arquitetura *feedforward* de 1 camada, uma vez que o fluxo de informações em sua estrutura segue sempre em direção a camada de saída, sem qualquer realimentação. Seu funcionamento se dá pela ponderação de todas as entradas $\{x_1, x_2, ..., x_n\}$ pelos respectivos pesos sinápticos $\{w_1, w_2, ..., w_n\}$. Após isso o valor resultante da composição de todas as entradas ponderadas, com o limiar de ativação $\{\theta\}$, é passado como argumento para a função de ativação que produz a saída $\{y\}$. Obtemos as

Figura 2.8 – Linearidade de dados



Fonte: Rocha (2020)

mesmas equações 2.1 e 2.2, onde a saída deverá alternar entre dois valores (0 ou 1). Por conta desse padrão de saída, a função de ativação é tipicamente do tipo degrau ou degrau bipolar

A tabela 2.1 resume aspectos dos parâmetros do perceptron

Tabela 2.1 – Aspectos característicos do *Perceptron*

Parâmetro	Variável	Tipo
Entrada	x_i	Reais ou binárias
Saídas	y	binária
Pesos sinápticos	w_i	Real (iniciado aleatoriamente)
Limiar	θ	Real (iniciado aleatoriamente)
Função de ativação	$g()$	Tipicamente: degrau e degrau bipolar
Processo de treinamento	-	Supervisionado
Regra de aprendizagem	-	Regra de Hebb

Nota: A variável i se refere a i-ésima entrada da rede

2.2.5.2 Processo de treinamento do Perceptron

O ajuste de pesos e o limiar do Perceptron é efetuado utilizando o treinamento supervisionado, ou seja, para cada amostra dos sinais de entradas espera-se uma respectiva saída. Como trata-se de uma rede que possui apenas duas saídas, então cada entrada é associada a uma classe e o treinamento é realizado pela regra Hebb. O postulado de aprendizado de Hebb é a primeira e mais famosa regra de aprendizagem e pode ser vista em (HEBB, 1949a).

De forma simples o treinamento acontece com base em duas situações, a primeira se dá quando a saída produzida coincide com a saída desejada, dessa forma os pesos sinápticos e o limiar permanecerão sem modificações. Caso o contrário aconteça, ou seja, a saída produzida não coincida com a saída desejada os pesos e limiares da rede devem ser incrementados

nas devidas proporções em relação aos valores de seus sinais de entrada. Esse processo é repetido sequencialmente em todas as amostras de treinamento, até que não exista mais atualizações de pesos, ou de outra forma, até que a saída desejada coincida com a saída da rede.

Em termos matemáticos a atualização dos pesos e limiares pode ser escrita da seguinte forma:

$$w_i^{atual} = w_i^{anterior} + \eta \cdot (d^{(k)} - y) \cdot x_i^{(k)} \tag{2.12}$$

$$\theta_i^{atual} = \theta_i^{anterior} + \eta \cdot (d^{(k)} - y) \cdot -1 \tag{2.13}$$

Onde:

- k representa a k-ésima amostra do treinamento
- $d^{(k)}$ é o valor desejado para a k-ésima amostra do treinamento
- η é uma constante que define a taxa de aprendizagem da rede.

Em relação ao valor de $\{\eta\}$, ele se refere a velocidade com que o processo de treinamento da rede será conduzido a um valor estável. Entretanto com a finalidade de evitar instabilidades no processo de treinamento, adota-se um valor compreendido em um intervalo de $0 < \eta > 1$. Dessa forma quando a faixa de separabilidade entre as duas classes do problema for muito estreita, o processo pode ser instável, por isso uma maneira de contornar esse problema determinar uma taxa de aprendizagem pequena, assim a convergência pode ser mais estável.

2.2.6 Projetos com Redes neurais Artificiais

Embora as RNAs sejam abstrações drásticas das contrapartes biológicas, a ideia das RNAs não é replicar a operação dos sistemas biológicos, mas fazer uso do que se sabe sobre a funcionalidade das redes biológicas para resolver problemas complexos. A atratividade das RNAs vem das notáveis características de processamento de informações do sistema biológico, como não linearidade, alto paralelismo, robustez, tolerância a falhas e falhas, aprendizado, capacidade de lidar com informações imprecisas e difusas e sua capacidade de generalizar (HAYKIN, 2001).

Para o desenvolvimento de projetos que envolvam RNAs é preciso conhecer as tarefas básicas executadas por uma RNA. De acordo com (MEDEIROS, 2018), são cinco, as tarefas que são as bases de todos os funcionamentos e aplicações, sendo elas:

1. **Associação de padrões** - As RNAs são frequentemente usadas para classificação, onde a tarefa é atribuir uma ou mais categorias a um determinado conjunto de dados. Exemplos incluem classificação de imagens, detecção de fraudes, diagnóstico médico, análise de sentimentos e muito mais. De forma geral, quando determinado conjunto de dados de ser associado a outro conjunto.

2. **Reconhecimento de padrões** - As RNAs são eficazes para tarefas de reconhecimento de padrões, como reconhecimento de fala, reconhecimento de escrita à mão, reconhecimento de objetos em imagens, reconhecimento de rosto, entre outros.

3. **Regressão** - As RNAs podem ser aplicadas em tarefas de regressão, onde o objetivo é prever um valor contínuo em vez de uma classe ou categoria. Exemplos incluem previsão de preços de imóveis, previsão de vendas, previsão de demanda, entre outros.

4. **Controle** - As redes neurais podem executar funções de controle de sistemas de forma automatizada, monitorando o *feedback* das saídas. Exemplos incluem sistemas de frenagem, controle de voo, entre outros.

5. **Filtragem** - As redes neurais podem ser usadas para extração de ruídos, isso inclui a identificação do ruído. Exemplos incluem filtragem adaptativa de sons,

Agora pensando em um projeto que é possível resolver ou otimizar com redes neurais artificias, o artigo (BASHEER; HAJMEER, 2000) define um planejamento de seis passos para qualquer projeto de RNA ser bem sucedido. Consistem nos seguintes passos:

1. O **passo 1** consiste na definição e formulação do problema, é crucial para estabelecer uma compreensão adequada do desafio em questão. É fundamental identificar corretamente as relações de causa e efeito envolvidas no problema. Além disso, nessa fase, é importante avaliar os benefícios das redes neurais em comparação com outras técnicas disponíveis antes de selecionar a abordagem de modelagem final.

2. O **passo 2** se resume ao início da rede, consiste na coleta de dados, pré-processamento dos dados para adequá-los ao tipo de rede neural escolhido, a escolha do tipo de rede neural a ser utilizada, a decisão sobre a regra de aprendizado e o particionamento dos dados em três subconjuntos distintos (subconjuntos de treinamento, teste e validação).

3. O **passo 3**, envolve o treinamento da rede utilizando os subconjuntos de dados, juntamento com a avaliação e observação dos erros de previsão. Esse passo tem como princípio fundamental a seleção dos hiperparâmetros [1] que são envolvidos no treinamento, que afetam os resultados e o desempenho da rede final, por exemplo, tamanho da rede, taxa de aprendizado, número de ciclos de treinamento, erro aceitável, etc.

[1] Em machine learning, um hiperparâmetro é um parâmetro cujo valor é usado para controlar o processo de aprendizado e é definido antes do treinamento da rede, geralmente pelo projetista da rede

4. O **passo 4** é importante para confirmar a capacidade de generalização do modelo baseado em RNA. Embora os testes contra os dados de teste tenham sido realizados durante o treinamento da rede, é recomendável avaliar a capacidade da "melhor"rede em lidar com exemplos não utilizados durante o desenvolvimento da rede, usando o subconjunto de validação.

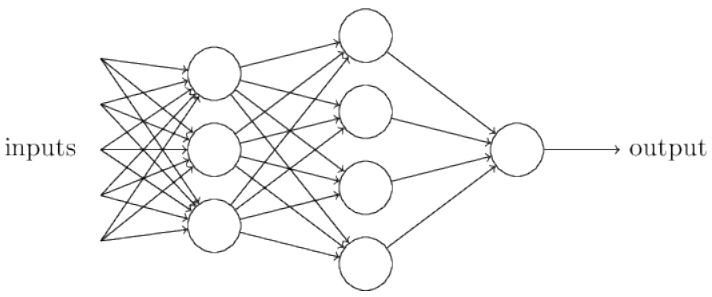
5. O **passo 5** é a fase de implementação do sistema, que envolve a incorporação da rede neural obtida em um sistema de trabalho apropriado, seja como controlador de hardware ou como parte de um programa de computador. É essencial realizar testes abrangentes no sistema integrado antes de disponibilizá-lo para o usuário final, garantindo assim sua funcionalidade e eficácia.

6. O **passo 6** é a fase de manutenção do sistema. Lida com a atualização contínua do sistema desenvolvido à medida que ocorrem mudanças no ambiente ou nas variáveis que afetam o funcionamento da rede neural.

2.3 Perceptron Multicamadas

As rede *Perceptron* de múltiplas camadas (PMC) ou também conhecidas como *multi-layers perceptron* (MLP) são identificadas pela presença de no mínimo uma camada oculta de neurônios entre as camadas de entrada e saída da rede neural. Essa camada oculta é também conhecida como camada escondida ou intermediária, cada neurônio em uma camada oculta recebe entradas ponderadas das saídas da camada anterior e aplica uma função de ativação para produzir uma saída. Um exemplo simples de uma rede PMC pode ser vista na figura 2.9 onde a rede possui 5 entradas, duas camadas ocultas de neurônios, sendo a primeira com 3 e a segunda com 4 neurônios e uma saída. É importante mencionar que essa rede é totalmente conectada, isso significa que que um neurônio em qualquer camada se conecta a todos os outros da camada posterior.

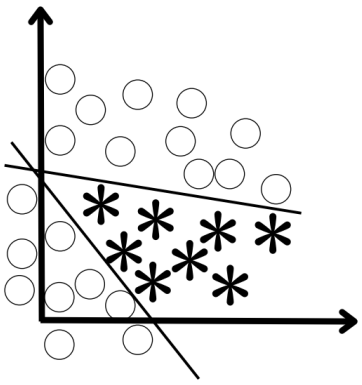
Figura 2.9 – Perceptron de múltiplas camadas



Fonte: Data Science Academy (2022)

As redes PMC são muito versáteis e poderosas, por consequência ela possui inúmeras aplicações e áreas que podem ser usadas e com sucesso. Algumas dessas áreas são a classificação de padrões, problemas de regressão de valores contínuos, reconhecimento de padrões, análise de dados e tantos outros problemas. Fazendo um paralelo com a rede Perceptron simples apresentada na seção 2.2.5, as redes PMC são capazes de resolver problemas não-lineares. A exemplo disso imagine um problema de classificação de duas classes que não é linearmente separável, ou seja, o perceptron de camada única não conseguiria convergir para posicionar um hiperplano capaz de separar as classes. Agora pensando na solução mais simples possível para esse problema, imagine uma rede Perceptron com duas camadas, sendo uma delas a camada oculta e a outra sendo a camada de saída. Essa abordagem é capaz de resolver esse problema de classificação onde os elementos estejam dentro de uma região convexa, como na figura 2.10.

(a) Região 1



(b) Região 2

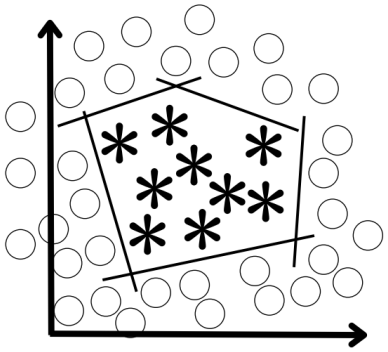


Figura 2.10 – Regiões convexas para classificação de padrões

Fonte: Produzidas pelo autor

A figura 2.10a precisaria de dois neurônios na camada oculta para realizar essa classificação de amostras, de maneira similar a figura 2.10b precisaria de 5 neurônios na camada oculta. Repare que foi dito que essa abordagem, onde a rede possui duas camadas, funciona para elementos dentro de uma região convexa [2]. O artigo (LIPPMANN, 1987) mostra que redes de 3 camadas, sendo duas delas camadas ocultas, conseguem classificar padrões em qualquer tipo de região, inclusive regiões não-convexas.

2.3.1 Funcionamento do *Perceptron* Multicamadas

Diferente do *Perceptron* simples apresentado na seção 2.2.5, a redes PMCs além de possuírem camadas ocultas, também podem produzir mais de uma saída, ou seja, várias saídas podem ser mapeadas com as PMCs. A arquitetura usada nessas redes é a *feedforward* de

[2] Uma região convexa é um conjunto no espaço (ou em um plano) onde, para quaisquer dois pontos dentro dessa região, o segmento de linha que os conecta também está completamente contido dentro dessa região.

múltiplas camadas conforme visto na seção **??**, além disso seu treinamento é supervisionado, conforme visto na seção 2.2.4. A propagação de sinais em redes desse tipo são realizadas em sentido a camada de saída, ou seja, os sinais entram nas camadas de entrada e são propagados para a camada de saída.

O principal fator de seu funcionamento está contido em duas características dessa rede, que são a presença de camadas ocultas e do algoritmo de treinamento dessa rede. As camadas ocultas são responsáveis por realizar um processamento interno dos dados, combinando as informações das camadas anteriores de forma ponderada. Cada neurônio em uma camada oculta recebe as saídas da camada anterior, realiza uma combinação linear ponderada dessas entradas e aplica uma função de ativação não linear para produzir uma saída.

Essa etapa de processamento não linear nas camadas ocultas permite que a rede aprenda a representar características ou padrões relevantes nos dados de entrada que são necessários para a tarefa em questão. A capacidade de aprender representações mais complexas é uma das principais vantagens das redes perceptron multicamadas em relação a modelos mais simples, como os *perceptrons* de camada única.

Ao adicionar mais camadas ocultas, a rede pode realizar transformações mais profundas e abstratas dos dados, permitindo uma representação hierárquica das informações. Cada camada oculta pode aprender a detectar características mais específicas e abstratas em relação à camada anterior, levando a uma progressiva extração de características mais significativas à medida que a informação é propagada pela rede.

2.3.2 Processo de treinamento do *Perceptron* Multicamadas

As *Perceptrons* multicamadas são usados para resolver diversos problemas complicados, isso porque seu treinamento supervisionado funciona com um algoritmo conhecido como algoritmo de retropropagação de erro, ou também conhecido como algoritmo *back-propagation*. Esse algoritmo funciona como uma generalização de um algoritmo conhecido como mínimo quadrado médio (LMS) ou regra delta ou método do gradiente descendente, desenvolvido por (WIDROW; HOFF, 1960). Basicamente o método consiste em encontrar, de forma iterativa, os valores dos parâmetros que minimizam determinada função de interesse.

Basicamente a rede funciona em uma sequência de passos que realizam a atualização dos pesos da rede. Trata-se de um processo bem complexo e existem diversas mudanças e variações dentro da forma que funciona esse algoritmo. A seguinte sequência de passos foi um resumo de dois ótimos livros, (HAYKIN, 2001) e (SILVA; SPATTI, 2016), nessas referências é possível ver com detalhes e em termos matemáticos os seguintes passos:

1. **Inicialização**:

Os pesos das conexões entre os neurônios são inicializados. Os pesos podem ser iniciados aleatoriamente, normalizados, constantes, com zeros, com uns e etc. Existem diversas formas de inicialização de pesos, e a escolha da forma mais adequada pode melhorar do desempenho da rede neural.

O Artigo (BROWNLEE, 2021) mostra como mudanças na inicialização impacta nos resultados e apresenta duas formas de se inicializar pesos de forma a melhorar o desempenho da rede. A primeira forma de se inicializar os pesos da rede é por meio da inicialização "xavier"ou "glorot"(GLOROT; BENGIO, 2010), indicada para rede que utilizam funções de ativação sigmóide e tangente hipoerbólica. A outra forma é a inicialização "he"(HE et al., 2015), indicada para redes que utilizam funções de ativação do tipo ReLU.

2. **Propagação para frente**:

   Também conhecida como *forward propagation*. Um exemplo de treinamento é apresentado à rede, e a propagação direta é realizada. Os dados de entrada são propagados através das camadas até que a saída final seja gerada. Cada neurônio em uma camada oculta recebe as saídas ponderadas da camada anterior, aplica uma função de ativação e produz uma saída. Aqui entram as funções de ativação vistas na seção 2.2.2.1.

3. **Cálculo do erro**:

   O erro é quantificado através da função de perda, comparando a saída prevista pela rede com a saída desejada. A função de perda é usada para medir o quão bem o modelo está realizando a tarefa durante o treinamento. Ela mede a diferença entre as saídas previstas pela rede e as saídas desejadas, representando o erro do modelo. A escolha da função de perda depende do tipo de tarefa que está sendo realizada, ou seja, para determinados tipos de problemas funções de perda se tornam mais adequadas.

4. **Propagação para trás**:

   Também conhecida como *backward propagation*. O erro calculado é propagado de volta às camadas ocultas, calculando o erro ponderado para cada neurônio com base nos pesos das conexões. Esse cálculo é realizado através do gradiente descendente. O gradiente é a derivada parcial da função de perda em relação aos pesos das conexões. Ele indica a direção e a magnitude do ajuste necessário para minimizar o erro. É preciso calcular primeiro o gradiente na camada de saída, utilizando a derivada parcial da função de perda em relação à saída da camada de saída. Em seguida o gradiente é propagado para trás, calculando o gradiente em cada camada oculta até chegar a primeira camada.

5. **Atualização dos pesos**:

Os pesos das conexões são atualizados usando os gradientes calculados na etapa anterior, com o objetivo de minimizar a função de perda. A base desse processo é a utilização da regra delta. A regra delta multiplica o erro pelo valor da entrada correspondente e multiplica esse resultado por uma **taxa de aprendizagem** (um hiperparâmetro que controla a velocidade de atualização dos pesos). Essa atualização de peso é feita para todos os neurônios de todas as camadas.

Em bibliotecas como o *KERAS*, é possível controlar a forma como os pesos são atualizados, o hiperparâmetro que define isso é o otimizador. Ele utiliza o algoritmo de retropropagação (backpropagation) para calcular os gradientes descendentes da função de perda em relação aos pesos e, em seguida, aplica as atualizações necessárias para minimizar a perda. Existem vários otimizadores disponíveis e cada otimizador possui suas próprias características e hiperparâmetros, permitindo ajustar o processo de atualização dos pesos de acordo com as necessidades do problema.

6. **Repetição do processo**:

   Os passos 2 a 5 são repetidos para cada exemplo de treinamento do conjunto de dados. Essa repetição é chamada de época (*epoch*). O algoritmo continua a ajustar os pesos iterativamente para minimizar o erro em todo o conjunto de dados.

7. **Convergência**:

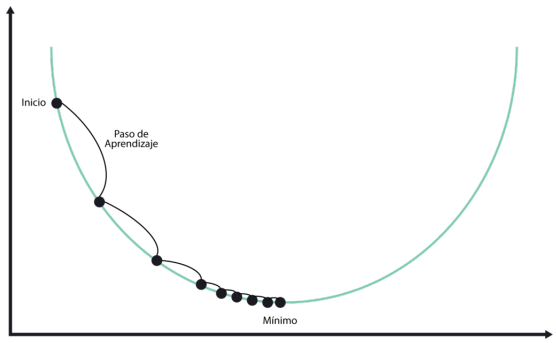
   O treinamento continua até que o erro de predição seja suficientemente baixo ou até que um número máximo de épocas seja alcançado. A convergência ocorre quando o modelo atinge uma boa capacidade de generalização e pode realizar previsões precisas em dados não vistos anteriormente.

A taxa de aprendizagem, mencionada brevemente no passo 5, é um hiperparâmetro crucial no treinamento de redes neurais. Ela controla a magnitude dos ajustes feitos nos pesos da rede durante o processo de aprendizado. Esse hiperparâmetro é capaz de controlar a velocidade de aprendizado e influência diretamente na convergência da rede. Na figura 2.11 podemos ver o resultado esperado para todas soluções, onde o erro mínimo (eixo y), é encontrado em relação ao conjunto de pesos (eixo x), em algumas épocas de atualização. A figura 2.12 representa o funcionamento de uma rede com taxas de aprendizagem muito pequenas, podemos ver que a convergência ótima pode até acontecer, mas a velocidade que isso ocorrerá é muito baixa. Já a figura 2.13 representa o funcionamento de redes com a taxa de aprendizagem muito alta, onde a convergência não ocorre e o erro é ainda aumentado.

2.3.3 Validação cruzada

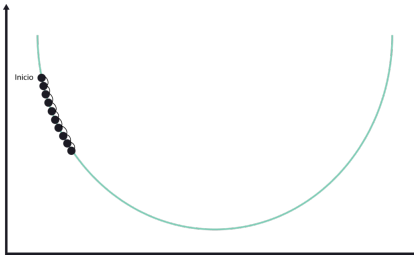
As especificações topológicas das redes PMC, ou seja, a determinação do número de camadas, funções de ativação, quantidade de neurônios na rede e tantas outras decisões de

Figura 2.11 – Taxa de aprendizagem otimizada



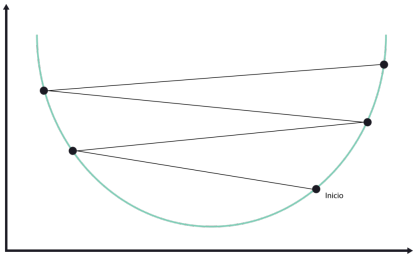
Fonte: (RUBIO, 2020)

Figura 2.12 – Taxa de aprendizagem baixa



Fonte: (RUBIO, 2020)

Figura 2.13 – Taxa de aprendizagem alta



Fonte: (RUBIO, 2020)

escolhas, são realizadas na maior parte das vezes de forma empírica. Algumas indicações sobre as escolhas existem, como apresentado na seção 2.2.2.1 em relação as funções de ativação. Porém mesmo com algumas indicações para resolução de problemas, existe mais de um caminho a ser tomado para cada decisão. Como a maior parte das decisões é tomada empiricamente, é preciso submeter esses hiperparâmetros em testes até selecionar um conjunto dos melhores para a rede. Esse processo é conhecido como *tuning*.

Quando os hiperparâmetros de uma rede neural são avaliados fazendo esse *tuning*, onde se escolhe as melhores combinações em relação aos resultados, existem alguns problemas que influenciam na aprendizagem e não são controlados. Fatores como a forma com que a matriz de pesos é iniciada, a complexidade do problema, a disposição das amostras de treinamento e a qualidade do conjunto de treinamento, são fatores que mudam os resultados. Por isso é importante avaliar também a capacidade de generalização da solução quando submetida a um conjunto de teste diferente do conjunto de treinamento. Uma forma muito conhecida que avalia essa capacidade é a validação cruzada.

A validação cruzada (*cross-validation*) é uma técnica utilizada no aprendizado de máquina para avaliar a capacidade de generalização de um modelo e estimar seu desempenho em dados não vistos. O objetivo da validação cruzada é mitigar o viés e a variância na avaliação

do modelo, proporcionando uma estimativa mais confiável do seu desempenho. (KOHAVI, 2001)

No treinamento de uma RNA, é comum dividir os dados disponíveis em um conjunto de treinamento e um conjunto de teste. O conjunto de treinamento é usado para treinar o modelo, enquanto o conjunto de teste é usado para avaliar seu desempenho. No entanto, essa divisão em apenas dois conjuntos pode levar a resultados enviesados, pois o modelo pode ser ajustado de forma otimizada para o conjunto de teste específico, assim como fatores específicos dessa divisão de dados pode afetar no resultado da rede.

A validação cruzada resolve esse problema dividindo os dados em vários subconjuntos. O modelo é treinado e avaliado várias vezes, alternando entre diferentes combinações de conjuntos de treinamento e teste. Por exemplo, na validação cruzada k-partições, os dados são divididos em k partes iguais. O modelo é treinado k vezes, usando k-1 partes como conjunto de treinamento e a parte restante como conjunto de teste. O desempenho é então avaliado pela média dos resultados obtidos em cada iteração. Esse modelo de validação cruzada pode ser visto na figura 2.14, onde k é igual a 5. É possível encontrar artigos que estudam maneiras de reduzir mais ainda o viés com técnicas mais avançadas de validação cruzada, no artigo (XIONG et al., 2020) é possível ver algumas variações da validação cruzada, assim como a avaliação desses modelos.

Figura 2.14 – Validação cruzada

Fonte: (PATRO, 2021)

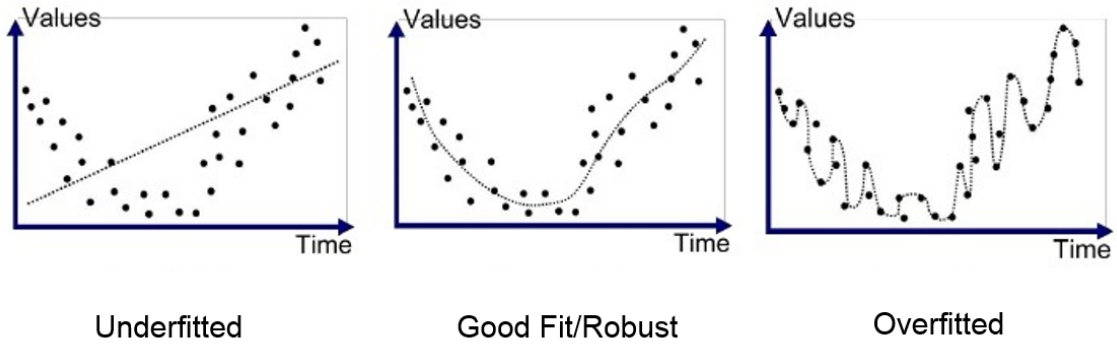
2.3.4 *Overfitting* e *Underdfitting*

Em relação a escolha da topologia utilizada é preciso tomar cuidado com a seleção de hiperparâmetros, pois um aumento indiscriminado de neurônios e de camadas não garante uma boa generalização da rede PMC, pelo contrário. Um dos problemas comuns que podem ocorrer em redes neurais se dá quando os erros durante o treinamento são baixos,

porém quando a rede neural é aplicada a um conjunto de teste, os erros percebidos são altos e diferentes do esperado. Isso é chamado de *overfitting* ou de sobreajuste (HASTIE; TIBSHIRANI; FRIEDMAN, 2001).

O *overfitting* ocorre quando um modelo se ajusta muito bem aos dados de treinamento, mas não generaliza bem para novos dados. Em outras palavras, o modelo se torna excessivamente complexo e memoriza o ruído e as flutuações nos dados de treinamento, em vez de capturar os padrões gerais subjacentes. Como resultado, o desempenho do modelo em dados não vistos é prejudicado. O caminho contrário do *overfitting* é o *underfitting*. Ele ocorre quando um modelo não é capaz de capturar bem os padrões nos dados de treinamento e, portanto, não generaliza bem para novos dados. Em vez de se ajustar aos dados de treinamento, o modelo é muito simples ou tem pouca capacidade de representação, resultando em um desempenho inferior tanto nos dados de treinamento quanto nos dados de teste. A figura 2.15 podemos ver a comparação do que seriam essas abordagens mencionadas, em comparação com uma curva com boa generalização

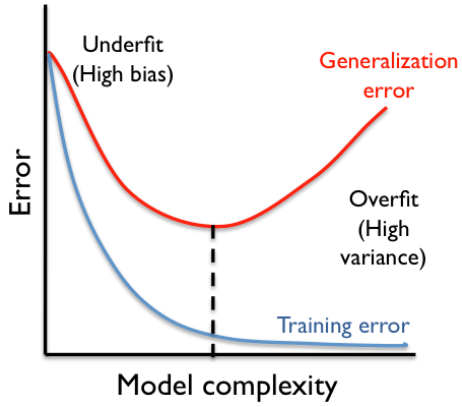
Figura 2.15 – Exemplos de curvas sub-ajustada, adequada e sobre-ajustada respectivamente



Fonte: (ABRACD, 2021)

De uma forma geral uma rede PMC deve ser capaz de se superar o *underfitting* e evitar o *overfitting*. Em uma rede que está sendo submetida a um processo de validação cruzada é possível realizar a parada antecipada do processo (DOMINGOS, 2012). Essa parada ocorre quando começar a existir uma elevação do erro a partir de determinada época de treinamento, uma vez que a rede em validação cruzada é constantemente avaliada pela aplicação do subconjunto de teste. A figura 2.16 representa a erro dos resultados em relação ao conjunto de teste e ao conjunto de treinamento, a complexidade do modelo nesse caso se resume a topologia utilizada, como um bom exemplo disso é possível citar o número de épocas, quantidade de dados, número de camadas, etc. É possível ver que em uma certa complexidade do modelo o resultado é ótimo e para complexidades maiores observa-se o *overfitting*.

