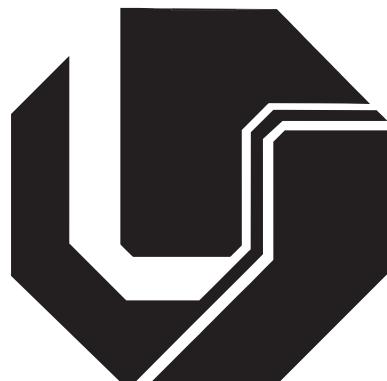


UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



**Inserção Automática de Componentes
em Ambientes Virtuais de Treinamento para Substações
de Energia utilizando Inteligência Artificial**

Leandro Sena Zuza

Uberlândia

2024

Leandro Sena Zuza

Inserção Automática de Componentes em Ambientes Virtuais de Treinamento para Substações de Energia utilizando Inteligência Artificial

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para obtenção do Título de Mestre em Ciências.

Outubro X, 2024.

Membros da Banca:

Prof. Alexandre Cardoso, Dr. (Orientador - UFU)

Prof. Daniel S. Caetano, Dr. (Coorientador - UFU)

Prof. Marcio, Dr. (Membro Interno - UFU)

Profa. Valéria Farinazzo Martins, Dra. (Membro Externo - Mackenzie)

Agradecimentos

Agradeço, antes de tudo, ao Grupo de Pesquisa em Realidade Virtual e Aumentada, mais conhecido como GRVA, por ser o subsídio de todo este trabalho e por ter me permitido conhecer pessoas fantásticas que viabilizaram sua confecção!

Ao meu orientador, Alexandre Cardoso, pela oportunidade de fazer parte de seu laboratório, por ser o substrato de tudo o que foi desenvolvido aqui e por todas as suas diretrizes e trocas de conhecimento!

À meu coorientador, Daniel Stefany, pelo planejamento conjunto deste texto e de seus frutos, sempre em busca do primor. Agradeço pela paciência em todas as nossas reuniões e pela parceria inestimável que desenvolvemos até aqui!

À minha mãe, Marilda Sena Pereira Zuza, que sempre me instou a estudar e me aperfeiçoar, desde minha mais tenra idade. Foi e será sempre minha inspiração!

Ao meu pai, Paulo da Silva Zuza, e aos meus irmãos, Lucas Sena Zuza e Guilherme Sena Zuza, por serem parte inerente a mim, e por consequência inerente ao meu desenvolvimento!

À Milena Bugoni, por estar ao meu lado desde o primeiro dia de mestrado e por seu esforço empenhado em me incentivar a concluir, o quanto antes, o meu mestrado!

“Então considerei que as botas apertadas são uma das maiores venturas da Terra,
porque, fazendo doer os pés, dão azo ao prazer de as descalçar.”
(Machado de Assis, em Memórias Póstumas de Brás Cubas)

Abstract

Training with Virtual Reality (VR) has gained prominence in corporate and industrial environments. Conducting educational maneuvers in virtual settings reduces the risk that inexperienced employees face in hostile and high-risk situations, such as those found in power substations. However, modeling these environments requires specialized labor and consumes countless hours of development. In light of this, this dissertation proposes a prototype to automate the creation of these Virtual Environments using YOLOv8, a Convolutional Neural Network (CNN). This work is pioneering and focuses on the identification and insertion of Air Core Reactors, essential equipment in power substations. To achieve this, images captured by Unmanned Aerial Vehicles (UAVs) in two power substations were used, selecting those in which the equipment was present. After this selection, the images were subjected to training using versions from YOLOv5 to YOLOv8, with methodical variations of parameters to identify the configuration that provided the best accuracy. The training results indicated that YOLOv8, compared to its previous versions, demonstrated superiority in terms of accuracy, particularly when using a batch size of 16 and the SGD optimizer. With the recognition model trained with this configuration, an application integrated into Unity software was developed, capable of receiving untrained photos, identifying the number of reactors present in the image, and retrieving their corresponding modeled objects, inserting them into the VR scene.

Keywords

Virtual Training Environments; Optimization; Power Substation; UAV; YOLOv8

Resumo

O treinamento com Realidade Virtual (RV) tem ganhado destaque nos ambientes corporativos e industriais. A realização de manobras educativas em ambientes virtuais reduz o risco que colaboradores inexperientes enfrentam em situações hostis e de alta periculosidade, como as encontradas em subestações de energia. Contudo, a modelagem desses ambientes exige mão de obra especializada e consome inúmeras horas de desenvolvimento. Diante disso, esta dissertação propõe um protótipo para automatizar a criação desses Ambientes Virtuais, utilizando a YOLOv8, uma Rede Neural Convolucional (RNC). Este trabalho é pioneiro e foca na identificação e inserção de Reatores de Núcleo de Ar, equipamentos essenciais nas subestações de energia. Para tanto, foram utilizadas imagens capturadas por Veículos Aéreos Não Tripulados (VANTs) em duas subestações de energia, selecionando aquelas em que o equipamento estava presente. Após essa seleção, as imagens foram submetidas ao treinamento das versões da YOLOv5 à YOLOv8, com variações metodológicas de parâmetros para identificar a configuração que proporcionasse a melhor precisão. Os resultados do treinamento indicaram que a YOLOv8, em comparação com suas versões anteriores, demonstrou superioridade em termos de precisão, especialmente ao utilizar um tamanho de *batch* igual a 16 e o otimizador SGD. Com o modelo de reconhecimento treinado com essa configuração, foi desenvolvida uma aplicação integrada ao software Unity, capaz de receber fotos não treinadas, identificar a quantidade de reatores presentes na imagem e buscar seus objetos modelados correspondentes, inserindo-os na cena de RV.

Palavras Chave

Ambientes Virtuais de Treinamento; Otimização; Subestação de Energia; VANTs; YOLOv8

Publicações

As publicações relacionadas à pesquisa e ao trabalho realizado são listadas a seguir:

1. Zuza, L. S., Cardoso, A., Lamounier, E., & Caetano, D. (2024). Análise dos Parâmetros da YOLOv8 na eficiência da identificação de reatores de núcleo de ar a partir de imagens de subestações de energia. In: *CISTI'2024 - 19ª Conferência Ibérica de Sistemas e Tecnologias de Informação*, 25-28 de junho de 2024, Salamanca, Espanha.

Sumário

1	INTRODUÇÃO	1
1.1	Motivação	1
1.2	Objetivos e Metas	3
1.3	Estrutura da Dissertação	3
2	FUNDAMENTAÇÃO TEÓRICA	5
2.1	Subestações de Energia	5
2.2	Realidade Virtual	7
2.3	Redes Neurais Artificiais	8
2.3.1	You Look Only Once	14
2.3.2	Batch	16
2.3.3	Otimizadores	17
2.3.4	Precisão e Recall	18
2.4	Considerações finais	19
3	TRABALHOS RELACIONADOS	20
3.1	Introdução	20
3.2	Alteração de Parâmetros da RNC	20
3.2.1	Identificação e Medição de Defeitos em Produtos Automotivos Usando Visão Computacional	20
3.3	Aplicação YOLOv8	22
3.3.1	Comparação de Modelos YOLOv5 e YOLOv8 para Detecção de Objetos em Áreas Rurais Usando Transferência de Aprendizado	22
3.3.2	UAV-YOLOv8: A Small-Object-Detection Model Based on Improved YOLOv8 for UAV Aerial Photography Scenarios	25
3.4	Considerações Finais	27
4	MATERIAIS E MÉTODOS	29
4.1	Introdução	29
4.2	Base de Dados	29
4.2.1	Captura das Fotos	29
4.2.2	Seleção das fotos	30
4.2.3	Rotulação das fotos	31
4.3	Treinamento	31
4.3.1	Materiais	31
4.3.2	Variação dos Parâmetros	32

4.4	Avaliação do Treinamento	32
4.5	Considerações Finais	33
5	DETALHES DA IMPLEMENTAÇÃO	34
5.1	Introdução	34
5.2	Estrutura geral da aplicação	34
5.3	Treinamento com a YOLO	36
5.3.1	Treinamento com a YOLOv5, YOLOv6 e YOLOv8	36
5.3.2	Treinamento com a YOLOv7	37
5.3.3	Variação dos otimizadores com a YOLOv8	38
5.4	API de Identificação	38
5.4.1	Implementação na Unity	40
5.5	Considerações Finais	42
6	RESULTADOS E DISCUSSÃO	43
6.1	Introdução	43
6.2	Resultados	43
6.2.1	Resultados e análise do treinamento com a YOLOv5	43
6.2.2	Resultados e análise do treinamento com a YOLOv6	44
6.2.3	Resultados e análise do treinamento com a YOLOv7	44
6.2.4	Análise dos Resultados do Treinamento com YOLOv8	45
6.2.5	Comparação de todas as versões	47
6.2.6	Avaliação da Implementação	49
6.3	Considerações Finais	49
7	CONCLUSÃO E TRABALHOS FUTUROS	51
7.1	Introdução	51
7.2	Conclusão	51
7.3	Trabalhos Futuros	52
	REFERÊNCIAS	53
A	SCRIPT API	58
A.1	api-rv.py	58
B	SCRIPT DO UNITY	59
B.1	ModelLoaderMenu.cs	59

Listas de ilustrações

Figura 1 – Representação dos componentes do sistema de geração e transmissão de energia (LOPES et al., 2012)	5
Figura 2 – Reatores de Núcleo de ar em uma subestação de energia (ELETRÔNICA, 2024)	6
Figura 3 – Representação de um NA (ADORNO, 2017)	9
Figura 4 – Topologia básica de uma rede neural (FLECK et al., 2016)	11
Figura 5 – Exemplificação de como uma imagem nada mais é que uma matriz, e como a filtragem busca padrões pré-estabelecidos dentro da imagem (LIMA; RUBIK; MORAIS, 2020)	12
Figura 6 – Esquema de funcionamento de uma RNC (VARGAS; PAES; VASCONCELOS, 2016)	13
Figura 7 – Operação de filtragem na camada convolucional. Nota-se que o filtro é uma multiplicação de matrizes (MUSIC, 2024)	13
Figura 8 – Disposição das 24 camadas convolucionais e das 2 camadas totalmente conectadas (REDMON et al., 2016)	15
Figura 9 – Processo de predição da Yolo (REDMON et al., 2016)	15
Figura 10 – Cálculo de IoU. (PADILLA; NETTO; SILVA, 2020)	18
Figura 11 – Aumento de desempenho da precisão média, versão 5 para 8. (JOCHER; CHAURASIA; QIU, 2023)	19
Figura 12 – Exemplo de marcação de foto utilizando labelImg, de um defeito em um vidro (GONZAGA, 2023)	21
Figura 13 – Bloco de aprendizado da ResNet (HE et al., 2016)	21
Figura 14 – YOLOv5 (DIAS; FIGUEIREDO; MAFRA, 2021)	25
Figura 15 – YOLOv8 (DIAS; FIGUEIREDO; MAFRA, 2021)	25
Figura 16 – Comparação da YOLOv8 com a rede modificada	27
Figura 17 – Comparação da YOLOv8 com a rede modificada	27
Figura 18 – Captura de foto em Porto Velho (à esquerda) e Araraquara (à direita), onde se notam quatro reatores de núcleo de ar na parte inferior.	30
Figura 19 – LabelImg realizando a marcação de reatores de núcleo de ar	32
Figura 20 – Estrutura geral da Aplicação	35
Figura 21 – Interface para o usuário.	35
Figura 22 – Demonstração dos Scripts de Treinamento das versões YOLOv5, em cima, e YOLOv6, em baixo.	37
Figura 23 – Comando de Execução da YOLOv7	38
Figura 24 – Demonstração do Script de Treinamento da YOLOv8, com aplicação do otimizador <i>Adam</i> e com <i>batch</i> configurado para 8	38

Figura 25 – <i>API</i> de identificação de imagem	39
Figura 26 – <i>API</i> de identificação de imagem	40
Figura 27 – Método de processamento da imagem e inserção dos modelos conforme a resposta da <i>API</i>	41
Figura 28 – Método de processamento da imagem e inserção dos modelos conforme a resposta da <i>API</i>	42
Figura 29 – Gráficos gerados a partir do treinamento com a YOLOv5, utilizando <i>batch</i> igual a 16. Nota-se a evolução repentina para uma precisão acima de 90%	44
Figura 30 – Gráficos gerados a partir do treinamento com a YOLOv6.	45
Figura 31 – Gráficos gerados a partir do treinamento com a YOLOv7, utilizando <i>batch</i> igual a 16. O comportamento alternado sobressaltou-se nessa arquitetura	46
Figura 32 – Comparação da YOLOv5 à YOLOv8, com otimizador <i>SGD</i> e variação de <i>batch</i> de 2, 4, 8 e 16	48

Listas de tabelas

Tabela 1 – Tabela comparativa de desempenho de modelos YOLO com diferentes otimizações e tamanhos de batch.	23
Tabela 2 – Resumo comparativo dos trabalhos relacionados	28
Tabela 3 – Resultados de diferentes tamanhos de batch aplicando YOLOv5	43
Tabela 4 – Resultados de diferentes tamanhos de batch aplicando YOLOv6	44
Tabela 5 – Resultados de diferentes tamanhos de batch aplicando YOLOv7	45
Tabela 6 – Resultados de diferentes tamanhos de batch aplicando YOLOv8 e otimizador <i>Adam</i>	46
Tabela 7 – Resultados de diferentes tamanhos de batch aplicando YOLOv8 e otimizador <i>AdamW</i>	46
Tabela 8 – Resultados de diferentes tamanhos de batch aplicando YOLOv8 e otimizador <i>SGD</i>	47
Tabela 9 – Resumo do desempenho na aplicação da YOLOv8 na melhor configuração encontrada	49
Tabela 10 – Resumo comparativo dos trabalhos relacionados	50

Lista de abreviaturas e siglas

ADAM	Adaptive Moment Estimation
AP	Average Precision
COCO	Common Objects in Context
FP	Falso Positivo
FN	Falso Negativo
GD	Gradiente Descendente
GRVA	Grupo de Estudos de Realidade Virtual e Aumentada
GUI	Graphical User Interface
IA	Inteligência Artificial
IoU	Intersection Over the Union
mAP	Mean-average Precision
NA	Neurônio Artificial
RNC	Rede Neural Convolucional
RV	Realidade Virtual
SGD	Stochastic Gradient Descent
VANT	Veículo Aéreo Não Tripulado
VP	Verdadeiro Positivo
YOLO	You Look Only Once
UFU	Universidade Federal de Uberlândia

1 Introdução

1.1 Motivação

As subestações de energia desempenham um papel fundamental no sistema de distribuição de energia elétrica no Brasil, permitindo a transferência eficiente e segura de eletricidade entre diferentes níveis de tensão. Elas são cruciais para garantir que a eletricidade gerada em usinas seja entregue aos consumidores com a qualidade e confiabilidade necessárias. Efetivamente, desempenham um papel crucial na estabilidade do sistema elétrico, facilitando a manutenção, controle e proteção da rede. Seu funcionamento envolve várias etapas, começando com a recepção da eletricidade gerada em usinas de energia. A eletricidade é então transformada em níveis de tensão adequados para distribuição por meio de transformadores. Nas subestações, também ocorrem operações de chaveamento, onde os dispositivos de comutação são usados para controlar o fluxo de eletricidade e direcioná-lo para as áreas desejadas da rede. As subestações também estão equipadas com sistemas de proteção que detectam e isolam falhas para evitar danos ao sistema elétrico e garantir a segurança dos equipamentos e dos operadores (RANDOLPH, 2013).

Contudo, falhas de segurança durante as rotinas de um colaborador não são raras no setor elétrico, podendo causar danos à sua saúde, e em alguns casos levando a óbito. De acordo com o estudo de (LIMA; OLIVEIRA, 2021), acidentes de trabalho são muito comuns em ambientes como subestações de energia. Para avaliar as causas, foram analisados diversos processos trabalhistas durante um período de tempo contra empresas que prestam serviço neste setor. Foram concluídas as existência de várias as causas, mas uma se destaca: a falta de treinamento. No estudo citado, este fato é atrelado à terceirização dos serviços. Enquanto um funcionário direto da companhia de energia da região recebia 6 meses de treinamento, o funcionário terceirizado era treinado em um período médio de 30 a 40 dias. A qualidade do conteúdo destes treinamentos para o funcionário terceirizado também era muito mais superficial. Em uma análise dentro deste estudo, a respeito de um acidente fatal, foi verificado que um técnico foi acionado para resolução de um problema de rompimento de cabo. Devido a um descuido, um dos colaboradores tocou em um cabo energizado, sem se preocupar em verificar se todas chaves estariam desligadas, acabando por o levar a óbito. Deste estudo, portanto, entendeu-se que a busca por custos mais baixos durante o treinamento, pessoas que se expõe ao risco, e o próprio funcionamento da transmissão de energia é colocado à prova.

Nesse cenário, intervenções tecnológicas podem desempenhar um papel crucial na melhoria das condições de treinamento de operadores no sistema elétrico. Soluções como utilização de aplicações de Realidade Virtual (RV) oferecem oportunidades significativas

para aprimorar a gestão e operação das subestações. O monitoramento em tempo real permite identificar falhas antes que se tornem graves, facilitando a tomada de decisões mais rápidas e eficazes, enquanto a RV possibilita treinamentos imersivos e realistas, preparando os operadores para enfrentar uma ampla gama de situações sem expô-los a riscos reais. Além disso, a automação de processos nas subestações reduz a dependência de intervenções humanas, aumentando a eficiência e confiabilidade do sistema elétrico. A implementação dessas tecnologias não só eleva a segurança dos colaboradores ao evitar exposições diretas a ambientes perigosos, mas também diminui os custos operacionais ao automatizar tarefas e agilizar a resolução de problemas. Dessa forma, as empresas do setor elétrico podem garantir um fornecimento de energia mais confiável e contínuo para os consumidores finais (ZHOU; FU; YANG, 2016).

Entretanto, a criação de ambientes de RV para treinamento em subestações de energia é um processo trabalhoso, complexo, exigindo tempo para seu desenvolvimento completo. Além disso, a modelagem gráfica desses ambientes demanda muitos recursos exigindo uma equipe especializada para desenvolver modelos 3D detalhados e sistemas de interação sofisticados. Além disso, o hardware utilizado precisa ser suficientemente poderoso para garantir que a experiência seja fluida e imersiva. Caso contrário, a qualidade do treinamento pode ser prejudicada e não atingir o resultado almejado (JUNIOR et al., 2022).

Em suma, para funções didáticas, como treinamentos, a RV se destaca como uma abordagem disruptiva em relação a métodos tradicionais, principalmente pelo alto nível de imersibilidade no contexto da aplicação, proporcionado ao usuário e ao alto resultado no aprendizado do conteúdo trabalhado. Contudo, para construir e preparar todo o ambiente para uma experiência imersiva em RV, faz-se necessário a elaboração de uma complexa estrutura que envolve desde a escolha do equipamento que será utilizado para a projeção ao usuário, como por exemplo, uma caverna de visualização, capacete de virtualização ou mesmo óculos de RV, até a criação, em softwares próprios para esse tipo de desenvolvimento, toda modelagem gráfica do ambiente até as interações que existentes na aplicação. Fatores como a capacidade gráfica e técnica são levadas em consideração nesta etapa, uma vez que aplicações com grande quantidade de interações e elementos, exigem do hardware que irá renderizar elevada capacidade de processamento. Se a demanda pela capacidade for alta, e não for suportada pelo hardware, será exigido do desenvolvedor redução na qualidade das texturas, assim como outros tratamentos para que toda a experiência durante a imersão não seja lenta ou mesmo careça de elementos que destitui a aplicação de imersibilidade (PALMEIRA et al., 2020).

Outra ferramenta que se tem tornado crucial na ciência para identificar padrões em dados complexos são as Redes Neurais Artificiais (RNA). Essas redes são particularmente eficazes na resolução de problemas que envolvem equações não lineares, ou seja, situa-

ções onde as soluções não são diretas e as interações entre variáveis são complexas. Tal abordagem se mostra essencial ao lidarmos com dados do mundo real, como imagens, que precisam ser processadas computacionalmente para reconhecer padrões. O funcionamento das RNAs é inspirado na estrutura e no funcionamento do cérebro humano; replicando o aprendizado natural, elas são capazes de enfrentar desafios complexos, tornando-se uma escolha excelente para pesquisas avançadas (OĞCU; DEMIREL; ZAIM, 2012).

Deste modo, motivado pelo possibilidade de elaborar uma ferramenta que simplifique o desenvolvimento de um sistema em RV voltado para aplicações de treinamento de colaboradores em subestações de energia, este trabalho se propõe a construir uma ferramenta que faça a inserção automática de componentes em um ambiente de RV de uma subestação de energia. Toda a automação será alimentada por um modelo treinado a partir de uma RNA, alimentada por fotos capturadas por VANTs em duas subestações de energia diferentes.

1.2 Objetivos e Metas

O objetivo geral desta pesquisa é propor um sistema de inserção automática de componentes em ambientes virtuais de treinamento para subestações de energia a partir de imagens aéreas coletadas do local a ser mapeado virtualmente. Para alcançar esse objetivo geral, foram estabelecidos os seguintes objetivos específicos:

- Realizar uma revisão da literatura científica, para identificar quais os algoritmos utilizados no reconhecimento de padrões em imagens aéreas obtidas por VANTs;
- Construir uma base de fotos de subestações de energia para compor o conjunto de dados usados para o treinamento da RNA.
- Estudar e avaliar quais são os hiperparâmetros do algoritmo de inteligência artificial a ser utilizado, que garanta maior eficiência no reconhecimento de componentes das subestações elétricas;;
- Desenvolver uma automação capaz de receber uma imagem, reconhecer componente(es) da subestação elétrica inserir no ambiente virtual.

