



UNIVERSIDADE ESTADUAL PAULISTA
“JÚLIO DE MESQUITA FILHO”
Campus de São José do Rio Preto

Tulio Henry das Neves Miranda

Evolução e Desempenho de Algoritmos de Detecção de Objetos em Ambientes Dinâmicos

São José do Rio Preto
2025

Tulio Henry das Neves Miranda

Evolução e Desempenho de Algoritmos de Detecção de Objetos em Ambientes Dinâmicos

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciências da Computação, junto ao Conselho de Curso de Bacharelado em Ciências da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Orientador:

Prof. Dr. Wallace Correa de Oliveira Casaca

**São José do Rio Preto
2025**

Tulio Henry das Neves Miranda

Evolução e Desempenho de Algoritmos de Detecção de Objetos em Ambientes Dinâmicos

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciências da Computação, junto ao Conselho de Curso de Bacharelado em Ciências da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Comissão Examinadora:

Prof. Dr. Wallace Correa de Oliveira Casaca

Prof. Dr. Lucas Correia Ribas

Prof. Dr. Rodrigo Capobiano Guido

**São José do Rio Preto
2025**

M672e

Miranda, Tulio Henry das Neves

Evolução e desempenho de algoritmos de detecção de objetos em ambientes dinâmicos / Tulio Henry das Neves Miranda. -- São José do Rio Preto, 2025

51 p. : il., tabs., fotos

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (UNESP), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto

Orientador: Wallace Correa de Oliveira Casaca

1. Inteligência artificial. 2. Visão computacional. 3. Deep learning (Machine learning). 4. Processamento de imagens. 5. Visão por computador. I. Título.

Dedico este trabalho aos meus pais, que sempre acreditaram no meu potencial mesmo quando eu duvidava de mim. À minha família, por todo o apoio, paciência e amor incondicional durante essa jornada

Não devemos temer as máquinas que pensam,
mas sim o que os humanos escolhem fazer com elas.

-Stephen Hawking

Agradecimentos

Agradeço ao meu orientador Prof. Dr Wallace Correa de Oliveira Casaca, aos meus colegas do LIDIA, aos técnicos, à seção administrativa, aos meus amigos e à minha família.

Resumo

Este trabalho apresenta um estudo aprofundado sobre a evolução e o desempenho das arquiteturas da família YOLO (*You Only Look Once*), amplamente utilizadas em tarefas de detecção de objetos em tempo real. A pesquisa tem como objetivo compreender como as modificações estruturais introduzidas nas versões mais recentes — YOLOv8, YOLOv9, YOLOv10 e YOLOv11 — impactam a precisão, a estabilidade de treinamento, a velocidade de inferência e o custo computacional em ambientes dinâmicos, caracterizados por variação de iluminação, movimento e sobreposição de objetos. Os experimentos foram conduzidos em ambiente controlado com parâmetros idênticos de treinamento, avaliando métricas como precisão, revocação, *F1-score*, mAP@0.5 e mAP@0.5:0.95. O YOLOv8 apresentou o melhor equilíbrio entre desempenho e estabilidade (mAP@0.5:0.95 = 0,801), enquanto o YOLOv11 destacou-se pela eficiência e menor tempo de execução, mantendo alta consistência. O YOLOv9 obteve maior sensibilidade, porém com custo computacional elevado, e o YOLOv10 mostrou-se ideal para aplicações em tempo real devido à remoção do *Non-Maximum Suppression* (NMS). Conclui-se que a evolução da série YOLO reflete um movimento contínuo em direção à otimização da eficiência computacional sem perda significativa de desempenho, representando avanços importantes para aplicações em sistemas embarcados, vigilância inteligente, robótica e veículos autônomos. Este estudo reforça a relevância da análise comparativa de arquiteturas como ferramenta essencial para orientar o uso de modelos de detecção conforme as necessidades de precisão, velocidade e recursos disponíveis.

Palavras-chave: Inteligência Artificial, Aprendizagem Profunda, Detecção de Objetos, YOLO.

Abstract

This work presents an in-depth study on the evolution and performance of the YOLO (You Only Look Once) architecture family, widely employed in real-time object detection. The main goal is to understand how the structural improvements introduced in recent versions — YOLOv8, YOLOv9, YOLOv10, and YOLOv11 — affect accuracy, training stability, inference speed, and computational cost in dynamic environments, characterized by motion, lighting variation, and object overlap. Experiments were performed in a controlled environment using identical training parameters, analyzing metrics such as precision, recall, F1-score, mAP@0.5, and mAP@0.5:0.95. YOLOv8 achieved the best balance between accuracy and stability (mAP@0.5:0.95 = 0.801), while YOLOv11 stood out for its computational efficiency and shorter training time. YOLOv9 showed higher sensitivity but required more resources, and YOLOv10 proved effective for real-time scenarios with its NMS-free detection strategy. The findings indicate that the evolution of the YOLO series represents a consistent trend toward greater computational efficiency without significant loss in detection accuracy, reinforcing its suitability for embedded systems, autonomous vehicles, robotics, and intelligent surveillance applications. Moreover, the comparative analysis conducted in this study contributes to understanding how each architectural innovation impacts practical deployment and scalability in modern computer vision systems.

Keywords: Artificial Intelligence, Deep Learning, Object Detection, YOLO.

Lista de Figuras

1.1	Exemplo da aplicação de detecção de alvos com a rede YOLO [3].	2
4.1	Amostras do <i>dataset</i>	20
4.2	Arquitetura simplificada do YOLOV8 [32]	24
4.3	Arquitetura simplificada do YOLOV9 [13]	26
4.4	Arquitetura simplificada do YOLOV10 [1]	28
4.5	Arquitetura simplificada do Yolov11 [23]	30
5.1	Graficos de Precision x Recall (PR)	33
5.2	PR com 100 epochs	33
5.3	PR com 200 epochs	33
5.4	Resultados gerais do modelo à 200 epochs	34
5.5	Evolução do mAP@ por epoch	35
5.6	Evolução das Métricas do Yolov10	36
5.7	Evolução do Precisão entre versões	38
5.8	Evolução do Recall entre versões	39
5.9	Evolução do mAP entre versões	39
5.10	Custo de tempo de treinamento por época em cada versão	40

Lista de Tabelas

5.1	Dados resultantes do YOLOv8	33
5.2	Dados resultantes do YOLOv9	34
5.3	Dados resultantes do YOLOv10	35
5.4	Dados resultantes do YOLOv11	37
5.5	Comparação de desempenho entre os modelos YOLOv8 a YOLOv11 . . .	37

Sumário

1	Introdução	1
1.1	Histórico e Contextualização	1
1.2	Detecção de Objetos	2
1.3	Objetivos	5
1.4	Contribuições do Trabalho	6
2	Revisão Bibliográfica	8
2.1	Panorama e Evolução dos Detectores	8
2.2	AlexNet	9
2.3	ResNet	10
2.4	Aplicações Práticas de CNNs e da Arquitetura YOLO	10
2.5	Arquiteturas da Família YOLO	11
3	Conceitos Básicos de Deep Learning	14
3.1	R-CNN e Faster R-CNN	14
3.2	Transformers	15
3.3	Comparações Metodológicas: One-stage, Two-stage e Transformers	16
4	Materiais e Métodos	17
4.1	Descrição do Ambiente e Ferramentas	17
4.1.1	Configuração de Treinamento dos Modelos	18
4.2	Dataset Utilizado	19
4.3	Critérios de Avaliação	21
4.4	Tópicos Técnicos Recorrentes na Literatura	22

4.5	Aprofundamento nos Modelos YOLO	23
4.5.1	YOLOv8	23
4.5.2	YOLOv9	25
4.5.3	YOLOv10	27
4.5.4	YOLOv11	29
5	Resultados e Discussão	32
5.1	Experimento 1 – YOLOv8	32
5.2	Experimento 2 – YOLOv9	34
5.3	Experimento 3 – YOLOv10	35
5.4	Experimento 4 – YOLOv11	36
5.5	Análise Comparativa	37
5.6	Considerações Finais	41
6	Conclusão	42
6.1	Comparação Direta Entre as Versões	42
6.1.1	Trade-offs	44
6.2	Desafios Encontrados	45
6.3	Posição da Presente Pesquisa	46
6.4	Trabalho Futuros	46

Capítulo 1

Introdução

1.1 Histórico e Contextualização

A Inteligência Artificial (IA) é uma das áreas mais transformadoras e promissoras da Ciência da Computação, revolucionando a forma como interagimos com a tecnologia e resolvemos problemas complexos. Em termos gerais, a IA refere-se à capacidade de máquinas e sistemas de realizar tarefas que, tradicionalmente, exigiriam a inteligência humana, como aprender, tomar decisões e resolver problemas. Embora o termo IA tenha sido cunhado na década de 1950, a área experimentou uma evolução acelerada apenas nos últimos 30 anos, impulsionada por avanços em hardware, algoritmos e pela disponibilidade de grandes volumes de dados.

No dia a dia das pessoas, convencionou-se chamar de IA o conjunto de técnicas e metodologias que permitem a sistemas computacionais simular capacidades cognitivas humanas. Dentre as principais linhas da IA, destacam-se: aprendizado de máquina, redes neurais artificiais, processamento de linguagem natural e visão computacional. O aprendizado de máquina, por exemplo, permite que sistemas aprendam padrões a partir de dados, sem serem explicitamente programados. Já as redes neurais artificiais, inspiradas no funcionamento do cérebro humano, são capazes de processar informações complexas, como imagens, sons e textos.

A detecção de objetos é uma tarefa desafiadora na visão computacional, com aplicações que abrangem desde veículos autônomos e vigilância até diagnósticos médicos e

automação industrial. A capacidade de identificar e localizar objetos em imagens ou vídeos em tempo real é essencial para sistemas inteligentes, e o desenvolvimento de algoritmos eficientes para essa tarefa tem sido foco de intensa pesquisa nas últimas décadas. Nesse cenário, a rede YOLO (*You Only Look Once*) emergiu como uma das arquiteturas mais revolucionárias da área, destacando-se por sua velocidade e precisão [25].

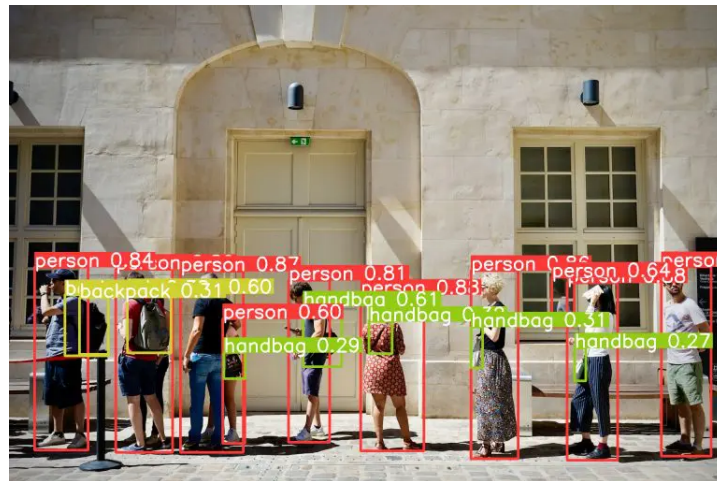


Figura 1.1: Exemplo da aplicação de detecção de alvos com a rede YOLO [3].

Desde sua primeira versão, a rede de aprendizado profundo YOLO tem passado por vários aprimoramentos, incorporando avanços em arquitetura, técnicas de aprendizado e otimização de desempenho (vide Figura 1.1 para uma ilustração). As versões mais recentes da referida rede, da YOLOv8 à YOLOv11, representam o nível mais alto em detecção de objetos, trazendo melhorias significativas em termos de acurácia, eficiência computacional e adaptabilidade para diferentes cenários. Essas versões não apenas aprimoram a arquitetura da rede em si, mas também introduzem novas funcionalidades, como a integração com *frameworks* modernos de aprendizado de máquina. [31].

1.2 Detecção de Objetos

Antes de adentrar no tecnicismo das técnicas e estratégias computacionais sobre detecção de objetos, foco deste TCC, vamos inicialmente apresentar os principais conceitos referentes às redes neurais convolucionais.

As Redes Neurais Convolucionais, ou *Convolutional Neural Networks* (CNNs), cons-

tituem uma das arquiteturas mais utilizadas dentro do campo da inteligência artificial, especialmente para aplicações que envolvem dados visuais, como imagens e vídeos. Inspiradas na estrutura do córtex visual dos seres vivos, as CNNs se destacam por sua capacidade de aprender automaticamente características relevantes dos dados de entrada, sem a necessidade de intervenção humana na extração de atributos [18].

Entre as principais vantagens das CNNs estão a capacidade de aprender automaticamente representações de dados, a eficiência no uso de parâmetros — graças ao compartilhamento de pesos — e a robustez frente a variações espaciais, como mudanças de posição e escala de objetos em uma imagem [10].

No entanto, essas redes também apresentam limitações. O treinamento de modelos convolucionais geralmente exige grandes quantidades de dados rotulados e considerável poder computacional. Além disso, as CNNs são frequentemente criticadas por sua falta de interpretabilidade, dificultando a compreensão das decisões tomadas pelo modelo — uma questão crítica em aplicações sensíveis como diagnóstico médico ou decisões judiciais [19].

Ainda assim, com o progresso tecnológico das CNNs, uma nova área foi favorecida: a identificação de objetos através da Inteligência Artificial. Isso permitiu que ela alcançasse novos níveis na comunidade científica e representasse um dos campos mais avançados da visão computacional. A sua principal função é reconhecer e localizar automaticamente objetos de interesse em imagens ou vídeos, atribuindo-lhes etiquetas e delimitando suas posições com caixas delimitadoras (*boundary boxes*). Com o progresso do aprendizado profundo (*deep learning*), particularmente com a utilização de Redes Neurais Convolucionais (CNNs), os sistemas de identificação de objetos alcançaram níveis de acurácia e rapidez anteriormente inalcançáveis por métodos convencionais. [38]

Historicamente, os primeiros modelos de detecção baseavam-se em técnicas de aprendizado de máquina com extração manual de características, como HOG (*Histogram of Oriented Gradients*) e classificadores SVM (*Support Vector Machine*). No entanto, esses métodos apresentavam limitações significativas quanto à capacidade de generalização e escalabilidade [6].

