Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Trabalho de Conclusão de Curso

Desenvolvimento de um Sistema de Classificação de Peças em Ambiente Industrial Utilizando Visão Computacional e a Arquitetura YOLO

Lucas Dantas Pereira

Campina Grande - PB

Maio de 2024

Lucas Dantas Pereira

Desenvolvimento de um Sistema de Classificação de Peças em Ambiente Industrial Utilizando Visão Computacional e a Arquitetura YOLO

Trabalho de Conclusão de Curso submetido à Coordenaria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Engenheiro Eletricista.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Coordenação de Graduação em Engenharia Elétrica - CGEE

Péricles Rezende Barros, Dr.

(Orientador)

Campina Grande - PB

Maio de 2024

Lucas Dantas Pereira

Desenvolvimento de um Sistema de Classificação de Peças em Ambiente Industrial Utilizando Visão Computacional e a Arquitetura YOLO

Trabalho de Conclusão de Curso submetido à Coordenaria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Engenheiro Eletricista.

Aprovado em _____ /_____ /_____

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Péricles Rezende Barros
Universidade Federal de Campina Grande
Orientador

Campina Grande - PB
Maio de 2024

"Dedico este trabalho principalmente aos meus pais, Orestes Pereira Filho e Gilvanete Dantas de Oliveira Pereira, e a todos que contribuíram de alguma maneira para a minha formação."

Agradecimentos

Agradeço primeiramente a Deus por me permitir trilhar meu caminho até aqui. Apesar de todos os dias de estresse e noites sem dormir, também vivi alegrias, conquistas e fiz amizades que levarei para a vida toda.

Agradeço à minha família por sempre estar ao meu lado, proporcionando apoio e suporte, além de todas as condições necessárias para que eu pudesse continuar minha jornada.

Aos meus professores, agradeço por todo o aprendizado e oportunidades que me foram proporcionadas. Em especial, agradeço ao professor Péricles Rezende Barros e à Anna Paula, que me orientaram neste trabalho e me ajudaram a concluir esta etapa.

Agradeço a todos os amigos que compartilharam essa jornada comigo na universidade, especialmente a Biancca Cavalcante, Dhara Pamplona, Douglas Almeida, Gabriel Marques, Isac Almeida, Larissa Teixeira, Lucas Viana, Marcos Henrique, Roberto Dourado e, por último mas não menos importante, Victor Hugo. Pelas noites de estudo, conversas intermináveis e celebrações de conquistas, saibam que vocês são laços que valorizo imensamente e pretendo levar para toda a vida. Este percurso não seria possível sem o apoio e o companheirismo de vocês. Obrigado por sempre acreditarem em mim.

Aos amigos que me acompanham desde a escola e aos que fiz durante o caminho, especialmente Karol Mendes, Lorena Caramel e Victor Henrique, agradeço por estarem sempre presentes e por serem fontes constantes de apoio e amizade verdadeira. Obrigado por me mostrarem que posso contar com vocês a qualquer dia e qualquer hora.

Por fim, agradeço a mim mesmo por me permitir concluir a graduação, por toda a dedicação e por, mesmo nos momentos difíceis, sempre ter seguido em frente e perseverado sem desistir.

"Se cheguei até aqui foi porque me apoiei no ombro dos gigantes.",
(Isaac Newton)

Resumo

O presente trabalho tem como objetivo fornecer uma alternativa para a detecção de peças, substituindo os sensores convencionais e, em algumas situações, oferecendo uma solução viável quando os sensores tradicionais não são eficientes. Para alcançar esse objetivo, o estudo foi desenvolvido com base no Sistema Modular de Produção, que simula um ambiente industrial em escala laboratorial. Utilizou-se o modelo de rede neural pré-treinado YOLO (*You Only Look Once*) para a identificação das peças, ajustando-o especificamente para essa tarefa. A implementação do algoritmo foi realizada em Python, devido à sua ampla gama de bibliotecas de aprendizado de máquina e facilidade de integração com ferramentas de visão computacional. A rede neural convolucional profunda demonstrou uma precisão de 98,9%, recall de 100% e F1-score de 99,3%, indicando um desempenho promissor do modelo.

Palavras-chave: Visão Computacional, YOLO, Redes Neurais Convolucionais, Detecção de Objetos, Sensores, Ambiente Industrial, Python, Aprendizado de Máquina, Algoritmos de Detecção, Processamento de imagens, Automação Industrial, Análise de Imagens, Inteligência Artificial, Transferência de Conhecimento, Redes Neurais Profundas.

Abstract

The present work aims to provide an alternative for part detection, replacing conventional sensors and, in some situations, offering a viable solution when traditional sensors are inefficient. To achieve this objective, the study was developed based on the Modular Production System, which simulates an industrial environment on a laboratory scale. The pre-trained neural network model YOLO (You Only Look Once) was used for part identification, specifically adjusting it for this task. The algorithm was implemented in Python due to its wide range of machine learning libraries and ease of integration with computer vision tools. The deep convolutional neural network demonstrated an accuracy of 98.9%, recall of 100%, and an F1-score of 99.3%, indicating a promising performance of the model.

Keywords: Computer Vision, YOLO, Convolutional Neural Networks, Object Detection, Sensors, Industrial Environment, Python, Machine Learning, Detection Algorithms, Image Processing, Industrial Automation, Image Analysis, Artificial Intelligence, Knowledge Transfer, Deep Neural Networks.

Lista de ilustrações

Figura 1 – Modelo de um neurônio artificial 5
Figura 2 – Rede neural simples e rede neural profunda Deep Learning 7
Figura 3 – Processo de convolução em imagem RGB 10
Figura 4 – Exemplo de *max pooling* 12
Figura 5 – Método YOLO para detecção de objetos 13
Figura 6 – Linha do tempo das versões do YOLO. 14
Figura 7 – Arquitetura de rede do YOLO. 16
Figura 8 – Arquitetura *Darknet-53* 16
Figura 9 – Estações do sistema modular de produção disponíveis no LIEC. 18
Figura 10 – Peças processadas pelo MPS. 19
Figura 11 – Componentes do Sistema de Controle e Interface de Operação. 20
Figura 12 – Adaptações Realizadas 22
Figura 13 – Imagens Extraídas da Base de Dados 23
Figura 14 – Ferramenta de Rotulagem Make Sence 24
Figura 15 – Previsões Feitas Pelo Algoritimo. 27
Figura 16 – Matriz de Confusão 29
Figura 17 – Matriz de Confusão do Modelo 31
Figura 18 – Matriz de Confusão: Resultado do Modelo 32
Figura 19 – Previsões Realizadas no MPS. 34

Lista de tabelas

Tabela 1 – Desempenho dos Modelos YOLOv8 25
Tabela 2 – Métricas de avaliação do modelo 32

Lista de abreviaturas e siglas

RNA	Redes Neurais Artificiais
CNN	*Convolutional Neural Network*
FN	Falso Negativo
FP	Falso Positivo
ReLU	*Rectified Linear Activation*
ResNet	Rede Neural Residual
Tanh	Tangente Hiperbólica
VN	Verdadeiro Negativo
VP	Verdadeiro Positivo
YOLO	*You Only Look Once*
ML	*Machine Learning*
LTU	*Linear Threshold Unit*
RGB	*Red, Green and Blue*
MPS	*Modular Production System*
LIEC	Laboratório de Instrumentação Eletrônica e Controle
CLP	Controlador Lógico Programável
E/S	Entradas e Saída
COCO	*Common Objects in Context*
CPU	Unidade Central de Processamento
GPU	Unidade de Processamento Gráfico
fps	*Frames* por Segundo

Sumário

1 INTRODUÇÃO 1
1.1 Objetivo Geral 2
1.2 Objetivos Específicos 2
1.3 Organização do Trabalho 2

2 FUNDAMENTAÇÃO TEÓRICA 3
2.1 Visão Computacional 3
2.2 Redes Neurais 4
2.2.1 Neurônio Artificial 5
2.2.2 Camadas 6
2.2.3 Redes Neurais Profundas 8
2.2.4 Redes Neurais Convolucionais 9
2.2.5 Camada de *Pooling* 11
2.3 YOLO 12
2.3.1 Detecção 12
2.3.2 Versões do YOLO 13
2.3.2.1 YOLOv1 14
2.3.2.2 YOLOv2: melhor, mais rápido e mais forte 14
2.3.2.3 YOLOv4 15
2.3.2.4 YOLOv5 15
2.3.2.5 YOLOv8 15
2.3.3 Arquitetura 15

3 MATERIAIS E MÉTODOS 18
3.1 Módulo FESTO 18
3.1.1 Componetes do Módulo 19
3.1.1.1 Peças 19
3.1.1.2 Sistema de Controle e Interface de Operação 19
3.1.2 Estações 20
3.1.2.1 Estação de Distribuição 20
3.1.2.2 Estação de separação 21
3.1.2.3 Estação de Classificação 21
3.1.3 Integração da Visão Computacional ao Sistema MPS: Motivação e Adaptações 21
3.2 Preparação do Ambiente de Trabalho 22
3.3 Base de Dados 23
3.4 Modelo 25

3.5 Treinamento 28

4 RESULTADOS OBTIDOS 29
4.1 Resultados do Modelo 30
4.2 Análise das Métricas 33
4.3 Processamento em Tempo Real 33

5 CONCLUSÕES 36
5.1 Perspectivas para o Futuro 36

REFERÊNCIAS BIBLIOGRÁFICAS 38

1 Introdução

A Revolução Industrial teve seu início no século XVIII, transformando radicalmente os processos produtivos, impulsionando assim a humanidade em direção a novos paradigmas econômicos e sociais. Ao decorrer dos anos, a evolução tecnológica foi um elemento constante, acarretando na atual era da Indústria 4.0, caracterizada pela integração intensiva de tecnologias digitais nos sistemas de produção. Nesse contexto, a crescente necessidade de aumento da eficiência e produtividade nas linhas de manufatura impulsiona a busca incessante por soluções inovadoras (NEVES, 2024).

Diante dos desafios impostos pela Indústria 4.0, a evolução da inteligência artificial emerge como uma ferramenta essencial. Devido isso, a visão computacional, que combina conceitos de processamento de imagem e aprendizado de máquina, destaca-se como uma abordagem única para aprimorar a automação industrial. A capacidade dos sistemas visuais de interpretar e compreender o ambiente ao seu redor torna-se fundamental para otimizar processos e garantir a qualidade na produção.

Além disso, a necessidade de integração de sistemas heterogêneos, a garantia da segurança cibernética e a capacitação da força de trabalho são desafios cruciais que as empresas enfrentam na busca pela transformação digital e pela eficiência na Indústria 4.0. Nesse contexto, a visão computacional surge como uma solução promissora para enfrentar esses desafios, oferecendo novas formas de coletar, processar e interpretar dados visuais para tomada de decisão e automação de processos industriais (SZELISKI, 2022).

Assim sendo, este trabalho propõe a utilização da visão computacional como alicerce para aprimorar a eficiência em ambientes industriais. Neste contexto, destaca-se o foco especial na identificação e classificação de peças em uma bancada didática de manufatura desenvolvida pela FESTO. Esta bancada é composta por três módulos simples - Distribuição, Separação e Classificação - que podem ser combinados para simular ambientes industriais. Com a implementação de um algoritmo baseado na arquitetura YOLO (*You Only Look Once*), o objetivo principal é oferecer uma solução ágil e precisa para a detecção de elementos na linha de produção.

A escolha do YOLO se fundamenta em sua notável eficácia em tempo real, proporcionando um desempenho excepcional na identificação simultânea de múltiplos objetos em imagens. Este modelo se destaca por sua abordagem única de detecção, onde a imagem é analisada em uma única iteração. Além disso, promete uma integração eficiente com sistemas de automação, permitindo uma resposta precisa às demandas da produção industrial (HUANG; PEDOEEM; CHEN, 2018)(HUSSAIN, 2023).

Este projeto propõe uma forma alternativa de detectar produtos em linhas de

produção, seja para classificar e separar, ou até mesmo para detectar produtos defeituosos, inserindo-se de maneira estratégica na realidade da Indústria 4.0. Ao oferecer uma solução prática para os desafios enfrentados pelas linhas de manufatura contemporâneas, este trabalho explora as capacidades do YOLO em um ambiente industrial.

1.1 Objetivo Geral

Este trabalho tem como objetivo geral desenvolver um sistema que possibilite a classificação das peças de manufatura presentes no módulo didático desenvolvido pela FESTO (RAMOS, 2019), utilizando técnicas de visão computacional.

1.2 Objetivos Específicos

Como forma de mapear e organizar as etapas necessárias para atingir o objetivo geral proposto, foram definidos os seguintes objetivos específicos:

- Estudar e avaliar bibliotecas de código livre baseadas em aprendizado de máquina e/ou inteligência artificial voltadas para o processamento de imagens e vídeos;
- Desenvolver um algoritmo capacitado para detectar as peças de manufatura do módulo FESTO;
- Efetuar as classificações adequadas das peças identificadas, proporcionando um sistema de reconhecimento robusto e preciso.

1.3 Organização do Trabalho

O presente trabalho está organizado em 5 capítulos. No Capítulo 1 é feita uma introdução do trabalho e é apresentado os objetivos gerais e específicos.

No Capítulo 2, são descritos os conceitos básicos necessários para a compreensão de Redes Neurais Artificiais e Redes Neurais Convolucionais, assim como é apresentada a arquitetura do YOLO.

O Capítulo 3 trata com detalhes a metodologia adotada e as atividades desenvolvidas objetivando identificar e reconhecer peças do MPS.

No Capitulo 4, são apresentados os resultados obtidos do trabalho e discussões acerca dos resultados.