1.3 Estrutura da Dissertação

A presente dissertação é composta por sete capítulos, descritos da seguinte forma.

- No Capítulo 1 são apresentadas as motivações, os objetivos e a estruturação do trabalho;

- No Capítulo 2 são apresentados os principais fundamentos teóricos relacionados ao trabalho;
- No Capítulo 3 é apresentado o estado da arte da linha de pesquisa principal desse trabalho;
- Nos Capítulos 4 e 5 são apresentados materiais/métodos e detalhes de implementação;
- No Capítulo 6, são discutidos e apresentados os resultados obtidos nesse trabalho a partir do sistema desenvolvido;
- Por fim, no Capítulo 7, são apresentadas as conclusões e as perspectivas para trabalhos futuros.

2 Fundamentação Teórica

2.1 Subestações de Energia

Atualmente, o sistema elétrico do Brasil é formado por um conjunto de usinas, subestações, linhas de transmissão e demais equipamentos, que viabilizam a produção, transmissão e distribuição de energia elétrica. A energia é gerada em usinas que variam conforme os recursos utilizados, como hidroelétricas, termoelétricas, eólicas, nucleares, entre outras, e é então transportada por linhas de transmissão até subestações de energia de empresas distribuidoras. Nestas subestações, a tensão é ajustada para os níveis adequados de consumo. Esse processo elétrico é composto por três etapas principais: geração, transmissão e distribuição (Figura 1). O sistema elétrico é composto por diversos elementos, sendo os mais destacados os geradores, responsáveis pela transformação de energia mecânica em elétrica, e os sistemas de transmissão e distribuição, que conduzem a energia até os consumidores. As subestações têm um papel fundamental ao ajustar a tensão na transmissão e distribuição, assegurando a confiabilidade e a qualidade do serviço elétrico fornecido aos clientes (LOPES et al., 2012).

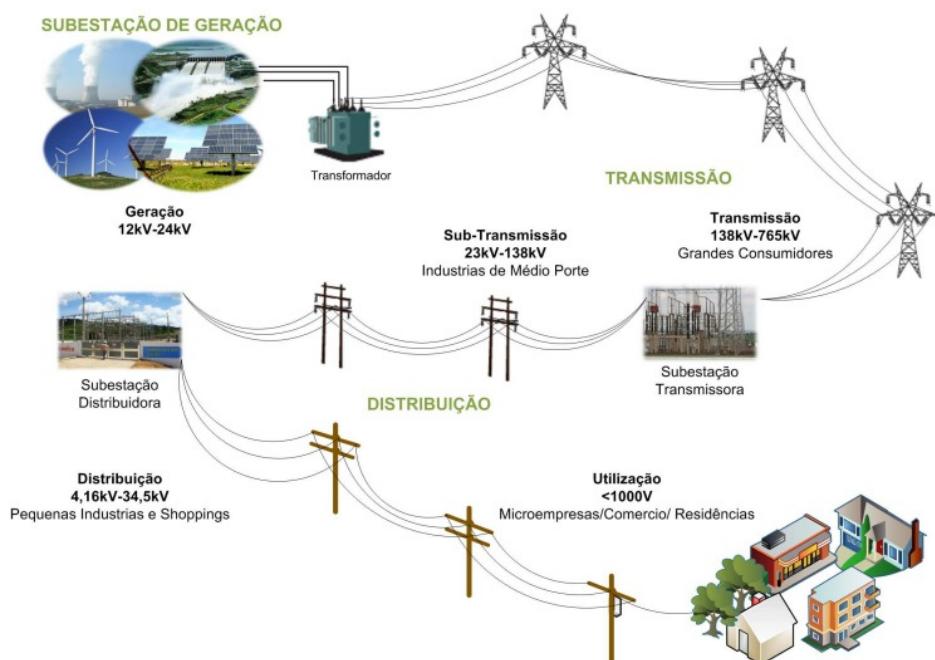


Figura 1 – Representação dos componentes do sistema de geração e transmissão de energia (LOPES et al., 2012)

De modo geral, portanto, a energia é obtida por geradores e transmitida para

subestações primárias. Há aí a primeira aferição de qualidade dos níveis de transmissão. Após isso, a energia transformada é enviada para subestações de distribuição, alterada para níveis mais baixos para serem transmitidas para setores industriais e urbanos (JUNIOR et al., 2022).

É importante destacar que a energia produzida no Brasil é oriunda majoritariamente de fontes renováveis, em especial da geração hidroelétrica, correspondendo a 69% de toda malha produtora. Ela é considerada uma energia limpa, de baixo custo e possui reduzida emissão de gases de efeito estufa. (ANDRADE; SANTOS, 2015).

Analizando mais detalhadamente as subestações, podemos afirmar que elas são encarregadas de regular a tensão tanto na transmissão quanto na distribuição de energia. Elas operam de forma autônoma, contando com sistemas supervisórios para o controle e supervisão remotos. Dentro das subestações, funções como comutação, proteção e controle também são desempenhadas. Um dos problemas que as subestações resolvem é a interrupção de curtos-circuitos, utilizando para isso disjuntores. Existem diversos tipos de subestações: as de transmissão, que conectam linhas de transmissão com diferentes voltagens; as de distribuição, que ligam linhas de transmissão a linhas de distribuição e regulam a tensão; e as subestações coletores, que são utilizadas na geração de energia eólica para conectar a geração às linhas de transmissão. Portanto, as subestações são fundamentais para garantir a confiabilidade e a qualidade do fornecimento de energia (LOPES et al., 2012).



Figura 2 – Reatores de Núcleo de ar em uma subestação de energia (ELETRÔNICA, 2024)

Dentre os diversos equipamentos que compõem a subestação de energia, um em

particular será objeto de estudo neste trabalho: os reatores de núcleo de ar (Figura 2), cujo interesse reside em seu formato geométrico, que será tratado posteriormente. Nota-se que possui um corpo cilíndrico. Em seu topo, pode possuir ou não uma proteção circular contra as intempéries. Estes dispositivos são essenciais para a regulação da reatividade e controle de correntes indutivas em sistemas de transmissão e distribuição de energia. Diferentemente dos reatores de núcleo de ferro, os reatores de núcleo de ar oferecem vantagens como uma resposta mais linear e a capacidade de operar sem saturação magnética, o que os torna ideais para aplicações onde é necessário um comportamento previsível e estável (JUNIOR, 2012).

Dentre os diversos equipamentos que compõem uma subestação de energia, este trabalho focará especificamente nos reatores de núcleo de ar. A escolha desse tipo de reator se justifica por suas características físicas quando capturadas em uma foto aérea obtida via VANT. Diferentemente de outros equipamentos presentes em seu entorno, os reatores de núcleo de ar têm um formato geométrico particular, com um corpo cilíndrico que pode incluir ou não uma proteção circular contra intempéries em seu topo. Esse formato possibilita que a RNA possa generalizar de maneira simplificada um modelo que represente os reatores, cumprindo assim a proposta do protótipo. Além disso, esses equipamentos são essenciais para distribuição de energia elétrica, atuando na regulação da reatividade e controle de correntes indutivas em sistemas de transmissão e distribuição de energia. Diferentemente dos reatores de núcleo de ferro, os reatores de núcleo de ar oferecem vantagens como uma resposta mais linear e a capacidade de operar sem saturação magnética, o que os torna ideais para aplicações onde é necessário um comportamento previsível e estável (JUNIOR, 2012).

2.2 Realidade Virtual

RV pode ser definida como um ambiente gerado a partir de um sistema computacional em que o usuário não apenas se sente dentro do contexto artificial, como também possibilita ao usuário a consciência de que pode navegar e interagir neste ambiente virtual. Trata-se, por isso, de uma interessante interface entre um ser humano e um computador. Nela, é possível que haja navegabilidade, interações e, principalmente, imersão em um ambiente sintético, gerado computacionalmente por meio de canais multisensoriais. Ou seja, podemos concluir que para que haja RV é necessário que existam esses três conceitos: interação, imersão e navegação (KALAWSKY, 1993).

De acordo com (CARDOSO et al., 2007), pode-se classificar a RV de duas formas diferentes: Imersiva e Não-imersiva. Na primeira, cria-se uma RV que isole o usuário do mundo real. Seus sentidos são bloqueados para recepção dos estímulos do seu entorno, para que sejam direcionados àqueles oriundos do mundo fictício. Para tanto, uma ampla

variedade de equipamentos, como fones de ouvidos, luvas de dados, óculos de RV, entre outros, são empregados para os resultados esperados. Já a RV Não-Imersiva não se isola do mundo real. Ou seja, o usuário tem consciência de que está em um ambiente artificial. De modo similar, uma ampla variedade de equipamentos, convencionais e não-convencionais, é empregada para gerar essa interação, incluindo dispositivos do cotidiano, como mouses, monitores de computador.

O surgimento da RV data de 1963, em que foi apresentado uma aplicação de manipulação de objetos tridimensionais em um computador, denominada Sketchpad (SUTHERLAND, 1963). A aplicação conseguia reproduzir interatividade por meio de uma caneta óptica, que era utilizada para seleção de objetos projetados em uma tela. Neste trabalho, surgiram alguns dos principais termos da RV, como representação visual, dispositivos especiais, e interações em tempo real. Em 1968, o mesmo autor do trabalho anterior, Sutherland, publicou outro trabalho marcante para a história: A Head-Mounted Three Dimensional Display (SUTHERLAND, 1968). Nele, foi introduzido o conceito de imersividade para uma RV; no caso, um capacete capaz de projetar fotos diretamente aos olhos do usuário, assim como rastrear o movimento da cabeça, afim de que este movimento fosse responsivo no ambiente virtual. Seguido a isso, uma série de outros equipamentos foram desenvolvidos a fim de sofisticar as soluções do ramo e propor novos modelos. A tendência sempre prevaleceu de surgir ferramentas mais simples e acessíveis ao usuário final (KIRNER; KIRNER, 2011).

Além das aplicações lúdicas, a RV atua de maneira séria e efetiva em diversas áreas técnicas. Como ferramenta para treinamento de operadores em ambientes de risco e de difícil simulação, torna-se uma alternativa viável para capacitação. O trabalho de (SILVA et al., 2012) demonstra bem essa ideia ao investigar técnicas de RV para que uma pessoa comum, sem treinamento anterior, possa, de maneira segura à uma subestação em funcionamento e à própria pessoa e todos ao seu redor, interagir com um transformador de energia, executando todos procedimentos que teria em um ambiente real, contudo de maneira virtual. Este treinamento já seria uma base interessante para que um profissional pudesse agilizar seu treinamento no mundo real ou mesmo absorvê-lo. Em trabalhos de alta periculosidade tudo é crítico; então, qualquer intervenção que possa atenuar os riscos inerentes à natureza do trabalho, traz grandes avanços ao processo de profissionalização de técnicos e pessoas interessadas.

2.3 Redes Neurais Artificiais

A RNA trata-se de um conjunto de técnicas que buscam simular o funcionamento do cérebro humano, a partir de algoritmos computacionais, a fim de resolver problemas específicos. Sua eficiência não necessariamente está relacionada com a quantidade de

interações que as unidades de processamento que o compõem realizam entre si, uma vez que outros fatores, como tamanho do conjunto de entradas ou mesmo a qualidade delas podem influenciar o treinamento. Comparam-se as RNA à mente humana devido a sua capacidade de aprendizado, podendo generalizar funções a partir de alguns exemplos informados, e delas prever o comportamento de valores para os quais não foi fornecido a resposta esperada. A base do funcionamento da RNA está no conceito do neurônio artificial (NA), que se traduz como uma pequena unidade de processamento, capaz de receber um sinal simples, e a partir dele gerar uma resposta. De acordo com (FLECK et al., 2016), a primeira noção de NA surge em 1943, no trabalho Warren McCulloch e Walter Pitts, no artigo: “A Logical Calculus of the Ideas Immanent in Nervous Activity”.

Matematicamente, o NA recebe um valor de entrada, realizando com ele o produto desse valor a um outro denominado peso. O resultado é comparado com um diferenciador: se for maior, será dada a saída verdadeira; se menor, falso. Na Figura 3, esse processo é demonstrado de maneira esquemática. A operação descrita, chamada também de *threshold logic*, i.e., lógica limiar, em tradução livre, mimetiza o funcionamento de um componente eletrônico chamado transistor, base do funcionamento dos processadores computacionais modernos, em que a passagem de corrente elétrica por ele é interrompida ou permitida com base no sinal de entrada (MCCULLOCH; PITTS, 1943).

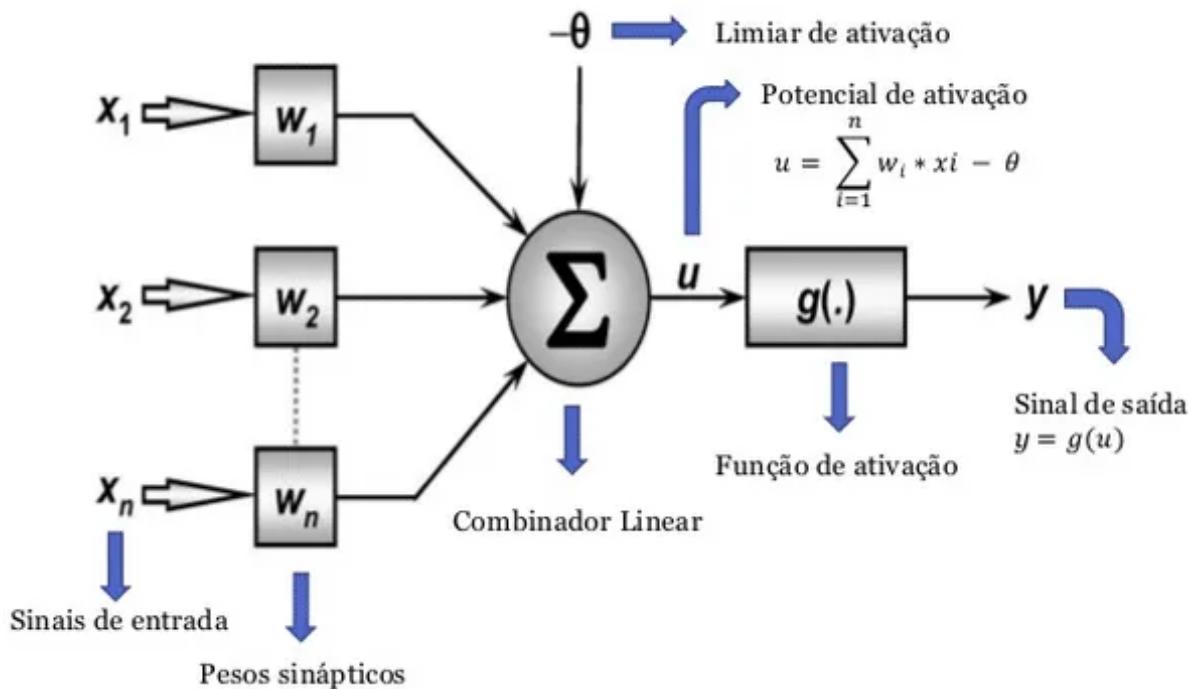


Figura 3 – Representação de um NA (ADORNO, 2017)

Aprofundando-se na álgebra relacionada do NA, pode-se visualizar o esquema do seu funcionamento na Figura 3, e também acompanhar a definição de cada envolvido nesta operação (BRASIL, 2023):

- **Sinais de entrada:** Constituem-se os sinais, ou valores, externos ao modelo, muitas vezes submetidos a algum tratamento prévio, responsáveis por alimentar a rede.
- **Pesos sinápticos:** Também chamados apenas de pesos, ponderam os sinais de cada entrada da rede.
- **Combinador linear:** Operação aritmética envolvendo os sinais de entrada a fim de gerar um potencial de ativação.
- **Bias:** O bias é um valor adicional que é somado à combinação linear das entradas antes de passar pela função de ativação, visando ajustá-la para que os dados melhor se adaptem à rede.
- **Limiar de ativação:** Também denominado *threshold*, determina o nível adequado em que o resultado obtido pelo combinador linear pode acionar a ativação.
- **Potencial de ativação:** É o resultado decorrente da discrepância entre o valor gerado pelo combinador linear e o limiar de ativação. Se esse resultado for positivo, indicando $u \geq 0$, o neurônio gera um potencial de excitação; caso contrário, o potencial será inibitório.
- **Função de ativação:** Sua função é restringir a saída de um neurônio dentro de um determinado intervalo de valores.
- **Sinal de saída:** Resultado final da saída, que pode ser utilizado como entrada para outros neurônios interligados sequencialmente.

Em suma, conforme (GOODFELLOW; BENGIO; COURVILLE, 2016), a RNA, enquanto um conjunto de NA, pode ser entendido como um processador robusto, distribuído de maneira paralela, com pequenas unidades de processamento. Sua semelhança ao cérebro humano, deve-se, portanto, à capacidade de aprendizado e aos sinais sinápticos, existentes entre os neurônios, responsáveis pelo processo de armazenamento de conhecimento (HAYKIN, 2001).

De acordo com (RAUBER, 2005), em 1958, após à concepção do NA, a primeira RNA a se notabilizar e a continuar sendo utilizada até o presente momento trata-se da Perceptron (ROSENBLATT, 1958). Este algoritmo possuía a capacidade de alterar os pesos dos neurônios, conforme o avanço do processamento da rede, de modo a resolver problemas da classificação linear. Seguido a ela, em 1960, houve a concepção da rede Adaline (WIDROW; HOFF et al., 1960). Diferentemente da perceptron, esta rede já se mostrava capaz de produzir resultados que iam além dos valores binários, sendo capaz de gerar respostas de valores presentes em todo o conjunto dos números reais. O cálculo da rede Adaline era baseado na regra Delta, que era capaz de aproximar os valores dos pesos gerados

para aqueles valores com menor erro possível. Ambas as redes, contudo, apresentavam uma limitação quanto à modificação dos pesos em todas suas multicamadas. Para esse cenário, foi proposto o conceito de backpropagation, ou, em tradução livre, retropropagação do erro (RUMELHART; HINTON; WILLIAMS, 1986). A ideia da retropropagação se inicia com o feedforward, em que todos os dados de entrada de uma RNA são transmitidos do início à camada de saída, sem nenhuma alteração de peso. Desta forma, calcula-se o erro total dessa primeira passagem, comparando o valor resultante com o esperado. Essa métrica de erro torna-se o guia para o ajuste dos pesos entre as conexões de cada neurônio. Com ela, é feito o caminho de volta, e calculado um gradiente de erro, que será o fator decisivo para a atualização dos pesos durante todo o treinamento.

A topologia básica de uma rede neural é dividida em três níveis de camada: a camada de entrada, a camada oculta e a camada de saída, conforme mostrado na Figura 4. A camada de entrada é, naturalmente, aquela que recebe os dados de entrada. Nela, cada nó representa uma característica ou atributo dos dados que estão sendo alimentados na rede neural. Por exemplo, em uma aplicação de reconhecimento de imagens, cada nó na camada de entrada pode representar um pixel da imagem. A camada oculta, que não se restringe a apenas uma, podendo ser várias, representa um processo intermediário entre a camada de entrada e a camada de saída. Cada neurônio em uma camada oculta recebe entradas das camadas anteriores, realiza algum tipo de transformação não linear dessas entradas e passa o resultado para a próxima camada. A presença de múltiplas camadas ocultas permite que a rede aprenda representações complexas e abstratas dos dados. Por fim, a camada de saída representa a camada final, e ela é responsável por gerar as saídas desejadas. A estrutura da camada de saída depende do tipo de problema que está sendo resolvido. Por exemplo, em um problema de classificação, cada nó na camada de saída pode representar uma classe diferente, e a saída pode ser interpretada como a probabilidade de pertencer a cada classe (RAUBER, 2005).

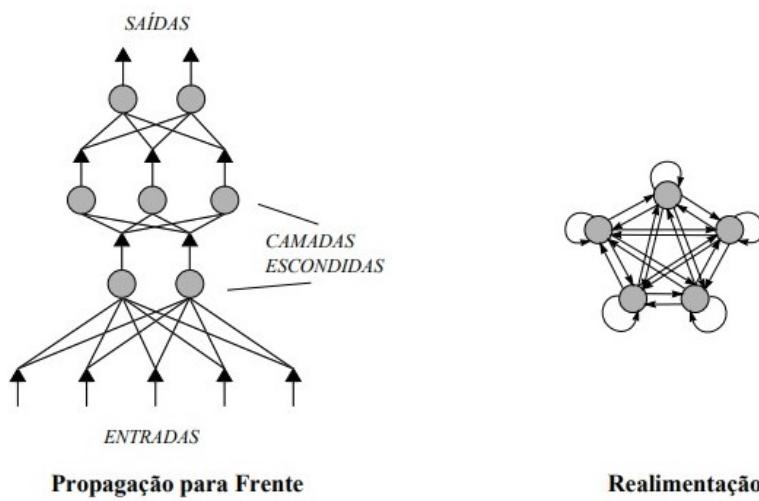


Figura 4 – Topologia básica de uma rede neural (FLECK et al., 2016)

Essencialmente, as redes neurais aprendem iterativamente ajustando os pesos de suas conexões através do processo de treinamento, onde são apresentados a um conjunto de dados de entrada e as correspondentes saídas desejadas. Com o tempo, a rede neural é capaz de aprender a mapear efetivamente os padrões nos dados de entrada para as saídas desejadas, tornando-se assim capaz de realizar tarefas como reconhecimento de padrões, classificação, regressão, entre outros (HAYKIN, 2001).

