Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

TomatoHealth

uma alternativa à escassez de datasets open-source de visão computacional para a agricultura

Heitor Barroso Cavalcante, Pedro Lucas R. S. Campos

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisora: Prof.ª Dr.ª Nina S. T. Hirata

Agradecimentos

"Agradeço primeiramente à minha orientadora, Profa. Dra. Nina S. T. Hirata, por ter aceitado me apoiar no início da minha vida acadêmica, proporcionando discussões valiosas e inúmeros aprendizados ao longo de todo o nosso período de colaboração.

Agradeço também à minha família, especialmente aos meus pais, Zákia e Martiniano, e à minha irmã Sofia, por seu incrível apoio e parceria em todos os momentos da minha vida. Obrigado por serem uma base sólida sobre a qual venho construindo a pessoa e o profissional que desejo me tornar.

Finalmente, agradeço ao IME-USP e à própria Universidade de São Paulo por promoverem excelentes oportunidades que pavimentaram minha trajetória acadêmica, desde bolsas de pesquisa, contato com profissionais de altíssimo nível e até mesmo um intercâmbio acadêmico internacional, que certamente contribuíram e continuam contribuindo significativamente para minha jornada."
– Heitor

"Agradeço à minha mãe, Ana Márcia, e ao meu pai, Danilo, por sempre me darem amor e suporte, e acreditarem em mim. Também agradeço à minha irmã, Giovana, que sempre foi uma inspiração para mim.

Ademais, agradeço ao IME-USP e às minhas orientadoras Profa. Dra. Nina S. T. Hirata (na orientação desta tese) e Profa. Dra. Kelly Rosa Braghetto (na orientação de minha iniciação científica), sem as quais não conseguiria entregar este trabalho.

Por fim gostaria de agradecer aos meus amigos Heitor e Hélcio e às gatas Sirigaita e Curianga, com os quais tive o prazer de compartilhar o apartamento durante a graduação."
– Pedro

Resumo

Heitor Barroso Cavalcante, Pedro Lucas R. S. Campos. **TomatoHealth: *uma alternativa à escassez de datasets open-source de visão computacional para a agricultura***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

Um problema latente da agricultura mundial atual é a perda da produção devido a problemas com pragas. De acordo com FAO (IPPC Secretariat, 2021), todo ano entre 20 e 40% da produção agrícola é perdida em função desses problemas. Na agricultura brasileira, a cultura do tomate é uma das principais fontes de emprego e renda, tendo um valor de produção bruto superior a 12,4 bilhões de reais em 2022, de acordo com dados do IBGE e uma matéria publicada no portal Revista Rural (*Tecnologias impulsionam produção nacional do tomate* 2022). Nesse contexto, os recentes avanços na área de Visão Computacional, devido ao contínuo melhoramento das técnicas de *deep learning*, tornam possível a implementação de modelos de detecção de doenças em imagens de plantas com o uso de modelos de redes neurais profundas. Entre os trabalhos produzidos, a maioria utiliza o *dataset open-source PlantVillage* (Hughes e Salathé, 2015). Contudo, uma série de questões relacionadas à qualidade dos dados desse conjunto faz com que a capacidade de generalização dos modelos treinados com ele deixe muito a desejar (Yao *et al.*, 2023). Dessa forma, fica evidente a necessidade de investir em alternativas para o desenvolvimento de conjuntos de dados abertos e robustos. Somente assim avançaremos no desenvolvimento de modelos de visão computacional aplicados à detecção de doenças em plantas, com resultados que realmente farão a diferença no dia a dia dos agricultores que se beneficiarão dessas ferramentas abertas e acessíveis. Por isso, em nosso trabalho, desenvolvemos o sistema *TomatoHealth*, que permite a identificação de doenças, o armazenamento de imagens enviadas por usuários em um *dataset* público, a rotulagem de imagens por meio de uma interface dedicada a usuários especialistas e o retreinamento do modelo empregado na plataforma com dados revisados.

Palavras-chave: *TomatoHealth*. Aprendizado de Máquina. Visão computacional. Detecção de objetos. *PlantVillage*. *PlantDoc*.

Abstract

Heitor Barroso Cavalcante, Pedro Lucas R. S. Campos. **TomatoHealth: *an alternative to the scarcity of open-source computer vision datasets for agriculture***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo.

A latent issue in modern global agriculture is the loss of production due to pest problems. According to the FAO (IPPC Secretariat, 2021), every year between 20% and 40% of agricultural production is lost due to these issues. In Brazilian agriculture, tomato cultivation is one of the main sources of employment and income, with a gross production value exceeding 12.4 billion reais in 2022, according to IBGE data and a report published on the Revista Rural portal (*Tecnologias impulsionam produção nacional do tomate* 2022). In this context, recent advancements in the field of Computer Vision, driven by the continuous improvement of deep learning techniques, have made it possible to implement disease detection models using deep neural networks on plant images. Among the works produced, most rely on the PlantVillage open-source dataset (Hughes and Salathé, 2015). However, several issues related to the quality of the data in this dataset significantly hinder the generalization capabilities of models trained with it (Yao *et al.*, 2023). Thus, it becomes evident that investing in alternatives for developing robust and open datasets is essential. Only by doing so can we advance the development of computer vision models for plant disease detection, achieving results that genuinely make a difference in the daily lives of farmers who will benefit from these open and accessible tools. Therefore, in our work, we developed the TomatoHealth system, which enables disease identification, the storage of user-submitted images in a public dataset, image labeling through a dedicated interface for expert users, and the retraining of the platform's model using reviewed data.

Keywords: TomatoHealth. Machine Learning. Computer Vision. Object Detection. PlantVillage. PlantDoc.

Lista de abreviaturas

IA	Inteligência Artificial (*Artificial Intelligence*)
CNN	Rede Neural Convolucional (*Convolutional Neural Network*)
RoI	Região de Interesse (*Region of Interest*)
IoU	Interseção sobre União (*Intersection over Union*)
RGB	Vermelho, Verde, Azul (*Red, Green, Blue*)
ML	Aprendizado de Máquina (*Machine Learning*)
SVM	Máquina de Vetores de Suporte (*Support Vector Machine*)
LLM	Grande Modelo de Linguagem (*Large Language Model*)
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo

Lista de símbolos

α	Taxa de aprendizado
θ	Parâmetros do modelo
$\Delta x, \Delta y$	Coordenadas relativas da bounding box
$\Delta w, \Delta h$	Largura e altura relativas da bounding box
c	Confiabilidade de que a grid cell contém um objeto
p_c	Probabilidade de classe
λ	Peso na função de perda
$\mathbb{1}$	Função indicadora

Lista de figuras

Lista de tabelas

Sumário

Introdução

Nos dias atuais, um problema latente para a agricultura mundial é a perda da produção devido a problemas com pragas. De acordo com a Organização das Nações Unidas para a Alimentação e a Agricultura (FAO), todo ano, entre 20 e 40% da produção agrícola é perdida em função desses problemas (IPPC Secretariat, 2021). Levando tal fato em consideração, é evidente a importância de se diagnosticar doenças em plantas de maneiras que sejam tanto eficientes quanto baratas, visando a minimização dos danos oriundos das pragas na agricultura mundial.

A detecção e diagnóstico de doenças em plantas pode ser feita por inspeção visual, avaliação microscópica de características morfológicas, ou ainda técnicas moleculares, sorológicas ou microbiológicas (Mahlein, 2016). Todas essas formas de análise são baseadas em tecnologias ou conhecimentos específicos, sendo portanto altamente dependentes de pessoal qualificado e especialista. Esse processo, além de ser feito em escala reduzida, é custoso e complexo, o que demonstra a necessidade de inovações tecnológicas que auxiliem nesta tarefa e a tornem mais simples e acessível para aqueles que desenvolvem atividades agrícolas.

Paralelamente à essa necessidade, podemos observar avanços tecnológicos substanciais que vêm sendo feitos no desenvolvimento de tarefas complexas como a criação de agentes conversacionais (AI4Science e Quantum, 2023; Minaee *et al.*, 2024) e direção autônoma (Badue *et al.*, 2019), por exemplo.

Assim, é considerável que o acúmulo científico que a humanidade alcançou nos últimos anos (evidenciado pelos avanços tecnológicos comentados) possibilite a construção de inovações tecnológicas que auxiliem os produtores rurais contra esse problema latente.

Na realidade, essa consideração se mostra realista. O artigo de Mahlein, 2016 discute o uso de técnicas de imageamento em agricultura, destacando a questão de detecção de doenças em plantas. O trabalho lista sensores convencionais que geram imagem RGB, câmeras multi ou hiper-espectrais, sensores termais, sensores de fluorescência, e outros sensores capazes de medir biomassa e outras características estruturais das plantas. Dentre essas tecnologias, câmeras convencionais que geram imagens RGB estão amplamente disponíveis, são portáteis e de fácil manuseio, principalmente em *smartphones*. Assim, o processamento computacional dessas imagens poderia ajudar na identificação precoce dessas doenças e, por conseguinte, diminuir as perdas envolvidas.

A percepção que técnicas de visão computacional podem ajudar nesse problema não é

algo novo. No cenário nacional, recentemente a Embrapa[1] publicou uma matéria intitulada "Inteligência artificial identifica plantas doentes simulando processo cerebral" na qual é discutida uma parceria estabelecida por ela com empresas privadas para desenvolver inovações na detecção automática de doenças em plantas. Além disso, embora aplicativos de celular voltados para a identificação de doenças em plantas estejam se tornando mais comuns, ainda enfrentam críticas devido à sua baixa precisão e limitações funcionais (Siddiqua *et al.*, 2022)

Motivação

A partir de uma perspectiva acadêmica, os recentes avanços na área de Visão Computacional, devido ao contínuo melhoramento das técnicas de *deep learning* (Goodfellow *et al.*, 2016), tornaram possíveis trabalhos sobre detecção de doenças em imagens de plantas que fazem uso de modelos de redes neurais profundas. Entre esses trabalhos, a maioria considera imagens de folhas de plantas obtidas utilizando-se câmeras convencionais e trata do problema de classificação de doenças (V. Singh e Misra, 2017; Mohanty *et al.*, 2016; Geetharamani e Arun Pandian, 2019; Thakur *et al.*, 2022).

Essas técnicas de *deep learning* são baseadas em redes neurais (Nielsen, 2015; Goodfellow *et al.*, 2016) que possuem a capacidade de processar dados em seu formato bruto e extrair automaticamente as representações (características) que são mais eficazes para a inferência final esperada. Essa capacidade, resultante do processo de treinamento da rede neural, faz com que os métodos sejam facilmente adaptados para imagens com diferentes características. Assim, o desenvolvimento de soluções para novos cenários de aplicação — como culturas diferentes e imagens com características diferentes — é bastante agilizado, o que reforça o potencial dessa abordagem.

O conjunto de dados público mais utilizado para o treinamento e avaliação dos modelos de visão computacional para detecção de doenças em plantas (e dos trabalhos supracitados) é o *PlantVillage dataset* (Hughes e Salathé, 2015; Yao *et al.*, 2023). Exemplos de imagens desse *dataset* são mostrados na figura 1.

Contudo, infelizmente uma série de questões relacionadas à qualidade dos dados do *PlantVillage* faz com que os ótimos resultados demonstrados nos trabalhos acadêmicos em questão não se traduzam para o contexto do cotidiano dos agricultores. Seja pela natureza das imagens, compostas de folhas de plantas em um plano de fundo de cor sólida e sem grandes variações na iluminação (o que não é reproduzido em fotos tiradas no campo) (Yao *et al.*, 2023; Hughes e Salathé, 2015; V. Singh e Misra, 2017; Coletta *et al.*, 2022) ou por problemas referentes a condições de captura (balanço de branco, exposição, foco) e padrões observados nos planos de fundo das imagens que se constituem como atributos que têm correlações espúrias com o rótulo das imagens e, por isso, acabam por facilitar muito a tarefa do modelo classificador desenvolvido (Noyan, 2022).

Como comentado, a consequência desses vieses é que, apesar de artigos corriqueiramente publicados obterem resultados muito impressionantes na tarefa de classificação de doenças de plantas, esses resultados não se generalizam para situações reais e, portanto,

[1] https://www.embrapa.br/

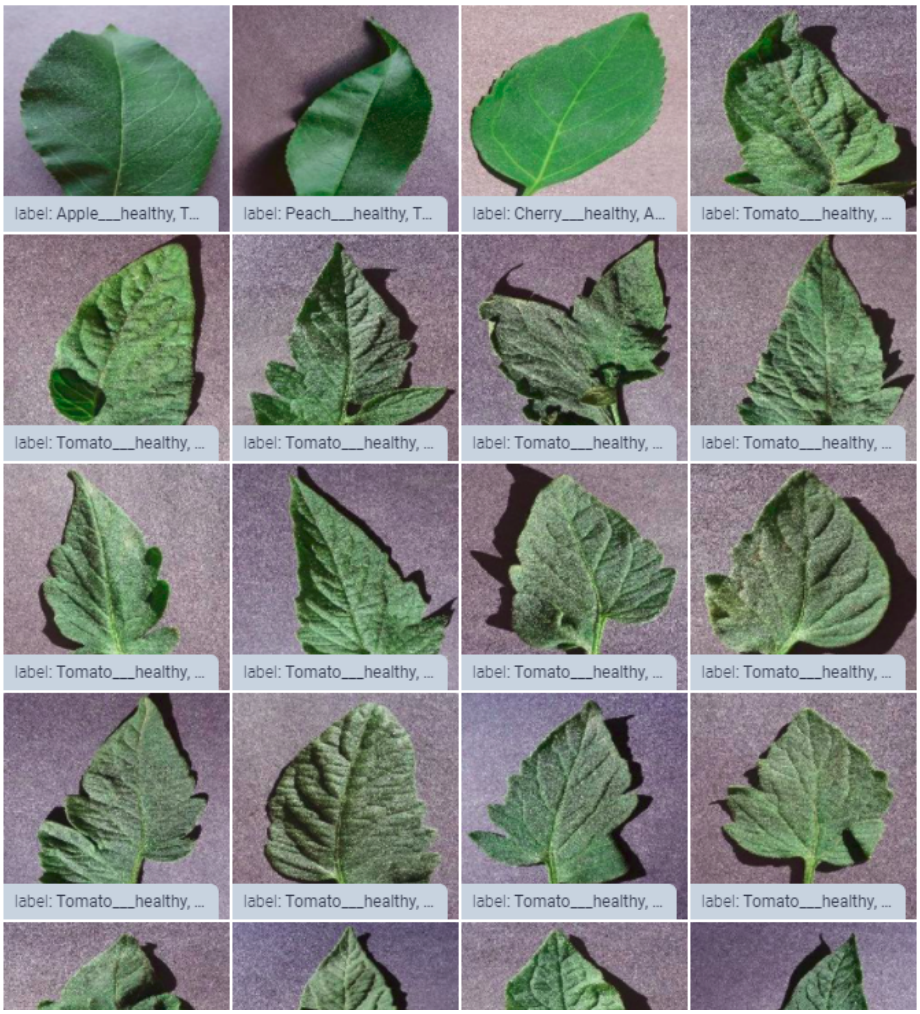


Figura 1: *Exemplos de imagens de folhas no dataset PlantVillage (extraído de https://paperswithcode.com/dataset/plantvillage).*

nos dão uma falsa ideia de que o problema de classificação de doenças de plantas utilizando visão computacional já está resolvido. Isso acaba por ter o resultado oposto do esperado, atrasando o desenvolvimento tecnológico e científico.

Dessa forma, fica evidente que é necessário investir em alternativas para o desenvolvimento de conjunto de dados abertos, robustos e que tenham passado por curadoria. Somente assim avançaremos no desenvolvimento de modelos de visão computacional aplicados à detecção de doenças em plantas, com resultados que realmente farão a diferença no dia a dia dos agricultores que se beneficiariam dessas ferramentas abertas e acessíveis. Essa é a principal motivação desse trabalho de formatura supervisionado.

Objetivo

O objetivo deste trabalho é desenvolver um sistema que possibilite a produção de melhores conjuntos de dados de visão computacional no domínio da agricultura. Para isso, a ideia proposta é a implementação de um sistema em que os usuários possam enviar fotos

de, por exemplo, folhas de plantas doentes, para obter o diagnóstico da doença pelo sistema. As imagens enviadas pelos usuários serão acumuladas em um banco de dados de imagens e posteriormente revisadas por usuários especialistas, que poderão corrigir os diagnósticos feitos pelo sistema. Depois disso, as imagens revisadas passam a fazer parte de um novo conjunto de dados, que será incrementado com o tempo com as imagens enviadas para a plataforma. Dessa forma, consideramos que será possível produzir conjuntos de dados de boa qualidade e volume de maneira colaborativa.

Capítulo 1

Descrição do problema e do sistema proposto

Como comentado, o presente trabalho é motivado pela carência de conjuntos de dados abertos e robustos de imagens de doenças em folhas de plantas, que possibilitem o desenvolvimento de modelos de visão computacional com aplicabilidade prática no cotidiano dos produtores rurais.

1.1 Considerações

Para que um conjunto de dados com essa potencialidade seja desenvolvido, é preciso levar em conta algumas considerações importantes. Dentre elas, podemos destacar as seguintes:

1. **Volume considerável de dados:**

   No contexto de *deep learning*, modelos de visão computacional já se provaram muito efetivos. Contudo, é consenso que grandes quantidades de dados são necessárias para evitar *overfitting*.

   A ideia geral de sistemas supervisionados de *machine learning*, é que, utilizando de algoritmos de otimização de uma função de perda — que quantifica a performance do modelo atual — os modelos em treinamento aprendam a reproduzir a função representada pelo conjunto de dados, em que os dados de entrada são as variáveis independentes (no caso, imagens) e os rótulos são as variáveis dependentes (Abu-Mostafa *et al.*, 2012).

   Nesse caso, quando não temos uma grande quantidade de dados, os modelos de aprendizado profundo (*deep learning*) têm a capacidade de aprender uma função que reproduz perfeitamente o conjunto de treinamento, é como se o modelo "decorasse" os dados ao invés de relacionar características importantes neles para chegar a respostas pertinentes (Abu-Mostafa *et al.*, 2012; Shorten e Khoshgoftaar, 2019).