O Capítulo 5 discute as conclusões do trabalho realizado e propõe direções futuras para complementar e aprimorar as atividades desenvolvidas.

2 Fundamentação Teórica

Neste capítulo será apresentada a revisão dos principais tópicos que foram utilizados no desenvolvimento deste trabalho. Entre eles, é válido destacar conceitos sobre ANN (*Artificial Neural Network*) e as CNNs (*Convolutional Neural Networks*), examinando seus métodos e as arquiteturas.

2.1 Visão Computacional

A visão computacional é uma área importante na interseção entre ciência da computação e inteligência artificial, pois há uma demanda crescente por sistemas inteligentes capazes de compreender e interagir com o mundo visual. Digamos que, em uma situação hipotética em que temos uma cena simples de uma bola sendo arremessada para uma pessoa e ela pega essa bola. Por trás dessa aparente simplicidade, está um dos processos mais intrincados que os seres humanos já tentaram desvendar: como o cérebro processa a informação visual para identificar, interpretar e reagir a objetos em movimento (VISãO. . . ,).

A tarefa de ensinar uma máquina a ver e compreender imagens de maneira semelhante à do ser humano é extremamente desafiadora, não apenas pela dificuldade de reproduzir matematicamente o complexo processo de visão humana, mas também pela própria natureza enigmática desse processo. Afinal, ainda não temos compreendido totalmente como o cérebro humano interpreta o mundo visual ao seu redor.

Em sua essência, a visão computacional visa modelar e replicar o complexo processo visual humano por meio de uma combinação de *hardware* e *software*. Essa área abrange não apenas o processamento de imagens, mas também técnicas sofisticadas de reconstrução tridimensional, análise de padrões e identificação de eventos e objetos. Seu alcance é amplo e diversificado, incluindo diagnósticos médicos e identificação facial em plataformas de mídia social, demonstrando sua versatilidade e eficácia em vários contextos.

Esta tecnologia promete revolucionar a forma como percebemos e interagimos com o mundo visual por meio de sua notável aplicação no campo do reconhecimento de imagens. Ao reproduzir a capacidade visual humana, ela permite que sistemas inteligentes compreendam e interajam melhor com o mundo.

A distinção entre processamento de imagens e visão computacional não é sempre clara. A visão computacional visa replicar a capacidade humana de interpretar e compreender uma imagem como um todo ou parcialmente. Isso ocorre porque o processamento de imagens envolve a conversão de uma imagem de entrada em um conjunto de valores

numéricos, que pode ou não produzir uma nova imagem. Conforme Gonzalez, Woods e Eddins (2006), os processos de visão computacional normalmente começam com o processamento de imagens, que pode ser classificado em três níveis de complexidade: baixo, médio e alto. As operações de baixo nível incluem tarefas primitivas, como redução de ruído e ajuste de contraste, enquanto as de nível médio envolvem segmentação da imagem em regiões ou classificação de objetos. Já os processos de alto nível estão relacionados às habilidades cognitivas associadas à interpretação visual humana.

Recentemente, avanços significativos na computação paralela impulsionadas pelo uso de GPUs e na capacidade de processamento computacional estimularam o desenvolvimento de redes neurais artificiais, especialmente no campo de aprendizagem profunda e das redes neurais convolucionais. A rapidez e a precisão com que essas tecnologias estão reconhecendo padrões estão revolucionando a visão computacional.

Com os avanços contínuos nas tecnologias de aprendizado de máquina e processamento de imagens, a visão computacional tem demonstrado um potencial significativo em diversos setores. Como resultado, a visão computacional não apenas imita a percepção visual humana, mas também a expande e aprimora, abrindo caminho para uma nova era de sistemas inteligentes que podem compreender e interagir com o mundo ao seu redor de forma mais sofisticada.

2.2 Redes Neurais

As redes neurais artificiais (RNA) são estruturas inspiradas no funcionamento do cérebro humano e representam um tipo específico de algoritmo de inteligência computacional. Elas se assemelham ao cérebro em dois aspectos fundamentais: a aquisição de conhecimento pela rede a partir de seu ambiente, através de um processo de aprendizagem, e o armazenamento desse conhecimento por meio de forças de conexão entre neurônios, conhecidas como pesos sinápticos. Esse processo está incluído no aprendizado de máquina ou *machine learning* (ML) (HAYKIN, 2009).

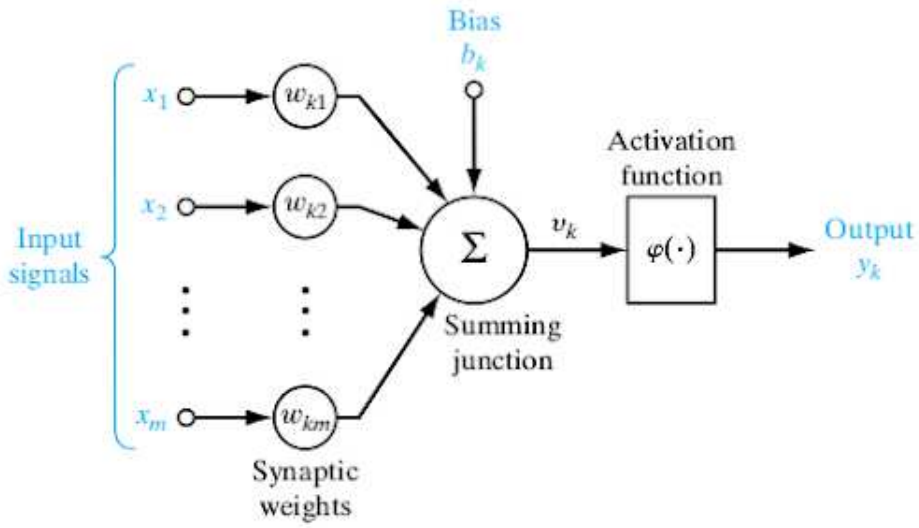
A ML é o campo da ciência da computação em que os computadores são capacitados para aprender a partir de dados. Uma definição clássica na literatura é a seguinte: ”Um programa de computador aprende com a experiência E em relação a uma tarefa T e uma medida de desempenho P se o seu desempenho em T, conforme medido por P, melhora com a experiência E” (MITCHELL, 1997).

As RNA são comumente descritas como uma rede de neurônios artificiais interligados, os quais são capazes de aprender por meio de exemplos, emulando o funcionamento dos neurônios biológicos. Este processo é conduzido através de uma aprendizagem interativa que se aprimora com a experiência, refletindo o modo como o cérebro humano opera. A seção a seguir apresenta a definição de um neurônio em uma rede neural artificial.

2.2.1 Neurônio Artificial

Na década de 1940 até 1960, o conceito de RNA era pouco relevante e as entradas eram apenas valores binários. Alguns anos depois do desenvolvimento desse conceito, o americano Frank Rosenblatt trouxe uma arquitetura simples de neurônio, conhecido como *percepetron* que é usado como exemplo didático até os dias atuais. O *perceptron* é composto apenas por um neurônio artificial diferente, chamado de unidade linear de limiar (LTU - *Linear Threshold Unit*), ele possui entradas (*inputs*) responsáveis por processa-la e gerar uma saída (*output*) que estão diretamente associadas a um peso. A LTU calcula a soma ponderadas das entradas e aplica uma função degrau a esta soma. O *percepetron* é capaz de resolver apenas problemas linearmente separáveis com classificação binária (GéRON, 2017). O neurônio artificial em termos matemáticos, é um componente que calcula a soma ponderada de várias entradas, aplica uma função e apresenta uma saída. A representação do neurônio é ilustrada na Figura 1.

Figura 1 – Modelo de um neurônio artificial



Fonte: Haykin (2009)

O modelo neural é composto por um conjunto de sinapses, cada uma representada por uma entrada X e seu respectivo peso W. Em seguida, há um somador encarregado de somar as entradas multiplicadas por seus pesos correspondentes. Por fim, a saída do neurônio é submetida a uma função de ativação, que controla a amplitude da saída do neurônio, limitando-a dentro de determinados limites (HAYKIN, 2009). Temos assim a seguinte expressão:

$$bk = X1 * Wk1 + X2 * Wk2 + \cdots + Xm * Wkm \tag{2.1}$$

Onde:

- Xm é a entrada.
- Wkm é o peso.
- bk é o somatório
- f(bk) é a função de ativação

A maioria dos modelos de redes neurais aprendem através de exemplos. Geralmente, é definido alguma regra de treinamento, onde os pesos de suas conexões são ajustados de acordo com os padrões apresentados.

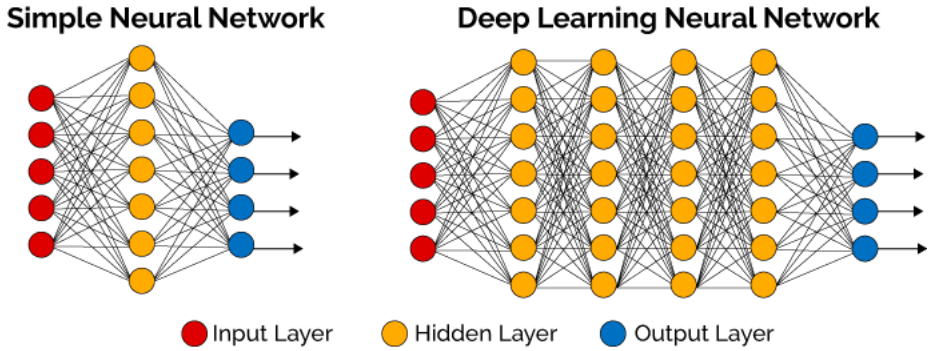
2.2.2 Camadas

Considerando que as RNA são modeladas com base no funcionamento do cérebro humano e de seus neurônios, essas redes são estruturadas em camadas (NETO, 2019), compostas por unidades organizadas sequencialmente para formar o sistema de uma RNA (GeeksforGeeks, 2023). Essas camadas podem ser divididas principalmente em três tipos:

- Camada de Entrada: Esta camada consiste nos elementos iniciais da rede, responsáveis por receber as características ou informações de entrada;
- Camadas Intermediárias ou Ocultas: Localizadas entre a camada de entrada e a de saída, essas camadas são responsáveis pelo processamento dos dados. Elas desempenham um papel crucial na extração e transformação das características do sistema, contribuindo para a representação dos padrões presentes nos dados;
- Camada de Saída: Esta camada fornece o valor final ou a resposta da rede neural após o processamento das informações pelas camadas intermediárias. Ela representa a saída ou a resposta do sistema neural em relação à entrada fornecida.

Essa organização em camadas permite que as redes neurais realizem uma variedade de tarefas, desde reconhecimento de padrões até tomada de decisões, em diversos domínios de aplicação. As camadas de redes neurais artificiais simples e profundas podem ser vistas na Figura 2.

Figura 2 – Rede neural simples e rede neural profunda Deep Learning



Fonte: Analytics Vidhya (2021)

Na representação acima, os círculos vermelhos representam os neurônios na camada de entrada, os círculos amarelos representam os neurônios nas camadas intermediárias e os círculos azuis representam os neurônios na camada de saída. As setas representam as saídas da rede neural, enquanto as linhas entre os neurônios representam as conexões que os conectam de uma camada para outra. É importante destacar que cada camada contém um determinado número de neurônios, que ajudam no processamento e representação da informação ao longo da rede.

Na maioria das redes neurais, as unidades são conectadas entre as diferentes camadas, e cada uma dessas conexões possui pesos que determinam sua influência nas outras unidades como pode ser visto na Eq 2.2. Dessa forma, à medida que os dados são transferidos de uma unidade para outra, a rede neural aprende mais sobre eles, culminando em uma camada de saída.

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.2}$$

Onde:

- y é a saída da camada.
- f é a função de ativação.
- Wi são os pesos associados às características de entrada xi
- b é o viés (bias).
- n é o número de características de entrada.

Durante o treinamento da rede neural, os pesos Wi estão relacionados com as características de entrada Xi. Eles são ajustados iterativamente com o objetivo de minimizar

a diferença entre as saídas previstas e os rótulos reais. Além disso, cada unidade incorpora um viés b, que é um parâmetro permitindo deslocamentos e ajustes na saída.

O resultado é então submetido a uma função de ativação, conforme ilustrado na equação. O objetivo desta função é alterar o valor que foi calculado no neurônio antes de ser transferido para a próxima camada ou visto como a saída final da rede. A função de ativação é semelhante a um mecanismo de decisão que escolhe se um neurônio envia ou não um sinal para outro. No entanto, sua função é mais do que isso. O valor de saída das redes neurais artificiais pode ser transformado de várias maneiras. Isso inclui a conversão desse valor em uma classe final, que é essencial para atividades de classificação e outras tarefas relacionadas ao aprendizado de máquina (MARTINS, 2018).

Existe uma grande variedade de funções de ativação disponíveis, cada uma com suas próprias características e usos específicos. Essas funções são essenciais para a capacidade das redes neurais de aprender e representar padrões de dados de forma eficaz. Essas funções incluem softmax, ativação linear, sigmoide, ativação linear retificada (ReLU) e tanh (tangente hiperbólica). A ReLU é uma das opções mais populares e usadas. Ela tem uma característica distinta. Se o valor de entrada for positivo, ela ativa um neurônio e mantém o valor inalterado, enquanto se for negativo, ele o zera. Essa conduta tem um impacto significativo na resolução de problemas de dissipação do gradiente, que são comuns no treinamento de redes neurais profundas. Assim, escolher a função de ativação adequada é essencial para a rede neural aprender de forma eficiente e obter os padrões dos dados de entrada para realizar tarefas específicas com precisão.

Essa diversidade de funções permite que as redes neurais se adaptem a diferentes contextos e desafios, influenciando de forma precisa e direta na capacidade do modelo de aprender e generalizar a partir dos dados de entrada.