A revolução no campo ocorreu com a introdução de arquiteturas baseadas em CNNs. A detecção de objetos passou a ser tratada como uma tarefa de aprendizado profundo, em que as próprias redes aprendem as representações necessárias para identificar objetos com diferentes formas, tamanhos e posições. Modelos como o R-CNN (*Regions with CNN Features*), introduzido por Girshick, foram precursores dessa mudança [9].

Posteriormente, surgiram variações mais rápidas e eficientes, como o Fast R-CNN e o Faster R-CNN, que incorporaram redes de proposta de regiões (*Region Proposal Networks*) diretamente na arquitetura da CNN, reduzindo significativamente o tempo de inferência [9] [27].

Outra frente de pesquisa relevante é aquela representada por meio de modelos de detecção em tempo real, como o YOLO (*You Only Look Once*) e o SSD (*Single Shot MultiBox Detector*), que oferecem maior velocidade com desempenho competitivo. Redes dessa natureza realizam a detecção de objetos como uma única tarefa de regressão, tratando simultaneamente a localização e a classificação [26] [20].

Aplicações Práticas

As IAs detectoras de objetos são amplamente aplicadas em diversas áreas, tais como:

- **Segurança e Vigilância:** Sistemas de monitoramento inteligente usam detecção de objetos para identificar atividades suspeitas, pessoas ou veículos em tempo real [26].
- **Agricultura de Precisão:** Drones com visão computacional detectam pragas, frutas ou plantas doentes automaticamente, permitindo uma gestão mais eficiente das lavouras [29].
- **Carros Autônomos:** Detectores de objetos são essenciais para reconhecer pedestres, outros veículos, semáforos e placas de trânsito, garantindo segurança na navegação [5].
- **Medicina:** Utilizadas para localizar tumores, órgãos e anomalias em imagens radiológicas, auxiliando no diagnóstico assistido por IA [19].

- **Indústria e Logística:** Empregadas em sistemas de inspeção de qualidade e rastreamento de produtos em linhas de produção automatizadas [37].

Desafios e Limitações

Apesar do progresso, ainda existem desafios técnicos relevantes na detecção de objetos, como:

- **Ocultações e sobreposições:** Objetos parcialmente cobertos são mais difíceis de detectar corretamente.
- **Variações de escala e iluminação:** A mudança na dimensão dos objetos ou em condições de luz pode prejudicar a precisão.
- **Desbalanceamento de classes:** Alguns objetos aparecem com mais frequência do que outros nos dados de treinamento, o que pode afetar a capacidade da IA de detectar classes menos representadas.
- **Custo Computacional:** Modelos de última geração, como o YOLOv8 ou o DETR (*Detection Transformer*), exigem alto poder computacional tanto para treinamento quanto para inferência.

A detecção de objetos por meio de inteligência artificial tem se mostrado uma tecnologia promissora, com aplicações diretas em diversos setores da sociedade. A evolução constante dos algoritmos e o avanço do hardware têm contribuído para tornar essa tecnologia mais precisa, rápida e acessível. No entanto, sua adoção em larga escala ainda exige atenção a aspectos como robustez, ética, explicabilidade e eficiência computacional.

1.3 Objetivos

Este Trabalho de Conclusão de Curso tem como objetivo geral implementar, treinar e realizar uma análise comparativa entre as versões mais recentes da arquitetura

de detecção de objetos YOLO (*You Only Look Once*), especificamente as versões YOLOv8, YOLOv9, YOLOv10 e YOLOv11. A proposta visa examinar o desempenho, as características estruturais e a aplicabilidade de cada uma dessas versões em diferentes contextos de detecção de objetos.

A análise será conduzida com base em métricas amplamente consolidadas na literatura, tais como precisão (*precision*), revocação (*recall*), *F1-score* e tempo de inferência [12], utilizando conjuntos de dados personalizados desenvolvidos pelo discente. A comparação entre as versões permitirá avaliar não apenas a eficácia das melhorias implementadas ao longo do tempo, mas também os compromissos existentes entre desempenho e eficiência computacional. Dessa forma, os objetivos específicos deste trabalho são:

- Implementar, treinar e ajustar as versões YOLOv8, YOLOv9, YOLOv10 e YOLOv11, comparando seus desempenhos com base em diferentes métricas de avaliação.
- Investigar o impacto das versões analisadas em cenários de detecção de objetos em tempo real, considerando aspectos como consumo de recursos computacionais, velocidade de inferência e qualidade dos resultados.
- Discutir os principais desafios enfrentados por cada versão da arquitetura YOLO, destacando as melhorias introduzidas e suas limitações remanescentes.

A relevância desta comparação reside na possibilidade de identificar quais inovações trouxeram ganhos efetivos de desempenho e quais ainda enfrentam limitações decorrentes de *trade-offs* entre precisão e velocidade. Além disso, os resultados obtidos podem contribuir para orientar a escolha da versão mais adequada da arquitetura YOLO em aplicações específicas, levando em conta fatores como o hardware disponível, os requisitos de tempo real e a complexidade do ambiente de detecção.

1.4 Contribuições do Trabalho

As principais contribuições deste trabalho podem ser assim resumidas:

- Realização de uma revisão prática e comparativa das versões YOLOv8 a YOLOv11, com ênfase em aplicações reais de detecção de objetos.
- Condução de uma avaliação sistemática sob condições controladas, utilizando as mesmas configurações de treinamento, métricas padronizadas e base de dados unificada, assegurando rigor e validade na comparação.
- Elaboração de uma análise crítica dos *trade-offs* entre acurácia e eficiência computacional, oferecendo subsídios concretos para decisões de engenharia em projetos de visão computacional.
- Registro estruturado de desafios enfrentados e soluções adotadas, constituindo material de apoio para pesquisadores e profissionais interessados em explorar versões mais recentes do YOLO.

Assim, este trabalho configura-se como uma referência prática e aplicada para estudantes e profissionais que desejam compreender e explorar o potencial das versões modernas do YOLO, equilibrando desempenho técnico e viabilidade de implantação.

Capítulo 2

Revisão Bibliográfica

A primeira etapa tratou, além da recapitulação de conceitos básicos, de uma revisão bibliográfica abrangente sobre as diferentes arquiteturas da YOLO, com foco nas versões 8 a 11, analisando artigos científicos, documentações oficiais e materiais técnicos para compreender as arquiteturas, técnicas de aprendizado profundo e inovações introduzidas em cada versão.

Essa revisão vai incluir:

- O estudo de arquiteturas das Redes Neurais, detalhando as estruturas das redes, como seus *backbones*, *necks* e *heads* de predição e outras estruturas que o compõe.
- As técnicas de Aprendizado Profundo, com uma análise dos métodos, como normalização em lote (*Batch Normalization*), aumento de dados (*Data Augmentation*) e funções de perda (*Loss Functions*) utilizadas para melhorar o desempenho.
- E inovações específicas, como a identificação das principais melhorias em cada versão, como o uso de *anchors* dinâmicos, treinamento multi-escala e otimizações pós-processamento.

2.1 Panorama e Evolução dos Detectores

A detecção de objetos percorreu um arco histórico que vai de métodos baseados em engenharia de atributos, como HOG [6], às CNNs profundas inauguradas em larga

escala por AlexNet [17] e consolidadas por arquiteturas residuais [11]. Esse movimento está bem documentado por revisões abrangentes [38] e pelas bases teóricas de *deep learning* [10], que explicam por que redes convolucionais substituíram pipelines manuais. No contexto específico de detecção, a transição dos modelos *two-stage* (R-CNN, Fast/Faster R-CNN) [9, 8, 27] para *one-stage* (SSD, YOLO) [20, 26, 24] marcou um ponto de inflexão: unificar a predição de caixas e classes em uma única passagem de rede, reduzindo latência e tornando viáveis aplicações em tempo real.

Uma síntese histórica do ecossistema YOLO é oferecida por Peiyuan Jiang [14], que descreve a progressão de versões anteriores (até v4/v5) e discute tendências como detecção em múltiplas escalas, *heads* desacoplados e avanços em *losses*. Mais recentemente, [36] sumariza a linha YOLOv1→YOLOv10, destacando as motivações por trás de cada salto (robustez, rapidez, eficiência de *deploy*), e situando os modelos em *benchmarks* padronizados. Essa visão de longo prazo é central para pesquisa: mesmo quando hiperparâmetros aparentam ser “iguais”, diferenças de desenho arquitetural e protocolos de treinamento (atribuição de rótulos, *schedulers*, perdas) mudam o comportamento final.

2.2 AlexNet

O surgimento da AlexNet no desafio ImageNet de 2012 marcou o início da era moderna do *deep learning*. Krizhevsky, Sutskever e Hinton treinaram uma rede profunda de oito camadas convolucionais utilizando GPUs para acelerar o processamento, algo inédito até então. O modelo reduziu em quase 50% o erro de classificação top-5 do ImageNet em relação ao segundo colocado, provando que redes profundas eram superiores em reconhecimento de padrões complexos. Além disso, introduziu técnicas que se tornaram padrão, como a função de ativação ReLU, que acelerava a convergência; o uso de *dropout* para reduzir *overfitting* e a estratégia de *data augmentation* para aumentar a robustez. A AlexNet não apenas quebrou recordes, mas também trouxe uma mudança de paradigma na comunidade, estabelecendo a viabilidade prática das CNNs em grandes bases de dados [17].

2.3 ResNet

A publicação das *Residual Networks* (ResNet) por He et al. representou outro salto conceitual. Modelos com mais de 50 ou 100 camadas eram impraticáveis devido ao problema do gradiente desaparecendo, o que limitava a profundidade das redes. A ResNet resolveu esse problema introduzindo conexões de atalho (*skip connections*), que permitiam que a rede aprendesse funções residuais em vez de funções diretas. Essa inovação viabilizou o treinamento de redes extremamente profundas, com até 152 camadas, sem perda de acurácia. O impacto foi imediato: a ResNet venceu o desafio ImageNet de 2015 com grande margem e se tornou um *backbone* padrão em múltiplas tarefas, incluindo detecção de objetos, segmentação e reconhecimento facial. Até hoje, muitas versões do YOLO e outros detectores utilizam ResNets ou variações como CSP-ResNet em seus *backbones* [11].

2.4 Aplicações Práticas de CNNs e da Arquitetura YOLO

Em cenários de detecção e rastreamento contínuo, o YOLO tem sido amplamente comparado a métodos mais complexos, como R-CNN e *Faster* R-CNN, demonstrando menor latência e operação estável mesmo sob cargas intensas de processamento. Essa eficiência faz do YOLO a escolha ideal para aplicações que exigem resposta imediata, como sistemas de vigilância, monitoramento de tráfego, drones e robótica móvel. A integração entre precisão e velocidade o consolida como uma solução prática para detecção embarcada, em contraste com modelos *two-stage* mais pesados [16].

No campo médico, embora estudos como o de Litjens et al. não sejam centrados especificamente na arquitetura YOLO, eles demonstram o impacto de detectores rápidos na triagem assistida por IA, como na detecção de lesões, nódulos e anomalias em imagens radiológicas [19]. Nesses contextos, a eficiência do modelo é fundamental para o aumento do *throughput* clínico, permitindo diagnósticos mais ágeis e escaláveis. As evoluções estruturais do YOLO — como cabeçalhos desacoplados, abordagens *anchor-free*

e melhorias na regressão de *bounding boxes* — reforçam essa aplicabilidade em sistemas que demandam alta precisão aliada a desempenho em tempo real [14, 38].

Na indústria e no controle de tráfego, aplicações práticas de CNNs e YOLO são igualmente expressivas. Estudos recentes mostram que, embora versões mais novas do YOLO apresentem ganhos marginais em *benchmarks* ideais, fatores ambientais como variação de iluminação, oclusões e objetos pequenos podem modificar a hierarquia de desempenho entre versões. Tais achados sustentam a observação de que o YOLOv8, por exemplo, pode superar modelos mais recentes em condições reais devido à sua estabilidade e maturidade arquitetural [14, 16].

Além das aplicações diretas em detecção, CNNs e modelos derivados como o YOLO vêm sendo empregados em reconhecimento facial, autenticação biométrica e sistemas de vigilância, onde a capacidade de extrair características discriminantes permite identificar indivíduos mesmo sob variações de iluminação e ângulo de visão. Em paralelo, a combinação entre CNNs e redes generativas adversárias (GANs) possibilitou avanços em geração e aprimoramento de imagens, incluindo restauração, reconstrução e técnicas de super-resolução [10].

2.5 Arquiteturas da Família YOLO

Nas últimas décadas, a detecção automática de objetos em imagens e vídeos, destaca-se o algoritmo YOLO (*You Only Look Once*), cuja abordagem inovadora trouxe avanços significativos em termos de desempenho, precisão e viabilidade para aplicações em tempo real. Desde sua primeira versão, lançada em 2016, o YOLO evoluiu continuamente, incorporando melhorias estruturais e funcionais que o tornaram uma das soluções mais eficazes na área [26].

O YOLO diferencia-se por tratar a tarefa de detecção de objetos como um único problema de regressão, o que contrasta com abordagens anteriores, como o R-CNN e suas variantes, que segmentavam o processo em múltiplas etapas, como a extração de regiões propostas e posterior classificação. No YOLO, a imagem de entrada é dividida em uma grade regular, e cada célula dessa grade é responsável por prever um conjunto

de caixas delimitadoras e suas respectivas probabilidades de classe, resultando em um processo mais simples, direto e eficiente. Essa arquitetura permite que a rede seja treinada de forma fim a fim (*end-to-end*), o que favorece tanto a precisão quanto a velocidade da detecção.