   Por isso, esses sistemas necessitam de grandes quantidades de dados para funciona-

rem de maneira satisfatória (Shorten e Khoshgoftaar, 2019). Então, para construir um conjunto de dados de qualidade para detecção de doenças em folhas de plantas, é importante que ele tenha um volume considerável.

2. **Os dados devem contemplar a variabilidade dos casos de uso real da tecnologia:** Para que modelos de *machine learning* sejam capazes de generalizar bem, é necessário que, o conjunto de dados com os quais são treinados representem bem a diversidade dos dados que estarão presentes no cotidiano de uso da tecnologia resultante. Isso ocorre porque, quando os dados presentes nos casos de uso real da ferramenta forem muito diferentes dos dados de treinamento, o modelo simplesmente não terá tido contato com dados de tal natureza e, portanto performará pobremente.

   Um exemplo claro disso é o tipo de imagens presentes no *dataset PlantVillage*. Essas imagens, com aspecto de "laboratório" (folhas retiradas das plantas e dispostas em superfícies planas e de cor sólida) certamente são diferentes das imagens disponíveis no contexto do campo: características como planos de fundo com alta variabilidade (plantas, paisagens, etc.) e folhas ainda presas nos galhos são exemplos disso.

3. **Os dados devem passar por uma curadoria especializada:**

   No domínio de detecção de doenças em plantas, uma avaliação de especialistas sobre os dados, para produzir rótulos corretos é de suma importância. Isso ocorre pois muitas vezes, não é fácil diferenciar as doenças entre si, devido à elevada similaridade entre sintomas e pelo simples fato dessa identificação não ser conhecimento "comum".

4. **Não deve haver duplicatas ou "quase-duplicatas" nos dados:**

   A presença de dados duplicados, ou "quase-duplicados" em conjuntos de dados é maléfica para o treinamento de modelos de *machine learning*. Isso ocorre pois durante a etapa de treinamento, o modelo pode decorar padrões presentes nas imagens duplicadas e isso implica no comprometimento da performance dos modelos ao se deparar com dados inéditos (Barz e Denzler, 2019).

   Além disso, a inclusão desses dados "repetidos" pode levar a uma similaridade elevada entre os conjuntos de treino e teste do *dataset*. Nesse caso, como a avaliação do modelo é realizada com base no conjunto de teste, a presença de dados muito semelhantes pode artificialmente inflar as métricas de desempenho, resultando em uma avaliação irreal das capacidades do modelo e na pior capacidade de generalização no momento de implantação do modelo em campo (Barz e Denzler, 2019).

1.2 Solução proposta

Com o levantamento dessas considerações, formulamos uma abordagem de construção colaborativa de conjuntos de dados para detecção de doenças de plantas que se propõe a ser uma possível solução para a escassez de *datasets* robustos, volumosos, confiáveis e abertos nesse domínio.

1.2.1 Redução do escopo

Para o desenvolvimento dessa solução, primeiramente reduzimos o escopo das tarefas propostas por grandes *datasets* desse domínio (Yao *et al.*, 2023). Tanto o *PlantDoc* (um *dataset* alternativo ao *PlantVillage*, que oferece imagens sem a característica de imagens de "laboratório") quanto o próprio *PlantVillage*, por exemplo, têm em suas classes o seguinte formato:

```
(<espécie> : <condição>)
```

Ou seja, além de identificar a doença que afeta a planta, o modelo também tem a tarefa de identificar a espécie da planta em questão. Na realidade, para o caso de uso abordado nesse trabalho (detecção de doenças em plantas) é esperado que o agricultor saiba, no mínimo, a espécie da planta que está cultivando. Desse modo, decidimos simplificar a tarefa para a **detecção de doenças nas folhas de tomate**.

A escolha dessa cultura na redução do escopo do trabalho se deu pelo fato da hortaliça ser uma das mais consumidas no mundo e da tomaticultura ser de imensa importância no setor da agricultura brasileira, sendo uma das principais fontes de emprego e renda da produção agrícola e tendo um valor de produção bruto superior a 12,4 bilhões de reais em 2022, de acordo com dados do IBGE e matéria publicada no portal Revista Rural.[1]

1.3 *TomatoHealth*

Desse modo, a ideia da solução proposta é implementar uma plataforma chamada *TomatoHealth* que oferece a funcionalidade de detecção de doenças em folhas de tomate por meio de uma aplicação *web*. Por meio dessa aplicação, os usuários podem submeter imagens das folhas de suas plantas e a aplicação retorna uma resposta sobre as doenças detectadas.

Para que isso ofereça uma possível solução para a escassez de conjuntos de dados robustos na tarefa de detecção de doenças em plantas, as imagens enviadas pelos usuários passam a fazer parte de um banco de dados.

Então, a partir desse repositório de imagens enviadas pelos usuários, uma outra frente de atuação da plataforma permite que, para cada imagem armazenada no banco de dados, usuários privilegiados (especialistas do domínio) façam revisões da inferência efetuada pelo modelo de detecção de doenças.

Depois dessa revisão, as imagens revisadas podem passar a integrar o conjunto de dados usado para a tarefa de detecção de doenças em folhas de tomate, fechando um ciclo de atualização contínua do conjunto de dados e melhorando, iterativamente, a robustez do conjunto de dados original.

É importante destacar que a solução implementada por essa primeira versão do *TomatoHealth* aborda de maneira efetiva as seguintes considerações feitas sobre o problema tratado nesse trabalho de formatura supervisionado:

[1] https://www.revistarural.com.br/2022/10/14/tecnologias-impulsionam-producao-nacional-do-tomate/

1. **Volume considerável de dados:** pois iterativamente, o volume do conjunto de dados é aumentado com imagens que passaram pelo crivo de especialistas;
2. **Os dados contemplam a variabilidade dos casos de uso real da tecnologia:** pois as imagens que têm a possibilidade de passar a integrar o conjunto de dados são tiradas justamente do contexto de aplicação real da tecnologia;
3. **Os dados passam por uma curadoria especializada:** pois a presença da figura dos especialistas atua na revisão e curadoria de todas as imagens que passam a fazer parte do conjunto de dados;

Note que a consideração

4. **Não deve haver duplicatas ou "quase-duplicatas" nos dados**

não foi abordada por essa primeira versão de implementação do *TomatoHealth*. Contudo, já planejamos a integração de algoritmos que façam busca por imagens semelhantes no conjunto de dados existente e as exiba para o especialista que está fazendo a revisão.

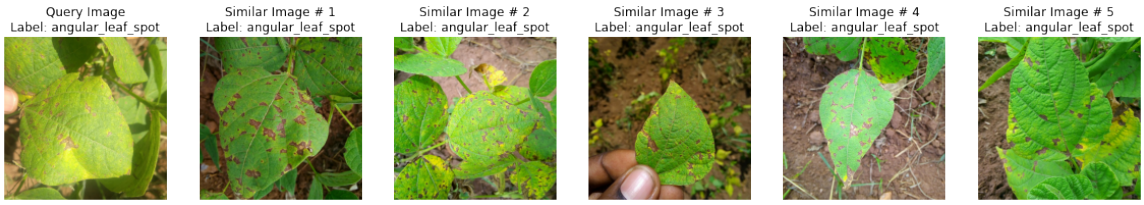


Figura 1.1: *Exemplos de resultados do algoritmo de busca por imagens com base em similaridade. (extraído de https://huggingface.co/blog/image-similarity).*

Caso a imagem que está sendo revisada apresente um grau de semelhança elevada com outras imagens que já fazem parte do conjunto de dados, o especialista terá a opção de descartar a amostra em questão. Essa funcionalidade, que fará com que a solução proposta contemple todas as considerações feitas no trabalho será melhor descrita no capítulo 4.

1.4 Detalhes "abstratos" sobre o sistema

Nesse capítulo, iremos descrever as capacidades esperadas do sistema desenvolvido sem detalhes de implementação.

1.4.1 *Storyboard* do sistema

Uma ótima forma de começar o desenvolvimento de um sistema é demonstrar seus casos de uso e aplicações no mundo real por meio de um *Storyboard*. Uma técnica muito empregada por estúdios de animação, pois produzir desenhos para cada quadro de uma animação é uma tarefa demorada, e desenhar um quadro que não será usado é um desperdício (similar ao desenvolvimento de código). Além de ser rápido de esboçar, o *Storyboard* permite demonstrar informações sobre um sistema que seria dificilmente explicado por texto, e também transmite informações essenciais de um sistema: usuários, contexto, sequência de tarefas de seu sistema e como seu sistema traz satisfação (Klemmer, 2016). Sendo assim, a Figura 1.2 demonstra o *Storyboard* feito para o *TomatoHealth*.

Na Figura 1.2, enumeramos as ações dos quadrinhos:

1. Um tomateiro doente
2. Um fazendeiro encontra seu tomateiro doente
3. Ele tira uma foto da sua folha
4. Ele fica alegre em descobrir a doença e como cuidar de seu tomateiro, com a ajuda de uma Inteligência Artificial
5. A imagem tirada pelo fazendeiro é salva num banco de dados abertos
6. Um especialista analisa as imagens salvas
7. As imagens analisadas por um especialista são usadas para treinar a Inteligência Artificial

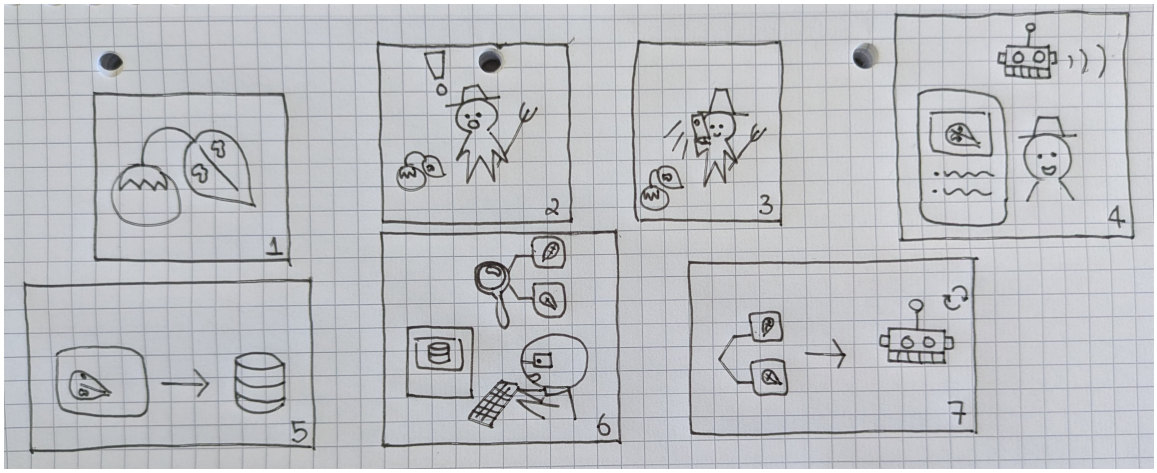


Figura 1.2: *Storyboard do desenvolvimento do sistema*

1.4.2 Descrição geral sobre o sistema

Já tendo uma breve noção sobre a atuação do *TomatoHealth* por nosso *Storyboard*, também é importante descrever o sistema de maneira geral, para comentar sua pertinência aos assuntos tratados nesse trabalho, e destacar como ele se relaciona ao problema.

Os próximos capítulos separam os componentes do sistema do *TomatoHealth*, para que o capítulo 3 os detalhe de maneira técnica.

Interface do Usuário Geral

Nossa interface *web* deve permitir ao usuário geral:

- entender o papel da ferramenta
- tirar ou escolher uma foto de folha de tomate
- enviar essa foto para análise
- receber um *feedback* sobre a saúde de seu tomateiro

Uma parte importante dessa interface é estimular o usuário a tirar uma foto de sua folha a partir de seu celular, para que essas imagens quando armazenadas e classificadas sirvam de dados em casos de usos reais em nosso *dataset*.

Com esse objetivo em mente, priorizamos o desenvolvimento *mobile-first*, e permitimos o usuário a acessar a câmera de seu celular, assim como o instruímos sobre como tirar uma foto adequada, tanto para um *feedback* quanto um dado melhor em nosso *dataset*.

Interface do Usuário Especialista

Na interface do usuário especialista, ele deve ser capaz de:

- autenticar com seu e-mail e sua senha
- ver as imagens tiradas por usuários comuns
- demarcar na imagem as diferentes classes que uma folha de tomate pode ter

Com isso, as imagens ficam com suas classes demarcadas por um quadrado, e inseridas em nosso banco de dados, para retreinamento do nosso modelo.

Back-end

Nosso *back-end* deve comportar:

- a criação de usuários especialistas
- um banco de dados com as imagens enviadas por usuários comuns e as imagens a serem usadas no modelo de detecção
- o treinamento do modelo
- um servidor que recebe uma imagem, realiza a detecção, e faz uma chamada para um *Large-Scale Transformer-based Language Model* recomendar cuidados a serem feitos com o tomateiro de acordo com as classes detectadas
- a inserção de imagens enviadas por usuários comuns e de imagens anotadas por especialistas no banco de dados

Dessa forma, o sistema faz o treinamento do modelo de maneira periódica com dados revisados por especialistas, e retorna um *feedback* com as detecções do modelo a um usuário comum.

1.5 Sistemas similares – detecção de doenças e sistemas colaborativos

Ao pesquisar por *"plant diagnosis"* na *Google Play*,[2] encontramos dezenas de aplicativos capazes de identificar culturas e/ou doenças por meio de fotos. Um dos mais populares,

[2] https://play.google.com/store/apps/

com mais de 10 milhões de downloads, é o *Plantix*.[3]

Ademais, na pesquisa por sistemas no âmbito de ciência colaborativa, encontramos duas referências semelhantes quanto a nossa proposta de sistema por compartilharem fotos fornecidas por usuários para a utilização em pesquisa, são elas: *iNaturalist*[4] e *Zooniverse*.[5]

1.5.1 *Plantix*

Segundo sua página, o aplicativo tem o mesmo propósito de nosso *TomatoHealth*, mas para diversas culturas. Entretanto, além do sistema identificar doenças e propor tratamentos, ele fornece ao usuário o contato com especialistas para sanar dúvidas, e também conta com uma biblioteca sobre doenças em plantas.

1.5.2 *iNaturalist*

O *iNaturalist* é uma comunidade onde usuários observam organismos vivos com suas câmeras, obtêm detalhes, e compartilham suas fotos com outros entusiastas. A plataforma também faz com que as fotos sejam compartilhadas com repositórios de dados abertos como o *Global Biodiversity Information Facility* (GBIF), e conecta usuários comuns com especialistas, capazes de identificar diferentes espécies.

1.5.3 *Zooniverse*

Já o *Zooniverse* é constituído por projetos em diferentes áreas da ciência: Artes, Biologia, Clima, História, etc., em quais voluntários classificam dados disponibilizados por um projeto e se conectam com especialistas. O propósito da ferramenta é descrito como "pesquisa com base em pessoas", de forma a destacar o papel do voluntário na análise dos dados e no fomento de pesquisas. No dia de hoje (2 de novembro de 2024), o site conta com 2.783.522 voluntários e 846.112.376 classificações.

[3] https://plantix.net/pt/

[4] https://www.inaturalist.org/

[5] https://www.zooniverse.org/

Capítulo 2

Aplicação de *Machine Learning* no projeto

No projeto *TomatoHealth*, que possui a funcionalidade de detecção de doenças em imagens de folhas de tomate, as técnicas de *Machine Learning* desempenham um papel fundamental. No campo da visão computacional aplicada à agricultura, é comum o uso de modelos de Inteligência Artificial para tarefas como classificação de imagens ou detecção de objetos.

Esses métodos tornam-se indispensáveis em situações onde a complexidade do problema torna impraticável ou mesmo impossível desenvolver um programa de computador tradicional baseado em regras fixas capaz de oferecer uma boa solução. Como por exemplo, determinar a doença presente em uma folha a partir de sua imagem. Em casos como esse, recorremos a técnicas de aprendizado de máquina, que oferecem a capacidade de aprender padrões complexos a partir de dados.

No contexto de aprendizado de máquina supervisionado, é necessário dispor de um conjunto de dados rotulados. Esses dados são usados para treinar o modelo de IA, permitindo que ele aprenda a identificar os padrões que caracterizam cada rótulo. Uma vez treinado, o modelo é capaz de generalizar esse conhecimento para realizar predições precisas em novos dados de entrada, predizendo seus rótulos.

2.1 Classificação vs Detecção

Para que o processo de treinamento do modelo de visão computacional seja possível, primeiro é necessário escolher a arquitetura de rede neural que será utilizada. Contudo, ainda antes disso, é necessário definir qual é a tarefa que devemos realizar e um *dataset* já existente para efetuar o treinamento desse primeiro modelo que será integrado à plataforma.

Até agora, a expressão: “detecção de doenças em imagens de folhas de plantas”; que resume qual é o objetivo do modelo de visão computacional incorporado ao *TomatoHealth* ainda não foi muito bem definida.

É importante esclarecer isso pois, nesse domínio, não há um consenso entre os *datasets* publicamente disponíveis. Isso ocorre pois a tarefa proposta pelo *dataset PlantVillage*, e, portanto, a tarefa mais popular nesse domínio, é a de **classificação**. Isto é, o modelo treinado recebe uma imagem como entrada e então classifica essa imagem como tendo um dentre os diversos rótulos possíveis.

Já o *dataset PlantDoc*, propõe uma tarefa de **detecção de objetos**. Como já comentado, esse *dataset* surgiu para ser uma alternativa ao *PlantVillage*. Podemos comentar sobre a mudança das caracteríticas das imagens, que no *PlantDoc* são oriundas do campo e não obtidas em um "ambiente controlado", mas uma das melhorias conceituais mais importantes que foi proposta é justamente a mudança da tarefa de classificação para a de detecção de objetos.

Ao pensar no domínio da agricultura e nos casos de uso dessa tecnologia de visão computacional para identificar doenças em folhas de plantas, não é difícil perceber que a tarefa de detecção de objetos é a mais adequada.

Para que a tarefa de classificação de imagens seja coerente nesse contexto, é necessário que somente uma folha esteja presente na imagem. Para isso, um agricultor no campo teria de retirar a folha da planta, colocá-la sobre uma supefície plana e então efetuar a captura da imagem.

