Projeto 2 IIA - Yolo-MS para Detecção de Árvores

Caio Medeiros Balaniuk - 231025190
Davi Henrique Vieira Lima - 231013529
Lucca Schoen de Almeida - 231018900
Departamento de Ciência da Computação, UnB

***Resumo*— Este trabalho tem como objetivo aplicar e avaliar a arquitetura YOLO-MS para a detecção de copas de árvores individuais em imagens RGB de alta resolução de ambientes urbanos. A análise é baseada na comparação dos resultados com os 21 métodos testados no artigo de Zamboni et al. (2021). Utilizamos a base de dados disponibilizada pelo artigo original e medimos o desempenho com as mesmas métricas propostas, demonstrando a eficácia da abordagem recente YOLO-MS.**

***Abstract*— This work aims to apply and evaluate the YOLO-MS architecture for detecting individual tree crowns in high-resolution RGB images of urban environments. The analysis is based on a comparison with the 21 methods tested in the paper by Zamboni et al. (2021). We use the dataset made available by the original article and evaluate the performance using the same proposed metrics, demonstrating the effectiveness of the recent YOLO-MS approach.**

***Keywords*— Deep Learning, Object Detection, Urban Tree Detection, YOLO-MS, Remote Sensing, Cross-Validation, Aerial Imagery**

I. Introduction

A contagem de árvores em ambientes urbanos é uma tarefa fundamental para o planejamento urbano, a sustentabilidade ambiental e o monitoramento ecológico. Com o avanço das imagens aéreas em alta resolução, tornou-se possível aplicar técnicas modernas de aprendizado profundo (deep learning) para automatizar essa tarefa com alta precisão.

O trabalho de Zamboni *et al.* (2021) propõe um benchmark com 21 modelos de detecção de objetos voltados à identificação de copas de árvores individuais em áreas urbanas, oferecendo uma base sólida para comparações entre diferentes abordagens.

Neste projeto, propomos a aplicação do modelo YOLO-MS (2023), uma arquitetura recente e eficiente para detecção em tempo real, e avaliamos seu desempenho em relação ao benchmark proposto por Zamboni *et al.*. Utilizamos o mesmo conjunto de dados e métricas para garantir uma comparação justa e reprodutível. O código-fonte completo, bem como as instruções detalhadas para a reprodução dos experimentos, estão disponíveis em nosso repositório público no GitHub: https://github.com/DaviHVL/Yolo-MS-para-deteccao-de-arvores. Recomendamos fortemente consultar o arquivo *README.md* presente no repositório para orientações completas sobre instalação, uso e estrutura do projeto.

II. Fundamentação Teórica

II.I. Detecção de Objetos

A detecção de objetos é uma tarefa central na visão computacional, cujo objetivo é localizar e classificar automaticamente instâncias de objetos em imagens. Ao longo dos anos, diversos modelos têm sido propostos para essa finalidade, destacando-se arquiteturas como Faster R-CNN, RetinaNet e a família YOLO (You Only Look Once). Essas abordagens diferem tanto na estrutura quanto na estratégia de detecção, podendo ser classificadas em métodos baseados em *anchors* — que utilizam caixas pré-definidas para gerar previsões — e métodos *anchor-free*, que evitam esse processo e focam em representações mais diretas. Cada uma dessas abordagens apresenta diferentes compromissos entre acurácia, velocidade de inferência e custo computacional, sendo escolhidas conforme o cenário de aplicação.

II.II. YOLO-MS

YOLO (You Only Look Once) é uma arquitetura de detecção de objetos que realiza a tarefa em uma única etapa, tratando o problema como uma tarefa de regressão direta. A imagem é dividida em uma grade, e para cada célula são previstos múltiplos bounding boxes, junto com as classes e as confianças correspondentes. Essa abordagem permite que o modelo opere em tempo real, com boa precisão e alta velocidade de inferência.

