

CLODOALDO DE SOUZA FARIA JÚNIOR

**Avaliação de Viabilidade do Uso de Deep
Learning em Computadores de Placa Única para
Aplicações em Agricultura**

Presidente Prudente

2025

CLODOALDO DE SOUZA FARIA JÚNIOR

**Avaliação de Viabilidade do Uso de Deep Learning em
Computadores de Placa Única para Aplicações em
Agricultura**

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciências Cartográficas da Faculdade de Ciências e Tecnologia da Universidade Estadual Paulista “Júlio de Mesquita Filho” - Campus de Presidente Prudente, para obtenção do título de Mestre em Ciências Cartográficas.

Universidade Estadual Paulista – UNESP

Faculdade de Ciências e Tecnologia

Programa de Pós-Graduação em Ciências Cartográficas

Orientador: Dr. Milton Hirokazu Shimabukuro

Coorientador: Dr. Antonio Maria Garcia Tommaselli

Presidente Prudente

2025

F224a

Faria Junior, Clodoaldo Souza

Avaliação de viabilidade do uso de Deep Learning em computadores de placa única para aplicações em agricultura / Clodoaldo Souza Faria Junior. -- Presidente Prudente, 2025
92 p.

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências e Tecnologia, Presidente Prudente

Orientador: Milton Hirokazu Shimabukuro

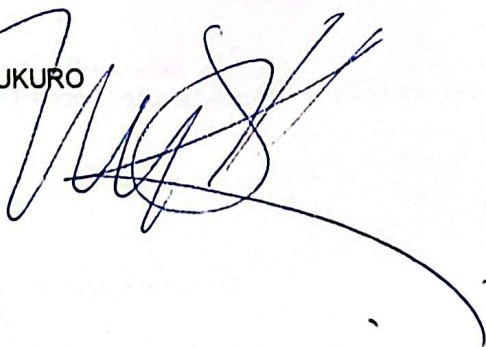
Coorientador: Antonio Maria Garcia Tommaselli

1. Deep Learning. 2. Agricultura Digital. 3. Visão Computacional. 4. YOLO. 5. NVIDIA Jetson Nano. I. Título.

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE CLODOALDO DE SOUZA FARIA JÚNIOR, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS CARTOGRÁFICAS, DA FACULDADE DE CIÊNCIAS E TECNOLOGIA - CÂMPUS DE PRESIDENTE PRUDENTE.

Aos 05 dias do mês de junho do ano de 2025, às 9h, no(a) Sala de Apresentação de Projetos da Central de Laboratórios do Depto. de Cartografia, realizou-se a defesa de DISSERTAÇÃO DE MESTRADO de CLODOALDO DE SOUZA FARIA JÚNIOR, intitulada **AVALIAÇÃO DE VIABILIDADE DO USO DE DEEP LEARNING EM COMPUTADORES DE PLACA ÚNICA PARA APLICAÇÕES EM AGRICULTURA**. A Comissão Examinadora foi constituída pelos seguintes membros: Prof. Dr. MILTON HIROKAZU SHIMABUKURO (Orientador(a) - Participação Presencial) do(a) Departamento de Matemática e Computação / Faculdade de Ciências e Tecnologia de Presidente Prudente - FCT/Unesp, Prof. Dr. NILTON NOBUHIRO IMAI (Participação Presencial) do(a) Departamento de Cartografia / Universidade Estadual Paulista "Júlio de Mesquita Filho" - Faculdade de Ciências e Tecnologia, Prof. Dr. FRANCISCO ASSIS DA SILVA (Participação Presencial) do(a) Universidade do Oeste Paulista - UNOESTE. Após a exposição pelo mestrando e arguição pelos membros da Comissão Examinadora que participaram do ato, de forma presencial e/ou virtual, o discente recebeu o conceito final: APROVADO . Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelo(a) Presidente(a) da Comissão Examinadora.

Prof. Dr. MILTON HIROKAZU SHIMABUKURO



Impacto Potencial Desta Pesquisa

A pesquisa tem o potencial de impulsionar aplicações na agricultura ao demonstrar a viabilidade do uso de *Deep Learning* em dispositivos acessíveis e eficientes, promovendo a agricultura de precisão e o uso sustentável de recursos. Esse avanço está alinhado com os Objetivos de Desenvolvimento Sustentável (ODS) estabelecidos pela Organização das Nações Unidas (ONU), em especial o ODS 2 (Fome Zero e Agricultura Sustentável), o ODS 9 (Indústria, Inovação e Infraestrutura) e o ODS 12 (Consumo e Produção Responsáveis), ao otimizar recursos operacionais e fomentar a sustentabilidade por meio de tecnologias inovadoras.

Potential Impact of This Research

The research has the potential to boost agricultural applications by demonstrating the feasibility of using Deep Learning on accessible and efficient devices, promoting precision agriculture and the sustainable use of resources. This advancement aligns with the Sustainable Development Goals (SDGs) established by the United Nations (UN), particularly SDG 2 (Zero Hunger and Sustainable Agriculture), SDG 9 (Industry, Innovation and Infrastructure), and SDG 12 (Responsible Consumption and Production), by optimizing operational resources and fostering sustainability through innovative technologies.

Agradecimentos

Agradeço sinceramente a todos que, de alguma forma, contribuíram para a realização deste trabalho. Cada apoio, orientação e incentivo foram essenciais para a conclusão desta jornada.

O presente trabalho foi realizado com apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), Brasil. Processo nº 2021/06029-7 e nº 2021/11296-4. Além disso, esse trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 88887.821780/2023-00.

Sou imensamente grato ao meu orientador, Milton Hirokazu Shimabukuro, e ao meu coorientador, Antonio Maria Garcia Tommaselli. As orientações detalhadas, o vasto conhecimento compartilhado e a dedicação de ambos foram imprescindíveis para o desenvolvimento desta dissertação.

Ao Programa de Pós-Graduação em Ciências Cartográficas (PPGCC), minha gratidão pela estrutura acadêmica que me proporcionou crescimento intelectual, aprendizagem contínua e um ambiente estimulante de troca de ideias, que foi crucial para o desenvolvimento desta pesquisa.

À Inspectral, agradeço a assistência técnica, os recursos disponibilizados e o valioso conhecimento adquirido durante o período em que estive na empresa. Gostaria de agradecer de forma especial ao Alisson e à Nariane, cujos conhecimentos e apoio foram indispensáveis para a realização deste projeto, bem como a toda a equipe de engenharia, que foi fundamental para o meu desenvolvimento pessoal e profissional.

Aos meus amigos e à minha família, agradeço imensamente pelo apoio constante e pela força que me deram, sempre me incentivando e me acompanhando em cada etapa deste processo.

E, finalmente, à minha namorada, Isabella Subtil Norberto, minha parceira em todos os momentos desta jornada. Sua paciência, compreensão, e apoio incondicional foram fundamentais para que eu pudesse concluir este trabalho com sucesso.

"Todos os modelos estão errados, mas alguns são úteis."
(George E. P. Box)

Resumo

Na agricultura a aplicação de *Deep Learning* por meio de computadores de placa única, emerge como uma solução potencial para implementar modelos complexos de inteligência artificial em ambientes de campo e com processamento local. Esses modelos são usualmente desenvolvidos para execução em computadores, servidores e máquinas de alta potência, e podem requerer ajustes para aplicações em campo ou em tempo real. A presente dissertação objetivou avaliar a implementação e a execução de técnicas de *Deep Learning* utilizando processamento em computadores de placa única (*Single Board Computer*, SBC) com GPU (*Graphics Processing Unit*). Os experimentos foram realizados utilizando um conjunto de dados coletado no projeto temático (FAPESP) "Sensoriamento Remoto de Alta Resolução para Agricultura Digital", da FCT/Unesp-Presidente Prudente. O conjunto de dados foi obtido em um pomar de laranjas utilizando a câmera Sony A7R Multispectral Sextuple, que possui um sistema de seis lentes - Q1 a Q6 -, as quais capturam em diferentes comprimentos de onda. Os modelos de detecção e segmentação utilizados foram os da família YOLO (*You Only Look Once*), especificamente os modelos YOLOv8 e YOLOv9 com suas variantes de arquitetura. Foi utilizado para uma delimitação inicial das instâncias de interesse, e posterior ajuste fino manual, um método para rotulação semiautomática baseado no *Segment Anything Model* (SAM). A investigação compreendeu três experimentos. O primeiro experimento avaliou os modelos YOLOv8 e YOLOv9, na forma previamente treinadas na base de dados Microsoft COCO de Lin *et al.* (2014), quanto às suas capacidades de segmentação de laranjas, explorando as potencialidades do *Transfer Learning* e do *Fine Tuning*. A inferência, foi realizada no SBC NVIDIA Jetson Nano. O principal objetivo deste experimento foi analisar se, mesmo com o uso de um modelo pré-treinado, a rede seria capaz de realizar a detecção e segmentação dos alvos (no caso, laranjas) de maneira satisfatória. No segundo experimento, foi construído um conjunto de dados original por meio da rotulação semiautomática com SAM. A partir desse conjunto, os modelos YOLOv8 e YOLOv9 foram treinados para a detecção e segmentação de folhas, aplicando-se *Transfer Learning* e *Fine Tuning*, conforme validados no primeiro experimento. O treinamento foi conduzido na plataforma Google Colab, enquanto a inferência ocorreu na NVIDIA Jetson Nano. No que se refere às variantes arquiteturais, os resultados mais consistentes foram obtidos com modelos de complexidade intermediária (YOLOv8-s e YOLOv8-m). Esse desempenho pode estar relacionado ao fato de que arquiteturas com um número elevado de parâmetros podem não apresentar uma configuração otimizada para o conjunto de dados analisado. No experimento final, foi realizada a segmentação de folhas em imagens das lentes Q4 e Q6, seguida da identificação de pares homólogos de folhas. A inferência foi conduzida na NVIDIA Jetson Nano, utilizando as arquiteturas previamente treinadas e exploradas no segundo experimento. Para a tomada de decisão sobre a correspondência entre os pares de folhas considerou-se, além da posição na imagem, a intersecção entre

as instâncias como parâmetros. A busca pelos pares para análise foi otimizada por meio da subdivisão da imagem em quadrantes, baseada na estrutura de dados QuadTree. Os resultados demonstraram uma relação direta a eficiência da detecção/segmentação e a precisão da correspondência. Em conclusão, demonstrou-se a exequibilidade de modelos de *Deep Learning* na NVIDIA Jetson Nano para aplicações agrícolas que envolvem detecção e segmentação de instâncias e os tempos de processamento sugerem a viabilidade de aplicação em tempo real.

Palavras-chave: *Deep Learning*. Agricultura Digital. Visão Computacional. YOLO. NVIDIA Jetson Nano.

Abstract

In agriculture, the application of Deep Learning through single-board computers emerges as a potential solution for implementing complex artificial intelligence models in field environments with local processing. These models are usually developed for execution on computers, servers, and high-performance machines, and may require adjustments for field or real-time applications. This dissertation aimed to evaluate the implementation and execution of Deep Learning techniques using processing on single-board computers (SBC) with GPUs (Graphics Processing Units). The experiments were conducted using a dataset collected as part of the thematic project (FAPESP) “High-Resolution Remote Sensing for Digital Agriculture,” from FCT/Unesp - Presidente Prudente. The dataset was obtained in an orange orchard using the Sony A7R Multispectral Sextuple camera, which features a system of six lenses, Q1 to Q6, capturing in different wavelengths. The detection and segmentation models used were from the YOLO (You Only Look Once) family, specifically the YOLOv8 and YOLOv9 models with their architectural variants. For the initial delimitation of the instances of interest, followed by manual fine-tuning, a semi-automatic labeling method based on the Segment Anything Model (SAM) was used. The investigation comprised three experiments. The first experiment evaluated the YOLOv8 and YOLOv9 models, previously trained on the Microsoft COCO dataset by Lin *et al.* (2014), in terms of their ability to segment oranges, exploring the potential of Transfer Learning and Fine Tuning. Inference was performed on the NVIDIA Jetson Nano SBC. The main objective of this experiment was to analyze whether, even with the use of a pre-trained model, the network would be capable of detecting and segmenting the targets (in this case, oranges) satisfactorily. In the second experiment, an original dataset was built using semi-automatic labeling with SAM. Based on this dataset, YOLOv8 and YOLOv9 models were trained for leaf detection and segmentation, applying Transfer Learning and Fine Tuning, as validated in the first experiment. Training was conducted on the Google Colab platform, while inference was performed on the NVIDIA Jetson Nano. Regarding architectural variants, the most consistent results were obtained with models of intermediate complexity (YOLOv8-s and YOLOv8-m). This performance may be related to the fact that architectures with a high number of parameters may not provide an optimized configuration for the analyzed dataset. In the final experiment, leaf segmentation was performed on images from lenses Q4 and Q6, followed by the identification of homologous leaf pairs. Inference was carried out on the NVIDIA Jetson Nano, using the architectures previously trained and explored in the second experiment. For the decision-making process regarding the matching between leaf pairs, both the position in the image and the intersection between instances were considered as parameters. The search for the pairs to be analyzed was optimized through the subdivision of the image into quadrants, based on the QuadTree data structure. The results showed a direct relationship

between detection/segmentation efficiency and matching accuracy. In conclusion, the feasibility of Deep Learning models on the NVIDIA Jetson Nano was demonstrated for agricultural applications involving instance detection and segmentation, and the processing times suggest the viability for real-time application.

Keywords: Deep Learning. Digital Agriculture. Computer Vision. YOLO. NVIDIA Jetson Nano.

Lista de figuras

Figura 1 – Um modelo básico de sistemas SoC.	24
Figura 2 – Arquitetura de um sistema utilizando <i>edge computing</i>	27
Figura 3 – Exemplo do contexto do <i>Deep Learning</i> , <i>Machine Learning</i> e <i>Artificial Intelligence</i>	28
Figura 4 – Comparativo entre <i>Machine Learning</i> e <i>Deep Learning</i>	29
Figura 5 – Rede Neural Profunda.	30
Figura 6 – Comparação do formato dos números de ponto flutuante do FP16 e do FP32.	32
Figura 7 – Utilização de FP16 e FP32 durante a etapa de treinamento de uma RNA.	32
Figura 8 – Dimensionamento de perdas por fatores pré-definidos baseado na utilização de FP16 ou FP32.	33
Figura 9 – Arquitetura da rede YOLO original.	34
Figura 10 – Exemplo de uma previsão de uma rede YOLO.	36
Figura 11 – Diagrama de alto nível de uma arquitetura moderna que usa <i>Backbone</i> , <i>Neck</i> e <i>Head</i>	37
Figura 12 – Arquitetura de uma FPN para detecção de objetos em múltiplas escalas.	37
Figura 13 – Arquitetura da YOLOv8.	40
Figura 14 – Arquitetura da YOLOv9.	42
Figura 15 – Comparação da arquitetura GELAN com arquiteturas do estado da arte disponíveis à época de sua proposição.	42
Figura 16 – Estrutura da PGI.	43
Figura 17 – Exemplos de operações de aumento de dados aplicadas aos dados de treinamento.	44
Figura 18 – Representação da aplicação do <i>Transfer Learning</i> e <i>Fine Tuning</i>	45
Figura 19 – Construção da estrutura de dados de uma QuadTree.	46
Figura 20 – Câmara multiespectral, modelo Sony A7R Multispectral Sextuple, com o sensor de sistema de seis cabeças.	47
Figura 21 – Descrição da placa NVIDIA Jetson Nano	48
Figura 22 – Mapa da localização do município de Matão.	50
Figura 23 – Descrição do comprimento de onda central e a composição de cada uma das subimagens obtidas pela câmara Sony A7R Multispectral Sextuple.	50
Figura 24 – Combinações de bandas de cada uma das subimagens da câmara Sony A7R Multispectral Sextuple.	51
Figura 25 – Fluxograma do processamento.	52
Figura 26 – Exemplo da criação do conjunto de dados utilizando SAM.	55
Figura 27 – Exemplo de matriz de confusão.	56

Figura 28 – Segmentação obtida para primeira imagem analisada do Q6 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	60
Figura 29 – Segmentação obtida para a segunda imagem analisada do Q6 da câmara Agrowing usando as redes pré-treinadas (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	61
Figura 30 – Segmentação obtida para a terceira imagem analisada do Q6 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	62
Figura 31 – Segmentação obtida para a primeira imagem analisada do Q4 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	64
Figura 32 – Segmentação obtida para a segunda imagem analisada do Q4 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	65
Figura 33 – Segmentação obtida para a terceira imagem analisada do Q4 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	67
Figura 34 – Segmentação obtida para a primeira imagem analisada do comprimento utilizando o comprimento de onda do 650 nm da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	68
Figura 35 – Resultados obtidos para as redes treinadas para a primeira imagem analisada (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	72
Figura 36 – Resultados obtidos para as redes treinadas para a segunda imagem analisada (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	73
Figura 37 – Busca de pares de laranjas correspondentes para a primeira cena analisada utilizando a YOLOv8-x para as composições coloridas formadas utilizando as subimagens Q4 e Q6.	78

Figura 38 – Resultado da correspondência de folhas a partir da segmentação aplicando (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e. . . .	79
Figura 39 – Resultado da segmentação de folhas aplicando (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.	81

Lista de tabelas

Tabela 1 – Especificações da Câmara multiespectral, modelo Sony A7R Multispectral Sextuple.	47
Tabela 2 – Descrição dos experimentos realizados.	58
Tabela 3 – Estatísticas da confiança, em %, para cada uma das arquiteturas utilizadas obtidas através da segmentação da primeira imagem e utilizando as bandas do Q6 da câmara Agrowing.	59
Tabela 4 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da segunda imagem e utilizando as bandas do Q6 da câmara Agrowing.	61
Tabela 5 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da terceira imagem e utilizando as bandas do Q6 da câmara Agrowing.	63
Tabela 6 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da primeira imagem e utilizando as bandas do Q4 da câmara Agrowing.	64
Tabela 7 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da segunda imagem e utilizando as bandas do Q4 da câmara Agrowing.	66
Tabela 8 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da terceira imagem e utilizando as bandas do Q4 da câmara Agrowing.	67
Tabela 9 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da primeira imagem e utilizando o Q6 da câmara Agrowing.	68
Tabela 10 – Métricas das arquiteturas obtidas durante o processo de treinamento. .	70
Tabela 11 – Métricas das arquiteturas treinadas obtidas durante o processo de validação.	71
Tabela 12 – Métricas das arquiteturas treinadas e validadas obtidas durante o processo de teste.	71
Tabela 13 – Níveis de confiança obtidos para as redes treinadas para a primeira imagem analisada.	72
Tabela 14 – Níveis de confiança obtidos para as redes treinadas para a segunda imagem analisada.	73
Tabela 15 – Média do tempo de execução para cada uma das arquiteturas executadas no Google Colab sem placa de vídeo.	75

Tabela 16 – Média do tempo de execução para cada uma das arquiteturas executadas no Google Colab com a Tesla T4, comparando resultados sem e com FP16.	75
Tabela 17 – Média do tempo de execução para cada uma das arquiteturas executadas na NVIDIA Jetson Nano, agrupadas por FP32 e FP16.	76
Tabela 18 – Média do tempo de execução para cada uma das arquiteturas executadas no Google Colab com a Tesla T4 e as métricas obtidas utilizando TensorRT.	77
Tabela 19 – Estatísticas da métrica de acurácia da busca de objetos semelhantes para a segunda cena analisada para cada uma das arquiteturas treinadas para as bandas Q4 e Q6.	79
Tabela 20 – Estatísticas da métrica de acurácia da busca de objetos semelhantes para a segunda cena analisada para cada uma das arquiteturas treinadas para as bandas Q4 e Q6.	80

Sumário

1	INTRODUÇÃO	18
1.1	Objetivos	21
1.1.1	Objetivos Gerais	21
1.1.2	Objetivos Específicos	21
1.2	Organização do documento	21
2	REVISÃO DA LITERATURA	22
2.1	Agricultura 4.0	22
2.2	Computador de Placa Única	23
2.3	Aceleração na execução utilizando GPU e processamento paralelo	25
2.4	Computação de Borda	26
2.5	Inteligência Artificial	27
2.5.1	<i>Deep Learning</i>	28
2.5.2	Processo de ajuste dos parâmetros de uma RNA	30
2.5.3	Precisão no treinamento de Redes Neurais Artificiais	31
2.6	Arquitetura YOLO	33
2.6.1	A evolução da YOLO	37
2.6.2	YOLOv8	39
2.6.3	YOLOv9	41
2.6.4	Técnicas de aumento de dados, <i>Transfer Learning</i> e <i>Fine Tuning</i>	43
2.7	QuadTree	44
3	MATERIAIS E MÉTODOS	47
3.1	Câmara multiespectral	47
3.2	Placa NVIDIA Jetson Nano	48
3.3	Conjuntos de dados e preparação	49
3.4	Frameworks de inteligência artificial, rotulagem e experimentos realizados	51
3.5	Técnicas de análise dos resultados	55
4	RESULTADOS E DISCUSSÕES	58
4.1	Experimento 1: Utilização das arquiteturas pré-treinadas	58
4.2	Experimento 2: Adequação e avaliação das arquiteturas para a segmentação de folhas	69
4.2.1	Avaliação dos níveis de confiança dos resultados	71
4.3	Execução das redes nas diferentes plataformas	74

4.4	Experimento 3: Busca de objetos semelhantes	77
5	CONCLUSÕES	82
	REFERÊNCIAS	84

1 INTRODUÇÃO

A busca por soluções computacionais para agilizar a extração de informações e oferecer suporte para a tomada de decisão é um cenário em diversas áreas de aplicação. Como exemplo, na agricultura, a detecção precoce de doenças em plantas é fundamental para prevenir a disseminação de infecções e garantir tratamentos eficazes, pois isso afeta a produção e, por consequência, a segurança alimentar. Cada doença tende a exibir padrões específicos de sintomas, o que auxilia na identificação e no diagnóstico da anormalidade. Esse monitoramento é vital para o gerenciamento agrícola, pois permite a tomada de decisões informadas que impactam diretamente a produção, possibilitando o estabelecimento de uma estratégia preventiva ou a mitigação das consequências (Liu *et al.*, 2017; Ebrahimi *et al.*, 2017).

Ainda, a automação de maquinários agrícolas tem se mostrado uma alternativa promissora para reduzir custos e aumentar a eficiência nas atividades de campo. A utilização de robôs para a detecção e eliminação de plantas daninhas é um exemplo concreto dessa abordagem, oferecendo maior precisão e agilidade nas operações. Modelos baseados em aprendizado profundo, como o YOLOv8, apresentam alta acurácia na identificação dessas plantas, viabilizando a capina automatizada em tempo real, mesmo diante das limitações computacionais inerentes ao ambiente agrícola (Bajraktari; Toyman, 2025). Além disso, o controle robótico localizado de plantas daninhas tem se destacado como uma estratégia eficaz na agricultura de precisão, promovendo a redução do uso de herbicidas e o consequente impacto ambiental. Estudos de campo demonstram que sistemas robóticos baseados em visão computacional e *Deep Learning* podem reduzir significativamente o consumo de defensivos, mantendo a eficácia do controle e contribuindo para a sustentabilidade das lavouras (Azghadi *et al.*, 2025).

A computação de borda, que consiste no processamento de dados próximo à fonte de geração ou coleta, surge como uma solução viável em contextos nos quais os dados podem ser processados no local da geração ou coleta. Esse paradigma pode ser combinado ao sensoriamento inteligente, proporcionando melhor desempenho, menor latência e redução de custos por meio do processamento local dos dados. A computação de borda permite o processamento de fluxos de dados sem a necessidade de enviá-los para um serviço centralizado, melhorando o tempo de resposta dos sistemas, economizando largura de banda de rede e melhorando a escalabilidade ao reduzir a dependência de serviços centralizados. Além disso, a computação de borda melhora a privacidade dos dados, pois nem todos os dados precisam ser enviados para a nuvem ou para um *data center* para processamento posterior. Atualmente, uma das principais demandas nesse cenário é a

aplicação de métodos de inteligência artificial. Embora esses métodos sejam frequentemente desenvolvidos para computadores, servidores e máquinas de alta potência, nem sempre são adequados para aplicações em tempo real. Nesse contexto, os computadores de placa única (SBC, do inglês *Single Board Computer*) surgem como uma solução eficaz, viabilizando o uso de modelos complexos de IA em campo e em tempo real, embora apresentem limitações relativas de recursos de processamento. Os algoritmos de *Machine Learning* (do português, Aprendizado de Máquina), treinados para alcançar alto desempenho, demandam uma quantidade significativa de memória durante o processo de treinamento. Por isso, uma implementação eficaz exige ajustes e modificações na arquitetura dos modelos, para garantir sua adequação às limitações de hardware e aos requisitos das aplicações em tempo real (Akhtar *et al.*, 2021; Maraveas, 2022; Biglari; Tang, 2023; Badiger, 2023; Grzesik; Mrozek, 2024).

Os SBCs constituem uma classe de dispositivos compactos que integram todos os componentes essenciais de um computador em uma única placa de circuito impresso. Essas soluções tecnológicas oferecem uma abordagem conveniente para uma ampla gama de aplicações, que vão desde projetos de hobby até sistemas complexos de controle industrial e embarcado. Um exemplo de SBC com GPU é o Jetson Nano, desenvolvido pela NVIDIA, projetado especificamente para atender às demandas de aplicações em inteligência artificial e *machine learning* em dispositivos embarcados. O Jetson Nano se destaca pelo seu alto desempenho em computação gráfica e processamento de dados, tudo em um formato compacto. Devido às suas dimensões reduzidas e ao grande poder computacional, esse dispositivo tem sido amplamente adotado em diversos projetos, incluindo, mas não se limitando a, visão computacional, robótica, automação industrial e veículos autônomos. Sua capacidade de executar modelos de *machine learning* e algoritmos de inteligência artificial diretamente no dispositivo o torna uma escolha adequada para aplicações que requerem baixa latência e processamento local de dados. Além disso, o uso crescente dos SBCs está associado à computação de borda, na qual a eficiência e a capacidade de processamento local são cruciais para a tomada de decisões ágeis e a otimização de recursos em ambientes dinâmicos (Singh, 2017; Assunção *et al.*, 2022; Shek *et al.*, 2023)

No trabalho de Sampurno *et al.* (2024), foi desenvolvido um sistema de visão inteligente para detectar plantas daninhas não cortadas dentro de fileiras de pomares, com o objetivo de automatizar o processo de remoção. Esse sistema visa superar as limitações dos cortadores de grama convencionais, que não conseguem operar efetivamente dentro das fileiras devido à estrutura confinada dos pomares. O sistema utiliza algoritmos de segmentação de instâncias YOLO e foi treinado em um conjunto de dados personalizado obtido de um pomar de peras. Quatro versões do modelo YOLO, com diferentes variantes na arquitetura, foram avaliadas em termos de precisão de detecção, complexidade do modelo e velocidade de inferência. Os resultados indicaram que as variantes mais simples foram mais eficientes.

Já no estudo de Trinh *et al.* (2024), foi proposta uma metodologia para a detecção eficiente de doenças nas folhas de arroz utilizando uma rede neural convolucional (CNN, do inglês *Convolutional Neural Network*). Uma variante do modelo YOLOv8 é apresentada, substituindo a função de perda original por uma combinação de EIoU loss e α -IoU loss (Zhang *et al.*, 2022)¹. O processo foi dividido em duas etapas: coleta e separação automática de imagens de diferentes doenças nas folhas de arroz, e implementação do modelo treinado em dispositivos IoT² (*Internet of Things*) para detecção e identificação das doenças. Os resultados experimentais demonstraram que o modelo proposto alcançou uma precisão de 89,9% em um conjunto de dados de 3175 imagens, superando métodos anteriores, incluindo versões anteriores do modelo YOLO.

No trabalho de Li, Güldenring e Nalpantidis (2023), foi proposta uma metodologia para robôs autônomos de capina detectarem com precisão o caule conjunto de ervas daninhas nas pastagens. Foram identificados pontos-chave nos caules articulados e caixas delimitadoras ao redor das plantas daninhas. Duas arquiteturas, uma de dois estágios e outra de estágio único, baseadas no detector de objetos YOLOv5, foram comparadas. Os resultados demonstram que a detecção de caules articulados como uma segunda tarefa melhora o desempenho em ambas as arquiteturas. A arquitetura de estágio único supera seus concorrentes, alcançando uma taxa de sucesso de 56,3% na detecção de caules articulados e operando em tempo real a 12,2 FPS (do inglês, *Frames Per Second* no Nvidia Jetson NX, adequado para uso em robôs agrícolas).

Esta dissertação explorou o uso de *Deep Learning* em SBC, visando aplicações para agricultura, com ênfase nos modelos YOLOv8 e YOLOv9. Os experimentos foram realizados no SBC NVIDIA Jetson Nano, pois esta possui GPU e condições de alimentação de energia para uso em campo. Para a criação dos conjuntos de dados de treinamento, adotou-se um procedimento semi-automatizado de rotulação; inicialmente, o modelo *Segment Anything Model* (SAM) foi utilizado para delimitar e identificar o objeto de interesse, seguido por uma verificação manual para assegurar a precisão dos segmentos. Esse método buscou agilizar a formação de conjuntos de dados robustos e diversificados, fundamentais para o treinamento eficaz dos modelos de *Deep Learning*.

¹ O EIoU loss (*Efficient Intersection over Union*) é uma função de perda que estende o IoU tradicional ao incorporar penalizações relacionadas ao deslocamento dos centros, discrepâncias nas dimensões (largura e altura) e desalinhamento nas proporções das *bouding boxes* preditas e reais, resultando em uma melhoria na conformidade geométrica das predições

² A Internet das Coisas (IoT) descreve a rede de objetos físicos incorporados a sensores, software e outras tecnologias visando conectar e trocar dados com outros dispositivos e sistemas pela internet. Esses dispositivos variam de objetos domésticos comuns a ferramentas industriais sofisticadas.

1.1 Objetivos

1.1.1 Objetivos Gerais

Avaliar a viabilidade do uso de modelos de *Deep Learning* em SBC para aplicações em agricultura.

1.1.2 Objetivos Específicos

- Definir um processo semiautomatizado para a construção de conjuntos de dados de treinamento, utilizando um modelo de segmentação, e realizar refinamentos manuais para assegurar a qualidade dos dados;
- Analisar o desempenho computacional e a eficiência da tarefa de detecção e segmentação, visando o uso de Computadores de Placa Única para aplicações em agricultura em campo;
- Investigar modelos de *Deep Learning* para aplicação na NVIDIA Jetson Nano e o seu desempenho para aplicações em agricultura.

1.2 Organização do documento

O documento está dividido em cinco capítulos. O Capítulo 2 traz uma revisão sobre a Agricultura 4.0, computação de borda e aceleradores GPU (*Graphics Processing Unit*). Também aborda o SBC NVIDIA Jetson Nano. Além disso, discute conceitos de inteligência artificial, *Deep Learning* e a arquitetura YOLO. O foco está na segmentação de imagens e nos modelos YOLOv8 e YOLOv9, com suas variantes. O Capítulo 3 apresenta os materiais e métodos usados, sendo detalhadas as especificações da câmera, conjuntos de dados e técnicas como aumento de dados, *Transfer Learning*, *Fine Tuning* e a estrutura QuadTree. O Capítulo 4 mostra os resultados obtidos, discutindo sobre o uso das arquiteturas pré-treinadas e o desempenho dos modelos YOLOv8 e YOLOv9. Foram realizadas comparações entre plataformas e os testes com imagens de cenário com instâncias da agricultura são apresentados. Esses testes foram feitos com a placa NVIDIA Jetson Nano. O Capítulo 5 traz as conclusões, destacando as contribuições dos experimentos e perspectivas futuras, sendo que o foco permanece em plataformas acessíveis para IA, principalmente segmentação de instâncias para aplicações em agricultura.

