ESCOLA SALESIANA SÃO JOSÉ

CENTRO PROFISSIONAL DOM BOSCO - CPDB

Leonardo Adriano da Silva

Matheus Mancilha Marinho

Miguel Nascimento Venzel

DETECÇÃO DE PRAGAS NA AGRICULTURA ATRAVÉS DA VISÃO COMPUTACIONAL EM TEMPO REAL

CAMPINAS
2024

Leonardo Adriano da Silva
Matheus Mancilha Marinho
Miguel Nascimento Venzel

DETECÇÃO DE PRAGAS NA AGRICULTURA ATRAVÉS DA VISÃO COMPUTACIONAL EM TEMPO REAL

Relatório referente ao Projeto de Conclusão de Curso, apresentado ao Centro Profissional Dom Bosco da Escola Salesiana São José, como parte dos requisitos necessários à obtenção do título de Técnico em Informática.

Orientador: Adriana Maia da Silva Coelho

Coorientador: Geraldo Moreno Florentino Junior

CAMPINAS
2024

AGRADECIMENTOS

Os meus colegas de curso, quem convivi intensamente durante os últimos anos, pelo companheirismo e pela troca de experiências, que foram fundamentais para o crescimento pessoal e acadêmico. Aos professores, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional.

“Agricultura e tecnologia estão atreladas. Hoje, num caminho inverso, as inovações tecnológicas é que ajudam as plantações e produtores.”

Adriano Naspolini

RESUMO

A agricultura enfrenta desafios significativos, especialmente no combate às pragas, que impactam negativamente as plantações. Agricultores de pequeno e médio porte frequentemente não têm acesso a tecnologias avançadas devido ao seu alto custo, resultando em um uso elevado de agrotóxicos para combater pestes. Este projeto foi desenvolvido com o objetivo de oferecer uma alternativa acessível para esses agricultores, auxiliando no monitoramento e combate a doenças em suas plantações, o que, por sua vez, irá ajudar na redução do uso de agrotóxicos. A solução proposta é voltada principalmente para pequenos e médios agricultores, mas também pode beneficiar grandes produtores ao proporcionar economia financeira. No projeto, foi utilizado um drone simples, equipado com uma câmera de qualidade aceitável, para sobrevoar as plantações. O drone está integrado a uma inteligência artificial, cuja rede neural é o YOLOv8, treinada para detectar quatro doenças comuns em plantações de café: Mancha de Olho Pardo, Bicho-Mineiro, Ferrugem das Folhas e Ácaro Vermelho. Para facilitar a visualização dos dados, foi criado um aplicativo em Python, utilizando o framework Kivy, onde o produtor pode visualizar um mapa com as localizações das doenças na plantação. As coordenadas das áreas afetadas são armazenadas em um banco de dados e, com a ajuda da biblioteca Folium, é criado um mapa com as marcações das doenças nele. O drone pode operar de forma manual e autônoma seguindo rotas predefinidas no aplicativo RX Drone, que utiliza a funcionalidade de marcar pontos no mapa. Essa funcionalidade é possível graças ao GPS integrado do drone.

Palavras-chave: Inteligência Artificial, Monitoramento de Doenças, Drone.

ABSTRACT

Agriculture faces significant challenges, particularly in combating pests that negatively impact crops. Small and medium-sized farmers often lack access to advanced technologies due to their high cost, resulting in increased use of pesticides. This project aims to provide an accessible alternative for these farmers by assisting in the monitoring and control of plant diseases, which in turn will help reduce pesticide use. The proposed solution primarily targets small and medium-sized farmers but can also benefit large producers by offering financial savings. The project utilizes a simple drone equipped with a reasonably good quality camera to survey the fields. This drone is integrated with artificial intelligence, specifically the YOLOv8 neural network, trained to detect four common diseases in coffee plantations: Brown Spot, Leaf Miner, Leaf Rust, and Red Mite. To facilitate data visualization, a Python application using the Kivy framework was developed, allowing producers to view a map highlighting disease locations within their plantations. A database stores the coordinates of affected areas, and the Folium library is used to create a map with disease markers. The drone can operate both manually and autonomously, following predefined routes in the RX Drone application, which includes functionality to mark points on the map. This capability is made possible by the drone's integrated GPS.

Keywords: Artificial Intelligence, Disease Monitoring, Drone.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ANAC – Agência Nacional de Aviação Civil
C2F – Cross Stage Partial Fusion
CNN – Convolutional Neural Network
FPN – Feature Pyramid Network
GPS – Global Positioning System
IA – Inteligência Artificial
IBGE – Instituto Brasileiro de Geografia e Estatística
JSON – JavaScript Object Notation
NMS – Non Maximum Suppression
PAN – Path Aggregation Network
SiLU – Sigmoid Linear Unit
SPPF – Spatial Pyramid Pooling Fast
YOLO – You Only Look Once

SUMÁRIO

1. INTRODUÇÃO

Este projeto é fundamentado na utilização de drones e Inteligência Artificial como uma solução acessível e eficiente para um dos desafios primordiais enfrentados pelos produtores do agronegócio: o controle de pragas nas plantações. Ao explorar as dificuldades enfrentadas por esses agricultores, surgiu a ideia de desenvolver um algoritmo de Inteligência Artificial que aplicado em um drone sobrevoa as plantações analisando-as em tempo real. Essa análise possibilita a identificação das doenças presentes nas plantas, possibilitando intervenções rápidas e eficazes.

O grande diferencial desse sistema reside não apenas na sua eficiência, mas também no seu preço acessível. O projeto tem a preocupação de tornar a tecnologia disponível para os pequenos e médios agricultores, muitas vezes limitados por recursos financeiros, porém isso não impede que os grandes agricultores adquiram essa tecnologia, pois como seu preço não é alto eles alcançarão uma redução de custos. Ao oferecer uma solução de baixo custo, a tecnologia no setor agrícola se torna viável a todos e contribuí para a modernização das práticas agrícolas em todas as escalas.

Com o mapeamento da plantação e o fornecimento de informações detalhadas sobre as pragas e doenças, esse sistema permite aos agricultores tomarem medidas direcionadas para o controle desses problemas. Isso não apenas reduzirá a necessidade de aplicação generalizada de agrotóxicos e pesticidas, mas também promoverá uma agricultura mais sustentável com menos danos ao meio ambiente.

Portanto, este projeto propõe uma solução eficiente e econômica para o controle de pragas e doenças nas plantações. Ao combinar drones e inteligência artificial, beneficiará os agricultores, tornando essa tecnologia mais disponível, sustentável e inclusiva.

2. MATERIAIS E MÉTODOS

A metodologia de pesquisa utilizada nesse projeto foi a de Engenharia, onde foram usadas as técnicas de pesquisas bibliográficas. Os conceitos analisados foram: a situação em que pequenos e médios agricultores se encontram. Buscou-se quais tecnologias de precisão eram mais acessíveis a eles. Como material principal bibliográfico, foi utilizado “Estado atual da agricultura digital no Brasil: Inclusão dos agricultores familiares e pequenos produtores rurais” (2021). Os principais autores que contribuíram com o trabalho foram: Antônio Márcio Buainain, Pedro Cavalcante e Letícia Consoline (2021).

Momentaneamente o drone observado para se usar no projeto (o modelo L900 Pro) possui uma câmera, com resolução HD, o suficiente para a necessidade do projeto na questão da qualidade de imagem e vídeo para as detecções que são feitas das plantações.

O drone também possui uma bateria com capacidades baixas, mas utilizável para os voos de treinamento, além disso seu baixo preço traz um custo-benefício comparando o que ele entrega e a suas funcionalidades no projeto.

O Google Colab e o Visual Studio foram usados como editores de código, onde foi possível fazer a programação da IA, a conexão com o drone, desenvolver o aplicativo, entre outras tarefas. O Google Colab é uma ferramenta muito utilizada para edição e teste de programas em Python, pois permite acesso a outros projetos feitos em Python e possibilita rodar a inteligência artificial na nuvem.

Para o voo autônomo foi utilizado o RX Drone, aplicativo do drone onde ele possuiu uma função que permite o usuário marcar pontos no mapa e assim o drone irá voar de maneira autônoma seguindo o trajeto definido pelo usuário, o aplicativo também teve a utilidade na parte do mapeamento das áreas com doenças pois através dele é possível extrair as coordenadas do drone quando ele detectar as doenças.

O Kivy é uma biblioteca em Python que é voltada para o desenvolvimento de aplicativos móveis ou desktop, essa biblioteca é Open Source, ou seja, código aberto e ela irá ajudar na parte de programar o aplicativo onde o usuário consegue ver onde há doenças na plantação, rota de voo e entre outras utilidades.

Para realizar o espelhamento da tela do celular com o drone, é utilizado uma ferramenta de código aberto chamado Scrcpy, onde ativando a depuração USB do celular é possível espelhar a tela do celular no computador.

Outra ferramenta utilizada foi o mapa do Estado de São Paulo no formato OpenStreetMap vector tile esse mapa baixado vem em um formato de arquivo de banco de dados para armazenar tiles de mapas. Através do site Map Tiler DATA foi possível baixar os arquivos do mapa. Este tipo de tile de mapa possibilita a criação e o acesso a mapas em Python sem a necessidade de estar conectado à internet.

Como as tiles do mapa são armazenadas em um tipo de arquivo específico para bancos de dados, foi necessário converter esse formato para PNG. Assim, outro material utilizado no projeto foi o script mbtilesToPngs.py, desenvolvido em Python, que realiza a conversão de arquivos mbtiles para uma estrutura de pastas contendo os arquivos do mapa no formato PNG.

2.1. Inteligência Artificial

O YOLOv8 é a oitava versão do modelo de Inteligência Artificial YOLO, uma rede neural convolucional projetada para detecção de objetos. Esses modelos são amplamente utilizados em aplicações que envolvem visão computacional. A rede neural YOLO foi desenvolvida pela empresa Ultralytics, que também é responsável por outras versões do YOLO, como o YOLOv4 e o YOLOv5, entre outras. A Ultralytics oferece uma biblioteca em Python que inclui funções e métodos para rodar e treinar o modelo YOLOv8, tornando-o uma ferramenta mais simples de usar e treinar em comparação com outros modelos de IA voltados para visão computacional.