2.2.3 Redes Neurais Profundas

Uma rede neural profunda se distingue pelo seu arranjo de camadas ocultas, que é notavelmente mais extenso em comparação com uma rede neural simples, como foi mostrado na Figura 2. O processo de aprendizagem é significativamente afetado por esse aumento na profundidade da rede, especialmente em tarefas de alta complexidade. A rede neural profunda permite que as máquinas trabalhem com mais precisão ao lidar com essas tarefas difíceis. Segundo Deng e Yu (2014), o aprendizado profundo é uma derivação do campo de aprendizado de máquina, onde as redes neurais artificiais aprendem e se adaptam a grandes conjuntos de dados. Essas redes usam várias camadas para extrair características importantes gradualmente de uma entrada bruta. Esse processo permite que a rede desenvolva a capacidade de prever e classificar informações com maior acurácia e eficiência.

O número de camadas intermediárias (ocultas) entre as entradas e as saídas de uma rede neural é conhecido como sua profundidade. Em outras palavras, a profundidade de uma rede aumenta com o número de camadas que ela possui. Brigade (2016) afirmou que as máquinas treinadas com aprendizado profundo para reconhecimento de imagens têm taxas de precisão superiores às dos seres humanos em alguns cenários.

Uma subcategoria específica de Redes Neurais Artificiais emergiu com o tempo, incorporando a técnica de convolução em suas camadas, permitindo avanços fundamentados no *Deep Learning*. As Redes Neurais Convolucionais (CNNs) serão examinadas mais detalhadamente a seguir.

2.2.4 Redes Neurais Convolucionais

As redes neurais convolucionais representam uma subdivisão do campo do *Deep Learning*, originadas das Redes Neurais Artificiais, e são notáveis por incorporar camadas de convolução em sua arquitetura. As CNNs e o avanço da quantidade e qualidade dos dados foram fundamentais para o avanço científico do aprendizado profundo. Contudo, as CNNs são especificamente desenvolvidas para lidar com o processamento de imagens (MARTINS, 2018).

A camada de convolução é uma camada fundamental para a construção de uma rede neural convolucional. A função dessa camada é convoluir as imagens de entrada com um conjunto de filtros cujos valores dos pesos são aprendidos durante o processo de treinamento. Os filtros são elementos que extraem certas características das imagens.

A camada de convolução desempenha um papel central na arquitetura de uma CNN, sendo fundamental para processar imagens de forma eficiente. Sua principal função é aplicar a operação de convolução nas imagens de entrada utilizando uma série de filtros, cujos pesos são ajustados durante o processo de treinamento da rede. Esses filtros funcionam como elementos que extraem características específicas das imagens, como bordas, texturas ou padrões.

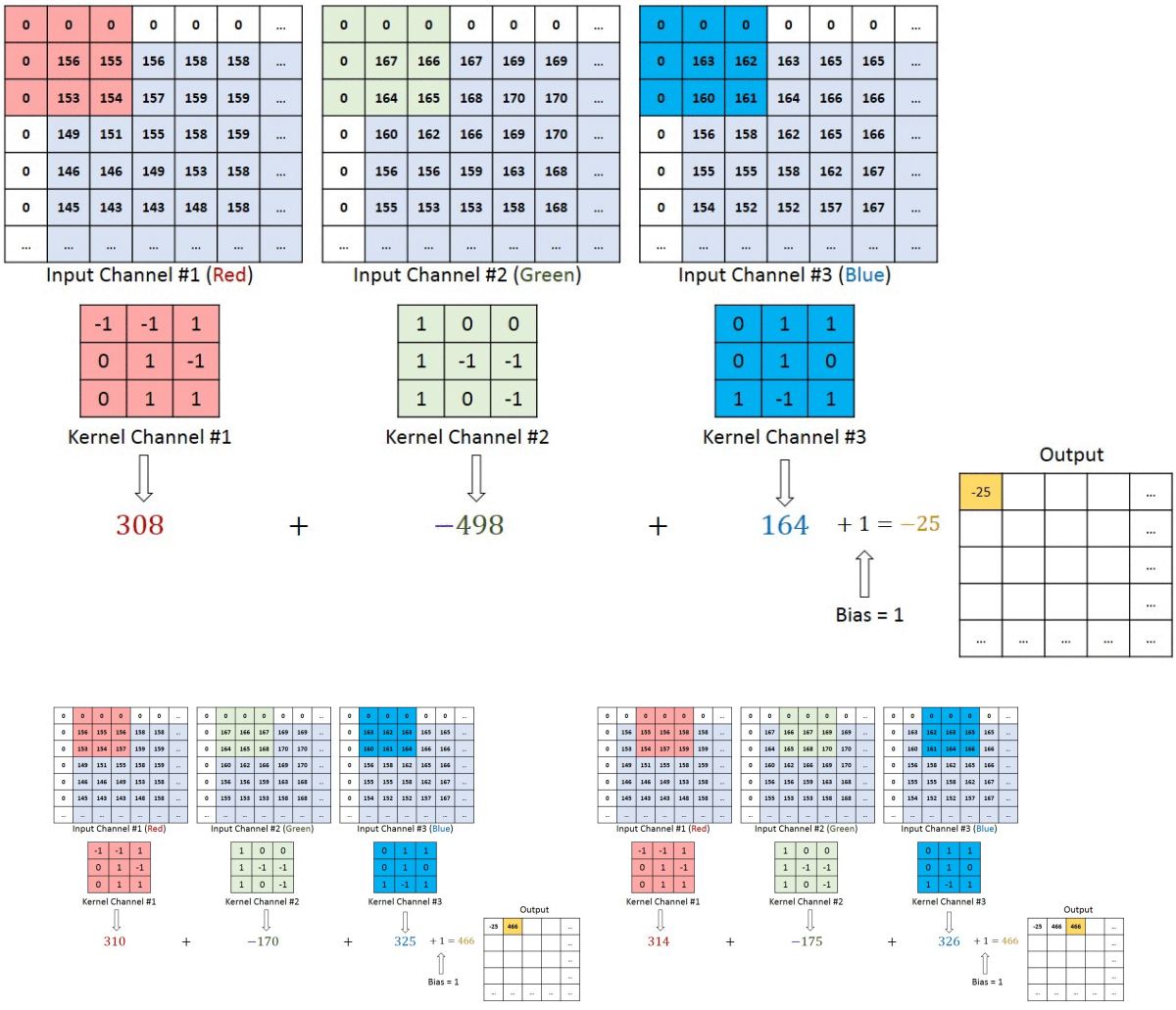
Um aspecto importante é que as características a serem extraídas não são pré-definidas; ao contrário, são aprendidas pela rede durante o treinamento. Isso significa que a CNN se adapta dinamicamente ao problema em questão, ajustando seus filtros e pesos internos para resolver eficientemente a tarefa de interesse. Dessa forma, a rede é capaz de aprender a reconhecer e interpretar os padrões relevantes para a tarefa específica, independentemente da complexidade ou da natureza dos dados de entrada.

Na prática, os conjuntos de filtros são movidos sobre os dados de entrada, os quais são representados por uma matriz. Essa matriz é aplicada em regiões específicas dos dados de entrada através de operações de multiplicação. Assim, o filtro executa o produto entre a matriz de convolução e os elementos correspondentes dos dados de entrada, produzindo um

valor resultante. Esse valor resultante é então transmitido como entrada para a próxima camada da rede neural.

Considerando que uma imagem geralmente é representada em três dimensões (altura x largura x número de canais), a Figura 3 ilustra um exemplo de processo de convolução em uma imagem RGB. A imagem de entrada possui dimensões de 6 x 6 x 3, enquanto o filtro é 3 x 3 x 3 e a imagem de saída tem dimensões de 4 x 4. Durante a operação convolucional, o filtro é inicialmente posicionado no canto superior esquerdo da imagem de entrada, onde ocorre a multiplicação pixel a pixel entre a imagem de entrada e o filtro, seguida pela soma desses valores. Esse resultado representa o pixel correspondente na imagem de saída (canto superior esquerdo). Para calcular os pixels subsequentes, o filtro é deslocado pela imagem de entrada em uma unidade e a operação de multiplicação e soma é repetida. Esse processo continua até que o filtro tenha percorrido toda a imagem de entrada.

Figura 3 – Processo de convolução em imagem RGB



Fonte: Alves (2018)

Na camada convolucional, para cada par (imagem e filtro), é gerada uma imagem de saída denominada mapa de características. Além de extrair as características da imagem, essa camada também reduz o número de pixels na imagem de saída, o que consequentemente diminui a quantidade de dados na rede.

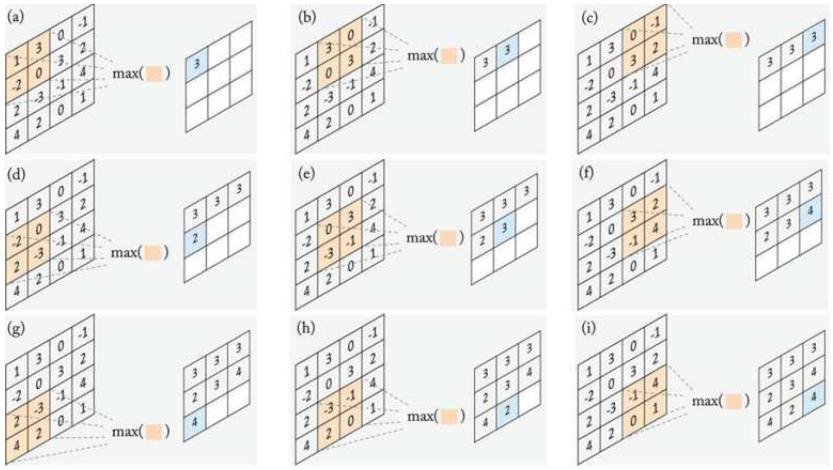
2.2.5 Camada de *Pooling*

A camada de *pooling* desempenha um papel crucial na redução do tamanho da imagem, o que acelera o processamento da rede neural enquanto preserva as características mais importantes. Ao reduzir a imagem, diminui-se o número de parâmetros, resultando em menos cálculos e, consequentemente, em uma rede mais eficiente. A saída dessa camada é uma versão compacta da imagem original, mantendo a integridade dos dados relevantes. Normalmente, essa camada é inserida entre duas camadas de convolução para otimizar o fluxo de informações.

O processo de *pooling* consiste em receber o mapa de características gerado pela camada de convolução anterior e criar um mapa de características condensado. Este processo é conduzido por uma função de *pooling*, que pode operar selecionando o valor mínimo, médio ou máximo de uma região específica. Em aplicações de visão computacional, a técnica mais frequentemente utilizada é o *max pooling*, que seleciona o valor máximo de cada região, preservando assim as características mais salientes da imagem.

Tomando como exemplo a função de *max pooling*, a imagem de entrada é dividida em regiões e, para cada região, a saída corresponde ao valor máximo encontrado. A Figura 4 ilustra o funcionamento da camada de *pooling* usando a função de *max pooling*. Os hiperparâmetros neste exemplo incluem um filtro de tamanho 2 e um *stride* (passo) de 2. Dada uma imagem de entrada com um conjunto específico de características, um valor alto na saída indica a detecção de uma característica particular. Por exemplo, o quadrante superior direito da saída pode sinalizar a presença dessa característica específica.

Figura 4 – Exemplo de *max pooling*



Fonte: Kocakanat e Serif (2021)

2.3 YOLO

O algoritmo YOLO (*You Only Look Once*), desenvolvido por Redmon et al. (2016), destina-se à classificação e detecção de objetos em uma única passagem. Diferenciando-se de abordagens anteriores, onde sistemas inicialmente concebidos como classificadores eram adaptados para detecção, muitas vezes utilizando janelas deslizantes em várias escalas sobre a imagem. O YOLO trata a detecção como um problema de regressão. Neste método, uma grade de retângulos é posicionada sobre a imagem, e as probabilidades das classes de objetos são previstas para cada retângulo.

Essa abordagem permite ao YOLO operar em tempo real, sendo preferido em vários domínios devido à sua velocidade e precisão, tornando-o uma solução robusta para tarefas de detecção de objetos em aplicações onde a resposta em tempo real é crucial.

2.3.1 Detecção

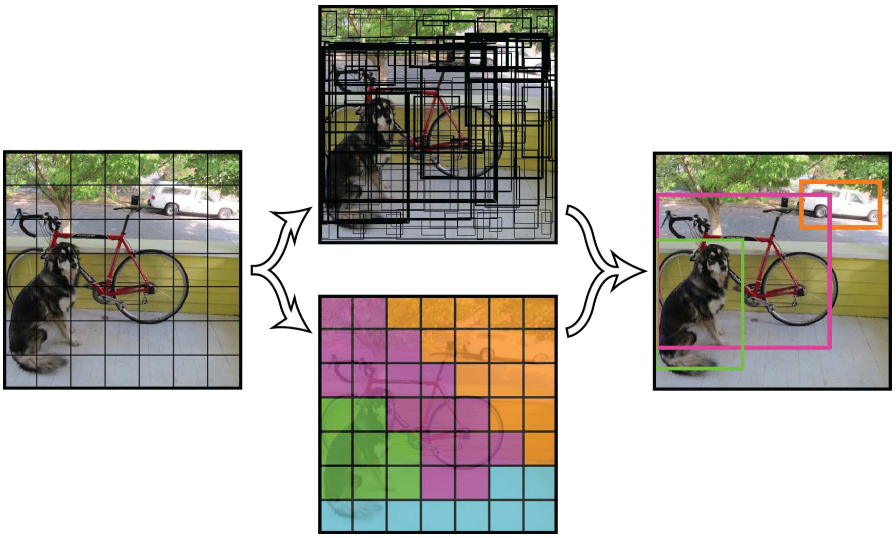
Porém, o que exatamente envolve a detecção de objetos? Conforme Mantripragada (2020) descreve, a detecção de objetos abarca duas principais atividades: a classificação e a localização de objetos, podendo ser aplicada tanto em imagens estáticas quanto em vídeos em tempo real.