A primeira versão do algoritmo, conhecida como YOLOv1, apresentou bons resultados em tarefas envolvendo objetos de tamanho médio e grande. No entanto, seu desempenho era limitado quando se tratava de detectar objetos pequenos ou múltiplos objetos próximos uns dos outros, problemas que foram parcialmente solucionados nas versões subsequentes. A versão YOLOv2, também conhecida como YOLO9000, introduziu o conceito de anchor boxes, além de melhorias na arquitetura da rede neural e suporte para detecção de milhares de classes, combinando aprendizado supervisionado e não supervisionado. [14]

Com a chegada do YOLOv3, houve uma reformulação significativa da arquitetura da rede, que passou a utilizar o *backbone Darknet-53*. Essa versão implementou detecção em múltiplas escalas e obteve maior precisão em comparação às versões anteriores, consolidando-se como uma das preferidas para aplicações industriais. Já o YOLOv4, lançado em 2020, incorporou técnicas modernas de regularização, otimização e aumento de dados, sendo projetado especificamente para maximizar a precisão e a velocidade de inferência mesmo em hardwares convencionais. Essa versão representou um marco importante, por ter sido projetada de forma independente por Alexey Bochkovskiy, após Joseph Redmon, criador original do YOLO, anunciar seu afastamento do projeto por razões éticas [25].

Em paralelo, foi desenvolvida a versão YOLOv5, pela empresa Ultralytics, que não faz parte da linha oficial criada por Redmon e Bochkovskiy, mas se tornou amplamente adotada na comunidade por sua acessibilidade, documentação robusta e compatibilidade com o *framework* PyTorch. Essa versão popularizou o uso do YOLO em ambientes de produção e pesquisa aplicada. Posteriormente, as versões YOLOv6 e YOLOv7 foram introduzidas com foco em otimizações específicas para aplicações industriais e dispositivos embarcados, trazendo melhorias tanto na arquitetura da rede quanto na eficiência

computacional.

A versão mais recente, YOLOv11, também desenvolvida pela Ultralytics, representa uma reformulação profunda do modelo, incorporando uma nova arquitetura base, suporte nativo a tarefas adicionais como segmentação e rastreamento, além de oferecer APIs simplificadas para treinamento, inferência e implantação. Essa versão destaca-se por sua modularidade e flexibilidade, o que a torna adequada para uma ampla gama de aplicações, desde a inspeção visual automatizada até a análise de imagens médicas e agrícolas [31].

A evolução das versões do YOLO reflete não apenas avanços tecnológicos, mas também a crescente demanda por soluções que aliem alta performance, baixo consumo de recursos computacionais e facilidade de integração em sistemas reais. Em contextos como segurança pública, automação industrial, transporte autônomo e monitoramento ambiental, o uso de modelos YOLO tem-se mostrado vantajoso pela capacidade de realizar detecções com precisão e em tempo real, fatores essenciais para aplicações em ambientes dinâmicos e críticos.

Portanto, o YOLO consolidou-se como uma das principais abordagens para detecção de objetos, e sua contínua evolução demonstra o vigor e a relevância da área de visão computacional baseada em aprendizado profundo. O conhecimento sobre essas versões e suas particularidades permite uma escolha mais consciente e adequada do modelo a ser empregado em diferentes contextos e restrições computacionais.

Capítulo 3

Conceitos Básicos de Deep Learning

O livro "Deep Learning" de Goodfellow, Bengio e Courville [10] consolidou o conhecimento do campo em uma única referência, sendo considerado a obra fundamental para pesquisadores e profissionais da área.

A publicação combina rigor matemático com aplicações práticas, cobrindo fundamentos como redes neurais, convoluções, regularização e aprendizado não supervisionado. Na área de visão computacional, o livro descreve como CNNs revolucionaram a extração de características, substituindo métodos manuais como SIFT e HOG.

A obra também contextualiza a evolução histórica do aprendizado profundo e apresenta direções futuras, como o uso de modelos generativos e *attention mechanisms*. Para qualquer pesquisa que envolva YOLO ou detecção de objetos, este livro serve como pilar teórico essencial [10].

3.1 R-CNN e Faster R-CNN

O trabalho de Girshick introduziu o R-CNN, é considerado um marco inicial na aplicação de redes neurais convolucionais para detecção de objetos. O modelo seguia a abordagem em duas etapas (two-stage): primeiro, utilizava-se um algoritmo seletor de regiões (*Selective Search*) para propor possíveis áreas de interesse em uma imagem, e depois cada região era processada individualmente por uma CNN para classificação e regressão de *bounding boxes*. Apesar de alcançar resultados de alta precisão, o R-CNN

apresentava forte limitação de desempenho, uma vez que o processamento de cada região era independente, tornando o método inadequado para aplicações em tempo real [9] .

Como evolução do R-CNN, Ren et al. apresentaram o Faster R-CNN, que introduziu a Rede de Propostas de Regiões (RPN), substituindo a etapa externa de geração de propostas por um mecanismo integrado à rede. Isso reduziu consideravelmente o custo computacional, permitindo maior eficiência. Ainda assim, o modelo continuava sendo um método em duas etapas, o que limitava sua aplicabilidade em cenários de baixa latência. Contudo, o Faster R-CNN consolidou-se como referência no estado da arte até o surgimento das arquiteturas *one-stage* [27].

3.2 Transformers

Os *Transformers*, introduzidos por Vaswani et al. no artigo *Attention Is All You Need*, marcaram uma mudança fundamental no aprendizado profundo ao substituir as arquiteturas recorrentes e convolucionais pelo mecanismo de *self-attention*. Esse mecanismo permite que o modelo identifique relações entre todos os elementos de uma entrada simultaneamente, sem depender da posição ou da distância entre eles. Cada token é projetado em três vetores, *Query*, *Key* e *Value*, e a similaridade entre *Query* e *Key* determina a importância relativa de cada elemento, ponderando os valores correspondentes. Essa operação paralelizada, expandida em múltiplas cabeças de atenção, fornece uma compreensão contextual rica e global, favorecendo o aprendizado de dependências de longo alcance [33].

Além do *self-attention*, os Transformers empregam camadas de normalização, projeções lineares e redes totalmente conectadas (*feed-forward*), interligadas por conexões residuais que estabilizam o fluxo de gradientes. Por não possuírem estrutura recursiva ou convolucional, os modelos utilizam codificações posicionais (*positional encodings*) para preservar a ordem dos elementos da sequência. Essa arquitetura altamente paralelizável tornou possível o treinamento de modelos em larga escala, como BERT, GPT e T5, que redefiniram o estado da arte em diversas tarefas de linguagem natural. Posteriormente, a mesma lógica foi aplicada à visão computacional, com variantes como o *Vision Trans-*

former (ViT) e o *Swin Transformer*, mostrando que o *self-attention* poderia substituir convoluções em tarefas visuais complexas [7, 21].

3.3 Comparações Metodológicas: One-stage, Two-stage e Transformers

A literatura contrasta dois paradigmas: os *two-stage* (ex.: Faster R-CNN) [27], com *region proposals* e maior precisão em contextos complexos, e os *one-stage* (YOLO, SSD) [20, 26], que priorizam latência reduzida e *throughput* alto. Em paralelo, *Transformers* trouxeram uma formulação *end-to-end* com o DETR [4], eliminando *anchors* e NMS na fase de inferência à custa de treinamento mais custoso e requisitos de dados maiores. Esse eixo comparativo — *accuracy* vs *latency* vs *data hunger* — aparece de forma recorrente nas revisões [38, 14].

No ciclo mais recente, YOLOv9 e YOLOv10 representam duas linhas de investigação complementares: YOLOv9 propõe mecanismos para aproveitar melhor o fluxo de gradientes e a agregação eficiente de camadas, enquanto YOLOv10 mira eficiência de ponta a ponta e treinamento que dispensa NMS (via estratégias de atribuição *one-to-one/one-to-many* e *heads/stages* treinados para reduzir redundância) [35, 34]. O panorama de [36] ajuda a enxergar como essas ideias se conectam com a linhagem original do YOLO.

Capítulo 4

Materiais e Métodos

Neste capítulo, são apresentados os materiais, ferramentas e metodologias empregados na condução dos experimentos. Isso inclui a descrição do ambiente computacional, as ferramentas utilizadas, os procedimentos adotados para a coleta e o tratamento dos dados, bem como os parâmetros definidos para o treinamento dos modelos de detecção de objetos.

A padronização do processo experimental visa garantir a reprodutibilidade dos resultados e permitir uma comparação justa entre os diferentes modelos analisados.

4.1 Descrição do Ambiente e Ferramentas

Os experimentos foram realizados em um ambiente de desenvolvimento preparado para suportar treinamentos intensivos de redes neurais profundas. A configuração do hardware e software utilizada foi a seguinte:

- **Sistema Operacional:** Ubuntu 22.04 LTS (Linux).
- **Processador (CPU):** I9-13900 K (5.8GHz Max Turbo, Cache 36MB, 24 Núcleos, 32 Threads).
- **Memória RAM:** 128GB DDR5.
- **Placa de Vídeo (GPU):** 2 NVIDIA GeForce RTX 4090 de 24GB.

- **Armazenamento:** HDD 8TB 7200 RPM e NVME 2TB 7300 Leitura/ 7000 gravação.

Do ponto de vista de software e *frameworks*, foram utilizadas as seguintes ferramentas:

- **Python** 3.11 como linguagem principal.
- **PyTorch** 2.0.1 como *backend* de *deep learning*.
- **Ultralytics YOLO:** *framework* oficial para treinamento e inferência de modelos YOLOv8, YOLOv9, YOLOv10 e YOLOv11, disponível via *pip*.
- Pandas e Matplotlib para análise e visualização de dados.
- *Jupyter Notebook* e *Visual Studio Code* como ambientes de desenvolvimento.

Este ambiente permitiu a execução eficiente dos experimentos, com suporte a aceleração via CUDA, garantindo um tempo de treinamento reduzido e o melhor aproveitamento da GPU.

4.1.1 Configuração de Treinamento dos Modelos

Para garantir uma comparação justa entre as versões do modelo YOLO, foram adotados critérios padronizados de treinamento. Todos os modelos foram treinados com a mesma base de dados, mesmas proporções de divisão e mesmos parâmetros, excetuando-se as particularidades arquiteturais de cada versão.

Os parâmetros de treinamento definidos foram:

- **Tamanho das imagens:** 640x640 *pixels*
- **Número de épocas:** 100 e 200
- **Batch size:** 8
- **Otimizador:** *Stochastic Gradient Descent* (SGD) com *momentum* de 0.937

- **Scheduler de taxa de aprendizado:** *Cosine Annealing*
- **Taxa de aprendizado final (lr0):** 0.000298
- **Taxa de aprendizado inicial (lrf):** 0.01
- **Loss function:** objetiva de classificação, localização (*bounding box regression*) e confiança
- **Augmentações:** *flip* horizontal, escala, brilho/contraste e *blur* leve
- **Early stopping** com paciência de 10 épocas sem melhoria de mAP

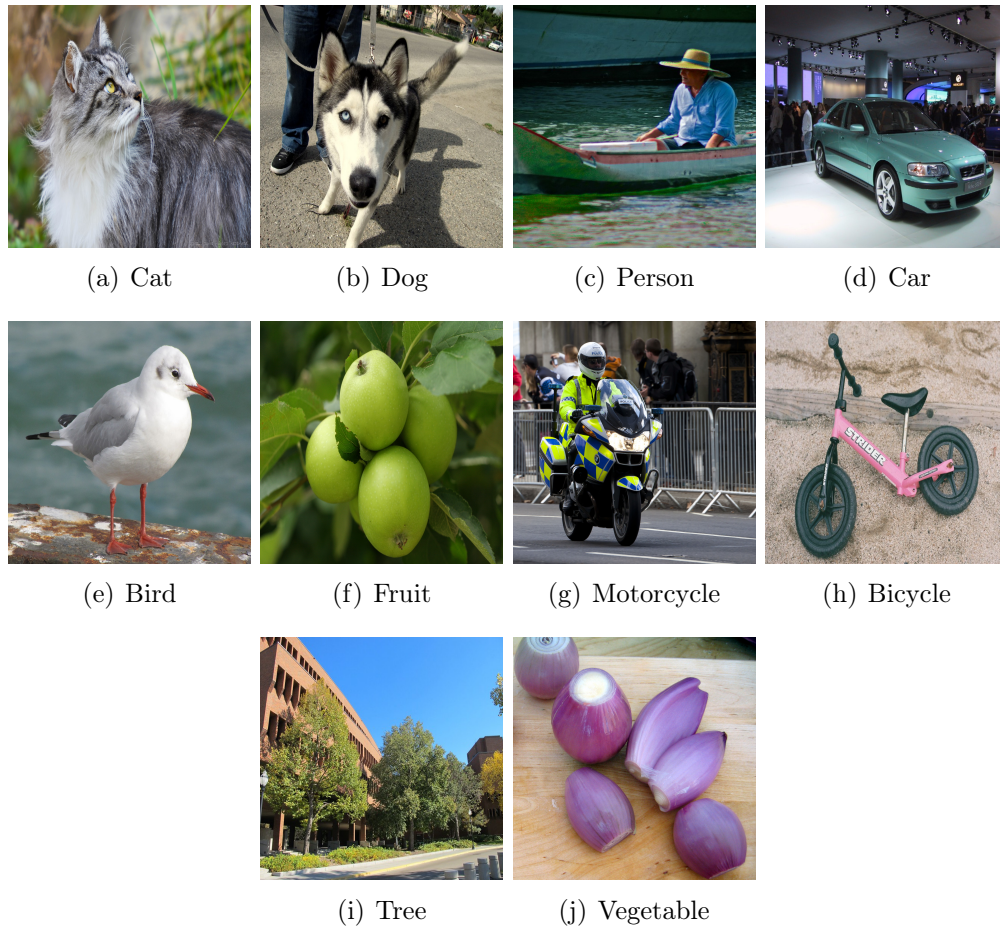
As versões dos modelos utilizadas foram as “medium” (YOLOv8m, YOLOv9m, etc.), com objetivo de avaliar o equilíbrio entre desempenho e velocidade. O processo de treinamento foi monitorado utilizando o *TensorBoard* da própria *Ultralytics*, que permite a visualização em tempo real das métricas de *loss*, mAP e outras estatísticas relevantes.