As principais características do YOLOv8 que o destacam são sua velocidade e precisão, pois este modelo é relativamente leve, permitindo escolher o tamanho e a capacidade dos modelos pré-treinados. Outros pontos positivos relacionados ao YOLOv8 incluem sua arquitetura eficiente, que proporciona detecção com alta eficiência e velocidade; sua versatilidade, que permite detectar diferentes tipos de objetos; e, por fim, sua facilidade de uso, já que, como mencionado, o YOLO é mais fácil de utilizar em comparação com outros modelos de inteligência artificial.

2.1.1 Estrutura do YOLO

A versão 8 do YOLO (YOLOv8) é uma rede neural avançada projetada para suportar uma variedade de tarefas de visão computacional, incluindo detecção de objetos, segmentação de imagens, obturação, classificação e estimativa de pose. YOLOv8 é uma rede neural convolucional que realiza a previsão de caixas delimitadoras e a probabilidade de classe em uma única avaliação, oferecendo eficiência e precisão em diferentes tarefas.

Entre as principais inovações do YOLOv8 estão a ausência de âncoras, a previsão de um número reduzido de caixas e um processo de supressão não máxima (NMS) otimizado para maior rapidez. A arquitetura do YOLOv8 incorpora novas características, como a Feature Pyramid Network (FPN) e a Path Aggregation Network (PAN), que melhoram a capacidade de detecção e segmentação. O modelo é desenvolvido em PyTorch e é compatível tanto com CPUs quanto com GPUs, permitindo flexibilidade na implementação.

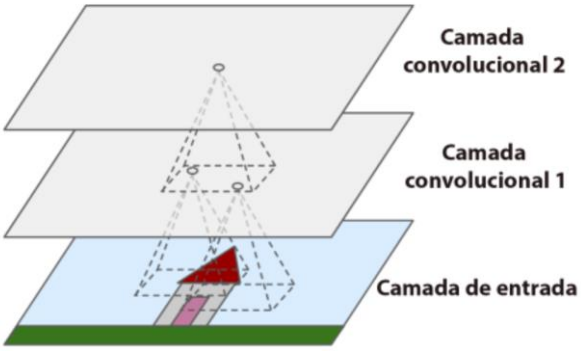
O YOLOv8 representa uma evolução das versões anteriores da série YOLO, integrando novas funcionalidades e aprimoramentos que visam aumentar a precisão e a eficiência em tarefas de visão computacional.

2.1.1.1 Redes Convolucionais

O termo "convolucionais" refere-se à operação de convolução, que é fundamental no processamento de dados em redes neurais convolucionais. As redes neurais convolucionais (CNNs) são amplamente empregadas na análise de imagens. Uma rede convolucional é uma rede com múltiplas camadas projetadas especificamente para reconhecer formas bidimensionais com alta invariância em relação à translação, escalamento, inclinação e outras formas de distorção.

Em uma rede neural convolucional, as camadas mais básicas identificam características simples, como bordas e cores. À medida que se avança para camadas mais profundas, essas informações simples são combinadas para identificar padrões mais complexos, como formas e texturas. Dessa forma, a rede convolucional se destaca pela presença de camadas especializadas que identificam padrões de maneira hierárquica.

Figura 1: Camadas da CNN com seus campos receptivos



Fonte: Adaptado de Géron, 2019

O reconhecimento dos padrões ocorre de forma escalonada: pixels se combinam em arestas, formando padrões que são agrupados para formar partes de objetos, e esses objetos, por sua vez, constituem cenas. Ao analisar uma imagem, a camada inicial da rede é examinada primeiro. Cada neurônio (ou unidade de processamento) dessa camada está atento apenas a uma pequena parte da imagem; por exemplo, se o campo receptivo do neurônio é de 3 pixels, isso indica que ele está observando uma área de 3 pixels na imagem original. À medida que se avança para camadas mais profundas, cada neurônio começa a observar uma área maior da imagem. Isso ocorre porque, conforme se sobe na hierarquia das camadas, os neurônios da camada seguinte recebem informações de áreas maiores da imagem original. Assim, um neurônio na camada mais profunda pode estar analisando uma área consideravelmente maior da imagem, combinando informações de diversas partes da imagem que foram analisadas anteriormente.

Cada campo receptivo atua como um filtro em busca de padrões específicos. Durante o treinamento da rede, a CNN ajusta os pesos desses filtros para que os neurônios se ativem ao detectar padrões específicos, como bordas ou texturas, dentro do campo receptivo. Esses pesos são ajustados para garantir que a rede aprenda a identificar e destacar características importantes na imagem.

A operação de convolução, que é o processo central, consiste em aplicar esses filtros de forma eficiente. À medida que a imagem é processada pelas camadas convolucionais, a rede destaca diferentes padrões e características, como bordas e texturas. Isso não só ajuda na identificação dos elementos visuais importantes, mas também reduz a complexidade da imagem e a quantidade de dados que a rede precisa

processar, tornando o modelo mais eficiente e ajudando-o a generalizar melhor para novas imagens.

Essas técnicas são extremamente úteis em tarefas como o reconhecimento de objetos, pois a rede aprende a extrair e interpretar características relevantes de maneira eficaz. Além disso, contribuem para a transferência de aprendizagem, na qual um modelo treinado em uma tarefa pode aplicar o conhecimento adquirido a novas tarefas relacionadas, aproveitando o aprendizado prévio sobre a identificação de padrões em imagens.

2.1.1.2 Backbone, Neck e Head

O backbone, conhecido como extrator de recursos, é responsável por extrair características significativas da entrada. Este componente captura padrões simples nas camadas iniciais, como bordas e texturas, e pode ter várias escalas de representação à medida que avança, capturando características de diferentes níveis de abstração. Assim, fornece uma representação rica e hierárquica da entrada.

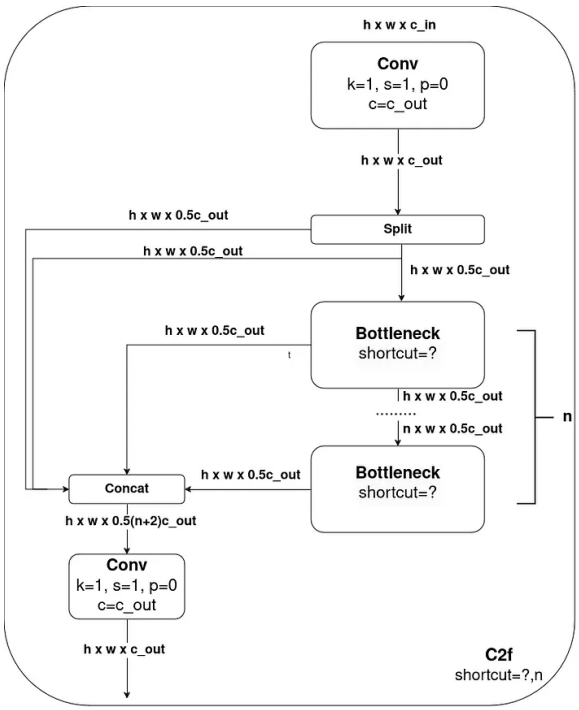
O YOLOv8 utiliza um backbone CSPDarknet53, semelhante ao YOLOv5, porém com algumas modificações no CSPLayer, agora denominado módulo C2f. O módulo C2f, que incorpora um gargalo parcial entre estágios com duas convoluções, substitui o CSPLayer utilizado no YOLOv5 e combina recursos de alto nível com informações contextuais, resultando em uma melhoria na precisão da detecção. Adicionalmente, uma camada de agrupamento rápido de pirâmide espacial (SPPF) acelera a computação, agrupando recursos em um mapa de tamanho fixo. Cada convolução é acompanhada de normalização em lote e ativação SiLU.

O CSPDarknet53, utilizado como backbone, é uma arquitetura rede neural convolucional projetada para detecção de objetos, fundamentada na arquitetura DarkNet-53. Essa rede emprega uma estratégia CSPNet que particiona o mapa de recursos da camada base em duas partes, mesclando-as por meio de uma hierarquia entre estágios. A implementação da estratégia de divisão e mesclagem possibilita um fluxo mais eficiente de gradientes pela rede, contribuindo para a melhoria do desempenho do modelo.

A arquitetura YOLOv8 utiliza um novo módulo C2f em vez da tradicional arquitetura de rede neural Feature Pyramid Network (FPN). Esse módulo combina características semânticas de alto nível com informações espaciais de baixo nível, resultando em uma precisão de detecção aprimorada, especialmente para objetos

pequenos. O C2F é uma abordagem inovadora que mescla características extraídas de diferentes níveis da rede, melhorando a capacidade de detecção de objetos, destacando-se pela sua estrutura única que facilita a fusão de informações de forma mais eficiente.

Figura 2: Bloco C2F



Fonte: medium.com

Na imagem do bloco C2f, um mapa de características resultante da convolução inicial é dividido em duas partes distintas: uma parte vai para o bloco Bottleneck, enquanto a outra segue diretamente para o bloco Concat. Essa combinação de informações provenientes de diferentes caminhos de processamento é benéfica para preservar e fundir características importantes. O bloco Bottleneck é projetado para reduzir a dimensionalidade, permitindo que a rede aprenda representações mais compactas e eficientes, enquanto o bloco Concat desempenha um papel crucial na fusão das informações.