YOLO-MS propõe uma reinterpretação das representações em múltiplas escalas, combinando eficientemente informações espaciais e semânticas. Ele introduz módulos como MSRF e RePPM para melhorar a qualidade da detecção, especialmente em objetos de tamanhos variados, mantendo a eficiência em tempo real. YOLO-MS propõe uma reinterpretação das representações em múltiplas escalas, combinando eficientemente informações espaciais e semânticas. Ele introduz módulos como MSRF e RePPM para melhorar a qualidade da detecção, especialmente em objetos de tamanhos variados, mantendo a eficiência em tempo real.

III. Trabalhos Relacionados

A utilização do YOLO-MS em outros problemas do mundo real é extremamente relevante no cenário atual. A alta eficiência combinada com a agilidade no processamento da informação se mostra como um excelente alicerce para soluções que exigem rapidez. Seguem projetos que também utilizam a rede neural:

- MS-YOLO (fluxo óptico + radar-milímetro);
  Song et al. (2022) usam YOLO-v5 aprimorado com fusão de radar e visão para percepção veicular.[1]

- MS-YOLO para detecção de afogamento;
  Song, Yao et al. (2024, Sensors) propõem uma versão leve para salvar vidas em ambientes aquáticos, com módulos MD-C2F, EMA, BiFPN e MSI-SPPF, obtendo 86,4 % em média no seu dataset, com apenas 7,3 GFLOPs;[2]

- MS-YOLOv8 em defeitos de pavimentação;
  Han et al. (2024) usam YOLOv8 modificado para detectar doenças em pavimentos;[3]

- MS-YOLO para imagens aéreas;
  Springer (2024) desenvolve MS-YOLO com multi-subnets e agregação dinâmica, alcançando ganhos expressivos em AP no VisDrone e DIOR (e.g. +23,9 %, de 30,9 para 38,3 AP)[4].
-

IV. Metodologia

Neste trabalho, adotamos uma abordagem sistemática para aplicar e avaliar a arquitetura YOLO-MS na tarefa de detecção de copas de árvores em ambientes urbanos. O desenvolvimento do projeto seguiu as seguintes etapas principais:

1) Preparação do Ambiente

Inicialmente, dois repositórios fundamentais foram clonados: o repositório *individual_urban_tree_crown_detection* [6], que fornece as imagens RGB e as anotações no formato YOLO, e o repositório *YOLO-MS* [5], que contém a implementação da arquitetura YOLO-MS e suas variantes dentro do framework MMYOLO.

2) Conversão para o Formato COCO

Como o MMYOLO utiliza o formato COCO para anotações, foi desenvolvido o script, em Python, responsável por converter os dados do formato YOLO para COCO. Durante esse processo, foram gerados arquivos de anotações de treino, teste e validação para cada um dos cinco folds definidos, sendo armazenados nas suas respectivas estruturas.

Cada *fold* utilizado no processo de validação cruzada **representa uma divisão distinta dos dados em subconjuntos de treinamento, validação e teste**. É importante destacar que, neste projeto, cada *fold* não corresponde a uma partição isolada dos dados originais, mas sim a uma divisão completa e independente do conjunto de dados em três conjuntos complementares (*train*, *val* e *test*), com diferentes amostras em cada um. Ou seja, os *folds* não são subconjuntos mutuamente exclusivos, mas sim diferentes formas de particionar o dataset completo para avaliar a capacidade de generalização do modelo em múltiplos cenários de separação dos dados.