4.2 Dataset Utilizado

A base de dados utilizada neste trabalho foi composta por um conjunto balanceado de 10.000 imagens, distribuídas igualmente entre 10 classes de objetos demonstradas na Figura 4.1: *fruit*, *person*, *car*, *tree*, *bicycle*, *dog*, *cat*, *bird*, *motorcycle* e *vegetable*. A escolha dessas categorias buscou representar um conjunto diversificado de elementos do cotidiano, abrangendo tanto objetos naturais quanto artificiais, de modo a favorecer a generalização do modelo em contextos variados.

Fonte das Imagens: As imagens foram obtidas majoritariamente do repositório público Open Images Dataset [28], conhecido por conter anotações detalhadas e de alta qualidade. Para complementar classes com menor representatividade, foram incluídas imagens coletadas manualmente de fontes abertas adicionais, garantindo maior equilíbrio entre as categorias.

Conversão para o formato YOLO: Cada imagem foi acompanhada de um arquivo de anotação no formato `.txt` seguindo o padrão do YOLO, em que as coordenadas

Figura 4.1: Amostras do *dataset*

são normalizadas em relação às dimensões da imagem. Esse formato assegura compatibilidade direta com os modelos utilizados e otimiza o desempenho durante o treinamento. O formato segue esses valores:

$$[class][x_center][y_center][width][height]$$

Pré-processamento: Antes do treinamento, todas as imagens passaram por um processo de padronização, incluindo:

- Redimensionamento para resolução de 640×640 pixels;
- Conversão para o espaço de cores RGB;
- Normalização dos valores de pixel;

- Remoção de imagens duplicadas, desbalanceadas ou com baixa qualidade visual.

Essas etapas visaram reduzir ruídos e inconsistências, promovendo maior estabilidade no processo de aprendizagem.

Divisão do conjunto de dados: O *dataset* foi dividido conforme proporções comumente adotadas em tarefas de detecção de objetos:

- 70% das imagens destinadas ao treinamento;
- 30% reservadas para teste e validação.

Essa divisão garantiu que o modelo fosse avaliado em dados não vistos previamente, permitindo uma análise mais fidedigna do seu desempenho real.

Relevância do conjunto de dados: A composição deste *dataset* representa um diferencial importante do trabalho, pois alia diversidade visual, equilíbrio entre classes e qualidade nas anotações. Essa combinação fornece uma base sólida para a avaliação comparativa entre diferentes arquiteturas de detecção de objetos, permitindo testar a capacidade dos modelos em reconhecer padrões variados e contextos distintos. Além disso, a escolha por classes amplamente representativas reforça o potencial de aplicação prática dos resultados obtidos, aproximando o experimento de cenários reais de uso em visão computacional.

4.3 Critérios de Avaliação

As revisões [14, 38] enfatizam a importância de *benchmarks* padronizados (ex.: COCO, VOC) e métricas como mAP@[0.5:0.95] , *precision/recall* e latência. Trabalhos aplicados [16] destacam FPS, *throughput* e consumo de memória como métricas imprescindíveis para tempo real. O que justifica reportar mAP@0.5 e mAP@[0.5:0.95] , tempo de inferência (ou FPS) e tamanho do modelo/parametrização, além de registrar *seeds* e variação entre execuções, dado que a literatura reconhece sensibilidade a inicializações.

Deste modo, a avaliação dos modelos foi realizada com base em métricas amplamente utilizadas na literatura para tarefas de detecção de objetos [12]. As métricas consideradas para este trabalho foram:

Precisão (*Precision*): proporção de predições corretas dentre todas as predições realizadas.

Revocação (*Recall*): proporção de objetos verdadeiros corretamente identificados pelo modelo.

F1-Score: média harmônica entre precisão e revocação.

mAP@0.5 (*mean Average Precision*): média da acurácia em todas as classes com $\text{IoU} \geq 0,5$.

mAP@0.5:0.95: média ponderada considerando múltiplos limiares de IoU entre 0,5 e 0,95, em passos de 0,05.

Tempo de Duração do Treino (segundos): tempo em que o modelo demorou para ser treinado nas especificações de nosso ambiente.

Essas métricas fornecem uma visão abrangente tanto da qualidade das predições quanto da viabilidade de aplicação dos modelos em ambientes de tempo real.

4.4 Tópicos Técnicos Recorrentes na Literatura

Esquemas de ancoragem e atribuição: A transição de *anchor-based* para *anchor-free* e o uso de atribuição de rótulos mista (one-to-one/one-to-many) influenciam fortemente convergência e qualidade das caixas. Mesmo com “hiperparâmetros iguais”, detalhes do algoritmo de atribuição e do balanceamento de perdas mudam a dinâmica do treinamento [20, 24, 34].

Funções de perda e regressão de caixas: A literatura compara perdas IoU/CIoU/GIoU e ajustes de regressão, buscando localização mais estável sob *jitter* e ruído de *labels*. As revisões [14, 38] mostram que ganhos modestos e consistentes vêm de *losses* mais informativas e de melhor uso dos gradientes [35].

Data augmentation, regularização e schedulers: Estratégias de *augmentation* (*Mosaic*, *MixUp*), EMA, *cosine decay* e *warmup* aparecem como *knobs* de generalização.

Pequenas diferenças implementacionais entre famílias/versões podem explicar variações de resultado — ponto que reforça análises como a sua, com protocolo padronizado e ablação quando possível [14, 36].

NMS-free e eficiência de deploy: A proposta de treinar o modelo para reduzir a dependência do NMS ganha tração com YOLOv10 , aproximando detectores *one-stage* da promessa *end-to-end* do DETR sem arcar com o custo total de *training* dos *Transformers*. Isso é especialmente relevante para edge/embarque [34, 4].

4.5 Aprofundamento nos Modelos YOLO

4.5.1 YOLOv8

Na base da arquitetura, o YOLOv8 adota um *backbone* inspirado na ideia de CSP (*Cross Stage Partial connections*), que já havia sido introduzido em versões anteriores, mas com modificações importantes. Essa estrutura melhora a reutilização de gradientes e reduz redundâncias no fluxo de informação, permitindo que a rede extraia representações visuais ricas sem aumentar drasticamente os custos computacionais. Além disso, o *backbone* combina convoluções otimizadas com blocos de agregação espacial e contextual, o que ajuda a melhorar o aprendizado de padrões de diferentes escalas. Esse aspecto é essencial, já que objetos em imagens podem variar em tamanho e a rede precisa ter robustez para lidar com múltiplas resoluções [31].

A sua principal inovação está na cabeça de detecção (*detection head*). Enquanto versões anteriores do YOLO utilizavam uma cabeça de detecção ancorada (*anchor-based*), neste a equipe optou por uma abordagem *anchor-free*, o que simplifica tanto a configuração quanto a adaptação a diferentes *datasets*. Isso significa que o modelo não depende mais de caixas pré-definidas para prever objetos, mas aprende diretamente os pontos de referência (*keypoints*) e *bounding boxes* durante o treinamento. Essa mudança reduz a necessidade de cálculos adicionais, melhora a generalização e torna o treinamento menos sensível a erros na escolha de parâmetros de ancoragem, que historicamente eram um gargalo.

Outro componente essencial é o uso de DFL (*Distribution Focal Loss*) na regressão de *bounding boxes*. Essa função de perda permite que o modelo refine as previsões de localização com maior precisão, aproximando a regressão de coordenadas a uma distribuição contínua ao invés de tratá-la como um problema de classificação discreta. O resultado é um ganho consistente em métricas como $\text{mAP}@0.5:0.95$, sem comprometer a velocidade. A rede também emprega variações de perdas adaptadas para classificação e detecção, equilibrando *recall* e precisão durante o treinamento.

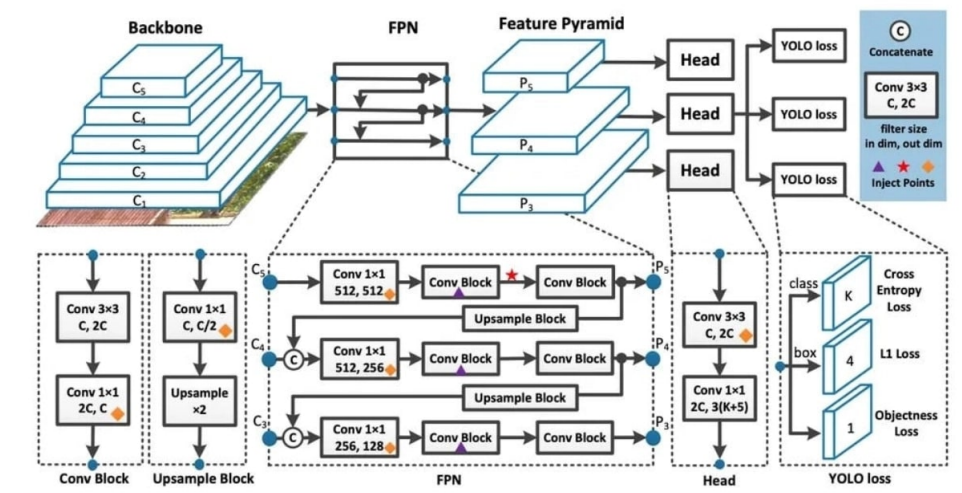


Figura 4.2: Arquitetura simplificada do YOLOv8 [32]

Ilustrada na Figura 4.2 temos a estrutura modular do YOLOv8, que representa um marco evolutivo na transição para modelos *anchor-free*. A imagem evidencia a divisão clássica em *backbone*, *neck* e *head*, com o *backbone* baseado em CSP (*Cross Stage Partial connections*) e blocos otimizados de convolução. Essa organização visual demonstra a eficiência do fluxo de informação entre camadas e a reutilização de gradientes — fatores diretamente responsáveis pela redução das perdas observadas nos experimentos. Além disso, o diagrama evidencia a simplicidade do pipeline *end-to-end*, indicando um modelo mais leve, generalista e de treinamento estável.

Em termos de desempenho, o YOLOv8 foi projetado para alta velocidade e versatilidade. Ele suporta diferentes tamanhos de arquitetura — desde versões extremamente leves, como YOLOv8n (nano), até versões robustas, como YOLOv8x (*extra-large*). Essa modularidade possibilita que o usuário selecione o modelo mais adequado ao seu caso:

dispositivos de borda, como drones ou câmeras de vigilância com restrições de hardware, podem utilizar versões menores; aplicações que exigem maior precisão em ambientes de alta performance, como centros de processamento em nuvem, podem adotar versões maiores.

Ele também se destaca pela integração com o ecossistema da *Ultralytics*, distribuído com uma API amigável, suporte a exportação para diferentes formatos (ONNX, TensorRT, CoreML, OpenVINO) e recursos de treinamento automatizados, incluindo *hyperparameter tuning*. Além disso, há suporte direto para tarefas além da detecção de objetos, como segmentação e *pose estimation*, o que amplia sua aplicabilidade em diversos domínios.

Acompanhado por uma infraestrutura de treinamento robusta, integrada ao ecossistema da *Ultralytics*, com suporte a *transfer learning*, possui quantização e exportação para múltiplos formatos (ONNX, TensorRT, CoreML), fatores que explicam sua rápida adoção no mercado e na comunidade acadêmica.

4.5.2 YOLOv9

O ponto central do YOLOv9 é a busca por maior eficiência no fluxo de gradientes durante o treinamento, algo que vinha sendo identificado como um gargalo em arquiteturas anteriores. Para isso, a versão incorporou dois componentes principais: o GELAN (*Generalized Efficient Layer Aggregation Network*) e o PGI (*Programmable Gradient Information*) [35].

O GELAN atua como o coração do *backbone* do modelo. Ele foi concebido como uma evolução dos blocos CSP, aproveitando os conceitos de agregação de camadas para tornar o aprendizado mais eficiente e flexível. A ideia é combinar informações de múltiplas profundidades de forma a reutilizar representações sem redundância, mas sem perder a riqueza semântica necessária para capturar padrões complexos. Essa abordagem proporciona um ganho de eficiência computacional, pois reduz a duplicação de cálculos e melhora a adaptação da rede em tarefas que exigem generalização. Ao mesmo tempo, mantém a compatibilidade com implementações leves, o que garante que

o YOLOv9 continue aplicável a cenários de tempo real.

O segundo pilar, o PGI, é talvez a inovação mais marcante. Trata-se de uma técnica que permite “programar” o fluxo de gradientes, priorizando a retropropagação das informações mais relevantes para o aprendizado de representações úteis. Em arquiteturas tradicionais, a retropropagação tende a ser difusa, afetando todas as camadas de forma relativamente uniforme. O PGI quebra esse padrão ao introduzir um mecanismo seletivo, que orienta melhor o aprendizado da rede para as partes mais críticas da tarefa de detecção. Isso resulta em ganhos consistentes no mAP@0.5:0.95, que é a métrica mais exigente de avaliação da qualidade de *bounding boxes*.