2 REVISÃO DA LITERATURA

2.1 Agricultura 4.0

A Agricultura 4.0 é caracterizada pela presença de dispositivos e sistemas inteligentes no campo, que proporcionam melhorias significativas no setor (Mourarias, 2021). Também chamada de agricultura digital, ela representa uma evolução da agricultura de precisão, integrando tecnologias digitais à gestão dos processos produtivos da agropecuária, incluindo a zootecnia. Nesse contexto, o campo teria passado por três revoluções, estando agora na quarta (Zhai *et al.*, 2020).

A primeira revolução no campo teria se iniciado por volta de 10.000 anos a.C., durante o período Neolítico, e teria sido marcada pelo uso da força bruta do animal e do homem, além do uso de ferramentas simples. A segunda revolução, no século XIX, foi marcada pela utilização de maquinário agrícola e de produtos químicos. Na terceira revolução, nos séculos XX e XXI, o surgimento do primeiro controlador lógico programável possibilitou o uso de softwares e o desenvolvimento de robôs para o campo. Atualmente, a quarta revolução seria marcada por sistemas e dispositivos inteligentes, por meio de tecnologias como a Internet das Coisas, o Big Data, a Inteligência Artificial, a computação em nuvem etc. Esta revolução possibilita a implementação das fazendas inteligentes, em que sensores e robôs realizam o monitoramento e o manejo das lavouras, dispensando a presença humana nestas tarefas (Fountas *et al.*, 2020; Mourarias, 2021).

Esse avanço é particularmente relevante para enfrentar desafios emergentes, como o aumento de doenças de plantas que, nos últimos anos, impactaram significativamente a produção agrícola e a segurança alimentar. Dessa forma, torna-se essencial monitorar e extrair rapidamente informações dos dados coletados, oferecendo suporte eficaz para a tomada de decisão. Por exemplo, a detecção precoce de doenças em plantas é vital para o gerenciamento agrícola, pois permite decisões fundamentais que impactam diretamente a produção (Liu *et al.*, 2017; Ebrahimi *et al.*, 2017).

Ainda, de acordo com Dutot, Nelson e Tyson (2013), as folhas das plantas são a principal fonte para identificar doenças, já que a maioria dos sintomas aparece nelas. Tradicionalmente, há dois métodos principais para essa identificação, ambos com suas limitações. O primeiro é a inspeção visual feita por especialistas, que exige conhecimento técnico e pode ser um processo demorado. O segundo é a detecção automatizada, baseada em processamento de imagens, que, embora eficiente, pode apresentar falhas em condições adversas ou quando os dados estão incompletos.

Dessa forma, verifica-se que a utilização de métodos baseados em captura por visão

computacional mostram-se extremamente úteis. Todavia, a principal limitação reside na execução desses modelos robustos em campo. Nesse contexto, os computadores de placa única, detalhado a seguir, emergem como uma solução.

2.2 Computador de Placa Única

Um computador de placa única (ou *Single-Board Computer* - SBC) é um sistema computacional contido numa única placa de circuito impresso (PCI). Os componentes fundamentais nas arquiteturas SBC incluem o núcleo do processador, os bancos de memória, os barramentos e as conexões para os periféricos. O barramento de dados interno desempenha um papel crucial na interconexão eficiente dos diversos blocos do sistema. As arquiteturas de processadores *multicore*, operando acima de 1GHz, tornaram-se características comuns nas plataformas SBC. Os SBCs ao contrário dos computadores pessoais convencionais, em sua maioria possui *slots* para funções periféricas ou de expansão. Isso se deve ao fato de que todos os componentes essenciais, como processadores, memória RAM, GPU e outras funcionalidades, estão integrados em uma única placa, o que impede a realização de atualizações nesses sistemas (Pajankar, 2017).

Com os avanços na tecnologia de eletrônica e semicondutores, os preços dos SBCs se tornaram bastante acessíveis. A relação custo-benefício é uma das características mais significativas desses dispositivos, que se mostram uma ferramenta de desenvolvimento adequada para uma variedade de aplicações, incluindo o desenvolvimento de novos produtos, depuração, testes e sistemas de automação (Freitas, 2018).

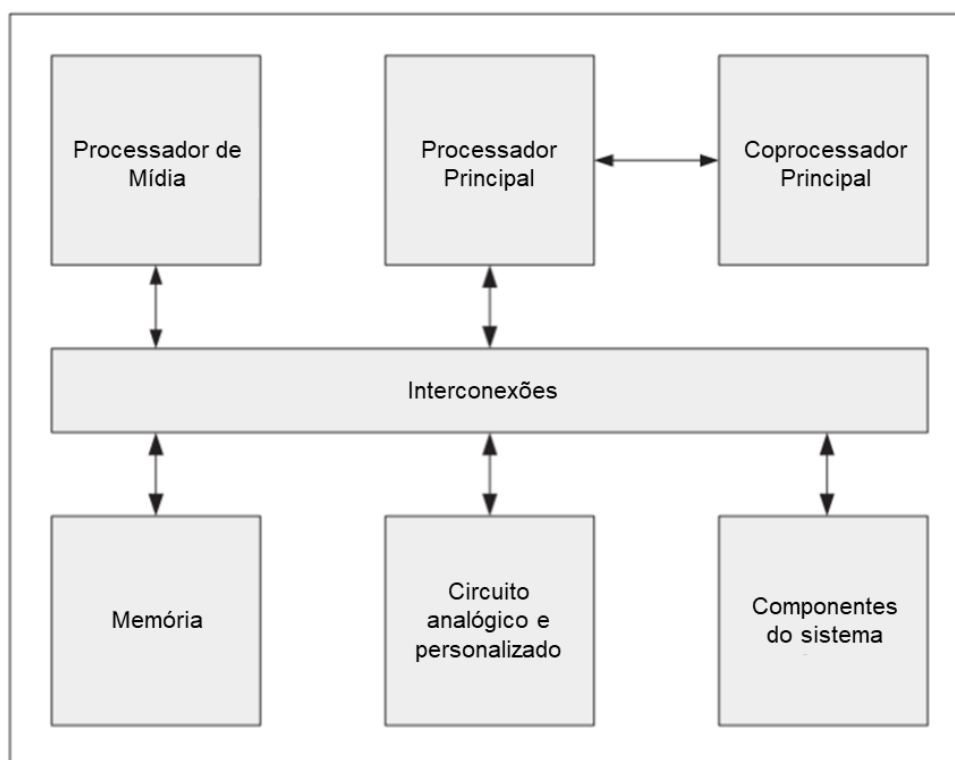
Uma característica proeminente dos SBCs é a adoção do sistema em um chip (*System-on-a-Chip*, SOC), que é um circuito integrado (CI) contendo todos os componentes essenciais de um computador ou outro sistema eletrônico em um único chip (Flynn; Luk, 2011), cuja a estrutura básica pode ser visualizada na Figura 1.

Os SoCs são amplamente empregados em dispositivos eletrônicos móveis devido à sua eficiência no consumo de energia e versatilidade. Esses chips são capazes de executar um sistema operacional completo, como o Linux. O progresso significativo dos SoCs acessíveis tem proporcionado aos desenvolvedores a oportunidade de utilizar linguagens de programação de alto nível para criar sistemas (Pajankar, 2017).

SoCs mais avançados geralmente incorporam uma GPU, dedicada ao processamento gráfico e otimizada para exibição de gráficos, vídeos e jogos. Além disso, os SoCs possuem controladores de periféricos que possibilitam a conexão com uma variedade de dispositivos externos, como câmeras, sensores e portas USB (Beasley; Clarke; Watson, 2020).

O Controlador de Memória gerencia o acesso eficiente à memória do sistema, distribuindo dados entre os diferentes blocos do SoC. Uma Unidade de Processamento de

Figura 1 – Um modelo básico de sistemas SoC.



Fonte: Adaptado de Flynn e Luk (2011).

Sinais (DSP, do inglês *Digital Signal Processing*) pode estar presente para o processamento de sinais analógicos e digitais, sendo comumente usada em aplicações de áudio, vídeo e sensores.

O SoC também incorpora um Controlador de Energia (PMIC, *Power Management Integrated Circuit*), que gerencia a distribuição de energia, otimizando o consumo de cada componente. Interfaces de comunicação, como UART³, SPI⁴ e I2C⁵, facilitam a interação do SoC com outros dispositivos, permitindo a troca de dados e comandos.

O bloco de segurança é uma adição comum em SoCs de dispositivos sensíveis, garantindo a proteção de dados e a integridade do sistema. Por fim, módulos de *clock* e gerenciamento de temporização sincronizam as operações entre os diferentes componentes do SoC, assegurando seu funcionamento correto. A integração eficiente desses componentes em um único chip permite que os SoCs ofereçam alto desempenho, eficiência energética e versatilidade em uma ampla gama de aplicações, que vão desde dispositivos móveis até

³ O UART é um protocolo de comunicação serial assíncrona amplamente utilizado em sistemas embarcados para transmissão de dados entre dispositivos, como sensores e microcontroladores, por meio de linhas de transmissão (TX) e recepção (RX).

⁴ O SPI é uma interface serial síncrona para a comunicação entre dispositivos periféricos. É comumente utilizado para conectar sensores, memórias flash, displays e outros dispositivos periféricos ao SoC.

⁵ O I2C é um barramento serial que permite a comunicação entre vários componentes utilizando apenas dois fios. Essa interface é frequentemente empregada para conectar sensores, EEPROMs, e outros dispositivos de baixa taxa de transferência.

sistemas embarcados e Internet das Coisas.

2.3 Aceleração na execução utilizando GPU e processamento paralelo

Buck (2007) relata que a computação de propósito geral utilizando GPUs emergiu como uma solução para resolver uma variedade de problemas intensivos em computação em tempo consideravelmente inferior quando comparada aos processadores convencionais.

A GPU é um componente eletrônico desenvolvido para manipular e otimizar a memória de maneira ágil, acelerando significativamente o processo de geração de imagens em um *buffer* (área de memória temporária usada para armazenar as imagens enquanto são processadas antes de serem exibidas) de quadros. Esse *buffer* permite que as informações visuais sejam carregadas e atualizadas rapidamente, resultando em uma transição suave para o dispositivo de exibição. Tipicamente, as GPUs atuam como coprocessadores especializados na execução eficiente de tarefas relacionadas à renderização de informações gráficas bidimensionais e tridimensionais, resultando em uma apresentação visual em um monitor ou tela (Florencio, 2020).

O trabalho de Owens *et al.* (2008) detalha as etapas para o processamento e aceleração utilizando GPU. Ele explica que a entrada para uma GPU consiste em uma lista de primitivos geométricos, geralmente triângulos, expressos em um sistema de coordenadas tridimensional global. Esses primitivos geométricos passam por um processo de sombreamento e são mapeados na tela, onde são combinados para formar a imagem final. As GPUs modernas representam uma evolução significativa em relação ao tradicional *pipeline* gráfico de função fixa, transformando-se em processadores paralelos programáveis com capacidade computacional superior à encontrada em CPUs (*Central Processing Unit*, ou Unidade Central de Processamento) *multicore*. Nesse contexto, a GPU transcendeu sua função original de unidade de processamento dedicada à aceleração gráfica, assumindo o papel de um processador programável caracterizado por notável paralelismo e elevada largura de banda de memória (Lindholm *et al.*, 2008).

A GPU distribui os recursos do processador entre diferentes estágios, implementando uma abordagem espacial na divisão do *pipeline*, ao contrário da abordagem temporal típica das CPUs. Cada parte do processador que atua em um estágio específico direciona sua saída diretamente para a próxima etapa do *pipeline*.

No cenário atual, observa-se a existência de um extenso ecossistema de *GPU Computing*, abrangendo centenas de milhões de GPUs compatíveis com a arquitetura CUDA (*Compute Unified Device Architecture*), desenvolvida pela NVIDIA. Este ambiente tem sido explorado por diversos desenvolvedores e pesquisadores que empregam GPUs em uma

variedade de aplicações que demandam processamento paralelo, incluindo, notadamente, o *Deep Learning* (DL) (Ye *et al.*, 2024). A CUDA é uma plataforma de computação paralela e um modelo de programação que permite aumentos significativos de desempenho ao explorar a potência das unidades de processamento gráfico (GPUs). Assim, o programa escrito para esse modelo define como os componentes da aplicação compartilham informações e coordenam suas atividades. O compilador é responsável por mapear essa estrutura lógica em instruções para o *hardware* específico, utilizando bibliotecas que acessam primitivas do *hardware* e do sistema operacional (Garland *et al.*, 2008; Quadros, 2016).

Essa capacidade de processamento é diretamente influenciada pelas microarquitecturas das GPUs, que, conforme destacado por Hennessy e Patterson (2011), referem-se à organização interna dos componentes de um computador. Por exemplo, diferentes microarquitecturas podem implementar o mesmo conjunto de instruções x86, mas com variações na organização do *pipeline* e do cache. Com cada nova geração, a NVIDIA modifica a organização das GPUs, adaptando-as para executar instruções de maneira ainda mais eficiente. Essas evoluções permitem otimizações importantes para aplicações de inteligência artificial, conforme detalhado por Reuther *et al.* (2019).

2.4 Computação de Borda

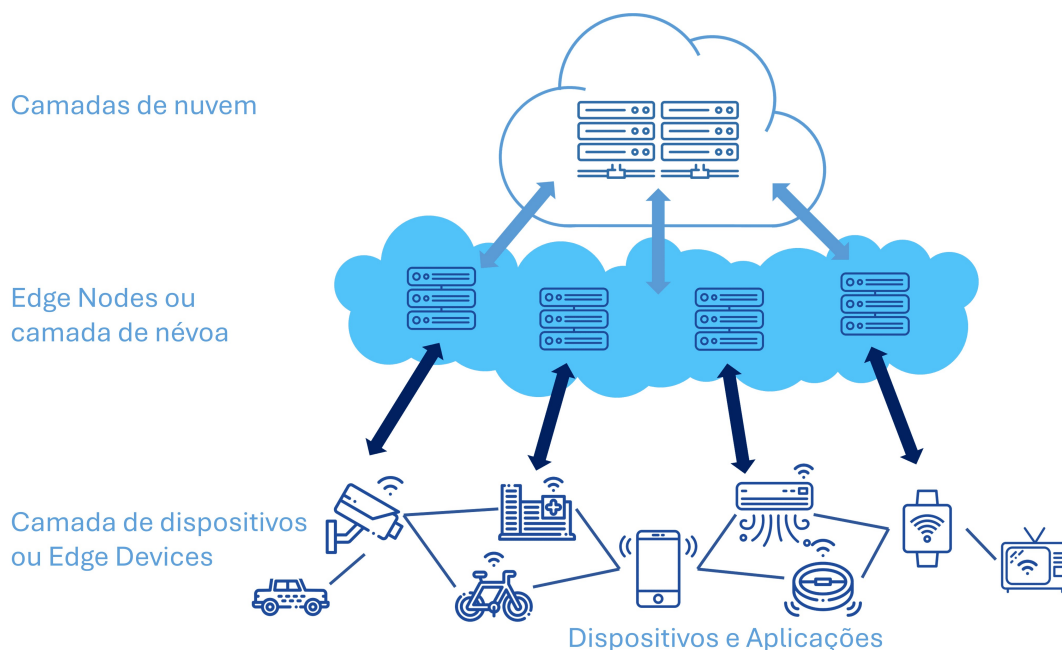
Considerando a alta latência e as preocupações de privacidade associadas à computação em nuvem, muitos reconhecem a computação de borda (*Edge Computing*) como uma alternativa viável. Nessa abordagem, o processamento de dados é transferido para dispositivos localizados na periferia da rede (Tanwar; Tanwar; James, 2021).

Essa estratégia visa atender às necessidades de processamento com alta latência. Entretanto, em casos em que a aplicação requer o processamento de grandes volumes de dados, surge a necessidade de armazenamento adicional. A arquitetura resultante pode ser observada na Figura 2, que ilustra a divisão em três partes essenciais: a camada de dispositivo, a camada de névoa e a camada de nuvem (Moreira, 2021).

O processamento nas camadas de dispositivos reduz significativamente o volume de dados entre dispositivos e o núcleo de processamento, diminuindo, conseqüentemente, o tráfego na rede, os custos associados a isso e melhorando a qualidade do serviço (QoS, do inglês *Quality of Service*). A Computação de Borda elimina, ou pelo menos reduz, a carga de trabalho do *middleware*⁶, que pode se tornar um ponto único de falha caso haja centralização (Chang *et al.*, 2014).

Como o volume de dados que trafega pela rede é significativamente reduzido, há

⁶ *Middleware* é um tipo de *software* intermediário que atua como uma ponte entre diferentes aplicações, sistemas operacionais ou serviços, facilitando a comunicação, a troca de dados e o gerenciamento de processos entre eles.

Figura 2 – Arquitetura de um sistema utilizando *edge computing*.

Fonte: Autor.

a possibilidade de criptografá-lo para o tráfego na rede, sendo posteriormente descryptografado pelo *middleware*. Caso o tipo de criptografia não seja reconhecido, os dados são descartados devido à possibilidade de serem parte de um ataque. Dessa forma, a segurança do sistema também é beneficiada. Outra vantagem é a escalabilidade do sistema, que, por estar em um ambiente distribuído, também é beneficiada (Lewis *et al.*, 2014). A integração com Inteligência Artificial (IA) amplia ainda mais o potencial da computação de borda, permitindo que os dispositivos no campo analisem dados e tomem decisões de forma autônoma. Na próxima seção, será mencionada a utilização de IA nesse contexto.

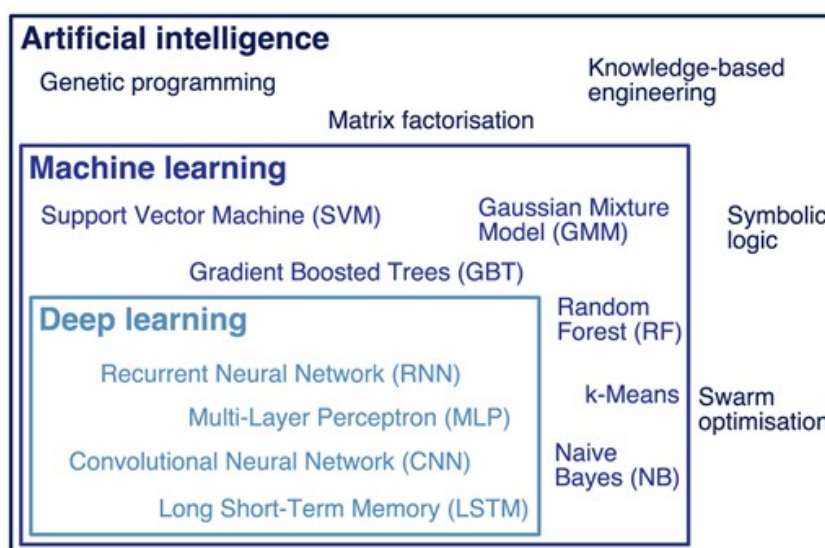
2.5 Inteligência Artificial

A Inteligência Artificial (IA) consiste em um amplo campo da Ciência da Computação que se concentra na criação de sistemas capazes de realizar tarefas que, tradicionalmente, exigiriam inteligência humana. O objetivo da IA é criar sistemas capazes de compreender o ambiente, tomar decisões apropriadas para alcançar objetivos e aprender com a experiência (Silva; Spatti; Flauzino, 2016).

Dentro dessa área, encontra-se o *Machine Learning*, um subcampo da IA que se concentra no desenvolvimento de algoritmos e modelos que permitem aos computadores aprender a partir de dados e melhorar seu desempenho em uma tarefa específica ao longo do tempo, sem necessidade de programação explícita. O *Deep Learning* é uma subárea do *Machine Learning* que utiliza Redes Neurais Artificiais (RNAs) com muitas camadas

(profundas) para aprender padrões complexos (Rubbens *et al.*, 2023). Esta técnica foi a principal abordagem utilizada na presente dissertação de mestrado, a qual é descrita mais detalhadamente a seguir. Uma imagem de representação do contexto de cada um dos termos é apresentada na Figura 3. O *Deep Learning* será detalhado na próxima seção.

Figura 3 – Exemplo do contexto do *Deep Learning*, *Machine Learning* e *Artificial Intelligence*.



Fonte: Rubbens *et al.* (2023).

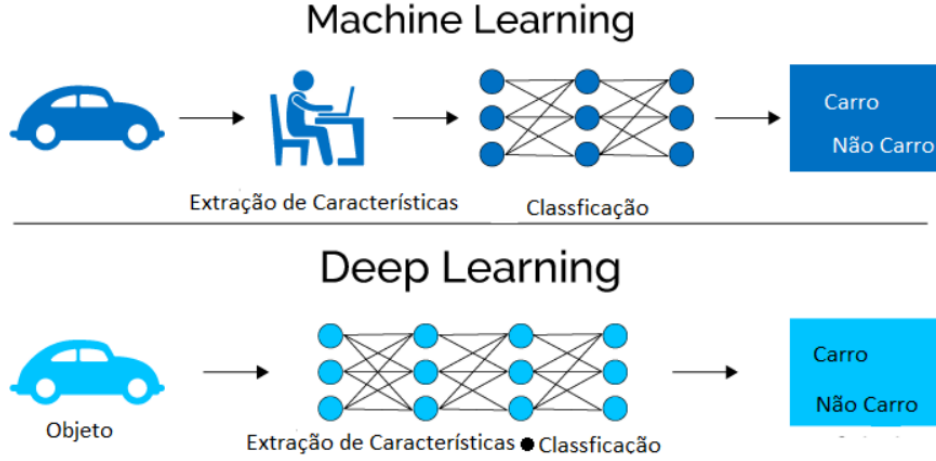
2.5.1 *Deep Learning*

Deep Learning é uma subárea de *Machine Learning* que utiliza algoritmos para processar dados, baseado no processamento realizado pelo cérebro humano. Essa abordagem é materializada em RNAs com um nível mais elevado de complexidade. A particularidade dessa solução reside na capacidade de permitir que computadores aprendam a partir de experiências anteriores, capacitando-os a compreender o mundo por meio de uma hierarquia de conceitos. Nessa estrutura, os conceitos mais complexos são definidos e compreendidos em relação aos mais simples, conforme destacado por Goodfellow, Bengio e Courville (2016).

O diferencial das técnicas de *Deep Learning* em relação a outros métodos está em sua habilidade de lidar eficazmente com grandes volumes de dados. Além disso, essa abordagem possui a flexibilidade para adicionar camadas para a extração de características, conforme apresentado na Figura 4. Essa capacidade de lidar com a complexidade e de se adaptar a diferentes níveis de abstração é fundamental para a eficácia do *Deep Learning* em diversas aplicações. Ao abordar um problema de classificação simples, a aplicação de *Machine Learning* implica a necessidade de extrair características da imagem antes de aplicar um modelo de classificação. Em contrapartida, no contexto do *Deep Learning* (DL), é possível eliminar a etapa prévia de extração de características, pois o próprio modelo,

após o treinamento, é capaz de realizar tanto a extração quanto a classificação (Ponti; Costa, 2018).

Figura 4 – Comparativo entre *Machine Learning* e *Deep Learning*.



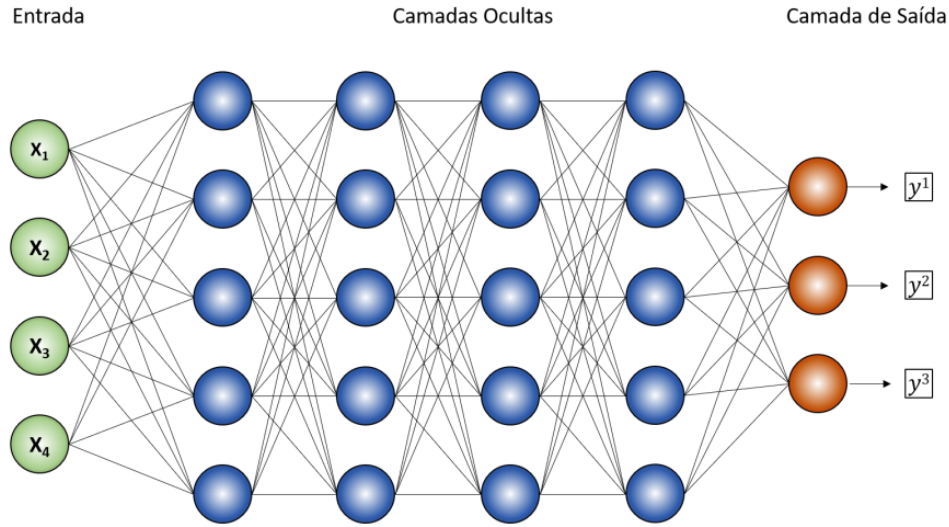
Fonte: Ponti e Costa (2018).

Uma distinção adicional entre os modelos de *Machine Learning* e *Deep Learning* refere-se à arquitetura da RNA. Redes Neurais simples, frequentemente denominadas de Redes Rasas (*Shallow Networks*), buscam uma única função $f(\cdot)$ a partir de um conjunto de parâmetros para gerar o resultado desejado. Essa arquitetura é justificada pela estrutura das RNAs, que geralmente possuem apenas uma camada oculta, isto é, uma camada intermediária entre a entrada e a saída, responsável por transformar os dados e extrair características relevantes, embora seus valores não sejam diretamente observáveis, conforme ilustrado na Figura 4. Em contrapartida, no contexto de *Deep Learning* (DL), a aprendizagem da função $f(\cdot)$ ocorre por meio de composições de funções, conforme ilustrado na Equação (2.1). Cada função $f_i(\cdot)$ opera sobre um vetor x_i de entrada, gerando um novo vetor x_{i+1} como saída. O índice i refere-se a cada camada da rede. Essa abordagem é fundamental para redes profundas, que possuem múltiplas camadas ocultas, como representado na Figura 5. Assim, uma das ideias centrais em *Deep Learning* consiste em aprender representações sucessivas dos dados (X), ou seja, os vetores $x_i|_{i=1}^n$ intermediários, ao longo dessas camadas (Ponti; Costa, 2018).

$$f(x) = f_i(\dots f_2(f_1(x_1)\dots)) \quad (2.1)$$

Matematicamente, essa sequência de transformações pode ser interpretada como um processo capaz de desdobrar as diversas variedades presentes nos dados originais. Em outras palavras, as estruturas complexas dos dados, inicialmente entrelaçadas, são desembaraçadas por meio dessas transformações sequenciais.

Figura 5 – Rede Neural Profunda.



Fonte: Ponti e Costa (2018).

2.5.2 Processo de ajuste dos parâmetros de uma RNA

O processo de ajuste de uma RNA possui basicamente duas fases. A primeira fase é a etapa de aprendizagem, também conhecida como processo de treinamento, que consiste em três passos: a RNA é estimulada pelo ambiente, ou seja, diante de suas entradas, ela sofre modificações em seus parâmetros e, com isso, passa a responder de maneira diferente ao meio ambiente, devido às modificações sofridas (Goodfellow; Bengio; Courville, 2016).

Os parâmetros atualizados durante o treinamento correspondem aos hiperparâmetros das camadas densas, que são estruturas nas quais cada neurônio está conectado a todos os neurônios da camada anterior. Esses hiperparâmetros incluem os pesos sinápticos, que representam a força das conexões entre os neurônios, e os limiares (ou biases), que determinam o nível de ativação necessário para que um neurônio dispare. Outra fase é a inferência, na qual a RNA é destinada a realizar a classificação e o reconhecimento, conforme Silva, Spatti e Flauzino (2016). Considerando a Teoria Hebbiana sobre a plasticidade sináptica, descrita por Hebb (1949), para o treinamento de neurônios, tem-se que, no processo, quando a saída produzida pela rede coincide com a saída desejada, os pesos sinápticos e limiares são incrementados (ajuste excitatório) proporcionalmente aos valores de seus sinais de entrada. Se a saída for diferente do valor desejado, ocorre o ajuste inibitório, que decrementa os pesos sinápticos e o limiar. Assim, a Equação (2.2) mostra como as regras de aprendizagem são realizadas, na qual η representa a taxa de aprendizagem, ou seja, a sensibilidade com que os pesos variam em cada atualização.

$$w_i^{atual} = w_i^{anterior} + \eta \cdot (t - y) \cdot x_i \quad (2.2)$$

A variável t representa os *targets*, ou seja, as saídas esperadas. Os w_i são os pesos

sinápticos, a variável y é a saída de cada neurônio e a variável x_i representa os dados de entrada para um neurônio.

O cálculo do erro em uma rede neural artificial (RNA) torna-se mais complexo à medida que se adicionam neurônios e camadas extras, pois se torna mais difícil identificar quais pesos sinápticos contribuem para os erros. Para lidar com essa questão, é utilizado o método de *backpropagation*, no qual os erros são propagados para trás através da rede. O cálculo por *backpropagation* do erro é feito pela função soma do quadrado dos erros, que garante a atualização dos pesos sinápticos sempre que necessário. Essa função é descrita na Equação (2.3).

$$E(t, y) = \frac{1}{2} \sum_{k=1}^N (t_k - y_k)^2 \quad (2.3)$$

Na expressão, o N é o número de nós existentes na camada de saída, o y_k são as saídas de cada nó e o t_k são os *targets*. Por trás dessa regra de aprendizagem está a minimização do erro de rede por descida de gradiente, utilizando-se a derivada da função de erro buscando reduzir o erro. Trata-se de um processo de otimização, que consiste em adaptar os valores dos pesos sinápticos para minimizar a função de erro.

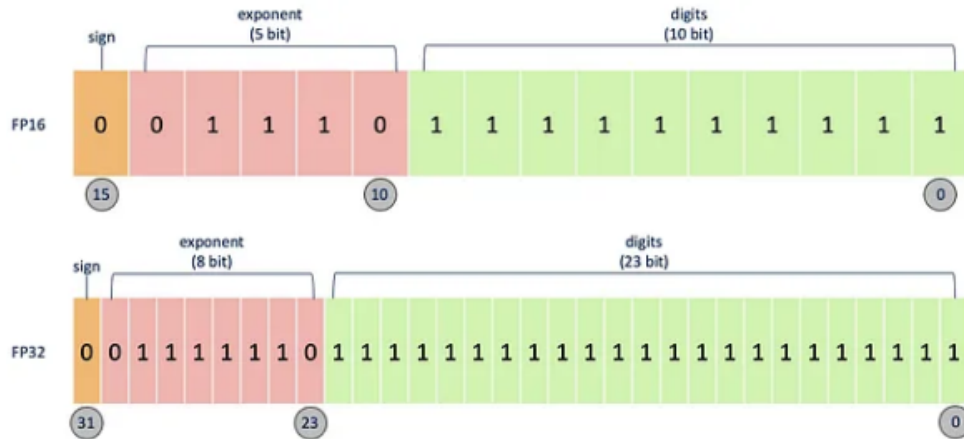
2.5.3 Precisão no treinamento de Redes Neurais Artificiais

Conforme discutido por Micikevicius *et al.* (2017), o artigo introduz uma metodologia para o treinamento de redes neurais profundas utilizando números de ponto flutuante de meia precisão (FP16), sem comprometer a precisão do modelo em níveis significativos ou exigir a modificação dos hiperparâmetros. Essa abordagem resulta em uma redução de quase 50% nos requisitos de memória e, em GPUs mais recentes, acelera as operações aritméticas.

No cenário convencional de treinamento de Redes Neurais Artificiais, é comum o uso de FP32, que proporciona maior precisão na representação dos dados, mas demanda uma quantidade significativamente maior de memória para armazená-los. Micikevicius *et al.* (2017) propõe a adoção de FP16 para o armazenamento de pesos sinápticos, funções de ativação e gradientes durante as iterações do treinamento. A Figura 6 ilustra esses formatos em representação binária.