Figura 2.16 – Curva de complexidade do modelo vs erro para dados de treino e teste.



Fonte: (ABRACD, 2021)

Fica claro que o *overfitting* é o caso mais difícil de verificar que está ocorrendo, por isso existem algumas formas de implementar técnicas com a finalidade evitar que o sobreajuste aconteça (LAKSHMANAN; GÖRNER; GILLARD, 2021). Entre eles podemos citar:

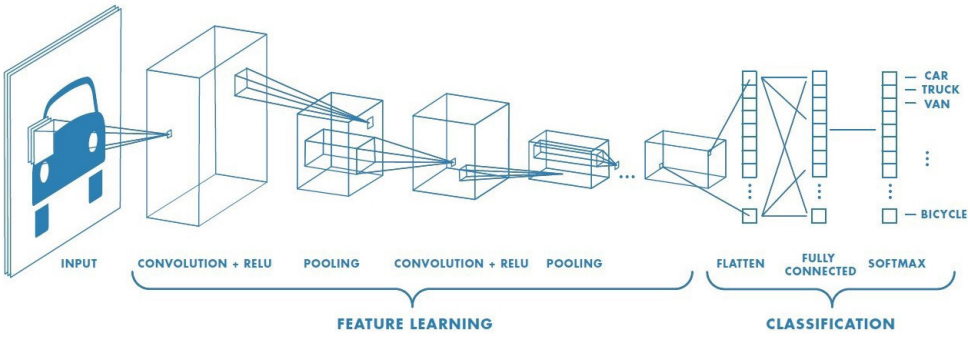
- **Regularização**: A regularização é uma técnica usada para evitar o *overfitting*. Essa técnica permite que se aplique penalidades nos parâmetros da camada ou na atividade da camada durante a otimização. Essas penalidades são somadas na função de perda durante o processo de treinamento e têm o efeito de penalizar pesos excessivos.
- ***Dropout***: Essa técnica envolve aleatoriamente desligar (zerar) uma porcentagem dos neurônios durante o treinamento, o que ajuda a evitar o *overfitting* e melhorar a capacidade de generalização. Isso força a rede a aprender características redundantes de diferentes maneiras e evita que unidades específicas dependam demais de outras unidades, tornando a rede mais robusta (GOODFELLOW; BENGIO; COURVILLE, 2016). No artigo (POERNOMO; KANG, 2018) é apresentado o funcionamento e a importância dessa técnica e um estudo sobre duas extensões do *Dropout* baseadas no comportamento de unidades ocultas no modelo CNN.
- **Aumento de dados**: O *data augmentation* é a capacidade de gerar mais exemplos de treinamento através de técnicas de aumento de dados pode ajudar a melhorar a capacidade de generalização do modelo.

2.4 Redes neurais convolucionais

Nessa seção discutiremos sobre uma classe especial de perceptrons de múltiplas camadas, conhecidas como redes convolutivas. As redes neurais convolucionais (RNCs) são uma classe especializada de redes neurais artificiais, projetadas especialmente para

processar dados que possuem uma estrutura bidimensional. As RNCs, conhecidas também como Convolutional Neural Network (CNN), são amplamente utilizadas em diversas áreas, como visão computacional, reconhecimento de padrões, processamento de imagens e até mesmo em jogos. As RNCs se destacam por sua capacidade de aprender automaticamente recursos relevantes de entrada, permitindo que sejam usadas para tarefas complexas, como classificação de imagens, detecção de objetos e segmentação semântica.

Figura 2.17 – Rede neural convolucional



Fonte: (MATHWORKS, 2018)

Essas redes tem algumas capacidades diferentes das PMCs, elas são capazes de reconhecer formas com um alto grau de invariância, translação, escalonamento, inclinação e outras tantas formas de distorção. As rede convolutivas, de forma geral, segue os seguintes passos de funcionamento: extração de características, mapeamento dessas características e subamostragem dos dados. Essa estrutura de funcionamento tem inspiração neurobiológica com origem no seguinte trabalho (HUBEL; WIESEL, 1977), que estuda o córtex visual do macaco. Mas só em 1988 com o artigo (LECUN et al., 1998) que o tema foi estabelecido.

2.4.1 Funcionamento das redes de convolução

A Rede Neural Convolucional se trata de um é um algoritmo de aprendizado profundo que pode captar uma imagem de entrada, atribuir importância (pesos e vieses que podem ser aprendidos) a vários aspectos. As RNCs possuem camadas, assim como nas redes perceptron onde existem camadas de neurônios, essas redes possuem camadas que desempenham um papel crucial no processamento e aprendizado dos dados de entrada. Portanto as camadas fundamentais são: Camada de convolução, camada de ativação e camada de *pooling*.

2.4.1.1 Camada de convolução

A camada de convolução é o coração de uma RNC. Ela é responsável por aplicar filtros convolucionais nas entradas, extraindo características relevantes em diferentes regiões da imagem. Cada filtro convolucional é uma pequena matriz que percorre a imagem e realiza operações de convolução, multiplicando seus valores com os valores correspondentes da imagem e somando-os, gerando assim o mapa de características da imagem. Um aprofundamento nas definições matemáticas em relação ao operador de convolução podem ser vistas nessa referência (CONTRIBUTORS, 2023a). Já a convolução aplicada diretamente em imagens é definida como a soma ponderada de produtos entre um detector de características e a região correspondente na imagem, matematicamente pode ser definido como (CONTRIBUTORS, 2023b):

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x-dx, y-dy) \tag{2.14}$$

Onde:

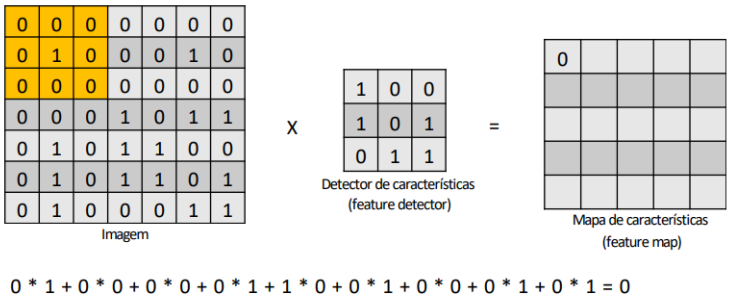
- $g(x,y)$ representa o mapa de características
- ω representa o detector de características, também conhecido como *Kernel* ou filtro convolucional.
- $f(x,y)$ representa a imagem propriamente dita

Na figura 2.18 é possível ver a forma que funciona o operador de convolução. Esse processo é realizado para a imagem toda, vemos que a imagem é uma matriz 7x7, já o mapa de características gerado, é uma matriz 5x5, ou seja, houve uma redução na dimensão da imagem. Quando o tamanho do filtro é maior que 1x1, ocorre uma sobreposição parcial entre as regiões cobertas pelo filtro em diferentes posições. Conforme o filtro é deslizado pelo volume de entrada, ocorrem sobreposições parciais em todas as posições. Essa sobreposição tem o efeito de reduzir as dimensões do mapa de características resultante.

O volume resultante da camada convolucional trata-se das dimensões de altura, largura e quantidade de mapa de características e existem 3 parâmetros que controlam essas dimensões: Profundidade, passo e *zero-padding* (KARN, 2016).

A profundidade refere-se à quantidade de filtros utilizados na camada convolucional. Cada filtro gera um mapa de características diferente, e quanto mais filtros forem usados, maior será a profundidade da saída. É importante destacar que o número de filtros está diretamente relacionado à complexidade computacional do problema. Camadas convolucionais com um grande número de filtros demandam mais tempo e memória para processamento.

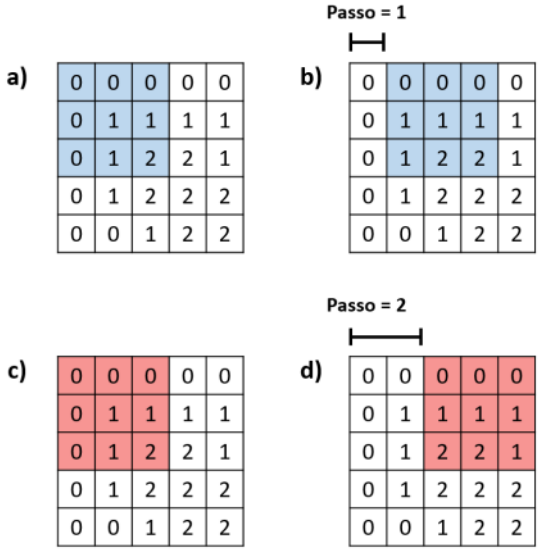
Figura 2.18 – representação de convolução



Fonte: (EXPERT, 2023)

O passo, também conhecido como *stride*, especifica o tamanho do salto na operação de convolução. Na figura 2.19 é possível ver o papel que diferentes números de saltos tem nesse processo. Quanto maior for o valor do passo, menor serão a altura e a largura do volume resultante. No entanto, é importante notar que características importantes podem ser perdidas com um valor de passo muito elevado. Por esse motivo, é incomum usar um passo maior que 2.

Figura 2.19 – Influência do passo no deslocamento do filtro na imagem



Fonte: (ARAÚJO et al., 2017)

A operação de *zero-padding* consiste em preencher a borda do volume de entrada com zeros. Essa operação tem a vantagem de permitir o controle das dimensões de altura e largura do volume resultante, tornando-os iguais aos valores do volume de entrada. Isso pode ser útil em certos casos para garantir que o tamanho do volume seja mantido durante

as operações de convolução.

Com esses parâmetros é possível calcular a altura (A_{mp}) e largura (L_{mp}) do volume resultante de uma camada convolucional, com as seguintes equações:

$$A_{mp} = \frac{A - F + 2P}{S} + 1 \tag{2.15}$$

$$L_{mp} = \frac{L - F + 2P}{S} + 1 \tag{2.16}$$

Onde A e L correspondem a altura de largura do volume de entrada, F é o tamanho do filtro usado, P é o valor do *zero-padding* e S é o valor do passo

Durante o treinamento da RNC, os filtros convolucionais são ajustados automaticamente, de forma a otimizar o desempenho da rede em tarefas específicas. Isso permite que a rede aprenda a identificar e extrair características relevantes a partir dos dados de entrada. Eles podem captar diversas características variando seus valores, a figura 2.20 aplica um filtro de identidade na imagem, gerando assim a mesma imagem, já a figura 2.21 é aplicado um filtro de relevo, onde o objetivo é dar uma ilusão de profundidade na imagem. As figuras 2.20 e 2.21 foram retiradas do site (POWELL, 2023), nesse endereço podem ser testados e criados diversos tipos de filtros.

Figura 2.20 – Aplicação de um filtro identidade

Fonte: (POWELL, 2023)

Figura 2.21 – Aplicação de um filtro de relevo

Fonte: (POWELL, 2023)

A camada de convolução é então a junção de todos os mapas de características formados ao longo do treinamento. É importante salientar que faz parte do treinamento selecionar os mapas que mais se ajustam na resolução do problema, isso significa que a rede será treinada para variar os valores do filtro de características.

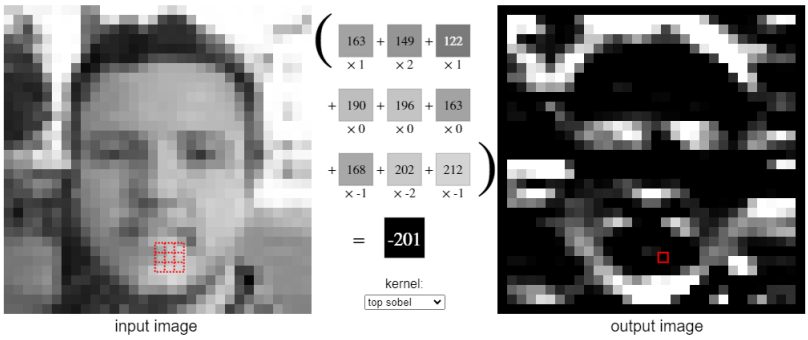
2.4.1.2 Camada de ativação

A camada de ativação introduz não linearidades na rede, permitindo que a RNC aprenda representações mais complexas e não lineares dos dados. A função mais comumente usada em RNCs é a função ReLU (*Rectified Linear Unit*), que retorna zero para valores negativos e o próprio valor de entrada para valores positivos. Por exemplo na figura 2.22, o *kernel* escolhido foi o sobel, que é usado para mostrar apenas as diferenças nos valores

de pixel adjacentes em uma direção específica. Observe que na figura o pixel em destaque possui um valor negativo, a função ReLU, por exemplo transformaria o valor desse pixel em zero, isso cria melhores padrões e ajuda o treinamento da rede.

Dessa forma a função de ativação é aplicada elemento por elemento ao mapa de características, ativando ou desativando certas partes da imagem, dependendo dos valores de entrada. Isso ajuda a introduzir não linearidades e a capturar relacionamentos mais complexos entre as características extraídas pela camada de convolução.

Figura 2.22 – Aplicação de um filtro sobel



Fonte: (POWELL, 2023)

2.4.1.3 Camada de *Pooling*

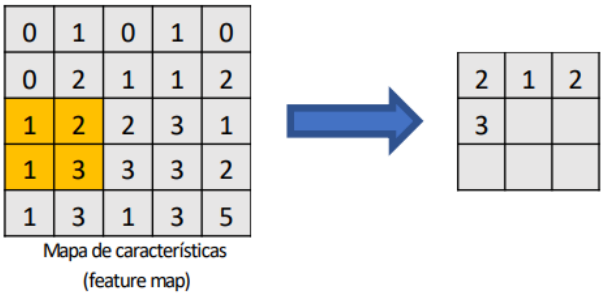
A camada de *pooling* é usada para reduzir a dimensionalidade dos dados de saída da camada de convolução, mantendo as características mais importantes. Isso é feito dividindo a imagem em regiões não sobrepostas e, em seguida, selecionando o valor máximo (*max pooling*) ou a média (*average pooling*) dentro de cada região (GOODFELLOW; BENGIO; COURVILLE, 2016). Na figura 2.23 podemos ver a aplicação do *max polling*, onde a dimensão inicial da imagem era 5x5 e se tornou 3x3 preservando as características mais relevantes .

O objetivo do *pooling* é fornecer uma representação mais compacta dos dados, reduzindo a quantidade de informações processadas e preservando as características mais relevantes. Além disso, o *pooling* ajuda a tornar a rede mais robusta a pequenas variações ou deslocamentos nas características detectadas pela camada de convolução.

2.4.2 Arquitetura da rede

A arquitetura típica de uma RNC é projetada para aprender de forma hierárquica e discriminativa, com camadas de convolução iniciais capazes de detectar características de baixo nível, como bordas e texturas, e camadas posteriores capazes de aprender representações mais complexas e abstratas. Através de uma combinação de convoluções, operações

Figura 2.23 – Aplicação do *Pooling*

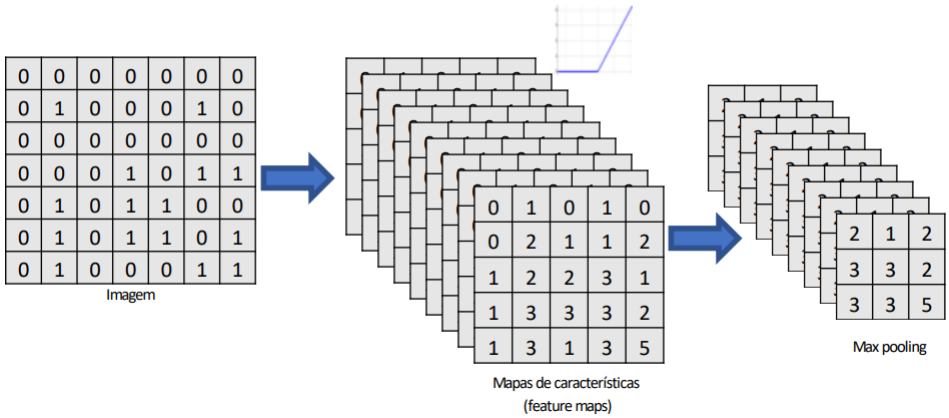


Fonte: (EXPERT, 2023)

de *pooling* e funções de ativação, as camadas da RNC conseguem capturar informações relevantes nas diferentes regiões das imagens.

A arquitetura da rede é composta pelas 3 camadas vistas anteriormente: camadas de convolução, camadas de ativação e camadas de *pooling*. Esses componentes básicos são empilhados em um formato de arquitetura em cascata. A estrutura de uma RNC típica varia dependendo da tarefa e do modelo específico, permitindo ajustes no número de camadas, tamanho dos filtros, conexões entre as camadas e outras características específicas. A Figura 2.24 ilustra um conjunto de camadas sendo aplicadas em sequência, mas é importante destacar que múltiplos conjuntos de camadas podem ser empilhados um após o outro para aumentar a profundidade da rede.

Figura 2.24 – Camadas resultantes do processo de convolução



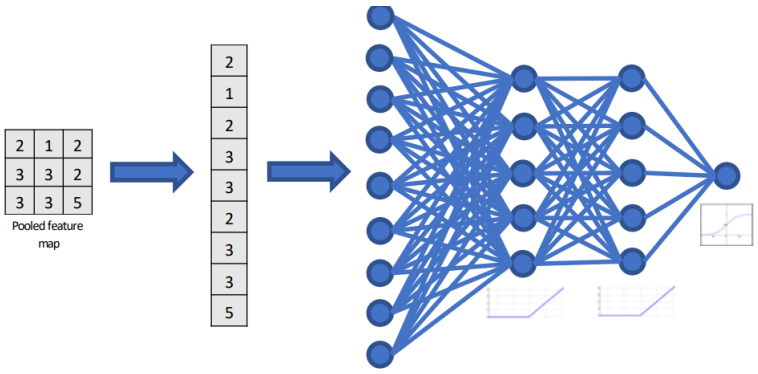
Fonte: (EXPERT, 2023)

Essa arquitetura em cascata é projetada para extrair gradualmente informações mais complexas e abstratas à medida que a informação percorre a rede. No final da sequência de camadas, é aplicada a etapa de *Flattening*, que consiste em transformar a matriz resultante

das camadas anteriores em um vetor unidimensional. Esse vetor é então passado para uma rede perceptron de múltiplas camadas (PMC). Na Figura 2.25, podemos observar esse processo, que ocorre após as camadas de convolução, ativação e *pooling*. Essa etapa leva a uma rede de camadas totalmente conectadas, que executa a tarefa final de classificação ou regressão, utilizando as informações extraídas nas camadas anteriores para fazer previsões precisas.

O *Flattening* é importante para permitir que as informações extraídas pelas camadas anteriores sejam processadas pelas camadas totalmente conectadas. Ao transformar a matriz em um vetor, a estrutura espacial dos dados é perdida, mas é possível preservar as informações relevantes necessárias para a tarefa final. Essa etapa também ajuda a reduzir a complexidade computacional, já que uma representação unidimensional é mais adequada para as camadas totalmente conectadas.

Figura 2.25 – Aplicação do *Flattening*



Fonte: (EXPERT, 2023)

É importante ressaltar que a arquitetura exata de uma RNC pode variar de acordo com a aplicação e o objetivo do modelo. Em problemas mais complexos, como reconhecimento de objetos em imagens ou segmentação de imagens, pode ser necessário empilhar mais camadas e utilizar técnicas adicionais.

3 Referências bibliográficas

3.1 Trabalhos anteriores

Esse trabalho é uma continuação do trabalho de mestrado (FRANCO, 2007), realizado em 2007 na Universidade de Brasília (UnB) pelo Luciano Duarte. Neste trabalho, foi realizado o desenvolvimento de um sistema de visão computacional em tempo real com o objetivo de capturar e medir imagens durante o processo de soldagem GMAW no modo curto-circuito. A captura das imagens foi sincronizada com a detecção de curto-circuito, visando obter imagens do arco com pouco ou nenhum brilho, para permitir que um algoritmo de processamento de imagens possa avaliar a posição do arame de soldagem e da poça fundida.

Durante o desenvolvimento desse sistema, foram estudadas as principais tecnologias envolvidas, como o processo de soldagem GMAW e os motivos para escolher a soldagem no modo curto-circuito. Além disso, foi realizada a análise e monitoramento dos sinais de soldagem, como tensão e corrente, e determinou-se o melhor momento para a captura das imagens.

Posteriormente, foi estabelecida toda a infraestrutura necessária para a criação e experimentação do sistema. Isso incluiu a montagem de um sistema computacional industrial para captura de imagens, o uso de uma mesa linear para o deslocamento do corpo de prova, a seleção do sistema mais adequado para detecção e sincronização de imagens, bem como o desenvolvimento e validação de uma série de softwares, como o software de detecção de curto-circuitos e o software de captura e medição de imagens.

Os resultados obtidos com esse sistema alcançaram os objetivos propostos, uma vez que foi possível obter imagens de boa qualidade, sem saturação, sincronizadas com a detecção de curto-circuito. Isso permitiu a medição precisa da poça de solda e o processamento robusto das imagens obtidas.

Um outro projeto desenvolvido por Rodrigo Ferreira em 2015 na Universidade de Brasília, (FERNANDES, 2015), é muito similar em objetivo final a esse projeto atual, somente os métodos de realização para se chegar no objetivo que são diferentes. Ele tem como objetivo o desenvolvimento de métodos para obtenção em tempo real de parâmetros de soldagem GMAW. O projeto realiza processos de visão computacional para detectar parâmetros de soldagem como largura da poça, posição do arame implementados em Matlab, utilizando várias técnicas e filtros para essa função. Ao final um outro objetivo apresentado é a implementação da solução em um FPGA para reduzir o tempo de processamento.

Dando continuidade nesse projeto, em 2016 também na Universidade de Brasília, Felipe Fraga realizou um projeto de mestrado, que consistia no desenvolvimento de um

sensor de visão para o monitoramento dos parâmetros geométricos da poça de soldagem, (SILVA, 2016). Esse trabalho melhorou a obtenção das imagens obtidas por meio da soldagem GMAW e desenvolveu um um processo de visão computacional para dimensionar a largura da poça de fusão. O autor deixa como sugestão para trabalhos futuros o uso de inteligência artificial para a identificação dos parâmetros.

3.2 Histórico das Redes Neurais Convolucionais

Nos últimos anos, a área de aprendizagem profunda e visão computacional tem visto avanços significativos graças ao desenvolvimento de arquiteturas de redes neurais poderosas e inovadoras. Compreender as arquiteturas de redes mais importantes se tornou um aspecto fundamental para os pesquisadores e profissionais que buscam explorar o potencial dessas tecnologias. Essas arquiteturas pioneiras e influentes estabeleceram as bases teóricas e práticas para o campo, impulsionando o progresso em diversas aplicações de reconhecimento de padrões, processamento de imagem e análise de dados. Veremos como essas arquiteturas fornecem referências teóricas, orientação para problemas específicos, melhorias no desempenho, compreensão dos avanços do campo e facilitação da comunicação e colaboração.

No caso do projeto arquiteturas como a *LeNet*, *AlexNet* e *VGGNet* serão usadas e combinadas como base para a nossa própria arquitetura. A ideia básica toda da *LeNet* e como as camadas são ordenadas será a base de construção do projeto. A questão da profundidade e técnicas como *dropout* serão usadas e testadas, inspiradas na *AlexNet*. O uso de filtros convolucionais pequenos de tamanho 3x3, serão usados, inspirados pela *VGGNet*

3.2.1 *LeNet* (1988)

A *LeNet* é uma arquitetura de rede neural convolucional (RNC) pioneira, desenvolvida por Yann LeCun e seus colegas no início da década de 1990 (LECUN et al., 1998). Ela foi projetada especialmente para tarefas de reconhecimento de caracteres escritos à mão, como o reconhecimento de dígitos em cheques bancários.A LeNet foi uma das primeiras arquiteturas de CNN a obter sucesso em aplicações práticas de visão computacional.

A seção 2.4 mostra toda a estrutura de uma rede neural convolucional, e a estrutura apresentada trata-se da própria arquitetura LeNet. Essa arquitetura foi escolhida para a apresentação do tema pois as novas arquiteturas que foram propostas nos últimos anos, são melhorias da LeNet, isso porque a grande maioria compartilha dos mesmo conceitos fundamentais apresentados.

3.2.2 *AlexNet* (2012)

A *AlexNet* é uma arquitetura de rede neural convolucional que teve um impacto significativo no campo da visão computacional. Ela foi desenvolvida por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton (KRIZHEVSKY; SUTSKEVER; HINTON, 2017) e ganhou o desafio *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) em 2012. A vitória da AlexNet nessa competição marcou um ponto de virada para as redes neurais convolucionais e atraiu grande atenção para o campo da aprendizagem profunda.

Uma das principais contribuições da *AlexNet* foi a demonstração do potencial das RNCs profundas no reconhecimento de imagens em grande escala. A arquitetura da *AlexNet* era muito mais profunda do que as arquiteturas convolucionais anteriores e apresentava um número significativo de parâmetros treináveis. Isso foi possível graças ao uso de unidades de processamento gráfico (GPUs) para acelerar o treinamento da rede.

A *AlexNet* consiste em cinco camadas convolucionais seguidas por três camadas totalmente conectadas. Além da arquitetura em si, a *AlexNet* também introduziu outras técnicas importantes para o treinamento de redes neurais profundas, como o uso de regularização por *dropout* e treinamento com taxas de aprendizado adaptativas.

3.2.3 VGGNet (2014)

A VGGNet, ou VGG (*Visual Geometry Group*), é uma arquitetura de rede neural convolucional desenvolvida pelos pesquisadores do *Visual Geometry Group* da Universidade de Oxford. Ela foi proposta por Karen Simonyan e Andrew Zisserman em 2014 (SIMONYAN; ZISSERMAN, 2014) e se destacou por sua simplicidade arquitetônica e desempenho impressionante em várias tarefas de visão computacional.

A principal característica da *VGGNet* é sua profundidade. Ela possui uma estrutura mais profunda do que a maioria das arquiteturas existentes na época, chegando a 19 camadas convolucionais. Essa profundidade permitiu à *VGGNet* aprender representações de alto nível mais complexas e discriminativas das imagens.

Um aspecto notável da *VGGNet* é o uso de filtros convolucionais de tamanho pequeno (3x3) em todas as camadas convolucionais. Esses filtros menores têm um campo receptivo limitado, mas permitem que a rede aprenda representações mais ricas e capture detalhes finos das imagens. A utilização repetida de camadas convolucionais com filtros 3x3 resulta em uma representação mais profunda e não linear das características.

3.2.4 *GoogLeNet* e *Inception* (2014)

A *GoogLeNet*, também conhecida como Inception, é uma arquitetura de rede neural convolucional desenvolvida pelos pesquisadores do Google. Foi proposta por Christian

Szegedy et al. em 2014 (SZEGEDY et al., 2017) e ficou famosa por sua arquitetura inovadora, que introduziu o conceito de módulos *Inception*.

Uma das principais motivações por trás do desenvolvimento da *GoogLeNet* era lidar com o desafio de projetar redes mais profundas e eficientes computacionalmente. Em vez de simplesmente aumentar a profundidade da rede, a *GoogLeNet* adotou uma abordagem de largura, onde várias operações de convolução são executadas em paralelo e, em seguida, concatenadas.

O principal componente da arquitetura *GoogLeNet* é o módulo *Inception*. Ele consiste em várias convoluções em paralelo com diferentes tamanhos de filtro (1x1, 3x3, 5x5) e uma operação de *pooling*. Esses diferentes filtros convolucionais capturam informações em diferentes escalas e níveis de abstração, permitindo que a rede aprenda representações mais ricas e complexas das imagens.

Outra inovação importante introduzida pela *GoogLeNet* é o uso de camadas de convolução 1x1 para reduzir a dimensionalidade dos mapas de características antes da aplicação de convoluções mais computacionalmente intensivas. Essas convoluções 1x1 são conhecidas como convoluções de *bottlenecks* e ajudam a reduzir o número de parâmetros e o custo computacional da rede. Na figura A.49 é possível ver um pouco de como funciona essa arquitetura

3.2.5 *ResNet* (2015)

A ResNet (*Residual Network*) é uma arquitetura de rede neural profunda desenvolvida por Kaiming He em 2015 (HE et al., 2015). Ela foi projetada para superar o desafio do desvanecimento do gradiente que ocorre em redes neurais muito profundas. A ResNet introduziu o conceito de blocos residuais, que permitiu a criação de redes extremamente profundas, com mais de 100 camadas, sem comprometer o desempenho.