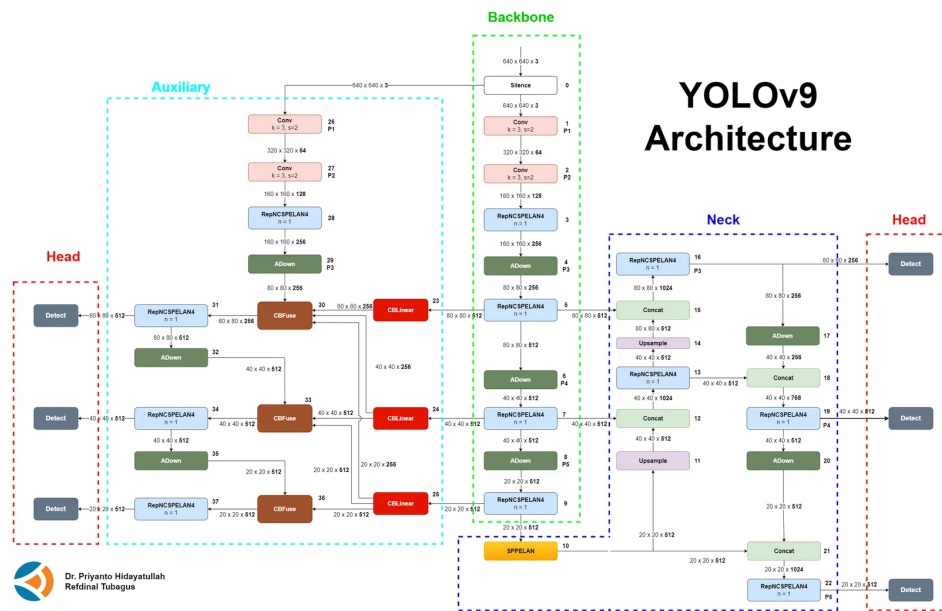


Figura 4.3: Arquitetura simplificada do YOLOv9 [13]

Mostra-se na Figura 4.3 a arquitetura do YOLOv9, na qual se destaca o uso do GELAN no *backbone* e o módulo PGI. Visualmente, observa-se uma ampliação das conexões internas entre blocos convolucionais, reforçando a ideia de agregação eficiente de camadas. Essa organização é projetada para melhorar a passagem de gradientes e, consequentemente, o aprendizado de características em diferentes profundidades. Contudo, o próprio esquema revela uma estrutura mais densa e complexa, o que se traduz nos resultados experimentais.

Assim como o YOLOv8, o YOLOv9 também é *anchor-free*, o que mantém o modelo mais simples e adaptável a diferentes *datasets*. A regressão de caixas é auxiliada pelo

uso do DFL (*Distribution Focal Loss*), permitindo previsões mais refinadas e suaves nas fronteiras dos objetos. O treinamento também se beneficia da compatibilidade com técnicas modernas de *data augmentation* e do suporte nativo a pipelines de pré-processamento otimizados.

Portanto, o modelo representa um ponto de equilíbrio entre eficiência computacional e qualidade de predição, sendo bastante explorado em aplicações industriais e em contextos de *edge computing*.

4.5.3 YOLOv10

Enquanto o YOLOv8 buscava simplicidade com sua arquitetura *anchor-free* e o YOLOv9 inovava no fluxo de gradientes com GELAN e PGI, o YOLOv10 se destaca por atacar um gargalo histórico: a dependência do NMS (*Non-Maximum Suppression*) na fase de pós-processamento.

O NMS foi, durante anos, uma parte essencial de qualquer detector de objetos. Ele atua eliminando predições redundantes de *bounding boxes*, escolhendo as mais confiáveis para cada objeto. Entretanto, esse processo adiciona complexidade e custo, além de ser uma heurística manual que nem sempre generaliza bem em cenários com objetos densamente agrupados. O YOLOv10 propõe um treinamento que elimina essa etapa, tornando o em um modelo *NMS-free*. Isso é alcançado por meio de uma estratégia de atribuição mais sofisticada, chamada de *one-to-one* e *one-to-many matching*, em que as caixas previstas são ligadas diretamente aos objetos-alvo durante o treinamento, evitando redundância desde a fase de aprendizado.

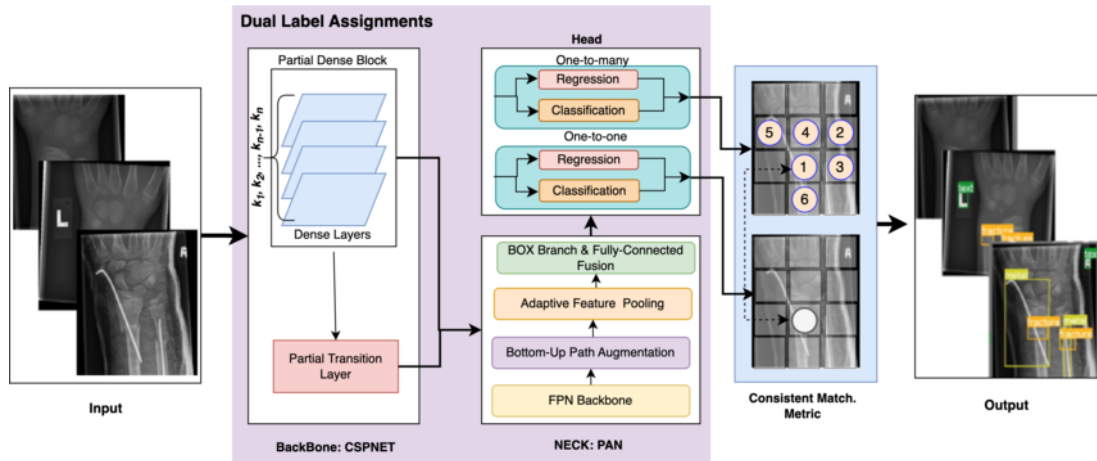


Figura 4.4: Arquitetura simplificada do YOLOv10 [1]

Revelado na Figura 4.4 vemos a proposta inovadora do YOLOv10, que rompe com a dependência do NMS. O diagrama mostra uma cabeça de detecção reorganizada, com múltiplos estágios de refinamento e uma nova estratégia de atribuição *one-to-one* / *one-to-many*, que substitui o NMS por aprendizado supervisionado direto. Visualmente, nota-se um pipeline mais linear, com menos blocos redundantes entre o *backbone* e o *head*, reforçando o foco dessa versão em eficiência operacional e otimização para aplicações em tempo real. Além dessa mudança estrutural, também incorpora melhorias nos cabeçalhos de detecção (*heads*), ajustando-os para que sejam capazes de lidar com diferentes estágios de refinamento sem precisar de filtros adicionais.

Essa reorganização da cabeça de detecção reduz tanto a redundância de predições quanto a sobrecarga computacional, mantendo a filosofia de redes rápidas e aplicáveis em tempo real. Em paralelo, há uma reorganização da pipeline de treinamento, que se beneficia da combinação de diferentes formas de atribuição entre predições e objetos, tornando o modelo mais robusto em *datasets* variados.

Outra característica importante do YOLOv10 é sua eficiência energética e de latência. Com a remoção do NMS e a redução de redundâncias, o modelo consegue entregar resultados competitivos com menor tempo de inferência. Nos experimentos comparativos, embora não tenha atingido a mesma precisão absoluta do YOLOv8 em algumas métricas, ele se destacou pela baixa latência e *throughput* elevado, o que o torna particularmente adequado para aplicações em dispositivos embarcados, sistemas autônomos

e aplicações móveis, onde cada milissegundo importa.

Ele também dialoga com tendências mais recentes da visão computacional ao trazer uma aproximação do conceito *token-to-token*, inspirado nas arquiteturas de *Transformers*. Embora não seja um *Transformer* puro como o DETR, incorpora a ideia de reduzir redundâncias representacionais, tratando partes da imagem de maneira mais independente, o que aproxima a arquitetura da granularidade dos *tokens*. Isso não apenas simplifica a inferência, mas também amplia a escalabilidade do modelo para tarefas mais complexas sem incorrer nos altos custos de treinamento característicos dos *Transformers*.

Na prática, isso significa que o YOLOv10 é mais rápido em pipeline *end-to-end*, além de apresentar ganhos em precisão e estabilidade. Essa versão foi projetada especialmente para aplicações em tempo real em que velocidade é um ponto crítico, como veículos autônomos e sistemas de monitoramento em tempo real.

4.5.4 YOLOv11

Enquanto o YOLOv8 foi a base sólida da transição, o YOLOv9 trouxe inovações no fluxo de gradientes com o GELAN e PGI, e o YOLOv10 eliminou o NMS em nome da eficiência, o YOLOv11 surge como uma síntese dessas ideias, buscando entregar equilíbrio entre precisão, estabilidade e generalização em múltiplos cenários.

Na sua estrutura, o YOLOv11 mantém a filosofia *one-stage* herdada da família, mas refina a organização de camadas intermediárias, integrando mecanismos de agregação de informações mais eficientes e uma melhor adaptação para diferentes tamanhos de *input*. Isso permite que o modelo seja facilmente escalado desde aplicações em dispositivos móveis até servidores com GPUs robustas, mantendo consistência de desempenho. Um dos pontos centrais é a sua capacidade de adaptação multi-escala, aprimorada em relação às versões anteriores, o que o torna particularmente competitivo em contextos onde objetos variam muito em tamanho ou aparecem em densidades distintas.

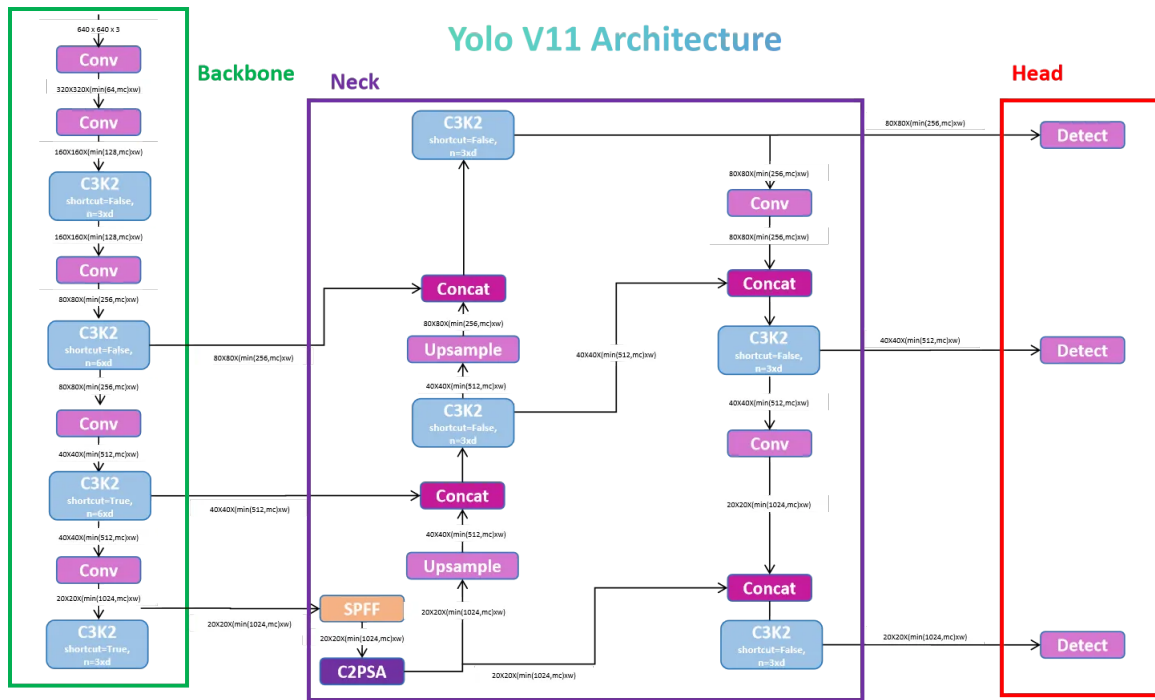


Figura 4.5: Arquitetura simplificada do YOLOv11 [23]

Na Figura 4.5 é apresentada a evolução do YOLOv11, uma síntese equilibrada das melhorias introduzidas nas versões anteriores. O diagrama mostra uma arquitetura modular e escalável, com mecanismos de agregação multi-escala mais refinados e conexões intermediárias otimizadas para estabilidade de treinamento. Visualmente, percebe-se um design mais coeso, sem as redundâncias das versões anteriores, o que garante maior generalização e eficiência, especialmente em cenários de hardware limitado ou *datasets* heterogêneos.

Outro avanço importante está na otimização do treinamento, ele incorpora estratégias de balanceamento de perdas e refinamento de atribuição que reduzem tanto a sobrecarga de redundância quanto a sensibilidade a hiperparâmetros. Isso significa que, em comparação ao seus antecessores, o YOLOv11 apresenta maior estabilidade no processo de convergência, reduzindo flutuações e mantendo curvas de aprendizado mais consistentes. Essa característica se reflete nos gráficos de precisão e *recall*, onde o modelo tende a mostrar crescimento mais suave e sustentado, atingindo bons resultados sem oscilações abruptas.

Outro aspecto notável do modelo é sua generalização. Enquanto versões anteriores

poderiam ser vistas como mais “especializadas”, como o YOLOv8 em precisão, o YOLOv9 em eficiência de gradiente e o YOLOv10 em latência, o YOLOv11 se posiciona como um modelo versátil, capaz de atender a diferentes tipos de aplicações com resultados equilibrados. Esse caráter híbrido o torna uma escolha sólida para projetos em que não se deseja abrir mão nem de precisão nem de velocidade, como em sistemas de monitoramento de tráfego, robótica industrial ou aplicações médicas, onde tanto a acurácia quanto a confiabilidade temporal são cruciais.

Em termos de uso prático, a versão 11 se beneficia de uma maior maturidade do ecossistema, sendo lançado já com suporte consistente em bibliotecas de treinamento e inferência modernas. Isso facilita a adoção e garante que desenvolvedores e pesquisadores possam integrá-lo rapidamente em pipelines já existentes. Além disso, o modelo apresenta resiliência a datasets heterogêneos, performando bem em contextos que exigem maior generalização, como ambientes não controlados ou aplicações em larga escala.

Capítulo 5

Resultados e Discussão

Neste capítulo, são apresentados e analisados os resultados obtidos ao longo dos experimentos realizados. Foram conduzidos quatro testes principais, cada um correspondente a uma versão distinta do modelo *You Only Look Once* (YOLO), abrangendo as versões YOLOv8, YOLOv9, YOLOv10 e YOLOv11.

A estrutura experimental foi mantida constante em todos os casos, variando apenas a arquitetura e os pesos pré-treinados de cada modelo. Essa padronização visa garantir condições equitativas de comparação, permitindo que as diferenças observadas nos resultados sejam atribuídas exclusivamente às particularidades estruturais e funcionais de cada versão.

A seguir, são apresentados os resultados quantitativos obtidos, incluindo métricas como mAP@0.5:0.95, precisão, *recall* e tempo de inferência, e uma discussão qualitativa que busca interpretar como as inovações arquiteturais de cada versão influenciaram seu desempenho geral.