A figura a seguir demonstra isso:

```
(YOLO-MS-mmyolo) C:\Users\davih\Projeto2_IIA\data>python convert_yolo_to_coco.py

=== Fold 0 ===
  ✔train: 132 imgs, 2124 annots → annotations\fold_0\instances_train.json
  ✔val: 44 imgs, 582 annots → annotations\fold_0\instances_val.json
  ✔test: 44 imgs, 676 annots → annotations\fold_0\instances_test.json

=== Fold 1 ===
  ✔train: 132 imgs, 2103 annots → annotations\fold_1\instances_train.json
  ✔val: 44 imgs, 683 annots → annotations\fold_1\instances_val.json
  ✔test: 44 imgs, 596 annots → annotations\fold_1\instances_test.json

=== Fold 2 ===
  ✔train: 132 imgs, 2013 annots → annotations\fold_2\instances_train.json
  ✔val: 44 imgs, 705 annots → annotations\fold_2\instances_val.json
  ✔test: 44 imgs, 664 annots → annotations\fold_2\instances_test.json

=== Fold 3 ===
  ✔train: 132 imgs, 2044 annots → annotations\fold_3\instances_train.json
  ✔val: 44 imgs, 699 annots → annotations\fold_3\instances_val.json
  ✔test: 44 imgs, 639 annots → annotations\fold_3\instances_test.json

=== Fold 4 ===
  ✔train: 132 imgs, 2040 annots → annotations\fold_4\instances_train.json
  ✔val: 44 imgs, 645 annots → annotations\fold_4\instances_val.json
  ✔test: 44 imgs, 697 annots → annotations\fold_4\instances_test.json
```

Fig. 1. Conversão do dataset ao formato COCO e criação dos folds para validação cruzada

3) Configuração do Modelo

A arquitetura escolhida para este trabalho foi a **YOLO-MS-XS**, uma variação leve e eficiente do modelo YOLO-MS, desenhada especificamente para aplicações em tempo real com restrições computacionais. Essa versão preserva a acurácia do modelo base ao mesmo tempo em que reduz a complexidade, sendo ideal para tarefas de detecção de objetos em imagens urbanas de alta resolução.

Para adaptar o modelo ao problema específico de detecção de copas de árvores, foram criados cinco arquivos de configuração ".py", cada um correspondente a um dos cinco folds utilizados na validação cruzada. Esses arquivos foram organizados em uma pasta específica do repositório e foram construídos a partir da herança do arquivo base *yoloms-xs_syncbn_fast_8xb32-300e_coco.py*, originalmente treinado no dataset COCO.

Cada arquivo de configuração define os **hiperparâmetros de treinamento**, como:

- A taxa de aprendizado é definida como **5e-4,** usando o otimizador AdamW.
- O número total de épocas estabelecidas é 100.
- A estrutura do treinamento baseada em *EpochBasedTrainLoop;*
- A frequência de validação é definida como 10.
- O uso de *EarlyStoppingHook* com monitoramento da métrica *coco/bbox_mAP_50*, paciência de 10 épocas e delta mínimo de 0.001.

Além disso, as configurações definem os caminhos dos arquivos de anotação COCO gerados previamente, ajustando corretamente os diretórios *data_root* e *data_prefix* para cada fold.

As metainformações do dataset, como o nome da única classe (**tree**) e a paleta de cores associada, também foram incluídas para serem utilizadas pelos dataloaders de treinamento, validação e teste.

O modelo foi ajustado para detectar apenas uma classe e utiliza pesos **pré-treinados** armazenados no arquivo nomeado *yoloms-xs_pre_trained.pth*, disponível no *Model Zoo* oficial do repositório *YOLO-MS* [5]. O uso desses pesos fornece uma inicialização mais eficaz e acelera a convergência do treinamento.

Com essa estrutura modular, foi possível automatizar o treinamento e avaliação de cada fold, mantendo consistência nas configurações e possibilitando comparações justas entre os experimentos.

4) Treinamento e Avaliação

Após configurar o ambiente de acordo com o documento *install_mmyolo.md*, o treinamento foi executado utilizando o script *train.py* do MMYOLO, com os arquivos de configuração criados. O modelo foi avaliado com o script *test.py*, utilizando o melhor checkpoint salvo com base na métrica *bbox_mAP_50*. Todas as métricas propostas por Zamboni et al. [1] foram utilizadas para garantir comparabilidade com os 21 métodos analisados no benchmark original.

5) Análise de Resultados

Os resultados foram coletados e organizados para permitir a análise por fold e a média geral dos desempenhos, destacando a eficácia da arquitetura YOLO-MS na tarefa específica de detecção de copas de árvores individuais em ambientes urbanos.