Ao adotar o FP16, observa-se uma redução no intervalo de valores que podem ser armazenados, o que resulta em uma diminuição da precisão, especialmente quando os números se aproximam de valores muito pequenos em módulo. Esse efeito está relacionado ao limite da precisão do formato de ponto flutuante utilizado. Como consequência, ocorre uma redução na acurácia do modelo, já que os cálculos são limitados pela precisão dos valores de ponto flutuante, o que pode impactar negativamente o desempenho do

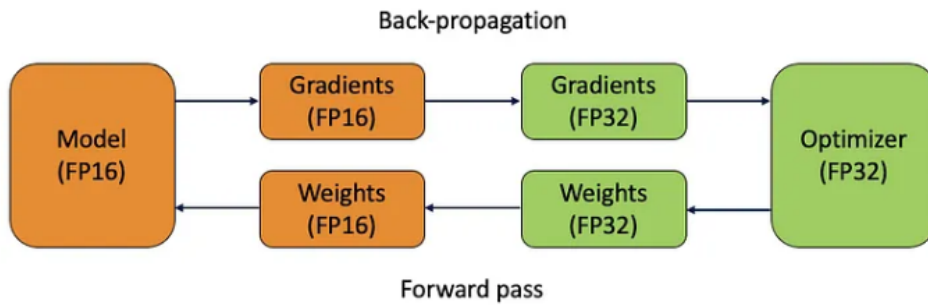
Figura 6 – Comparação do formato dos números de ponto flutuante do FP16 e do FP32.



Fonte: Ahmed e Naveed (2021).

modelo. Todavia, a metodologia, proposta por Micikevicius *et al.* (2017), foi planejada para mitigar tais efeitos, de modo a assegurar que eventuais perdas de precisão não comprometessem de forma significativa os resultados obtidos (Anagnostopoulos, 2022). Diante dessa consideração, essa abordagem consiste em armazenar uma cópia mestre dos pesos sinápticos em FP32 e convertê-la para FP16 a cada iteração do treinamento. Ao término da iteração, os gradientes de peso obtidos são utilizados para atualizar os pesos sinápticos mestres durante a etapa do otimizador. Essa metodologia é ilustrada na Figura 7.

Figura 7 – Utilização de FP16 e FP32 durante a etapa de treinamento de uma RNA.



Fonte: Ahmed e Naveed (2021).

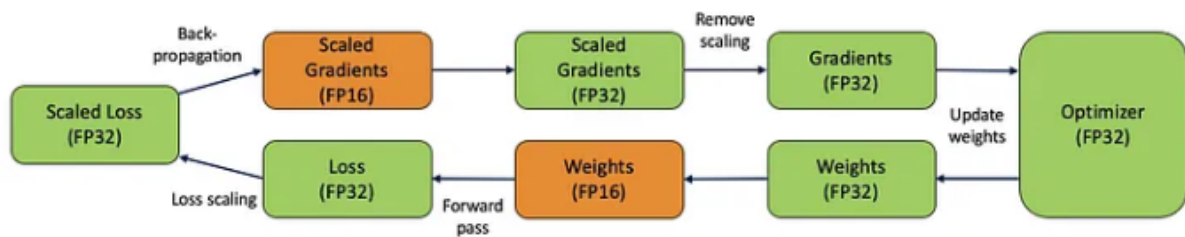
Dessa forma, ao adotar a estratégia de manter uma cópia FP32 dos pesos sinápticos, obtêm-se benefícios. Com isso, a redução do consumo de memória, torna-se viável alocar recursos adicionais para outros elementos do modelo, como aumentar o tamanho da arquitetura, o tamanho do conjunto de amostras ou até mesmo o tamanho dos dados de entrada. Em virtude da taxa de aprendizagem ser frequentemente pequena, a multiplicação pelos gradientes de peso pode resultar em valores muito pequenos. No contexto do FP16, qualquer número com magnitude inferior a 2^{-24} é equiparado a zero, pois não pode ser representado. Assim, ao concluir as atualizações no FP32, esses valores podem ser

preservados (Ahmed; Naveed, 2021; Anagnostopoulos, 2022).

Micikevicius *et al.* (2017) sugere a aplicação do escalonamento de perda, um processo no qual o valor da perda é multiplicado por um fator de escala após a etapa de *feed forward* e antes da *back propagation*. Tal procedimento se justifica pela ocorrência, em alguns casos, de pequenos valores de gradiente antes mesmo de serem multiplicados pela taxa de aprendizado.

Nesse contexto, emprega-se a regra da cadeia para estabelecer que todos os gradientes sejam posteriormente dimensionados pelo mesmo fator. Em outras palavras, uma vez calculados, os gradientes podem ser divididos pelo mesmo fator de escala antes de serem utilizados para atualizar os pesos sinápticos mestres em FP32. A aplicação prática da regra da cadeia é ilustrada na Figura 8.

Figura 8 – Dimensionamento de perdas por fatores pré-definidos baseado na utilização de FP16 ou FP32.



Fonte: Ahmed e Naveed (2021).

Em 2018, a NVIDIA lançou uma extensão para o PyTorch denominada Apex, que incorporava a capacidade de *Automatic Mixed Precision* (AMP). Essa extensão ofereceu uma solução simplificada para a implementação de treinamento de precisão mista no PyTorch. Com apenas algumas linhas de código, tornou-se possível migrar o treinamento de FP32 para a precisão mista na GPU. Essa transição permite reduzir o tempo de treinamento em uma faixa que varia de 1,5 a 5,5 vezes, sem comprometer significativamente o desempenho do modelo, conforme demonstrado em Li, Güldenring e Nalpantidis (2023).

2.6 Arquitetura YOLO

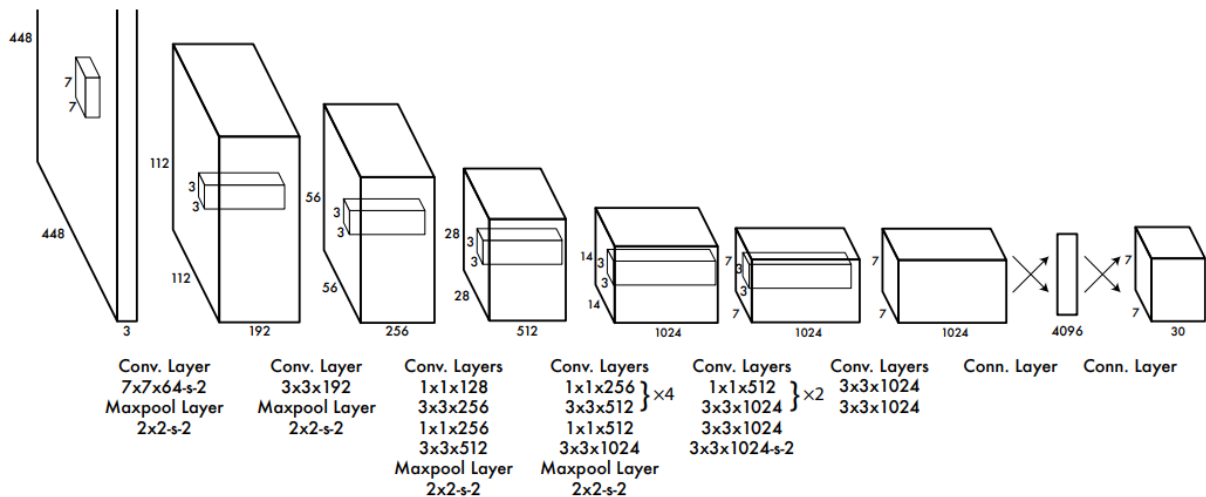
A YOLO (*You-Only-Look-Once*) é um algoritmo de detecção de objetos em tempo real que combina o que antes era um processo de várias etapas, utilizando uma única RNA para realizar classificação e predição de *bounding boxes* para objetos detectados (Terven; Córdova-Esparza; Romero-González, 2023). Nessa dissertação, optou-se por utilizar YOLO ao invés de outras arquiteturas devido à sua alta eficiência e rapidez na detecção de objetos, o que é crucial para aplicações em tempo real. O nome YOLO deriva de sua capacidade de analisar a imagem na totalidade, permitindo assim capturar o contexto dos objetos

detectados. Isso reduz o número de detecções falsas positivas em comparação com as Redes Convolucionais Recorrentes (R-CNNs), que analisam diferentes partes da imagem separadamente. Além disso, a YOLO pode generalizar as representações de vários objetos, tornando-a mais aplicável a uma variedade de novos ambientes (Jiang *et al.*, 2022).

A YOLO é otimizada para o desempenho de detecção e classificação e pode ser executada mais rapidamente do que a execução de duas redes neurais separadas para detecção e classificação de objetos individualmente. Isso é alcançado ao redirecionar os classificadores de imagens tradicionais para a tarefa de regressão, visando identificar *bounding boxes* para objetos. Essa rede segmenta a imagem em regiões e simultaneamente prevê as *bounding boxes* e as probabilidades associadas às classes dos objetos em cada região. Essa abordagem resulta em uma maior velocidade durante o processo de detecção. Apesar do aumento significativo na velocidade de detecção, a YOLO apresenta uma diminuição na precisão da localização de pequenos objetos, principalmente quando comparada com arquiteturas de dois estágios (Redmon *et al.*, 2016; Terven; Córdova-Esparza; Romero-González, 2023). No entanto, as versões mais recentes da YOLO resolvem esse detalhe.

A arquitetura original da YOLO é apresentada na Figura 9. Esta rede de detecção possui 24 camadas convolucionais seguidas por 2 camadas totalmente conectadas. Camadas convolucionais alternadas de 1×1 reduzem o espaço de características das camadas anteriores. As camadas convolucionais são pré-treinadas na tarefa de classificação do ImageNet com metade da resolução (imagem de entrada de 224×224) e depois a resolução é dobrada para a detecção (Redmon *et al.*, 2016).

Figura 9 – Arquitetura da rede YOLO original.



Fonte: Redmon *et al.* (2016)

A YOLO se baseia na ideia de dividir uma imagem em imagens menores, sendo dividida em uma grade quadrada de dimensões $S \times S$. Assim, cada elemento da grade em que reside o centro de um objeto é responsável por detectar esse objeto. Cada uma

das células irá prever *bounding boxes* B e um *score* de confiança para cada *bounding box*. Assim, o *score* da classificação varia de 0,0 a 1,0, sendo que 0,0 representa o menor nível de confiança e 1,0 o maior. Em outras palavras, se o objeto não existir na célula, o nível de confiança deve ser o menor, enquanto, se o modelo estiver completamente certo da predição, o valor deve ser o maior (Redmon *et al.*, 2016).

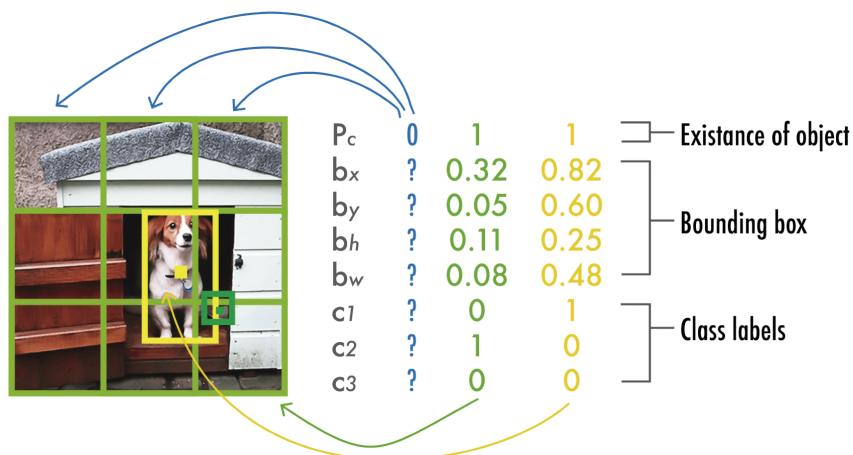
O nível de confiança indica a probabilidade de a *bounding box* conter um objeto. Se o nível de confiança for baixo, próximo de 0,0, isso sugere que o modelo não tem certeza de que há um objeto presente ou que a caixa está mal posicionada. Se o nível de confiança for alto, próximo de 1,0, significa que o modelo está confiante de que um objeto está presente e de que a *bounding box* está corretamente ajustada à sua posição. Portanto, cada uma das *bounding boxes* é composta por 5 valores: a posição b_x , a posição b_y , a largura b_h , a altura b_w e a confiança P_c . As coordenadas, por sua vez, representam a localização do centro da *bounding box* prevista, e a largura e altura são frações relativas ao tamanho inteiro da imagem (Redmon *et al.*, 2016). Contudo, o nível de confiança por si só não é suficiente para avaliar a qualidade da detecção. Para isso, utiliza-se também a métrica IoU (*Intersection over Union*), que quantifica o grau de sobreposição entre a *bounding box* prevista e a caixa real. A IoU é definida como a razão entre a área de interseção da *bounding box* prevista com a caixa real e a área total da união entre ambas. Esse valor varia entre 0 e 1, sendo utilizado como critério para validar se uma detecção pode ser considerada correta (Rezatofighi *et al.*, 2019).

A imagem é dividida em uma grade $S \times S$. Cada quadrado dessa grade é chamado de célula. Para cada célula, o YOLO prevê as *bounding boxes*, os respectivos escores de confiança e a classe do objeto. A predição de classe é representada por um vetor de tamanho igual ao número de classes C . Apesar de cada célula poder prever múltiplas caixas delimitadoras e seus *scores* de confiança, ela prevê apenas uma única classe, o que constitui uma limitação do algoritmo: se houver objetos de classes diferentes dentro da mesma célula, o modelo não conseguirá classificá-los corretamente. Assim, cada saída por célula tem a forma $C + B \times 5$, onde B é o número de caixas delimitadoras previstas. B é multiplicado por 5 porque inclui b_x, b_y, b_w, b_h, P_c para cada caixa. Como existem $S \times S$ células na grade, a saída final do modelo é um tensor $S \times S \times (C + B \times 5)$ (Redmon *et al.*, 2016).

Na Figura 10 é apresentado um exemplo de saída do modelo ao prever apenas uma única caixa delimitadora por célula. Nessa imagem, cada célula da grade faz previsões sobre a presença de objetos e suas características. A imagem original mostra um cachorro em frente a uma casa de cachorro, dividida em uma grade de 3×3 (representado em verde), onde cada célula é responsável por prever se há um objeto dentro dela e suas características. Cada célula realiza previsões em três categorias principais: a existência de objeto (P_c), a caixa delimitadora (*bounding box*) (representado em amarelo) e as etiquetas de classe

(*class labels*).

Figura 10 – Exemplo de uma previsão de uma rede YOLO.



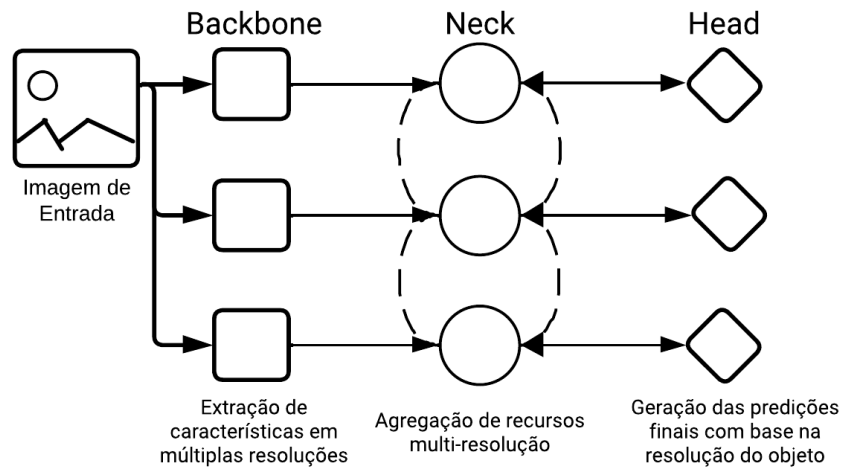
Fonte: Terven, Córdoba-Esparza e Romero-González (2023)

A arquitetura de um modelo das arquiteturas modernas são compreendidas em três componentes principais: o *backbone*, o *neck* e o *head*, como apresentado na Figura 11. A componente *backbone* é encarregada de extrair características das imagens de entrada durante o treinamento. Como essa etapa pode ser computacionalmente intensiva, é comum utilizar modelos previamente treinados em grandes conjuntos de dados, diversos e representativos, aplicando a técnica conhecida como *Transfer Learning*. Esse processo consiste em aproveitar os conhecimentos adquiridos por uma rede neural já treinada, como a capacidade de identificar padrões visuais básicos, incluindo bordas, texturas e formas, e aplicá-los em uma nova tarefa. Dessa forma, a rede inicia o treinamento com uma base sólida de extração de características, sendo necessário apenas ajustá-la ao domínio ou contexto específico desejado (Wu *et al.*, 2019).

O *neck* é um componente intermediário que conecta o *backbone* à *head*, sendo responsável por agregar e refinar as características extraídas pelo *backbone*. Essa etapa frequentemente tem como foco a melhoria das informações espaciais e semânticas em diferentes escalas. Para isso, podem ser utilizadas camadas convolucionais adicionais, FPNs (*Feature Pyramid Networks*) ou outros mecanismos voltados ao aprimoramento da representação das características. A FPN, ilustrada na Figura 12, é uma arquitetura projetada para melhorar a predição de objetos e o desempenho na extração de características das imagens. Ela se beneficia de múltiplas camadas convolucionais para construir uma representação piramidal dos características, possibilitando uma predição mais eficaz em diferentes níveis de escala (Lin *et al.*, 2017a; Terven; Córdoba-Esparza; Romero-González, 2023).

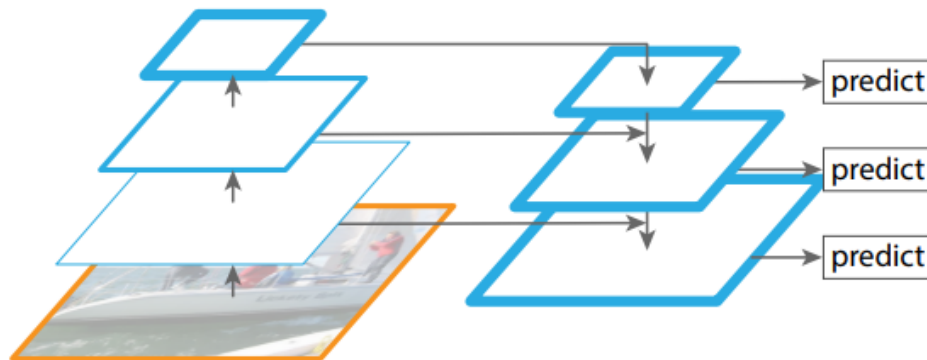
A componente *head* representa a saída final do modelo e está associada à classificação. No contexto da detecção de objetos, os autores ressaltam que essa camada geralmente

Figura 11 – Diagrama de alto nível de uma arquitetura moderna que usa *Backbone*, *Neck* e *Head*.



Fonte: Autor.

Figura 12 – Arquitetura de uma FPN para detecção de objetos em múltiplas escalas.



Fonte: Lin *et al.* (2017a)

considera a qualidade e a distribuição das previsões da rede neural, combinando-as com outras fontes de verdade para calcular e otimizar a definição das classes. No caso do YOLO, essa utiliza métricas de regressão para a predição das *bounding boxes* a fim de definir as classes dos objetos (Bochkovskiy; Wang; Liao, 2020).

2.6.1 A evolução da YOLO

A primeira versão da YOLO foi lançada em 2015, introduzindo uma abordagem inovadora para detecção de objetos. Esta versão tratava o problema como uma regressão para *bounding boxes* espacialmente separadas e com probabilidades de classe associadas. Ao final do processo, uma etapa de pós-processamento, como a supressão não máxima (*Non-Maximum Suppression* – NMS), é aplicada para filtrar as *bounding boxes* sobrepostas e reter apenas as detecções mais confiáveis (Terven; Córdova-Esparza; Romero-González, 2023).

Posteriormente, a YOLOv2 ou YOLO9000, proposta por Redmon e Farhadi (2017), trouxe melhorias em relação à versão original. Entre as principais melhorias, destaca-se a inclusão de normalização de lote, classificadores de alta resolução, caixas de ancoragem, entre outras. O principal objetivo era aumentar tanto a velocidade quanto a precisão da predição.

A YOLOv3, proposta em Redmon e Farhadi (2018), refinou ainda mais a arquitetura e os métodos utilizados no treinamento. Incorporou redes de pirâmides de características (FPN, do inglês *Feature Pyramid Networks*) (Lin *et al.*, 2017b) e a previsão em diferentes escalas para melhorar o desempenho de detecção, principalmente para objetos pequenos. Além disso, introduziu múltiplas escalas para detecção e superou a precisão das versões anteriores.

Já a YOLOv4, proposta por Bochkovskiy, Wang e Liao (2020), trouxe melhorias significativas de velocidade e precisão em relação ao seu antecessor. Esta versão concentrou-se em melhorar o *backbone* da rede, incorporando várias técnicas de última geração, como o uso do CSPDarknet53 como *backbone*, a função de ativação Mish (Misra, 2019) e a introdução das Conexões Residuais Ponderadas (WRC, do inglês *Weighted Residuals Connections*) (Shen; Gan; Zeng, 2016), além de outras novas abordagens para aumentar o desempenho. Maiores detalhes podem ser obtidos em Misra (2019). No entanto, este ano marcou o momento em que Joseph Redmon, um dos autores originais da YOLO, deixou a pesquisa em visão computacional.

Dois meses após o lançamento da YOLOv4, Jocher *et al.* (2022) introduziram a YOLOv5 na empresa Ultralytics, marcando um avanço significativo na série YOLO. Embora a YOLOv5 não tenha sido uma continuação direta do trabalho dos criadores originais, tornou-se uma das versões mais populares na comunidade de código aberto. Essa versão foi otimizada e simplificada, com foco na compatibilidade, tornando o modelo mais acessível e fácil de implementar em diversas aplicações. Além disso, foi projetada para ser mais modular e flexível. Diferentemente das versões anteriores, que utilizavam o *framework* Darknet, a YOLOv5 foi implementada em PyTorch, facilitando sua integração em ambientes modernos de aprendizado profundo.

Em Li *et al.* (2022), foi lançada a YOLOv6, que alcançou o melhor desempenho de precisão entre outros detectores, mantendo uma velocidade de inferência semelhante em uma NVIDIA Tesla T4. Por outro lado, em Wang, Bochkovskiy e Liao (2022), foi lançada a YOLOv7, introduzindo três novos pontos importantes: ELAN (*Efficient Learning and Adaptation Network*) para aprendizagem eficiente, dimensionamento de modelos para adaptabilidade e uma estratégia de *bag-of-freebies* para melhorar a precisão e eficiência. A estratégia de *bag-of-freebies* refere-se a um conjunto de técnicas que melhoram o desempenho do modelo durante o treinamento sem aumentar o custo de inferência, sendo que maiores detalhes podem ser lidos em Zhang *et al.* (2019).

A YOLOv7 superou a YOLOv4 ao reduzir significativamente os parâmetros e a computação em 75% e 36%, respectivamente, enquanto melhorava a precisão média em 1,5%. Além disso, a YOLOv7-tiny também reduziu os parâmetros e a computação em 39% e 49%, respectivamente, sem comprometer o mAP (Wang; Bochkovskiy; Liao, 2022).

2.6.2 YOLOv8

A YOLOv8 ainda não havia sido descrita em artigo na época da escrita desta dissertação, o que limita a disponibilidade de informações sobre sua metodologia de desenvolvimento e os estudos realizados durante sua criação. Por esse motivo, nesta dissertação foram analisadas as informações disponíveis no repositório oficial, juntamente com a documentação limitada disponibilizada por Jocher, Chaurasia e Qiu (2023), além de alguns artigos que descrevem a arquitetura do modelo. Esses artigos, no entanto, não representam fontes oficiais.

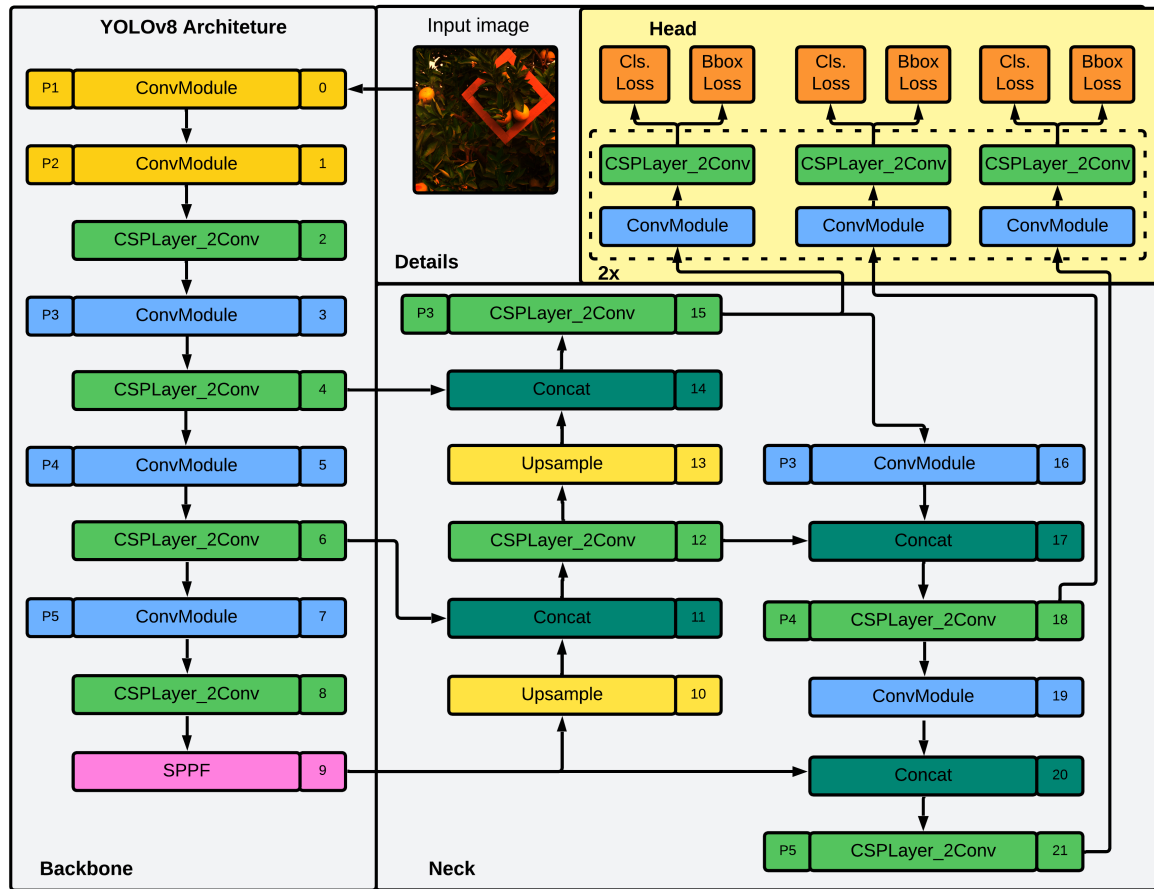
O modelo YOLOv8 é um dos mais recentes da família de algoritmos YOLO, e sua estrutura é basicamente dividida em três partes: a rede *backbone*, responsável pela extração de características da imagem; a *neck*, que realiza a fusão de características de diferentes níveis; e a saída *Decoupled Head*, na qual o processamento das tarefas de detecção, classificação e segmentação é separado em ramos distintos. Essa separação permite que cada ramo se concentre exclusivamente em sua respectiva tarefa, melhorando a precisão e a eficiência geral do modelo (Huangfu; Li, 2023; Jocher; Chaurasia; Qiu, 2023). A arquitetura do modelo pode ser visualizada na Figura 13.

A YOLOv8 usa a rede de pirâmide de características (SPPF, do inglês *Spatial Pyramid Pooling Fast*), um módulo que combina informações de diferentes regiões da imagem ao aplicar operações de *pooling* em múltiplas escalas, reduzindo o esforço computacional e a propagação de *features* ao longo da rede (He *et al.*, 2015; Jocher *et al.*, 2022; Terven; Córdova-Esparza; Romero-González, 2023).

A *neck* da YOLOv8 é modelado conforme a estrutura da *Efficient Layer Aggregation Networks* (ELAN), originalmente proposta na YOLOv7. Essa arquitetura permite uma integração eficiente de características (*feature maps*) de diferentes níveis da rede, combinando informações de camadas superficiais (detalhes finos) e profundas (contexto global). Como resultado, a arquitetura melhora a extração de características multiescala, essencial para detectar objetos de tamanhos variados.

Além disso, é adotado o módulo *Cross Stage Partial Network Fusion* (C2f), que realiza uma fusão parcial de características em estágios cruzados da rede, utilizando conexões residuais (*skip connections*) para conectar *features* de camadas de diferentes níveis da arquitetura. Com isso, as conexões cruzadas reduzem o problema do desaparecimento do gradiente, permitindo treinamento mais estável e eficiente (Huangfu; Li, 2023; Wang;

Figura 13 – Arquitetura da YOLOv8.



Fonte: Autor. Adaptada de Jocher, Chaurasia e Qiu (2023).

Bochkovskiy; Liao, 2023).

A YOLOv8 utiliza um modelo livre de âncora, que não depende de caixas de referência pré-definidas (*anchor boxes*) para prever objetos, o que simplifica a arquitetura e melhora a flexibilidade em diferentes tamanhos de objetos. Na camada de saída da YOLOv8, Jocher, Chaurasia e Qiu (2023) usaram a função sigmoide como função de ativação para o *score*, representando a probabilidade de que a *bounding box* contenha um objeto.

Ademais, é utilizada a função softmax para as probabilidades de classe, representando a probabilidade de objetos pertencerem a cada classe possível. Em relação às funções de perda, usa-se *Complete Intersection over Union* (CIoU), que leva em conta a sobreposição, distância central e a proporção das *bounding boxes* para otimizar sua precisão, e *Distribution Focal Loss* (DFL), que ajusta a distribuição dos valores previstos para refinar a localização dos limites das caixas. Essas abordagens contribuem significativamente para o aprimoramento do desempenho na detecção de objetos, especialmente no caso de objetos

de menor porte.

Neste trabalho, foi utilizada a variante YOLOv8-Seg, voltada para tarefas de segmentação. Essa versão conta com o *backbone* CSPDarknet53 como extrator de características, seguido por um módulo C2f. Posteriormente, um dos ramos da *head* é empregado para aprender a prever as máscaras a partir da imagem de entrada (Jocher; Chaurasia; Qiu, 2023). Ademais, de forma análoga à versão padrão da YOLOv8, i.e. a versão de classificação, a YOLOv8-Seg apresenta cinco configurações distintas de modelo, denominadas YOLOv8n-seg, YOLOv8s-seg, YOLOv8m-seg, YOLOv8l-seg e YOLOv8x-seg.

2.6.3 YOLOv9

A YOLOv9, proposta por Wang, Yeh e Liao (2024), foi desenvolvida com foco na eficiência, visando alcançar desempenho em tempo real em uma ampla variedade de dispositivos. Suas principais inovações incluem a integração da nova arquitetura GELAN (*Generalized Efficient Layer Aggregation Network*), que otimiza a predição ao agregar camadas de forma eficiente, reduzindo a quantidade de parâmetros e operações computacionais, e a técnica de treinamento PGI (*Programmable Gradient Information*), que ajusta dinamicamente os gradientes de aprendizagem, tornando-os mais estáveis e confiáveis, especialmente em modelos mais complexos. A arquitetura da YOLOv9 é apresentada na Figura 14.