5.1 Experimento 1 – YOLOv8

Modelo: YOLOv8m

Tabela 5.1: Dados resultantes do YOLOv8

Epoch	Precision	Recall	mAP 50	mAP 50-95	Train Box Loss	Train Cls Loss	Train Dfl Loss	Duração (sec)
100	0.88910	0.86884	0.92019	0.75171	0.71736	0.48711	1.04845	56280.5
200	0.90855	0.89215	0.94030	0.80148	0.53781	0.36454	0.93372	23495.9

Apresenta-se a Tabela 5.1 os resultados quantitativos obtidos no treinamento do modelo YOLOv8m, considerado a base de comparação para as demais versões. Observamos que, com 100 épocas, o modelo já atinge valores elevados de precisão (0,889) e *recall* (0,868), além de um mAP@0.5 de 92,0% e mAP@0.5:0.95 de 75,1%, demonstrando rápida convergência. Ao estender o treinamento para 200 épocas, há um ganho visível em todas as métricas — mAP@0.5 sobe para 94,0% e mAP@0.5:0.95 para 80,1% —, com redução expressiva nas perdas (*Box*, *Cls* e *DFL*). Isso mostra que o YOLOv8 mantém aprendizado efetivo mesmo em longos treinamentos, indicando excelente estabilidade e equilíbrio entre precisão e revocação. O custo temporal de 23.495 segundos (aprox. 6h30min) reflete seu alto poder de generalização, ainda que a um custo computacional considerável, coerente com sua complexidade estrutural.

Figura 5.1: Graficos de Precision x Recall (PR)

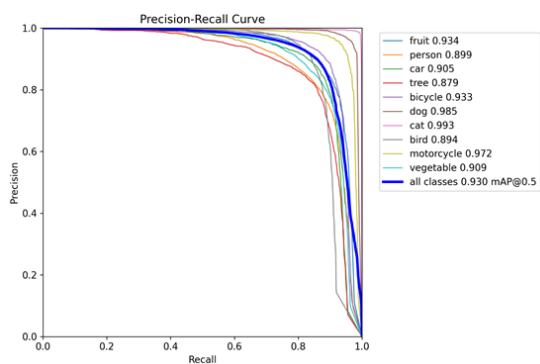


Figura 5.2: PR com 100 epochs

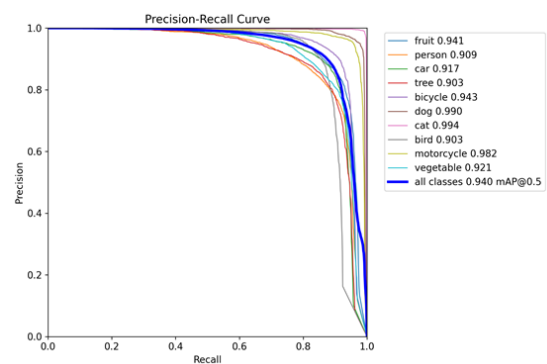


Figura 5.3: PR com 200 epochs

No gráfico PR 5.1 demonstra-se a relação entre a precisão e o *recall* ao longo do treinamento do YOLOv8. Observa-se uma curva suave e consistente, evidenciando que o modelo conseguiu equilibrar bem a identificação correta de objetos (*recall*) e a minimização de falsos positivos (precisão). Esse comportamento indica convergência estável e capacidade de generalização sólida, o que é típico de arquiteturas maduras e

bem otimizadas.

Comparando os resultados com 100 e 200 épocas, nota-se que o aumento no número de épocas favoreceu o refinamento das fronteiras de decisão do modelo. As curvas deslocam-se para regiões de maior área sob a curva (AUC), indicando que o YOLOv8 continuou aprendendo de forma produtiva, sem sinais de *overfitting*. Isso reforça sua robustez em treinamentos prolongados e estabilidade da função de perda.

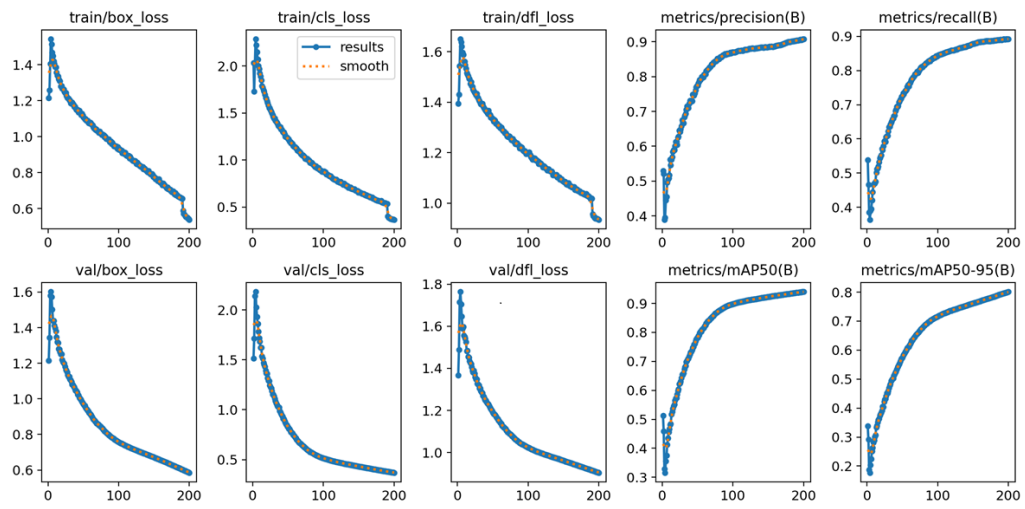


Figura 5.4: Resultados gerais do modelo à 200 epochs

No gráfico 5.4, visualiza-se o comportamento integrado das métricas. O modelo atingiu precisão e *recall* próximos de 0,9 e mAP@0.5:0.95 de 0,80, desempenho considerado de excelência em detecção de objetos genéricos. As perdas (*Box*, *Cls* e *DFL*) mostram redução progressiva, evidenciando aprendizado consistente e convergência efetiva.

5.2 Experimento 2 – YOLOv9

Modelo: YOLOv9m

Tabela 5.2: Dados resultantes do YOLOv9

Epoch	Precision	Recall	mAP@0.5	mAP@0.5:0.95	Train Box Loss	Train Cls Loss	Train Dfl Loss	Duração (s)
100	0.88849	0.86887	0.91767	0.75100	0.73414	0.49409	1.07869	11167.2
200	0.89130	0.88353	0.92633	0.77700	0.57616	0.3889	0.97957	53785.7

Evidencia-se na Tabela 5.2 que o YOLOv9m apresenta desempenho sólido, mas ligeiramente inferior ao YOLOv8 nas métricas principais. Com 200 épocas, o modelo

alcançou $\text{mAP}@0.5$ de 92,6% e $\text{mAP}@0.5:0.95$ de 77,7%, resultados competitivos que confirmam a eficiência do mecanismo PGI (*Programmable Gradient Information*). Entretanto, as perdas (*Train Box*, *Cls* e *DFL*) mostram-se um pouco superiores, sugerindo que o fluxo de gradientes mais intenso pode introduzir instabilidade durante o aprendizado. O tempo total de treinamento, de 53.785 segundos, é cerca de duas vezes maior que o YOLOv8, o que reduz sua eficiência global em termos de custo-benefício, ainda que mantenha excelente sensibilidade na detecção.

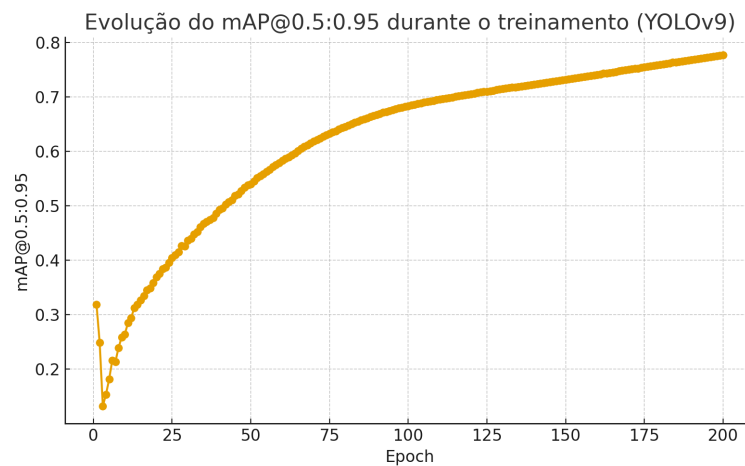


Figura 5.5: Evolução do $\text{mAP}@$ por epoch

O YOLOv9 apresentou evolução mais irregular. A curva de mAP 5.5 cresce rapidamente nas primeiras épocas, mas estabiliza-se em valores ligeiramente inferiores ao YOLOv8. Essa flutuação pode ser atribuída ao novo mecanismo PGI, que intensifica o fluxo de gradientes em fases específicas, mas pode gerar instabilidade local. Ainda assim, o modelo mostrou ganhos progressivos e convergência satisfatória.

5.3 Experimento 3 – YOLOv10

Modelo: YOLOv10m

Tabela 5.3: Dados resultantes do YOLOv10

Epoch	Precision	Recall	$\text{mAP}@0.5$	$\text{mAP}@0.5:0.95$	Train Box Loss	Train Cls Loss	Train Dfl Loss	Duração (s)
100	0.88854	0.85973	0.91731	0.75529	1.56769	1.03262	2.14018	10324.2
200	0.88925	0.88460	0.93028	0.78533	1.22148	0.77392	1.93554	76198.5

Os resultados do YOLOv10m destacam a eficiência e a economia de recursos da arquitetura *NMS-free*. Apesar de uma leve queda em precisão (0,889) e *recall* (0,884), o modelo entrega mAP@0.5 de 93,0% e mAP@0.5:0.95 de 78,5%, valores apenas ligeiramente inferiores aos melhores resultados obtidos. As perdas são mais elevadas em comparação às versões anteriores (especialmente Box e DFL Loss), refletindo o ajuste estrutural do modelo para eliminar redundâncias de detecção. O tempo total de treinamento de 76.198 segundos demonstra que, embora mais rápido em inferência, o treinamento ainda é intensivo. Esse modelo mostra-se ideal quando o foco está em tempo real e baixa latência, mesmo com leve perda de acurácia.

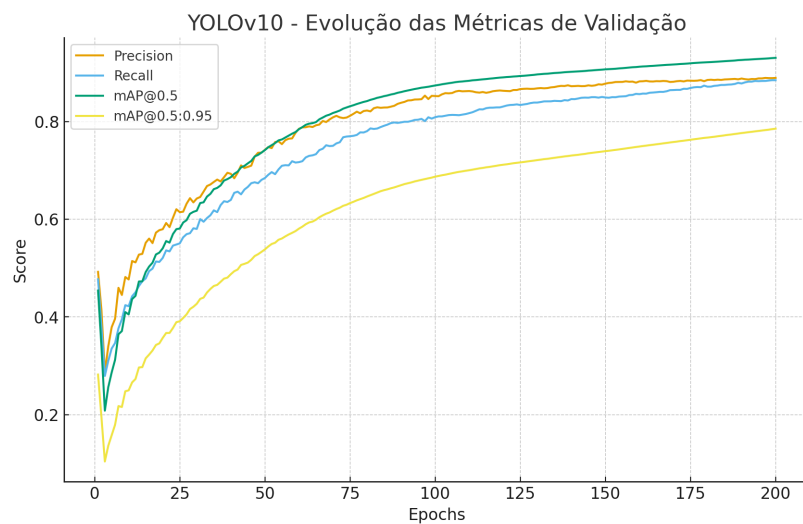


Figura 5.6: Evolução das Métricas do YOLOv10

Expõe-se na Figura 5.6 uma curva de crescimento mais moderado de precisão e *recall*, refletindo o impacto da remoção do NMS no pipeline. Embora a precisão final seja um pouco inferior, o modelo se destaca por estabilidade e rapidez de convergência. O YOLOv10 prioriza eficiência e velocidade de inferência, o que se alinha à proposta de aplicações em tempo real, mesmo que com leve perda de acurácia.

5.4 Experimento 4 – YOLOv11

Modelo: YOLOv11m

Tabela 5.4: Dados resultantes do YOLOv11

Epoch	Precision	Recall	mAP@0.5	mAP@0.5:0.95	Train Box Loss	Train Cls Loss	Train Dfl Loss	Duração (s)
100	0.89004	0.85911	0.91456	0.74466	0.74715	0.51514	1.05605	9607.18
200	0.89601	0.87824	0.92623	0.77623	0.56833	0.39112	0.94893	32082.7

O modelo YOLOv11m demonstrou a evolução significativa em eficiência temporal e estabilidade de aprendizado. Em 200 épocas, alcançou mAP@0.5 de 92,6% e mAP@0.5:0.95 de 77,6%, com precisão de 0,896 e *recall* de 0,878, valores muito próximos aos do YOLOv9, mas com menor custo computacional. A queda consistente das perdas (Box: 0,568; Cls: 0,391; DFL: 0,948) revela um treinamento mais estável, o que indica maior equilíbrio entre as classes e convergência eficiente. O tempo de execução, 32.082 segundos, é o menor entre as quatro versões, demonstrando que o YOLOv11 alcança um excelente compromisso entre desempenho e velocidade, sendo especialmente adequado a sistemas embarcados e dispositivos de borda.

5.5 Análise Comparativa

Os resultados quantitativos obtidos com os quatro modelos estão dispostos na Tabela 5.5:

Tabela 5.5: Comparação de desempenho entre os modelos YOLOv8 a YOLOv11

Modelo	mAP@0.5	mAP@0.5:0.95	Precisão	Revocação	F1-Score	Duração (s)
YOLOv8	94.0%	80.1%	90.8%	89.2%	90.02%	23495.9
YOLOv9	92.6%	77.7%	89.1%	88.3%	88.74%	53785.7
YOLOv10	93.0%	78.5%	89.0%	85.6%	88.72%	76198.5
YOLOv11	92.6%	77.6%	89.6%	87.8%	88.71%	32082.7

Expressado na Tabela 5.5, há uma comparação resumida entre os modelos em relação às suas métricas e ao tempo de inferência em seu ápice. Nele conseguimos perceber a variação dos dados em cada modelo, mostrando que o YOLOV8 conseguiu atingir resultados melhores que seus sucessores em nosso *dataset*. Deste modo, infere-se que mesmo com uma tecnologia mais antiga, é possível manter valores ainda relevantes em suas

métricas. Enquanto isso, seus sucessores ainda mantiveram resultados excelentes, mas devido sua especialização e avanços na sua arquitetura, acabaram sendo ultrapassados em nossos testes de caso. Por fim, o YOLOV11 manifestou-se como o mais próximo do YOLOV8, mostrando que o avanço da tecnologia continua aprimorando sua capacidade de detecção e ainda trazendo as inovações para os futuros modelos.