Observação: para um detalhamento em nível de código acerca do desenvolvimento, leia a seção “Desenvolvimento do Projeto” do *README.md* do repositório https://github.com/DaviHVL/Yolo-MS-para-deteccao-de-arvores.

V. Resultados

Como o modelo foi treinado utilizando validação cruzada com cinco folds, foi necessário analisar individualmente as métricas de desempenho obtidas em cada divisão. Em seguida, foi calculada a média entre os folds, a fim de fornecer uma estimativa mais robusta, confiável e representativa da performance geral do modelo em diferentes subconjuntos dos dados.

Fold 0

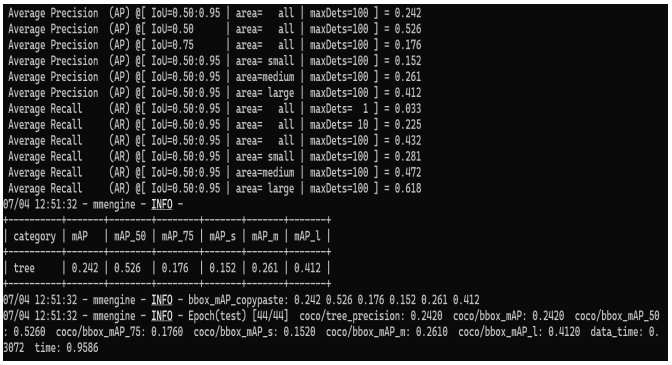

Fig. 2. Métricas de desempenho do Fold 0

Fold 1

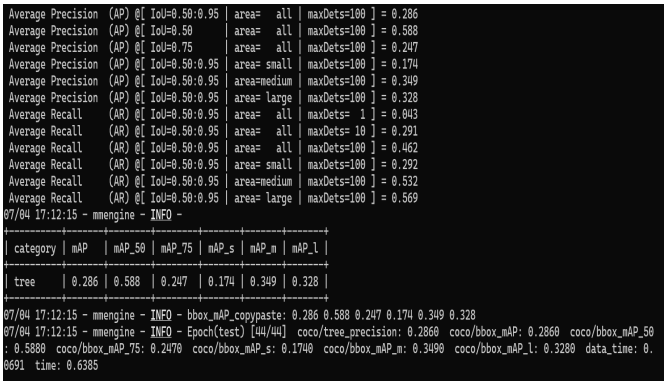

Fig. 3. Métricas de desempenho do Fold 1

Fold 2

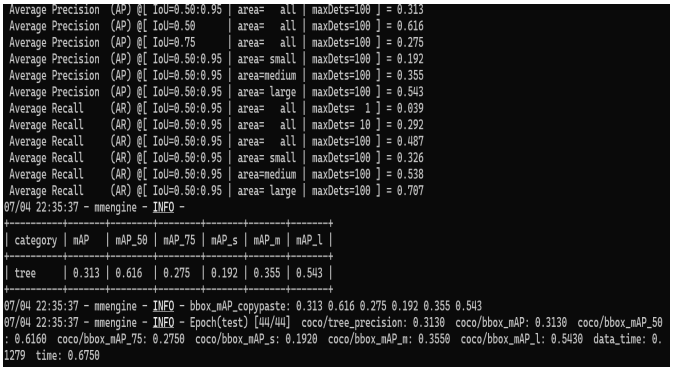

Fig. 4. Métricas de desempenho do Fold 2

Fold 3

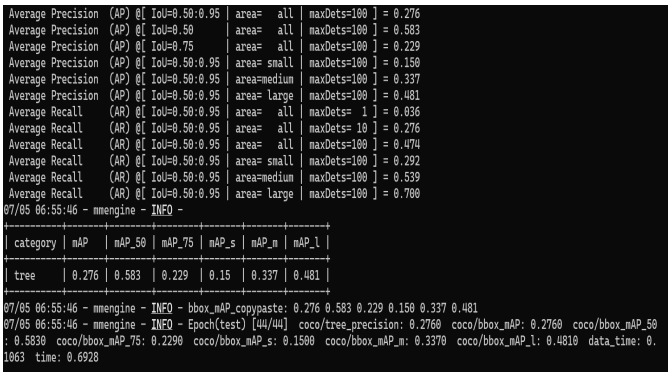

Fig. 5. Métricas de desempenho do Fold 3

Fold 4

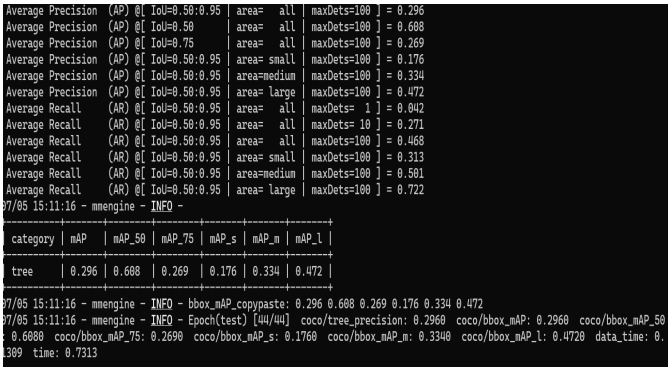

Fig. 6. Métricas de desempenho do Fold 4

Após a análise individual dos cinco folds da validação cruzada, serão considerados os valores obtidos para cada uma das 12 métricas avaliadas. Em seguida, será calculada a média aritmética de cada métrica, permitindo uma estimativa mais representativa e estável da performance geral do modelo YOLO-MS. É possível perceber que as métricas seguem um padrão semelhante entre si, com variações em parâmetros como o IoU (Intersection over Union), que define o grau mínimo de sobreposição entre a caixa predita e a real para considerar uma detecção válida. Além disso, outros fatores como o área (indicando se o objeto é pequeno, médio ou grande) e o maxDets (número máximo de detecções consideradas por imagem) influenciam diretamente nos valores reportados.