A estrutura do YOLO é composta por diversas camadas convolucionais, as quais refinam gradativamente as previsões, capturando diversas características da imagem de entrada. Essa abordagem contribui para aprimorar a acurácia do modelo. Além disso, o YOLO é reconhecido por sua eficiência em cenários com objetos sobrepostos, uma vez que é capaz de detectar múltiplos objetos dentro de uma mesma célula da grade. Essas características fazem do YOLO uma ferramenta valiosa para aplicações que demandam

detecção de objetos em tempo real e com precisão.

Redmon et al. (2016) explicam que o YOLO divide a imagem de entrada em uma grade de células, com cada célula responsável por prever caixas delimitadoras e probabilidades de classe para os objetos dentro dela. Essas caixas delimitadoras são ponderadas pelas probabilidades previstas para eliminar as redundantes ou sobrepostas, garantindo que cada objeto esteja associado a uma célula específica, o que simplifica a tarefa de detecção de objetos e mantém as detecções mais precisas (REDMON; FARHADI, 2018). A rede utiliza características de toda a imagem para realizar a predição de uma caixa delimitadora, efetuando a predição para todas as classes simultaneamente. Caso o centro do objeto esteja em uma célula da grade, esta célula se torna responsável pela detecção do objeto. Cada célula realiza a predição de caixas delimitadoras e o valor de confiança para elas. Todo esse processo pode ser conferido na Figura 5.

Figura 5 – Método YOLO para detecção de objetos

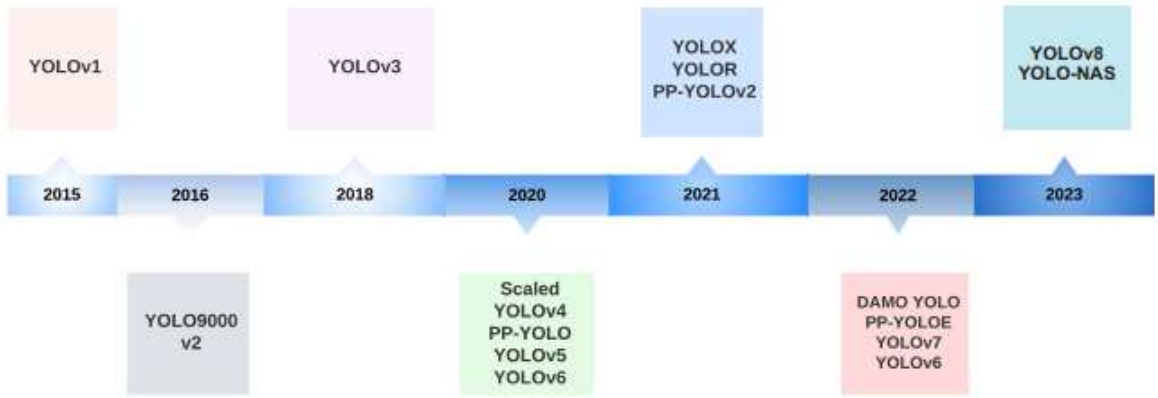


Fonte: Redmon et al. (2016)

2.3.2 Versões do YOLO

Desde sua criação, a família de algoritmos YOLO passou por diversas iterações, cada uma trazendo avanços significativos em relação às suas predecessoras. A evolução das versões do YOLO é detalhada na Figura 6.

Figura 6 – Linha do tempo das versões do YOLO.



Fonte: Terven, Córdova-Esparza e Romero-González (2023)

As novas versões do YOLO foram desenvolvidas para superar as limitações anteriores, melhorando a arquitetura da rede neural e introduzindo novas técnicas de regularização e ancoragem de caixas. Cada iteração se baseou nas melhorias da versão anterior, resultando em algoritmos cada vez mais robustos e eficientes. Essa evolução contínua permitiu que o YOLO se tornasse uma das abordagens mais rápidas e precisas para a detecção de objetos (TERVEN; CóRDOVA-ESPARZA; ROMERO-GONZáLEZ, 2023). As principais versões serão comentadas nas subseções seguintes.

2.3.2.1 YOLOv1

O YOLOv1 representou um avanço significativo na detecção de objetos ao unificar as etapas de detecção, permitindo a detecção de todas as caixas delimitadoras simultaneamente. Sua arquitetura é composta por 24 camadas convolucionais seguidas por 2 camadas totalmente conectadas, responsáveis por prever as coordenadas e probabilidades das caixas delimitadoras. O YOLOv1 é baseado na estrutura da *Darknet*, desenvolvida por Redmon et al. (2016)., escrita principalmente na linguagem de programação C, especificamente adaptada para o YOLO, proporcionando uma base sólida para suas operações.

2.3.2.2 YOLOv2: melhor, mais rápido e mais forte

A segunda versão do YOLO (REDMON; FARHADI, 2017) trouxe avanços significativos, incluindo a capacidade de aumentar o número de classes disponíveis para classificação e melhorias nos resultados da rede neural. Com essas aprimoramentos na arquitetura do YOLOv2, foi possível desenvolver um modelo capaz de classificar objetos em até 9000 classes distintas. Além disso, o YOLOv2 manteve sua reputação como uma das opções mais rápidas para a detecção de objetos na época.

2.3.2.3 YOLOv4

O YOLOv4 viu sua continuidade sob novos desenvolvedores, uma vez que o autor original encerrou suas pesquisas em visão computacional, devido à percepção de seu impacto na sociedade. Os responsáveis pela continuação do projeto foram Alexey Bochkovskiy, Chien-Yao Wang e Hong-Yuan Mark Liao. Esta nova versão teve como objetivo principal criar um sistema eficiente, capaz de operar em ambientes produtivos e otimizado para computação paralela. Dessa forma, possibilita que qualquer indivíduo com uma GPU convencional seja capaz de treinar e testar o modelo com resultados satisfatórios, também incluiu melhorias na detecção de objetos pequenos (TERVEN; CóRDOVA-ESPARZA; ROMERO-GONZáLEZ, 2023).

2.3.2.4 YOLOv5

O YOLOv5 foi lançado alguns meses após o YOLOv4, em 2020, por Glen Jocher, fundador e *CEO* da *Ultralytics*. Ele utiliza muitas das melhorias descritas na seção sobre o YOLOv4, mas foi desenvolvido em *PyTorch*, em vez de *Darknet*. O YOLOv5 incorpora um algoritmo da *Ultralytics* chamado *AutoAnchor*. Esta ferramenta de pré-treinamento verifica e ajusta as caixas de âncoras se elas estiverem inadequadas para o conjunto de dados e configurações de treinamento, como o tamanho da imagem. Inicialmente, ele aplica uma função *k-means* aos rótulos do conjunto de dados para gerar condições iniciais para um algoritmo de evolução genética (GE). O algoritmo GE então evolui essas âncoras ao longo de 1000 gerações, usando a perda de CIoU e a Melhor Recuperação Possível como sua função de aptidão(TERVEN; CóRDOVA-ESPARZA; ROMERO-GONZáLEZ, 2023).

2.3.2.5 YOLOv8

Por fim, temos o YOLOv8, a última versão oficial lançada do YOLO. Ele foi desenvolvido pela mesma equipe responsável pela versão v5. O YOLOv8 oferece cinco versões em escala: YOLOv8n (nano), YOLOv8s (pequeno), YOLOv8m (médio), YOLOv8l (grande) e YOLOv8x (extragrande). Cada versão possui um conjunto de pesos associados, o que aumenta a carga computacional e a precisão da detecção, mas também eleva o tempo de processamento. Portanto, a escolha da escala deve ser ajustada conforme a necessidade específica. Além disso, o YOLOv8 suporta múltiplas tarefas de visão computacional, como detecção de objetos, segmentação, estimativa de pose, rastreamento e classificação(TERVEN; CóRDOVA-ESPARZA; ROMERO-GONZáLEZ, 2023).

2.3.3 Arquitetura

Explorando mais a fundo o funcionamento do YOLO, LI (2019) propôs um esquema, ilustrado na Figura 7, que simplifica a arquitetura do YOLO. Em linhas gerais, a arquitetura é composta por dois componentes principais: o Extrator de Recursos e o Detector de

Recursos. Quando uma nova imagem é recebida, ela passa inicialmente pelo extrator de recursos para obter informações sobre as escalas das imagens. Em seguida, essas informações são encaminhadas para o detector, onde as *bounding boxes* são geradas e as classes são identificadas.

Figura 7 – Arquitetura de rede do YOLO.

Fonte: LI (2019)

De maneira mais detalhada, o YOLOv3 emprega o *Darknet-53*, uma rede neural que consiste em 53 camadas de convolução, construídas sequencialmente com camadas de convolução de "3x3"e "1x1". A arquitetura do *Darknet-53* é ilustrada na Figura 8 (REDMON; FARHADI, 2018).

Figura 8 – Arquitetura *Darknet-53*

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fonte: Redmon e Farhadi (2018)

O detector de recursos, por sua vez, recebe a saída do extrator de recursos. Após

a aplicação da escala das imagens, o detector identifica os objetos dentro da imagem, aplicando as *bounding boxes* juntamente com a identificação da classe.

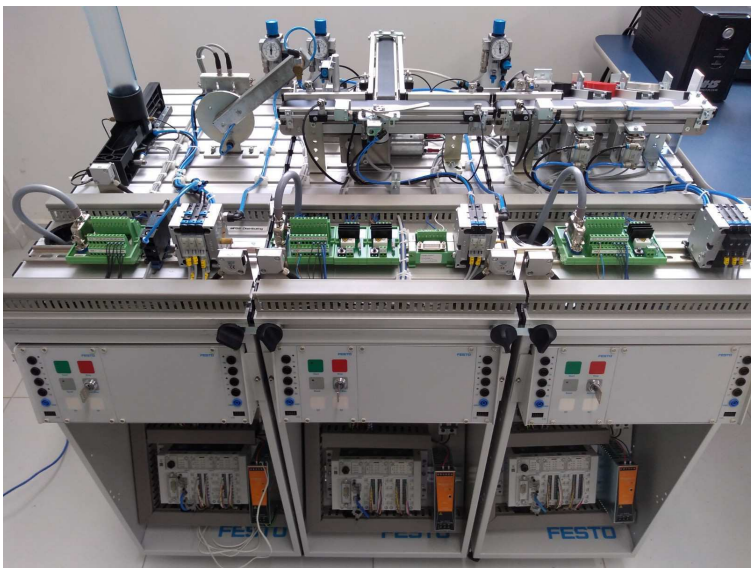
3 Materiais e Métodos

Este capítulo descreve os materiais, técnicas e abordagens utilizadas na contrução deste trabalho. Ele está organizado em seções que foram desenvolvidas para atingir o objetivo proposto. Na primeira seção desse capitulo, o módulo Festo é apresentado, no qual descrevemos como funciona, os motivos de utilizá-lo nesse trabalho e as adaptações necessárias para o desenvolvimento. Posteriomente, na segunda seção será comentada sobre a coleta de dados e o pre-processamento das imagens. Na terceira seção será enfatizado sobre o processado de treinamento do modelo. Por fim, na última seção será feita uma avaliação de desempenho do modelo.

3.1 Módulo FESTO

O Sistema Modular de Produção (MPS - *Modular Production System*) fabricado pela empresa alemã FESTO é composto por estações que realizam funções específicas de manufatura em escala laboratorial. Três módulos foram adquiridos pelo LIEC e podem ser combinados para executar tarefas mais complexas. O MPS é utilizado para o treinamento de pessoal em diferentes áreas relacionadas à produção, tais como: planejamento, montagem, programação, comissionamento, operação e manutenção (RAMOS, 2019).

Figura 9 – Estações do sistema modular de produção disponíveis no LIEC.



Fonte: Ramos (2019)

Podemos observar na Figura 9 os três módulos combinados. O primeiro módulo à esquerda é a estação de distribuição, responsável por alimentar a estação seguinte com

peças. No meio, temos a estação de separação, que identifica se o orifício da peça está voltado para cima ou para baixo, separando-as em dois caminhos distintos. Por fim, à direita, temos a estação de classificação, responsável por categorizar e separar as peças.

3.1.1 Componetes do Módulo

3.1.1.1 Peças

As peças representam produtos em uma linha de produção de manufatura e possuem uma configuração cilíndrica. Uma das bases desses componentes apresenta um orifício que identifica sua posição durante o processamento. O MPS inclui três categorias distintas de componentes: preto, vermelho e metálico, conforme ilustrado na Figura 10.

