INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL-MECÂNICA
CURSO DE GRADUAÇÂO EM ENGENHARIA MECATRÔNICA

HASSAN IMAD HAMDAN

COMPARAÇÃO ENTRE ABORDAGENS BASEADAS EM APRENDIZADO DE MÁQUINA E EM VISÃO COMPUTACIONAL CLÁSSICA NA LOCALIZAÇÃO DE REBITES METÁLICOS

FLORIANÓPOLIS, 2022

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL-MECÂNICA
CURSO DE GRADUAÇÂO EM ENGENHARIA MECATRÔNICA

HASSAN IMAD HAMDAN

COMPARAÇÃO ENTRE ABORDAGENS BASEADAS EM APRENDIZADO DE MÁQUINA E EM VISÃO COMPUTACIONAL CLÁSSICA NA LOCALIZAÇÃO DE REBITES METÁLICOS

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Bacharel em Engenharia Mecatrônica.

Orientador:
Prof. Delcino Picinin Junior, Dr.Eng.

Coorientador:
Felipe Alfredo Nack, Eng.

FLORIANÓPOLIS, 2022

Ficha de identificação da obra elaborada pelo autor.

Hamdan, Hassan
COMPARAÇÃO ENTRE ABORDAGENS BASEADAS EM APRENDIZADO DE MÁQUINA E EM VISÃO COMPUTACIONAL CLÁSSICA NA LOCALIZAÇÃO DE REBITES METÁLICOS / Hassan Hamdan; orientação de Delcino Picinin Junior; coorientação de Felipe Alfredo Nack. - Florianópolis, SC, 2022.
67 p.
Trabalho de Conclusão de Curso (TCC) - Instituto Federal de Santa Catarina, Câmpus Florianópolis. Bacharelado em Engenharia Mecatrônica. Departamento Acadêmico de Metal Mecânica.
Inclui Referências.

1. Visão Computacional. 2. Aprendizado de Máquina. 3. Aprendizado Profundo. 4. Detecção de Coordenadas. I. Picinin Junior, Delcino . II. Alfredo Nack, Felipe . III. Instituto Federal de Santa Catarina. IV. COMPARAÇÃO ENTRE ABORDAGENS BASEADAS EM APRENDIZADO DE MÁQUINA E EM VISÃO COMPUTACIONAL CLÁSSICA NA LOCALIZAÇÃO

COMPARAÇÃO ENTRE ABORDAGENS BASEADAS EM APRENDIZADO DE MÁQUINA E EM VISÃO COMPUTACIONAL CLÁSSICA NA LOCALIZAÇÃO DE REBITES METÁLICOS

HASSAN IMAD HAMDAN

Este trabalho foi julgado adequado para obtenção do título de Engenheiro em Mecatrônica e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Mecatrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 14 de Dezembro, 2022.

Banca Examinadora:

gov.br
Documento assinado digitalmente
DELCINO PICININ JUNIOR
Data: 06/02/2023 12:32:05-0300
Verifique em https://verificador.iti.br

Delcino Picinin Junior, Dr.Eng.

ICP-Edu
Documento assinado digitalmente
Felipe Alfredo Nack
Data: 06/02/2023 21:08:09-0300
CPF: ***.213.429-**
Verifique as assinaturas em https://v.ufsc.br

Felipe Alfredo Nack, Eng.

GOV.BR
Documento assinado digitalmente
FRANCISCO RAFAEL MOREIRA DA MOTA
Data: 06/02/2023 18:50:42-0300
CPF: ***.003.153-**
Verifique as assinaturas em https://v.ufsc.br

Francisco Rafael Moreira da Mota , Dr.Eng.

gov.br
Documento assinado digitalmente
RAIMUNDO RICARDO MATOS DA CUNHA
Data: 06/02/2023 17:40:58-0300
Verifique em https://verificador.iti.br

Raimundo Ricardo Matos da Cunha , Dr.Eng.

AGRADECIMENTOS

Aos meus pais Mona Safadi e Imad Hamdan, por demonstrarem apoio as minhas decisões, e por serem pessoas incríveis que me criaram e educaram, resultando na pessoa que sou, sempre lutando para me oferecer tudo do bom e do melhor.

A minhas irmãs Sarah Hamdan e Amira Hamdan, por estarem presentes em todos os momentos ao longo de minha vida.

Aos meus amigos e colegas, por me acompanharem durante toda minha jornada acadêmica, contribuindo com apoio, durante momentos de frustração.

Ao professores Maurício Stivanello e Delcino Junior, por toda orientação e suporte durante o desenvolvimento deste projeto.

A todos os professores e Funcionários do Instituto Federal de Santa Catarina, por tornarem minha formação acadêmica possível.

"Se pudermos tornar os computadores mais inteligentes, e entender melhor o mundo e o meio ambiente, isso pode tornar tornar a vida muito melhor para muitos de nós."
(Andrew Ng)

RESUMO

A necessidade por linhas de produção mais rápidas e eficientes em diversos setores da indústria, torna necessário a implementação de Sistemas de Visão, podendo estes serem responsáveis pela detecção ou classificação de tais produtos. Entretanto, apesar do baixo custo de implementação, abordagens clássicas que usufruem de métodos mais convencionais, podem não ser a solução mais viável se comparadas com a utilização de Aprendizado de Máquina e Aprendizado Profundo.

Sendo assim, este trabalho adota uma abordagem utilizando Aprendizado de Máquina, mais especificamente Redes Neurais Convolucionais, realizando uma comparação com uma abordagem adotando a Visão Computacional Classica. Tal perspectiva de solução com *machine learning* procura contornar as deficiências encontradas na Visão Computacional Clássica que demanda da utilização de métodos mais complexos e demasiadamente sensíveis, suscetíveis a ruídos.

Palavras-chave: Sistemas de Visão, Visão Computacional, Aprendizado de Máquina, Redes Neurais Convolucionais, Aprendizado Profundo, Detecção de Coordenadas.

ABSTRACT

The need for faster and more efficient production lines in various sectors of the industry, makes it necessary to implement vision systems, which may be responsible for the detection or classification of such products. However, despite the low cost of implementation, classic approaches that take advantage of more conventional methods may not be the most viable solution if detected using Machine Learning and Deep Learning.

Therefore, this work adopts an approach using Machine Learning, more specifically Convolutional Neural Networks, performing a comparison with an approach adopting the Classical Computer Vision. The solution's perspective with machine learning seeks to circumvent the shortcomings found in classical computational vision that demand the use of more complex and overly sensitive methods, susceptible to noise.

Keywords: Computer vision, Machine Learning, Convolutional Neural Networks, Deep Learning, Coordinate Detection.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
IFSC	Instituto Federal de Santa Catarina
RNAs	Redes Neurais Artificiais
RNCs	Redes Neurais Convolucionais
Resnet	Redes Neurais Residuais

SUMÁRIO

1 INTRODUÇÃO

Desde o final do século XVIII, com o início da Primeira Revolução Industrial, nota-se uma grande mudança nos meios de produção, e a grande competição entre empresas estimula, cada vez mais, linhas de produção mais eficientes. Sendo assim, processos automatizados vêm sendo implementados ao longo de diversas áreas da indústria, eliminando a necessidade de um trabalhador humano executando tarefas, muitas vezes, repetitivas e assim, diminuindo as taxas de erro durante o processo.

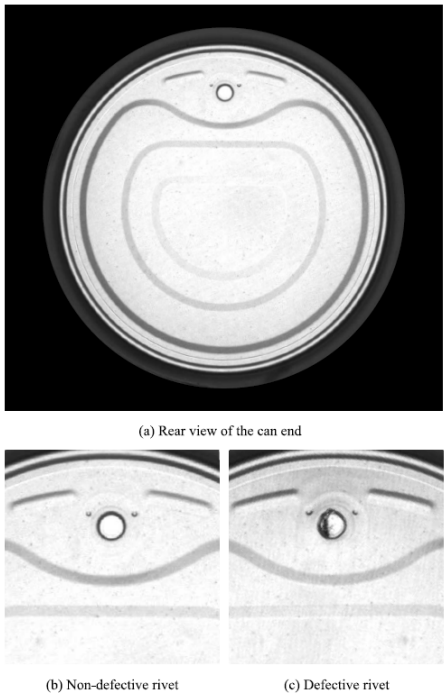
Devido a esses fatores, a aplicação de visão computacional na indústria cresce ano após ano. Onde muitas vezes era necessário um operário fazendo a verificação visual do produto em uma linha de produção, hoje foram substituídos por sistemas de visão responsáveis desde a contagem para controle até a detecção de defeitos e anomalias no produto. Ambos os cenários possuem suas vantagens e desvantagens, sendo a verificação manual a mais barata porém menos eficiente, não satisfazendo as necessidades do mercado altamente competitivo em que vivemos.

O uso de aprendizado de máquina pode ser uma solução mais cara se comparada com a inspeção visual realizada por um operário. Isso deve-se a fatores técnicos e de infraestrutura necessários para realização de tal solução. Dentre essas razões, estão a necessidade de computadores com grande capacidade de processamento para o tratamento da base de dados por parte da máquina e, além disso, as alterações necessárias na linha de produção, como, por exemplo, a instalação de dispositivos de captura de imagem.

Os problemas acarretados por produtos defeituosos em uma linha de produção podem causar altos prejuízos em empresas de maior escala, causando a urgência de implementar sistemas de detecção automatizados. A aplicação de tais soluções variam, podendo ser implementadas utilizando visão computacional na obtenção de bordas e assim como ilustrado na Figura 1, detecção de defeitos , ou em certos casos, a obtenção de coordenadas. Caso seja feito o uso de tecnologia como o Aprendizado de Máquina, soluções como segmentação e classificação de objetos, se tornam possíveis de serem aplicadas.

A tecnologia de *Machine Learning*, ou em português, Aprendizado de Máquina é baseada no uso de Inteligência Artificial (IA), permitindo que "computadores aprendam por conta própria", por meio do uso de bases de dados (*datasets*) e reconhecimento de padrões. Devido a esse “aprendizado”, a eficiência destes sistemas pode aumentar seguidamente, obtendo resultados mais precisos e confiáveis ao longo do tempo.

Figura 1 – Exemplo de aplicação para detecção de defeitos



Fonte: Stivanello e Marcelino (2019).

Dentre os tipos de tarefas incumbidas a um sistema de visão estão a classificação, avaliação e estimação da localização e orientação de peças e partes. A classificação é muito utilizada para indicar a qual classe uma dada peça pertence, por exemplo. Por sua vez, a localização é utilizada para encontrar a posição na qual determinada peça ou objeto se encontra na imagem. No presente trabalho pretende-se explorar a possibilidade de obter valores de posicionamento de um objeto específico, no caso, o rebite, através da regressão obtida por Redes Neurais Convolucionais , voltada para aplicações industriais.

Além disso, outro ponto que vale ser destacado, é a suscetibilidade da visão computacional clássica a ruídos. Por utilizar de métodos e algoritmos que usufruem de regras predefinidas e sensíveis, a alternativa em questão pode encontrar dificuldades para detecção de pontos devido a grande quantidade de ruídos e poluição visual em uma imagem. Tais problemas não afetam com tanta intensidade ao utilizar a abordagem de aprendizado de máquina, pois a mesma utiliza uma base de dados como fonte do aprendizado, possibilitando se adaptar a tais ruídos.

1.1 Justificativa

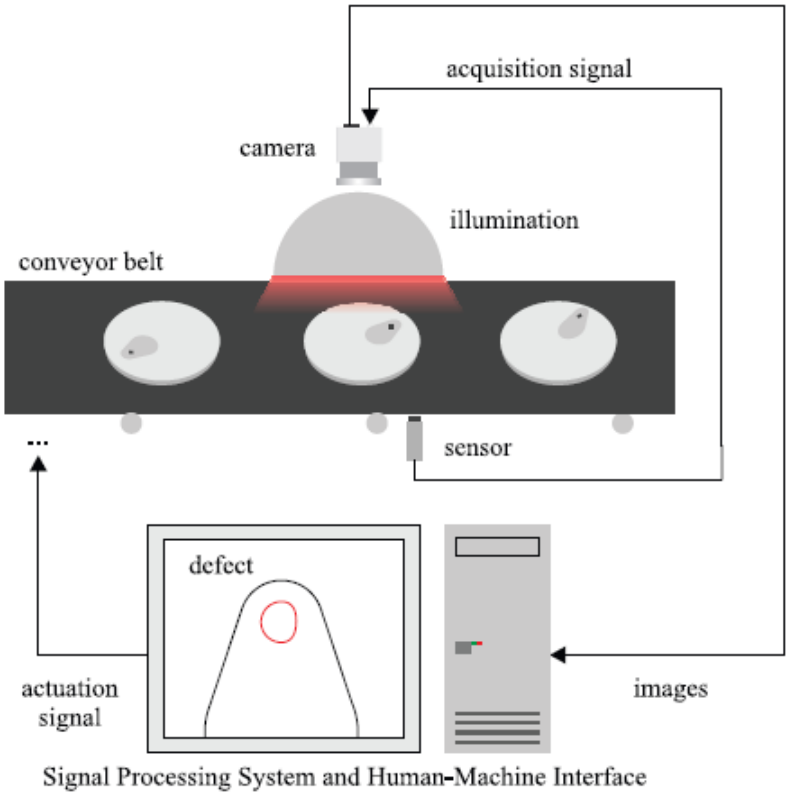
Com a crescente demanda de diversos setores da indústria, por sistemas de alta velocidade e confiabilidade de inspeção, a utilização de visão computacional aliada à técnicas de aprendizado da máquina se torna uma alternativa interessante a

ser avaliada, por já ter sido provada eficaz em diversas aplicações. Sendo assim, o tema proposto pode ser mostrar relevante no desenvolvimento de certos setores da indústria.

A visão computacional já vem sendo amplamente utilizada em diversas aplicações como na detecção e classificação de objetos, contribuindo para o desenvolvimento de sistemas autônomos e assim, aumentando a eficiência com a diminuição de erros, cortes de gastos e, consequentemente, maior lucratividade ao longo prazo.

Vale ressaltar que este projeto tem como objetivo realizar um estudo comparativo na detecção da posição através de coordenadas por meio de redes neurais convolucionais, ou do inglês, *Convolutional Neural Networks* (CNNs) e Visão Computacional Clássica. Entretanto, tal detecção pode ser de grande relevância, por exemplo, em uma linha de montagem onde o encontro de peças só ocorre se sua orientação e posição estiverem de acordo com o planejado para que o processo de montagem ocorra normalmente, pois em caso de um componente, mesmo que não defeituoso, estar com a orientação incorreta, diversos problemas serão acarretados nas etapas posteriores da montagem. Ou até mesmo, na detecção de deformação de rebite metálicos pertencentes a uma linha de produção, como é exibido na figura 2.

Figura 2 – Exemplo de aplicação de um sistema de detecção



Fonte: Stivanello e Marcelino (2019).

1.2 Objetivo Geral

Avaliar a efetividade da utilização de técnicas baseadas em aprendizado de máquina na localização de rebites, para assim, a partir dos resultados obtidos, realizar um estudo comparativo entre as solução de aprendizado de máquina e visão computacional clássica.

1.3 Objetivos Específicos

a) Estudo teórico de técnicas e trabalhos correlatos;

b) Preparação da base de imagens a ser utilizada;

c) Implementação dos algoritmos, incluindo a seleção das arquiteturas e obtenção dos modelos de RNC;

d) Validação da solução;

e) Realização de estudo comparativo dos resultados em relação a uma abordagem clássica.

2 FUNDAMENTAÇÃO TEÓRICA

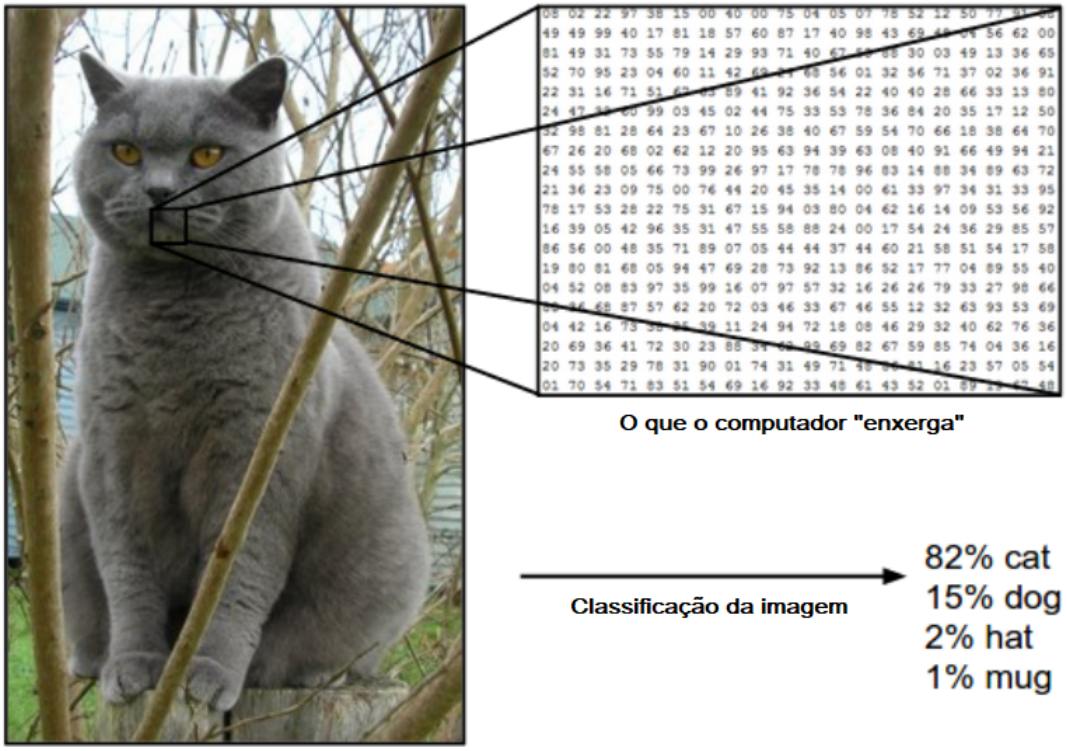
Nesta seção são abordados alguns tópicos essenciais para o entendimento do estudo realizado envolvendo Visão Computacional. Inicialmente, é apresentada a importância da Visão Computacional na indústria e conceitos básicos sobre as Redes Neurais Artificiais, especificando o tipo utilizado no projeto desenvolvido e suas características. Além disso, são também explicadas as concepções fundamentais da Visão Computacional Clássica e seus principais aspectos, com comentários sobre o método matemático da Regressão Linear, utilizado na solução de Redes Neurais Convolucionais.

2.1 A Visão Computacional na Indústria

Técnicas variadas de Visão Computacional revolucionam a indústria há décadas, principalmente nas áreas de: Inspeção de Objetos e Controle Adaptativo de Robôs. Robótica e Visão Computacional são ambas tecnologias com exponencial e contínuo avanço, tornando soluções já existentes rapidamente obsoletas. Apesar disso, tais soluções possuem ainda espaço para progresso na indústria, cada vez mais tornando-as precisas e eficientes, com uso em diversos âmbitos.

Dentre os tipos de possíveis aplicações, estão a classificação de imagens, onde uma dada imagem é identificada e, a partir de suas características, é classificada em uma classe previamente conhecida e definida durante a preparação da base de dados. Na Figura 3, é possível visualizar um exemplo de imagem que, ao ser processada por uma rede neural de classificação, retornou com 82% de eficiência que a imagem exibida é referente a um gato.