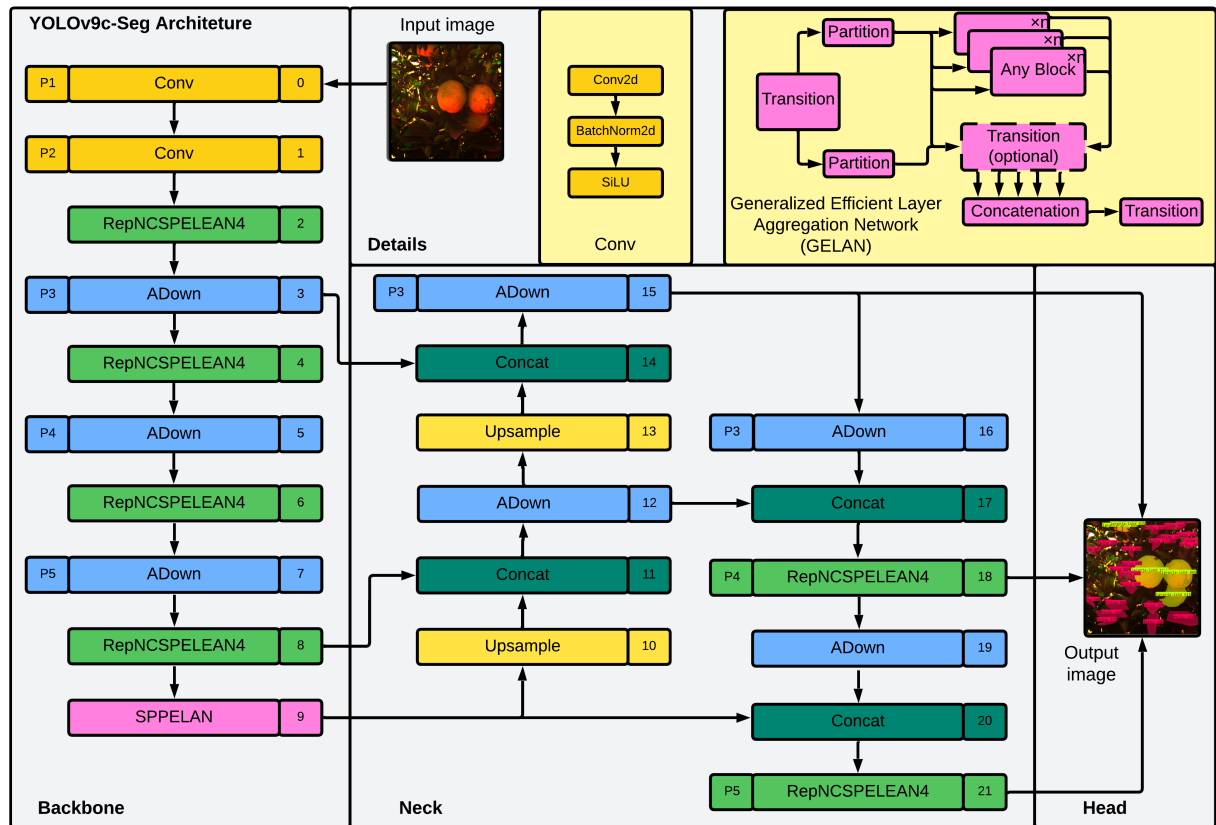
A GELAN é composta por dois blocos principais empilhados: blocos de agregação de camadas eficientes, que realizam transformações agregadas em múltiplas ramificações da rede para capturar características em diferentes escalas com baixo custo computacional, e blocos computacionais, responsáveis pela propagação eficiente de informações entre as camadas (Wang; Yeh; Liao, 2024). A arquitetura da GELAN é ilustrada na Figura 15.

Para contornar o problema do treinamento instável em modelos menores da YOLOv9, causado por informações de gradiente pouco confiáveis, Wang, Yeh e Liao (2024) propõem o método PGI. Essa técnica distribui gradientes auxiliares de forma progressiva e controlada ao longo da rede, o que permite estabilizar o processo de aprendizagem ao reforçar o fluxo de informação de maneira mais equilibrada. Como resultado, observa-se um ganho de desempenho mesmo em arquiteturas compactas e com recursos limitados.

O PGI possui dois componentes principais. O primeiro são os ramos auxiliares reversíveis, que fornecem gradientes menos ruidosos e mantêm conexões reversíveis com a entrada. O segundo componente é a integração de gradiente multinível, que evita a interferência causada por divergências provenientes dos diferentes ramos laterais. Com essa abordagem, os gradientes de todos os ramos são combinados antes de serem reintegrados ao modelo principal. Essa técnica pode ser visualizada na Figura 16.

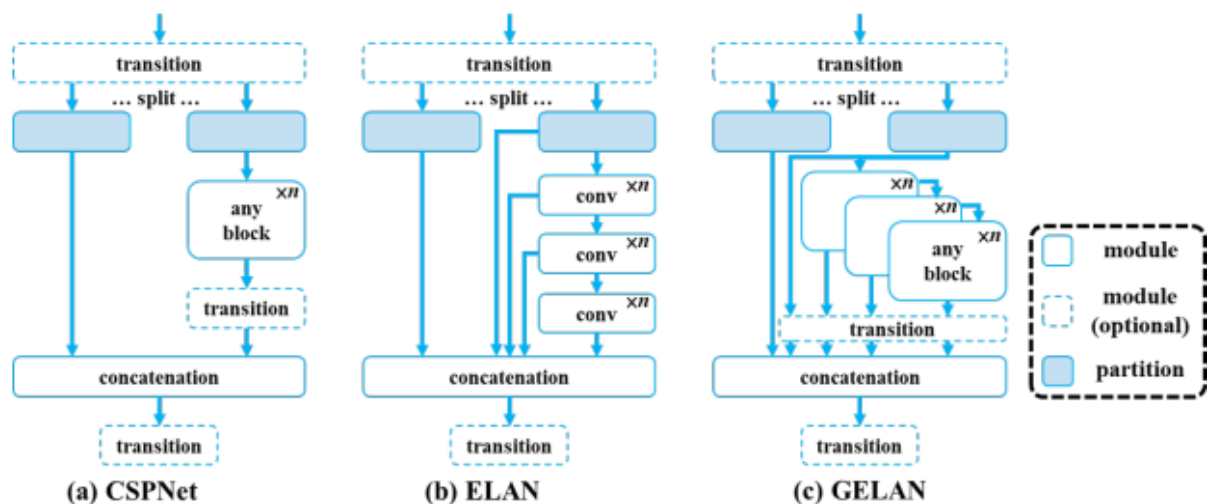
Os experimentos realizados por Wang, Yeh e Liao (2024) mostraram que o PGI

Figura 14 – Arquitetura da YOLOv9.



Fonte: Autor. Adaptada de Wang, Yeh e Liao (2024).

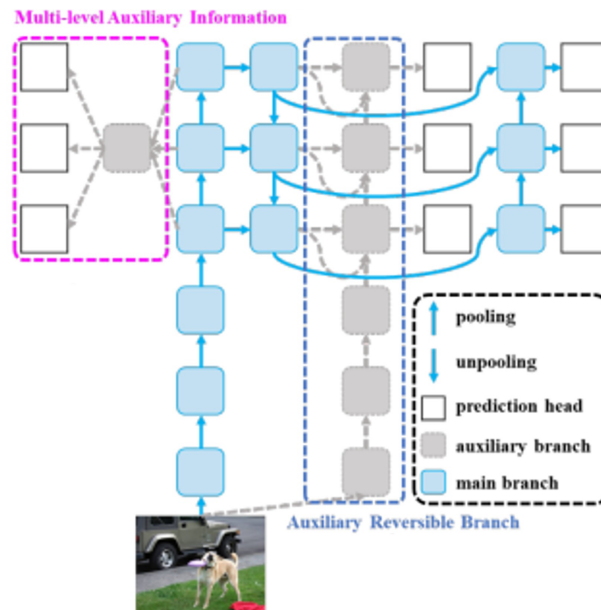
Figura 15 – Comparação da arquitetura GELAN com arquiteturas do estado da arte disponíveis à época de sua proposição.



Fonte: Wang, Yeh e Liao (2024).

melhorou a precisão em todos os tamanhos de modelo, especialmente em configurações menores. Por exemplo, aumentou as pontuações de AP do YOLOv9-Small em 0.1-0.4% em relação à linha de base GELAN-Small. Os ganhos foram ainda mais significativos

Figura 16 – Estrutura da PGI.



Fonte: Adaptado de Wang, Yeh e Liao (2024).

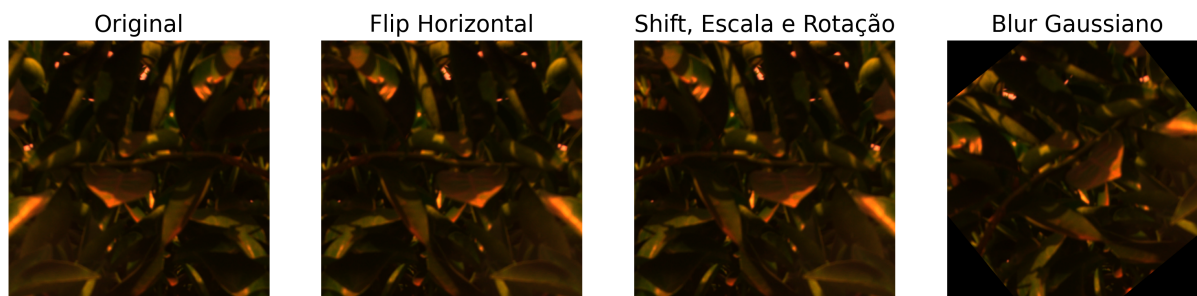
para modelos mais profundos, como o YOLOv9-e, com 55.6% de mAP. Portanto, o PGI permite que modelos menores e eficientes sejam treinados para níveis de precisão mais altos, anteriormente alcançados apenas por modelos superparametrizados.

2.6.4 Técnicas de aumento de dados, *Transfer Learning* e *Fine Tuning*

Embora as redes neurais profundas mostrem capacidade para resolver problemas complexos, ainda há dificuldade em construir modelos que generalizem bem com dados diferentes, o que leva ao problema de *overfitting* por consequência. Para contornar esse problema, diversas técnicas são propostas na literatura, entre as quais se destaca o aumento de dados. Essa técnica é comumente utilizada para expandir a dimensionalidade e a variedade do conjunto de dados.

A técnica escolhida para o presente trabalho foi baseada na biblioteca Albumentation, de autoria de Buslaev *et al.* (2020), utilizada em inúmeras competições e que conta com mais de 70 técnicas diferentes de aumento implementadas, divididas em dois tipos: transformações de píxel e transformações em nível espacial. O principal ganho dessa biblioteca é a possibilidade de utilizar essas técnicas de aumento juntamente com outras bibliotecas de *Machine Learning*, tais como Tensorflow, Pytorch e a biblioteca da Ultralytics. Além disso, essa biblioteca é otimizada para a execução dessas tarefas, sendo que a grande maioria de suas operações é mais rápida que a TorchVision, a ImgAug e a Keras, que são outras bibliotecas comumente utilizadas para essa operação. A Figura 17 apresenta exemplos das transformações realizadas nas imagens de treinamento para aumentar os dados.

Figura 17 – Exemplos de operações de aumento de dados aplicadas aos dados de treinamento.



Fonte: Autor.

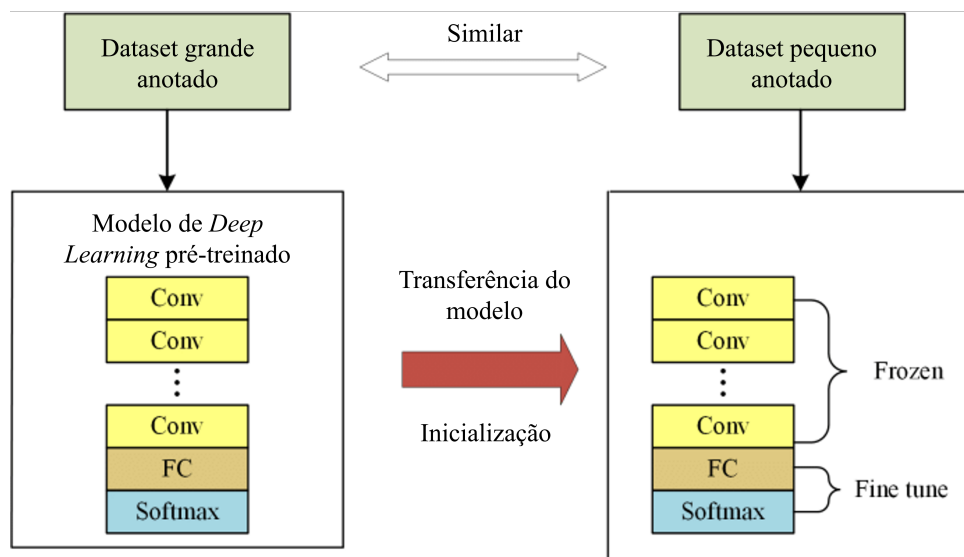
Comumente, para minimizar o tempo de treinamento e aumentar a probabilidade de obter um modelo de alta qualidade, utilizam-se, além da técnica de aumento de dados, duas outras técnicas denominadas *Transfer Learning* e *Fine Tuning*. O *Transfer Learning*, ou aprendizagem por transferência, é uma técnica de *Machine Learning* na qual um modelo treinado para uma tarefa é reutilizado como ponto de partida para um modelo em uma segunda tarefa relacionada. Essa abordagem é comumente utilizada em cenários nos quais os dados para a nova tarefa são escassos ou difíceis de obter, mas há abundância de dados para uma tarefa semelhante. Ao aproveitar o conhecimento adquirido pelo modelo pré-treinado, é possível alcançar um desempenho elevado na nova tarefa com menos dados e menos tempo de treinamento (Yu; Xiu; Li, 2022).

O processo de ajuste fino, ou *Fine Tuning*, é um passo adicional no *Transfer Learning*, no qual um modelo pré-treinado é adaptado de forma mais específica à nova tarefa. Isso envolve re-treinar o modelo, geralmente com uma taxa de aprendizado menor, com um novo conjunto de dados mais específico. O *Fine Tuning* permite que o modelo ajuste seus parâmetros para melhor se adequar aos detalhes e nuances da nova tarefa, mantendo, ao mesmo tempo, o conhecimento previamente adquirido (Yu; Xiu; Li, 2022).

Para realizar o *Fine Tuning*, normalmente se congela as primeiras camadas do modelo pré-treinado, que capturam características mais gerais e universais das imagens, e se treina apenas as camadas finais, que são mais específicas para a tarefa original. Uma representação dessas duas técnicas sendo aplicadas pode ser visualizada na Figura 18 (Chen; Cen; Xiong, 2020).

2.7 QuadTree

Visando realizar uma busca otimizada das informações para a correspondência de objetos em imagens, utiliza-se o IoU em conjunto com o QuadTree. O IoU (*Intersection over Union*) é uma medida que avalia a sobreposição entre a área prevista pelo modelo e a

Figura 18 – Representação da aplicação do *Transfer Learning* e *Fine Tuning*.

Fonte: Chen, Cen e Xiong (2020).

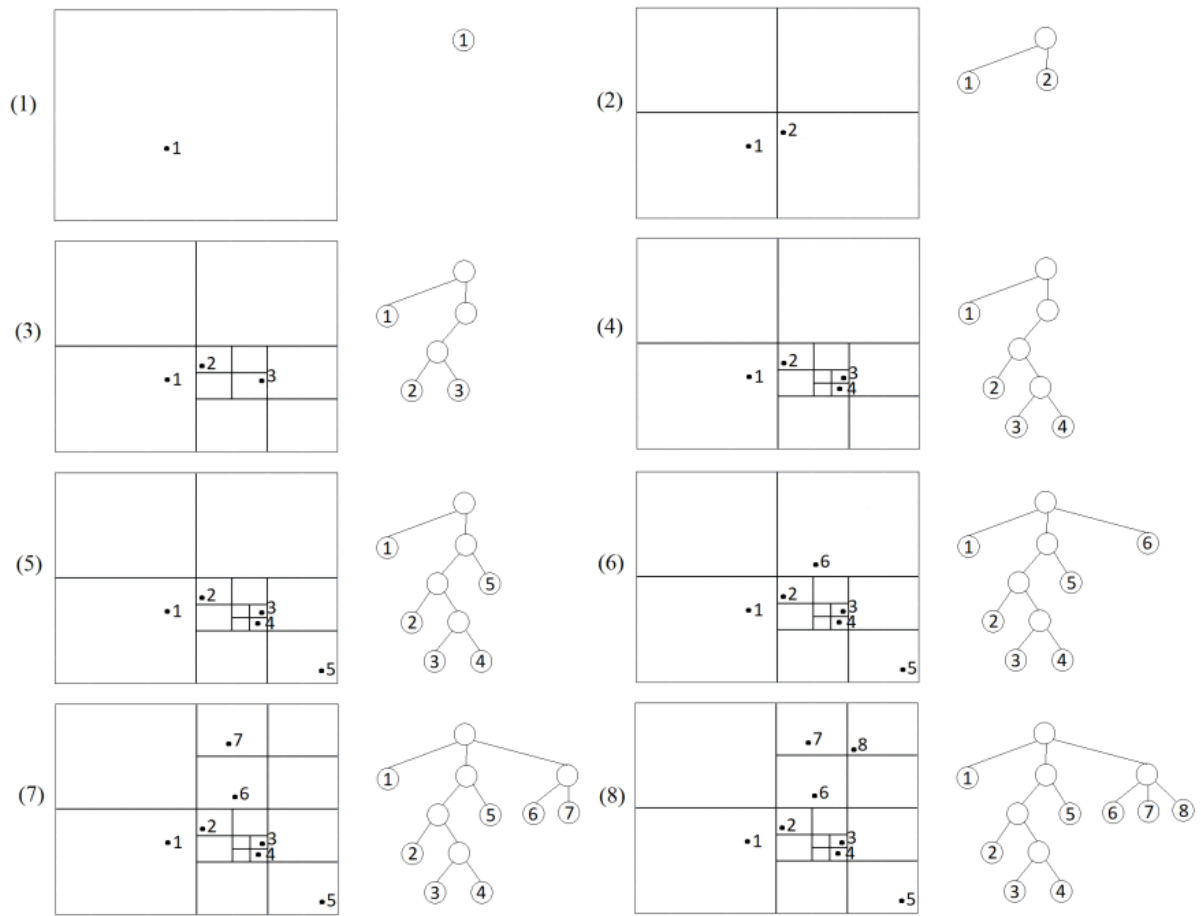
área verdadeira do objeto em questão. Já o QuadTree é uma estrutura de dados espacial empregada para subdividir um domínio 2D. Consiste em uma árvore na qual os nós internos possuem quatro nós filhos (quadrantes), enquanto os nós sem filhos são denominados nós folhas (Samet, 1984). Essa estrutura exige um particionamento hierárquico de uma região alinhada aos eixos cartesianos, onde o nó raiz representa a caixa delimitadora e cada nó filho é menor que seu pai e se encontra em um nível inferior (Camata; Coutinho, 2010).

O QuadTree funciona por meio de uma decomposição recursiva do espaço $\mathbb{R}$, utilizando separadores paralelos aos eixos das coordenadas e abcissas, resultando em quatro regiões de mesmo tamanho. É importante estabelecer uma relação entre os arranjos hierárquicos espaciais e as regiões que correspondem aos nós da QuadTree (Coelho, 2022).

O nó raiz da QuadTree corresponde ao espaço de características que contém todos os pontos de interesse, ou seja, à base de toda a hierarquia espacial. Esse nó raiz é denominado célula raiz. Cada região de mesmo tamanho resultante da divisão da célula raiz é chamada de célula. Os termos subcélula e supercélula são usados para definir a relação hierárquica entre as células. Assim, o termo subcélula é usado para definir uma célula que está contida em outra, enquanto a célula que contém a subcélula é definida como supercélula (Mehta; Sahni, 2004). Um exemplo da construção de uma QuadTree pode ser visualizado na Figura 19.

O presente trabalho não constrói a árvore completamente, mas utiliza um algoritmo inspirado em sua estrutura para reduzir o espaço de busca ao longo da imagem. Com isso, a busca por objetos semelhantes em diferentes imagens é otimizada, facilitando o processo de correspondência entre elas. Para diminuir o espaço de busca utilizando o método proposto, inicialmente realiza-se a varredura nos quatro quadrantes da imagem. Ao identificar um

Figura 19 – Construção da estrutura de dados de uma QuadTree.



Fonte: Coelho (2022).

objeto em um dos quadrantes, utilizando o IoU como critério de verificação, esse quadrante é subdividido em quatro partes menores, e a busca prossegue. Esse processo se repete até que sejam realizadas dez iterações. Esse número foi definido com base em testes empíricos, os quais indicaram que ele proporciona um bom equilíbrio entre desempenho e eficiência na busca.

3 MATERIAIS E MÉTODOS

3.1 Câmara multiespectral

Nessa dissertação de mestrado, foi utilizado um conjunto de dados de agricultura obtido com a câmara mutiespectral Sony A7R Multispectral Sextuple, equipada com um sistema de seis lentes desenvolvido pela Agrowing. Esse sistema possui diferentes filtros que produzem 14 bandas espectrais. As especificações do sensor estão apresentadas na Tabela 1, e a câmara utilizada, na Figura 20.

Tabela 1 – Especificações da Câmara multiespectral, modelo Sony A7R Multispectral Sextuple.

Parâmetro	Especificação
Tipo	Montagem única com seis lentes
Campo de visada	Diagonal 40.8° Horizontal 33.0° Vertical 25.0°
Distorção das lentes	<2%
Bandas multiespectrais (nm)	405;430;450;490;525;550;560;570 630;650;686;710;735;850
Distância focal	21.8 mm
Abertura	Fixo F6
Dimensões	64.5x35 mm
Peso das lentes	190 g
Peso total do sensor incluindo a bateria	705 g

Fonte: Agrowing (2024).

Figura 20 – Câmara multiespectral, modelo Sony A7R Multispectral Sextuple, com o sensor de sistema de seis cabeças.



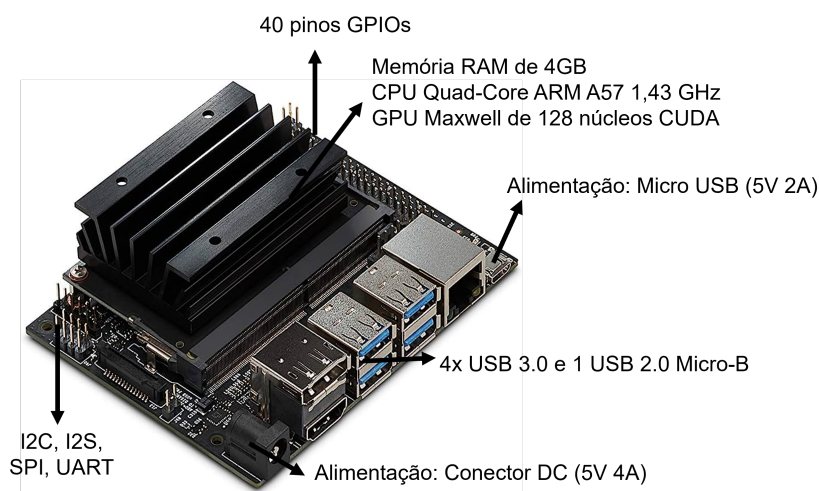
Fonte: Agrowing (2024).

3.2 Placa NVIDIA Jetson Nano

Para o presente trabalho, foi escolhida a placa NVIDIA Jetson Nano. Suas características são particularmente adequadas para operações em campo e em tempo real, nas quais a inferência de modelos de IA precisa ser realizada localmente. Assim, a NVIDIA Jetson Nano se mostra uma escolha interessante para aplicações baseadas em computação de borda, conforme explicado na Seção 2.4.

A Jetson Nano, desenvolvida pela NVIDIA, é um computador de placa única projetado para implementar soluções de inteligência artificial (IA) em sistemas embarcados. A mesma é ilustrada na Figura 21. Lançada em 2019, essa plataforma oferece alto desempenho a um custo acessível, permitindo a execução de algoritmos de IA em projetos compactos. Além de sua versatilidade em aplicações de IA, a Jetson Nano utiliza a plataforma CUDA para maximizar o desempenho de suas GPUs integradas (Assunção *et al.*, 2022).

Figura 21 – Descrição da placa NVIDIA Jetson Nano



Fonte: Autor.

Do ponto de vista arquitetônico, a NVIDIA Jetson Nano é equipada com uma GPU NVIDIA Maxwell, contendo 128 núcleos CUDA, complementada por um processador quad-core ARM Cortex-A57. Essa configuração proporciona um equilíbrio otimizado entre poder computacional e eficiência energética, conferindo à placa adaptabilidade para uma variedade de aplicações (Kulkarni, 2021). Cada núcleo CUDA é capaz de executar uma pequena parte de um cálculo em paralelo com outros núcleos, permitindo assim uma abundância de cálculos a serem realizados simultaneamente.

Um aspecto importante sobre o processador ARM Cortex-A57 é que ele foi projetado para dispositivos móveis e embarcados, enquanto os processadores x86 (Intel e AMD) são projetados para uma gama mais ampla de tarefas e demandas de desempenho, estando voltados para desktops e laptops. Dessa forma, não é possível realizar comparações

de desempenho entre eles diretamente. O máximo que se poderia fazer nesse caso seria comparar com alguns processadores Intel Atom ou Celeron de baixa potência ou com a linha Ryzen Embedded, que poderiam ser comparados, especialmente em aplicações de sistemas embarcados e de baixa potência (ARM Developer, 2009; Kulkarni, 2021).

Um aspecto importante da NVIDIA Jetson Nano é o suporte a uma variedade de frameworks de IA populares, como TensorFlow (TensorFlow, 2018), PyTorch (Imambi; Prakash; Kanagachidambaresan, 2021) e Caffe (Jia *et al.*, 2014), simplificando o desenvolvimento e implementação de modelos de *Machine Learning* e redes neurais profundas. NVIDIA Jetson Nano encontra aplicação em uma diversidade de projetos práticos, incluindo sistemas de visão em robótica, drones, dispositivos de IoT e reconhecimento de objetos, destacando-se como uma ferramenta versátil no domínio de sistemas embarcados com capacidades para execução de algoritmos de IA (Assunção *et al.*, 2022).

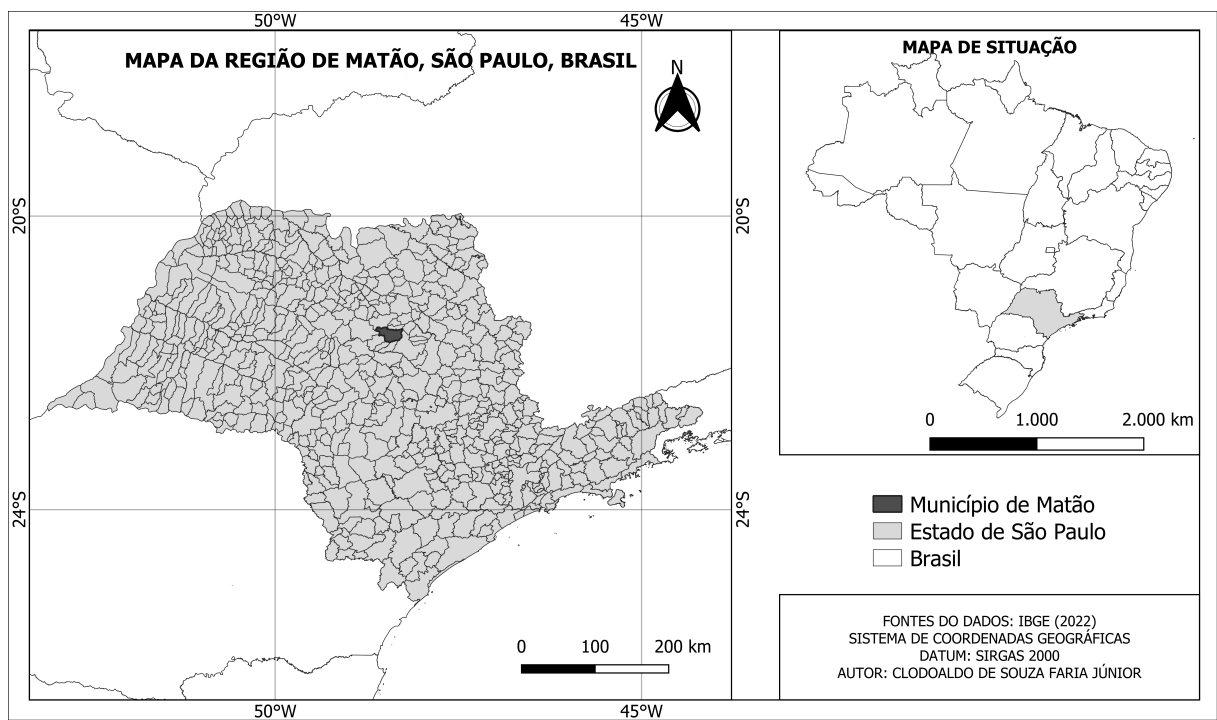
A Jetson Nano possui uma memória RAM DDR4, uma decisão estratégica fundamentada em diversos fatores, como demandas específicas e capacidades do dispositivo. A implementação de memória DDR4 na Jetson Nano proporciona um aprimoramento significativo de desempenho em relação às gerações anteriores. O aumento da largura de banda e as taxas de transferência mais elevadas permitem que a Jetson Nano processe cargas de trabalho intensivas em dados de maneira mais eficiente, o que é essencial para aplicações de visão computacional e *machine learning*. Além disso, a DDR4 contribui para a eficiência energética da Jetson Nano ao operar com tensões mais baixas em comparação com suas predecessoras. Essa característica é especialmente vantajosa em cenários nos quais o consumo de energia é um fator crítico, como em dispositivos embarcados e sistemas autônomos (Latif *et al.*, 2021; Assunção *et al.*, 2022).

3.3 Conjuntos de dados e preparação

Quanto ao conjunto de dados utilizado, a equipe do projeto temático "Sensoriamento Remoto de Alta Resolução para Agricultura Digital", da FCT/Unesp-Presidente Prudente, realizou uma coleta terrestre com a câmera Sony A7R Multispectral Sextuple, sem utilização de tripé ou qualquer tipo de apoio. A área em que os dados foram obtidos consiste em um pomar produtor de cítricos localizado no município de Matão, no estado de São Paulo. Um mapa da localização do município de Matão está apresentado na Figura 22.

Embora alvos quadrados tenham sido utilizados para sinalizar pontos infestados por uma doença durante a coleta de dados, eles foram desconsiderados neste trabalho, uma vez que o foco não esteve na identificação da doença, mas sim na avaliação da presença de objetos com potencial para servir como alvos visuais capazes de aprimorar o desempenho dos modelos, por apresentarem características distintivas em relação aos demais elementos da cena.

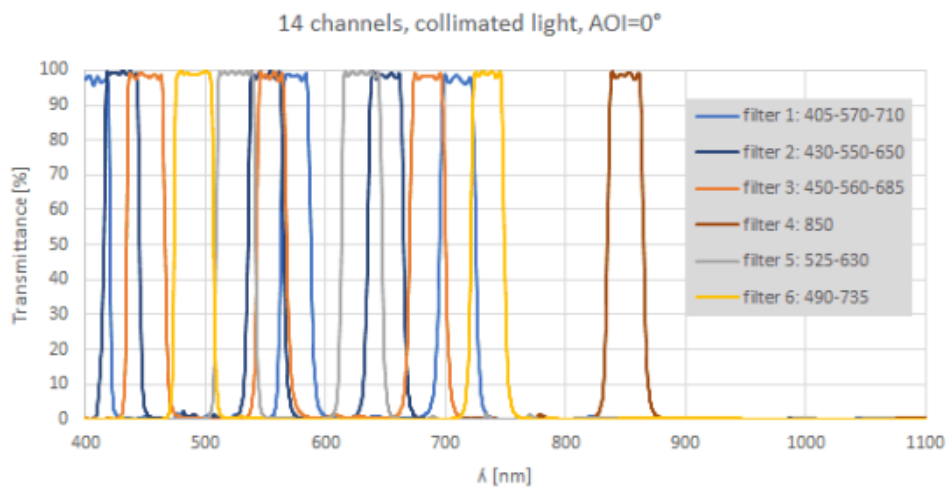
Figura 22 – Mapa da localização do município de Matão.



Fonte: Autor.