- Average Precision | IoU = 0.50 : 0.95 | area = all | maxDets= 100 => 0.2826
- Average Precision | IoU = 0.50 | area = all| maxDets = 100 => 0.5842
- Average Precision | IoU = 0.75 | area = all | maxDets = 100 => 0.2392
- Average Precision | IoU = 0.50 : 0.95 | area = small | maxDets = 100 => 0.1688
- Average Precision | IoU = 0.50 : 0.95 | area = mediuml | maxDets = 100 => 0.3272
- Average Precision | IoU = 0.50 : 0.95 | area = large | maxDets = 100 => 0.4472
- Average Recall | IoU = 0.50 : 0.95 | area = all | maxDets = 1 => 0.0386
- Average Recall | IoU = 0.50 : 0.95 | area = all | maxDets = 10 => 0.2710
- Average Recall | IoU = 0.50 : 0.95 | area = all | maxDets = 100 => 0.4646
- Average Recall | IoU = 0.50 : 0.95 | area = small | maxDets = 100 => 0.3008
- Average Recall | IoU = 0.50 : 0.95 | area = medium | maxDets = 100 => 0.5164
- Average Recall | IoU = 0.50 : 0.95 | area = large | maxDets = 100 => 0.6632

Observação: para uma melhor visualização das imagens, entre na pasta *midia* do repositório

https://github.com/DaviHVL/Yolo-MS-para-deteccao-de-arvores.

VI. Discussão

VI.I. Análise Comparativa de Desempenho

Levando em consideração que os modelos apresentados no artigo de referência foram avaliados com base na métrica AP50 (Average Precision com limiar de IoU ≥ 0.50), optamos por utilizar este mesmo indicador para garantir uma comparação justa e coerente entre os métodos. Nesse contexto, o modelo YOLO-MS, empregado neste projeto, obteve uma precisão média de **0,5842**, calculada a partir da validação cruzada em cinco folds. Este valor representa o desempenho médio do modelo na tarefa de detecção de copas de árvores em imagens RGB de alta resolução.

