INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Desenvolvimento de um Sistema de Visão Computacional para a Deteção e Monitorização de Infestações de Vespa-das-Galhas-do-Castanheiro em Castanheiros Infetados

Ricardo Manuel Góis Bento - a26182

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Engenharia Eletrotécnica e de Computadores.

Trabalho realizado sob a orientação de:
Professor Doutor José Fernando Lopes Barbosa

Bragança
2024

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Desenvolvimento de um Sistema de Visão Computacional para a Deteção e Monitorização de Infestações de Vespa-das-Galhas-do-Castanheiro em Castanheiros Infetados

Ricardo Manuel Góis Bento - a26182

Dissertação apresentada à Escola Superior de Tecnologia e de Gestão de Bragança para obtenção do Grau de Mestre em Engenharia Eletrotécnica e de Computadores.

Trabalho realizado sob a orientação de:
Professor Doutor José Fernando Lopes Barbosa

Bragança
2024

Dedicatória

Para a minha tia Maria Bernardete Barreira Bento, com todo o meu amor e carinho.

Agradecimentos

Em primeiro lugar, queria agradecer ao meu orientador, Professor Doutor José Fernando Lopes Barbosa, por toda a dedicação, apoio, conhecimento transmitido, motivação e disponibilidade, mas acima de tudo, pela paciência e confiança demonstrada ao longo da minha formação académica, especialmente este último ano académico.

Queria agradecer a todos os meus amigos e colegas de trabalho que direta ou indiretamente contribuíram para o desenvolvimento desta dissertação, através de conhecimentos, e sobretudo apoio e motivação.

E por último à minha família, em especial ao meu pai e sobretudo à minha tia Maria Bernardete que sempre me ajudou e acreditou em mim e nas minhas capacidades, sem ela nada disto teria sido possível e o meu futuro académico e profissional teria sido muito diferente daquilo que é hoje em dia.

A todos vocês, um muito obrigado do fundo do meu coração. É incrível poder partilhar esta vida convosco.

Resumo

A doença da vespa-das-galhas-do-castanheiro (Dryocosmus kuriphilus) constitui uma das maiores ameaças à produção de castanha, afetando significativamente a saúde e produtividade dos castanheiros. Esta praga provoca deformações nas folhas e ramos dos castanheiros, resultando no enfraquecimento da árvore e numa diminuição considerável da produção de castanha. A identificação precoce e o combate eficiente à vespa-das-galhas são cruciais para minimizar os prejuízos e garantir a sustentabilidade da produção de castanha.

Este trabalho propõe um sistema de deteção automática da presença infeções nas galhas, utilizando técnicas de visão computacional, que visam facilitar a identificação precoce da praga e, assim, melhorar a eficácia das ações de controlo. O sistema foi desenvolvido com recurso a *Convolutional Neural network* (CNN) e ao modelo YOLOv8.2, treinado com um *dataset* composto por imagens anotadas de galhas de castanheiro. Este *dataset* foi construído e aumentado através de técnicas de *data augmentation* para melhorar a capacidade de generalização do modelo e aumentar a robustez do sistema perante diferentes condições de iluminação e ângulos de captura.

A deteção das galhas infetadas é realizada através da análise de imagens capturadas por um drone, com uma velocidade controlada de forma a garantir a captura de detalhes suficientes para uma deteção precisa, ou outro meio de captação de imagens de alta definição. Esta abordagem permite realizar um rastreio eficiente e de baixo custo, cobrindo grandes áreas em menos tempo e com menor intervenção humana.

Os resultados obtidos demonstram que o sistema desenvolvido tem potencial para ser utilizado numa fase de pré-análise, oferecendo apoio à decisão para os produtores e cooperativas. Este sistema auxilia no combate à vespa-das-galhas-do-castanheiro, permitindo uma identificação precoce e uma intervenção direcionada, contribuindo para a redução dos danos causados pela praga. Apesar de ainda não estar totalmente maduro para aplicação final, o sistema mostra-se promissor na proteção das produções de castanha.

Palavras-chave: vespa-das-galhas-do-castanheiro, deteção automática, visão computacional, YOLOv8.2, *Convolutional Neural network*, drone, rastreio de pragas

Abstract

The chestnut gall wasp disease (Dryocosmus kuriphilus) is one of the greatest threats to chestnut production, significantly affecting the health and productivity of chestnut trees. This pest causes deformations in the leaves and branches of chestnut trees, resulting in tree weakening and a considerable reduction in chestnut yield. Early identification and efficient control of the gall wasp are crucial to minimize damage and ensure the sustainability of chestnut production.

This work proposes an automatic detection system for the presence of infections in galls using computer vision techniques, aimed at facilitating the early identification of the pest and thus improving the effectiveness of control actions. The system was developed using Convolutional Neural Networks (CNN) and the YOLOv8.2 model, trained with a dataset composed of annotated images of chestnut galls. This dataset was constructed and augmented through data augmentation techniques to improve the model's generalization capability and increase the system's robustness under different lighting conditions and capture angles.

The detection of infected galls is performed through the analysis of images captured by a drone, with a controlled speed to ensure the capture of sufficient details for accurate detection, or by other high-definition imaging means. This approach allows for efficient and cost-effective screening, covering large areas in less time and with minimal human intervention.

The results obtained demonstrate that the developed system has the potential to be used in a pre-analysis phase, providing decision support to producers and cooperatives. This system assists in combating the chestnut gall wasp, enabling early identification and targeted intervention, contributing to reducing the damage caused by the pest. Although it is not yet fully mature for final application, the system shows promise in protecting chestnut production.

Keywords: chestnut gall wasp, automatic detection, computer vision, YOLOv8.2, Convolutional Neural Network, drone, pest monitoring.

Conteúdo

Lista de Figuras

Lista de Abreviações

YOLO - *You Only Look Once*

IA - Inteligência Artificial

RGB - *Red Green Blue*

CNNs - *Convolutional Neural network*

VANTs - Veículos Aéreos Não Tripulados

ML - *Machine Learning*

ANN - Artificial Neural Network

SGD - *Stochastic* Gradient Descent

CNNs - Convolutional Neural Network

GPU - Unidades de Processamento Gráfico

TPUs - *Tensor Processing Units*

1 Introdução

1.1 Contextualização

A produção de castanhas é uma atividade agrícola de extrema importância, não só no contexto económico, mas também nos âmbitos social e ambiental em várias regiões do mundo, com destaque especial na Europa. Em países como Portugal, Espanha, Itália e França, os castanheiros (Castanea spp.) são cultivados não apenas pelo valor comercial das castanhas, mas também pelo importante papel cultural e ecológico que desempenham. Os castanheiros são árvores de vida longa que contribuem significativamente para a estabilização do solo, manutenção da biodiversidade e preservação das paisagens rurais, tornando-se assim pilares fundamentais nas regiões onde são encontrados.

No entanto, esta atividade enfrenta desafios substanciais devido à presença de diversas pragas e doenças, sendo a vespa-das-galhas-do-castanheiro (*Dryocosmus kuriphilus*) uma das ameaças mais preocupantes. Originária da China, esta praga foi introduzida acidentalmente na Europa, onde se espalhou rapidamente, causando danos severos às plantações de castanheiros. Desde a sua primeira identificação na China, em 1941, a vespa-das-galhas-do-castanheiro tornou-se uma das pragas mais destrutivas para os castanheiros em várias partes do mundo. Na Europa, a sua presença foi relatada pela primeira vez em Itália, em 2002, e desde então alastrou-se a outros países, incluindo França, Espanha, Portugal, Suíça, Eslovénia, Croácia e Grécia. A rápida disseminação da vespa deve-se, em grande parte, ao transporte de material vegetal infetado e à dispersão natural dos insetos adultos[1].

Em Portugal, a vespa-das-galhas-do-castanheiro foi detetada pela primeira vez em 2014, na região do Minho, e rapidamente se espalhou para outras áreas, como Trás-os-Montes, uma das principais regiões produtoras de castanha no país. Vinhais, um dos maiores concelhos produtores, tem sido severamente afetado, o que gera grande preocupação entre os agricultores locais devido aos impactos significativos que a praga pode ter sobre as plantações[2].

A infestação pela vespa-das-galhas-do-castanheiro não só reduz a produção de castanhas, mas também prejudica o crescimento saudável das árvores, resultando em perdas económicas significativas para os produtores. Do ponto de vista ecológico, a vespa ameaça a biodiversidade local, uma vez que as galhas formadas pelas larvas comprometem a capacidade

das árvores de realizar fotossíntese e tornam-nas mais suscetíveis a outras pragas e doenças. Além disso, a morte de castanheiros em decorrência de infestações severas pode impactar negativamente o ecossistema local[3].

Os métodos tradicionais de deteção de infestações, que dependem principalmente de inspeções visuais por técnicos ou agricultores, são demorados e laboriosos, além de dependerem da experiência do observador. Tais métodos muitas vezes não detetam infestações nos estágios iniciais, quando o controlo seria mais eficaz. Enquanto métodos biológicos e químicos de controlo da praga, como a introdução de inimigos naturais e o uso de inseticidas, são utilizados, cada um apresenta limitações e desafios significativos, destacando a necessidade urgente de soluções inovadoras [3].

Para enfrentar esses desafios, a utilização de sistemas avançados de visão por computador, como o YOLO (*You Only Look Once*), tem-se mostrado promissora. YOLO é um sistema de deteção de objetos em tempo real que revolucionou o campo da visão computacional pela sua capacidade de detetar múltiplos objetos em imagens e vídeos com elevada rapidez e precisão [4]. Este sistema baseia-se em redes neuronais convolucionais que processam imagens em tempo real, permitindo a identificação precisa e eficiente de padrões complexos, como as galhas infetadas pela vespa-das-galhas-do-castanheiro.

O YOLO funciona dividindo uma imagem em regiões e, simultaneamente, prediz as probabilidades de diferentes classes de objetos, juntamente com *bounding boxes*, para cada região. A abordagem única de YOLO permite uma deteção extremamente rápida, tornando-o ideal para aplicações em tempo real, como na agricultura de precisão, onde é crucial monitorizar grandes áreas de cultivo de forma eficiente e eficaz [5].

A aplicação da visão artificial na agricultura, e particularmente na deteção de pragas e doenças, tem crescido exponencialmente, facilitada por tecnologias como drones e sistemas de inteligência artificial (IA). Drones equipados com câmaras de alta resolução podem capturar imagens de grandes áreas agrícolas, que são posteriormente analisadas por sistemas como o YOLO, para detetar a presença de pragas, doenças ou outros fatores que possam impactar a saúde das culturas [6]. Estas tecnologias proporcionam aos agricultores uma ferramenta poderosa para a monitorização das suas culturas, permitindo a tomada de decisões mais informadas e a implementação de medidas de controlo mais eficazes.

Este trabalho procura explorar o potencial destas tecnologias para desenvolver um sistema de deteção precoce, acessível e eficiente, que contribua para a sustentabilidade da produção de castanhas e promova práticas agrícolas mais modernas e eficazes. Através desta

abordagem inovadora, espera-se proteger uma cultura vital para regiões como Trás-os-Montes e assegurar o futuro da produção de castanhas em Portugal.

1.2 Importância do estudo

1.2.1 Inovação Tecnológica na Agricultura: Drones e Inteligência Artificial

A agricultura moderna enfrenta desafios significativos relacionados à sustentabilidade e à necessidade de aumentar a produtividade, minimizando simultaneamente os impactos ambientais. Neste contexto, a tecnologia de drones combinada com a IA revela-se uma solução inovadora, sobretudo na deteção precoce e na gestão de pragas. Este capítulo explora como essa combinação tecnológica não só melhora a eficiência operacional como também contribui para a redução de custos, apoiado por estudos relevantes na área.

Os drones, equipados com câmaras de alta resolução e sensores avançados, proporcionam uma visão abrangente e detalhada das culturas, que seria difícil ou mesmo impossível de obter através de métodos convencionais. Drones com câmaras RGB (*Red Green Blue*) e sensores infravermelhos são capazes de identificar variações sutis na cor e saúde das plantas, permitindo a deteção precoce de pragas e doenças antes de serem visíveis ao olho humano. A capacidade de cobrir rapidamente grandes áreas e realizar voos frequentes assegura uma vigilância contínua e atualizada das condições das culturas, facilitando intervenções precisas e oportunas[7].

A utilização de drones também reduz significativamente os custos associados à monitorização de extensas áreas agrícolas. Estudos demonstram que o uso de drones pode diminuir os custos de monitorização em até 60% em comparação com os métodos terrestres tradicionais. Esta redução deve-se não apenas à diminuição da necessidade de mão-de-obra, mas também à eficiência com que os drones podem coletar dados e aplicar tratamentos. Por exemplo, drones podem ser utilizados para aplicar pesticidas de forma direcionada, reduzindo o volume necessário de produtos químicos e, consequentemente, os custos e o impacto ambiental [8].

A integração de IA nos sistemas de drones eleva ainda mais a capacidade de diagnóstico. Modelos de aprendizagem automática, como as *Convolutional Neural network* (CNNs ou *ConvNet*) usadas em algoritmos de deteção de objetos, podem identificar e classificar

problemas nas plantações com alta precisão. Outros estudos, relatam que algoritmos de IA podem diferenciar entre diferentes tipos de stress biótico e abiótico com uma precisão superior a 90%, permitindo não apenas uma resposta rápida, mas também uma utilização mais racional de inseticidas, fungicidas e outros insumos agrícolas, baseando-se na identificação precisa do tipo de praga ou doença presente [9].

A longo prazo, a adoção de drones e IA na agricultura tem o potencial de transformar as práticas agrícolas para um modelo mais sustentável. A implementação de tecnologias inteligentes pode contribuir para a agricultura de precisão, que maximiza os rendimentos enquanto minimiza os danos ambientais, sendo crucial para o futuro da segurança alimentar global e a conservação dos recursos naturais [10].

Para concluir, a integração de drones e IA no tratamento de pragas na agricultura não é apenas uma tendência tecnológica, mas uma necessidade emergente para enfrentar os desafios globais de produção de alimentos de maneira mais sustentável. Com estudos que já demonstram vários benefícios significativos, esta tecnologia promete ser um pilar fundamental na agricultura do futuro.

1.3 Motivação Pessoal e Contributo à Comunidade Local

Sendo natural de Trás-os-Montes, mais especificamente do concelho de Vinhais, a minha ligação com a produção de castanhas e a vida rural é bastante profunda. Vinhais é um dos principais concelhos produtores de castanha em Portugal, e a propagação da vespa-das-galhas-do-castanheiro tem começado a afetar diretamente a economia local e a subsistência dos agricultores da minha comunidade. Crescendo nesta região, tive a oportunidade de observar de perto os desafios enfrentados pelos produtores de castanha devido a esta e outras pragas devastadoras.

A motivação para o meu projeto de mestrado é, em grande medida, de cariz pessoal. Tenho o desejo de contribuir de forma significativa para a minha comunidade, desenvolvendo uma solução prática e eficaz que possa atenuar os impactos das infestações de vespa-das-galhas-do-castanheiro. Com base no potencial de tecnologias avançadas, como drones e IA, encaro esta investigação como uma oportunidade para trazer inovação e renovada esperança aos produtores de castanha em Vinhais e de outras regiões afetadas.

Estou convicto de que, ao implementar um sistema de deteção precoce e preciso, poderei não só ajudar a preservar a produção de castanhas, mas também promover práticas agrícolas mais sustentáveis e eficientes. Esta é a minha forma de retribuir à comunidade, utilizando os

conhecimentos adquiridos na minha licenciatura em Engenharia Eletrotécnica e de Computadores e agora também no Mestrado em Engenharia Eletrotécnica e de Computadores, para resolver um problema real e premente que afeta a vida de muitos agricultores na minha região.

1.4 Objetivos

O objetivo geral deste trabalho é desenvolver e analisar a viabilidade um sistema automatizado de baixo custo para a identificação de infestações de vespa-das-galhas-do-castanheiro (Dryocosmus kuriphilus) em castanheiros, utilizando, para a captação de imagem, drones equipados com câmaras ou outros dispositivos de captação de imagem e modelos de deteção de objetos em tempo real, contribuindo assim para uma gestão mais eficiente e sustentável das plantações de castanheiros:

1. **Desenvolver um sistema de baixo custo**: Implementar uma solução acessível para os agricultores, utilizando drones equipados com câmaras ou outro dispositivo de captação de imagem e tecnologias de código aberto.
2. **Implementar e treinar um modelo de deteção de objetos em tempo real**: Utilizar o YOLOv8 ou outro modelo adequado para reconhecer galhas infetadas pela vespa do castanheiro, garantindo alta precisão e eficiência.
3. **Avaliar a precisão e eficácia do sistema**: Realizar testes comparativos entre o sistema proposto e os métodos tradicionais de deteção, ajustando o modelo conforme necessário para melhorar o desempenho.
4. **Analisar a viabilidade de diferentes dispositivos de captura de imagem**: Comparar a eficácia de drones, câmaras instaladas em tratores e telemóveis em termos de qualidade de imagem, custo e praticidade.
5. **Viabilidade de um modelo de negócio:** Avaliar possíveis modelos de negócio de forma a rentabilizar a ideia e otimizar o produto.

2 Estado da Arte

Neste capítulo vai ser discutida a utilização dos vários métodos tradicionais e a inovação do estado de arte na deteção de pragas relacionado com a agricultura.

2.1 Métodos Tradicionais de Monitorização de Pragas

Os métodos tradicionais de monitorização de pragas têm sido amplamente utilizados na agricultura ao longo dos anos. Apesar das suas limitações, continuam a ser ferramentas essenciais, especialmente em contextos onde as soluções tecnológicas modernas ainda não estão plenamente implementadas. A seguir, vão ser abordados os principais métodos tradicionais, destacando as suas características, vantagens e desvantagens.

Inspeção Visual

A inspeção visual consiste na observação direta das plantas por agricultores ou técnicos especializados, que procuram identificar sinais de infestação, como a presença de insetos, danos nas folhas ou outros sintomas de doenças. Este método é particularmente acessível, pois não exige equipamentos sofisticados. No entanto, a sua eficácia está fortemente dependente da experiência e da atenção do observador, o que pode resultar em variabilidade nos resultados [11].

Este método é mais adequado para pequenas explorações agrícolas, onde a extensão da área a ser monitorizada é limitada. Em áreas de cultivo extensivas, a inspeção visual torna-se menos prática devido ao tempo e à mão-de-obra necessários [11]. Além disso, a deteção de pragas através da inspeção visual ocorre frequentemente numa fase em que os danos já são visíveis, o que pode significar que a praga já causou impactos significativos e irreversíveis na produção [11].

Uso de Armadilhas

As armadilhas, como as armadilhas de feromonas, são utilizadas para atrair e capturar insetos, permitindo uma monitorização mais precisa das populações de pragas. Este método é eficaz para a monitorização de pragas específicas, que respondem aos estímulos químicos das feromonas. As armadilhas fornecem dados quantitativos sobre a presença de pragas, o que é útil para tomar decisões informadas sobre o momento ideal para a aplicação de medidas de controlo [12].

No entanto, a eficácia das armadilhas pode ser afetada por fatores como o posicionamento, a densidade das armadilhas e as condições ambientais. Uma colocação inadequada pode levar a resultados que não refletem corretamente a realidade da infestação, comprometendo a eficácia das intervenções subsequentes. Além disso, as armadilhas requerem manutenção regular, o que implica um esforço contínuo por parte dos agricultores [12].

Amostragem de Solo e Planta

A amostragem de solo e planta envolve a coleta de amostras que são posteriormente analisadas em laboratório para deteção de pragas, ovos ou patógenos. Este método é particularmente útil para pragas que afetam as raízes das plantas ou que vivem no solo, sendo capaz de identificar infestações numa fase precoce [13].

Apesar da sua utilidade, a amostragem de solo e planta é um processo que exige tempo e recursos, especialmente em grandes áreas de cultivo. A representatividade das amostras é crucial para garantir que os resultados obtidos refletem a realidade do campo. A interpretação dos resultados também requer conhecimentos técnicos, o que pode ser uma limitação para alguns agricultores [13].

Observações Continuadas

As observações continuadas consistem no registo regular das condições das culturas e da presença de pragas ao longo do tempo. Este método permite identificar padrões de infestação e ajustar as práticas de gestão conforme necessário. É uma prática comum em sistemas de gestão integrado de pragas, onde se procura minimizar o uso de pesticidas através de uma monitorização contínua [12].

A eficácia das observações continuadas depende da consistência dos registos e da capacidade do agricultor ou técnico em interpretar os dados corretamente. Embora este método seja valioso, especialmente em conjunto com outras abordagens, a sua eficácia pode ser limitada

pela variabilidade nas condições de campo e pela subjetividade inerente ao processo de observação [12].

2.1.1 Vantagens e Limitações dos Métodos Tradicionais

Os métodos tradicionais de monitorização de pragas oferecem várias vantagens que justificam a sua ampla adoção, mas também apresentam limitações que podem comprometer a eficácia da gestão de pragas em algumas situações.

Vantagens:

- **Custo inicial reduzido**: Muitos dos métodos tradicionais, como a inspeção visual e o uso de armadilhas, não requerem grandes investimentos em equipamentos tecnológicos avançados, tornando-os acessíveis para a maioria dos agricultores [11][12].
- **Facilidade de implementação**: Estes métodos são relativamente fáceis de implementar e não exigem conhecimentos técnicos especializados, o que facilita a sua adoção em diferentes contextos agrícolas [11][12].
- **Adaptabilidade**: Podem ser aplicados em diversas culturas e condições ambientais, sendo métodos versáteis e flexíveis [13][12].

Limitações:

- **Subjetividade e variabilidade**: A eficácia da inspeção visual e das observações continuadas pode variar significativamente dependendo da experiência do observador, o que pode levar a inconsistências nos resultados [11].
- **Elevada exigência de tempo e mão-de-obra**: Muitos destes métodos requerem uma quantidade significativa de tempo e esforço humano, especialmente em grandes áreas de cultivo, o que pode limitar a sua eficácia em explorações agrícolas maiores [11].
- **Deteção tardia**: Em muitos casos, os métodos tradicionais só permitem a deteção de pragas quando os danos já são visíveis, limitando as opções de intervenção e potencialmente aumentando os custos de controlo [12][13].

Em resumo, os métodos tradicionais de monitorização de pragas, como a inspeção visual, o uso de armadilhas, a amostragem de solo e planta, e as observações continuadas, têm desempenhado um papel fundamental na gestão de pragas ao longo dos anos. No entanto,

apresentam limitações significativas, especialmente em termos de precisão e eficiência, que podem ser superadas através da integração com tecnologias mais modernas. A combinação destes métodos com soluções tecnológicas emergentes oferece um caminho promissor para uma gestão de pragas mais eficaz e sustentável.

2.2 Tecnologias Emergentes na Monitorização de Pragas

A monitorização de pragas é uma prática essencial para a preservação da saúde das culturas agrícolas, e a evolução tecnológica tem proporcionado novas ferramentas que revolucionam essa área. As tecnologias emergentes, como o uso de drones, a visão computacional e a IA, representam uma mudança significativa em relação aos métodos tradicionais, oferecem soluções mais eficazes, rápidas e precisas. A aquisição de imagem é um dos componentes mais importantes para a identificação de objetos, neste caso, mais precisamente de pragas. Neste capítulo, iremos analisar mais detalhadamente como essas inovações estão a ser aplicadas no campo da agricultura e as suas implicações para o futuro da monitorização de pragas.

2.2.1 Tipos de sistemas para deteção de pragas baseados em visão computacional

Sistemas de monitorização baseados em câmaras estáticas: Sistemas de câmaras estáticas são utilizados em estufas e campos agrícolas, capturando imagens de alta resolução ao longo do tempo para identificar anomalias que possam indicar doenças, como a descoloração das folhas ou manchas. Estes sistemas podem incluir câmaras RGB ou multiespectrais e são frequentemente associados a redes de sensores que recolhem dados ambientais como temperatura e humidade, fornecendo uma visão holística da saúde da planta [14].

Robôs agrícolas: Robôs agrícolas móveis equipados com câmaras avançadas percorrem os campos de forma autónoma ou semiautónoma, capturando imagens de alta precisão para identificar doenças nas plantas. Estes robôs utilizam câmaras RGB, multiespectrais ou até híper espectrais para obter informações detalhadas das culturas e podem incluir algoritmos de aprendizagem automática (*machine learning*) para deteção em tempo real. Um exemplo comum são os robôs utilizados em culturas de cana de açúcar para detetar doenças [15].

Figura 2.1- Exemplo de Robôs agrícolas equipados com câmaras[16]

Sistemas portáteis ou manuais: Dispositivos portáteis equipados com câmaras e aplicações de processamento de imagem permitem aos agricultores ou técnicos identificar doenças no terreno. Estes dispositivos podem ser usados com smartphones ou câmaras portáteis específicas e são uma solução de baixo custo, muito popular entre agricultores de pequena e média escala. Tecnologias de IA aplicadas a estas imagens permitem a identificação de doenças através de apps móveis [17].

Sistemas de análise por imagem com câmaras híper espectrais: As câmaras híper espectrais captam imagens em centenas de bandas espectrais, permitindo a deteção precoce de doenças através da identificação de mudanças subtis na composição química das folhas e tecidos das plantas. Estes sistemas são utilizados em estufas e campos experimentais para deteção de doenças fúngicas e virais antes de se manifestarem visualmente. A análise computacional avançada é essencial para interpretar os dados híper espectrais [18].

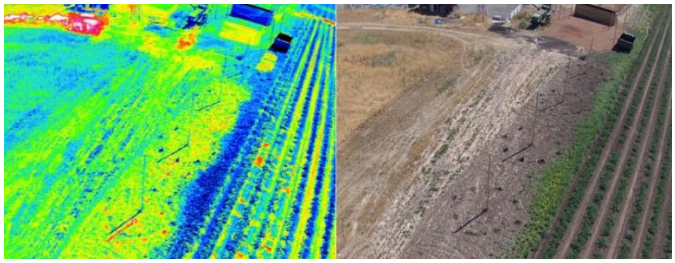


Figura 2.2- Sistemas de análise por imagem com câmaras híper espectrais [19]

Sistemas de vigilância por satélite: Satélites equipados com sensores de imagem multiespectral são capazes de monitorizar grandes áreas de cultivo, detetando padrões associados a doenças, como mudanças na refletividade das folhas. Estes sistemas são particularmente úteis em culturas de grande escala, como trigo e milho, permitindo a monitorização contínua em regiões de difícil acesso [20].

Sistemas baseados em câmaras térmicas: Câmaras térmicas captam a luz infravermelha e detetam variações de temperatura nas plantas, um sinal precoce de doenças ou stress hídrico. As plantas infetadas tendem a exibir áreas com temperaturas diferentes das

plantas saudáveis. Estas câmaras são utilizadas tanto em campos agrícolas como em estufas para monitorizar a saúde das plantas e prevenir infeções fúngicas ou bacterianas antes de os sintomas serem visíveis [21].

Sistemas de monitorização baseados em Drones: Os drones são conhecidos pela sua versatilidade, pois podem ser equipados com diferentes tipos de câmaras e alcançar zonas, muitas vezes inalcançáveis facilmente, este tema será abordado mais detalhadamente no subcapítulo 2.2.2 [22].

Figura 2.3- Exemplo de Sistemas baseados em câmaras térmicas e drones [23]

2.2.2 Uso de Drones

Os drones, também conhecidos como veículos aéreos não tripulados (VANTs), têm-se tornado uma ferramenta essencial na agricultura moderna, especialmente no contexto da monitorização de pragas. A capacidade de capturar imagens de alta resolução e dados precisos de grandes áreas de cultivo em tempo real faz dos drones uma solução altamente eficaz para a deteção precoce e gestão de pragas. Esta subsecção explora em detalhe as aplicações, benefícios, desafios e o futuro do uso de drones na agricultura, com foco particular na monitorização de pragas.

Os drones oferecem uma série de aplicações práticas e inovadoras na monitorização de pragas, transformando a forma como os agricultores abordam a proteção das suas culturas.

Uma das principais vantagens dos drones no contexto da agricultura é a sua capacidade de cobrir grandes áreas de forma eficiente. Esta característica é especialmente relevante em explorações agrícolas de grande dimensão, onde os métodos tradicionais de monitorização, como a inspeção manual, são frequentemente impraticáveis devido ao tempo e aos recursos necessários. Os drones permitem uma monitorização rápida e detalhada das culturas, capturando imagens de alta resolução que oferecem uma visão abrangente do estado das plantações. A rapidez com que os drones conseguem recolher dados é um dos seus maiores trunfos. Por exemplo, um drone equipado com uma câmara de alta resolução pode cobrir uma área de 100 hectares em aproximadamente uma hora, enquanto uma equipa de técnicos

precisaria de um dia inteiro para realizar a mesma tarefa manualmente. Esta eficiência operacional reduz significativamente os custos associados à monitorização e permite a deteção precoce de pragas, antes que estas possam causar danos generalizados nas culturas [22]. Esses valores de tempo aumentam substancialmente, quando falamos da deteção de doenças localizadas e de mais difícil acesso, como a vespa-das-galhas-do-castanheiro.

Além disso, os drones permitem uma cobertura uniforme e sistemática das áreas de cultivo. Graças à sua capacidade de voo programado, os drones podem seguir rotas predefinidas que garantem que toda a área de interesse seja monitorizada de forma homogénea. Este planeamento automatizado assegura que nenhuma parte da plantação é negligenciada, o que é crucial para a deteção eficaz de pragas que podem começar a infestar áreas específicas antes de se espalharem para outras partes da cultura [24]. Outro benefício significativo dos drones é a sua capacidade de realizar monitorização em tempo real. As imagens e os dados capturados podem ser transmitidos imediatamente para os operadores ou analisados por software especializado em terra, permitindo uma avaliação quase instantânea do estado das culturas. Esta capacidade é especialmente útil em situações que exigem uma resposta rápida, como a deteção de um surto de pragas que requer intervenção imediata.

Figura 2.4- Exemplo de drone a sobrevoar uma plantação [25]

Os drones também têm a vantagem de poderem monitorizar áreas de cultivo que seriam difíceis ou perigosas de aceder por métodos convencionais. Isto inclui terrenos acidentados, áreas densamente plantadas ou zonas onde a presença de água ou vegetação alta torna a inspeção manual impraticável. A flexibilidade dos drones em alcançar estas áreas garante que todas as partes da plantação sejam monitorizadas adequadamente, independentemente das dificuldades do terreno. Além disso, os drones são capazes de voar a baixas altitudes, permitindo a captura de imagens detalhadas que revelam o estado das culturas com maior precisão do que seria possível a partir de um avião ou satélite[22].

A capacidade dos drones de capturar dados em grande escala é ainda complementada pela sua integração com software de análise de dados e mapeamento. As imagens capturadas pelos drones podem ser processadas para gerar mapas detalhados que mostram a distribuição de pragas, o estado de saúde das plantas e outras métricas importantes para a gestão das culturas. Esta integração com ferramentas de análise permite uma gestão mais eficiente e sustentável das plantações, já que os agricultores podem concentrar os seus esforços de controlo onde são mais necessários, reduzindo o uso de pesticidas e minimizando o impacto ambiental [24].

Em resumo, a capacidade dos drones de cobrir grandes áreas de forma eficiente representa uma revolução na monitorização de pragas. A rapidez, precisão e abrangência que oferecem superam largamente as capacidades dos métodos tradicionais, permitindo uma gestão mais eficaz das pragas e uma maior produtividade agrícola. Contudo, a adoção generalizada desta tecnologia ainda enfrenta desafios, como os custos iniciais e a necessidade de formação especializada. À medida que a tecnologia avança e se torna mais acessível, espera-se que os drones se tornem uma ferramenta padrão na agricultura global [22].