Já no caso da tarefa de detecção de objetos, bastaria identificar a planta de interesse e efetuar a captura da imagem de modo que as folhas afetadas pela doença estejam presentes na imagem.

Ao compararmos essas duas abordagens diferentes, a etapa adicional inerente à tarefa de classificação (de ser necessário retirar uma folha da planta e só então efetuar a captura da imagem) só seria justificada se isso significasse em uma performance muito superior dos modelos de visão computacional baseadas no conceito de classificação, o que não é o caso (D. Singh *et al.*, 2020).

Desse modo, é mais interessante trabalharmos com a tarefa de detecção, em que, dado uma imagem de planta com diversas folhas, as folhas presentes na imagem serão detectadas e então classificadas de acordo com as diversas doenças possíveis.

Considerando o fato do *PlantDoc* ter sido construído após a constatação de que haviam problemas profundos com o *PlantVillage* e ter proposto essas melhorias para tentar se consolidar como um novo padrão de *dataset* público para essa tarefa, decidimos escolhê-lo para ser o *dataset* de base para o modelo de visão computacional do *TomatoHealth*.

Para performar essa tarefa de detecção de objetos, escolhemos a família de modelos *YOLO*, que explicaremos em detalhes no capítulo 2.3

2.2 Entendendo detecção de objetos em *Machine Learning*

A detecção de objetos tem o objetivo de resolver uma das questões fundamentais de aplicações de visão computacional. Identificar o que são os objetos e onde eles estão locali-

zados nas imagens. Isso é feito por meio da detecção de instâncias de objetos pertencentes a classes específicas nas imagens em questão, com o uso de *bounding boxes* — retângulos que envolvem os objetos detectados e indicam sua classe (Zou *et al.*, 2019).

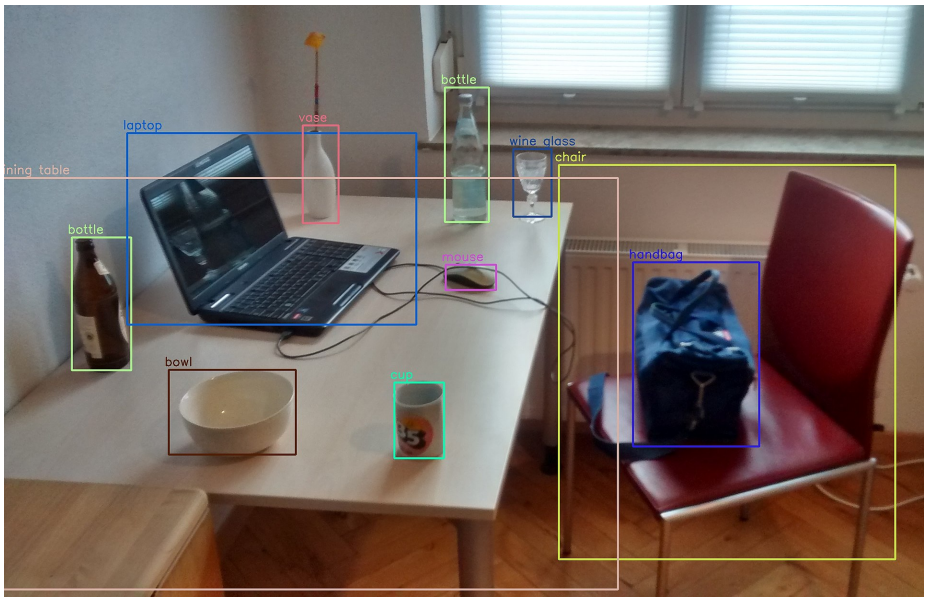


Figura 2.1: *Exemplo de detecção de objetos em uma imagem utilizando bounding boxes. Imagem extraída de MTheiler, trabalho próprio, licenciada sob CC BY-SA 4.0. Disponível em: https://commons.wikimedia.org/w/index.php?curid=75843378.*

A tarefa de detectar objetos em imagens não é nova, e por ter muitas aplicações como detecção de faces e corpos humanos e servir como base para outras tarefas como direção autônoma, *image captioning* — que corresponde à tarefa de descrever imagens com textos de linguagem natural —, aplicações em sistemas com câmeras de segurança, etc. Há uma evolução rápida nesse campo da ciência desde os anos 1990 (Zou *et al.*, 2019; Zhao *et al.*, 2018). Podemos observar essa evolução ao considerarmos o crescimento exponencial no número de publicações que vêm ocorrendo nessa área de conhecimento:

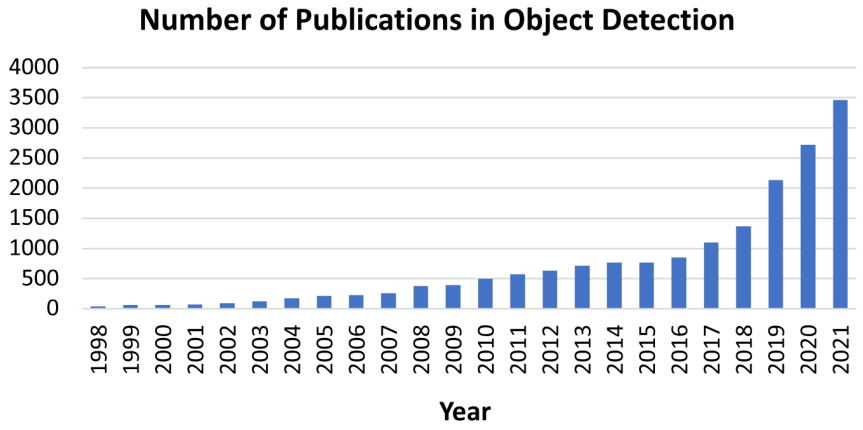


Figura 2.2: *Crescimento do número de publicações relacionadas à detecção de objetos ao longo dos anos. (extraído de (Zou et al., 2019)).*

O motivo da aceleração significativa na produção de conhecimento científico nessa área do conhecimento foi uma mudança de paradigma considerável: a introdução de técnicas de *deep learning* nessa tarefa de visão computacional. Dessa forma, podemos afirmar que a história da detecção de objetos pode ser compreendida a partir de dois períodos históricos que iremos abordar a seguir: detecção de objetos com uma abordagem tradicional (anterior a 2014) e detecção de objetos baseada em *deep learning* (depois de 2014) (Zou *et al.*, 2019).

2.2.1 Detecção de objetos com uma abordagem tradicional

Antes do advento das técnicas de *deep learning*, os métodos para detecção de objetos em imagens dependiam da extração manual de representações (características) eficazes para a inferência esperada. Os algoritmos mais eminentes dessa época utilizavam janelas deslizantes na imagem com filtros compostos pelas características extraídas manualmente.

Por exemplo, no algoritmo de Viola-Jones, a imagem é percorrida por janelas deslizantes de diversas escalas contendo características presentes em rostos humanos (Haar *features*). Dessa forma, a depender da correspondência das características com a imagem, uma face poderia ser acusada como detectada ou não naquela localidade, isso é feito com classificadores em cascata (Viola e Jones, 2001).

Por outro lado, o método *HOG (Histogram of Oriented Gradients)* é um método em que as caraterísticas da imagem são obtidas por meio da computação de gradientes em pequenas porções da imagem. Para cada pequena localidade, o histograma resultante era concatenado aos demais, formando um mapa de características representativo de bordas e texturas. Com a aplicação desse mapa de características em classificadores como *SVMs* (*Support Vector Machines* (Cortes e Vapnik, 1995)), é possível detectar objetos e essa combinação foi inicialmente implementada para detecção de faces (Dalal e Triggs, 2005).

De maneira geral, perceba que podemos resumir as abordagens tradicionais de detecção de objetos como a extração manual de características das imagens (Haar *features* ou mapas de gradientes oriundos de *HOG*) e então essas características passavam por classificadores simples (Viola e Jones, 2001; Dalal e Triggs, 2005).

Contudo, essas abordagens eram limitadas e os novos métodos com *deep learning* superaram os desempenhos desses algoritmos mais tradicionais (Zou *et al.*, 2019).

2.2.2 Detecção de objetos baseada em *deep learning*

Depois de um tempo, a performance de detectores de objetos com representações extraídas manualmente chegou a ficar saturada. Então, com o advento das técnicas de *deep learning*, especialmente das redes neurais convolucionais (*CNNs*), passou a ser possível processar imagens em seu formato bruto e obter automaticamente representações de alto nível de suas características. Isso impulsionou um avanço significativo na precisão e eficiência dos métodos de detecção de objetos, marcando um salto evolutivo nessa tarefa de visão computacional (Zou *et al.*, 2019).

Fundamentos de redes neurais convolucionais

Redes neurais podem ser entendidas como funções não lineares extremamente complexas. Esse artifício matemático é constituído por nós — neurônios — organizados em camadas. Cada nó efetua uma combinação linear dos valores oriundos dos nós da camada anterior e então passa o resultado dessa operação por uma função não linear — função de ativação.

Podemos pensar que a relação entre os dados de entrada de um *dataset* e cada um dos seus rótulos é regida por uma função. Por meio do método do gradiente descendente e da técnica de *backpropagation*, conseguimos fazer com que os parâmetros da rede neural (pesos das combinações lineares) se adaptem de modo a reproduzir essa função.

As redes neurais convolucionais são uma evolução das redes neurais convencionais. Elas são especialmente boas lidando com imagens. São constituídas por três tipos de camadas diferentes:

Camadas convolucionais aplicam a operação de convolução nos dados que chegam à essa camada. Nesse caso, os pesos do *kernel* da convolução são aprendidos como parâmetros da rede neural.

Camadas de *pooling* são camadas utilizadas para redimensionar os dados de entrada. Tanto aqui quanto em relação às camadas convolucionais, podemos pensar que os dados de entrada são imagens. Geralmente, as camadas de *pooling* não apresentam parâmetros aprendíveis pela rede (treináveis). A técnica mais utilizada é a denominada *max-pooling*, em que a camada visa preservar o sinal privilegiado pela camada convolucional anterior.

Camadas *fully connected* são camadas semelhantes às presentes nas redes neurais convencionais. Geralmente são utilizadas para relacionar — de maneira treinável — os resultados das camadas abordadas anteriormente. Constituem camadas que utilizam as características extraídas pelas camadas convolucionais e de *pooling*, para efetuar a classificação. Isso, no caso de um problema de classificação, evidentemente.

Principais modelos para detecção de objetos com *deep learning*

Com o avanço do *deep learning* e das redes convolucionais, esses novos conceitos foram introduzidos ao contexto de detecção de objetos com a introdução de *Regions with CNN features* (*RCNNs*) (Zou *et al.*, 2019; Girshick, Donahue *et al.*, 2014; Girshick, Donahue *et al.*, 2016).

A ideia por trás do conceito de *RCNN* é bem direta. Primeiro um conjunto de regiões na imagem é obtido por meio de algoritmos de busca. Depois disso, as representações de cada região é obtida ao passá-las (adaptadas para um tamanho fixo) por uma rede convolucional. Por fim, classificadores como *SVMs* são utilizados para categorizar essas regiões em diferentes classes (Zou *et al.*, 2019).

Ao longo do tempo, esforços foram sendo feitos para aumentar a velocidade da detecção de objetos com *CNNs*. A abordagem de He *et al.*, 2014 utiliza o conceito de *spatial pyramidal pooling*, com esse conceito, as regiões de interesse (*RoI*) são obtidas diretamente no mapa de *features*, o que aumenta muito a velocidade de todo o processo.

O *spatial pyramidal pooling* funciona ao subdividir a *RoI* em "grades de células" e aplicar a operação de *max pooling* em cada célula. Desse modo, ao final do *spatial pyramidal pooling*, teremos um vetor de representações de tamanho fixo, independente do tamanho da *RoI* (He *et al.*, 2014). Depois disso, esses vetores passam por algoritmos de classificação como *Support Vector Machines (SVMs)*. A possibilidade de uso desse vetor de tamanho fixo foi uma das principais contribuições da *SPPNet* (Zou *et al.*, 2019).

Depois disso, Girshick, 2015 introduziu a *Fast R-CNN*, que possibilitou o treinamento de uma única rede neural capaz de realizar simultaneamente a regressão das *bounding boxes* (isto é, um ajuste fino das *RoIs* inicialmente propostas por algoritmos de busca) e das classificações. Isso representou um avanço em relação à *SPPnet*, que processava a obtenção das *RoIs* e a classificação das regiões de interesse em etapas distintas, o que tornava o processo mais lento e menos eficiente.

É notável que, a princípio, os métodos de detecção baseados em *CNNs* eram baseados em duas etapas: a proposta das regiões de interesse e, depois, a classificação dessas regiões. A partir de 2015, a tarefa de detecção de objetos com *CNNs* passou a ser feita em somente uma etapa.

A arquitetura *YOLO (You Only Look Once)* é uma grande representante dessa nova abordagem, permitindo obter as coordenadas das *bounding boxes* e suas classificações em uma única passagem pela rede neural, viabilizando a detecção de objetos em tempo real. Pioneira na detecção de objetos em uma etapa, a arquitetura *YOLO* ganhou popularidade por seus bons resultados em acurácia e velocidade (Zou *et al.*, 2019), sendo, por isso, a escolhida para a tarefa de detecção de objetos no *TomatoHealth*.

2.3 *YOLO* (*You Only Look Once*)

2.3.1 Contextualização

A arquitetura de rede neural para detecção de objetos *YOLO* foi introduzida em 2016 com o trabalho de Redmon *et al.*, 2016. Desde então, diversas evoluções e melhorias foram incorporadas ao modelo. Ao longo dos anos, novas versões dessa arquitetura foram lançadas, indo do *YOLOv1* ao *YOLOv8* em 2023 e até mesmo modelos mais atuais como o *YOLO11*, lançado em 2024 (Redmon *et al.*, 2016; Ultralytics, 2023b; Ultralytics, 2024). Por haver mais informação disponível sobre o modelo *YOLOv8* e pelo fato de ainda ser um modelo novo e com performances muito satisfatórias, essa foi a arquitetura escolhida para o projeto. Além disso, apesar dessa evolução entre versões, os conceitos fundamentais e as ideias por trás do *YOLO* ainda podem ser utilizadas para entender seu funcionamento.

Na aplicação *TomatoHealth*, utilizamos o modelo *YOLOv8*. No entanto, o propósito deste texto não é aprofundar em detalhes específicos ou otimizações exclusivas desta versão, mas sim oferecer uma visão geral de como o *framework YOLO* funciona. A seguir, apresentaremos os conceitos primeiro introduzidos em 2016 com o *YOLOv1* e então discutiremos os avanços e melhorias que foram incorporados à arquitetura do *YOLOv8*.

2.3.2 Conceitos fundamentais do *YOLO*

Como já comentado, a principal ideia proposta pela arquitetura *YOLO* é efetuar a detecção de objetos em uma só etapa. Dessa maneira, uma única rede neural faz a predição das *bounding boxes* e das classificações. Com somente um *forward pass*.

Passo a passo do *YOLOv1*: No *YOLOv1*, isso é feito por meio do artifício de enxergar o problema de detecção de objetos como puramente um problema de regressão. Isso significa que o problema é formulado de tal modo que os valores preditos pelo modelo são contínuos. Mesmo que a classificação esteja envolvida na tarefa de detecção de objetos, o *YOLOv1* não trabalha com a predição de valores discretos para as classes.

O funcionamento da arquitetura *YOLO* segue o seguinte passo a passo:

- Escolhemos uma imagem como *input* para o modelo;
- A redimensionamos para um tamanho fixo — no artigo original, 448×448;
- Dividimos a imagem em $S \times S$ quadrados (*grid cells*) — no artigo original, $S = 7$;

É importante evidenciar que, nesse contexto, cada *grid cell* terá a capacidade de prever um objeto cujo centro esteja localizado no seu interior.

Figura 2.3: *Figura que ilustra a as grid cells do YOLOv1 em uma imagem. Imagem extraída de* Redmon *et al., 2016.*

Para treinar o modelo *YOLOv1*, considere o seguinte rótulo de *bounding box*:(x, y, w, h); onde (x, y) corresponde ao centro da *bounding box* e w e h à sua largura e altura, respectivamente. Perceba que no rótulo, os valores de x, y, w e h são valores absolutos relativos aos *pixels* da imagem inteira. Contudo, a previsão do modelo e, portanto, os valores alvo são relativos à *grid cell* em que o centro da *bounding box* está localizado. Dessa forma, uma importante etapa de pre-processamento dos dados que passam pela arquitetura do

YOLOv1 é a normalização desses valores para que sejam relativos à *grid cell*. Isso é feito da seguinte maneira:

Os componentes de uma *bounding box* predita são: $(\Delta x, \Delta y, \Delta w, \Delta h)$; onde:

$$\Delta x = (x - x_a) \,/\, (L/S) \tag{2.1}$$
$$\Delta y = (y - y_a) \,/\, (L/S) \tag{2.2}$$

Perceba que (L/S) é justamente o tamanho do lado de uma *grid cell*, pois L é o comprimento do lado da imagem. Além disso, (x_a, y_a) corresponde às coordenadas do canto esquerdo superior da *grid cell*. Desse modo, note que Δx e Δy correspondem ao centro da *bounding box* em uma posição relativa à *grid cell* em que ele se encontra.

De maneira análoga:

$$\Delta x = (x - x_a) \,/\, (L/S) \tag{2.3}$$
$$\Delta y = (y - y_a) \,/\, (L/S) \tag{2.4}$$

Então, Δw e Δh correspondem ao tamanho da *bounding box* em relação às medidas de uma *grid cell*.

Além desse processamento, outra coisa que deve ser feita é a codificação dos rótulos da classe de cada objeto. Esse processo é chamado *One-hot encoding* e nele, a classe de um objeto é definida por um vetor de tamanho N em que N é a quantidade de classes que o problema possui e é tal que:

$$y_i = \begin{cases} 1, & \text{se } i = k \\ 0, & \text{se } i \neq k \end{cases} \tag{2.5}$$

onde i representa o índice da classe atual, e k é o índice da classe alvo para o objeto em questão. Ou seja, o vetor resultante do *one-hot encoding* possui valor 1 apenas na posição da classe do objeto e 0 nas demais posições.