As bandas foram processadas com o software Agrowing Basic, que permitiu a extração de cada uma. A Figura 23 ilustra os comprimentos de onda. As combinações das bandas podem ser visualizadas na Figura 24.

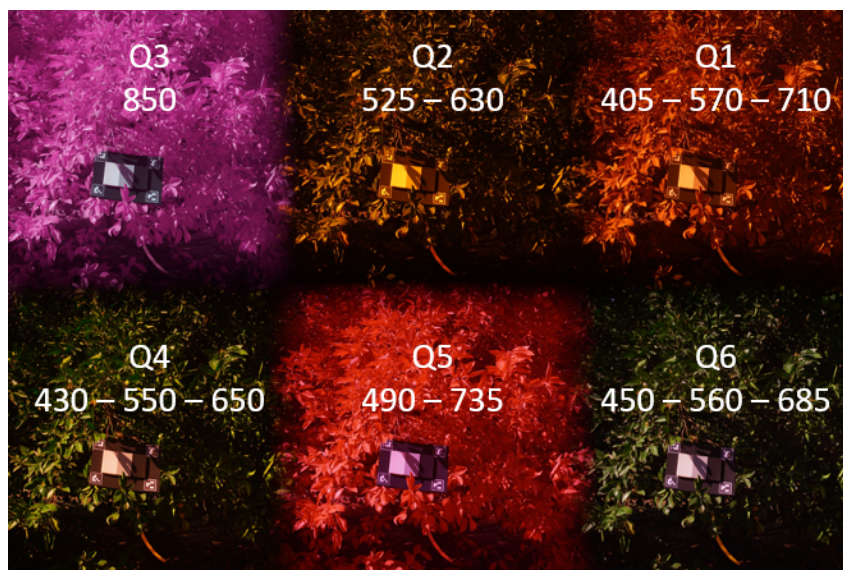
Figura 23 – Descrição do comprimento de onda central e a composição de cada uma das subimagens obtidas pela câmara Sony A7R Multispectral Sextuple.



Fonte: Agrowing (2024).

Como as composições coloridas das subimagens foram utilizadas como entrada para as redes, foi necessário ajustar o contraste das imagens para melhorar a visualização e a compreensão dos detalhes. Isso facilitou a distinção dos alvos. Para isso, foi realizado

Figura 24 – Combinações de bandas de cada uma das subimagens da câmara Sony A7R Multispectral Sextuple.



Fonte: Autor.

um breve estudo sobre metodologias possíveis, das quais foi escolhida o *stretching* do histograma utilizando os percentis de 2% e 98%.

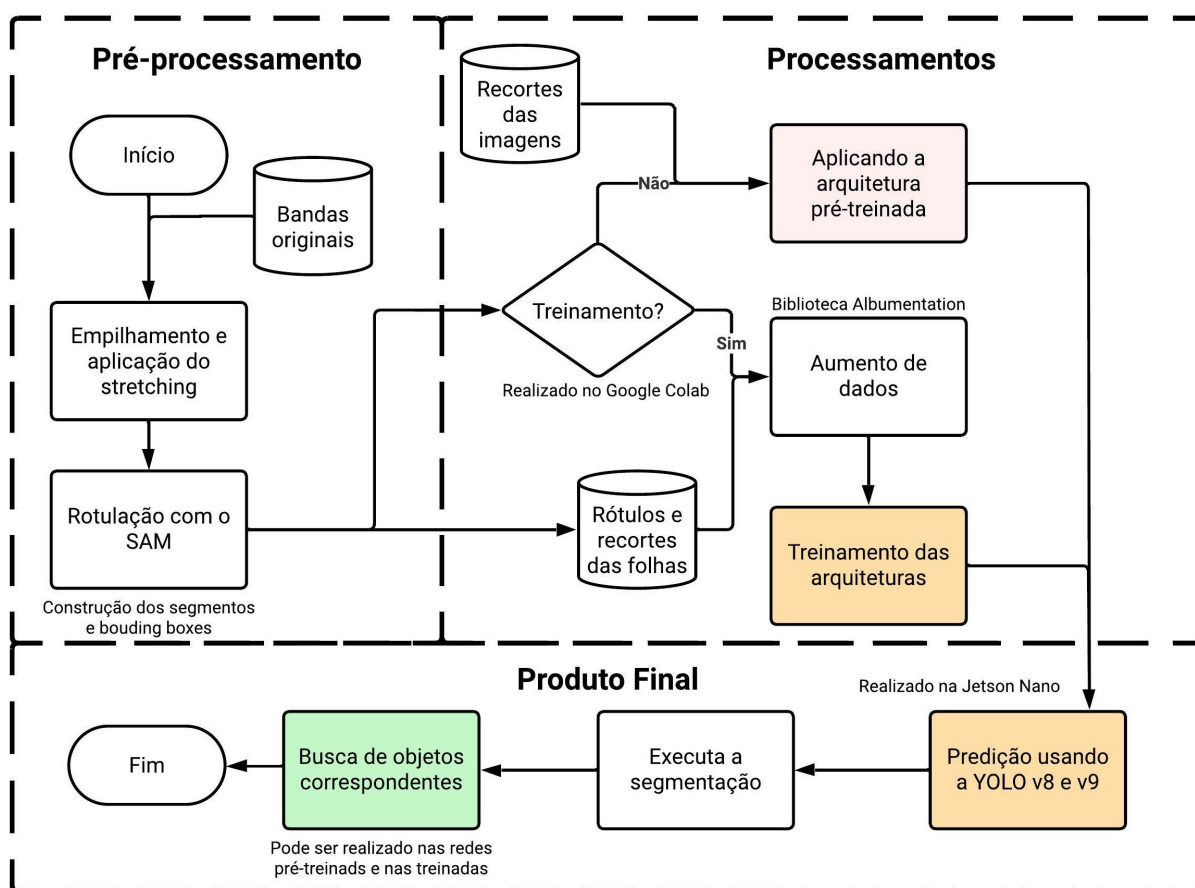
Para facilitar a visualização e o entendimento de detalhes, além de permitir uma entrada adequada para os modelos, as imagens originais foram recortadas. As arquiteturas da YOLO utilizadas, por padrão, operam com imagens de 640x640 pixels e três canais (RGB), o que favorece a visualização dos resultados e dispensa ajustes adicionais para redimensionamento. Os recortes foram realizados sem sobreposição, ignorando a parte restante da imagem quando não era possível recortar integralmente. Assim, os dados apresentados para as arquiteturas subsequentes correspondem a dados brutos, apenas recortados e com a correção de contraste. Ambas as operações podem ser realizadas automaticamente em campo.

3.4 Frameworks de inteligência artificial, rotulagem e experimentos realizados

A YOLO foi adotada como arquitetura base devido ao seu suporte para execução em hardware embarcado, como o NVIDIA Jetson Nano, e à sua capacidade de realizar detecções em tempo real. Além disso, algumas versões da YOLO incorporam funcionalidades de segmentação, permitindo não apenas identificar a posição dos objetos, mas também seus contornos. Para a implementação e execução, foi utilizada a biblioteca da Ultralytics, que oferece implementações eficientes da YOLO e modelos pré-treinados. Essa biblioteca utiliza PyTorch como *backend* para o desenvolvimento das arquiteturas. Antes de sua

escolha, foram avaliadas outras opções, com PyTorch, TensorFlow e Keras. No entanto, a da Ultralytics se destacou por oferecer uma implementação mais simplificada e compatível com a NVIDIA Jetson Nano, justificando sua adoção. A Figura 25 apresenta o fluxograma do processamento e as metodologias adotadas ao longo deste trabalho, que serão descritas a seguir.

Figura 25 – Fluxograma do processamento.



Fonte: Autor.

As bibliotecas Pandas (McKinney, 2010; team, 2020), NumPy (Harris *et al.*, 2020) e Scikit-Learn (Pedregosa *et al.*, 2011) foram usadas para o tratamento, manuseio e extração de métricas durante o treinamento e a análise dos dados. Além disso, utilizou-se a biblioteca Pyplot, integrante do pacote Matplotlib (Hunter, 2007), para elaborar gráficos. A biblioteca SciencePlots (Garrett, 2021) foi empregada para fornecer estilos personalizados ao Matplotlib, facilitando a formatação de figuras para trabalhos acadêmicos.

Para o treinamento dos modelos, foi criado um conjunto de dados a partir dos dados no projeto temático "Sensoriamento Remoto de Alta Resolução para Agricultura Digital" da FCT/Unesp-Presidente Prudente. Esses dados consistem nos dados corrigidos de contraste via *stretching* e recortados com dimensão 640x640 pixels. Essa etapa foi realizada por meio de uma metodologia semiautomática de rotulagem, com o emprego

do algoritmo *Segment Anything Model* (SAM). A delimitação inicial das feições a serem segmentadas foi feita automaticamente, e uma avaliação visual foi realizada para verificar se o segmento criado pelo SAM apresentava um resultado satisfatório ou se precisava de refinamento manual. Para mais detalhes sobre o SAM, recomenda-se a leitura de Kirillov *et al.* (2023). Esse processo de rotulação semiautomática permitiu um ganho significativo na velocidade de rotulação de dados para treinamento, validação e teste. Embora não seja tão oneroso quanto o processo manual, a rotulação semiautomática ainda requer uma etapa final de avaliação e ajuste manual, quando são feitos pequenos ajustes, se necessário.

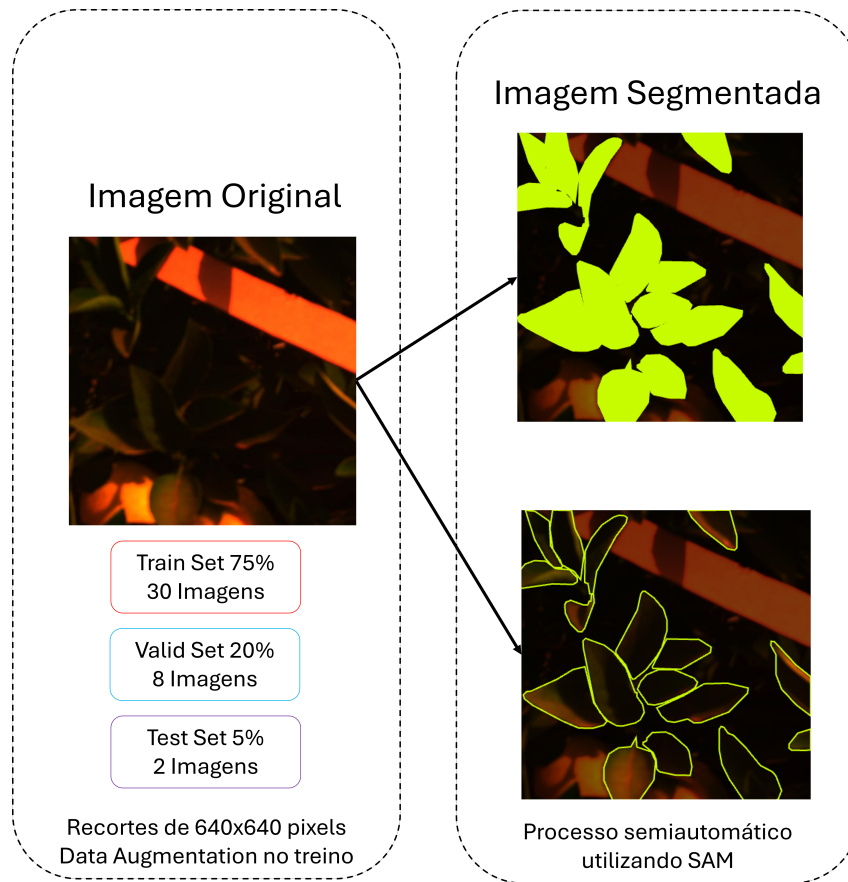
Com isso, o presente trabalho realizou três experimentos baseados nas etapas apresentados na Figura 25. O primeiro teste realizado consistiu em localizar, classificar e segmentar objetos utilizando apenas as redes pré-treinadas YOLOv8 e YOLOv9 com o banco de dados MS COCO. Esse teste consistiu na avaliação da etapa apresentada em rosa no fluxograma na Figura 25. Nesse momento, somente foi validada a capacidade em relação às laranjas, visto que existem dados desse tipo no banco de dados MS COCO de (Lin *et al.*, 2014), não sendo trabalhado, com folhas de árvores. Esse experimento está detalhado na Seção 4.1 dos resultados. O objetivo desse teste foi validar se, mesmo utilizando o modelo pré-treinado, a rede era capaz de realizar a localização, a classificação e a segmentação das laranjas de maneira satisfatória. Para isso, foram exploradas várias combinações de dados para determinar qual proporcionaria uma melhor explicabilidade dos objetos, principalmente no que se refere à informação de cor, que seria um fator determinante na identificação dos objetos. Para tanto, foram consideradas duas composições coloridas e uma banda individual, a saber, as subimagens Q4 e Q6, bem como o comprimento de onda de 650 nm. Vale ressaltar que foram selecionadas bandas para formarem composições RGB com o objetivo de testar a generalização dos modelos, mesmo com dados aparentemente semelhantes. Para cada uma das combinações de bandas, foram testadas todas as variantes da YOLOv8 e da YOLOv9, a fim de visualizar o ganho de informação e compreender qual das combinações de bandas seria mais relevante para o processo de identificação, bem como a necessidade de complexidade do modelo, ou seja, a quantidade de parâmetros necessários. Além disso, ao conseguir segmentar as laranjas de forma satisfatória, torna-se possível utilizar técnicas de *Transfer Learning* e *Fine Tuning* para aplicar o conhecimento dessas redes pré-treinadas em aplicações similares, como é o caso da aplicação para folhas, que será descrita a seguir, com a realização de somente alguns ajustes. Um aspecto relevante na validação das composições coloridas está relacionado ao fato de que, em diversos trabalhos da literatura, a informação de cor é utilizada predominantemente nas camadas iniciais das redes neurais convolucionais. À medida que a imagem percorre camadas mais profundas, sua representação torna-se progressivamente mais abstrata (Zeiler; Fergus, 2013; Kinger; Kulkarni, 2021). Com isso, o modelo deixa de operar diretamente sobre canais de cor, passando a atuar sobre mapas de características (*feature maps*), que representam formas, partes de objetos e relações espaciais. Dessa forma, a realização desta

etapa visa verificar a real contribuição da informação de cor para o desempenho do modelo.

O segundo experimento, detalhado na Seção 4.2, o qual é apresentado em amarelo no fluxograma na Figura 25, consistiu em treinar redes YOLO utilizando um conjunto de dados rotulados semi-automaticamente com o auxílio do algoritmo Segment Anything Model, de Kirillov *et al.* (2023). Um exemplo da criação desses conjuntos foi apresentado na Figura 26. Esse treinamento foi realizado utilizando o Google Colab, e o modelo pronto é aplicado para inferência na placa NVIDIA Jetson Nano, sendo que o modelo foi salvo no formato ".pt", que, no PyTorch é uma extensão de arquivo usada para salvar modelos treinados, *checkpoints* ou outros objetos relacionados à aprendizagem de máquina. Esse formato armazena o estado de um modelo, incluindo os pesos e parâmetros treinados, permitindo que o modelo seja carregado posteriormente para inferência e/ou retreinamento. O treinamento no Google Colab foi utilizando uma placa de vídeo NVIDIA Tesla T4 de 20 GB de memória de vídeo e 15 GB de memória RAM. Além disso, foram utilizadas técnicas de *Transfer Learning* e *Fine Tuning* a partir do modelo com o conjunto de dados MS COCO. O objetivo foi ajustar os pesos para possibilitar a segmentação de folhas a partir da transferência do conhecimento prévio obtido com o uso desse conjunto de dados. Com a utilização do algoritmo SAM, um total de 30 imagens foram rotuladas para o treinamento, contendo 398 folhas segmentadas, 8 imagens de validação, que possuem 107 rótulos de folhas, e 2 imagens para testes, contendo 22 rótulos. Assim, utilizou-se uma proporção de 75%, 20% e 5% para treinamento, validação e teste. Apesar da baixa porcentagem de dados para teste, um ponto importante é que cada imagem é recortada em dimensões de 640×640 , o que aumenta a quantidade de dados por conjunto utilizados. Além disso, os dados de treinamento passaram por uma etapa de aumento de dados, o que resultou em uma multiplicação do conjunto de treinamento por cerca de 6 vezes o tamanho, resultando em 2.388 segmentos utilizados para treinamento no final.

No terceiro experimento (o qual será discutido na Seção 4.4), apresentado em verde no fluxograma na Figura 25, avaliou-se a capacidade de buscar feições correspondentes em imagens distintas, baseadas nos segmentos construídos. Para isso, realizou-se a segmentação de ambas as composições coloridas, ou seja, de Q4 e Q6, encontrando cada uma das feições em ambas as composições e, em seguida, realizando a segmentação. Além disso, foram empregadas, neste experimento, técnicas de *Transfer Learning* e *Fine Tuning* baseadas no conjunto de dados COCO de Lin *et al.* (2014). Após a segmentação da subimagem Q4, busca-se a feição correspondente na subimagem Q6. Geralmente, são observadas diferentes colorações e deslocamento, e a feição pode não estar completamente visível devido ao contraste da imagem e à distância do local de tomada da cena. Entretanto, ao realizar essa busca, foi utilizada uma sobreposição mínima de 50% juntamente com uma busca baseada na estrutura de dados QuadTree, a qual foi descrita na Seção 2.7.

Figura 26 – Exemplo da criação do conjunto de dados utilizando SAM.



Fonte: Autor.

3.5 Técnicas de análise dos resultados

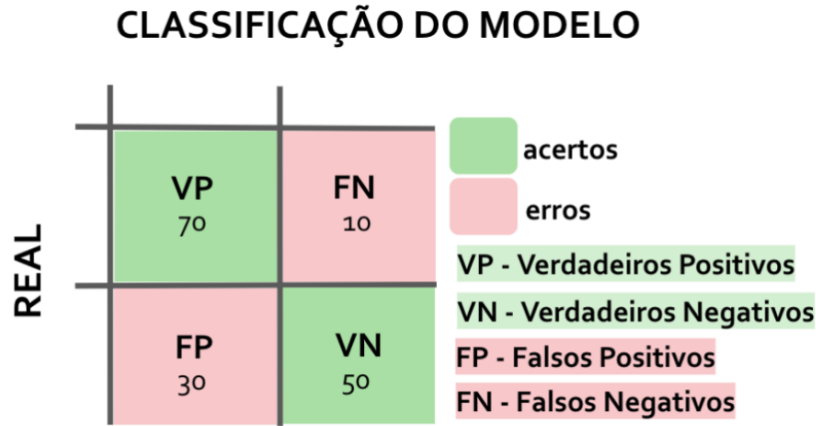
Uma das métricas mais utilizadas é a *Accuracy*, que quantifica a proporção de acertos em relação ao número total de exemplos. Ela deve ser usada quando a proporção de exemplos para cada classe é semelhante e o impacto dos acertos e erros é equivalente para cada uma. A *Accuracy* pode ser calculada utilizando a Equação (3.1).

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN} \quad (3.1)$$

Onde VP , VN , FP e FN são, respectivamente, os valores de verdadeiro positivo, verdadeiro negativo, falso positivo e falso negativo, ilustrados em um exemplo de matriz de confusão na Figura 27.

Outra métrica é a *Precision*, que consiste no número de exemplos classificados como pertencentes a uma classe que realmente são dela (verdadeiros positivos), dividido pela soma desse número com o número de exemplos classificados nesta classe, mas que pertencem a outras (falsos positivos). O cálculo dessa métrica é apresentado na Equação (3.2).

Figura 27 – Exemplo de matriz de confusão.



Fonte: Ruback, Avila e Cantero (2021).

$$Precision = \frac{VP}{VP + FP} \quad (3.2)$$

O *recall*, por sua vez, mostra o número de exemplos classificados como pertencentes a uma classe que realmente são dela, dividido pela quantidade total de exemplos que pertencem a esta classe, mesmo que sejam classificados em outra. No caso binário, são positivos verdadeiros divididos pelo total de positivos. Esse cálculo é feito usando a Equação (3.3).

$$Recall = \frac{VP}{VP + FN} \quad (3.3)$$

O F1-Score é a média harmônica entre precisão e *recall*, utilizado para avaliar o equilíbrio entre essas duas métricas. Esse é definido pela Equação (3.4).

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (3.4)$$

O mAP, *mean Average Precision*, por sua vez, é calculado pela Equação (3.5), onde U representa o número de imagens avaliadas, k é o número de detecções consideradas por imagem, $P(u, i)$ é a precisão da detecção i na imagem u , e $rel(u, i)$ é 1 se a detecção i corresponde a um objeto real (verdadeiro positivo) na imagem u , e 0 caso contrário.

$$mAP = \frac{1}{|U|} \sum_{u=1}^{|U|} \sum_{i=1}^k P(u, i) \times rel(u, i) \quad (3.5)$$

O mAP mede a proporção de itens relevantes entre os itens retornados pelo modelo para todas as imagens. A média, referida como *average*, é a média da precisão calculada dentro da lista de resultados para cada imagem. A média, referida como *mean*, é a média

das médias acima, ou seja, a média da precisão média para cada imagem. A Equação (3.5) supõe que o número de itens relevantes possíveis para cada imagem é, no mínimo, k . Ou seja, é possível preencher a lista com k itens relevantes.

Por fim, tem-se o IoU (*Intersection over Union*), que é calculado a partir da Equação (3.6). O IoU é uma medida que avalia a sobreposição entre a área prevista pelo modelo e a área real do objeto em questão. Ela é calculada dividindo a área da interseção das duas áreas pela área da união das mesmas. Esse índice fornece uma métrica quantitativa da precisão da segmentação, indicando o quão bem a previsão do modelo corresponde à realidade. No entanto, essa métrica somente pode ser calculada se a *bounding box* original for conhecida, ou seja, não funciona durante as etapas de teste e predição.

$$IoU = \frac{\text{Área de Sobreposição}}{\text{Área de União}} \quad (3.6)$$

No âmbito da avaliação de desempenho de arquiteturas, é comum a utilização de métricas derivadas do mAP em diferentes limiares de IoU. O mAP50, considera como verdadeira positiva uma predição cuja sobreposição com o objeto de referência atinge, no mínimo, 50% de IoU. Por sua vez, o mAP50-95, é definido como a média dos valores de *Average Precision* calculados em múltiplos limiares de IoU, variando de 0,50 a 0,95, com incrementos de 0,05. Essa abordagem mais abrangente permite uma avaliação mais criteriosa, penalizando tanto classificações incorretas quanto localizações imprecisas.

4 RESULTADOS E DISCUSSÕES

Conforme a descrição dos testes na Seção 3.4, a Tabela 2 apresenta um resumo dos experimentos que foram realizados, incluindo objetivos, dados utilizados, metodologias e plataformas empregadas.

Tabela 2 – Descrição dos experimentos realizados.

#	Objetivo	Dados Utilizados	Metodologia	Plataforma	Avaliação dos Resultados
1	Localizar, classificar e segmentar objetos (laranjas) com redes pré-treinadas YOLOv8 e YOLOv9	Conjunto de dados MS COCO (laranjas), subimagens Q4, Q6, comprimeto de onda 650 nm	<i>Transfer Learning</i> e <i>Fine Tuning</i>	Google Colab e Jetson Nano para inferência	Verificação visual e métricas apresentadas na Seção 3.5.
2	Treinar redes YOLO com dados rotulados (utilizando o Segment Anything Model) para segmentação de folhas	30 imagens rotuladas com 398 folhas segmentadas, 8 imagens de validação e 2 imagens de teste	<i>Transfer Learning</i> , <i>Fine Tuning</i> e Aumento de Dados	Google Colab (treinamento) e Jetson Nano (para inferência)	Avaliação da qualidade da segmentação de folhas com base nas métricas da Seção 3.5.
3	Buscar feições similares baseadas nos segmentos construídos (Q4 e Q6)	Subimagens Q4 e Q6 com segmentações e feições correspondentes	Redes treinadas no Experimento 2 e Estrutura de QuadTree	Jetson Nano	Avaliação da capacidade de correspondência de feições com base na sobreposição de segmentos e precisão na correspondência entre as imagens.

Fonte: Autor.

4.1 Experimento 1: Utilização das arquiteturas pré-treinadas

O primeiro teste consistiu em analisar a segmentação, utilizando redes pré-treinadas, nos Q4, Q6 e a banda 650nm da câmara Agrowing com os modelos YOLOv8 e YOLOv9.

Foram selecionadas três cenas distintas, com a coleta de imagens em cada uma dessas bandas. Ao final, tem-se como objetivo comparar o ganho de informação proporcionado por cada banda e avaliar o desempenho das diferentes arquiteturas nessas condições, bem como a possibilidade de uso direto desses modelos em um Computadores de Placa Única.

O primeiro resultado avaliado consistiu na combinação das bandas do Q6, compostas por 685nm (vermelho), 560 nm (verde) e 450nm (azul), resultando em uma imagem colorida. Essa composição foi realçada utilizando a técnica de *stretching*, ignorando os percentis 2 e 98 dos dados, com o objetivo de eliminar *outliers* e permitir uma melhor visualização das imagens, conforme mencionado no Capítulo 3. Com isso, tem-se a Figura 28, onde em (a) é apresentada a imagem original e os outros quadros consistem na aplicação de cada uma das variações dos modelos para a segmentação. Para tal, não foi utilizado filtro com base em confiança ou IoU para seleção das feições, ou seja, esse resultado é a pura aplicação da rede pré-treinada para análise da capacidade de segmentação dessa rede. As estatísticas das confianças obtidas para a primeira análise são apresentadas na Tabela 3, onde a confiança, é uma medida que indica a certeza do modelo de que uma determinada *bounding box* contém um objeto. É importante destacar que esses valores referem-se às detecções (ou segmentos) realizadas em uma única imagem, ou seja, não se tratam de resultados obtidos a partir de múltiplas execuções do modelo. Dentre as estatísticas, destaca-se o percentil 95, que representa o valor abaixo do qual se encontram 95% das confianças. A análise desse percentil é útil para compreender o comportamento do modelo nas detecções de maior confiabilidade, ao mesmo tempo em que contribui para filtrar *outliers* e fornece uma visão mais realista do desempenho.

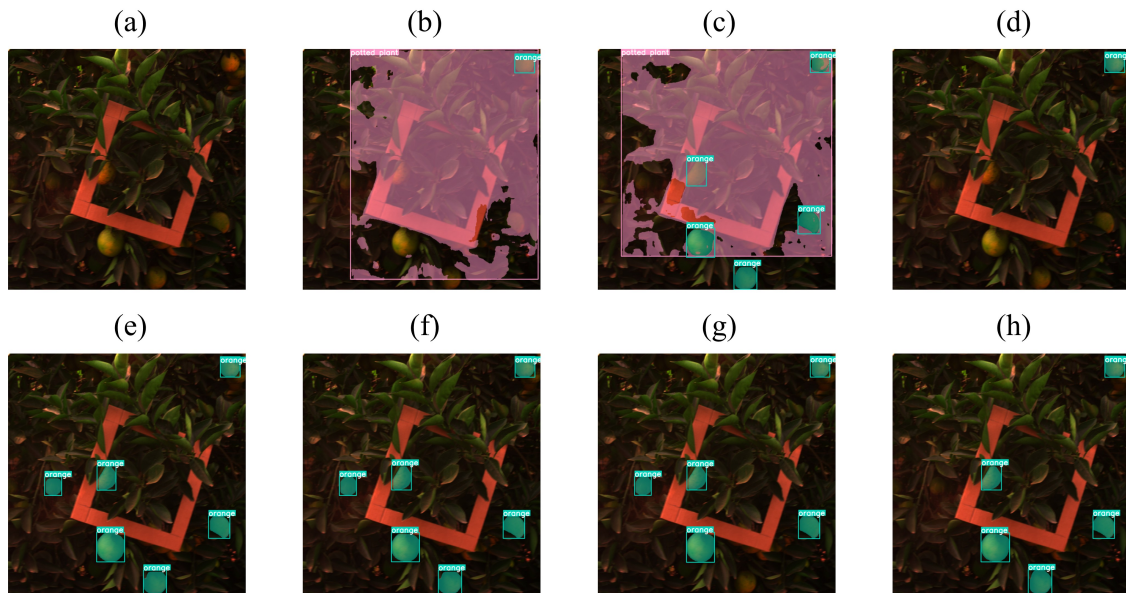
Tabela 3 – Estatísticas da confiança, em %, para cada uma das arquiteturas utilizadas obtidas através da segmentação da primeira imagem e utilizando as bandas do Q6 da câmara Agrowing.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão
YOLOv8-n	45,52	45,52	47,17	1,84
YOLOv8-s	52,53	52,69	72,01	14,16
YOLOv8-m	56,97	56,97	56,97	0,00
YOLOv8-l	48,16	47,35	61,24	10,39
YOLOv8-x	54,05	47,23	75,86	13,86
YOLOv9-c	49,73	51,16	62,61	13,10
YOLOv9-e	56,11	65,08	77,23	19,08

Fonte: Autor.

Em relação à Figura 28, é possível analisar dois aspectos básicos. Com o aumento da complexidade (variantes) da arquitetura, há a possibilidade de segmentar mais laranjas e em condições mais difíceis. Isso se aplica às variações da YOLOv8. No entanto, a YOLOv8-s, apresentada na Figura 28(c), apresenta um resultado satisfatório para a segmentação das laranjas. Nas Figuras 28(b) e 28(c) tem-se uma segmentação que é classificada como

Figura 28 – Segmentação obtida para primeira imagem analisada do Q6 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

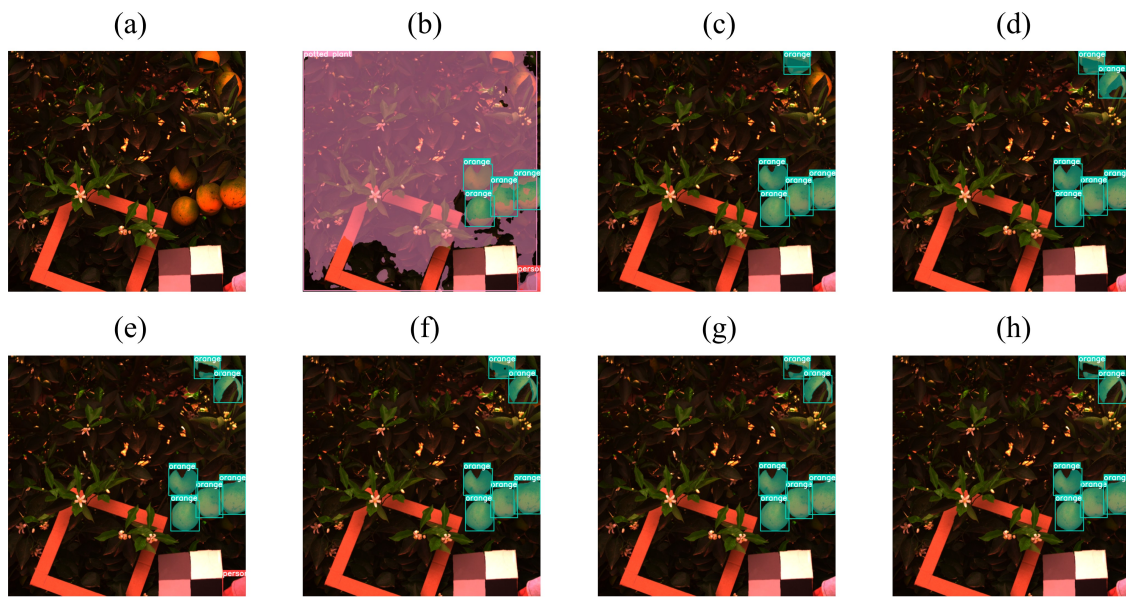
um vaso de plantas. Isso indica que a arquitetura não conseguiu compreender o contexto, visto que o conjunto de dados que foi utilizado para o treinamento dessa não dispõe de todas as classes disponíveis na imagem analisada. Todavia, caso o interesse fosse somente identificar laranjas, um filtro para essa classe já permitiria a aplicação, sendo que, nesse caso, a YOLOv8-s seria uma rede com ótimo custo-benefício.