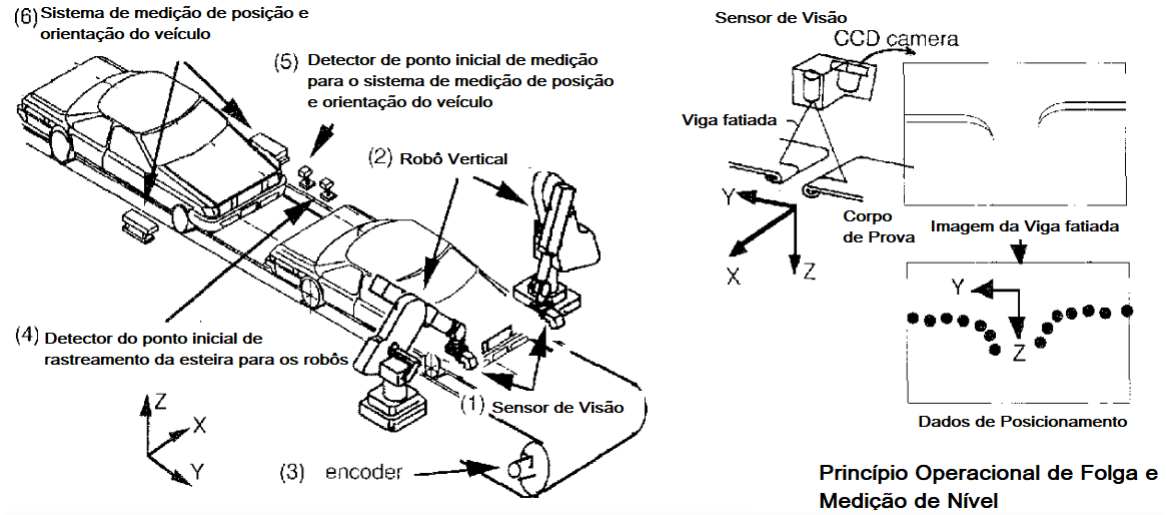
Figura 3 – Exemplo do uso de Visão Computacional para Classificação



Fonte: Adaptado de Sagar (2019).

Outra aplicação de Visão Computacional bastante utilizada, é na detecção de posicionamento e orientação de objetos. Na Figura 4, é ilustrada uma situação onde tal aplicação é empregada a uma linha de produção de veículos. Em uma linha de montagem, o posicionamento e orientação das peças envolvidas devem estar devidamente alinhadas para que o processo ocorra de acordo com esperado. Sendo assim, caso o sistema aplicado detecte que há qualquer anomalia no posicionamento de tais peças, as mesmas poderam ser rearranjadas, garantindo a fluidez do processo.

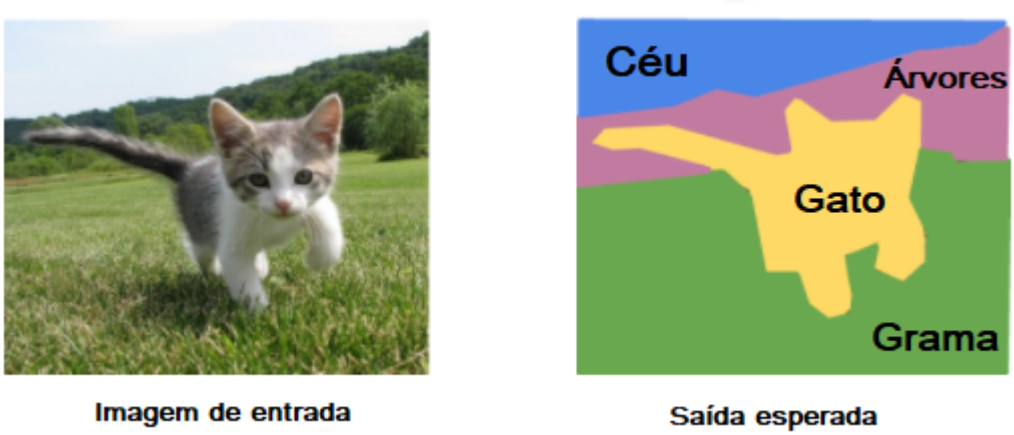
Figura 4 – Exemplo do uso de Visão Computacional para Localização



Fonte: Adaptado de M. Kondo, S. Tachiki, M. Ishida, K. Higuchi (1995).

Por fim, uma aplicação que vêm sendo aplicada em sistemas autônomos, é na segmentação de imagens. Diferente da classificação que atribui a imagem a uma classe, a segmentação é responsável por identificar e atribuir uma classe para cada *pixel* de uma dada imagem, geralmente, resultando em múltiplas classes. Na Figura 5, é possível visualizar um exemplo aplicado de segmentação, onde uma imagem de entrada de um gato em um campo aberto, resulta em 4 diferentes classes: céu, árvores, gato e grama. Vale destacar que, assim como na classificação, tais classes devem ser já previamente conhecidas e definidas.

Figura 5 – Exemplo do uso de Visão Computacional para Segmentação



Fonte: Adaptado de Great Learning Team (2022).

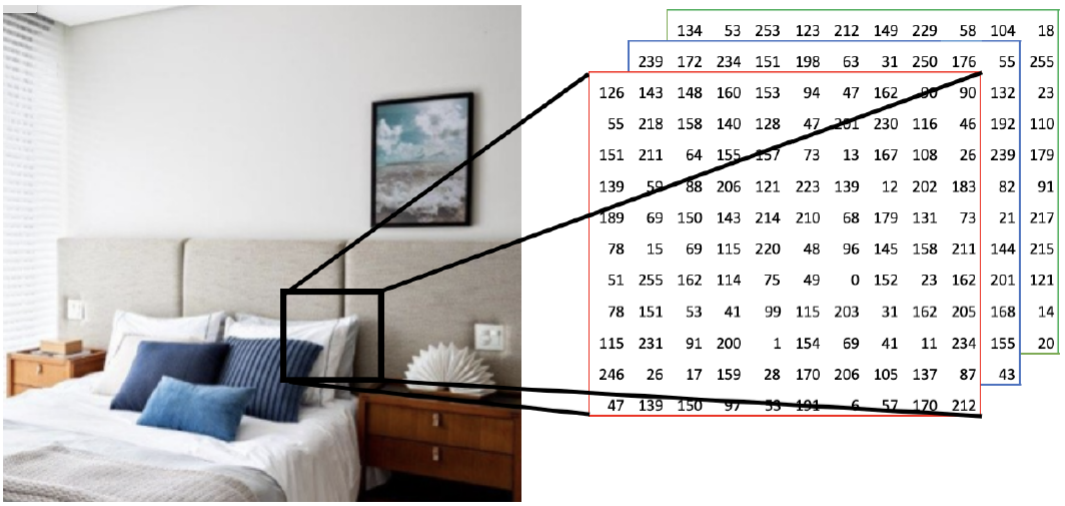
2.2 Visão Computacional e o Processamento de Imagens Digitais

Considerada a ciência responsável por envolver teorias e tecnologias para o desenvolvimento de sistemas autônomos, a Visão Computacional, como seu nome sugere, obtem os dados multidimensionais de entrada a partir de uma ou mais imagens. Para que a computação de tais dados ocorra, a imagem no formato digital (detalhada previamente na seção de Redes Neurais Convolucionais) é submetida a diversas etapas de processamento. Vale lembrar que, não foram utilizadas etapas de processamento relacionadas à operações morfológicas voltadas para correção de imperfeição da imagens, devido a aquisição de tais imagens a partir de um ambiente controlado, resultando em imagens de alta qualidade e resolução e assim, tornando desnecessário o uso de processos corretivos.

2.2.1 Representação de uma imagem digital

Com o objetivo de determinar como uma imagem é representada por um computador. Em sua forma digital, uma imagem é composta por uma matriz *n* por *m* representando seus *pixels*. Seus valores simbolizam a intensidade do respectivo *pixel*, podendo variar entre 0 e 255. A dimensão *d* dessa matriz de *pixels* varia de acordo com o tipo de imagem a ser trabalhada. Caso a mesma seja acinzentada, *d* será igual a 1, entretanto, caso a imagem seja colorida, *d* será igual a 3 como, por exemplo, no caso do sistema RGB (vermelho, azul e verde) (BRIGNOLI, 2021).

Figura 6 – Representação de uma imagem no formato digital



Fonte: Brignoli (2021).

Ao considerar uma imagem de tamanho de 64x64, ou seja, 64 *pixels* de altura e de largura, são utilizados um total de 4096 *pixels*, esses serão os dados de

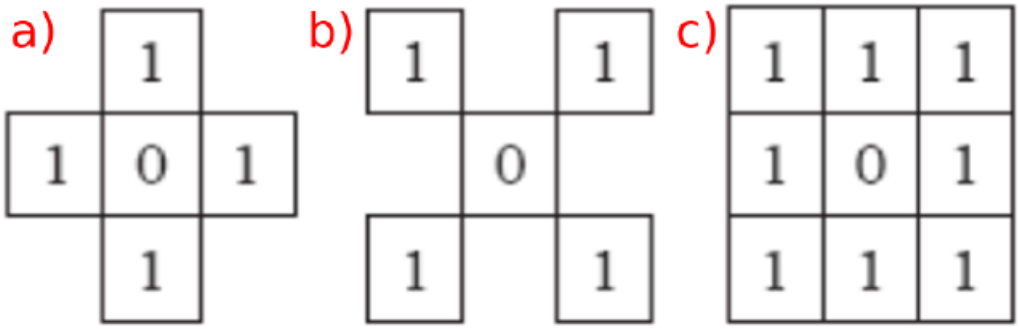
entrada em uma Rede Neural Artificial, onde cada camada está interligada e trafegando tais dados. Sendo assim, apenas na primeira camada oculta serão necessários 4096 neurônios considerando uma imagem acinzentada, enquanto no caso de uma imagem colorida esse valor triplica, ou seja, serão necessários 12288 pesos.

Nota-se que a imagem usada na exemplificação acima (64x64) possui uma resolução baixa para os padrões atuais, concluindo-se que esses números podem escalar com relativa facilidade seguindo os requisitos necessários da indústria de maneira geral.

2.2.2 Conectividade de *Pixels*

Ao visualizar uma imagem como um aglomerado de *pixels*, cada um com sua intensidade, percebe-se que esses são diretamente ligados uns aos outros. Tal conceito é chamado de conectividade ou vizinhança de *pixels*. Na Figura 7, visualiza-se que cada *pixel* possui 4 vizinhos na horizontal, 4 na vertical e 4 na diagonal, sendo a vizinhança podendo ser considerada uma dos seguintes tipos: 4-vizinhança, vizinhança diagonal ou 8-vizinhança. No caso do tipo 4-vizinhança representado na Figura 7.a, são considerados somente os vizinhos verticais e horizontais. Já no caso da vizinhança diagonal representada na Figura 7.b, são considerados apenas os vizinhos diagonais. E por fim, ilustrado na Figura 7.c, o tipo 8-vizinhança são considerados todos os vizinhos ao redor do *pixel*, ou seja, verticais, horizontais e diagonais (VALMORBIDA, 2018).

Figura 7 – Representação da conectividade de *pixels*



Fonte: Valmorbida (2018).

Matematicamente, dado um *pixel p* qualquer de coordenadas (x,y), seguindo o padrão de 8-vizinhança, é definido que o mesmo está conectado a um vizinho superior (x,y+1), um vizinho inferior (x,y-1), aos vizinhos laterais dados por (x+1,y) e (x-1,y), e por fim, os vizinhos diagonais dados por (x-1,y-1), (x+1,y+1), (x+1,y-1) e (x-1,y+1).

2.2.3 Processo de Limiarização

Considerado um dos processos mais relevantes durante o processamento de imagens está a segmentação e, ligado a ela, a técnica de limiarização ou binarização.

Com a finalidade de separar os objetos de interesse, a limiarização é uma técnica amplamente utilizada na indústria e no âmbito acadêmico, por necessitar de baixo poder computacional, barateando os custos da solução de maneira geral. Além disso, é considerada uma técnica relativamente simples de ser utilizada se comparada com as demais disponíveis.

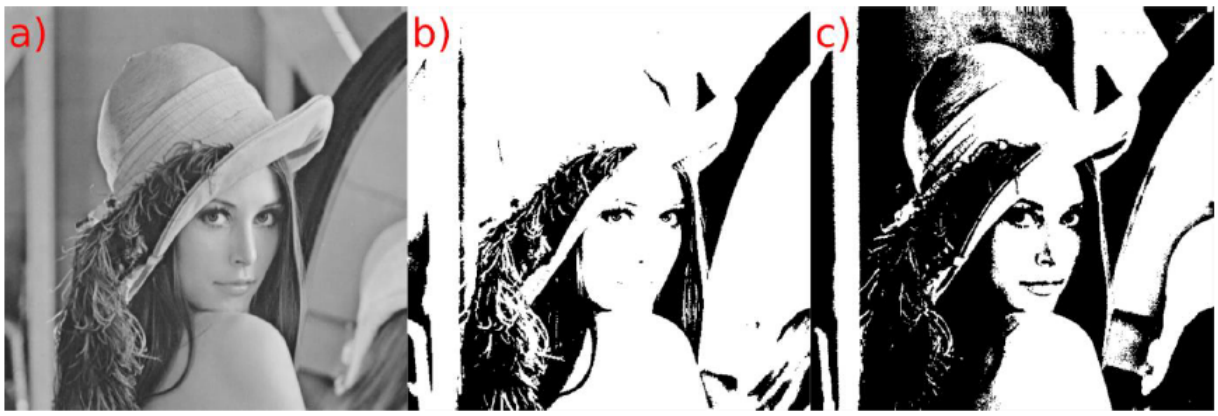
Assim como descrito por Valmorbida, a limiarização consiste em:

> "Fixar um limiar dentro dos níveis de cinza possíveis para um *pixel* de uma imagem. Após o limiar ser fixado a imagem é varrida e todos os *pixels* com valor abaixo do limiar fixado, tem seu valor ajustado para 0 e todos os *pixels* com valores acima do limiar são ajustados para o valor binário 1". (2018, v.1, p.27)

Sendo assim, percebe-se que a imagem resultante do processo de limiarização é binária, possuindo somente dois valores possíveis, onde os *pixels* pretos de intensidade 0 (ausência de cor/luz) são inerentes ao fundo da imagem, entretanto, os *pixels* brancos de intensidade 1 (presença de todas as cores/luz) são constituidores das formas relacionadas aos objetos de interesse da imagem.

Assim como é evidenciado na Figura 8, o parâmetro de limiarização é fundamental para obter uma imagem qualificada para ser devidamente trabalhada e processada.

Figura 8 – Etapas de Limiarização aplicadas a uma imagem



Fonte: Valmorbida (2018).

No caso da Figura 8.a, pode-se visualizar a imagem original que, após a aplicação da técnica de binarização, resultou nas figuras 8.b e 8.c, com o parâmetro de limiarização igual a 100 e 150, respectivamente. Sendo o último, o resultado de melhor qualidade e, consequentemente, ideal para ser trabalhado.

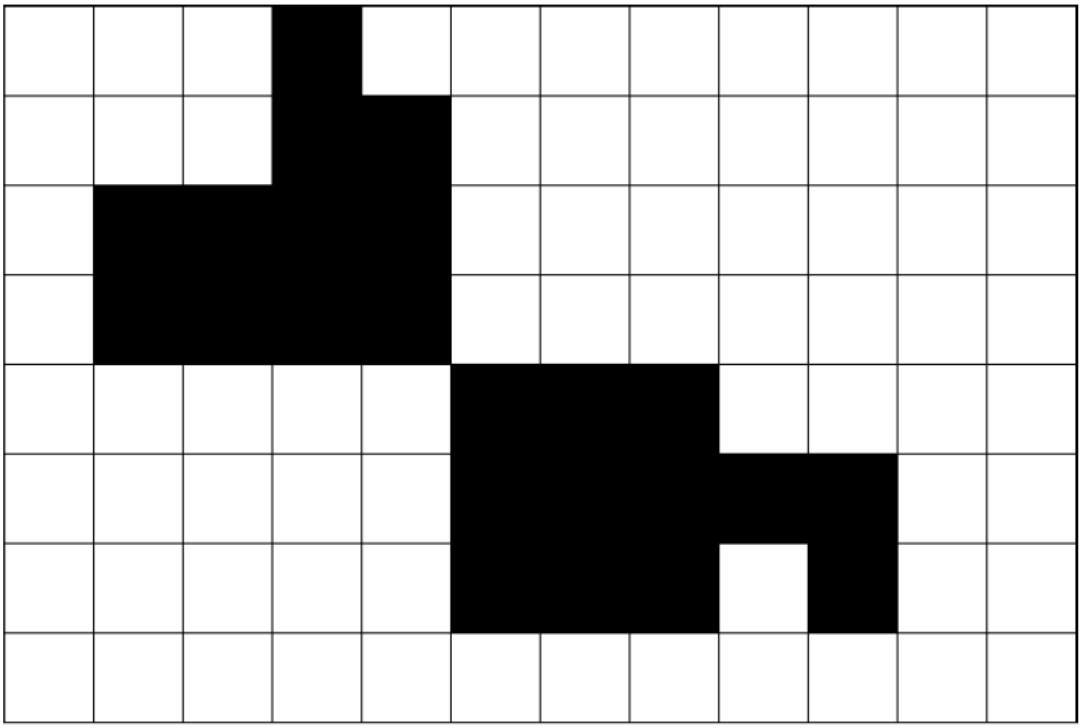
2.2.4 Rotulação de Elementos Correlacionados

Com o objetivo de realizar a diferenciação dos elementos encontrados em uma imagem, os algoritmos de rotulação por componentes conexos são bastante utilizados ao desenvolver um sistema de visão. Pois, após realizar a segmentação de uma imagem, os diversos objetos possíveis contidos na mesma não estão distintos dentre si.

Para o melhor entendimento do algoritmo de elementos conexos, além da concepção de vizinhança de *pixels*, explicada na seção anterior, também é necessário esclarecer o conceito de semelhança de *pixels*. Ao utilizar imagens acinzentadas, *pixels* equivalentes são aqueles que apresentam tons de cinza parecidos, entretanto, em imagens binarizadas, *pixels* equivalentes são aqueles que apresentam o mesmo valor de intensidade (VALMORBIDA, 2018).

Além disso, para que dois *pixels* sejam considerados conectados, ambos devem ser equivalente e compartilharem da mesma vizinhança. Na Figura 9, pode-se visualizar a diferença ao optar pelos diferentes tipos de vizinhança, sendo exibidos abaixo os tipos: 4-vizinhança e 8-vizinhança (já mencionados anteriormente).

Figura 9 – Representação dos tipos 4-vizinhança e 8-vizinhança



Fonte: Valmorbida (2018).

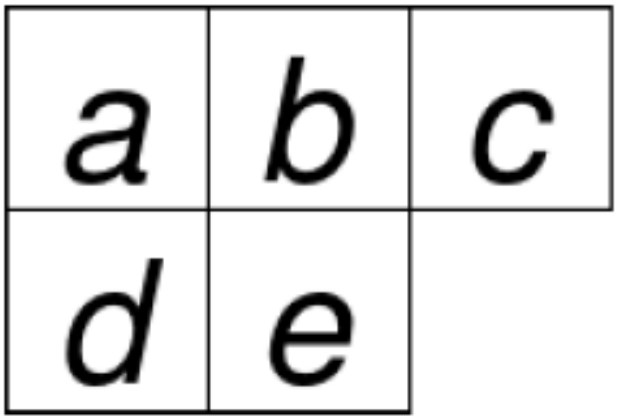
Como fica evidente na Figura acima, ao utilizar a vizinhança do tipo 4-vizinhança (aglomerado à esquerda), obtém-se dois componentes correlacionados distintos sendo considerados, diferentemente do tipo 8-vizinhança (aglomerado à

direita) que resultou apenas um componente sendo considerado. Sendo assim, conclui-se o quão relevante e essencial a escolha correta do tipo de vizinhança é para a obtenção de um resultado satisfatório.

Para a realização da rotulação dos componentes conexos, é feita uma varredura na imagem, geralmente, da esquerda para direita e de cima para baixo, podendo variar o sentido. A quantidade de varreduras realizadas também é variável, sendo realizadas uma (*one-pass* ou passada única) ou mais (*multi-pass* ou passadas múltiplas). Tal varredura tem como objetivo a atribuição de rótulos provisórios a cada *pixel* pertencente à imagem que são considerados parte de um objeto.