A operação de divisão (Split) no bloco C2f separa o mapa de características resultante em duas partes. A camada de convolução (Conv) é uma operação que aplica filtros (kernels) ao mapa de características de entrada para extrair características significativas. O cálculo "h vezes w vezes c_in" representa a altura, largura e canais de entrada, sendo essencial para determinar o tamanho da entrada

que a rede processará. Por fim, o cálculo no último bloco convolucional, "h vezes w vezes 0.5 vezes (n+2)c_out", fornece uma medida da dimensão do mapa de características resultante que será alimentada na próxima etapa de processamento na rede YOLOv8. No bloco C2f, o número de blocos Bottleneck utilizados é definido pelo parâmetro depth_multiple do modelo, que determina a profundidade da rede e o número de blocos Bottleneck. Ao final do processamento, os mapas de características do bloco Bottleneck e do bloco Concat são concatenados e inseridos em um bloco convolucional final.

O neck realiza operações de fusão de características, integrando informações contextuais ao montar pirâmides de características. Ele coleta mapas de características de diferentes estágios do backbone e reduz a resolução espacial e a dimensionalidade das características para facilitar o processamento. Embora essa redução aumente a velocidade do modelo, pode comprometer sua qualidade. Além disso, o neck concatena características de diferentes escalas, garantindo que a rede consiga detectar objetos de tamanhos variados, enquanto integra informações contextuais para melhorar a precisão da detecção, considerando o contexto mais amplo da cena.

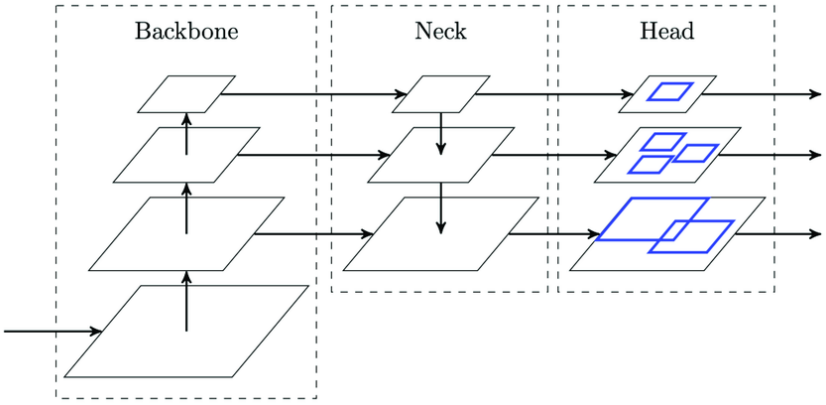
O neck é responsável por aumentar a amostragem do mapa de características, combinando as características adquiridas das várias camadas da seção backbone. Nessa seção, o neck aumenta o mapa de recursos em dobro, sem alterar o canal de saída. Ele contém operações para ajustar as resoluções e características em diferentes escalas, o que tem grande influência na detecção de objetos pequenos e grandes com a mesma eficiência.

Dentre as operações do neck, destaca-se o downsampling, que reduz a resolução dos mapas de características em determinados pontos, permitindo que a rede capture informações de contexto global, como grandes áreas da imagem. Por outro lado, o upsampling aumenta a resolução dos mapas de características, auxiliando a rede na detecção de objetos menores e mais detalhados.

O head é a parte final da rede, responsável por gerar as saídas, como as caixas delimitadoras e as pontuações de confiança para a detecção de objetos. Ou seja, o head desenha as caixas ao redor dos objetos detectados pelo YOLO, atribui uma porcentagem de certeza sobre a detecção e classifica os objetos dentro das caixas de acordo com suas respectivas classes.

Na arquitetura YOLOv8, a cabeça de detecção foi separada da cabeça de classificação. Essa separação envolve o cálculo da perda e a filtragem das caixas delimitadoras nos objetos detectados. As caixas de predição do YOLOv8 são formadas por cabeças desacopladas, que preveem simultaneamente as pontuações de classificação e as coordenadas de regressão. As pontuações de classificação são representadas por uma matriz bidimensional, indicando a presença de um objeto em cada pixel da imagem. Por sua vez, as coordenadas de regressão são representadas por uma matriz quadrimensional, que indica o desvio do centro do objeto em relação a cada pixel.

Figura 3: A arquitetura do modelo do YOLO abordando o Backbone, Neck e Head

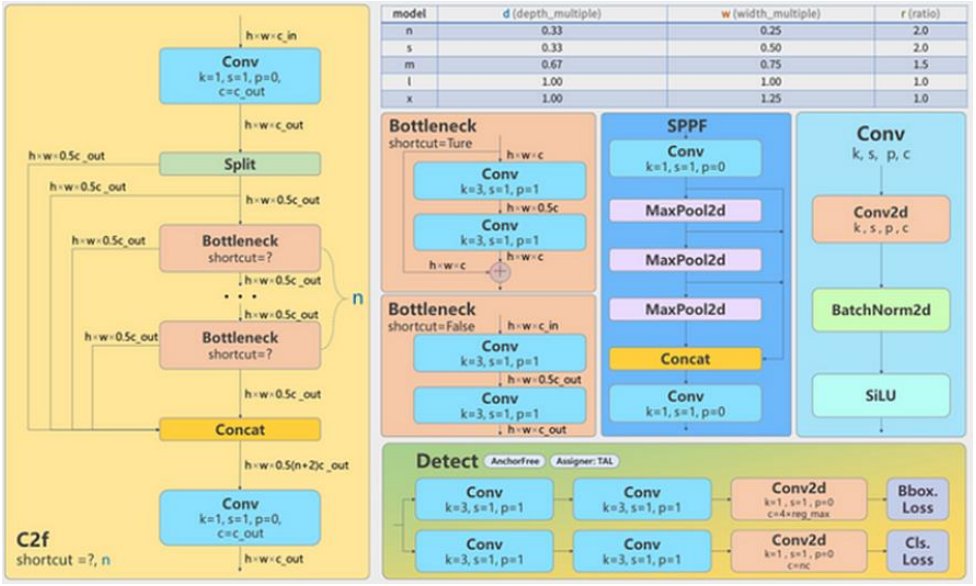


Fonte: researchgate.net

Resumidamente a arquitetura do YOLO funciona com o backbone extraindo os recursos da camada da entrada, o neck aprimora e integra as informações e monta uma pirâmide de características e o head mostrando as caixas de detecção de objetos.

2.1.1.3 Visualização da Arquitetura do YOLO

Figura 4: Principais Blocos da Estrutura do YOLOv8



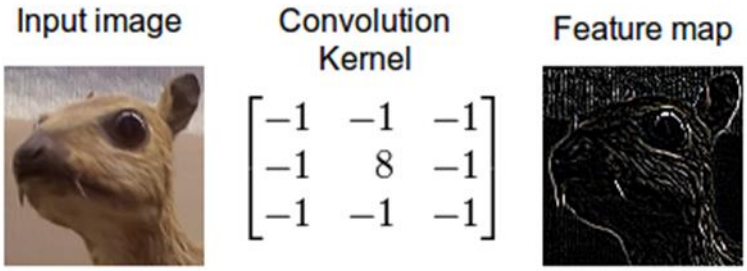
Fonte: medium.com

A arquitetura YOLO adota a abordagem de analisar características locais, em vez de analisar a imagem como um todo. O objetivo dessa abordagem é a otimização, pois reduz o esforço computacional e contribui para a detecção em tempo real. A velocidade de detecção é alta, permitindo a análise de múltiplos quadros por segundo, ou seja, a detecção em vídeo.

Para extrair mapas de características, o algoritmo utiliza a operação de convolução várias vezes. A convolução é uma operação matemática que combina duas funções para criar uma terceira. Na visão computacional e no processamento de sinais, a convolução é frequentemente utilizada para aplicar filtros a imagens ou sinais, destacando padrões específicos. Em redes neurais convolucionais (CNNs), a convolução é empregada para extrair características das entradas, como imagens. As operações de convolução são estruturadas por Kernels (K), Strides (s) e Paddings (p).

O kernel é uma pequena matriz de números que desliza pela entrada da imagem ou sinal durante a operação de convolução. Ele multiplica seus valores pelos valores correspondentes da entrada, soma os produtos e coloca o resultado na saída. Seu objetivo é detectar características específicas na entrada ao aplicar operações locais.

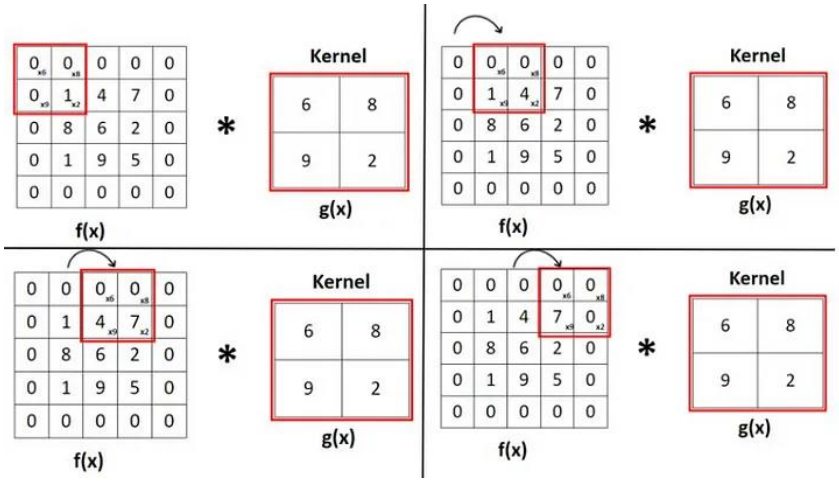
Figura 5 : Caso real de aplicação de convolução para extrair um recurso.



Fonte: developer.nvidia.com

O termo "stride" refere-se à quantidade de deslocamento do kernel enquanto este desliza sobre a imagem ou o sinal durante a operação de convolução. Um stride igual a 1 indica que o kernel move-se uma posição de cada vez, enquanto um stride igual a 2 implica que o kernel avança duas posições a cada movimento. Strides maiores resultam em uma saída com dimensões espaciais reduzidas, isto é, menor largura e altura. Em contraste, strides menores preservam uma maior quantidade de informações espaciais da entrada. Embora strides maiores reduzam o número de cálculos necessários e aumentem a velocidade da operação, essa redução pode impactar negativamente a qualidade da detecção de características, devido à menor retenção de detalhes.