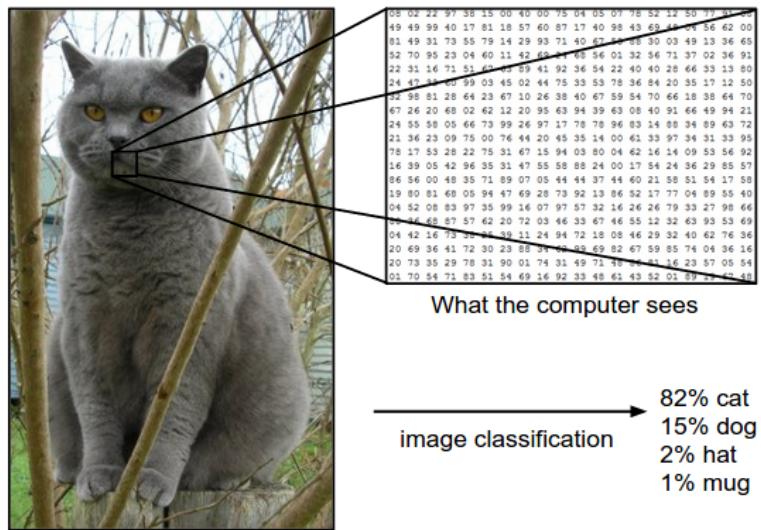


Figura 5 – Exemplificação de como uma imagem nada mais é que uma matriz, e como a filtragem busca padrões pré-estabelecidos dentro da imagem (LIMA; RUBIK; MORAIS, 2020)

Contudo, para problemas envolvendo aprendizado por meio de imagens, a forma de lidar com as informações é diferente. De fato, imagens são dados, organizados em formato de matrizes de duas dimensões ou três dimensões (se for considerada a camada de cores), em que cada unidade de informação se chama pixel, conforme se nota no exemplo da Figura 5. Para processar esses dados, um tipo de RNA destaca-se: a Rede Neural Convolucional (RNC). Comparando ambas, nota-se que a RNA é composta por camadas intrinsecamente conectadas, em que cada neurônio de uma camada se conecta a todos os neurônios da camada seguinte. Isso faz com que ela seja adequada para dados vetorizados e para problemas de classificação e regressão sem uma estrutura espacial ou temporal específica. Por sua vez, a RNC recorre a dois tipos de camadas (Figura 6): a convolucional, cujo objetivo é extrair características locais, e a de *pooling*, responsável por manter a estrutura espacial essencial (VARGAS; PAES; VASCONCELOS, 2016).

A camada de convolução, portanto, é responsável pela *feature extractor*, i.e., pela extração de características de interesse em uma imagem. Basicamente, são aplicados filtros, também chamados de *kernel*, à imagem a fim de buscar padrões. São comumente empregados filtros de detecção de bordas (*edge detection*), desfoque (*blur*), nitidez (*sharpen*). Conforme na Figura 7, cada elemento do filtro é multiplicado pelo elemento de mesma posição na região em que o filtro está sendo aplicado naquele instante. Ao final dessas

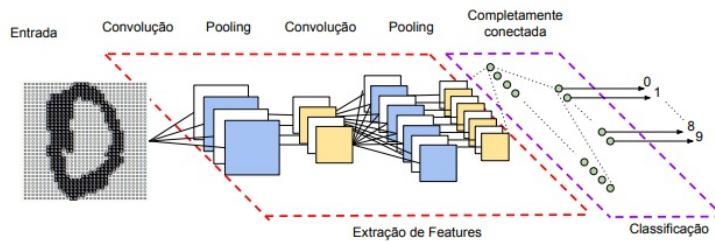


Figura 6 – Esquema de funcionamento de uma RNC (VARGAS; PAES; VASCONCELOS, 2016)

operações matriciais, adiciona-se os resultados dessas multiplicações para ter um único valor como saída, que será o pixel correspondente na imagem filtrada (RODRIGUES, 2023)

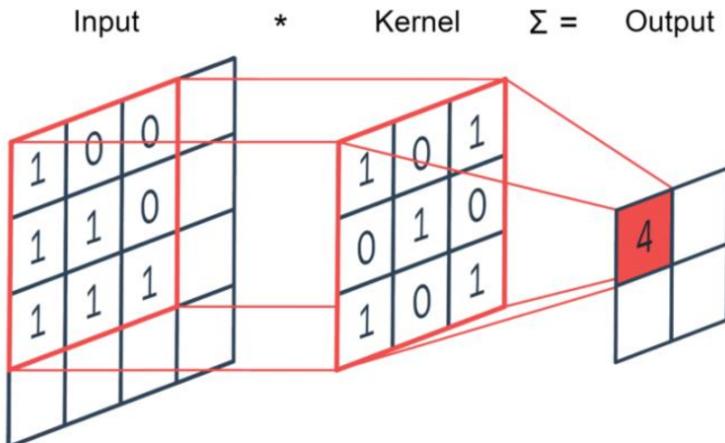


Figura 7 – Operação de filtragem na camada convolucional. Nota-se que o filtro é uma multiplicação de matrizes (MUSIC, 2024)

A camada de *pooling* é aplicada após cada camada convolucional, com o objetivo de reduzir as dimensões das imagens enquanto mantém a profundidade do volume. Esse processo promove a invariância da RNC às transformações geométricas, permitindo à rede reconhecer objetos na imagem independentemente de sua posição. Além disso, o uso da camada de *pooling* contribui para diminuir significativamente o custo computacional da rede. Existem diferentes tipos de *pooling*, como o *max pooling*, que seleciona o valor máximo em uma região específica da imagem, e o *average pooling*, que calcula a média dos valores nesta região. O *max pooling* é o mais comum, pois preserva as características mais salientes das imagens, enquanto reduz o ruído. A camada de *pooling*, ao reduzir o tamanho espacial dos mapas de características, também ajuda a evitar um problema comum no aprendizado de máquina, em que um modelo aprende não apenas os padrões relevantes dos dados de treinamento, mas também o ruído e as particularidades específicas desses dados. Chama-se este problema de *overfitting*. Evitando-o, a rede consegue construir

um treinamento mais generalizável e que possa gerar resultados relevantes para dados diferentes do modelo treinado (LIMA; RUBIK; MORAIS, 2020).

Por último, a camada totalmente conectada emprega uma função de ativação na sua camada de saída para realizar a classificação. Ser totalmente conectada significa que todos os neurônios da camada anterior estão ligados a todos os neurônios da camada subsequente. Ela recebe as saídas da camada de convolução e da camada de *pooling* e, em seguida, utiliza essas informações para determinar a classe à qual a imagem de entrada pertence (SILVA, 2018).

As RNC representam por si só uma revolução no campo da visão computacional, tornando-se a base para uma variedade de aplicações, desde reconhecimento de imagens até análise de vídeos. Entre as arquiteturas mais notáveis estão a LeNet, AlexNet, VGGNet, GoogLeNet e ResNet, cada uma contribuindo com avanços significativos em profundidade, eficiência e precisão. Além dessas, uma destaca-se em especial: a YOLO (“You Only Look Once”), uma abordagem inovadora para treinamento e detecção de objetos (CARBONI, 2021).

2.3.1 You Look Only Once

YOLO, uma RNC, é conhecida por sua eficácia e precisão na detecção de objetos em imagens e vídeos. Desenvolvida por Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi em 2015, a arquitetura do YOLO foi apresentada em sua primeira versão no artigo "You Only Look Once: Unified, Real-Time Object Detection" (REDMON et al., 2016).

A arquitetura do YOLO é singular: ao contrário de outras RNCs que fazem múltiplas passagens pela imagem em busca de objetos, o YOLO analisa a imagem inteira de uma vez, justificando seu nome, que em tradução livre significa “você olha apenas uma vez”. Essa abordagem permite ao YOLO aplicar uma única rede neural à imagem completa. Durante o processamento, a imagem é dividida em regiões menores, e a rede prevê caixas delimitadoras, as probabilidades de existência de objetos nessas caixas, e a classe provável de cada objeto. Sua classificação, enquanto rede neural, vai além da RNC, pois ela utiliza uma arquitetura conhecida como Darknet, um tipo de rede neural profunda (variação da RNC) e sua implementação original foi desenvolvida em C, embora agora esteja disponível em várias outras linguagens de programação, graças ao apoio da comunidade e de empresas. Quando utilizada para treinamento sua estrutura básica, estabelecida em sua primeira versão, segue, evidentemente, o padrão de uma RNC. As camadas convolucionais iniciais da rede extraem características da imagem, enquanto as camadas totalmente conectadas preveem as probabilidades de saída e as coordenadas (Figura 8). Em sua primeira versão, a rede possuia 24 camadas convolucionais seguidas por 2 camadas totalmente conectadas (COMPUTACIONAL, 2024).

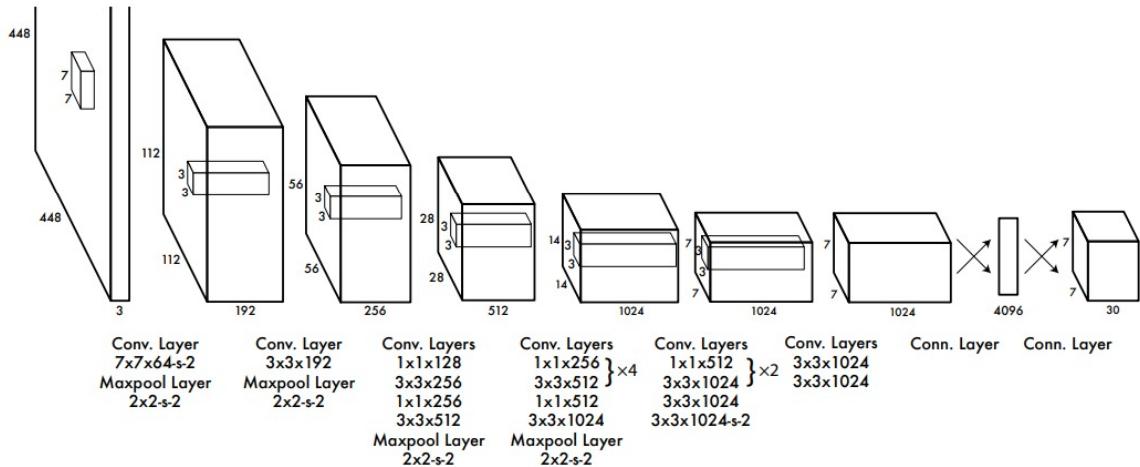


Figura 8 – Disposição das 24 camadas convolucionais e das 2 camadas totalmente conectadas (REDMON et al., 2016)

Embora existam várias versões do YOLO, há um funcionamento geral da arquitetura. O primeiro passo do YOLO é dividir a imagem em uma grade de células S por S. Nas versões iniciais, essa grade era de 13x13, totalizando 169 células, enquanto nas versões mais recentes é de 19x19. Cada célula é responsável por prever/detectar 5 caixas delimitadoras, pois pode haver múltiplos objetos em uma única célula. Portanto, na versão de exemplo do YOLO, há um total de 845 caixas delimitadoras (13x13x5). Na Figura 9, esse processo é representado. Notam-se diversas caixas, aquelas com bordas mais grossas indicam que a probabilidade de a demarcação estar representando o objeto que se propõe é alta. (JIANG et al., 2022).

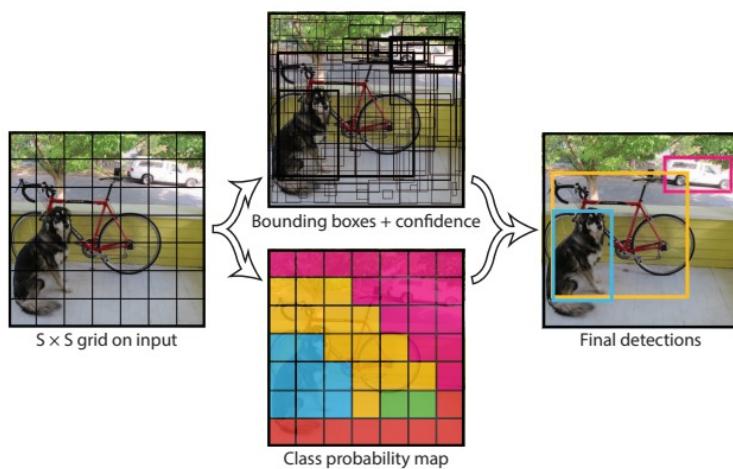


Figura 9 – Processo de predição da Yolo (REDMON et al., 2016)

A partir da YOLOv3, foram introduzidos conceitos para definir algumas partes de sua arquitetura. Primeiramente, define-se a camada de *backbone*, ou espinha dorsal. Basicamente, é nesta etapa que ocorre a extração de características significativas para o

processo de classificação. Ela captura características hierárquicas em diferentes escalas, com características de nível inferior (como bordas e texturas) nas camadas iniciais e características de nível superior (como partes de objetos e informações semânticas) nas camadas mais profundas. O *neck*, ou pescoço, conecta a camada *backbone* à *head*, ou cabeça, agregando e refinando as características extraídas, muitas vezes focando em melhorar a informação espacial e semântica em diferentes escalas. A *head* processa as características fornecidas pelo *neck*, gerando previsões para cada candidato a objeto. Resumindo, a partir da YOLOv3, a arquitetura foi dividida em três partes: *backbone* para extração de características; *neck* para agregar e refinar características; e *head* para fazer previsões. A espinha dorsal usa uma CNN para capturar características hierárquicas, o pescoço melhora a informação espacial e semântica, e a head gera as previsões finais, que são refinadas por pós-processamento para eliminar sobreposições (TERVEN; CORDOVA-ESPARZA, 2023).

Atualmente, a oitava versão da YOLO, ou, YOLOv8 , é considerada estado da arte em se tratando de análise e treinamento de imagens (ULTRALYTICS, 2024). YOLOv8 traz uma série de avanços significativos em relação às versões anteriores, destacando-se pela melhoria no desempenho e precisão das detecções. A arquitetura foi otimizada para equilibrar velocidade e precisão, utilizando *backbones*, modernos como CSPDarknet para uma extração de características mais eficiente. O processo de treinamento foi acelerado e aprimorado com melhores métodos de regularização e otimização, enquanto a facilidade de uso e integração foi aprimorada, atendendo às demandas tanto de dispositivos de alta performance quanto de dispositivos móveis com menor capacidade de processamento (HUSSAIN, 2023).

2.3.2 Batch

Um dos aspectos cruciais do funcionamento da YOLO é o conceito de *batch* (em tradução livre, lote) durante o treinamento da rede neural. Ao agrupar várias imagens em lotes para processamento simultâneo, a YOLO aproveita a capacidade de processamento paralelo das GPUs, acelerando significativamente o treinamento. Durante a propagação direta, cada imagem no lote é processada pela rede neural para gerar previsões de detecção de objetos. Em seguida, a perda é calculada em relação às anotações verdadeiras, e os pesos da rede são atualizados para minimizar essa perda, usando algoritmos de otimização como o gradiente descendente. Esse processo é repetido para vários lotes de imagens até que a rede converja para uma solução adequada. Assim, o uso eficiente de lotes na YOLO não apenas acelera o treinamento, mas também contribui para a robustez e eficácia dos modelos de detecção de objetos resultantes (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.3.3 Otimizadores

O objetivo dos algoritmos de aprendizado de máquina supervisionados é otimizar a redução de uma função objetivo, geralmente chamada de função de custo J . Reduzir essa função é crucial, pois a rede precisa aprender os pesos que melhor representam o relacionamento entre os dados, formando um modelo preditivo para fazer previsões em novos conjuntos de dados. O algoritmo de otimização mais simples para encontrar esses pesos é o Gradiente Descendente (GD), onde se realiza pequenos passos em direção ao mínimo global da função de custo. No entanto, o GD enfrenta dificuldades com conjuntos de dados muito grandes devido à sua abordagem em lote, tornando o treinamento computacionalmente caro. Uma alternativa popular é o Gradiente Descendente Estocástico (SGD, sigla para a expressão em inglês “Stochastic Gradient Descent”), que aplica atualizações de peso com base em amostras aleatórias de dados de treinamento, resultando em convergência mais rápida. O SGD possui três parâmetros importantes que podem influenciar no treinamento da rede neural: a taxa de aprendizagem, que determina quão rápido o otimizador tentará treinar a rede neural, com uma taxa muito alta podendo levar a falhas no treinamento e uma taxa muito baixa resultando em treinamento lento; o momentum, que indica quanto a direção da mudança de peso anterior deve influenciar na etapa atual, acelerando o treinamento ao permitir que o otimizador persista em direções com histórico de melhoria; e o decay, utilizado para diminuir a taxa de aprendizagem conforme os erros diminuem durante o treinamento, evitando oscilações excessivas e permitindo uma convergência mais suave do modelo (SILVA, 2018).

Além do SGD, outro algoritmo de otimização recorrente é o Adam (“Adaptive Moment Estimation”, em tradução livre, “Estimação Adaptativa de Momento”). O otimizador Adam é um algoritmo popular de otimização de gradientes utilizado em treinamento de redes neurais e outras tarefas de aprendizado de máquina. Ele calcula e mantém estimativas adaptativas dos momentos do gradiente, incluindo o primeiro momento (média) e o segundo momento (variância). Essas estimativas são utilizadas para calcular uma correção de gradiente, chamada de momento Adam, que ajusta adaptativamente a taxa de aprendizagem para cada parâmetro. Além disso, o Adam inclui termos de regularização L2 para evitar que os pesos cresçam excessivamente durante o treinamento. Essa abordagem combinada de momentos adaptativos e ajuste de taxa de aprendizagem torna o Adam eficiente, fácil de implementar e eficaz em uma variedade de problemas de otimização (KINGMA; BA, 2014).

Há também uma variação do Adam, que seria o AdamW. Esta variante do otimizador Adam que inclui regularização de peso ponderada diretamente no processo de atualização dos parâmetros, ao contrário do Adam tradicional, onde a regularização é aplicada separadamente após a atualização. Isso ajuda a corrigir problemas de decaimento de pesos em modelos de aprendizado profundo, resultando em melhor desempenho e

capacidade de generalização (ZHOU et al., 2024).

2.3.4 Precisão e Recall

A fim de avaliar o desempenho de um treinamento na arquitetura YOLO, é preciso entender os resultados fornecidos pelo modelo. De acordo com (PADILLA; NETTO; SILVA, 2020), basicamente a YOLO utiliza a métrica chamada *Average Precision* (AP) (“Precisão Média”, em tradução livre). Ela se baseia no conceito de IoU (“Intersection over the Union”, Intersecção sobre a União, em tradução livre), que calcula uma razão entre a interseção da detecção feita pelo algoritmo com relação à marcação (*bounding box*) realizada em cima da área dividida pela união dessas duas áreas (Figura 10). Essa razão poderá ser comparada com um valor pré-estabelecido, o *thresholds*, que será referido por L. A partir desse valor, é possível que no processo de detecção retorne três diferentes resultados na pesquisa pela classe desejada. São eles: Verdadeiro Positivo (VP), Falso Positivo (FP) e Falso Negativo (FN). VP trata-se dos resultados considerados corretos que a rede neural retorna, que seriam todos resultados que $\text{IoU} > L$. FP já seriam os resultados em que $\text{IoU} < L$, que são tidos como incorretos. FN, por sua vez, trata dos resultados totalmente fora do esperado.

$$\text{Recall} = \frac{VP}{VP + FN} \quad (1)$$

$$\text{Precisão} = \frac{VP}{VP + FP} \quad (2)$$

Com esses valores, pode-se calcular os resultados da saída que o treinamento da rede YOLO fornece. Em (1), tem-se o cálculo da *Recall*, que calcula a capacidade da rede de detectar todos os objetos relevantes em uma imagem. Já a Precisão (2), refere-se à capacidade da rede de encontrar apenas resultados relevantes.

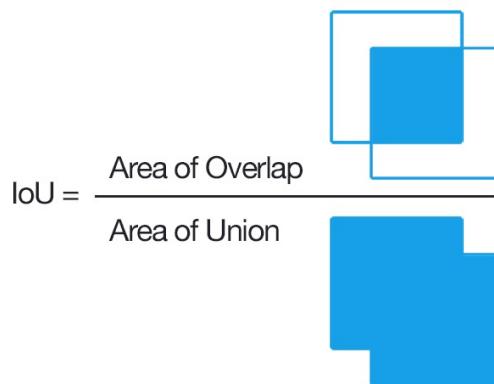


Figura 10 – Cálculo de IoU. (PADILLA; NETTO; SILVA, 2020)

A arquitetura YOLO disponibiliza um *dataset*, ou seja, um banco de imagens e *weights*, comum em todas as versões, chamado de COCO (“Common Objects in Context”,

que em tradução livre seria “Classes Comuns de Objetos”) com classes pré-treinadas e imagens para realização de treinamentos. A partir dele, verificou-se por meio de testes a eficiência das quatro últimas versões da YOLO, a fim de identificar se nas mais recentes houve melhorias significativas em termos de performance e precisão. Na Figura 11, é apresentado o comparativo das versões. Nota-se que a v8, para um menor número de parâmetros que as demais, apresentou uma mAP₅₀₋₉₅ maior que nas versões v5, v6 e v7. Além disso, com relação a velocidade de processamento, a v8 também se sobressai, com maior rapidez no processamento, ao processar maior quantidade de imagens para um mesmo intervalo de tempo (PADILLA; NETTO; SILVA, 2020).