Quando comparado aos resultados dos modelos analisados por Zamboni et al. (2021), observa-se que o YOLO-MS apresenta um desempenho inferior à média geral dos métodos testados. Seu valor de AP50 aproxima-se do obtido pelo YOLOv3 (**0,591**), uma arquitetura de referência entre os modelos de detecção de uma etapa (anchor-based one-stage), e fica consideravelmente abaixo do RetinaNet (**0,650**), outro modelo clássico muito utilizado em aplicações de sensoriamento remoto.

A comparação torna-se ainda mais expressiva ao analisarmos modelos mais recentes e sofisticados. O YOLO-MS, embora seja uma evolução na família YOLO por sua capacidade de aprendizado de representações multiescala, encontra-se em uma categoria de detectores de um estágio. No estudo de referência, os modelos de melhor desempenho, como o **FSAF** (Feature Selective Anchor-Free), com um AP50 de **0,701**, e o **Double Heads**, com **0,699**, pertencem a classes de arquiteturas distintas que oferecem vantagens intrínsecas para o desafio em questão.

O FSAF, um detector livre de âncoras (*anchor-free*), elimina a necessidade de caixas de ancoragem pré-definidas, permitindo uma seleção de características mais flexível e dinâmica, o que é particularmente vantajoso para detectar objetos com grande variação de escala e forma, como as copas de árvores. Já o Double Heads, um representante dos detectores de dois estágios, utiliza cabeças de predição separadas para a classificação e a regressão da caixa delimitadora. Essa abordagem desacoplada tem se mostrado eficaz para resolver o desalinhamento espacial entre as duas tarefas, resultando em uma localização mais precisa dos objetos, especialmente em cenários densos e com sobreposição, como é o caso de florestas urbanas.

Essas arquiteturas refletem avanços técnicos importantes, como mecanismos de atenção, aprendizado livre de âncoras e estratégias de refinamento multiestágio, que contribuem diretamente para a elevação da precisão na detecção de objetos complexos em ambientes urbanos — como é o caso das árvores com sobreposição de copas, variações de iluminação e densidade heterogênea do cenário.

VI.II. Potencial de melhoria do YOLO-MS

Apesar do resultado inicial, o desempenho do YOLO-MS pode ser significativamente aprimorado. A performance de um modelo de deep learning não depende apenas de sua arquitetura, mas também de um ajuste fino de seus hiperparâmetros para o conjunto de dados específico. A seguir, detalhamos um plano de otimização focado no ajuste de hiperparâmetros e em outras técnicas avançadas.

A busca sistemática pelos melhores hiperparâmetros é um passo crucial para maximizar a performance do YOLO-MS no dataset de copas de árvores. Recomenda-se a utilização de técnicas de otimização automatizada, como algoritmos genéticos ou a busca Bayesiana, para explorar o espaço de hiperparâmetros de forma eficiente. Os principais hiperparâmetros a serem ajustados incluem:

- **Número de Épocas de Treinamento:** O treinamento por um número maior de épocas pode permitir que o modelo convirja para uma solução mais ótima. É fundamental monitorar a curva de perda na validação para evitar o sobreajuste (*overfitting*), utilizando técnicas como o *early stopping*.
- **Taxa de Aprendizagem (*Learning Rate*):** Este é um dos hiperparâmetros mais críticos. Uma taxa de aprendizagem inicial (geralmente `lr0`) muito alta pode fazer com que o modelo divirja, enquanto uma taxa muito baixa pode levar a uma convergência extremamente lenta. A utilização de um agendador da taxa de aprendizagem (*learning rate scheduler*), como o "Cosine Annealing", pode ajudar o modelo a escapar de mínimos locais e a encontrar melhores soluções.
- **Tamanho do Lote (*Batch Size*):** O tamanho do lote de imagens processadas por iteração influencia a estabilidade do treinamento e a utilização da memória da GPU. Lotes maiores podem levar a uma estimativa mais estável do gradiente, mas requerem mais recursos computacionais. É importante encontrar um equilíbrio que maximize a eficiência do treinamento.
- **Otimizador e seus Parâmetros:** Embora o Adam e o SGD com momento sejam otimizadores comuns, experimentar diferentes algoritmos e seus respectivos parâmetros (como momento e decaimento de peso - *weight decay*) pode levar a ganhos de desempenho.