Figura 6: Exemplo de Stride

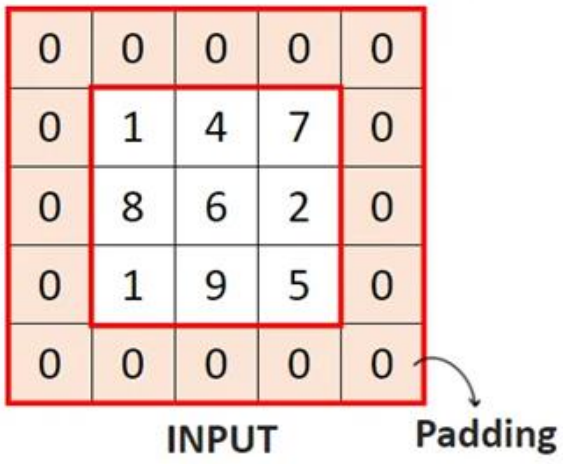


Fonte: medium.com

O termo "padding" refere-se ao preenchimento, ou seja, à adição de pixels extras ao redor das bordas da imagem de entrada antes da aplicação de operações de convolução. Esse procedimento visa garantir que as informações nas bordas

sejam tratadas com a mesma importância que as informações no centro da imagem durante as operações. Quando um filtro (ou kernel) é aplicado a uma imagem, ele percorre a imagem pixel por pixel. Na ausência de preenchimento, os pixels nas bordas da imagem são menos contemplados pelo filtro em comparação com os pixels no centro, uma vez que possuem menos contexto ao seu redor. Isso pode resultar em perda de informações nas regiões periféricas da imagem, pois essas áreas não têm um contexto tão amplo quanto os pixels centrais durante a operação de convolução.

Figura 7: Exemplo de Preenchimento 0



Fonte: medium.com

Convolução e Probabilidade: Em termos de probabilidade, a operação de convolução pode ser compreendida como um processo de combinação de duas distribuições de probabilidade. No contexto de processamento de sinais e visão computacional, a convolução pode ser interpretada como a soma ponderada de eventos aleatórios. Especificamente, ela pode ser vista como a combinação de duas distribuições de probabilidade, pX(x)p_X(x)pX(x) e pY(y)p_Y(y)pY(y), associadas a duas variáveis aleatórias XXX e YYY, respectivamente.

Figura 8: Convolução de duas variáveis aleatórias contínuas

$$(X * Y)(z) = \int_{-\infty}^{\infty} p_X(x) \cdot p_Y(z - x)\, dx \qquad (1)$$

Fonte: medium.com

No bloco de convolução do YOLOv8, a operação inicia-se com uma Convolução 2D, na qual um filtro é aplicado à entrada para extrair características locais. Em seguida, realiza-se a Batch Normalization 2D, que normaliza as ativações resultantes da convolução ao calcular as médias e os desvios padrão. Após essa etapa, a função de ativação SiLU é aplicada à saída da convolução e, opcionalmente, da normalização em lote. Finalmente, a saída da função SiLU é encaminhada para as camadas subsequentes da rede neural. A não linearidade introduzida pela função SiLU é essencial para a aprendizagem de representações não lineares dos dados.

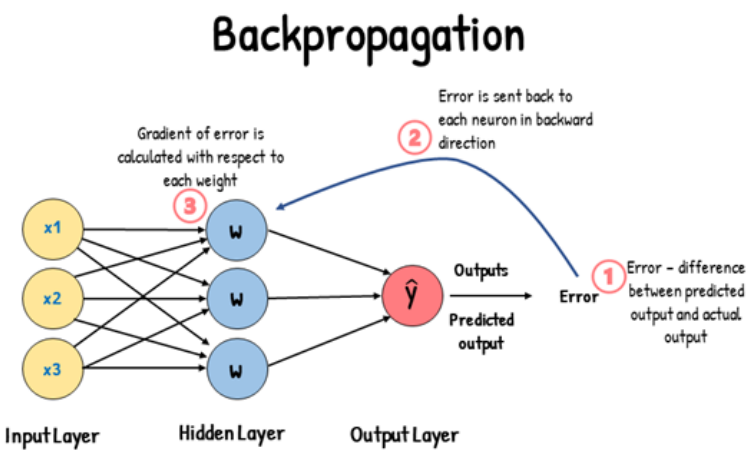
2.1.2 Treinamento do YOLOv8

Um dos passos mais importantes na aplicação de visão computacional, especialmente na detecção, é o treinamento do modelo. Antes de iniciar o treinamento, é necessário definir claramente os objetivos para a utilização da ferramenta, a fim de que o modelo possa detectar os elementos desejados. É essencial utilizar um conjunto de dados abrangente, com um número significativo de imagens, pois quanto mais treinada for a inteligência artificial, melhor será seu desempenho. O treinamento de modelos é o processo pelo qual o modelo é ensinado a reconhecer padrões visuais e a fazer previsões com base nos dados fornecidos. O treinamento influencia diretamente o desempenho e a precisão da inteligência artificial.

Um modelo de visão computacional é treinado ajustando seus parâmetros internos para minimizar os erros. Inicialmente, o modelo é alimentado com um grande conjunto de imagens rotuladas (datasets). O modelo realiza previsões sobre os objetos presentes nas imagens, e essas previsões são comparadas com os rótulos reais para calcular os erros. Esses erros indicam a distância entre as previsões do modelo e os valores verdadeiros.

Durante o treinamento, o modelo faz previsões de forma iterativa, calcula erros e atualiza seus parâmetros por meio de um processo denominado retropropagação. Nesse processo, o modelo ajusta seus parâmetros internos (weights e biases) para reduzir os erros. Ao repetir esse ciclo diversas vezes, o modelo melhora gradualmente sua precisão, aprendendo a reconhecer padrões complexos, como formas, cores e texturas.

Figura 9: Retropropagação



Fonte: medium.com

Este processo de aprendizagem permite que o modelo execute diversas tarefas, incluindo a detecção de objetos, a segmentação de instâncias e a classificação de imagens. O objetivo final é criar um modelo que possa interpretar dados visuais de forma eficaz em aplicações do mundo real.

Para treinar o modelo YOLO (You Only Look Once), é necessário escolher uma de suas variantes, como YOLOv8n.pt, YOLOv8s, entre outras. Além disso, é muito importante possuir um conjunto de dados (dataset) contendo as imagens e suas respectivas informações. Essas informações incluem as classes dos objetos, as quais podem ser definidas utilizando ferramentas de anotação de objetos em imagens, como o LabelImg. Com as anotações e as imagens prontas, é possível proceder com o treinamento. Outro aspecto crucial do treinamento é a definição adequada das configurações para garantir que o YOLOv8 opere da melhor maneira possível, de acordo com os objetivos estabelecidos.

2.1.2.1 Treinamento de Subconjunto, Multiescala, Precisão Mista e Distribuído

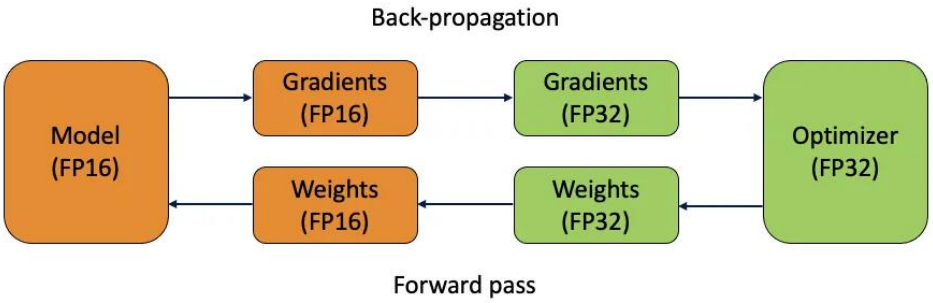
O treinamento de subconjunto refere-se ao uso de uma parte reduzida de um conjunto de dados maior para treinar um modelo de aprendizado de máquina. Esse tipo de treinamento é particularmente útil durante o desenvolvimento, pois permite testar o modelo e experimentar diferentes configurações de forma mais rápida e econômica. Além disso, economiza tempo e recursos, tornando-se uma alternativa mais ágil e leve.

No contexto do YOLOv8, é possível implementar facilmente o treinamento de subconjunto utilizando o parâmetro “fraction”. Esse parâmetro permite especificar qual fração do conjunto de dados será utilizada para o treinamento. Por exemplo, ao definir “fraction=0.1”, o modelo será treinado com 10% dos dados. Essa técnica é útil para realizar iterações rápidas e ajustar o modelo antes de investir mais tempo e recursos no treinamento completo. O treinamento de subconjunto facilita o progresso rápido e a identificação rápida de problemas.

O treinamento de multiescala é uma técnica que melhora a capacidade de generalização da inteligência artificial, pois, ao treinar o modelo em imagens de diferentes tamanhos, ele aprende a detectar objetos em várias escalas e distâncias, tornando-se, assim, mais robusto. Quando se treina o YOLOv8, pode-se ativar o treinamento multiescala definindo o parâmetro scale. Esse parâmetro ajusta o tamanho das imagens de treinamento por um fator especificado, simulando objetos em diferentes distâncias. Por exemplo, definir scale=0.5 reduzirá o tamanho da imagem pela metade, enquanto scale=2.0 dobrará o tamanho da imagem. A configuração desse parâmetro permite que o modelo experimente uma variedade de escalas de imagem, melhorando os recursos de detecção em diferentes tamanhos e cenários de objetos.

O treinamento de precisão mista utiliza tipos de pontos flutuantes de 16 bits (FP16) e 32 bits (FP32). Os pontos fortes do FP16 são aproveitados para uma computação mais rápida, otimizando o desempenho, enquanto os pontos fortes do FP32 são utilizados para manter a precisão quando necessário. Grande parte das operações da rede neural é realizada em FP16 devido à sua otimização de desempenho e tempo. No entanto, uma cópia mestre dos pesos é mantida em FP32 para garantir a precisão durante as etapas de atualização dos pesos. Modelos e lotes podem ser ajustados conforme as restrições do hardware.