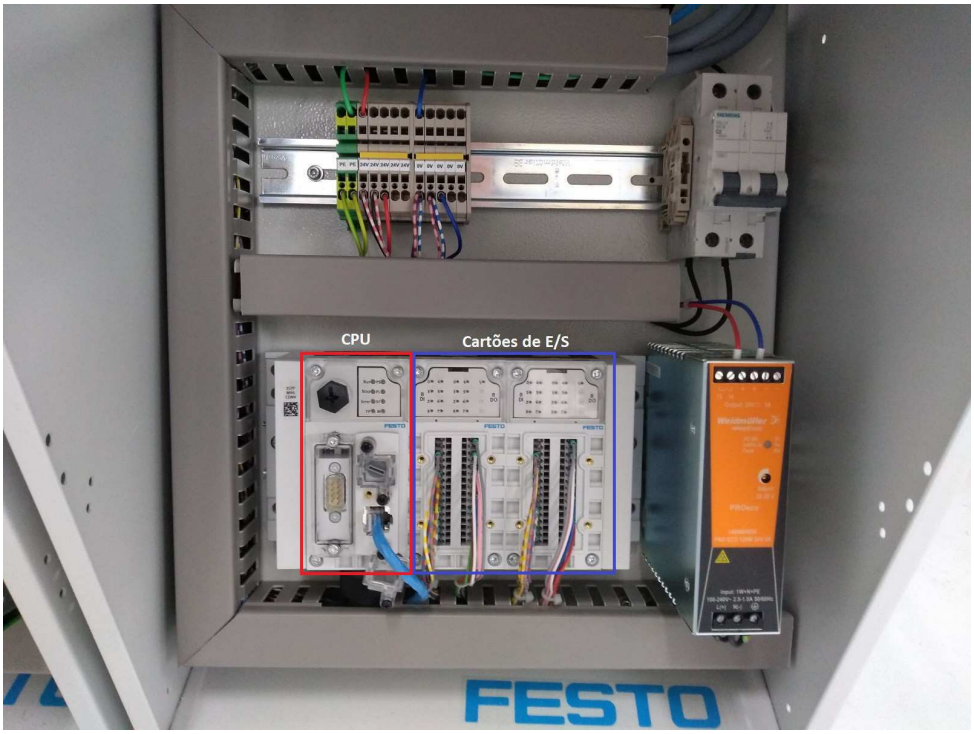
Figura 10 – Peças processadas pelo MPS.

Fonte: Ramos (2019)

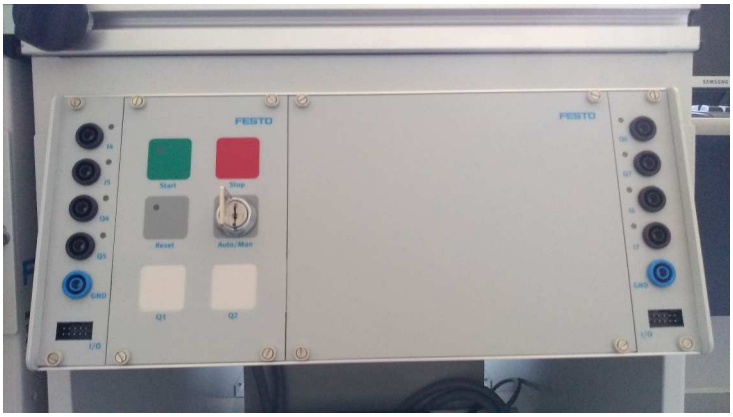
3.1.1.2 Sistema de Controle e Interface de Operação

Cada estação utiliza um controlador lógico programável (CLP) para seu gerenciamento. Além disso, cada estação possui um painel de controle que permite a interação entre o operador e o sistema por meio de botões, lâmpadas e uma interface com o CLP, contendo entradas e saídas (E/S) para comunicação. O CLP utilizado, fabricado pela FESTO, possui uma estrutura modular que inclui a CPU CPX-CEC-C1-V3 e dois cartões de E/S digitais CPX-AB-8-KL-4POL, responsáveis por receber sinais de entrada e enviar sinais de saída para controlar atuadores e sinalizadores. Esta configuração pode ser visualizada na Figura 11, que apresenta o CLP com os módulos e o painel de controle (RAMOS, 2019).

Figura 11 – Componentes do Sistema de Controle e Interface de Operação.



a) CLP da FESTO



b) Painel de uma estação do MPS

Fonte: Ramos (2019)

3.1.2 Estações

3.1.2.1 Estação de Distribuição

A estação de distribuição é o módulo inicial do sistema de manufatura, encarregada de fornecer as peças para as próximas estações. Esta estação é composta por dois subsistemas principais: um depósito, onde as peças são armazenadas, e um atuador rotativo equipado com uma ventosa, que transporta as peças do depósito para a estação subsequente.

3.1.2.2 Estação de separação

A estação de separação é um módulo intermediário, responsável por determinar o trajeto das peças com base em sua posição. Um sensor localizado no início da esteira principal verifica a orientação da peça, gerando um sinal lógico distinto dependendo de o orifício da peça estar voltado para cima ou para baixo.

3.1.2.3 Estação de Classificação

A estação de classificação é o módulo final do MPS, responsável por separar as peças de acordo com seu tipo: preta, vermelha ou metálica. Para isso, são utilizados dois sensores: um sensor óptico difuso, que identifica peças vermelhas, e um sensor indutivo, que detecta peças metálicas. As peças pretas são identificadas quando nenhum dos sensores gera um sinal lógico alto. Dessa maneira, é possível distinguir entre peças vermelhas, metálicas e pretas, permitindo a separação adequada de cada uma.

3.1.3 Integração da Visão Computacional ao Sistema MPS: Motivação e Adaptações

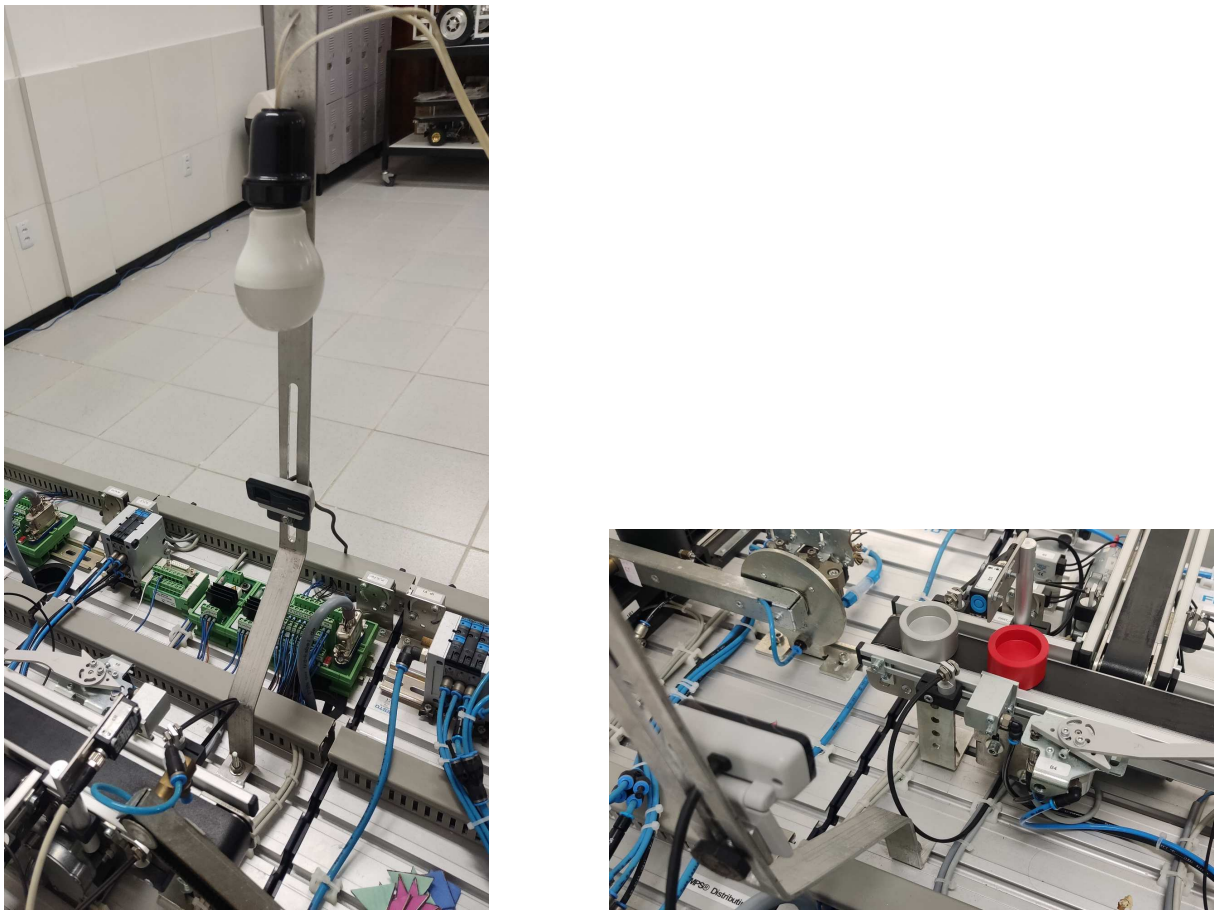
Compreendendo o funcionamento de cada estação individualmente e em conjunto, torna-se possível propor uma nova solução para a classificação das peças. No contexto da automação industrial, a visão computacional emerge como uma alternativa viável e eficiente. Utilizando tecnologias avançadas de captura e análise de imagens, é possível realizar todas as classificações necessárias para o funcionamento adequado do MPS, substituindo os métodos tradicionais baseados em sensores.

A implementação da visão computacional no MPS permite substituir tanto o sensor responsável pela seleção das peças quanto os dois sensores encarregados da classificação. Esta substituição não apenas simplifica o sistema, mas também aumenta a precisão e a eficiência do processo. Além disso, a visão computacional possibilita uma infinidade de novas formas de operação, como a adição de novos tipos de peças sem a necessidade de *hardware* adicional. Esta flexibilidade permite uma adaptação mais rápida a novas exigências de produção e uma maior capacidade de resposta a mudanças nas características das peças, melhorando significativamente a qualidade do controle no sistema de manufatura.

Para incorporar a classificação das peças por meio da visão computacional ao sistema MPS, foram realizadas modificações específicas na estação de separação. Identificada como o primeiro ponto de contato das peças após serem posicionadas pela estação de distribuição, essa estação foi considerada estratégica para a instalação de uma câmera. Assim, um suporte foi fixado na bancada, permitindo posicionar uma *webcam* para capturar imagens em tempo real. Além disso, o suporte inclui um ajuste de altura para o posicionamento da *webcam* e um orifício para acoplar uma luminária na parte superior, proporcionando uma

iluminação adequada da esteira e garantindo uma qualidade de imagem otimizada. Essas adaptações foram cruciais para possibilitar a integração eficaz da visão computacional ao sistema MPS, permitindo a captura e análise precisas das peças em movimento durante o processo de classificação. Como pode ser observado na Figura 12.

Figura 12 – Adaptações Realizadas



a) Suporte com Webcam e Luminária b) Posicionamento do Suporte

Fonte: Elaborado Pelo Autor

3.2 Preparação do Ambiente de Trabalho

Após as adaptações realizadas no MPS, o próximo passo consistiu na instalação dos *softwares* e bibliotecas necessários para o desenvolvimento do projeto. Foi configurado um ambiente de trabalho no *Visual Studio Code* e em seguida foi instalada a extensão do *Jupyter Notebook*. A linguagem de programação *Python* foi selecionada devido à sua ampla gama de bibliotecas de aprendizado de máquina. Especificamente, foi instalada a biblioteca YOLOv8, uma API de alto nível desenvolvida pela *Ultralytics* e que opera sobre o *PyTorch*, para o desenvolvimento e treinamento dos modelos. Além disso, outras bibliotecas, como *OpenCV*, *NumPy* e *Pandas*, foram empregadas para diversas operações

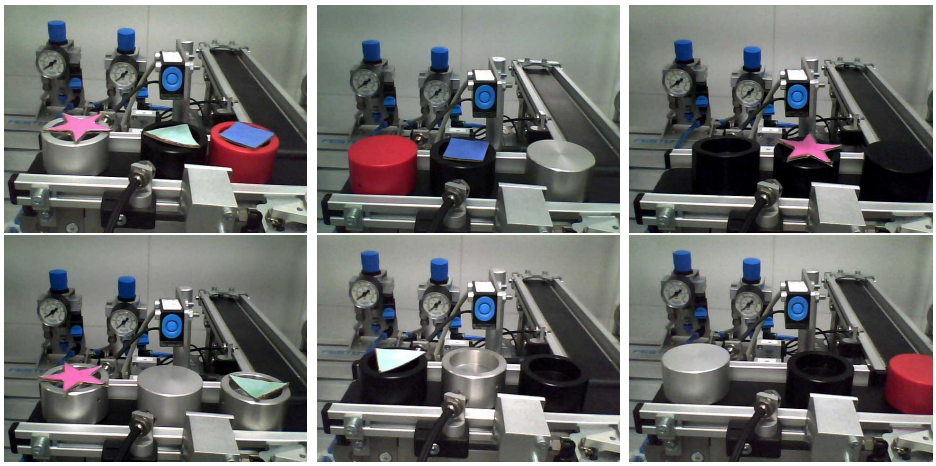
e manipulações de dados.

3.3 Base de Dados

Após a instalação e configuração do ambiente de trabalho, tornou-se necessário criar uma base de dados personalizada, considerando que as peças do MPS são muito específicas e não há material disponível para *download*. A base de dados criada contém 526 imagens das peças em diferentes posições, todas com resolução de 640 x 480 x 3.

Após a construção da base de dados, o próximo passo foi realizar a rotulagem (*labeling*). Este processo envolve a marcação manual das "*bounding boxes*" nas imagens, identificando os objetos de interesse dentro de cada imagem para possibilitar o treinamento do modelo. No contexto deste trabalho, os objetos de interesse são as peças do MPS. Adicionalmente, foram incluídas três formas geométricas (triângulo, quadrado e estrela) posicionadas sobre as peças. Essas formas foram adicionadas para aumentar a complexidade do sistema e testar a eficiência do algoritmo, conforme ilustrado na Figura 13.