2.2.2.1 Vantagens do Uso de Drones

O uso de drones na monitorização de pragas traz várias vantagens que têm impulsionado a sua adoção na agricultura:

Precisão e Detalhe: Os drones capturam imagens de alta resolução, permitindo a identificação de pragas e sintomas de doenças com grande precisão. Esta capacidade de deteção detalhada é particularmente importante para a identificação de pragas em estágios iniciais de infestação, quando as medidas de controlo podem ser mais eficazes [22].

Eficiência Operacional: Em comparação com os métodos tradicionais, que exigem mão-de-obra intensiva, os drones permitem uma monitorização mais rápida e eficiente. Um único voo de drone pode cobrir uma vasta área de cultivo, reduzindo significativamente o tempo e os recursos necessários para inspecionar manualmente as culturas [24].

Redução de Custos a Longo Prazo: Embora o investimento inicial em drones e os seus sistemas associados possa ser elevado, a redução dos custos com mão-de-obra e a minimização das perdas de produção devido à deteção precoce de pragas pode justificar o investimento a longo prazo. Além disso, a capacidade de aplicar intervenções de forma precisa e localizada pode reduzir a quantidade de pesticidas necessários, resultando em economia de custos e menor impacto ambiental [8].

Acesso a Áreas de Difícil Alcance: Os drones podem alcançar áreas de cultivo que seriam difíceis ou perigosas de inspecionar manualmente, como terrenos acidentados ou zonas densamente plantadas. Esta flexibilidade permite uma cobertura completa da área de cultivo, garantindo que nenhuma zona é deixada sem monitorização [8].

2.2.2.2 Desafios e Limitações

Apesar das vantagens significativas, o uso de drones na monitorização de pragas enfrenta alguns desafios e limitações que precisam ser considerados:

Custo de Aquisição e Manutenção: O custo inicial dos drones, especialmente dos modelos equipados com sensores avançados, pode ser proibitivo para pequenos agricultores. Além disso, os drones requerem manutenção regular, incluindo a substituição de peças, atualização de software e reparos ocasionais, o que pode aumentar os custos operacionais [8].

Dependência das Condições Meteorológicas: O uso de drones é altamente dependente das condições meteorológicas. Ventos fortes, chuva, nevoeiro ou temperaturas extremas podem limitar ou impedir o voo dos drones, comprometendo a recolha de dados em momentos críticos [22]. Esta limitação pode ser particularmente problemática em regiões onde as condições meteorológicas são voláteis e imprevisíveis.

Regulamentação e Complexidade Operacional: O uso de drones está sujeito a regulamentações rigorosas, que variam de acordo com o país e a região. Estas regulamentações podem incluir restrições de voo, requisitos de licenciamento e limites de altitude, que podem complicar a operação de drones em áreas agrícolas. Além disso, a operação eficaz de drones requer conhecimentos técnicos especializados, incluindo a habilidade de planear missões de voo, interpretar dados de sensores e realizar manutenções básicas [22].

2.2.3 Inteligência Artificial

A Inteligência Artificial (IA) é uma área interdisciplinar da ciência da computação que se concentra na criação de sistemas capazes de executar tarefas que normalmente exigem inteligência humana. Estas tarefas incluem reconhecimento de padrões, tomada de decisões, processamento de linguagem natural, visão computacional e previsão de eventos. O conceito de IA remonta à década de 1950, quando matemáticos e cientistas, como Alan Turing, começaram a explorar a possibilidade de máquinas que poderiam "pensar". Turing propôs o

famoso "Teste de Turing" em 1950, que definiu um padrão para determinar se uma máquina poderia exibir comportamento inteligente indistinguível do humano [26].

A partir da conferência de Dartmouth em 1956, organizada por John McCarthy, Marvin Minsky, Nathaniel Rochester e Claude Shannon, o termo "Inteligência Artificial" foi formalmente estabelecido como uma nova disciplina académica [27]. Desde então, a IA evoluiu por meio de diferentes eras, incluindo os chamados "verões da IA", períodos de intenso desenvolvimento e entusiasmo, e os "invernos da IA", períodos de menor progresso e desilusão com os resultados alcançados.

2.2.4 Aprendizagem de Máquina (*Machine Learning*)

A aprendizagem de máquina, ou *machine learning* (ML), é uma subárea da IA que se foca no desenvolvimento de algoritmos que permitem que os sistemas aprendam a partir de dados e façam previsões ou tomem decisões sem serem explicitamente programados para realizar essas tarefas [28].

As tecnologias baseadas em ML estão cada vez mais presentes em diversas áreas do nosso dia a dia. Um exemplo notável são os assistentes virtuais, como Siri, Alexa e Google Assistant, que utilizam essa tecnologia para compreender e responder a comandos de voz. Sistemas de recomendação, como os da Netflix, Amazon e Spotify, também se beneficiam do ML ao sugerirem filmes, produtos e músicas de acordo com as preferências dos utilizadores[29].

Outra aplicação importante é o reconhecimento de imagens, presente em tecnologias de reconhecimento facial usadas em segurança e redes sociais, assim como no setor agrícola [30]. No setor financeiro, a deteção de fraude é aprimorada por modelos de ML que identificam atividades suspeitas em transações. No campo da mobilidade, os carros autónomos, como os da Tesla, empregam essa tecnologia para navegar e tomar decisões em tempo real [31].

Além disso, a filtragem de spam em serviços de email, como o Gmail, utiliza ML para identificar e bloquear mensagens indesejadas [32]. Ferramentas de análise de sentimentos também fazem uso dessa tecnologia para interpretar o sentimento das pessoas em redes sociais, fornecendo insights sobre tópicos, marcas e produtos [33]. No setor de saúde, o diagnóstico médico assistido por computador ajuda médicos a identificar doenças com base em imagens e dados clínicos [34].

A personalização de publicidade online, observada em plataformas como Google Ads e Facebook Ads, é outro exemplo de como o ML está a ser utilizada para direcionar anúncios de

forma mais eficaz [35]. Finalmente, o processamento de linguagem natural (NLP) permite a tradução automática de idiomas, como no Google Tradutor, e a geração automatizada de textos, facilitando assim a comunicação entre pessoas que falam diferentes idiomas [36].

Em suma, o ML visa criar modelos matemáticos que, ao aprenderem a partir de exemplos observados, conseguem identificar padrões e tomar decisões informadas sobre novos dados, permitindo assim a aplicação desse conhecimento em situações então desconhecidas de forma eficaz e automatizada [37].

2.2.4.1 Fundamentos do *Machine Learning*

No *machine learning*, o objetivo é criar um modelo que represente de forma eficaz a relação entre as variáveis de entrada (características ou *features*) e a saída (rótulo ou *target*) [28]. O modelo é treinado com a utilização de um conjunto de dados de treino, que consiste em pares de entradas e saídas conhecidas. O processo de treino envolve a otimização dos parâmetros do modelo para minimizar o erro de previsão, geralmente medido por uma função de perda [37].

Um dos métodos mais comuns de ML é **a aprendizagem supervisionada**, onde o modelo é treinado com um conjunto de dados rotulados [38]. A aprendizagem supervisionada inclui tarefas de regressão (onde a saída é um valor contínuo) e classificação (onde a saída é uma classe ou categoria) [39]. A função objetivo, geralmente representada como uma função de perda, é minimizada durante o treino. Para regressão, uma função de perda comum é o Erro Quadrático Médio (*MSE - Mean Squared Error*):

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Onde:

- y_i são os valores reais,
- $\hat{y}_i$ são as previsões do modelo,
- N é o número de exemplos [40].

Outro método importante é a **aprendizagem não supervisionada**, onde o modelo tenta encontrar padrões nos dados sem usar saídas rotuladas. Exemplos incluem *clustering* (agrupamento de dados em clusters) e análise de componentes principais (PCA) [41].

A **aprendizagem por reforço** é outro paradigma importante no ML onde um agente aprende a tomar decisões sequenciais interagindo com um ambiente para maximizar uma recompensa acumulada ao longo do tempo. O agente observa o estado do ambiente, escolhe ações com base numa política, e recebe recompensas que guiam o seu comportamento [42]. O objetivo é aprender uma estratégia que equilibre exploração (experimentar novas ações) e aproveitamento (utilizar o conhecimento atual para maximizar a recompensa). Este tipo de aprendizagem é aplicado em áreas como robótica, jogos e sistemas de recomendação, utilizando algoritmos como Q-Learning e redes neuronais profundas [43].

2.2.5 Aprendizagem Profunda (*Deep Learning*)

A aprendizagem profunda, ou *deep learning*, é uma subárea do ML que tem atraído enorme atenção devido à sua capacidade de resolver problemas complexos que envolvem grandes volumes de dados e padrões difíceis de identificar por métodos convencionais. O *deep learning* destaca-se principalmente pela sua utilização de *Artificial Neural Network* (ANN) com muitas camadas, o que permite a modelagem de dados em níveis progressivamente mais altos de abstração [30]. Esta capacidade tem implicações profundas em diversos campos, incluindo a visão computacional, processamento de linguagem natural, e particularmente na agricultura, onde é usada para monitorização de pragas e doenças [44][45].

2.2.5.1 Evolução e Importância do *Deep Learning*

Historicamente, as primeiras abordagens de ML dependiam fortemente de algoritmos que requeriam a extração manual de características pelos especialistas, ou seja, os engenheiros tinham de definir explicitamente quais atributos dos dados eram importantes para realizar uma tarefa específica, como classificação ou previsão. No entanto, este processo não só era moroso, como também limitado pela capacidade humana de identificar as características relevantes, o que se revelava insuficiente em situações onde as relações entre os dados eram extremamente complexas ou não lineares [26].

Com o advento da aprendizagem profunda, este paradigma mudou drasticamente. Em vez de exigir a intervenção humana para a definição das características, as redes neuronais profundas conseguem aprender automaticamente as representações dos dados, extraindo características diretamente a partir dos dados brutos. Isso é feito através de múltiplas camadas de neurónios artificiais, onde cada camada aprende a representar os dados de uma forma mais

abstrata e complexa do que a anterior [27]. Esta característica torna o *deep learning* especialmente poderoso em tarefas onde os dados são ricos e multifacetados, como a análise de imagens, áudio ou texto.

2.2.5.2 Arquitetura e Funcionamento das Redes Neuronais Profundas

Uma rede neuronal profunda é composta por várias camadas de neurónios artificiais, que são organizados em três tipos principais de camadas: a camada de entrada, várias camadas ocultas, e a camada de saída. Cada neurónio em uma camada está conectado aos neurónios da camada seguinte, formando uma rede complexa de conexões que permite a passagem e transformação dos dados através das camadas.

A operação básica em cada neurónio pode ser descrita por uma função matemática que calcula uma soma ponderada das entradas, aplica um viés, e depois passa o resultado por uma função de ativação não-linear. A função de ativação é crucial porque permite que a rede capture relações não lineares entre as entradas e as saídas, o que é essencial para a capacidade da rede de modelar dados complexos [37].

Embora a arquitetura exata de uma rede neuronal possa variar dependendo da tarefa, a ideia central é que, à medida que os dados passam por cada camada, a rede vai aprendendo a extrair características cada vez mais abstratas e de alto nível. Por exemplo, numa tarefa de reconhecimento de imagens, as camadas iniciais podem aprender a identificar bordas e texturas, enquanto as camadas mais profundas podem identificar formas mais complexas e objetos completos.

2.2.5.3 Algoritmos de Treino e *Backpropagation*

O treino de uma rede neuronal profunda envolve a otimização dos seus parâmetros (os pesos e os viéses) para que a rede consiga fazer previsões precisas quando confrontada com novos dados. Este processo de treino é realizado através de um algoritmo de otimização, sendo o mais comum o Gradiente Descendente Estocástico (do inglês *Stochastic* Gradient Descent SGD) [46]. O objetivo do SGD é minimizar uma função de perda, que mede a diferença entre a saída prevista pela rede e a saída esperada.

A função de perda é, por sua vez, minimizada utilizando um processo chamado ***backpropagation***. No *backpropagation*, o erro calculado na saída da rede é propagado de volta através das camadas, e os pesos das conexões são ajustados de acordo com o gradiente do erro

em relação a cada peso. Este processo é repetido muitas vezes (em ciclos chamados épocas) até que a função de perda atinja um valor mínimo aceitável [42].

2.2.5.4 Capacidade de Generalização e Regularização

Uma das preocupações centrais no treino de redes neuronais profundas é a capacidade de generalização, ou seja, a capacidade do modelo de funcionar bem não apenas nos dados de treino, mas também em novos dados que nunca viu antes. Um modelo que se ajusta demasiado bem aos dados de treino pode sofrer de *overfitting*, o que significa que ele memoriza os exemplos de treino em vez de aprender padrões gerais aplicáveis a novos dados [30].

Para combater o *overfitting*, várias técnicas de regularização são utilizadas. Uma das mais comuns é a regularização L2, que penaliza a magnitude dos pesos, incentivando a rede a manter os pesos pequenos e, portanto, menos propensos a ajustes excessivos. Outra técnica amplamente utilizada é o *dropout*, onde, durante o treino, alguns neurónios são desativados aleatoriamente, forçando a rede a ser mais robusta e a não depender excessivamente de uma pequena parte da rede para fazer previsões [47].

2.2.5.5 Aplicações em Agricultura: Monitorização de Pragas e Doenças

No contexto da agricultura, a aprendizagem profunda tem sido aplicada com sucesso em várias tarefas críticas, como a monitorização de pragas e doenças em plantações. A capacidade das redes neuronais profundas de analisar grandes volumes de dados e de identificar padrões complexos permite que sejam usadas para detetar sinais precoces de infestações ou doenças que poderiam passar despercebidos aos métodos tradicionais.

Por exemplo, na deteção da vespa-das-galhas-do-castanheiro, redes neuronais profundas podem ser treinadas para analisar imagens de castanheiros e identificar sinais de infestação, como galhas, nas folhas, ou ramos. A capacidade de identificar automaticamente estas características em imagens de alta resolução capturadas por drones ou outras tecnologias de campo permite intervenções mais rápidas e eficazes, reduzindo o impacto da praga nas colheitas [45].

2.2.5.6 Desafios e Futuro do *Deep Learning* na Agricultura

Apesar das suas vantagens, a implementação de soluções baseadas em *deep learning* na agricultura enfrenta desafios significativos. A necessidade de grandes volumes de dados de alta

qualidade para treinar os modelos é um dos principais obstáculos, uma vez que a recolha e anotação desses dados pode ser demorada e dispendiosa. Além disso, as redes neuronais profundas são computacionalmente intensivas, exigindo hardware especializado como GPUs de alta performance para treinar os modelos de forma eficiente [28].

Outro desafio é a complexidade e a falta de interpretabilidade dos modelos de *deep learning*. Como as redes neuronais profundas são frequentemente vistas como "caixas pretas", pode ser difícil para os utilizadores finais, como agricultores e técnicos agrícolas, entender como e por que razão a rede toma certas decisões. Esta falta de transparência pode limitar a confiança e a adoção generalizada destas tecnologias [48].

No entanto, à medida que a tecnologia evolui, espera-se que estas barreiras sejam superadas. Novas técnicas de *deep learning*, como *transfer learning* e *few-shot learning*, estão a ser desenvolvidas para reduzir a necessidade de grandes quantidades de dados e para melhorar a capacidade dos modelos de generalizar a partir de menos exemplos. Além disso, esforços contínuos para melhorar a interpretabilidade dos modelos estão a tornar estas ferramentas mais acessíveis e confiáveis para uma maior gama de utilizadores na agricultura.

2.2.6 *Convolutional Neural Network* (CNNs)

As CNNs são uma arquitetura essencial dentro do campo da aprendizagem profunda, projetada especificamente para processar dados que têm uma estrutura de grade, como imagens e vídeos. As CNNs são amplamente utilizadas em tarefas de visão computacional, incluindo deteção de objetos, reconhecimento facial, e análise de vídeos. No contexto agrícola, as CNNs têm sido fundamentais para o desenvolvimento de sistemas automatizados de monitorização de pragas e doenças, como a deteção da vespa-das-galhas-do-castanheiro.

2.2.6.1 Relação entre *Artificial Neural Network* (ANNs) e CNNs

Para entender as CNNs, é importante começar com as *Artificial Neural Network,* que são a base de muitos modelos de ML. As ANNs consistem em camadas de neurónios artificiais, onde cada neurónio recebe um conjunto de entradas, aplica pesos a essas entradas, soma-as e passa o resultado por uma função de ativação. As ANNs são geralmente usadas para tarefas como classificação, regressão e reconhecimento de padrões em dados estruturados ou não estruturados [37].

No entanto, as ANNs tradicionais apresentam limitações quando se trata de processar dados com estruturas espaciais, como imagens. Estas redes tratam cada entrada (cada pixel de uma imagem, por exemplo) como uma característica independente, ignorando as relações espaciais entre os dados. É aqui que entram as CNNs, que foram projetadas para capturar essas relações espaciais de forma eficiente [30].

As CNNs são, portanto, uma extensão das ANNs, mas com camadas especializadas que lidam explicitamente com a natureza espacial dos dados. Enquanto uma ANN tradicional tem camadas totalmente conectadas (onde cada neurónio está ligado a todos os neurónios da camada anterior e seguinte), uma CNN introduz camadas de convolução e *pooling* que reduzem a complexidade do modelo ao mesmo tempo que preservam as relações espaciais importantes.

2.2.6.2 Arquitetura das CNNs

A arquitetura de uma CNN é composta por várias camadas, cada uma com um papel específico no processamento dos dados:

1. **Camadas de Convolução (*Convolutional Layers*):**

As camadas de convolução são a espinha dorsal de uma CNN. Nestas camadas, pequenos filtros (ou *kernels*) são aplicados sobre a entrada para extrair mapas de características (*feature maps*). Cada filtro desliza pela imagem e executa uma convolução, que é uma operação matemática onde cada valor do filtro é multiplicado pelo valor correspondente da entrada, e os resultados são somados.

Este processo permite que a CNN detete padrões básicos, como bordas e texturas, nas primeiras camadas, e características mais complexas, como formas e objetos completos, nas camadas posteriores. Um dos principais benefícios da convolução é que ela preserva as relações espaciais entre os pixels, o que é essencial para tarefas de visão computacional [49].

A operação matemática de convolução pode ser representada como:

$$Output_{ij} = \sum_{m=1}^{M} \sum_{n=1}^{N} W_{mn} \cdot X_{i+m-1,j+n-1} + b$$

Onde:

- $Output_{ij}$: É o valor resultante na posição (i,j) do mapa de características.
- W_{mn}: É o valor do elemento na posição (m,n) no filtro ou *kernel*.

- $X_{i+m-1,j+n-1}$: É o valor do elemento na posição $(i+m-1, j+n-1)$ da imagem de entrada.
- b: O termo de viés adicionado após a operação de convolução.

2. **Camadas de *Pooling* (*Pooling Layers*):**

Depois das camadas de convolução, as camadas de *pooling* são utilizadas para reduzir a dimensionalidade dos mapas de características, o que ajuda a diminuir a carga computacional e a tornar a rede mais robusta a variações pequenas nos dados, como mudanças na posição e escala dos objetos na imagem. A forma mais comum de *pooling* é o *max pooling*, que seleciona o valor máximo em cada sub-região do mapa de características.

O *max pooling* pode ser matematicamente expresso como:

$$Output_{ij} = max\{X_{i,j}\}$$

Onde, $X_{i,j}$ representa os valores dentro de uma janela específica do mapa de características e $Output_{ij}$ é o valor máximo dessa janela [50].

3. **Funções de Ativação (*Activation Functions*)**:

As funções de ativação são cruciais para as ANNs porque introduzem não-linearidade, permitindo que a rede aprenda representações complexas dos dados. As funções de ativação mais comuns incluem:

Sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Esta função mapeia a entrada num intervalo entre 0 e 1, sendo útil em saídas binárias [51].

***ReLU* (*Rectified Linear Unit*)**:

$$f(x) = max\,(0, x)$$

A função *ReLU* é amplamente usada em redes neuronais profundas devido à sua eficiência computacional e à mitigação do problema do gradiente desaparecido, comum em funções como a *sigmoid* [52].

Tanh:

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

A função *Tanh* mapeia a entrada num intervalo entre -1 e 1, o que pode ser útil em algumas arquiteturas de rede, por exemplo em problemas onde se espera uma saída negativa [30].

4. **Camadas Totalmente Conectadas (*Fully Connected Layers*)**:

Nas últimas etapas da CNN, os mapas de características são "achatados" em um vetor e passados por uma ou mais camadas totalmente conectadas. Cada neurónio nestas camadas está conectado a todos os neurónios da camada anterior, e estas camadas são responsáveis por combinar as características extraídas para produzir a saída final do modelo, como a classificação de uma imagem [53].

5. **Camada de Saída (*Output Layer*)**:

A camada final da CNN geralmente utiliza uma função de ativação como a *softmax* para produzir a previsão final, que no caso da deteção de doenças pode ser a probabilidade de presença de uma condição específica, como a vespa-das-galhas-do-castanheiro, em diferentes partes da imagem [28].

2.2.6.3 Propagação Direta e *Backpropagation*

O treino de uma rede neuronal envolve dois passos principais: propagação direta (*forward propagation*) e *backpropagation*.

1. **Propagação Direta (*Forward Propagation*)**:

Na propagação direta, as entradas são passadas pela rede, camada por camada, até que uma saída seja produzida. Cada neurónio em uma camada recebe as saídas da camada anterior, aplica pesos e um viés, e depois processa o resultado através de uma função de ativação. O erro ou diferença entre a saída prevista e o valor real é então calculado usando uma função de perda. Em tarefas de classificação, uma função de perda amplamente utilizada é a Entropia Cruzada **(***Cross-Entropy Loss***)**, definida como:

$$L(y, \hat{y}) = \sum_{i=1}^{N} y_i log(\widehat{y_i})$$

Aqui, y_i é o rótulo verdadeiro, e y_i é a probabilidade prevista pela rede para a classe correta. A função de perda é minimizada utilizando um algoritmo de otimização como o

Gradiente Descendente Estocástico (SGD), que ajusta os pesos com base no gradiente da função de perda. Este processo é conhecido como *backpropagation*, onde o erro é propagado de volta através das camadas da rede para atualizar os pesos de forma a melhorar a precisão do modelo [28], [54].

2. ***Backpropagation***:

A etapa de *backpropagation* envolve a computação do gradiente da função de perda em relação a cada peso na rede, usando a regra da cadeia (derivada parcial). Esses gradientes são então usados para atualizar os pesos, reduzindo o erro. A atualização dos pesos é realizada de acordo com a seguinte fórmula:

$$w_i \leftarrow w_i - \alpha \frac{\partial \text{MSE}}{\partial w_i}$$

Onde:

- w_ié o peso associado ao neurónio *i*,
- α é a taxa de aprendizagem, um parâmetro que controla o tamanho do passo dado na direção do gradiente,
- $\frac{\partial \text{MSE}}{\partial w_i}$ é o gradiente da função de perda em relação ao peso w_i [28], [54].

2.2.6.4 Otimização e Regularização

A otimização em *deep learning* refere-se ao processo de ajuste dos pesos para minimizar a função de perda. Uma das técnicas mais comuns é o Gradiente Descendente Estocástico (SGD - *Stochastic Gradient Descent*), onde a atualização dos pesos é baseada num subconjunto aleatório dos dados (*mini-batch*), o que acelera a convergência [55].

Além do SGD, outras técnicas de otimização incluem:

Momentum: que acumula uma fração do vetor de atualização anterior para acelerar a convergência em direções consistentes [55].

Adam (*Adaptive Moment Estimation*): que combina o *momentum* com a adaptação da taxa de Aprendizagem, sendo uma das técnicas mais eficazes para uma variedade de problemas [55].

A *regularização* é uma técnica usada para evitar o *overfitting*, onde o modelo se ajusta muito bem aos dados de treino, mas não generaliza bem para novos dados. Métodos comuns de regularização incluem:

L2 *Regularization* (*Ridge Regression*): que adiciona um termo à função de perda para penalizar pesos grandes:

$$Erro\ Total = MSE + \lambda \sum_{i=1}^{n} w_i^2$$

Dropout: onde, durante o treino, alguns neurónios são "desligados" aleatoriamente, forçando a rede a aprender representações mais robustas e distribuídas [55].

2.2.6.5 Aplicação de CNNs na Agricultura

No campo da agricultura, as CNNs têm demonstrado um enorme potencial, particularmente na deteção e monitorização de pragas e doenças em plantações. As CNNs são especialmente eficazes em tarefas de visão computacional, onde a deteção de padrões espaciais, como formas e texturas, é crucial [45].

Durante o treino de uma CNN para detetar pragas, o modelo é alimentado com um grande número de imagens de plantas, algumas das quais contêm pragas. A CNN aprende a distinguir entre imagens saudáveis e imagens com pragas, identificando características visuais associadas a cada classe.

A CNN consiste em várias camadas de convolução, onde filtros (ou *kernels*) são aplicados às imagens de entrada para extrair características importantes, seguidas por camadas de pooling que reduzem a dimensionalidade dos dados, mantendo as características mais relevantes. A última camada é geralmente uma camada totalmente conectada que realiza a classificação final, atribuindo uma probabilidade a cada classe (presença ou ausência de pragas) [45].

No caso da vespa-das-galhas-do-castanheiro, uma CNN pode ser treinada para analisar imagens de castanheiros e identificar características visuais que indicam a presença de galhas infetadas, como deformações nas folhas ou ramos. Este tipo de análise pode ser realizado de forma automatizada e em tempo real, permitindo uma resposta rápida às infestações e a minimização do impacto nas colheitas [56].

Além da deteção de pragas, as CNNs têm sido aplicadas em outras áreas da agricultura, como a análise de qualidade de frutos, monitorização da saúde das plantas através de imagens

aéreas capturadas por drones, e até na previsão de rendimentos agrícolas com base em características visuais das plantas [57].

2.2.6.6 Vantagens e Limitações das CNNs

Vantagens:

Autonomia na Extração de Características: Uma das maiores vantagens das CNNs é a sua capacidade de aprender características diretamente dos dados de entrada, eliminando a necessidade de intervenção humana na definição dessas características. Isto é especialmente útil em contextos como a deteção de pragas, onde as características relevantes podem ser difíceis de definir manualmente [45].

Escalabilidade: As CNNs são altamente escaláveis, o que significa que podem ser aplicadas a grandes volumes de dados, como milhares ou milhões de imagens. Isto é particularmente útil na agricultura de precisão, onde são necessários dados em grande escala para monitorizar vastas áreas de cultivo [45].

Robustez a Variações nos Dados: Graças às camadas de convolução e *pooling*, as CNNs são robustas a pequenas variações nos dados de entrada, como mudanças de iluminação, ângulo de captura da imagem, ou pequenas variações na aparência das pragas [58].

Limitações:

Necessidade de Grandes Volumes de Dados: As CNNs requerem grandes volumes de dados rotulados para treinar de forma eficaz. Na agricultura, isso pode ser um desafio, pois a recolha e anotação de dados de campo pode ser dispendiosa e morosa [28].

Custo Computacional Elevado: O treino de CNNs em grande escala exige recursos computacionais significativos, incluindo o uso de GPUs (Unidades de Processamento Gráfico), o que pode ser uma barreira para a adoção destas tecnologias em ambientes com recursos limitados [5].

Complexidade e Interpretação: As CNNs são frequentemente vistas como "caixas pretas", onde é difícil interpretar ou explicar como e por que razão a rede chegou a uma determinada conclusão. Isto pode ser um problema em situações onde a transparência e a interpretabilidade são cruciais, como na tomada de decisões agrícolas [59].

Em suma, as *Convolutional Neural Network* são uma poderosa ferramenta dentro do arsenal da aprendizagem profunda, oferecendo capacidades avançadas de processamento de

imagens que são particularmente úteis na agricultura moderna. No entanto, como qualquer tecnologia, as CNNs têm as suas próprias limitações, que devem ser consideradas na sua aplicação prática. À medida que a tecnologia continua a evoluir, espera-se que as CNNs se tornem ainda mais eficientes e acessíveis, facilitando a sua adoção em larga escala na agricultura de precisão e em outras áreas críticas.

2.2.7 Visão Computacional

Visão computacional e visão artificial são frequentemente usadas como sinônimos, mas há uma ligeira diferença entre os termos. Visão computacional refere-se ao campo da ciência que desenvolve algoritmos e técnicas para permitir que computadores processem e compreendam imagens. Já visão artificial é um termo mais amplo, muitas vezes usado para descrever sistemas que utilizam tecnologias de visão para realizar tarefas específicas, geralmente no contexto de automação industrial. No entanto, na prática, ambos os termos são usados de forma intercambiável [60].

A visão computacional é um campo vibrante e de rápido crescimento, que desempenha um papel fundamental em diversas áreas da tecnologia moderna, desde a automação industrial até a análise de imagens médicas e os veículos autónomos. Para facilitar o desenvolvimento de soluções de visão computacional, uma ampla variedade de ferramentas e bibliotecas foram criadas ao longo dos anos, tanto para profissionais quanto para iniciantes. Essas ferramentas ajudam a construir, treinar e implementar modelos que processam e analisam imagens e vídeos, permitindo a deteção de objetos, reconhecimento facial, segmentação de imagem, rastreamento de movimento e muito mais [60].

Neste texto, vamos explorar em profundidade as principais ferramentas de visão computacional utilizadas por desenvolvedores, pesquisadores e engenheiros. Vamos detalhar como cada ferramenta se destaca em termos de funcionalidades, casos de uso e compatibilidade com diversas linguagens e plataformas. Faremos uma análise de bibliotecas populares como OpenCV, TensorFlow, Keras e PyTorch, além de *frameworks* especializados, como YOLO e Fastai [2,3]. Também discutiremos ferramentas proprietárias, como as fornecidas pela NVIDIA e Google, que oferecem poder computacional e integração em grande escala para projetos de visão computacional[4].

Além disso, vamos analisar o uso de ambientes como Google Colab, Amazon SageMaker, e NVIDIA Jetson, que permitem a execução de tarefas intensivas de visão computacional em ambientes otimizados [61]. No final, discutiremos o impacto dessas

ferramentas no campo da visão computacional e os avanços recentes que estão moldando o futuro da área [62].

2.2.7.1 Ferramentas e Técnicas de Visão Computacional

Para facilitar o desenvolvimento de soluções de visão computacional, uma ampla variedade de ferramentas e bibliotecas foram criadas ao longo dos anos, tanto para profissionais quanto para iniciantes. Essas ferramentas ajudam a construir, treinar e implementar modelos que processam e analisam imagens e vídeos, permitindo a deteção de objetos, reconhecimento facial, segmentação de imagem, rastreamento de movimento e muito mais [63].

A aplicação de visão computacional na agricultura tem-se expandido significativamente, impulsionada pelo desenvolvimento de diversas ferramentas e bibliotecas que facilitam a implementação de algoritmos complexos para análise de imagens e vídeos. Estas ferramentas permitem a automatização de tarefas que vão desde a deteção de pragas e doenças até à monitorização do crescimento das culturas[45].

Neste subcapítulo, serão discutidas algumas das principais ferramentas de visão computacional amplamente utilizadas na agricultura, com base em bibliotecas como OpenCV, TensorFlow, PyTorch, e plataformas como Roboflow [61]. As bibliotecas mencionadas são amplamente utilizadas devido à sua flexibilidade, integração com outros sistemas e suporte a diversos algoritmos de aprendizagem profunda e visão computacional.

2.2.7.1.1 Principais Ferramentas de Visão Computacional

As ferramentas de código aberto desempenham um papel fundamental no desenvolvimento de soluções de visão computacional, proporcionando a flexibilidade e escalabilidade necessárias para criar e implementar modelos eficazes. Estas ferramentas são amplamente utilizadas por investigadores e engenheiros, pois permitem a criação de soluções personalizadas para uma vasta gama de aplicações. A seguir, exploramos as principais bibliotecas de código aberto amplamente adotadas na visão computacional: *OpenCV*, *TensorFlow*, *PyTorch*, *YOLO* e Fastai.