Figura 10: Loop de treinamento de precisão mista



Fonte: medium.com

Para implementar o treinamento de precisão mista, é necessário modificar o script de treinamento e assegurar que o hardware utilizado (como a GPU) ofereça suporte adequado. Por exemplo, bibliotecas como TensorFlow e PyTorch fornecem suporte para utilizar a placa de vídeo no treinamento do modelo. No contexto do YOLOv8, a configuração do treinamento de precisão mista é simplificada com o uso do parâmetro "amp", um sinalizador que deve ser ativado na configuração de treinamento. Quando "amp" está definido como "True", o treinamento de precisão mista é ativado. Essa abordagem é uma excelente opção para otimizar o treinamento do modelo, devido à sua simplicidade e eficácia na melhoria do desempenho.

O treinamento distribuído é uma técnica utilizada para lidar com grandes conjuntos de dados, oferecendo uma vantagem significativa ao reduzir o tempo de treinamento ao distribuir a carga de trabalho por várias máquinas ou GPUs. Essa abordagem permite uma aceleração considerável no processo de treinamento, tornando-a ideal para situações que exigem processamento intensivo e gerenciamento eficiente de grandes volumes de dados.

2.1.2.2 Tamanho do Lote, Uso da GPU e Cache

Ao treinar o YOLOv8 com um grande dataset, o uso da GPU torna-se essencial para otimizar o desempenho do treinamento. Outro fator crucial é o tamanho do lote, que representa o número de amostras de dados que um modelo de aprendizado de máquina processa em uma única iteração de treinamento. A relação entre os lotes e a GPU é significativa, pois, ao utilizar o máximo de lotes suportados pela GPU, é possível aproveitar ao máximo suas capacidades e, assim, reduzir o tempo de treinamento do modelo. Portanto, o uso da GPU juntamente com tamanhos de lote

apropriados resulta em um desempenho aprimorado no treinamento da IA permitindo que o modelo seja treinado em um período menor.

No YOLOv8, é possível definir o parâmetro "batch_size" na configuração do treinamento para adequar-se à capacidade da GPU. Além disso, ao definir "batch = 1" no script de treinamento do YOLO, o sistema determina automaticamente o tamanho do lote que pode ser processado de forma eficiente, com base nas capacidades dos dispositivos. Dessa forma, é possível maximizar o desempenho dos recursos da GPU e melhorar o processo de treinamento em geral.

O armazenamento em cache é uma técnica importante que melhora a eficiência do treinamento de modelos de aprendizado de máquina. Ao armazenar imagens pré-processadas na memória RAM, essa técnica reduz o tempo que a GPU gasta esperando os dados serem carregados a partir do disco, uma vez que a velocidade da memória RAM é significativamente maior do que a do disco. No YOLOv8, o cache pode ser controlado durante o treinamento com as seguintes opções: "cache = True", que armazena as imagens pré-processadas na memória RAM; "cache = disk", que armazena as imagens no disco, mais lento que na memória RAM, porém mais rápido do que carregar dados novos a cada vez; e "cache = False", que desativa o cache, dependendo inteiramente do disco e sendo a opção mais lenta.

2.1.2.3 Taxa de aprendizagem

A implementação de programadores de taxas de aprendizagem ajusta dinamicamente a taxa de aprendizagem durante o treinamento. Um programador de taxa de aprendizagem é uma ferramenta ou técnica utilizada para ajustar a taxa de aprendizagem de forma adaptativa ao longo do treinamento de um modelo de rede neural. Esses programadores são essenciais porque permitem a modificação da taxa de aprendizagem com base no comportamento do treinamento, como a redução da taxa de aprendizagem após o modelo ter alcançado uma certa estabilidade. A taxa de aprendizagem é um fator crucial que influencia a eficiência do treinamento e a qualidade final do modelo.

No treinamento do YOLOv8, o parâmetro que auxilia na gestão do agendamento da taxa de aprendizagem é o "lrf". Este parâmetro define a taxa de aprendizagem final como uma fração da taxa de aprendizagem inicial, facilitando a adaptação dinâmica da taxa de aprendizagem ao longo do processo de treinamento.

2.1.2.4 Épocas e parada antecipada

Ao treinar um modelo, uma época refere-se a uma passagem completa por todo o conjunto de dados de treinamento. Durante uma época, o modelo processa cada exemplo no conjunto de treinamento uma vez e, seguindo a lógica da rede convolucional para aprendizado, atualiza seus parâmetros. Normalmente, são necessárias várias épocas para permitir que o modelo aprenda e refine seus parâmetros ao longo do tempo. É importante atentar-se ao número de épocas durante o treinamento. Um bom ponto de partida é 300 épocas; no entanto, se o modelo for ajustado antecipadamente, pode-se reduzir o número de épocas. Caso o overfitting não ocorra após 300 épocas, é possível estender o treinamento para 600, 1.200 ou mais épocas.

Overfitting refere-se a um problema onde o modelo aprende excessivamente sobre os dados de treinamento, incluindo o ruído e as particularidades específicas desses dados, em vez de generalizar para novas amostras. Isso resulta em um modelo que tem um desempenho excelente nos dados de treinamento, mas apresenta um desempenho insatisfatório em dados não vistos. O número ideal de épocas pode variar com base no tamanho do conjunto de dados e nos objetivos do projeto. Conjuntos de dados maiores podem exigir mais épocas para que o modelo aprenda de forma eficaz, enquanto conjuntos de dados menores podem necessitar de menos épocas para evitar o ajuste excessivo, pois pode haver um ponto em que a inteligência artificial pare de aprender durante as épocas, dependendo do dataset. O parâmetro "Epochs" define o número de épocas no script de treinamento.

A parada antecipada é uma técnica valiosa para otimizar o treinamento do modelo. Ao monitorar o desempenho de validação, é possível interromper o treinamento quando o modelo parar de apresentar melhorias. Essa técnica economiza tempo e recursos computacionais, além de evitar o overfitting. O processo envolve a definição de um parâmetro de paciência, que determina o número de épocas a esperar por melhorias nas métricas de validação antes de interromper o treinamento. Se o desempenho não melhorar dentro dessas épocas, o treinamento será interrompido para evitar desperdício de tempo e recursos. No YOLOv8, isso é feito definindo o parâmetro "patience" na configuração do treinamento. Por exemplo, "patience = 5" significa que o treinamento será interrompido se não houver melhorias nas métricas de validação por 5 épocas consecutivas. O uso deste método garante que o processo

de treinamento do modelo continue evoluindo, trazendo melhorias contínuas e mantendo o modelo eficiente e com desempenho ideal, sem computação excessiva.

2.1.2.5 Otimizadores Comuns

Um otimizador é um algoritmo que ajusta os pesos da rede neural para minimizar a função de perda, que mede o desempenho do modelo. Em outras palavras, o otimizador auxilia o modelo a aprender ajustando seus parâmetros para reduzir os erros. A escolha do otimizador correto afeta diretamente a rapidez e a precisão com que o modelo aprende. Além disso, é possível ajustar os parâmetros do otimizador para melhorar o desempenho do modelo. Ajustar a taxa de aprendizagem define o tamanho das etapas ao atualizar os parâmetros. Para garantir estabilidade, recomenda-se começar com uma taxa de aprendizagem moderada e reduzi-la gradualmente ao longo do tempo para melhorar a aprendizagem a longo prazo. Também, definir o parâmetro de impulso (momentum) determina a influência das atualizações anteriores nas atuais. Um valor comum para o momentum é em torno de 0,9, geralmente fornecendo um bom equilíbrio.

SGD (Stochastic Gradient Descent): Este otimizador atualiza os parâmetros do modelo utilizando o gradiente da função de perda em relação aos parâmetros. O gradiente indica a direção e a magnitude de como os pesos devem ser ajustados para minimizar a função de perda, que quantifica os erros entre as previsões do modelo e os valores reais. O SGD é simples e eficiente, mas pode demorar para convergir e pode ficar preso em mínimos locais, devido às suas atualizações ruidosas, que podem causar oscilações ou uma convergência lenta, especialmente em regiões complexas da função de perda com múltiplos mínimos locais.

ADAM (Adaptive Moment Estimation): Combina os benefícios do SGD com momentum (uma técnica usada em otimizadores de redes neurais para acelerar a convergência) e RMSProp. Ele ajusta a taxa de aprendizagem para cada parâmetro com base nas estimativas dos primeiros e segundos momentos dos gradientes. Além disso, é adequado para dados ruidosos e gradientes esparsos (gradientes em que a maioria dos valores é zero ou muito próximo de zero). Esse otimizador é recomendado para o YOLOv8, pois é eficiente e geralmente requer menos ajustes.

RMSProp (Root Mean Square Propagation): Este otimizador adaptativo ajusta a taxa de aprendizagem para cada parâmetro individualmente e divide o gradiente por uma média móvel dos quadrados dos gradientes recentes. Isso ajuda a estabilizar as

atualizações, evitando grandes oscilações e acelerando a convergência. É eficiente no tratamento do problema de gradiente evanescente, comum em redes neurais recorrentes, e contribui para uma aprendizagem mais estável e eficiente em redes neurais profundas.

No YOLOv8, é possível definir qual otimizador utilizar através do parâmetro "optimizer". Esse parâmetro permite escolher entre vários otimizadores, incluindo os mencionados, e também pode ser configurado para seleção automática com base na configuração do modelo, deixando o parâmetro como "auto", ou seja, "optimizer = auto".

O treinamento de modelos de visão computacional geralmente requer desempenho computacional elevado e envolve seguir boas práticas, estratégias de otimização e resolução de problemas à medida que surgem. Técnicas como ajuste de tamanhos de lote, treinamento de precisão mista e uso de pesos pré-treinados podem melhorar o desempenho dos modelos e acelerar o treinamento, economizando recursos computacionais e tempo.