Em relação às variantes da YOLOv9, observa-se que todas conseguem realizar a segmentação de forma adequada. Ao avaliar a confiança dessas segmentações, observa-se um aumento de 2 pontos percentuais conforme o aumento da complexidade da arquitetura. Todavia, a análise dos segmentos obtidos na 28(h), a arquitetura YOLOv9-e acaba por perder uma laranja e localizando outra que na 28(g) não foi localizada, com isso, tem-se que essas redes possuem desempenho equivalentes para essa aplicação com base no presente conjunto de dados analisado.

Ao analisar a Figura 29, que apresenta uma composição utilizando as bandas do Q6 para outra imagem, observa-se que a arquitetura YOLOv8-n ainda identifica um vaso de plantas. Essa limitação ocorre porque a rede não compreende completamente o contexto da imagem, resultando em interpretações equivocadas. Nesse caso específico, a confiança da detecção é de apenas 36%. Ademais, a rede não conseguiu localizar duas laranjas posicionadas no topo da imagem, mesmo quando as segmentações apresentavam níveis de

confiança superiores a 46%. Já a YOLOv8-s identificou uma laranja a mais em comparação com a rede anterior, porém apresentou uma duplicidade de *bounding boxes*, situação que pode ser solucionada mediante a aplicação de um algoritmo de eliminação de sobreposições. As estatísticas detalhadas da confiança são apresentadas na Tabela 4.

Figura 29 – Segmentação obtida para a segunda imagem analisada do Q6 da câmara Agrowing usando as redes pré-treinadas (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Tabela 4 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da segunda imagem e utilizando as bandas do Q6 da câmara Agrowing.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão
YOLOv8-n	49,53	51,56	62,89	11,35
YOLOv8-s	63,24	69,27	85,31	20,12
YOLOv8-m	79,55	83,55	88,47	8,35
YOLOv8-l	70,98	75,76	84,52	14,94
YOLOv8-x	76,31	83,51	88,43	14,04
YOLOv9-c	71,22	74,54	75,03	5,41
YOLOv9-e	87,23	87,17	91,06	3,27

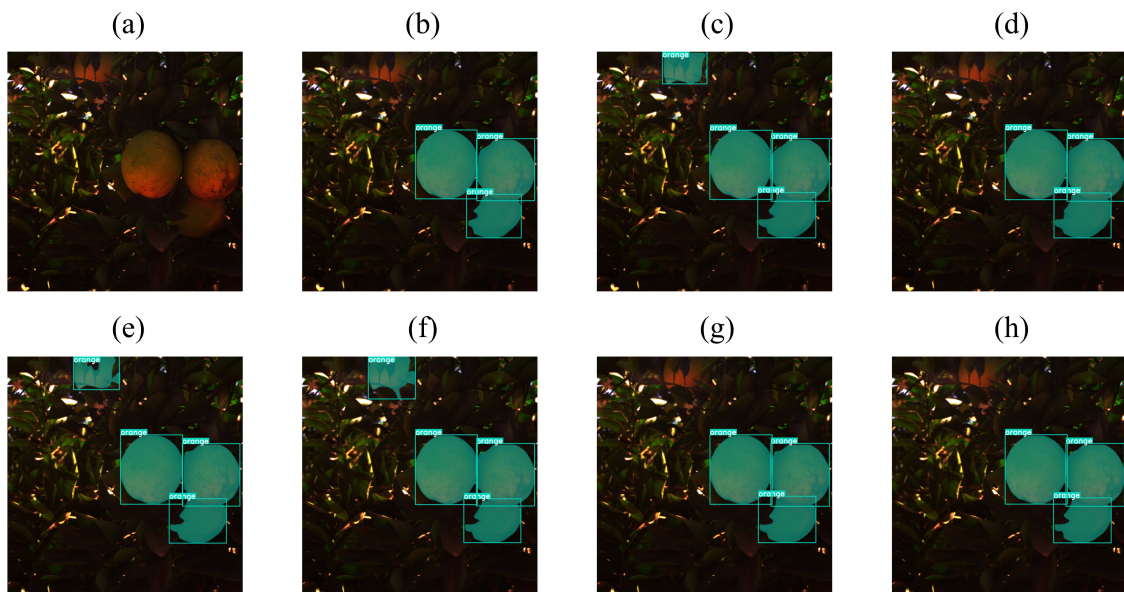
Fonte: Autor.

Ao examinar as arquiteturas YOLOv8-m (Figura 29d), YOLOv8-l (Figura 29e) e YOLOv8-x (Figura 29f), observa-se uma segmentação precisa de todas as laranjas presentes na imagem. Um aspecto distintivo reside no tratamento da folha sobreposta

às laranjas. A arquitetura YOLOv8-l (Figura 29e) demonstrou desempenho superior na distinção entre laranjas e elementos não pertencentes, além de identificar com precisão uma mão parcialmente visível no canto inferior direito da imagem. É importante ressaltar a correlação entre o aumento de parâmetros da rede e o incremento da confiança nas detecções, embora esse aprimoramento implique diretamente no aumento do tempo de processamento.

Na análise da terceira imagem, apresentada na Figura 30(a), que representa o cenário menos complexo entre os três, observaram-se resultados similares entre as diferentes arquiteturas. No entanto, identificou-se uma limitação significativa: as arquiteturas YOLOv8-n, YOLOv8-s, YOLOv9-g e YOLOv9-e não conseguiram detectar a laranja posicionada no topo da imagem. Um aspecto crítico revelado foi a incapacidade de todas as arquiteturas em excluir as folhas sobrepostas à laranja no topo da imagem, resultando na inclusão indevida desses elementos na segmentação. Essa constatação evidencia uma limitação técnica relacionada à capacidade de distinção entre o fruto e elementos adjacentes, especialmente considerando o tamanho do galho e das folhas. Apesar das restrições identificadas, as arquiteturas demonstraram níveis de confiança expressivos. A YOLOv8-n, mesmo sendo a arquitetura com menor complexidade, apresentou valores de confiança superiores a 65%. Os detalhes estatísticos da confiança são apresentados na Tabela 5.

Figura 30 – Segmentação obtida para a terceira imagem analisada do Q6 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

No contexto da combinação de bandas do Q6, identifica-se um potencial promissor

Tabela 5 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da terceira imagem e utilizando as bandas do Q6 da câmara Agrowing.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão
YOLOv8-n	55,83	65,87	74,54	21,38
YOLOv8-s	66,18	74,93	86,39	23,28
YOLOv8-m	83,01	89,17	92,97	11,82
YOLOv8-l	77,75	79,84	92,14	13,22
YOLOv8-x	74,85	79,73	91,38	17,08
YOLOv9-c	81,82	81,22	91,08	8,23
YOLOv9-e	84,58	87,16	92,11	7,87

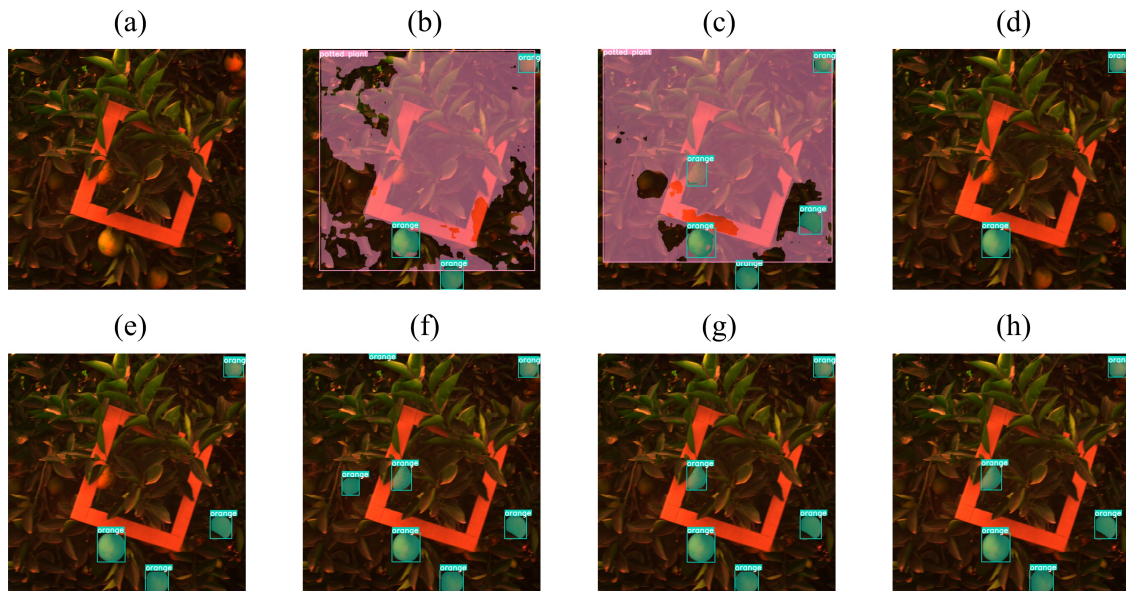
Fonte: Autor.

para segmentação de objetos e extração de características. Entretanto, é imperativo atentar para o processo de *stretching* do histograma, uma vez que a inadequada aplicação dessa técnica pode resultar em imagens com baixa luminosidade, comprometendo a localização precisa dos objetos de interesse. Adicionalmente, ressalta-se a possibilidade de implementação de estratégias de refinamento na análise, tais como a inserção de filtros de classes para eliminação de categorias irrelevantes, aplicação de filtragem por níveis de confiança e precisão da segmentação

Na análise do Q4, identificaram-se padrões de segmentação e níveis de confiança comparáveis aos resultados obtidos anteriormente com o Q6. Na Figura 31 são ilustrados os resultados da primeira imagem, revelando nuances significativas na performance da arquitetura YOLOv8-n. Notou-se um aumento expressivo na capacidade de segmentação da YOLOv8-n nesta configuração. Enquanto na análise prévia a rede havia identificado apenas uma laranja, nesta análise foram segmentadas três laranjas. Essa variação demonstra a sensibilidade da arquitetura às bandas do Q4. Adicionalmente, observou-se que o segmento correspondente às laranjas apresentou um aumento nos níveis de confiança, enquanto o segmento referente ao vaso de plantas registrou uma redução. Essa variação sugere uma possível influência das bandas na precisão da detecção de diferentes objetos. Os detalhes estatísticos das confianças estão apresentados na Tabela 6.

Na análise da arquitetura YOLOv8-s, observou-se uma melhoria nos resultados, i.e., um aumento nos valores de níveis de confiança para as medições de ambos os objetos. É importante ressaltar que, dependendo do contexto de aplicação, a presença de segmentos de classes irrelevantes pode ser mitigada mediante estratégias de filtragem direcionadas. Essa abordagem permite refinar a segmentação, selecionando exclusivamente as classes de interesse para análise. Quanto à arquitetura YOLOv8-m, identificou-se uma redução na quantidade de laranjas segmentadas. Enquanto na configuração anterior foram localizadas seis laranjas, na presente análise o número foi reduzido para quatro, evidenciando uma perda na capacidade de segmentação, ao realizar uma comparação com os resultados

Figura 31 – Segmentação obtida para a primeira imagem analisada do Q4 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Tabela 6 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da primeira imagem e utilizando as bandas do Q4 da câmara Agrowing.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão
YOLOv8-n	37,04	37,24	44,87	6,56
YOLOv8-s	57,62	57,78	71,62	10,51
YOLOv8-m	54,63	54,63	59,68	5,61
YOLOv8-l	50,19	48,04	63,69	10,89
YOLOv8-x	50,79	46,68	71,43	13,54
YOLOv9-c	46,69	41,76	63,07	12,38
YOLOv9-e	64,14	72,49	83,03	17,69

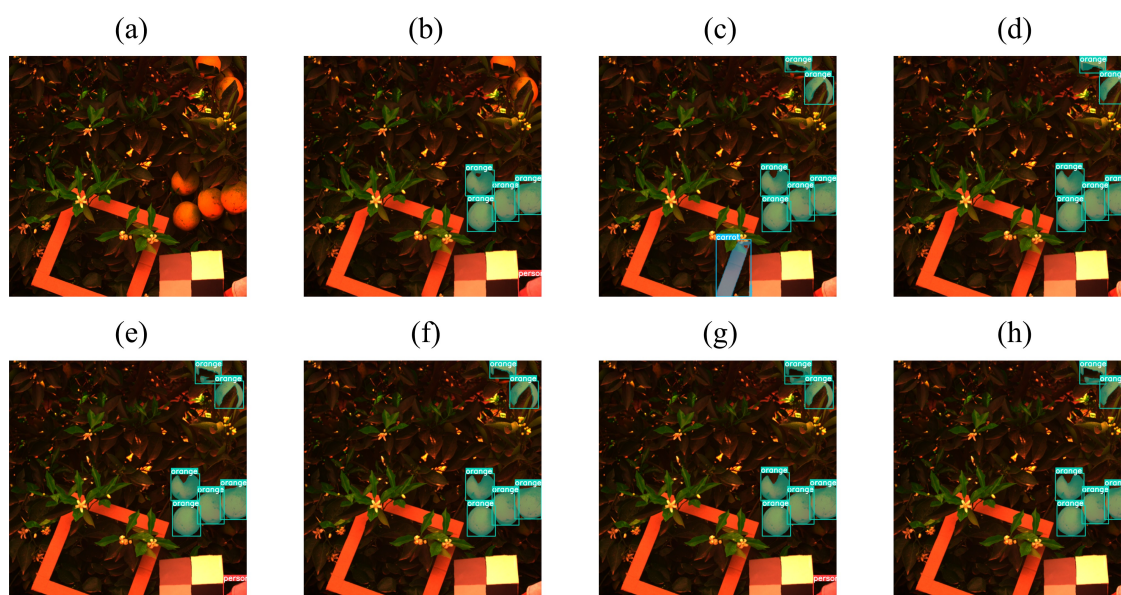
Fonte: Autor.

apresentados na Figura 28.

Já para a YOLOv8-x observou-se um aumento de uma laranja no topo da imagem da Figura 31(f) que até então não havia sido encontrada por nenhuma rede pré-treinada. Em relação às arquiteturas YOLOv9-c e YOLOv9-e, foram observados resultados semelhantes. Apenas a YOLOv9-c não conseguiu localizar uma laranja que anteriormente era possível, mas conseguiu localizar outra que antes não conseguia. Assim, tem-se o mesmo número de objetos localizados.

Ao analisar a segunda imagem apresentada na Figura 32, observa-se que houve uma melhoria significativa nos resultados obtidos para a segmentação, sendo que as arquiteturas YOLOv8-n e YOLOv8-s, apresentadas na Figura 32(b) e (c), conseguiram aumentar a quantidade de laranjas segmentadas e a confiança das medições. Todavia, na YOLOv8-s, observa-se que uma parte dos alvos foi segmentada como uma cenoura, com do nível de confiança de 25%. Com isso, esse resultado pode ser removido via filtragem do mínimo de confiança para uma medição ou pela classe que foi encontrada. As estatística da confiança são apresentadas na Tabela 7.

Figura 32 – Segmentação obtida para a segunda imagem analisada do Q4 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Na análise da arquitetura YOLOv8-m, observou-se um desempenho consistente entre as configurações das subimagens Q6 e Q4. A variação na média da confiança foi marginal, passando de 88,47% (Q6) para 88,99% (Q4). A complexidade da segmentação revelou-se na variabilidade da confiança dos segmentos individuais. Alguns segmentos de laranja experimentaram um incremento na confiança, enquanto outros apresentaram ligeira redução. Adicionalmente, a quantidade total de segmentos permaneceu constante. O aspecto mais notável desta análise reside na qualidade aprimorada da segmentação ao utilizar a subimagem do Q4. Nesta configuração, conseguiu-se uma separação mais precisa entre a folha sobreposta e o objeto laranja subjacente, demonstrando maior sensibilidade da arquitetura à configuração espectral utilizada.

As análises realizadas para a arquitetura YOLOv8-m estendem-se igualmente às

Tabela 7 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da segunda imagem e utilizando as bandas do Q4 da câmara Agrowing.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão
YOLOv8-n	55,92	60,18	70,26	14,53
YOLOv8-s	54,90	61,62	82,12	21,07
YOLOv8-m	75,92	81,39	88,99	16,06
YOLOv8-l	68,52	68,17	85,62	14,54
YOLOv8-x	74,30	79,76	85,16	11,66
YOLOv9-c	59,12	66,18	76,96	18,58
YOLOv9-e	85,98	88,46	90,64	5,33

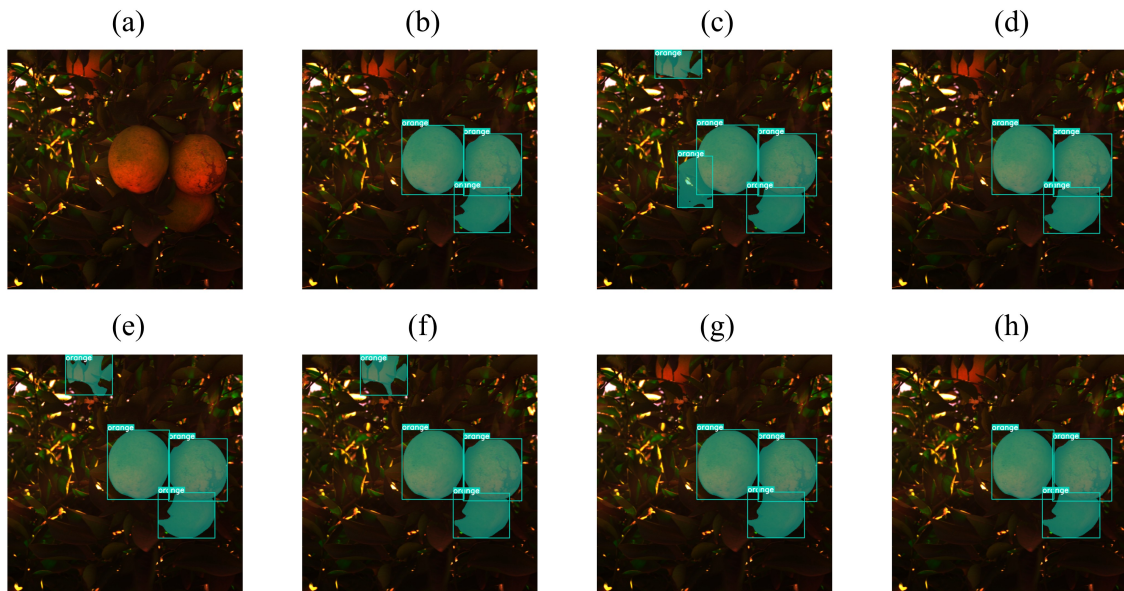
Fonte: Autor.

variações YOLOv8-l e YOLOv8-x, caracterizadas por uma delimitação de segmentos progressivamente mais refinada. A análise comparativa dos percentis 95 da confiança revelou variações sutis, com oscilações de 84,52% para 85,62% e de 88,43% para 85,16%, respectivamente. No que concerne às arquiteturas baseadas na YOLOv9, observaram-se comportamentos diferenciados. A versão YOLOv9-c demonstrou dificuldades específicas na segmentação de uma laranja posicionada no topo da imagem, um desafio não identificado em análises anteriores. Paradoxalmente, essa mesma configuração apresentou capacidade de localizar e segmentar uma figura humana no canto da imagem. Já a YOLOv9-e manteve características de desempenho consistentes, sem alterações significativas.

A análise da última imagem, representada na Figura 33, evidenciou um incremento nos níveis de confiança, particularmente nas arquiteturas de menor complexidade. As configurações YOLOv8-n e YOLOv8-s demonstraram variações expressivas: para o Q6, registraram-se percentis 95 de 74,54% e 86,39%, enquanto no Q4 esses valores evoluíram para 84,52% e 87,07%, respectivamente. As demais arquiteturas apresentaram oscilações marginais, resultando em um desempenho globalmente equivalente. Os detalhes estatísticos constam na Tabela 8. Os resultados evidenciam a complexidade e sensibilidade das diferentes arquiteturas de redes neurais, destacando como variações sutis na configuração de bandas podem impactar significativamente os processos de segmentação e identificação de objetos.

Por fim, analisou-se a utilização de uma rede pré-treinada empregando exclusivamente a banda de comprimento de onda de 650 nm. Ao realizar o teste com a primeira imagem apresentada na Figura 34, observou-se uma redução significativa no número de laranjas detectadas. Essa redução foi verificada em todas as arquiteturas analisadas, exceto na YOLOv8-x. Quando utilizadas as bandas do Q6, eram detectadas seis laranjas, conforme previamente ilustrado na Figura 28. Contudo, ao utilizar apenas a banda de 650 nm, foi possível identificar cinco laranjas. Além disso, houve uma diminuição expressiva no percentil 95 da confiança, passando de 75,86% para 45,44%. Esses resultados, obtidos para

Figura 33 – Segmentação obtida para a terceira imagem analisada do Q4 da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Tabela 8 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da terceira imagem e utilizando as bandas do Q4 da câmara Agrowing.

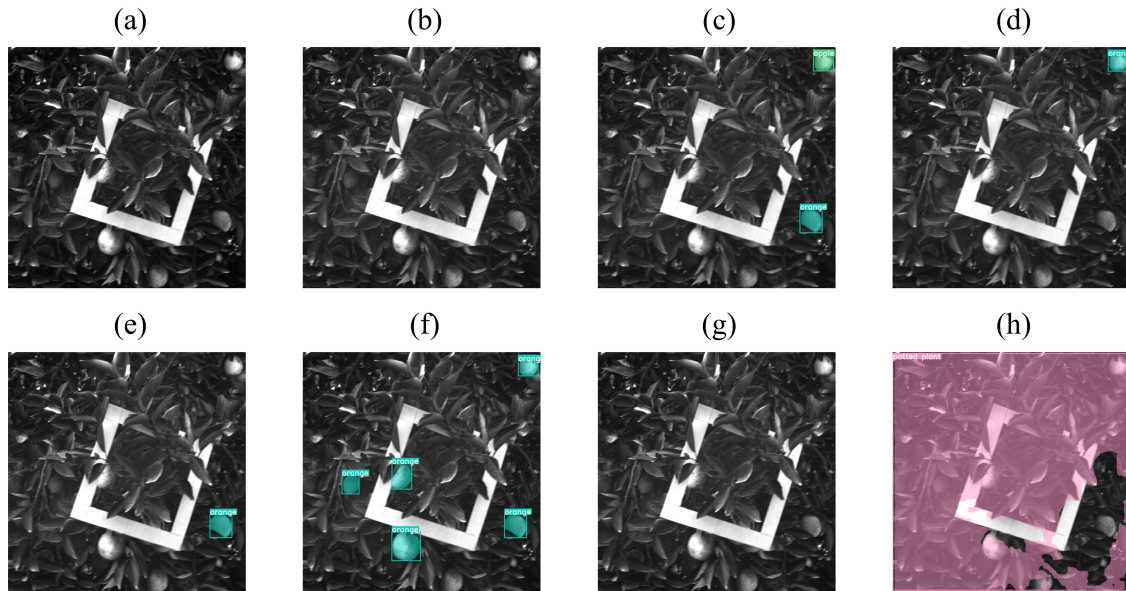
Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão
YOLOv8-n	64,55	67,47	84,52	19,15
YOLOv8-s	59,40	63,13	87,07	24,31
YOLOv8-m	82,81	87,67	92,44	10,83
YOLOv8-l	79,14	85,38	92,95	16,17
YOLOv8-x	71,72	72,63	90,07	16,37
YOLOv9-c	79,80	83,53	90,57	11,27
YOLOv9-e	87,19	90,62	93,05	6,85

Fonte: Autor.

o comprimento de onda de 650 nm, estão detalhados na Tabela 9. Um aspecto notável é que, mesmo na ausência de informação cromática na análise, foi possível classificar corretamente os objetos como laranjas.

Ao comparar a segmentação obtida com as bandas do Q4 àquela gerada pelas bandas do Q6, observou-se uma redução no percentil 95 da confiança, de 71,43% para 45,44%, especificamente para a arquitetura YOLOv8-x. Adicionalmente, as arquiteturas YOLOv8-n, YOLOv8-m e YOLOv9-c não foram capazes de localizar nenhuma laranja,

Figura 34 – Segmentação obtida para a primeira imagem analisada do comprimento utilizando o comprimento de onda do 650 nm da câmara Agrowing usando as redes pré-treinadas, sem quaisquer tipo de filtragem (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Tabela 9 – Estatísticas da confiança para cada uma das arquiteturas utilizadas obtidas através da segmentação da primeira imagem e utilizando o Q6 da câmara Agrowing.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão
YOLOv8-n	0,00	0,00	0,00	0,00
YOLOv8-s	39,88	39,88	49,46	10,64
YOLOv8-m	0,00	0,00	0,00	0,00
YOLOv8-l	30,29	30,29	30,29	0,00
YOLOv8-x	38,59	38,59	45,44	7,61
YOLOv9-c	0,00	0,00	0,00	0,00
YOLOv9-e	42,18	42,18	42,18	0,00

Fonte: Autor.

enquanto a YOLOv8-l e a YOLOv9-e identificaram apenas uma laranja cada. Em função disso, o desvio-padrão associado a essas arquiteturas encontra-se zerado, conforme indicado na Tabela 9.

Em síntese, a partir de todos os resultados e análises obtidos ao longo da seção, tem-se que, ao utilizar somente uma banda, como a de 650 nm apresentada ao longo da seção, as arquiteturas da família YOLOv8 comumente apresentam os melhores resultados. Mesmo assim, há a necessidade de cores para distinguir corretamente os objetos, sendo recomendada

a utilização de uma composição colorida. Além disso, as arquiteturas da YOLOv9 não conseguem trabalhar bem com somente a informação de uma banda, apresentando baixo nível de confiança e localizando poucas laranjas. Por fim, observa-se a presença de falsos negativos utilizando arquiteturas baseadas na YOLOv9.

Ao analisar a combinação de bandas, observou-se que a ocorrência de falsos negativos ocorreu somente nas redes mais simples, sendo essas as redes YOLOv8-n e YOLOv8-s. Conforme a complexidade da rede aumenta, os níveis de confiança também aumentam, conforme análise do percentil 95. No entanto, observa-se uma leve perda do nível de confiança da YOLOv8-l para a YOLOv8-x. Enquanto o maior ganho do nível de confiança é da YOLOv8-s para a YOLOv8-m, não se tem um aumento significativo do nível de confiança posteriormente, mas a complexidade computacional (tempo de processamento) aumenta demasiadamente.

Ao analisar os resultados obtidos com a YOLOv9-c e a YOLOv9-e, observou-se que eles são superiores até mesmo aos da YOLOv8-x, que apresenta a maior quantidade de parâmetros e consegue localizar a grande maioria das laranjas. Dessa forma, destaca-se a otimização de recursos e o custo-benefício de sua utilização. Tais observações foram levadas em consideração nos testes que foram realizados e descritos nas próximas seções.

4.2 Experimento 2: Adequação e avaliação das arquiteturas para a segmentação de folhas

Conforme descrito na Seção 3.4, esta seção apresenta o detalhamento do processo de treinamento e a avaliação dos resultados obtidos com as diferentes variações das arquiteturas YOLOv8 e YOLOv9 para segmentação de folhas. Todos os parâmetros das redes foram mantidos constantes para assegurar uma comparação justa, enquanto os valores das sementes (um dos parâmetros de configuração da rede) para geração de comportamento randômico foram idênticos em todos os testes, garantindo a reprodutibilidade dos resultados. O experimento utilizou redes YOLO treinadas a partir de um conjunto de dados rotulados de forma semi-automática, aplicando o algoritmo Segment Anything Model de Kirillov *et al.* (2023). Para este estudo, foram empregadas as técnicas de *Transfer Learning* e *Fine Tuning*, utilizando os modelos pré-treinados como base. O objetivo foi ajustar os pesos das redes, transferindo o conhecimento prévio adquirido no treinamento com o MS COCO para viabilizar a segmentação de folhas nesta nova tarefa.

Nas Tabelas 10, 11 e 12, são apresentados os resultados obtidos para todas as arquiteturas durante as etapas de treinamento, validação e teste. Em termos de *precision* no treinamento, o modelo com o melhor desempenho foi a YOLOv8-s (99,24%), seguida pela YOLOv9-c (98,99%). Todos os modelos apresentaram resultados semelhantes em relação ao recall, exceto a YOLOv8-x (93,66%). Quanto às métricas mAP50 e mAP50-95,

a YOLOv8-s (99,49% e 96,17%) obteve os melhores resultados quantitativos, seguida pela YOLOv8-n (99,43% e 96,68%).

Tabela 10 – Métricas das arquiteturas obtidas durante o processo de treinamento.

Modelo	<i>Precision</i>	<i>Recall</i>	F1-Score	mAP50	mAP50-95
YOLOv8-n	98,22	99,50	98,85	99,43	96,87
YOLOv8-s	99,25	99,46	99,36	99,49	96,17
YOLOv8-m	97,75	98,10	97,92	99,39	95,82
YOLOv8-l	96,57	99,04	97,79	99,29	94,91
YOLOv8-x	93,67	95,73	94,69	98,03	89,72
YOLOv9-c	98,99	98,59	98,79	99,45	94,82
YOLOv9-e	97,27	98,32	97,79	99,37	92,90

Fonte: Autor.

A YOLOv8-x apresentou os valores mais baixos em todas as métricas. Isso pode sugerir que o aumento no tamanho do modelo não resultou em ganhos de desempenho, possivelmente devido à quantidade desnecessária de parâmetros da variação utilizada. Por outro lado, a YOLOv9-c mostrou resultados consistentes e próximos dos melhores valores, com F1-score (98,78%) e mAP50 (99,44%) quase equivalentes aos melhores modelos. Embora não tenha se destacado em nenhuma métrica específica, o modelo mostrou grande estabilidade, mantendo um equilíbrio entre as métricas e evitando resultados insatisfatórios. Além disso, observa-se uma tendência de queda na *Precision* e no mAP à medida que os modelos YOLOv8 aumentam de tamanho, o que pode indicar que modelos menores são suficientes para o conjunto de dados utilizado.

Durante a validação, conforme apresentado na Tabela 11, a YOLOv9-c (91,96%) obteve os melhores resultados em termos de *precision*, enquanto a YOLOv8-x (85,47%) apresentou o melhor desempenho no F1-score. Em relação às métricas de segmentação, a YOLOv8-x obteve os melhores resultados nas métricas mAP50 e mAP50-95, com valores de 86,87% e 72,11%, respectivamente, isso indica que o modelo conseguiu equilibrar *Precision* e *recall*, além de oferecer um desempenho superior em cenários mais complexos (mAP50-95). Por outro lado, a YOLOv8-m apresentou o melhor equilíbrio entre as métricas, com um *F1-Score* de 80,65% e valores competitivos de mAP50-95 81,42% e mAP50-95 (71,19%). Isso o torna uma escolha robusta para aplicações que demandam desempenho estável.

Durante a fase de testes, conforme apresentado na Tabela 12, observa-se que o modelo com melhor desempenho em termos de *precision* foi a YOLOv8-s (96,25%), seguido pela YOLOv8-l (93,02%). Em relação ao *Recall*, os melhores resultados foram obtidos pelos modelos YOLOv8-s (77,27%) e YOLOv8-m (77,15%), que apresentaram valores quase idênticos, seguidos pela YOLOv8-x (72,72%). Quanto ao F1-score, a melhor arquitetura durante a fase de testes foi a YOLOv8-s (85,72%), seguida pela YOLOv8-x (79,63%). Para as métricas de segmentação, os modelos com melhor desempenho foram a

Tabela 11 – Métricas das arquiteturas treinadas obtidas durante o processo de validação.