O **OpenCV (*Open Source Computer Vision Library*)** é uma das bibliotecas mais utilizadas no campo da visão computacional. Desenvolvida pela Intel, o OpenCV tornou-se uma referência para a manipulação de imagens e vídeos. A sua versatilidade permite realizar desde operações básicas, como a deteção de bordas, até funcionalidades mais avançadas, como a utilização de algoritmos de aprendizagem automática integrados. A compatibilidade com

várias linguagens de programação, como Python, C++, Java e MATLAB, o que facilita a sua integração em diversas plataformas. Além disso, o OpenCV é amplamente usado em áreas como segurança, vigilância, robótica e análise de imagens médicas. Uma das grandes vantagens do OpenCV é a extensa comunidade ativa que contribui para o desenvolvimento contínuo da biblioteca, mantendo-a atualizada e altamente personalizável para diferentes necessidades [63], [60].

Outra ferramenta amplamente usada na visão computacional é o TensorFlow, desenvolvido pela Google. O TensorFlow é uma das bibliotecas mais poderosas para *deep learning*, sendo amplamente utilizada para construir, treinar e implementar modelos de aprendizagem profunda, especialmente CNNs. Com suporte para TPUs (*Tensor Processing Units*), o TensorFlow consegue acelerar significativamente o treino de grandes redes neuronais, o que é crucial para soluções complexas de visão computacional. A API de alto nível **Keras**, baseada no TensorFlow, facilita ainda mais o desenvolvimento de redes neuronais convolucionais, tornando o processo de criação de modelos de *deep learning* acessível tanto para principiantes como para programadores que necessitam de prototipagem rápida [64]. O TensorFlow e o Keras são amplamente utilizados em diversas áreas, como análise de vídeos em tempo real e diagnósticos médicos baseados em imagem [65].

O **PyTorch**, desenvolvido pela Facebook AI Research, é também amplamente utilizado para *deep learning*, oferecendo maior flexibilidade e facilidade de uso, principalmente devido ao seu suporte a gráficos computacionais dinâmicos. Isto facilita a depuração e a experimentação rápida, tornando o PyTorch uma escolha comum tanto para investigação como para produção, especialmente em áreas como deteção de objetos e segmentação de imagens médicas. A popularidade do PyTorch deve-se à sua flexibilidade e integração com bibliotecas adicionais, como o Fastai [66].

Uma ferramenta essencial para a deteção de objetos em tempo real é o **YOLO (*You Only Look Once*)**, que se destaca pela sua rapidez e precisão. Desenvolvido por Joseph Redmon e Ali Farhadi, o YOLO introduziu um novo paradigma ao realizar a deteção de objetos numa única passagem pela imagem, permitindo deteções rápidas sem comprometer a precisão [4]. Isto faz do YOLO uma ferramenta amplamente utilizada em aplicações que exigem processamento em tempo real, como veículos autónomos, vigilância por vídeo e monitorização agrícola. As versões mais recentes, como o YOLOv8, apresentam melhorias em termos de precisão e velocidade, mantendo um desempenho eficiente[67]. Um sistema baseado no YOLO enquadra-se na categoria de visão computacional. YOLO é um algoritmo utilizado para a deteção de objetos em imagens e vídeos, que é uma aplicação típica da visão computacional.

Este sistema pode também ser descrito como parte de um sistema de visão artificial, especialmente se for integrado num contexto mais amplo de automação, como o uso de um drone para inspeção. Portanto, o YOLO é uma ferramenta de visão computacional que pode fazer parte de um sistema maior de visão artificial.

Já o **Fastai** é uma biblioteca de alto nível construída sobre o PyTorch, desenhada para simplificar o desenvolvimento de modelos de aprendizagem profunda. A biblioteca abstrai muitos dos detalhes técnicos envolvidos na criação de modelos, permitindo que os programadores se concentrem em experimentar rapidamente novas ideias. O Fastai inclui funcionalidades como *transfer learning* e *data augmentation*, que ajudam a melhorar o desempenho dos modelos, especialmente quando os dados são limitados [68]. A sua simplicidade torna-o ideal para investigação, ensino e desenvolvimento rápido, sem sacrificar a flexibilidade.

E por fim, o **Roboflow** é uma plataforma abrangente para a gestão de *datasets* e machine learning em visão computacional, que facilita a criação, anotação, preparação e treino de modelos. Esta ferramenta tem inúmeras vantagens se comparada com o **LabelImg**, pois permite não só fazer anotações, que é a única função do LabelImg, como também permite aos utilizadores fazer upload de *datasets*, aplicar técnicas de data augmentation (como rotações, *flips*, ajustes de brilho), e organizar as anotações para tarefas como deteção de objetos, segmentação de instâncias e classificação de imagens. O Roboflow inclui funcionalidades para pré-processamento de dados, como redimensionamento e normalização, e permite integração direta com modelos populares, como YOLO, TensorFlow e PyTorch, facilitando a exportação e o treino destes modelos.

A plataforma também oferece diferentes tipos de treino diretamente na sua interface, possibilitando que os utilizadores treinem modelos para deteção de objetos, segmentação de imagens e classificação sem necessidade de configuração externa. Além disso, o Roboflow suporta a implementação (*deployment*) dos modelos treinados, permitindo que sejam utilizados na *cloud* para inferência em tempo real através de APIs REST, facilitando a integração das previsões com aplicações e serviços. Assim, o Roboflow oferece um pipeline completo, desde a anotação dos dados até ao treino e implementação de modelos de visão computacional.

Em conclusão, as ferramentas de código aberto fornecem uma plataforma robusta e flexível para a construção de soluções de visão computacional. Desde o OpenCV, amplamente utilizado para o processamento de imagens tradicional, até ao YOLO, ideal para a deteção de objetos em tempo real, cada biblioteca tem os seus pontos fortes. A escolha da ferramenta certa

depende das necessidades específicas do projeto, incluindo requisitos de desempenho, escalabilidade e facilidade de uso.

2.2.7.2 *Softwares* proprietários de Visão Computacional

Existem também softwares proprietários de Visão Computacional que são amplamente utilizados em várias indústrias devido à sua robustez e capacidade de integração com diferentes sistemas. O MATLAB destaca-se pelas suas extensas bibliotecas de processamento de imagem e pela facilidade de utilização em projetos de engenharia[69]. O HALCON, da MVTec, é largamente utilizado na automação industrial, oferecendo uma plataforma flexível para a deteção de objetos e inspeção de qualidade [70]. O Cognex VisionPro é outra ferramenta popular em linhas de produção, com funções avançadas para inspeção visual e deteção de defeitos [71]. No contexto de soluções baseadas na nuvem, o Amazon Rekognition permite a análise de imagens e vídeos para reconhecimento de objetos e rostos, com foco na escalabilidade e integração com outros serviços da AWS [72], enquanto o Google Cloud Vision oferece ferramentas poderosas para categorização de imagens e OCR (Reconhecimento Ótico de Caracteres) [73]. Finalmente, o Microsoft Azure Computer Vision combina reconhecimento de objetos e análise de imagem com a escalabilidade da plataforma de nuvem Azure, ideal para soluções empresariais que requerem processamento em grande escala [74]. Estes softwares não serão utilizados para este contexto.

2.2.7.3 Justificação da Escolha do YOLOv8.2

A escolha do **YOLOv8.2** como modelo para a deteção de doenças nas galhas causadas pela vespa do castanheiro baseia-se em diversos fatores, nomeadamente a sua precisão, eficiência e capacidade de deteção em tempo real. Esta escolha é justificada pela necessidade de um modelo capaz de identificar pequenos objetos (galhas e rebentos) em imagens de campo, que são frequentemente influenciadas por variações de iluminação e condições ambientais adversas. O YOLOv8.2 representa uma evolução significativa das versões anteriores da arquitetura YOLO, introduzindo melhorias tanto em termos de precisão como de velocidade, tornando-o ideal para este projeto.

Deteção em Tempo Real

Uma das principais vantagens do YOLOv8.2 é a sua capacidade de realizar deteção de objetos em tempo real, o que é fundamental em aplicações de campo, como a monitorização de plantações, onde o modelo pode ser implementado em drones ou sistemas de câmaras móveis. A arquitetura YOLO foi originalmente concebida para maximizar a velocidade de inferência, ao dividir uma imagem em grades e realizar a deteção de todos os objetos relevantes numa única passagem pela imagem (*one pass inference*) [75]. Esta abordagem distingue-se de métodos tradicionais, como o Faster R-CNN, que primeiro gera propostas de regiões e depois classifica essas regiões, resultando em maior tempo de processamento.

No contexto do projeto de deteção de galhas, a rapidez de deteção é crucial para permitir monitorização contínua e eficiente em áreas vastas, como plantações de castanheiros. Esta capacidade de inferência em tempo real é potenciada pelo design otimizado do YOLOv8.2, que permite alcançar um compromisso ideal entre velocidade e precisão, mesmo quando utilizado em hardware de menor capacidade, como GPUs em dispositivos móveis ou drones [75].

Precisão na Deteção de Objetos Pequenos

Outro desafio significativo no contexto da deteção de doenças nas galhas é a necessidade de identificar objetos de pequenas dimensões e com variações visuais subtis, que podem passar despercebidos em imagens de baixa resolução ou com ruído. O *YOLOv8.2* oferece uma melhoria na deteção de objetos pequenos graças à introdução de uma *anchor-free design*, que ajusta automaticamente o tamanho das *bounding boxes* em função das características dos objetos detetados [75]. Este ajuste é crucial quando se trabalha com objetos como as galhas, que podem variar em tamanho e posição em diferentes partes das árvores.

Além disso, o YOLOv8.2 utiliza uma *distributed prediction layer* que permite que o modelo se concentre em diferentes escalas de objetos em diferentes camadas, melhorando assim a precisão global da deteção, particularmente em cenários onde há uma mistura de objetos grandes e pequenos na mesma imagem [67]. No caso deste projeto, onde se requer a deteção de galhas de vários tamanhos e em diferentes condições de iluminação, esta característica do YOLOv8.2 oferece uma clara vantagem em termos de robustez e versatilidade.

Eficiência no Treino e Integração com *Data Augmentation*

A disponibilidade de apenas 800 imagens anotadas é um fator limitante para o treino de um modelo de visão artificial. No entanto, o YOLOv8.2 foi otimizado para treinar de forma

eficaz com conjuntos de dados limitados, aproveitando técnicas como *transfer learning* e *data augmentation* [67]. O *transfer learning* permite iniciar o treino a partir de um modelo previamente treinado em grandes bases de dados, como o COCO, ajustando os parâmetros para o domínio específico da deteção de galhas. Esta técnica reduz significativamente o tempo de treino e melhora a precisão mesmo com um número reduzido de exemplos anotados.

Além disso, o YOLOv8.2 integra facilmente técnicas de *data augmentation*, que permitem aumentar virtualmente o tamanho do conjunto de dados através da criação de variações artificiais das imagens, como rotações, inversões horizontais e verticais, ajustes de iluminação e escalamento. Esta capacidade de gerar novas variações das imagens de treino permite ao modelo generalizar melhor para novas condições, como mudanças sazonais de luz ou variações na aparência das galhas devido ao progresso da doença [75].

Arquitetura Eficiente e Capacidade de Escalabilidade

O YOLOv8.2 tem uma arquitetura modular e flexível, o que permite que o modelo seja ajustado conforme as necessidades do projeto. Isto significa que pode ser configurado tanto para maximizar a precisão da deteção como para aumentar a velocidade de inferência, dependendo dos requisitos. A arquitetura do YOLOv8.2 é composta por duas partes principais: o *backbone* e a *detection head*. O *backbone* é responsável por extrair as características principais da imagem, como padrões, texturas e formas, utilizando camadas convolucionais. Já a *detection head* é a parte que processa essas características extraídas e faz a previsão dos objetos, gerando as *bounding boxes* e classificando-os. A combinação de camadas mais superficiais (rasas) e mais profundas permite ao modelo captar tanto detalhes finos quanto informações mais gerais, resultando num equilíbrio entre a capacidade de generalização do modelo e a precisão nas deteções [75].

Este equilíbrio é particularmente útil no contexto agrícola, onde o sistema pode necessitar de ser ajustado para diferentes tipos de plantações, ou para a deteção de outras pragas ou doenças além da vespa-das-galhas-do-castanheiro. A arquitetura flexível do YOLOv8.2 também permite uma adaptação fácil a diferentes configurações de hardware, seja em dispositivos de baixo consumo energético, como câmaras inteligentes, ou em sistemas de alto desempenho, como servidores com GPUs dedicadas [75].

Comparação com Outras Abordagens

Quando comparado com outras arquiteturas populares, como Faster R-CNN ou SSD (*Single Shot Multibox Detector*), o YOLOv8.2 oferece um desempenho superior em cenários que exigem tempo de resposta rápido e deteção em tempo real. Enquanto o Faster R-CNN alcança alta precisão, o seu tempo de inferência é significativamente maior, o que torna esta arquitetura menos adequada para aplicações que requerem processamento contínuo de grandes quantidades de dados de imagem, como a monitorização agrícola com drones. Por outro lado, o SSD, embora mais rápido que o Faster R-CNN, não atinge a mesma precisão que o YOLOv8.2, especialmente na deteção de objetos pequenos [76].

O YOLOv8.2, por outro lado, consegue combinar o melhor dos dois mundos, fornecendo alta precisão e deteção eficiente de objetos pequenos, ao mesmo tempo que mantém um tempo de inferência suficientemente rápido para aplicações de campo. A sua arquitetura simplificada e eficiente permite que a deteção seja feita de forma simultânea em várias escalas, algo fundamental no contexto de inspeção de plantações, onde os objetos de interesse (neste caso, galhas) podem variar em tamanho, forma e posição dentro da imagem [75].

Em suma, A escolha do YOLOv8.2 para o projeto de deteção de doenças nas galhas causadas pela vespa-das-galhas-do-castanheiro é a mais apropriada devido à sua combinação única de rapidez, precisão e flexibilidade. A capacidade de detetar objetos pequenos com alta precisão, aliada à sua eficiência em tempo real e à facilidade de integração com técnicas de *transfer learning* e *data augmentation*, tornam-no o modelo ideal para um projeto que depende da monitorização eficiente de plantações e da identificação de sintomas de doenças em condições variáveis. A arquitetura escalável do YOLOv8.2 também garante que, à medida que o projeto evolui, o modelo pode ser ajustado para lidar com novos desafios e cenários.

2.3 Especificidades da Vespa-das-Galhas-do-Castanheiro

A *Dryocosmus kuriphilus*, mais conhecida como Vespa-das-galhas-do-castanheiro, é originária da China, onde se desenvolveu como uma praga associada a espécies locais de castanheiros (*Castanea mollissima* e *Castanea crenata*). Embora a sua presença na China fosse bem controlada, devido à existência de predadores naturais e ao desenvolvimento de mecanismos de resistência nas árvores hospedeiras, a introdução acidental da praga em outros países teve consequências graves, especialmente em regiões onde a cultura do castanheiro possui elevada importância económica. A sua rápida disseminação ocorreu devido à ausência

de controles eficazes e à vulnerabilidade das espécies locais de castanheiros, resultando numa propagação alarmante da praga ao longo das últimas décadas [77].

Figura 2.5- *Dryocosmus kuriphilus* Adulta, Estado larvar e sintomas [78]

2.3.1 Origem e Introdução em Novos Ecossistemas

A *Dryocosmus kuriphilus*, conhecida como Vespa-das-galhas-do-castanheiro, teve origem na China, onde evoluiu ao longo de milénios como uma praga que afeta espécies nativas de castanheiros asiáticos (*Castanea mollissima* e *Castanea crenata*). Nas suas regiões de origem, o impacto da vespa é relativamente contido, uma vez que os castanheiros locais desenvolveram, ao longo do tempo, mecanismos de defesa natural contra as vespas. Além disso, na Ásia, existe uma comunidade de parasitoides e predadores naturais que ajudam a manter as populações da praga em níveis equilibrados. No entanto, com a globalização do comércio de plantas e a crescente movimentação de mudas e material vegetal entre continentes, a *D. kuriphilus* foi inadvertidamente introduzida em outros ecossistemas, onde as árvores hospedeiras não possuíam defesas evolutivas eficazes contra a praga, resultando na sua rápida e destrutiva disseminação [3].

A primeira introdução documentada fora da China ocorreu no Japão, em 1941. Acredita-se que a praga tenha chegado ao país através de mudas de castanheiros infestadas, transportadas sem os devidos cuidados fitossanitários. No Japão, a Vespa-das-galhas-do-castanheiro encontrou condições ideais para se estabelecer, dada a presença abundante de castanheiros nativos e a ausência de inimigos naturais locais, levando a uma rápida proliferação [1].

Após a introdução no Japão, a praga espalhou-se rapidamente para outros países asiáticos, como a Coreia do Sul, onde os castanheiros também eram amplamente cultivados. A introdução subsequente ocorreu nos Estados Unidos, em 1974, quando foi detetada na Geórgia. Mais uma vez, o transporte inadvertido de mudas de castanheiros infestadas foi o provável veículo para a introdução. No contexto norte-americano, a vespa começou a infestar o castanheiro-americano (*Castanea dentata*), que já estava severamente enfraquecido por outras doenças, como o cancro-do-castanheiro (*Cryphonectria parasitica*), agravando ainda mais o declínio da espécie nos EUA [1].

A Europa foi atingida pela primeira vez em 2002, quando a Vespa-das-galhas-do-castanheiro foi identificada na região do Piemonte, norte da Itália. Assim como nas introduções anteriores, o transporte de mudas infestadas foi o principal meio de propagação. A rápida propagação em solo europeu ocorreu em grande parte devido à ausência de inimigos naturais e ao fato de os castanheiros europeus (*Castanea sativa*) não estarem preparados evolutivamente para resistir ao ataque da praga. Em poucos anos, a vespa espalhou-se para a França, Eslovénia e Croácia, atingindo posteriormente a Península Ibérica, sendo detetada em Portugal em 2014, nas regiões de Entre-Douro-e-Minho e Trás-os-Montes, este último, um importante centro de produção de castanhas [79].

A capacidade de voo da vespa é limitada, estudos indicam que, em condições normais, o voo da vespa pode alcançar apenas algumas centenas de metros, o que significa que a dispersão natural da praga ocorre de forma lenta. Sendo o transporte direto de mudas e material vegetal o fator mais impactante para a dispersão de longas distâncias. Embora a *D. kuriphilus* seja capaz de voar apenas pequenas distâncias, a infestação pode se espalhar rapidamente em áreas próximas, levando à infestação de grandes florestas ou pomares ao longo do tempo. A capacidade da praga de se adaptar a diferentes climas também facilitou a sua propagação global. Embora seja nativa de regiões temperadas da China, a vespa conseguiu se estabelecer em uma variedade de climas temperados, como o mediterrânico, europeu e até subtropical, o que tem amplificado o seu impacto em diversas regiões do globo [80].

A introdução da *Dryocosmus kuriphilus* em novos ecossistemas tem resultado em sérios desequilíbrios ecológicos. Em regiões como a Europa e os EUA, onde não existem parasitóides ou predadores nativos eficazes para controlar a vespa, as populações crescem sem obstáculos. A ausência de mecanismos de defesa por parte dos castanheiros locais agrava ainda mais a situação. Esses desequilíbrios ecológicos resultam em perdas agronómicas e económicas severas, principalmente em regiões onde a cultura do castanheiro tem grande importância, seja pelo valor económico da castanha, seja pelo papel ecológico das árvores nos ecossistemas florestais [3].

Como consequência dessa introdução em novos ambientes, tornou-se crucial o desenvolvimento de estratégias fitossanitárias eficazes para evitar novas dispersões. Regulamentações sobre o transporte de mudas e material vegetal, assim como a inspeção rigorosa de viveiros, são medidas essenciais para conter a propagação da praga. Em várias regiões da Europa e nos Estados Unidos, legislações rigorosas de quarentena foram estabelecidas para impedir a movimentação de plantas infestadas e para eliminar focos de infestações emergentes. No entanto, como demonstra a experiência em múltiplas regiões do

mundo, uma vez que a Vespa-das-galhas-do-castanheiro se instala em uma nova área, torna-se extremamente difícil erradicá-la completamente, reforçando a importância das medidas preventivas [3].

2.3.2 Ciclo de Vida da Vespa-das-galhas-do-castanheiro

A *Dryocosmus kuriphilus* tem um ciclo de vida anual (univoltino), e cada fêmea pode viver entre 2 e 10 dias após atingir a fase adulta. As fêmeas emergem no início do verão e depositam os ovos nas gemas dos castanheiros. Cada fêmea pode colocar mais de 100 ovos ao longo da sua vida, com cerca de 20 a 30 ovos depositados num único gomo [81], [82].

Os ovos permanecem nas gemas durante o inverno, eclodindo na primavera seguinte, quando as larvas começam a desenvolver-se dentro dos tecidos vegetais, causando a formação de galhas. Estas galhas são facilmente visíveis após o abrolhamento dos brotos e podem ser encontradas nos ramos e folhas dos castanheiros. As larvas desenvolvem-se no interior das galhas até ao início do verão, quando atingem a fase adulta, completando assim o seu ciclo de vida [81],[82].

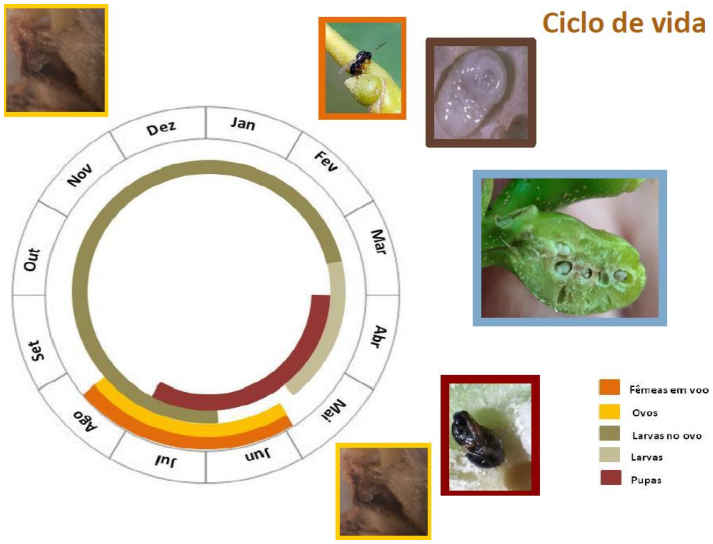


Figura 2.6- Ciclo de vida da Vespa do Castanheiro [81]

2.3.3 Impacto Agronómico

O principal impacto agronómico da Vespa-das-galhas-do-castanheiro está relacionado com a formação de galhas nos ramos e folhas dos castanheiros, o que interfere no crescimento vegetativo e na frutificação. As galhas impedem a adequada circulação de nutrientes e água, resultando numa redução significativa do crescimento dos ramos. Como consequência, a

capacidade de produção de castanhas pode diminuir drasticamente, com perdas de rendimento que podem chegar a 80% em casos de infestação severa.

Além disso, a praga afeta negativamente a apicultura, uma vez que a produção de flores é reduzida devido ao menor vigor das árvores afetadas. Em infestações prolongadas e não controladas, a praga pode, em alguns casos, levar à morte das árvores, especialmente em castanheiros jovens e suscetíveis [81].

Figura 2.7- Sinais e sintomas da presença da Vespa do Castanheiro [81]

2.3.4 Impacto Económico

O mercado global de castanhas tem registado um crescimento constante, com um valor estimado em mais de 3,5 mil milhões de dólares em 2023. A China é o principal produtor, contribuindo com cerca de 80% da produção mundial, seguida por outros grandes produtores, como Itália, Coreia, Espanha e Portugal. A crescente procura de castanhas tanto para o consumo direto quanto para a indústria de alimentos e cosméticos tem sido o principal motor deste aumento de valor [83].

A Vespa-das-galhas-do-castanheiro (*Dryocosmus kuriphilus*) tem tido um impacto económico devastador nas regiões produtoras de castanha, afetando significativamente o rendimento agrícola. Esta praga pode reduzir a produção em 70-80%, dependendo da intensidade da infestação. As árvores infestadas apresentam uma diminuição na produção de frutos, o que afeta diretamente a rentabilidade das colheitas. Para além disso, os custos acrescidos com o controlo da praga, como o uso de parasitóides naturais, como o *Torymus sinensis*, aumentam os encargos para os agricultores [84]. As medidas de controlo biológico têm-se mostrado as mais eficazes, uma vez que o uso de inseticidas não tem sido uma opção viável devido ao impacto ambiental e à limitada eficácia destes produtos [80].

Esta praga não só afeta a produção de castanhas, mas também tem implicações a nível ecológico, uma vez que o declínio das árvores castanheiras afeta a biodiversidade e a estrutura das florestas em regiões onde a cultura do castanheiro é predominante. A disseminação da vespa tem resultado em grandes prejuízos, tanto económicos quanto ambientais, com necessidade de uma resposta coordenada entre governos, produtores e cientistas para conter a praga e minimizar os seus impactos [80].

2.3.4.1 Impacto económico em Portugal

A produção de castanhas em Portugal tem um papel crucial na economia agrícola, especialmente na região de Trás-os-Montes, sendo o maior produtor do país. Além das perdas diretas, os custos com o controlo da praga aumentam substancialmente. Uma das estratégias de controlo mais eficazes tem sido o uso do parasitoide *Torymus sinensis*, que tem sido implementado em Portugal como medida de controlo biológico. No entanto, esta abordagem envolve elevados custos e exige monitorização constante, o que representa um desafio adicional para os produtores [82], [84].

Outro fator agravante tem sido a mudança climática, que favorece a disseminação de pragas e doenças, comprometendo ainda mais a produção de castanhas em várias regiões do país. As árvores afetadas muitas vezes apresentam dificuldades em recuperar, o que agrava a baixa produtividade média em algumas áreas [84], [85].

Apesar desses desafios, o mercado global de castanhas tem continuado a crescer, com Portugal a desempenhar um papel importante nas exportações. O valor anual das exportações de castanhas portuguesas está estimado em cerca de 65 milhões de euros. No entanto, a competitividade no mercado internacional, especialmente em comparação com grandes produtores como Itália e Grécia, exige maior organização da produção e melhorias nos processos de comercialização [86].

2.3.5 Estratégias atuais de controlo e gestão

2.3.5.1 Controlo Biológico

O uso de parasitóides naturais, especialmente o Torymus sinensis, é a estratégia mais eficaz e amplamente adotada para o controlo da praga. Este parasitóide é introduzido intencionalmente nas áreas afetadas, onde as fêmeas de Torymus depositam os seus ovos dentro das galhas formadas pela D. kuriphilus. As larvas do parasitóide, ao eclodirem, alimentam-se

das larvas da vespa, interrompendo o seu ciclo de vida. Esta abordagem tem demonstrado resultados positivos na redução das populações de vespa sem impactos negativos no ecossistema [80], [85].

No entanto, a introdução de Torymus sinensis exige uma monitorização contínua para garantir que o parasitóide seja eficaz em controlar a praga sem se tornar ele próprio uma ameaça. Outro desafio é o tempo de resposta, já que o parasitóide demora várias gerações até conseguir reduzir substancialmente as populações da praga [80], [85].

2.3.5.2 Monitorização e Deteção Precoce

A monitorização rigorosa das plantações é uma medida essencial para gerir a infestação da Vespa-das-galhas-do-castanheiro. A prática consiste em inspeções visuais regulares durante as épocas do ano mais favoráveis à praga, especialmente no período de formação de galhas, que ocorre na primavera. A deteção precoce permite que os produtores possam tomar medidas rápidas para evitar a proliferação massiva [80].

As inspeções geralmente envolvem a análise de amostras de ramos e folhas para identificar a presença de galhas, sendo que o tamanho e a densidade destas galhas são indicadores da gravidade da infestação. Em muitas regiões, esta prática é combinada com mapas de infestação, que ajudam a identificar as zonas mais afetadas e a priorizar as áreas para intervenções com o Torymus sinensis ou outras práticas de controlo [80], [85].

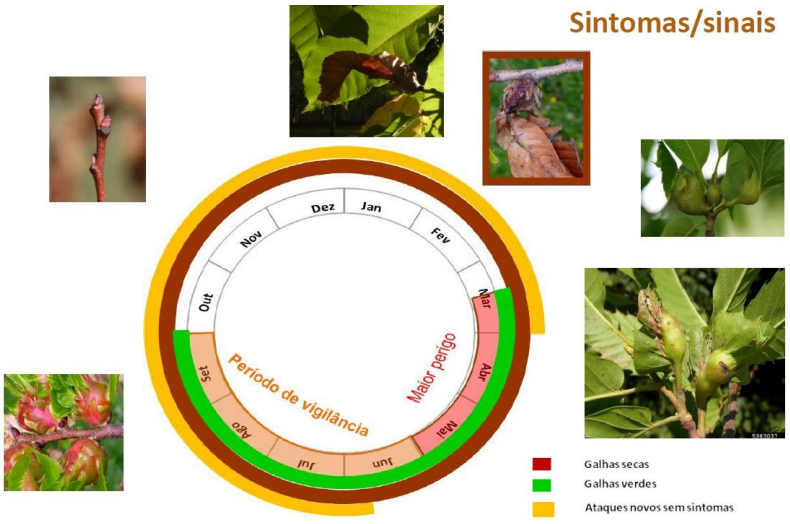


Figura 2.8- Período de deteção das diferentes fases e sintomas [81]

2.3.5.3 Podas e Remoção Mecânica de Galhas

Uma técnica complementar ao controlo biológico é a remoção manual ou mecânica de galhas, especialmente em pomares de menor dimensão. Durante o período de dormência da árvore, no outono e inverno, as galhas contendo larvas podem ser podadas e removidas, reduzindo assim a população de vespas para a estação seguinte.

Esta estratégia é prática para pequenas propriedades e áreas com baixa densidade de infestação, onde o esforço manual pode ser eficaz. No entanto, torna-se pouco viável em plantações de grande escala, devido ao trabalho intensivo necessário para inspecionar e remover as galhas. Além disso, a poda inadequada ou tardia pode comprometer a recuperação da planta, agravando o problema em vez de o resolver [85]. Também não é aconselhável remover as galhas afetadas, nas zonas onde o parasitóide já foi largado, uma vez que esse processo irá reduzir a sua proliferação [81].

2.3.5.4 Quarentena e Regulamentação

Para prevenir a propagação da praga para áreas não infestadas, foram implementadas medidas rigorosas de quarentena em muitos países. Em Portugal, por exemplo, a movimentação de material vegetal de castanheiro é regulamentada com base em certificações fitossanitárias, que verificam se as plantas estão livres de infestação antes de serem transportadas.

Estas regulamentações têm-se mostrado eficazes na contenção da praga, mas exigem uma fiscalização constante, especialmente em viveiros e áreas de produção de mudas. As barreiras regulamentares também envolvem a destruição de árvores gravemente infestadas, de forma a evitar a propagação da praga para novas regiões. Este tipo de intervenção é crucial em áreas onde o controlo biológico ainda não foi implementado ou não é viável [80], [85].

2.3.5.5 Investigação e Desenvolvimento

A pesquisa científica sobre a resistência genética dos castanheiros à praga é uma área promissora. Há esforços contínuos para desenvolver híbridos de castanheiros que sejam mais resistentes à *D. kuriphilus*, o que poderia reduzir a dependência de intervenções biológicas e manuais. Além disso, estão a ser exploradas novas técnicas de controlo químico que não afetem o ambiente nem os agentes biológicos de controlo.

Contudo, a investigação nesta área requer tempo e recursos, e os resultados ainda estão longe de serem implementados em larga escala. A adoção de castanheiros geneticamente

resistentes poderia alterar a dinâmica da produção de castanhas e reduzir a dependência de soluções como o Torymus sinensis, embora ainda seja necessária mais pesquisa para garantir que estas soluções sejam viáveis a longo prazo [80], [85].