2.1.2.6 Modelos Pré-treinados

Modelos pré-treinados são modelos de aprendizado profundo que foram treinados previamente em grandes conjuntos de dados para resolver tarefas específicas, como classificação de imagens, entre outras. Esses modelos já "aprenderam" a identificar padrões e características úteis. Em vez de treinar um modelo do zero, é possível utilizar um modelo pré-treinado como base e adaptá-lo para um objetivo específico. Essa abordagem pode acelerar o treinamento e melhorar o desempenho do modelo. O ajuste de um modelo pré-treinado envolve iniciar com esses pesos e, em seguida, continuar o treinamento com um novo conjunto de dados.

O parâmetro "pretrained" facilita a aprendizagem por transferência no YOLOv8. Quando o "pretrained" está definido como "True", o treinamento utiliza pesos pré-treinados padrão. Alternativamente, pode-se especificar um caminho para um modelo pré-treinado personalizado. O uso de pesos pré-treinados e a aprendizagem por transferência aumentam efetivamente os recursos do modelo e reduzem os custos do treinamento.

2.1.2.7 Versões de modelos de rede YOLOv8

As versões do YOLOv8 diferem em tamanho e capacidade de detecção de objetos, com cada versão projetada para equilibrar precisão e velocidade de acordo com as necessidades específicas e limitações de hardware. Por exemplo, o YOLOv8n é a versão mais leve, adequada para dispositivos com recursos limitados, enquanto o YOLOv8x é a versão mais robusta, destinada a dispositivos com alta capacidade computacional.

A tabela a seguir apresenta os modelos do YOLOv8 treinados no COCO (Microsoft Common Objects in Context), um conjunto de dados amplamente utilizado em tarefas de visão computacional, como a detecção de objetos. A tabela inclui informações sobre o tamanho dos pixels (resolução da imagem usada para a detecção), mAP@50-95 (média da precisão média em diferentes limiares de Interseção sobre União, IoU, de 50% a 95%), velocidade (tempo de inferência em milissegundos usando TensorRT), número de parâmetros do modelo (expresso em milhões) e FLOPs (número de operações de ponto flutuante por segundo, expresso em bilhões).

Figura 11: Modelos do YOLOv8

Modelo	tamanho (pixéis)	mAPval 50-95	Velocidade CPU ONNX (ms)	Velocidade A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Fonte: github.com

O YOLOv8 oferece diferentes versões para atender a várias necessidades de hardware e objetivos específicos. A versão YOLOv8n é a mais simples e leve, ideal para dispositivos com recursos limitados, como celulares. Embora ofereça menor precisão em comparação com os modelos maiores, é mais rápida e eficiente em

termos de uso de hardware, proporcionando um bom equilíbrio entre desempenho e consumo reduzido de recursos, resultando em menor precisão e maior desempenho.

A versão YOLOv8s é pequena e adequada para aplicações em tempo real em hardware relativamente limitado, mas não tão restrito quanto o modelo Nano. Este modelo é projetado para oferecer um equilíbrio entre precisão e velocidade, sendo útil em cenários que exigem um bom desempenho com menos potência de computação, apresentando uma precisão um pouco melhor que o Nano e um pouco mais de peso.

O YOLOv8m possui um tamanho médio e é versátil para uma variedade de tarefas de detecção de objetos, proporcionando um equilíbrio entre desempenho e precisão. É o modelo com maior precisão entre os modelos leves e oferece um desempenho razoável, resultando em um desempenho médio e precisão média.

A versão YOLOv8l é ideal para dispositivos que suportam uma carga computacional maior, oferecendo alta precisão. No entanto, para utilizar este modelo é necessário um poder computacional mais significativo. Sua maior capacidade de modelagem resulta em um melhor desempenho na detecção e classificação de imagens mais complexas, proporcionando uma precisão de alta qualidade e um desempenho mais pesado.

Finalmente, o YOLOv8x é o modelo mais preciso e mais pesado, utilizado em situações onde a precisão é crucial e o poder computacional não é uma limitação, como em servidores e máquinas de alto desempenho. Suas características incluem máxima precisão, maior capacidade de detecção e uma maior demanda em termos computacionais e de memória.

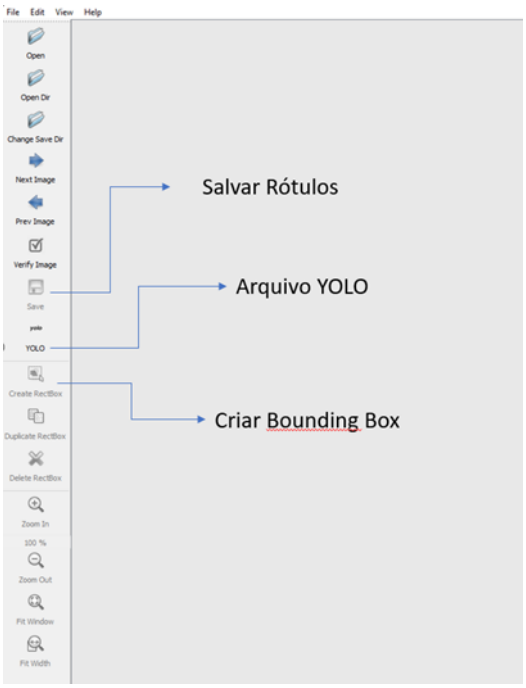
Cada um desses modelos é projetado para atender a diferentes tipos de hardware e objetivos, permitindo o uso da rede neural YOLO em dispositivos com variadas capacidades computacionais.

2.1.2.8 Dataset

O dataset é uma parte essencial no treinamento do YOLOv8, pois é com ele que será possível treinar o modelo para a detecção dos objetos desejados. O dataset utilizado para o treinamento do YOLOv8 deve possuir características específicas para garantir a eficácia do processo. Em particular, é necessário que o dataset contenha imagens devidamente rotuladas, uma etapa crucial do processo de anotação de objetos nas imagens. Para realizar essa anotação, é necessário utilizar ferramentas que extraem informações das imagens e as atribuem a classes específicas, facilitando

o treinamento do modelo. Uma dessas ferramentas é o LabelImg, que permite a criação e a edição dessas anotações de forma eficiente.

Figura 12: Labelimg



Fonte: autoria própria

No LabelImg, ao selecionar uma imagem, é possível criar uma bounding box para marcar o objeto que deve ser detectado e indicar a qual classe ele pertence. Após a criação da bounding box, é necessário salvar o arquivo no formato específico do YOLO. Isso resultará na geração de um arquivo de texto com as informações da imagem, tendo o mesmo nome do arquivo de imagem.

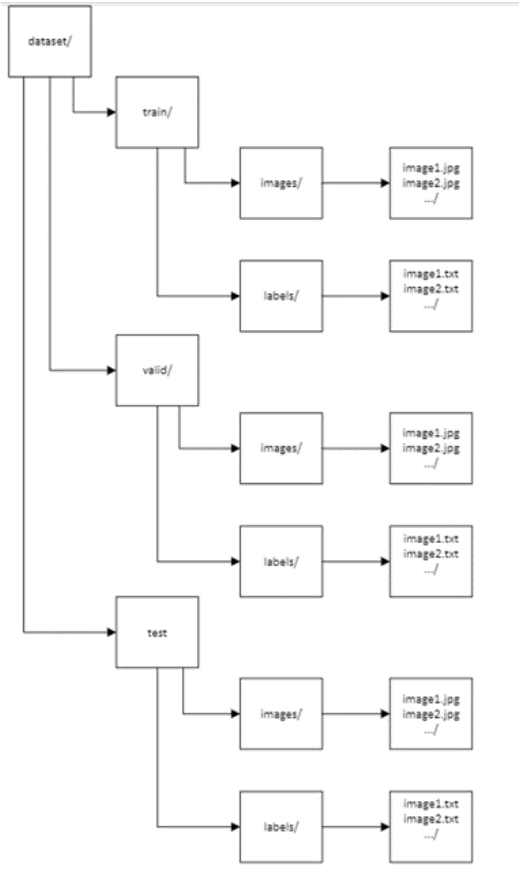
O dataset para treinamento do YOLOv8 deve seguir uma estrutura específica, composta por três pastas, sendo uma delas opcional. Dentro de cada pasta, devem existir duas subpastas: uma chamada "images", que contém as imagens do dataset, e outra chamada "labels", que contém as anotações em arquivos de texto.

A primeira pasta é a "train", que armazena as imagens e anotações usadas para treinar o modelo. A maior parte do conteúdo do dataset está nessa pasta, onde as imagens são acompanhadas por arquivos de anotações que descrevem as caixas delimitadoras e as classes dos objetos presentes nas imagens. A importância dessa pasta reside no fato de que o modelo aprende a partir dos dados contidos nela.

Outra pasta essencial é a "valid", que contém as imagens e anotações usadas para validar o modelo durante o treinamento. Seu conteúdo é um subconjunto do dataset que não é utilizado diretamente para treinar o modelo, mas para avaliar seu desempenho em dados que não foram vistos durante o treinamento. A importância dessa pasta é fundamental para monitorar o desempenho do modelo e detectar problemas como overfitting.

Por fim, a pasta opcional "test" é destinada a conter as imagens e anotações para testar o modelo após o treinamento. Assim como a pasta "valid", o conteúdo da pasta "test" é um subconjunto do dataset, mas é completamente separado dos dados de treino e validação, permitindo a avaliação do modelo de forma independente.