Além dos algoritmos de passada única ou passadas múltiplas que realizam uma ou múltiplas varreduras, respectivamente, há também o algoritmo de passada dupla que, como o nome sugere, realiza duas varreduras à imagem. Tal algoritmo é considerado de fácil implementação e, por isso, é o mais utilizado de forma geral. Na Figura 10, para melhor entendimento do algoritmo, é exibida uma representação do *pixel* a ser examinado "*e*"e seus respectivos vizinhos (considerando uma vizinhança do tipo 8-vizinhança).

Figura 10 – Máscara de *pixels* a serem analisados



Fonte: Valmorbida (2018).

Considerando que os *pixels* da Figura acima são provenientes de uma imagem binarizada, caso o *pixel e*, ao ser analisado, retorne o valor 0, ou seja, faz parte do fundo da imagem, conclui-se que o mesmo não faz parte de um objeto e não possui rótulos a serem atribuídos, sendo assim, permanecendo com seu valor igual a 0. Entretanto, caso o valor referente ao *pixel e* seja igual a 1, ou seja, indicando a participação do mesmo em um objeto, serão analisados os *pixels* vizinhos "*a*", "*b*", "*c*"e "*d*"à procura de rótulos já previamente atribuidos. Caso nenhum deles possua qualquer atribuição de rótulos, o *pixel "e"* será rotulado novamente. Em contrapartida, caso apenas um dos *pixels* vizinhos possua um rótulo, o *pixel "e"* recebe este mesmo

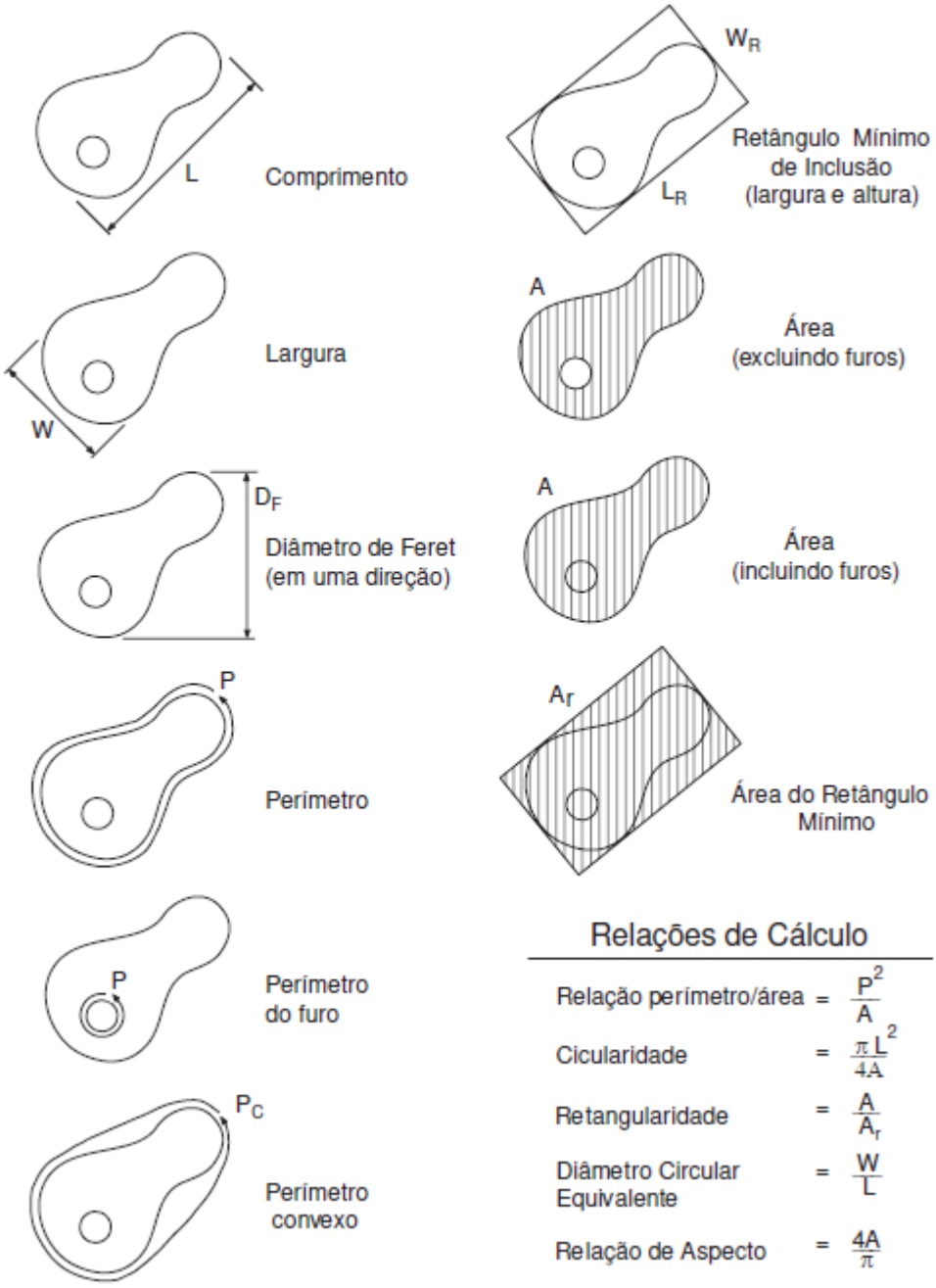
rótulo. Por fim, caso um dos *pixels* vizinhos possua múltiplos rótulos, o *pixel "e"* recebe o rótulo de menor valor e a relação de equivalência entre rótulos é armazenada em um vetor (VALMORBIDA, 2018).

Sendo finalizado o processo descrito (*two-passing*) e, consequentemente, todos *pixels* temporariamente rotulados, uma segunda varredura é feita e cada rótulo previamente atribuído é revisado a partir de uma consulta no vetor de equivalência, por fim, atribuindo um rótulo final e definitivo para o *pixel*.

2.2.5 Descritores

Por definição, descritores são quaisquer parâmetros que possam descrever qualquer característica de uma imagem. Anteriormente, foi apresentado o processo de varredura realizado nos *pixels* de uma imagem, rotulando-os. Feito tal procedimento, são obtidos descritores responsáveis por fornecerem dados relevantes para análise da imagem a ser estudada. Na Figura 11, é possível visualizar os descritores mais utilizados em aplicações de sistemas de visão.

Figura 11 – Descritores mais comuns



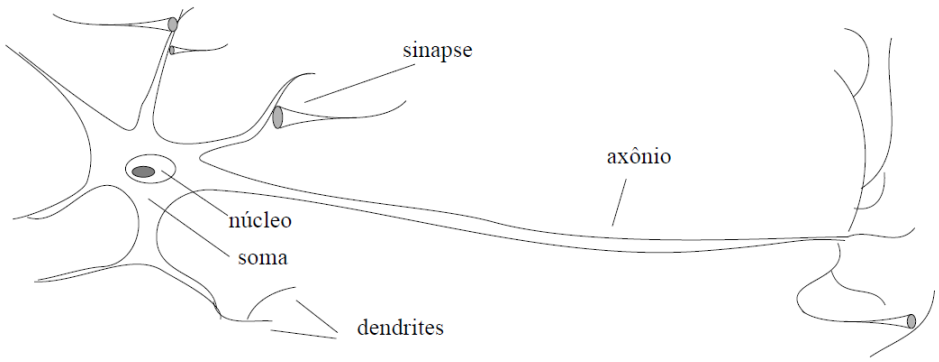
Fonte: Esquef (2003).

Além dos descritores exibidos na Figura acima, estão também dentre os mais comuns: a *bounding box* que é o descritor representado a partir de um quadrado ao redor do objeto a ser analisado e a circularidade, que é o descritor representado por indíce (que varia de 0 a 1) obtido através de uma relação entre descritores de comprimento e área, indicando o quão próximo de um círculo o objeto detectado está. Vale notar que, a combinação de descritores pode resultar em novos descritores, assim como ocorre com a circularidade.

2.3 Redes Neurais Artificias

As Redes Neurais Artificais (RNAs) ou Redes Neuronais Artificiais são modelos computacionais com a capacidade de realizar o aprendizado da máquina a partir do reconhecimento de padrões provenientes de uma base de dados.Tais redes, buscam emular uma rede de neurônios biológicos presentes no sistema nervoso central dos seres humanos, podendo ser visualizada na Figura 12.

Figura 12 – Estrutura de um Neurônio Biológico



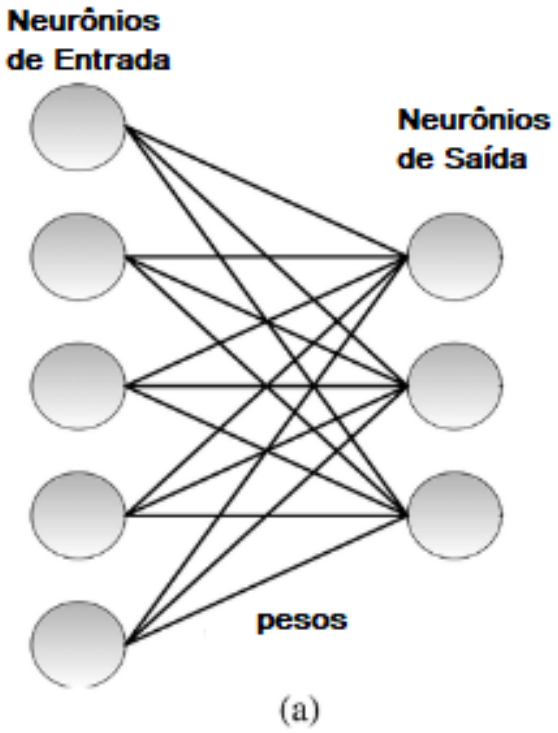
Fonte: Rauber (2005).

2.3.1 Descrição Geral de uma RNA

De acordo com Rauber (2005, v.1, p5) "uma rede neural artificial (RNA) tem duas facetas elementares: a arquitetura e o algoritmo de aprendizagem. Essa divisão surge naturalmente pelo paradigma como a rede é treinada". Ao contrário de um computador que segue a arquitetura de Von Neumann, que é previamente programado, uma rede neural é treinada através de exemplos de padrões, sendo assim, o conhecimento a ser adquirido tem que estar contido dentro destes exemplos.

A rede neural artificial mais básica é conhecida como Perceptron, modelo proposto em 1959 por Frank Rosenblatt. Tal modelo é capaz de realizar tarefas de classificação, atribuindo rótulos para objetos que podem pertencer a diferentes variedades de classes. Assim como é possível visualizar na Figura 13, o Perceptron básico é formado por duas camadas de neurônios: a de entrada (sensorial), responsável pela distribuição das entradas e a camada de saída (computacional), responsável pela rotulação das classes. Vale ressaltar que cada entrada está relacionada a um peso, responsável por multiplicar tal entrada durante o processamento, realizando uma combinação linear de tais valores. Tal relação é descrita por Ferneda (2006, v.1, p.26) como "o comportamento das conexões entre os neurônios é simulado por meio de seus pesos. Os valores de tais pesos podem ser negativos ou positivos, dependendo de as conexões serem inibitórias ou excitatórias".

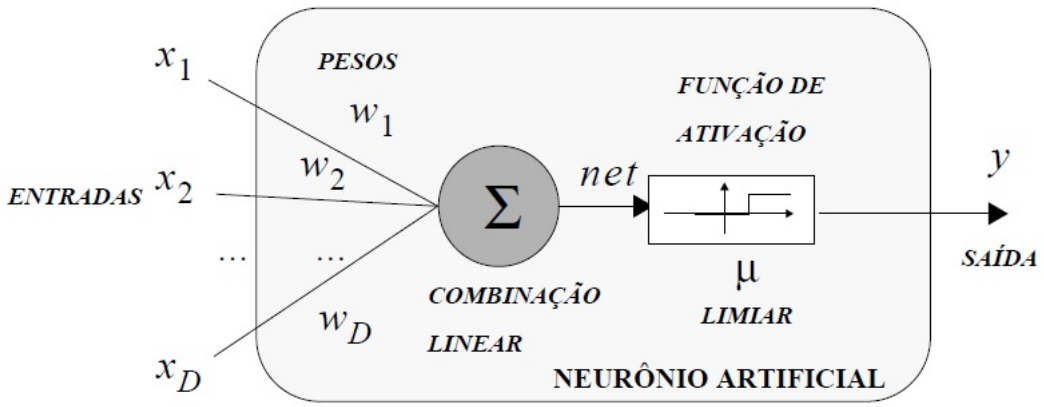
Figura 13 – Arquitetura de um Perceptron básico



Fonte: Adaptado de Camunãs-Mesa, Linares-Barranco e Serrano-Gotarredona (2019).

Na Figura 14, pode-se observar uma combinação conhecida por *net*, que é aplicada a uma função de ativação, sendo essa responsável pelo processamento matemático de cada Perceptron, simulando as sinapses químico-elétricas realizadas entre neurônios.

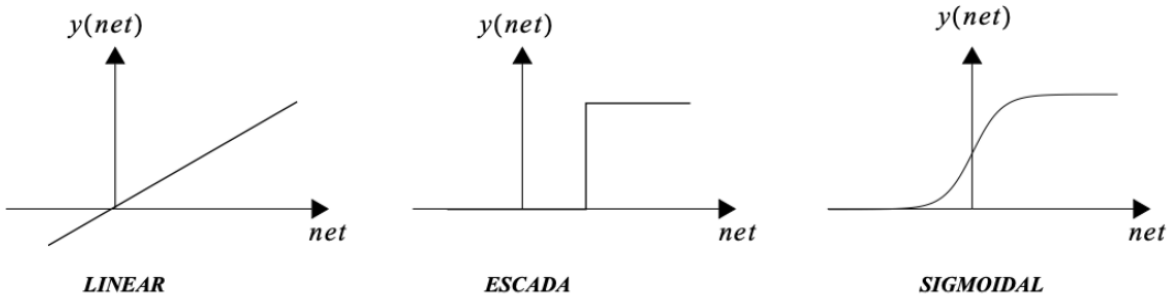
Figura 14 – Detalhamento da arquitetura de um Perceptron



Fonte: Rauber (2005).

Dentre as funções de ativação mais comuns está a função sigmoidal. Entretanto, assim como ilustrado na Figura 15, existem outras funções com características e propriedades distintas. Para escolher uma função dentre as alternativas disponíveis, é necessária a realização de um estudo nos dados de entrada da rede neural, pois determinada entrada funciona de maneira mais otimizada com determinado tipo de função. Na imagem abaixo, pode-se observar no eixo Y dos gráficos exibidos a saída conhecida por *ynet*, que por sua vez é responsável por prever se a o dado de entrada pertence a uma categoria de interesse específica ou não.

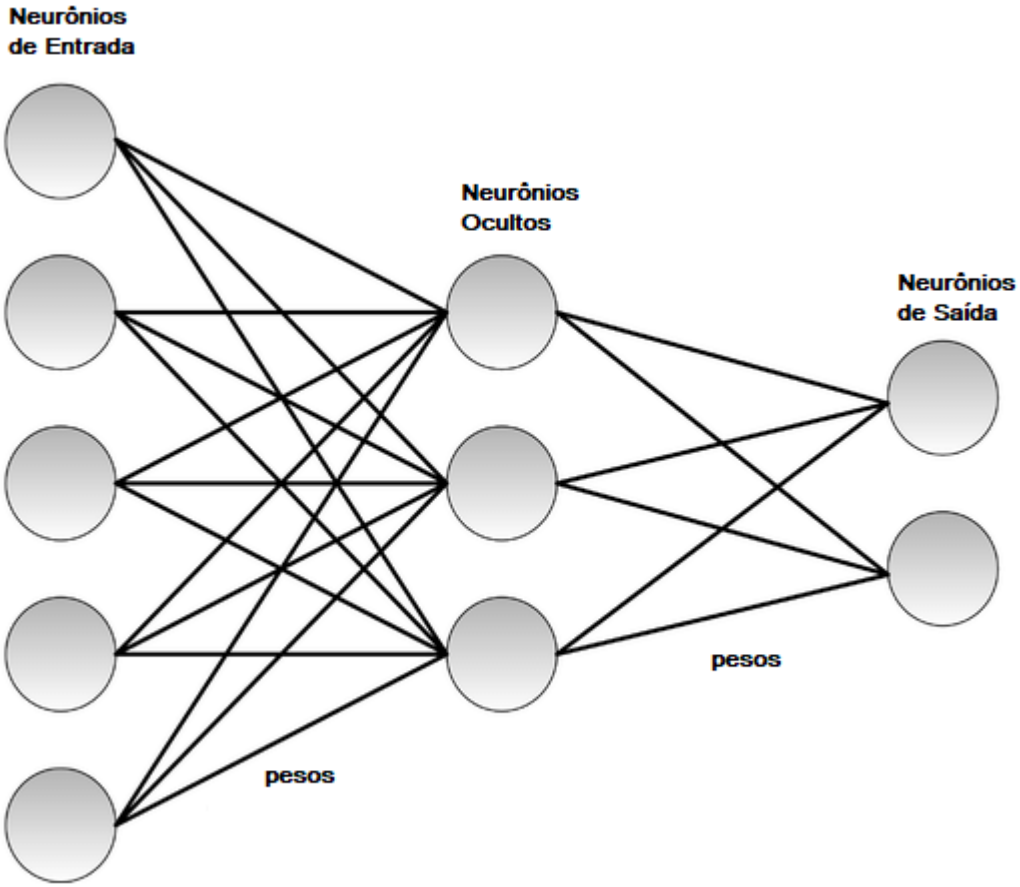
Figura 15 – Funções de ativação mais utilizadas



Fonte: Rauber (2005).

Já em 1986, uma nova abordagem ao Perceptron foi proposta, conhecida como *Multilayer Perceptron* (MLP), ou do português, Perceptron Multicamadas. O MLP apresenta uma topologia de pelo menos 3 camadas, tendo as já conhecidas camadas de entrada e saída aliadas a uma ou mais camadas extras, conhecidas como camadas ocultas, exibidas na Figura 16. Assim como na de entrada, uma camada oculta é composta por neurônios computacionais, responsáveis pelo mapeamento intermediário do problema, realizando uma separação linear para camada de saída.

Figura 16 – Arquitetura de um Perceptron Multicamadas

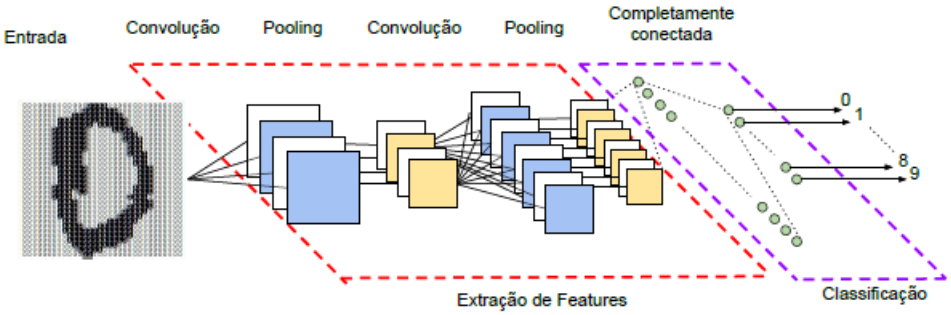


Fonte: Adaptado de Camunãs-Mesa, Linares-Barranco e Serrano-Gotarredona (2019).

2.3.2 Redes Neurais Convolucionais

Uma Rede Neural Convolucional ou, do inglês, *Convolutional Neural Network* (CNN), é uma classe de rede neural artificial que possuem múltiplas camadas, incluindo camadas convolucionais, camadas completamente conectadas, camadas não-lineares e camadas de agrupamento (*pooling*), sendo que as duas últimas não possuem parâmetros a serem atribuídos. A recepção dos dados de entrada é realizada a partir da camada convolucional que, de acordo com Vargas, Carvalho e Vasconcelos (2016, v.1, p1) "é composta por diversos neurônios, cada um responsável por aplicar um filtro em um pedaço específico da imagem."