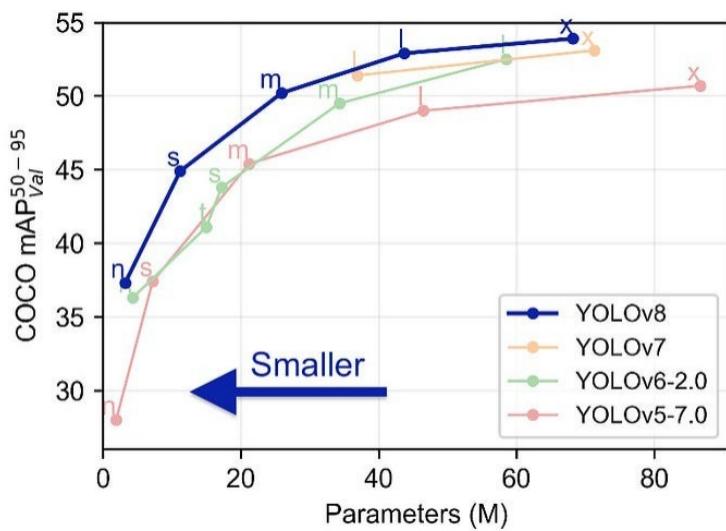


Figura 11 – Aumento de desempenho da precisão média, versão 5 para 8.
(JOCHER; CHAURASIA; QIU, 2023)

2.4 Considerações finais

Neste capítulo, foi apresentado o arcabouço teórico necessário para o entendimento da proposta dessa dissertação. A apresentação das redes neurais, e em específico, o modo que a arquitetura da YOLOv8 sobrepõe-se em termos de eficiência em relação às demais arquiteturas abordadas em outros trabalhos científicos, demonstra o direcionamento assertivo deste trabalho. Além disso, a apresentação da RV como uma disciplina inovadora e muito útil para servir à diversos propósitos dentro da indústria e ciência, corroboram para o entendimento da proposta desta dissertação.

3 Trabalhos Relacionados

3.1 Introdução

Este capítulo apresenta os trabalhos relacionados à utilização de RNC para treinamento de imagens, com e sem variação de parâmetros; a trabalhos que aplicam especificamente a versão 8 do YOLO; aqueles que utilizaram VANT e os que criaram automação para sistemas de RV. O objetivo é encontrar o estado da arte de cada trabalho, destacando as principais contribuições.

No final do capítulo, os estudos são comparados para justificar o sistema abordado neste trabalho.

3.2 Alteração de Parâmetros da RNC

3.2.1 Identificação e Medição de Defeitos em Produtos Automotivos Usando Visão Computacional

A dissertação apresentada por (GONZAGA, 2023) investigou a aplicabilidade e eficiência de um sistema de visão computacional para identificar defeitos visuais no controle de qualidade de peças automotivas reposição. O intuito do trabalho foi de avaliar a possibilidade de automatizar de modo confiável o processo de inspeção e precificação do reparo destas peças.

Para este trabalho, foi utilizado um *dataset* composto por imagens de vidros automotivos, fornecidos por uma empresa do ramo. Os defeitos analisados incluíram bolhas, delaminação, irisação, ostra e grau em vidros, além de manchas em peças como faróis, lanternas e retrovisores. Todos esses defeitos foram identificados nas fotos fornecidas. Assim como na presente dissertação, foi utilizado o software LabelImg para realizar a marcação da ocorrência de cada uma dessas anomalias no conjunto de fotos, como se vê na Figura 12.

Em sua fundamentação, foram entendidas as RNC como ferramentas eficazes para o processamento de imagens. Avaliou-se a rede ResNet (HE et al., 2016), como uma possibilidade de arquitetura para processar as imagens, justamente por se mostrar uma ferramenta poderosa para a identificação de padrões. Para tanto, ela se vale de uma série de camadas residuais com conexões de salto, permitindo que os gradientes fluam mais facilmente através da rede durante o treinamento, mitigando o problema do desaparecimento do gradiente. Esse problema ocorre quando, em redes profundas,



Figura 12 – Exemplo de marcação de foto utilizando labelImg, de um defeito em um vidro (GONZAGA, 2023)

os gradientes se tornam extremamente pequenos durante a retropropagação através das camadas, dificultando a atualização dos pesos nas camadas iniciais e, consequentemente, a aprendizagem adequada da rede. A ResNet supera essa dificuldade utilizando conexões que pulam uma ou mais camadas, somando a entrada diretamente à saída dessas camadas puladas, permitindo a construção de redes muito mais profundas sem a degradação do desempenho (Figura 13). Com isso, a ResNet consegue capturar e aprender padrões complexos presentes nas imagens, resultando em uma melhoria significativa na precisão das tarefas de classificação e reconhecimento de imagens.

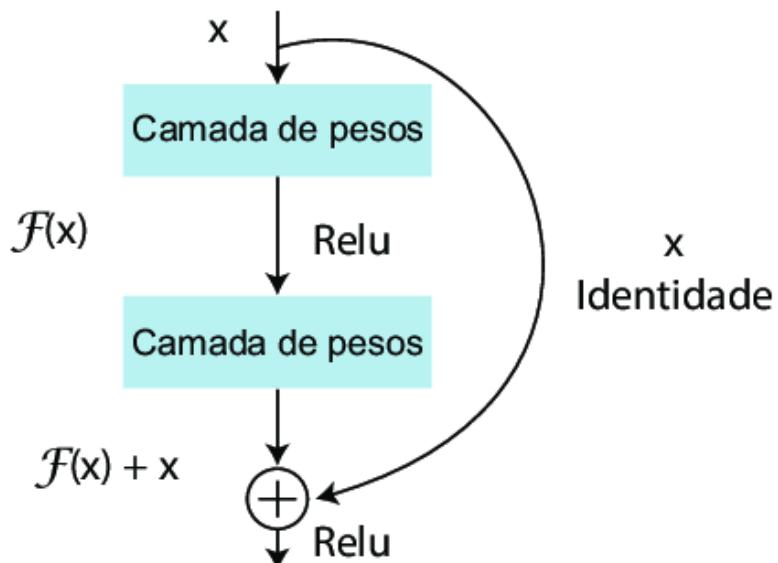


Figura 13 – Bloco de aprendizado da ResNet (HE et al., 2016)

Contudo, a ResNet, devido à sua grande quantidade de camadas e processamento, acaba por ser morosa durante o treinamento. Além disso, sua utilização não é trivial e exige configurações específicas. Deste modo, o trabalho explora as vantagens da YOLO como

alternativa. A YOLO é extremamente rápida, treinando em imagens completas com uma única passagem pelas imagens, o que reduz significativamente o tempo de processamento. Sua arquitetura simples e eficiente permite a detecção de múltiplos objetos em uma única passagem pela rede, tornando o treinamento mais direto e eficaz em termos de recursos computacionais. Além disso, a YOLO é altamente generalizável e menos propensa a erros de *background*, sendo uma solução mais prática e veloz para detecção de objetos.

Para o treinamento, foram analisadas 3.397 imagens com defeitos específicos da rotina da empresa. Destas, 70% foram utilizadas para treinamento e 30% para validação. Todas as imagens foram anotadas manualmente. Em cada uma delas, pelo menos um dos defeitos estudados deveria constar nela para que a imagem fosse colocada para treinamento. Essa parte do processo garantiu que o modelo gerado fosse significativo.

Além disso, foram variados os tamanhos do valor do *batch size* (entre 2 e 32) e testados três otimizadores diferentes: SGD, Adam e AdamW. A YOLOv5 oferece variações da arquitetura para diferentes propósitos. Neste trabalho, foram testadas 10 variações da arquitetura, juntamente com a alteração de parâmetros. Todos os outros hiperparâmetros foram mantidos na configuração padrão. Cada treinamento foi realizado em 300 épocas.

Na Tabela 1, é apresentado o resultado do treinamento, com os respectivos valores, com a variação de parâmetro. Nota-se, disso, que o otimizador SGD, na variação YOLOv5x, com batch size de 8, alcançou a melhor precisão média, com mAP de 0,72921, mostrando com isso a melhor configuração. Com esse treinamento aplicado na identificação de imagens, resultou-se em 83,33% de precisão na especificação correta dos produtos defeituosos, mostrando com isso que a automação foi bem sucedida em seu propósito.

3.3 Aplicação YOLOv8

3.3.1 Comparação de Modelos YOLOv5 e YOLOv8 para Detecção de Objetos em Áreas Rurais Usando Transferência de Aprendizado

O artigo de (DIAS; FIGUEIREDO; MAFRA, 2021), apresenta um estudo comparativo entre duas versões da YOLO, a YOLOv5 e YOLOv8. Nele, avalia-se o desempenho de ambos os modelos na detecção de objetos em cenários característicos de ambientes rurais, onde os desafios incluem a presença de objetos agrícolas, plantações, animais e estruturas. Além disso, é utilizada uma técnica de transferência de aprendizado com modelos pré-treinados para adaptar ambas arquiteturas ao contexto proposto.

Como embasamento teórico, buscou-se um estudo similar ao apresentado, em que foi realizada também uma comparação dos modelos YOLOv5 e YOLOv8 para detecção de veículos e placas em sistemas de transporte inteligentes, usando transferência de aprendizado com dados da plataforma Kaggle (AFONSO et al., 2023). Após uma avaliação

Otimização Adam				
ARQUITETURA	Batch 2	Batch 4	Batch 8	Batch 16
YOLOv6n	0.27538	0.34865	0.38961	0.41800
YOLOv5n	0.34196	0.36812	0.34965	0.44277
YOLOv6s	0.29176	0.29275	0.34995	0.39963
YOLOv5s	0.22021	0.27696	0.30976	0.40907
YOLOv6m	0.22398	0.29198	0.33198	0.42154
YOLOv5m	0.26330	0.31970	0.38044	0.44705
YOLOv6l	0.22940	0.30710	0.34329	0.44475
YOLOv6n	0.30030	0.39497	-	-
YOLOv5x	0.32160	0.38688	-	-
Otimização AdamW				
ARQUITETURA	Batch 2	Batch 4	Batch 8	Batch 16
YOLOv6n	0.42883	0.43552	0.49657	-
YOLOv5n	0.48022	0.48880	0.57565	-
YOLOv6s	0.44942	0.49605	0.58261	-
YOLOv5s	0.43433	0.52185	0.55821	-
YOLOv6m	0.51829	0.53139	0.60474	-
YOLOv5m	0.46789	0.50103	0.58947	-
YOLOv6l	0.46769	0.50310	0.58947	-
YOLOv6n	-	-	-	-
YOLOv5x	-	-	-	-
Otimização SGD				
ARQUITETURA	Batch 2	Batch 4	Batch 8	Batch 16
YOLOv6n	0.57286	0.55400	0.65654	-
YOLOv5n	0.59021	0.60678	0.60235	-
YOLOv6s	0.60723	0.61912	0.61684	-
YOLOv5s	0.67145	0.70897	0.71185	-
YOLOv6m	0.60228	0.71413	0.71037	-
YOLOv5m	0.61179	0.71564	0.70514	-
YOLOv6l	0.62377	0.71564	0.70514	-
YOLOv6n	0.71112	0.71067	-	-
YOLOv5x	0.70527	0.71120	-	-

Tabela 1 – Tabela comparativa de desempenho de modelos YOLO com diferentes otimizações e tamanhos de batch.

abrangente, concluíram que o YOLOv8 superou ligeiramente o YOLOv5, além de ter um tempo de treinamento menor.

Já no trabalho citado de (SILVA et al., 2018), foi estudada a detecção e contagem de plantas usando Inteligência Artificial (IA), aplicando modelos de visão computacional para detectar e contar eucaliptos em plantações. O modelo R-CNN Resnet 101 alcançou 95% de precisão com um tempo de inferência de 578 milissegundos por imagem, destacando-se como o mais promissor para o desenvolvimento de software automatizado na silvicultura.

Por fim, citando (GOMES, 2022), foi explorada a detecção de objetos com a

arquitetura YOLO, comparando várias versões do modelo usando o conjunto de dados COCO. O YOLOv5 se destacou, alcançando uma acurácia de 67,9% na métrica *Mean-average Precision* (mAP), superando significativamente as versões anteriores.

A literatura indica que a detecção de objetos tem atraído muita atenção, com muitos estudos focando na identificação e avaliação dos melhores modelos conforme o cenário de aplicação. Este trabalho é similar ao estudo (AFONSO et al., 2023) ao comparar YOLOv5 e YOLOv8, mas difere na área de aplicação.

Em sua conceituação, reforça-se a definição de objeto, a entidade desejável de ser reconhecida pela RNC. É ele qualquer forma visual reconhecível em uma imagem, como carros, pessoas, animais, ou prédios. Cada objeto será representado por uma classe por sua vez é representado por uma classe. Assim, a principal tarefa da detecção de objetos de uma RNC é identificar e localizar a presença dessas classes em imagens ou vídeos.

A metodologia deste trabalho envolveu a coleta de 452 imagens de 1800 pixels, utilizando um VANT. As classes definidas foram: cafezal, milharal, soja, estrada, casa, carro e pasto. As imagens foram anotadas usando a plataforma Roboflow para criar caixas delimitadoras. O pré-processamento incluiu orientação automática e redimensionamento das imagens para 640 x 640 pixels. O aumento de dados foi realizado com rotações aleatórias e adição de ruído, ampliando o conjunto para 1100 imagens. O *dataset* foi dividido em treinamento (960 imagens), validação (95 imagens) e teste (45 imagens), e exportado nos formatos *YOLOv5 PyTorch* e *YOLOv8*.

No estudo, foi utilizado para complementar, pesos pré-treinados no conjunto MS COCO, com ajuste fino dos modelos para o novo conjunto de dados. A taxa de aprendizado foi de 0,0001 para YOLOv8x e 0,00001 para YOLOv5x, com um batch size igual a 128. Assim como nos demais trabalhos, foram adotadas as métricas de precisão, *recall* e mAP.

Os resultados deste estudo demonstraram que o modelo YOLOv8x superou o YOLOv5x em termos de precisão e eficiência na detecção de objetos em áreas rurais. Especificamente, o YOLOv8x alcançou maior acurácia nas predições e menor tempo total de inferência, apesar de necessitar de um tempo de treinamento ligeiramente superior.

O YOLOv8x mostrou-se mais eficaz na detecção de objetos, atingindo um mAP de 0,767 (Figura 14). Além disso, apresentou maior confiança nas detecções, com um tempo médio de inferência de 15,9 ms. Já a YOLOv5x, por sua vez, atingiu um mAP de 0,735 (Figura 15), com um desempenho inferior em comparação ao YOLOv8x tanto em precisão quanto no tempo total de inferência. O tempo médio de inferência para o YOLOv5x foi de 17,2 ms.

Este trabalho investigou a importância da detecção de objetos em imagens de áreas rurais, comparando os modelos YOLOv5 e YOLOv8. A detecção precisa de objetos é crucial em várias aplicações agrícolas, de monitoramento e preservação ambiental, aumentando a

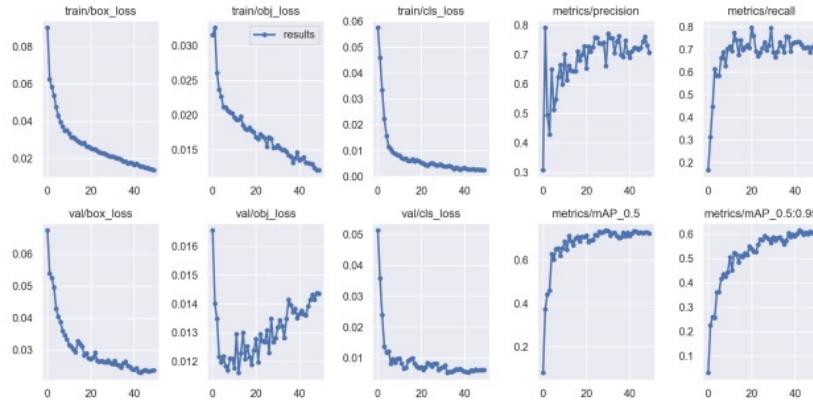


Figura 14 – YOLOv5 (DIAS; FIGUEIREDO; MAFRA, 2021)

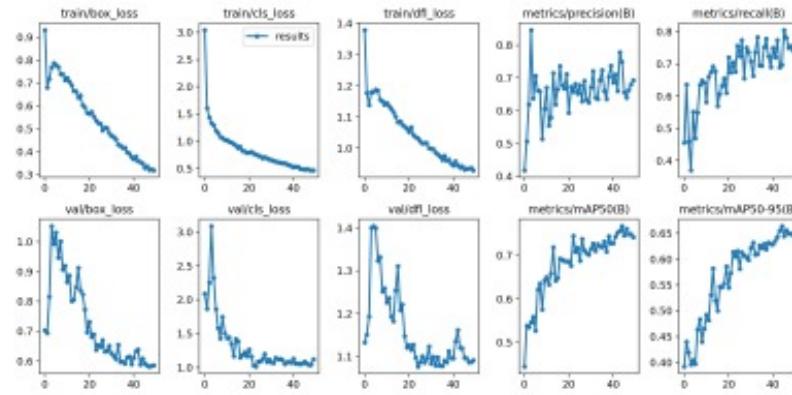


Figura 15 – YOLOv8 (DIAS; FIGUEIREDO; MAFRA, 2021)

eficiência, produtividade e sustentabilidade nessas áreas. Os modelos YOLOv5 e YOLOv8 destacaram-se na detecção de objetos em tempo real, com o YOLOv5 sendo eficiente no treinamento e o YOLOv8 se sobressaindo pela precisão na detecção.

3.3.2 UAV-YOLOv8: A Small-Object-Detection Model Based on Improved YOLOv8 for UAV Aerial Photography Scenarios

O artigo de (WANG et al., 2023) se inicia apresentando o crescente uso de VANTs para diversas aplicações da ciência e engenharia. Contudo, devido a altura do voo, o uso de suas imagens para captura de objetos específicos delas mostra-se um desafio, uma vez que a proporção existente entre objeto e a foto como um todo é muito pequena. Além disso, realizar o processamento dessas imagens em tempo real, de maneira embarcada em um VANT, dada as limitações de hardwares desses equipamentos, mostra-se outro desafio. Disto, entende-se que a primeira motivação do trabalho está em melhorar o desempenho da detecção de objetos considerando os recursos limitados da plataforma de hardware.

Para a tarefa de detecção, como nos demais trabalhos citados nessa dissertação, recorre-se às RNC. Entre elas, a principal diferença está na quantidade de estágios de

deteção dos algoritmos, que são aqueles que fazem a detecção em dois estágios, como a Fast R-CNN, e também aqueles que a realizam em apenas um, que é o caso da YOLO (GIRSHICK, 2015).

A alteração da arquitetura da RNC influí diretamente no resultado desejado. Contudo, a simples troca de um algoritmo para outro não satisfaz completamente a necessidade de aumentar a acurácia desejada, levando à busca por alterações na estrutura da rede para obter mais interessantes.

Assim, o trabalho de (WANG et al., 2023) propõe um modelo de detecção de objetos baseado em fotografias obtidas por meio de VANTs, chamado UAV-YOLOv8, que se vale da YOLOv8 como rede *backbone*. No geral, essa nova arquitetura melhora significativamente a precisão e a eficiência da detecção de objetos em imagens capturadas por UAVs, mantendo um baixo consumo de recursos, o que o torna adequado para plataformas de UAVs com capacidades limitadas. Ele é especialmente otimizado para enfrentar os desafios das imagens aéreas, como a detecção de pequenos objetos em fundos complexos.

A oitava versão da YOLO notabiliza-se por ser o estado da arte na ciência de detecção de objetos. Sua estrutura utiliza a arquitetura modificada CSPDarknet53 como *backbone*, com módulos C2f e SPPF para otimizar a extração e processamento das características de imagem. O *neck* adota a estrutura PAN-FPN, que melhora a fusão de informações posicionais e semânticas, tornando o modelo mais leve e eficiente. Por fim a head desacoplada usa ramos separados para classificação e regressão de caixas, melhorando a precisão e a robustez da detecção, sem o uso de âncoras (REDMON; FARHADI, 2018).

Neste artigo, para melhorar sua eficácia nesses cenários, foi proposta uma otimização do modelo utilizando a função de perda WIoU v3, que aprimora a generalização; o mecanismo de atenção BiFormer, que foca em características de alta relevância; e o módulo de processamento de características FFNB, que melhora a fusão de características de diferentes escalas. Essas melhorias resultaram no UAV-YOLOv8, que expande a detecção de 3 para 5 escalas, aprimorando significativamente a detecção de pequenos objetos.

A experimentação do novo modelo foi proposta de forma a avaliar a precisão de detecção de objetos pequenos e a eficiência de recursos do modelo proposto, UAV-YOLOv8s. Como *dataset*, o estudo fez uso do VisDrone2019, que inclui fotografias aéreas obtidas por UAVs em várias cidades da China, capturando diferentes cenários e condições de iluminação. Escolheu-se o VisDrone2019 devido à sua riqueza de informações e diversidade. Foi desenvolvido pela Universidade de Tianjin e inclui uma variedade de categorias de objetos de detecção e distribuições variadas de objetos, abrangendo tanto cenários com alta quanto baixa densidade de objetos, e fotos tiradas durante o dia e a noite.