3 Especificações do Sistema de Classificação Automática da Vespa-das-Galhas-do-Castanheiro em castanheiros infetados

Para que o projeto de classificação automática da vespa-das-galhas-do-castanheiro em castanheiros afetados funcione corretamente, várias etapas rigorosas devem ser seguidas, desde a aquisição de imagens até ao resultado final. Cada uma destas etapas assegura a captura adequada das imagens, o processamento correto, o treino eficaz do modelo e, por fim, a obtenção de previsões precisas para a classificação das doenças em árvores. Nos subcapítulos seguintes vai ser descrita de forma detalhada cada uma dessas etapas.

3.1 Requisitos do Sistema

Os requisitos do sistema são essenciais para garantir a precisão e a eficiência na tarefa de classificação automática de doenças em árvores.

Requisitos Funcionais:

- O sistema deve ser capaz de detetar doenças automaticamente em árvores a partir de imagens capturadas ou armazenadas.
- O modelo de classificação deve ser baseado no YOLOv8.2 ou similar, permitindo a deteção em tempo real ou próximo ao tempo real.
- O sistema deve ser capaz de **classificar múltiplas imagens de doenças** com precisão.

Requisitos Não Funcionais:

- Baixa latência no tempo de processamento das imagens.
- Interface simples e intuitiva para carregar imagens e obter resultados.
- O sistema deve ser escalável, permitindo a análise de grandes volumes de dados de imagens

3.2 Arquitetura do Sistema

Para realizar os procedimentos descritos no subcapítulo anterior, é necessário recorrer a um conjunto de ferramentas que irão formar todo o ecossistema necessário para o desenvolvimento e implementação da solução.

Uma vez que este projeto careceu de financiamento, a escolha do *hardware* e de *softwares* foi cuidadosamente planeada para garantir que são acessíveis e económicos. Por esta razão, optou-se por um hardware modesto e pela utilização de softwares de código aberto (*open source*), permitindo uma implementação eficiente e de baixo custo.

3.2.1 Componentes de *Hardware* (placa gráfica, Processador)

O *hardware* utilizado será um computador pessoal, com sistema operacional Windows 11 Home. As Características técnicas do computador são: Processador Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40 GHz, Memória RAM 16,0 GB DDR4L, placa gráfica NVIDIA GeForce GTX 1650 4GB, armazenamento 256GB SSD e 512GB HDD e *smartphone* Poco X6 Pro 5G de 64MP.

3.2.2 Componentes de *Software* (YOLOv8, LabelImg, Roboflow, YOLOv8 Test Interface)

YOLOv8.2: O modelo principal utilizado para a deteção de doenças é o YOLOv8.2, uma versão avançada e otimizada para deteção de objetos com alta precisão e rapidez. Este modelo é especializado em tarefas de *object detection*, onde consegue identificar múltiplos objetos numa imagem com alta eficiência. No contexto deste projeto, o YOLOv8.2 foi treinado para identificar visualmente diferente a doença da vespa-das-galhas-do-castanheiro em árvores, com base em padrões de galhas, manchas ou outras anomalias visuais características. Este modelo oferece uma boa combinação de rapidez e precisão, o que é essencial para a deteção automática em tempo real ou quase real[87].

LabelImg: O software LabelImg foi utilizado para a anotação manual das imagens. A anotação das imagens é uma etapa crucial, onde foram marcadas manualmente as áreas específicas das árvores que estão afetadas por doenças, como as galhas ou lesões nas folhas e ramos. As anotações criadas com o LabelImg contêm informações sobre as classes (doenças) e

as coordenadas das *bounding boxes* que definem as áreas afetadas. Estas anotações foram depois convertidas para o formato compatível com o YOLOv8.2, sendo normalmente o formato TXT, com a estrutura apropriada para o treino do modelo.

Roboflow: Após a anotação das imagens no LabelImg, as mesmas foram carregadas na plataforma Roboflow. O Roboflow permite realizar o pré-processamento das imagens e gerar *augmentation* (aumentação de dados) diretamente na plataforma, o que foi essencial para melhorar a diversidade e robustez do conjunto de dados. Com o Roboflow, foi possível aplicar transformações como rotação, redimensionamento, e ajustes de contraste e brilho nas imagens anotadas entre outras técnicas para obter um *dataset* maior. Estas *augmentations* são essenciais para garantir que o modelo YOLOv8.2 se torne mais robusto, sendo capaz de generalizar melhor para novos exemplos, que possam ter variações de iluminação, orientação ou outras condições no campo [88].

Google Colab: O treino do modelo foi realizado no Google Colab, uma plataforma que oferece recursos gratuitos de GPU. A utilização do Google Colab permite que modelos como o YOLOv8.2 sejam treinados com *datasets* grandes e complexos sem a necessidade de hardware próprio. A integração com o Google Drive facilitou o armazenamento e acesso aos conjuntos de dados, enquanto o uso das GPUs do Colab acelerou o processo de treino, reduzindo o tempo necessário para ajustar o modelo, a GPU fornecida pelo Google Colab foi NVIDIA T4 GPU 16GB, Python 3 Google Compute Engine backend (GPU), RAM: 1.22 GB/12.67 GB, Armazenamento: 32.28 GB/112.64 GB [89].

YOLOv8 Test Interface: Este programa é uma aplicação GUI (*Graphical User Interface*) e foi criada utilizando a biblioteca tkinter, para a interface e ultralytics, permitir a utilização de modelos YOLOv8 de forma simples e interativa, visando a deteção de objetos em imagens ou vídeos. Através de uma interface gráfica, o utilizador pode escolher um modelo pré-treinado, carregar ficheiros de imagem ou vídeo e visualizar as deteções em tempo real. O propósito é facilitar o uso de modelos de visão artificial, permitindo a troca de modelos e o controlo do processamento, sendo ideal para projetos de pesquisa e protótipos que envolvem análise visual.

3.3 Métodos de Captura e Armazenamento de Imagens

A primeira etapa para a criação do modelo, é a aquisição de imagens, que são a base do sistema. Estas imagens podem ser obtidas por diversas fontes, tais como câmaras de alta

resolução, drones para cobrir grandes áreas de plantações, ou até através da recolha de imagens disponíveis na internet. Neste caso, não foi possível proceder à captura de imagens no terreno devido ao facto de terem de ser coletadas em uma janela de tempo no período adequado. As imagens precisam de mostrar de forma clara as árvores e os sintomas visuais associados às doenças, como galhas, manchas ou outros sinais visíveis. A qualidade e a clareza destas imagens são fundamentais para o sucesso do sistema.

Devido à indisponibilidade para capturar imagens no terreno, no período adequado, foram utilizadas imagens encontradas na internet para construir o *dataset*. Estas imagens foram selecionadas para garantir uma diversidade suficiente de exemplos da doença em árvores, representando diferentes estágios de infeção [90].

As imagens foram posteriormente anotadas no LabelImg e processadas no Roboflow, sendo então o *dataset* armazenado no Google Drive e Roboflow para integração com o Google Colab. Esta abordagem facilitou o treino e a organização do conjunto de dados, além de garantir o armazenamento eficiente.

Para imagens sem doença, foi utilizado um *smartphone* Poco X6 Pro 5G de 64MP e o formato das imagens escolhido foi de 1:1 uma vez que é o formato padrão (640x640 pixels) do modelo YOLO.

3.4 Técnicas de anotação, processamento e análise de imagens

Em seguida, as imagens adquiridas passam por um processo de anotação, uma fase crítica para a criação de um *dataset* rotulado. Este processo é feito com a ajuda de ferramentas como o LabelImg, ou Roboflow. Durante a anotação, definem-se as classes de doenças e as coordenadas das áreas afetadas, assim os ficheiros de anotação são gerados num formato compatível com o YOLOv8.2, como o formato TXT. Esta fase é essencial para que o modelo seja capaz de aprender e reconhecer padrões específicos nas imagens [91].

Para anotações inicialmente foi utilizado o LabelImg, mas os *datasets* finais foram realizados com o Roboflow, pois como explicado anteriormente este último possui enumeras vantagens, especialmente utilizando a versão pro.

Figura 3.1- Interface LabelImg

Uma ferramenta que pode ser muito útil a nível de anotação é a função AI Label Assist do Roboflow, que utiliza como base, um modelo já treinado para dar sugestões de anotações e assim reduzir drasticamente o tempo demorado para realizar anotações.

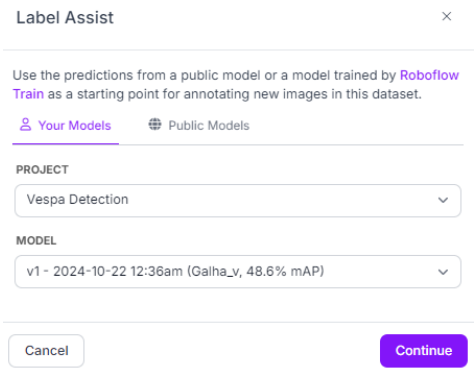


Figura 3.2- Roboflow interface onde é possível selecionar um modelo treinado pelo utilizador

Assim que o modelo é selecionado, está pronto a ser utilizado, ainda assim a supervisão e algumas correções das anotações é sempre necessária e aconselhada.

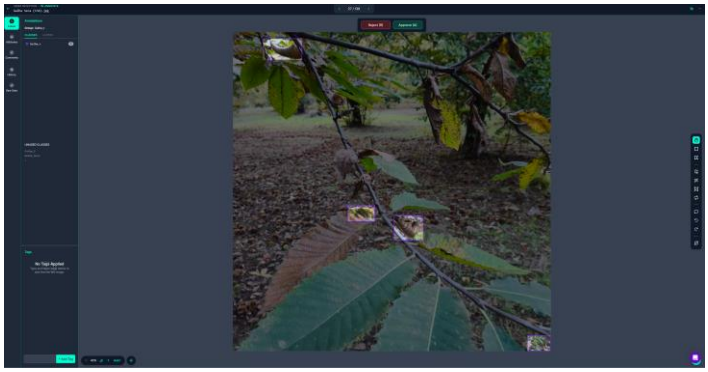


Figura 3.3- Interface de anotações do Roboflow

Depois das anotações efetuadas, é necessário separar o *dataset* em 3 tipos categorias (*Train*, *Valid*, *Test*), o Roboflow é uma ferramenta excelente para este processo, uma vez que só necessitamos de escolher qual a percentagem para cada categoria e a separação é automática, com a possibilidade de ser efetuada também manualmente se o utilizado quiser realizar alterações.

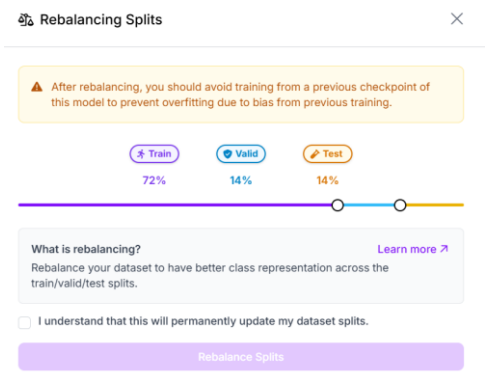


Figura 3.4- Interface para divisão do *dataset*

Em seguida, é possível realizar o pré-processamento das imagens, onde podem redimensionadas para um tamanho padronizado, por exemplo, aqui foi escolhido 640x640 pixels. Esta uniformização garante que o modelo recebe entradas consistentes durante o treino. Além disso, são aplicadas técnicas de normalização dos valores dos pixels, de modo a garantir que o treino ocorre de forma mais eficiente e são também utilizadas técnicas de *data augmentation [92]*.

O pré-processamento das imagens é uma etapa fundamental para garantir que os dados sejam adequados para o treino do modelo. Este pré-processamento foi realizado em duas etapas, primeiramente no Roboflow e posteriormente, também realizado diretamente no Google Colab, com diferentes técnicas já incorporadas no modelo de treino. As principais técnicas utilizadas foram as seguintes: *resize*, *normalization* e *data augmentation*.

As técnicas utilizadas incluem:

Resize das imagens para um tamanho padrão, como 640x640 pixels, para garantir consistência no input do modelo. Isto assegura que todas as imagens tenham dimensões uniformes, facilitando o processamento pelo YOLOv8.2.

Normalization dos valores dos pixels, para garantir que estejam numa faixa de valores adequada (por exemplo, entre 0 e 1), permitindo ao modelo convergir mais rapidamente durante o treino e melhorar a sua performance.

Data Augmentation aplicada através de operações como rotações, inversões, variações de brilho e contraste. Estas técnicas são essenciais para aumentar a diversidade do *dataset*,

criando várias versões da mesma imagem, o que ajuda o modelo a generalizar melhor para novos dados, evitando que se ajuste demasiado ao *dataset* de treino.

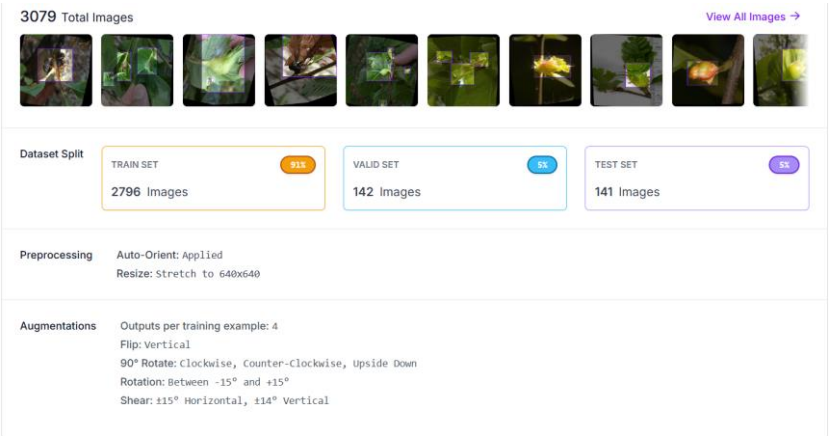


Figura 3.5- Exemplo de um *dataset* anotado no Roboflow com técnicas de *augmentation aplicadas*

Treino com Roboflow: O Roboflow tem também uma ferramenta integrada, não só de *augmentation*, mas também de treino, neste caso foi utilizado o Modelo de Arquitetura Roboflow 3.0 para treinar e os resultados serão discutidos no capítulo 4.

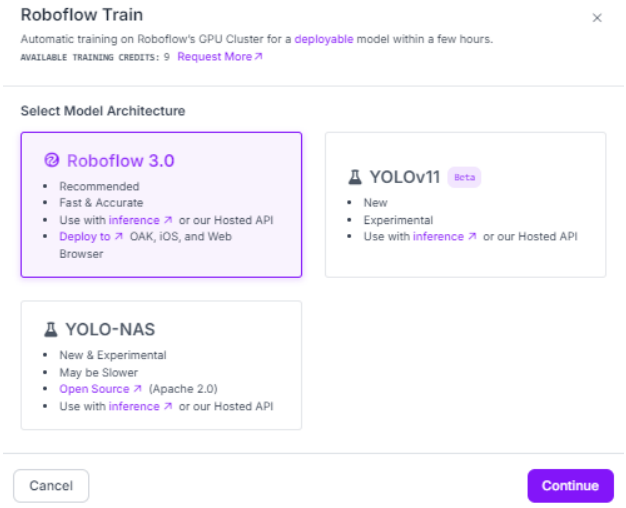


Figura 3.6- Possibilidades de treino com o Roboflow Train

Este treino foi muito importante, pois foi permitido rever todas as anotações de forma muito mais rápida e intuitiva devido ao modo de observação da *Confusion Matrix*, que ao contrário do modelo treinado no Google Colab, permite verificar diretamente as labels com previsões corretas e incorretas e corrigir as bounding boxes muito mais rapidamente. Também é possível analisar dados como *Precision*, *Recall*, *mAP*, *Box Loss* e todos os gráficos de treino.

No entanto, ainda não é possível parametrizar de forma tão personalizada como com o Google Colab, nem mesmo definir *early stopping*, o que torna este modo de treino menos eficiente no que diz respeito a evitar *overfitting*.

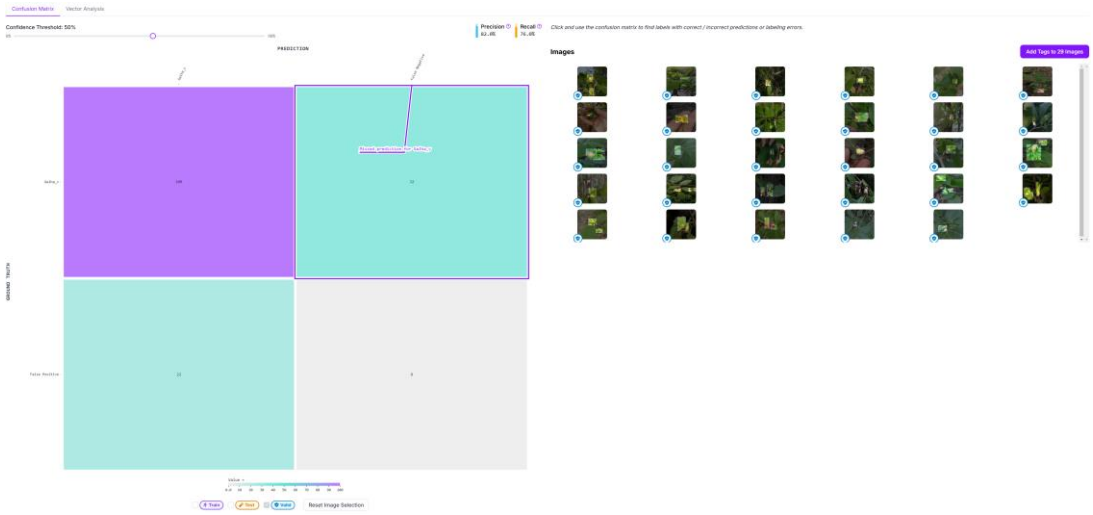


Figura 3.7- Exemplo de *Confusion Matrix* onde se podem encontrar as *labels* as previsões corretas / incorretas ou erros de *labeling*, com apenas as imagens de treino selecionadas.

Treino com Google Colab: Outra opção, depois das imagens anotadas e pré-processadas, ou não, é a possibilidade de exportar e utilizar esse mesmo *dataset* para treinar o modelo na nuvem utilizando o Google Colab, para isso pode ser utilizado o YOLOv8 *notebook* [93].

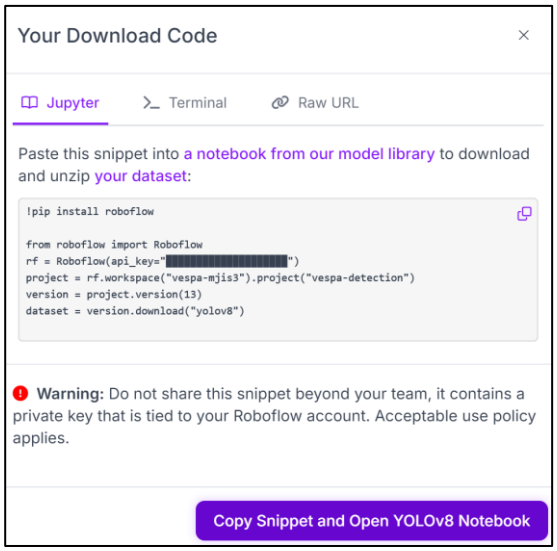


Figura 3.8- Código gerado com a informação relativa ao dataset para ser integrado com Google Colab para treino do modelo YOLOv8.2

Para treinar o modelo foram efetuadas várias combinações de testes e parâmetros de treino diferentes. Alguns dos primeiros modelos testados apresentavam resultados muito bons,

devido ao número reduzido de imagens anotadas comparadas com os resultados, levantaram algumas suspeitas de imagens duplicadas. O que levou a uma análise mais detalhada do *dataset*, depois de verificado o *dataset* com um software de deteção de duplicação de imagens (PictureEcho) [94], a conclusão foi que algumas das imagens de treino e validação estavam repetidas, o que levou a resultados equivocados e a ser necessária a revisão e novos testes com o *dataset* de imagens revisto.

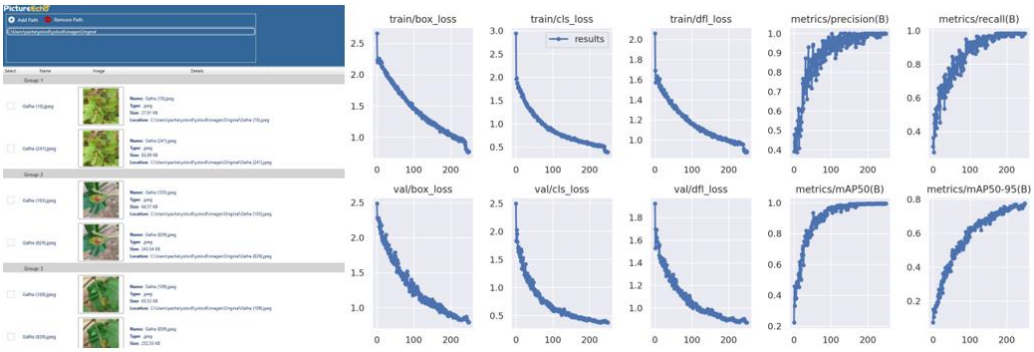


Figura 3.9- *Software* PictureEcho e exemplo de resultados de treino realizado com imagens duplicadas

Depois do *dataset* revisto e corrigido, de todos os testes, o que obteve melhores resultados, foi com o seguinte com os parâmetros:

```
      !yolo task=detect mode=train model=yolov8s.pt
data={dataset.location}/data.yaml epochs=100 imgsz=640 plots=True \
      patience=10 optimizer=AdamW lr0=0.01 weight_decay=0.0005 augment=True
save_period=10
```

Explicação de cada parâmetro:

task=*detect*: Define a tarefa como deteção de objetos.

mode=*train*: YOLOv8 em modo de treino.

model=*yolov8s.pt*: Usa o modelo YOLOv8 *small* pré-treinado como base.

data={*dataset.location*}/*data.yaml*: Especifica o ficheiro *data.yaml* com as informações do *dataset*.

epochs=100: Define o treino para 100 épocas.

imgsz=640: Configura as imagens para 640x640 *pixels*.

plots=*True*: Ativa gráficos de desempenho durante o treino.

patience=10: Interrompe o treino se a perda de validação não melhorar em 10 épocas (*early stopping*).

optimizer=AdamW: Usa o otimizador AdamW para melhor regularização.

lr0=0.01: Define a taxa de aprendizagem inicial para 0.01.

weight_decay=0.0005: Aplica uma regularização de *weight decay* para evitar *overfitting*.

augment=*True*: Ativa *data augmentation*, incluindo *flip* horizontal, redimensionamento, rotação, translação, alteração de cor, cisalhamento, mosaico e recortes.

save_period=10: Salva o modelo a cada 10 épocas.

Depois de atualizado o código que está disponível na íntegra no Capítulo 7 (Anexos), inicia-se a fase de treino do modelo. Neste projeto, como explicado anteriormente, o modelo utilizado é o YOLOv8.2. O treino é realizado no *Google Colab*, que fornece recursos de GPU, que permite acelerar o processo de treino e tornar o sistema mais eficiente. Durante o treino, o modelo aprende a reconhecer os padrões nas imagens anotadas, associando-os às respetivas classes de doenças [89]. Uma desvantagem do uso da versão free do Google Colab, é que só se pode utilizar a GPU por um máximo de 3 horas e 40 minutos consecutivos, passado esse tempo, é necessário, ou comprar unidades de computação ou esperar várias horas para poder utilizar o poder de computação da GPU de forma gratuita.

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     85/100      3.99G      1.668      1.332      1.687         31        640: 100% 175/175 [01:05<00:00,  2.67it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.31it/s]
                   all        142        132      0.635      0.711      0.698      0.241
EarlyStopping: Training stopped early as no improvement observed in last 10 epochs. Best results observed at epoch 75, best model saved as best.pt.
To update EarlyStopping(patience=10) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.