Sobre os conceitos envolvidos com as *grid cells* do *YOLOv1*, resta explicar que, cada *grid cell* terá um número B de *bounding boxes* candidatas a englobarem o objeto cujo centro está dentro de seus limites. No artigo original, $B = 2$. Assim, a *bounding box* que tiver o maior valor de confiança de que contém um objeto será a escolhida como *bounding box* para essa *grid cell*.

Para continuar o entendimento dos passos envolvidos na detecção de objetos do *YOLOv1*, precisamos saber como é o resultado da tarefa de detecção de objetos sobre uma imagem. Desse modo, após o *forward pass*, temos o seguinte vetor de predição:

$$\left[\Delta\hat{x}_1,\ \Delta\hat{y}_1,\ \Delta\hat{w}_1,\ \Delta\hat{h}_1,\ \hat{c}_1,\ \Delta\hat{x}_2,\ \Delta\hat{y}_2,\ \Delta\hat{w}_2,\ \Delta\hat{h}_2,\ \hat{c}_2,\ \hat{p}_1,\ \hat{p}_2,\ \ldots,\ \hat{p}_{20}\right]$$

onde:

$$\left[\Delta\hat{x}_i,\ \Delta\hat{y}_i,\ \Delta\hat{w}_i,\ \Delta\hat{h}_i,\ \hat{c}_i\right]_{i=1}^{B}$$

corresponde às coordenadas das *bounding boxes* preditas pela *grid cell* em questão e $\hat{c}$ a confiança de que, nessa *grid cell* o centro de um objeto esteja localizado.

Além disso,

$$\left[\hat{p}_1,\ \hat{p}_2,\ \ldots,\ \hat{p}_{20}\right]$$

São os valores das probabilidades preditas para cada classe, no modelo proposto por *One-hot encoding*. Temos o número de 20 classes pois o *YOLOv1* foi concebido para atuar com o *dataset* PASCAL VOC (Redmon *et al.*, 2016).

Assim, para cada *grid* cell, temos um vetor de tamanho 1×30 de resultado. Como são 7×7 *grid cells* em uma imagem, temos de resultado de um *forward pass* um vetor de tamanho $7 \times 7 \times 30$.

Nesse ponto, perceba que, para obtermos as coordenadas e as dimensões das *bounding boxes*, precisamos passar pelo seguinte processo (estamos revertendo a normalização feita no início do *forward pass*):

$$\begin{cases} x_1 = \Delta x_1 \times (L/S) + x_a \\ y_1 = \Delta y_1 \times (L/S) + y_a \\ w_1 = \Delta w_1 \times L \\ h_1 = \Delta h_1 \times L \end{cases}$$

E, para obter as informações sobre a classe, temos:

- **Classe do objeto**: A classe l do objeto é dada por:

$$l = \arg\max(p_1, p_2, \ldots, p_{20})$$

- **Confiança da classe**: A confiança da classe ($\hat{c}_1$ ou $\hat{c}_2$) é calculada multiplicando a confiança da *bounding box* pelo valor máximo de probabilidade p entre as classes. Dessa forma:

$$p = \max(p_1, p_2, \ldots, p_{20})$$

e

$$\hat{c} = c_i \times p$$

onde $c_i = \max\{c_1, c_2\}$.

Então, ao exibir as informações sobre o objeto detectado, levamos em consideração a confiança $\hat{c}$ e a classe l.

Arquitetura: A arquitetura do *YOLOv1* é composta basicamente por uma rede neural convolucional convencional. É uma arquitetura inspirada pela *GoogleNet*, em que temos 24

camadas convolucionais e 2 camadas *fully connected* ao final da rede para que possamos obter o vetor de saída desejado. As 24 camadas convolucionais são responsáveis por gerar o mapa de representações da imagem e geralmente constituem uma porção da rede que é chamada de *backbone*. Por outro lado, as camadas *fully connected* são responsáveis por usar o mapa de representações para obter as predições do modelo.

Treinamento do *YOLOv1* O treinamento do modelo *YOLOv1* geralmente começa com uma rede neural pré-treinada no *dataset ImageNet* (Deng *et al.*, 2009). Depois disso, finalizamos o treinamento (*fine tune*) no *dataset* de detecção de objetos. O processo de treinamento do *YOLOv1* com *dataset* de detecção de objetos pode ser resumido na figura 2.4.

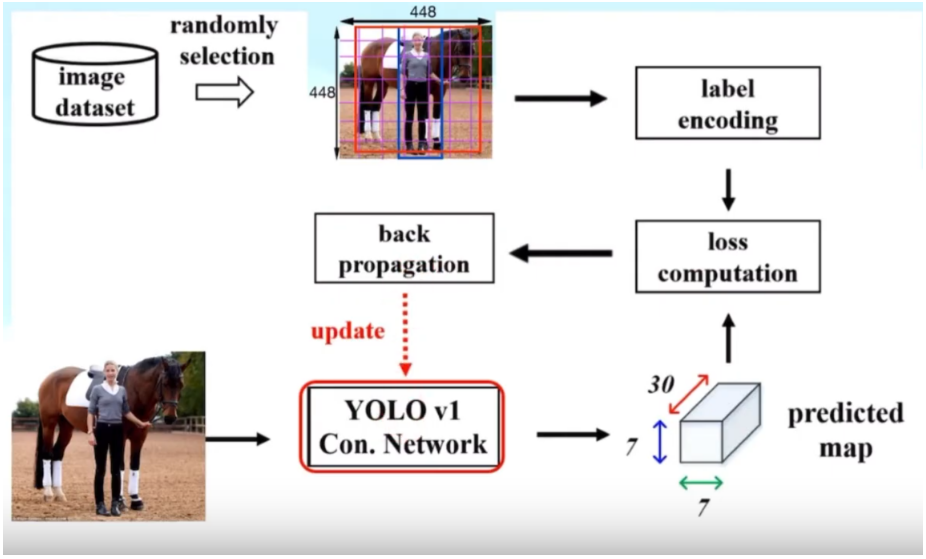


Figura 2.4: *Ilustração do processo de treinamento do YOLOv1. Disponível em: https://www.youtube.com/watch?v=zgbPj4lSc58&list=PL1u-h-YIOL0sZJsku-vq7cUGbqDEeDK0a&index=1&t=1517s.*

Escolhemos uma imagem aleatória do conjunto de dados, efetuamos o pré-processamento de seus rótulos como comentado, passamos a imagem pela rede neural, computamos a função de perda ao comparar os resultados do *forward pass* com os rótulos processados e então efetuamos o processo de *back propagation*, atualizando os pesos do modelo com base no gradiente da função de perda.

Função de perda A função de perda introduzida pelo *YOLOv1* é a soma das funções de perda por todas as *grid cells*:

$$L = \sum_{i=1}^{S^2} L_i$$

A função de perda é dividida nos casos em que o centro de um objeto está ou não localizado dentro de uma *grid cell*. Isso é feito por meio das funções indicadoras $\mathbb{1}_i^{\text{obj}}$ e $\mathbb{1}_i^{\text{no\_obj}}$, que assumem respectivamente o valor 1 quando há e não há objetos na *grid cell* i, e assumem 0

caso contrário. Além disso, desejamos dar mais importância para a presença de objetos na imagem, por isso, há a constante $\lambda_{\text{no\_obj}}$, que no artigo original tem o valor de 0,5:

$$L = \sum_{i=1}^{S^2} \mathbb{1}_i^{\text{obj}} \cdot L_{i,\text{obj}} + \lambda_{\text{no\_obj}} \sum_{i=1}^{S^2} \mathbb{1}_i^{\text{no\_obj}} \cdot L_{i,\text{no\_obj}}$$

Agora, vamos definir $L_{i,\text{obj}}$. Essa função é definida como uma soma ponderada da função de perda da confiança de que há um objeto cujo centro está na *grid cell i* (*objectness loss*), a função de perda da classificação e a função de perda das coordenadas da *bounding box*.

$$L_{i,\text{obj}} = \lambda_{\text{coord}} \cdot L_{i,\text{obj}}^{\text{box}} + L_{i,\text{obj}}^{\text{conf}} + L_{i,\text{obj}}^{\text{cls}}$$

No artigo original, λ_{coord} tem o valor de 5.

A função de perda para as *bounding boxes*, $L_{i,\text{obj}}^{\text{box}}$, é definida como a soma dos erros quadráticos entre os valores preditos e os valores presentes no rótulo da imagem em questão (*ground truth*). A expressão é dada por:

$$L_{i,\text{obj}}^{\text{box}} = (\Delta x_i - \Delta\hat{x}_i)^2 + (\Delta y_i - \Delta\hat{y}_i)^2 + \left(\sqrt{\Delta w_i} - \sqrt{\Delta\hat{w}_i}\right)^2 + \left(\sqrt{\Delta h_i} - \sqrt{\Delta\hat{h}_i}\right)^2$$

onde: Δx_i e Δy_i são as coordenadas reais do centro da *bounding box*; $\Delta\hat{x}_i$ e $\Delta\hat{y}_i$ são as coordenadas preditas; Δw_i e Δh_i são a largura e a altura reais da *bounding box*; e $\Delta\hat{w}_i$ e $\Delta\hat{h}_i$ são os valores preditos.

A raiz quadrada em w e h está presente na computação dessa função de perda pois, quanto maior a *bounding box*, maiores serão as diferenças $(\Delta w_i - \Delta\hat{w}_i)$ e $\left(\Delta h_i - \Delta\hat{h}_i\right)$. Esse aumento ocorre ainda mais ao usarmos o quadrado dessas diferenças. Por isso, a função de perda penaliza mais caixas maiores. Para tentar melhorar essa situação, a ideia é introduzir a raiz quadrada para diminuir a penalização de caixas de maior tamanho, já que maiores valores em $(\Delta w_i - \Delta\hat{w}_i)$ e $\left(\Delta h_i - \Delta\hat{h}_i\right)$, após o uso da raiz quadrada, não impactarão tanto no valor final da função de perda.

Já a função de perda da confiança da presença de centro de objeto na *grid cell i*, é somente o erro quadrático da confiança predita e do valor rótulo (nesse caso, sempre 1).

$$L_{i,\text{obj}}^{\text{conf}} = (c_i - \hat{c}_i)^2$$

Por fim, a função de perda de classificação também é constituída por erro quadrático.

$$L_{i,\text{obj}}^{\text{cls}} = \sum_{c=1}^{20} (p_{i,c} - \hat{p}_{i,c})^2$$

Em que p e $\hat{p}$ são os vetores das probabilidades.

Agora, precisamos computar a função de perda no caso em que não há objetos cujo centro está localizado na *grid cell i*. Nesse caso, somente levamos em consideração a *objectness score*:

$$L_{i,\text{no\_obj}} = \sum_{j=1}^{B} (c_{i,j} - \hat{c}_{i,j})^2$$

Em que o valor de $\hat{c}_{i,j}$ é 0 e B igual a 2.

Limitações Apesar de ser uma arquitetura disruptiva que revolucionou o cenário da visão computacional e detecção de objetos, o *YOLOv1* apresentava algumas limitações. Principalmente a incapacidade de detectar muitos objetos, principalmente quando os objetos são pequenos e estão amontoados. (limitação devido às *grid cells*). Além disso, como todas as *bounding boxes* têm sua origem em um processo adaptativo relativo às *grid cells*, o *YOLOv1* tem dificuldade em adaptar as caixas a objetos com proporções variadas (Terven *et al.*, 2023).

2.3.3 Métricas

Para avaliar diferentes modelos de detecção de objetos, temos diversas métricas para quantificar a qualidade do modelo em identificar corretamente objetos em uma imagem. Vamos detalhar as métricas mais comuns, com base nas definições matemáticas do artigo (Padilla *et al.*, 2020).

- ***Intersection over Union (IoU*:** *Intersection over Union* mede a área de sobreposição entre a *bounding box* predita (B_p) e a *bounding box* real (*ground truth*) (B_{gt}).

$$\text{IoU} = \frac{\text{Área}\left(B_p \cap B_{gt}\right)}{\text{Área}\left(B_p \cap B_{gt}\right)}$$

  Uma das funções mais importantes dessa métrica é definir:

  - Verdadeiros positivos (*TP*): uma detecção correta de uma *bounding box* real (*ground-truth* (Padilla *et al.*, 2020);
  - Falsos positivos (*FP*): uma detecção incorreta de um objeto inexistente ou uma detecção mal posicionada de um objeto existente (Padilla *et al.*, 2020);

  - Falsos negativos (*FN*): uma *bounding box* real (*ground-truth*) não detectada (Padilla *et al.*, 2020).

  Desse modo, consideramos a detecção correta de uma *bounding box* caso a classe detectada seja a correta e $IoU > t$.

- **Precisão e *Recall*:** A *Precisão* (P) mede a proporção de predições corretas entre todas as predições feitas:

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

  O *Recall* (R) mede a proporção de objetos reais detectados entre todos os objetos presentes na imagem:

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- ***Average Precision (AP)*:** A *Average Precision* (AP) é uma medida que combina a precisão e o *recall*, calculando a área sob a curva de precisão-*recall*. Primeiramente, precisamos entender como obtemos diversos valores de precisão e *recall* para que seja possível traçar essa curva.

  Isso é feito ao escolhermos diversos valores t do *threshold* de confiança para classificação. Para cada valor de t, obtemos um par de valores de precisão e *recall*. Depois disso, traçamos a curva. Então, um método comum para o cálculo do AP é a **interpolação em todos os pontos** (Padilla *et al.*, 2020):

$$AP_{\text{all}} = \sum_{n} (R_{n+1} - R_n) P_{\text{interp}}(R_{n+1})$$

  onde onde

$$P_{\text{interp}}(R_{n+1}) = \max_{\tilde{R} : \tilde{R} \geq R_{n+1}} P(\tilde{R}).$$

  Uma noção visual pode ser vista na figura 2.5.

- ***Mean Average Precision (mAP)*:** A *Mean Average Precision* (mAP) é a média do AP calculada sobre todas as classes em um conjunto de dados. Se um modelo é avaliado em N classes, a mAP é dada por:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

  Onde AP_i é a AP para a classe i.

Essas métricas, especialmente o AP e o mAP, são amplamente utilizadas para avaliar e comparar o desempenho de diferentes modelos de detecção de objetos. Além disso, há diferentes variantes do mAP para diferentes valores de IoU. Por exemplo, AP@50 (que gera a curva AP com o valor de t em 0,5); AP@75 e AP@50:5:95, que entrega diversos valores.

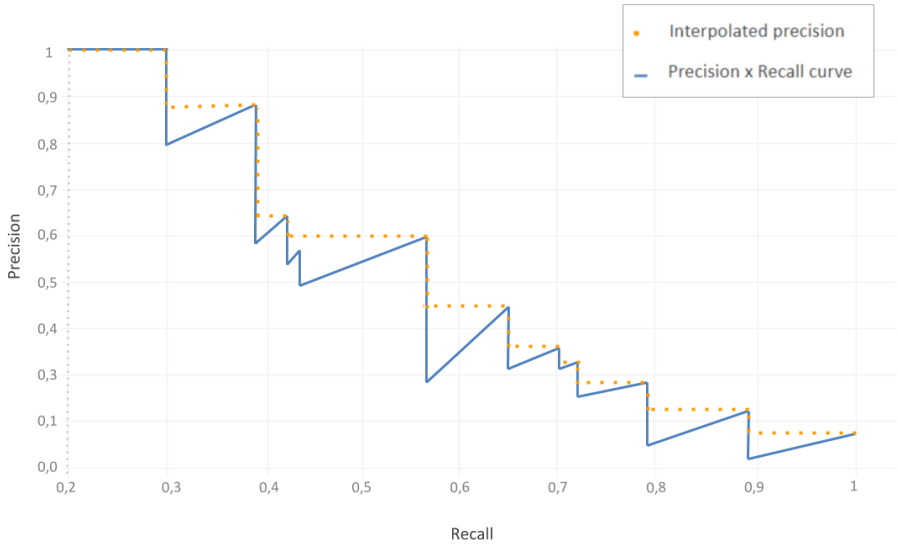


Figura 2.5: *Exemplo de curva de precisão-recall usando interpolação em todos os pontos para o cálculo da Average Precision (AP). Disponível em: https://manalelaidouni.github.io/assets/img/pexels/all_point_interpolated_AP_(2)-848db820-1379-456f-bad8-8e45ea562fd0.png.*

2.3.4 Avanços da família *YOLO*

***Backbone, neck* e *head*:** Com o avanço das arquiteturas para detecção de objetos, nomenclaturas para diferentes partes das redes neural convolucionais começaram a ser utilizadas. Nesse contexto, se destacam o **backbone**, **neck** e **head**. Por isso, como parte da explicação sobre a evolução da família *YOLO*, é interessante abordar a função de cada uma dessas partes dos detectores de objetos.

- **Backbone:** o *backbone* da rede convolucional, muitas vezes referido como uma rede em si, é muito comum em arquiteturas de modelos de detecção de objetos. A função do *backbone* é realizar a extração das representações da imagem. Além disso, essas características extraídas são codificadas em uma dimensão adequada para o *neck*, que será explicado a seguir. O *backbone* deve ser capaz de capturar representações de alto nível e baixo nível da imagem de entrada (Shroff, 2021).
- **Neck:** depois do *backbone*, o *neck* é responsável por transformar ainda mais as representações que foram extraídas anteriormente. O seu objetivo é melhorar a qualidade dessas representações de modo que elas sejam capazes de fornecer ainda mais informação para a detecção de objetos (Shroff, 2021).
- **Head:** a *head* do modelo de detecção de objetos é a sua parte final. Ela é responsável por produzir as predições do modelo e se baseia nas informações extraídas pelo *backbone* e *neck* (Shroff, 2021).