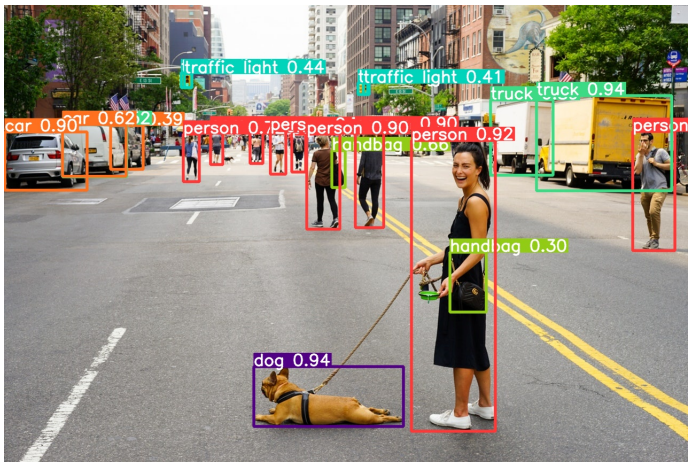
O desvanecimento do gradiente ocorre quando o sinal do gradiente diminui à medida que é propagado pelas camadas profundas da rede, dificultando o treinamento eficaz. A ResNet aborda esse problema usando conexões residuais, que permitem que as informações originais sejam transmitidas diretamente através da rede, ao invés de serem totalmente transformadas em cada camada.

A ideia central por trás dos blocos residuais é a introdução de atalhos (*shortcut connections*) que saltam uma ou mais camadas, permitindo que o gradiente seja transmitido diretamente para camadas posteriores. Esses atalhos adicionais permitem que a rede aprenda as diferenças residuais, ou seja, as diferenças entre a entrada e a saída esperada em cada bloco. Isso facilita o treinamento de redes profundas, pois as informações originais são preservadas e o gradiente pode fluir mais facilmente.

3.2.6 YOLO

O YOLO (*You Only Look Once*) é um *framework* popular desenvolvido inicialmente por Joseph Redmon e Ali Farhadi em 2016 (REDMON et al., 2016), usado para a detecção de objetos em tempo real, que se destaca por sua eficiência e velocidade. Ele aborda o problema de detecção de objetos como um problema de regressão, onde um único modelo de rede neural é treinado para prever a localização e a classe dos objetos em uma imagem em uma única passada. Na figura 3.26 podemos ver como funciona o sistema de detecção de objetos usado no YOLO, onde um conjunto de caixas delimitadoras são colocadas na imagem com o objetivo de dimensionar e classificar os objetos. O YOLO divide a imagem

Figura 3.26 – Funcionamento do YOLO para detecção de objetos



Fonte: Produzido pelo autor

de entrada em uma grade de células e cada célula é responsável por prever um conjunto de caixas delimitadoras e as classes correspondentes aos objetos contidos nessas caixas. A arquitetura é baseada em uma rede neural convolucional profunda. Ela usa várias camadas convolucionais, juntamente com camadas de normalização, ativação e *pooling*, para extrair características da imagem de entrada em várias escalas. Essas características são então usadas para fazer previsões de detecção em diferentes resoluções.

Uma característica importante do YOLO é o uso de blocos residuais, inspirados na arquitetura *ResNet*, que ajudam a resolver o problema de desaparecimento de gradientes e permitem o treinamento de redes neurais mais profundas. Esses blocos residuais ajudam a melhorar a capacidade de representação do modelo e a detectar objetos de diferentes tamanhos e escalas.

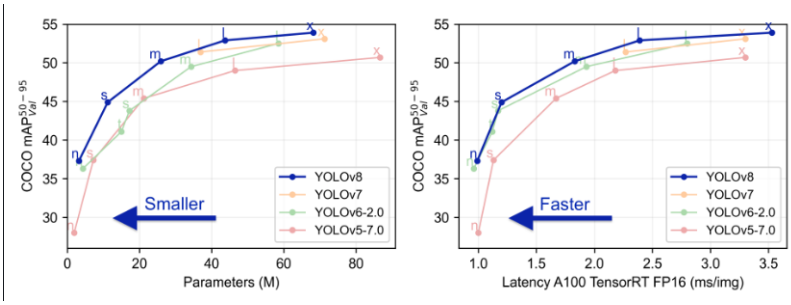
Além disso, o YOLO utiliza várias técnicas para melhorar o desempenho, como ativações Mish, camadas SPP (*Spatial Pyramid Pooling*) para capturar informações contextuais em diferentes escalas e camadas PANet (*Path Aggregation Network*) para melhorar a

comunicação entre diferentes escalas de recursos.

O treinamento do YOLO envolve a otimização de uma função de perda que combina termos de regressão e classificação. O modelo é treinado em um conjunto de dados rotulados, onde as caixas delimitadoras e as classes dos objetos são fornecidas. O processo de treinamento envolve o ajuste dos pesos da rede neural para minimizar a função de perda, usando algoritmos de otimização, como o gradiente descendente estocástico.

O YOLO vem sendo atualizado e desde de sua criação novas versões mais rápidas e poderosas vem sendo criadas, é possível ver essa evolução e o modo de funcionamento das demais versões nesse artigo (TERVEN; CORDOVA-ESPARZA, 2023). Na data que esse relatório está sendo escrito já está disponível o YOLOv8 criado pela *Ultralytics* (JOCHER; CHAURASIA; QIU, 2023), que é o modelo mais recente do *framework*. Na figura 3.27 podemos ver uma rápida comparação entre as últimas versões. O YOLOv8 oferece suporte a uma gama completa de tarefas de IA de visão, incluindo detecção , segmentação , estimativa de pose , rastreamento e classificação.

Figura 3.27 – Evolução do YOLOv8 em comparação com outras versões



Fonte: (JOCHER; CHAURASIA; QIU, 2023)

A intenção do projeto é realizar a criação de uma arquitetura que se aproxime de arquiteturas como o YOLO, onde seja possível prever dimensões dos parâmetros da soldagem. Caso precise, uma alternativa será usar a arquitetura da rede YOLOv8 para fazer um treinamento na rede por meio de *transfer learning*, onde a rede já treinada é usada para o treinamento de um novo conjunto de dados. Essa abordagem servirá para resolver o problema da detecção dos parâmetros assim como analisar o desempenho desse tipo de abordagem na solução.

3.3 Trabalhos similares

Alguns trabalhos desempenharam um papel fundamental no desenvolvimento deste projeto, fornecendo orientações e direcionamento para a pesquisa. Eles serviram como fonte de inspiração, permitindo explorar diferentes aspectos, aplicar técnicas adequadas e organizar

a execução do projeto de maneira lógica. A contribuição desses trabalhos foi fundamental para o sucesso do projeto, proporcionando uma base sólida, evitando obstáculos conhecidos e alcançando resultados significativos. Eles guiaram a pesquisa e o desenvolvimento, ajudando a estabelecer um quadro conceitual e teórico, bem como identificar as melhores práticas.

O primeiro trabalho a ser comentado é o (ZHOU et al., 2023), que realiza o monitoramento de qualidade nos procedimentos de soldagem de bocal de motor de foguete líquido (LREN) . Esse artigo propõe o uso de uma arquitetura chamada de WDS-net (Rede de segmentação de detecção de solda) para a detecção de parâmetros de solda, classificação de desses parâmetros em situações de falha e situações sem falha e além disso o trabalho conta com a segmentação de imagem. Esse trabalho utiliza como arquitetura base para o projeto uma rede YOLOv3, no menor modelo possível, devido ao seu tamanho e velocidade de resposta, sendo mais adequada para ambientes industrias por conta dessas características.

Outro trabalho é uma revisão de processos de detecção de soldagens a arco elétrico realizados por robôs, (XU et al., 2022). Esse trabalho primeiro aborda diversas técnicas de detecção de aspectos de soldagem, técnicas como sensores de visão, sensor acústico e muitos outros. Além disso o autor discute os aspectos de soldagem e faz uma revisão entre as arquiteturas, essa revisão tem como objetivo apresentar os resultados de cada arquitetura e apresentar a eficiência nos aspectos de soldagem apresentados. Esse trabalho apresenta algumas informações sobre caminhos a seguir no processo de detecção e é útil para apresentar as abordagens conhecidas.

O artigo (AMARNATH; SUDHARSHAN; SRINIVAS, 2023) analisa e compara as metodologias existentes usadas no passado e o que está sendo usado agora para detecção de defeitos em processos de soldagem. Os autores discutem como redes convolucionais podem resolver os problemas enfrentados pela visão de máquina tradicional. Para isso são verificados o funcionamento de redes CNN com diferentes arquiteturas, para classificação de imagens de soldagem entre classes de defeitos, analisando o funcionamento dessa rede e a acurácia dos resultados. No artigo também se discute como *Vision Transformers* podem ser incorporados para obter melhores resultados.

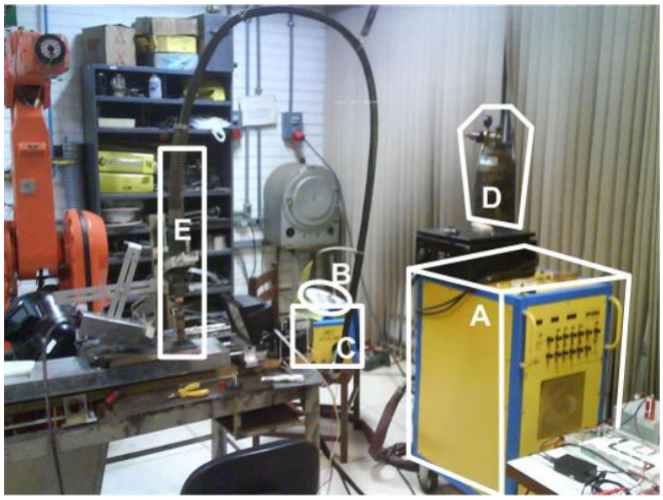
O artigo (PENTTILÄ et al., 2019) implementa um controle completo do processo de soldagem usando redes neurais. O trabalho apresenta um sistema de controle em malha fechada onde se utiliza um sensor a laser para escanear o processo e captar dados do processo de soldagem, apos esse passo os dados passam por uma rede neural que analisa os dados e atualiza os parâmetros de soldagem. É um trabalho interessante focado no processo final de um projeto de análise de parâmetros, onde a análise de resultado é feita na própria qualidade da soldagem.

4 Metodologia

4.1 Experimento GMAW

Com base no trabalho de Luciano et al. (2007) (FRANCO, 2007), foram conduzidos seis experimentos para a captura de imagens durante o processo de soldagem do tipo transferência por curto-circuito, apresentada na seção 2.1. O sistema de soldagem utilizado, ilustrado na Figura 4.28, é composto por uma fonte de soldagem (A), uma bobina de arame de soldagem (B) com seu respectivo tracionador (C), um cilindro de gás de proteção (D) e uma tocha de soldagem (E). A tocha de soldagem é composta por um bocal condutor que possui três conexões, uma para o gás de proteção, uma para a fonte de corrente e uma para o arame de soldagem. A partir do sistema montado um determinado conjunto de configurações foi utilizado para a realização desses seis experimentos.

Figura 4.28 – Sistema de soldagem utilizado nos experimentos



Fonte: (FRANCO, 2007)

As configurações gerais dos experimentos podem ser vistas a seguir

- Distância entre o bocal de gás e a chapa = 17mm
- Distância entre a lente e o ponto de toque do arame na chapa = 165mm
- Ângulo entre a câmera e a chapa = 35 graus
- Lente de 12,5mm mais 5mm de espaçado

- Gás de proteção: Argônio + 6% de O_2
- Taxa de captura da câmera = 1000 fps

Na tabela 4.2 vemos alguns parâmetros escolhidos para a realização dos experimentos.

Tabela 4.2 – Parâmetros do experimento

Experimento	Tensão	Tempo de exposição	wfs
1	19,5 V	50ms	7 m/min
2	19.3 V	50ms	7 m/min
3	19.5 V	50ms	7 m/min
4	19.6 V	50ms	7 m/min
5	19.6 V	60ms	7 m/min
6	19.8 V	55ms	7 m/min

A tensão se refere a tensão é aplicada entre a peça de trabalho (metal base) e o eletrodo consumível (arame de solda). A quantidade de tensão aplicada afeta vários aspectos do processo de soldagem. Ela influencia a estabilidade do arco elétrico, o controle da transferência de metal de adição, a taxa de deposição, a penetração e a aparência do cordão de solda. A tensão adequada deve ser selecionada de acordo com o tipo de material, a espessura da peça de trabalho, o diâmetro do arame de solda e outros fatores específicos da aplicação de soldagem.

A velocidade de alimentação do arame, também conhecida como *wire feed speed* (wfs) é uma medida de quão rápido o arame está se movendo através da tocha de soldagem, medida em metros por minutos. É importante destacar que a *wire-feed speed* varia dependendo do tipo de material, do diâmetro do arame, da aplicação de soldagem e das configurações do equipamento. A escolha adequada da velocidade de alimentação do arame é essencial para garantir a fusão adequada do metal de adição e a formação de uma junta soldada de qualidade.

O tempo de exposição se refere ao tempo de exposição do obturador da câmera, Normalmente é mantido relativamente curto durante a soldagem, geralmente em frações de segundo. Isso ajuda a evitar a superexposição da imagem, reduzindo a quantidade de luz e radiação capturada pela câmera.

4.1.1 Resultados gerados

As imagens geradas durante um processo de soldagem de transferência por curto-circuito podem apresentar vários aspectos que fornecem informações valiosas sobre o processo e a qualidade da solda. Esses aspectos podem ser analisados para avaliar a presença de defeitos, a forma do cordão de solda, a intensidade luminosa e outras características

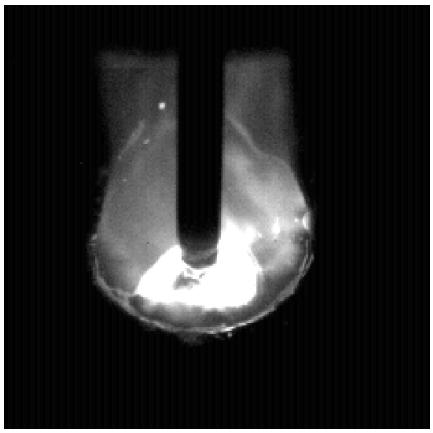
importantes. A seguir, destacam-se alguns aspectos das imagens geradas durante a soldagem por curto-circuito:

- **Intensidade luminosa**: A intensidade luminosa é um aspecto característico das imagens de soldagem que pode variar de acordo com a corrente e a tensão. Durante o processo de soldagem por curto-circuito, o arco elétrico produz uma intensa luminosidade que pode diferenciar a etapa do processo de soldagem. Usaremos isso inclusive para diferenciar na rotulação os momentos de curto dos momentos de não-curto.
- **Formato do poça de solda**: A poça de solda é formado pela fusão do metal de adição com o material base. A imagem capturada pode revelar o formato do cordão, incluindo sua largura, altura e geometria geral. O formato do cordão de solda é um indicativo importante da qualidade da solda e da eficiência do processo.
- **Defeitos**: As imagens geradas durante a soldagem por curto-circuito também podem revelar a presença de defeitos, como porosidades, trincas ou falta de fusão. Esses defeitos aparecerão como irregularidades na imagem, que podem indicar problemas durante o processo de soldagem ou inadequações nas configurações dos parâmetros.
- **Respingos**: Durante a soldagem, podem ocorrer respingos de metal derretido que são projetados para fora do cordão de solda. Esses respingos podem ser observados nas imagens e indicam a ocorrência de salpicos. O controle dos respingos é importante para evitar danos ao material adjacente e garantir a qualidade da solda.
- **Aspectos do arame**: A imagem capturada pode revelar o formato do arame assim como sua a posição em relação à poça de fusão. Isso é importante para verificar se o arame está sendo posicionado corretamente e se está fornecendo a quantidade adequada de metal de adição para obter uma solda adequada. Além disso é possível estimar o diâmetro do arame de solda com base na imagem capturada. Isso pode ser útil para verificar se o diâmetro do arame utilizado está de acordo com as especificações do processo de soldagem.

Esses são apenas alguns dos aspectos que podem ser observados nas imagens geradas durante a soldagem por curto-circuito. A análise dessas imagens auxilia na avaliação da qualidade da solda, no ajuste dos parâmetros do processo e no controle de possíveis defeitos. Na figura 4.29 é possível ver o exemplo de duas imagens geradas durante o processo de soldagem, sendo o 4.29a um imagem em situação de curto e a figura 4.29b gerada a partir de um momento de não-curto

Figura 4.29 – Imagens obtidas a partir do experimento 3

(a) Imagem com curto



(b) Imagem sem curto

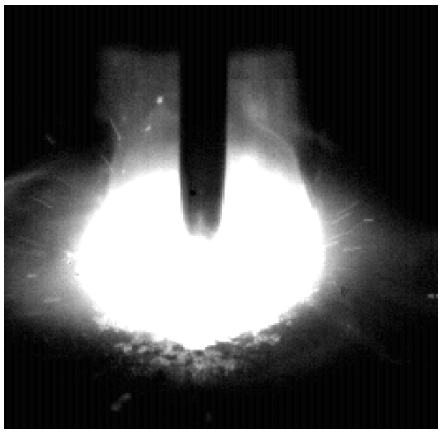


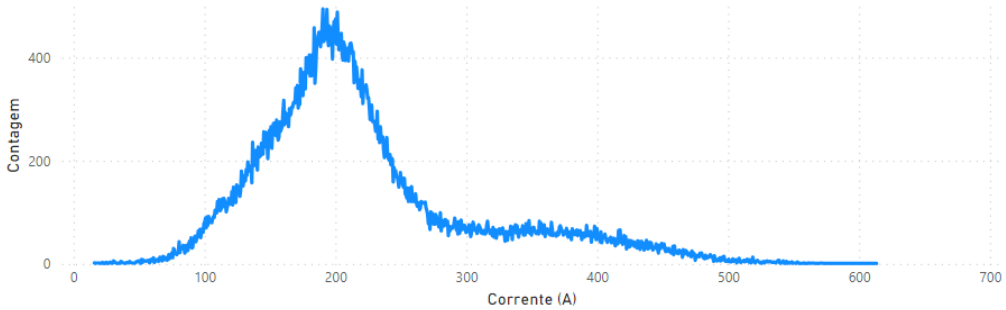
Figura 4.30 – Imagens obtidas a partir do experimento 3

Fonte: Produzidas pelo autor

No processo de soldagem por curto-circuito, a corrente é um dos parâmetros essenciais que afetam o desempenho e o resultado da solda. Durante a soldagem por curto-circuito, a corrente elétrica flui através do arco formado entre o eletrodo consumível (arame de solda) e a peça de trabalho (metal base).

É importante ressaltar que certa variação da corrente é esperada durante a soldagem de transferência por curto-circuito. No entanto, é essencial que essa variação esteja dentro de uma faixa aceitável para garantir a qualidade da solda. Valores de corrente muito baixos podem resultar em falta de fusão e falta de penetração, enquanto valores muito altos podem causar problemas como respingos excessivos e risco de porosidade. Na figura 4.31 observa-se os valores de corrente obtidos durante o experimento 1, foram realizadas 100 mil leituras de valores de corrente e vemos as ocorrências desses valores no gráfico.

Figura 4.31 – Valores de corrente observados durante a realização do experimento 1



Fonte: produzida pelo autor

4.2 Rotulagem dos dados

A partir dos seis experimentos realizados foi possível conseguir um total de 6000 imagens, devemos então ser capazes de conseguir extrair parâmetros importantes dessas imagens e fazer assim um banco de dados com as saídas desejadas. Para a rotulagem de dados usaremos o software da ***Label Box***, que fornece um mecanismo personalizável criado para produzir dados de treinamento de alta qualidade. Esse software permite a rotulagem de dados de várias formas, é possível classificar imagens em diversas categorias, fazer seleção de diversos elementos em imagens, obter o dimensionamento de objetos na imagem e muitas outras funcionalidades. O *software* também permite a distribuição do processo de rotulagem entre várias pessoas, no caso desse projeto tivemos a participação de 4 pessoas no processo de rotulação.

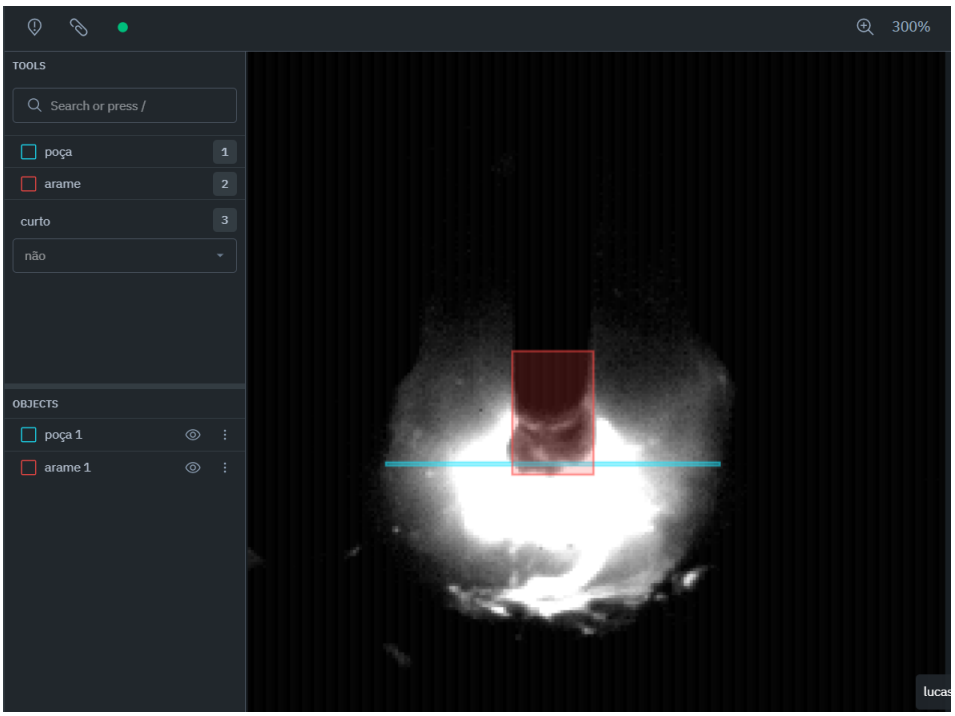
Para iniciar a rotulação foi preciso hospedar 6 mil imagens 232x230px do tipo "bmp"no site da *Label Box*. Esse formato também é conhecido como *Device Independent Bitmap* ou (DIB) e é um tipo de arquivo rasterizado não compactado projetado para exibir imagens de alta qualidade no Windows. Após colocar as imagens no site o processo de rotulagem é simples, foi criado um processo personalizado de etiquetamento, onde era preciso classificar a imagem em momento de curto-circuito ou não e delimitar as dimensões e posição da poça de soldagem e do arame. No caso da poça, é importante determinar a posição da poça na imagem, e sua largura. Seguindo a mesma lógica, em relação ao arame o objetivo é determinar a posição do arame na imagem e a altura da ponta final do arame. A figura mostra 4.32 como funcionava esse processo de rotulagem dentro do *Label Box*.

Esse processo foi de rotulação foi realizado então cerca de seis mil vezes, apenas algumas imagens onde a imagem não era conclusiva ou não era possível identificar o arame a rotulação foi pulada, esse fenômeno aconteceu cerca de 38 vezes. Ao final da rotulação foi possível exportar todos os dados em formato Json, onde foram obtidos mais de 40 objetos para cada imagem. O código 4.1 apresenta uma versão reduzida dessa saída, onde é possível ver como o *software* entrega os valores da rotulagem. Os dados do etiquetamento podem ser vistos e até mesmo outras informações como o tempo gasto na rotulação da imagem, imagens puladas, id único e nome da imagem são fornecidos.

Código 4.1 – Saída de uma rotulação reduzida

```
{"ID":"cl4rdfn831it1074jcj3o43lf",
"Label":{"objects":[{"title":"poça","value":"poca",
"bbox":{"top":136,"left":46,"height":23,"width":110}},
{"title":"arame","value":"arame",
"bbox":{"top":117,"left":90,"height":21,"width":23}}],
"classifications":[{"title":"curto","value":"curto",
"answer":{"title":"não","value":"nao"}}]},
"Seconds to Label":25.05,"Seconds to Create":25.05,
"External ID":"Img990.bmp","Skipped":false}
```

Figura 4.32 – Rotulagem no *Label Box*



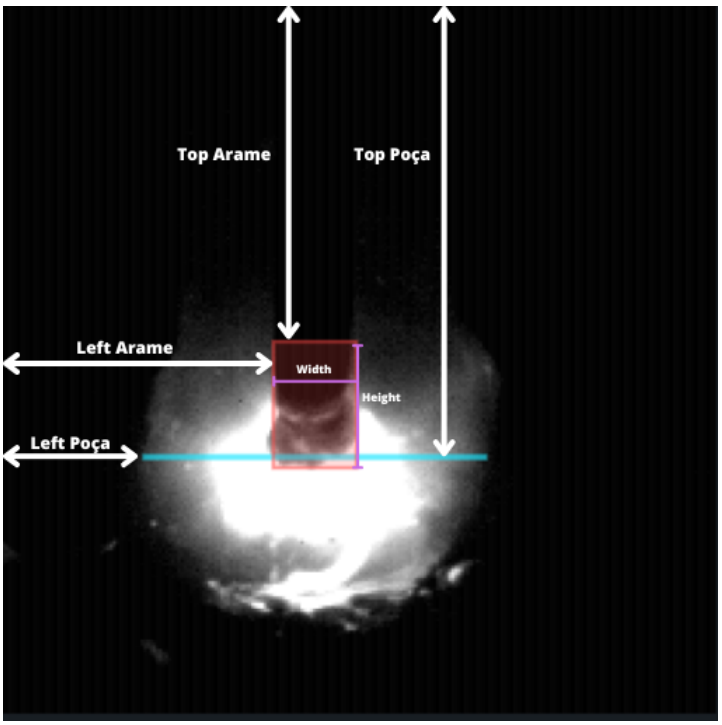
Fonte: Produzido pelo autor

Para cada *box* etiquetada nas imagens temos as seguintes dimensões: *top*, *left*, *height* e *width*. A variável *top* representa a distância entre aresta superior da caixa até a aresta superior da imagem, em *pixels*. A variável *left* representa a distância entre a aresta esquerda da caixa até a aresta esquerda da imagem, em *pixels*. A variável *height* representa a altura da caixa em *pixels*. A variável *width* representa a largura da caixa em *pixels*. Na figura 4.33 é possível ver como essas dimensões fornecidas funcionam.

Após obtermos os dados, foi desenvolvido um pequeno código em *Python* para transformar o arquivo *JSON* em uma planilha utilizável para o treinamento supervisionado da rede neural a ser criada (conforme apresentado no Código A.10). Esse processo resultou em uma planilha contendo todas as informações das rotulagens, juntamente com alguns dados adicionais calculados.

Os valores de dimensões presentes na planilha foram calculados em relação à dimensão da imagem original e estão expressos como porcentagem. Por exemplo, se o valor "Esquerda"(*Left*) de uma determinada poça em uma imagem é igual a 90 *pixels* e a largura total da imagem é 232 *pixels*, isso representa 39% da largura da imagem, ou seja, $90/232 = 0.3879$. Além disso, a altura do final do arame foi calculada somando os valores "Top"(Topo) e "Altura"(*Height*) do arame. Dessa forma, a planilha resultante contém todas as informações relevantes das rotulagens, incluindo as dimensões normalizadas em relação à imagem e a altura do final do arame. Esses dados fornecem as informações necessárias para o treinamento

Figura 4.33 – Exemplo das dimensões obtidas



Fonte: Produzido pelo autor

supervisionado da rede neural.

Uma modificação realizada nesse arquivo csv, foi em relação os nome das imagens, uma vez que foram realizados 6 experimentos, com a geração de 1000 imagens cada, numeradas de 0 até 999, foi preciso realizar uma mudança nos nomes dessas variáveis para que não existam 6 nomes iguais e 6 classificações diferentes para cada imagem. Por isso os nomes das imagens foram alterados de acordo com o experimento que a imagem fazia parte.

4.2.1 Análise dos dados

A análise dos dados desempenha um papel fundamental no treinamento de redes neurais, particularmente na compreensão e análise da sua distribuição. A qualidade dos dados utilizados durante o treinamento é essencial para garantir o desempenho e a precisão dos modelos de redes neurais. Ao analisar, é possível obter informações valiosas sobre sua distribuição e características. Isso envolve examinar estatísticas descritivas, como média, mediana, desvio padrão e outros indicadores relevantes. Além disso, a visualização dos dados por meio de gráficos, histogramas, diagramas de dispersão e outras técnicas nos permite compreender melhor a maneira como os dados estão distribuídos.