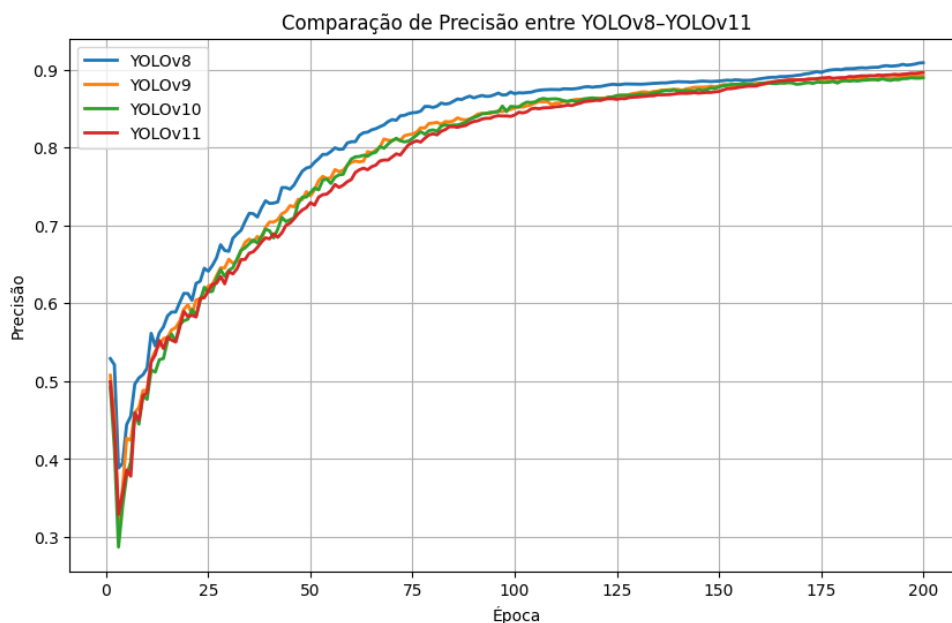


Figura 5.7: Evolução do Precisão entre versões

No gráfico de precisão 5.7, novamente o YOLOv8 demonstrou maior consistência, atingindo valores elevados e com pouca oscilação ao longo do treinamento. Isso evidencia sua capacidade de minimizar falsos positivos de maneira mais eficiente. O YOLOv9 obteve resultados aceitáveis, mas apresentou maior instabilidade, oscilando em algumas épocas. O YOLOv10 registrou valores mais baixos, o que sugere que sua especialização na eliminação de redundância de detecções acaba impactando negativamente na precisão geral. Já o YOLOv11, embora não tenha alcançado os melhores resultados, apresentou uma linha estável e próxima do YOLOv9, indicando bom equilíbrio entre custo e qualidade.

No gráfico de *recall* 5.8, a situação se inverte parcialmente. O YOLOv8 manteve bons valores, mas o YOLOv9, em algumas fases do treinamento, conseguiu acompanhar ou até superar levemente sua curva, mostrando que, apesar de menor precisão, o modelo é mais sensível em identificar objetos, mesmo que às custas de mais falsos positivos. O

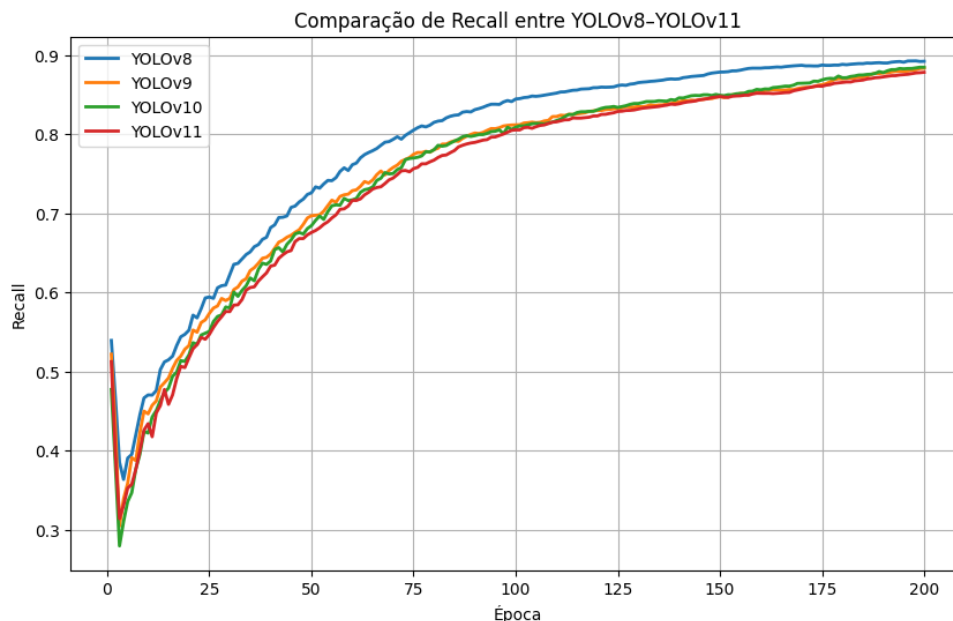


Figura 5.8: Evolução do Recall entre versões

YOLOv10 ficou abaixo de ambos, com *recall* inferior, reforçando que sua ênfase está em reduzir redundâncias e otimizar inferência, enquanto o YOLOv11 novamente ficou em posição intermediária, conseguindo resultados sólidos e equilibrados.

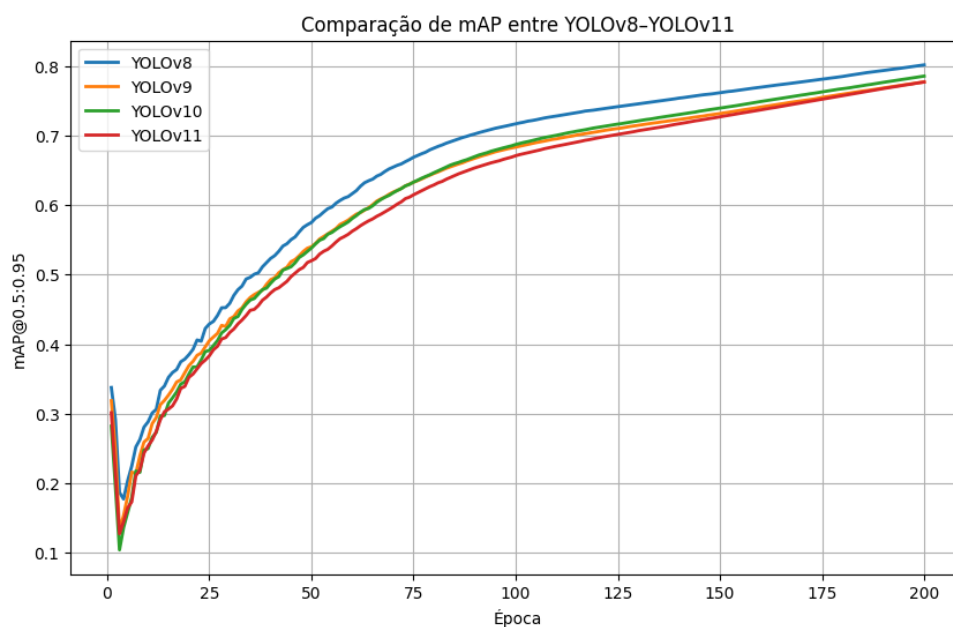


Figura 5.9: Evolução do mAP entre versões

Apresentado no gráfico 5.9, vemos de forma clara a evolução da acurácia global das quatro versões do YOLO. O YOLOv8 se destacou com uma curva ascendente mais estável e consistente, alcançando os maiores valores finais, o que confirma sua robustez e

eficiência no treinamento. Já o YOLOv9 apresentou um crescimento competitivo nas primeiras épocas, mas sua curva se estabilizou em patamares inferiores, indicando menor capacidade de generalização. O YOLOv10, apesar de trazer inovações arquiteturais, obteve resultados mais modestos, sugerindo que sua otimização está mais voltada para eficiência do que para maximizar a precisão. O YOLOv11 se manteve em uma posição intermediária, superando o YOLOv10 em mAP em diversos momentos, mas sem alcançar a estabilidade e a performance do YOLOv8.

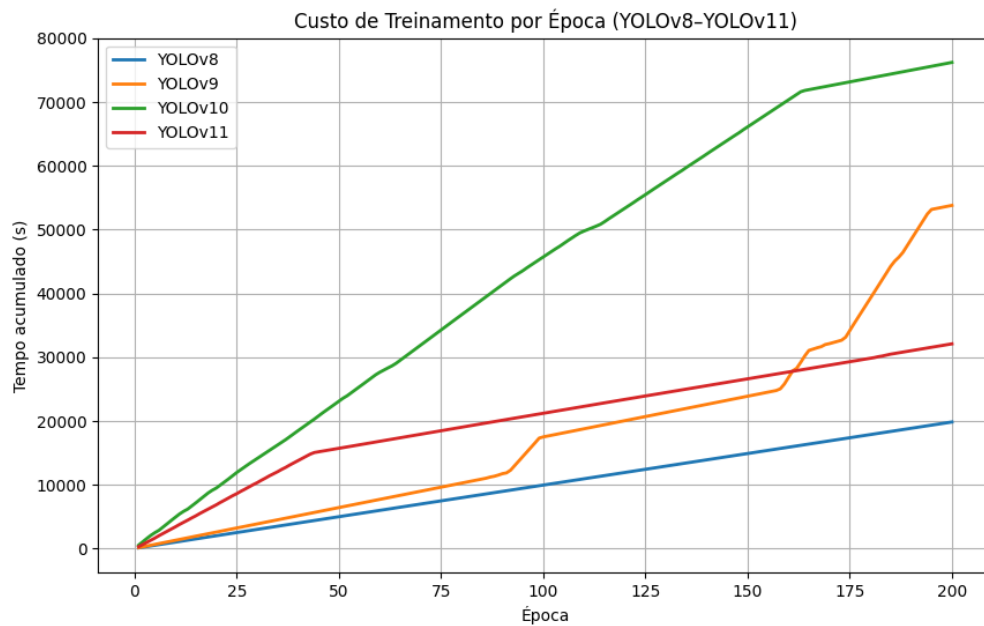


Figura 5.10: Custo de tempo de treinamento por época em cada versão

Observando o gráfico de custo por época 5.10, notamos outro contraste importante. O YOLOv8, apesar do ótimo desempenho, apresentou maior tempo de treinamento, refletindo o custo de sua alta acurácia. O YOLOv9 foi similar em termos de custo computacional, mas sem entregar os mesmos ganhos, o que o torna menos eficiente em termos de custo-benefício. O YOLOv10, mesmo com precisão e *recall* mais baixos, conseguiu tempos menores e inferência mais rápida, o que o torna adequado para cenários em que a velocidade é prioridade. Já o YOLOv11 apresentou os melhores resultados nesse quesito, com tempos reduzidos de forma significativa, confirmando sua vocação para aplicações em dispositivos embarcados e ambientes com restrições de hardware.

5.6 Considerações Finais

A análise comparativa entre YOLOv8, YOLOv9, YOLOv10 e YOLOv11 revelou que cada versão carrega características próprias em sua arquitetura que impactam diretamente nos resultados de precisão, *recall*, mAP e tempo de treinamento. Os resultados obtidos confirmam que as versões mais recentes do YOLO vêm incorporando avanços relevantes tanto em arquitetura quanto em desempenho. No entanto, cada versão carrega consigo uma proposta de valor específica. O modelo ideal, portanto, deve ser escolhido com base no tipo de aplicação desejada:

De modo geral, os resultados obtidos confirmam que não há um “melhor modelo absoluto”, mas sim arquiteturas com propostas específicas. O YOLOv8 apresentou-se como a opção mais estável e confiável em cenários gerais, enquanto o YOLOv9 se mostra promissor em contextos que exigem maior precisão em objetos pequenos ou *datasets* complexos. O YOLOv10, por sua vez, se posiciona como a versão mais interessante em aplicações de alta escala e baixa latência, enquanto o YOLOv11 oferece um compromisso adequado entre desempenho e eficiência, especialmente em dispositivos de borda.

Assim, a escolha da versão ideal deve levar em conta não apenas as métricas quantitativas obtidas em *benchmarks*, mas principalmente o contexto de aplicação, as restrições de hardware e a disponibilidade de dados. O estudo evidencia, portanto, a importância de alinhar os avanços arquiteturais das versões do YOLO às necessidades práticas do cenário de uso, reforçando que a evolução da família YOLO não segue apenas a busca pela maior acurácia, mas também pela diversificação de aplicações e eficiência em diferentes ambientes.

Capítulo 6

Conclusão

Este trabalho teve como objetivo principal comparar o desempenho de diferentes versões do modelo YOLO (You Only Look Once), especificamente as versões 8 a 11, aplicadas à tarefa de detecção de objetos. Através de uma abordagem experimental padronizada, foi possível avaliar o impacto das evoluções arquiteturais de cada versão em métricas como mAP, precisão, revocação e tempo de inferência. Nesta conclusão, discutem-se os principais achados do estudo, os *trade-offs* observados entre desempenho e eficiência, os desafios enfrentados ao longo do desenvolvimento, e as contribuições oferecidas pelo trabalho.