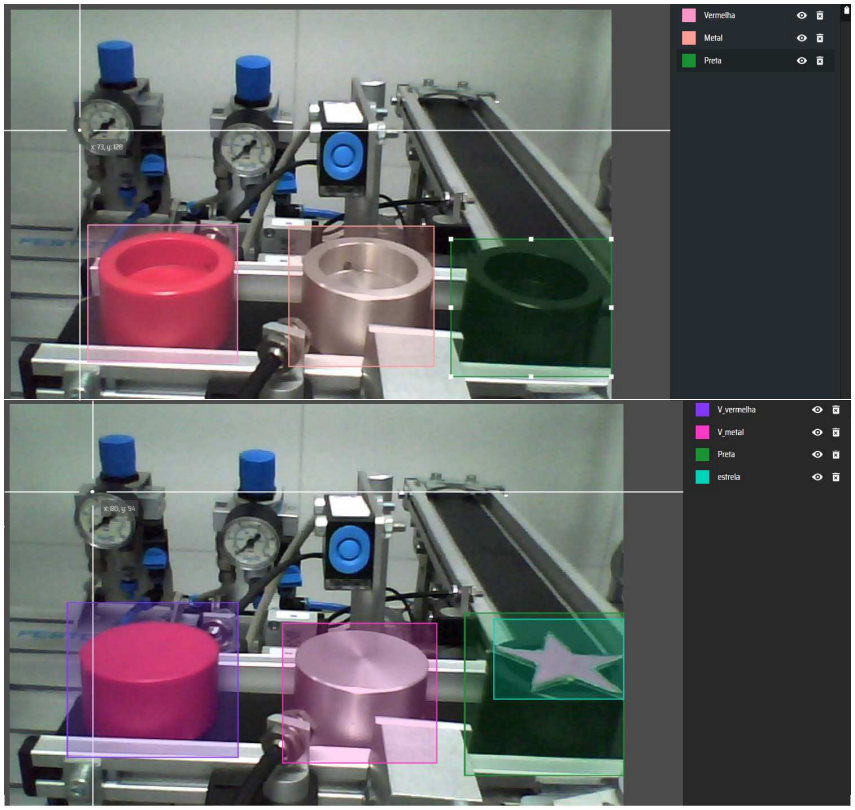
Figura 13 – Imagens Extraídas da Base de Dados



Fonte: Elaborado Pelo Autor

Para realizar a rotulagem, utilizou-se o *Make Sense*, uma ferramenta *online* que facilita a rotulagem de bases de dados e é compatível com diversos algoritmos de visão computacional, incluindo o YOLO. Após abrir a ferramenta, foi necessário fazer o *upload* das imagens e criar os *labels* (classes) que seriam treinadas pelo algoritmo. Para este trabalho, foram criados nove *labels*: Vermelha, Preta, Metal, V_Vermelho, V_Metal, V_Preta, Triângulo, Estrela e Quadrado. Os *labels* que iniciam com "V_" representam as peças do MPS com o orifício voltado para baixo. A rotulagem realizada com a ferramenta pode ser vista na Figura 14.

Figura 14 – Ferramenta de Rotulagem Make Sence



Fonte: Elaborado Pelo Autor

A base de dados foi segmentada em três partes: treino, validação e teste. Essa divisão é fundamental para assegurar que a rede neural desenvolva a capacidade de generalização necessária para classificar corretamente novos dados, em vez de apenas memorizar os padrões específicos do conjunto de treino. Tendo em vista que o problema de *overfitting* pode comprometer a capacidade do modelo de performar bem em dados não vistos. Portanto, a separação em treino, validação e teste permite avaliar e ajustar a rede neural de forma eficaz, garantindo que ela aprenda padrões gerais das classes e não apenas os detalhes específicos dos exemplos de treinamento.

A base de dados foi dividida em três conjuntos: treinamento, validação e teste, seguindo a proporção de 70% para treinamento, 23% para validação e 7% para teste. A escolha dessa proporção foi determinada pelo autor, uma vez que não há um padrão universal de divisão. A proporção varia conforme a complexidade do problema e o tamanho da base de dados em questão.

A amostra de imagens destinada ao treinamento é utilizada para que o algoritmo possa aprender e ajustar seus parâmetros com base nos dados fornecidos. A amostra de validação é empregada para avaliar continuamente o desempenho do algoritmo ao longo do processo de treinamento, garantindo que os padrões aprendidos sejam generalizáveis e não específicos apenas ao conjunto de treino, evitando assim o *overfitting*. Após o término

do treinamento, realiza-se uma verificação final utilizando o conjunto de teste, composto por exemplos inéditos para o algoritmo. Esta etapa final é crucial para assegurar que os padrões reconhecidos pela rede sejam aplicáveis de maneira geral e não limitados aos conjuntos de treino e validação.

3.4 Modelo

Após a criação da base de dados e a realização da rotulagem, foram baixados modelos pré-treinados do YOLOv8, disponíveis na documentação do YOLOv8. Esses modelos foram ajustados para detectar as peças do MPS, os quais são baseados no conjunto de dados COCO (*Common Objects in Context*), um recurso fundamental na pesquisa de visão computacional desenvolvido pela *Microsoft Research*.

O COCO contém mais de 330.000 imagens, incluindo 80 categorias de objetos com anotações detalhadas, como caixas delimitadoras, máscaras de segmentação e pontos-chave. As imagens do COCO refletem cenários realistas e complexos, com objetos sobrepostos e em diversas poses, o que torna a detecção e segmentação desafiadoras. O COCO é utilizado para várias tarefas, como detecção de objetos, segmentação de imagens e estimativa de poses, sendo amplamente empregado para treinar e avaliar modelos de redes neurais convolucionais. Além disso, serve como um *benchmark* padrão em competições e pesquisas, contribuindo significativamente para avanços em algoritmos de visão computacional e aplicações em robótica e sistemas autônomos (ARAUJO, 2020). Essa versão do YOLO conta com cinco modelos que podem ser vistos na Tabela 1.

Tabela 1 – Desempenho dos Modelos YOLOv8

Modelo	**Tamanho (pixéis)**	**mAPval 50-95**	**Velocidade CPU ONNX (ms)**	**Velocidade A100 TensorRT (ms)**	**Params (M)**	**FLOPs (B)**
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Fonte: Documentação do YOLOv8.

Para a avaliação de modelos de aprendizado profundo, especialmente aqueles voltados para tarefas de visão computacional como a detecção de objetos, utilizam-se diversas métricas. A seguir, detalhamos as principais métricas utilizadas:

- **mAPval 50-95:** A *Mean Average Precision* (mAP) é uma métrica padrão para avaliar a precisão de modelos de detecção de objetos. O valor de mAP é calculado considerando a média das precisões em vários limiares de *Intersection over Union* (IoU), que variam de 0,50 a 0,95. Esta métrica fornece uma média da precisão do

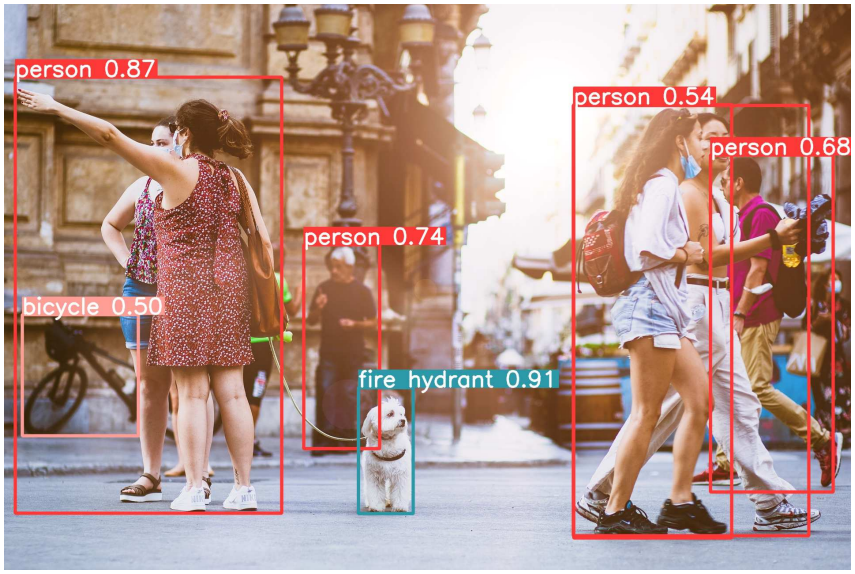
modelo, levando em conta diferentes graus de sobreposição entre as predições do modelo e as caixas delimitadoras reais (*ground truth*).

- **Velocidade CPU ONNX (ms):** Esta métrica indica a velocidade de inferência do modelo quando convertido para o formato ONNX (*Open Neural Network Exchange*) e executado em uma CPU. A unidade de medida é milissegundos (ms), representando o tempo médio necessário para o modelo processar uma imagem ou um conjunto de dados de entrada. Esta métrica é essencial para avaliar a eficiência do modelo em ambientes onde os recursos computacionais são limitados.
- **Velocidade A100 TensorRT (ms):** Esta métrica mede a velocidade de inferência do modelo quando otimizado com TensorRT (um otimizador de inferência de alta performance desenvolvido pela NVIDIA) e executado em uma GPU NVIDIA A100. Também medido em ms, esta métrica reflete o tempo necessário para o modelo processar uma imagem ou um conjunto de dados de entrada, sendo crucial para aplicações que exigem alta performance e baixa latência.
- **Params (M):** Esta métrica refere-se ao número de parâmetros treináveis no modelo, medido em milhões (M). Os parâmetros incluem pesos e vieses que são ajustados durante o treinamento do modelo. Esta métrica é uma indicação direta da capacidade e da complexidade do modelo. Modelos com um número maior de parâmetros tendem a ter maior capacidade de representação, mas também exigem mais dados e poder computacional para o treinamento e a inferência.
- **FLOPs (B) (*Floating Point Operations*):** FLOPs representa o número de operações de ponto flutuante que o modelo realiza para processar uma entrada. Medido em bilhões (B), esta métrica indica a complexidade computacional do modelo. Um número maior de FLOPs geralmente implica em maior consumo de recursos computacionais, mas também pode estar associado a uma maior capacidade de processamento de características complexas nos dados de entrada.

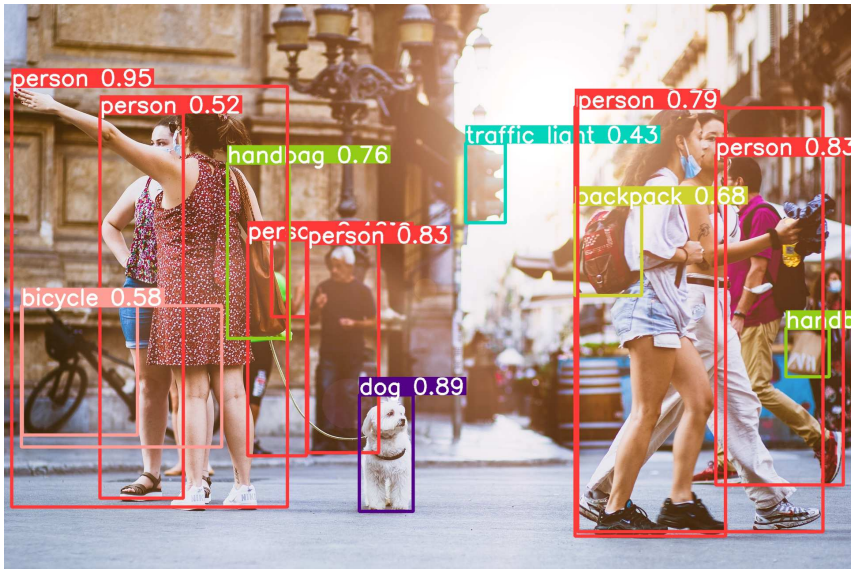
Na seleção dos modelos de detecção, é crucial considerar os requisitos específicos do sistema em questão, como podemos ver na Tabela 1, conforme a precisão dos modelos vão aumentando o tempo de processamento também aumenta. Se a prioridade for a velocidade de processamento e a precisão puder ser ligeiramente comprometida, os modelos mais simples, como YOLOv8n ou YOLOv8s, são preferíveis. Por outro lado, se a precisão na detecção de objetos for fundamental e a velocidade não for um problema, o modelo mais robusto, YOLOv8x, seria mais indicado. No entanto, se o objetivo for alcançar um equilíbrio entre precisão e velocidade, o modelo intermediário YOLOv8m seria a escolha mais apropriada. Portanto, a seleção do modelo adequado depende da ponderação cuidadosa desses requisitos específicos para atender às necessidades do sistema de forma eficiente e eficaz.

Para comparar o desempenho entre o modelo mais simples e o modelo mais robusto, foi realizado um teste de detecção em uma pasta que contém cinco imagens. Na Figura 15, podem ser visto a previsão dos modelos em uma imagem.

Figura 15 – Previsões Feitas Pelo Algoritimo.



a) Yolov8n



b) Yolov8x

Fonte: Elaborado Pelo Autor

Como pode ser observado, o YOLOv8x foi superior em relação ao YOLOv8n. Além de conseguir detectar mais objetos, ele também fez todas as previsões corretas, ao contrário do modelo mais simples que errou na detecção do cachorro. No entanto, o modelo mais simples levou 5,25 segundos para fazer todas as detecções em todas as imagens da pasta, enquanto o YOLOv8x levou 8,27 segundos.

O uso de modelos pré-treinados é uma prática comum na resolução de problemas com redes neurais e está associado a uma técnica chamada *Transfer Learning* (ou transferência de aprendizado). Essa técnica aproveita o conhecimento adquirido por uma rede neural ao resolver um problema e o aplica para resolver um problema semelhante. Modelos pré-treinados estão disponíveis em bibliotecas de *Deep Learning*, como *TensorFlow*, *Keras* e YOLO. A transferência de aprendizado não só acelera o treinamento da rede, mas também ajuda a evitar o *overfitting* (ou superajuste) (NETO, 2019).