Modelo	<i>Precision</i>	<i>Recall</i>	F1-Score	mAP50	mAP50-95
YOLOv8-n	88,06	71,96	79,20	78,98	67,63
YOLOv8-s	83,18	71,03	76,62	78,88	68,40
YOLOv8-m	84,61	77,06	80,66	81,42	71,20
YOLOv8-l	82,46	76,64	79,44	82,52	68,53
YOLOv8-x	86,58	84,40	85,47	86,88	72,12
YOLOv9-c	91,97	69,16	78,95	80,76	64,03
YOLOv9-e	81,14	76,64	78,83	79,66	66,22

Fonte: Autor.

Tabela 12 – Métricas das arquiteturas treinadas e validadas obtidas durante o processo de teste.

Modelo	<i>Precision</i>	<i>Recall</i>	F1-Score	mAP50	mAP50-95
YOLOv8-n	88,22	68,07	76,84	72,86	59,42
YOLOv8-s	96,26	77,27	85,73	79,46	66,24
YOLOv8-m	80,93	77,15	78,99	73,85	58,94
YOLOv8-l	93,02	68,18	78,69	78,19	62,89
YOLOv8-x	87,99	72,73	79,64	76,92	63,25
YOLOv9-c	92,50	63,64	75,40	80,12	64,57
YOLOv9-e	86,34	68,18	76,20	73,25	57,41

Fonte: Autor.

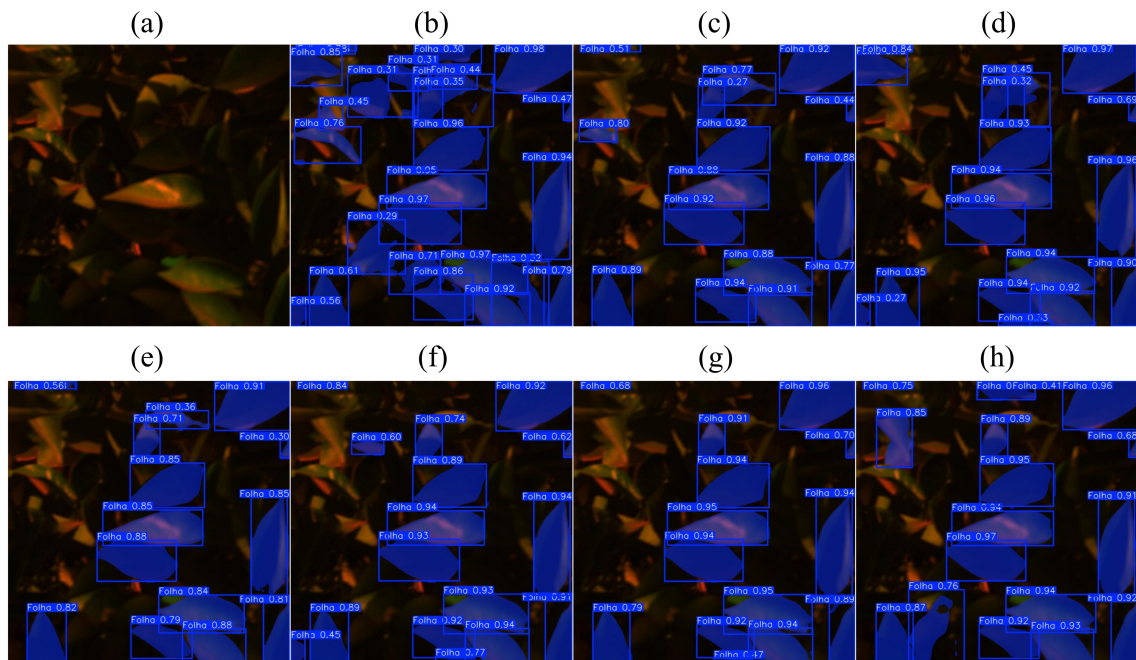
YOLOv8-s (79,46% e 66,24% para mAP50 e mAP50-95, respectivamente) e a YOLOv9c (80,12% e 64,56% para mAP50 e mAP50-95, respectivamente). Percebe-se que, pelos resultados obtidos, valores *Precision* são mais elevados em relação aos valores *Recall*. Isso implica que o modelo realiza um número menor de classificações como folhas, mas a maioria delas é correta. Com base nos resultados quantitativos apresentados, a YOLOv8-s demonstra uma boa relação entre desempenho e eficiência. No entanto, a YOLOv8-x se mostra promissora para determinadas aplicações, desde que sua eficiência seja devidamente avaliada, especialmente no que se refere ao custo computacional.

4.2.1 Avaliação dos níveis de confiança dos resultados

Em termos do nível de confiança, foram realizados testes com duas imagens diferentes do conjunto de dados de teste as quais foram analisadas e detalhadas a seguir. A primeira imagem analisada é apresenta na Figura 35 e as métricas de confiança dessa são apresentadas na Tabela 13.

Com base na Tabela 13, a YOLOv8n apresenta a maior média de confiança (85,61%) e a melhor confiança mediana (92,86%), mas tem um desvio padrão relativamente menor (14,06%), indicando boas previsões, mas com variabilidade moderada. A YOLOv8-s tem uma confiança média ligeiramente inferior à YOLOv8-n (82,66%), mas apresenta uma boa

Figura 35 – Resultados obtidos para as redes treinadas para a primeira imagem analisada (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Tabela 13 – Níveis de confiança obtidos para as redes treinadas para a primeira imagem analisada.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão	Qtd. de instâncias
YOLOv8-n	85,61	92,86	95,68	14,06	14
YOLOv8-s	82,66	90,05	94,21	14,48	16
YOLOv8-m	82,20	91,44	96,07	18,79	17
YOLOv8-l	74,00	91,28	96,12	26,54	18
YOLOv8-x	63,81	60,70	97,06	25,33	27
YOLOv9-c	78,07	88,15	92,73	19,84	15
YOLOv9-e	71,34	82,48	89,21	21,71	15

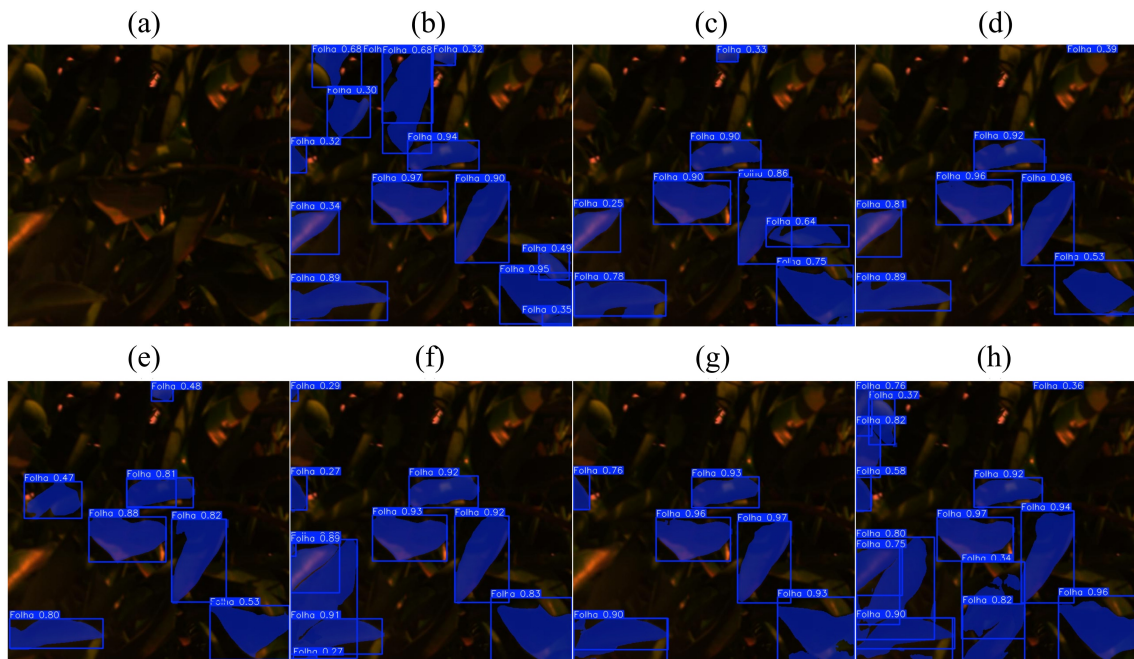
Fonte: Autor.

confiança mediana (90,05%) e um desvio padrão relativamente alto (14,48%). O YOLOv8-x apresenta a menor média de confiança (63,81%) e a maior variabilidade (25,33%), sugerindo que seu desempenho pode ser mais instável em relação à confiança. As versões YOLOv9-c e YOLOv9-e, que são uma evolução do YOLOv9, têm um desempenho um pouco inferior a alguns modelos YOLOv8 em termos de média de confiança, com YOLOv9-c sendo ligeiramente melhor que YOLOv9-e.

Os resultados para a segunda imagem analisada estão apresentados na Figura 36 e na Tabela 14. Esses resultados apresentam variabilidade nas estimativas de desempenho, o que explica as quedas observadas nas métricas de confiança. A imagem analisada neste

estudo é mais difícil de avaliar empiricamente. Os modelos que apresentaram as maiores quedas foram o YOLOv8-s e o YOLOv9-e, enquanto o YOLOv8-l apresentou os melhores resultados.

Figura 36 – Resultados obtidos para as redes treinadas para a segunda imagem analisada (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Tabela 14 – Níveis de confiança obtidos para as redes treinadas para a segunda imagem analisada.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão	Qtd. de instâncias
YOLOv8-n	83,10	92,67	97,02	20,29	7
YOLOv8-s	66,56	83,21	92,57	29,43	11
YOLOv8-m	71,21	80,70	96,16	23,30	16
YOLOv8-l	78,08	88,90	96,18	20,98	7
YOLOv8-x	59,96	50,38	95,79	26,13	15
YOLOv9-c	67,49	76,09	90,02	23,64	8
YOLOv9-e	63,38	66,68	85,61	20,37	8

Fonte: Autor.

A arquitetura YOLOv8-n apresenta uma média de confiança de 83,10%, ligeiramente abaixo da média de 85,61% apresentada na Tabela 13. No entanto, a confiança no percentil de 95% (97,02%) melhorou ligeiramente, indicando que o modelo ainda apresenta boas previsões em um alto percentual de confiança. O desvio padrão aumentou para 20,29%, sugerindo maior variabilidade nas previsões. A YOLOv8-s teve uma redução considerável

em sua média de confiança (66,56%) e mediana (83,21%), além de maior variabilidade no desvio padrão (29,43%). A YOLOv8-m apresentou uma queda na média de confiança (71,21%) e na mediana (80,70%) em comparação com a Tabela 13, o que pode indicar uma diminuição na precisão do modelo. O desvio padrão aumentou para 23,30%, sugerindo maior variação nas previsões. O número de medições foi ligeiramente reduzido, para 16, o que pode contribuir para a maior variabilidade dos resultados. A arquitetura YOLOv8-l, por sua vez, obteve uma melhora na média de confiança (78,08%) e na mediana (88,90%), com desvio padrão relativamente baixo (20,98%), indicando maior consistência.

A arquitetura YOLOv8-x apresentou uma leve queda na média de confiança (59,96%) e na mediana (50,38%), além de um aumento no desvio padrão (26,13%), sugerindo maior instabilidade nas previsões. A YOLOv9c, por sua vez, apresentou uma diminuição na média de confiança (67,49%) e na mediana (76,09%) em comparação com a Tabela 13, além de um aumento no desvio padrão (23,64%). A YOLOv9e também mostrou uma queda na média de confiança (63,38%) e na mediana (66,68%) em relação à Tabela 13, além de um desvio padrão maior (20,37%).

Notou-se uma redução da quantidade de medições realizadas, devido à dificuldade de se detectar e segmentar a imagem. Apesar disso, as arquiteturas YOLOv8-n e YOLOv8-l ainda demonstram um bom desempenho com menos medições. Entretanto, modelos como YOLOv8-s, YOLOv9-c e YOLOv9-e sofreram quedas significativas em suas métricas. No entanto, observa-se que as arquiteturas conseguem detectar e segmentar as folhas de maneira satisfatória, apesar da redução da confiança em alguns casos.

4.3 Execução das redes nas diferentes plataformas

Os tempos de execução obtidos para os experimentos são apresentados nas Tabelas 15, 16 e 17, detalhando os resultados na Google Colab sem placa de vídeo, com a placa de vídeo Tesla T4, e na NVIDIA Jetson Nano com sua GPU integrada, respectivamente. Um aspecto importante a destacar é que o modelo foi previamente treinado, conforme mencionado na metodologia. Em relação às métricas do modelo, isto é, a eficiência na tarefa, observa-se que não há quaisquer perdas, visto que as execuções em ambos os ambientes utilizam a mesma biblioteca, sendo, portanto, comparáveis. A média do tempo foi obtida a partir de 30 execuções consecutivas, com a eliminação de *outliers*. Cada teste consistiu na conexão da NVIDIA Jetson Nano, na execução do processo e, em seguida, no desligamento do dispositivo, repetindo o procedimento 30 vezes para cada modelo e variação.

Ao analisar as Tabelas 15, 16 e 17, observa-se que a introdução de uma GPU Tesla T4 no Google Colab resulta em uma redução significativa nos tempos de execução para todos os modelos. Por exemplo, o modelo YOLOv8-n, que levou 168 ms para detecção sem GPU,

Tabela 15 – Média do tempo de execução para cada uma das arquiteturas executadas no Google Colab sem placa de vídeo.

Modelo	Parâmetros	Deteção	Deteção + Segmentação
YOLOv8-n	3M	168 ms	203 ms
YOLOv8-s	11M	401 ms	523 ms
YOLOv8-m	27M	947 ms	1210 ms
YOLOv8-l	45M	1725 ms	2302 ms
YOLOv8-x	71M	2799 ms	3388 ms
YOLOv9-c	27M	1425 ms	1943 ms
YOLOv9-e	59M	2947 ms	3436 ms

Fonte: Autor.

Tabela 16 – Média do tempo de execução para cada uma das arquiteturas executadas no Google Colab com a Tesla T4, comparando resultados sem e com FP16.

Modelo	Sem FP16		Com FP16	
	Deteção	Deteção + Segmentação	Deteção	Deteção + Segmentação
YOLOv8-n	8 ms	9 ms	7 ms	8 ms
YOLOv8-s	13 ms	14 ms	12 ms	14 ms
YOLOv8-m	16 ms	20 ms	15 ms	18 ms
YOLOv8-l	20 ms	22 ms	19 ms	22 ms
YOLOv8-x	32 ms	39 ms	31 ms	38 ms
YOLOv9-c	21 ms	26 ms	16 ms	18 ms
YOLOv9-e	33 ms	37 ms	28 ms	30 ms

Fonte: Autor.

reduziu esse tempo para apenas 8 ms com a GPU. Essa aceleração destaca a importância de utilizar hardware especializado para tarefas intensivas de processamento, como a visão computacional. A GPU permite que modelos mais complexos, que seriam impraticáveis sem aceleração, possam ser usados em aplicações de tempo real. A NVIDIA Jetson Nano, embora menos potente que a Tesla T4, oferece uma solução intermediária que equilibra desempenho e eficiência energética. Seus tempos de execução são significativamente mais baixos do que os do Google Colab sem GPU, mas mais altos do que os do Google Colab com GPU.

A escolha do modelo YOLO adequado e do ambiente computacional deve ser orientada pelos requisitos específicos da aplicação. Modelos mais simples e rápidos, como o YOLOv8-n, são ideais para aplicações que exigem alta velocidade de processamento com recursos limitados. Modelos mais complexos, como o YOLOv8-x e o YOLOv9e, são mais adequados para aplicações que podem se beneficiar de maior precisão e que têm acesso a hardware acelerado, como GPUs poderosas. A NVIDIA Jetson Nano, equipada com uma GPU baseada na arquitetura Maxwell, mostrou desempenho satisfatório em comparação com o Google Colab sem GPU, que utiliza CPUs tradicionais. Embora a Jetson Nano

Tabela 17 – Média do tempo de execução para cada uma das arquiteturas executadas na NVIDIA Jetson Nano, agrupadas por FP32 e FP16.

Modelo	FP32		FP16	
	Detecção	Detecção + Segmentação	Detecção	Detecção + Segmentação
YOLOv8-n	220 ms	310 ms	213 ms	301 ms
YOLOv8-s	482 ms	645 ms	419 ms	600 ms
YOLOv8-m	1086 ms	1385 ms	1035 ms	1370 ms
YOLOv8-l	1773 ms	2427 ms	1712 ms	2170ms
YOLOv8-x	2939 ms	3455 ms	2839 ms	3190 ms
YOLOv9-c	1222 ms	1685 ms	1193 ms	1650 ms
YOLOv9-e	2493 ms	3131 ms	2412 ms	2876 ms

Fonte: Autor.

tenha menos núcleos CUDA e memória em comparação com a Tesla T4, que utiliza a arquitetura Turing com núcleos tensores especializados para tarefas de *Deep Learning*, ela ainda proporciona tempos de resposta adequados para muitas aplicações. Essa eficiência é particularmente relevante em ambientes de computação de borda, como em algumas aplicações na agricultura digital.

Ao analisar o comportamento utilizando o formato FP16 apresentado na Tabela 16 observa-se que a diferença de FP32 para FP16, não se faz muito significativa nas variações mais leves das arquiteturas, sendo que na maioria dos casos a redução foi de cerca de 1 milissegundo, isso é observado para todos as variações da YOLOv8. Apesar disso, ao se executar as variações da YOLOv9, observa-se uma redução significativa dos tempos de execução, sendo que, para a YOLOv9c observa-se uma redução de 23,81% (21ms para 16ms) e 30,77% (26 ms para 18 ms) para detecção e segmentação, respectivamente. Enquanto, para a YOLOv9e, tem-se uma redução de 15,15% (33 ms para 28ms) e 18,92% (37 ms para 30ms) para detecção e segmentação, respectivamente.

Enquanto, para a execução das arquiteturas na NVIDIA Jetson Nano, a redução se manteve comumente inferior a 4% do tempo, as únicas arquiteturas que se destacam foram a YOLOv8-s, com uma redução de 13,07% para detecção e 6,98% para segmentação e os processos de segmentação da YOLOv8-l, YOLOv8-x e YOLOv9-e com reduções de 10,59%, 7,67% e 8,14%, respectivamente. Com isso, nota-se que apesar de ocorrer uma redução no tempo de execução utilizando FP16, esse tempo não é tão significativo para a maioria dos casos, dito isso, explorou-se outras alternativas para melhorar ainda mais o desempenho, com isso, analisou-se o tempo de execução utilizando TensorRT.

Ao executar os códigos utilizando o TensorRT, realiza-se a conversão do modelo do formato ".pt" para ".engine". Esse processo de conversão costuma demorar cerca de 30 minutos no Google Colab com a Tesla T4. Com o modelo pronto, é possível executá-lo em quaisquer plataformas que atendam aos requisitos mínimos. Essa metodologia resulta em

uma redução no tempo de execução dos modelos. O TensorRT aplica várias técnicas de otimização, como fusão de camadas e quantização, para melhorar a eficiência da inferência. No entanto, ao contrário do que foi observado utilizando FP16, para as arquiteturas mais leves, a redução percentual do tempo de execução é maior. Os tempos de execução para cada uma das arquiteturas no Google Colab são apresentados na Tabela 18.

Tabela 18 – Média do tempo de execução para cada uma das arquiteturas executadas no Google Colab com a Tesla T4 e as métricas obtidas utilizando TensorRT.

Modelo	Deteção	D + S	Red. Percentual (D)	Red. Percentual (D + S)
YOLOv8-n	3 ms	4 ms	62,50%	55,55%
YOLOv8-s	5 ms	6 ms	61,54%	57,14%
YOLOv8-m	11 ms	13 ms	31,25%	35,00%
YOLOv8-l	17 ms	20 ms	15,00%	9,09%
YOLOv8-x	25 ms	29 ms	21,88%	25,64%
YOLOv9-c	13 ms	16 ms	38,09%	38,46%
YOLOv9-e	29 ms	32 ms	12,12%	13,51%

Fonte: Autor.

A utilização de FP16, embora beneficie o processo de treinamento, não proporciona uma melhoria significativa no tempo de execução durante a inferência. Por outro lado, os ganhos obtidos com o uso do TensorRT são expressivos. Contudo, essa otimização está atualmente limitada ao ambiente do Google Colab, não sendo compatível com a NVIDIA Jetson Nano, devido à versão do JetPack disponível para o dispositivo e à falta de integração com a biblioteca da Ultralytics.

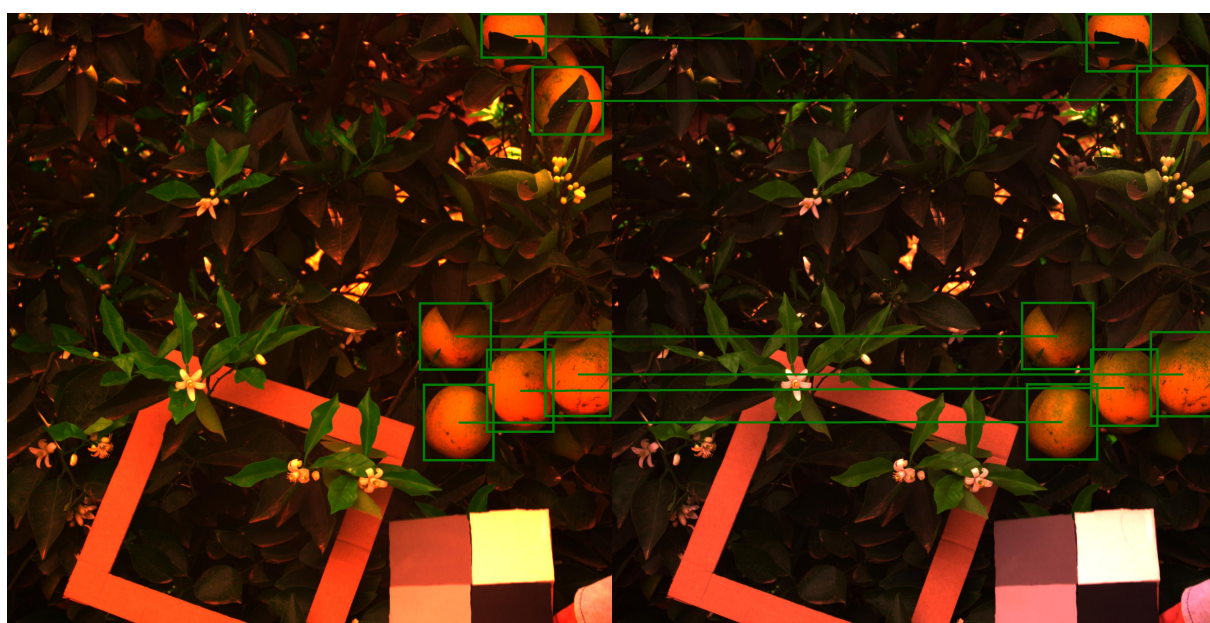
Em relação aos tempos de execução utilizando a placa NVIDIA Jetson Nano, observou-se que, com o uso da GPU, é possível executar os modelos. Por outro lado, sem o uso da GPU, a execução torna-se inviável nessa plataforma. Além disso, considerando um tempo de aquisição em campo de 3 segundos, todas as arquiteturas são passíveis de execução com FP16. No entanto, ao utilizar FP32, apenas a arquitetura YOLOv9-e para segmentação não atenderia ao tempo limite. Dessa forma, os resultados demonstram-se satisfatórios e apresentam indícios promissores de que a execução em tempo real em campo é viável, especialmente com o uso de FP16.

4.4 Experimento 3: Busca de objetos semelhantes

Conforme o experimento detalhado anteriormente na Seção 3.4, nesta seção foi avaliada a possibilidade de utilizar os resultados obtidos até então para a busca de objetos semelhantes em diferentes bandas e combinações, sendo essa uma possibilidade de aplicação na agricultura. Para isso, foi empregada a busca com base na estrutura QuadTree, conforme descrito na Seção 2.7.

Na cena apresentada na Figura 37, observa-se o primeiro resultado obtido a partir do método baseado em QuadTree para a busca de objetos semelhantes, o qual é realizado unicamente para validar o funcionamento da metodologia proposta. Neste caso, foi utilizada a rede pré-treinada YOLOv8-x para realizar a segmentação e a construção da estrutura. Verifica-se que, em ambas as composições coloridas, foram segmentadas 6 laranjas, e a busca por objetos correspondentes se mostrou eficaz, uma vez que 100% dos objetos segmentados corretamente foram encontrados na presente análise.

Figura 37 – Busca de pares de laranjas correspondentes para a primeira cena analisada utilizando a YOLOv8-x para as composições coloridas formadas utilizando as subimagens Q4 e Q6.

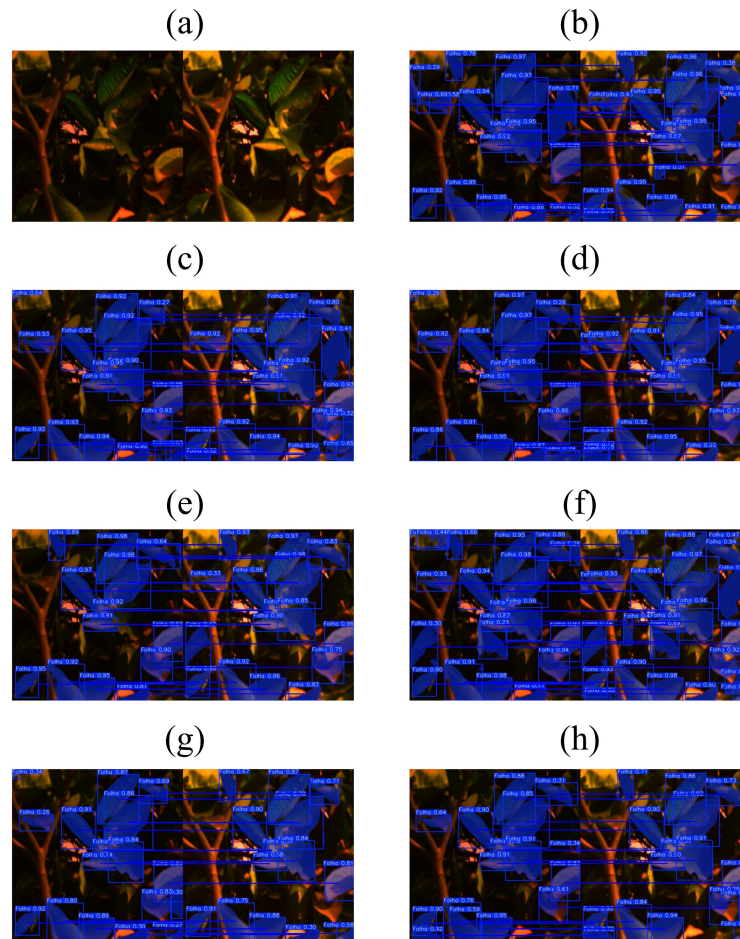


Fonte: Autor.

O resultado para a busca de objetos correspondentes é apresentado na Figura 38. Para essa análise, foram utilizados recortes dos Q4 (composição colorida utilizando 430, 550 e 650 nm) e Q6 (composição colorida utilizando 450, 560 e 685 nm), realizando-se a segmentação em ambos e, posteriormente, a busca de objetos correspondentes com os modelos treinados discutidos na seção 4.2. O primeiro aspecto importante é que a metodologia baseada no QuadTree se mostra robusta o suficiente para realizar a busca de objetos correspondentes. Quando a feição não existe em um das subimagens, não ocorre confusão com outras feições, mesmo que estas sejam próximas. Dessa forma, verifica-se que a qualidade do resultado está diretamente correlacionada com a qualidade da segmentação. As métricas de similaridade obtidas, apresentadas na Figura 38, são detalhadas na Tabela 19.

A Tabela 19 apresenta os resultados para a segunda imagem analisada. A média de acurácia varia entre 91,44% (YOLOv8-n) e 95,32% (YOLOv8-l), com a mediana seguindo

Figura 38 – Resultado da correspondência de folhas a partir da segmentação aplicando (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

Tabela 19 – Estatísticas da métrica de acurácia da busca de objetos semelhantes para a segunda cena analisada para cada uma das arquiteturas treinadas para as bandas Q4 e Q6.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão	Pares de correspondência
YOLOv8-n	91,44	96,15	98,63	11,84	16
YOLOv8-s	93,99	96,39	98,33	7,95	15
YOLOv8-m	92,13	95,34	98,39	8,89	14
YOLOv8-l	95,32	96,17	99,02	4,06	13
YOLOv8-x	94,06	95,81	98,90	5,56	16
YOLOv9-e	93,67	93,45	98,28	3,59	13
YOLOv9-c	93,22	94,36	97,06	2,96	13

Fonte: Autor.

a mesma tendência, indicando que o YOLOv8-l foi a arquitetura que obteve o melhor desempenho. As arquiteturas YOLOv9 apresentaram resultados médios ligeiramente

inferiores às variantes YOLOv8, com destaque para o YOLOv9-e, que obteve a maior média (93,67%) entre as versões dessa família. Esse comportamento reflete a robustez do YOLOv8-l em comparação com as demais, tanto em *precision* média quanto em consistência dos resultados (mediana). O percentil 95 confirma a tendência observada na média e na mediana. O YOLOv8-l alcançou o maior valor (99,02%), seguido de perto pelo YOLOv8-x (98,90%) e pelo YOLOv8-n (98,63%). As arquiteturas YOLOv9, por sua vez, apresentaram percentis ligeiramente inferiores, sugerindo que, em cenários mais desafiadores, a capacidade de atingir altos níveis de acurácia é menos frequente.

Ao analisar o desvio-padrão, nota-se que a YOLOv9-c foi o modelo mais consistente (desvio-padrão de 2,96%), seguido pelo YOLOv9e (3,59%). Já no grupo YOLOv8, o YOLOv8-l também se destacou com um baixo desvio-padrão (4,06%), confirmando sua estabilidade em diferentes contextos de análise. A quantidade de Pares de correspondência varia entre 13 e 16, com as versões YOLOv8-n e YOLOv8-x liderando com 16 ocorrências cada. A redução na quantidade de Pares de correspondência em alguns modelos, como o YOLOv8-l e o YOLOv9, pode indicar que essas arquiteturas estão priorizando a *precision* em detrimento da quantidade.

Ao analisar a Figura 39 e a Tabela 20, que se referem a uma cena mais complexa do que a segunda imagem analisada (apresentada na Figura 38 e na Tabela 19), observa-se que as médias caíram significativamente, variando entre 71,72% e 80,55%. Dentre as arquiteturas, a YOLOv9-c foi a que apresentou a melhor média (80,55%) e a mediana mais alta (80,48%), enquanto a YOLOv8-x apresentou o pior resultado tanto na média quanto na mediana. Esse comportamento reflete a maior complexidade da cena, que impactou negativamente todas as arquiteturas.