Figura 17 – Uma Rede Neural Convolucional e suas diferentes camadas

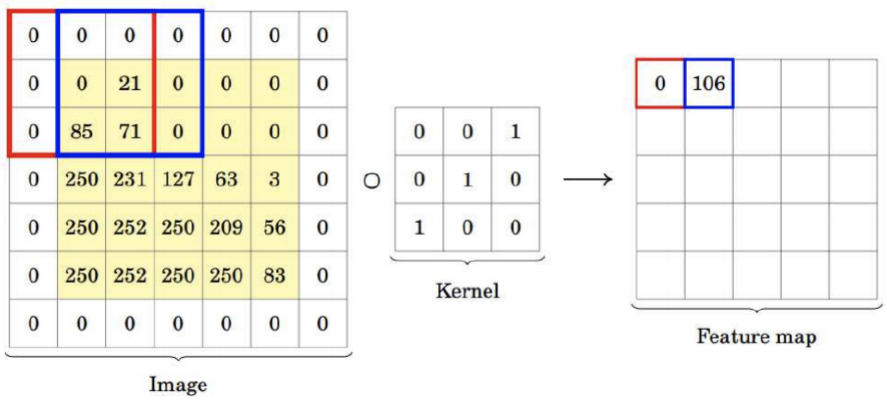


Fonte: Vargas, Carvalho e Vasconcelos (2016).

Entende-se então que, cada neurônio será atribuído a um conjunto de *pixels* da camada precedente, e que serão atribuídos pesos a tais ligações. Assim como mencionado por Vargas, Carvalho e Vasconcelos (2016, v.1, p2) "a combinação das entradas de um neurônio, utilizando os pesos respectivos de cada uma de suas conexões, produz uma saída passada para a camada seguinte.". Ressalta-se também que as atribuições de pesos às conexões podem ser entendidas como uma matriz representativa do filtro de uma convolução de imagens, também descritos na bibliografia como *kernel* ou máscara.

Tais camadas de convolução utilizam segmentos provenientes da imagem, como por exemplo blocos de *pixel* 3x3 que podem ser visualizado na Figura 18.

Figura 18 – Camada de Convolução 3x3



Fonte: Brignoli (2021).

Ainda sobre esses segmentos, pode-se afirmar que:

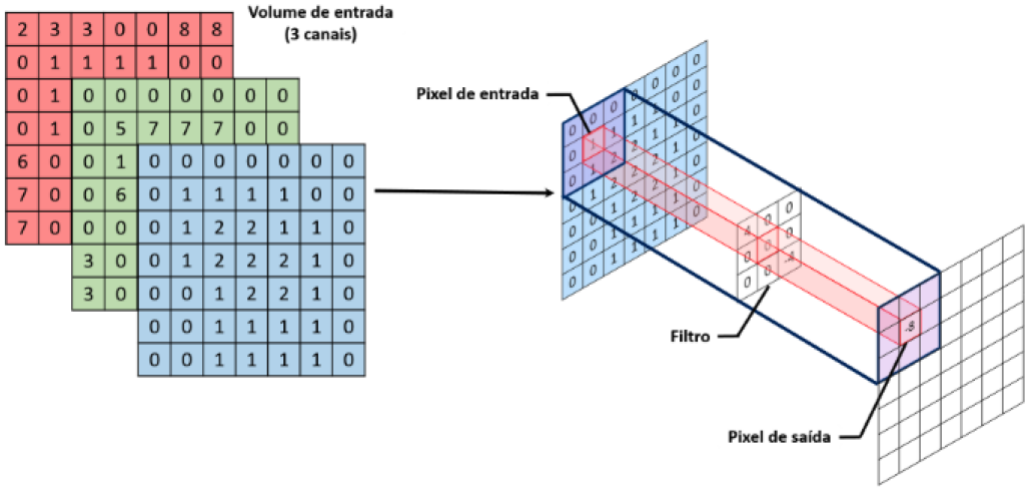
> "Estes segmentos são chamados de filtros ou kernels, e são desliza-dos sobre o volume de entrada para obter uma série de mapas de

> características, tais como uma borda de uma determinada orientação, ou uma coloração específica. Em azul observa-se o mesmo filtro, agora movimentado 1 pixel para a direita, e destinado ao segundo neurônio na camada subsequente". (BRIGNOLI, 2021, p.22)

Os filtros citados anteriormente são referentes às estruturas que se adaptam, com o objetivo de obter determinados coeficientes ao longo do processo de treinamento da rede, sendo basicamente uma matriz de valores utilizados nas operações de transformação sobre uma imagem. É possível observar na Figura 19 que o resultado proveniente de tais operações é aplicado a uma função de transformação não linear e, em seguida, disposto na camada seguinte (BRIGNOLI, 2021).

Além disso, há outras variáveis a serem ajustadas chamadas de hiperparâmetros, como o passo (*stride*), o *padding* (estratégia necessária para manter a mesma dimensão após a camada de convolução) e a quantidade de filtros a serem utilizados.

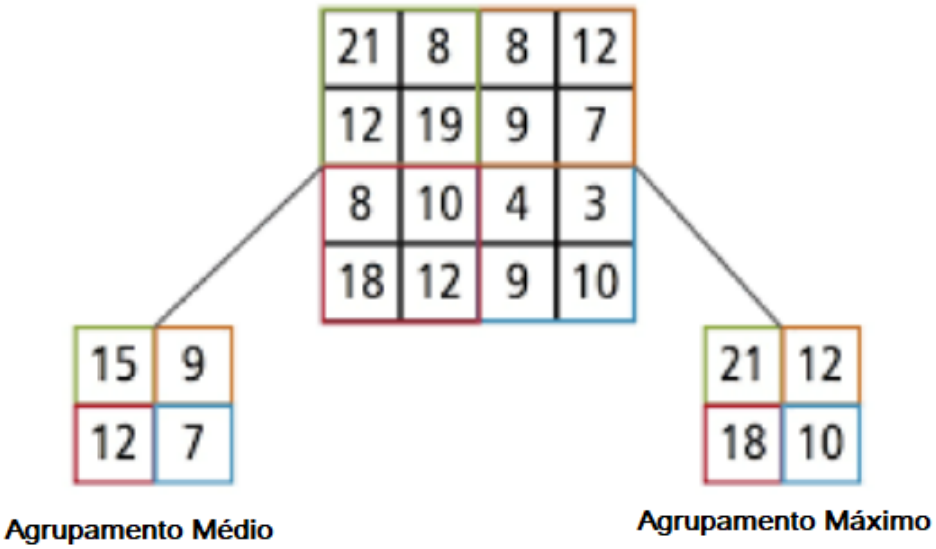
Figura 19 – Convolução entre um filtro 3x3 e o volume de entrada



Fonte: Araújo (2017).

Conhecidas pelo termo em inglês *pool*, as camadas de agrupamento tem como objetivo o controle do sobreajuste a partir de cálculos na rede e a redução no número de parâmetros. A redução é geralmente feita utilizando um filtro de tamanho 2x2, permitindo diminuir pela metade a quantidade de parâmetros. Na Figura 20, pode-se visualizar uma simplificação do que ocorre na camada de agrupamento, separada em duas classificações: o *Average Pooling* (Agrupamento Médio), do qual cada *pixel* resultante é uma média dos anteriores; e o *Max Pooling* (Agrupamento Máximo), do qual cada *pixel* resultante é equivalente ao maior valor dos anteriores (HOWARD, 2018).

Figura 20 – Visualização da camada de Agrupamento

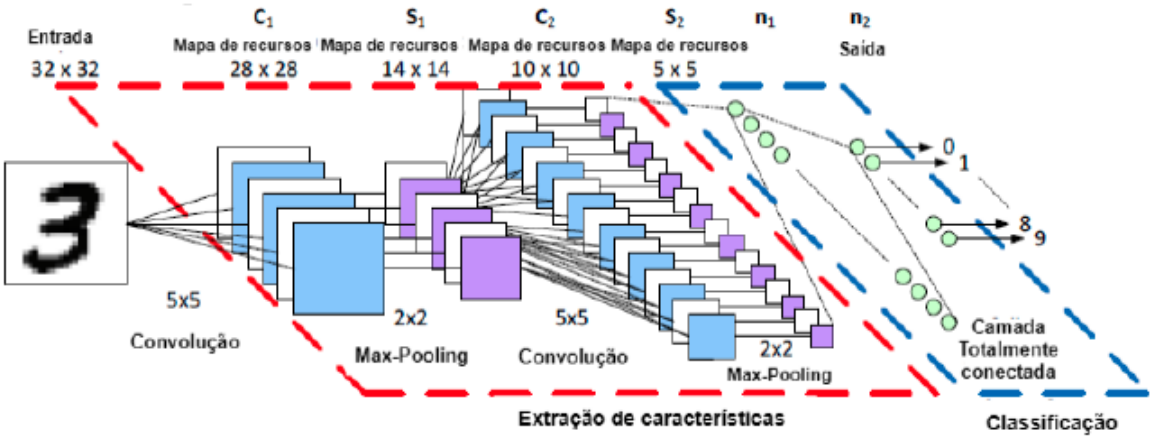


Fonte: Karpathy (2020).

Finalmente, seguidos de sequenciais camadas de convolução e agrupamentos, como é mostrado por Brignoli (2021, p.23, apud SOUZA et. al., 2020) "os mapas de características estão achatados em forma de uma matriz unidimensional, e então a camada totalmente conectada é responsável por reconectar estes dados e realizar a tarefa de classificação das imagens."

Na Figura 21 é apresentado um exemplo de aplicação de uma Rede Neural Convolucional que busca identificar valores numéricos entre 0 e 9, sendo assim, há 10 possibilidades de resultado e, consequentemente, 10 classes possíveis. Observa-se que o dado de entrada é uma imagem de tamanho 32x32 do número 3, que é, por sua vez, aplicada a primeira camada de convolução com o uso de um filtro 5x5, tendo como resultado C1 de tamanho 28x28. Após isso, é executada a camada de agrupamento com um filtro 2x2 que, conforme comentado anteriormente, tem como característica a diminuição das dimensões pela metade, ou seja, resultando em S1 de tamanho 12x12.

Figura 21 – Exemplo de aplicação de uma Rede Neural Convolucional



Fonte: Brignoli, Ramon (2021).

O processo descrito anteriormente é refeito, resultando sequencialmente em C2 e S2, para que assim, a etapa final possa ser realizada, onde a camada conectada recebe os neurônios de saída (círculos verdes) e os classifica de acordo com as classes previamente definidas durante o processo de treinamento (BRIGNOLI, 2021).

2.3.3 Redes Neurais Residuais (Resnet)

Ao usar um número elevado de camadas em uma rede, a mesma estará sujeita a alguns problemas relacionados ao treinamento: *overfitting* e o *underfitting*, que são relacionados ao treinamento excessivo ou pouco treinamento da rede, respectivamente. O *overfitting* pode ser causado pelo uso demasiado de neurônios que, por sua vez, são treinados em um conjunto limitado de dados. Já o *underfitting* é geralmente ocasionado ao se utilizar um número baixo de camadas e, consequentemente, um número insuficiente de neurônios necessários para haver um processamento adequado do conjunto de dados.

Outro problema muito comum em Redes Neurais mais profundas é o da degradação. A degradação ocorre ao adicionar um grande número de camadas a uma rede de arquitetura simples, acarretando no gradiente de desaparecimento, proveniente da gradativa perda de informações. Tal problema pode levar a um tempo de treinamento muito elevado, exigindo grande poder computacional de processamento.

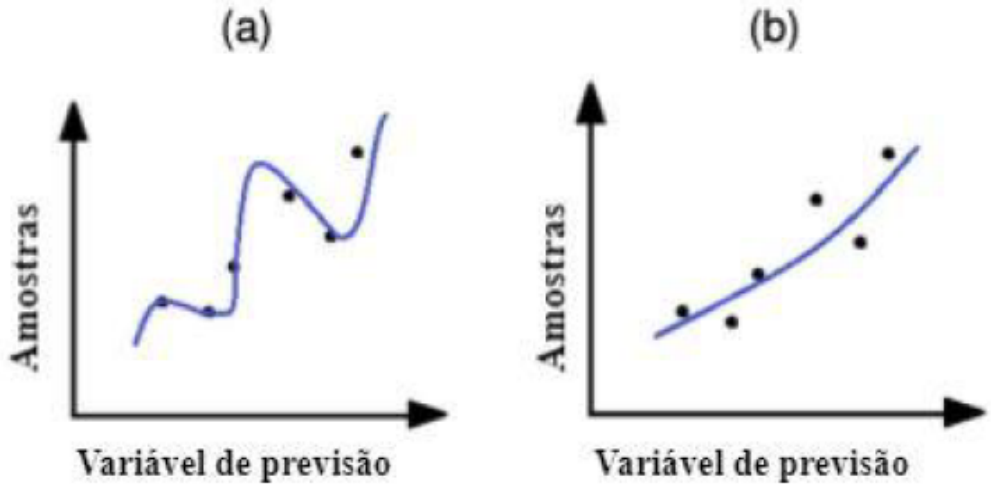
Uma maneira encontrada para aliviar tais problemas, foi o desenvolvimento das Redes Neurais Residuais, mais conhecidas pela abreviatura Resnet que, assim como descrito por Brignoli:

> "É construída por blocos residuais que realizam operações em uma entrada x. O resultado da operação f(x) é adicionado à entrada

> original x, e esta solução indica que um modelo mais profundo não deve produzir um erro de treinamento maior do que sua versão da rede mais rasa". (2021, p.22, HE et. al, 2016)

Sendo assim, percebe-se que a Degradação pode ser uma barreira para obtenção de resultado precisos em redes muito profundas, devido a alocação exagerada de camadas que faz com que, ao invés de focar no sinal principal, ruídos sejam memorizados, ocasionando numa queda intransigente de performance.

Figura 22 – Representação de (a) um modelo com memória e (b) um modelo exato



Fonte: Lemos (2021).

Pode-se perceber na Figura 22 que o modelo "a", por possuir um número elevado de camadas em sua composição, realiza ajustes excessivos no processo de previsão. Entretanto, o modelo "b", por possuir um número inferior de camadas, é capaz de traçar uma trajetória de previsão mais "suave", definindo com maior clareza a fronteira de decisão sem realizar demasiados ajustes.

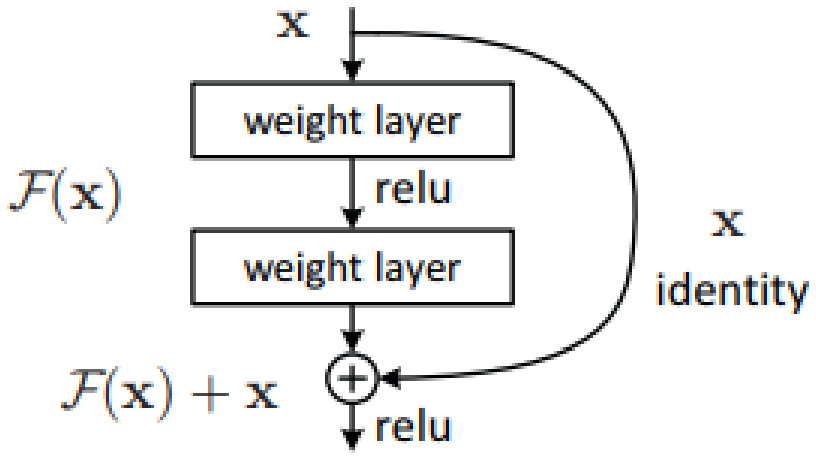
Em resposta a problemas como da Degradação, foi desenvolvida a Resnet. As Redes Neurais Residuais são um tipo de rede neural convolucional, sendo então definida pelo nome ResnetX, onde X é a quantidade de camadas presentes em sua arquitetura. Introduzidas em 2015, provinda do inglês, *Residual Neural Networks*, tornou-se rapidamente um dos modelos de *deep learning* (aprendizado profundo) mais utilizado de todos os tempos.

Destre os tipos de Resnet disponíveis, a Resnet18, que é composta por 18 camadas em sua estrutura, é considerada a menor de todas.

2.3.3.1 Blocos Residuais e Conexões Diretas

Treinar uma rede neural pode ser uma tarefa difícil. Dificuldades como o conhecido e já mencionado problema de degradação, da utilização de Redes Neurais muito profundas, foram amenizadas através da utilização dos Blocos Residuais. Assim como exibido na Figura 23, tais blocos constituem a Resnet.

Figura 23 – Blocos Residuais de uma Resnet



Fonte: He, Xiangyu, Shaoqing e Jian (2015).

Na Figura 23, percebe-se uma conexão direta que ignora certas camadas da estrutura do modelo exibido. Essa ligação direta é conhecida, do inglês, como *skip connection* e é considerada o principal diferencial dos blocos residuais, pois devido a ela o valor resultante será alterado, ignorando os pesos atribuídos às camadas, seguida da adição de um *bias term*. O *bias term* é o erro resultante ao treinar um modelo, devido à distinção entre as previsões médias e os valores corretos a serem previstos. Dada a função de ativação H(x), desconsiderando a conexão direta, obtém-se que:

$$H(x) = f(wx + b) \tag{1}$$

Entretanto, ao introduzir a conexão direta ao modelo e considerar-se que:

$$f(x) = f(wx + b) \tag{2}$$

Sendo assim, obtém-se que a função de ativação H(x) com a inserção de conexões diretas resulta em:

$$H(x) = f(x) + x \tag{3}$$

Um cuidado a ser tomado é em relação às dimensões de entrada e saída, que podem variar e se divergir ao utilizar redes neurais convolucionais. Pode-se solucionar esse problema adicionando uma camada convolucional 1x1 às entradas, combinando as dimensões. Neste caso, a saída da função de ativação H(x) é dada por:

$$H(x) = f(x) + w1x \tag{4}$$

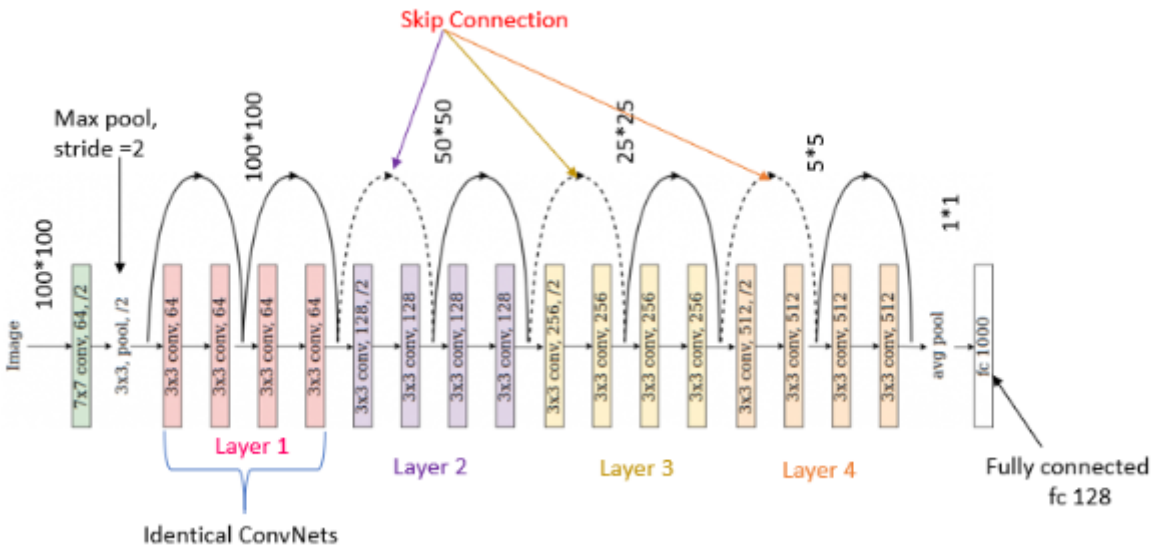
Na equação 4 um parâmetro w1 é adicionado à função, diferentemente das equações anteriores onde nenhum parâmetro adicional é incluído.

A técnica de conexões diretas é responsável por resolver o problema de fuga de gradiente ao utilizar RNCs, permitindo a alternância do fluxo do gradiente ao percorrer a estrutura da rede. Além disso, a conexão direta alivia o problema caso alguma camada da estrutura prejudique o desempenho, pois a mesma será ignorada, regulando a perfomance.