Figura 13: Estrutura de pastas do dataset do YOLO



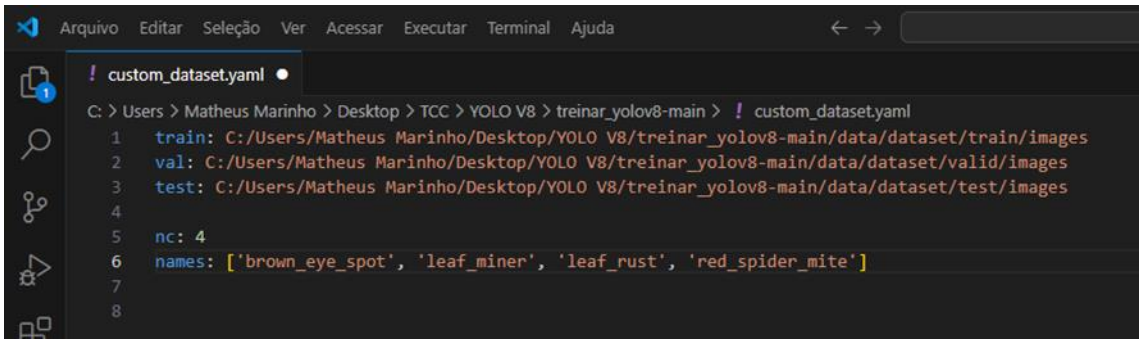
Fonte: autoria própria

A estrutura das pastas do dataset do YOLOv8 está vinculada a um arquivo YAML, que é essencial para a configuração do treinamento do modelo. Este arquivo YAML estabelece a conexão entre o código de treinamento e o dataset, desempenhando um papel crucial na configuração do treinamento. Além disso, o

arquivo YAML define as classes de objetos presentes no dataset, permitindo que o modelo compreenda e categorize adequadamente os diferentes tipos de objetos.

Portanto, a criação e a configuração correta do arquivo YAML são fundamentais para garantir que o modelo de YOLOv8 seja treinado corretamente, reconhecendo e distinguindo as classes de objetos conforme especificado.

Figura 14: Arquivo YAML do dataset do YOLO



Fonte: autoria própria

Na imagem acima, observa-se um arquivo YAML que estabelece a conexão com as pastas do dataset. Neste arquivo, as seções "train", "val" e "test" especificam os caminhos das respectivas pastas que contêm as imagens e anotações. A variável "nc" refere-se ao número total de classes presentes no dataset, enquanto "names" lista os nomes dessas classes.

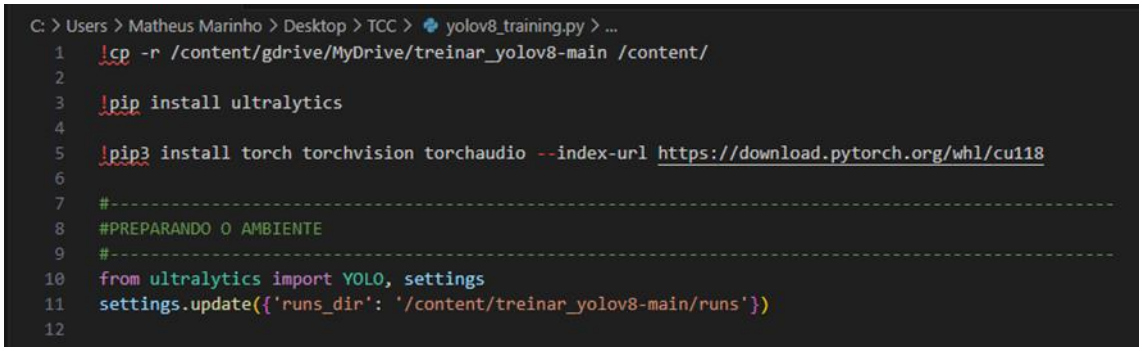
Os benefícios de utilizar um arquivo YAML são diversos. Primeiramente, a centralização das configurações em um único arquivo facilita a gestão e a organização, permitindo uma visão consolidada das configurações do dataset. Além disso, a flexibilidade proporcionada pelo arquivo YAML é significativa, pois permite ajustar rapidamente os caminhos do dataset e outros parâmetros sem a necessidade de modificar o código principal. Isso torna o processo de configuração e manutenção do treinamento mais eficiente e menos propenso a erros.

2.1.2.9 Treinamento do modelo utilizado no projeto

O modelo utilizado neste projeto foi treinado para detectar quatro tipos de doenças na plantação de café. As classes que o modelo irá identificar são: "brown_eye_spot", "leaf_miner", "leaf_rust" e "red_spider_mite". Estes nomes das classes estão em inglês, pois o dataset utilizado para o treinamento do modelo possui

os nomes das classes nesse idioma. A tradução para o português dessas classes é: Mancha de Olho Pardo, Minador de Folhas, Ferrugem das Folhas e Ácaro Vermelho. Essas doenças foram escolhidas para o protótipo inicial devido à sua alta incidência em plantações de café. O dataset que incluía essas doenças continha um número significativo de imagens, o que possibilitou um treinamento eficaz. Com o sucesso do protótipo na detecção dessas doenças, é possível treinar o modelo para identificar outras doenças adicionais, utilizando um dataset apropriado para essas novas classes.

Figura 15: Configurações para o Treinamento do Modelo



Fonte: autoria própria

O modelo deste projeto foi treinado no Google Colab Pro devido ao seu alto desempenho e capacidade computacional. A primeira linha de código refere-se à importação do dataset para o ambiente virtual para treinar o modelo. Logo abaixo, a terceira linha se refere à instalação da biblioteca da Ultralytics no ambiente virtual, pois essa biblioteca contém todas as funcionalidades necessárias para treinar o YOLO. A quinta linha corresponde à instalação do PyTorch, que é essencial para utilizar o poder da GPU no treinamento do modelo. Na décima linha, a biblioteca da Ultralytics é importada, junto com a classe principal do modelo YOLO para detecção de objetos e o módulo "settings", que contém configurações e parâmetros que podem ser usados para ajustar o comportamento do modelo.

Figura 16: Código do Treinamento do Modelo

```
13  #-----------------------------------------------------------------------------------------------------
14  #TREINAMENTO DA IA
15  #-----------------------------------------------------------------------------------------------------
16  model = YOLO('yolov8m')
17
18  model.train(data='/content/treinar_yolov8-main/custom_dataset.yaml',
19              epochs=100,
20              patience=10,
21              batch=16,
22              imgsz=640,
23              workers=8,
24              pretrained=True,
25              resume=False,
26              single_cls=False,
27              box=7.5,
28              cls=0.5,
29              dfl=1.5,
30              val=True,
31              degrees=0.3,
32              hsv_s=0.3,
33              hsv_v=0.3,
34              scale=0.5,
35              project='/content/treinar_yolov8-main/runs',
36              name='IA_Treinada',
37              save_period=3,
38              )
```

Fonte: autoria própria

A parte principal do código configura os parâmetros para o treinamento do YOLO. A escolha da versão do modelo foi para o "YOLOv8m", pois seu desempenho se adequa às limitações de hardware do projeto e à maioria dos computadores modernos, oferecendo uma boa precisão. Na linha 18, a função "model.train" é utilizada para iniciar o treinamento do modelo. O parâmetro "data" especifica o caminho do arquivo YAML. O parâmetro "epochs" define o número de épocas e foi ajustado para 100 devido à quantidade de arquivos no dataset. O parâmetro "patience" foi definido como 10, o que significa que o treinamento será interrompido se o modelo não apresentar melhorias em 10 épocas consecutivas. O parâmetro "batch" é o número de imagens por lote durante o treinamento e foi definido como -1 para que o número de lotes se ajuste à capacidade do poder computacional. O parâmetro "imgsz" define o tamanho das imagens para 640x640 pixels, impactando diretamente o desempenho do treinamento.

O parâmetro "workers" especifica o número de trabalhadores (threads) para carregar os dados. O parâmetro "pretrained" foi definido como "true" para utilizar pesos pré-treinados. O parâmetro "resume" foi definido como "false", indicando que o treinamento não será retomado a partir de um checkpoint anterior. O parâmetro "single_cls" foi definido como "false", pois o conjunto de dados contém múltiplas classes. O parâmetro "box" foi definido como 7.5, ajustando a ponderação da perda de caixa delimitadora para melhorar a IoU. O parâmetro "cls" foi definido como 0.5,

ajustando a ponderação da perda de classe para aprimorar a classificação das caixas delimitadoras. O parâmetro "dfl" ajusta a perda focal de distribuição, e o valor de 1.5 aumenta a ênfase na precisão dos limites das caixas delimitadoras. O parâmetro "val" foi definido como "true", permitindo validação durante o treinamento. O parâmetro "degrees" foi definido como 0.3, permitindo que as imagens sejam rotacionadas aleatoriamente em até 0.3 graus para aumentar a diversidade do conjunto de dados.

O parâmetro "hsv_s" foi definido como 0.3, ajustando a saturação no espaço de cor HSV em até 30%, e "hsv_v" também foi definido como 0.3 para ajustar a luminosidade de forma semelhante. O parâmetro "scale" permite o redimensionamento aleatório das imagens em até 50%. O parâmetro "project" especifica o caminho para salvar os resultados do treinamento, e "name" define o nome do experimento. Por fim, "save_period" foi definido como 3, indicando que um checkpoint será salvo a cada 3 épocas.

Esse foi o primeiro treinamento do modelo "YOLOv8" utilizado neste projeto. O treinamento do YOLO permite diversas alterações em suas configurações e a utilização de várias técnicas de otimização. Com este primeiro teste, foi possível avaliar o funcionamento do modelo e identificar melhorias a serem feitas nos parâmetros de treinamento para criar um modelo mais eficiente.

2.1.3 Uso do YOLOV8 em tempo real e fotos

Com o YOLOv8, é possível realizar várias tarefas de visão computacional. Neste projeto, o YOLOv8 será utilizado para a detecção de objetos, com o objetivo específico de identificar doenças nas plantações de café. O YOLOv8 é versátil e pode ser aplicado tanto em tempo real quanto em fotos estáticas, adaptando-se às necessidades do projeto para fornecer resultados eficazes na identificação de doenças.