Abordagem baseada em âncoras: Desde o lançamento do *YOLOv1*, a abordagem para detecção de objetos evoluiu significativamente. As versões subsequentes introduziram conceitos importantes, como o uso de *âncoras* (*anchors*) no *YOLOv2* e *YOLOv3*. As âncoras

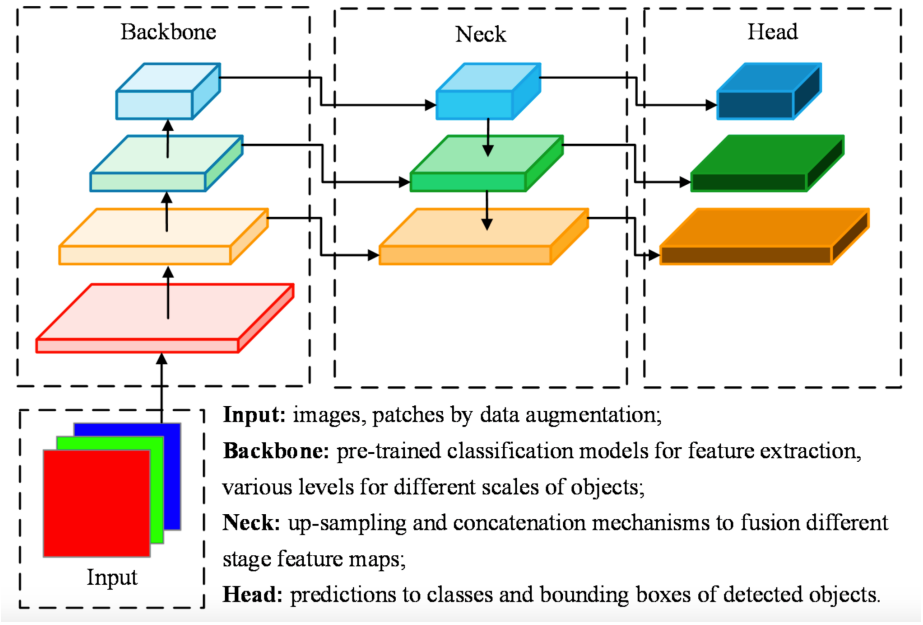


Figura 2.6: *Estrutura de detecção de objetos mostrando a separação entre Backbone, Neck e Head. Disponível em: https://velog.io/@peterkim/Object-DetectionìŮŘìĎœ-ëğŘíŢŸëŁŤ-Backbone-Neck-Head.*

são *bounding boxes* predefinidas que servem como referências para o modelo detectar objetos de diferentes escalas e proporções. Isso é feito geralmente utilizando o algoritmo de clusterização não supervisionado *K-means* nas *bounding boxes* presentes nos dados de treinamento do modelo de detecção. Essa técnica melhorou a precisão na detecção, mas adicionou complexidade ao processo (Terven *et al.*, 2023).

***YOLOX*:** Com o surgimento do *YOLOX* Ge *et al.*, 2021, foi adotada uma abordagem *anchor-free*, eliminando a necessidade de âncoras predefinidas. Em vez disso, o modelo efetua a regressão das *bounding boxes* diretamente pela *CNN*, simplificando o processo de detecção. Além disso, o *YOLOX* introduziu à família de modelos *YOLO* a abordagem de uma *head* desacoplada, ou seja: cada "parte" (*branch*) da *CNN* poderia se especializar na sua tarefa (uma delas na regressão das *bounding boxes* e a outra na tarefa de classificação). Nesse ponto, a tarefa de detecção de objetos na arquitetura *YOLO* deixou de ser uma tarefa centrada em regressão e passou a ser uma tarefa desacoplada de classificação e regressão (Ge *et al.*, 2021). De qualquer forma, é interessante ressaltar que, apesar do desacoplamento, o treinamento é feito de maneira única. Com uma só passagem da imagem pela rede neural, obtemos todos os resultados da tarefa de detecção de objetos, o que não retorna à abordagem tradicional (de detecção em múltiplas etapas).

YOLOv8: O *YOLOv8*, desenvolvido pela *Ultralytics* e lançado em janeiro de 2023 (Ultralytics, 2023b), incorporou essa abordagem *anchor-free* e introduziu diversas melhorias na arquitetura:

- **Melhorias no *backbone* e *neck***: O *YOLOv8* utiliza um *backbone* mais eficiente para extração de representações, aliado a um *neck* aprimorado que facilita a fusão de informações de múltiplas escalas. Isso melhora a capacidade do modelo em detectar

objetos de diferentes tamanhos, especialmente objetos pequenos em imagens de alta resolução;

- **Desacoplamento da *head***: De modo semelhante ao *YOLOX*, o *YOLOv8* implementa uma *head* desacoplada, onde as tarefas de regressão das *bounding boxes* e de classificação são realizadas de forma separada. Essa separação permite otimizar cada tarefa individualmente, aumentando a precisão geral do modelo.
- **Funções de perda específicas**: No processo de treinamento, o modelo utiliza uma combinação de *Complete Intersection over Union* (CIoU) (Zheng *et al.*, 2020) e *Distribution Focal Loss* (DFL) (Li *et al.*, 2020) na função de perda da regressão das *bounding boxes*. Para a perda de classificação, é empregada a *Binary Cross-Entropy* (BCE). Essa abordagem especializada para cada tipo de perda contribui para melhorias significativas na precisão da detecção (Terven *et al.*, 2023).

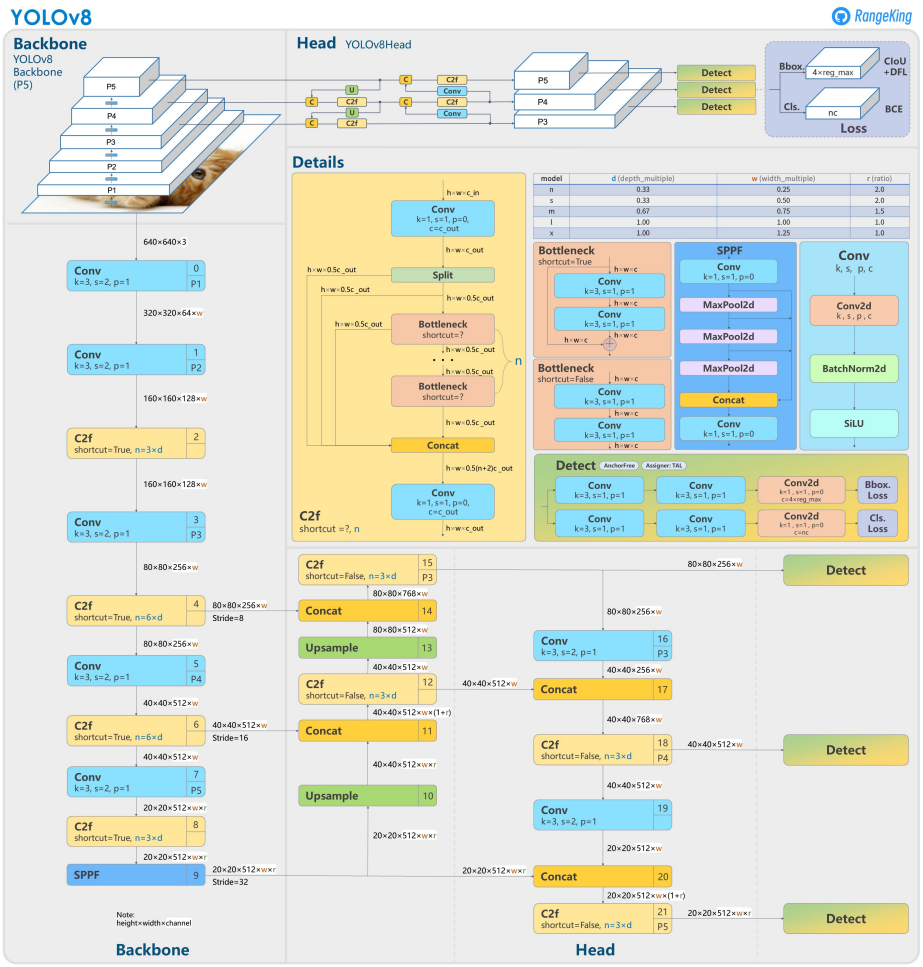


Figura 2.7: *Figura que ilustra a arquitetura do YOLOv8. Imagem extraída de King. Disponível em: https://github.com/ultralytics/ultralytics/issues/189.*

Essas inovações resultaram em um desempenho superior do *YOLOv8* em comparação com suas versões anteriores. Em testes realizados no conjunto de dados COCO, o *YOLOv8* demonstrou um aumento no *Mean Average Precision* (mAP) e uma velocidade de inferência mais rápida (Ultralytics, 2023a).

Infelizmente, não houve a publicação de um artigo científico oficial da *Ultralytics* para que possamos saber com facilidade o passo a passo do funcionamento do modelo *YOLOv8*. Além disso, o repositório do *Github* que contém a codificação da arquitetura do modelo está passando por atualizações constantes, com as modificações do código fonte do *YOLOv8* para novos códigos, que originaram o *YOLO11*. Dessa forma, consideramos que a profundidade da explicação feita sobre o *YOLOv8* é o bastante para o intuito desse trabalho de formatura supervisionado.

2.4 Materiais e métodos

Depois de escolhermos o modelo *YOLOv8* para realizar a detecção de objetos no projeto *TomatoHealth*, chegou o momento de treiná-lo para que ele tivesse a capacidade de detectar as doenças em folhas de tomate. Como comentado anteriormente, o *dataset* escolhido para essa tarefa foi o *PlantDoc*, mas com o escopo reduzido para doenças em folhas de tomate.

2.4.1 Processamento do *dataset*

Para obtermos o *dataset* das doenças em folhas de tomate, precisamos então realizar uma filtragem no *PlantDoc* para que somente imagens que possuam rótulos de doenças de tomate (e esses rótulos em si) permanecessem na versão do *dataset* que seria utilizada para treinar o modelo.

Para fazer isso, primeiro obtemos o conjunto de dados *PlantDoc* a partir da plataforma *Dataset Ninja*[1] — essa plataforma disponibiliza conjuntos de dados para uso em projetos de *Machine Learning*. Depois disso, no processo de filtragem do *dataset*, também precisamos fazer adaptações no formato dos *rótulos* e da estrutura de diretórios para que tivéssemos um conjunto de dados adaptado para a arquitetura *YOLO*.

Sobre os rótulos, o formato inicial define as *bounding boxes* a partir do seu vértice superior esquerdo e do seu vértice inferior direito. Então precisamos fazer uma transformação para obter o tamanho, largura e centro de cada caixa. O código responsável por esse processamento pode ser encontrado no repositório de códigos do projeto.[2]

Inicialmente, a estrutura de diretórios apresentava o seguinte formato:

```
treino/
|-- imgs/
|-- rótulos/
teste/
|-- imgs/
|-- rótulos/
```

Para o uso com o *YOLO*, a organização de diretórios precisa ser ajustada para separar as imagens e os rótulos em diretórios específicos, com subdiretórios para treino e teste. A estrutura correta é:

[1] https://datasetninja.com/plantdoc

[2] https://github.com/heitorc62/TCC

```
imgs/
|-- treino/
|-- teste/
rótulos/
|-- treino/
|-- teste/
```

Ao final de todo esse processamento, restaram as seguintes classes no *dataset*:

ID	Nome Original (Inglês)	Nome em Português
0	Tomato Early blight leaf	Folha de tomate com pinta-preta
1	Tomato leaf	Folha de tomate saudável
2	Tomato leaf bacterial spot	Folha de tomate com mancha bacteriana
3	Tomato leaf late blight	Folha de tomate com requeima
4	Tomato leaf mosaic virus	Folha de tomate com vírus do mosaico
5	Tomato leaf yellow virus	Folha de tomate com Begomovirus (yellow leaf curl virus)
6	Tomato mold leaf	Folha de tomate com mofo
7	Tomato Septoria leaf spot	Folha de tomate com septoriose (fungo)
8	Tomato two spotted spider mites leaf	Folha de tomate com ácaro-rajado (spider mite)

Tabela 2.1: *Nomes originais das classes e suas traduções em português.*

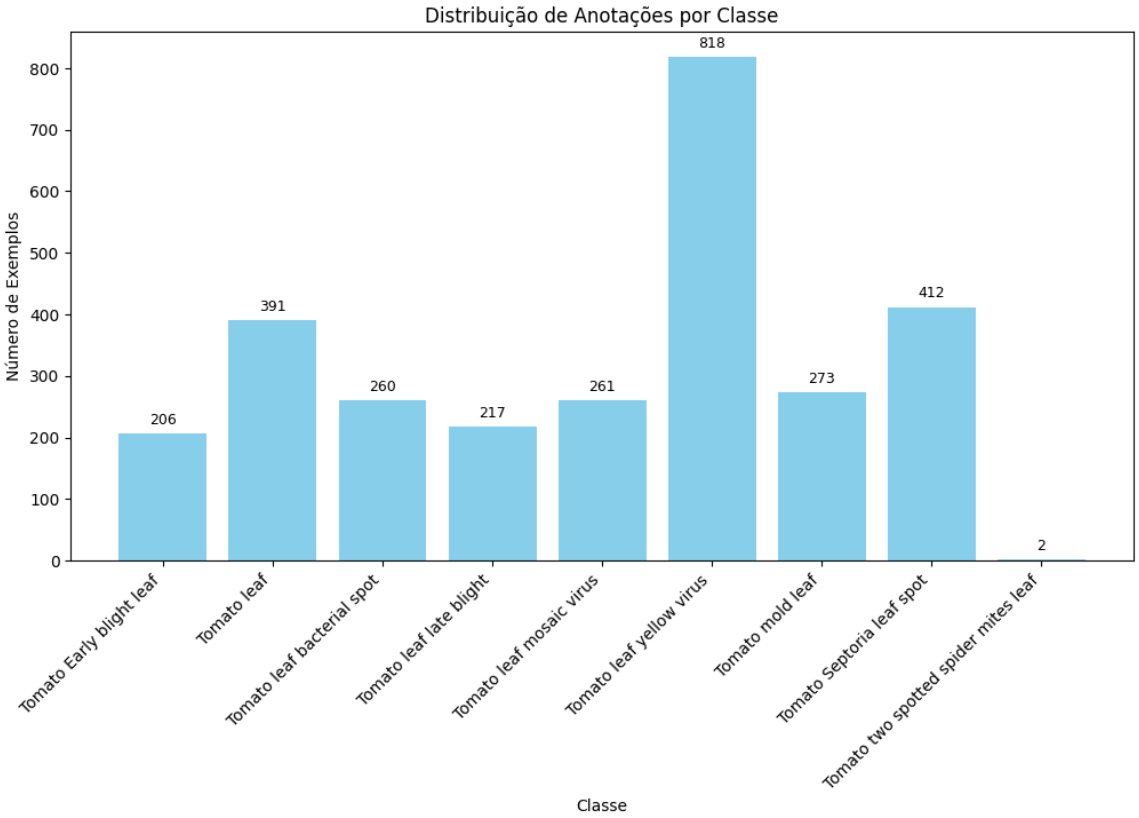


Figura 2.8: *Gráfico de barras mostrando a distribuição de anotações por classe após o processamento do conjunto de dados PlantDoc. Fonte: acervo dos autores.*

Totalizando 721 imagens, 2840 anotações e uma média de 3.93897 anotações por imagem. Na figura 2.8 é possível enxergar a distribuição de anotações por classes. É notável que a

classe *Tomato two spotted spider mites leaf* possui um número muito pequeno de exemplos. Contudo, como o intuito do projeto é, justamente estender o atual conjunto de dados com novas imagens e novas anotações, decidimos não desconsiderar essa classe e mantê-la no conjunto de dados.

2.4.2 Treinamento

Após o pré-processamento, o conjunto de dados foi adequadamente estruturado para ser utilizado pelo *YOLO*. Com isso, o próximo passo foi realizar o treinamento do modelo. Executamos um processo de ajuste fino (*fine-tuning*) do *YOLOv8n* (uma versão mais leve do *YOLOv8*), previamente treinado no *dataset* COCO[3] (Lin *et al.*, 2014).

Para realizar o ajuste fino, foram utilizadas duas abordagens principais:

- **Linha de comando com *Bash***: A primeira opção é rodar o treinamento diretamente no terminal com um comando *bash*, que especifica configurações, como a tarefa, o número de épocas e o tamanho da imagem:

```
yolo task=detect mode=train model=yolov8n.pt data=tomato.yaml epochs
    =300 imgsz=640 device=0
```

  - `task=detect`: define a tarefa como detecção de objetos.
  - `model=yolov8n.pt`: indica o modelo *YOLOv8* pré-treinado a ser utilizado.
  - `data=tomato.yaml`: aponta para o arquivo de configuração dos dados, que mapeia as classes para seus nomes e os caminhos das imagens.
  - `epochs=50`: número de épocas para o treinamento.
  - `imgsz=640`: define o tamanho das imagens usadas no treinamento.
  - `device=0`: especifica o dispositivo (*GPU*) para acelerar o treinamento.

- **Uso da biblioteca *Python* da *Ultralytics***: A segunda opção é utilizar funções disponibilizadas pela biblioteca de *Python* da *Ultralytics*:

```
results = model.train(task="detect", data=data_yaml, epochs=epochs,
    imgsz=img_size, device=device)
```

Para treinar o modelo, utilizamos a rede de computadores (*vision, e-science*) com placas de vídeo (*GPU*). Mais especificamente a máquina chamada *deepthree*, com 200*GB* de RAM e equipada com a placa de vídeo *NVIDIA GeForce GTX 1080 Ti* com 12*GB* de memória.

2.4.3 Resultados

Com o *dataset* que obtivemos após o processamento e utilizando o *YOLOv8n*, decidimos fazer um treinamento por 300 épocas. As ferramentas fornecidas pela *Ultralytics* facilitam muito esse processo. O parâmetro *EarlyStopping(patience=100)* define que o treinamento para após 100 épocas sem melhorias nas métricas de avaliação. Então, após 183 épocas o treinamento foi finalizado já que o melhor modelo encontrado foi do final da época de

[3] https://yolov8.org/yolov8-coco-dataset/

número 83. O treino com 183 épocas e utilizando as configurações de hardware mencionadas no capítulo 2.4.2, leva cerca de 30 minutos e resulta em um arquivo de pesos de cerca de 6*MB*. Os resultados obtidos podem ser vistos na figura 2.9.