2.3.3.2 Arquitetura da Resnet18

Na Figura 24, é exibida a arquitetura de uma Rede Residual de 18 camadas, mais conhecida pelo nome Resnet18. Tal estrutura é inspirada pelo modelo VGG-19 do qual os blocos residuais e conexões diretas são adicionados. Feita a inclusão de tais elementos, a arquitetura é convertida em uma Resnet.

Figura 24 – Arquitetura da Resnet18



Fonte: Singhal (2020).

2.3.4 Tipos de Arquitetura de uma RNC

Existem diversos tipos e aplicações de sistemas de visão e, tais tipos fazem com que a arquitetura de uma RNC varie dependendo da aplicação. Assim como mostrado por Cuttle (2018, v.1, p.27): "Para cada problema, costumam ser testadas arquiteturas de maneira a encontrar a ideal para aquele específico, não havendo uma arquitetura que resolva todos os problemas". Abordado anteriormente, na Figura 21, visualiza-se a arquitetura de uma RNC de classificação, onde a modelo é responsável por atribuir a imagem a uma classe previamente definida.

Entretanto, na Figura 25, é possível visualizar uma imagem resultado de uma RNC de segmentação, por exemplo.

Figura 25 – Imagem segmentada por uma RNC

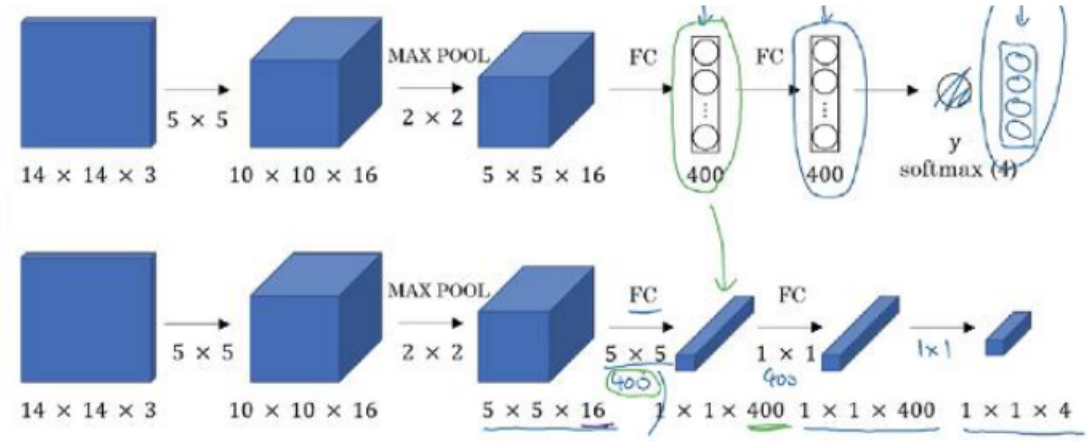
Fonte: Cunha (2020).

Neste caso, a arquitetura utilizada é conhecida como RTC, sigla proveniente de Redes Totalmente Convolucionais. Na Figura 26 é ilustrada a arquitetura de tal rede, onde ao realizar uma convolução com 400 filtros de tamanho 5x5x16 um aglomerado de dados de dimensões 5x5x16, resultando em um vetor de grandeza 1x1x400. Este procedimento é conhecido por processo de achatamento, ou do inglês, *flattening process*.

Entretanto, assim como descrito por Cunha:

> "No caso da segmentação de imagens, ao invés de analisar todas as características extraídas num vetor e retornar uma única divisão de probabilidades para a imagem, é necessário analisá-la pixel a pixel, e para isso é preciso substituir esses elementos que caracterizam a camada totalmente conectada. A nova camada de convolução utilizará filtros que ao invés de cobrir todo o bloco, cobrirão apenas 1 pixel ao longo do bloco". (2022, v.1, p.36)

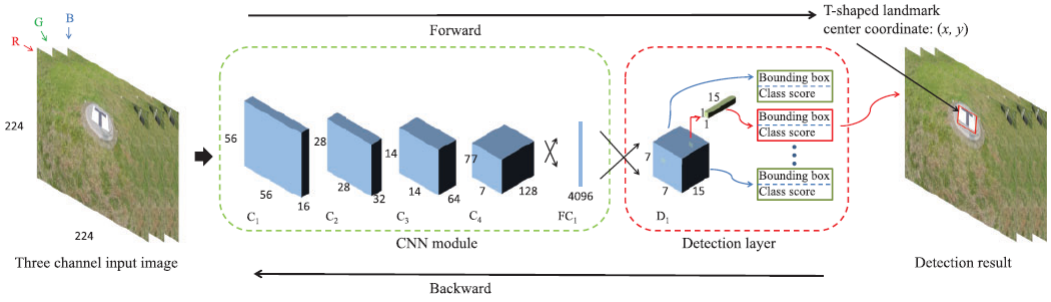
Figura 26 – Arquitetura de uma RTC de Segmentação



Fonte: Cunha (2020).

Já na Figura 27, é possível visualizar uma arquitetura referente a uma RNC de localização. Neste exemplo de detecção, cada entrada no modelo é referente a uma imagem de três canais (RGB), resultando em uma saída referente a localização prevista de um ponto de referência detectado na imagem.

Figura 27 – Arquitetura de uma RNC de Localização



Fonte: Yu et.al (2018).

Percebe-se, também na Figura 27, que tal arquitetura é dividida em dois módulos: módulo RNC e módulo de detecção. No primeiro, é onde a imagem é processada em quatro camadas de convolução, sendo elas, C1, C2, C3 e C4, seguidas por uma camada totalmente conectada. Já no segundo módulo, há uma camada de detecção responsável pelo processo de regressão.

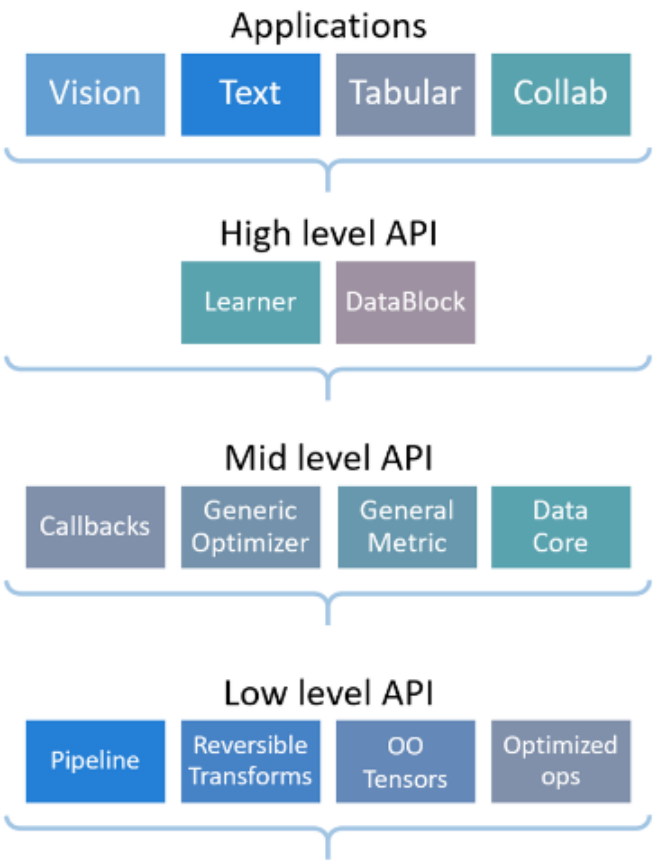
2.4 Fast.ai e o Processo de Treinamento de Dados

Criada em 2016 por Jeremy Howard e Rachel Thomas, a Fast.ai é uma biblioteca de aprendizado profundo capaz de fornecer, de forma simples e rápida, resultados de alta qualidade para soluções voltadas à aprendizado de máquina, possibilitando

o uso em projetos já existentes ou, até mesmo, em novas abordagens. A Fast.ai tem como principal foco proporcionar flexibilidade e performance à solução, sem qualquer tipo de comprometimento entre ambas. Salienta-se que a Fast.ai é uma biblioteca disponível apenas para a linguagem de programação *Python*, permitindo a flexibilidade da biblioteca *PyTorch*.

Em questão de arquitetura, o *design* da mesma foi criado visando dois objetivos: ser acessível e de fácil e rápida produção, sem perder sua fácil adaptabilidade. Assim como ilustra a Figura 28, a construção da arquitetura é feita sobre a hierarquia de APIs de baixo nível, que fornecem blocos de construção combináveis. Sendo assim, caso o usuário deseja reescrever ou personalizar parte da API de alto nível ou, até mesmo, adicionar uma característica em particular para atender os requisitos de um projeto, o mesmo não terá que aprender a utilizar as camadas inferiores da API (baixo nível).

Figura 28 – Arquitetura utilizada pela Fast.ai



Fonte: Equipe Fast.ai (2022).

Assim como ilustrado na Figura 28, na API de alto nível, há o *DataBlock*. O *DataBlock* é responsável pela resolução de problemas relacionados aos dados de entrada da rede, como na coleta desses dados, identificação do problema, separação

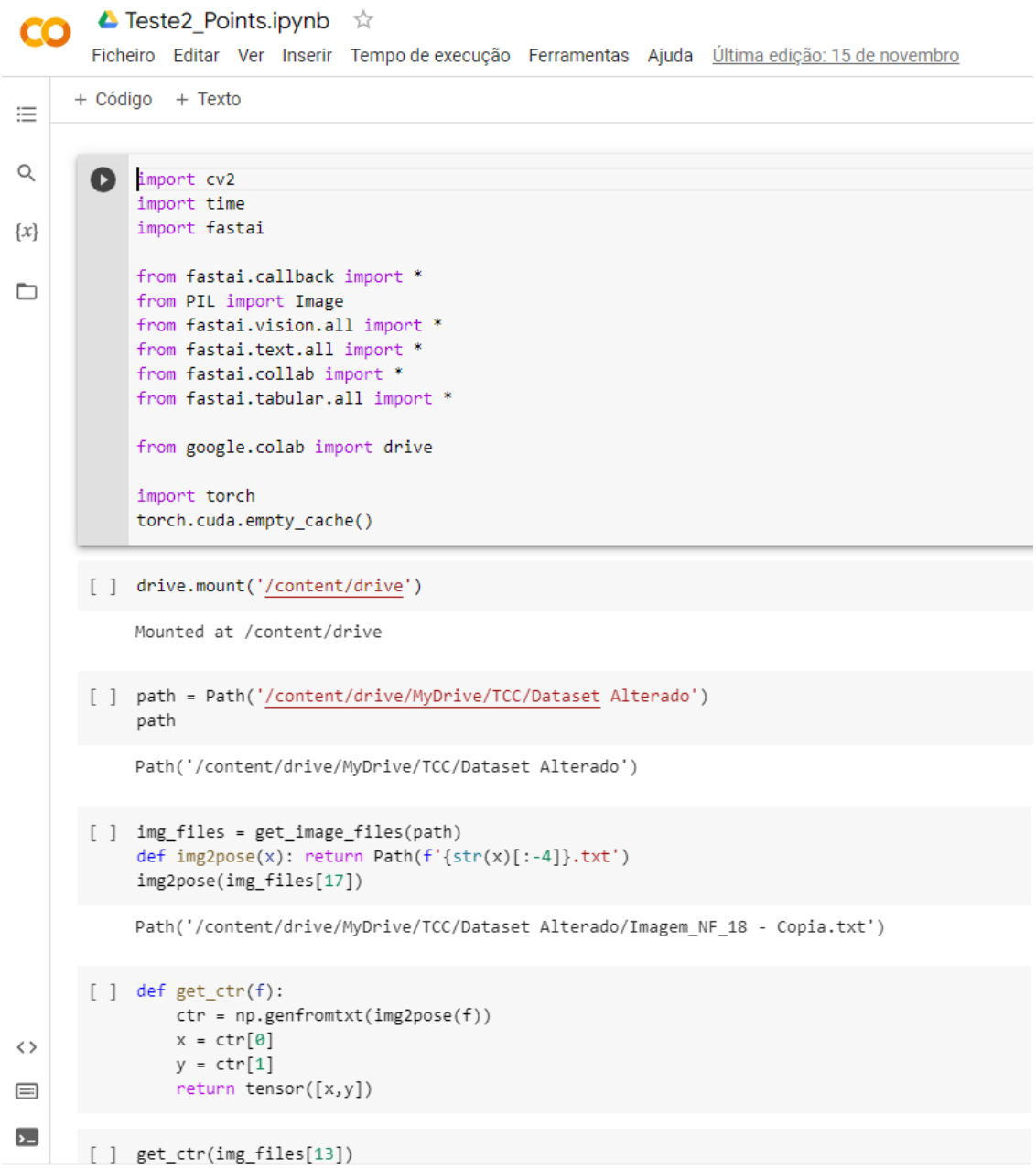
do banco de dados utilizado durante o treinamente, dentre outras funcionalidades e validações. É a partir disso, que é definido as etapas necessárias para preparação dos dados a serem processados por um modelo de aprendizado profundo.

Além disso, ainda no *DataBlock*, é necessário ser apontado de onde as imagens de entrada virão, e como as mesmas serão distribuídas e/ou classificadas. Ainda, também é necessário a definir como será feita a divisão das imagens do banco de dados dentre os processos de treinamento e validação. Sendo assim, preparando os dados para o treinamento do modelo.

Em relação ao processo de treinamento, a função responsável é a *Learner*. Essa função de treinamento recebe como parâmetro obrigatório os dados previamente preparados e pré-processados pelo *DataBlock*, assim como a arquitetura a ser utilizada. A partir disso, a função compila e realiza baterias de treinamento (conhecidas como épocas), obtendo valores prévios de desempenho. Tendo o modelo treinado, é possível utilizar a função *fine_tune*, responsável por aperfeiçoar os resultados obtidos, buscando obter valores maiores de eficiência.

Os processos descritos, que são realizadas pelas funções da Fast.ai, exigem poder de processamento elevado, fazendo grande uso da GPU. Sendo assim, ao trabalhar com RNCs, é comum ser utilizado ambientes de desenvolvimento em nuvem, como por exemplo, o *Google Colaboratory*. O *Google Colab* é um ambiente em nuvem da Google, disponível de forma (limitada) gratuita, assim como é possível visualizar na Figura 29, a plataforma possui uma interface simples e prática, de fácil integração com o *Google Drive*, que pode ser usufruido no armazenamento de bases de dados.

Figura 29 – Interface do *Google Colab*



Fonte: Elaborado pelo autor (2022).

2.5 Trabalhos Correlatos

É possível achar trabalhos que usufruem da Visão Computacional e Redes Neurais para a detecção e localização de objetos em diversas áreas de estudo. Um projeto que vale ser citado é o trabalho desenvolvido por Felipe Conter (2022), que, através da detecção de objetos, propôs um método responsável pela realização de uma estimativa do ponto central 3D de objetos para aplicação robótica.

Seu projeto teve sucesso ao provar que quanto maior a distância entre os

pontos de fixação, menor foi a distância euclidiana entre ponto central 3D estimado pela rede neural e o ponto central real. Tal feito foi possível graças a integração do YOLOv3 com um novo método de estimativa do ponto central 3D de objetos através do ROS.

Um segundo trabalho que vale ser mencionado é da equipe da Fast.ai (disponível como tutorial no site da Fast.ai) que, utilizando de biblioteca proprietária focada em aprendizado profundo, desenvolveu uma solução capaz de detectar as coordenadas em uma imagem, do rosto de um ser humano, localizando o centro do mesmo.

Tal projeto obteve êxito ao apresentar resultados com eficiência superior à 99%, exibindo suas previsões corretamente com certa pluralidade. Esses resultados foram possíveis graças a arquitetura robusta da solução e a utilização da base dados *Biwi*, que possui centenas de imagens e suas respectivas coordenadas obtidas com o auxílio do sensor de movimentos da Microsoft, o *Kinect*.

Em relação a trabalhos acadêmicos realizando estudos comparativos entre o uso de Redes Neurais e Visão Clássica, não foram encontrados artigos. Entretanto, um projeto com similaridades é o artigo desenvolvido por Maurício Stivanello e Kleber Marcellino (2019), que propuseram um sistema de detecção *online* de rebites metálicos utilizando Visão Computacional Clássica.

A solução obteve êxito ao detectar os rebites com precisão e acurácia superiores a 90%, mantendo tais resultados até mesmo para rebites com deformações estruturais. Com esses resultados, concluiu-se que o classificador de distância mínima, aliado aos descritores selecionados, podem atingir taxas de detecção equivalentes, ou até mesmo, superiores se comparadas com métodos mais custosos financeiramente.

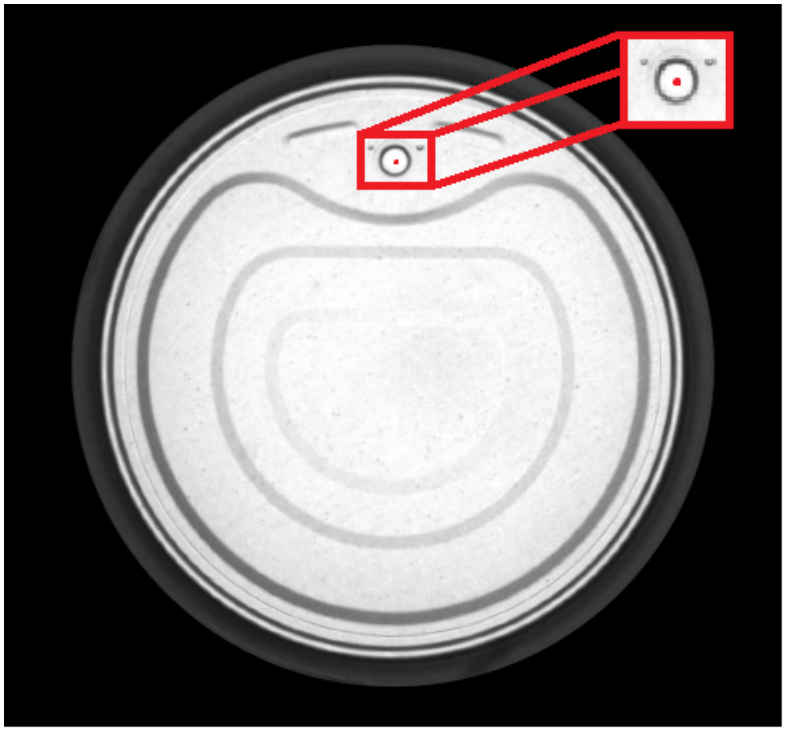
3 METODOLOGIA

Neste capítulo será detalhada a metodologia empregada para o desenvolvimento do projeto proposto, assim como, as ferramentas e estratégias utilizadas para realização do estudo comparativo entre as soluções utilizando Redes Neurais Convolucionais e Visão Computacional Clássica, assim como métricas e parâmetros estabelecidos.

3.1 Descrição geral dos requisitos da aplicação

A solução desenvolvida neste projeto, tem como objetivo principal a localização de rebites metálicos presentes em imagens capturadas por uma câmera do fundo da tampa de enlatados, de modo que se possa inspecionar a integridade dos mesmos. Assim como é mostrado na Figura 30, a ideia consiste na detecção do centro do rebite para que, a partir do mesmo, seja adquirida uma subjanela e suas coordenadas.

Figura 30 – Ilustração representando proposta final da solução



Fonte: Elaborado pelo autor (2022).