85 epochs completed in 1.601 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 22.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11,126,358 parameters, 0 gradients, 28.4 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:04<00:00,  1.07it/s]
                   all        142        132      0.687      0.766      0.752      0.284
               Galha_v        107        132      0.687      0.766      0.752      0.284
Speed: 0.4ms preprocess, 19.6ms inference, 0.0ms loss, 3.2ms postprocess per image
Results saved to runs/detect/train
💡 Learn more at https://docs.ultralytics.com/modes/train
```

Figura 3.10- Treino parou às 85 *epochs* para evitar *overfitting*

Depois do treino efetuado de cerca de 1 hora e 40 minutos, o modelo foi sujeito a uma fase de validação e teste. Nesta fase, utilizou-se um conjunto de dados de imagens diferente do que foi usado durante o treino, para avaliar o desempenho do modelo. As métricas utilizadas para medir o desempenho e os resultados serão apresentados e discutidos no Capítulo 4.

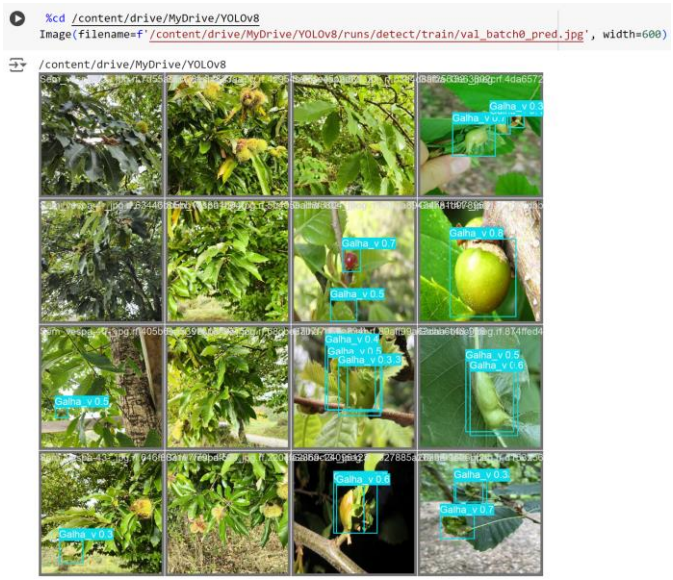


Figura 3.11- Dados de *inference* do *dataset* de validação

Finalmente, os resultados finais são obtidos e podem ser apresentados de forma visual, com as áreas afetadas destacadas nas imagens. Estes resultados incluem a classificação das doenças e as suas localizações nas imagens, assim como métricas associadas à precisão do modelo. Os resultados são armazenados numa base de dados ou num sistema de armazenamento na nuvem, como o *Google Drive* ou o *Amazon S3*, para possibilitar futuras análises, comparações e gestão das doenças em plantações de árvores [89]

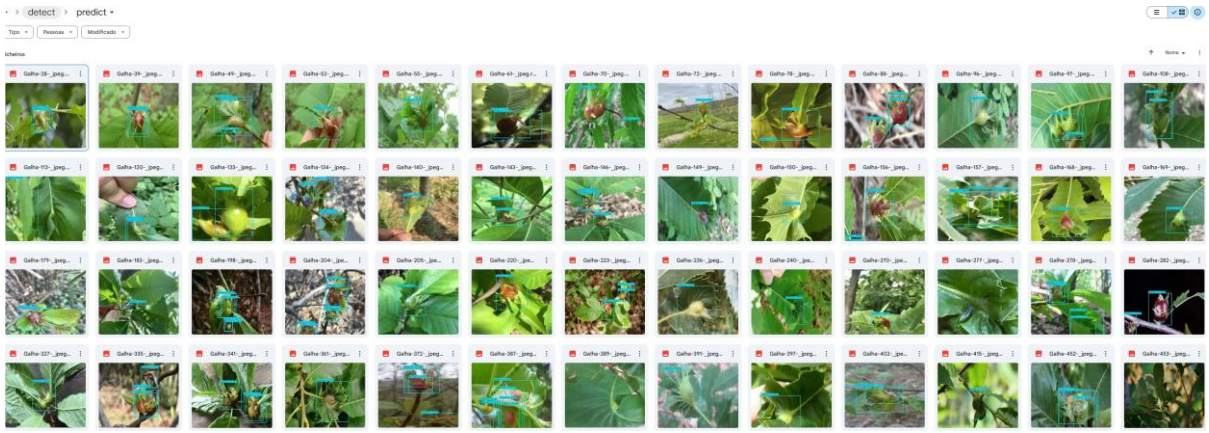


Figura 3.12- Resultados da previsão do modelo guardados no Google Drive

Depois desse passo, o modelo está pronto para a fase de *inference*, onde o modelo treinado é utilizado para fazer previsões em novas imagens ou vídeos. Neste caso, as imagens são carregadas na aplicação YOLOv8 Test Interface, que utiliza o YOLOv8.2 para identificar e classificar automaticamente as áreas afetadas pelas doenças. O sistema gera previsões em tempo real ou quase real, indicando a doença presente e em quais áreas da árvore ou plantação essa doença se manifesta [95].

3.5 YOLOv8 Test Interface

De uma forma sucinta, o programa começa por solicitar ao utilizador que selecione um modelo YOLO (.pt). Após o modelo escolhido, a interface principal é aberta, permitindo ao utilizador selecionar imagens para análise. Os resultados são apresentados, e o utilizador pode obter estatísticas sobre quantas imagens tiveram deteções. Em seguida, vai ser efetuada uma breve explicação das funcionalidades da aplicação YOLOv8 Test Interface:

Seleção de Modelo YOLO: O utilizador pode selecionar um modelo YOLOv8 (.pt) pré-treinado para ser utilizado nas inferências.

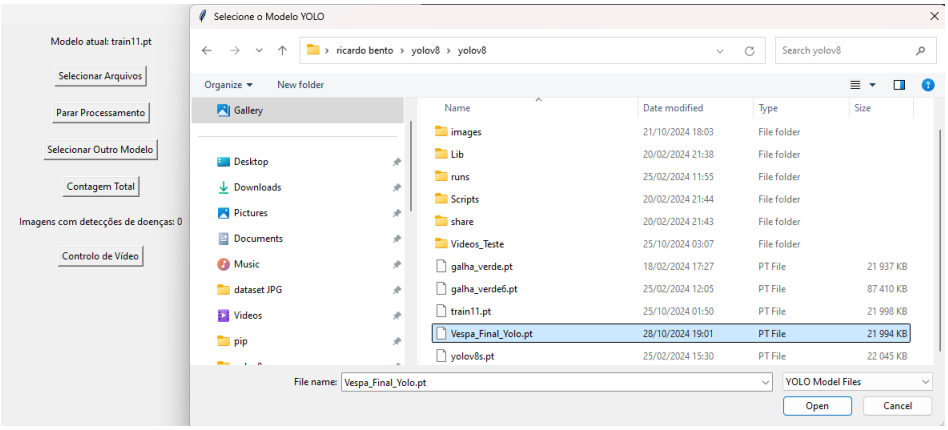


Figura 3.13- Exemplo da funcionalidade de como escolher o modelo YOLO

Interface Gráfica: A aplicação fornece uma interface onde o utilizador pode selecionar imagens para processamento, parar o processamento em curso e trocar o modelo YOLO.

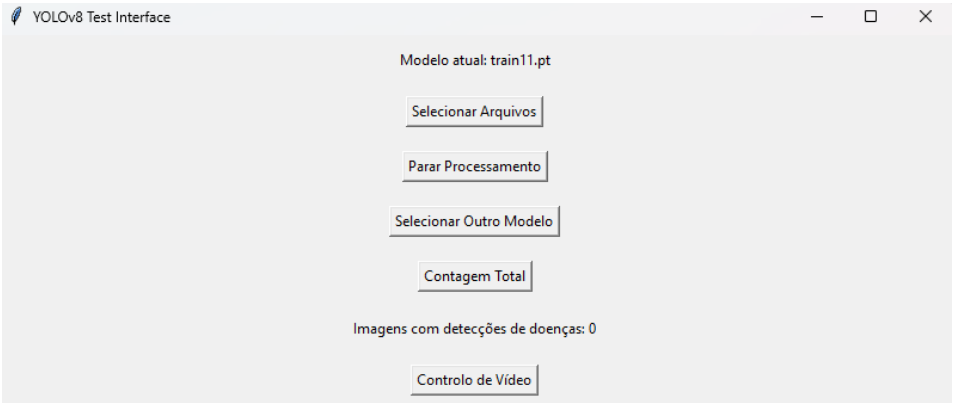


Figura 3.14- Interface gráfica com as opções disponíveis na aplicação

Processamento de Múltiplas Imagens: Permite o upload de várias imagens em simultâneo para serem processadas, mostrando a deteção diretamente de todas elas, uma a uma.

Parar Processamento: Um botão que permite interromper o processamento das imagens caso o utilizador decida parar a visualização.

Visualização dos Resultados em imagens: Os resultados das deteções são apresentados em tempo real, exibindo as imagens anotadas com as deteções feitas.

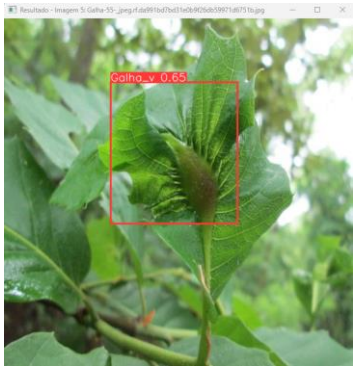
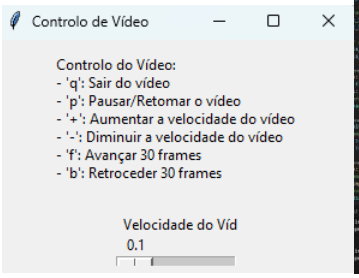


Figura 3.15- Exemplo da Visualização dos Resultados em uma imagem, com o modelo final YOLOV8.2

Visualização dos Resultados em vídeos: O programa permite o upload de imagens e vídeos, foi criado também uma instrução e um controlo de vídeo para facilitar a visualização e no caso de se querer captar uma imagem especifica, poder parar, avançar e diminuir a velocidade do vídeo. O código fonte pode sem encontrado no Capítulo 7.



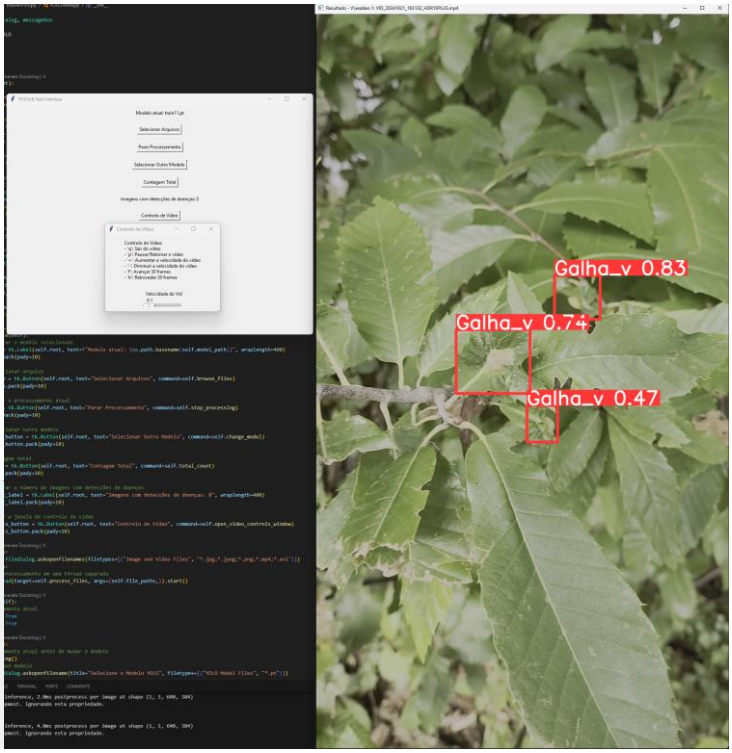


Figura 3.16- Menu de controlo de vídeo, com o modelo final YOLOV8.2

Figura 3.17- Visualização de resultados em um vídeo de teste

Contagem Total: Apresenta o total de imagens verificadas e o número de deteções encontradas.

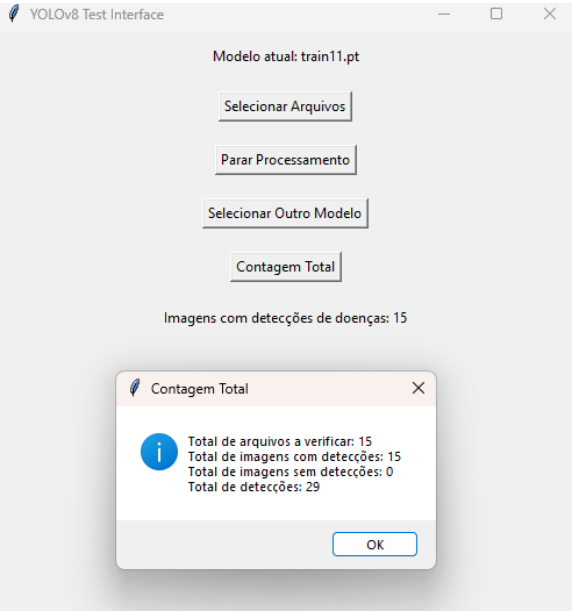


Figura 3.18- Exemplo de Contagem Total

4 Resultados e Discussão

No capítulo seguinte vão ser discutidos e analisados os resultados obtidos durante o trabalho, as métricas utilizadas para essa análise, assim como as suas implicações e o potencial económico da ideia. O foco será na avaliação do *dataset* final treinado com o modelo YOLOv8.2 e *augmentation*, que foi o que melhor desempenho apresentou. Será feita a comparação com e sem *augmentation.* E serão discutidos os resultados do treino no Roboflow 3.0 e YOLOv8.2.

Para esta dissertação foram testadas dezenas de combinações diferentes e foram efetuados testes intermédios, a comparação desses resultados com os finais, não seria justa, uma vez que em cada iteração, os erros foram corrigidos e o *dataset* foi alterado e melhorado, nomeadamente com o uso do Roboflow 3.0 como explicado no capítulo anterior. Essa comparação poderia levar a uma interpretação errada de resultados, no entanto esses resultados podem ser consultados nos anexos (Capítulo 7.1).

4.1 Validação de Modelos

A validação de modelos é um processo fundamental para assegurar que o sistema desenvolvido possui um desempenho satisfatório e adequado aos objetivos da deteção de infestações por Vespa-das-galhas-do-castanheiro. Neste estudo, foram considerados vários critérios e métodos de validação, visando uma análise abrangente e rigorosa da eficácia do modelo, temos então os resultados de treino e de validação:

Resultados de Treino: Os resultados de treino mostram o desempenho do modelo nos dados com os quais ele aprendeu. Valores elevados de *precision* e *recall* nos dados de treino indicam que o modelo consegue capturar bem os padrões do dataset de treino, mas esses valores não refletem necessariamente a sua capacidade de generalizar para novos dados.

Focar demasiado nos resultados de treino pode levar a uma interpretação errada do desempenho, especialmente se o modelo estiver a sobre ajustar-se aos dados de treino (*overfitting*), aprendendo ruídos ou detalhes específicos em vez dos padrões gerais.

Resultados de Validação: Os dados de validação são independentes do treino, servindo para avaliar o modelo em situações mais próximas das reais. Um bom desempenho nos dados de validação indica que o modelo consegue generalizar e detetar infestações em novos dados, o que é essencial para garantir a sua aplicabilidade.

As métricas de validação, como *precision*, *recall*, F1 Score e mAP, mostram de forma mais confiável se o modelo está pronto para uso em campo ou se precisa de ajustes para melhorar a capacidade de generalização.

Dito isto, deve sempre avaliar-se o desempenho do modelo com base nos resultados de validação para garantir que ele é eficaz em dados que não foram vistos antes, o que é crucial para a sua aplicação prática.

4.1.1 Critérios e métodos de validação YOLOv8.2 e Roboflow 3.0

Os principais critérios usados para validar o modelo foram a ***precision***, ***recall***, **F1 *Score*** e **mAP (*Mean Average Precision*)**. Estes critérios foram selecionados por serem amplamente utilizados na área de visão computacional, oferecendo uma visão clara da capacidade do modelo em distinguir corretamente a classe "Galha_v" do fundo.

- **Precisão (*Precision*)**: Mede a proporção de deteções corretas em relação ao total de previsões realizadas. No caso do modelo, a *precision* alcançada foi de 59.8% para a classe "Galha_v", conforme observado na *Confusion Matrix*. Isso indica que, das previsões que o modelo realizou como infestações, 59.8% estavam corretas.
- ***Recall***: Mede a capacidade do modelo de detetar todas as infestações presentes no conjunto de dados. O modelo alcançou um recall de 83.3%, o que sugere que, embora o modelo seja preciso, há ainda uma parte das infestações que não está a ser corretamente detetada.
- **F1 *Score***: Este é o valor que combina *precision* e *recall*, sendo particularmente útil para avaliar modelos em situações onde há um desbalanceamento entre classes. A curva F1-*Confidence* indica que o valor máximo de F1 (0.71) foi atingido com uma confiança de aproximadamente 0.416. Este valor oferece um equilíbrio entre *precision* e *recall*, útil para afinar o modelo.
- **mAP (*Mean Average Precision*)**: A média de precisão a vários limiares (mAP@0.5 e mAP@0.5:0.95) é uma métrica robusta para a avaliação de modelos de deteção de objetos. Neste caso, o mAP@0.5 é de 0.746, o que indica uma boa capacidade de deteção com um critério de sobreposição de 50%. No entanto, o valor tende a baixar com limiares mais altos, refletindo o desafio de uma localização precisa.

- ***Confusion Matrix*** revelou que o modelo detetou corretamente 110 instâncias de Galha_v, mas também gerou 74 falsos positivos e 22 falsos negativos. Esses erros destacam áreas onde o modelo ainda tem dificuldades, especialmente em diferenciar infestações de elementos de fundo, sugerindo a necessidade de ajustes na *data augmentation* e no limiar de confiança para melhorar a *precision*.

4.1.2 Testes de eficácia e precisão YOLOv8.2

Para testar a eficácia do modelo, foram realizados diversos ensaios com um conjunto de validação, aplicando diferentes níveis de confiança e observando as variações nos resultados de *precision* e *recall*. Os testes mostraram que, ao ajustar o limiar de confiança para 0.416, o modelo maximiza o F1 Score, atingindo um equilíbrio entre *precision* e *recall*.

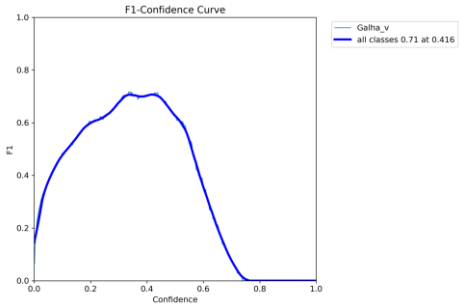


Figura 4.1- F1-Confidence Curve, do modelo YOLOv8.2 com augmentation

Os gráficos de *Precision-Confidence* e *Recall-Confidence* são particularmente elucidativos. A curva de *Precision-Confidence* mostra que a precisão do modelo atinge o seu valor máximo (1.00) com uma confiança de 0.693, mas este valor reduz significativamente o *recall*. O *trade-off* entre *precision* e *recall* é evidente, e a escolha de um limiar ótimo de confiança depende do objetivo da aplicação: se a prioridade for evitar falsos negativos (não detetar uma infestação presente), então um limiar mais baixo pode ser preferível. Caso o objetivo seja minimizar falsos positivos, um limiar mais alto é recomendado.

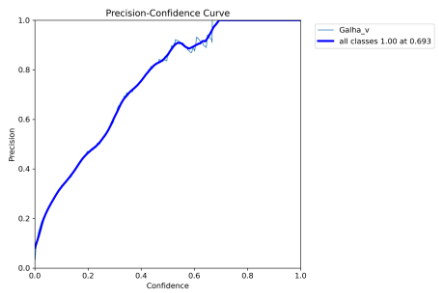


Figura 4.2- Precision-Confidence Curve, do modelo YOLOv8.2 com augmentation

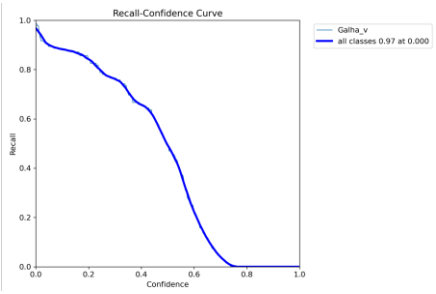


Figura 4.3- Recall-Confidence Curve, do modelo YOLOv8.2 com augmentation

4.2 Resultados da Deteção de Infestações YOLOv8.2

Os resultados da deteção de infestações pelo modelo desenvolvido foram analisados em termos de métricas de desempenho e exemplos visuais de deteções corretas e incorretas.

4.2.1 Resultados das Métricas YOLOv8.2

Olhando e avaliando os dados de validação, o modelo apresenta uma *precision* de 58.9% e um *recall* de 83.3% para a classe "Galha_v". Em baixo a explicação detalhada dos cálculos.

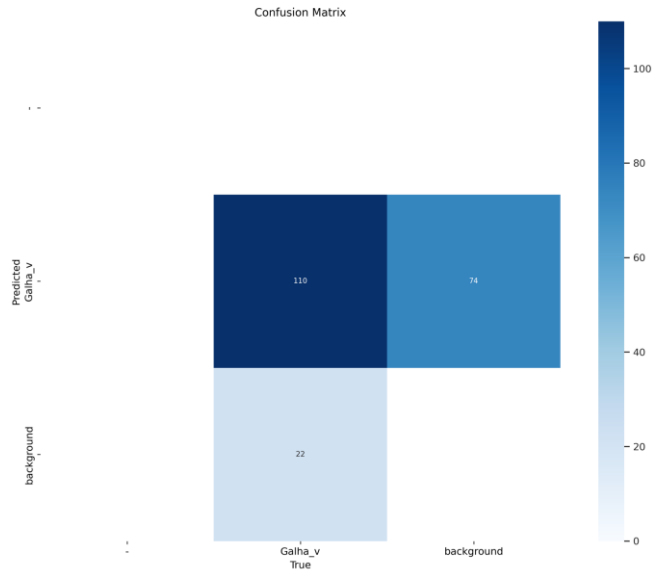


Figura 4.4- *Confusion Matrix*, do modelo YOLOv8.2 com augmentation

Olhando para a *Confusion Matrix* temos:

- TP (*True Positives*) para Galha_v: 110
- FP (*False Positives*) para Galha_v: 74
- FN (*False Negatives*) para Galha_v: 22

Cálculo da *Precision*

A fórmula da *Precision* é:

$$Precision = \frac{TP}{TP + FP}$$

Substituindo os valores temos:

$$Precision = \frac{110}{110 + 74} = \frac{110}{184} \approx 0.598\ ou\ 59.8\%$$

Cálculo *Recall*

O cálculo do *Recall* é:

$$Recall = \frac{TP}{TP + FN}$$

Substituindo os valores temos:

$$Recall = \frac{110}{110 + 22} = \frac{110}{132} \approx 0.833\ ou\ 83.3\%$$

A *precision* de 59.8% indica que, das previsões de "Galha_v" realizadas pelo modelo, 59.8% foram corretas. O *recall* de 83.3% indica que o modelo conseguiu identificar 83.3% das infestações reais presentes no conjunto de dados.

O F1 Score máximo de 0.71 foi alcançado com uma confiança de 0.416. A média de precisão (mAP@0.5) de 74.6% indica que o modelo possui uma boa capacidade de deteção global, mas com margem para melhorias.

A curva *Precision-Recall* mostra o desempenho do modelo na deteção da classe Galha_v, indicando uma elevada *precision* inicial que diminui à medida que o *recall* aumenta. No início da curva, o modelo mantém uma *precision* próxima de 1.0 com valores baixos de *recall*, sugerindo que as previsões mais confiantes são altamente precisas. Contudo, à medida que o recall aumenta, o que representa uma tentativa do modelo de captar mais instâncias de Galha_v, a precisão cai significativamente, refletindo um aumento nos falsos positivos. O valor de mAP@0.5 de 0.746 indica uma precisão média satisfatória para um limiar de sobreposição de 50%, mas sugere que o modelo tem espaço para melhorar, especialmente em manter a *precision* ao aumentar o *recall*. Em resumo, o modelo é eficaz na identificação inicial de infestações com alta confiança, mas compromete a precisão ao expandir o alcance para

maximizar o *recall*, sendo necessário encontrar um limiar de confiança ideal de acordo com a prioridade entre precisão e recall.

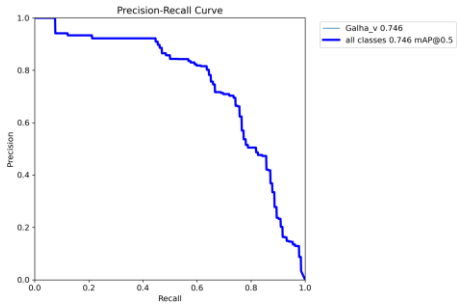


Figura 4.5- Precision Recall Curve, do modelo YOLOv8.2 com augmentation

4.2.2 Resultados das Métricas Roboflow 3.0

Vão ser agora analisados os dados de treino e validação do modelo Roboflow 3.0

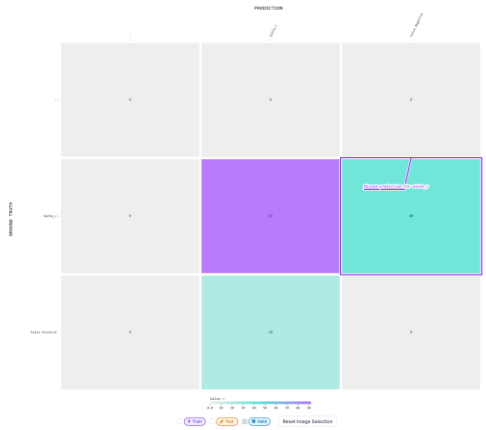


Figura 4.6- *Confusion Matrix*, do modelo Roboflow 3.0 com augmentation

Efetuando os mesmos cálculos, os valores indicam que o modelo tem uma precisão de 82.1%, o que significa que, das previsões feitas como Galha_v, 82.1% estavam corretas. O recall de 69.7% sugere que o modelo identificou 69.7% de todas as instâncias reais de Galha_v, indicando que há uma parte das infestações que não está a ser detetada.

Infelizmente o nível de informação fornecida depois do treino é mais limitada no Roboflow 3.0.

4.2.3 Exemplos Visuais de Sucesso e Erro

A análise visual dos resultados fornece uma visão adicional sobre a eficácia do modelo. As imagens de validação com previsões mostram que o modelo deteta corretamente muitas das

infestações em condições de iluminação e foco favoráveis. No entanto, observa-se que, em imagens com sombras, folhas sobrepostas ou com as galhas parcialmente ocultas, o modelo apresenta uma maior taxa de falsos negativos ou identifica elementos incorretamente como Galha_v (falsos positivos).

Figura 4.7- Comparação das anotações manuais val_batch2_labels e da previsão de validação val_batch2_pred, do modelo YOLOv8.2 com augmentation

Figura 4.8- Comparação da previsão com Roboflow 3.0 (esquerda) e de anotações manuais (direita)

Estes exemplos visuais ilustram a necessidade de melhorar a robustez do modelo para cenários mais desafiantes. O uso de um conjunto de dados mais diversificado, incluindo condições ambientais variadas, pode ser uma solução para reduzir esses erros.

4.3 Análise dos dados coletados

A análise dos dados do modelo YOLOv8.2 revelam alguns padrões importantes sobre a distribuição das infestações de Vespa-das-galhas-do-castanheiro e a capacidade do modelo de as identificar.

Precision (59.8%): A precisão do modelo indica que, das instâncias que foram identificadas como Galha_v (infestações), apenas 59.8% eram realmente infestações. Este valor

relativamente baixo sugere que o modelo tem uma tendência a gerar falsos positivos, ou seja, a classificar elementos de fundo como infestações. Esta taxa de falsos positivos pode ser problemática em contextos onde é importante minimizar alertas falsos, pois aumenta a necessidade de verificação manual das deteções. Tal facto pode ter acontecido, porque as imagens de fundo (sem anotações), foram tiradas numa altura que já havia ouriços. Como tal esse facto pode estar a influenciar negativamente o modelo.

Recall (83.3%): O *recall* do modelo indica que ele conseguiu identificar 83.3% das infestações reais. Este valor é bastante elevado e reflete uma boa capacidade do modelo em detetar infestações presentes nas imagens, mas há ainda 16.7% das infestações que não foram detetadas (falsos negativos). Um recall elevado é geralmente desejável em sistemas onde o objetivo é garantir a máxima deteção possível de infestações, mesmo que com alguns falsos positivos. Tendo em conta que a doença nunca se manifesta apenas em uma folha ou galho, o facto de haver falsos negativos, na prática, não afetaria o modelo, porque algumas galhas adjacentes, estatisticamente falando, continuariam a ser detetadas.

4.3.1 Comparação entre YOLOv8.2 e Roboflow 3.0

Para avaliar o desempenho dos modelos treinados para deteção de infestações na classe Galha_v, foi realizada uma comparação entre duas abordagens distintas: o YOLOv8.2 (Modelo 1) e o Roboflow 3.0 (Modelo 2). Cada modelo possui características específicas que influenciam a sua eficácia em termos de precisão e recall, permitindo uma análise mais profunda das suas vantagens e limitações.

YOLOv8.2: O YOLOv8.2 é amplamente reconhecido pela sua eficiência em tarefas de deteção em tempo real. Neste estudo, o modelo apresentou uma precisão de 59.8% e um recall de 83.3%. Estes resultados indicam que o YOLOv8.2 é altamente eficaz a capturar a maioria das infestações reais de Galha_v (elevado recall), tornando-o ideal para cenários onde a prioridade é maximizar a deteção. No entanto, a precisão relativamente baixa sugere que o modelo gera um número considerável de falsos positivos, ou seja, identifica o background como Galha_v em várias situações.

Apesar de ser um modelo robusto para identificar um grande número de infestações, a quantidade de falsos positivos pode tornar-se problemática, especialmente em casos onde os alarmes falsos são indesejáveis. Ainda assim, o YOLOv8.2 é uma excelente opção quando o objetivo é garantir que a maioria das infestações seja detetada, mesmo que haja necessidade de filtrar manualmente alguns falsos positivos.

Roboflow 3.0: O modelo Roboflow 3.0, por outro lado, foi otimizado para obter uma maior precisão nas suas previsões, com um valor final de 82.1% de precisão, enquanto o recall é de 69.7%. Esta abordagem torna-o mais seletivo na identificação de Galha_v, resultando em menos falsos positivos em comparação com o YOLOv8.2. A alta precisão implica que, quando o Roboflow 3.0 identifica uma infestação, há uma maior probabilidade de que essa deteção esteja correta, tornando-o ideal para aplicações onde a confiabilidade das previsões é essencial.

No entanto, o compromisso para alcançar esta precisão elevada reflete-se num recall mais baixo, o que significa que o Roboflow 3.0 não deteta uma parte significativa das infestações reais (falsos negativos). Este comportamento pode ser uma limitação em contextos onde é crucial garantir a máxima abrangência na deteção.

4.3.2 Distribuição Geográfica das Infestações

Para compreender a distribuição espacial das infestações, foram analisados os locais das deteções corretas e incorretas. Embora não tenhamos dados geográficos exatos nesta análise, observa-se que o modelo funciona melhor em áreas com condições homogêneas de luz e pouca interferência visual. Esses dados indicam a necessidade de adaptar o modelo para uma aplicação em campo, onde fatores como variações de luz e densidade foliar podem impactar a deteção.

4.3.3 Padrões Identificados

Foi identificado que o modelo responde bem a imagens com boa iluminação e galhas visíveis. Imagens em condições adversas (como sombras ou galhas parcialmente ocultas) são onde o modelo apresenta maior dificuldade. Este padrão sugere que futuras versões do modelo poderiam beneficiar de uma *data augmentation* que simule condições de iluminação variadas.

O facto de diferentes tipos de estágio da doença terem várias cores e a doença se manifestar, tanto nas galhas como nas folhas, pode ter afetado o treino porque devido ao número reduzido de imagens foi criada apenas uma *label* (Galha_v), o ideal seria separar cada tipo de "defeito" em uma categoria distinta, se o número de imagens assim o permitisse.

4.4 Eficácia das técnicas utilizadas e comparação de modelos

A eficácia das técnicas utilizadas no treino do modelo foi avaliada tanto em termos de *data augmentation* quanto na definição do limiar de confiança para maximizar a precisão e recall.

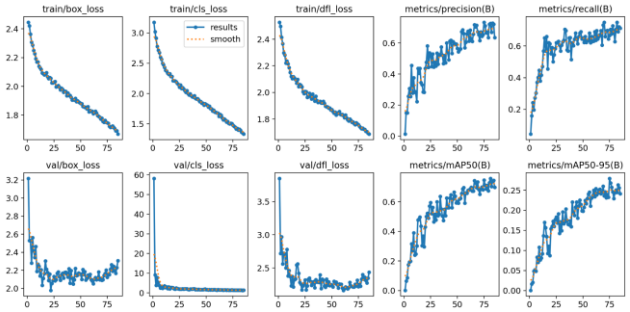


Figura 4.9- Treino em 85 *Epochs*, do modelo YOLOv8.2 com *augmentation*, consultar Capítulo 7.1 para detalhes.

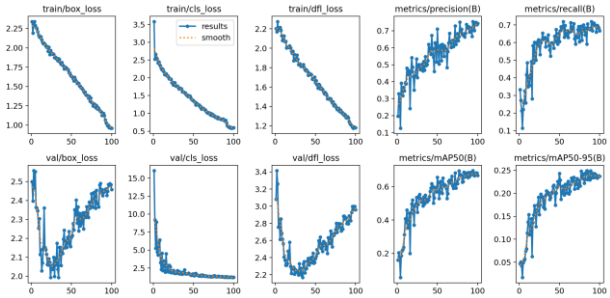


Figura 4.10- Treino em 100 *Epochs YOLOv8.2* sem *data augmentation*, consultar Capítulo 7.1 para detalhes.

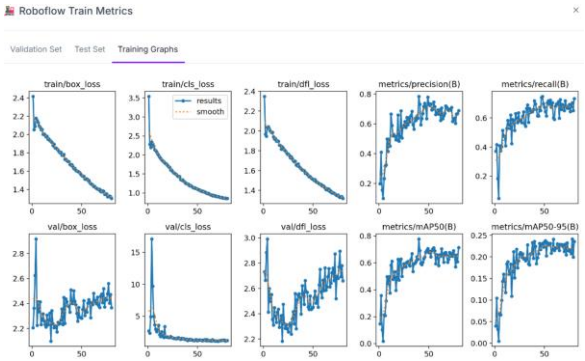


Figura 4.11- Treino em 80 *Epochs* Roboflow com técnicas de *data augmentation*, Capítulo 7.1 para detalhes

4.4.1 Impacto da *Data Augmentation*

As técnicas de data *augmentation* aplicadas — incluindo *flip*, rotação, alteração de cor e mosaico — definitivamente ajudaram a aumentar a diversidade dos dados e a melhorar a capacidade do modelo de generalizar para diferentes condições. Na Figura (4.9) podemos verificar no gráfico de val/box_loss, um aumento ao fim de 40 *epochs*, o que mostra um sinal

claro de *overfitting*, algo que não se verifica na Figura (4.8), em que o *dataset* tem vários parâmetros de *augmentation* definidos. Na Figura (4.10), também se verifica alguma tendência para *overfitting.*

Outro facto que se verifica é que no modelo sem augmentation, olhando para os dados de validação, o modelo não consegue generalizar nem detetar minimamente nenhuma imagem de doença e identifica tudo como fundo, isso deve-se ao facto do *overfitting* verificado anteriormente, o que o torna completamente inviável.

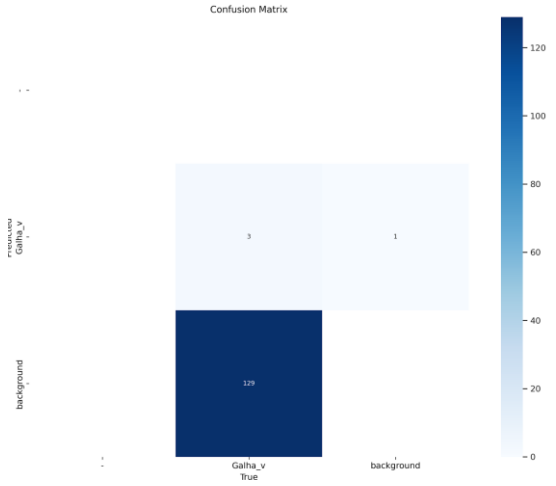


Figura 4.12- Confusion Matrix YOLOv8.2 sem augmentation

4.4.2 Escolha do Limiar de Confiança

A escolha do limiar de confiança revelou-se crucial para atingir o equilíbrio ideal entre *precision* e *recall*. A análise das curvas de F1-*Confidence* e *Precision-Confidence* sugere que um limiar de confiança em torno de 0.4-0.5 é ideal para um compromisso entre minimizar falsos positivos e maximizar o *recall.*

4.4.3 Limitações e Sugestões de Melhoria

Embora os modelos tenham apresentado bons resultados, a análise indica que a inclusão de mais dados em condições desafiantes pode melhorar o desempenho. Além disso, a experimentar outras arquiteturas de rede ou técnicas de regularização poderá fornecer melhorias adicionais.

O treino dos modelos pode ter sido afetado pela variedade de cores e complexidade visual que a doença apresenta, já que ela se manifesta de maneiras diferentes nas galhas e nas

folhas, dependendo do estágio. Devido ao número limitado de imagens do *dataset*, houve a necessidade de ser criada apenas uma *classe*, "Galha_v," para identificar a presença da doença, porque de outra forma as *classes* ficariam muito desequilibradas em termos de imagens. Esta escolha, embora prática, pode ter limitado o modelo, que acaba por "ver" todas as manifestações da doença como se fossem iguais. Idealmente, com mais imagens, seria possível criar categorias distintas para cada tipo de "defeito" ou estágio, o que permitiria ao modelo diferenciar melhor cada situação e responder de forma mais precisa. Essa abordagem mais detalhada ajudaria muito na deteção e monitorização da doença, oferecendo informações mais ricas e, assim, melhorando a gestão de pragas ao longo do tempo.

4.5 Comparação com métodos tradicionais

Os modelos de visão computacional, como **YOLOv8.2** e **Roboflow 3.0**, apresentam vantagens significativas em relação aos métodos tradicionais de deteção de infestações, como inspeções manuais.

Precisão e Confiabilidade: O YOLOv8.2, com um recall elevado (83.3%), é mais abrangente na deteção, mas com uma precisão mais baixa (59.8%), gerando mais falsos positivos. O Roboflow 3.0, por outro lado, oferece maior precisão (82.1%), reduzindo falsos positivos, embora o seu recall seja menor (69.7%), o que implica que algumas infestações podem não ser detetadas.

Eficiência e Custo: A utilização de drones com estes modelos permite inspeções rápidas e frequentes em grandes áreas, reduzindo os custos de mão-de-obra associados aos métodos tradicionais e permitindo monitorização contínua. Em comparação, as inspeções manuais são demoradas, dispendiosas e sujeitas a erro humano.

Deteção Precoce e Escalabilidade: Os modelos YOLOv8.2 e Roboflow 3.0 possibilitam uma deteção precoce mais consistente e são escaláveis para grandes plantações. Enquanto os métodos tradicionais enfrentam dificuldades em escalabilidade e monitorização contínua, os modelos de visão computacional podem ser facilmente ajustados e aplicados em qualquer contexto, proporcionando flexibilidade e adaptabilidade.

Cobertura Contínua e Flexível: Ambos os modelos podem ser programados para realizar inspeções de forma contínua ou em intervalos frequentes, possibilitando uma monitorização mais detalhada e um controle mais rápido da propagação de infestações.

4.6 Implicações práticas dos resultados

A escolha entre YOLOv8.2 e Roboflow 3.0 depende fortemente dos objetivos específicos da aplicação. O YOLOv8.2, com o seu recall mais alto, é preferível em situações onde é fundamental detetar o maior número possível de infestações, mesmo que isso implique lidar com mais falsos positivos. Esta característica torna-o adequado para inspeções em larga escala, onde a prioridade é garantir uma cobertura abrangente da área de estudo.

Por outro lado, o Roboflow 3.0, com a sua precisão mais elevada, é indicado para aplicações que exigem um nível elevado de confiança nas deteções. Com menos falsos positivos, o Roboflow 3.0 é particularmente útil em contextos onde alarmes falsos podem ter consequências significativas, como no planeamento de intervenções específicas. No entanto, o compromisso pela precisão resulta num recall mais baixo, o que pode ser uma desvantagem quando a deteção abrangente de infestações é crucial.

4.7 Potencial económico da Ideia

A implementação de um sistema de deteção de infestações com recurso a drones e modelos de visão computacional, como o YOLO ou Roboflow, representam uma inovação significativa para a gestão agrícola. Este sistema oferece potencial económico a longo prazo, especialmente quando o custo inicial é diluído ao longo do tempo e partilhado entre vários utilizadores. Neste capítulo, são discutidas estratégias para maximizar a viabilidade económica desta tecnologia, analisando formas de diluir os custos e avaliar o retorno do investimento. Um hectare de souto pode produzir entre 1000 a 3000 kg de castanhas por ano, o que significa um valor de 1000 a 8000€ por hectare dependendo dos anos. [86]

4.7.1 Diluição do Custo dos Drones ao Longo do Tempo

Investir em drones de qualidade e software de visão computacional representa uma despesa inicial significativa. No entanto, é possível diluir este custo ao longo de vários anos, aproveitando a longevidade do equipamento e atualizações regulares de software, que garantem o desempenho sem necessidade de substituir o hardware frequentemente.

Custo do Equipamento e Manutenção: O preço de um drone adequado para a agricultura, equipado com câmaras de alta resolução e sensores, pode variar entre 3,000€ e

10,000€. Este valor cobre a aquisição inicial, e a vida útil de um drone bem mantido ronda os 3 a 5 anos, dependendo das condições de uso e do tipo de manutenção. Para prolongar a vida do equipamento, a manutenção anual, que inclui ajustes, calibração e reparação de componentes desgastados, tem um custo estimado entre 200€ e 500€.

Atualizações de *Software*: Para maximizar o retorno do investimento, é vantajoso focar as melhorias no *software* de visão computacional, em vez de substituir o drone. Atualizações periódicas do modelo YOLOv8.2 podem aumentar a precisão da deteção e melhorar a funcionalidade do sistema, com um custo anual estimado entre 500€ e 1,500€. Este tipo de atualização permite que o equipamento continue eficaz ao longo do tempo, diluindo o custo inicial de aquisição.

Custo de Operação por Inspeção: O custo de operação de um voo completo, que cobre 100 hectares, varia entre 500€ e 1,500€ por inspeção, incluindo mão-de-obra e manutenção do equipamento. Cada voo tem uma duração de 10 a 15 horas, o que permite monitorizar áreas extensas em tempo relativamente curto. Com quatro inspeções anuais, realizadas durante a época crítica de infestações, o custo anual de operação é estimado entre 2,000€ e 6,000€**.**

4.7.2 Diluição do Custo pelo Varrimento de Toda a Área

A capacidade dos drones de monitorizar grandes áreas de forma rápida e eficiente é uma vantagem importante para diluir os custos de operação entre os utilizadores. A partilha do serviço entre múltiplos agricultores permite que o custo médio por hectare seja reduzido, tornando o sistema mais acessível.

Economia de Escala: Quando o sistema cobre uma área extensa (como 100 hectares), o custo de operação torna-se proporcionalmente menor para cada hectare. Para quatro inspeções anuais, o custo médio por hectare fica entre 20€ e 60€, um valor que pode ser facilmente repartido entre os utilizadores, beneficiando cada agricultor com uma monitorização frequente a um custo acessível.