Essa análise da distribuição dos dados permite identificar possíveis desequilíbrios, padrões, tendências ou anomalias nos conjuntos de dados. Essas informações são fundamen-

tais para tomar decisões informadas durante o treinamento das redes neurais. Por exemplo, se os dados estiverem desequilibrados, com uma classe dominante em relação às outras, o modelo pode ter dificuldade em aprender corretamente a distinguir as classes minoritárias. Além disso, ao analisar a distribuição dos dados, é possível selecionar as técnicas de pré-processamento adequadas. Dependendo da natureza dos dados e de sua distribuição, pode ser necessário aplicar transformações, como normalização, padronização e balanceamentos. Essas técnicas ajudam a preparar os dados para o treinamento da rede neural, melhorando sua capacidade de generalização e reduzindo o impacto de valores extremos.

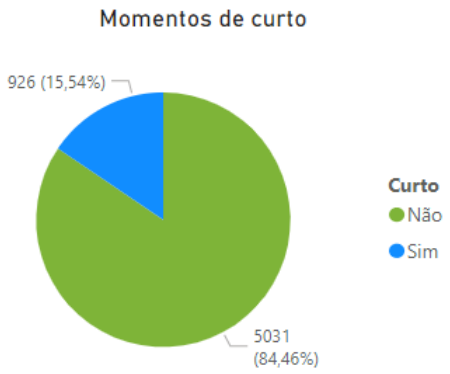
Outro aspecto importante da análise dos dados é a identificação de possíveis vieses ou distorções presentes nos conjuntos de dados. É essencial verificar se os dados são representativos do problema. Caso existam vieses, como a falta de diversidade ou a presença de dados incorretos, o modelo de rede neural pode aprender a partir dessas características indesejáveis, resultando em previsões enviesadas ou imprecisas. Claro que em nossa rotulagem de 6 mil imagens feitas a mão, alguns dados não foram rotulados corretamente, assim como a precisão das etiquetas colocadas não foi muito minuciosa muitas das vezes, ainda sim confiamos que a maior parte dos dados tem uma boa acurácia em suas rotulagens e veremos a seguir como se deu esse resultado. As visualizações dos dados mostrados foi feito no *software* ***Power BI***.

- **Curto**: Existem dois momentos nesse experimento, um é quando a imagem é capturada em um momento onde a soldagem está em estado de curto e o outro momento é quando a soldagem não está em momento de curto. A figura mostra uma demonstração da diferença desses dois momentos

  Em relação ao momento de curto a figura 4.34, mostra qual o percentual encontrado em relação as imagens. É possível ver que 84,46% das imagens não apresentam momentos de curto enquanto 15,54% das imagens foram feitas em momentos de curto. Agora a figura 4.35, apresenta em quais experimentos tivemos o maior número de momentos de curto, o experimento 4 por exemplo, representa 18,68% de todos os momentos de curto disponíveis para o treinamento da rede, tendo uma diferença de 4% em relação ao experimento que menos contribui para o treinamento, que é o experimento 1.

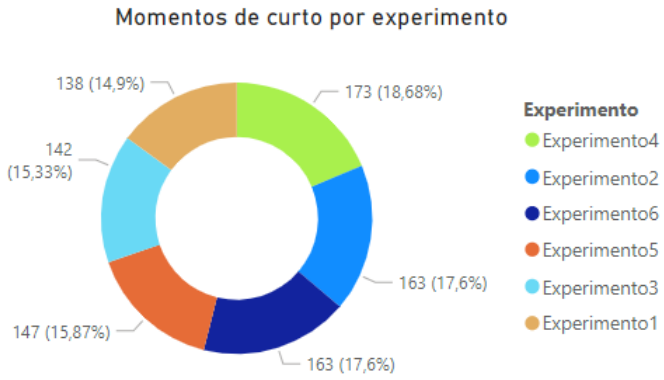
- **Arame**: Em relação as dimensões do arame, deseja-se medir 2 parâmetros, que são: a posição que se inicia o arame e a posição da ponta final do arame. Na figura 4.36, é possível ver a distribuição da métrica que mede a distância que se inicia o arame em relação a borda esquerda da imagem. Já na figura 4.37 vemos um comparativo da posição da ponta do arame em momentos de curto e de não curto, observa-se que nos momentos de não curto o arame tende a ter valores maiores, isso significa que a ponta do arame é mais baixa nesse estado de soldagem.

Figura 4.34 – Porcentagem de imagens em momentos de curto



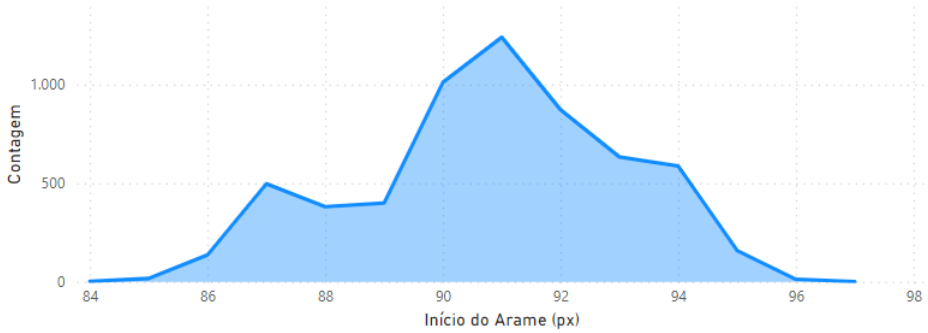
Fonte: Produzido pelo autor

Figura 4.35 – Porcentagem de imagens em momentos de curto por experimento



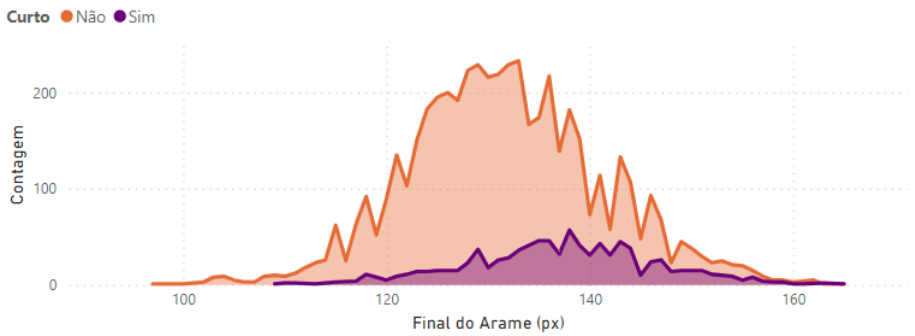
Fonte: Produzido pelo autor

Figura 4.36 – Distribuição da posição inicial do arame



Fonte: Produzido pelo autor

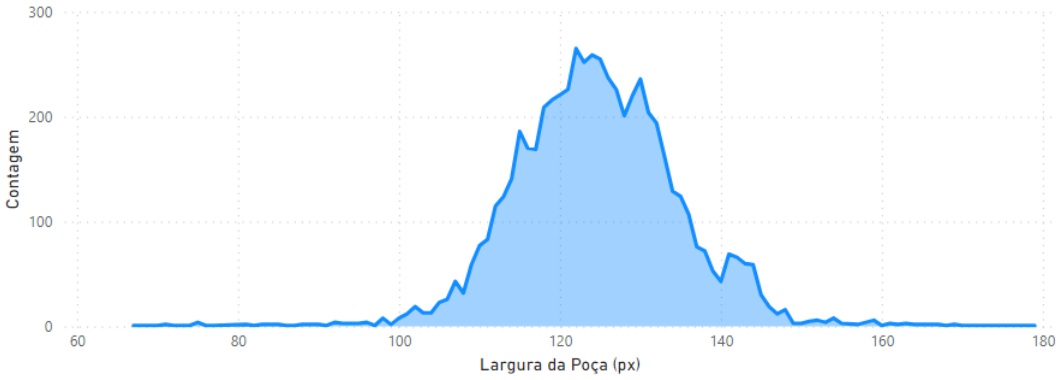
Figura 4.37 – Distribuição da altura do arame comparando momentos de curto-circuito



Fonte: Produzido pelo autor

- **Poça**: Ao analisarmos a poça de soldagem, observa-se dois parâmetros cruciais a serem estudados: a largura da poça e sua posição de início. Na figura 4.38, é possível ver a distribuição das dimensões de largura de poça rotulados, é possível observar que existe uma grande distribuição entre esses valores, indo um pouco um pouco mais afundo nesse parâmetro a figura 4.39 mostra a relação que esses dados tem com o momento de curto, vemos que para momentos de não curto as dimensões da poça são menores, mas da mesma forma os dois valores mostram que existe uma grande distribuição nesses valores.

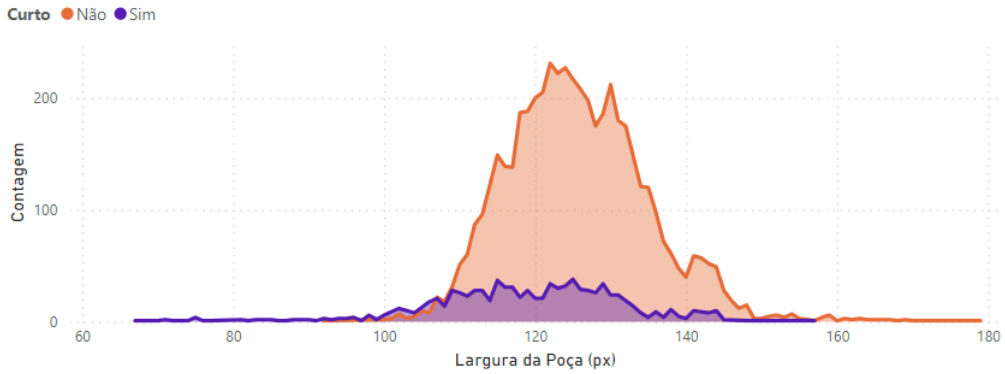
Figura 4.38 – Distribuição da largura da poça



Fonte: Produzido pelo autor

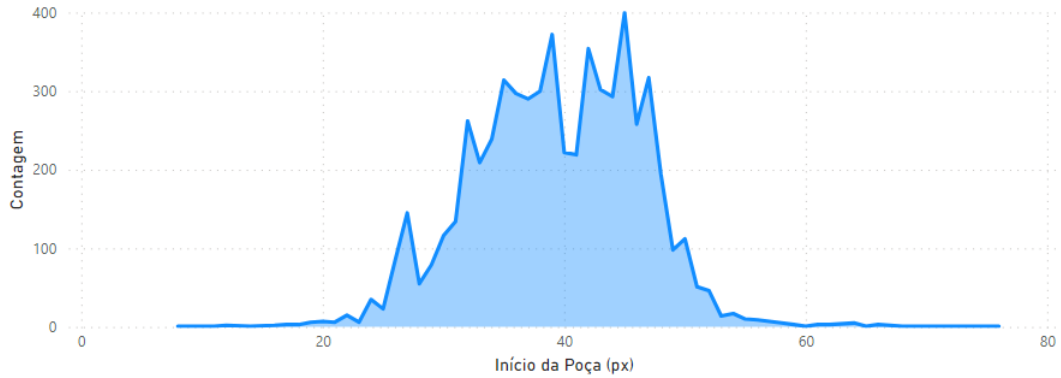
Já em relação ao início da poça, mostrado na figura 4.33 como **Left Poça**, é possível ver sua distribuição na figura 4.40, onde temos também uma boa distribuição. é possível concluir que em relação a poça de soldagem, os parâmetros serão mais dispersos no treinamento quando comparados ao arame.

Figura 4.39 – Distribuição da largura da poça em relação ao momento de curto



Fonte: Produzido pelo autor

Figura 4.40 – Distribuição da posição inicial da poça



Fonte: Produzido pelo autor

4.3 Construindo a rede neural

Nesse projeto o objetivo é a construção de uma única rede neural capaz de estipular os valores de parâmetros relacionados a soldagem. Faremos isso a partir de redes neurais convolucionais que são especialmente adequadas para o processamento de imagens, pois são capazes de aprender e extrair características relevantes das imagens de forma automatizada. Elas são compostas por camadas convolucionais, de pooling e camadas totalmente conectadas. O funcionamento e teoria por trás dessas redes foram apresentadas nas seções 2.3 e 2.4, a arquitetura base da rede será inspirada em 3 principais arquiteturas: *LeNet*, *AlexNet* e *VGGNet* conforme apresentadas na seção 3.2 .

Essas redes por sua vez precisam ser executadas em alguma linguagem de programação, a escolha para esse projeto é a linguagem *Python* por uma série de motivos. O *Python*

possui várias bibliotecas populares e poderosas voltadas para aprendizado de máquina e processamento de imagens, como o *TensorFlow*, *Keras*, *PyTorch* e *scikit-learn*. Essas bibliotecas oferecem uma ampla gama de funcionalidades e recursos específicos para o desenvolvimento de redes neurais convolucionais. No artigo

Além disso o *Python* possui um ecossistema de pacotes e ferramentas abrangente, que vai além das bibliotecas específicas de aprendizado de máquina. Isso inclui pacotes para manipulação de dados, visualização, pré-processamento de imagens, entre outros. Essas ferramentas facilitam o desenvolvimento completo do pipeline de análise de imagens, desde a leitura e pré-processamento até o treinamento e avaliação da CNN. Por último outra vantagem da linguagem é a portabilidade, o que significa que o código pode ser executado em diferentes sistemas operacionais, como *Windows*, *macOS* e *Linux*. Isso permite que a CNN desenvolvida em *Python* seja implantada em diferentes ambientes, como servidores locais, nuvem ou dispositivos embarcados.

Dito isso para a construção do primeiro modelo de rede neural convolucional a ser estudado, iremos construir uma rede neural convolucional para a classificação de momentos de curto em *Python*. Por isso a saída da nossa rede terá um neurônio capaz de decidir se a imagem está em um momento de curto ou não. A construção dessa rede será bem simples e servirá apenas como base para as outras redes que serão criadas, por isso o foco será detalhar ao máximo o processo.

Iniciaremos com a entrada dos dados e seus respectivos tratamentos. Essa primeira rede neural terá como base de dados 1000 imagens e as correspondentes 1000 rotulagens de momentos de curto que a rede deverá fazer a previsão. Para esse projeto todas as imagens estão presentes dentro de um diretório e suas respectivas classes e saídas estão mapeadas em um arquivo CSV.

O processo de entrada desses dados ser visto no código 4.2. Na linha 1 temos a importação da biblioteca **ImageDataGenerator** sua documentação pode ser vista em (TENSORFLOW, 2023). Utilizamos essa função pré-definida do *TensorFlow* para facilitar e simplificar a associação de uma imagem ao seu respectivo nome. Ao extrairmos as imagens do diretório, a biblioteca automaticamente estabelece a relação entre o nome da imagem, a própria imagem e sua classe correspondente.

Na linha 8, essa biblioteca também oferece uma maneira de processar os dados, normalizando-os. Esse processo envolve a divisão do valor de cada pixel da imagem por 255, o que resulta em valores de pixel variando entre 0 e 1. Essa normalização facilita o processo de treinamento. Além disso, a biblioteca oferece a opção de dividir os dados entre conjuntos de treinamento e validação. No código apresentado, a divisão foi configurada para 75% dos dados para treinamento e 25% para validação. Embora existam diversas configurações disponíveis para aumentar a base de imagens, como rotação e mudança de direção, essas técnicas não serão utilizadas neste projeto inicialmente.

Na linha 2 temos a importação da biblioteca *pandas*. Sua documentação pode ser vista em (PANDAS, 2023). Essa biblioteca nos permite extrair o *data frame* contendo os dados das rotulagens. No *data frame*, as colunas contendo o nome da imagem e a classificação do momento de curto serão utilizadas na rede neural.

A função **flow from dataframe** é usada para determinar o conjunto de treino e validação, ela também relaciona a imagem com a linha do *data frame*. Nessa função precisamos especificar o caminho do diretório das imagens, o tamanho das imagens, o rótulo que a rede deverá prever, o tipo de rede neural, que no nosso caso será de classificação binária e o tipo da imagem, que significa dizer quantos canais de cor a imagem tem. O tamanho do *batch size* será estudado mais a frente, mas ele define o número de amostras de treinamento que serão propagadas pela rede neural antes de ocorrer uma atualização dos pesos.

Código 4.2 – Entrada de dados para classificação de momentos de curto

```
1  from keras.preprocessing.image import ImageDataGenerator
2  import pandas as pd
3
4  trainDf = pd.read_excel("/diretorio/dataframe/Experimento1.xlsx")
5
6  trainDf["Curto"] = trainDf["Curto"].astype(str)
7
8  datagen =
       ImageDataGenerator(rescale=1./255.,validation_split=0.25)#
       Normalização dos dados e separação do conjunto de validação
9
10 train_generator = datagen.flow_from_dataframe(
11     dataframe=trainDf, # Dataframe com os dados
12     directory='/diretorio/imagens/Experimento1',
13     x_col='Nome da imagem', # Coluna com os nomes das imagens
14     y_col= "Curto",  # Coluna com os rótulos
15     subset='training',   # Conjunto de treinamento
16     batch_size=25,
17     color_mode='grayscale',  # Para imagens com um canal de cor
18     class_mode='binary',  # Para classificação binária
19     target_size=(230, 230)   # Tamanho desejado das imagens
20 )
21
22 validation_generator = datagen.flow_from_dataframe(
23     dataframe=trainDf,
24     directory='/diretorio/imagens/Experimento1',
25     x_col='Nome da imagem',
26     y_col= "Curto",
27     subset='validation',  # Conjunto de validação
28     batch_size=25,
29     color_mode='grayscale',
30     class_mode='binary',
31     target_size=(230, 230)
32 )
```

No código 4.3 fazemos então a criação da rede neural propriamente dita. Será listada a função de cada camada e seus atributos mais relevantes. Vale lembrar que toda a documentação pode ser encontrada em (KERAS, 2023).

1. **Modelo sequencial**: Na linha 5, temos a criação da rede no modelo sequencial, que consiste no agrupamento em sequência de camadas.

2. **Camada de convolução**: Essa camada foi apresentada na seção 2.4.1.1. Ela é responsável por aplicar os *kernels* na imagem e extrair características. Na linha 7 vemos sua aplicação, que conta com os seguintes atributos:

   - ***filters***: Primeiro parâmetro fornecido, trata-se do número de filtros de saída na convolução
   - ***kernel size***: Segundo parâmetro fornecido, especifica a altura e a largura da janela de convolução 2D
   - ***strides***: Especifica os passos da convolução ao longo da altura e largura. Padrão *default* é (1,1).
   - ***padding***: Pode receber dois valores: *valid*, que indica que a imagem não terá um preenchimento de zeros as bordas da imagem ou *same* que indica que a imagem terá preenchimento de zeros nas bordas para a saída da convolução ter a mesma dimensão da entrada.
   - ***activation***: Trata-se da camada de ativação apresentada na seção 2.4.1.2. Nela é possível aplicar uma função de ativação nos valores do mapa de características. Diferentes funções de ativação foram explicadas e na seção 2.2.2.1.
   - ***kernel initializer***: Inicializador para a matriz de pesos do *kernel*. Padrão *default* é *glorot uniform*.
   - ***input shape***: Trata-se da dimensão da entrada de dados, o primeiro valor é a largura da imagem, o segundo valor é a altura da imagem e por último a quantidade de canais de cor

3. **Camada de *pooling***: Essa camada foi apresentada na seção 2.4.1.3, mas resumidamente, nessa camada reduzimos a dimensão da imagem e seleciona o mais importante da imagem. Na linha 11, vemos a aplicação dessa camada que contém alguns atributos, sendo o mais importante o *pool size*, que basicamente fornece as dimensões da janela usada para o *pooling*.

4. **Camada de *flatten***: Essa camada realiza o achatamento da matriz, ela transforma uma matriz multidimensional de tensores em um vetor unidimensional e é usada como a primeira camada após as camadas convolucionais.

5. **Camada Densa**: É uma camada onde cada neurônio é conectado a todos os neurônios da camada anterior. Seu funcionamento foi explicado na seção 2.3, mas resumidamente a camada realiza operações de multiplicação matricial entre os valores de entrada e os pesos associados a cada conexão, seguidas de uma função de ativação. Essa camada é responsável por aprender relações não lineares nos dados e mapear as características extraídas pelas camadas anteriores para as classes ou valores de saída desejados. Nas linhas 15 e 19 vemos sua aplicação, que conta com os seguintes atributos:

    - ***units***: É o primeiro parâmetro fornecido, trata-se do número de neurônios presentes nessa camada.
    - ***activation***: Trata-se da função de ativação presente nos neurônios. Para consultar algumas das funções de ativação existentes basta voltar na seção 2.2.2.1, onde são apresentadas as funções de ativação presente na biblioteca *Keras*.
    - ***kernel initializer***: Inicializador para os pesos das conexões dos neurônios. Padrão *default* é *glorot uniform*.

6. ***Dropout***: A camada de *Dropout* é usada para mitigar sobreajuste em redes neurais, apresentado na seção 2.3.4. Essa camada funciona desligando aleatoriamente um percentual especificado de unidades (neurônios) em cada etapa de treinamento. No caso apresentado foram desligados 15% dos neurônios.

7. **Compilação da rede**: Essa função permite especificar várias configurações importantes que afetam o processo de treinamento da rede, incluindo o otimizador, a função de perda e as métricas a serem monitoradas durante o treinamento. Uma boa descrição do funcionamento dessas configurações pode ser vista na seção 2.3. Na linha 21, vemos a função *compile()*, que conta com os seguintes atributos:

    - ***optimizer***: O otimizador é responsável por ajustar os pesos da rede neural durante o treinamento com o objetivo de minimizar a função de perda. Existem vários otimizadores disponíveis, como o *Adam*, *RMSprop*, *SGD*, entre outros. Cada otimizador possui diferentes algoritmos de atualização de peso e hiperparâmetros que afetam o desempenho e a velocidade de convergência do treinamento. Nesse caso a função *Adam* foi escolhida e trata-se de um método de descida de gradiente estocástico baseado na estimativa adaptativa de momentos de primeira e segunda ordem.
    - ***loss function***: A função de perda é usada para calcular a diferença entre as previsões da rede e os rótulos (targets) verdadeiros durante o treinamento. Ela fornece uma medida de quão bem a rede está performando em termos de sua capacidade de fazer previsões corretas. A escolha da função de perda depende do tipo de

problema que está sendo abordado, como classificação binária, classificação multiclasse ou regressão. Nesse caso a função *binary crossentropy* foi escolhida por se tratar de uma classificação binária.

- ***metrics***: As métricas são usadas para avaliar o desempenho da rede durante o treinamento, mas não são usadas para ajustar os pesos da rede. Essas métricas são calculadas durante o treinamento e fornecem informações sobre a acurácia, precisão, recall, entre outras medidas. As métricas podem ser úteis para monitorar o desempenho da rede em tempo real, mas não afetam diretamente o processo de treinamento.

Existem mais hiperparâmetros de configurações presentes nessas camadas, mas faremos apenas o estudo dessas citadas acima.

Código 4.3 – construção da rede neural

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

# Criação da rede neural
classificador = Sequential()
# Camada de convolução
classificador.add(Conv2D(128, (3, 3),
                  activation='relu',
                  input_shape=(230, 230, 1)))
# Camada de pooling
classificador.add(MaxPooling2D((2, 2)))
# Camada de achatamento
classificador.add(Flatten())
# Camada densa
classificador.add(Dense(100, activation='relu'))
# Dropout para evitar overfitting
classificador.add(Dropout(0.15))
# Camada de saída
classificador.add(Dense(1, activation='sigmoid'))
# Compilador da rede
classificador.compile(optimizer='adam',
    loss='binary_crossentropy', metrics=['binary_accuracy'])
```

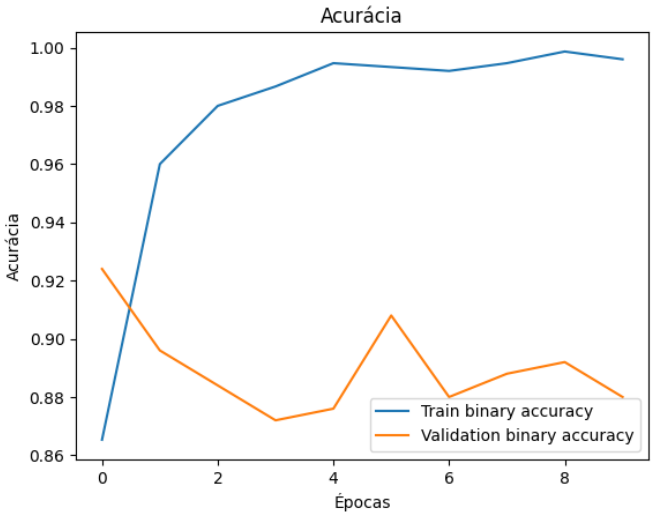
Por fim o código 4.4, mostra como o treinamento da rede é realizado. Nessa parte a rede é treinada especificando o conjunto de dados que será usado para o treinamento e o conjunto de dados que será usado para validação durante o treinamento. Além disso é possível especificar o número de épocas de treinamento, que no caso foi escolhido um treinamento de 10 épocas.

Código 4.4 – Treinamento da rede

```
classificador.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

É possível visualizar o histórico de treinamento durantes as 10 épocas, e serve para visualizar como os parâmetros estão ajustados para o problema. Na figura 4.41 vemos o gráfico de acurácia durante o treinamento das bases de treinamento e teste para o modelo apresentado. Vemos nesse gráfico que o conjunto de validação começou com uma boa estimativa por alguma inicialização aleatória de pesos e disposição dos dados, mas ao longo do treinamento esse resultado foi piorando. Já o conjunto de treinamento foi se aproximando do máximo valor de acurácia.

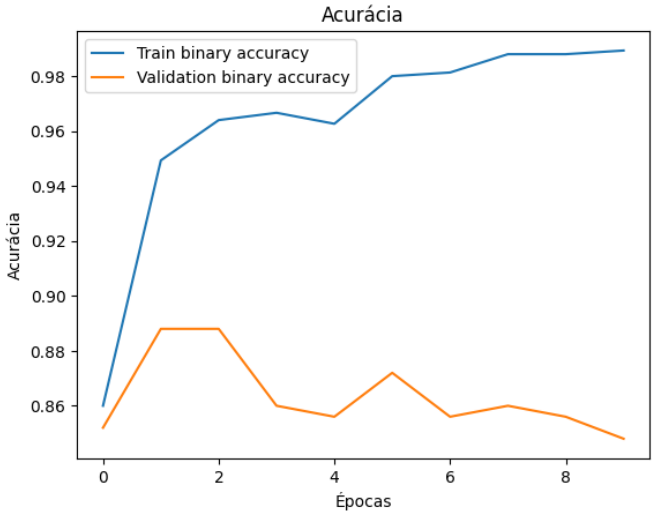
Figura 4.41 – Acurácia no treinamento do modelo simples



Fonte: Produzido pelo autor

Para fins de avaliação de topologia, realizamos o treinamento de um outra rede similar a essa porém com o triplo de camadas a mais, ou seja, existem 3 camadas de convolução e 3 camadas densas na rede, o código pode ser visto em A.1. O resultado desse treinamento pode ser visto na figura 4.42 que apresenta a acurácia entre o treinamento, das bases de treino e teste para essa nova topologia utilizada. É perceptível o resultado inferior dessa topologia em relação a anterior, mas alguns pontos devem ser analisados, não foi considerada uma validação cruzada para a validação do modelo, a base de dados do experimento 1 possui a menor quantidade de momentos de curto entre todos os experimentos e é possível que dados relevantes para o treinamento estão contidos no conjunto de validação.

Figura 4.42 – Acurácia no treinamento do modelo mais robusto



Fonte: Produzido pelo autor

4.4 *Tuning* de hiperparâmetros

Para essa parte do projeto a construção de uma rede convolucional foi uma barreira vencida, porém nosso objetivo é a criação de uma RNC com múltiplas saídas para tarefas de regressão. Essa abordagem é bem diferente da abordagem realizada que classifica as imagens em dois estados.