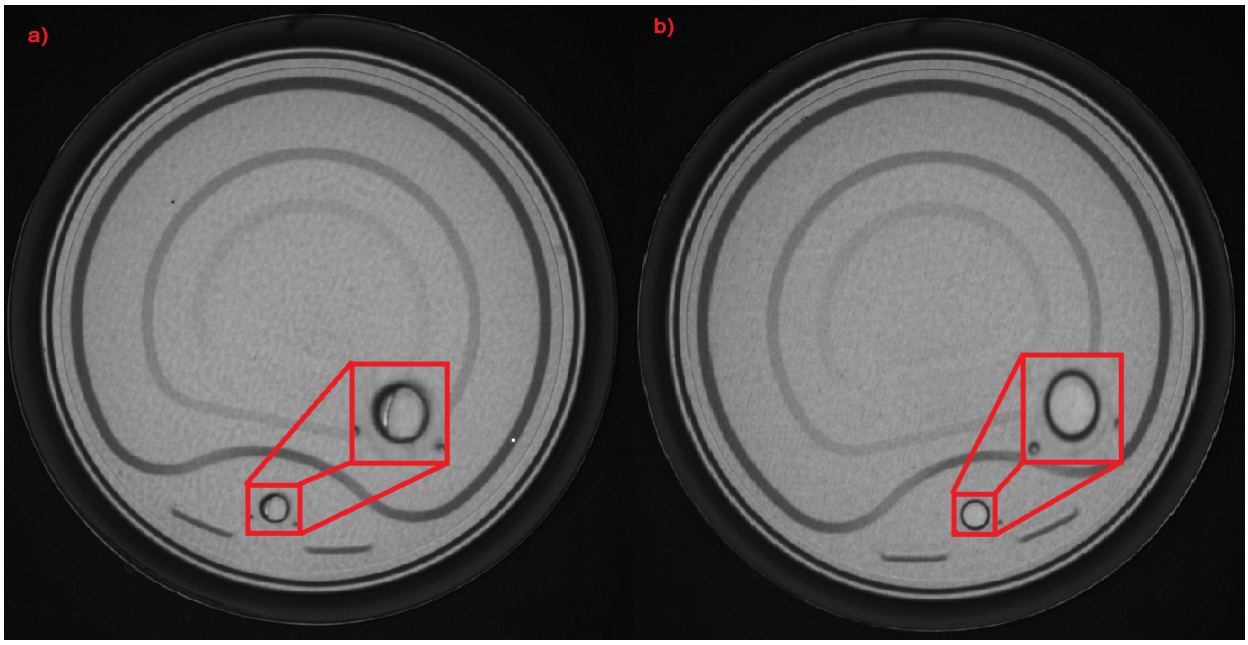
A ideia é que, a partir de tal recorte, seja possível, de maneira mais otimizada, realizar a classificação do rebite presente na imagem como sendo defeituoso ou não defeituoso. Vale ressaltar que, em sua resolução original, o rebite pertencente ao modelo de latinha utilizada, possui um diâmetro de, aproximadamente, 40 *pixels*. Sendo

assim, para a obtenção da subjanela contendo o objeto, deverá ser obtido um quadrado em que seus lados estejam a uma distâcia superior de 40 *pixels* do centróide alcançado.

3.2 Preparação da Base de Dados

Inicialmente, foi fornecido por um dos autores do trabalho "*A Machine Vision System for Online Metal Can- End Rivet Inspection*", professor Maurício Stivanello, uma base de dados contendo 152 imagens de latas metálicas. Dentre as imagens fornecidas, 41 eram referentes às tampas que possuiam seu rebite fraturado ou deformado de alguma maneira, já as 111 imagens restantes, são relativas às tampas que não possuiam qualquer defeito estrutural em seu rebite. Na Figura 31.a, é possível visualizar uma imagem referente a uma tampa com seu rebite danificado. Já na Figura 31.b, observa-se uma imagem refente a uma tampa com seu rebite não fraturado.

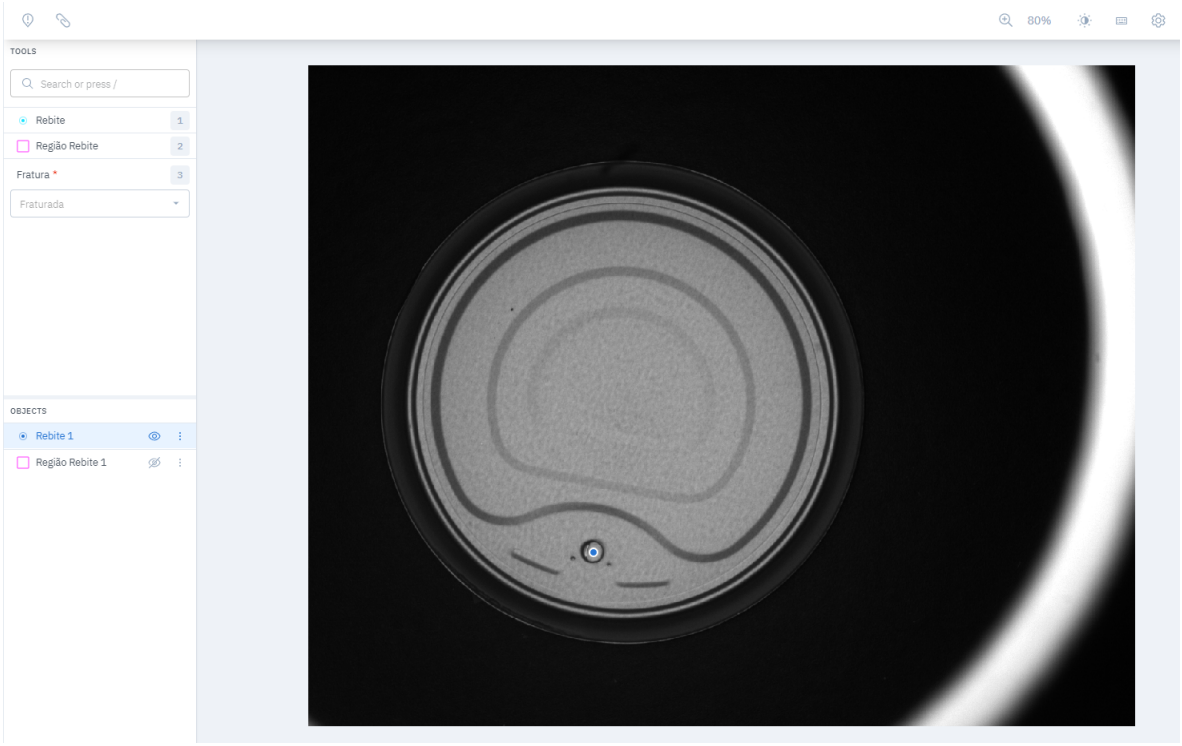
Figura 31 – Exemplos das classes Fraturadas e Não Fraturadas



Fonte: Elaborado pelo autor (2022).

Recebida a base de dados e devidamente organizada, iniciou-se o processo de marcação manual do centro do rebite, com o objetivo de obter e armazenar as coordenadas X e Y de referência. Tal processo foi feito a partir da plataforma gratuita LabelBox. Assim como é possível visualizar na Figura 33, esta plataforma permite a marcação de um ponto na imagem que, posteriormente, possibilita a extração da coordenada de posição.

Figura 32 – Interface de marcação da plataforma LabelBox



Fonte: Elaborado pelo autor (2022).

Além da posição do ponto marcado, a plataforma possibilita designar a imagem uma classe previamente definida. Sendo assim, as imagens marcadas foram separadas dentre duas classes: "Fraturadas", referentes às latas com qualquer defeito em seu rebite, e as "Não Fraturadas", referentes às latas sem qualquer dano em sua estrutura.

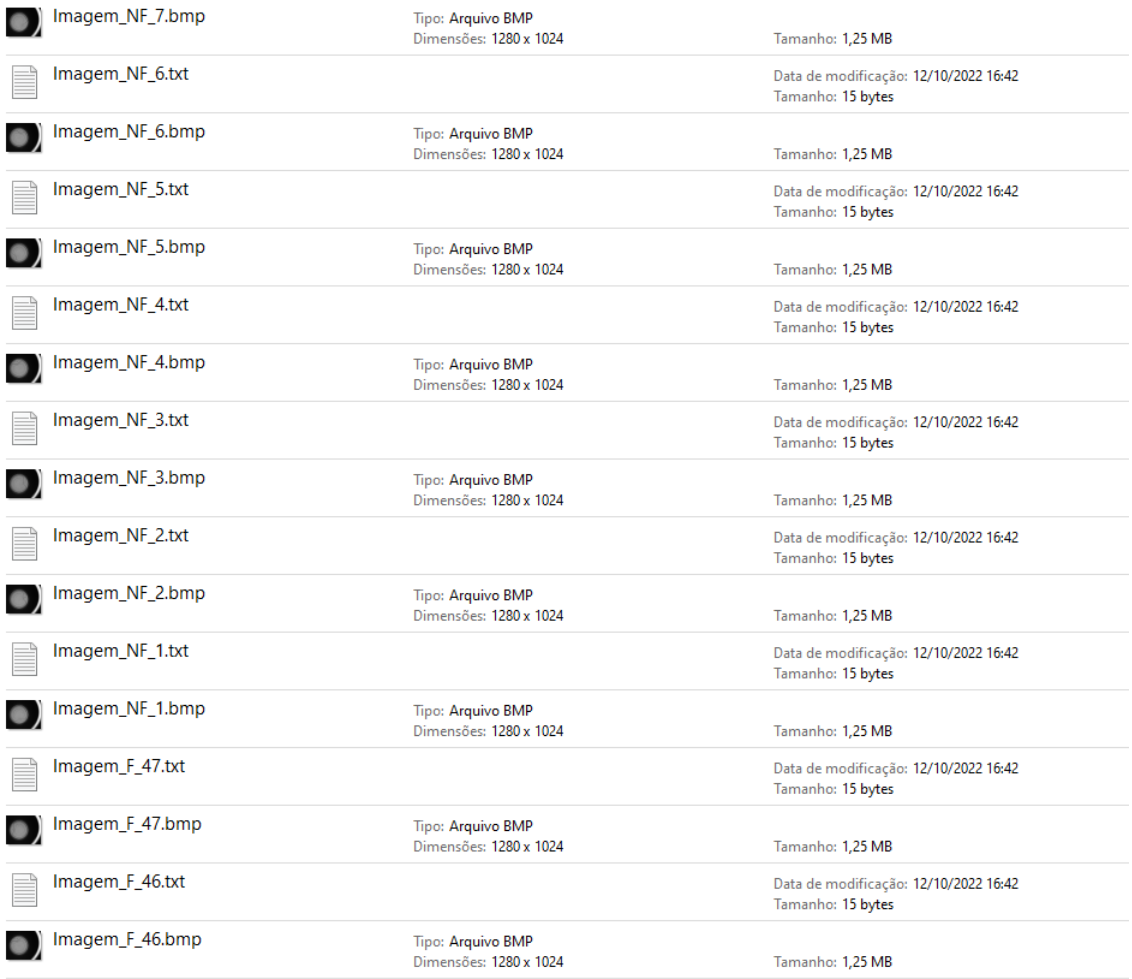
Feita a anotação, marcação e classificação de cada imagem a partir da plataforma, foi então utilizado a função de "*Data Export*", que gera um arquivo .json com todas as informações de cada imagem de maneira já organizada e atribuindo um ID a cada uma delas, dentre tais informações estão: ID da imagem, descritores aplicados, nome da imagem, categoria e os dados referentes a posição e tamanho de cada descritor. Após isso, utilizando um conversor *online*, foi realizada a conversão do arquivo ".json"para ".xlsx", possibilitando manipular e organizar os dados obtidos na plataforma de planilhas Microsoft Excel.

Tendo todos os dados já organizados e validados, foi desenvolvido um código simples utilizando a biblioteca Pandas. Tal código tinha como função ler as coordenadas X e Y de dada imagem, inserir tais coordenadas em um arquivo .txt e renomear o arquivo de acordo com seu ID e classe, seguindo o padrão: "Imagem_F_ID"para imagens da classe "Fraturada"ou "Imagem_NF_ID". Vale ressaltar que, ambos os arquivos de imagem e texto (contendo as coordenadas), possuem a mesma nomenclatura, buscando facilitar o processo de treinamento de modelos de aprendizado profundo,

posteriormente.

Realizados todos os procedimentos descritos, assim como ilustra a Figura 33, a base de dados foi preparada e validada para ser utilizada pelos algoritmos propostos.

Figura 33 – Print de uma parte da Base de Dados



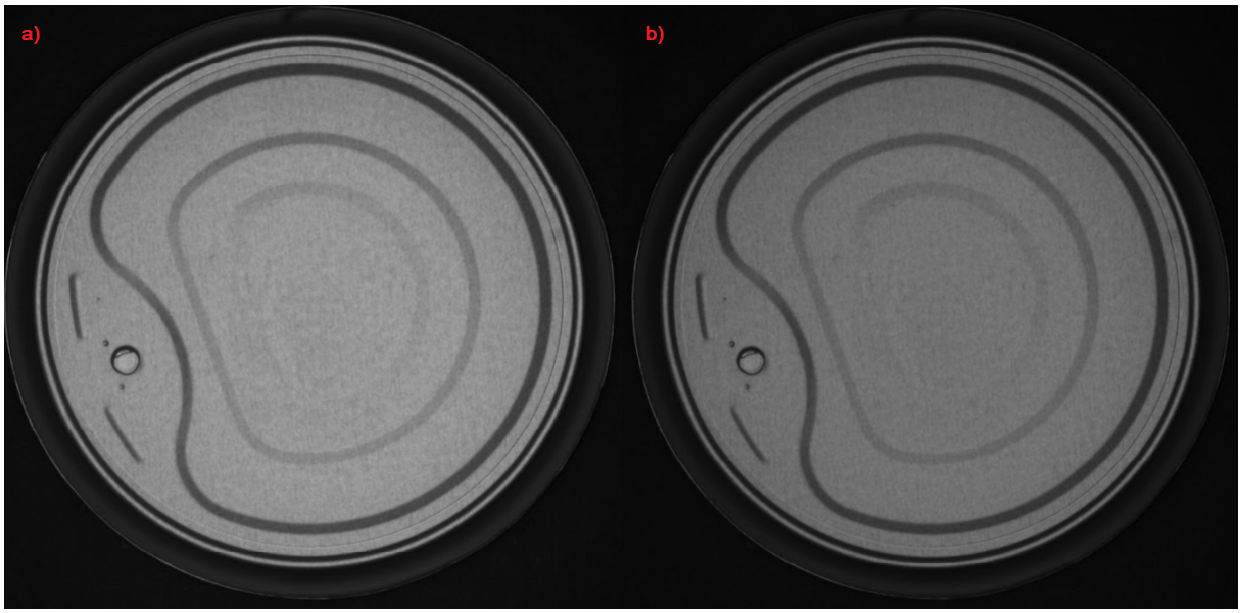
Fonte: Elaborado pelo autor (2022).

Além disso, foram desenvolvidas versões alternativas da base de dados descrita. Com o objetivo de simular situações mais próximas da realidade, foram inseridos de maneira artificial ruídos e variações de brilho às imagens, de modo que se pudesse avaliar a robustez dos algoritmos de localização propostos quando da presença de tais alterações ou artefatos. Vale destacar que, tais alterações foram feitas de maneira aleatória, com alterações no brilho variando de -30% á 30%. Finalmente, aplicou-se um Ruído Gaussiano de baixa intensidade, criando um leve ruído visual na imagem.

Todas as alterações aplicadas resultaram em uma versão alternativa final da base de dados, utilizada posteriormente para fins de estudo dos resultados obtidos

e, assim como se visualiza na Figura 34, as variações realizadas foram "suaves", buscando não afetar consideravelmente a imagem.

Figura 34 – Antes (a) e Depois (b) de uma Imagem após as alterações visuais

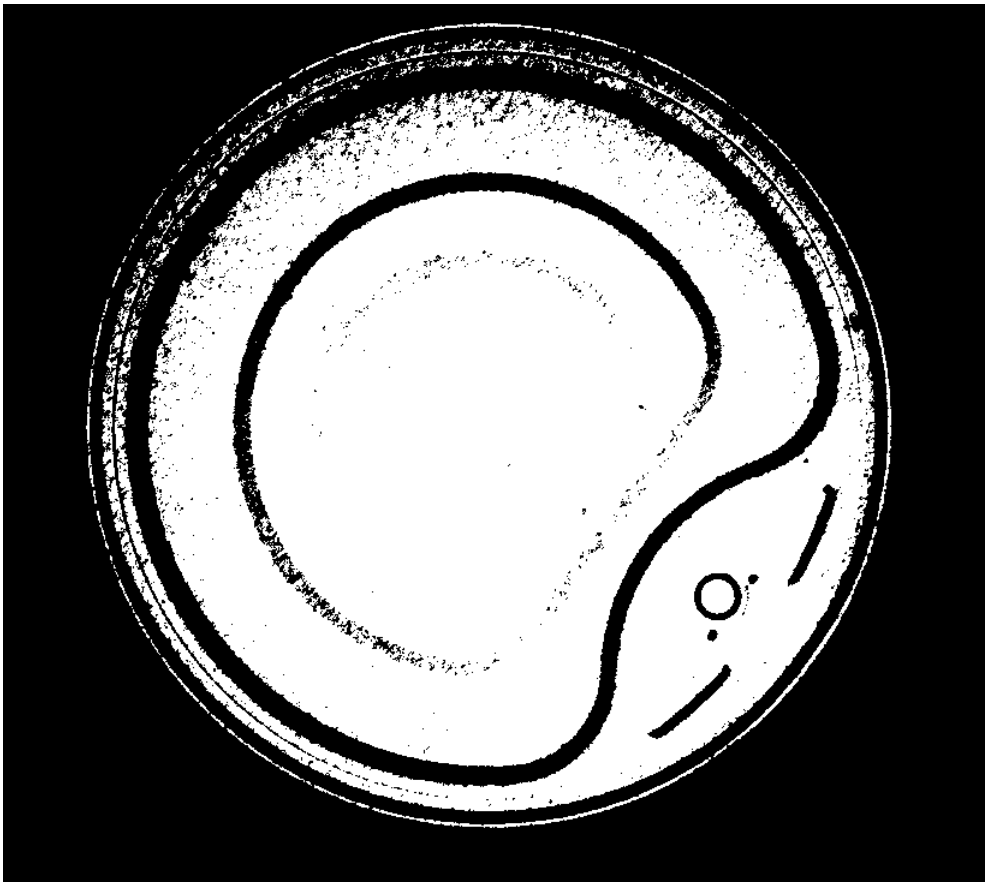


Fonte: Elaborado pelo autor (2022).

3.3 Abordagem de localização de rebite utilizando Visão Clássica

Inicialmente, é realizado o carregamento das bibliotecas a serem utilizadas e a conexão ao Google Drive (para acesso à base de dados). Tendo acesso a base de dados, é feita uma verificação sobre cada arquivo da base e, caso o arquivo em questão seja do formato ".bmp", ou seja, uma imagem, será então realizada o processamento da mesma. O primeiro método a ser aplicado à imagem, é o da limiarização, com o objetivo de separar o fundo da tampa das regiões de borda ou com alteração de profundidade evidenciadas por diferenças de intensidade. Após alguns testes feitos, a partir do visual das imagens obtidos após a aplicação do processo de binarização, percebeu-se que o fator de limiarização ideal era 127. Na Figura 35, é possível visualizar o resultado de uma das imagens testadas após o processo de binarização.