Dentro do escopo deste projeto, a escolha do modelo pré-treinado foi uma etapa crucial. Inicialmente, optamos pelo YOLOv8n devido à sua velocidade de processamento e desempenho em cenários menos complexos, como o do MPS. Este modelo foi considerado adequado devido à relativa simplicidade das peças a serem detectadas, somando-se ao fato de que o MPS envolve apenas nove tipos de classes. No entanto, sabe-se que a escolha do modelo ideal nem sempre é uma decisão imediata. Caso o modelo YOLOv8n não atenda às nossas expectativas em termos de desempenho e precisão, estamos preparados para realizar ajustes e implementar melhorias no modelo seguinte, até alcançarmos a solução mais adequada para as necessidades específicas do sistema.

3.5 Treinamento

Uma vez definida a base de dados a ser utilizada, o modelo de rede neural e a arquitetura devidamente configurada para a tarefa de detectar as peças do MPS, o próximo passo foi o treinamento dos modelos. Após o treinamento, a detecção foi testada em imagens específicas do conjunto de teste e em um vídeo.

4 Resultados Obtidos

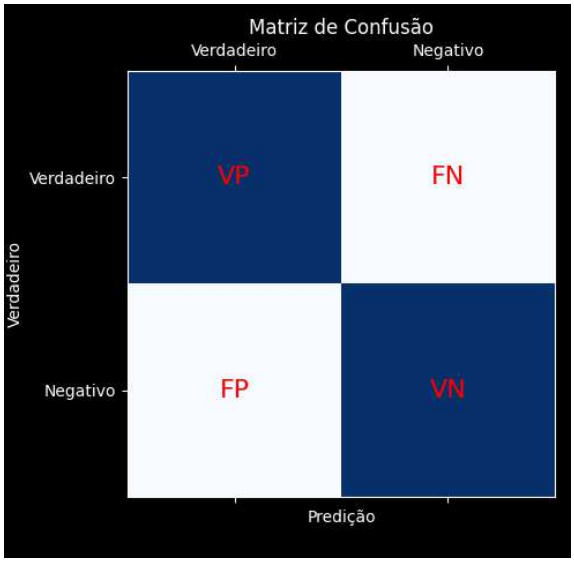
Esta seção aborda diversas métricas de desempenho empregadas para avaliar modelos de redes neurais. Nas subseções seguintes, apresentamos os resultados de desempenho do algoritmo de detecção com base nessas métricas.

Em problemas de classificação, para avaliar as previsões da rede, contamos a quantidade de acertos, quando a classe prevista corresponde ao gabarito, e a quantidade de erros, quando a previsão da classe não coincide com o gabarito. Contudo, para uma análise mais detalhada do desempenho de um algoritmo de classificação, é frequentemente utilizada a Matriz de Confusão. Esta matriz oferece quatro métricas distintas e específicas:

- Verdadeiros Positivos (VP) - Números de exemplos da classe positiva classificados corretamente;
- Verdadeiros Negativos (VN) - Números de exemplos da classe negativa classificados corretamente;
- Falsos Positivos (FP) - Números de exemplos da classe negativa classificados incorretamente;
- Falsos Negativos (FN) - Números de exemplos da classe positiva classificados incorretamente.

A Figura 16 ilustra uma matriz de confusão.

Figura 16 – Matriz de Confusão



Fonte: Elaborado Pelo Autor.

Essa representação possibilita uma análise aprofundada do desempenho do classificador em cada classe, oferecendo informações essenciais sobre sua precisão, *recall* e habilidade de distinguir entre diferentes classes. Dessa forma, a partir da matriz de confusão, é possível derivar outras métricas de desempenho (PAGANO et al., 2023), tais como:

- **Acurácia**: Indica a porcentagem de previsões corretas em comparação com o total de previsões feitas, sendo calculada como a soma dos Verdadeiros Positivos e Verdadeiros Negativos dividida pelo número total de amostras.

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.1}$$

  É uma métrica simples e fácil de interpretar, mas em casos onde o conjunto de dados de validação está desbalanceado, a acurácia pode se tornar tendenciosa.

- **Precisão**: Refere-se à proporção de exemplos classificados corretamente como positivos, em relação ao total de exemplos preditos como positivos, indicando sua precisão.

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{4.2}$$

- ***Recall***: Conhecido como sensibilidade, indica a completude do modelo, representando a proporção de exemplos positivos corretamente identificados pelo modelo.

$$Recall = \frac{VP}{VP + FN} \tag{4.3}$$

- ***F1-Score***: Esta métrica combina precisão e recall em um único valor, proporcionando uma média harmônica entre essas duas medidas. Por isso, oferece uma avaliação mais abrangente do modelo, especialmente útil em situações com desequilíbrio considerável entre as classes.

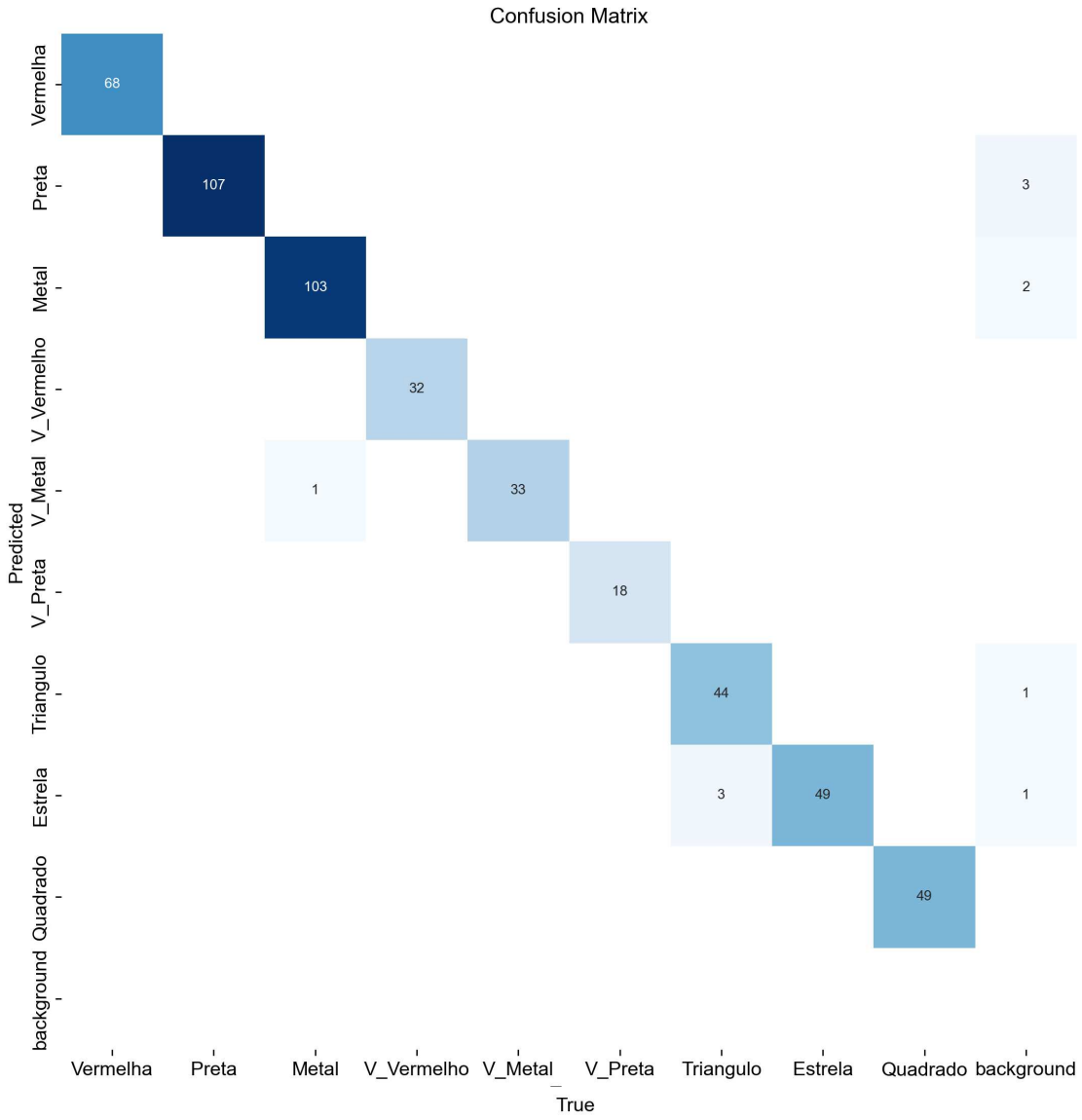
$$\text{F1-Score} = 2 \times \frac{\text{Precisão} \times \text{Revocação}}{\text{Precisão} + \text{Revocação}} \tag{4.4}$$

4.1 Resultados do Modelo

Com base nas métricas de desempenho apresentadas na Seção Resultados Obtidos, é apresentada nesta seção a avaliação do modelo empregado para identificar as peças do MPS.

Logo após a realização das predições do modelos de treinamento, é possível inferir a matriz de confusão. O modelo pode ser visto na Figura 17. Dessa forma, são concebíveis algumas informações importantes analisando a matriz. É importante esclarecer que as métricas que serão avaliadas são do modelo como um todo, e não em cada uma de suas categorias.

Figura 17 – Matriz de Confusão do Modelo



Fonte: Elaborado Pelo Autor.

A diagonal principal, destacada em azul na matriz de confusão, indica os verdadeiros positivos, onde as predições correspondem aos valores reais, resultando em um total de 503 casos corretamente identificados como positivos.

Os valores fora da diagonal principal são os verdadeiros negativos, indicando que o modelo detectou uma peça, porém classificou erroneamente sua categoria, totalizando 4 casos corretamente identificados como negativos.

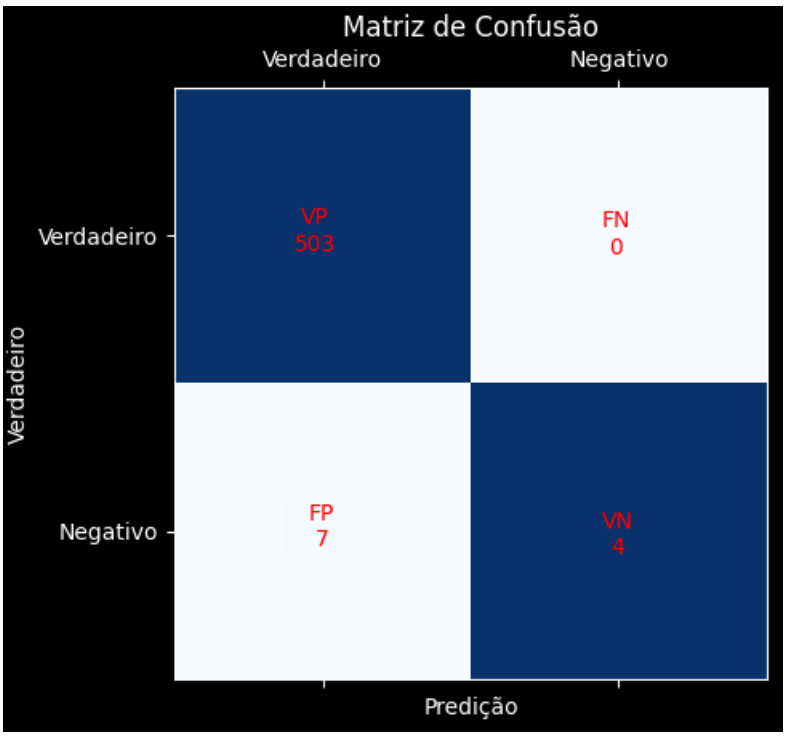
Na coluna rotulada como "*background*" no conjunto de valores verdadeiros, encontramos os falsos positivos, onde o modelo identifica algo na imagem mesmo quando não há nenhum objeto presente, totalizando 7 casos incorretamente classificados como positivos.

Por fim, na linha rotulada como "*background*" no conjunto de previsões, estão os

falsos negativos, representando os casos em que o modelo falha em detectar uma peça que está presente na imagem, totalizando 0 casos erroneamente classificados como negativos.

Após essa análise, podemos destacar que a detecção das peças se enquadra como um problema de classificação binária, no qual cada imagem de entrada é atribuída a uma de duas classes. Dessa forma, podemos representar a matriz de confusão conforme destacado na Figura 18.

Figura 18 – Matriz de Confusão: Resultado do Modelo



Fonte: Elaborado Pelo Autor.

A partir da matriz de confusão gerada, é possível calcular as métricas de desempenho importantes, mencionadas anteriormente. Essas métricas foram calculadas e dispostas na tabela 2.

Tabela 2 – Métricas de avaliação do modelo

Métrica	**Valor**
Precisão	0.9863
Recall	1
F1-Score	0.9931
Acurácia	0.9864

Fonte: Elaborado Pelo Autor.

4.2 Análise das Métricas

Os resultados obtidos a partir das métricas de desempenho indicam que o modelo desenvolvido apresenta uma performance excepcional na tarefa de detecção. A análise detalhada dessas métricas permitirá uma compreensão mais profunda do comportamento do modelo, destacando suas forças e possíveis áreas de melhoria. A seguir, serão discutidos os resultados de precisão, *recall*, *F1-Score* e acurácia para fornecer uma visão abrangente da eficácia do modelo.

- **Precisão**: Uma precisão de 0.9863 indica que 98,63% das predições feitas pelo modelo foram corretas. Isso sugere que o modelo tem uma baixa taxa de falsos positivos, o que é crucial em aplicações onde a penalidade por detectar incorretamente é alta.
- ***Recall***: Um *recall* de 1.0 significa que o modelo foi capaz de identificar todos os exemplos reais da classe positiva. Isso indica que o modelo não deixou escapar nenhum exemplo positivo, o que é essencial em situações onde a penalidade por não detectar é alta.
- ***F1-Score***: Um *F1-Score* de 0.9931 sugere que o modelo tem um excelente balanço entre precisão e *recall*. Isso significa que ele é eficaz tanto em identificar exemplos positivos quanto em minimizar predições incorretas.
- **Acurácia**: Uma acurácia de 0.9864 indica que 98,64% de todas as predições feitas pelo modelo estavam corretas. Isso sugere que o modelo é geralmente confiável para a tarefa de detecção.