Uma RNC com múltiplas saídas possui uma arquitetura em que as camadas convolucionais e de *pooling* são compartilhadas entre todas as saídas, enquanto as camadas totalmente conectadas são específicas para cada saída. Isso permite que a rede extraia características relevantes em comum para todas as saídas e, ao mesmo tempo, aprenda representações específicas para cada tarefa de regressão. Por exemplo, em uma RNC com múltiplas saídas para prever as coordenadas x e y de um objeto em uma imagem, as primeiras camadas convolucionais e de *pooling* seriam compartilhadas para extrair características relevantes da imagem inteira. Em seguida, camadas totalmente conectadas seriam adicionadas para cada saída, aprendendo a relacionar as características extraídas às coordenadas x e y.

Enquanto em uma rede de classificação a função de perda mais comum é a entropia cruzada, em uma rede de regressão é utilizada uma função de perda adequada para tarefas de regressão. A função de perda mais simples e amplamente utilizada para regressão é o erro quadrático médio (MSE, do inglês *Mean Squared Error*), que calcula a média dos quadrados das diferenças entre as previsões da rede e os valores reais. A função de perda MSE é adequada para tarefas de regressão, pois penaliza mais fortemente as grandes discrepâncias entre as previsões e os valores reais, contribuindo para que a rede aprenda a fazer previsões mais precisas.

A abordagem com saídas de regressão tem uma grande diferença também em relação as saídas e interpretação de resultados. Em uma rede de classificação, a saída é uma probabilidade ou uma distribuição de probabilidade sobre as classes diferentes. A classe com a probabilidade mais alta é considerada como a previsão final da rede. Já em uma RNC de regressão, a saída é um valor numérico para cada tarefa de regressão. Por exemplo, no caso das coordenadas x e y de um objeto em uma imagem, as saídas seriam dois valores numéricos representando as coordenadas x e y. A interpretação dos resultados também é diferente. Enquanto em uma rede de classificação, a previsão final é associada a uma classe específica, em uma RNC de regressão, as previsões são interpretadas como valores contínuos que representam estimativas para a variável-alvo.

Para realizar o projeto de uma rede neural convolucional com múltiplas saídas, o primeiro passo consiste em realizar um estudo para identificar os melhores hiperparâmetros a serem definidos para o nosso problema específico. Inicialmente, iremos criar um modelo de rede neural convolucional com apenas uma saída de regressão, visando facilitar a avaliação do modelo. Nesse caso, o parâmetro mais desafiador e escolhido para o treinamento será o tamanho da poça. Além disso, é crucial ter uma forma eficiente de avaliar o desempenho do modelo. Portanto, iremos utilizar a validação cruzada para avaliar todos os testes realizados. Essa abordagem nos permite obter uma estimativa mais confiável do desempenho do modelo ao considerar diferentes partições dos dados para treinamento e validação.

Essa rede neural também terá um certo grau de automação, no sentido de que informaremos à rede um conjunto de hiperparâmetros. Em seguida, o algoritmo irá realizar testes com diferentes combinações desses hiperparâmetros e, posteriormente, classificará essas combinações com base na métrica escolhida para avaliação. Dessa forma, será possível determinar qual topologia produz o melhor desempenho de acordo com a métrica escolhida. Para isso utilizaremos uma biblioteca do *Scikitlearn* que contém uma função chamada de *GridSeachCV*, sua documentação pode ser vista em (SCIKITLEARN, 2023). Essa função basicamente faz uma pesquisa exaustiva sobre valores de hiperparâmetros especificados, o treinamento é realizado e ao final o algoritmo pontua as combinações com uma avaliação feita por meio de validação cruzada. Existem algumas alternativas a essa função, existem outras bibliotecas que desempenham o mesmo papel e isso poderia ser realizado manualmente também testando todas as combinações, porém essa biblioteca foi escolhida pela facilidade de uso.

4.4.1 Hiperparâmetros: Otimizador e Função de perda

O ideal seria fazer um *tuning* de todos os hiperparâmetros de uma só vez, mas para isso seria necessário uma máquina muito potente e consumiria um tempo de processamento muito alto. Por isso a abordagem utilizada nesse projeto é realizar essa escolha da topologia em lotes, nesse caso serão estudados os hiperparâmetros: *optimizer* e *loss function*. A escolha

do otimizador e da função de perda é fundamental para a atualização dos pesos e são os responsáveis por fazerem a rede chegar na convergência.

Para esse e todos os testes de topologias realizados nessa seção, os dados serão obtidos e organizados de uma forma diferente da apresentada até o momento. As imagens de treinamento e dados da saída de regressão serão obtidos e organizados da forma mais manual possível, para que se tenha o melhor controle possível dos dados e facilite no uso da função de *tuning*. O código referente a obtenção dos dados está disponível em A.2.

O modelo de rede neural que servirá como base para os testes das topologias, pode ser visto no código 4.5. Nesse código recebemos os dois hiperparâmetros a serem testados. Para todos os testes de tuning o processo de obtenção dos dados será feito dessa mesma forma.

Código 4.5 – Modelo de RNC para treinamento dos parâmetros: *optimizer* e *loss function*

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

def create_model(loss, optimizer):
    model = Sequential()
    model.add(Conv2D(64, (3, 3), activation='relu',
       input_shape=(100, 100, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(units = 100, activation='relu'))
    model.add(Dropout(0.15))
    model.add(Dense(units = 100, activation='relu'))
    model.add(Dropout(0.15))
    model.add(Dense(1, activation='linear'))

    model.compile(optimizer=optimizer, loss=loss)
    return model
```

No código 4.6 definimos os hiperparâmetros a serem testados na rede neural. Explicaremos o funcionamento de cada um a seguir

- ***Loss function***: Foram escolhidas as funções de erro que fazem mais sentido para problemas de regressão.

    1. *mean squared error*: Calcula o erro quadrático médio entre rótulos e previsões. Matematicamente é definido como:

$$\text{loss} = MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{prev})^2 \tag{4.1}$$

2. *mean absolute error*: Calcula o erro absoluto médio entre rótulos e previsões. Matematicamente é definido como:

$$\text{loss} = MAE = \frac{1}{n}\sum_{i=1}^{n} |y_{true} - y_{prev}| \tag{4.2}$$

3. *mean absolute percentage error*: Calcula o erro percentual médio absoluto entre rótulos e previsões. Matematicamente é definido como:

$$\text{loss} = 100 * \left(\frac{1}{n}\sum_{i=1}^{n} \left|\frac{y_{true} - y_{prev}}{y_{true}}\right|\right) \tag{4.3}$$

4. *cosine similarity*: Calcula a semelhança de cosseno entre rótulos e previsões. Matematicamente definido por:

$$\text{loss} = \frac{\vec{y_{true}} \cdot \vec{y_{prev}}}{|y_{true}||y_{prev}|} \tag{4.4}$$

- ***Optimizer***: Foram escolhidos 6 otimizadores para teste. A documentação desses otimizadores podem ser visualizadas em (KERAS, 2023).

1. *Adam*: A otimização de Adam é um método de descida de gradiente estocástico baseado na estimativa adaptativa de momentos de primeira e segunda ordem.
2. *SGD*: Otimizador de descida de gradiente (com impulso)
3. *RMSprop*: Otimizador que implementa o algoritmo RMSprop, que consiste em manter uma média móvel (descontada) do quadrado dos gradientes e divide o gradiente pela raiz dessa média.
4. *Adagrad*: Adagrad é um otimizador com taxas de aprendizado específicas de parâmetros, que são adaptadas em relação à frequência com que um parâmetro é atualizado durante o treinamento. Quanto mais atualizações um parâmetro recebe, menores são as atualizações.
5. *Adadelta*: A otimização Adadelta é um método de descida de gradiente estocástico baseado na taxa de aprendizado adaptável por dimensão, para resolver duas desvantagens: A queda contínua das taxas de aprendizado ao longo do treinamento e a necessidade de uma taxa de aprendizado global selecionada manualmente.
6. *Adamax*: A otimização Adamax é uma variante de Adam baseada na norma do infinito, é um método de otimização baseado em gradiente de primeira ordem. Devido à sua capacidade de ajustar a taxa de aprendizado com base nas características dos dados, ele é adequado para aprender processos variantes no tempo, por exemplo, dados de fala com condições de ruído alteradas dinamicamente.

A métrica selecionada para avaliação dos resultados será o *neg mean squared error* (erro quadrático médio negativo), que será utilizado para facilitar a escolha dos hiperparâmetros. A métrica será avaliada com base no valor médio, ou seja, quanto menor o valor, melhor será o desempenho do modelo, indicando um erro médio quadrático menor.

Além disso, será avaliado o desvio padrão da métrica escolhida, que indica a variabilidade do desempenho do modelo em diferentes conjuntos de dados. Um valor menor do desvio padrão indica maior confiabilidade nos resultados, ou seja, menor variabilidade no desempenho do modelo. Portanto, a avaliação levará em consideração tanto o valor médio quanto o desvio padrão da métrica escolhida para avaliar o desempenho geral do modelo.

A validação cruzada funcionará da seguinte forma, dividiremos a base de de dados em 5 partições, fazendo 5 treinamentos na rede. Cada treinamento usará 4 partições para treinamento e uma partição para validação. Ao final será feito a média do desempenho desses 5 treinamentos. Um dado interessante a ser mencionado é que para fazer esse processo serão testadas 24 combinações de hiperparâmetros, e cada combinação será treinada 5 vezes e cada treinamento terá 3 épocas.

Código 4.6 – Treinamento da rede neural para os parâmetros: *optimizer* e *loss function*

```
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import GridSearchCV
# Parâmetros a serem testados
param_grid = {
    'batch_size':[20],
    'optimizer': ['adam',
                  'SGD',
                  'RMSprop',
                  'Adagrad',
                  'Adadelta',
                  'Adamax'],
    'loss':['mean_squared_error',
            'mean_absolute_error',
            'mean_absolute_percentage_error',
            'cosine_similarity'],
    'epochs': [3]
}
# Criação do modelo de regressão
regressor = KerasRegressor(build_fn=create_model)
# Criação do tuning, k-folds = 5
grid = GridSearchCV(estimator=regressor,
                    param_grid=param_grid,
                    scoring='neg_mean_squared_error',
                    cv=5)
# Treinamento da rede
grid.fit(X_train,y_train)
```

Após os 24 testes serem realizados, selecionamos 7 combinações de hiperparâmetros

na tabela 4.3 para visualização. Sendo as 10 primeiras combinações as melhores e as duas últimas combinações as piores, para comparação. Fica claro a escolha do otimizador como o *SGD*, pelas aparições mais ao topo da tabela nas métricas de desempenho médio. Para a *loss function* escolhemos o erro quadrático médio, pelo mesmo motivo.

Tabela 4.3 – *Tuning* dos parâmetros: *optimizer* e *loss function*

	Desempenho	**Desvio padrão**	***optimizer***	***loss function***
1	0.0038	0.001	SGD	mean squared error
2	0.0043	0.0013	Adagrad	mean absolute error
3	0.0044	0.0009	Adam	mean squared error
4	0.0045	0.0028	Adamax	mean absolute error
5	0.0046	0.0018	SGD	mean absolute error
6	0.0047	0.0033	Adamax	mean absolute percentage error
7	0.0048	0.0018	Adam	mean absolute error
8	0.0051	0.0012	adagrad	mean squared error
9	0.0051	0.0019	adamax	mean squared error
10	0.0054	0.0035	Adam	mean absolute percentage error
23	0.2800	0.0395	Adagrad	cosine similarity
24	3.6806	3.3389	Adamax	cosine similarity

4.4.2 Hiperparâmetros: Número de camadas, número de *kernels* e neurônios

A ideia por trás desse teste é selecionar em um primeiro momento a quantidade de número de camadas convolucionais e densas para a rede. No segundo momento o objetivo é selecionar números de *kernels* e neurônios para as camadas obtidas no primeiro momento. A decisão de escolha da melhor topologia para o projeto será a que vai oferecer o menor custo operacional, ou seja, caso se obtenha resultados semelhantes entre duas topologias, a topologia selecionada será a mais enxuta.

O número de camadas em uma rede neural é conhecido como a profundidade da rede. Adicionar camadas adicionais permite que a rede aprenda representações mais complexas dos dados. No entanto, adicionar camadas em excesso pode levar ao *overfitting*, onde a rede se ajusta demais aos dados de treinamento e não generaliza bem para novos dados.

A importância do ajuste do número de camadas reside em encontrar o equilíbrio certo entre a capacidade de aprendizado da rede e a capacidade de generalização. Para problemas mais simples, uma rede com poucas camadas pode ser suficiente para obter bons resultados. Já para problemas mais complexos e com dados mais desafiadores, é possível explorar a adição de camadas intermediárias para extrair representações mais abstratas e discriminativas.

A quantidade de neurônios em cada camada é um fator importante que determina a capacidade da rede neural para modelar relações complexas nos dados. Aumentar o número de neurônios em uma camada aumenta a capacidade de representação da rede, permitindo

que ela aprenda relações mais detalhadas e não lineares. No entanto, um número excessivo de neurônios pode levar ao *overfitting*.

Para realizar o teste de hiperparâmetros, conduzimos uma análise para determinar o número ideal de camadas para o nosso projeto. A estrutura do modelo de rede é composta por um laço de repetição que adiciona camadas de convolução e outro laço de repetição que adiciona camadas densas. Serão testados cinco valores diferentes para o número de camadas, tanto nas camadas de convolução quanto nas camadas densas, o código pode ser visto em A.3.

A tabela 4.4 fornece os 5 melhores resultados desses testes de combinações. É possível ver que nas primeiras combinações, as topologias melhor classificadas são as abordagens com o maior número de camadas, isso significa que os testes não foram conclusivos. O melhor a se fazer é variar o número de camadas para valores maiores, obter os resultados para comparação e verificar se esses valores maiores são os melhores de fato. No caso, esses testes foram realizados e observa-se que conforme o número de camadas aumenta não existe uma melhora na estimativa do modelo

é possível ver também que a combinação de número 3, que emprega um número de camadas intermediário, tem um desempenho muito próximo da combinação 1. Por isso escolhemos essa topologia para nossa rede, que apresenta 4 camadas de convolução e 4 camadas densas.

Tabela 4.4 – *Tuning* dos parâmetros: Número de camadas

	Desempenho	**Desvio padrão**	**N° camadas de conv.**	**N° camadas densas**
1	0.0032	0.0013	5 camadas	4 camadas
2	0.0032	0.0014	5 camadas	5 camadas
3	0.0033	0.0016	4 camadas	4 camadas
4	0.0033	0.0013	5 camadas	3 camadas
5	0.0033	0.0015	4 camadas	5 camadas

O próximo passo é então variar os números de *kernels* e neurônios nessas camadas. Para as camadas de convolução será seguida uma lógica de aumentar o número que *kernels* a cada camada criada. Isso será feito por que à medida que as camadas convolucionais vão se aprofundando, elas tendem a aprender representações mais abstratas e complexas das características presentes nas imagens. Aumentar o número de *kernels* permite que a rede capture uma gama maior de informações e detalhes, pois cada *kernel* pode aprender a detectar diferentes tipos de padrões. Além disso, é importante notar que as primeiras camadas de convolução geralmente aprendem características mais simples, como bordas, texturas e gradientes. À medida que a informação passa pelas camadas, as características aprendidas se tornam mais complexas e abstratas, relacionadas a objetos específicos ou a combinações de objetos. Isso garante a rede uma melhor capacidade de generalização, faz a rede lidar melhor com variações e aumenta a capacidade de representação.

Em relação ao número de neurônios nas camadas densas, não tem nenhuma lógica a ser seguida. No código A.4 vemos os valores de teste iniciais para ter um direcionamento sobre qual direção seguir em relação aos valores testados. Com esses testes obtemos a tabela 4.5, que apresenta as 5 melhores combinações.

Vemos que a rede se comporta melhor para valores mais baixos, uma vez que os valores altos de neurônios e *kernels* fazem o erro quadrático médio da rede aumentar. Escolheremos então a segunda combinação apresentada, que contém a sequência de 16, 32, 64 e 128 *kernels* nas camadas de convolução e 30 neurônios nas camadas densas. Não iremos fazer outros testes com valores menores ainda, pois o objetivo é ter um direcionamento de que caminho seguir na implementação da rede final, nesse caso descobrimos que valores menores são mais adequados ao problema.

Tabela 4.5 – *Tuning* dos parâmetros: quantidade de *kernels* e neurônios

	Desempenho	**Desvio padrão**	**N° de *kernels***	**N° de neurônios**
1	0.0032	0.0010	[100, 200, 300, 400]	[30, 30, 30, 30]
2	0.0032	0.0011	[16, 32, 64, 128]	[30, 30, 30, 30]
3	0.0033	0.001	[100, 200, 300, 400]	[100, 100, 100, 100]
4	0.0033	0.0011	[8, 16, 32, 64]	[30, 30, 30, 30]
5	0.0033	0.0011	[8, 16, 32, 64]	[400, 300, 200, 100]

4.4.3 Hiperparâmetros: funções de ativação e inicialização de pesos

Nessa seção o objetivo é encontrar as melhores funções de ativação e encontrar a melhor forma de se iniciar os pesos de ambas camadas. As funções de ativação são responsáveis por introduzir não-linearidades na saída dos neurônios. Elas desempenham um papel fundamental na capacidade da rede neural de modelar relações complexas nos dados. Já a inicialização dos pesos das conexões entre os neurônios é uma etapa crítica no treinamento de redes neurais. Inicializar os pesos corretamente pode ajudar a evitar problemas como o desvanecimento ou a explosão do gradiente e a melhorar a convergência do modelo.

Para isso testaremos as seguintes funções de ativação: **Linear**, **Sigmóide**, **Tangente hiperbólica**, **ReLU**, **SoftPlus**, **ELU**, **SELU**. Todas essas funções foram apresentadas na seção 2.2.2.1.

Agora para a inicialização de pesos, definido como *kernel initializer* dentro da biblioteca *Keras*, será apresentada algumas formas de inicialização disponíveis.

1. *Glorot normal*: O inicializador normal Glorot, também chamado de inicializador normal Xavier, amostra os pesos de forma aleatória a partir de uma distribuição normal com média zero e uma variância [1] específica. A variância é calculada com base na

[1] A variância é uma medida estatística que descreve a dispersão dos valores em torno da média de uma distribuição. Em termos simples, ela indica o quão espalhados os dados estão.

média do número de conexões de entrada e saída (GLOROT; BENGIO, 2010).

$$var = \frac{2}{fan_{in} + fan_{out}} \tag{4.5}$$

Onde fan_{in} é o número de unidades de entrada no tensor de peso e fan_{out} é o número de unidades de saída

2. *Glorot uniforme*: O inicializador uniforme Glorot, também chamado de inicializador uniforme Xavier, amostra os pesos de forma aleatória a partir de uma distribuição uniforme centrada em zero e com uma variância específica. A variância é calculada com base na média do número de conexões de entrada e saída para um neurônio específico (GLOROT; BENGIO, 2010).

$$var = \frac{6}{fan_{in} + fan_{out}} \tag{4.6}$$

3. *He normal*: O inicializador He Normal amostra os pesos de forma aleatória a partir de uma distribuição normal com média zero e uma variância específica. A variância é calculada apenas com base no número de conexões de entrada para um neurônio específico (HE et al., 2015).

$$var = \frac{2}{fan_{in}} \tag{4.7}$$

4. *He uniforme*: O inicializador He Uniforme amostra os pesos de forma aleatória a partir de uma distribuição uniforme centrada em zero e com uma variância específica. A variância é calculada apenas com base no número de conexões de entrada para um neurônio específico (HE et al., 2015).

$$var = \frac{6}{fan_{in}} \tag{4.8}$$

5. *Random Uniforme*: Inicializador que gera tensores aleatórios com distribuição uniforme.

6. zeros: Inicializador que gera tensores inicializados em 0.

7. *truncated normal*: Inicializador que gera uma distribuição normal truncada.

Na tabela 4.6 vemos os melhores parâmetros para cada camada e no código A.5. O objetivo desse estudo não é diminuir o valor do erro sempre, mas ter certeza dos melhores hiperparâmetros e comprovar com dados as escolhas feitas na rede. Por isso a escolha dos hiperparâmetros nesse caso vai ser com base na frequência de aparição nos melhores resultados da rede, uma vez que os valores de desempenho são bem próximos. Por isso para as camadas convolucionais a função de ativação escolhida será a ReLU e o inicializador de pesos será o *Glorot normal*. Para as camadas densas a função de ativação escolhida foi a tangente hiperbólica e para iniciar os pesos foi a *Truncated Normal*. Os outros parâmetros apresentados, que não estão na tabela, geraram um erro quadrático médio mais alto, mas ainda assim os resultados para quase todos os parâmetros estudados foram baixos.

Tabela 4.6 – *Tuning* dos parâmetros: Funções de ativação e inicializadores de pesos

	Desempenho	**Desvio padrão**	**Camada de convolução**	**Camada densa**
1	0.0031	0.0011	tanh e he uniform	tanh e truncated normal
2	0.0031	0.0012	tanh e random uniform	relu e truncate normal
3	0.0031	0.0012	relu e glorot normal	elu e truncate normal
4	0.0031	0.0012	tanh e glorot normal	tanh e truncated normal
5	0.0031	0.0012	relu e glorot normal	tanh e truncate normal
6	0.0031	0.0012	relu e random uniform	tanh e truncated normal
7	0.0032	0.0012	relu e random uniform	elu e truncate normal
8	0.0032	0.0012	relu e glorot normal	relu e truncate normal
9	0.0032	0.0011	tanh e random uniform	tanh e truncated normal
10	0.0032	0.0012	relu e he uniform	tanh e truncated normal

4.4.4 Hiperparâmetros: *Batch size* e *dropout*

Duas das configurações importantes a serem consideradas são os hiperparâmetros de *dropout* e o tamanho do lote (*batch size*).

A camada de *dropout* é uma técnica regularizadora usada em redes neurais para evitar o *overfitting*. Ela ajuda a reduzir a dependência excessiva entre os neurônios, permitindo que diferentes subconjuntos de neurônios sejam ativados durante o treinamento. A importância do ajuste reside em encontrar um equilíbrio entre a regularização e a capacidade de aprendizado da rede. Um valor muito baixo de *dropout* pode não fornecer uma regularização adequada, enquanto um valor muito alto pode prejudicar a capacidade do modelo de aprender informações relevantes.

O tamanho do lote (*batch size*) refere-se à quantidade de exemplos de treinamento usados em cada passo de atualização dos pesos durante o treinamento da rede neural. É uma configuração importante que influencia o tempo de treinamento, a estabilidade do treinamento e a utilização eficiente dos recursos computacionais.

Tamanhos de lote maiores podem fornecer estimativas de gradiente mais estáveis, reduzindo a variância dos gradientes calculados em cada atualização de peso. No entanto, tamanhos de lote menores podem ajudar a evitar mínimos locais e a melhorar a generalização, permitindo atualizações de peso mais frequentes.

Na tabela 4.7, vemos as melhores e piores combinações para os hiperparâmetros escolhidos para teste, os valores que foram testados podem ser vistos no código A.6. Para a escolha da combinação fica claro que valores maiores de tamanho do lote deixam o resultado é ruim, assim como valores maiores para o *dropout* também piora os resultados. Por isso a combinação escolhida para a rede será a 1, que promove um valor intermediário entre as combinações e possui um desempenho bom.

Tabela 4.7 – *Tuning* dos parâmetros: Funções de ativação e inicializadores de pesos

	Desempenho	**Desvio padrão**	***Batch size***	***dropout***
1	0.0031	0.0011	20	0.2
2	0.0031	0.0013	10	0.1
3	0.0031	0.0013	10	0.15
4	0.0031	0.0012	32	0.1
5	0.0039	0.0012	64	0.3
6	0.0107	0.0030	128	0.2

4.5 Treinamento do modelo final

Após todo o processo de *tuning* de parâmetros, chega o passo crucial para o desenvolvimento do projeto. Durante o desenvolvimento do modelo, é comum realizamos várias iterações, explorando diferentes combinações de hiperparâmetros, arquiteturas de rede e técnicas de otimização. Cada iteração envolve treinamento e avaliação do modelo, ajustando os parâmetros com base nos resultados obtidos. Essa experimentação iterativa permite aprender com os erros e sucessos, refinando gradualmente a rede neural.

Uma das considerações mais importantes é a escolha da arquitetura final da rede neural. Com base nos resultados das iterações anteriores, é possível identificar a arquitetura que melhor se adequa ao problema em questão. Isso pode incluir o número e o tipo de camadas, a presença de camadas compartilhadas ou ramificadas, e a incorporação de técnicas como *dropout*. No nosso caso combinamos algumas arquiteturas já conhecidas, como por exemplo a *LeNet*, *AlexNet* e *VGGNet*

Além da arquitetura, os hiperparâmetros também desempenham um papel crucial no desempenho da rede neural. Durante as iterações, foram testados diferentes valores para taxa de aprendizado, tamanho do lote, número de épocas e outros parâmetros relevantes. Com base nos resultados obtidos, é necessário atribuir os hiperparâmetros finais que proporcionam o melhor desempenho. Esses valores serão aqueles escolhidos com base nas melhores configurações encontradas durante a experimentação. Para facilitar, listaremos os hiperparâmetros escolhidos após os testes de *tuning*.

- **Função de perda**: Erro quadrático Médio
- **Otimizador**: SGD
- **Número de camadas convolucionais**: 4
- **Número de camadas densas**: 4
- **Número de filtros convolucionais**: 16, 32, 64, 128
- **Número de neurônios** : 30, 30, 30, 30

- **Função de ativação convolucional**: Relu
- **Função de ativação densa**: Tanh
- **Inicializador de pesos da camada convolucional**: Glorot Normal
- **Inicializador de pesos da camada densa** : Truncate normal
- ***Batch Size***: 20
- ***Dropout***: 0.2

Agora é o momento de treinar o modelo usando todos os dados disponíveis. No caso nossa base completa temos 6000 imagens, e dividiremos os conjuntos de em treino com 5000 imagens, teste com 500 imagens e validação com 500 imagens. O conjunto de treinamento é usado diretamente no treinamento da rede, o conjunto de validação é usado para avaliar a rede e verificar resultados ao longo do treinamento, e o conjunto de teste é usado após o treinamento para obter as métricas reais do treinamento com um conjunto que não teve contato algum com a rede.

A rede neural final é treinada com o objetivo de aprender padrões, representações e correlações nos dados que permitam fazer previsões precisas. Para esse passo realizaremos a confecção de duas redes neurais diferentes, uma rede completa com múltiplas saídas e uma rede com uma única saída de regressão que testaremos para cada valor numérico desejado.

4.5.1 Rede com múltiplas saídas

A construção de uma rede neural com múltiplas saídas é uma abordagem poderosa que permite resolver problemas complexos que envolvem várias tarefas ou previsões simultâneas. Essa arquitetura é particularmente útil quando há dependências entre as diferentes saídas ou quando as saídas estão relacionadas de alguma forma. Existem diferentes maneiras de construir uma rede neural com múltiplas saídas, e a escolha da arquitetura depende das características específicas do problema que você está enfrentando.

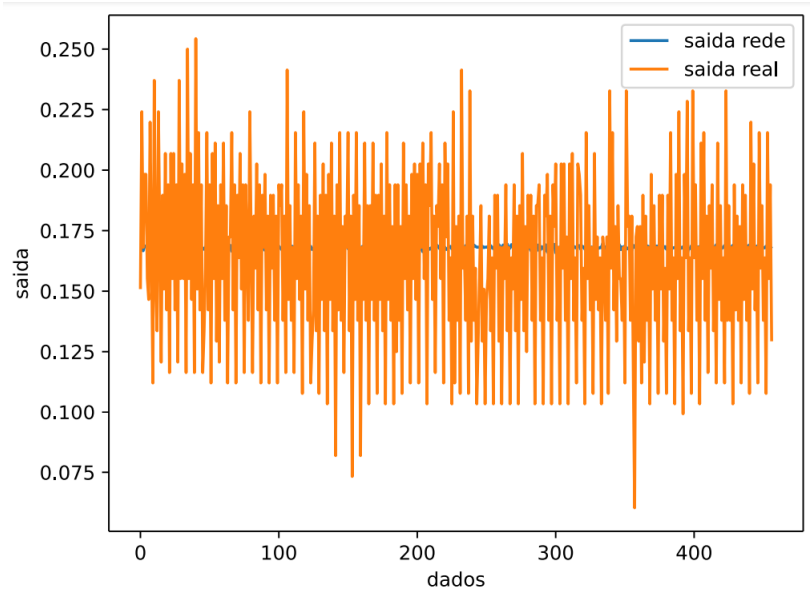
Uma abordagem comum é usar camadas compartilhadas nas etapas iniciais da rede. Essas camadas são responsáveis por extrair características relevantes dos dados de entrada e criar representações intermediárias que serão compartilhadas entre as diferentes saídas. Isso permite que a rede aprenda padrões gerais e promove a eficiência de aprendizado, uma vez que as características comuns são aprendidas em conjunto. Após as camadas compartilhadas, cada saída tem sua própria ramificação, que consiste em camadas específicas para a tarefa correspondente. Essas camadas são responsáveis por processar as representações intermediárias e gerar as previsões para cada saída individualmente.

Testamos essa abordagem e o código referente a essa arquitetura pode ser visto em A.7. Esse código consiste em usar a mesma arquitetura para várias saídas, sendo que as

ramificações se encontram apenas nas camadas de saída da rede, ou seja, para cada valor desejado na saída, existe uma ramificação que contém uma camada totalmente conectada com um neurônio de ativação linear.

Durante nossas observações, notamos uma tendência na rede em gerar como saída a média dos valores correspondentes. Por exemplo, ao analisar a distância entre o início da poça e a borda esquerda da imagem, percebemos que essa medida pode variar de 20 a 50 *pixels*, com uma média de 37 *pixels*. Surpreendentemente, descobrimos que a saída gerada para todas as imagens tende a ser em torno de 37 *pixels*. Para ilustrar esse comportamento, na figura 4.43, apresentamos um gráfico que compara a saída real ou esperada com a saída gerada após o processamento do conjunto de teste pela rede treinada. Constatamos que o resultado é insatisfatório e não atende às nossas expectativas nesse caso específico. Portanto, optamos por não seguir essa abordagem e decidimos construir uma rede separada para cada saída da rede original.

Figura 4.43 – Distribuição da posição da poça de soldagem na imagem (real x previsão)



Fonte: Produzido pelo autor

4.5.2 Rede com uma saída

Para essa abordagem, adotaremos a mesma estrutura utilizada nos testes de ajuste de parâmetros, com os mesmos hiperparâmetros mencionados no início desta seção (ver seção 4.5). O objetivo da rede é gerar uma saída numérica referente a quatro dimensões da imagem, que são: Largura da poça; Posição da poça na imagem; Posição do arame na imagem; Altura do Arame. Além disso, essa arquitetura deve ser capaz de classificar a imagem em dois momentos: momento de curto-circuito e momento onde não ocorre curto-circuito. O código

A.8 mostra o algoritmo completo, desde a importação dos dados e bibliotecas até a avaliação dos resultados. Esse código será usado para todas as saídas, iremos analisar os resultados gerados para cada uma delas a seguir.

Avaliaremos os resultados do conjunto de teste na saída da rede em comparação com a saída esperada. A avaliação será por meio de 3 métricas comumente usadas para avaliar o desempenho de modelos em problemas de regressão, que são MSE, MAE e R2.

O MSE é uma medida de erro médio quadrático. Ele calcula a média dos quadrados das diferenças entre as previsões do modelo e os valores reais. Quanto menor o valor do MSE, melhor o desempenho do modelo.

O MAE é uma medida de erro médio absoluto. Ele calcula a média das diferenças absolutas entre as previsões do modelo e os valores reais. Assim como o MSE, o MAE mede a qualidade das previsões, mas considera as diferenças absolutas em vez dos quadrados.

O R2, também conhecido como coeficiente de determinação, é uma medida estatística que indica a proporção da variância dos valores de destino que é explicada pelas previsões do modelo. O R2 varia de 0 a 1, onde 1 indica um ajuste perfeito e 0 indica que o modelo não explica a variabilidade dos dados. O R2 é uma métrica útil para avaliar a capacidade do modelo em capturar as variações dos dados e pode ajudar a comparar diferentes modelos.

Na tabela 4.8 vemos a relação de todas as saídas de regressão e a avaliação segundo as métricas apresentadas. Na figura 4.44 vemos as representações das curvas dos valores de saída da rede quando o conjunto de testes é submetido e a saída esperada que são os valores de rotulação do mesmo conjunto.

Tabela 4.8 – Resultados da rede neural em relação a 3 métricas de avaliação

Dimensão	**MAE**	**MSE**	**R2**
Posição da poça	0.0277	0.00114	0.114
Largura da poça	0.034	0.0018	0.29
Posição do Arame	0.0083	9.9e-05	0.21
Altura do arame	0.024	0.0012	0.39

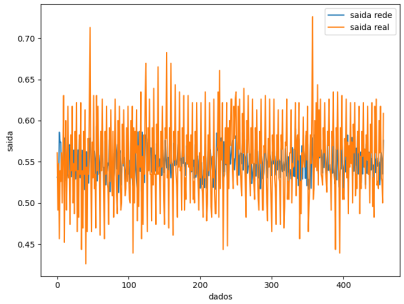
4.5.2.1 Momento de curto-circuito

Para a classificação dos momentos de curto, foi usado como métrica de avaliação a acurácia da saída da rede, que avalia a taxa de acerto da classificação. Nesse caso para a mesma rede a acurácia do conjunto de teste foi de 0.9737, ou seja, a rede foi capaz de acertar 97% das classificações de momento de curto e não curto no conjunto de testes. Esse resultado de classificação foi o melhor encontrado até agora, comparando com os resultados obtidos na seção 4.3.

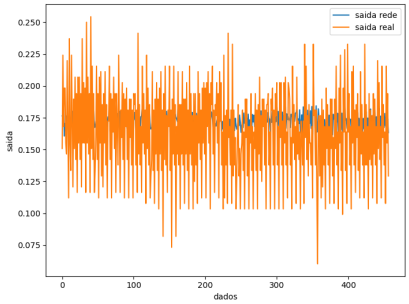
Na figura 4.45 é possível ver o mesmo gráfico gerado na avaliação das outras saídas a título de comparação. Vemos que as linhas quase que se sobrepõe.

Figura 4.44 – Comparação entre valores reais e predições

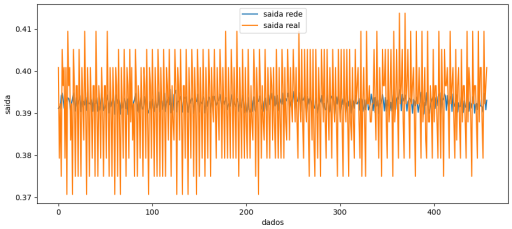
(a) Largura da poça



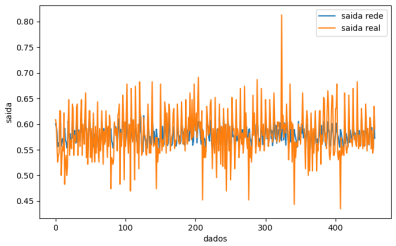
(b) Posição da poça



(c) Posição do Arame

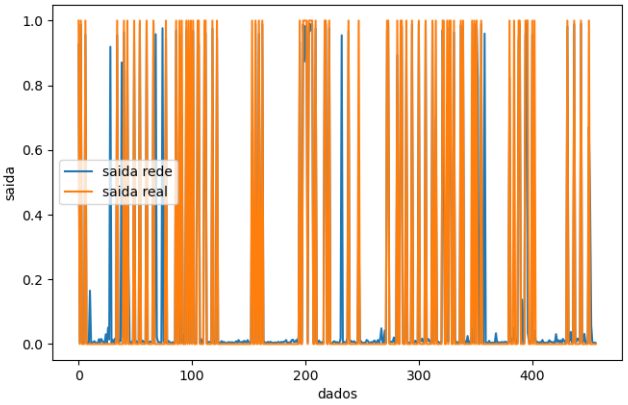


(d) Altura do arame



Fonte: Produzidas pelo autor

Figura 4.45 – Classificação dos momentos de curto-circuito (real x previsão)



Fonte: Produzido pelo autor

4.5.3 Mudança na saída

Os resultados apresentam erros baixos, mas isso é considerando o conjunto de todos os resultados juntos, a rede minimizou o erro de todo o conjunto das saídas, e conteve as saídas numérias em valores dentro de uma faixa média de valores previstos. A métrica R2 mostra que os valores não estão próximos o suficiente do desejável e o modelo estuda e criado não se aplica a variabilidade dos dados.

É preciso pensar em uma outra arquitetura para conseguir aplicar os dados de forma que a saída seja suficientemente boa e que dê um saída mais assertiva sobre as dimensões dos parâmetros de soldagem.

Uma possível abordagem para lidar com isso é alterar o tipo de saída da rede. Até agora, utilizou-se valores percentuais normalizados para representar a posição dos parâmetros na imagem. Em outras palavras, as dimensões rotuladas na imagem foram transformadas em valores de porcentagem, variando de 0 a 1. Por exemplo, um valor de 100 pixels para a largura da poça foi calculado da seguinte maneira: $100/232 = 0{,}43$. A ideia agora seria mudar esse valor de 0,43 para 43 e testar o funcionamento da rede para valores que variam de 0 a 100.

Foi realizado todos os testes de parâmetros novamente com os mesmos hiperparâmetros testados na seção 4.4, e o resultado das mesmas tabelas apresentadas na seção podem ser vistas no apêndice B. Os resultados variaram bastante após essa abordagem, mas o resultado final teve os mesmos resultados obtidos e em alguns casos o valor da saída da rede comparado pela métrica R2 foi até pior.

4.6 YOLOv8

O objetivo principal do nosso projeto era construir e treinar uma rede neural convolucional capaz de identificar parâmetros no processo de soldagem. Especificamente, queríamos alcançar duas tarefas: classificar o momento em que ocorre a soldagem e identificar as dimensões dos parâmetros envolvidos nesse processo.

Felizmente, existem alguns *frameworks* disponíveis que são projetados exatamente para realizar esse tipo de função. Decidimos aproveitar essa abordagem e utilizar um desses *frameworks*, combinando-o com a técnica de *transfer learning*. Essa técnica nos permite treinar uma rede pré-treinada, que já possui uma arquitetura estabelecida e é capaz de identificar dimensões nos parâmetros de soldagem.

Dessa forma, é possível aproveitar o conhecimento prévio da rede pré-treinada e adaptá-la para o nosso problema específico. Isso nos permite obter melhores resultados, aproveitando a capacidade de detecção de padrões já desenvolvida pela rede pré-treinada.

Para realizar o *transfer learning*, inicializamos a rede neural com pesos pré-treinados em um grande conjunto de dados. Esses pesos pré-treinados contêm informações valiosas sobre características gerais das imagens, aprendidas durante o treinamento em um conjunto de dados diversificado.

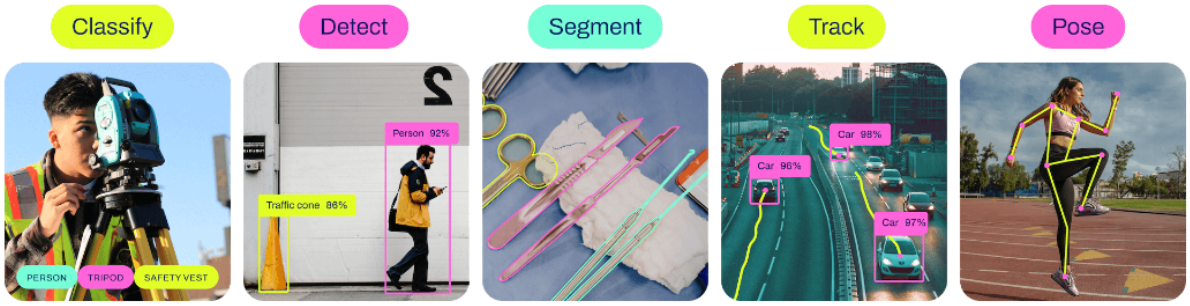
Em vez de treinar a rede do zero, aproveitamos os pesos pré-treinados como uma inicialização inicial. Em seguida, continuamos o treinamento da rede em um conjunto de dados específico para a nova tarefa que desejamos resolver. No caso do projeto usaremos as imagens do processo de soldagem juntamente com a rotulação desses dados. A ideia é

permitir que a rede ajuste suas características aprendidas anteriormente para se adaptar ao novo problema.

Dessa forma a rede pré-treinada que usaremos é o YOLOv8. O YOLO é um *framework* de detecção de objetos que aborda o problema como um desafio de regressão, no qual uma única passagem pela rede neural é capaz de prever a localização e a classe dos objetos em uma imagem. Sua arquitetura profunda baseia-se em camadas convolucionais, normalização, ativação e *pooling*, permitindo a extração de características relevantes para a detecção. Esse *framework* foi discutido mais profundamente na seção 3.2.6.

Como mencionado, usaremos o YOLOv8, criado pela *Ultralytics* (JOCHER; CHAURASIA; QIU, 2023). Esse *framework* tem a capacidade de classificar, segmentar, detectar, rastrear e estimar a pose, conforme mostrado na figura 4.46. No caso do projeto usaremos a detecção para estimar os parâmetros do processo de soldagem.

Figura 4.46 – Processos de visão computacional usando YOLOv8



Fonte: (JOCHER; CHAURASIA; QIU, 2023)

4.6.1 Preparação dos dados

Para o uso do YOLOv8 foi preciso preparar os dados em um formato especifico, isso consistia em mudar a estrutura de pastas e o formato com que os dados foram rotulados. Esse formato especifico do YOLOv8 consiste em ter para cada uma das imagens do projeto um arquivo .txt com as informações da caixa delimitadora e das suas respectivas coordenadas relativas na imagem.

O nome do arquivo .txt deve ser igual e relativo a imagem que está sendo rotulada, ou seja, para a imagem **Img679.bmp** o arquivo de texto deve ser **Img679.txt**. Cada linha do arquivo .txt corresponde a uma detecção de objeto e segue o seguinte padrão:

<classe1> <x1 center> <y1 center> <width1> <height1>

<classe2> <x2 center> <y2 center> <width2> <height2>

Onde:

- <classe>: O nome ou o ID da classe do objeto detectado. Cada classe é representada por um número inteiro
- <x center>: A coordenada x do centro da caixa delimitadora, normalizada em relação à largura da imagem.
- <y center>: A coordenada y do centro da caixa delimitadora, normalizada em relação à altura da imagem.
- <width>: A largura da caixa delimitadora, normalizada em relação à largura da imagem.
- <height>: A altura da caixa delimitadora, normalizada em relação à altura da imagem.

Para isso foi preciso realizar uma transformação no tipo de rotulação realizado anteriormente no projeto. No código A.11 é possível ver a forma que os arquivos de texto relativos a cada imagem foram criados. Além disso foi preciso criar um arquivo auxiliar do tipo .yaml, com o objetivo de listar as classes de rotulagem e os caminhos relativos de cada subconjunto de dados.

4.6.2 Treinamento do YOLOv8

Para o treinamento do modelo usamos como base o modelo mais básico do YOLOv8, o YOLOv8 nano, que consiste em 225 camadas, 3011238 parâmetros e 3011222 gradientes. Esse é o menor modelo de YOLOv8 e escolhemos ele justamente por ter a menor robustez, pois essa característica garante uma vantagem em ambientes industriais, devido a velocidade e tamanho da rede.

Realizamos o treinamento com 50 épocas, um *batch size* de 16, uma taxa de aprendizado de 0.01. Os demais parâmetros como otimizadores, *dropout* e tantos outros estudados foram usados os valores de *default* no próprio modelo, que podem ser vistos na documentação do *framework* (JOCHER; CHAURASIA; QIU, 2023).

No código 4.7 é possível ver o modo básico de treinamento para realizar um *transfer learning*, nesse exemplo usamos o modo de detecção para realizar o treinamento, o tamanho da imagem escolhido é de 256 por conta de uma restrição da biblioteca onde o tamanho da imagem deve ser múltiplo de 32, para se adequar ao número de camadas. Usamos o subconjunto de dados com 5 mil imagens para o treinamento do modelo e avaliamos com um conjunto de validação de 500 imagens.

Código 4.7 – Treinamento do YOLOv8 por meio de *transfer learning*