6.1 Comparação Direta Entre as Versões

Os resultados experimentais demonstraram que a versão YOLOv8 obteve o melhor desempenho geral em termos de precisão e *recall*, produzindo os maiores valores de mAP@0.5 e mAP@0.5:0.95. Isso pode ser explicado pela maturidade do modelo: sua implementação já é amplamente validada pela comunidade e beneficia-se de uma configuração padrão robusta, capaz de generalizar bem mesmo para *datasets* de tamanho médio ou pequeno. Assim, o YOLOv8 mostrou-se a versão mais estável e equilibrada neste cenário.

Para a versão YOLOv9, embora não tenha superado o YOLOv8 em termos de métricas finais em nosso experimento, foram identificadas contribuições arquiteturais impor-

tantes — tais como a introdução de mecanismos de atenção para aproveitamento mais eficaz dos gradientes durante o treinamento. De acordo com Murat AA et al. essa evolução inerente às versões mais novas aponta para melhorias, porém depende fortemente de cenários específicos (por exemplo, *datasets* maiores ou objetos mais complexos) para que as vantagens se manifestem de fato. Nos nossos experimentos, tais vantagens não se traduziram em ganhos expressivos, o que sugere que essa versão exige um ajuste de hiperparâmetros ou um volume de dados maior para alcançar seu potencial máximo [22].

A versão YOLOv10 introduziu uma abordagem end-to-end com menor dependência de NMS, o que a torna particularmente interessante para pipelines de baixa latência. Conforme discutido por Sapkota et al., embora versões recentes tragam ganhos em eficiência e precisão, esses ganhos ocorrem em nichos de aplicação e não traduzem uma superioridade absoluta para todos os cenários. No experimento deste trabalho, o YOLOv10 apresentou métricas intermediárias, com perdas de treinamento mais elevadas, indicando que, embora conceitualmente promissora, a versão demanda maior esforço de configuração para render resultados competitivos. Em cenários em que o processamento em tempo real e com grande volume de dados é prioritário, o YOLOv10 pode se tornar o mais adequado modelo, ainda que não tenha obtido as melhores métricas gerais nos testes [30].

Finalmente, o YOLOv11 apresentou resultados próximos aos do YOLOv9 — com valores de mAP e *recall* ligeiramente inferiores ao YOLOv8, porém com desempenho competitivo em termos de tempo de inferência e eficiência computacional. A revisão de Sapkota et al. relata que especificamente o YOLOv11 foi projetado com foco em eficiência para hardware restrito, por exemplo em borda ou IoT, reduzindo parâmetros e latência para atender tais exigências. Assim, embora o YOLOv11 não tenha superado o YOLOv8 em métricas de precisão em nosso estudo, mostra-se como opção mais indicada para aplicações embarcadas, onde a eficiência operacional é mais relevante que a máxima precisão [30].

Além disso, o estudo ODverse33 demonstra empiricamente que “a nova versão do

YOLO nem sempre é melhor em todos os domínios”. Em sua avaliação sobre 33 *datasets* de múltiplos domínios (v5 a v11), foi constatado que, em determinados domínios e condições, versões anteriores de YOLO ainda competem ou até mesmo superam versões mais recentes. Esse resultado reforça a tese central deste trabalho: mais importante do que a versão mais nova é a adequação da versão ao perfil da aplicação (tipo de dados, restrições de latência, hardware disponível, escala da implantação) [15].

De forma geral, os resultados obtidos neste estudo, indicam que as versões da série YOLO devem ser consideradas como alternativas que se complementam, e não como substituições automáticas lineares. Ou seja, cada versão responde a necessidades específicas — seja busca por maior precisão, menor latência, eficiência energética ou facilidade de implantação. Portanto, a escolha do modelo ideal depende fortemente do contexto de uso — um ponto de vista que se alinha diretamente com os desafios de escalabilidade, elasticidade e hardware heterogêneo em ambientes distribuídos ou em nuvem [22, 30, 15].

A Tabela 5.5 e os gráficos de desempenho confirmam que, embora as melhorias entre versões não sejam sempre exponenciais, elas são consistentes e direcionadas a desafios reais enfrentados em aplicações práticas de detecção.

6.1.1 Trade-offs

Um dos pontos mais relevantes da análise é o conjunto de *trade-offs* (compensações) envolvidos na escolha de cada versão:

Desempenho vs. Velocidade: versões mais recentes como YOLOv10 e YOLOv11 oferecem maior acurácia, mas exigem maior poder computacional, especialmente no tempo de inferência.

Complexidade vs. Portabilidade: modelos como YOLOv8, mais leves e simples, são preferíveis para aplicações móveis ou *edge computing*, mesmo que sacrifiquem um pouco de precisão.

Curva de aprendizado vs. performance: versões mais recentes exigem maior conhecimento técnico para ajustes finos, configuração e *deploy*, o que pode ser uma

barreira em projetos com prazos curtos ou equipes pequenas.

Em suma, não existe uma "melhor versão absoluta", mas sim a melhor versão para cada contexto específico de aplicação [2].

6.2 Desafios Encontrados

Durante o desenvolvimento do trabalho, foram enfrentados diversos desafios técnicos e práticos:

- **Instabilidade nas versões mais recentes:** o YOLOv11 ainda possui documentação limitada e sofreu atualizações durante os testes, o que exigiu adaptações e repetição de experimentos.
- **Gerenciamento e balanceamento de *dataset*:** foi necessário refinar o dataset para evitar desbalanceamento de classes, que poderia distorcer as métricas de avaliação.
- **Uso eficiente da GPU:** a configuração de *batch size* e *augmentation* precisou ser ajustada para evitar erros de falta de memória e gargalos durante o treinamento.
- **Diferenças nos hiperparâmetros padrões:** a configuração dos hiperparâmetros precisaram ser alteradas de modo que todas as versões apresentassem a mesma configuração geral, para serem avaliadas e comparadas corretamente.
- **Compatibilidade entre formatos:** garantir que os arquivos anotados estivessem corretamente formatados para cada versão do YOLO (especialmente ao alternar entre versões) demandou cuidado extra.

Esses desafios foram superados com pesquisa contínua, documentação abundante e específica, testes iterativos, apoio de ferramentas e repositórios abertos da comunidade.

6.3 Posição da Presente Pesquisa

À luz desses trabalhos, esse estudo se posiciona como uma avaliação comparativa pragmática: dialoga com as revisões históricas [14, 38], incorpora o estado da arte recente [36, 35, 34] e contrasta com aplicações de tempo real [16]. O achado de que YOLOv8 pode superar versões mais novas em certo cenário é coerente com a literatura: maturidade/estabilidade do pipeline vs inovações que dependem de dados maiores, maior diversidade ou ajuste fino para expressarem vantagem. Esse contraste realça a recomendação central observada nas revisões: a escolha do detector deve considerar não só o “*State of the Art*” nominal, mas o contexto de dados, restrições de *deploy* e as métricas alvo.

6.4 Trabalho Futuros

Com base nas limitações e oportunidades identificadas durante a execução do projeto, sugerem-se os seguintes desdobramentos futuros:

- **Ampliação do dataset:** testar os modelos com um conjunto de dados maior e mais diverso, inclusive com classes desbalanceadas ou cenários adversos.
- **Testes Avançado em tempo real:** avaliar o comportamento dos modelos em aplicações reais embarcadas, como robôs móveis, drones ou sistemas de vigilância.
- **Quantização e compressão dos modelos:** explorar estratégias para reduzir o tamanho dos modelos e acelerar a inferência sem perda significativa de desempenho.
- **Análise energética e consumo de recursos:** quantificar o custo energético de cada versão do YOLO.

Além disso, com o surgimento de novas versões (como futuras YOLOv12 ou variações como RT-DETR ou *Segment Anything*), abre-se a possibilidade de expandir a comparação para modelos baseados em arquiteturas diferentes, incorporando novos paradigmas como atenção, segmentação e multitarefas.

Bibliografia

- [1] Ammar Ahmed e Abdul Manaf. *Pediatric Wrist Fracture Detection in X-rays via YOLOv10 Algorithm and Dual Label Assignment System*. Jul. de 2024. DOI: 10.48550/arXiv.2407.15689.
- [2] Alexey Bochkovskiy, Chien-Yao Wang e Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV]. URL: <https://arxiv.org/abs/2004.10934>.
- [3] Gaudenz Boesch. *YOLOv10: Real-Time Object Detection Evolved*. <https://viso.ai/deep-learning/yolov10/>. Acessado em: 04 de fevereiro de 2025. 2024.
- [4] Nicolas Carion et al. “End-to-End Object Detection with Transformers”. Em: *European Conference on Computer Vision (ECCV)*. Springer, 2020.
- [5] Chenyi Chen et al. “DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving”. Em: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015.
- [6] Navneet Dalal e Bill Triggs. “Histograms of Oriented Gradients for Human Detection”. Em: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2005.
- [7] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. Em: *International Conference on Learning Representations (ICLR)* (2021). URL: <https://arxiv.org/abs/2010.11929>.
- [8] Ross Girshick. “Fast R-CNN”. Em: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015.

- [9] Ross Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation”. Em: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.
- [10] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. Cambridge: MIT Press, 2016.
- [11] Kaiming He et al. “Deep Residual Learning for Image Recognition”. Em: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [12] Steven Hicks et al. “On evaluation metrics for medical applications of artificial intelligence”. Em: *Scientific Reports* 12 (abr. de 2022). DOI: 10.1038/s41598-022-09954-8.
- [13] Dr. Priyanto Hidayatullah e Refdinal Tubagus. *YOLOv9 Architecture Explained*. Acesso em: 23 set. 2025. 2023. URL: <https://stunningvisionai.com/article/yolov9-architecture>.
- [14] Peiyuan Jiang et al. “A Review of Yolo Algorithm Developments”. Em: *Procedia Computer Science* 199 (2022). The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19, pp. 1066–1073. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2022.01.135>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050922001363>.
- [15] Tianyou Jiang e Yang Zhong. *ODverse33: Is the New YOLO Version Always Better? A Multi Domain benchmark from YOLO v5 to v11*. 2025. arXiv: 2502.14314 [cs.CV]. URL: <https://arxiv.org/abs/2502.14314>.
- [16] N. Murali Krishna et al. “Object Detection and Tracking Using Yolo”. Em: (2021), pp. 1–7. DOI: 10.1109/ICIRCA51532.2021.9544598.
- [17] Alex Krizhevsky, Ilya Sutskever e Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. Em: *Advances in Neural Information Processing Systems (NeurIPS)*. 2012.

- [18] Yann LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. Em: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [19] Geert Litjens et al. “A Survey on Deep Learning in Medical Image Analysis”. Em: *Medical Image Analysis* 42 (2017), pp. 60–88.
- [20] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. Em: *European Conference on Computer Vision (ECCV)*. Springer, 2016.
- [21] Ze Liu et al. “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”. Em: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 10012–10022. URL: <https://arxiv.org/abs/2103.14030>.
- [22] A. A. Murat et al. “A comprehensive review on YOLO versions for object detection”. Em: *Journal of King Saud University - Computer and Information Sciences* (2025). DOI: 10.1016/j.jksuci.2025.102154.
- [23] S Nikhileswara Rao. *YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced Speed and Accuracy*. Acesso em: 23 set. 2025. 2024. URL: <https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-%20enhanced-speed-and-accuracy-2dbe2d376f71>.
- [24] Joseph Redmon e Ali Farhadi. “YOLOv3: An Incremental Improvement”. Em: *arXiv preprint arXiv:1804.02767* (2018).
- [25] Joseph Redmon e Ali Farhadi. “YOLOv3: An Incremental Improvement”. Em: *ArXiv* abs/1804.02767 (2018). URL: <https://api.semanticscholar.org/CorpusID:4714433>.
- [26] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. Em: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
- [27] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. Em: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.

- [28] Google Research. *Open Images Dataset V7 and Extensions*. <https://storage.googleapis.com/openimages/web/index.html>. Acesso em: 20 abr. 2025. 2016.
- [29] Artur dos Santos Ferreira, Daniel M. Freitas e et al. “Weed detection in soybean crops using ConvNets”. Em: *Computers and Electronics in Agriculture* 143 (2017), pp. 314–324.
- [30] A. Sapkota et al. “YOLO advances to its genesis: a decadal and comprehensive review of the YOLO series”. Em: *Artificial Intelligence Review* (2025). DOI: 10.1007/s10462-025-11253-3.
- [31] ULTRALYTICS. *YOLOv8 Documentation*. <https://docs.ultralytics.com/>. Acessado em: 04 de fevereiro de 2025. 2022.
- [32] Ultralytics. *What is YOLOv8*. Acesso em: 23 set. 2025. 2023. URL: <https://yolov8.org/what-is-yolov8/>.
- [33] Ashish Vaswani et al. “Attention Is All You Need”. Em: *Advances in Neural Information Processing Systems (NeurIPS)* 30 (2017). URL: <https://arxiv.org/abs/1706.03762>.
- [34] Chien-Yao Wang, Alexey Bochkovskiy e Hong-Yuan Mark Liao. “YOLOv10: Real-Time End-to-End Object Detection”. Em: *arXiv preprint arXiv:2405.14458* (2024). URL: <https://arxiv.org/abs/2405.14458>.
- [35] Chien-Yao Wang, Alexey Bochkovskiy e Hong-Yuan Mark Liao. “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information”. Em: *arXiv preprint arXiv:2402.13616* (2024). URL: <https://arxiv.org/abs/2402.13616>.
- [36] Chien-Yao Wang e Hong-Yuan Mark Liao. *YOLOv1 to YOLOv10: The fastest and most accurate real-time object detection systems*. 2024. arXiv: 2408.09332 [cs.CV]. URL: <https://arxiv.org/abs/2408.09332>.
- [37] X. Zheng, S. Zheng, Y. Kong et al. “Recent advances in surface defect inspection of industrial products using deep learning techniques”. Em: *The International Journal of Advanced Manufacturing Technology* 113 (2021), pp. 35–58. DOI: 10.

1007/s00170-021-06592-8. URL: <https://link.springer.com/article/10.1007/s00170-021-06592-8>.

- [38] Zhiqiang Zou et al. “Object Detection in 20 Years: A Survey”. Em: *arXiv preprint arXiv:1905.05055* (2019).