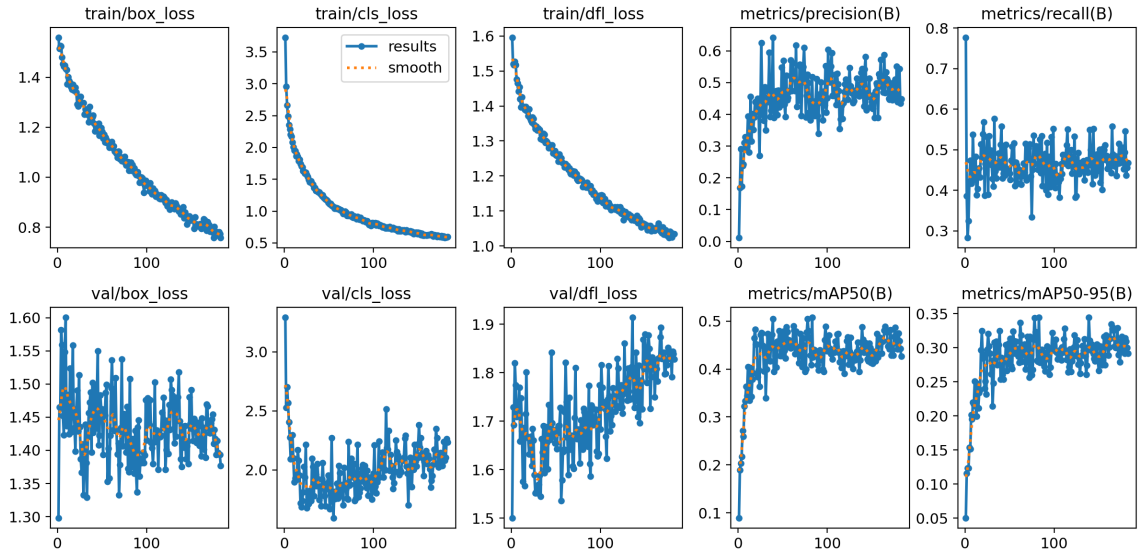
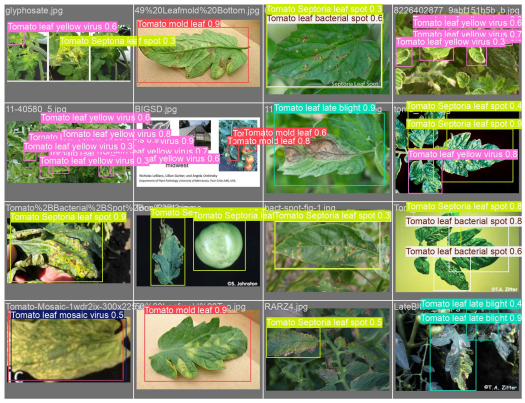


Figura 2.9: *Resultados obtidos após 183 épocas de treinamento. Fonte: acervo dos autores.*

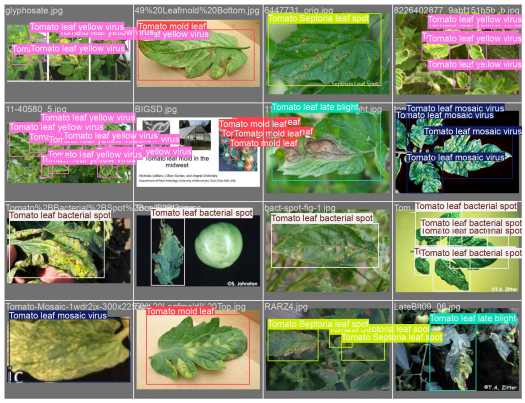
O melhor valor da métrica *mAP@50-95* obtido após o treinamento foi: 34,437; Se compararmos esse valor com os melhores resultados do *YOLOv8* na referência de performance do *dataset COCO* (37.3) conseguimos notar que, pelo menos na teoria, o conjunto de dados está performando bem.

Apesar do intuito do *TomatoHealth* ser a construção de um conjunto de dados robusto no domínio de detecção de doenças em folhas de tomate, pensamos que em um momento inicial é interessante ter um modelo que seja o melhor possível.

Assim, devido ao pequeno número de exemplos que possuímos inicialmente nesse *dataset*, reconhecemos que essa métrica de 34,437 pode estar superestimada e isso pode ser observado nas figuras 2.10a e 2.10b (predições incorretas no conjunto de validação). Então, consideramos uma estratégia para tentar melhorar essa performance inicial. Essa estratégia será discutida no capítulo 4.1.



(a) *Exemplo de deteccao de objetos em imagens do conjunto de validação. Imagem extraída do acervo dos autores.*



(b) *Exemplo de rótulos de objetos. Imagem extraída do acervo dos autores.*

Figura 2.10: *Comparação entre a detecção de objetos e os rótulos originais.*

Capítulo 3

Detalhes sobre a implementação do sistema

Nesse capítulo, apresentaremos detalhes técnicos sobre a implementação do sistema *TomatoHealth*, tecnologias utilizadas e decisões do desenvolvimento tomadas para um sistema com as capacidades do descrito no capítulo 1.4.

3.1 Arquitetura cliente-servidor e contêineres *Docker*

O *TomatoHealth* implementa uma arquitetura de cliente-servidor, com dois tipos de clientes diferentes: clientes de usuários comuns e de especialistas, que se diferenciam conforme os requisitos do capítulo 1.4.2.

Do ponto de vista da arquitetura cliente-servidor e da descrição abstrata para o usuário geral, os tópicos de mais difícil implementação são: tirar, enviar e receber uma foto para análise; já que entender o papel da ferramenta é uma tarefa do *front-end*. Para realizar essas tarefas difíceis, o cliente faz uma requisição *POST HTTP* enviando uma imagem com o uso da *API FormData*.[1] E por sua vez, o servidor salva essa imagem no banco de dados *S3 MinIO*, realiza a detecção das classes presentes, faz um prompt para a *LLM* `gpt-4o-mini` da *OpenAI* por *API*[2] (pergunta ao modelo como tratar um tomateiro com as doenças detectadas) e finalmente responde o *POST* do cliente com o *output* da *LLM*.

Já para o cliente especialista, as tarefas de ver as imagens tiradas por usuários comuns e demarcar as diferentes classes são solucionadas com a utilização da ferramenta de anotação *Label Studio*. Essa ferramenta já inclui uma interface de rotulagem de dados que se comunica com o banco onde essas imagens são guardadas, e ela pode ser facilmente alterada para os propósitos da nossa aplicação.

Um modelo simples para nossa arquitetura pode ser visto na figura 3.1.

[1] https://developer.mozilla.org/en-US/docs/Web/API/FormData

[2] https://platform.openai.com/docs/api-reference/introduction

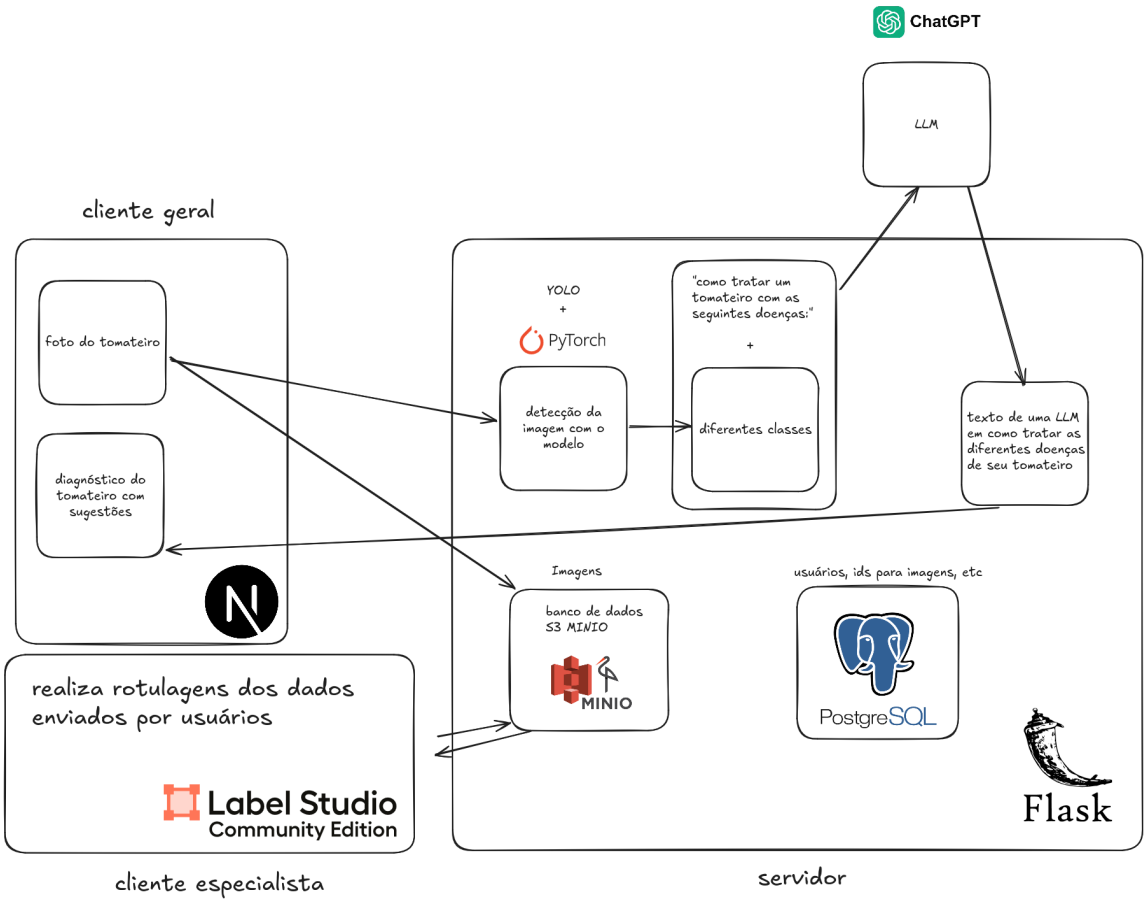


Figura 3.1: *Modelo simples para a arquitetura implementada. A imagem apresenta as logos das tecnologias usadas: ChatGPT, Next.js, Label Studio, MinIO, PostgreSQL e Flask. Fonte: acervo dos autores.*

Além disso, utilizamos contêineres *Docker*,[3] pois facilitam o processo de desenvolvimento, uma vez que lidamos com o uso de diferentes *frameworks* e bancos de dados, além de utilizarmos a ferramenta *Docker Compose* para o controle de inicialização, execução e comunicação entre os contêineres. A figura 3.2 demonstra os diferentes contêineres de nossa aplicação.

3.2 *Back-end*

O *back-end* do projeto foi implementado para contemplar os requisitos do capítulo 1.4.2. A maior parte do código foi feito utilizando *Flask*,[4] um *micro-framework Python* para aplicações *web*.

Com essa ferramenta, implementamos os seguintes *endpoints* principais (*URLs* que representam funcionalidades no contexto da aplicação):

[3] https://www.docker.com/

[4] https://flask.palletsprojects.com/en/stable/

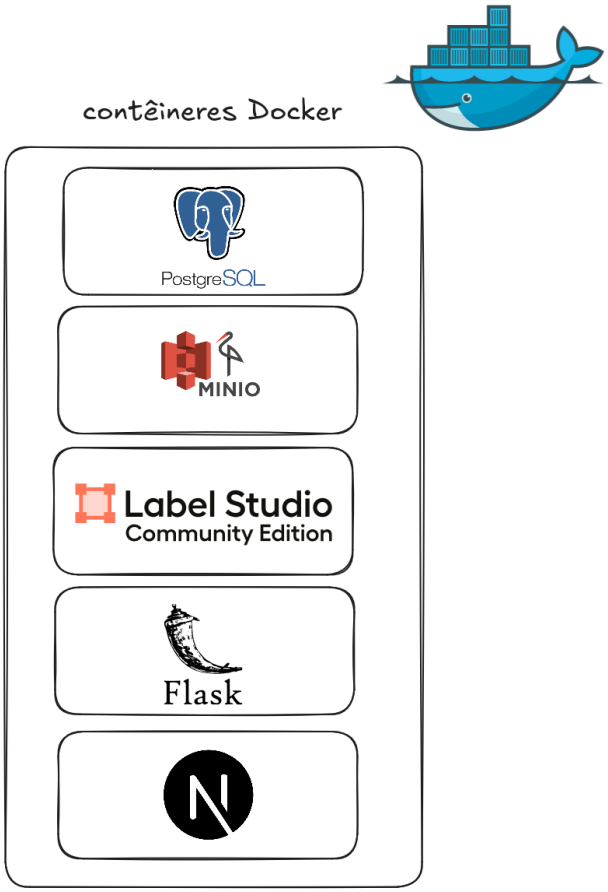


Figura 3.2: *Modelo simples para os contêineres da nossa aplicação. A imagem apresenta as logos das tecnologias usadas: Docker, Next.js, Label Studio, MinIO, PostgreSQL e Flask. Fonte: acervo dos autores.*

1. `/process_image`:
   (a) Recebe a imagem da folha do tomateiro enviada por um usuário regular;
   (b) Obtém as respostas do modelo de detecção de objetos em relação às doenças presentes na imagem de folha do tomateiro: Isso é feito ao invocar o método de detecção de objetos do modelo de visão computacional (objeto em *Python*) que é instanciado no início da aplicação

```
# Run detection using YOLOv8
detections = detection_model.predict(image, imgsz=640, conf=
    detection_threshold, device=device, verbose=False)
```

   (c) Salva a imagem enviada pelo usuário em um banco de dados de imagens: O banco de dados utilizado para a armazenagem de imagens é o *S3 MinIO*,[5] uma alternativa local ao sistema de armazenamento em nuvem *S3* da AWS. Além de salvar as imagens enviadas pelos usuários, o *MinIO* armazena todas as imagens do conjunto de dados em um diretório especial, no qual as imagens revisadas passarão a ser localizadas caso seja de interesse de um usuário administrador atualizar o conjunto de dados.

[5] https://min.io/

(d) Obtém a resposta de um modelo de linguagem que será entregue para o usuário: Utilizamos a *API* da *OpenAI*[6] para obter a resposta do modelo de linguagem;

(e) Envia a resposta à requisição;

2. `/register_admin`: Além do programa em *Flask*, uma parte importante do *back-end* da aplicação *TomatoHealth* é um programa de linha de comando que oferece uma maneria simplificada dos administradores do sistema interagirem com ele. Com isso, é possível criar uma conta de administrador (que será registrada no banco de dados *PostgreSQL*.[7] Com essa conta, temos acesso a todas as funcionalidades oferecidas pelo sistema, que serão explicadas a seguir.

3. `/admin_login` Depois de registrado um administrador, precisamos fazer *login* na plataforma para receber um *token* de autenticação *JWT*. Isso é necessário pois as operações sensíveis, são protegidas.

4. `/invite_reviewer` Depois que um administrador está autenticado, ele pode convidar um usuário especialista para a plataforma. Isso é feito ao informar o *e-mail* do especialista na requisição feita para `/invite_reviewer`. Com isso, a ferramenta *SendGrid* se encarrega de enviar um *e-mail* convidando o usuário especialista a fazer parte da plataforma.

5. `/register_reviewer/<token>` No *e-mail* recebido pelo usuário especialista, há um link que leva para a página de registro. Essa página é renderizada pela aplicação *Flask* ao acessarmos `/register_reviewer/<token>` pelo navegador *web*.

6. `/login` Depois que o usuário especialista criou uma conta na plataforma, ele já pode efetuar o *login*. Esse *endpoint* também é renderizado pela aplicação e após um *login* bem sucedido, o usuário especialista deve ser redirecionado para o endpoint `/labelling_tool`.

7. `/labelling_tool` Nesse *endpoint*, o usuário especialista será redirecionado para a plataforma de anotação das imagens enviadas pelos usuários. Então, é efetuada uma busca no banco de dados por imagens que ainda não foram revisadas. Ao encontrar uma imagem, uma tarefa no *label-studio* é criada (revisão dessa imagem) e então o usuário especialista é redirecionado para essa tarefa, nesse momento, ele pode efetuar as revisões e, ao clicar no botão de finalizar a revisão, a imagem será marcada como revisada.

8. `/update_dataset` Em um momento que há diversas imagens marcadas como revisadas, um usuário administrador pode utilizar o *endpoint* `/update_dataset` para mover todas as imagens com o *status* de revisadas para o diretório do *MinIO* que contém o conjunto de dados inteiro.

9. `/update_weights` O processo de retreinamento do modelo (com a versão atualizada do conjunto de dados) será explicado a seguir. De qualquer maneira, um arquivo de pesos da rede neural é o resultado do treinamento e, para atualizar o modelo que

[6] https://platform.openai.com/docs/overview

[7] https://www.postgresql.org/

está rodando na aplicação, basta enviar esse arquivo em uma requisição feita para o endpoint `/update_weights`.

3.2.1 *Machine Learning* no *back-end* da aplicação

Treinamento inicial: antes de podermos iniciar a operação do sistema, seja em ambiente de desenvolvimento ou produção, um modelo de visão computacional é treinado para detectar as doenças dado uma imagem.

Após esse treinamento, obtemos um arquivo com a estrutura e os valores dos pesos do modelo treinado. Então, no programa *Flask* do servidor, esse arquivo é usado para instanciar um objeto de rede neural (o modelo treinado). Assim, quando uma imagem é enviada via requisição para o endpoint `/process_image`, obtemos os resultados das doenças detectadas pelo modelo nessa imagem ao invocar o método de inferência de seu objeto e usar a imagem recebida como argumento:

```
detections = detection\_model.predict(image)
```

Retreinamento: para fazer o retreinamento do modelo de visão computacional, utilizamos o programa de linha de comando do administrador. Além disso, precisamos que o repositório do projeto esteja clonado na máquina responsável por esse processo (não só na máquina que roda a aplicação). Isso ocorre pois no repositório há um *script Python* que faz o download do novo conjunto de dados do *MinIO* e utiliza esses novos dados para treinar um novo modelo *YOLO*. Ao final desse processo, o arquivo de pesos resultante é enviado automaticamente para o *endpoint* `/update_weights`.