```
!yolo task=detect mode=train model=yolov8n.pt
   data=/content/drive/MyDrive/datasets/Yolo/data.yaml epochs=50
   imgsz=256 batch=16 lr0=0.01
```

4.6.3 Avaliação do modelo

O *framework* trás uma série de resultados do treinamento, que compara funções de perda da classificação e da regressão relativa ao tamanho das caixas delimitadoras ao longo das 50 épocas de treinamento. Separamos esses dados na figura A.50 em anexo. Em anexo é possível ver também um exemplo de um conjunto de imagens que foram submetidas ao treinamento com as respectivas caixas delimitadoras e classes. Na figura A.52 é possível ver o conjunto de imagens de validação usadas no treinamento, devidamente rotuladas. Já na figura A.53 é possível ver as mesmas imagens rotuladas depois de passarem pela rede. Vemos que os resultados são bem satisfatórios e apresentam um bom resultado.

O treinamento da rede proporciona para a rede a realização de duas tarefas, uma é a regressão para dimensionar as caixas delimitadoras e a outra é a classificação dos objetos no imagem. Falando da classificação, um dado que obtemos após o treinamento do modelo é que a rede foi capaz de identificar 99,5% das vezes o arame e a poça de soldagem com mais de 50% de confiança no conjunto de validação.

A fim de conferir a saída mostraremos o resultado da rede um uma imagem, comparando os resultados obtidos com os valores esperados. Dessa forma uma imagem que não participou do treinamento da rede será submetida ao modelo e analisaremos todas as saídas dessa predição. A imagem escolhida pode ser visualizada na figura 4.47a e a saída do modelo de treinamento podem ser vistos na figura 4.47b. Analisando a saída é possível ver que os resultados são bons e as caixas delimitadoras parecem comportar com uma certa precisão as medidas da poça e do arame. O código 4.8 foi usado para realizar essa predição, onde usamos a rede com os pesos treinadas para realizar a predição da imagem

Código 4.8 – Realiza a predição de uma imagem com a rede treinada

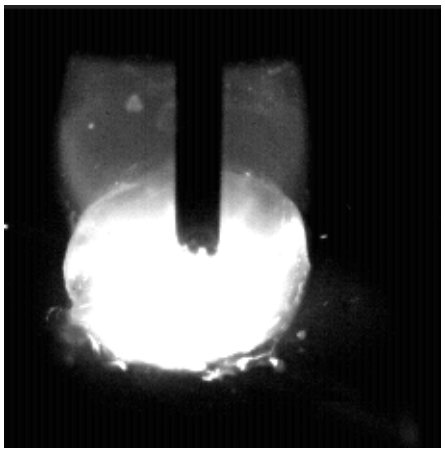
```
from ultralytics import YOLO
model = YOLO('/path/redetreinada.pt')
results = model.track(source='/path/Imgem.bmp', conf=0.7,
   save=True)
```

Os valores esperados e as saídas encontradas para essa imagem podem ser vistas na tabela 4.9, os valores foram transformados para as dimensões correspondentes da imagem para facilitar a visualização. Vemos que os valores da rotulação são bem próximos do esperado e visualmente os resultados parecem estar perfeitos.

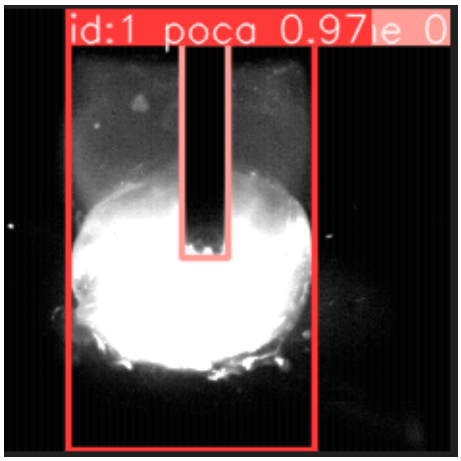
O conjunto de testes inteiro foi submetido a rede neural para a predição dos valores. Com o conjunto de testes devidamente rotulado pela rede, analisaremos as métricas de saída

Figura 4.47 – Comparação entre valores reais e predições

(a) Imagem original



(b) Imagem após passar pela rede



Fonte: Produzidas pelo autor

Tabela 4.9 – Comparação entre as dimensões reais e dimensões de saídas da rede neural

Dimensão	Valor real	Valor de previsão
Centro da poça (eixo x)	97	97.41
Largura da poça	117	127.85
Centro do arame (eixo x)	104.4	104.60
Largura do arame	23	24.41
Centro do arame (eixo y)	62.5	64.60
Altura do arame	125	129.20

da mesma forma feita na seção 4.5.2. Usaremos então as métricas MSE, MAE e R2 para avaliar o desempenho da saída da rede para 5 saídas importantes, que são: Posição de poça, que é dado pelo valor central da Poça no eixo x da imagem. Largura da poça. Posição do arame no eixo x, que é dado pelo valor central da arame no eixo x da imagem. Posição do arame no eixo y, que é dado pelo valor central da arame no eixo y da imagem. Altura do Arame. O código A.9 mostra como é realizada a predição em todo o conjunto de dados de teste, e como os dados podem ser armazenados em uma tabela.

A tabela 4.10 mostra os resultados de cada saída em relação as métricas. Vemos que para as métricas de erro quadrático médio e erro absoluto médio os valores se mantêm baixos e para a métrica R2, que para nós nesse momento é a métrica mais importante, observa-se que seu valor está bem mais próximo de 1, isso indica que o modelo está mais adequado a variabilidade dos dados. Esse é um resultado interessante e importante para essa saída e era esperado dado a qualidade da saída observada da rede.

Na figura 4.48 é possível ver os gráficos das métricas apresentadas na tabela 4.9. Para cada dimensão foi gerado um gráfico que mostra duas curvas, uma é formada pelo conjunto de valores da saída da rede e a outra representa o conjunto de valores referentes a saída esperada da rede. Vemos que as curvas são muito parecidas, diferente dos resultados

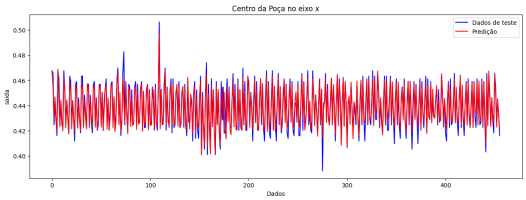
Tabela 4.10 – Resultados da rede neural em relação a 3 métricas de avaliação

Dimensão	MAE	MSE	R2
Posição da Poça	0.0039	3.535e-05	0.89
Largura da Poça	0.0105	3.1e-04	0.87
Posição do Arame no eixo X	0.00168	7.29e-06	0.93
Posição do Arame no eixo Y	0.0035	3.59e-05	0.88
Altura do Arame	0.0071	1.42e-04	0.89

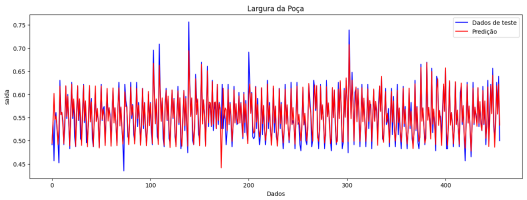
anteriores.

Figura 4.48 – Comparação entre valores reais e predições

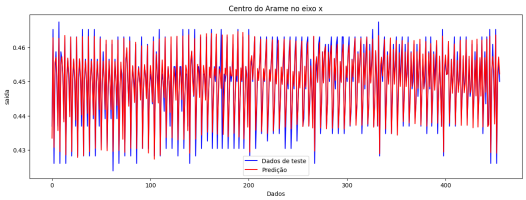
(a) Posição da Poça de soldagem



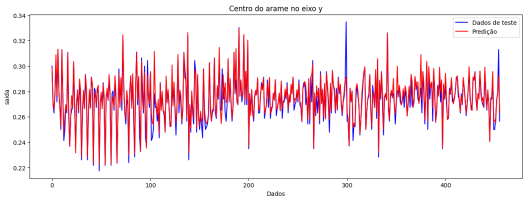
(b) Largura da Poça de soldagem



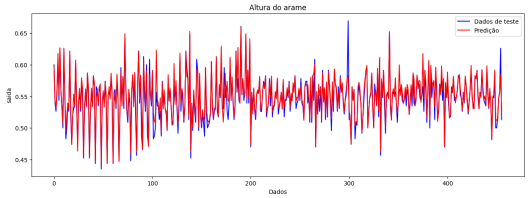
(c) Posição do Arame no eixo X



(d) Posição do Arame no eixo Y



(e) Altura do Arame



Fonte: Produzidas pelo autor

5 Avaliação dos resultados

5.1 Processo de soldagem

A primeira parte do projeto foi a realização dos seis experimentos onde cada um resultou em 1000 imagens 230x232 no formato bmp. Dentre essas imagens a grande parte não apresenta problema algum, porém foram observadas algumas imagens com alguns problemas como: falha de segmentação, respingos de metal e imagens com baixa luminosidade. Devido a esses problemas cerca de 38 imagens não foram rotuladas e foram ignoradas nos processos de treinamento e avaliação das redes neurais. Foi observado também que algumas mudanças nos parâmetros de soldagem variam entre os experimentos e isso causou leves mudanças, dificilmente observáveis, nas imagens geradas.

5.2 Rotulação dos dados

A rotulagem dos dados foi realizada manualmente, o que introduz um certo grau de imprecisão que deve ser levado em consideração ao avaliar a precisão das saídas dos modelos de redes neurais. É importante ressaltar que diferentes pessoas realizaram a rotulagem, resultando em percepções variadas sobre a rotulação de uma imagem. Além disso, a percepção também foi refinada ao longo do tempo.

No processo de rotulagem dos dados, constatamos que aproximadamente 15% das imagens são momentos em que ocorre um curto-circuito. Isso indica que a rede pode encontrar dificuldades em identificar esses momentos de curto, uma vez que sua tendência natural é se concentrar nos momentos em que não há curto-circuito.

Em relação às dimensões observadas, verificamos que as dimensões reais que refletem as condições reais da soldagem são mais claramente observadas nos momentos em que ocorre o curto-circuito. Isso ocorre porque esses momentos apresentam menor luminosidade, o que proporciona maior clareza na imagem. Portanto, para uma consideração real dos parâmetros de soldagem, é necessário observar os momentos em que ocorre o curto-circuito.

Quanto ao arame, em relação à posição do arame na imagem, há uma variação moderada, especialmente ao compararmos os diferentes experimentos. Isso ocorre devido à utilização de diferentes parâmetros nas capturas das imagens durante os experimentos e a posição da câmera, isso reflete em uma variação desses valores.

Em relação à posição final do arame, ou seja, onde o arame se transforma na poça de solda, observamos uma grande variação nos valores rotulados. Essa variação ocorre devido ao processo de fusão que não ocorre de forma uniforme. Além disso, há uma tendência para

valores maiores quando observamos momentos de curto-circuito na imagem. Isso é plausível, pois o brilho da imagem naquele local específico oculta a dimensão real da imagem.

Em relação à poça de soldagem, notamos uma grande variação tanto na largura da poça quanto na sua posição em relação à imagem. Esses dois fatos estão intimamente relacionados, pois a largura da poça de soldagem pode influenciar a sua posição na imagem. Isso ocorre porque diferentes parâmetros e condições durante o processo de soldagem podem resultar em variações na forma e na localização da poça de soldagem.

Além disso, observamos que os menores valores da poça de soldagem estão associados aos momentos em que ocorre o curto-circuito. Essa relação é natural e pode ser justificada pelo fato de que, nesses momentos, o brilho da imagem é menor. Como mencionado anteriormente, os momentos de curto-circuito apresentam menor luminosidade, o que pode levar a uma percepção reduzida da largura da poça de soldagem na imagem.

5.3 Construção da rede neural

Nessa etapa do projeto, iniciamos a construção da rede neural, com foco inicial na classificação dos momentos de curto-circuito. Ao desenvolver a primeira rede genérica, observamos que o conjunto de treinamento se adequava ao modelo, porém detectamos um problema de sobreajuste (*overfitting*) ocorrendo precocemente, por volta da terceira época de treinamento. A partir dessa observação, chegamos a duas conclusões: a primeira é que o modelo inicial pode ser treinado e utilizado para a classificação do problema, e a segunda é que o modelo ainda não está totalmente ajustado ao problema em questão.

Com base nessas informações, procedemos à seleção dos hiperparâmetros mais adequados ao problema. Essa escolha foi feita com base em dados, utilizando as métricas obtidas durante a avaliação do modelo por meio de ajustes validados com o processo de validação cruzada (*cross validation*). Optamos por utilizar a métrica de largura de poça para avaliar as redes construídas, tratando-a como um valor numérico e aplicando regressão linear na saída. Dessa forma, avaliamos diversos modelos de rede neural, testando diferentes combinações de hiperparâmetros e analisando os resultados com a mesma métrica. Nosso objetivo ao analisar esses resultados não era apenas minimizar o erro quadrático médio, mas também confirmar a escolha das combinações e garantir que determinado modelo não aumentaria o erro.

Após a seleção de todos os hiperparâmetros, avaliamos o modelo construído e conduzimos dois estudos em relação à saída da rede neural. Analisamos o comportamento da rede neural tanto com múltiplas saídas como com apenas uma saída. Observamos que, com múltiplas saídas, a rede generaliza cada saída de forma excessiva e não é capaz de prever resultados precisos. Basicamente, cada saída se torna uma média dos valores de rótulo de cada saída correspondente. De uma forma simples a rede não é robusta o suficiente para

dividir sua capacidade entre várias saídas.

Em seguida, o modelo construído foi treinado para ter apenas uma saída e o seu desempenho foi avaliado. Nesse resultado, deve-se considerar duas categorias de saída: a classificação dos momentos de curto-circuito e as saídas numéricas de regressão. Em relação à regressão, observa-se uma melhor adequação das métricas de avaliação nas saídas, porém, ainda assim, a capacidade de adaptação à variabilidade dos dados foi limitada. Isso resultou em um desempenho não tão satisfatório, pois, mesmo com erros absolutos e quadráticos bastante baixos, a métrica R2 e os gráficos gerados indicaram que os resultados não eram precisos o suficiente.

Em relação à classificação, obtivemos resultados positivos. A mesma rede neural desenvolvida para o problema de regressão alcançou uma taxa de acerto de 97% nas classificações de imagens. Esse resultado foi altamente satisfatório e quase excelente, especialmente considerando a possibilidade de existirem casos em que as etiquetas de classificação estejam incorretas.

A escolha das arquiteturas de base para o modelo, como LeNet, AlexNet e VGG, que foram originalmente desenvolvidas para a classificação de imagens, provavelmente influenciou os resultados observados. Essas arquiteturas são altamente eficazes na extração de características e na classificação precisa de objetos em imagens. É importante ressaltar que a classificação é uma tarefa específica na qual o modelo busca identificar a classe correta de uma determinada imagem. Por outro lado, a regressão tem como objetivo prever valores numéricos, o que requer uma abordagem diferente e uma modelagem mais adequada aos dados de entrada.

Devido à natureza das arquiteturas escolhidas, que foram desenvolvidas com foco na classificação, é compreensível que tenhamos obtido resultados satisfatórios nessa tarefa. No entanto, a mesma abordagem pode não ser tão eficaz quando aplicada à regressão, onde a relação entre as variáveis de entrada e saída é contínua e não discreta.

Portanto, a discrepância nos resultados entre a classificação e a regressão pode ser atribuída à escolha das arquiteturas de base, que não são otimizadas para a tarefa de regressão. Isso destaca a importância de explorar arquiteturas mais adequadas e ajustadas especificamente para problemas de regressão ao trabalhar com dados numéricos.

5.4 YOLOv8

Após analisarmos os resultados obtidos, decidimos empregar uma técnica chamada *transfer learning*. Essa abordagem consiste em aproveitar os conhecimentos adquiridos por um modelo pré-treinado em um conjunto de dados relacionado e aplicá-lo em um novo conjunto de dados. No nosso caso, utilizamos um modelo pré-treinado no problema de

detecção de objetos chamado YOLO (*You Only Look Once*).

Para utilizar o YOLO, realizamos a devida transformação dos dados para o formato aceito pelo *framework* e procedemos com o treinamento do modelo. O resultado obtido foi visualmente impressionante, com a rede neural sendo capaz de aprender, rotular e classificar os parâmetros com excelência. Ao analisar as métricas, constatamos que os erros foram minimizados e, pela métrica R2, a rotulação se adaptou muito bem à variabilidade dos dados. Além disso, ao observar os gráficos gerados, podemos constatar que as curvas de predição e os valores reais são muito semelhantes.

Considerando esses resultados, podemos concluir que o uso do YOLO foi extremamente benéfico para o projeto. Essa técnica de *transfer learning* permitiu que aproveitássemos o conhecimento prévio do modelo treinado em um conjunto de dados relacionado, acelerando o processo de treinamento e melhorando significativamente a capacidade de detecção e classificação dos parâmetros desejados. A aplicação do YOLO trouxe resultados visuais satisfatórios, além de apresentar baixos erros e uma boa adaptação à variabilidade dos dados.

6 Conclusões

A rotulagem manual dos dados apresentou um certo grau de imprecisão, decorrente de diferentes percepções dos rotuladores e refinamentos ao longo do tempo. A detecção de momentos de curto-circuito foi um desafio, já que a tendência natural da rede era concentrar-se nos momentos sem curto-circuito. Além disso, observou-se que as dimensões reais do processo de soldagem são mais claras nos momentos de curto-circuito, devido à menor luminosidade, e, portanto, esses momentos são essenciais para uma consideração real dos parâmetros de soldagem.

Ao desenvolver a rede neural personalizada, notou-se o problema de sobreajuste e a necessidade de ajustar cuidadosamente os hiperparâmetros para melhorar a performance. A escolha de arquiteturas de base originalmente desenvolvidas para classificação de imagens influenciou os resultados positivos na classificação dos momentos de curto-circuito, mas apresentou limitações na tarefa de regressão.

Para abordar essas limitações, optou-se pelo *transfer learning* com a arquitetura YOLOv8, pré-treinada em detecção de objetos. Os resultados foram impressionantes, com uma excelente capacidade de aprendizado e classificação dos parâmetros. A utilização do YOLOv8 permitiu minimizar os erros e adaptar-se melhor à variabilidade dos dados, com resultados satisfatórios tanto em métricas quanto em gráficos.

Como próximos passos para o projeto, sugere-se alguns caminhos. O primeiro é a exploração de outras arquiteturas adequadas a problemas de regressão, que possam dar maior controle sobre os processos intermediários do treinamento, essa abordagem permite também a realização de estudos comparando arquiteturas afim de avaliar sua eficácia. O segundo caminho é a realização de mais experimentos de soldagem, porém com características de falha. Isso pode incluir a criação de um novo conjunto de dados que contemple diferentes tipos de falhas, como porosidade, descontinuidades, falta de fusão, entre outros.

Ao incluir características de falha nos experimentos, será possível treinar a inteligência artificial para identificar e classificar não apenas os parâmetros ideais do processo de soldagem, mas também as características indesejáveis ou problemáticas. Isso contribuirá para o desenvolvimento de um sistema mais abrangente e eficaz na detecção e análise de falhas em soldas. Além disso, os experimentos com características de falha permitirão expandir o conjunto de dados utilizado no treinamento da rede neural, tornando-o mais diversificado e representativo das possíveis variações e desafios encontrados na soldagem. Isso fortalecerá o modelo e o preparará para lidar com uma gama mais ampla de situações reais.

Outra direção interessante para pesquisas futuras seria explorar abordagens de interpretabilidade do modelo, entendendo quais características visuais estão sendo aprendidas e

como elas se relacionam com os parâmetros de soldagem. Isso pode ajudar a fornecer *insights* valiosos para engenheiros e técnicos no processo de otimização do processo de soldagem.

Em conclusão, este trabalho de graduação contribuiu para o campo da classificação de parâmetros de soldagem GMAW por meio do uso de redes neurais. As descobertas e contribuições aqui apresentadas abrem caminho para pesquisas futuras, visando melhorar a precisão e a eficácia dos modelos de classificação de parâmetros. A inteligência artificial tem o potencial de revolucionar a indústria de soldagem, proporcionando um processo mais eficiente e confiável, e este estudo é apenas o começo dessa jornada.

Referências

ABRACD. **Overfitting e underfitting em Machine Learning**. Disponível em: https://abracd.org/overfitting-e-underfitting-em-machine-learning/ – acesso em 31 de Maio de 2023. 2021. Citado nas pp. 47, 48.

AMARNATH, M.; SUDHARSHAN, N.; SRINIVAS, P. Automatic detection of defects in welding using deep learning-a systematic review. **Materials Today: Proceedings**, Elsevier, 2023. Citado na p. 62.

ARAÚJO, F. H.; CARNEIRO, A.; SILVA, R. R.; MEDEIROS, F. N.; USHIZIMA, D. M. Redes neurais convolucionais com tensorflow: Teoria e prática. **SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. III Escola Regional de Informática do Piauí. Livro Anais-Artigos e Minicursos**, Sociedade Brasileira de Computação, v. 1, p. 382–406, 2017. Citado nas pp. 51, 129.

BALMER. **Processo soldagem MIG e MAG**. [Online; acessado em 14 de junho de 2023]. 2021. Disponível em: <https://www.balmer.com.br/blog/processo-de-soldagem-mig-mag/>. Citado na p. 23.

BASHEER, I. A.; HAJMEER, M. Artificial neural networks: fundamentals, computing, design, and application. **Journal of Microbiological Methods**, v. 43, p. 3–31, nov. 2000. Citado na p. 39.

BAUCHSPIESS, A. **Aplicações em Engenharia de Redes Neurais Artificiais, Lógica Fuzzy e Sistemas Neuro-Fuzzy**. Nov. 2008. Brasília. Introdução aos Sistemas Inteligentes. Citado na p. 28.

BROWNLEE, J. **Weight Initialization for Deep Learning Neural Networks**. Disponível em: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/ – acesso em 27 de Maio de 2023. 2021. Citado na p. 43.

CONTRIBUTORS, W. **Convolução**. 2023a. [Online; acessado em 7 de junho de 2023]. Disponível em: <https://en.wikipedia.org/w/index.php?title=Convolution&oldid=1155936911>. Citado na p. 50.

CONTRIBUTORS, W. **Kernel (image processing) — Wikipedia, The Free Encyclopedia**. [Online; accessed 7-June-2023]. 2023b. Disponível em: <https://en.wikipedia.org/w/index.php?title=Kernel_(image_processing)&oldid=1133314414]>. Citado na p. 50.

DATA SCIENCE ACADEMY. **Deep learning book**. Disponível em: https://www.deeplearningbook.com.br/ – acesso em 15 Maio 2023. 2022. Citado na p. 40.

DOMINGOS, P. A few useful things to know about machine learning. **Communications of the ACM**, ACM New York, NY, USA, v. 55, n. 10, p. 78–87, 2012. Citado na p. 47.

ERTUGRUL, O. F. A novel type of activation function in artificial neural networks: Trained activation function. **Elsevier**, v. 99, p. 148–157, mar. 2018. Citado na p. 34.

EXPERT, I. **Curso de Deep Learning**. [Online; acessado em 7 de junho de 2023]. 2023. Disponível em: <https://iaexpert.academy/>. Citado nas pp. 51, 54, 55.

FERNANDES, R. F. MONITORAÇÃO NO PROCESSO DE SOLDAGEM GMAW POR MEIO DE VISÃO COMPUTACIONAL E DESENVOLVIMENTO DE MÉTODOS PARA APLICAÇÃO EM FPGA. **Faculdade de Tecnologia - Universidade de Brasília**, 2015. Trabalho de Graduação em Engenharia de Controle e Automação. Citado na p. 56.

FORTES, C. **Soldagem MIG/MAG**. 1. ed.: ESAB, 2005. Citado na p. 24.

FRANCO, L. D. N. Sincronização, captura e análise de imagens da poça de soldagem no processo GMAW convencional, no modo de transferência metálica por curto-circuito. **Departamento de Engenharia Mecânica - Universidade de Brasília**, 2007. Dissertação de Mestrado em Sistemas Mecatrônicos. Citado nas pp. 24–26, 56, 63.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010. v. 9. (Proceedings of Machine Learning Research), p. 249–256. Disponível em: <https://proceedings.mlr.press/v9/glorot10a.html>. Citado nas pp. 43, 88.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. MIT press, 2016. Citado nas pp. 48, 53.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. New York, NY, USA: Springer New York Inc., 2001. (Springer Series in Statistics). Citado na p. 47.

HAYKIN, S. **Redes neurais princípios e Prática**. 2. ed. Porto Alegre: Bookman, 2001. Traduzido por: Paulo Martins Engel. ISBN 85-7307-718-2. Citado nas pp. 26, 36, 38, 42.

HE, K.; ZHANG, X.; REN, S.; SUN, J. **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**. 2015. arXiv: 1502.01852 [cs.CV]. Citado nas pp. 43, 59, 88.

HEBB, D. O. a neuropsychological theory. **The Organization of Behavior**, Wiley, New York, 1949a. Citado na p. 37.

HEBB, D. O. The first stage of perception: growth of the assembly. **The Organization of Behavior**, Wiley, New York, v. 4, n. 60, p. 78–60, 1949b. Citado na p. 35.

HECHT-NIELSEN, R. **Neurocomputing**. Pearson, 1990. Citado na p. 28.

HUBEL, D. H.; WIESEL, T. N. Ferrier lecture-Functional architecture of macaque monkey visual cortex. **Proceedings of the Royal Society of London. Series B. Biological Sciences**, The Royal Society London, v. 198, n. 1130, p. 1–59, 1977. Citado na p. 49.

JAIN, A. K.; MAO, J.; MOHIUDDIN, K. M. Artificial neural networks: A tutorial. **Computer**, IEEE, v. 29, n. 3, p. 31–44, 1996. Citado na p. 31.

JOCHER, G.; CHAURASIA, A.; QIU, J. **YOLO by Ultralytics**. [Online; acessado em 29 de abril de 2023]. 2023. Disponível em: <https://github.com/ultralytics/ultralytics>. Citado nas pp. 61, 96, 97.

KANDEL, E. R.; SCHWARTZ, J. H.; JESSELL, T. M. **Priciples of neural science**. 4. ed. New York: McGraw-Hill, 2000. ISBN 0-8385-8068-8. Citado na p. 29.

KARN, U. **An Intuitive Explanation of Convolutional Neural Networks**. [Online; acessado em 23 de junho de 2023]. 2016. Disponível em: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>. Citado na p. 50.

KERAS. **Keras API reference**. [Online; acessado em 7 de junho de 2023]. 2023. Disponível em: <https://keras.io/api/>. Citado nas pp. 31, 76, 83.

KHAN ACADEMY. **Anatomia de um neurônio**. Disponível em: https://pt.khanacademy.org/science/biology/human-biology/neuron-nervous-system/v/anatomy-of-a-neuron – acesso em 11 Maio. 2023. 2015. Citado na p. 29.

KOHAVI, R. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. v. 14, mar. 2001. Citado na p. 46.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Communications of the ACM**, AcM New York, NY, USA, v. 60, n. 6, p. 84–90, 2017. Citado na p. 58.

LAKSHMANAN, V.; GÖRNER, M.; GILLARD, R. **Practical Machine Learning for Computer Vision**. "O'Reilly Media, Inc.", 2021. Citado na p. 48.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998. DOI: 10.1109/5.726791. Citado nas pp. 49, 57.

LIPPMANN, R. An introduction to computing with neural nets. eng. **IEEE ASSP magazine**, IEEE, v. 4, n. 2, p. 4–22, 1987. ISSN 0740-7467. Citado na p. 41.

MARQUES, P. V.; MODENESI, P. J.; BRACARENSE, A. Q. **Soldagem Fundamentos e tecnologia**. Editora UFMG, 2009. v. 3. Citado na p. 23.

MATHWORKS. **What Is a Convolutional Neural Network?** 2018. Brasília. [Online; acessado em 20 de junho de 2023]. Disponível em: <https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html>. Citado na p. 49.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, p. 198–211, 1943. Citado na p. 29.

MEDEIROS, L. F. de. **inteligência Artificial Aplicada: uma abordagem introdutória**. 1. ed. Curitiba. PR. Brasil: intersaberes, 2018. Citado na p. 38.

MÜLLER, A.; GUIDO, S. **Introduction to Machine Learning with Python: A Guide for Data Scientists**. O'Reilly Media, nov. 2016. ISBN 1449369413. Citado na p. 36.

NASCIMENTO JR., C. L.; YONEYAMA, T. **Inteligência Artificial em controle e automação**. Edgard Blucher, 2000. Citado na p. 28.

NORRISH, J. **Advanced Welding Processes**. Elsevier Science, 2006. (Woodhead Publishing Series in Welding and Other Joining Technologies). ISBN 9781845691301. Disponível em: <https://books.google.com.br/books?id=M4P3PQAACAAJ>. Citado na p. 23.

OKUNMURA, T.; TANIGUCHI, C. **Engenharia de soldagens e aplicações**. 1. ed. Rio de janeiro, RJ, Brasil: LTC, 1982. v. 1. Citado nas pp. 17, 22.

PANDAS. **Pandas Documentation**. [Online; acessado em 7 de junho de 2023]. 2023. Disponível em: <https://pandas.pydata.org/docs/>. Citado na p. 75.

PATRO, R. **Cross-Validation: K Fold vs Monte Carlo**. Disponível em: https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b – acesso em 31 de Maio de 2023. 2021. Citado na p. 46.

PENTTILÄ, S.; KAH, P.; RATAVA, J.; ESKELINEN, H. Artificial Neural Network controlled GMAW system: penetration and quality assurance in a multi-pass butt weld application. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 105, p. 3369–3385, 2019. Citado na p. 62.

POERNOMO, A.; KANG, D.-K. Biased Dropout and Crossmap Dropout: Learning towards effective Dropout regularization in convolutional neural network. **Neural Networks**, v. 104, p. 60–67, 2018. ISSN 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2018.03.016. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608018301096>. Citado na p. 48.

POWELL, V. **Image Kernels: Explained Visually**. [Online; acessado em 7 de junho de 2023]. 2023. Disponível em: <https://setosa.io/ev/image-kernels/>. Citado nas pp. 52, 53.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. 2016. P. 779–788. Citado na p. 60.

ROCHA, J. C. **Introdução ao Perceptron multicamadas passo a passo**. Disponível em: https://juliocprocha.wordpress.com/2020/03/30/introducao-ao-perceptron-multicamadas-passo-a-passo/ – acesso em 12 Maio. 2023. 2020. Citado na p. 37.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, p. 386–408, 1958. Citado na p. 36.

RUBIO, J. L. **Regresión Lineal y Descenso de Gradiente**. Disponível em: https://iadelta.com/inteligencia-artificial/regresion/lineal-y-gradiente/ – acesso em 31 de Maio de 2023. 2020. Citado na p. 45.

SCIKITLEARN. **Scikitlearn User Guide**. [Online; acessado em 7 de junho de 2023]. 2023. Disponível em: <https://scikit-learn.org/stable/user_guide.html>. Citado na p. 81.

SILVA, F. F. P. DESENVOLVIMENTO DE UM SENSOR DE VISÃO PARA MONITORAÇÃO DOS PARÂMETROS GEOMÉTRICOS DA POÇA DE SOLDA NO PROCESSO GTAW PULSADO. **Departamento de Engenharia Mecânica - Universidade de Brasìlia**, 2016. Dissertação de Mestrado em Sistemas Mecatrônicos. Citado na p. 57.

SILVA, I. N. da; SPATTI, D. H. **Redes Neurais Artificiais para engenharia e ciências aplicadas**. 2. ed. Universidade de São Paulo: Artliber editora ltda., 2016. Citado nas pp. 29, 30, 35, 42.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014. Citado na p. 58.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2. ed.: Bradford Books, nov. 2018. An optional note. ISBN 9780262039246. Citado na p. 36.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V.; ALEMI, A. Inception-v4, inception-resnet and the impact of residual connections on learning. In: 1. PROCEEDINGS of the AAAI conference on artificial intelligence. 2017. v. 31. Citado na p. 59.

TENSORFLOW. **TensorFlow documentation Keras**. [Online; acessado em 7 de junho de 2023]. 2023. Disponível em: <https://www.tensorflow.org/api_docs/python/tf/keras>. Citado na p. 74.

TERVEN, J.; CORDOVA-ESPARZA, D. A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond. **arXiv preprint arXiv:2304.00501**, 2023. Citado na p. 61.

THE ASIMOV INSTITUTE. **THE NEURAL NETWORK ZOO**. Disponível em: https://www.asimovinstitute.org/neural-network-zoo/ – acesso em 27 de Maio de 2023. 2016. Citado nas pp. 35, 130.

WEMAN, K. Welding processes handbook. **Cambridge UK Philadelphia PA USA by Woodhead Pub**, n. 2, p. 1–12, 2012. Citado na p. 17.

WIDROW, B.; HOFF, M. E. **Adaptive switching circuits**. 1960. Citado na p. 42.

WINSTON, P. H. **Artificial Intelligence**. 3. ed.: Addison-Wesley, 1977. Citado na p. 18.

XIONG, Z.; CUI, Y.; LIU, Z.; ZHAO, Y.; HU, M.; HU, J. Evaluating explorative prediction power of machine learning algorithms for materials discovery using k-fold forward cross-validation. **Computational Materials Science**, v. 171, p. 109203, 2020. ISSN 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2019.109203. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0927025619305026>. Citado na p. 46.

XU, F.; XU, Y.; ZHANG, H.; CHEN, S. Application of sensing technology in intelligent robotic arc welding: A review. **Journal of Manufacturing Processes**, Elsevier, v. 79, p. 854–880, 2022. Citado na p. 62.

ZHOU, Y.; CHANG, B.; ZOU, H.; SUN, L.; WANG, L.; DU, D. Online visual monitoring method for liquid rocket engine nozzle welding based on a multi-task deep learning model. **Journal of Manufacturing Systems**, Elsevier, v. 68, p. 1–11, 2023. Citado na p. 62.

Apêndices

Apêndice A – Códigos de programação

A.1 Classificação de momento de curto

Código A.1 – Rede neural convolucional para a classificação de imagens em momento de curto

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
    Dropout
from keras.preprocessing.image import ImageDataGenerator
import pandas as pd
import matplotlib.pyplot as plt

trainDf =
    pd.read_excel("/content/drive/MyDrive/TCC/Experimento1.xlsx")

trainDf["Curto"] = trainDf["Curto"].astype(str)

datagen = ImageDataGenerator(rescale=1./255.,validation_split=0.25)

train_generator = datagen.flow_from_dataframe(
    dataframe=trainDf, # Dataframe com os dados
    directory='/content/drive/MyDrive/TCC/Experimento1', #
        Diretório com as imagens
    x_col='Nome da imagem', # Coluna com os nomes das imagens
    y_col= "Curto",    # Coluna com os rótulos
    subset='training',  # Conjunto de treinamento
    batch_size=25, # Tamanho do batch
    color_mode='grayscale', # Para imagens com um canal de cor
    class_mode='binary',  # Para classificação binária
    target_size=(230, 230)  # Tamanho desejado das imagens
)

validation_generator = datagen.flow_from_dataframe(
    dataframe=trainDf,
    directory='/content/drive/MyDrive/TCC/Experimento1',
    x_col='Nome da imagem',
    y_col= "Curto",
    subset='validation',  # Conjunto de validação
    batch_size=25,
    color_mode='grayscale',
    class_mode='binary',
    target_size=(230, 230)
)
# Criação da rede neural
```

```
classificador = Sequential()
# Camada de convolução
classificador.add(Conv2D(128, (3, 3), activation='relu',
   input_shape=(230, 230, 1)))
# Camada de pooling
classificador.add(MaxPooling2D((2, 2)))
classificador.add(Conv2D(128, (3, 3), activation='relu'))
classificador.add(MaxPooling2D((2, 2)))
classificador.add(Conv2D(128, (3, 3), activation='relu'))
classificador.add(MaxPooling2D((2, 2)))
# Camada de achatamento
classificador.add(Flatten())
# Camada densa
classificador.add(Dense(100, activation='relu'))
# Dropout para evitar overfitting
classificador.add(Dropout(0.15))
classificador.add(Dense(100, activation='relu'))
classificador.add(Dropout(0.15))
classificador.add(Dense(100, activation='relu'))
classificador.add(Dropout(0.15))
 # Camada de saída
classificador.add(Dense(1, activation='sigmoid'))

classificador.compile(optimizer='adam',
   loss='binary_crossentropy', metrics=['binary_accuracy'])
# Treinamento da rede neural
history = classificador.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)

train_metric = history.history['binary_accuracy']
val_metric = history.history['val_binary_accuracy']

# Plotar o gráfico de função de erro
plt.plot(train_metric, label='Train binary accuracy')
plt.plot(val_metric, label='Validation binary accuracy')
plt.title('Acurácia')
plt.xlabel('Epocas')
plt.ylabel('Acurácia')
plt.legend()
plt.show()

# Avaliação do modelo
evaluation_results =
   classificador.evaluate_generator(generator=validation_generator)
print("Loss: ", evaluation_results[0])
print("Accuracy: ", evaluation_results[1])
```