Figura 35 – Imagem após o processo de Limiarização

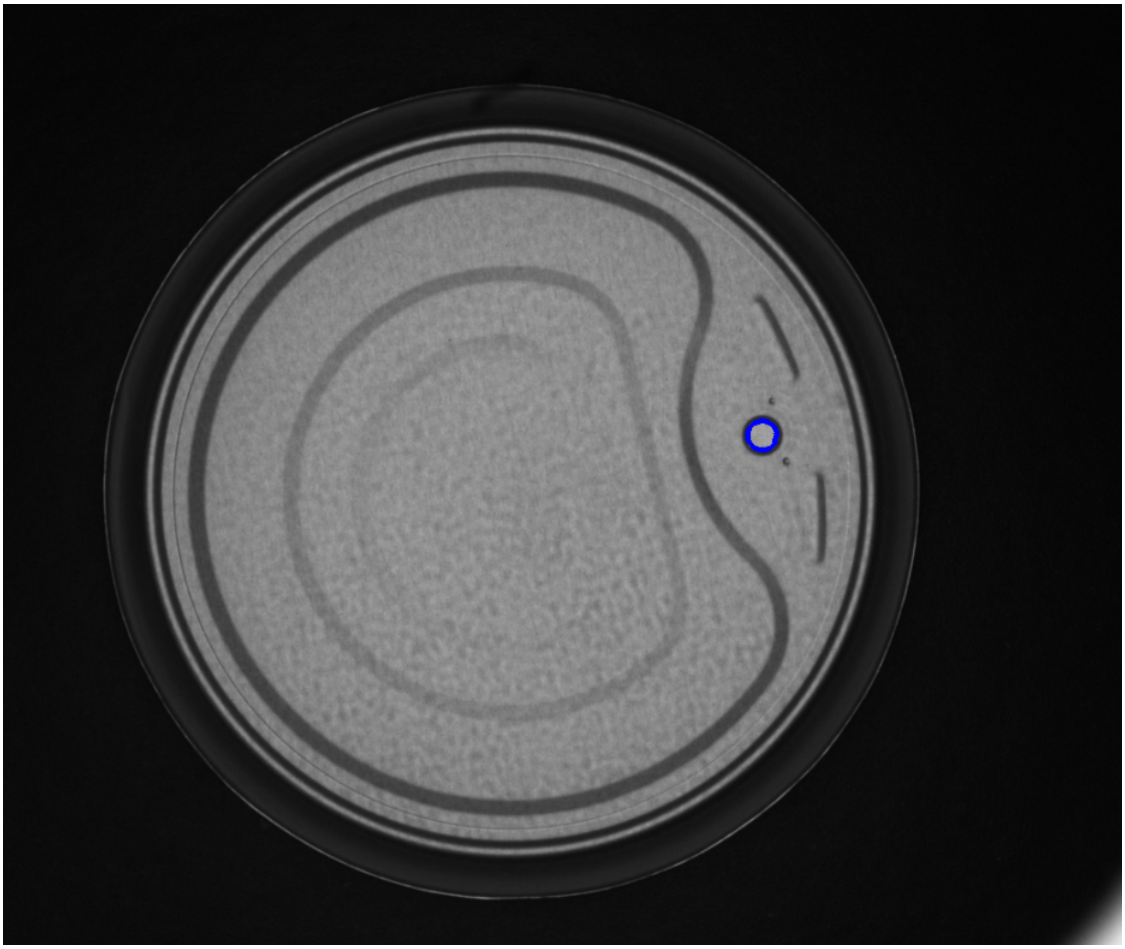


Fonte: Elaborado pelo autor (2022).

Aplicado o método de limiarização, é então executada a função "*findCountours* pertencente a *OpenCV*, que implementa um algoritmo de identificação e agrupamento de blocos de *pixels* conectados. Assim como é possível visualizar na Figura 36, a função será responsável por buscar possíveis contornos pertencentes à imagem, retornando uma lista de contornos encontrados. Sendo assim, é navegado por tal lista e, caso o contorno possua uma área superior a 220 e inferior a 1000 *pixels*, será obtido a circularidade da região obtida. Assim como explicado anteriormente na seção de descritores (seção 2.2.5), a circularidade é um valor que varia de 0 a 1, sendo 1 referente a uma região composta por um círculo perfeito, ou seja, 100% circular. Caso o valor obtido ao calcular a circularidade seja maior ou igual a 0,75, ou seja, seja pelo menos 75% circular, isso significa que o contorno obtido é referente ao rebite metálico da imagem.

Tendo sido o rebite localizado, é então calculado o centro do contorno encontrado. Na Figura 36, visualiza-se o contorno encontrado, aplicado sobre a imagem, para validar o resultado alcançado.

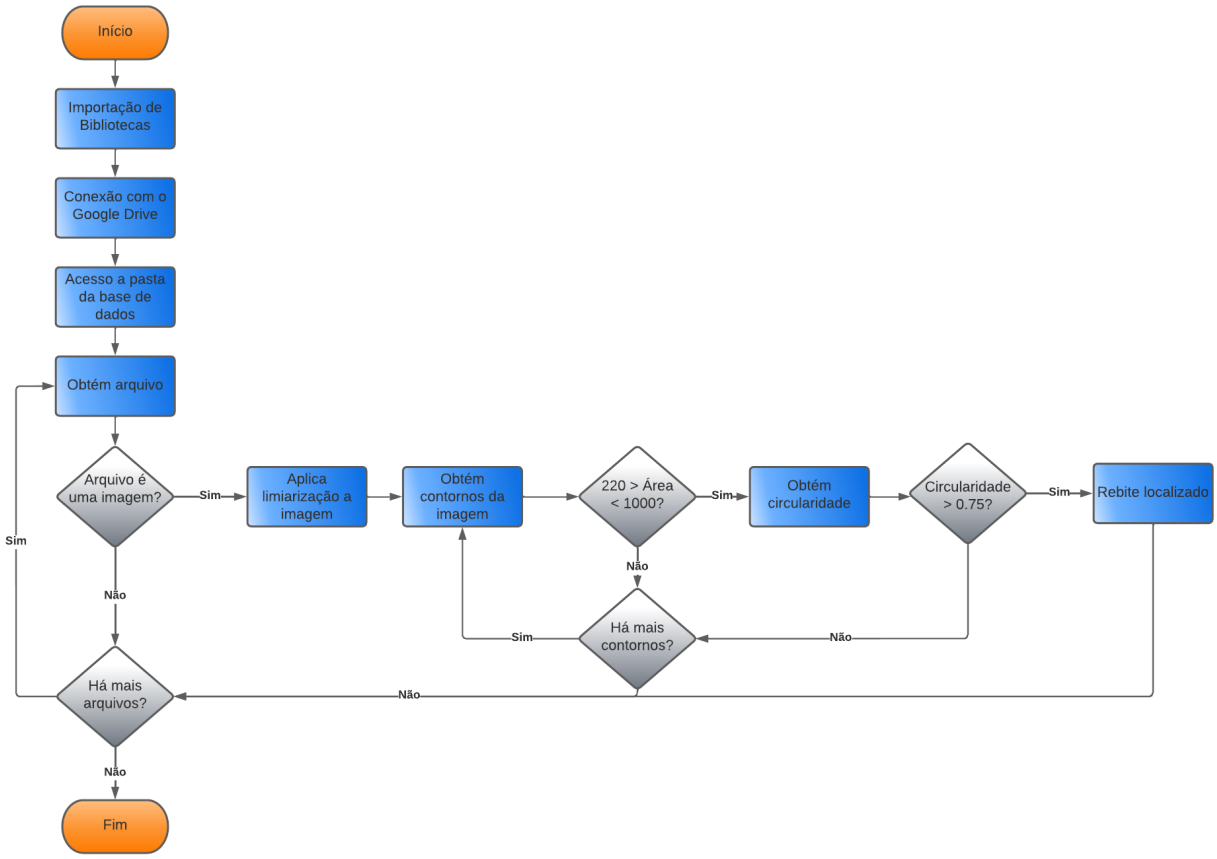
Figura 36 – Imagem após encontro do rebite



Fonte: Elaborado pelo autor (2022).

No fim do processo, foram então inseridos os resultados obtidos em uma planilha, sendo esses dados o nome do arquivo, a classe da imagem obtida a partir do nome do arquivo, ou seja, "F"para a classe "Fraturadas"e "NF"para a classe "Não Fraturadas". Além disso, também foram armazenadas as coordenadas XY do centro do rebite. Na Figura 37, é possível enxergar o processo por completo descrito em um fluxograma, visando melhor entendimento do mesmo.

Figura 37 – Descrição do processo realizado pelo algoritmo de Visão Clássica



Fonte: Elaborado pelo autor (2022).

Desenvolvida na plataforma em nuvem da Google, o Google Colab, o algoritmo referente à abordagem de Visão Clássica utiliza como base a biblioteca *OpenCV*. Apesar da mesma ser ter seu código fonte em linguagem C++, a *OpenCV* pode ser utilizada, também, em linguagem *Python*.

3.4 Abordagem de localização de rebites empregando técnica de Aprendizado Profundo

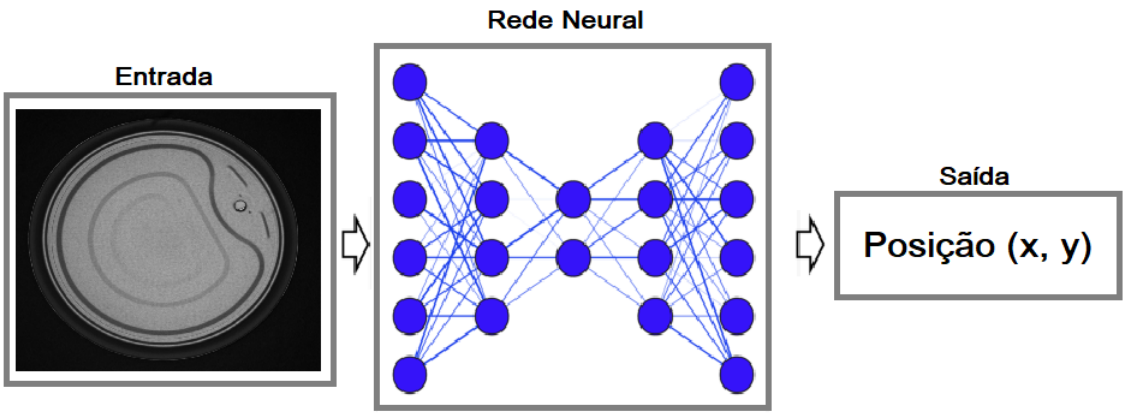
Diferente da abordagem anterior que, baseia-se na sequência de de processamanto de *pixels* por Visão Clássica, a segunda solução desenvolvida é responsável pela obtenção dos resultados obtido utilizando técnicas de Aprendizado Profundo, empregando Redes Neurais Convolucionais.

Os modelos utilizados tinham como arquitetura base a Resnet18, uma Rede Neural Residual de 18 camadas, considerada a menor dentre sua família. Apesar de possuir um número de camadas inferior à Resnet34 e Resnet50, que possuem 34 e 50 camadas, respectivamente, a Resnet18 se mostrou suficiente para o problema

prosposto.

Assim como exposto na Figura 38, a ideia do processo consiste na imagem contendo o objeto a ser detectado, ou seja, o rebite metálico, como entrada da rede. O dado de entrada é então processado pela rede neural, resultando nas coordenadas X e Y de referência, que neste caso, são os dados de saída da rede.

Figura 38 – Visualização do processo de Aprendizado Profundo



Fonte: Elaborado pelo autor (2022).

Para melhor entendimento do processo realizado pela rede neural e, também, detalhamento dos parâmetros utilizados, será descrito o processo de desenvolvimento do algoritmo referente à abordagem de Aprendizado Profundo.

Primeiramente, assim como na solução anteriror, é feito o carregamento das bibliotecas a serem utilizadas e a conexão ao Google Drive (para acesso à base de dados). Com o acesso a base de dados, é obtido todas as imagens da mesma a partir do uso da função *get_image_files*, responsável por retornar uma lista de caminhos de qualquer imagem presente em uma dada pasta, independente do formato da mesma. Tendo o caminho de cada imagem, é então feita a definição de uma nova função, nomeada obtemTXT.

Assim como é possível visualizar no Código 1, a função "obtemTXT"tem como objetivo receber um valor *x* referente ao nome do arquivo de imagem, para que assim, seja obtido o caminho do arquivo ".txt"referente a tal imagem. Sendo assim, são adquiradas as coordenadas XY relacionadas ao gabarito do centro do rebite metálico, marcado de maneira manual durante o processo de preparação da base de dados.

Graças a função obtemTXT é possível acessar as coordenadas do centro do rebite de uma imagem. Sendo assim, é realizada a definição de outra função, nomeada *get_ctr*. Bem como é exibido no Código 2, essa função é responsável por obter um valor *y* referente ao caminho de um arquivo de texto. Após a obtenção do mesmo, é

Codigo 1 – Definição da função "obtemTXT"

```
# Definição da função que obtém e relaciona cada imagem e seu respectivo bloco de notas:
img_files = get_image_files(path)
def obtemTXT(x): return Path(f'{str(x)[:-4]}.txt')
```

utilizado a função *genfromtxt*, responsável por retornar as informações de um bloco de notas em uma lista. Por fim, é obtida a partir de tal lista as coordenadas X e Y, nas posições 0 e 1, respectivamente.

Codigo 2 – Definição da função "*get_ctr*"

```
# Definição da função que obtém as coordenadas XY do bloco de notas de cada imagem:
def get_ctr(f):
    ctr = np.genfromtxt(obtemTXT(f))
    x = ctr[0]
    y = ctr[1]
    return tensor([x,y])
```

Com as funções "obtemTXT"e "*get_ctr*"definidas, é então possível estabelecer os parâmetros necessários da função *dataBlock*, já detalhada anteriormente. Assim como é possível perceber no Código 3, os parâmetros necessários para tal função são: *blocks*, *get_items*, *get_ctr*, *splitter*, *batch_tfms*.

Codigo 3 – Definição do *DataBlock*

```
# Criação do datablock a partir dos dados da base:
dataset = DataBlock(
    blocks=(ImageBlock, PointBlock),
    get_items=get_image_files,
    get_y=get_ctr,
    splitter=FuncSplitter(lambda o: o.parent.name=='13'),
    batch_tfms=[Rotate(), Flip(), Dihedral(), Normalize.from_stats(*imagenet_stats)
    ]
)
```

O parâmetro *blocks* é responsável por definir as transformações a serem realizadas durante o processo. Neste caso, serão utilizadas as transformações *ImageBlock* e *PointBlock*, responsáveis por trabalhar com imagens e pontos, respectivamente. Já em relação ao parâmetro *get_items*, o mesmo é encarregado de receber os itens a

serem processados, que neste caso são imagens. Devido a isto, o valor deste parâmetro é o resultado obtido pela função *get_image_files*. Responsável por receber as coordenadas XY do objeto a ser detectado, o parâmetro *get_y* possui como valor, os valores alcançados pela função *get_ctr*. Por fim, o parâmetro *batch_tfms* é encarregado por receber os métodos de aumento de dados a serem realizados, como rotacionar e espelhar a imagem, por exemplo. Tal estratégia busca suprir a falta de uma base de dados robusta para realização de projetos, buscando atingir maiores valores de eficiência.

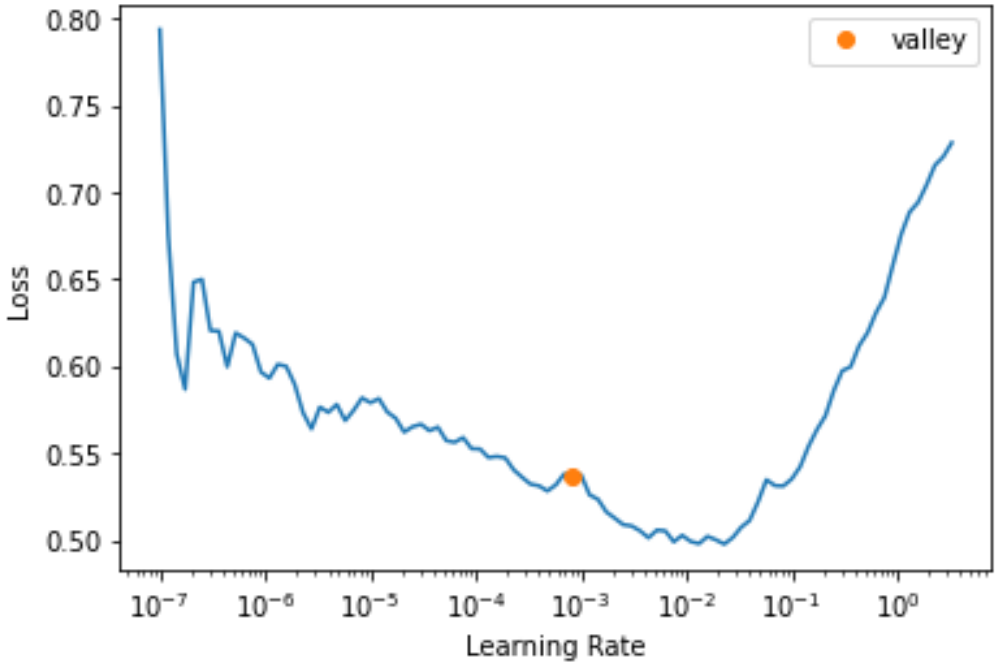
No código 4, são apresentadas a funções responsáveis pelos processos de treinamento da rede neural. Inicialmente, é especificado o tamanho do lote (ou do inglês, *batch size*) de imagens a serem a processados por vez. Tal valor é especificado através do parâmetro *bs*=8, da função *dataloaders*. Após isso, através do uso da função *learner*, detalhada no capítulo anterior, é definida a arquitetura a ser utilizada, a Resnet18.

Codigo 4 – Definição das funções de treinamento do modelo

```
19 # Processo de aprendizagem à máquina e obtenção dos resultados:
20 dls = dataset.dataloaders(path,bs=8) # Tamanho do "Batch" = 8
21 dls.show_batch()
22 learn = vision_learner(dls, resnet18, y_range=(-1,1)) # Utilização da rede neural
       Resnet18
23 learn.lr_find() # Obtenção da curva de aprendizado da máquina
24 learn.fine_tune(60, 5e-3) # Utilização de 60 épocas para o treinamento
25 learn.show_results() # Exibição dos resultados obtidos
```

Tendo o tamanho do lote e arquitetura parametrizados, é então utilizada a função *lr_find* para a aquisição do gráfico referente à curva de aprendizado, ilustrada na Figura 39. Este gráfico é obtido a partir das informações de "Perda"(*Loss*) e "Taxa de Aprendizado"(*Learning Rate*), referentes aos eixos Y e X, respectivamente. Com o modelo prévio já definido, é então usada a função *fine_tune*.

Figura 39 – Gráfico da Curva de Aprendizado do Modelo



Fonte: Elaborado pelo autor (2022).

Tendo como principal parâmetro a quantidade de "épocas"a serem realizadas, a função *fine_tune* é responsável pelo aperfeiçoamento do modelo e, consequentemente, aumento da eficiência da mesma. O termo "épocas", refere-se à quantidade de vezes que a rede neural sera exposta aos dados de treino, sendo esses a imagem e seu gabarito referente. Vale ressaltar que, apesar de um número elevado de épocas, geralmente, significar valores mais elevados de eficiência, após certo valor, o modelo perde um ganho significativo de rendimento com o aumento de épocas. Sendo assim, se torna necessária a realização de testes com diferentes quantidades para tal parâmetro, buscando sempre otimizar o processo de treinamento, equilibrando eficiência e tempo de processamento.

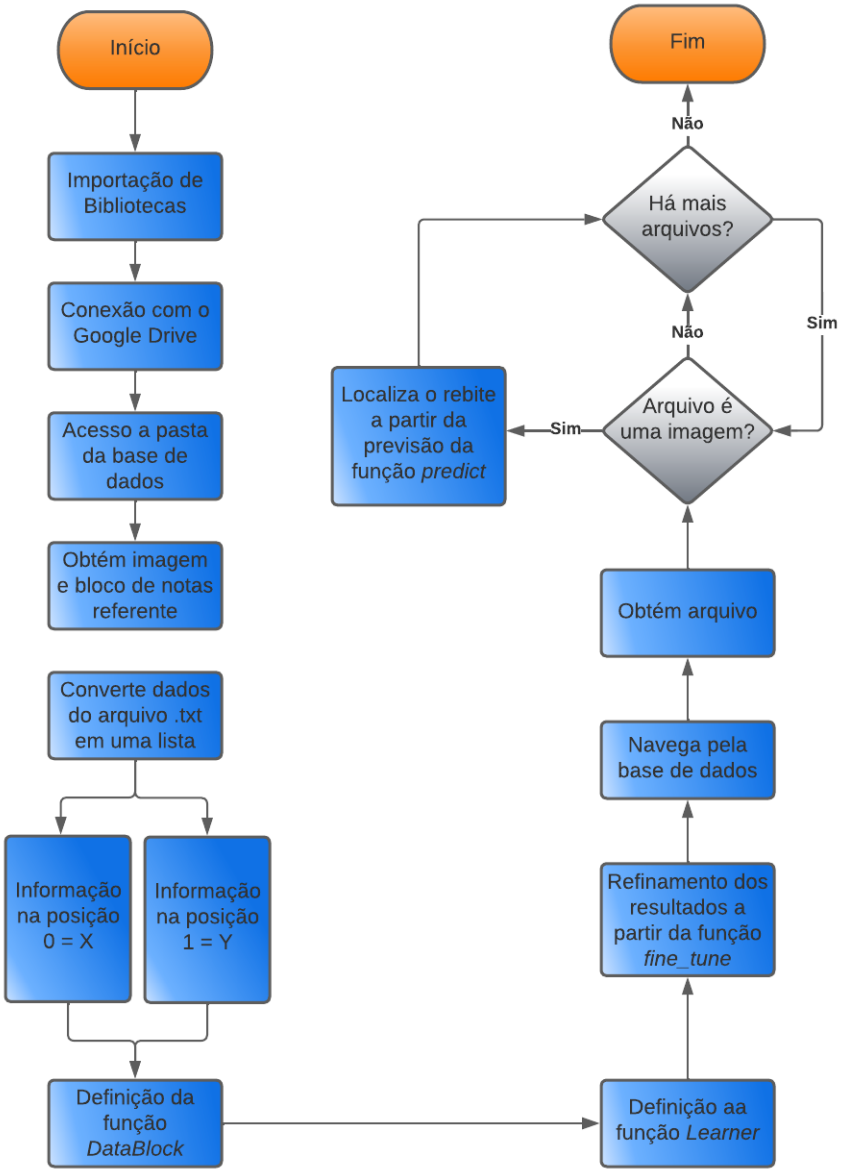
Assim como é exibido na linha 24 do código 39, durante os testes realizados neste projeto, percebeu-se que o valor ideal de épocas a serem utilizados é igual a 60. Os valores obtidos de eficiência e tempo de execução serão posteriormente discutidos no capítulo de "Apresentação de Resultados".