Ao implementar de forma sistemática essas melhorias, é esperado que o desempenho do YOLO-MS no dataset de copas de árvores melhore substancialmente, reduzindo a diferença em relação aos modelos de melhor desempenho e potencialmente superando outros detectores de um estágio, como o RetinaNet.

VII. Conclusão

O presente trabalho demonstrou a aplicação da arquitetura YOLO-MS para a tarefa de detecção de copas de árvores individuais em imagens urbanas de alta resolução. A partir do treinamento com a base de dados utilizada por Zamboni et al. (2021), obtivemos um desempenho satisfatório com Average Precision (AP50) de 0.5842, o que representa uma performance competitiva em relação aos métodos avaliados no benchmark original, cujo melhor modelo reportado atingiu AP50 de 0.686.

Apesar de não alcançar o melhor resultado absoluto, o YOLO-MS se destacou pela sua capacidade de realizar detecção em tempo real, aliando precisão e eficiência. Isso o torna uma excelente alternativa para aplicações práticas de monitoramento ambiental urbano, como mapeamento de áreas verdes e planejamento urbano sustentável.

Adicionalmente, os resultados de AP e AR segmentados por tamanho de objeto indicam que o modelo teve desempenho expressivo em árvores de tamanho médio e grande, o que é coerente com o padrão de copas encontrado em ambientes urbanos.

Como trabalho futuro, propõe-se:

- Ajustar hiperparâmetros para melhorar a precisão em árvores pequenas;
- Explorar técnicas de aumento de dados e treinamento semi-supervisionado;
- Comparar outras variantes modernas do YOLO com o mesmo conjunto de dados;
- Integrar os resultados com dados geoespaciais para geração de mapas urbanos inteligentes.

Conclui-se que o YOLO-MS é uma solução promissora e prática para detecção de árvores em imagens aéreas, com potencial de aprimoramento e expansão para outras aplicações ambientais. Implementamos e avaliamos o modelo YOLO-MS para a tarefa de detecção de copas de árvores urbanas. Os resultados demonstram sua viabilidade para aplicações de monitoramento urbano em tempo real. Trabalhos futuros incluem o teste de variantes do YOLO-MS e integração com sistemas de georreferenciamento.

VIII. *References*

O desenvolvimento deste projeto foi fundamentado em estudos recentes sobre detecção de objetos aplicados à identificação de copas de árvores em imagens aéreas de alta resolução. Foram considerados trabalhos que avaliam algoritmos de última geração, métricas de desempenho amplamente aceitas pela comunidade científica e práticas consolidadas em visão computacional. A seguir, apresentam-se as principais referências utilizadas ao longo da implementação e análise comparativa dos resultados obtidos.

- https://ieee-sensorsalert.org/articles/ms-yolo-object-detection-based-on-yolov5-optimized-fusion-millimeter-wave-radar-and-machine-vision/?utm [1]
- https://www.mdpi.com/1424-8220/24/21/6955?utm [2]
- https://www.mdpi.com/1424-8220/24/14/4569?utm [3]
- https://link.springer.com/article/10.1007/s12145-024-01265-y?utm [4]
- https://github.com/FishAndWasabi/YOLO-MS [5]
- https://github.com/pedrozamboni/individual_urban_tree_crown_detection.git [6]
- https://journals.ieeeauthorcenter.ieee.org/create-your-ieee-journal-article/create-graphics-for-your-article/file-formatting/ [7]
- https://arxiv.org/abs/2308.05480?utm [8]