Cobrança por Inspeção Periódica: Para otimizar o custo e assegurar a previsibilidade das despesas, é vantajoso estabelecer um plano de inspeções regulares. Assim, cada agricultor pode pagar uma pequena taxa por hectare, entre 50€ e 150€ por inspeção, garantindo a cobertura de toda a área durante o período de maior risco de infestação.

4.7.3 Estrutura Cooperativa e Venda de Serviço a Clientes Externos

Para tornar o sistema ainda mais viável, a criação de uma cooperativa agrícola representa uma solução prática e economicamente vantajosa. Com uma estrutura cooperativa, os agricultores partilham os custos e beneficiam de uma monitorização contínua e eficiente, enquanto a cooperativa gera receitas adicionais ao vender o serviço de deteção a clientes externos.

Formação de Cooperativas para Aquisição e Operação dos Drones: Numa cooperativa, os agricultores podem partilhar os custos de aquisição e operação do drone. Esta estrutura é ideal para pequenos e médios produtores, que assim têm acesso à tecnologia sem precisar de suportar o custo total. Um plano de subscrição anual para os membros, com um custo entre 150€ e 350€ por hectare, cobre todas as inspeções previstas, oferecendo um serviço completo e sustentável.

Venda de Serviços a Terceiros: Para aumentar a sustentabilidade, a cooperativa pode oferecer o serviço de monitorização a agricultores não-membros ou terceiros, gerando uma receita adicional. O custo de inspeção para estes clientes pode variar entre 50€ e 200€ por hectare por inspeção. A receita adicional gerada permite amortizar o custo inicial do equipamento e reduzir o custo para os membros da cooperativa.

Subscrições e Planos Personalizados: A cooperativa pode também oferecer pacotes de subscrição ajustados às necessidades de cada agricultor, permitindo uma previsão de receitas que facilita a gestão dos custos operacionais. Com uma taxa anual entre 150€ e 350€ por hectare, os agricultores beneficiam de um serviço de monitorização regular, o que contribui para uma maior previsibilidade e segurança na deteção de infestações.

4.7.4 Retorno do Investimento e Sustentabilidade

Com uma cobertura de 200 hectares e um modelo de subscrição para membros da cooperativa, o sistema apresenta um retorno económico positivo:

Receita Anual da Cooperativa: Com uma taxa de subscrição entre 150€ e 350€ por hectare, a cooperativa pode gerar uma receita anual entre 30,000€ e 70,000€, cobrindo os custos de operação e manutenção do drone e gerando uma margem sustentável para a cooperativa.

Recuperação do Investimento: O investimento inicial necessário para a aquisição do drone e instalação do sistema de visão computacional pode ser recuperado em 1 a 2 anos,

dependendo do custo de subscrição e do número de hectares cobertos. Este modelo garante um retorno significativo a médio prazo, assegurando a viabilidade económica do sistema.

Economias Indiretas e Aumento de Produtividade: A deteção precoce de infestações é uma mais-valia, pois permite agir rapidamente para conter a praga e evitar perdas de produtividade que podem chegar a 80% em áreas não monitorizadas. Estas economias indiretas, aliadas à sustentabilidade do sistema, reforçam o valor económico do modelo a longo prazo, ao garantir um aumento de produtividade e um menor custo com controle de pragas.

4.7.5 Considerações Finais sobre o Potencial Económico

A utilização de drones para monitorização de pragas agrícolas está cada vez mais vigente e tem um grande potencial económico, particularmente em sistemas cooperativos onde os custos são partilhados entre múltiplos utilizadores. A estrutura da cooperativa permite que agricultores de pequena e média dimensão beneficiem de tecnologia avançada, que de outra forma não seria possível, e a um custo acessível, ao mesmo tempo que gera uma receita constante para cobrir o investimento inicial e as despesas operacionais.

Ao possibilitar a deteção precoce e eficaz das infestações, este sistema contribui para a sustentabilidade da produção agrícola e reduz custos a longo prazo. A combinação de economia de escala, monitorização contínua e um modelo de subscrição viável faz com que a utilização de drones e visão computacional não seja apenas uma solução tecnológica avançada, mas também uma estratégia economicamente sustentável para a gestão de plantações agrícolas.

5 Conclusão

5.1 Resumo dos principais achados

Os principais resultados deste estudo evidenciam a eficiência dos modelos YOLOv8.2 e Roboflow na deteção de doenças em árvores, em particular a vespa-das-galhas-do-castanheiro. Os modelos demonstram uma boa precisão na identificação das galhas, mesmo em condições de variabilidade no ambiente, como iluminação e ângulos diferentes e com um dataset relativamente reduzido. A utilização de técnicas de data *augmentation*, como rotação, redimensionamento, inversão (*flip*) foi fundamental para melhorar a generalização do modelo e compensar o número limitado de imagens disponíveis no dataset. Este procedimento permitiu que o sistema se tornasse mais robusto em simulações de cenários reais de campo (imagens de vídeo de teste).

Para o desenvolvimento de uma aplicação profissional, capaz de atingir o máximo de produtividade, é essencial dispor de recursos computacionais adequados. Durante este estudo, verificou-se que o uso de GPU's poderosas, ou a subscrição da versão Pro do Google Colab, são necessários para realizar múltiplos treinos consecutivos sem interrupções, pois a versão gratuita apresenta limitações significativas, como a falta de créditos e tempo de execução reduzido, com um máximo de cerca de 3horas e 40 minutos de treino consecutivo, e quando o tempo se esgota, por vezes é necessário esperar 48h para poder voltar a utilizar o poder computacional de forma gratuita. Uma situação semelhante também se verifica com o Roboflow, em que os créditos são limitados. Embora a versão gratuita do Roboflow ofereça uma gama útil de funcionalidades, a versão *premium* dispõe de recursos adicionais que tornam o processo de anotação, treino e *deployement* do modelo muito mais eficiente. Também se torna mais prazeroso realizar essas atividades, uma vez que a produtividade é muito elevada e os resultados surgem mais rápido. Funcionalidades como o aumento de limites de armazenamento, ferramentas avançadas de anotação e integração facilitada com ambientes de produção são essenciais para quem procura desenvolver soluções escaláveis e profissionais.

Ao criar o modelo, deve ser analisada a sua aplicação prática, e quando analisado o comportamento da doença fisicamente, e as suas particularidades, podemos observar que uma árvore normalmente, não tem apenas uma infeção por galha ou árvore, normalmente são encontrados vários focos de infeção com dezenas de ovos. Posto isso, mesmo que o modelo não preveja todos os pontos de infeção, baseado nos dados adquiridos e testes efetuados, com

certeza o modelo vai prever grande parte das galhas infetadas, o que mesmo não detetando todas as galhas infetadas, cumpriria com o propósito funcional e prático do modelo.

Em resumo, o estudo mostra que, embora o YOLOv8.2 e o Roboflow 3.0 tenham demonstrado uma performance robusta na deteção de doenças, alcançar o pleno potencial de uma aplicação prática e escalável requer investimentos em infraestrutura computacional e ferramentas de *software* profissionais, como as versões *premium* de Google Colab e Roboflow. Estes recursos permitem que o processo de desenvolvimento seja mais rápido, eficiente e, sobretudo, sustentável em ambientes de produção.

5.2 Contribuições do estudo para a gestão de pragas

Embora o sistema desenvolvido ainda necessite de melhorias, mesmo assim, já demonstra um potencial significativo no suporte para a gestão de pragas, especialmente em áreas com escassez de mão de obra e elevada densidade de castanheiros. A sua capacidade de automatizar verificações em grandes superfícies reduz a necessidade de intervenção humana constante, o que é particularmente vantajoso em regiões de difícil acesso ou com recursos limitados. Ao proporcionar uma monitorização eficiente e contínua, este tipo de sistema pode otimizar o tempo e os custos operacionais, permitindo assim, uma gestão mais eficaz das pragas, mesmo em cenários onde a mão de obra é limitada, garantindo uma vigilância mais frequente e precisa.

5.3 Recomendações para futuras investigações

Com base nos resultados obtidos e nas limitações identificadas ao longo deste trabalho, recomenda-se o desenvolvimento de futuras investigações que explorem diferentes abordagens e tecnologias para aprimorar a deteção de pragas, como a vespa-das-galhas-do-castanheiro. Entre as sugestões para trabalhos futuros, destacam-se:

Utilização de Câmaras Multiespectrais: As câmaras multiespectrais são capazes de capturar informações em vários comprimentos de onda, permitindo uma análise mais detalhada da saúde das plantas. A aplicação desta tecnologia poderia melhorar a deteção precoce de infestações, uma vez que permite a identificação de anomalias invisíveis ao olho humano e até mesmo a câmaras RGB. Futuros estudos podem investigar a eficácia das câmaras multiespectrais na deteção de galhas em comparação com os métodos tradicionais.

Drones com Capacidade de Zoom Ótico: Investigar o uso de do modelo treinado em drones equipados com câmaras de alta resolução e zoom ótico seria uma valiosa adição para capturar imagens mais pormenorizadas das galhas em áreas de difícil acesso. Esta funcionalidade permitiria a inspeção detalhada de cada árvore sem a necessidade de uma proximidade excessiva, aumentando assim a eficiência e a precisão na deteção.

Integração de Câmaras em Tratores: Outra área promissora para investigações futuras seria a implementação de câmaras em tratores ou outros veículos agrícolas durante as atividades regulares, como a lavra. Este sistema poderia fornecer uma monitorização constante e em tempo real, agindo como uma medida preventiva enquanto se realizam tarefas diárias nas plantações. Esta abordagem poderia reduzir custos ao integrar a tecnologia no fluxo normal de trabalho.

Comparação de Diferentes Modelos de Visão Computacional: Futuros estudos deveriam considerar a utilização de múltiplos modelos de IA, como diferentes versões do YOLO (e.g., YOLOv10, YOLOv11), SSD ou Faster R-CNN, para comparar o desempenho na deteção de galhas infetadas. A comparação entre modelos permitirá identificar aquele que oferece o melhor compromisso entre precisão, velocidade e capacidade de generalização, otimizando assim o processo de deteção.

Criação de Diferentes Classes para Estágios e Manifestações da Doença e Integração de Outras Doenças: Para otimizar a deteção de pragas e doenças que afetam o castanheiro, recomenda-se não apenas a criação de classes específicas para os variados estágios e manifestações da vespa-das-galhas-do-castanheiro, mas também a integração de outras doenças que afetam esta espécie. Um modelo capaz de detetar múltiplas doenças aumentaria a versatilidade e eficácia do sistema, permitindo uma gestão mais abrangente e eficiente. Com essa abordagem, cada doença seria representada por classes específicas, otimizando a identificação e o tratamento direcionado. As classes sugeridas incluem:

Classes para Vespa-das-galhas-do-castanheiro:

Galha_Inicial: Caracterizadas por pequenas galhas de coloração esverdeada, ainda sem impacto nas folhas.

Galha_Intermediária: Normalmente são galhas de tamanho médio, coloração entre o amarelo e o alaranjado, com possíveis alterações nas folhas próximas.

Galha_Avançada: Estas galhas são maiores com coloração castanha ou vermelha, associadas a necrose das folhas adjacentes (folhas secas).

Folha_Manchada: Folhas que apresentam manchas ainda num estágio inicial de maturação, o que não é normal, sinal precoce de infeção.

Folha_Necrosada: Folhas em estado avançado de necrose, evidenciando uma infeção mais severa, são relativamente fáceis de detetar, pois aparecem numa altura que as restantes folhas ainda têm uma coloração verde e aspeto saudável.

Classes para Outras Doenças:

Cancro-do-Castanheiro_Inicial: Pequenas lesões no tronco e nos ramos, que podem evoluir para cancro, causado pelo fungo *Cryphonectria parasitica*.

Cancro-do-Castanheiro_Avançado: Lesões mais extensas e escuras no tronco e ramos, com sinais de descamação e necrose avançada.

Tinta-do-Castanheiro_Inicial: Folhas com coloração amarela e sinais de murchamento, característicos da infeção causada por *Phytophthora cinnamomi*.

Tinta-do-Castanheiro_Avançada: Raízes apodrecidas e descoloração geral da planta, especialmente nas folhas e tronco, indicando infeção severa.

Estas recomendações têm o potencial de não só melhorar a precisão e eficiência do sistema de deteção de pragas, mas também de ampliar a aplicabilidade das tecnologias propostas a diferentes contextos agrícolas e geográficos. A realização de investigações que envolvam estas técnicas com certeza trará avanços significativos na gestão sustentável de pragas, promovendo a saúde das culturas e a viabilidade económica dos produtores e da região de Trás-os-Montes.

6 Referências

6.1 Lista de todas as fontes bibliográficas consultadas e citadas no trabalho

[1] K. Ipekdal, F. Colombari, and A. Battisti, “From Padova to Yalova: biocontrol of the Asian chestnut gall wasp in Italy and Turkey,” 2022, pp. 191–214.

[2] “Vespa da Galha do Castanheiro - Uma praga recente que está instalada na região | CM Sabrosa.” Accessed: Oct. 17, 2024. [Online]. Available: https://www.sabrosa.pt/pages/442?news_id=139

[3] E. Marcolin *et al.*, “Impact of the Asian gall wasp Dryocosmus kuriphilus on the radial growth of the European chestnut Castanea sativa,” *Journal of Applied Ecology*, vol. 58, no. 6, pp. 1212–1224, Jun. 2021, doi: 10.1111/1365-2664.13861.

[4] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” *Cornell University*, Apr. 2018, Accessed: Oct. 18, 2024. [Online]. Available: https://arxiv.org/abs/1804.02767v1

[5] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 2020, Accessed: Oct. 17, 2024. [Online]. Available: https://github.com/AlexeyAB/darknet.

[6] F. Iost Filho, W. Heldens, Z. Kong, and E. de Lange, “Drones: Innovative Technology for Use in Precision Pest Management,” *J Econ Entomol*, vol. 113, Oct. 2019, doi: 10.1093/jee/toz268.

[7] J. Kim, S. Kim, C. Ju, and H. Son, “Unmanned Aerial Vehicles in Agriculture: A Review of Perspective of Platform, Control, and Applications,” *IEEE Access*, vol. PP, p. 1, Oct. 2019, doi: 10.1109/ACCESS.2019.2932119.

[8] K. Zhichkin, V. Nosov, L. Zhichkina, O. Anichkina, I. Borodina, and A. Beketov, “Efficiency of using drones in agricultural production,” *E3S Web of Conferences*, vol. 381, Oct. 2023, doi: 10.1051/e3sconf/202338101048.

[9] Y. Li, H. Wang, L. M. Dang, A. Sadeghi, and H. Moon, “Crop pest recognition in natural scenes using convolutional neural networks,” *Comput Electron Agric*, vol. 169, Oct. 2020, doi: 10.1016/j.compag.2019.105174.

[10] N. Banerjee, A. Dambale, and H. Upadhyay, “SMART FARMING TECHNOLOGIES FOR SUSTAINABLE AGRICULTURE,” vol. 4, pp. 13–19, Oct. 2024.

[11] “Manejo Integrado de Pragas (MIP): Métodos e Vantagens.” Accessed: Oct. 18, 2024. [Online]. Available: https://eos.com/pt/blog/manejo-integrado-de-pragas/

[12] “MIP (Manejo Integrado de Pragas): tudo o que você precisa saber sobre ele - Blog da Aegro.” Accessed: Oct. 18, 2024. [Online]. Available: https://blog.aegro.com.br/manejo-integrado-de-pragas/

[13] F. Técnica, “Alternativas de Escoamento dos Subprodutos do Algodão e Culturas Acessórias na África Projeto Além do Algodão (Projeto-País: Tanzânia) AMOSTRAGEM E MONITORAMENTO DE PRAGAS E DOENÇAS”, Accessed: Oct. 18, 2024. [Online]. Available: https://centrodeexcelencia.org.br/wp-content/uploads/2024/03/amostragem-e-monitoramento-de-pragas_1903.pdf

[14] T. Chen and H. Yin, “Camera-based plant growth monitoring for automated plant cultivation with controlled environment agriculture,” *Smart Agricultural Technology*, vol. 8, p. 100449, Aug. 2024, doi: 10.1016/J.ATECH.2024.100449.

[15] T. Bakker, K. Asselt, J. Bontsema, J. Müller, and G. van Straten, “Autonomous navigation using a robot platform in a sugar beet field,” *Biosyst Eng*, vol. 109, pp. 357–368, Oct. 2011, doi: 10.1016/j.biosystemseng.2011.05.001.

[16] Gaurav Roy, “Revolucionando a agricultura: o papel da robótica no aumento da produtividade e da sustentabilidade - Securities.io.” Accessed: May 30, 2024. [Online]. Available: https://www.securities.io/pt/revolucionando-a-agricultura-o-papel-da-rob%C3%B3tica-no-aumento-da-produtividade-e-da-sustentabilidade/

[17] C. Zhou *et al.*, “A smartphone application for site-specific pest management based on deep learning and spatial interpolation,” *Comput Electron Agric*, vol. 218, p. 108726, Mar. 2024, doi: 10.1016/j.compag.2024.108726.

[18] A. Gaikwad and V. Kharbadkar, “Hyperspectral Imaging: A Tool for Plant Disease Detection,” Oct. 2024. doi: 10.13140/RG.2.2.23479.09129.

[19] “Câmeras com sensores RGB e multispectral: entenda como são!” Accessed: May 30, 2024. [Online]. Available: https://www.modelismobh.com.br/blog/camera-com-sensores-rgb-e-multispectral-conheca-as-diferencas-e-as-cameras-mais-usadas-em-drones/

[20] Solanki Preeti, Gupta Vaibhav, and Kuntal Preeti, “Satellite Imagery For Crop Monitoring,” *J Cardiovasc Dis Res*, vol. 12, no. 1, pp. 937–949, 2021.

[21] H. Xu, S. Zhu, Y. Ying, and H. Jiang, “Early detection of plant disease using infrared thermal imaging,” *Proc SPIE*, vol. 6381, p. 35, Oct. 2006, doi: 10.1117/12.685534.

[22] J. C. de Alarcão Júnior and D. N. C. Nuñez, “O uso de drones na agricultura 4.0,” *Brazilian Journal of Science*, vol. 3, no. 1, pp. 1–13, Jul. 2023, doi: 10.14295/bjs.v3i1.438.

[23] “Tecnologia na agricultura: O que são as imagens térmicas?” Accessed: May 30, 2024. [Online]. Available: https://geoagri.com.br/public/blog/23/tecnologia-na-agricultura-o-que-sao-as-imagens-termicas

[24] L. Deng, Z. Mao, X. Li, Z. Hu, F. Duan, and Y. Yan, “UAV-based multispectral remote sensing for precision agriculture: A comparison between different cameras,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 146, pp. 124–136, Dec. 2018, doi: 10.1016/j.isprsjprs.2018.09.008.

[25] “Drones DJI Agras.” Accessed: Jun. 23, 2024. [Online]. Available: https://redproventum.net/productos/drones-dji-para-la-agricultura/

[26] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[27] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955,” *AIMag*, vol. 27, no. 4, p. 12, Dec. 2006.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[29] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science (1979)*, vol. 349, no. 6245, pp. 255–260, 2015.

[30] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[31] M. Bojarski *et al.*, “End to End Learning for Self-Driving Cars,” Apr. 2016.

[32] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, “A Bayesian Approach to Filtering Junk E-Mail,” *AAAI’98 Workshop on Learning for Text Categorization*, vol. 62, Oct. 1998.

[33] B. Pang and L. Lee, “Opinion Mining and Sentiment Analysis,” *Foundations and Trends® in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008, doi: 10.1561/1500000011.

[34] A. Esteva *et al.*, “Dermatologist-level classification of skin cancer with deep neural networks,” *Nature*, vol. 542, no. 7639, pp. 115–118, Feb. 2017, doi: 10.1038/nature21056.

[35] Y. Zhang, R. Jin, and Z.-H. Zhou, “Understanding bag-of-words model: A statistical framework,” *International Journal of Machine Learning and Cybernetics*, vol. 1, pp. 43–52, Oct. 2010, doi: 10.1007/s13042-010-0001-0.

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” Oct. 2018. doi: 10.48550/arXiv.1810.04805.

[37] C. Bishop, “Pattern Recognition and Machine Learning,” in *Journal of Electronic Imaging*, vol. 16, 2006, pp. 140–155. doi: 10.1117/1.2819119.

[38] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. 2009.

[39] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2021.

[40] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[41] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Wiley-Interscience, 2000.

[42] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.

[43] V. Mnih *et al.*, “Playing Atari with deep reinforcement learning,” *arXiv Preprint*, 2013, [Online]. Available: https://arxiv.org/abs/1312.5602

[44] A. Kamilaris and F. Prenafeta Boldú, “Deep Learning in Agriculture: A Survey,” *Comput Electron Agric*, vol. 147, Oct. 2018, doi: 10.1016/j.compag.2018.02.016.

[45] K. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, “Machine Learning in Agriculture: A Review,” *Sensors*, vol. 18, no. 8, p. 2674, Aug. 2018, doi: 10.3390/s18082674.

[46] J. D. Kelleher, B. Mac Namee, and A. D’Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*. MIT Press, 2015.

[47] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, doi: 10.1016/j.neunet.2014.09.003.

[48] Z. C. Lipton, “The Mythos of Model Interpretability,” *Queue*, vol. 16, no. 3, pp. 31–57, Jun. 2018, doi: 10.1145/3236386.3241340.

[49] V. Dumoulin, F. Visin, and G. E. P. Box, “Object detection _ssd :A guide to convolution arithmetic for deep learning,” *arXiv:1603.07285 [cs, stat]*, pp. 1–31, 2018.

[50] D. Scherer, A. Müller, and S. Behnke, “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition,” 2010, pp. 92–101. doi: 10.1007/978-3-642-15825-4_10.

[51] J. Han and C. Moraga, “The influence of the sigmoid function parameters on the speed of backpropagation learning,” 1995, pp. 195–201. doi: 10.1007/3-540-59497-3_175.

[52] V. Nair and G. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair,” in *Proceedings of ICML*, Oct. 2010, pp. 807–814.

[53] A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Neural Information Processing Systems*, vol. 25, Oct. 2012, doi: 10.1145/3065386.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.

[55] S. Ruder, “An overview of gradient descent optimization algorithms,” Sep. 2016.

[56] L. Pádua, P. Marques, L. Martins, A. Sousa, E. Peres, and J. J. Sousa, “Monitoring of Chestnut Trees Using Machine Learning Techniques Applied to UAV-Based Multispectral Data,” *Remote Sens (Basel)*, vol. 12, no. 18, p. 3032, Sep. 2020, doi: 10.3390/rs12183032.

[57] D. Anderson and C. S. Chan, “A Comprehensive Survey of Deep Learning in Remote Sensing: Theories, Tools and Challenges for the Community,” *J Appl Remote Sens*, Oct. 2017.

[58] C. Shorten and T. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J Big Data*, vol. 6, Oct. 2019, doi: 10.1186/s40537-019-0197-0.

[59] M. Hafizur and M. H. R. Masum, “Visualizing and Understanding Convolutional Networks,” Oct. 2022. doi: 10.13140/RG.2.2.12182.22080.

[60] R. Szeliski, *Computer Vision: Algorithms and Applications*. London: Springer-Verlag, 2011.

[61] E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Apress, 2019.

[62] A. Lavin, “Fastai: A layered API for deep learning,” *arXiv Preprint*, 2018, [Online]. Available: https://arxiv.org/abs/1807.01205

[63] G. Bradski and A. Kaehler, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*, 2nd ed. Sebastopol: O’Reilly Media, 2016.

[64] M. Abadi *et al.*, “TensorFlow: A system for large-scale machine learning,” Oct. 2016, doi: 10.48550/arXiv.1605.08695.

[65] F. Chollet, *Deep Learning with Python*, 2nd ed. Shelter Island: Manning Publications, 2021.

[66] A. Paszke, S. Gross, S. Chintala, G. Chanan, T. Killeen, and others, “PyTorch: An imperative style, high-performance deep learning library,” *Adv Neural Inf Process Syst*, vol. 32, pp. 8024–8035, 2019.

[67] G. Jocher, A. Chaurasia, J. Qiu, and N. Singh, “YOLO by Ultralytics, version YOLOv8.” Accessed: Oct. 19, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[68] J. Howard and S. Gugger, *Deep Learning for Coders with fastai and PyTorch: AI Applications Without a PhD*. Sebastopol: O’Reilly Media, 2020. [Online]. Available: https://www.oreilly.com/library/view/deep-learning-for/9781492045526/

[69] MathWorks, “Image and Video Processing Solutions,” 2024. Accessed: Feb. 19, 2024. [Online]. Available: https://www.mathworks.com/solutions/image-video-processing.html

[70] Mvt. S. GmbH, “HALCON – The Power of Machine Vision,” 2021. Accessed: Dec. 10, 2023. [Online]. Available: https://www.mvtec.com/products/halcon/

[71] C. Corporation, “Automation Equipment Providers and Integrators,” 2024. Accessed: Feb. 19, 2024. [Online]. Available: https://www.cognex.com/industries/automation-equipment-providers-integrators

[72] Inc. Amazon Web Services, “Amazon Rekognition,” 2023. Accessed: Feb. 19, 2024. [Online]. Available: https://aws.amazon.com/rekognition/

[73] G. Cloud, “Cloud Vision API,” 2023. Accessed: Feb. 20, 2023. [Online]. Available: https://cloud.google.com/vision

[74] Microsoft, “Azure Cognitive Services: Computer Vision,” 2023. Accessed: Feb. 20, 2024. [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/

[75] Ultralytics, “What is YOLOv8?,” 2023. Accessed: May 19, 2024. [Online]. Available: https://yolov8.org/what-is-yolov8/

[76] R. Rustemgaliev, “YOLOv8, EfficientDet, Faster R-CNN, or YOLOv5 for Remote Sensing,” 2024. Accessed: Aug. 19, 2023. [Online]. Available: https://medium.com/@rustemgal/yolov8-efficientdet-faster-r-cnn-or-yolov5-for-remote-sensing-12487c40ef68

[77] E. Gehring, B. Bellosi, A. Quacchia, and M. Conedera, “Assessing the impact of Dryocosmus kuriphilus on the chestnut tree: branch architecture matters,” *J Pest Sci (2004)*, vol. 91, Oct. 2018, doi: 10.1007/s10340-017-0857-9.

[78] “Dryocosmus kuriphilus (Oriental chestnut gall wasp),” CABI Compendium. Accessed: Oct. 28, 2023. [Online]. Available: https://www.cabidigitallibrary.org/doi/full/10.1079/cabicompendium.20005

[79] “EPPO Global Database.” Accessed: Dec. 20, 2023. [Online]. Available: https://gd.eppo.int/reporting/article-2823

[80] F. Paparella, C. Ferracini, A. Portaluri, A. Manzo, and A. Alma, “Biological control of the chestnut gall wasp with T. sinensis: A mathematical model,” *Ecol Modell*, vol. 338, pp. 17–36, Oct. 2016, doi: 10.1016/J.ECOLMODEL.2016.07.023.

[81] E. M. R. de Sousa, “Programa Nacional de Monitorização: Vespa do castanheiro (Dryocosmus kuriphilus),” 2019. [Online]. Available: https://www.icnf.pt/api/file/doc/f78ff2c665838a55

[82] “ICNF - Instituto da Conservação da Natureza e das Florestas.” Accessed: Nov. 20, 2023. [Online]. Available: https://www.icnf.pt/florestas/fitossanidade/agentesbioticosnocivos/vespadasgalhasdocastanheiro

[83] “Chestnut Market Size, Share, Industry Report 2024-2032.” Accessed: Oct. 18, 2024. [Online]. Available: https://www.imarcgroup.com/chestnut-market

[84] FAO, “Using one wasp to fight another: FAO establishes biological control plan to protect Albanian chestnuts.” Accessed: Mar. 20, 2024. [Online]. Available: https://www.fao.org/europe/news/detail/using-one-wasp-to-fight-another--fao-establishes-biological-control-plan-to-protect-albanian-chestnuts/en

[85] A. Amorim, R. Rodrigues, L. J. R. Nunes, M. Freitas, and L. Moura, “Dryocosmus kuriphilus Yasumatsu (Hymenoptera: Cynipidae) in Minho (Northern Portugal): Bioecology, Native Parasitoid Communities and Biological Control with Torymus sinensis Kamijo (Hymenoptera: Torymidae),” *Agronomy 2022, Vol. 12, Page 2184*, vol. 12, no. 9, p. 2184, Sep. 2022, doi: 10.3390/AGRONOMY12092184.

[86] “Estudo Económico de Desenvolvimento da Fileira da Castanha”, Accessed: Feb. 20, 2024. [Online]. Available: https://www.compete2020.gov.pt/admin/images/Castanha_Estudo.pdf

[87] Solawetz Jacob, “What is YOLOv8? A Complete Guide.” Accessed: Oct. 01, 2024. [Online]. Available: https://blog.roboflow.com/what-is-yolov8/

[88] Gallagher James, “Launch: Evaluate Computer Vision Models on Roboflow.”

[89] “train-yolov8-object-detection-on-custom-dataset.ipynb - Colab.” Accessed: May 25, 2024. [Online]. Available: https://colab.research.google.com/github/roboflow-ai/notebooks/blob/main/notebooks/train-yolov8-object-detection-on-custom-dataset.ipynb

[90] “Fotos de Vespa das galhas do castanheiro (Dryocosmus kuriphilus) · BioDiversity4All.” Accessed: Oct. 21, 2024. [Online]. Available: https://www.biodiversity4all.org/taxa/357839-Dryocosmus-kuriphilus/browse_photos?order_by=created_at

[91] “Roboflow - Ultralytics YOLO Docs.” Accessed: May 20, 2024. [Online]. Available: https://docs.ultralytics.com/pt/integrations/roboflow/

[92] Jain Abhishek, “Data Augmentation. Data augmentation is a technique used… | by Abhishek Jain | Medium.” Accessed: Apr. 21, 2024. [Online]. Available: https://medium.com/@abhishekjainindore24/data-augmentation-00c72f5f4c54

[93] “train-yolov8-object-detection-on-custom-dataset.ipynb - Colab.” Accessed: Aug. 30, 2024. [Online]. Available: https://colab.research.google.com/github/roboflow/notebooks/blob/main/notebooks/train-yolov8-object-detection-on-custom-dataset.ipynb

[94] “Best Duplicate Photo Finder, Visual Similarity Duplicate Image and Exact Duplicate Picture Finder.” Accessed: Sep. 30, 2024. [Online]. Available: https://www.pictureecho.com/?gad_source=1&gclid=Cj0KCQjwj4K5BhDYARIsAD1Ly2q_cwYQiKyAgGeydVcTYGKJJXfnRGjVM1wH8Z196eXSyf7s0fk01BgaAhlMEALw_wcB

[95] “Predict - Ultralytics YOLO Docs.” Accessed: Sep. 21, 2024. [Online]. Available: https://docs.ultralytics.com/modes/predict/

7 Anexos

7.1 Dados brutos, gráficos detalhados, e códigos de software

7.1.1 Código de *setup* Google Colab treino e resultados para 100 *epochs* sem *data augmentation*:

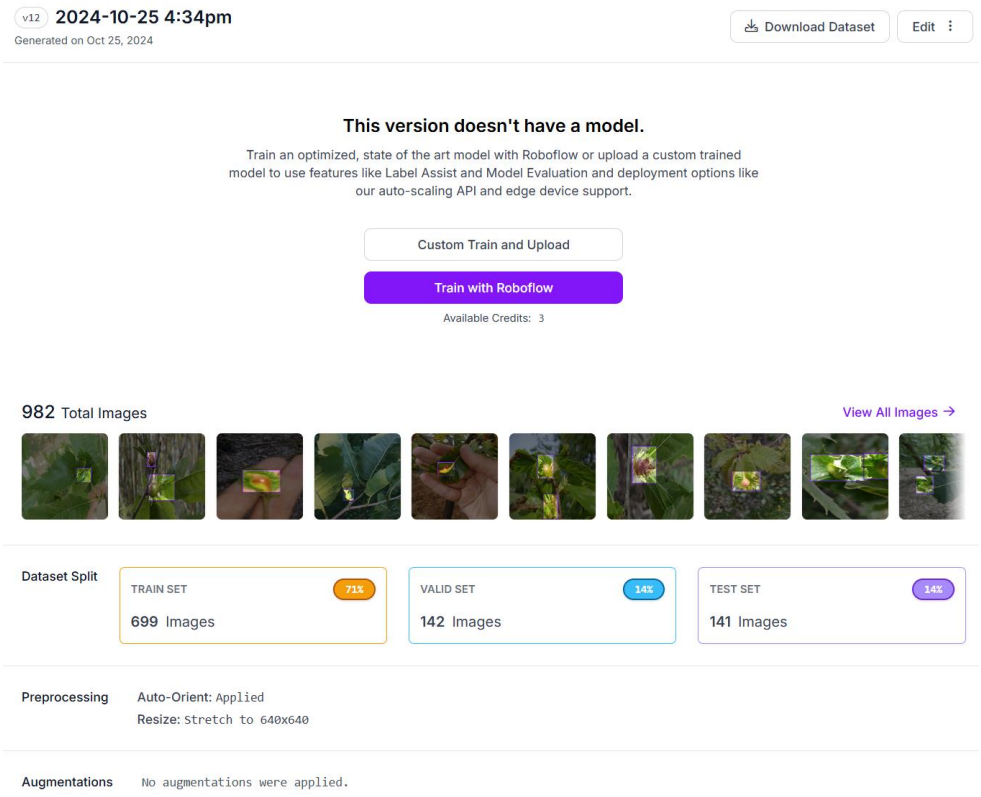


Figura 7.1 - Dataset usado para treino