Com o modelo treinado e validado, foi então executada a função *predict* sobre cada imagem presente na base de dados, retornando os valores previstos pela rede neural para X e Y, indicando a localização do rebite metálico.

No fim do processo, assim como na abordagem anterior, os resultados obtidos foram inseridos em uma planilha, sendo esses dados o nome do arquivo, a classe da imagem e coordenadas XY do centro do rebite. A aquisição dessas informações, foi obtida da mesma maneira descrita anteriormente na seção referente à

solução de Visão Clássica, na Figura 40, é possível enxergar o processo por completo descrito em um fluxograma, visando melhor entendimento do mesmo.

Figura 40 – Descrição do processo realizado pelo algoritomo de Aprendizado Profundo



Fonte: Elaborado pelo autor (2022).

Assim como na abordagem anterior, o algoritmo referente à solução de Aprendizado Profundo foi desenvolvido por completo na plataforma em nuvem Google Colab. Entretanto, neste caso, a biblioteca utilizada foi a Fast.ai, já detalhada anteriormente no capítulo 2. Assim como na *OpenCV*, a Fast.ai permite o desenvolvimento do código utilizando a linguagem de programação *Python*.

4 APRESENTAÇÃO DOS RESULTADOS

Neste capítulo, serão apresentados os resultados obtidos referentes às soluções desenvolvidas. Para assim, poder ser realizado o estudo comparativo entre ambas, analisando eficiência e desvio dos valores obtido.

4.1 Métricas e Parâmetros de Avaliação utilizados

A métrica utilizada na avaliação de localização dos rebites alcançada pelos algoritmos propostos é baseada na distância euclidiana. Matematicamente, a Distância Euclidiana é definida pela distância entre dois pontos. Sendo assim, considerando que, x1 e x1 são relacionados às coordenadas de gabarito e, x2 e y2 são relacionados às coordenadas obtidas pela rede neural, como é mostrado na equação 5, a Distância Eucliada é dada por:

$$\sqrt{(x1 - x2)^2 + (y1 - y2)^2} \tag{5}$$

Obtendo o valor referente à Distância Euclidiana, assim como é possível visualizar na tabela 1, considerou-se as seguintes condições como critério avaliativo: caso o valor obtido seja superior a 40 *pixels*, ou seja, maior que o diâmetro médio do rebite, considera-se que o *status* do posicionamento do rebite metálico como "Não localizado". Entretanto, caso a Distância Euclidiana seja inferior ou igual a 40 *pixels*, o *status* considerado é de "Localizado", indicando que o ponto retornado está contido no rebite metálico.

Tabela 1 – Condições de Avaliação dos Resultados Obtidos

Condição	*Status* Obtido
Distância Euclidiana > 40	Não localizado
Distância Euclidiana <= 40	Localizado

Fonte: Elaborado pelo autor (2022).

Tais critérios avaliativos, foram estabelecidos visando obter a eficiência sobre a taxa de sucesso na localização das soluções desenvolvidas. Sendo assim, caso a solução apresente todos os resultados com o *status* "Localizado", por exemplo, conclui-se que a mesma obteve uma eficiência de 100%.

Vale lembrar que, assim como ilustrado nas equações 6 e 7, também foi calculado o desvio médio em módulo dos valores de X e Y para o gabarito. Tal cálculo foi realizado para fins de estudo.

$$distX = |x1 - x2| \tag{6}$$

$$distY = |y1 - y2| \tag{7}$$

4.2 Avaliação dos resultados obtidos com a solução baseada em Visão Clássica

Para obtenção de dados para a realização de um estudo comparativo, foram realizados dois testes: sendo o Teste 1 referente aos resultados obtidos utilizando as imagens originais, e o Teste 2, referente aos resultados obtidos utilizando as imagens com adição artificial de ruído.

Inicialmente, como é possível visualizar na tabela 2, a abordagem em questão obteve baixos valores de desvio médio, resultado em uma baixa distância euclidiana média e, consequentemente, elevadas taxas de sucesso na localização.

Tabela 2 – Resultados Obtidos pela Visão Clássica no Teste 1

Parâmetro Avaliado	Resultado Obtido
Desvio médio em X (distX)	0,97 *pixels*
Desvio médio em Y (distY)	1,40 *pixels*
Distância euclidiana média (Fraturadas)	2,17 *pixels*
Distância euclidiana média (Não Fraturadas)	1,72 *pixels*
Taxa de Sucesso (Fraturadas)	100%
Taxa de Sucesso (Não Fraturadas)	100%
Taxa de Sucesso Geral (Ambas as Classes)	100%

Fonte: Elaborado pelo autor (2022).

Também, percebe-se a diferença nos valores obtidos nas diferentes categorias de imagens: "Fraturadas"e "Não Fraturadas". No caso das imagens onde a estrutura do rebite estava conservada, ou seja, não fraturado, os resultados obtidos foram mais satisfatórios em relação a outra classe. Isso ocorre devido a geometria com circularidade mais elevada dos rebites não fraturados, que se convertem em resultados mais precisos.

Entretanto, assim como mencionado anteriormente, buscando simular resultados mais próximos da realidade, um segundo teste foi realizado utilizando imagens ruidosas. Os resultados de tal teste podem ser visualizados na tabela 3.

Observa-se que, com a adição de ruídos na imagem, os resultados obtidos pela Visão Clássico são significamente impactados, com uma queda de 56,1% e 37,84% nas taxas de sucesso na localização das classes "Fraturadas"e "Não Fraturadas", respectivamente. Ou seja, o impacto causado pelos ruídos, juntamente com as deformações na geometria do rebite, resultaram em resultados piores nas imagens da classe "Fraturadas". Apesar disso, em ambos os testes e ambas as classes, a distância euclidiana média se manteve inferior a 40 *pixels* - valor limite estipulado a partir do diâmetro do rebite.

Tabela 3 – Resultados Obtidos pela Visão Clássica no Teste 2

Parâmetro Avaliado	Resultado Obtido
Desvio médio em X (distX)	18,27 *pixels*
Desvio médio em Y (distY)	19,43 *pixels*
Distância euclidiana média (Fraturadas)	30,1 *pixels*
Distância euclidiana média (Não Fraturadas)	29,2 *pixels*
Taxa de Sucesso (Fraturadas)	43,90%
Taxa de Sucesso (Não Fraturadas)	62,16%
Taxa de Sucesso Geral (Ambas as Classes)	57,2%

Fonte: Elaborado pelo autor (2022).

Vale ressaltar que, mesmo com a adição de ruídos, nota-se resultados mais apropriados na classe de "Não Fraturadas", obtendo uma distância euclidiana e taxa de sucesso na localização do rebite inferiores em relação às "Fraturadas". Além disso, em 65 imagens - que representam, aproximadamente, 43% da base - o rebite não foi localizado, gerando impacto negativo considerável nos resultados obtidos.

Outro ponto a ser destacado, é o tempo de execução que, em ambos os testes, se manteve próximo de 23s para obter as coordenadas do rebite de todas as imagens. Sendo assim, percebeu-se um tempo médio de 0,151s por imagem processada, considerando a base utilizada de 152 imagens.

4.3 Avaliação dos resultados obtidos com a solução baseada em Aprendizado Profundo

Assim como na abordagem anterior, múltiplos testes foram realizados para obtenção de dados para estudo da solução. No entanto, no caso da abordagem de Aprendizado Profundo foram realizados três testes: Testes 1 e 2, similares aos do método anterior e o Teste 3, referente a um modelo treinado utilizando tanto imagens ruidosas, quanto as imagens originais (feitas em ambiente controlado). Ressalta-se que, nos Testes 1 e 2, a base de dados utilizada para treinamento do modelo em questão, possui apenas as 152 imagens originais.

Inicialmente, assim como é demonstrado na tabela 4, os resultados obtidos pela solução baseada em Aprendizado Profundo, não obteve resultados tão satisfatórios se comparada com a solução anterior, com valores de desvio médio - em X e Y - bem superiores. Porém, ainda sim, o rebite foi localizado corretamente na grande maioria dos casos, resultando em uma taxa de sucesso na localização do rebite superior a 75% em ambas as classes.

Constata-se, assim como a abordagem baseada em Visão Clássica, resultados mais satisfatórios por parte das imagens pertecentes à classe "Não Fraturadas", obtendo uma taxa de sucesso na localização do rebite de 87,39%. Apesar disso, em

Tabela 4 – Resultados Obtidos pelo Aprendizado da Máquina no Teste 1

Parâmetro Avaliado	Resultado Obtido
Desvio médio em X (distX)	18,26 *pixels*
Desvio médio em Y (distY)	16,44 *pixels*
Distância euclidiana média (Fraturadas)	30,94 *pixels*
Distância euclidiana média (Não Fraturadas)	28,54 *pixels*
Taxa de Sucesso (Fraturadas)	75,61%
Taxa de Sucesso (Não Fraturadas)	87,39%
Taxa de Sucesso Geral (Ambas as Classes)	84,21%

Fonte: Elaborado pelo autor (2022).

ambas as classes, a distância euclidiana média se manteve abaixo do limite estipulado de 40 *pixels*.

Após a realização do Teste 2, foram obtidos os resultados detalhados na tabela 5, que comprovam o impacto de ruídos na eficiência da solução, alcançando uma distância euclidiana superior ao limite de 40 *pixels* no caso da classe "Fraturadas". Apesar disso, percebe-se uma inversão no quesito de resultados mais precisos em relação às soluções, tendo a abordagem de Aprendizado Profundo obtido valores mais próximos do esperado se comparados aos valores da Visão Clássica.

Tabela 5 – Resultados Obtidos pelo Aprendizado da Máquina no Teste 2

Parâmetro Avaliado	Resultado Obtido
Desvio médio em X (distX)	27,11 *pixels*
Desvio médio em Y (distY)	19,48 *pixels*
Distância euclidiana média (Fraturadas)	46,72 *pixels*
Distância euclidiana média (Não Fraturadas)	37,99 *pixels*
Taxa de Sucesso (Fraturadas)	43,24%
Taxa de Sucesso (Não Fraturadas)	61,90%
Taxa de Sucesso Geral (Ambas as Classes)	56,58%

Fonte: Elaborado pelo autor (2022).

Todavia, com a vantagem de adaptação da solução de Aprendizado Profundo, realizou-se um terceiro teste. No Teste 3, utilizou-se uma base de dados mais robusta de 304 imagens, contendo as 152 imagens originais e as 152 com adição artificial de ruídos. Tal implementação foi realizada buscando "preparar"o modelo para o processamento de imagens ruidosas.

Refeito o treinamento do modelo com a nova base de dados, adquiriu-se os resultados exibidos na tabela 6. Ao visualizar tais resultados, percebe-se uma melhoria nos valores obtidos, com um aumento de, aproximadamente, 5% e 8% nas taxas de sucesso na localização dos rebites das classes "Não Fraturadas"e "Fraturadas", respectivamente. Essa melhoria pôde ser vista, também, nos demais parâmetros, que

obtiveram resultados mais próximos do esperado.

Tabela 6 – Resultados Obtidos pelo Aprendizado da Máquina no Teste 3

Parâmetro Avaliado	Resultado Obtido
Desvio médio em X (distX)	23,11 *pixels*
Desvio médio em Y (distY)	15,72 *pixels*
Distância euclidiana média (Fraturadas)	40,31 *pixels*
Distância euclidiana média (Não Fraturadas)	32,48 *pixels*
Taxa de Sucesso (Fraturadas)	51,2%
Taxa de Sucesso (Não Fraturadas)	67,70%
Taxa de Sucesso Geral (Ambas as Classes)	62,5%

Fonte: Elaborado pelo autor (2022).

Vale ressaltar que, tal estratégia de melhoria pode ser implementada apenas na solução de Aprendizado Profundo, visto que a Visão Clássica segue regras predefinidas. Por essa razão, o Teste 3 foi realizado apenas na abordagem envolvendo Aprendizado Profundo.

Em relação tempo de execução, houve uma duração média de, aproximadamente, 17s para o processamento de uma base de 152 imagens nos Testes 1 e 2, resultando em um tempo médio de 0,111s por imagem. Já em relação ao Teste 3, a duração média para processamento de uma base de 304 imagens, foi de 41s, resultando em um tempo médio de 0,135s por imagem. Sendo assim, conclui-se que houve um aumento de, aproximadamente, 22% no tempo de execução médio por imagem.

5 CONSIDERAÇÕES FINAIS

O presente trabalho descreve o desenvolvimento de possíveis abordagens para a concepção de um sistema de detecção, utilizando técnicas de Aprendizado Profundo baseados em regressão e metodologias mais tradicionais de sistemas de visão, referentes à Visão Clássica.

Os resultados obtidos neste projeto, demonstram que ambas as abordagens são viáveis em cenários ideais, atigindo taxas de sucesso de 100% e 84,21%, nos resultados referentes às soluções de Visão Clássica e Aprendizado Profundo, respectivamente.

Entretanto, com uma redução final de, aproximadamente, 21,7% na taxa de sucesso geral, o uso de RNCs se mostrou mais eficiente na presença de ruídos na imagem, adaptando-se com a inserção das mesmas na base de dados utilizada para o treinamento, resultando em um impacto significativamente menor se comparado com a Visão Clássica, que obteve uma redução de, aproximadamente, 42,8% na taxa de sucesso geral.

Além disso, outra conclusão obtida, foi o impacto da integridade física do rebite métalico nos resultados obtidos. Nos casos em que o rebite estava danificado e, consequentemente, sua geometria afetada, os resultados obtidos foram inferiores se comparados com os resultados referentes ao rebites não fraturados. No geral, em ambas as soluções, a diferença na taxa de sucesso entre as classes variou entre 10% e 20%, sendo tais diferenças mais perceptíveis com a presença de ruídos a imagem.

Em relação ao tempo médio de execução, ambas as soluções obtiveram resultados muito próximos, com tempos de 0,111s e 0,135s para solução de Aprendizado Profundo, resultando em um tempo médio de 0,123s. Já a abordagem de Visão Clássica, obteve um tempo médio de 0,151s, ou seja, há uma desvantagem de, aproximadamente, 22,7% em relação à o RNCs. Destaca-se que, o uso de uma plataforma em nuvem, como o Google Colab, pode acarretar em pequenas variações nos tempos de execução.

Sendo assim, conclui-se que, de maneira geral, a solução referente a abordagem de Aprendizado Profundo se mostrou mais robusta e flexível com a presença de ruídos, se tornando mais viável para situações em que há exposição para tais ruídos. Por outro lado, a Visão Clássica se mostrou mais eficiente em ambientes controlados, onde a presença de ruídos não é tão presente.

Por isso, percebe-se que ambas as soluções se mostraram viáveis em diferentes cenários de aplicabilidade, possuindo vantagens e desvantagens. Pois, a solução utilizando RNCs pode se mostrar mais custosa no quesito de equipamento necessário para realização do processamento de dados. De outra forma, a solução

utilizando Visão Clássica, também, pode ser custosa no quesito de controle do ambiente, tal controle seria necessário para evitar possíveis ruídos.

Por fim, sugere-se como trabalho futuro uma ampliação na base de dados, para assim, atingir maiores taxas de sucesso. Além disso, recomenda-se a implementação de um recorte na imagem a partir do ponto médio encontrado, dando um maior enfoque para o rebite metálico, para que a partir desse recorte, seja implementado à solução a funcionalidade de classificação do rebite.

REFERÊNCIAS

ARAÚJO, F. Redes neurais convolucionais com tensorflow: Teoria e prática. *Escola Regional de informática do Piauí*, p. 382–406, 2017. 65

BRIGNOLI, R. Estudo de redes neurais convolucionais e sua aplicação para classificação de ambientes. *Instituto Federal de Santa Catarina*, p. 54, 2021. 21

CAMUÑAS-MESA, L. A.; LINARES-BARRANCO, B.; SERRANO-GOTARREDONA, T. *Neuromorphic Spiking Neural Networks and Their Memristor-CMOS Hardware Implementations*. *MDPI Sevilla*, p. 28, 2019. Disponível em: https://www.researchgate.net/publication/335438509_Neuromorphic_Spiking_Neural_Networks_and_Their_Memristor-CMOS_Hardware_Implementations. Acesso em: 16 out 2022. 65

CONTER, F. P. Detecção de objetos e estimativa de posição 3D através da aplicação de redes neurais convolucionais. *Universidade Tecnológica Federal do Paraná*, p. 89, 2022. 65

CUNHA, L. C. da. Redes neurais convolucionais e segmentação de imagens - uma revisão bibliográfica. *Universidade Federal de Ouro Preto*, p. 55, 2020. 65

ESQUEF, I. A.; ALBUQUERQUE, M. P. de; ALBUQUERQUE, M. P. de. Processamento digital de imagens. *Centro Brasileiro de Pesquisas Fisicas*, p. 89, 2003. 65

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. *Universidade de São Paulo*, p. 30, 2006. 65

KARPATHY, A. ***Convolutional Neural Networks (CNNs / ConvNets)***. Notas do curso CS231n: Convolutional Neural Networks for Visual Recognition. Disponível em: https://cs231n.github.io/convolutional-networks/. Acesso em: 2 nov 2022. 65

PUROHIT, R. ***Fully Convolutional Network (Semantic Segmentation)***. Great Learning Blog, 2022. Disponível em: https://www.mygreatlearning.com/blog/fcn-fully-convolutional-network-semantic-segmentation/. Acesso em: 3 nov 2022. 65

RAUBER, T. W. Redes neurais artificiais. *Universidade Federal do Espírito Santo*, p. 28, 2005. 65

SAGAR, A. ***Deep Learning for Image Classification with Less Data***. KD Nuggets, 2019. Disponível em: https://www.kdnuggets.com/2019/11/deep-learning-image-classification-less-data.html. Acesso em: 18 out 2022. 65

SHAKHADRI, S. A. G. ***Build ResNet from Scratch With Python***. Analytics Vidhya, 2021. Disponível em: https://www.analyticsvidhya.com/blog/2021/06/build-resnet-from-scratch-with-python/. Acesso em: 19 out 2022. 65

SINGHAL, G. ***Transfer Learning with ResNet in PyTorch***. PluralSight, 2019. Disponível em: https://www.pluralsight.com/guides/introduction-to-resnet. Acesso em: 18 out 2022. 65

VALMORBIDA, A. D. Desenvolvimento de um sistema de processamento de imagens para inspeção automatizada utilizando FPGA. *Instituto Federal de Santa Catarina*, p. 91, 2018. 22

VARGAS, A. C. G.; CARVALHO, A. M. P.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. *Universidade Federal Fluminense*, p. 4, 2016. 65