A.2 Obtenção de dados

Código A.2 – Obtenção de imagens e *dataframe* para treinamento

```
import numpy as np
import pandas as pd
import os
from PIL import Image

# Base para rotulos de imagens
trainDf = pd.read_excel("/dataframe/Experimento1.xlsx")
y_train = trainDf[["percentageWidthPoca","Imagem"]].values

# Base de imagens
image_dir = '/diretorio/imagens'
images = []
for image_file in os.listdir(image_dir):
    image_path = os.path.join(image_dir, image_file)
    image = Image.open(image_path)
    # Redimensionar para o tamanho desejado
    image = image.resize((230, 230))
    image = np.array(image)
    images.append(image)

# Ordenação do dataframe e mudança da tipagem
y_train = np.array(y_train)
y_train = sorted(y_train, key=lambda x: x[1])
y_train = np.array(y_train)
y_train = pd.DataFrame(y_train)
y_train = y_train.drop(1,axis=1)
y_train = np.array(y_train)
y_train = y_train.astype('float32')

# Mundanças da dimensão e tipo da imagem
X_train = np.array(images)
X_train = X_train.reshape(X_train.shape[0], 230, 230, 1)
X_train = X_train / 255.0
X_train32 = X_train.astype('float32')
```

A.3 *Tuning* de hiperparâmetros

Código A.3 – Treinamento da rede neural para os hiperparâmetros: Números de camadas

```
def create_model(conv_filters, dense_units,):

    model = Sequential()

    model.add(Conv2D(32, (3, 3), activation='relu',
       input_shape=(100, 100, 1)))
    model.add(MaxPooling2D((2, 2)))
```

```
7
8       for filters in conv_filters:
9         model.add(Conv2D(filters, (3, 3), activation='relu'))
10        model.add(MaxPooling2D((2, 2)))
11
12      model.add(Flatten())
13
14      for units in dense_units:
15        model.add(Dense(units = units, activation='relu'))
16        model.add(Dropout(0.15))
17
18      model.add(Dense(1, activation='linear'))
19
20      model.compile(optimizer='SGD', loss='mean_squared_error')
21      return model
```

Código A.4 – Treinamento da rede neural para os hiperparâmetros: Números de *kernels* e *neurônios*

```
1  def create_model(conv_filters, dense_units):
2
3       model = Sequential()
4
5       model.add(Conv2D(conv_filters[0], (3, 3), activation='relu',
           input_shape=(230, 230, 1)))
6       model.add(MaxPooling2D((2, 2)))
7
8       #... Camadas de convolução
9
10      model.add(Flatten())
11      model.add(Dense(units = dense_units[0], activation='relu'))
12      model.add(Dropout(0.15))
13
14      #... Camadas densas
15
16      model.add(Dense(1, activation='linear'))
17      model.compile(optimizer='SGD', loss='mean_squared_error')
18      return model
19
20 param_grid = {
21      'batch_size':[20],
22      'conv_filters': [[8, 16, 32, 64],
23                       [16, 32, 64, 128],
24                       [64, 128, 256, 512],
25                       [128, 256, 512, 1024],
26                       [100, 200, 300, 400]],
27      'dense_units': [[400,300,200,100],
28                       [230,230,230,230],
29                       [100,200,300,100],
30                       [100, 100, 100, 100],
31                       [30,30,30,30],
32                       [100,200,300,400]],
33      'epochs': [3]
```

```
}
```

Código A.5 – Treinamento da rede neural para os hiperparâmetros: Funções de ativação e inicialização de pesos

```
def create_model(conv_activation, dense_activation,
    conv_kernel_initializer, dense_kernel_initializer):

    model = Sequential()
    model.add(Conv2D(100, (3, 3),
              activation=conv_activation,
              input_shape=(230, 230, 1),
              kernel_initializer=conv_kernel_initializer))
    model.add(MaxPooling2D((2, 2)))

    #... Camadas de convolução

    model.add(Flatten())
    model.add(Dense(units = 30,
              activation=dense_activation,
              kernel_initializer=dense_kernel_initializer))
    model.add(Dropout(0.15))

    #... Camadas densas

    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='SGD', loss='mean_squared_error')
    return model

param_grid = {
    'batch_size':[20],
    'conv_activation':
        ['relu',
        'softplus',
        'elu',
        'selu',
        'tanh',
        'sigmoid'],
    'dense_activation':
        ['relu',
        'selu',
        'elu',
        'tanh',
        'linear'],
    'conv_kernel_initializer':
        ['glorot_uniform',
        'random_uniform',
        'glorot_normal',
        'he_normal',
        'he_uniform'],
    'dense_kernel_initializer':
        ['glorot_uniform',
```

```
        'random_uniform',
        'glorot_normal',
        'he_normal',
        'he_uniform'],
    'epochs': [3]
}
```