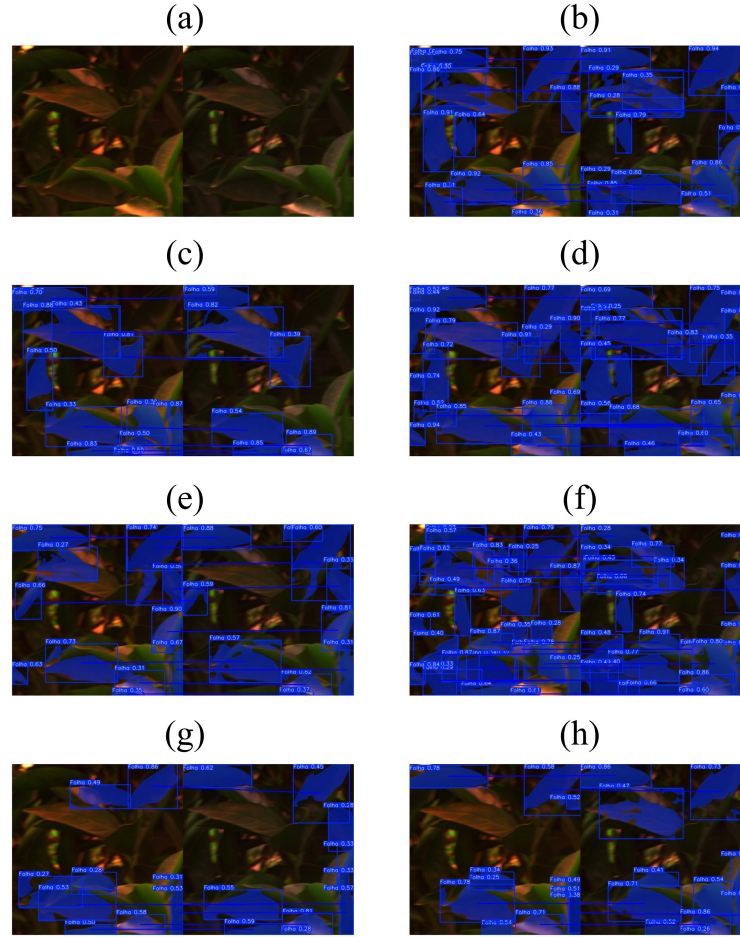
Tabela 20 – Estatísticas da métrica de acurácia da busca de objetos semelhantes para a segunda cena analisada para cada uma das arquiteturas treinadas para as bandas Q4 e Q6.

Arquitetura	Média	Mediana	Percentil 95	Desvio-padrão	Pares de correspondência
YOLOv8-n	77,82	78,13	89,38	7,34	7
YOLOv8-s	73,27	74,54	83,33	9,46	7
YOLOv8-m	80,08	79,89	91,79	8,37	10
YOLOv8-l	76,10	74,95	88,33	10,29	10
YOLOv8-x	71,72	72,64	85,35	10,39	14
YOLOv9-e	73,85	79,20	83,49	11,74	6
YOLOv9-c	80,55	80,48	83,15	2,05	5

Fonte: Autor.

Os valores do percentil 95 caíram, com o YOLOv8-m atingindo o maior valor (91,79%), seguido pelo YOLOv8-n (89,38%). Em relação à consistência, da mesma forma que ocorreu com o YOLOv9-c apresentado na Tabela 19, a consistência permaneceu como

Figura 39 – Resultado da segmentação de folhas aplicando (a) Imagem original, (b) YOLOv8-n, (c) YOLOv8-s, (d) YOLOv8-m, (e) YOLOv8-l, (f) YOLOv8-x, (g) YOLOv9-c, (h) YOLOv9-e.



Fonte: Autor.

ponto forte do YOLOv9-c (2,05%), enquanto o YOLOv9-e e o YOLOv8-l apresentaram maior variabilidade (11,74% e 10,29%, respectivamente). Ao realizar uma síntese dos resultados, nota-se que os resultados são dependentes da qualidade da segmentação realizada. Da mesma forma que foi apresentado anteriormente, as arquiteturas YOLOv8-l e YOLOv8-m se destacam como as principais candidatas para a aplicação, principalmente em termos de alta *precision* e bom desempenho geral, enquanto a YOLOv9-c é ideal para cenários em que a consistência e a estabilidade dos resultados são mais críticas do que a cobertura ou a máxima *precision*.

5 CONCLUSÕES

O presente trabalho analisou a viabilidade do uso de *Deep Learning* em Computadores de Placa Única em aplicações de agricultura, com foco na análise da NVIDIA Jetson Nano. Foi apresentado um método semiautomático para a construção dos conjuntos de dados utilizados ao longo do estudo. Esse método consiste na utilização do modelo SAM (*Segment Anything Model*) para a criação dos segmentos iniciais durante a rotulação dos dados, seguida de uma correção manual, se necessária, dos segmentos. O método permitiu a criação de conjuntos de dados de forma mais ágil. Além disso, contribuiu para a minimização de erros humanos e reduziu o tempo necessário para criar um conjunto de dados suficientemente grande para o treinamento de métodos de *Deep Learning*.

Ao longo dos testes realizados, verificou-se que utilizar apenas uma banda reduz a qualidade dos resultados, uma vez que a presença de cor afeta a confiança das previsões, podendo ser confundida com outras feições, ou até mesmo não detectada. Dessa forma, optou-se por trabalhar com composições coloridas, utilizando as bandas da subimagem 6 (Q6) ou 4 (Q4) da câmara Agrowing, que apresentaram os melhores resultados ao utilizar as redes pré-treinadas. O contraste e a informação de cor influenciam diretamente na qualidade dos resultados, evidenciando a necessidade de tratamento das imagens.

Com base nos resultados das redes pré-treinadas, validou-se a viabilidade de utilizar os pesos pré-treinados com o conjunto de dados MS COCO, implementando, assim, um processo de *Transfer Learning* e *Fine Tuning*, devido à semelhança das classes. Em relação ao treinamento das arquiteturas, observou-se que essas, comumente, apresentam resultados superiores a 80% de *precision* durante os testes. Como consequência, os modelos realizam um número menor de classificações como folhas, mas a maioria dessas classificações é correta.

Em relação às arquiteturas, observa-se que, embora em alguns casos específicos as arquiteturas mais pesadas se destaquem (YOLOv8-x e YOLOv9-c), comumente os melhores resultados, com maior consistência, são obtidos pelas arquiteturas intermediárias (YOLOv8-s e YOLOv8-m). Isso sugere que a complexidade e a quantidade de parâmetros presentes nos modelos mais pesados são excessivas para o conjunto de dados utilizado. Tais conclusões são corroboradas tanto pelas métricas de avaliação quanto pelo nível de confiança dos resultados obtidos pelas redes treinadas para a tarefa de segmentação de folhas.

No que diz respeito à correspondência, o desempenho é diretamente proporcional à qualidade da segmentação obtida, uma vez que as feições extraídas a partir dessa segmentação são utilizadas na busca por objetos semelhantes, por meio da estrutura

de dados do QuadTree. Dessa forma, ao se empregar um método de correspondência baseado em objetos segmentados, observa-se que as arquiteturas que obtêm os melhores resultados na segmentação também se destacam nessa etapa. Com isso, o método baseado em QuadTree revela-se adequado para a atividade proposta, sendo seu desempenho diretamente influenciado pela quantidade e pela qualidade das feições fornecidas como entrada.

Em relação ao tempo de execução, observou-se que a utilização de uma placa de vídeo (GPU) é fundamental, uma vez que os tempos de execução das arquiteturas na NVIDIA Jetson Nano são comparáveis aos obtidos no Google Colab sem o uso de GPU. No entanto, a Jetson Nano apresenta um custo significativamente reduzido, o que torna essa solução satisfatória para aplicações em campo. Verificou-se ainda uma redução de até 13,07% no tempo de execução ao se utilizar FP16 em vez do padrão FP32, comumente adotado pelas bibliotecas. Contudo, essa redução pode ser mínima dependendo da complexidade do modelo, resultando em um ganho modesto. Em contrapartida, com a utilização do TensorRT, observa-se uma redução substancial no tempo de inferência, sendo que, quanto menor a arquitetura, maior é o ganho obtido, podendo alcançar até 62,50% de redução. Considerando um tempo de aquisição em campo de 3 segundos, todas as arquiteturas são viáveis com FP16, enquanto, com FP32, apenas a YOLOv9-e para segmentação não atenderia ao tempo limite ao ser executada na NVIDIA Jetson Nano. Dessa forma, os resultados obtidos são satisfatórios e apresentam indícios promissores para a viabilidade de execução em tempo real em campo, especialmente com o uso de FP16 e a otimização por TensorRT. Com base nos experimentos realizados, conclui-se que a NVIDIA Jetson Nano é uma solução viável para aplicações em agricultura, demonstrando capacidade de executar inferências eficientes com um tempo de resposta aceitável. Para aplicações em tempo real é preciso considerar também o tempo de aquisição e pré-processamento das imagens. Com isso, este trabalho alcançou seu objetivo ao apresentar uma metodologia que integra a segmentação de feições com arquiteturas de *Deep Learning*, como a YOLOv8 e a YOLOv9, para segmentação e correspondência de imagens baseada em objetos segmentados, em computadores de placa única.

Para trabalhos futuros, recomenda-se a exploração de outras arquiteturas, como YOLOv10, YOLOv11 e SAM-2 (*Segment Anything Model 2*), que, no momento da redação deste trabalho, representam o estado da arte. Além disso, sugere-se a realização de comparações com outras arquiteturas, com o objetivo de refinar os resultados obtidos. Recomenda-se também a execução de testes em outros Computadores de Placa Única, como a NVIDIA Jetson TX2 e Jetson Xavier NX, bem como em plataformas como Raspberry Pi e Banana Pi, apesar de estas não possuírem placa de vídeo.

Referências

- Agrowing. **ALPHA 7Rxx Sextuple**. 2024. <<https://agrowing.com/products/alpha-7riv-sextuple/>>. Acessado em 09 de março de 2024. Citado 2 vezes nas páginas 47 e 50.
- AHMED, S.; NAVEED, H. **OpTorch: Optimized deep learning architectures for resource limited environments**. 2021. Citado 2 vezes nas páginas 32 e 33.
- AKHTAR, M. N.; SHAIKH, A. J.; KHAN, A.; AWAIS, H.; BAKAR, E. A.; OTHMAN, A. R. Smart sensing with edge computing in precision agriculture for soil assessment and heavy metal monitoring: A review. **Agriculture**, MDPI, v. 11, n. 6, p. 475, 2021. Citado na página 19.
- ANAGNOSTOPOULOS, K. Training e acceleration of deep reinforcement learning agents. 2022. Citado 2 vezes nas páginas 32 e 33.
- ARM Developer. **Cortex-A5**. 2009. <<https://developer.arm.com/Processors/Cortex-A5>>. Acessado em 11 de março de 2024. Citado na página 49.
- ASSUNÇÃO, E.; GASPAR, P. D.; MESQUITA, R.; SIMÕES, M. P.; ALIBABAEI, K.; VEIROS, A.; PROENÇA, H. Real-time weed control application using a jetson nano edge device and a spray mechanism. **Remote Sensing**, MDPI, v. 14, n. 17, p. 4217, 2022. Citado 3 vezes nas páginas 19, 48 e 49.
- AZGHADI, M. R.; OLSEN, A.; WOOD, J.; SALEH, A.; CALVERT, B.; GRANSHAW, T.; FILLOLS, E.; PHILIPPA, B. Precision robotic spot-spraying: Reducing herbicide use and enhancing environmental outcomes in sugarcane. **Computers and Electronics in Agriculture**, Elsevier, v. 235, p. 110365, 2025. Citado na página 18.
- BADIGER, D. J. A. M. M. A survey on low power sbcs for optimized implementation of deep learning models. **Journal of Survey in Fisheries Sciences**, v. 10, n. 2S, p. 4173–4189, 2023. Citado na página 19.
- BAJRAKTARI, A.; TOYLAN, H. Autonomous agricultural robot using yolov8 and bytetrack for weed detection and destruction. **Machines**, MDPI, v. 13, n. 3, p. 219, 2025. Citado na página 18.
- BEASLEY, A. E.; CLARKE, C. T.; WATSON, R. J. An opengl compliant hardware implementation of a graphic processing unit using field programmable gate array–system on chip technology. **ACM Transactions on Reconfigurable Technology and Systems (TRETS)**, ACM New York, NY, USA, v. 14, n. 1, p. 1–24, 2020. Citado na página 23.
- BIGLARI, A.; TANG, W. A review of embedded machine learning based on hardware, application, and sensing scheme. **Sensors**, MDPI, v. 23, n. 4, p. 2131, 2023. Citado na página 19.
- BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. **arXiv preprint arXiv:2004.10934**, 2020. Citado 2 vezes nas páginas 37 e 38.

- BUCK, I. Gpu computing: Programming a massively parallel processor. In: IEEE. **International Symposium on Code Generation and Optimization (CGO'07)**. [S.l.], 2007. p. 17–17. Citado na página 25.
- BUSLAEV, A.; IGLOVIKOV, V. I.; KHVEDCHENYA, E.; PARINOV, A.; DRUZHININ, M.; KALININ, A. A. Albumentations: Fast and flexible image augmentations. **Information**, v. 11, n. 2, 2020. ISSN 2078-2489. Disponível em: <<https://www.mdpi.com/2078-2489/11/2/125>>. Citado na página 43.
- CAMATA, J. J.; COUTINHO, A. L. Parallel linear octree meshing with immersed surfaces. In: IEEE. **2010 22nd International Symposium on Computer Architecture and High Performance Computing**. [S.l.], 2010. p. 151–158. Citado na página 45.
- CHANG, H.; HARI, A.; MUKHERJEE, S.; LAKSHMAN, T. Bringing the cloud to the edge. In: IEEE. **2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. [S.l.], 2014. p. 346–351. Citado na página 26.
- CHEN, Z.; CEN, J.; XIONG, J. Rolling bearing fault diagnosis using time-frequency analysis and deep transfer convolutional neural network. **Ieee Access**, IEEE, v. 8, p. 150248–150261, 2020. Citado 2 vezes nas páginas 44 e 45.
- COELHO, R. A. **Detetores de mudança de conceito por meio do mapeamento espacial do fluxo de dados usando quadtree**. Tese (Doutorado) — Universidade Federal de Minas Gerais, 2022. Citado 2 vezes nas páginas 45 e 46.
- DUTOT, M.; NELSON, L.; TYSON, R. Predicting the spread of postharvest disease in stored fruit, with application to apples. **Postharvest biology and technology**, Elsevier, v. 85, p. 45–56, 2013. Citado na página 22.
- EBRAHIMI, M.; KHOSHTAGHAZA, M. H.; MINAEI, S.; JAMSHIDI, B. Vision-based pest detection based on svm classification method. **Computers and Electronics in Agriculture**, Elsevier, v. 137, p. 52–58, 2017. Citado 2 vezes nas páginas 18 e 22.
- FLORENCIO, F. de A. **Estudo Comparativo do Desempenho de Bibliotecas para Redes Neurais Convolucionais em Diferentes Microarquiteturas de GPU**. Dissertação (Mestrado) — Universidade Federal de Sergipe, 2020. Citado na página 25.
- FLYNN, M. J.; LUK, W. **Computer system design: system-on-chip**. [S.l.]: John Wiley e Sons, 2011. Citado 2 vezes nas páginas 23 e 24.
- FOUNTAS, S.; ESPEJO-GARCIA, B.; KASIMATI, A.; MYLONAS, N.; DARRA, N. The future of digital agriculture: technologies and opportunities. **IT professional**, IEEE, v. 22, n. 1, p. 24–28, 2020. Citado na página 22.
- FREITAS, T. A. **Implementação de sistema para sonificação de microscopia de varredura por sonda**. Dissertação (Mestrado) — Universidade Federal de Minas Gerais, 2018. Citado na página 23.
- GARLAND, M.; GRAND, S. L.; NICKOLLS, J.; ANDERSON, J.; HARDWICK, J.; MORTON, S.; PHILLIPS, E.; ZHANG, Y.; VOLKOV, V. Parallel computing experiences with cuda. **IEEE micro**, IEEE, v. 28, n. 4, p. 13–27, 2008. Citado na página 26.
- GARRETT, J. D. garrettj403/SciencePlots. Zenodo, set. 2021. Disponível em: <<http://doi.org/10.5281/zenodo.4106649>>. Citado na página 52.

- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning. Adaptive computation and machine learning**. [S.l.]: The MIT Press, 2016. Citado 2 vezes nas páginas 28 e 30.
- GRZESIK, P.; MROZEK, D. Combining machine learning and edge computing: Opportunities, challenges, platforms, frameworks, and use cases. **Electronics**, MDPI, v. 13, n. 3, p. 640, 2024. Citado na página 19.
- HARRIS, C. R.; MILLMAN, K. J.; WALT, S. J. van der; GOMMERS, R.; VIRTANEN, P.; COUNAPEAU, D.; WIESER, E.; TAYLOR, J.; BERG, S.; SMITH, N. J.; KERN, R.; PICUS, M.; HOYER, S.; KERKWIJK, M. H. van; BRETT, M.; HALDANE, A.; RÍO, J. F. del; WIEBE, M.; PETERSON, P.; GÉRARD-MARCHANT, P.; SHEPPARD, K.; REDDY, T.; WECKESSER, W.; ABBASI, H.; GOHLKE, C.; OLIPHANT, T. E. Array programming with NumPy. **Nature**, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <<https://doi.org/10.1038/s41586-020-2649-2>>. Citado na página 52.
- HE, K.; ZHANG, X.; REN, S.; SUN, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 37, n. 9, p. 1904–1916, 2015. Citado na página 39.
- HEBB, D. O. The organization of behavior. **New York**, 1949. Citado na página 30.
- HENNESSY, J. L.; PATTERSON, D. A. **Computer architecture: a quantitative approach**. [S.l.]: Elsevier, 2011. Citado na página 26.
- HUANGFU, Z.; LI, S. Lightweight you only look once v8: An upgraded you only look once v8 algorithm for small object identification in unmanned aerial vehicle images. **Applied Sciences**, MDPI, v. 13, n. 22, p. 12369, 2023. Citado 2 vezes nas páginas 39 e 40.
- HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science e Engineering**, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. Citado na página 52.
- IMAMBI, S.; PRAKASH, K. B.; KANAGACHIDAMBARESAN, G. Pytorch. **Programming with TensorFlow: Solution for Edge Computing Applications**, Springer, p. 87–104, 2021. Citado na página 49.
- JIA, Y.; SHELHAMER, E.; DONAHUE, J.; KARAYEV, S.; LONG, J.; GIRSHICK, R.; GUADARRAMA, S.; DARRELL, T. Caffe: Convolutional architecture for fast feature embedding. **arXiv preprint arXiv:1408.5093**, 2014. Citado na página 49.
- JIANG, P.; ERGU, D.; LIU, F.; CAI, Y.; MA, B. A review of yolo algorithm developments. **Procedia Computer Science**, v. 199, p. 1066–1073, 2022. ISSN 1877-0509. The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 2021): Developing Global Digital Economy after COVID-19. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S1877050922001363>>. Citado na página 34.
- JOCHER, G.; CHAURASIA, A.; QIU, J. **Ultralytics YOLOv8**. 2023. Disponível em: <<https://github.com/ultralytics/ultralytics>>. Citado 3 vezes nas páginas 39, 40 e 41.

JOCHER, G.; CHAURASIA, A.; STOKEN, A.; BOROVEC, J.; NANOCODE012; KWON, Y.; MICHAEL, K.; TAOXIE; FANG, J.; IMYHXY; LORNA; YIFU), WONG, C.; V, A.; MONTES, D.; WANG, Z.; FATI, C.; NADAR, J.; LAUGHING; UNGLVKITDE; SONCK, V.; TKIANAI; YXNONG; SKALSKI, P.; HOGAN, A.; NAIR, D.; STROBEL, M.; JAIN, M. **ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation**. Zenodo, 2022. Disponível em: <<https://doi.org/10.5281/zenodo.7347926>>. Citado 2 vezes nas páginas 38 e 39.

KINGER, S.; KULKARNI, V. Explainable ai for deep learning based disease detection. In: **Proceedings of the 2021 Thirteenth International Conference on Contemporary Computing**. [S.l.: s.n.], 2021. p. 209–216. Citado na página 53.

KIRILLOV, A.; MINTUN, E.; RAVI, N.; MAO, H.; ROLLAND, C.; GUSTAFSON, L.; XIAO, T.; WHITEHEAD, S.; BERG, A. C.; LO, W.-Y. *et al.* Segment anything. **arXiv preprint arXiv:2304.02643**, 2023. Citado 3 vezes nas páginas 53, 54 e 69.

KULKARNI, K. K. **Machine Learning and Vision Using Edge Devices for Multimodal Chatbots and Bio-meteorological Sensing**. Tese (Doutorado) — Arizona State University, 2021. Citado 2 vezes nas páginas 48 e 49.

LATIF, S.; DRISS, M.; BOULILA, W.; HUMA, Z. E.; JAMAL, S. S.; IDREES, Z.; AHMAD, J. Deep learning for the industrial internet of things (iiot): A comprehensive survey of techniques, implementation frameworks, potential applications, and future directions. **Sensors**, MDPI, v. 21, n. 22, p. 7518, 2021. Citado na página 49.

LEWIS, G.; ECHEVERRÍA, S.; SIMANTA, S.; BRADSHAW, B.; ROOT, J. Tactical cloudlets: Moving cloud computing to the edge. In: IEEE. **2014 IEEE Military Communications Conference**. [S.l.], 2014. p. 1440–1446. Citado na página 27.

LI, C.; LI, L.; JIANG, H.; WENG, K.; GENG, Y.; LI, L.; KE, Z.; LI, Q.; CHENG, M.; NIE, W. *et al.* Yolov6: A single-stage object detection framework for industrial applications. **arXiv preprint arXiv:2209.02976**, 2022. Citado na página 38.

LI, J.; GÜLDENRING, R.; NALPANTIDIS, L. Real-time joint-stem prediction for agricultural robots in grasslands using multi-task learning. **Agronomy**, MDPI, v. 13, n. 9, p. 2365, 2023. Citado 2 vezes nas páginas 20 e 33.

LIN, T.-Y.; DOLLÁR, P.; GIRSHICK, R.; HE, K.; HARIHARAN, B.; BELONGIE, S. Feature pyramid networks for object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 2117–2125. Citado 2 vezes nas páginas 36 e 37.

LIN, T.-Y.; DOLLÁR, P.; GIRSHICK, R.; HE, K.; HARIHARAN, B.; BELONGIE, S. Feature pyramid networks for object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 2117–2125. Citado na página 38.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. In: SPRINGER. **Computer Vision—ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13**. [S.l.], 2014. p. 740–755. Citado 4 vezes nas páginas 7, 9, 53 e 54.

- LINDHOLM, E.; NICKOLLS, J.; OBERMAN, S.; MONTRYM, J. Nvidia tesla: A unified graphics and computing architecture. **IEEE micro**, IEEE, v. 28, n. 2, p. 39–55, 2008. Citado na página 25.
- LIU, B.; ZHANG, Y.; HE, D.; LI, Y. Identification of apple leaf diseases based on deep convolutional neural networks. **Symmetry**, MDPI, v. 10, n. 1, p. 11, 2017. Citado 2 vezes nas páginas 18 e 22.
- MARAVEAS, C. Incorporating artificial intelligence technology in smart greenhouses: Current state of the art. **Applied Sciences**, MDPI, v. 13, n. 1, p. 14, 2022. Citado na página 19.
- MCKINNEY Wes. Data Structures for Statistical Computing in Python. In: WALT Stéfan van der; MILLMAN Jarrod (Ed.). **Proceedings of the 9th Python in Science Conference**. [S.l.: s.n.], 2010. p. 56 – 61. Citado na página 52.
- MEHTA, D. P.; SAHNI, S. **Handbook of data structures and applications**. [S.l.]: Chapman and Hall/CRC, 2004. Citado na página 45.
- MICIKEVICIUS, P.; NARANG, S.; ALBEN, J.; DIAMOS, G.; ELSEIN, E.; GARCIA, D.; GINSBURG, B.; HOUSTON, M.; KUCHAIEV, O.; VENKATESH, G. *et al.* Mixed precision training. **arXiv preprint arXiv:1710.03740**, 2017. Citado 3 vezes nas páginas 31, 32 e 33.
- MISRA, D. Mish: A self regularized non-monotonic activation function. **arXiv preprint arXiv:1908.08681**, 2019. Citado na página 38.
- MOREIRA, C. L. **Método de otimização da alocação de redes virtuais na estrutura física de uma rede substrato utilizando aprendizado por reforço**. Dissertação (Mestrado) — Centro Universitário FEI, 2021. Citado na página 26.
- MOURARIAS, M. N. **Ferramenta para recomendação de tecnologias 4.0: um caso de aplicação em uma produção de queijo artesanal**. Dissertação (Mestrado) — Universidade de São Paulo, 2021. Citado na página 22.
- OWENS, J. D.; HOUSTON, M.; LUEBKE, D.; GREEN, S.; STONE, J. E.; PHILLIPS, J. C. Gpu computing. **Proceedings of the IEEE**, IEEE, v. 96, n. 5, p. 879–899, 2008. Citado na página 25.
- PAJANKAR, A. Raspberry pi supercomputing and scientific programming. **Ashwin Pajankar**, Springer, 2017. Citado na página 23.
- PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. Citado na página 52.
- PONTI, M. A.; COSTA, G. B. P. D. Como funciona o deep learning. **arXiv preprint arXiv:1806.07908**, 2018. Citado 2 vezes nas páginas 29 e 30.
- QUADROS, A. E. R. **Processamento Paralelo em CUDA Aplicado ao Modelo de Geração de cenários sintéticos de vazões e Energias-GEVAZP**. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, 2016. Citado na página 26.

- REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 779–788. Citado 2 vezes nas páginas 34 e 35.
- REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2017. p. 7263–7271. Citado na página 38.
- REDMON, J.; FARHADI, A. Yolo3: An incremental improvement. **arXiv preprint arXiv:1804.02767**, 2018. Citado na página 38.
- REUTHER, A.; MICHALEAS, P.; JONES, M.; GADEPALLY, V.; SAMSI, S.; KEPNER, J. Survey and benchmarking of machine learning accelerators. In: IEEE. **2019 IEEE high performance extreme computing conference (HPEC)**. [S.l.], 2019. p. 1–9. Citado na página 26.
- REZATOFIGHI, H.; TSOI, N.; GWAK, J.; SADEGHIAN, A.; REID, I.; SAVARESE, S. Generalized intersection over union: A metric and a loss for bounding box regression. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [S.l.: s.n.], 2019. p. 658–666. Citado na página 35.
- RUBACK, L.; AVILA, S.; CANTERO, L. Vieses no aprendizado de máquina e suas implicações sociais: Um estudo de caso no reconhecimento facial. In: . [S.l.: s.n.], 2021. p. 90–101. Citado na página 56.
- RUBBENS, P.; BRODIE, S.; CORDIER, T.; DESTRO BARCELLOS, D.; DEVOS, P.; FERNANDES-SALVADOR, J. A.; FINCHAM, J. I.; GOMES, A.; HANDEGARD, N. O.; HOWELL, K.; JAMET, C.; KARTVEIT, K. H.; MOUSTAHFID, H.; PARCERISAS, C.; POLITIKOS, D.; SAUZÈDE, R.; SOKOLOVA, M.; UUSITALO, L.; VAN DEN BULCKE, L.; VAN HELMOND, A. T. M.; WATSON, J. T.; WELCH, H.; BELTRAN-PEREZ, O.; CHAFFRON, S.; GREENBERG, D. S.; KÜHN, B.; KIKO, R.; LO, M.; LOPES, R. M.; MÖLLER, K. O.; MICHAELS, W.; PALA, A.; ROMAGNAN, J.-B.; SCHUCHERT, P.; SEYDI, V.; VILLASANTE, S.; MALDE, K.; IRISSON, J.-O. Machine learning in marine ecology: an overview of techniques and applications. **ICES Journal of Marine Science**, v. 80, n. 7, p. 1829–1853, 08 2023. ISSN 1054-3139. Disponível em: <<https://doi.org/10.1093/icesjms/fsad100>>. Citado na página 28.
- SAMET, H. The quadtree and related hierarchical data structures. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 16, n. 2, p. 187–260, 1984. Citado na página 45.
- SAMPURNO, R. M.; LIU, Z.; ABEYRATHNA, R. R. D.; AHAMED, T. Intrarow uncut weed detection using you-only-look-once instance segmentation for orchard plantations. **Sensors**, MDPI, v. 24, n. 3, p. 893, 2024. Citado na página 19.
- SHEK, S. C. K.; BUTTERFIELD, J.; SPENCE, I.; MURPHY, A. Single board computing for image recognition on an autonomous drone. In: IEEE. **2023 IEEE Industrial Electronics and Applications Conference (IEACon)**. [S.l.], 2023. p. 24–29. Citado na página 19.

- SHEN, F.; GAN, R.; ZENG, G. Weighted residuals for very deep networks. In: IEEE. **2016 3rd international conference on systems and informatics (Icsai)**. [S.l.], 2016. p. 936–941. Citado na página 38.
- SILVA, I. N. da; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais Para Engenharia e Ciências Aplicadas: Fundamentos Teóricos e Aspectos Práticos**. 2. ed. São Paulo: Artliber Editora, 2016. Universidade de São Paulo. Citado 2 vezes nas páginas 27 e 30.
- SINGH, S. Optimize cloud computations using edge computing. In: IEEE. **2017 International Conference on Big Data, IoT and Data Science (BIG-IOT)**. [S.l.], 2017. p. 49–53. Citado na página 19.
- TANWAR, S.; TANWAR; JAMES. **Fog computing for Healthcare 4.0 environments**. [S.l.]: Springer, 2021. Citado na página 26.
- TEAM, T. pandas development. **pandas-dev/pandas: Pandas**. Zenodo, 2020. Disponível em: <<https://doi.org/10.5281/zenodo.3509134>>. Citado na página 52.
- TENSORFLOW, D. Tensorflow. **Site oficial**, Springer, 2018. Citado na página 49.
- TERVEN, J.; CÓRDOVA-ESPARZA, D.-M.; ROMERO-GONZÁLEZ, J.-A. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. **Machine Learning and Knowledge Extraction**, MDPI, v. 5, n. 4, p. 1680–1716, 2023. Citado 5 vezes nas páginas 33, 34, 36, 37 e 39.
- TRINH, D. C.; MAC, A. T.; DANG, K. G.; NGUYEN, H. T.; NGUYEN, H. T.; BUI, T. D. Alpha-eiou-yolov8: An improved algorithm for rice leaf disease detection. **AgriEngineering**, MDPI, v. 6, n. 1, p. 302–317, 2024. Citado na página 20.
- WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. **arXiv preprint arXiv:2207.02696**, 2022. Citado 2 vezes nas páginas 38 e 39.
- WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2023. p. 7464–7475. Citado 2 vezes nas páginas 39 e 40.
- WANG, C.-Y.; YEH, I.-H.; LIAO, H.-Y. M. Yolov9: Learning what you want to learn using programmable gradient information. **arXiv preprint arXiv:2402.13616**, 2024. Citado 3 vezes nas páginas 41, 42 e 43.
- WU, H.; ZHANG, J.; HUANG, K.; LIANG, K.; YU, Y. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. **arXiv preprint arXiv:1903.11816**, 2019. Citado na página 36.
- YE, Z.; GAO, W.; HU, Q.; SUN, P.; WANG, X.; LUO, Y.; ZHANG, T.; WEN, Y. Deep learning workload scheduling in gpu datacenters: A survey. **ACM Computing Surveys**, ACM New York, NY, v. 56, n. 6, p. 1–38, 2024. Citado na página 26.
- YU, F.; XIU, X.; LI, Y. A survey on deep transfer learning and beyond. **Mathematics**, MDPI, v. 10, n. 19, p. 3619, 2022. Citado na página 44.

ZEILER, M. D.; FERGUS, R. **Visualizing and Understanding Convolutional Networks**. 2013. Disponível em: <<https://arxiv.org/abs/1311.2901>>. Citado na página 53.

ZHAI, Z.; MARTÍNEZ, J. F.; BELTRAN, V.; MARTÍNEZ, N. L. Decision support systems for agriculture 4.0: Survey and challenges. **Computers and Electronics in Agriculture**, Elsevier, v. 170, p. 105256, 2020. Citado na página 22.

ZHANG, Y.-F.; REN, W.; ZHANG, Z.; JIA, Z.; WANG, L.; TAN, T. Focal and efficient iou loss for accurate bounding box regression. **Neurocomputing**, Elsevier, v. 506, p. 146–157, 2022. Citado na página 20.

ZHANG, Z.; HE, T.; ZHANG, H.; ZHANG, Z.; XIE, J.; LI, M. Bag of freebies for training object detection neural networks. **arXiv preprint arXiv:1902.04103**, 2019. Citado na página 38.