Uma funcionalidade interessante do projeto *TomatoHealth* é a possibilidade de fazer o retreinamento do modelo de maneira remota (como por exemplo por meio de *SSH*, isto é, acesso remoto). Para isso, um desafio importante que precisou ser superado é o fato de que precisaríamos de uma maneira de disponibilizar os dados do *MinIO* e o *endpoint* `/update_weights` de alguma maneira que fosse acessível remotamente (do computador em que o retreinamento ocorre). Para isso, utilizamos a tecnologia de túnel de portas *HTTP* da ferramenta *Ngrok*.[8]

Uso do modelo de linguagem (*LLM*): depois de obtermos a classificação da condição das folhas identificadas na imagem, precisamos pensar em como entregar os resultados de uma maneira interessante para o usuário final.

Desse modo, implementamos uma integração dos resultados do modelo de detecção com a *API* de modelos de linguagem da *OpenAI*. Isso foi feito por meio da criação de um *prompt* ao qual os resultados da tarefa de detecção de doenças é incorporado. Assim, o texto gerado pelo modelo de linguagem, que conterá informações sobre as doenças detectadas e dicas sobre como tratar a planta afetada é o retorno do *endpoint* `/process_image`. A estrutura do *prompt* usado pode ser consultada a seguir. Note que há condições que mudam a estrutura do *prompt*.

```
1    Se nenhuma folha de tomate foi detectada:
```

[8] https://ngrok.com/docs

```
2    "Não detectamos folhas de tomate na imagem."
3
4    Se apenas folhas saudáveis foram detectadas:
5    "De acordo com nosso algoritmo, seu tomateiro está saudável."
6
7    Caso contrário:
8    "A imagem de tamanho <image_width>x<image_height> pixels contém as seguintes
         doenças detectadas:
9
10   1. Doença: <class_name_1>
11     Confiança: <score_1>
12     Localização: De (<x1_1>, <y1_1>) até (<x2_1>, <y2_1>)
13
14   2. Doença: <class_name_2>
15     Confiança: <score_2>
16     Localização: De (<x1_2>, <y1_2>) até (<x2_2>, <y2_2>)
17
18   ...
19
20   Por favor, providencie conselhos sobre como controlar e tratar as doenças
         detectadas e sugestões para melhorar o crescimento e a saúde da planta de
         tomate no geral."
21
22   Se ocorrer um erro na consulta ao modelo:
23   "Erro ao consultar o LLM: <mensagem_de_erro>"
```

3.3 *Front-end*

3.3.1 Cliente comum

Para a interface de usuário geral, desenvolvemos duas páginas com a *framework React Next.js*, e utilizamos componentes da biblioteca *React MUI*. As duas páginas são:

1. página inicial
2. página de diagnóstico do tomateiro

Na primeira página, dividimos o site nas seguintes seções, como vistas na figura 3.3, para o usuário conhecer o propósito da ferramenta e como a utilizar:

- barra superior (*app bar*)
- destaque principal (*hero section*)
- *call to action* (leva o usuário a diagnosticar sua planta)
- como funciona o *TomatoHealth*
- por que usar nosso aplicativo
- rodapé

Na página de diagnóstico, como visto na figura 3.4: o usuário escolhe tirar uma foto, ou então fazer *upload* de sua galeria, para enviar ao servidor. O nosso *front-end* então exibe a mensagem do servidor: o output de uma *LLM*.

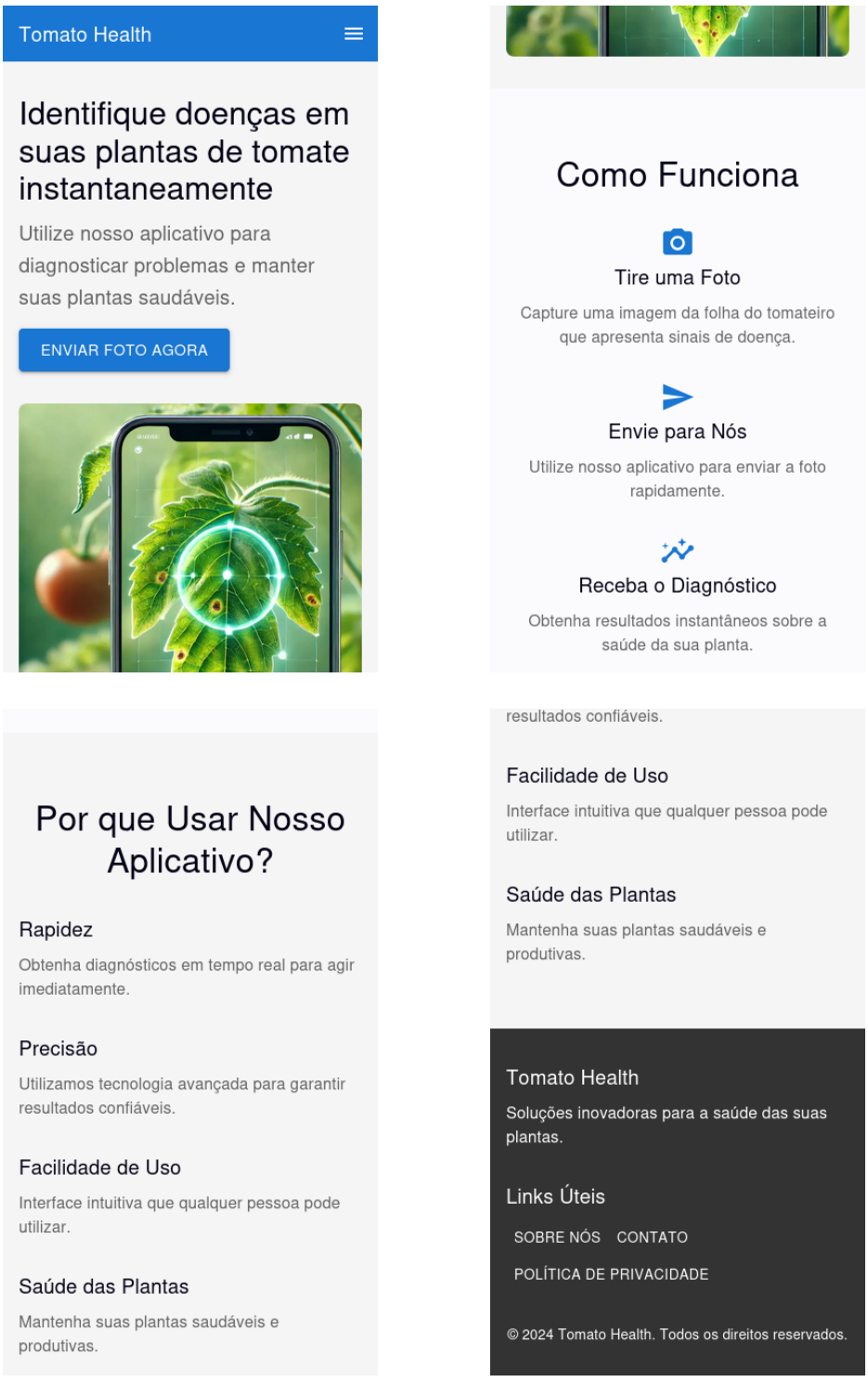


Figura 3.3: *Página inicial do TomatoHealth. Fonte da imagem utilizada: OpenAI, (2024), imagem gerada pelo modelo ChatGPT.*

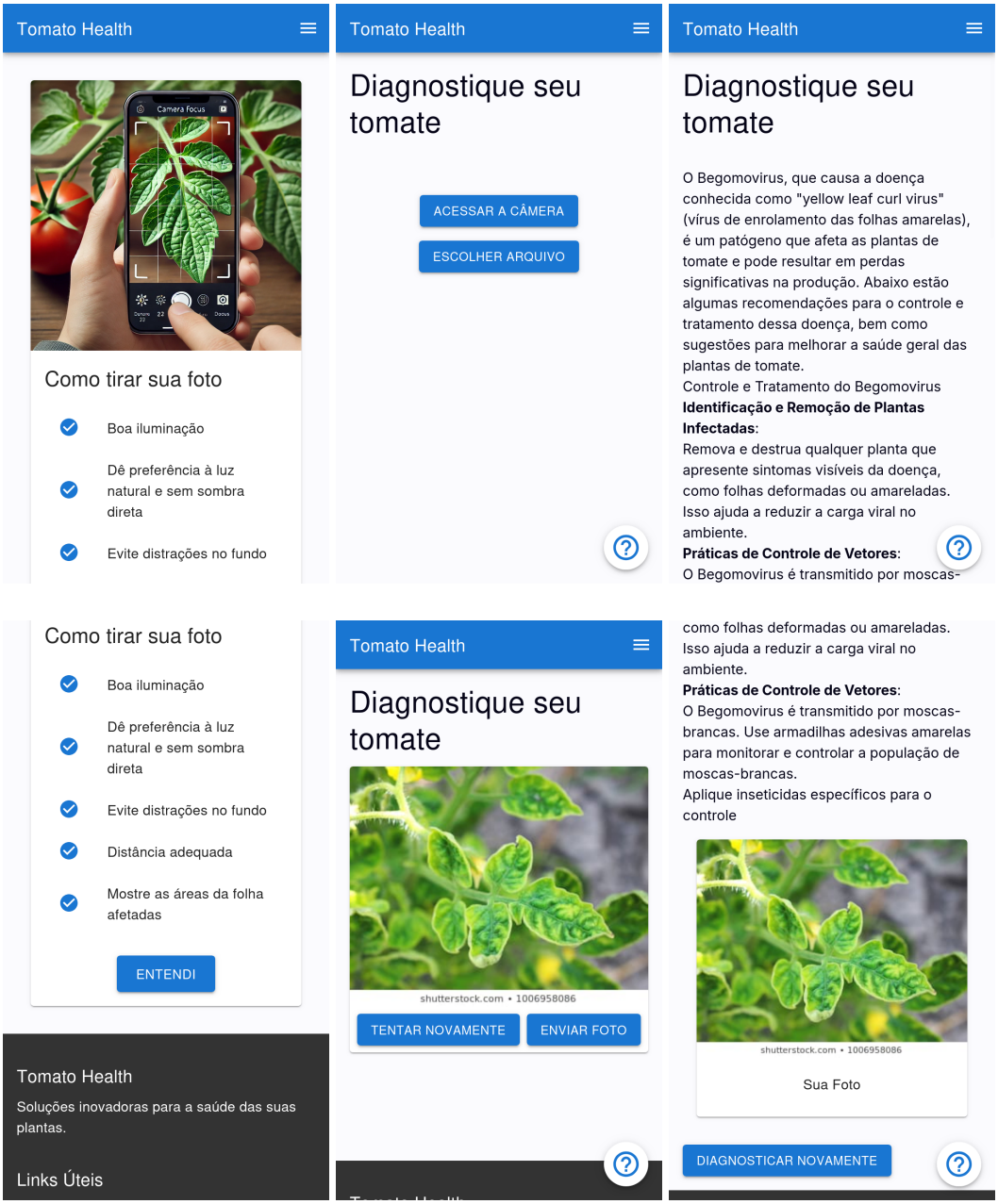


Figura 3.4: *Página de diagnóstico do TomatoHealth. Fonte da imagem utilizada: OpenAI, (2024), imagem gerada pelo modelo ChatGPT.*

3.3.2 Cliente especialista e ferramenta de anotação *Label Studio*

Na interface do usuário especialista, utilizamos o *software open-source* de rotulagem *Label Studio*. Segundo sua página no *Github*, "O Label Studio é uma ferramenta de rotulagem de dados de código aberto. Ela permite rotular tipos de dados como áudio, texto, imagens, vídeos e séries temporais com uma interface de usuário simples e direta e exportar para vários formatos de modelos. Ele pode ser usado para preparar dados brutos ou melhorar os dados de treinamento existentes para obter modelos de ML mais precisos".

Para não exibir detalhes desnecessários do sistema, modificamos seu código *front-end*

em *React*, de forma a excluir alguns componentes da página de anotação, como barras nas laterais e no topo da página, com a finalidade do usuário não visitar endereços não utilizadas pelo *TomatoHealth*.

Conforme a figura 3.5, o usuário especialista vê, por vez, uma única imagem que ele poderá demarcar classes de doenças com *bounding boxes*. Após ele efetuar a rotulagem, o banco de dados é atualizado com as novas informações e o usuário é redirecionado para a próxima imagem.

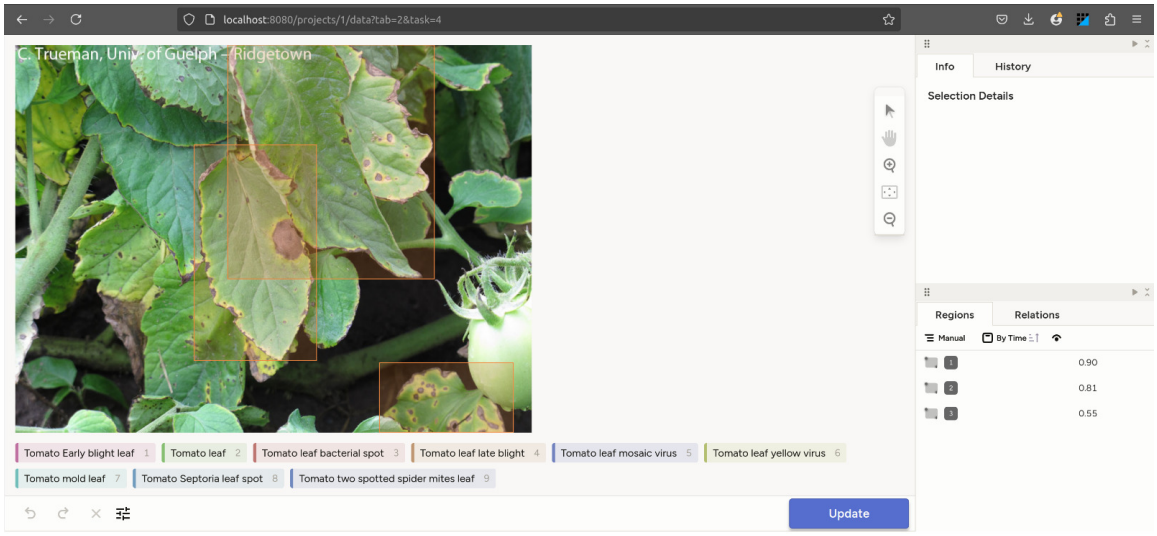


Figura 3.5: *Página de rotulagem do usuário especialista utilizando uma versão modificada do Label Studio. Fonte: acervo dos autores.*

Capítulo 4

Próximos passos

4.1 Integrando os *datasets*

Como comentado anteriormente, embora o *TomatoHealth* seja um projeto que tem o objetivo de construir um conjunto de dados robusto que possibilite o treinamento de modelos confiáveis e eficientes na tarefa de detecção de doenças em folhas de plantas, acreditamos que a performance inicial do modelo oferecido pela plataforma pode ser uma variável determinante para atrair um número considerável de usuários.

Com esse engajamento inicial, o objetivo do projeto se torna mais facilmente alcançável, pois esses primeiros usuários irão contribuir ativamente para o aumento da quantidade de imagens que poderão fazer parte das novas versões dos conjuntos de dados criados pelo *TomatoHealth*. Portanto, consideramos como um próximo passo do desenvolvimento do projeto, uma tarefa que aumenta a quantidade de dados disponíveis na primeira versão do *dataset* utilizado e, por isso, pode ser que produza um modelo inicialmente mais eficaz do que o atual.

Essa estratégia consiste em integrar o *dataset PlantVillage* com o conjunto de dados obtido a partir do processamento do *PlantDoc*. Apesar dos problemas previamente discutidos em relação ao *PlantVillage*, acreditamos que essa integração pode beneficiar a eficiência do modelo inicial do *TomatoHealth*. Essa abordagem é particularmente interessante porque o número de exemplos atualmente presentes no conjunto de dados do projeto é muito pequeno, enquanto o *PlantVillage* oferece uma quantidade consideravelmente maior de exemplos, o que pode contribuir para um treinamento mais robusto.

Na figura 4.1, observamos a distribuição de imagens por classes de doenças em folhas de tomate presentes no conjunto de dados *PlantVillage*. Note que as classes do *PlantDoc* representam um subconjunto das classes do *PlantVillage*. A classe `Tomato___Target_Spot`, embora não esteja presente no *PlantDoc*, poderia ser incorporada como uma possível adição futura.

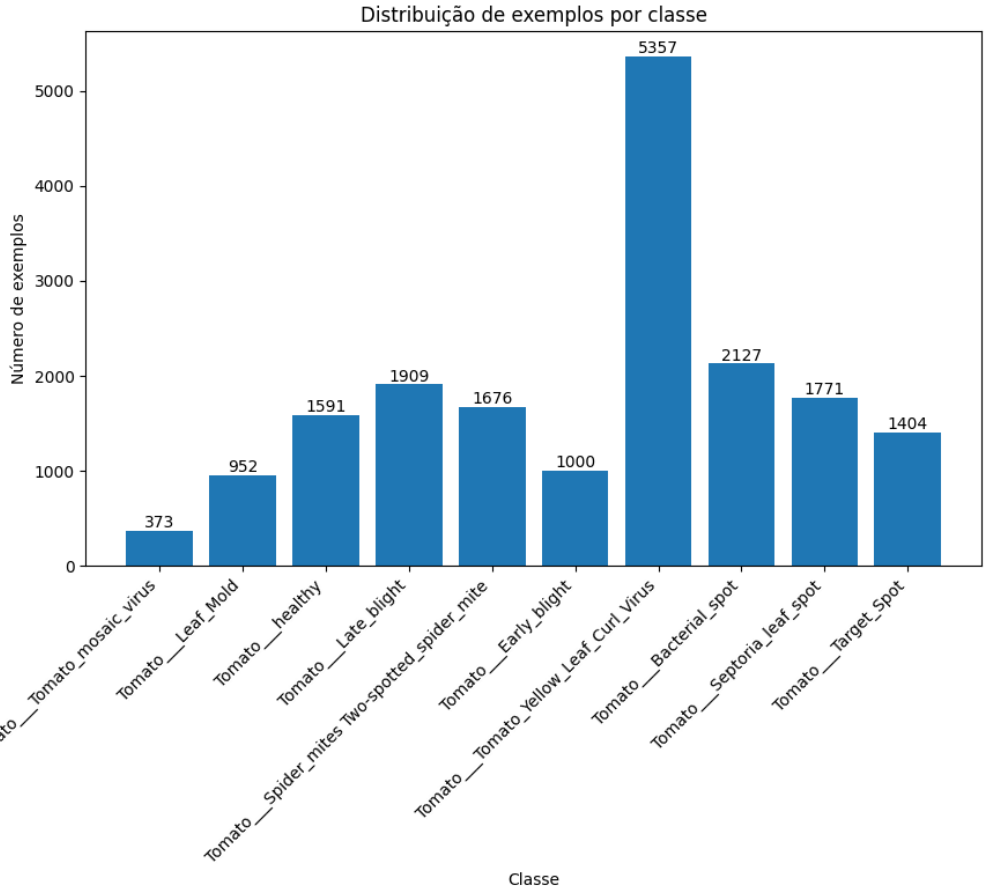


Figura 4.1: *Distribuição de exemplos por classes de tomate no PlantVillage. Fonte: acervo dos autores.*

Como o *PlantVillage* possui uma única folha por imagem e é destinado à tarefa de classificação, a ideia para fazer essa integração é utilizar a área inteira das imagens do *PlantVillage* como *bounding box*.