Código A.6 – Treinamento da rede neural para os hiperparâmetros: *Batch size* e *Dropout*

```
def create_model(dropout):
    model = Sequential()

    model.add(Conv2D(16, (3, 3), activation='relu',
       input_shape=(230, 230, 1)))
    model.add(MaxPooling2D((2, 2)))

    #... Camadas convolucionais

    model.add(Flatten())
    model.add(Dense(units = 30, activation='tanh',
       kernel_initializer='truncated_normal'))
    model.add(Dropout(dropout))

    #... Camadas densas

    model.add(Dense(1, activation='linear'))

    model.compile(optimizer='SGD', loss='mean_squared_error')
    return model

param_grid = {
    'batch_size':[10,20,32,64,128],
    'dropout':[0.1,0.15,0.2,0.3,0.4],
    'epochs': [4]
}
```

A.4 Redes neural Múltiplas saídas

Código A.7 – Criação do modelo de um rede neural para múltiplas saídas

```
inputs = Input(shape=(230,230,1))
regressor = Conv2D(16, (3, 3), activation='relu')(inputs)
regressor = MaxPooling2D((2, 2))(regressor)
regressor = Conv2D(32, (3, 3), activation='relu')(regressor)
regressor = MaxPooling2D((2, 2))(regressor)
regressor = Conv2D(32, (3, 3), activation='relu')(regressor)
regressor = MaxPooling2D((2, 2))(regressor)
regressor = Conv2D(64, (3, 3), activation='relu')(regressor)
regressor = MaxPooling2D((2, 2))(regressor)
regressor = Flatten()(regressor)
```

```
regressor = Dense(units = 30, activation='tanh',
    kernel_initializer='truncated_normal')(regressor)
regressor = Dropout(0.2)(regressor)
regressor = Dense(units = 30, activation='tanh',
    kernel_initializer='truncated_normal')(regressor)
regressor = Dropout(0.2)(regressor)
regressor = Dense(units = 30, activation='tanh',
    kernel_initializer='truncated_normal')(regressor)
regressor = Dropout(0.2)(regressor)
regressor = Dense(units = 30, activation='tanh',
    kernel_initializer='truncated_normal')(regressor)
regressor = Dropout(0.2)(regressor)
output1 = Dense(1, activation='sigmoid')(regressor)
output2 = Dense(1, activation='linear')(regressor)
output3 = Dense(1, activation='linear')(regressor)
output4 = Dense(1, activation='linear')(regressor)
output5 = Dense(1, activation='linear')(regressor)

model = Model(inputs=inputs,
    outputs=[output1,output2,output3,output4,output5])

model.compile(optimizer='SGD', loss='mean_squared_error')

history = model.fit(
    imagens_treino,[y_treino[:, 0], y_treino[:, 1], y_treino[:,
        2],y_treino[:, 3], y_treino[:, 4]],
    epochs=10,
    batch_size=20,
    validation_data=(imagens_validacao,
    [y_validacao[:,0],
    y_validacao[:,1],
    y_validacao[:,2],
    y_validacao[:,3],
    y_validacao[:,4]])
)
```

A.5 Rede neural saída única

Código A.8 – Algoritmo completo de um rede neural com saída única

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
    Dropout, Input
from keras.layers.normalization.batch_normalization import
    BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import np_utils
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from tensorflow.keras.models import Model
import cv2
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

trainDf = pd.read_excel("/Treino.xlsx")
testeDf = pd.read_excel("/Teste.xlsx")

imagens = []
tamanho_img = (230, 230)

# Loop pelas linhas do DataFrame
for index, row in trainDf.iterrows():
    nome_imagem = row['Imagem']
    caminho_imagem = '/Experimentos/' + nome_imagem
    # Carregar a imagem usando o OpenCV
    imagem = cv2.imread(caminho_imagem, cv2.IMREAD_GRAYSCALE)
    # Redimensionar a imagem
    imagem_redimensionada = cv2.resize(imagem, novo_tamanho)
    # Converter a imagem para um array NumPy
    array_imagem = np.array(imagem_redimensionada)
    # Adicionar o array da imagem e o rotulo as listas
    imagens.append(array_imagem)

# Tratamento para o formato dos dados
X_train = np.array(imagens)
X_train = X_train.reshape(X_train.shape[0], 230, 230, 1)
X_train = X_train / 255.0
X_train = X_train.astype('float32')

y_train = trainDf[["percentageLeftArame"]].values
y_train = np.array(y_train)
y_train = y_train
y_train = y_train.astype('float32')


imagens_teste = []
for index, row in testeDf.iterrows():
    nome_imagem = row['Imagem']
    caminho_imagem = 'Experimentos/' + nome_imagem
    imagem = cv2.imread(caminho_imagem, cv2.IMREAD_GRAYSCALE)
    imagem_redimensionada = cv2.resize(imagem, novo_tamanho)
    array_imagem = np.array(imagem_redimensionada)
    imagens_teste.append(array_imagem)
    rotulos_teste.append(row['Imagem'])

X_teste = np.array(imagens_teste)
X_teste = X_teste.reshape(X_teste.shape[0], 230, 230, 1)
X_teste = X_teste / 255.0
X_teste = X_teste.astype('float32')
```

```
61
62 y_teste = testeDf[["percentageLeftArame"]].values
63 y_teste = np.array(y_teste)
64 y_teste = y_teste
65 y_teste = y_teste.astype('float32')
66
67 # Divisão entre treino e validação
68 imagens_treino, imagens_validacao, y_treino, y_validacao =
       train_test_split(X_train, y_train, test_size=0.1,
       random_state=42)
69
70 # Criação do modelo
71 model = Sequential()
72
73 model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(230,
       230, 1)))
74 model.add(MaxPooling2D((2, 2)))
75 model.add(Conv2D(32, (3, 3), activation='relu'))
76 model.add(MaxPooling2D((2, 2)))
77 model.add(Conv2D(64, (3, 3), activation='relu'))
78 model.add(MaxPooling2D((2, 2)))
79 model.add(Conv2D(128, (3, 3), activation='relu'))
80 model.add(MaxPooling2D((2, 2)))
81
82 model.add(Flatten())
83
84 model.add(Dense(units = 30, activation='tanh',
       kernel_initializer='truncated_normal'))
85 model.add(Dropout(0.2))
86 model.add(Dense(units = 30, activation='tanh',
       kernel_initializer='truncated_normal'))
87 model.add(Dropout(0.2))
88 model.add(Dense(units = 30, activation='tanh',
       kernel_initializer='truncated_normal'))
89 model.add(Dropout(0.2))
90 model.add(Dense(units = 30, activation='tanh',
       kernel_initializer='truncated_normal'))
91 model.add(Dropout(0.2))
92
93 model.add(Dense(1, activation='linear'))
94
95 model.compile(optimizer='SGD', loss='mean_squared_error',
       metrics=['mean_absolute_error'])
96
97 # Treinamento
98 history = model.fit(
99     imagens_treino,y_treino,
100    epochs=10,
101    batch_size=20,
102    validation_data=(imagens_validacao,y_validacao)
103 )
104
```

```
# Avaliação do treinamento
y_pred = model.predict(X_teste)

plt.plot(y_pred, label="saida rede")
plt.plot(y_teste,label="saida real")
plt.xlabel('dados')
plt.ylabel('saida')
plt.legend()
plt.show()

mae = mean_absolute_error(y_teste, y_pred)
mse = mean_squared_error(y_teste, y_pred)
r2 = r2_score(y_teste, y_pred)
print(mae)
print(mse)
print(r2)
```

A.6 YOLOv8

Código A.9 – predição de todo o conjunto de dados de teste e o armazenamento dos dados em um arquivo CSV

```
from ultralytics import YOLO
model = YOLO('/content/drive/MyDrive/TCC/YOLOResults/best.pt')

import cv2
import os
import numpy as np
from PIL import Image

diretorio = '/content/drive/MyDrive/datasets/Yolo/test/images'


imagens = []
resultado = []

for nome_arquivo in os.listdir(diretorio):
      caminho_arquivo = os.path.join(diretorio, nome_arquivo)
      results = model.predict(source=caminho_arquivo,imgsz=256,
         conf=0.5, save_txt=True)
      resultado.append(results)

import csv
import os

# Diretório contendo os arquivos .txt
diretorio = '/content/runs/detect/predict/labels'

nome_arquivo_csv = 'saida.csv'

```

```
cabecalho = ['Arquivo', 'Conteudo']

with open(nome_arquivo_csv, 'w', newline='') as csv_file:
    writer = csv.writer(csv_file)
    writer.writerow(cabecalho)  # Escreve o cabeçalho no CSV

    for nome_arquivo in os.listdir(diretorio):
        if nome_arquivo.endswith('.txt'):
            caminho_arquivo = os.path.join(diretorio, nome_arquivo)
            with open(caminho_arquivo, 'r') as txt_file:
                conteudo = txt_file.read()

            # Escreve a linha no CSV
            writer.writerow([nome_arquivo, conteudo])

print("Conversão concluída. O arquivo CSV foi gerado com sucesso.")
```

A.7 Códigos auxiliares

Código A.10 – Código para transformar Json em CSV

```
import json
import csv

with open('export-2023-05-15T12_11_38.510Z.json') as json_file: # Nome do arquivo .json com as saidas dos rotulos
    data = json.load(json_file)

data_file = open('data_file.csv', 'w')
csv_writer = csv.writer(data_file)


for emp in data:
    csv_writer.writerow(emp.values()) # Cada objeto se tornará uma linha no arquivo .csv

data_file.close()
```

Código A.11 – Transformar rotulação para modelo YOLOv8

```
import pandas as pd
planilha = pd.read_excel('train.xlsx')

def convert_to_yolov8_format(row):
    classe1 = 0
    classe2 = 1

    # Coordenadas normalizadas para a primeira classe
    x1_center = row['yolo_xPocaCenter']
    y1_center = row['yolo_yPocaCenter']
```

```
11      width1 = row['yolo_xPocaw']
12      height1 = row['yolo_yPocah']
13
14      # Coordenadas normalizadas para a segunda classe
15      x2_center = row['yolo_xArameCenter']
16      y2_center = row['yolo_yArameCenter']
17      width2 = row['yolo_xAramew']
18      height2 = row['yolo_yArameh']
19
20      # Retorne as coordenadas no formato do YOLOv8
21      return f"{classe1} {x1_center} {y1_center} {width1}
           {height1}\n{classe2} {x2_center} {y2_center} {width2}
           {height2}"
22
23  for index, row in planilha.iterrows():
24      image_path = row['Imagem']
25      image_name = image_path.split('/')[-1].split('.')[0]
26
27      yolov8_labels = convert_to_yolov8_format(row)
28
29      with open(f'{image_name}.txt', 'w') as file:
30          file.write(yolov8_labels)
```

Apêndice B – Tabelas

Tabela B.11 – *Tuning* dos parâmetros: *optimizer* e *loss function*

	Desempenho	**Desvio padrão**	***optimizer***	***loss function***
1	38.27	13.69	Adamax	mean squared error
2	41.68	11.74	Adagrad	mean squared error
3	43.09	8.34	Adagrad	mean absolute error
4	43.12	14.95	Adamax	mean absolute error
5	46.78	17.85	Adam	mean absolute error
23	2687.33	123.03	SGD	cosine similarity
24	2687.68	124.34	Adadelta	cosine similarity

Tabela B.12 – *Tuning* dos parâmetros: Número de camadas

	Desempenho	**Desvio padrão**	**N° camadas de conv.**	**N° camadas densas**
1	23.43	8.02	4 camadas	1 camada
2	25.09	8.82	1 camada	1 camada
3	26.10	8.14	2 camadas	1 camada
4	26.3369	8.51	3 camadas	1 camada
5	30.52	10.15	4 camadas	2 camadas

Tabela B.13 – *Tuning* dos parâmetros: quantidade de *kernels* e neurônios

	Desempenho	**Desvio padrão**	**N° de *kernels***	**N° de neurônios**
1	26.21	6.58	[16, 32, 64, 128]	500
2	28.12	7.28	[64, 128,256,512]	100
3	30.13	8.4	[100, 200, 300, 400]	500
4	30.13	8.46	[128,256,512,1024]	100
5	30.43	7.19	[16, 32, 64, 128]	400

Tabela B.14 – *Tuning* dos parâmetros: Funções de ativação e inicializadores de pesos

	Desempenho	**Desvio padrão**	**Camada de convolução**	**Camada densa**
1	23.30	6.6	tanh e random uniform	tanh e he uniform
2	23.48	7.84	relu e random uniform	tanh e glorot normal
3	23.52	6.61	tanh e random uniform	tanh e truncate normal
4	23.64	7.00	tanh e glorot normal	tanh e glorot normal
5	23.74	7.02	tanh e he uniform	elu ehe uniform
6	35.39	11.5	tanh e he uniform	tanh e truncated normal
7	36.73	5.09	relu e random uniform	relu e truncate normal

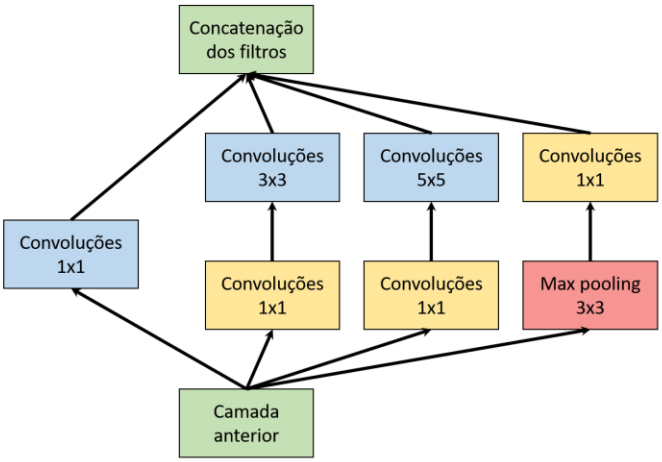
Tabela B.15 – *Tuning* dos parâmetros: Funções de ativação e inicializadores de pesos

	Desempenho	**Desvio padrão**	***Batch size***	***dropout***
1	22.45	7.2	10	0.3
2	22.46	7.22	10	0.15
3	22.53	6.91	20	0.15
4	22.53	7.21	10	0.1
5	22.57	7.15	10	0.4

Anexos

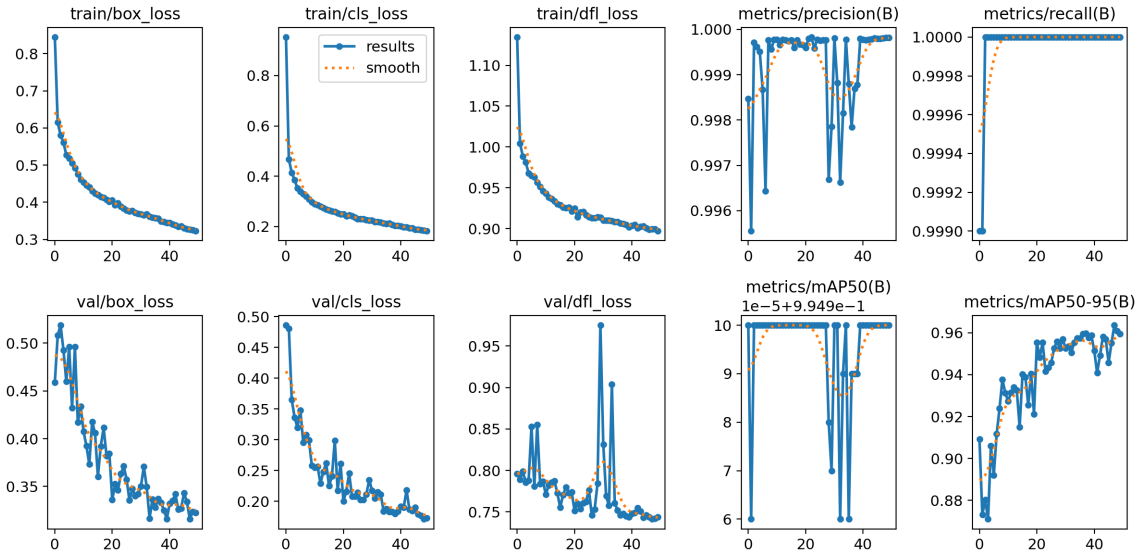
Anexo A – Figuras anexadas

Figura A.49 – Arquitetura do módulo *Inception*



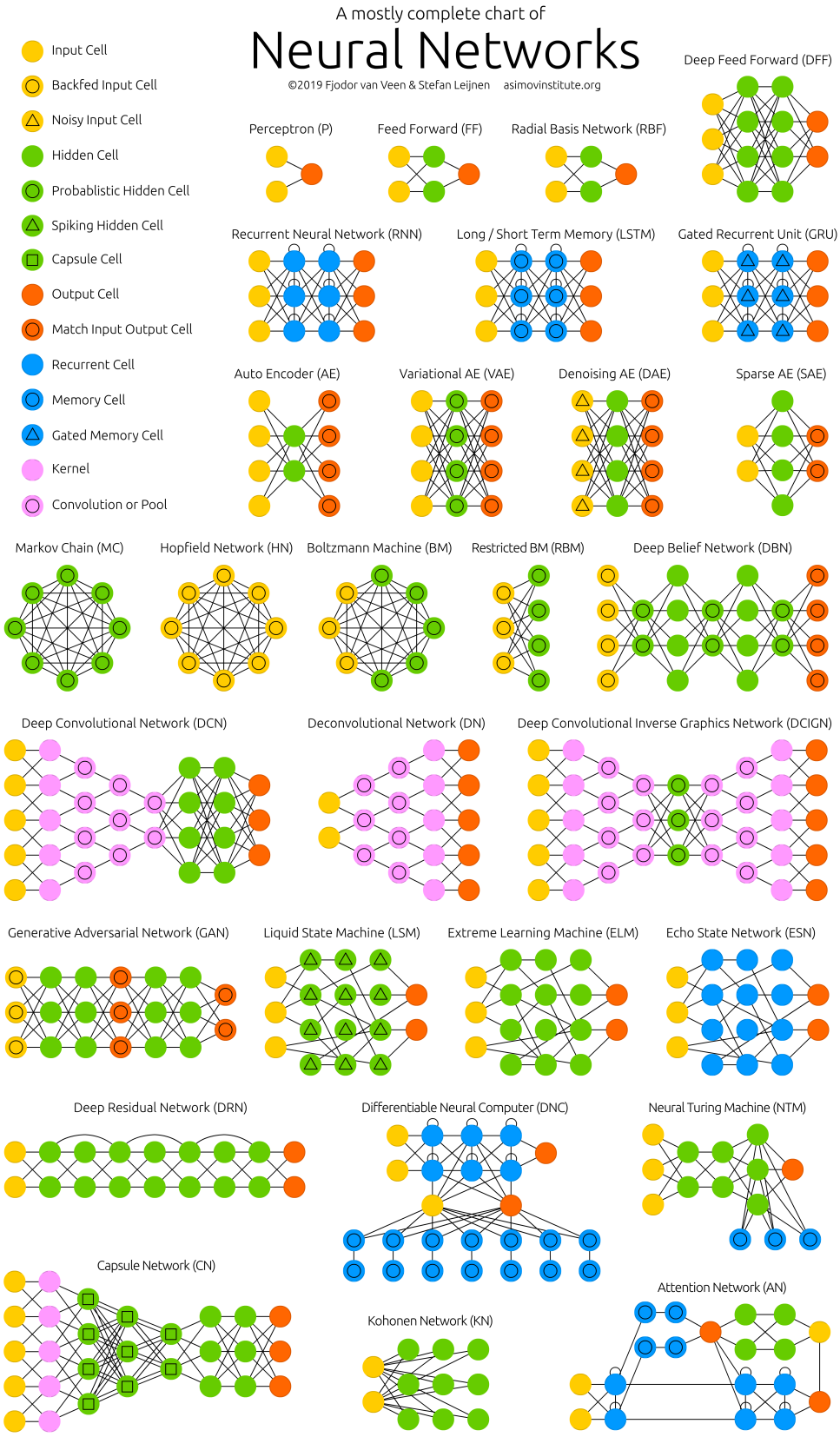
Fonte: ARAÚJO et al. (2017)

Figura A.50 – Resultado das métricas de treinamento da arquitetura do YOLOv8



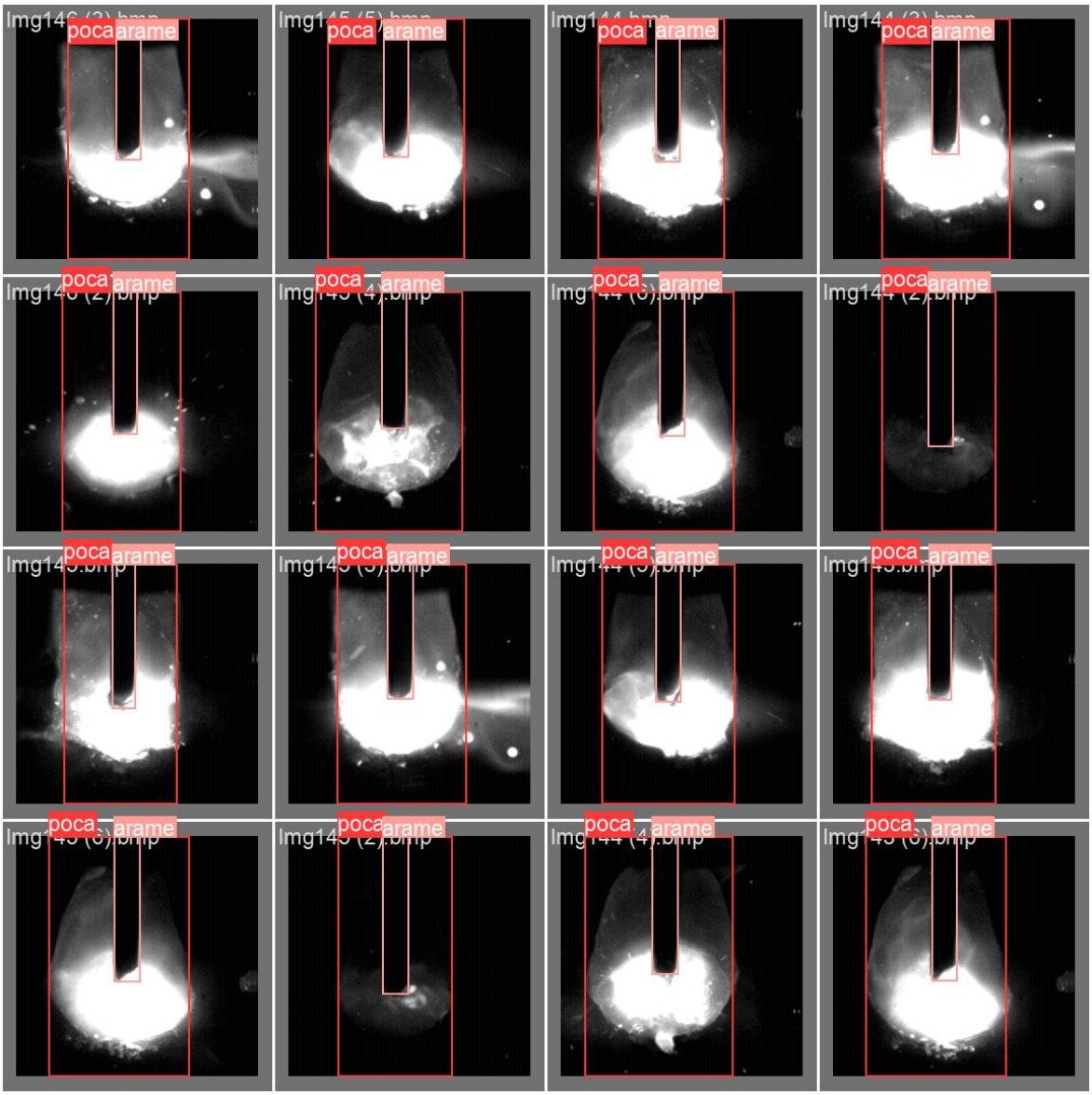
Fonte: Produzido pelo autor

Figura A.51 – Arquiteturas de redes neurais



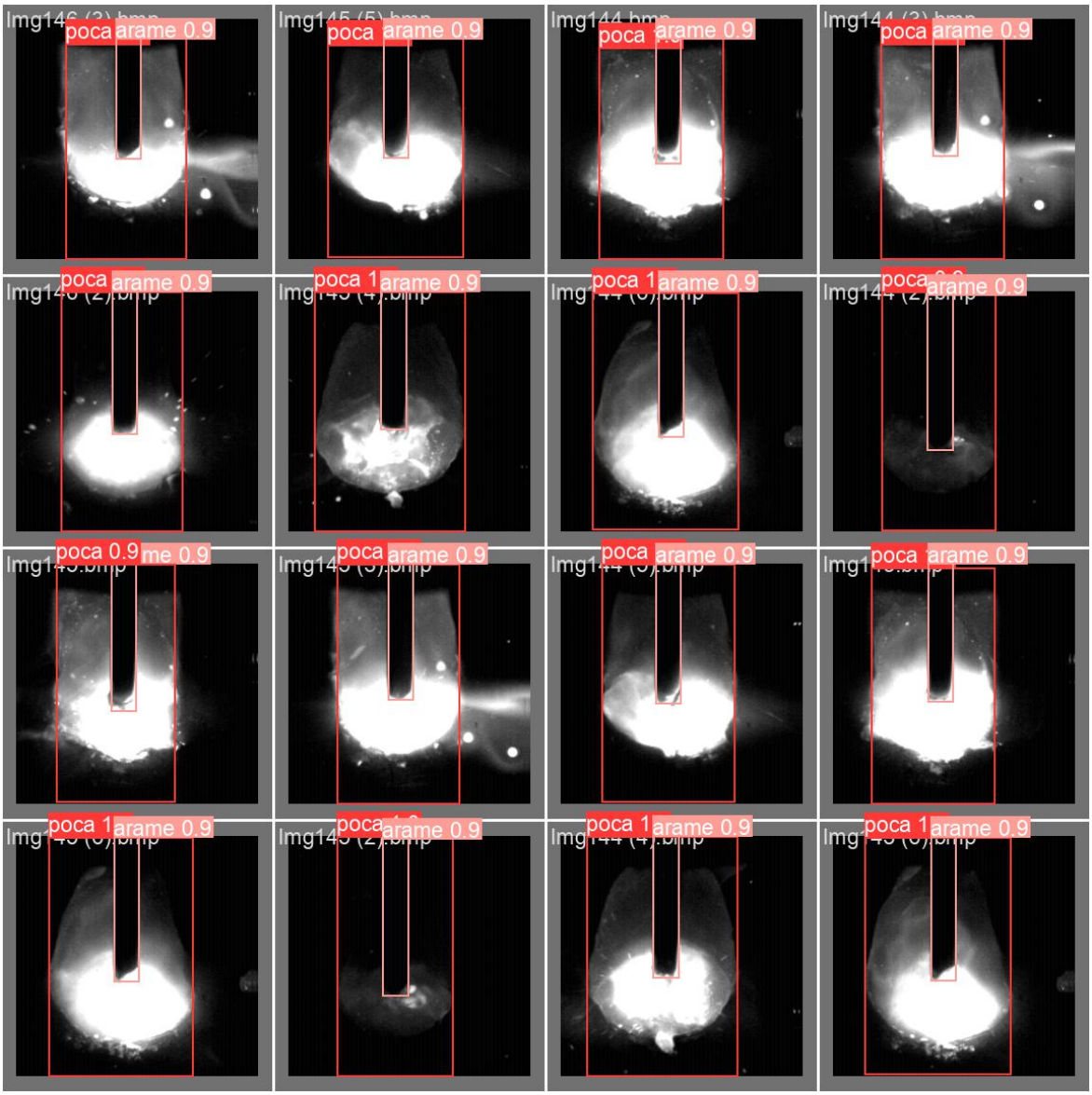
Fonte: THE ASIMOV INSTITUTE (2016)

Figura A.52 – Conjunto de imagens de validação com a rotulagem esperada



Fonte: Produzido pelo autor

Figura A.53 – Conjunto de imagens de validação após a rotulação da rede



Fonte: Produzido pelo autor