```
!nvidia-smi
import os
HOME = os.getcwd()
print(HOME)

# Pip install method (recommended)

!pip install ultralytics==8.2.103 -q

from IPython import display
display.clear_output()
```

```
import ultralytics
ultralytics.checks()
```

Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)

Setup complete ☑ (2 CPUs, 12.7 GB RAM, 32.0/112.6 GB disk)

```
from ultralytics import YOLO

from IPython.display import display, Image

!mkdir -p {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow --quiet

import roboflow

!pip install roboflow
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="My API Key, removed for privacy")
project = rf.workspace("vespa-mjis3").project("vespa-detection")
version = project.version(12)
dataset = version.download("yolov8")

Google Drive Connect
from google.colab import drive
drive.mount('/content/drive')

%cd /content/drive/MyDrive/YOLOv8

!yolo task=detect mode=train model=yolov8s.pt
data={dataset.location}/data.yaml epochs=100 imgsz=640 plots=True

New https://pypi.org/project/ultralytics/8.3.23 available 😃 Update
with 'pip install -U ultralytics'
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0
(Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8s.pt,
data=/content/{HOME}/datasets/datasets/Vespa-Detection-12/data.yaml,
epochs=100, time=None, patience=100, batch=16, imgsz=640, save=True,
save_period=-1, cache=False, device=None, workers=8, project=None,
name=train2, exist_ok=False, pretrained=True, optimizer=auto, verbose=True,
seed=0, deterministic=True, single_cls=False, rect=False, cos_lr=False,
close_mosaic=10, resume=False, amp=True, fraction=1.0, profile=False,
freeze=None, multi_scale=False, overlap_mask=True, mask_ratio=4,
dropout=0.0, val=True, split=val, save_json=False, save_hybrid=False,
conf=None, iou=0.7, max_det=300, half=False, dnn=False, plots=True,
source=None, vid_stride=1, stream_buffer=False, visualize=False,
augment=False, agnostic_nms=False, classes=None, retina_masks=False,
embed=None, show=False, save_frames=False, save_txt=False, save_conf=False,
```

```
save_crop=False, show_labels=True, show_conf=True, show_boxes=True,
line_width=None, format=torchscript, keras=False, optimize=False,
int8=False, dynamic=False, simplify=True, opset=None, workspace=4,
nms=False, lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005,
warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=7.5,
cls=0.5, dfl=1.5, pose=12.0, kobj=1.0, label_smoothing=0.0, nbs=64,
hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5,
shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, bgr=0.0, mosaic=1.0,
mixup=0.0, copy_paste=0.0, auto_augment=randaugment, erasing=0.4,
crop_fraction=1.0, cfg=None, tracker=botsort.yaml,
save_dir=runs/detect/train2
      Downloading https://ultralytics.com/assets/Arial.ttf to
'/root/.config/Ultralytics/Arial.ttf'...
      100% 755k/755k [00:00<00:00, 23.0MB/s]
      Overriding model.yaml nc=80 with nc=2

                         from  n    params  module
arguments
        0                  -1  1       928
ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
        1                  -1  1     18560
ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
        2                  -1  1     29056
ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
        3                  -1  1     73984
ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
        4                  -1  2    197632
ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
        5                  -1  1    295424
ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
        6                  -1  2    788480
ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
        7                  -1  1   1180672
ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
        8                  -1  1   1838080
ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
        9                  -1  1    656896
ultralytics.nn.modules.block.SPPF            [512, 512, 5]
       10                  -1  1         0
torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
       11             [-1, 6]  1         0
ultralytics.nn.modules.conv.Concat           [1]
       12                  -1  1    591360
ultralytics.nn.modules.block.C2f             [768, 256, 1]
       13                  -1  1         0
torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
       14             [-1, 4]  1         0
ultralytics.nn.modules.conv.Concat           [1]
       15                  -1  1    148224
ultralytics.nn.modules.block.C2f             [384, 128, 1]
       16                  -1  1    147712
ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
       17            [-1, 12]  1         0
ultralytics.nn.modules.conv.Concat           [1]
       18                  -1  1    493056
ultralytics.nn.modules.block.C2f             [384, 256, 1]
       19                  -1  1    590336
ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
       20             [-1, 9]  1         0
ultralytics.nn.modules.conv.Concat           [1]
```

```
       21                  -1  1   1969152
ultralytics.nn.modules.block.C2f             [768, 512, 1]
       22        [15, 18, 21]  1   2116822
ultralytics.nn.modules.head.Detect           [2, [128, 256, 512]]
      Model summary: 225 layers, 11,136,374 parameters, 11,136,358
gradients, 28.6 GFLOPs

      Transferred 349/355 items from pretrained weights
      TensorBoard: Start with 'tensorboard --logdir runs/detect/train2',
view at http://localhost:6006/
      Freezing layer 'model.22.dfl.conv.weight'
      AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
      AMP: checks passed ☑
      train: Scanning /content/{HOME}/datasets/datasets/Vespa-Detection-
12/train/labels... 699 images, 100 backgrounds, 0 corrupt: 100% 699/699
[00:00<00:00, 1757.09it/s]
      train: New cache created: /content/{HOME}/datasets/datasets/Vespa-
Detection-12/train/labels.cache
      /usr/local/lib/python3.10/dist-
packages/albumentations/__init__.py:13: UserWarning: A new version of
Albumentations is available: 1.4.20 (you have 1.4.15). Upgrade using: pip
install -U albumentations. To disable automatic update checks, set the
environment variable NO_ALBUMENTATIONS_UPDATE to 1.
        check_for_updates()
      albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01,
blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3,
method='weighted_average'), CLAHE(p=0.01, clip_limit=(1, 4.0),
tile_grid_size=(8, 8))
      val: Scanning /content/{HOME}/datasets/datasets/Vespa-Detection-
12/valid/labels... 142 images, 35 backgrounds, 0 corrupt: 100% 142/142
[00:00<00:00, 1359.40it/s]
      val: New cache created: /content/{HOME}/datasets/datasets/Vespa-
Detection-12/valid/labels.cache
      Plotting labels to runs/detect/train2/labels.jpg...
      optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and
'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momentum'
automatically...
      optimizer: AdamW(lr=0.001667, momentum=0.9) with parameter groups 57
weight(decay=0.0), 64 weight(decay=0.0005), 63 bias(decay=0.0)
```

TensorBoard: model graph visualization added ☑

Image sizes 640 train, 640 val

Using 2 dataloader workers

Logging results to **runs/detect/train2**

Starting training for 100 epochs...

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
1/100      4.08G      2.338      3.584      2.204         28        640: 100% 44/44 [00:19<00:00,  2.28it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:03<00:00,  1.61it/s]
             all        142        132      0.197      0.333       0.16     0.0469

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
2/100      3.88G      2.191      2.527      2.168         22        640: 100% 44/44 [00:17<00:00,  2.49it/s]
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  2.97it/s]
        all        142        132       0.33      0.273      0.205     0.0508

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
3/100      4.06G      2.302      2.661      2.276         42        640: 100% 44/44 [00:17<00:00,  2.48it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.44it/s]
        all        142        132       0.29      0.212       0.17      0.042

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
4/100      3.86G       2.34      2.567      2.184         40        640: 100% 44/44 [00:17<00:00,  2.53it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.13it/s]
        all        142        132      0.126      0.114     0.0563     0.0171

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
5/100      3.86G      2.289      2.526      2.211         19        640: 100% 44/44 [00:16<00:00,  2.65it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.16it/s]
        all        142        132      0.391       0.22      0.186     0.0492

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
6/100      4.02G       2.26       2.42      2.213         51        640: 100% 44/44 [00:16<00:00,  2.63it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.24it/s]
        all        142        132      0.344      0.258      0.215     0.0534

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
7/100      4.02G      2.251      2.447      2.184         45        640: 100% 44/44 [00:17<00:00,  2.45it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.20it/s]
        all        142        132      0.317      0.326      0.224     0.0568

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
8/100      4.02G       2.21      2.366       2.13         30        640: 100% 44/44 [00:17<00:00,  2.48it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.13it/s]
        all        142        132      0.366      0.258      0.237     0.0748

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
9/100      4.02G      2.165      2.323      2.079         19        640: 100% 44/44 [00:17<00:00,  2.49it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  2.93it/s]
        all        142        132      0.346      0.417      0.312     0.0793

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
10/100     4.02G      2.142      2.257       2.09         28        640: 100% 44/44 [00:17<00:00,  2.47it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  2.98it/s]
        all        142        132      0.394      0.485      0.408     0.114

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
11/100     4.02G      2.135      2.222      2.128         39        640: 100% 44/44 [00:17<00:00,  2.52it/s]
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.21it/s]
        all        142        132      0.388      0.348      0.385      0.138

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
12/100     4.02G      2.144      2.241      2.069         51        640: 100% 44/44 [00:18<00:00,  2.33it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.52it/s]
        all        142        132      0.479      0.409       0.46      0.154

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
13/100     4.05G      2.103      2.176      2.073         19        640: 100% 44/44 [00:17<00:00,  2.52it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.16it/s]
        all        142        132      0.442      0.356      0.354      0.109

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
14/100     4.05G      2.074       2.09      2.002         47        640: 100% 44/44 [00:17<00:00,  2.59it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.37it/s]
        all        142        132      0.465      0.416       0.38      0.121

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
15/100     3.86G      2.053      2.121      2.009         23        640: 100% 44/44 [00:16<00:00,  2.63it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.35it/s]
        all        142        132      0.451      0.492      0.441      0.145

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
16/100     3.84G      2.057      2.117      2.012         23        640: 100% 44/44 [00:17<00:00,  2.53it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.52it/s]
        all        142        132      0.242       0.28        0.2     0.0622

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
17/100       4G      2.044      2.055      2.008         34        640: 100% 44/44 [00:16<00:00,  2.64it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.43it/s]
        all        142        132      0.459      0.583      0.442      0.136

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
18/100     4.03G      2.026       2.06      2.025         34        640: 100% 44/44 [00:16<00:00,  2.60it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.32it/s]
        all        142        132      0.542      0.485      0.496      0.153

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
19/100     3.86G      2.007      2.024      2.004         27        640: 100% 44/44 [00:17<00:00,  2.56it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.35it/s]
        all        142        132      0.486      0.602       0.52      0.173

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
20/100     4.03G      2.013      1.972      1.972         25        640: 100% 44/44 [00:17<00:00,  2.55it/s]
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  2.90it/s]
        all        142        132       0.43      0.515      0.461      0.164

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
21/100      4.02G      1.957      1.968      1.948         33        640: 100% 44/44 [00:16<00:00,  2.61it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.49it/s]
        all        142        132       0.35        0.5      0.369      0.127

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
22/100      4.02G      2.007      1.962       1.97         39        640: 100% 44/44 [00:17<00:00,  2.56it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.15it/s]
        all        142        132       0.436      0.593      0.484      0.161

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
23/100      3.86G      1.912      1.861      1.873         39        640: 100% 44/44 [00:17<00:00,  2.51it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.46it/s]
        all        142        132       0.458      0.606      0.495      0.176

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
24/100      4.02G      1.965      1.879      1.935         35        640: 100% 44/44 [00:16<00:00,  2.74it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  2.62it/s]
        all        142        132       0.534      0.616      0.535      0.191

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
25/100      4.05G      1.932      1.795      1.881         30        640: 100% 44/44 [00:17<00:00,  2.51it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.16it/s]
        all        142        132       0.424      0.583      0.446      0.154

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
26/100      4.02G      1.919      1.852      1.878         16        640: 100% 44/44 [00:15<00:00,  2.88it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.01it/s]
        all        142        132        0.49      0.532      0.494      0.175

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
27/100      3.87G      1.921      1.865      1.928         31        640: 100% 44/44 [00:15<00:00,  2.78it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.25it/s]
        all        142        132       0.501      0.598      0.544      0.198

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
28/100      4.02G      1.917      1.768      1.888         51        640: 100% 44/44 [00:16<00:00,  2.71it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  1.90it/s]
        all        142        132       0.486      0.614      0.553      0.192

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
29/100        4G      1.873      1.769      1.833         33        640: 100% 44/44 [00:15<00:00,  2.77it/s]
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.24it/s]
        all        142        132      0.471      0.654      0.528      0.192

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
30/100      4.02G      1.878      1.766      1.842         20        640: 100% 44/44 [00:15<00:00,  2.86it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.05it/s]
        all        142        132      0.475      0.621      0.509       0.17

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
31/100      3.84G       1.86      1.737      1.838         37        640: 100% 44/44 [00:15<00:00,  2.84it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  1.92it/s]
        all        142        132      0.504      0.621      0.519      0.186

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
32/100      4.02G      1.861      1.691      1.812         34        640: 100% 44/44 [00:15<00:00,  2.89it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  1.80it/s]
        all        142        132      0.463      0.629      0.485      0.166

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
33/100      4.05G      1.833      1.697      1.816         33        640: 100% 44/44 [00:14<00:00,  2.94it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.10it/s]
        all        142        132      0.518      0.576      0.538      0.196

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
34/100      4.02G      1.858      1.706       1.82         30        640: 100% 44/44 [00:15<00:00,  2.90it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  2.02it/s]
        all        142        132      0.468      0.621      0.497      0.205

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
35/100      3.86G      1.848      1.692      1.833         33        640: 100% 44/44 [00:14<00:00,  2.94it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  1.96it/s]
        all        142        132      0.548      0.614      0.549      0.203

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
36/100      4.02G      1.811      1.612      1.801         27        640: 100% 44/44 [00:20<00:00,  2.19it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:00,  1.91it/s]
        all        142        132      0.494      0.644      0.525      0.201

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
37/100      4.02G      1.788      1.609      1.778         38        640: 100% 44/44 [00:19<00:00,  2.30it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.36it/s]
        all        142        132      0.501      0.583       0.51      0.197

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
38/100      3.97G      1.809      1.596      1.768         18        640: 100% 44/44 [00:17<00:00,  2.54it/s]
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.20it/s]
        all        142        132      0.523      0.557      0.521      0.186

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
39/100      3.87G      1.778      1.585      1.776         15        640: 100% 44/44 [00:17<00:00,  2.54it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.23it/s]
        all        142        132      0.619      0.523      0.519      0.191

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
40/100      4.02G      1.768      1.575       1.76         28        640: 100% 44/44 [00:18<00:00,  2.34it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.16it/s]
        all        142        132       0.58      0.598      0.539      0.199

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
41/100        4G      1.728      1.528      1.722         43        640: 100% 44/44 [00:17<00:00,  2.55it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.38it/s]
        all        142        132      0.599      0.598      0.572      0.202

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
42/100        4G      1.749      1.518      1.741         44        640: 100% 44/44 [00:17<00:00,  2.48it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.49it/s]
        all        142        132      0.561      0.598      0.591      0.215

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
43/100      3.86G      1.747      1.477      1.737         30        640: 100% 44/44 [00:17<00:00,  2.49it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.49it/s]
        all        142        132      0.672      0.568      0.618      0.217

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
44/100        4G      1.734      1.479       1.73         40        640: 100% 44/44 [00:17<00:00,  2.49it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.45it/s]
        all        142        132      0.594      0.636      0.603      0.201

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
45/100      4.02G      1.716      1.495      1.728         41        640: 100% 44/44 [00:17<00:00,  2.54it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.47it/s]
        all        142        132      0.544      0.606      0.501      0.174

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
46/100      4.02G      1.713       1.48      1.681         37        640: 100% 44/44 [00:17<00:00,  2.57it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.24it/s]
        all        142        132       0.46      0.674      0.532        0.2

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
47/100      3.81G      1.678      1.425      1.683         31        640: 100% 44/44 [00:17<00:00,  2.53it/s]
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.37it/s]
        all        142        132      0.629      0.621      0.593      0.212

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
48/100         4G      1.674      1.378      1.676         25        640: 100% 44/44 [00:17<00:00,  2.48it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.30it/s]
        all        142        132      0.616      0.591      0.594      0.226

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
49/100         4G      1.646      1.371      1.702         47        640: 100% 44/44 [00:17<00:00,  2.53it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.50it/s]
        all        142        132      0.516      0.653      0.568       0.22

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
50/100      3.97G      1.631      1.351      1.644         21        640: 100% 44/44 [00:17<00:00,  2.54it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.49it/s]
        all        142        132       0.71      0.606      0.656      0.243

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
51/100      3.86G      1.624      1.327      1.666         30        640: 100% 44/44 [00:17<00:00,  2.55it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.25it/s]
        all        142        132      0.583      0.667      0.624      0.229

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
52/100      4.02G      1.601      1.314      1.625         27        640: 100% 44/44 [00:17<00:00,  2.53it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.51it/s]
        all        142        132      0.504      0.689      0.529      0.187

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
53/100      4.02G       1.57      1.234       1.59         30        640: 100% 44/44 [00:16<00:00,  2.63it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.35it/s]
        all        142        132      0.609      0.674      0.567        0.2

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
54/100      4.02G      1.619      1.284      1.628         23        640: 100% 44/44 [00:17<00:00,  2.55it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.38it/s]
        all        142        132      0.615      0.689      0.632      0.216

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
55/100      3.87G      1.584       1.25      1.621         49        640: 100% 44/44 [00:17<00:00,  2.51it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.24it/s]
        all        142        132      0.524       0.72        0.6      0.206

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
56/100      4.02G      1.569       1.24      1.582         52        640: 100% 44/44 [00:17<00:00,  2.58it/s]
```

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.64it/s]
        all        142        132       0.62      0.631       0.56      0.196

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
57/100     4.02G      1.539      1.239      1.591         21        640: 100% 44/44 [00:16<00:00,  2.62it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.23it/s]
        all        142        132      0.617      0.697      0.651      0.219

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
58/100     4.05G      1.547      1.189      1.577         21        640: 100% 44/44 [00:16<00:00,  2.64it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.07it/s]
        all        142        132       0.55      0.644      0.605      0.209

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
59/100     3.79G      1.538      1.196      1.577         39        640: 100% 44/44 [00:17<00:00,  2.50it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.43it/s]
        all        142        132      0.609      0.652      0.611      0.201

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
60/100     4.02G      1.507      1.121      1.551         32        640: 100% 44/44 [00:17<00:00,  2.52it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.28it/s]
        all        142        132      0.586      0.667      0.606      0.221

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
61/100     4.02G      1.503      1.137      1.531         44        640: 100% 44/44 [00:17<00:00,  2.50it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.43it/s]
        all        142        132      0.593      0.629       0.59      0.226

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
62/100     4.02G      1.483      1.142      1.544         25        640: 100% 44/44 [00:17<00:00,  2.56it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.27it/s]
        all        142        132      0.483      0.659      0.529      0.199

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
63/100     3.84G       1.48      1.124      1.535         51        640: 100% 44/44 [00:17<00:00,  2.52it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.50it/s]
        all        142        132      0.577      0.661      0.558      0.204

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
64/100     4.02G       1.46      1.102      1.516         25        640: 100% 44/44 [00:16<00:00,  2.61it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.15it/s]
        all        142        132      0.627      0.714      0.632      0.219

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
65/100     4.02G      1.439      1.092      1.479         19        640: 100% 44/44 [00:17<00:00,  2.51it/s]
```

Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.41it/s]
all 142 132 0.684 0.682 0.64 0.233

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
66/100 4.03G 1.42 1.05 1.481 24 640: 100% 44/44 [00:17<00:00, 2.51it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.22it/s]
all 142 132 0.627 0.667 0.622 0.23

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
67/100 3.9G 1.427 1.022 1.484 26 640: 100% 44/44 [00:16<00:00, 2.62it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.22it/s]
all 142 132 0.602 0.689 0.641 0.226

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
68/100 4.02G 1.39 1.009 1.47 28 640: 100% 44/44 [00:18<00:00, 2.38it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.36it/s]
all 142 132 0.633 0.707 0.663 0.235

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
69/100 3.97G 1.387 1.024 1.479 21 640: 100% 44/44 [00:16<00:00, 2.62it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.31it/s]
all 142 132 0.644 0.621 0.606 0.209

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
70/100 4.02G 1.382 1.006 1.456 48 640: 100% 44/44 [00:17<00:00, 2.49it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.43it/s]
all 142 132 0.606 0.591 0.59 0.219

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
71/100 3.87G 1.349 0.954 1.436 24 640: 100% 44/44 [00:17<00:00, 2.53it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.42it/s]
all 142 132 0.64 0.636 0.611 0.227

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
72/100 4G 1.373 0.9958 1.448 26 640: 100% 44/44 [00:17<00:00, 2.55it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.19it/s]
all 142 132 0.728 0.629 0.682 0.232

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
73/100 4.02G 1.312 0.9433 1.408 36 640: 100% 44/44 [00:17<00:00, 2.56it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.45it/s]
all 142 132 0.62 0.674 0.591 0.211

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
74/100 3.9G 1.35 0.9707 1.42 23 640: 100% 44/44 [00:16<00:00, 2.63it/s]

```
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.39it/s]
        all        142        132      0.641      0.691      0.629      0.228

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
75/100     3.86G      1.296     0.9286      1.404         35        640: 100% 44/44 [00:17<00:00,  2.47it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.18it/s]
        all        142        132      0.683      0.605      0.618      0.222

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
76/100       4G      1.288     0.9285      1.388         34        640: 100% 44/44 [00:16<00:00,  2.63it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.43it/s]
        all        142        132       0.64      0.673      0.645      0.216

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
77/100     3.98G      1.249     0.8924      1.369         29        640: 100% 44/44 [00:17<00:00,  2.53it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.17it/s]
        all        142        132      0.665      0.693      0.695      0.248

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
78/100     4.02G      1.275     0.8982      1.378         32        640: 100% 44/44 [00:18<00:00,  2.41it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.51it/s]
        all        142        132       0.75      0.613      0.671      0.235

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
79/100     3.86G      1.264     0.9016      1.381         18        640: 100% 44/44 [00:17<00:00,  2.58it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.32it/s]
        all        142        132      0.691      0.667      0.672      0.232

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
80/100     3.98G      1.227     0.8542      1.349         32        640: 100% 44/44 [00:17<00:00,  2.53it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.41it/s]
        all        142        132      0.709      0.701      0.661      0.242

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
81/100     4.01G      1.254      0.882      1.364         28        640: 100% 44/44 [00:16<00:00,  2.60it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.24it/s]
        all        142        132      0.678      0.682      0.626      0.222

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
82/100     4.02G      1.212     0.8544      1.341         36        640: 100% 44/44 [00:16<00:00,  2.59it/s]
      Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.61it/s]
        all        142        132      0.674      0.644       0.63      0.231

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
83/100     3.87G      1.229     0.8409      1.341         31        640: 100% 44/44 [00:17<00:00,  2.51it/s]
```

Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.45it/s]
all 142 132 0.677 0.682 0.686 0.245

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
84/100 4.02G 1.216 0.8426 1.323 34 640: 100% 44/44 [00:17<00:00, 2.59it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.30it/s]
all 142 132 0.663 0.591 0.657 0.249

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
85/100 4.02G 1.188 0.8281 1.327 36 640: 100% 44/44 [00:17<00:00, 2.46it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.54it/s]
all 142 132 0.684 0.706 0.67 0.23

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
86/100 4.02G 1.177 0.8376 1.307 28 640: 100% 44/44 [00:17<00:00, 2.57it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.58it/s]
all 142 132 0.737 0.68 0.682 0.247

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
87/100 3.84G 1.171 0.8404 1.314 28 640: 100% 44/44 [00:16<00:00, 2.60it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.45it/s]
all 142 132 0.726 0.681 0.664 0.235

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
88/100 4.05G 1.126 0.8022 1.283 25 640: 100% 44/44 [00:17<00:00, 2.49it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.38it/s]
all 142 132 0.715 0.674 0.654 0.229

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
89/100 4G 1.133 0.7924 1.283 88 640: 100% 44/44 [00:17<00:00, 2.51it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.47it/s]
all 142 132 0.723 0.674 0.685 0.249

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
90/100 4G 1.151 0.8025 1.295 23 640: 100% 44/44 [00:17<00:00, 2.57it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.39it/s]
all 142 132 0.692 0.72 0.66 0.234

Closing dataloader mosaic

albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3, method='weighted_average'), CLAHE(p=0.01, clip_limit=(1, 4.0), tile_grid_size=(8, 8))

Epoch GPU_mem box_loss cls_loss dfl_loss Instances Size
91/100 3.84G 1.122 0.7354 1.281 14 640: 100% 44/44 [00:19<00:00, 2.30it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:01<00:00, 3.71it/s]
all 142 132 0.699 0.667 0.673 0.234

```
 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
92/100     4.02G      1.066     0.6545      1.245         43        640: 100% 44/44 [00:17<00:00,  2.57it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.62it/s]
                 all        142        132      0.716      0.705      0.685      0.243

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
93/100       4G      1.031     0.6326      1.226         14        640: 100% 44/44 [00:16<00:00,  2.61it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.60it/s]
                 all        142        132      0.666       0.71      0.673      0.234

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
94/100       4G      1.017     0.6228      1.214         24        640: 100% 44/44 [00:17<00:00,  2.51it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.58it/s]
                 all        142        132      0.754      0.697      0.696      0.243

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
95/100     3.84G     0.9907     0.5984      1.202         13        640: 100% 44/44 [00:17<00:00,  2.57it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.15it/s]
                 all        142        132      0.733      0.697      0.677      0.241

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
96/100       4G     0.9957     0.6119      1.185         15        640: 100% 44/44 [00:17<00:00,  2.49it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.68it/s]
                 all        142        132      0.701      0.659      0.664      0.235

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
97/100       4G     0.9656     0.5816      1.176         13        640: 100% 44/44 [00:17<00:00,  2.50it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.56it/s]
                 all        142        132      0.748      0.698      0.673      0.233

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
98/100       4G     0.9748     0.5964       1.19         17        640: 100% 44/44 [00:17<00:00,  2.53it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.59it/s]
                 all        142        132      0.749      0.689      0.683      0.238