4.2 Busca de imagens por similaridade

Como comentado anteriormente, é importante que um conjunto de dados robusto não tenha duplicatas ou "quase-duplicatas" em seus dados. No sistema atual, não há uma maneira de assegurar que somente imagens distintas farão parte do conjunto de dados resultante. Nesse sentido, para que os conjuntos de dados produzidos pelo *TomatoHealth* não tenham dados repetidos, pensamos em um próximo passo que apresente as imagens mais semelhantes à que está sendo revisada. Assim, caso o usuário especialista interprete que ela é semelhante demais com imagens já presentes no conjunto de dados atual, será possível cancelar sua adição ao conjunto de dados atualizado.

Para fazer isso, pensamos no seguinte passo a passo:

1. **Produzir representações em alto nível de cada imagem que faz parte do conjunto de dados:** essas representações, também conhecidas como *embeddings*, não são nada mais que o resultado da extração de características de cada imagem. É

mais simples e eficiente armazenar e comparar uma imagem com as demais se ela for representada como um vetor de 512 dimensões, por exemplo, ao invés de uma matriz de *pixels* de $256 \times 256 \times 3$ dimensões.

2. **Armazenar esses *embeddings* em um banco de dados de vetores:** podemos utilizar modelos de redes convolucionais pré-treinadas no *dataset ImageNet*, boas em extração de características, para produzir os *embeddings* de todas as imagens no conjunto de dados e então armazenar essas extrações em um banco de dados de vetores. Isso é importante pois, assim, só precisaremos obter essas representações uma única vez. Seguindo essa lógica, depois da revisão de toda imagem por um usuário especialista, devemos produzir sua representação vetorial e armazená-la no banco de dados vetorial.
3. **Fazer busca por similaridade no banco de dados vetorial:** no momento em que um usuário especialista for fazer a revisão de uma nova imagem, devemos fazer a busca por similaridade com as imagens presentes no banco de dados vetorial e, por conseguinte, no conjunto de dados. Isso é uma tarefa recorrente e há diversas ferramentas que disponibilizam esse serviço. Podemos comentar sobre a *Pinecone*,[1] que inclusive disponibiliza uma busca pelos K vetores mais semelhantes ao examinado.
4. **Exibir as imagens mais semelhantes:** basta exibir as imagens mais semelhantes ao usuário especialista.

4.3 Mudança de *LLM*

Atualmente, como descrito no artigo, utilizamos uma chamada de *API* a um modelo de linguagem (`gpt-4o-mini`) disponibilizado pela *OpenAI* para gerar a resposta de diagnóstico ao usuário. No entanto, tal serviço é pago e adiciona dependências a serviços de terceiros ao projeto *TomatoHealth*. Dessa forma, é interessante trabalhar para que essa funcionalidade seja incorporada de maneira mais natural ao projeto, sem implicar em custos monetários e que seja independente de terceiros. Isso pode ser feito com a adição de um programa que rode um modelo de linguagem (*LLM*) como um *container* no ecossistema do *Docker Compose*. Para isso, poderíamos utilizar a ferramenta *HuggingFace Transformers*,[2] que torna simples o processo de inferência e treinamento de modelos de linguagem atuais.

[1] https://www.pinecone.io/learn/what-is-similarity-search/\#What-Are-Vector-Representations

[2] https://huggingface.co/docs/transformers/index

Capítulo 5

Conclusão

Ao longo desse trabalho, a importância do desenvolvimento de soluções baratas e eficientes para a detecção e diagnóstico de doenças na produção agrícola ficou evidente. A análise da situação atual dos conjuntos de dados abertos e das aplicações baseadas neles revelou limitações significativas, como problemas de qualidade e uma relativa escassez de dados, que comprometem a eficácia dessas iniciativas.

Então, fica claro que o desenvolvimento dessas soluções exige a criação de conjuntos de dados abertos, livres e robustos capazes de sustentar o desenvolvimento de aplicações voltadas ao diagnóstico e à detecção de doenças em plantas por meio de imagens. Essa abordagem beneficia os agricultores e a comunidade científica em geral.

O objetivo deste trabalho de formatura foi justamente implementar uma solução promissora para essa demanda: o sistema *TomatoHealth*. Um sistema que oferece uma interface intuitiva para que agricultores diagnostiquem doenças em suas plantas — inicialmente voltada para tomateiros, mas com potencial de expansão para outras culturas. Para efetuar essa tarefa, o *TomatoHealth* conta com um modelo de detecção de objetos (*YOLOv8*) que inicialmente foi treinado com um subconjunto do *dataset PlantDoc*.

O diferencial do *TomatoHealth* está na possibilidade de melhoria contínua do conjunto de dados. As imagens enviadas pelos usuários passam pela revisão de especialistas e, então, são incorporadas ao conjunto de dados original. Ainda, podemos utilizar o novo conjunto de dados para retreinar o modelo de detecção de objetos e melhorar o desempenho da ferramenta de diagnóstico do *TomatoHealth*. Esse processo melhora o desempenho do diagnóstico e contribui para a construção de um *dataset* mais representativo das condições reais, já que utiliza fotos capturadas por dispositivos móveis diretamente em campo.

Por fim, este trabalho busca transformar a forma como os conjuntos de dados são produzidos, promovendo o desenvolvimento de soluções mais acessíveis e abertas. Essa abordagem, além de democratizar o acesso à tecnologia, visa expandir o alcance do conhecimento e impulsionar o progresso científico de maneira inclusiva.

Referências

[Abu-Mostafa *et al.* 2012] Yaser S. Abu-Mostafa, Hsuan-Tien Lin e Malik Magdon-Ismail. *Learning From Data*. AMLBook, 2012 (citado na pg. 5).

[AI4Science e Quantum 2023] Microsoft Research AI4Science e Microsoft Azure Quantum. *The Impact of Large Language Models on Scientific Discovery: a Preliminary Study using GPT-4*. 2023. doi: 10.48550/ARXIV.2311.07361. url: https://arxiv.org/abs/2311.07361 (citado na pg. 1).

[Badue *et al.* 2019] C. Badue *et al.* "Self-driving cars: a survey". *ArXiv* abs/1901.04407 (2019). doi: 10.1016/j.eswa.2020.113816 (citado na pg. 1).

[Barz e Denzler 2019] Björn Barz e Joachim Denzler. "Do we train on test data? purging cifar of near-duplicates". *Journal of Imaging* 6 (2019). doi: 10.3390/jimaging6060041 (citado na pg. 6).

[Coletta *et al.* 2022] Andrea Coletta *et al.* "Optimal deployment in crowdsensing for plant disease diagnosis in developing countries". *IEEE Internet of Things Journal* 9 (2022), pp. 6359–6373. doi: 10.1109/jiot.2020.3002332 (citado na pg. 2).

[Cortes e Vapnik 1995] Corinna Cortes e V. Vapnik. "Support-vector networks". *Machine Learning* 20 (1995), pp. 273–297. doi: 10.1023/A:1022627411411 (citado na pg. 16).

[Dalal e Triggs 2005] N. Dalal e B. Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 886–893 vol. 1. doi: 10.1109/CVPR.2005.177 (citado na pg. 16).

[Deng *et al.* 2009] Jia Deng *et al.* "Imagenet: a large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848 (citado na pg. 22).

[Ge *et al.* 2021] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li e Jian Sun. *YOLOX: Exceeding YOLO Series in 2021*. 2021. doi: 10.48550/ARXIV.2107.08430. url: https://arxiv.org/abs/2107.08430 (citado na pg. 27).

[Geetharamani e Arun Pandian 2019] G. Geetharamani e J. Arun Pandian. "Identification of plant leaf diseases using a nine-layer deep convolutional neural network". *Computers & Electrical Engineering* 76 (2019), pp. 323–338. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2019.04.011. URL: https://www.sciencedirect.com/science/article/pii/S0045790619300023 (citado na pg. 2).

[Girshick 2015] Ross Girshick. "Fast r-cnn". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169 (citado na pg. 18).

[Girshick, Donahue *et al.* 2014] Ross Girshick, Jeff Donahue, Trevor Darrell e Jitendra Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81 (citado na pg. 17).

[Girshick, Donahue *et al.* 2016] Ross Girshick, Jeff Donahue, Trevor Darrell e Jitendra Malik. "Region-based convolutional networks for accurate object detection and segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.1 (2016), pp. 142–158. DOI: 10.1109/TPAMI.2015.2437384 (citado na pg. 17).

[Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (citado na pg. 2).

[He *et al.* 2014] Kaiming He, X. Zhang, Shaoqing Ren e Jian Sun. "Spatial pyramid pooling in deep convolutional networks for visual recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37 (2014), pp. 1904–1916. DOI: 10.1007/978-3-319-10578-9_23 (citado nas pgs. 17, 18).

[Hughes e Salathé 2015] David P. Hughes e Marcel Salathé. "An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing". *CoRR* abs/1511.08060 (2015). arXiv: 1511.08060. URL: http://arxiv.org/abs/1511.08060 (citado nas pgs. iii, v, 2).

[IPPC Secretariat 2021] IPPC Secretariat. *Scientific review of the impact of climate change on plant pests – A global challenge to prevent and mitigate plant pest risks in agriculture, forestry and ecosystems*. Rome: FAO on behalf of the IPPC Secretariat, 2021. DOI: https://doi.org/10.4060/cb4769en (citado nas pgs. iii, v, 1).

[King 2023] Range King. *Brief summary of YOLOv8 model structure*. Accessed: 2023-10-10. 2023. URL: https://github.com/ultralytics/ultralytics/issues/189 (citado na pg. 28).

[Klemmer 2016] Scott R. Klemmer. *Lecture 8: Storyboarding*. Adapted from Amal Dar Aziz, Guide to Storyboarding. 2016 (citado na pg. 8).

[Li *et al.* 2020] Xiang Li *et al. Generalized Focal Loss: Learning Qualified and Distributed Bounding Boxes for Dense Object Detection.* 2020. DOI: 10.48550/ARXIV.2006.04388. URL: https://arxiv.org/abs/2006.04388 (citado na pg. 28).

[Lin *et al.* 2014] Tsung-Yi Lin *et al. Microsoft COCO: Common Objects in Context.* 2014. DOI: 10.48550/ARXIV.1405.0312. URL: https://arxiv.org/abs/1405.0312 (citado na pg. 31).

[Mahlein 2016] Anne-Katrin Mahlein. "Plant disease detection by imaging sensors – parallels and specific demands for precision agriculture and plant phenotyping". *Plant Disease* 100.2 (2016). PMID: 30694129, pp. 241–251. DOI: 10.1094/PDIS-03-15-0340-FE. eprint: https://doi.org/10.1094/PDIS-03-15-0340-FE. URL: https://doi.org/10.1094/PDIS-03-15-0340-FE (citado na pg. 1).

[Minaee *et al.* 2024] Shervin Minaee *et al. Large Language Models: A Survey.* 2024. DOI: 10.48550/ARXIV.2402.06196. URL: https://arxiv.org/abs/2402.06196 (citado na pg. 1).

[Mohanty *et al.* 2016] Sharada P. Mohanty, David P. Hughes e Marcel Salathé. "Using deep learning for image-based plant disease detection". *Frontiers in Plant Science* 7 (2016). ISSN: 1664-462X. DOI: 10.3389/fpls.2016.01419. URL: https://www.frontiersin.org/articles/10.3389/fpls.2016.01419 (citado na pg. 2).

[MTheiler 2019] MTheiler. *Bounding boxes in object detection.* https://commons.wikimedia.org/w/index.php?curid=75843378. CC BY-SA 4.0. 2019 (citado na pg. 15).

[Nielsen 2015] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015 (citado na pg. 2).

[Noyan 2022] Mehmet Alican Noyan. *Uncovering bias in the PlantVillage dataset.* 2022. eprint: arXiv:2206.04374 (citado na pg. 2).

[Padilla *et al.* 2020] Rafael Padilla, S. L. Netto e Eduardo A. B. da Silva. "A survey on performance metrics for object-detection algorithms". *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)* (2020), pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130 (citado nas pgs. 24, 25).

[Redmon *et al.* 2016] Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi. "You only look once: unified, real-time object detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91 (citado nas pgs. 18, 19, 21).

[Shorten e Khoshgoftaar 2019] Connor Shorten e T. Khoshgoftaar. "A survey on image data augmentation for deep learning". *Journal of Big Data* 6 (2019), pp. 1–48. DOI: 10.1186/s40537-019-0197-0 (citado nas pgs. 5, 6).

[Shroff 2021] Megha Shroff. *Know your Neural Network Architecture More by Understanding These Terms*. Accessed: 2024-11-01. 2021. URL: https://medium.com/@shroffmegha6695/know-your-neural-network-architecture-more-by-understanding-these-terms-67faf4ea0efb (citado na pg. 26).

[Siddiqua *et al.* 2022] Ayesha Siddiqua, Muhammad Ashad Kabir, Tanzina Ferdous, Israt Bintea Ali e Leslie A. Weston. "Evaluating plant disease detection mobile applications: quality and limitations". *Agronomy* 12.8 (ago. de 2022), p. 1869. ISSN: 2073-4395. DOI: 10.3390/agronomy12081869. URL: http://dx.doi.org/10.3390/agronomy12081869 (citado na pg. 2).

[D. Singh *et al.* 2020] Davinder Singh *et al.* "Plantdoc: a dataset for visual plant disease detection". In: *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*. CoDS COMAD 2020. ACM, jan. de 2020. DOI: 10.1145/3371158.3371196. URL: http://dx.doi.org/10.1145/3371158.3371196 (citado na pg. 14).

[V. Singh e Misra 2017] Vijai Singh e A.K. Misra. "Detection of plant leaf diseases using image segmentation and soft computing techniques". *Information Processing in Agriculture* 4.1 (2017), pp. 41–49. ISSN: 2214-3173. DOI: https://doi.org/10.1016/j.inpa.2016.10.005. URL: https://www.sciencedirect.com/science/article/pii/S2214317316300154 (citado na pg. 2).

[*Tecnologias impulsionam produção nacional do tomate* 2022] *Tecnologias impulsionam produção nacional do tomate*. Out. de 2022. URL: https://www.revistarural.com.br/2022/10/14/tecnologias-impulsionam-producao-nacional-do-tomate/ (acesso em 30/11/2024) (citado nas pgs. iii, v).

[Terven *et al.* 2023] Juan Terven, Diana-Margarita Córdova-Esparza e Julio-Alejandro Romero-González. "A comprehensive review of yolo architectures in computer vision: from yolov1 to yolov8 and yolo-nas". *Machine Learning and Knowledge Extraction* 5.4 (2023), pp. 1680–1716. ISSN: 2504-4990. DOI: 10.3390/make5040083. URL: https://www.mdpi.com/2504-4990/5/4/83 (citado nas pgs. 24, 27, 28).

[Thakur *et al.* 2022] Poornima Singh Thakur, Pritee Khanna, Tanuja Sheorey e Aparajita Ojha. *Explainable vision transformer enabled convolutional neural network for plant disease identification: PlantXViT*. 2022. DOI: 10.48550/ARXIV.2207.07919. URL: https://arxiv.org/abs/2207.07919 (citado na pg. 2).

[Ultralytics 2023a] Ultralytics. *YOLOv8 Models Documentation*. Accessed: 2023-10-10. 2023. URL: https://docs.ultralytics.com/models/yolov8/#__tabbed_1_1 (citado na pg. 28).

[Ultralytics 2023b] Ultralytics. *YOLOv8: A State-of-the-Art Object Detection Model*. Accessed: 2024-10-30. 2023. URL: https://github.com/ultralytics/ultralytics (citado nas pgs. 18, 27).

[Ultralytics 2024] Ultralytics. *Ultralytics YOLO11*. Accessed: 2024-10-30. 2024. URL: https://github.com/ultralytics/ultralytics (citado na pg. 18).

[Viola e Jones 2001] P. Viola e M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517 (citado na pg. 16).

[Yao *et al.* 2023] Jianping Yao, Son N. Tran, Samantha Sawyer e Saurabh Garg. "Machine learning for leaf disease classification: data, techniques and applications". *Artificial Intelligence Review* 56.S3 (out. de 2023), pp. 3571–3616. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10610-4. URL: http://dx.doi.org/10.1007/s10462-023-10610-4 (citado nas pgs. iii, v, 2, 7).

[Zhao *et al.* 2018] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu e Xindong Wu. "Object detection with deep learning: a review". *IEEE Transactions on Neural Networks and Learning Systems* 30 (2018), pp. 3212–3232. DOI: 10.1109/TNNLS.2018.2876865 (citado na pg. 15).

[Zheng *et al.* 2020] Zhaohui Zheng *et al.* "Distance-iou loss: faster and better learning for bounding box regression". *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (abr. de 2020), pp. 12993–13000. ISSN: 2159-5399. DOI: 10.1609/aaai.v34i07.6999. URL: http://dx.doi.org/10.1609/aaai.v34i07.6999 (citado na pg. 28).

[Zou *et al.* 2019] Zhengxia Zou, Zhenwei Shi, Yuhong Guo e Jieping Ye. "Object detection in 20 years: a survey". *Proceedings of the IEEE* 111 (2019), pp. 257–276. DOI: 10.1109/JPROC.2023.3238524 (citado nas pgs. viii, 15–18).