Assim, para a detecção de objetos em fotos capturadas por VANTs, em se tratando de objetos muitos, fundos complexos e recursos de hardware limitados, há um desafio em

manter a alta a precisão dos modelos. O UAV-YOLOv8, baseado no YOLOv8, é proposto para otimizar a detecção, introduzindo a função de perda WIoU v3 e o mecanismo de atenção BiFormer, melhorando o foco do modelo e reduzindo a perda de detecção de objetos pequenos. O modelo aprimorado alcança um aumento médio de 7,7% (Figura 16) na precisão sem aumentar seu tamanho, mas a adição de duas camadas de detecção aumenta a complexidade e o tempo de computação. Comparado com outros modelos mainstream, o UAV-YOLOv8 demonstrou desempenho superior, de modo geral (Figura 17). A precisão ainda é baixa para objetos muito pequenos, como bicicletas, e futuras pesquisas seriam necessárias na otimização dessa precisão, podendo elas ficarem na confecção do *dataset* para melhores resultados, alteração da arquitetura, entre outras abordagens.

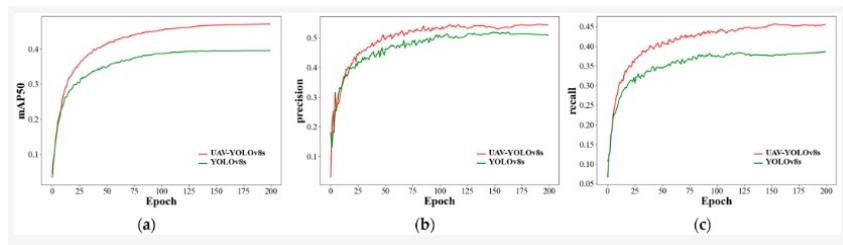


Figura 16 – Comparaçāo da YOLOv8 com a rede modificada

Models	Precision/%	Recall/%	mAP0.5/%	mAP0.5:0.95/%	Model Size/MB	Detection Time/ms	Parameter/ 10^6
YOLOv8n	43.8	33.0	33.3	19.3	6.6	4.2	3.0
YOLOv8s	50.9	38.2	39.3	23.5	22.5	7.7	11.1
YOLOv8m	56.0	42.5	44.6	27.1	49.6	16.6	25.9
YOLOv8l	57.5	44.3	46.5	28.7	83.5	25.6	43.7
Ours	54.4	45.6	47.0	29.2	21.5	19.5	10.3

Figura 17 – Comparaçāo da YOLOv8 com a rede modificada

3.4 Considerações Finais

A Tabela 2 apresenta um resumo de todos os trabalhos relacionados descritos neste capítulo, considerando os seguintes temas:

- *Treinamento utilizando YOLOv8*: Utilização de RNC para realizar treinamento em conjuntos de imagens;
- *Alteração de Parâmetros da RNC para treinamento*: Variação de parâmetros como batch size e otimizadores para treinamento;
- *Utilização de VANT*: Coleta de fotos utilizando VANTS;
- *Subestações de Energia*: Coleta, treinamento e inserção voltada para subestações de energia;

- *Automação de Inserção de RV:* Construção de sistema de inserção automática de objetos em um sistema de RV.

Tabela 2 – Resumo comparativo dos trabalhos relacionados

Trabalhos Relacionados	Alteração de Parâmetros da RNC	Treinamento utilizando YOLOv8	Utilização de VANT	Subestações de Energia	Automação de Inserção de RV
(GONZAGA, 2023)	✓	✓	✗	✗	✗
(DIAS; FIGUEIREDO; MAFRA, 2021)	✓	✓	✗	✗	✗
(WANG et al., 2023)	✓	✓	✓	✗	✗

Pela análise da Tabela 2, entende-se que não há ainda um trabalho que reúna todos os tópicos apresentados. É fato que há diversos trabalhos que exploram a detecção de objetos em diversos cenários, utilizando diversas versões da YOLO, inclusive utilizando VANT, para obtenção de imagens. Contudo, no contexto de subestações de energia e identificação de seus equipamentos, há uma lacuna de contribuições. Naturalmente, isso mostra o quanto prolífico esta dissertação se mostra, ao unir todas essas áreas de estudo num propósito ainda não explorado.

No próximo capítulo, serão detalhados os materiais e métodos utilizados para a solução proposta.

4 Materiais e Métodos

4.1 Introdução

Pesquisas relevantes ao tema de treinamento de RNC com imagens aéreas, RV e temas correlatos foram abordados no capítulo anterior, e se tornaram subsídio para compreender o estado da arte dessas temáticas. A partir dessa coletânea de trabalhos, foi traçada a metodologia aqui adotada.

Neste capítulo, portanto, apresenta-se os materiais utilizados nesta pesquisa, assim como os métodos executados para treinamento das imagens coletadas e a abordagem implementada na automação do sistema de RV.

4.2 Base de Dados

4.2.1 Captura das Fotos

Para o treinamento da rede neural YOLOv8, é necessário que sejam fornecidas imagens, ou seja, uma base de dados relevantes com objetos a serem identificados. Neste estudo, foram coletadas fotos aéreas por meio de VANT de modelo DJI Mavic 2, que é reconhecido por sua capacidade de captura de alta qualidade e manobrabilidade. Todas imagens capturadas foram obtidas no sobrevoo de duas subestações de energia, ambas no Brasil. A primeira subestação é localizada em Porto Velho, no estado de Rondônia, de coordenadas geográficas: -8.914705, -63.957887. A segunda, em Araraquara; coordenadas: -21.832444, -48.347566. No total, foram capturadas 1257 fotos, sendo 548 em Porto Velho e 709 em Araraquara. Todas as imagens foram capturadas em alta resolução e com dimensões variadas, variando entre 4000x3000 até 8000x6000 pixels.

De todo o conjunto de fotos coletadas, apenas 411 fotos foram selecionadas para serem processadas na RNC. Além disso, destas, 60% foram alocadas para o conjunto de treinamento, enquanto 20% para os conjuntos de teste e 20% de validação. De acordo com (SAMRAT, 2024), esta forma de divisão do conjunto de imagens produzirá resultados mais parciais do que uma divisão que preconiza um conjunto de validação e de teste menor.

Todo procedimento de coleta de imagens foi realizado por integrantes do GRVA, Grupo de Pesquisa em Realidade Virtual e Aumentada (GRVA, 2024), ligado à Faculdade de Engenharia Elétrica, pertencente à Universidade Federal de Uberlândia (UFU). Esta base de fotos, além de servir para esse trabalho, também foram subsídio para diversos outros projetos do laboratório, como (NETO et al., 2024) e (JÚNIOR et al., 2024).

As imagens são capturadas de modo a registrar o máximo de equipamentos possível, cobrindo toda a área das subestações. Todas as fotos são tiradas verticalmente em relação ao solo, garantindo uma visão abrangente e detalhada das instalações das subestações. Devido ao movimento do VANT, os objetos na foto são representados com pequenas variações de angulação em relação ao solo. Em Porto Velho, as foram fotos capturadas no dia 28 de junho de 2023 (Figura 18, à esquerda). Por sua vez, em Araraquara, as capturas foram realizadas dia 13 de junho de 2023 (Figura 18, à direita).



Figura 18 – Captura de foto em Porto Velho (à esquerda) e Araraquara (à direita), onde se notam quatro reatores de núcleo de ar na parte inferior.

4.2.2 Seleção das fotos

Conforme o trabalho de (SÁ et al., 2024), o treinamento eficiente com a RNC envolve o fornecimento de imagens relevantes da rede para que haja consistência no caminho desenvolvimento pelo processamento do algoritmo, de forma a conseguir o melhor resultado. Desta forma, foi realizada a seleção das imagens para o treinamento, selecionando apenas aquelas que contém ao menos um objeto de interesse, que neste trabalho, seria o reator de núcleo de ar.

O conjunto de treinamento é utilizado para ajustar os parâmetros internos do modelo. Durante essa fase, a RNC aprende a identificar padrões e características específicas das imagens que correspondem aos objetos de interesse. Naturalmente, objetivando minimizar a função de perda, que mede o quanto bem o modelo está se saindo na tarefa de detecção de objetos.

As fotos usadas para validação, por outro lado, são empregadas para monitorar o desempenho do modelo durante o treinamento. Elas permitem a realização de ajustes nos hiperparâmetros do modelo, como a taxa de aprendizado e o número de camadas. A validação contínua ajuda a prevenir o *overfitting*, que ocorre quando o modelo se ajusta demasiadamente aos dados de treinamento e perde a capacidade de generalizar para novos dados. Um desempenho consistente no conjunto de validação indica que o modelo está aprendendo de maneira robusta e equilibrada.

Finalmente, o conjunto de teste é utilizado após o treinamento e a validação para avaliar a capacidade de generalização do modelo em dados nunca antes vistos. Este conjunto não influencia o processo de treinamento, mas fornece uma estimativa imparcial do desempenho real do modelo. Um bom desempenho no conjunto de teste sugere que o modelo pode ser confiável para identificar objetos de interesse em situações práticas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Portanto, a divisão dos dados em conjuntos de treinamento, validação e teste é uma prática fundamental em aprendizado profundo, garantindo que o modelo YOLO, ou qualquer outro modelo de detecção de objetos, seja eficiente, robusto e generalizável.

4.2.3 Rotulação das fotos

Toda RNC necessita de dados rotulados para aprender a identificar e classificar corretamente os objetos em imagens. Utilizar um Software de marcação, como o LabelImg, é essencial nesse processo, pois permite que seja rotulado imagens que irão alimentar a rede, definindo nelas regiões de interesse e associando essas regiões a classes específicas. Nesta dissertação, a classe de interesse trata-se do reator de núcleo de ar. Sem dados rotulados, a RNC não teria a orientação necessária para distinguir diferentes objetos, comprometendo a sua capacidade de realizar previsões.

Para realizar as marcações das fotos selecionadas, foi utilizado o LabelImg, uma vez que trata-se de uma ferramenta de anotação gráfica de código aberto, de fácil instalação e manuseio. Sua interface é intuitiva, pode-se carregar imagens, desenhar caixas delimitadoras ao redor dos objetos de interesse e atribuir rótulos a esses objetos (Figura 19). As anotações geradas são salvas em um formato compatível, como XML ou TXT, já preparados para serem submetidos ao processamento da YOLO. É essencial que anotações feitas com o LabelImg sejam significativas, pois influenciam diretamente o desempenho do modelo treinado.

4.3 Treinamento

4.3.1 Materiais

Todo o treinamento foi realizado em um computador pessoal, portador do Sistema Operacional Windows 11 de 64 bits, equipado com um processador x64 com 24GB de memória RAM e um processador 11th Gen Intel(R) Core (TM) i7-1165G7@2.80GHz. Os scripts da YOLOv5, YOLOv6, YOLOv7 e YOLOv8, assim como o LabelImg, foram executados no ambiente de desenvolvimento PyCharm (PyCharm, 2023).

Para a implementação da automação, foi utilizado o software Unity (Unity Technologies, 2024). O Unity é uma plataforma amplamente utilizada para o desenvolvimento de

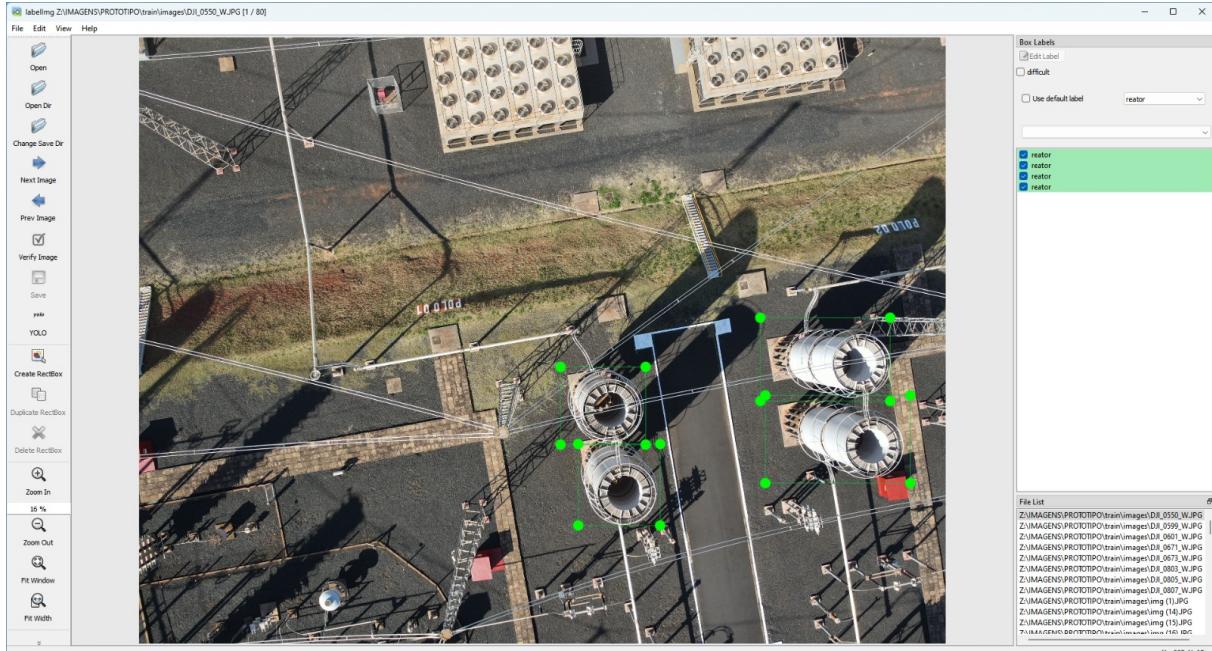


Figura 19 – LabelImg realizando a marcação de reatores de núcleo de ar

jogos e aplicações interativas 3D. Ele oferece um ambiente robusto e flexível, com uma vasta gama de ferramentas e recursos que facilitam a criação de experiências interativas. Além disso, o Unity permite integração com diversos outros softwares e sistemas, proporcionando um fluxo de trabalho eficiente para projetos de automação e simulação.

4.3.2 Variação dos Parâmetros

4.4 Avaliação do Treinamento

Os resultados de desempenho obtidos das RNCs foram estruturados da seguinte forma: compararam-se as métricas de mAP, mAP50, mAP50-95 e *recall* das versões YOLOv5, YOLOv6, YOLOv7 e YOLOv8, com variação do batch size de 2 a 16, utilizando o otimizador SGD. Esta primeira parte da experimentação teve como objetivo comparar todas as versões mais recentes e anteriores à YOLOv8, conforme realizado em seu artigo de apresentação (JOCHER; CHAURASIA; QIU, 2023), com a finalidade de demonstrar a superioridade desta arquitetura.

Conforme discutido na Fundamentação Teórica deste trabalho, a YOLOv8 foi proposta como um modelo superior às suas três versões anteriores. No artigo de apresentação, comparou-se a taxa de sucesso da versão 8 em relação às versões 5, 6 e 7, utilizando um conjunto de imagens comuns (JOCHER; CHAURASIA; QIU, 2023). Para validar essa afirmação, foi realizada uma experimentação similar, mantendo a configuração padrão de todos os hiperparâmetros, com exceção do batch size, que foi ajustado para verificar seu efeito no treinamento, seguindo o estudo de (GONZAGA, 2023).

Por configuração padrão da rede neural YOLO, entende-se a utilização do otimizador SGD, uma taxa de aprendizado de 0,01 e a entrada de imagens no formato 640x640 pixels.

Na segunda etapa, os mesmos parâmetros de desempenho foram avaliados, desta vez apenas para a YOLOv8, também com variação do *batch* de 2 a 16, mas com diferentes otimizadores: *Adam*, *AdamW* e *SGD*, conforme abordado no trabalho (GONZAGA, 2023). O objetivo dessa experimentação foi encontrar a configuração de melhor desempenho para a tarefa de identificação de reatores.

Por fim, a performance dos modelos foi avaliada na identificação de equipamentos utilizando um conjunto de fotos não presentes no treinamento, ou seja, o conjunto de teste. Esse procedimento visou verificar, na prática, a precisão real de cada configuração testada. A avaliação foi inspirada no trabalho de (GONZAGA, 2023), que aplicou o modelo com o maior mAP no conjunto de teste, comparando o número de detecções realizadas com o número esperado, para assim medir a efetividade do modelo em cenários reais.

4.5 Considerações Finais

Neste capítulo, foram descritas as especificações do sistema desenvolvido, com ênfase nos seus principais requisitos. O próximo capítulo trará informações detalhadas sobre a implementação, além de um aperfeiçoamento das especificações iniciais.

5 Detalhes da Implementação

5.1 Introdução

Este capítulo detalha a implementação do sistema de treinamento e inserção automática desenvolvida para a geração de ambientes virtuais a partir de imagens. A aplicação foi concebida para automatizar o processo de geração de modelos tridimensionais a partir das imagens processadas. O objetivo principal da implementação é oferecer uma solução eficiente para qualquer usuário que deseja criar ambientes virtuais, possa fazê-lo de maneira rápida e intuitiva, com base em imagens fornecidas.

A aplicação, desenvolvida dentro da plataforma Unity, permite ao usuário carregar imagens através de um menu gráfico e, a partir delas, o sistema realiza automaticamente o reconhecimento dos objetos presentes dentro da foto fornecida e os insere no ambiente virtual. Essa abordagem reduz significativamente a necessidade de intervenção manual, tornando o processo de criação de ambientes 3D mais eficiente. A automação deste processo é particularmente útil para usuários com pouca experiência técnica na geração de ambientes virtuais, permitindo-lhes focar na interatividade do ambiente ambiente sem se preocupar com os detalhes gráficos.

5.2 Estrutura geral da aplicação

A estrutura da aplicação desenvolvida neste trabalho está representada na Figura 20. A arquitetura do sistema é composta por quatro elementos principais que operam em conjunto para realizar a tarefa de criação de ambientes virtuais. O fluxo do processo inicia-se com o treinamento do modelo YOLO, que gera os pesos da RNC utilizados no reconhecimento de objetos nas imagens fornecidas.

O segundo componente importante da estrutura é a interface gráfica do usuário (*Graphical User Interface - GUI*) no Unity, que pode ser visualizada na Figura 21. Esta interface foi cuidadosamente projetada para facilitar a identificação e modelagem das cenas virtuais. O script que compõe a GUI é integrado diretamente ao Unity, proporcionando ao usuário uma ferramenta intuitiva para selecionar imagens e iniciar o processo de reconhecimento. A GUI comunica-se com uma *API* que aplica o modelo treinado às imagens fornecidas, enviando uma requisição *HTTP* com os dados necessários.

Conforme ilustrado no diagrama, a *API* recebe o caminho da imagem, aplica os pesos treinados da RNC, e retorna ao Unity a quantidade de objetos identificados e suas respectivas classes. No caso, o protótipo foi desenvolvido para trabalhar com apenas uma

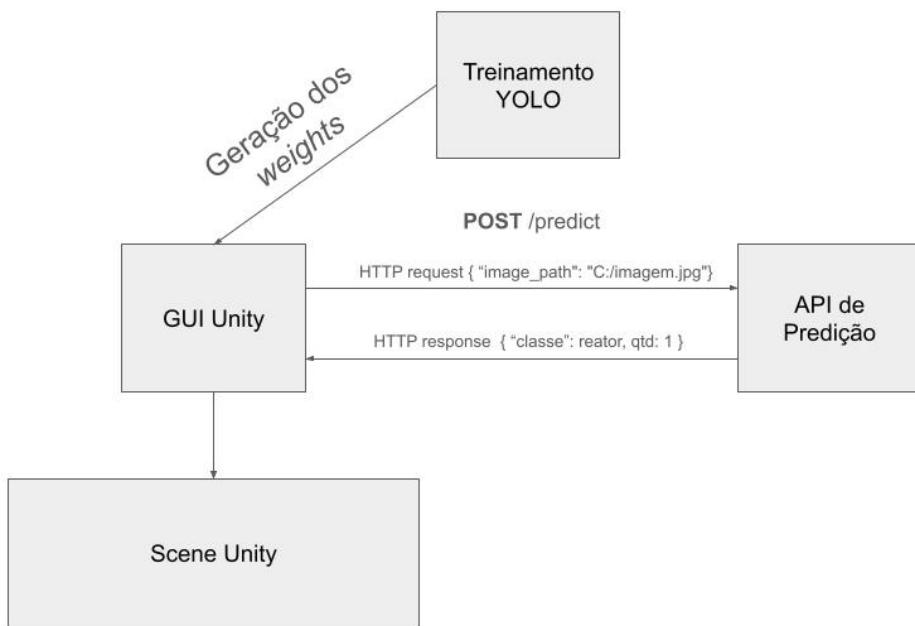


Figura 20 – Estrutura geral da Aplicação

classe, abrindo espaço para trabalhos futuros, assim como ele também não implementa a localização espacial dos objetos, os modelos virtuais gerados são dispostos de forma uniforme na cena do Unity após o processamento da imagem. Isso significa que, apesar da precisão na detecção dos objetos, ainda há espaço para melhorias no que diz respeito à colocação exata dos modelos dentro do ambiente virtual, o que será uma possível melhoria em versões futuras do sistema.