Os resultados obtidos sugerem que o modelo de detecção está funcionando com um alto nível de eficiência e precisão, sendo capaz de identificar corretamente todos os exemplos de interesse enquanto minimiza erros. Isso indica que o modelo é bem ajustado para a tarefa específica e pode ser confiável para uso em produção, embora testes adicionais em cenários variados e com dados não vistos devam ser realizados para validar ainda mais sua robustez e generalização.

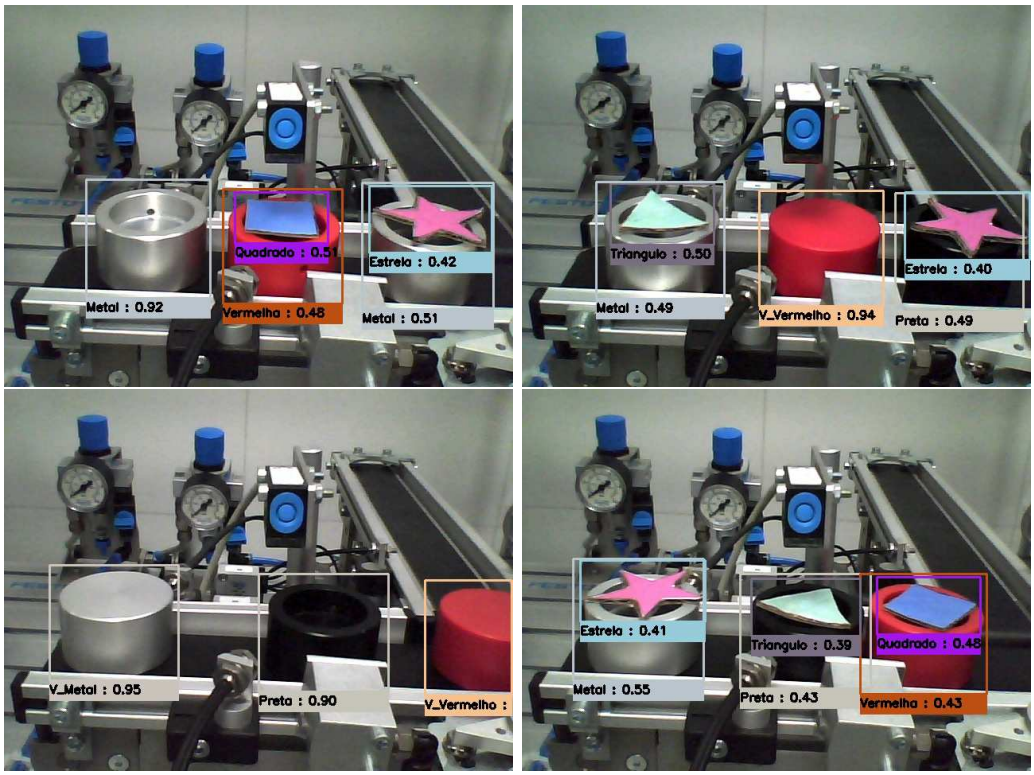
4.3 Processamento em Tempo Real

Para avaliar se o modelo é viável para aplicações em tempo real, foi conduzida a classificação de um vídeo. Este vídeo tem uma duração de 83 segundos, com uma taxa média de 30 quadros por segundo, tendo exatamente 2510 *frames*. O *hardware* utilizado para realizar os testes consiste em um processador Ryzen 5, uma placa de vídeo Nvidia RTX 3060 com 12GB de memória, e 16GB de RAM. Foram realizados dois testes distintos: o primeiro utilizando apenas o poder de processamento da CPU para a execução do modelo,

enquanto no segundo teste será empregada a aceleração por GPU. Estes experimentos visam determinar a capacidade do modelo em lidar com uma carga de trabalho realista, fornecendo *insights* valiosos sobre seu desempenho em cenários práticos.

No primeiro teste, que empregou exclusivamente a CPU, o tempo de processamento foi de 162.36 segundos, resultando em uma taxa de processamento de 15.5 *frames* por segundo (fps). Por outro lado, no segundo teste, que envolveu a aceleração por GPU, o processamento foi significativamente mais rápido, concluído em 40.20 segundos, o que representa uma taxa de 62.4 fps. Esses resultados destacam a diferença substancial na eficiência de processamento entre os dois métodos, demonstrando claramente os benefícios da utilização da GPU para acelerar o processamento de dados. Podemos observas algumas das previsões do modelo na Figura 19.

Figura 19 – Previsões Realizadas no MPS.



Fonte: Elaborado Pelo Autor.

Com os resultados obtidos, podemos avaliar tanto a eficiência do modelo na detecção das peças quanto seu desempenho em tempo real. Inicialmente, ao utilizar apenas a CPU, o modelo alcançou uma média de 15,5 fps. Essa taxa é adequada para operação em tempo real em sistemas onde a velocidade de processamento não é a principal exigência, considerando as limitações do *hardware* disponível. Essa capacidade de funcionar em tempo real, mesmo com processamento limitado, demonstra a robustez do modelo para aplicações práticas.

Quando impulsionado pela GPU, o desempenho do modelo foi significativamente melhorado, atingindo uma média de 62,4 fps. Essa alta taxa de processamento é ideal para sistemas que requerem respostas rápidas e precisas, como em aplicações de monitoramento ou robótica. Esses resultados indicam que o modelo é altamente adaptável, capaz de operar eficientemente tanto em ambientes com recursos limitados quanto em sistemas de alta performance, atendendo a uma ampla gama de requisitos operacionais e demonstrando sua versatilidade e eficácia em diferentes contextos de aplicação.

5 Conclusões

Este trabalho teve como objetivo a implementação de um algoritmo de visão computacional com intuito de identificar e reconhecer as peças do MPS. O desenvolvimento deste algoritmo envolveu a aplicação de conhecimentos adquiridos ao longo da graduação em Engenharia Elétrica, especialmente nas disciplinas de Técnicas de Programação, Sistemas de Automação Industrial e Instrumentação Eletrônica. A criação e otimização do modelo também se basearam em conceitos avançados de Visão Computacional, com ênfase nas Redes Neurais Convolucionais e na arquitetura YOLO. Esses conhecimentos foram essenciais para desenvolver um sistema eficiente e preciso, capaz de operar em ambientes industriais reais, demonstrando a integração prática da teoria aprendida durante o curso com as necessidades específicas do projeto.

Para a identificação e reconhecimento das peças, o algoritmo implementado mostrou-se eficaz na detecção e reconhecimento da maioria dos itens. Contudo, alguns erros de reconhecimento foram observados durante os testes. Após análise dos resultados, identificou-se que uma possível causa desses erros ocorre quando a peça está saindo do enquadramento da câmera, o que impede o algoritmo de distinguir corretamente a parte restante da peça. Além disso, o tamanho do *dataset* utilizado foi considerado pequeno para uma rede neural convolucional profunda, o que pode ter impactado a performance do modelo.

Para mitigar esses problemas, uma solução viável seria aumentar a base de dados, realizando um pré-processamento dos dados para garantir maior diversidade e qualidade. Adicionalmente, delimitar uma região de operação na qual o algoritmo opere eficientemente evitaria a detecção de peças que estejam parcialmente fora do enquadramento. Apesar dos desafios, os resultados obtidos na avaliação do modelo foram bastante satisfatórios, com uma precisão de 98,63% e um *recall* de 100%, demonstrando que o algoritmo possui um alto potencial de aplicação prática na detecção e reconhecimento de peças em ambientes industriais.

5.1 Perspectivas para o Futuro

Considerando os desafios encontrados neste trabalho, existem algumas perspectivas para a continuidade e melhorias do sistema, tais como:

- Expansão do banco de dados;
- Seleção de outras arquiteturas pré-treinadas ou algoritmos para comparar a eficiência com o YOLO;

- Implementação de protocolos de comunicação com CLP, como Modbus ou OPC, para integrar a automação do MPS utilizando a visão computacional.

Referências Bibliográficas

ALVES, G. *Entendendo Redes Convolucionais (CNNs)*. 2018. Acessado em 12 de maio de 2024. Disponível em: <https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184>. Citado na página 10.

Analytics Vidhya. *AI VS ML VS DL. Let's Understand The Difference*. 2021. Acesso em: 02 Maio 2024. Disponível em: <https://www.analyticsvidhya.com/blog/2021/07/ai-vs-ml-vs-dl-lets-understand-the-difference/>. Citado na página 7.

ARAUJO, J. G. *Precisa de imagens anotadas para treinar sua IA no Python? Use o COCO Dataset!* 2020. Disponível em: <https://medium.com/porto-seguro/precisa-de-imagens-para-treinar-sua-ia-use-a-api-do-coco-dataset-pycocotools-a87c7e21774b>. Citado na página 25.

BRIGADE, D. S. *A diferença entre inteligência artificial, machine learning e deep learning*. 2016. URL. Disponível em: <https://medium.com/data-science-brigade/a-diferen%C3%A7a-entreintelig%C3%AAncia-artificial-machine-learning-e-deep-learning-930b5cc2aa42/>. Citado na página 9.

DENG, L.; YU, D. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, v. 7, n. 3–4, p. 197–387, jun. 2014. Citado na página 8.

GONZALEZ, R. C.; WOODS, R. E.; EDDINS, S. L. *Digital Image Processing using MATLAB*. [S.l.]: Pearson, 2006. Citado na página 4.

GéRON, A. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, CA: O'Reilly Media, 2017. Citado na página 5.

HAYKIN, S. S. *Neural Networks and Learning Machines*. 3rd. ed. New York: Prentice Hall, 2009. Citado 2 vezes nas páginas 4 e 5.

HUANG, R.; PEDOEEM, J.; CHEN, C. YOLO-LITE: a real-time object detection algorithm optimized for non-gpu computers. In: *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.]: IEEE, 2018. p. 2503–2510. Citado na página 1.

HUSSAIN, M. YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, v. 11, n. 7, p. 677, 2023. Citado na página 1.

KOCAKANAT, K.; SERIF, T. Turkish traffic sign recognition: Comparison of training step numbers and lighting conditions. *European Journal of Science and Technology*, 11 2021. Citado na página 12.

LI, E. Y. *Dive Really Deep into YOLO v3: A Beginner's Guide*. 2019. Disponível em: <https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e>. Citado 2 vezes nas páginas 15 e 16.

MANTRIPRAGADA, M. *Digging deep into YOLO V3 - A hands-on guide Part 1*. 2020. Disponível em: <https://towardsdatascience.com/digging-deep-into-yolo-v3-a-hands-on-guide-part-1-78681f2c7e29>. Citado na página 12.

MARTINS, V. E. *Aplicação de Deep Learning para Detecção de Veias em Carne Suína*. Dissertação (Trabalho de conclusão de curso) — Universidade Estadual de Londrina, Londrina, 2018. Citado 2 vezes nas páginas 8 e 9.

MITCHELL, T. M. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN 978-0-07-042807-2. Citado na página 4.

NETO, R. P. da S. *Metodologia automática para classificação de lesões de mama utilizando Transfer Learning*. 2019. Trabalho de Conclusão de Curso. Disponível em: <https://ufpi.br/arquivos_download/arquivos/PICOS/Not%C3%ADcias/PICOS_2022/Biblioteca/2019/Sistemas_de_Informa%C3%A7%C3%A3o_2019/Rafael_Pedro_da_Silva_Neto.pdf>. Citado 2 vezes nas páginas 6 e 28.

NEVES, D. *Revolução Industrial*. 2024. Acesso em: 05 de março de 2024. Disponível em: <https://mundoeducacao.uol.com.br/historiageral/revolucao-industrial-2.htm>. Citado na página 1.

PAGANO, T. P. et al. Bias and unfairness in machine learning models: A systematic review on datasets, tools, fairness metrics, and identification and mitigation methods. *Big Data and Cognitive Computing*, v. 7, n. 1, 2023. ISSN 2504-2289. Disponível em: <https://www.mdpi.com/2504-2289/7/1/15>. Citado na página 30.

RAMOS, E. T. G. *Relatório de estágio supervisionado*. Campina Grande, Paraíba, Brasil, 2019. Curso de Bacharelado em Engenharia Elétrica, Centro de Engenharia Elétrica e Informática. Disponível em: <http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/20730>. Citado 4 vezes nas páginas 2, 18, 19 e 20.

REDMON, J.; FARHADI, A. Yolo9000: Better, faster, stronger. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2017. p. 7263–7271. Citado na página 14.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv*, 2018. Citado 2 vezes nas páginas 13 e 16.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 779–788. Citado 3 vezes nas páginas 12, 13 e 14.

SZELISKI, R. *Computer Vision: Algorithms and Applications*. [S.l.]: Springer Nature, 2022. Citado na página 1.

TERVEN, J.; CóRDOVA-ESPARZA, D.-M.; ROMERO-GONZáLEZ, J.-A. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, v. 5, n. 4, p. 1680–1716, 2023. ISSN 2504-4990. Disponível em: <https://www.mdpi.com/2504-4990/5/4/83>. Citado 2 vezes nas páginas 14 e 15.

VISãO Computacional – O Que é Visão Computacional? Acesso: 28 de Abril 2024. Disponível em: <https://blog.dsacademy.com.br/o-que-e-visao_computacional/>. Citado na página 3.