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
99/100     3.84G     0.9596     0.5877       1.18         23        640: 100% 44/44 [00:17<00:00,  2.57it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.55it/s]
                 all        142        132      0.736      0.682      0.682      0.238

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
100/100    4.03G     0.9615     0.5946      1.182         15        640: 100% 44/44 [00:17<00:00,  2.53it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:01<00:00,  3.37it/s]
                 all        142        132      0.743      0.667      0.666      0.237
```

100 epochs completed in 0.558 hours.

Optimizer stripped from runs/detect/train2/weights/last.pt, 22.5MB

Optimizer stripped from runs/detect/train2/weights/best.pt, 22.5MB

Validating runs/detect/train2/weights/best.pt...

Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)

Model summary (fused): 168 layers, 11,126,358 parameters, 0 gradients, 28.4 GFLOPs

Class Images Instances Box(P R mAP50 mAP50-95): 100% 5/5 [00:03<00:00, 1.57it/s]

all 142 132 0.667 0.697 0.695 0.248

Galha_v 107 132 0.667 0.697 0.695 0.248

Speed: 0.5ms preprocess, 5.3ms inference, 0.0ms loss, 6.3ms postprocess per image

Results saved to runs/detect/train2

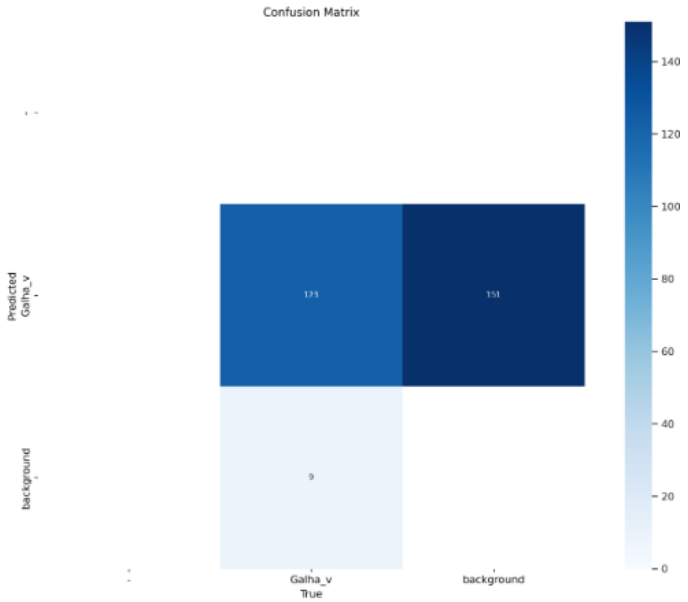
💡 Learn more at https://docs.ultralytics.com/modes/train

7.1.2 Parametrização de Roboflow e Google Colab para treino e resultados para 100 epochs:

```
!nvidia-smi
import os
HOME = os.getcwd()
print(HOME)
!pip install ultralytics==8.2.103 -q

from IPython import display
display.clear_output()

import ultralytics
ultralytics.checks()
from ultralytics import YOLO

from IPython.display import display, Image
!mkdir -p {HOME}/datasets
%cd {HOME}/datasets

!pip install roboflow --quiet

import roboflow
```

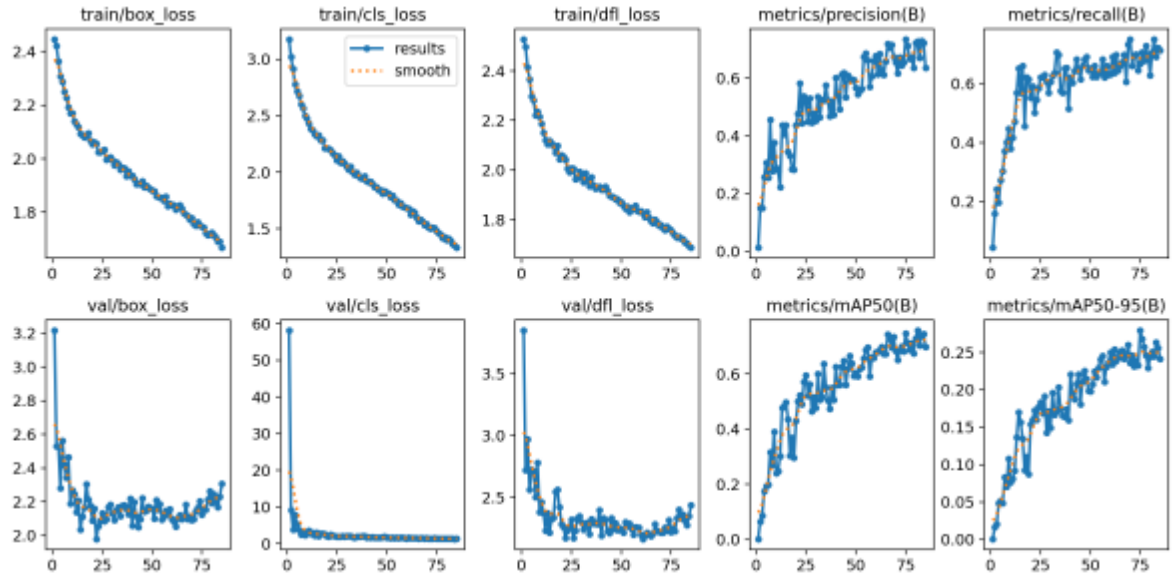
```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_keyPrivate key")
project = rf.workspace("vespa-mjis3").project("vespa-detection")
version = project.version(13)
dataset = version.download("yolov8")
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/YOLOv8
!yolo task=detect mode=train model=yolov8s.pt
data={dataset.location}/data.yaml epochs=100 imgsz=640 plots=True \
    patience=10 optimizer=AdamW lr0=0.01 weight_decay=0.0005
augment=True save_period=10
 %cd /content/drive/MyDrive/YOLOv8
Image(filename=f'/content/drive/MyDrive/YOLOv8/runs/detect/train/conf
usion_matrix.png', width=600)
```

```
/content/drive/MyDrive/YOLOv8
```




```
 %cd /content/drive/MyDrive/YOLOv8
Image(filename=f'/content/drive/MyDrive/YOLOv8/runs/detect/train/resu
lts.png', width=600)
```

```
/content/drive/MyDrive/YOLOv8
```




```
%cd /content/drive/MyDrive/YOLOv8
Image(filename=f'/content/drive/MyDrive/YOLOv8/runs/detect/train/val_batch0_pred.jpg', width=600)
```

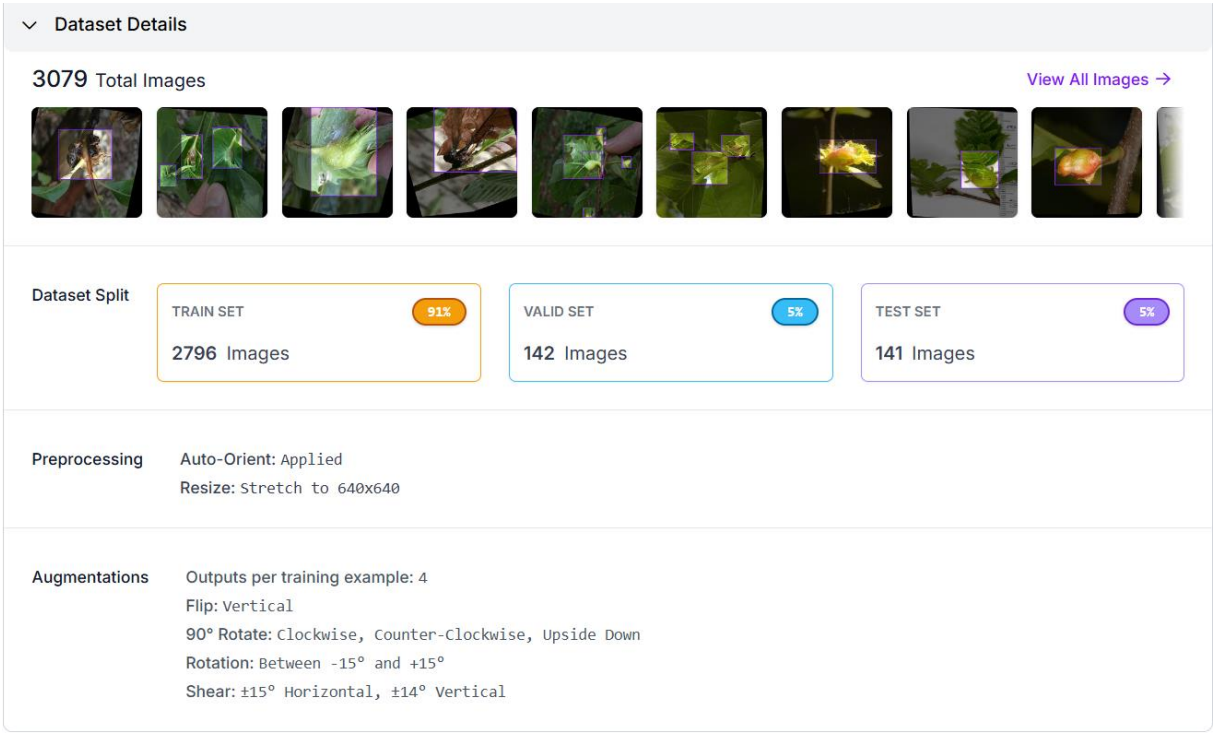
```
/content/drive/MyDrive/YOLOv8
```

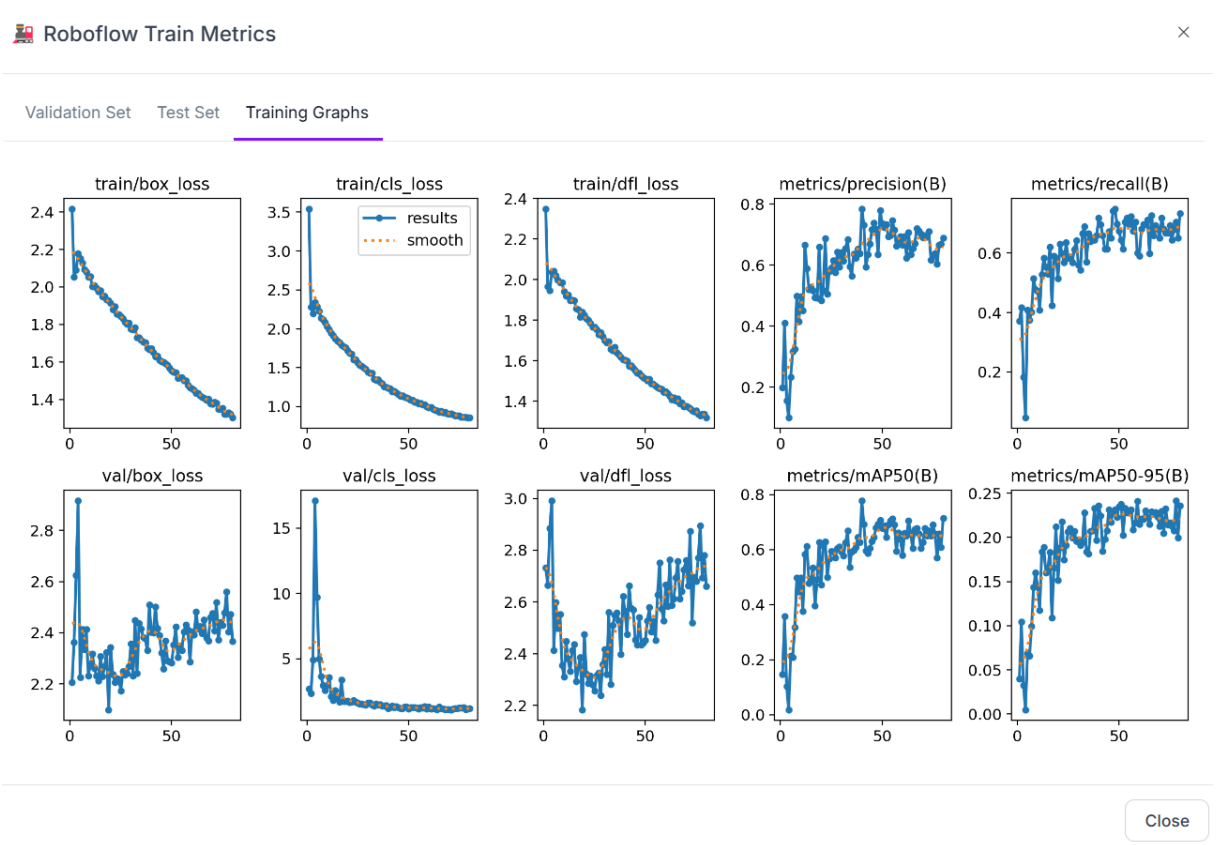

```
%cd /content/drive/MyDrive/YOLOv8
```

```
!yolo task=detect mode=val
model=/content/drive/MyDrive/YOLOv8/runs/detect/train/weights/best.pt
data={dataset.location}/data.yaml
```

```
/content/drive/MyDrive/YOLOv8
Ultralytics YOLOv8.2.103 🚀 Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11,126,358 parameters, 0 gradients, 28.4 GFLOPs
val: Scanning /content/datasets/Vespa-Detection-13/valid/labels.cache... 142 images, 35 backgrounds, 0 corrupt: 100% 142/142 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 9/9 [00:03<00:00,  2.32it/s]
                   all        142        132      0.734      0.667      0.746      0.276
               Galha_v        107        132      0.734      0.667      0.746      0.276
Speed: 1.7ms preprocess, 10.8ms inference, 0.0ms loss, 6.5ms postprocess per image
Results saved to runs/detect/val
💡 Learn more at https://docs.ultralytics.com/modes/val
```

7.1.3 Parametrização de Roboflow





v13 **2024-10-25 7:29pm**

Generated on Oct 25, 2024

Download Dataset | Edit

vespa-detection/13

Model Type: Roboflow 3.0 Object Detection (Accurate)
Checkpoint: COCOs

mAP	Precision	Recall
77.7%	78.5%	72.0%

Detailed Model Evaluation | Performance By Class | Visualize Model

Deploy Your Model

Try This Model

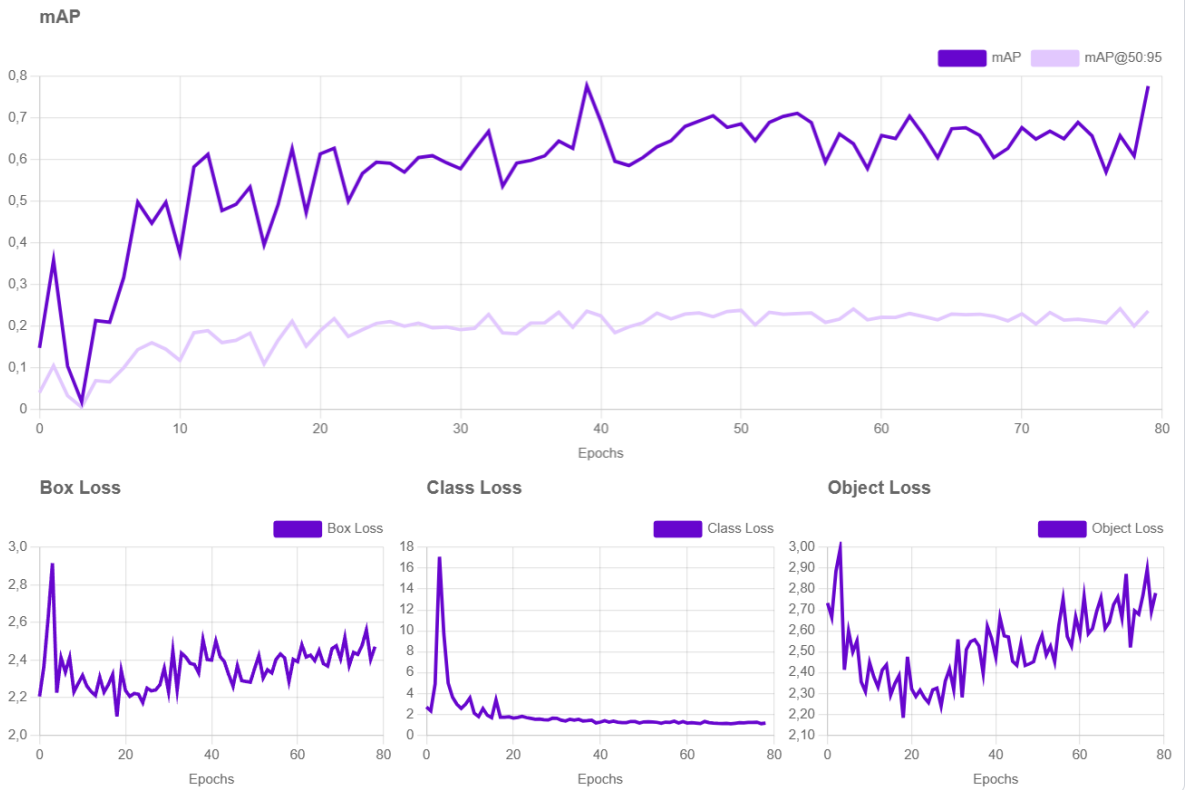
Try on mobile

- **Try Workflows** — Configure, integrate, and deploy your vision model
- **Use Curl Command** — Infer via command line
- **Code Samples** — In many languages
- **Example Web App** — Full sample code
- **Use Your Webcam** — Realtime in your browser
- **Deploy To NVIDIA Jetson** — Edge inference via Docker
- **Deploy To Luxonis OAK** — Edge inference via Docker

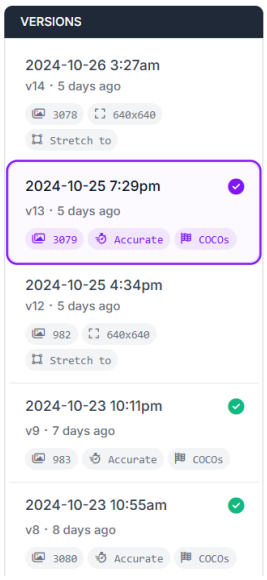
> Training Graphs

> Dataset Details

Training Graphs



Últimos testes efetuados com Roboflow 3.0:



7.1.4 Código fonte YOLOv8 Test Interface

```
import tkinter as tk
from tkinter import filedialog, messagebox
import cv2
from ultralytics import YOLO
import os
import threading

class YOLOTestApp:
    def __init__(self, root):
        self.root = root
        self.root.title("YOLOv8 Test Interface")
        self.root.geometry("800x600")

        # Variável para controlar o vídeo
        self.stop_video = False
        self.pause_video = False
        self.video_speed = 1

        # Variável para controlar a execução de imagens
        self.stop_image = False

        # Contador de imagens com detecções de doenças
        self.disease_count = 0
        # Contador de imagens sem detecções de doenças
        self.no_disease_count = 0
        self.total_detections = 0
        self.file_paths = []

        # Janela para selecionar diretório do modelo
        self.select_model_directory()

    def select_model_directory(self):
```

```
            model_path = filedialog.askopenfilename(title="Selecione o
Modelo YOLO", filetypes=[("YOLO Model Files", "*.pt")])
            if model_path:
                self.model = YOLO(model_path)
                self.model_path = model_path
                self.open_main_interface()
            else:
                messagebox.showerror("Erro", "Por favor, selecione um
modelo YOLO para continuar.")
                self.root.quit()

        def open_main_interface(self):
            # Label para mostrar o modelo selecionado
            self.model_label = tk.Label(self.root, text=f"Modelo atual:
{os.path.basename(self.model_path)}", wraplength=400)
            self.model_label.pack(pady=10)

            # Botão para selecionar arquivo
            self.select_button = tk.Button(self.root, text="Selecionar
Arquivos", command=self.browse_files)
            self.select_button.pack(pady=10)

            # Botão para parar o processamento atual
            self.stop_button = tk.Button(self.root, text="Parar
Processamento", command=self.stop_processing)
            self.stop_button.pack(pady=10)

            # Botão para selecionar outro modelo
            self.change_model_button = tk.Button(self.root,
text="Selecionar Outro Modelo", command=self.change_model)
            self.change_model_button.pack(pady=10)

            # Botão para contagem total
            self.count_button = tk.Button(self.root, text="Contagem
Total", command=self.total_count)
            self.count_button.pack(pady=10)

            # Label para mostrar o número de imagens com detecções de
doenças
            self.disease_count_label = tk.Label(self.root, text="Imagens
com detecções de doenças: 0", wraplength=400)
            self.disease_count_label.pack(pady=10)

            # Botão para abrir a janela de controle do vídeo
            self.video_controls_button = tk.Button(self.root,
text="Controlo de Vídeo", command=self.open_video_controls_window)
            self.video_controls_button.pack(pady=10)

        def browse_files(self):
            self.file_paths =
filedialog.askopenfilenames(filetypes=[("Image and Video Files",
"*.jpg;*.jpeg;*.png;*.mp4;*.avi")])
            if self.file_paths:
                # Executar o processamento em uma thread separada
                threading.Thread(target=self.process_files,
args=(self.file_paths,)).start()

        def stop_processing(self):
            # Parar o processamento atual
            self.stop_video = True
            self.stop_image = True
```

```
        def change_model(self):
            # Parar o processamento atual antes de mudar o modelo
            self.stop_processing()
            # Selecionar um novo modelo
            model_path = filedialog.askopenfilename(title="Selecione o
Modelo YOLO", filetypes=[("YOLO Model Files", "*.pt")])
            if model_path:
                self.model = YOLO(model_path)
                self.model_path = model_path
                self.model_label.config(text=f"Modelo atual:
{os.path.basename(model_path)}")
            else:
                messagebox.showerror("Erro", "Por favor, selecione um
modelo YOLO para continuar.")

        def process_files(self, file_paths):
            self.disease_count = 0
            self.no_disease_count = 0
            self.total_detections = 0
            for idx, file_path in enumerate(file_paths, start=1):
                if file_path.endswith(('.jpg', '.jpeg', '.png')):
                    self.process_image(file_path, idx)
                elif file_path.endswith(('.mp4', '.avi')):
                    self.process_video(file_path, idx)

            # Atualizar o label com o número de imagens com doenças
            self.disease_count_label.config(text=f"Imagens com detecções
de doenças: {self.disease_count}")

        def process_image(self, image_path, idx):
            # Carregar a imagem
            image = cv2.imread(image_path)
            self.stop_image = False
            # Realizar a detecção usando YOLOv8
            results = self.model(image)
            # Contabilizar imagens com detecção de doenças e número total
de detecções
            if len(results[0].boxes) > 0:
                self.disease_count += 1
                self.total_detections += len(results[0].boxes)
            else:
                self.no_disease_count += 1
                self.total_detections += len(results[0].boxes)
            # Mostrar os resultados
            for result in results:
                if self.stop_image:
                    break
                annotated_frame = result.plot()
                cv2.imshow(f"Resultado - Imagem {idx}:
{os.path.basename(image_path)}", annotated_frame)
                # Colocar a janela em destaque
                try:
                    cv2.setWindowProperty(f"Resultado - Imagem {idx}:
{os.path.basename(image_path)}", cv2.WND_PROP_TOPMOST, 1)
                except cv2.error:
                    print("Erro ao definir a janela como topmost.
Ignorando esta propriedade.")
                # Manter a imagem visível até que o usuário pare
manualmente
                while True:
```

```
                    if self.stop_image:
                        break
                    if cv2.waitKey(1) & 0xFF == ord('q'):
                        break
            cv2.destroyAllWindows()

        def process_video(self, video_path, idx):
            # Carregar o vídeo
            cap = cv2.VideoCapture(video_path)
            self.stop_video = False
            self.pause_video = False
            while cap.isOpened():
                if self.pause_video:
                    if cv2.waitKey(1) & 0xFF == ord('p'):
                        self.pause_video = not self.pause_video
                    continue

                ret, frame = cap.read()
                if not ret or self.stop_video:
                    break

                # Realizar a detecção usando YOLOv8
                results = self.model(frame)

                # Mostrar os resultados
                for result in results:
                    annotated_frame = result.plot()
                    cv2.imshow(f"Resultado - Vídeo {idx}:
{os.path.basename(video_path)}", annotated_frame)
                    # Colocar a janela em destaque
                    try:
                        cv2.setWindowProperty(f"Resultado - Vídeo {idx}:
{os.path.basename(video_path)}", cv2.WND_PROP_TOPMOST, 1)
                    except cv2.error:
                        print("Erro ao definir a janela como topmost.
Ignorando esta propriedade.")

                    # Verificar se o usuário deseja parar o processamento
ou fechar o vídeo
                    key = cv2.waitKey(int(1000 / (30 *
self.video_speed))) & 0xFF
                    if key == ord('q'):
                        self.stop_video = True
                        break
                    elif key == ord('p'):
                        self.pause_video = not self.pause_video
                    elif key == ord('+'):
                        self.video_speed = min(self.video_speed + 0.5,
5.0)  # Aumenta a velocidade do vídeo
                    elif key == ord('-'):
                        self.video_speed = max(self.video_speed - 0.5,
0.1)  # Diminui a velocidade do vídeo
                    elif key == ord('f'):
                        cap.set(cv2.CAP_PROP_POS_FRAMES,
cap.get(cv2.CAP_PROP_POS_FRAMES) + 30)  # Avança 30 frames
                    elif key == ord('b'):
                        cap.set(cv2.CAP_PROP_POS_FRAMES, max(0,
cap.get(cv2.CAP_PROP_POS_FRAMES) - 30))  # Retrocede 30 frames

            cap.release()
            cv2.destroyAllWindows()
```

```
        def open_video_controls_window(self):
            controls_window = tk.Toplevel(self.root)
            controls_window.title("Controlo de Vídeo")
            controls_window.geometry("300x200")

            # Instruções para controlar o vídeo
            instructions = (
                "Controlo do Vídeo:\n"
                "- 'q': Sair do vídeo\n"
                "- 'p': Pausar/Retomar o vídeo\n"
                "- '+': Aumentar a velocidade do vídeo\n"
                "- '-': Diminuir a velocidade do vídeo\n"
                "- 'f': Avançar 30 frames\n"
                "- 'b': Retroceder 30 frames"
            )
            instructions_label = tk.Label(controls_window,
text=instructions, justify="left")
            instructions_label.pack(pady=10)

            # Slider para controlar a velocidade do vídeo
            self.speed_slider = tk.Scale(controls_window, from_=0.1,
to=5.0, resolution=0.1, orient="horizontal", label="Velocidade do Vídeo")
            self.speed_slider.set(self.video_speed)
            self.speed_slider.pack(pady=10)
            self.speed_slider.bind("<Motion>", self.update_video_speed)

        def update_video_speed(self, event):
            self.video_speed = self.speed_slider.get()

        def total_count(self):
            images_without_detections = len(self.file_paths) -
self.disease_count
            messagebox.showinfo("Contagem Total", f"Total de arquivos a
verificar: {len(self.file_paths)}\nTotal de imagens com detecções:
{self.disease_count}\nTotal de imagens sem detecções:
{self.no_disease_count}\nTotal de detecções: {self.total_detections}")

    if __name__ == "__main__":
        root = tk.Tk()
        app = YOLOTestApp(root)
        root.mainloop()
```

7.2 Materiais suplementares que suportam a pesquisa

7.2.1 Regulamentação Portuguesa para o Uso de Drones

Licenciamento e Registo de Operadores

Em Portugal, qualquer operador de drones cujo dispositivo pese mais de 250 gramas ou seja utilizado para fins comerciais, incluindo a agricultura, deve registar-se na ANAC. Este registo é obrigatório e visa garantir que os operadores tenham conhecimento das regras de segurança e das limitações impostas para a operação dos drones. Além disso, todos os drones utilizados para fins profissionais, como a monitorização de pragas agrícolas, devem estar registados junto da ANAC, com uma identificação clara do operador [9].

Regras de Operação no Espaço Aéreo

A operação de drones em Portugal está sujeita a uma série de restrições e requisitos que variam conforme o tipo de operação e a área de voo. De acordo com a ANAC, os drones devem ser operados dentro da linha de visão direta do piloto (VLOS – Visual Line of Sight) e a uma altitude máxima de 120 metros acima do solo. Esta limitação de altura visa evitar conflitos com o tráfego aéreo tripulado, que opera em altitudes superiores [10].

Além disso, existem zonas onde o voo de drones é estritamente proibido ou limitado, tais como áreas próximas de aeroportos, instalações militares e outras infraestruturas críticas. Para operar em áreas restritas ou fora da linha de visão, é necessário obter uma autorização específica da ANAC. A operação noturna também é restrita e requer autorização prévia [11].

Formação e Certificação de Pilotos

Os pilotos de drones utilizados para fins comerciais, como na agricultura, devem possuir uma certificação específica. Esta formação inclui conhecimentos sobre as regras de operação, segurança aérea, meteorologia, e gestão de risco. Em Portugal, a formação de pilotos de drones é oferecida por entidades certificadas pela ANAC, e a certificação é obrigatória para operações profissionais [12].

Os operadores são também obrigados a garantir que os seus drones estão em condições seguras de operação, realizando manutenções regulares e verificações antes de cada voo. Qualquer incidente ou acidente envolvendo drones deve ser reportado à ANAC, em conformidade com a regulamentação vigente.

Proteção de Dados e Privacidade

Uma preocupação significativa no uso de drones, especialmente em áreas agrícolas próximas a zonas residenciais, é a proteção de dados e a privacidade. A legislação portuguesa obriga os operadores de drones a respeitar a privacidade dos indivíduos e a garantir que os

dados capturados, como imagens e vídeos, são tratados em conformidade com o Regulamento Geral sobre a Proteção de Dados (RGPD). Isto significa que a recolha de imagens de pessoas ou propriedades sem consentimento é proibida, e os operadores devem garantir que os dados são armazenados e utilizados de forma segura [13].

Seguro de Responsabilidade Civil

Outro requisito importante para a operação de drones em Portugal é a necessidade de seguro de responsabilidade civil. Este seguro cobre danos a terceiros que possam resultar da operação do drone, incluindo danos materiais e pessoais. A contratação deste seguro é obrigatória para drones utilizados em operações comerciais, incluindo na agricultura, e deve ser mantido ativo durante todo o período de operação [10].

Impacto da Regulamentação no Uso de Drones na Agricultura

A regulamentação rigorosa imposta pela ANAC garante que o uso de drones na agricultura em Portugal seja seguro e responsável, protegendo tanto os operadores como o público em geral. No entanto, estas normas também representam um desafio para os agricultores que desejam integrar drones nas suas práticas de monitorização de pragas e gestão de culturas.

O cumprimento dos requisitos de licenciamento, registo e certificação pode ser oneroso, especialmente para pequenos agricultores. Além disso, a necessidade de obter autorizações para voar em certas áreas ou em condições específicas pode limitar a flexibilidade na utilização dos drones. No entanto, os benefícios proporcionados pelos drones, em termos de eficiência e precisão na monitorização das culturas, frequentemente superam estas dificuldades, tornando-os uma ferramenta cada vez mais popular na agricultura moderna.

A evolução contínua da tecnologia de drones e o potencial ajustamento das regulamentações para acomodar as necessidades específicas do setor agrícola poderão tornar a sua utilização ainda mais acessível no futuro, contribuindo para a sustentabilidade e competitividade da agricultura em Portugal.

9. ANAC – Autoridade Nacional da Aviação Civil. "Regulamentação para Operação de Drones em Portugal." Disponível em: www.anac.pt.

10. Decreto-Lei n.º 58/2018, de 23 de Julho. "Regras Gerais para a Utilização de Drones no Espaço Aéreo Português."

11. ANAC – Autoridade Nacional da Aviação Civil. "Guia para Operadores de Drones." 2021. Disponível em: www.anac.pt.

12. Portal do Cidadão. "Formação e Certificação de Pilotos de Drones." Disponível em: www.portaldocidadao.pt.

13. Comissão Nacional de Proteção de Dados (CNPD). "Regras de Privacidade para o Uso de Drones." Disponível em: www.cnpd.pt.