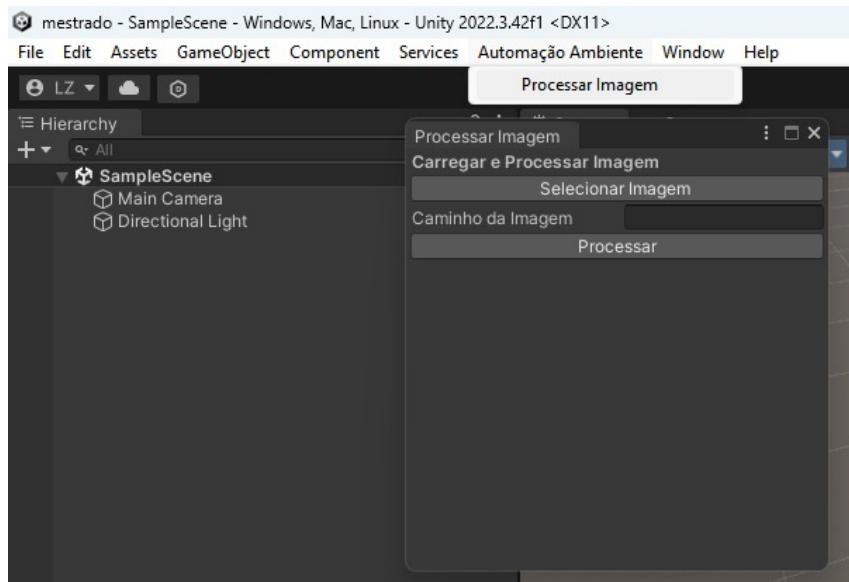


Figura 21 – Interface para o usuário.

5.3 Treinamento com a YOLO

O treinamento com o YOLO segue um padrão comum entre suas diferentes versões, mas cada versão possui características específicas em termos de pacotes necessários e processos de instalação.

5.3.1 Treinamento com a YOLOv5, YOLOv6 e YOLOv8

A YOLOv5¹, assim como a YOLOv6² e a YOLOv8³, foram desenvolvidas pela empresa *Ultralytics*, que também dá nome à biblioteca utilizada para a configuração do projeto. Para utilizar essa versão da arquitetura, foi necessário clonar o projeto para a máquina onde o treinamento foi executado. Além disso, foi utilizada a versão 3.1 do *Python*.

Em aplicações desenvolvidas em *Python*, é altamente recomendada a criação de um ambiente virtual para garantir que as bibliotecas sejam instaladas corretamente e que o ambiente não entre em conflito com outras dependências do sistema. Esse ambiente virtual isola o projeto e suas dependências de outros projetos e bibliotecas instaladas globalmente. As dependências necessárias para o projeto estão especificadas no arquivo *requirements.txt*.

Para configurar o ambiente virtual e instalar as dependências, foram executados no terminal dentro da pasta do projeto os seguintes comandos:

```
python -m venv ambiente_virtual
.\ambiente_virtual\Scripts\activate
pip install -r requirements.txt
```

Após isso, foi desenvolvido a estrutura básica do script de treinamento de todas essas versões da YOLO. A Figura 22 apresenta os scripts para o treinamento da YOLOv5, na imagem superior, e YOLOv6 na imagem inferior. Basicamente, a primeira linha designa a importação da biblioteca da *Ultralytics*. Logo em seguida, é especificado a arquitetura que será utilizada. Para a quinta versão, foi designado ‘YOLOv5n.pt’; para sexta versão, foi especificado ‘YOLOv5n.pt’. A última linha por fim, determina a modelagem do treinamento, que será especificada na função *train()*, do objeto *model*, do tipo *YOLO()*. Foram especificados três parâmetros diferentes. O primeiro, *data* recebe o arquivo de configuração “*config.yaml*”, em que é especificado a quantidade de classes a serem treinadas (neste caso, apenas uma, que é a classe “reator”), e a localização dos diretórios contendo as imagens e marcações para treinamento, validação e teste. Em seguida, há o parâmetro *epochs*, que configura a quantidade de iterações que a RNC irá realizar. Neste caso, foi configurado para 300 iterações. O parâmetro *batch*, naturalmente, configura o valor de

¹ <<https://github.com/ultralytics/yolov5>>

² <<https://github.com/meituan/YOLOv6>>

³ <<https://github.com/ultralytics/ultralytics>>

batch, que foi variado de 2 a 16. Por fim, *optimizer* determina o otimizador, que na comparação inicial foi utilizado no valor de ‘SGD’, aplicando o otimizador homônimo.

```

1  from ultralytics import YOLO
2
3  # Arquitetura de Treinamento
4  model = YOLO('yolov5n.pt')
5
6  #Configuração da RNC
7  results = model.train(data="config.yaml",
8                         epochs=300,
9                         batch=16,
10                        optimizer='SGD')
11
12 from ultralytics import YOLO
13
14 # Arquitetura de Treinamento
15 model = YOLO('yolov6n.pt')
16
17 #Configuração da RNC
18 results = model.train(data="config.yaml",
19                         epochs=300,
20                         batch=16,
21                        optimizer='SGD')
```

Figura 22 – Demonstração dos Scripts de Treinamento das versões YOLOv5, em cima, e YOLOv6, em baixo.

5.3.2 Treinamento com a YOLOv7

De todas as versões da YOLO, apenas a YOLOv7⁴ foi executada via terminal. Todo o preparo do ambiente foi igual às demais versões. Fez-se a cópia do repositório remoto no computador que a executou, utilizou-se a mesma versão do *Python*, e foi gerado um ambiente virtual para aquisição das dependências. Dentro do diretório da YOLOv7, ao contrário de executar um *script* por meio de um ambiente de desenvolvimento, fez a execução do comando demonstrado na Figura 23, diretamente via terminal. O início da instrução se inicia com *python*, para indicar qual o tipo de aplicação executada. Em seguida, é especificado o arquivo da classe *train.py*, que é a responsável por toda estruturação do treinamento da RNC. A parametrização é semelhante às demais, com *batch-size* determinando o valor de *batch*; *epochs*, épocas; e *data*, a configuração das classes e diretórios das imagens; *weights*, indicando a arquitetura utilizada que seria ‘*yolov7.pt*’. Diferindo das outras, há a especificação do tipo de processador por meio de *device*, que no caso foi escolhido CPU.

⁴ <<https://github.com/WongKinYiu/yolov7>>

```
Terminal: Local × + ▾
PS Z:\Mestrado\scripts-computer-vision\TESTE-YOLO-V7\yolov7
> python train.py --device CPU --batch-size 16 --epochs 300
--data config.yaml --weights 'yolov7.pt'
```

Figura 23 – Comando de Execução da YOLOv7

5.3.3 Variação dos otimizadores com a YOLOv8

Seguindo a mesma metodologia empregada no trabalho de (GONZAGA, 2023), foi variado o parâmetro referente ao otimizador. A estrutura do treinamento é igual à anteriormente referida para essa versão, apenas o valor do parâmetro do otimizador que será alterado. Os testes com o otimizador *Adam*, receberam a configuração *optimizer='Adam'*, para *Adam*; *optimizer='AdamW'*, para *AdamW*; e *optimizer='SGD'* para *SGD*. Na Figura 24 é demonstrado a configuração desse treinamento.

```
1  from ultralytics import YOLO
2
3  # Arquitetura de Treinamento
4  model = YOLO('yolov8n.pt')
5
6  #Configuração da RNC
7  results = model.train(data="config.yaml",
8                      epochs=300,
9                      batch=8,
10                     optimizer='Adam')
```

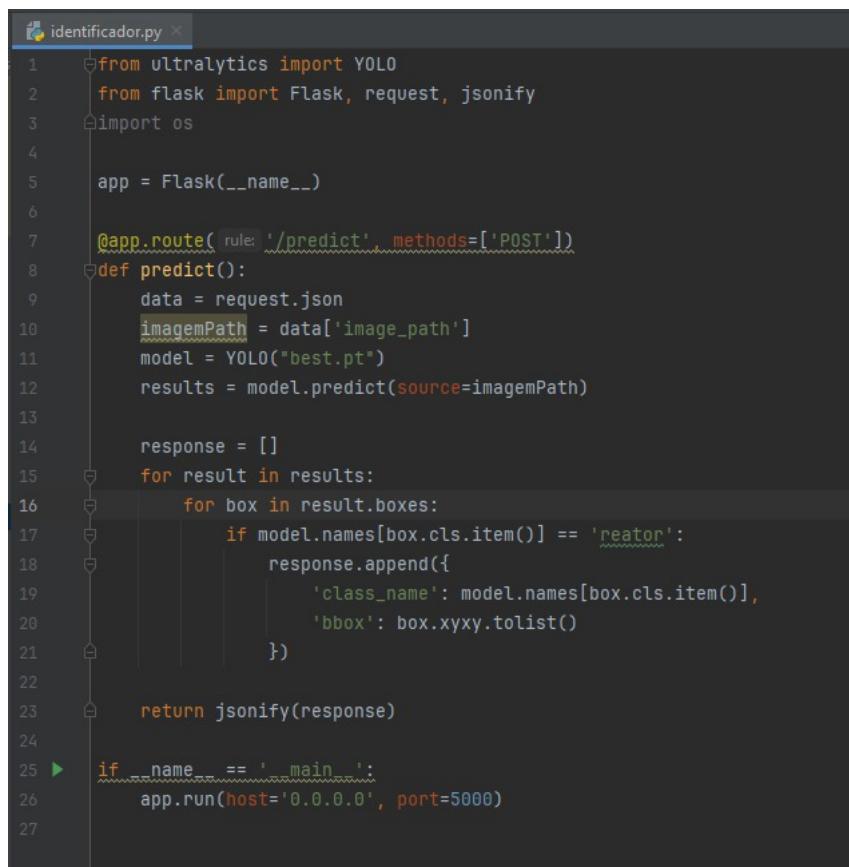
Figura 24 – Demonstração do Script de Treinamento da YOLOv8, com aplicação do otimizador *Adam* e com *batch* configurado para 8

Diferentemente dos demais, a fim de avaliar a melhor configuração da YOLOv8, para cada variação do otimizador foi realizado a variação do *batch*, nos valores de 2, 4, 8 e 16. Com isso, pode-se comparar o efeito dos otimizadores e tamanho de lote nas diversas configurações dessa arquitetura.

5.4 API de Identificação

A construção da aplicação responsável pelo processamento das imagens enviadas pela Unity envolveu a criação de um *endpoint* que permite a solicitação do serviço de processamento por aplicações externas. Sua estrutura é demonstrada na 25. O Unity envia uma requisição *HTTP* para o endereço "*localhost:5000/predict*", incluindo o caminho da

imagem no corpo da requisição. Ao receber a requisição, o *endpoint* decodifica seu corpo para extrair o endereço da localização da imagem a ser processada, que será armazenado na variável *imagePath*. Na linha seguinte, tem-se a atribuição *model = YOLO('best.pt')*. O arquivo *best.pt* carrega as informações relativas ao melhor treinamento da experimentação para ser aplicado no processo de predição. Na linha seguinte, aplica-se os pesos responsáveis pela predição à imagem especificada no caminho. Realizado esse processo, a iteração que é feita logo após, trata de extrair do arquivo resultante a localização de cada reator identificado. Passa-se essa informação por coordenadas de *bounding box*, idênticas ao processo de marcação. Além de poder contabilizar a quantidade de reatores detectados, é possível localizar visualmente os objetos detectados na foto processada.



```

1  from ultralytics import YOLO
2  from flask import Flask, request, jsonify
3  import os
4
5  app = Flask(__name__)
6
7  @app.route('/predict', methods=['POST'])
8  def predict():
9      data = request.json
10     imagePath = data['image_path']
11     model = YOLO("best.pt")
12     results = model.predict(source=imagePath)
13
14     response = []
15     for result in results:
16         for box in result.boxes:
17             if model.names[box.cls.item()] == 'reactor':
18                 response.append({
19                     'class_name': model.names[box.cls.item()],
20                     'bbox': box.xyxy.tolist()
21                 })
22
23     return jsonify(response)
24
25 if __name__ == '__main__':
26     app.run(host='0.0.0.0', port=5000)
27

```

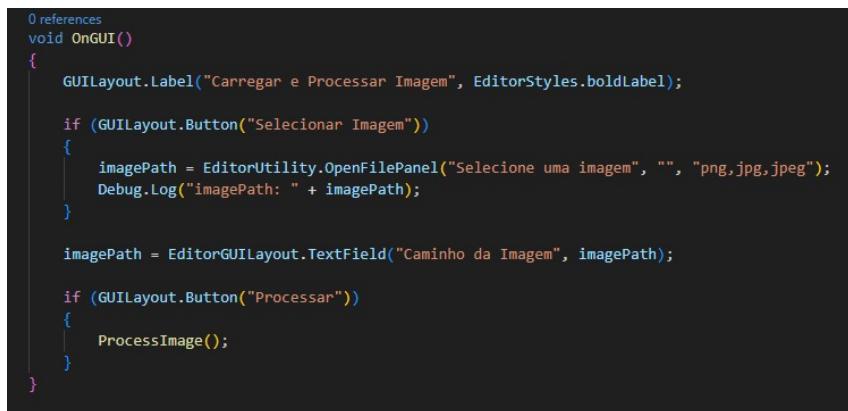
Figura 25 – API de identificação de imagem

Para executar esta aplicação, você precisará do Python 3.5 ou superior instalado em seu sistema. Pode-se instalar as dependências manualmente usando o comando *pip install*, especificando os pacotes necessários. Neste caso, como o script utiliza Flask e o YOLO da biblioteca *Ultralytics*, pode-se instalar essas bibliotecas com o seguinte comando: *pip install flask ultralytics*. Após instalar as dependências, supondo que o script seja salvo em um arquivo chamado *api-rv.py*, abra um terminal no diretório onde o script está localizado e execute o comando: *python api-rv.py*. Isso iniciará o servidor Flask e tornará o endpoint *"/predict"* acessível para aplicações externas, como o Unity. O código completo do script

api-rv.py está localizado no Apêndice A.

5.4.1 Implementação na Unity

O script desenvolvido no Unity é dividido em alguns métodos. Inicialmente, há a disposição na tela da interface gráfica, representada na Figura 26. Este código define o nome da janela, suas dimensões e os tipos de interações que o usuário pode realizar, como selecionar uma imagem e iniciar o processamento. A GUI foi desenvolvida com foco na usabilidade, garantindo que até mesmo usuários com pouca experiência em Unity possam operar a implementação com facilidade. O código completo que habilita esta implementação dentro do Unity, entitulado *ModelLoaderMenu.cs*, está localizado no Apêndice B. Copiá-lo em um arquivo homônimo e colocá-lo dentro do projeto Unity é suficiente para poder utilizá-lo.



```

0 references
void OnGUI()
{
    GUILayout.Label("Carregar e Processar Imagem", EditorStyles.boldLabel);

    if (GUILayout.Button("Selecionar Imagem"))
    {
        imagePath = EditorUtility.OpenFilePanel("Selecione uma imagem", "", "png,jpg,jpeg");
        Debug.Log("imagePath: " + imagePath);
    }

    imagePath = EditorGUILayout.TextField("Caminho da Imagem", imagePath);

    if (GUILayout.Button("Processar"))
    {
        ProcessImage();
    }
}

```

Figura 26 – API de identificação de imagem

Ao pressionar o botão de processamento, o método correspondente é ativado, conforme ilustrado na Figura 27. Este método chama outro, que é responsável por enviar a requisição *HTTP* para a *API* processar a resposta recebida. Com as informações retornadas pela *API*, o *script* percorre cada objeto identificado e o insere na cena do Unity, posicionando-os de acordo com uma disposição pré-definida.

É importante destacar que o modelo tridimensional do reator, nesta aplicação, foi armazenado dentro do próprio diretório do projeto, e é armazenado em uma variável *modelPath* conforme a linha:

```
string modelPath = "Assets/Model/FILTER_F10.fbx";
```

Naturalmente, em uma expansão deste protótipo, seria possível uma integração da aplicação com um sistema de armazenamento remoto, haja visto que a coleção da modelagem de diversos equipamentos pode ser custosa em termos de armazenamento, assim como possibilitaria um desacoplamento da *API* com seus componentes. A computação em nuvem, que é entendida como um modelo de entrega de serviços que permite o acesso sob

demandaria recursos computacionais compartilhados, como servidores, armazenamento e redes, pode ser uma solução viável para esse tipo de expansão (MELL, 2011). Este modelo permitiria o armazenamento e a gestão eficiente de modelos tridimensionais, ao mesmo tempo que possibilitaria escalabilidade e flexibilidade à medida que a coleção de modelos cresce, sem sobrecarregar os recursos locais.

```

async void ProcessImage()
{
    if (string.IsNullOrEmpty(imagePath))
    {
        Debug.LogError("Nenhuma imagem carregada para processar.");
        return;
    }

    Vector3 basePosition = Vector3.zero;
    Vector3 spacing = new Vector3(10, 0, 0);

    List<DetectionResult> detectionResults = await SendImageForPrediction(imagePath);

    if (detectionResults != null && detectionResults.Count >= 4)
    {
        string modelPath = "Assets/Model/FILTER_F10.fbx";
        GameObject model = AssetDatabase.LoadAssetAtPath<GameObject>(modelPath);

        if (model != null)
        {
            for (int i = 0; i < detectionResults.Count; i++)
            {
                GameObject modelInstance = (GameObject)PrefabUtility.InstantiatePrefab(model);
                if (modelInstance != null)
                {
                    modelInstance.transform.position = basePosition + spacing * i;
                    modelInstance.transform.rotation = Quaternion.Euler(-90, 0, 0);
                    Debug.Log("Reator inserido na posição: " + modelInstance.transform.position);
                }
                else
                {
                    Debug.LogError("Erro ao instanciar o modelo virtual.");
                }
            }
        }
        else
        {
            Debug.LogError("Erro ao carregar o modelo virtual. Verifique o caminho.");
        }
    }
}

```

Figura 27 – Método de processamento da imagem e inserção dos reatores conforme a resposta da API

Após o processamento da imagem, o número de reatores identificados pelo modelo é automaticamente renderizado na cena do Unity. Essa etapa garante que os objetos reconhecidos sejam visualmente representados no ambiente virtual, facilitando a interação e análise do usuário. Neste protótipo, os reatores identificados são posicionados com espaçamento padrão entre um e outro. Não há um posicionamento correto dos equipamentos, sendo esta uma demanda para implementações futuras. A Figura 28 ilustra um exemplo de imagem processada, na qual foram corretamente identificados e inseridos quatro reatores na cena. A quantidade de reatores e outros objetos detectados e inseridos na cena depende diretamente da precisão do modelo de reconhecimento utilizado, bem como da qualidade e clareza dos elementos presentes na imagem original. Dessa forma, um modelo bem treinado, aliado a uma imagem nítida e bem enquadrada, resulta em uma identificação mais precisa e uma correspondência fiel entre os objetos detectados e os inseridos no ambiente virtual.

Esse processo automatizado não apenas melhora a eficiência da criação de cenas no Unity, mas também minimiza a necessidade de ajustes manuais, proporcionando uma experiência mais fluida e prática para o usuário.

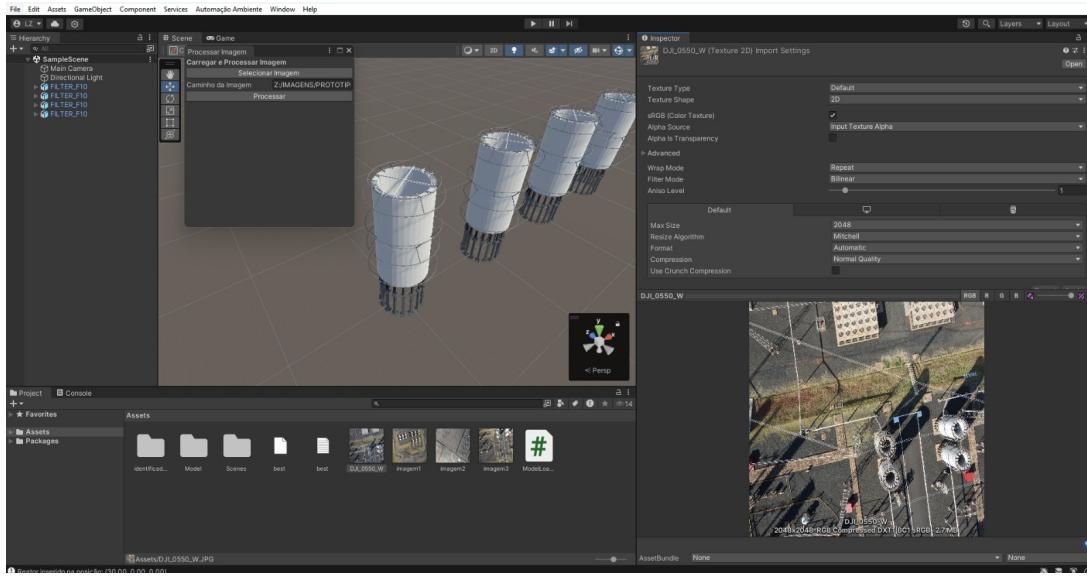


Figura 28 – Método de processamento da imagem e inserção dos modelos conforme a resposta da *API*

5.5 Considerações Finais

A implementação desenvolvida é altamente intuitiva e de fácil utilização. Para executá-la em um computador pessoal, basta inserir o script direcionado para o Unity dentro do projeto e, em paralelo, rodar o código da *API* em um arquivo separado. Essa aplicação se destaca como uma solução prática e eficiente, oferecendo um método simplificado que pode gerar um impacto significativo no desenvolvimento de ambientes de RV.

6 Resultados e Discussão

6.1 Introdução

Neste capítulo, será abordado a sequência de testes realizados para o treinamento da RNC, assim como os resultados obtidos. Além disso, será apresentado a eficiência da utilização dos pesos da configuração de treinamento da YOLO que obteve a melhor performance.

6.2 Resultados

6.2.1 Resultados e análise do treinamento com a YOLOv5

Na Tabela 3, apresenta-se o resultado da aplicação da YOLOv5 ao dataset de reatores de núcleo de ar, alterando o valor de batch de 2 a 16. Nela, é evidente notar a primeira característica marcante de se diminuir o tamanho do batch: há um aumento no tempo de processamento da RNC, uma vez que lotes menores de imagens estão sendo analisados em cada iteração, demandam maior tempo para cobrir todo o escopo do dataset. No treinamento com a arquitetura YOLOv5, a mAP50 apresenta valores que podem ser considerados altos, inclusive maiores que aqueles apresentados no estudo de (WANG et al., 2023), que também se dispôs a buscar modelos que identificassem objetos por meio de fotos obtidas de VANTs.

De modo geral, nesse sugere que o aumento do tamanho do batch de 2 para 8 resulta em melhorias significativas na precisão, recall, mAP50 e mAP50-95, ao mesmo tempo em que reduz o tempo de treinamento. No entanto, ao aumentar o batch size para 16, embora a precisão atinja seu valor mais alto, o recall e mAP50-95 apresentam uma pequena queda. Portanto, um batch size de 8 parece ser o mais eficiente em termos de equilíbrio entre desempenho e tempo de treinamento para este dataset específico.

Otimizador SGD					
Batch Size	Precisão	Recall	mAP50	mAP50-95	Tempo
2	88.724	87.456	88.785	49.821	48.75h
4	89.831	89.707	95.044	55.908	37.67h
8	92.416	91.922	96.197	60.488	33.46h
16	93.340	90.977	95.474	54.689	29.53h

Tabela 3 – Resultados de diferentes tamanhos de batch aplicando YOLOv5

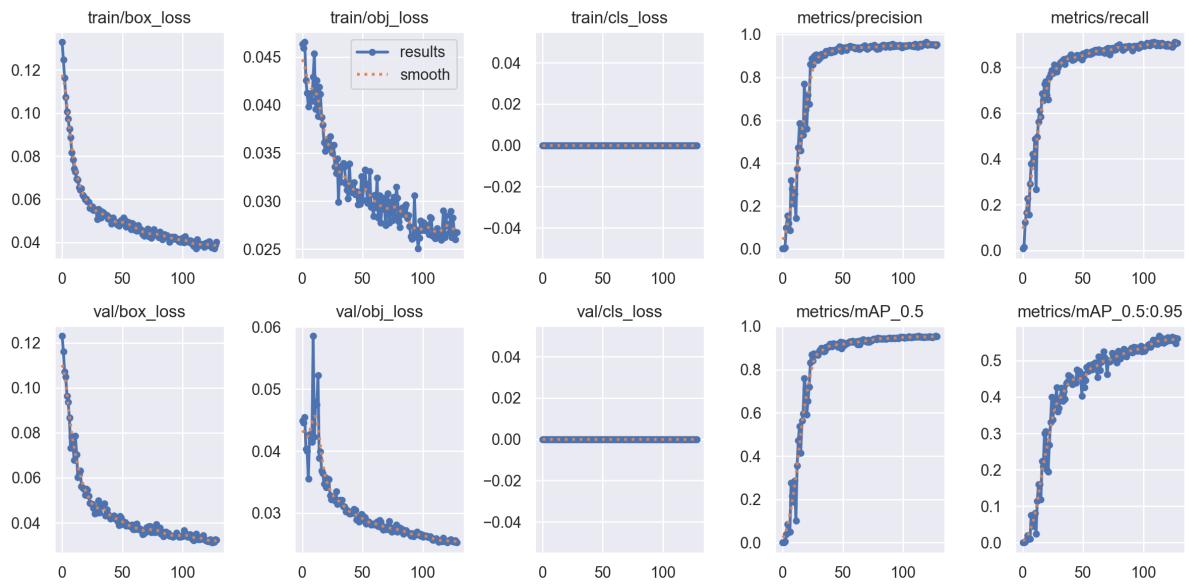


Figura 29 – Gráficos gerados a partir do treinamento com a YOLOv5, utilizando *batch* igual a 16. Nota-se a evolução repentina para uma precisão acima de 90%

6.2.2 Resultados e análise do treinamento com a YOLOv6

Embora a YOLOv6 tenha oferecido um leve aumento em comparação com a YOLOv5, ela não trouxe melhorias significativas no mAP50-95, que permaneceu na casa dos 50%. A precisão geral foi similar à YOLOv5, mas com uma pequena vantagem no batch size 16. A maior diferença foi observada no tempo de treinamento, que foi reduzido em todas as configurações, tornando a YOLOv6 mais eficiente em termos de tempo (Tabela 4).

Otimizador SGD					
Batch Size	Precisão	Recall	mAP50	mAP50-95	Tempo
2	88.523	86.225	87.895	50.705	50.78h
4	87.431	90.505	94.124	54.108	39.85h
8	91.578	90.592	95.298	53.128	32.65h
16	92.775	92.332	94.548	56.739	30.54h

Tabela 4 – Resultados de diferentes tamanhos de batch aplicando YOLOv6

6.2.3 Resultados e análise do treinamento com a YOLOv7

A YOLOv7 apresentou variabilidade significativa nos resultados, com quedas notáveis na precisão e recall, especialmente para batch sizes menores. Embora tenha demonstrado estabilidade no tempo de treinamento, seu desempenho geral foi menos consistente em comparação com as versões anteriores. O mAP50-95, por exemplo, não superou os valores obtidos com a YOLOv5, sendo inclusive inferior em algumas configurações, como

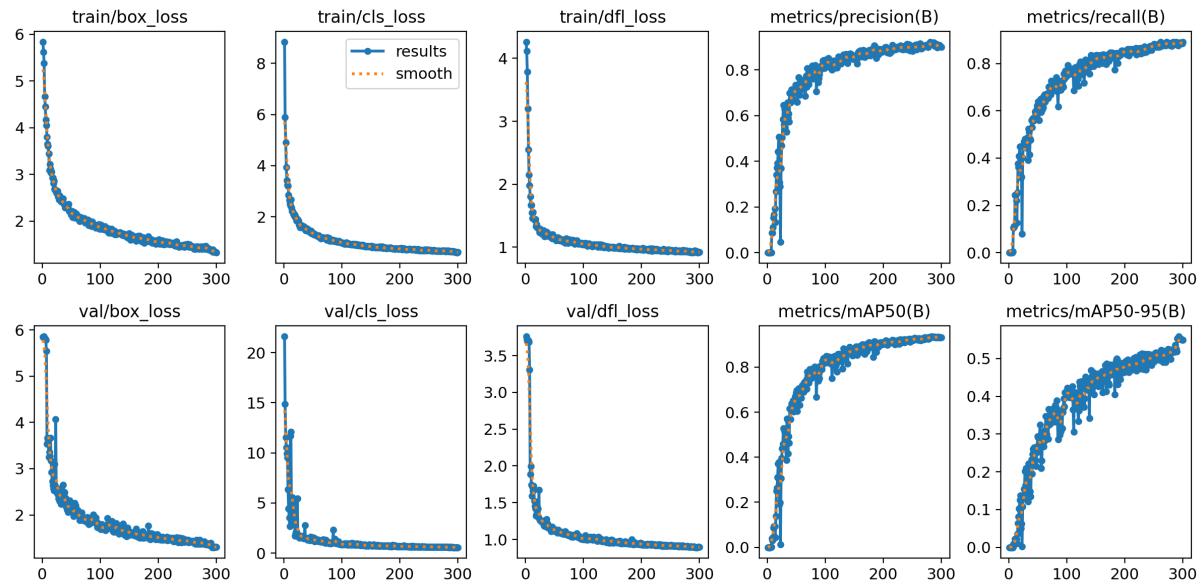


Figura 30 – Gráficos gerados a partir do treinamento com a YOLOv6.

observado na Tabela 5. Esta versão destacou-se pelo comportamento alternado, sugerindo um trade-off entre estabilidade temporal e eficácia.

Otimizador SGD					
Batch Size	Precisão	Recall	mAP50	mAP50-95	Tempo
2	66.84	59.45	62.42	53.45	36.69h
4	65.74	58.45	65.45	56.807	37.47h
8	70.65	59.582	68.57	54.71	38.46h
16	73.11	60.882	71.35	52.53	36.14h

Tabela 5 – Resultados de diferentes tamanhos de batch aplicando YOLOv7

Comparando as três versões, a YOLOv5 mostrou-se superior em termos de desempenho geral, com alta precisão e mAP50-95, aliada a um tempo de treinamento eficiente. A YOLOv6, embora menos precisa, destacou-se pela redução do tempo de treinamento, sendo uma escolha viável quando a eficiência temporal é prioritária. Já a YOLOv7, apesar de sua estabilidade no tempo de treinamento, apresentou maior variabilidade nos resultados e um desempenho inferior em relação às versões anteriores, sugerindo que sua aplicação deve ser cuidadosamente considerada dependendo do contexto e das necessidades do projeto.

6.2.4 Análise dos Resultados do Treinamento com YOLOv8

Apresentou o melhor desempenho geral, destacando-se tanto em precisão quanto em recall, mAP50 e mAP50-95. A combinação com o otimizador *AdamW* se mostrou superior, oferecendo o melhor equilíbrio entre desempenho e tempo de treinamento, especialmente com batch size de 16.

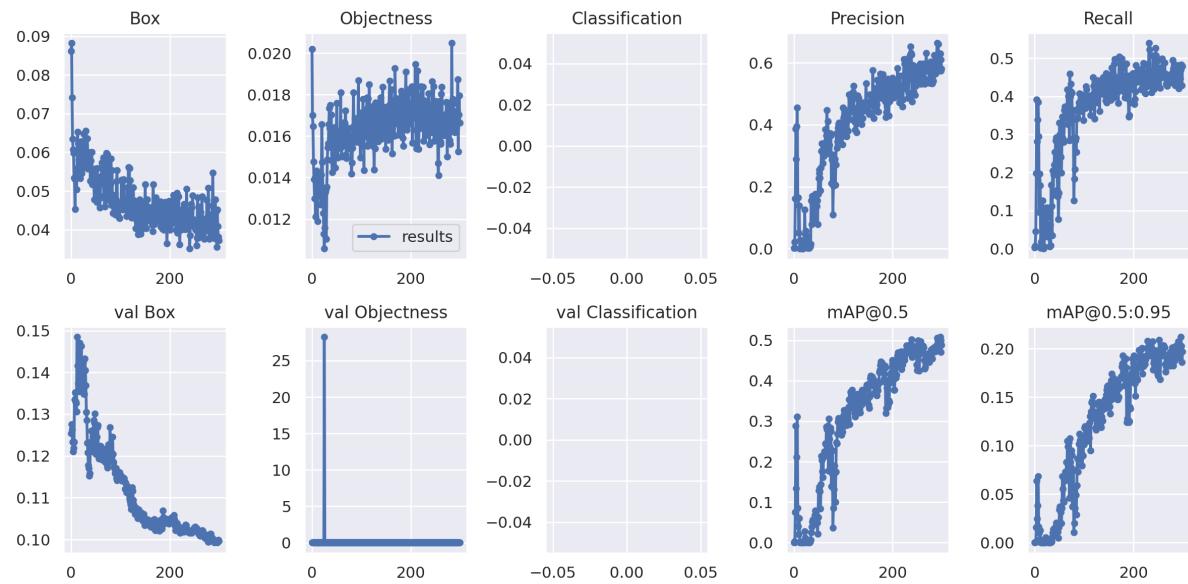


Figura 31 – Gráficos gerados a partir do treinamento com a YOLOv7, utilizando *batch* igual a 16. O comportamento alternado sobressaltou-se nessa arquitetura

Otimizador Adam					
Batch Size	Precisão	Recall	mAP50	mAP50-95	Tempo
2	91.323	90.102	94.022	54.987	33.071h
4	92.983	91.857	95.4	57.704	36.548h
8	92.383	91.969	95.006	57.465	31.221h
16	93.654	92.964	96.148	60.269	29.135h

Tabela 6 – Resultados de diferentes tamanhos de batch aplicando YOLOv8 e otimizador *Adam*

A Tabela 7 mostra os resultados com o otimizador *AdamW*, onde a melhoria na precisão e no recall é ainda mais pronunciada. A precisão cresce de 93.886% para 95.784%, e o recall sobe de 91.954% para 95.379% com o aumento do tamanho do batch. As métricas de mAP50 e mAP50-95 atingem 97.858% e 68.393%, respectivamente, para o maior batch size. Embora o tempo de treinamento com *AdamW* seja consistente, o batch size de 16 proporciona o tempo mais reduzido de 28.013 horas, mostrando uma vantagem competitiva em relação ao *Adam*.

Otimizador AdamW					
Batch Size	Precisão	Recall	mAP50	mAP50-95	Tempo
2	93.886	91.954	95.541	58.209	33.114h
4	93.701	92.102	95.426	57.508	32.744h
8	93.364	92.215	95.383	57.666	29.209h
16	95.784	95.379	97.858	68.393	28.013h

Tabela 7 – Resultados de diferentes tamanhos de batch aplicando YOLOv8 e otimizador *AdamW*

Por fim, a Tabela 8 apresenta os resultados com o otimizador *SGD*, que não só mostra uma melhoria contínua na precisão e no recall, com valores de 96.480% e 94.718% para o maior batch size, como também possui os melhores tempos de treinamento. O mAP50 e o mAP50-95 atingem 97.891% e 68.245%, respectivamente. O *SGD* combina o melhor desempenho em termos de métricas com a eficiência de tempo de treinamento, atingindo 27.633 horas para o maior batch size.

Otimizador <i>SGD</i>					
Batch Size	Precisão	Recall	mAP50	mAP50-95	Tempo
2	94.466	93.062	96.421	61.710	35.448h
4	94.989	93.779	96.759	65.201	31.018h
8	95.618	94.621	97.123	66.451	29.787h
16	96.480	94.718	97.891	68.245	27.633h

Tabela 8 – Resultados de diferentes tamanhos de batch aplicando YOLOv8 e otimizador *SGD*

Em resumo, os dados indicam que aumentar o tamanho do batch geralmente melhora o desempenho do modelo em termos de precisão, recall e métricas de mAP, independentemente do otimizador utilizado. Entre os otimizadores testados, o *SGD* oferece o melhor equilíbrio entre desempenho e eficiência de tempo, seguido pelo *AdamW*, que proporciona alta precisão e recall, mas com um tempo de treinamento competitivo. O *Adam*, embora ainda eficaz, não alcança o mesmo nível de desempenho que os outros dois otimizadores. Esses resultados destacam a importância de escolher cuidadosamente o otimizador e o tamanho do batch para otimizar o treinamento e o desempenho do YOLOv8.

6.2.5 Comparação de todas as versões

Semelhante ao estudo de (JOCHER; CHAURASIA; QIU, 2023), foi gerado um gráfico comparativo entre todas as versões da YOLO que aqui foram trabalhadas, tomando por base os resultados obtidos de mAP50-95 de cada, variando o tamanho de *batch* nos valores de 2, 4, 8 e 16. Na Figura 32, é apresentado o gráfico resultado da comparação. Nele, percebe-se visualmente que a YOLOv8, quando aplicada ao *dataset* deste trabalho, se destaca com uma superioridade na precisão, em comparação com as três últimas versões, ratificando a proposta de seu uso.

Com base nos resultados apresentados na Figura 32, bem como na Figura (PADILLA; NETTO; SILVA, 2020), em que foi utilizado um COCO para realizar os treinamento com a YOLO, pode-se inferir que o conjunto de imagens construído neste trabalho se comporta de maneira semelhante ao COCO, como discutido em (PADILLA; NETTO; SILVA, 2020). Esse comportamento reforça a adequação das imagens para comparar o desempenho das diferentes versões da YOLO.

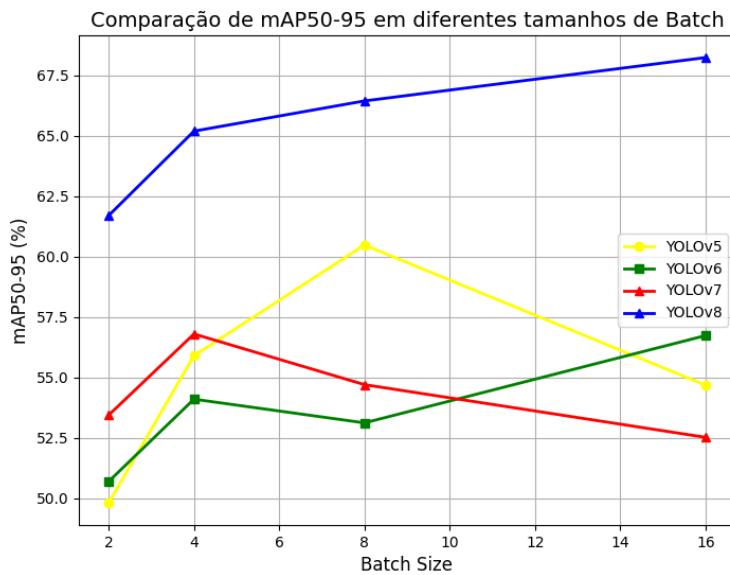


Figura 32 – Comparação da YOLOv5 à YOLOv8, com otimizador *SGD* e variação de *batch* de 2, 4, 8 e 16

As versões da YOLO não apresentaram um padrão claro de superioridade nem mostraram um crescimento consistente na precisão conforme o tamanho do *batch* aumentava. Isso ocorre porque, ao serem expostas a diferentes tamanhos de *batch*, as RNC podem não convergir de maneira previsível. A oscilação na precisão se deve às variações nos caminhos de aprendizado, que impactam os gradientes e tornam a otimização menos eficiente. Neste caso, tamanhos de *batch* como 16, resultaram em um desempenho inferior para os modelos YOLOv5 e YOLOv7. Esse fenômeno pode ser atribuído à estagnação do aprendizado em regiões conhecidas como "planícies" ou *plateaus*. Essas regiões são caracterizadas por uma diminuição na inclinação dos gradientes, levando a uma desaceleração no crescimento do aprendizado da RNC. Nessas situações, a rede pode se encontrar em um mínimo local, onde os ajustes dos pesos se tornam pouco eficazes, resultando em uma otimização subideal (DUCK, n.d.). Essa variabilidade no desempenho é uma característica comum em modelos de aprendizado profundo, especialmente em tarefas complexas como a detecção de objetos. A presença de planícies pode dificultar a exploração do espaço de parâmetros da rede, fazendo com que ela perca oportunidades de melhorar sua precisão (GOODFELLOW; BENGIO; COURVILLE, 2016).

A implementação do YOLOv8 com diferentes configurações de otimizadores e tamanhos de batch apresentou, no geral, resultados promissores, evidenciando uma tendência de melhora nas métricas conforme o aumento do tamanho do batch. Dentre as configurações avaliadas, a combinação do otimizador *SGD* com batch size de 16 se destacou por oferecer o melhor equilíbrio entre precisão, recall, mAP50, mAP50-95 e tempo de treinamento

razoavelmente curto. Portanto, essa configuração específica, foi naturalmente escolhida para os testes subsequentes com o peso treinado.

6.2.6 Avaliação da Implementação

Na separação do conjunto de imagens para treinamento, foi adotada a prática de reservar 20% do total de imagens para testes manuais. No caso, foram utilizadas 80 fotos para avaliar o desempenho do modelo treinado. A precisão média foi calculada aplicando o modelo em cada uma das 80 fotos, aferindo a quantidade de reatores detectados. A RNC apresentou resultados variados, identificando a quantidade exata de reatores em algumas imagens, enquanto em outras identificou um número superior ou inferior ao real. Para avaliar o desempenho, foi calculada a média absoluta da diferença entre os valores identificados e os valores reais presentes nas fotos, conforme a Equação (6.1), em que R_e trata-se dos valores de reatores estimados pela RNC; enquanto R_p , trata-se dos valores presentes de reatores em cada imagem.

$$\text{Média de Acertos da RNC} = \frac{1}{n} \sum_{i=1}^n \left(1 - \frac{|R_{e_i} - R_{p_i}|}{R_{p_i}} \right) \times 100\% \quad (6.1)$$

Para avaliar a eficácia do modelo na prática, foram realizados testes com um conjunto de 80 imagens não vistas anteriormente. A Tabela 9 resume o desempenho do modelo YOLOv8 com a configuração ideal (*SGD* e batch size 16), revelando que 72,34375% das imagens foram corretamente identificadas, refletindo uma taxa de acerto considerável. Conforme trabalho de (BARBADO et al., 2022), essa uma taxa de acerto para uma RNC acima de 70%, para uma aplicação em nível de protótipo, mostra-se demasiada satisfatória. Com isso, o propósito do trabalho, que se coloca como uma proposição de utilização de uma tecnologia em evidência para melhoria de um processo, é dado como atingido.

Configuração	Número de Imagens Testadas	Taxa Média de Acerto
<i>SGD</i> + Batch Size 16	80	72,34375%

Tabela 9 – Resumo do desempenho na aplicação da YOLOv8 na melhor configuração encontrada

6.3 Considerações Finais

O YOLOv8, como evolução natural das versões anteriores da família YOLO, traz melhorias significativas tanto na arquitetura do modelo quanto na eficiência do treinamento. Suas inovações permitem uma melhor detecção de objetos pequenos e maior precisão global, mantendo a rapidez e eficiência características dos modelos YOLO. A combinação de ajustes no tamanho do batch e a escolha do otimizador demonstram a flexibilidade do

YOLOv8, permitindo sua adaptação a diversos cenários de aplicação, desde ambientes de baixa capacidade computacional até sistemas de alta demanda.

Tabela 10 – Resumo comparativo dos trabalhos relacionados

Trabalhos Relacionados	Alteração de Parâmetros da RNC	Treinamento utilizando YOLOv8	Utilização de VANT	Subestações de Energia	Automação de Inserção de RV
(GONZAGA, 2023)	✓	✓	✗	✗	✗
(DIAS; FIGUEIREDO; MAFRA, 2021)	✓	✓	✗	✗	✗
(WANG et al., 2023)	✓	✓	✓	✗	✗
(ZUZA et al. 2024)	✓	✓	✓	✓	✓

Dessa forma, a escolha do *SGD* como otimizador, aliada a um tamanho de batch adequado, como o de 16, provou ser a configuração mais robusta para obter o equilíbrio ideal entre desempenho e custo computacional, tornando o YOLOv8 uma escolha vantajosa para aplicações de detecção de objetos, como os reatores de núcleo de ar, visando precisão e eficiência. Comparativamente, a Tabela 10 faz o paralelo desta dissertação com os trabalhos correlatos apresentados no Capítulo 3, ressaltando os principais objetivos e motivos aqui trabalhados, confirmando que a proposta se mostrou bem sucedida.

7 Conclusão e Trabalhos Futuros

7.1 Introdução

Neste trabalho, foi proposto um sistema de inserção automática de um equipamento específico de uma subestação de energia em um software de geração de ambientes de RV. Para isso, concorreram a captura de imagens aéreas para ser substrato ao treinamento da RNC; recorreu-se a uma base de modelos digitalizados do equipamento estudado; e foi desenvolvido um sistema automático que vincule essa peça em um ambiente virtual, apenas fornecendo uma imagem em que a mesma esteja representada em uma visão aérea. Esta seção apresentará uma conclusão sobre todo o estudo desenvolvido, assim como trabalhos futuros que poderão partir desse, tanto como melhorias, assim como outras contribuições possíveis à comunidade científica, no devido contexto.

7.2 Conclusão

Neste estudo, foi desenvolvido um sistema que permite a inserção automática de equipamentos específicos de subestações de energia em ambientes de RV. Por meio da captura de imagens aéreas e do treinamento da RNC YOLOv8, conseguiu-se identificar e posicionar automaticamente os equipamentos em um ambiente virtual devido a sua boa eficiência no reconhecimento dos diferentes componentes nas imagens. Isso permitiu a implementação de uma automação com o software Unity permitiu integrar de forma eficiente os modelos digitalizados dos equipamentos ao ambiente de RV, garantindo uma representação consideravelmente precisa, assim como intuitiva.

Além disso, com esta dissertação foi possível validar a superioridade de precisão no uso da YOLOv8 para o conjunto de imagens coletadas em subestações de energia por meio de VANTs, em relação a suas últimas versões. Isso confirma a extração feita a partir do trabalho de (JOCHER; CHAURASIA; QIU, 2023), para o conjunto de dados aqui apresentado.

Também neste trabalho, conclui-se que o estudo apresentado por (GONZAGA, 2023) mostrou-se substrato essencial para chegar à melhor configuração dos parâmetros da YOLO para obter a melhor precisão em sua oitava versão.

Os resultados abrem espaço para uma grande variedade de trabalhos que busquem utilizar de imagens aéreas combinadas com RNC, aplicadas à construção de ambientes de RV. Essa abordagem, se explorada ainda mais a fundo para outros equipamentos presentes em toda extensão da subestação, poderá elevar o nível da automação da criação

de ambientes virtuais, poupando esforço manual e, com os devidos ajustes, aumentando a precisão da criação.

7.3 Trabalhos Futuros

Este trabalho se propõe a ser um ponto de partida para uma série de futuras implementações e estudos. Primeiramente, a precisão e a eficiência do sistema podem ser aprimoradas com o uso de *datasets* mais amplos e variados para o treinamento do YOLOv8. Além disso, a inclusão de outros equipamentos presentes nas subestações de energia no processo de treinamento abriria oportunidades para novos estudos.

Naturalmente, uma melhoria significativa na aplicabilidade deste sistema envolveria a implementação de uma *API* com funcionalidades de geolocalização de cada equipamento. Dessa forma, além da detecção da quantidade de objetos na imagem, seria possível posicioná-los corretamente conforme sua disposição real na subestação. Em outras palavras, o objetivo seria desenvolver um sistema de detecção que posicione os objetos no ambiente virtual de acordo com a configuração apresentada na imagem usada para identificação. Para isso, seriam necessárias análises mais detalhadas ou o uso de dados complementares que viabilizassem tal inserção.

Finalmente, espera-se que este trabalho sirva como base para novas pesquisas na área de realidade virtual e inteligência artificial aplicada, contribuindo para o avanço tecnológico e oferecendo novas ferramentas e metodologias para a comunidade científica e industrial.

Referências

- ADORNO, V. *Redes Neurais Artificiais*. 2017. Medium. Disponível em: <<https://medium.com/@avinicius.adorno/redes-neurais-artificiais-418a34ea1a39>>. Citado 2 vezes nas páginas 10 e 9.
- AFONSO, M. H. et al. Vehicle and plate detection for intelligent transport systems: Performance evaluation of models yolov5 and yolov8. In: IEEE. *2023 IEEE International Conference on Computing (ICOCO)*. [S.l.], 2023. p. 328–333. Citado 2 vezes nas páginas 22 e 24.
- ANDRADE, A. de L.; SANTOS, M. A. D. Hydroelectric plants environmental viability: Strategic environmental assessment application in brazil. *Renewable and Sustainable Energy Reviews*, Elsevier, v. 52, p. 1413–1423, 2015. Citado na página 6.
- BARBADO, L. et al. Aplicação da rede convolucional yolo para análise de fluxo de veículos. In: SBC. *Congresso Latino-Americano de Software Livre e Tecnologias Abertas (Latinoware)*. [S.l.], 2022. p. 43–49. Citado na página 49.
- BRASIL, D. L. B. *O Neurônio Biológico e Matemático*. 2023. Acessado em: 2 de junho de 2024. Disponível em: <<https://www.deeplearningbook.com.br/o-neuronio-biologico-e-matematico/#:~:text=Limiar%20de%20ativa%C3%A7%C3%A3o%20%7B%20%CE%98%20%7D%3A,e%20o%20limiar%20de%20ativa%C3%A7%C3%A3o.>> Citado na página 9.
- CARBONI, D. F. Sistema para identificação de plantas invasoras em lavouras com detectores de objetos aplicados a imagens e vídeos. 2021. Citado na página 14.
- CARDOSO, A. et al. Tecnologias e ferramentas para o desenvolvimento de sistemas de realidade virtual e aumentada. *Editora Universitária UFPE*, p. 1–19, 2007. Citado na página 7.
- COMPUTACIONAL, V. *YOLO para Detecção de Objetos: Visão Geral*. 2024. Acessado em: 4 de junho de 2024. Disponível em: <<https://visaocomputacional.com.br/yolo-para-detectao-de-objetos-visao-geral>>. Citado na página 14.
- DIAS, R. L.; FIGUEIREDO, F. A. de; MAFRA, S. B. Comparação de modelos yolov5 e yolov8 para detecção de objetos em áreas rurais usando transferência de aprendizado. 2021. Citado 5 vezes nas páginas 10, 22, 25, 28 e 50.
- DUCK, T. O. *Local Minima, Saddle Points, and Plateaus*. n.d. Accessed: 2024-09-21. Disponível em: <<https://theorangeduck.com/page/local-minima-saddle-points-plateaus>>. Citado na página 48.
- ELETRÔNICA, A. E. *Reator Núcleo de Ar – Jauru Eletronorte 230kV*. 2024. Accessed: 2024-06-03. Disponível em: <<https://adseletrica.com.br/reactor-nucleo-de-ar-jauru-eletronorte-230kv>>. Citado 2 vezes nas páginas 10 e 6.

- FLECK, L. et al. Redes neurais artificiais: Princípios básicos. *Revista Eletrônica Científica Inovação e Tecnologia*, v. 1, n. 13, p. 47–57, 2016. Citado 3 vezes nas páginas 10, 9 e 11.
- GIRSHICK, R. Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1440–1448. Citado na página 26.
- GOMES, J. V. E. Detecção de objetos com a arquitetura yolo. 2022. Citado na página 23.
- GONZAGA, L. Identificação e medição de defeitos em produtos automotivos usando visão computacional. Serra, 2023. Citado 9 vezes nas páginas 10, 20, 21, 28, 32, 33, 38, 50 e 51.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016. Citado 4 vezes nas páginas 10, 16, 31 e 48.
- GRVA. *Grupo de Realidade Virtual e Aumentada (GRVA)*. 2024. Acesso em 16 de setembro de 2024. Disponível em: <<https://ppgeelt.feelt.ufu.br/grupos-de-pesquisa/grupo-de-realidade-virtual-e-aumentada-grva>>. Citado na página 29.
- HAYKIN, S. *Redes neurais: princípios e prática*. [S.l.]: Bookman Editora, 2001. Citado 2 vezes nas páginas 10 e 12.
- HE, K. et al. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. Citado 3 vezes nas páginas 10, 20 e 21.
- HUSSAIN, M. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, MDPI, v. 11, n. 7, p. 677, 2023. Citado na página 16.
- JIANG, P. et al. A review of yolo algorithm developments. *Procedia computer science*, Elsevier, v. 199, p. 1066–1073, 2022. Citado na página 15.
- JOCHER, G.; CHAURASIA, A.; QIU, J. *YOLO by Ultralytics*. Ultralytics. 2023. Citado 5 vezes nas páginas 10, 19, 32, 47 e 51.
- JUNIOR, C. d. L. B. et al. Uma proposta de sistema de autoria baseado em plantas baixas para projetar ambientes de realidade virtual: metodologia e estudo de caso aplicados a subestações de energia elétrica. Universidade Federal de Uberlândia, 2022. Citado 2 vezes nas páginas 2 e 6.
- JUNIOR, J. V. de O. Reatores para controle do fluxo de potência e suas consequências para a qualidade de energia. Universidade Federal de Minas Gerais, 2012. Citado na página 7.
- JÚNIOR, M. J. A. et al. Uma solução de realidade aumentada para manutenção assistida em subestações de energia elétrica em hvdc. *Revista Ibérica de Sistemas e Tecnologias de Informação*, Associação Ibérica de Sistemas e Tecnologias de Informação, n. E70, p. 547–560, 2024. Citado na página 29.
- KALAWSKY, R. *The science of virtual reality and virtual environments*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1993. Citado na página 7.
- KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado na página 17.

- KIRNER, C.; KIRNER, T. G. Evolução e tendências da realidade virtual e da realidade aumentada. *Realidade Virtual e Aumentada: Aplicações e Tendências. Cap*, v. 1, p. 10–25, 2011. Citado na página 8.
- LIMA, M.; RUBIK, E.; MORAIS, R. *Introdução ao reconhecimento de imagens*. 2020. Acessado em: 2 de junho de 2024. Disponível em: <<https://lamfo-unb.github.io/2020/12/05/Captcha-Break/>>. Citado 3 vezes nas páginas 10, 12 e 14.
- LIMA, M. E. A.; OLIVEIRA, R. C. Precarização e acidentes de trabalho: os riscos da terceirização no setor elétrico. *Revista Brasileira de Saúde Ocupacional*, SciELO Brasil, v. 46, p. e6, 2021. Citado na página 1.
- LOPES, Y. et al. Smart grid e iec 61850: Novos desafios em redes e telecomunicações para o sistema elétrico. *XXX Simpósio Brasileiro de Telecomunicações*, 2012. Citado 3 vezes nas páginas 10, 5 e 6.
- MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, p. 115–133, 1943. Citado na página 9.
- MELL, P. The nist definition of cloud computing. *NIST Special Publication*, p. 800–145, 2011. Citado na página 41.
- MUSIC, R. N. C. of. *A Short History of Neural Synthesis*. 2024. Acessado em: 2 de junho de 2024. Disponível em: <<https://www.rncm.ac.uk/research/research-activity/research-centres-rncm/prism/prism-blog/a-short-history-of-neural-synthesis/>>. Citado 2 vezes nas páginas 10 e 13.
- NETO, M. M. et al. Estratégias para concepção e digitalização de usinas e subestações de energia elétrica por captura da realidade: Fotogrametria associada ao escaneamento a laser versus modelagem paramétrica manual. *Revista Ibérica de Sistemas e Tecnologias de Informação*, Associação Ibérica de Sistemas e Tecnologias de Informacao, n. E66, p. 216–230, 2024. Citado na página 29.
- OĞCU, G.; DEMIREL, O. F.; ZAIM, S. Forecasting electricity consumption with neural networks and support vector regression. *Procedia-Social and Behavioral Sciences*, Elsevier, v. 58, p. 1576–1585, 2012. Citado na página 3.
- PADILLA, R.; NETTO, S. L.; SILVA, E. A. D. A survey on performance metrics for object-detection algorithms. In: IEEE. *2020 international conference on systems, signals and image processing (IWSSIP)*. [S.I.], 2020. p. 237–242. Citado 4 vezes nas páginas 10, 18, 19 e 47.
- PALMEIRA, E. G. Q. et al. O uncanny valley das mãos virtuais em aplicações de realidade virtual imersiva: uma revisão sistemática da literatura. *Revista Ibérica de Sistemas e Tecnologias de Informação*, Associação Ibérica de Sistemas e Tecnologias de Informacao, n. E31, p. 497–512, 2020. Citado na página 2.
- PyCharm. *PyCharm*. 2023. <<https://www.jetbrains.com/pt-br/pycharm/>>. Acesso em: 01 de nov. de 2023. Citado na página 31.
- RANDOLPH, J. Electric power substations engineering [book reviews]. *IEEE Power and Energy Magazine*, IEEE, v. 11, n. 3, p. 103–105, 2013. Citado na página 1.

- RAUBER, T. W. Redes neurais artificiais. *Universidade Federal do Espírito Santo*, v. 29, 2005. Citado 2 vezes nas páginas 10 e 11.
- 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 10, 14 e 15.
- REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. Citado na página 26.
- RODRIGUES, V. B. *Conhecendo a Visão do Computador: Redes Neurais Convolucionais*. 2023. Acessado em: 2 de junho de 2024. Disponível em: <<https://vitorborbarodrigues.medium.com/conhecendo-a-vis%C3%A3o-do-computador-redes-neurais-convolucionais-e1c2b14bf426>>. Citado na página 13.
- ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 10.
- RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *nature*, Nature Publishing Group UK London, v. 323, n. 6088, p. 533–536, 1986. Citado na página 11.
- SÁ, P. C. A. et al. Yolov8 para controle de produção pós-colheita e beneficiamento de frutos. *Revista de Engenharia e Pesquisa Aplicada*, v. 9, n. 1, p. 115–122, 2024. Citado na página 30.
- SAMRAT. *Splitting the Data to 60–20–20 Ratio vs. 80–10–10. Which One is Better?* 2024. <<https://medium.com/@itsmeSamrat/splitting-the-data-to-60-20-20-ratio-vs-80-10-10-which-one-is-better-bbc3503830d8>>. Accessed: 2024-09-18. Citado na página 29.
- SILVA, G. d. et al. Detecção e contagem de plantas utilizando técnicas de inteligência artificial e machine learning. *Graduação em Engenharia Eletrônica, Centro Tecnológico, Universidade Federal de Santa Catarina*, 2018. Citado na página 23.
- SILVA, R. C. et al. Virtual substation um sistema de realidade virtual para treinamento de operadores de subestações elétricas. Universidade Federal de Uberlândia, 2012. Citado na página 8.
- SILVA, R. E. V. d. Um estudo comparativo entre redes neurais convolucionais para a classificação de imagens. 2018. Citado 2 vezes nas páginas 14 e 17.
- SUTHERLAND, I. E. Sketchpad: A man-machine graphical communication system. p. 329–346, 1963. Citado na página 8.
- SUTHERLAND, I. E. A head-mounted three dimensional display. p. 757–764, 1968. Citado na página 8.
- TERVEN, J.; CORDOVA-ESPARZA, D. A comprehensive review of yolo: From yolov1 to yolov8 and beyond. *arXiv preprint arXiv:2304.00501*, 2023. Citado na página 16.
- ULTRALYTICS. *Ultralytics Repository*. 2024. Acessado em: 4 de junho de 2024. Disponível em: <<https://github.com/ultralytics/ultralytics>>. Citado na página 16.

Unity Technologies. *Unity: Desenvolvimento de jogos e aplicações 3D*. 2024. <<https://unity.com/>>. [Acessado em: 30 de julho de 2024]. Citado na página 31.

VARGAS, A. C. G.; PAES, A.; VASCONCELOS, C. N. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. In: SN. *Proceedings of the xxix conference on graphics, patterns and images*. [S.l.], 2016. v. 1, n. 4. Citado 3 vezes nas páginas 10, 12 e 13.

WANG, G. et al. Uav-yolov8: a small-object-detection model based on improved yolov8 for uav aerial photography scenarios. *Sensors*, MDPI, v. 23, n. 16, p. 7190, 2023. Citado 5 vezes nas páginas 25, 26, 28, 43 e 50.

WIDROW, B.; HOFF, M. E. et al. Adaptive switching circuits. v. 4, n. 1, p. 96–104, 1960. Citado na página 10.

ZHOU, K.; FU, C.; YANG, S. Big data driven smart energy management: From big data to big insights. *Renewable and sustainable energy reviews*, Elsevier, v. 56, p. 215–225, 2016. Citado na página 2.

ZHOU, P. et al. Towards understanding convergence and generalization of adamw. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, 2024. Citado na página 18.

A *Script API*

A.1 api-rv.py

```

from ultralytics import YOLO
from flask import Flask, request, jsonify
import os

app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json
    image_path = data['image_path']
    model = YOLO("best.pt")
    results = model.predict(source=image_path)

    response = []
    for result in results:
        for box in result.boxes:
            if model.names[box.cls.item()] == 'reactor':
                response.append({
                    'class_name': model.names[box.cls.item()],
                    'bbox': box.xyxy.tolist()
                })

    return jsonify(response)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

```

B Script do Unity

B.1 ModelLoaderMenu.cs

```

using UnityEditor;
using UnityEngine;
using System.Collections.Generic;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

public class ModelLoaderMenu : EditorWindow
{
    private string imagePath = "";
    private Texture2D loadedImage;

    [MenuItem("Automá̄ḡo Ambiente/Processar Imagem")]
    public static void ShowWindow()
    {
        ModelLoaderMenu window = GetWindow<ModelLoaderMenu>("Processar Imagem");
        window.minSize = new Vector2(300, 300);
        window.maxSize = new Vector2(300, 300);
    }

    void OnGUI()
    {
        GUILayout.Label("Carregar e Processar Imagem", EditorStyles.boldLabel);

        if (GUILayout.Button("Selecionar Imagem"))
        {
            imagePath = EditorUtility.OpenFilePanel("Selecione uma imagem", "", "png,jpg,jpeg");
            Debug.Log("imagePath: " + imagePath);
        }

        imagePath = EditorGUILayout.TextField("Caminho da Imagem",
            imagePath);

        if (GUILayout.Button("Processar"))
        {
            ProcessImage();
        }
    }
}

```

```
        }

    }

    async void ProcessImage()
    {
        if (string.IsNullOrEmpty(imagePath))
        {
            Debug.LogError("Nenhuma imagem carregada para processar.");
            return;
        }

        Vector3 basePosition = Vector3.zero;
        Vector3 spacing = new Vector3(10, 0, 0);

        List<DetectionResult> detectionResults = await
            SendImageForPrediction(imagePath);

        if (detectionResults != null && detectionResults.Count >= 4)
        {
            string modelPath = "Assets/Model/FILTER_F10.fbx";
            GameObject model = AssetDatabase.LoadAssetAtPath<
                GameObject>(modelPath);

            if (model != null)
            {
                for (int i = 0; i < detectionResults.Count; i++)
                {
                    GameObject modelInstance = (GameObject)
                        PrefabUtility.InstantiatePrefab(model);
                    if (modelInstance != null)
                    {
                        modelInstance.transform.position =
                            basePosition + spacing * i;
                        modelInstance.transform.rotation =
                            Quaternion.Euler(-90, 0, 0);
                        Debug.Log("Reator inserido na posição: " +
                            modelInstance.transform.position);
                    }
                else
                {
                    Debug.LogError("Erro ao instanciar o modelo ou
                        virtual.");
                }
            }
        }
    }
}
```

```
        {
            Debug.LogError("Erro ao carregar o modelo virtual. Verifique o caminho.");
        }
    }

async Task<List<DetectionResult>> SendImageForPrediction(string imagePath)
{
    string url = "http://192.168.12.13:5000/predict";
    var httpClient = new HttpClient();
    var content = new StringContent($"{{\"image_path\": \"{imagePath}\", \"}}", Encoding.UTF8, "application/json");

    HttpResponseMessage response = await httpClient.PostAsync(
        url, content);

    if (response.IsSuccessStatusCode)
    {
        string jsonResponse = await response.Content.
            ReadAsStringAsync();
        DetectionResultsWrapper wrapper = JsonUtility.FromJson<
            DetectionResultsWrapper>($"{{\"results\": {jsonResponse}}}");
        return wrapper.results;
    }
    else
    {
        Debug.LogError("Erro ao chamar o endpoint Flask: " +
            response.ReasonPhrase);
        return null;
    }
}

[System.Serializable]
public class DetectionResult
{
    public string ClassName;
    public List<float> BoundingBox;
}

[System.Serializable]
public class DetectionResultsWrapper
{
```

```
    public List<DetectionResult> results;  
}
```