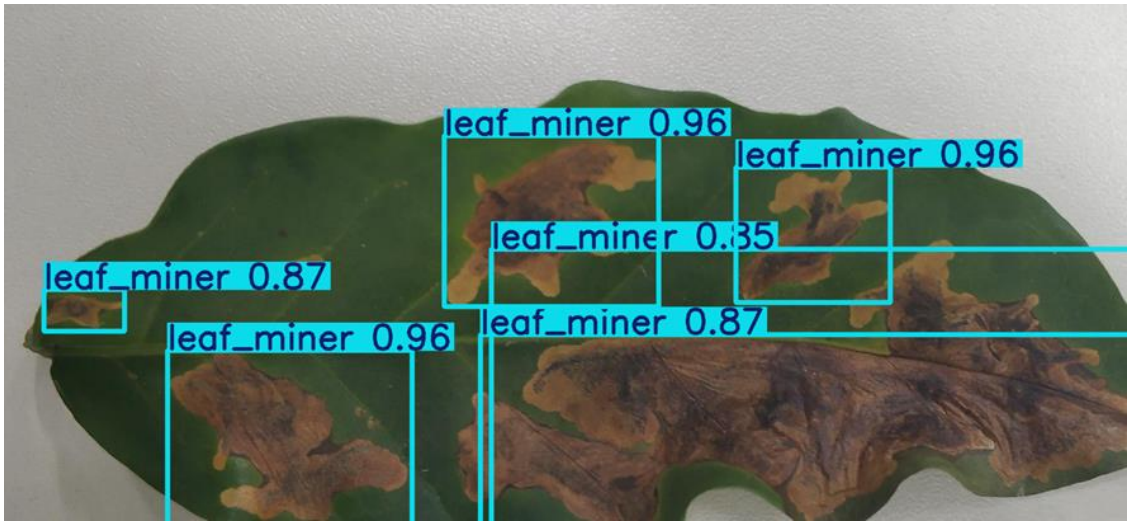
Figura 17: Código do YOLO para operar em tempo real

```
# Conectar ao dispositivo
device = adbutils.adb.device()

# Comando para obter o nome do modelo do dispositivo
nome_dispositivo = device.shell('getprop ro.product.model').strip()

# inicializar a classe WindowCapture
wincap = WindowCapture(nome_dispositivo)

# Carregar o modelo YOLOv8
model = YOLO("best.pt")

# Definir o limiar de confianca
limiar_confianca = 0.8
```

Fonte: autoria própria

Na figura acima, o YOLO foi configurado para operar em tempo real, realizando a detecção de objetos apenas para aqueles cuja confiança é superior a 80%. As bibliotecas utilizadas incluem Ultralytics, AdbUtils, CV2, Numpy e time. O script também incorpora um outro script chamado windowscapture.py, que fornece todas as funcionalidades necessárias para a detecção em tempo real com YOLO, acessando a imagem da câmera do drone. O modelo treinado, denominado best.pt, é o primeiro modelo desenvolvido neste projeto.

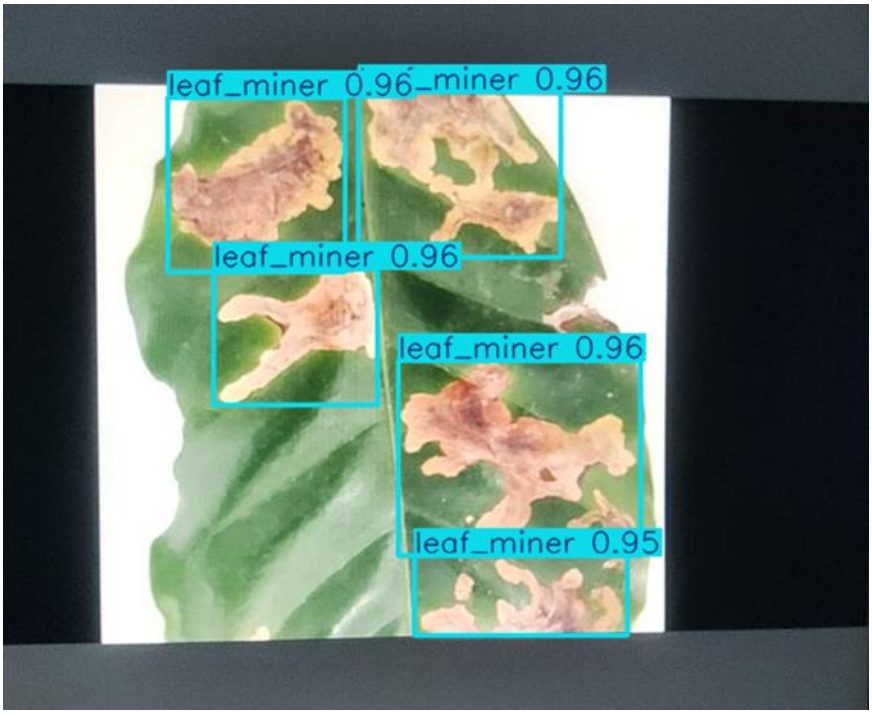
Figura 18: Teste de Detecção em Imagem Estática



Fonte: autoria própria

A detecção de objetos em fotos pode ser mais precisa do que em tempo real, pois as imagens carregadas geralmente têm uma resolução maior e são processadas de uma só vez. Como o modelo analisa apenas um quadro por vez, a detecção pode ser mais detalhada e precisa. Além disso, o processamento em foto elimina a necessidade de lidar com o fluxo contínuo de vídeo, permitindo que o modelo se concentre na análise detalhada de cada imagem individualmente. Isso pode resultar em uma maior precisão na identificação e classificação dos objetos presentes na imagem.

Figura 19: Teste de Detecção em Tempo Real



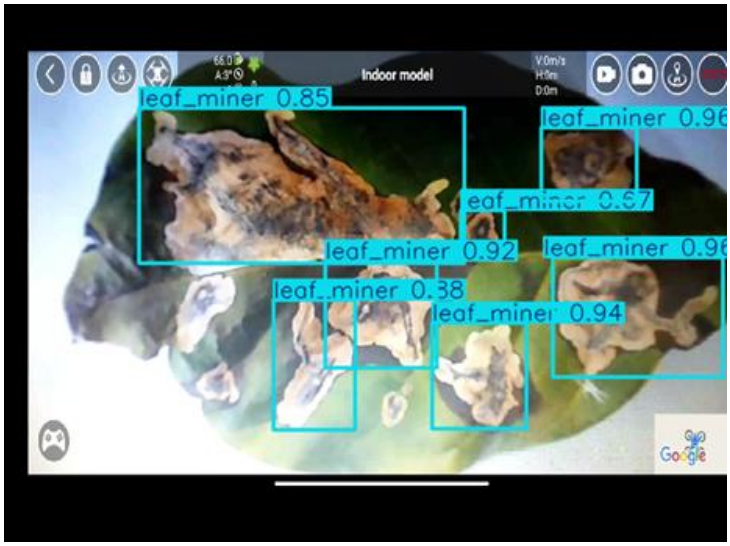
Fonte: autoria própria

A versão em tempo real do YOLO utiliza uma resolução menor para se adaptar às limitações de hardware e garantir uma detecção rápida. A redução na resolução ajuda a diminuir o tempo de processamento por quadro, permitindo que o modelo realize a detecção em um fluxo contínuo de vídeo. O número de quadros por segundo (fps) que o sistema pode processar e analisar dependerá das capacidades do hardware utilizado, como a potência da GPU e a eficiência do sistema de processamento. Com um hardware mais potente, é possível alcançar uma maior taxa de quadros por segundo, proporcionando uma detecção de objetos em tempo real mais fluida e eficiente.

2.1.4 Conexão do YOLO com o drone

O drone selecionado para este projeto não possui tecnologia de ponta, o que impede a realização de uma conexão direta entre o código da inteligência artificial (IA) e o dispositivo. Diante dessa limitação, optou-se pela utilização da ferramenta "scrcpy", responsável por espelhar a tela do celular no computador e adaptar o código do YOLOV8 para conseguir acessar a imagem da câmera do drone. O script windowscapture.py tem a função de capturar quadros de uma janela específica. Para isso, o usuário deve manter o celular conectado via USB ao computador, com a depuração USB ativada. Ao executar o código, o scrcpy é ativado, abrindo uma janela com o nome do dispositivo, que espelha a tela do celular. A função device.shell('getprop ro.product.model').strip() da biblioteca AdbUtils obtém o nome do celular conectado e armazena essa informação em uma variável. Essa variável é então utilizada para identificar o nome da janela que deve ser espelhada, já que o nome da janela espelhada pelo scrcpy corresponde ao nome do dispositivo conectado. Dessa forma, é possível realizar a detecção em tempo real com o YOLOv8, acessando a imagem da câmera do drone.

Figura 20: Teste de Detecção em Tempo Real com a câmera do drone



Fonte: autoria própria

2.2. Aplicativo

2.2.1. Kivy e KivyMD

Para o desenvolvimento do aplicativo foi utilizado o kivy e o kivyMD para criar o aplicativo móvel e desktop. O kivy como já mencionado é uma biblioteca em Python que é voltada para o desenvolvimento de aplicativos móveis ou desktop, já o kivyMD é uma coleção de widgets compatíveis com Material Design para uso com o framework gráfico multiplataforma Kivy.

Com isso, foi possível criar um aplicativo que tenha um design agradável de forma mais simples por conta do KivyMD.

Utilizou-se o Kivy e o KivyMD para criar o aplicativo móvel e desktop. O Kivy é uma biblioteca em Python voltada para o desenvolvimento de aplicativos móveis e desktop, enquanto o KivyMD é uma coleção de widgets compatíveis com o Material Design para uso com o framework gráfico multiplataforma Kivy.

Com isso, foi possível desenvolver um aplicativo com um design agradável de forma mais simples, graças ao KivyMD.

2.2.2 Construindo a Interface

2.2.2.1 Tela principal

Na tela principal, o objetivo foi exibir as últimas doenças detectadas e os tratamentos correspondentes. A tela apresenta três retângulos que mostram um texto extraído do banco de dados, indicando a doença detectada e suas coordenadas. Além disso, há outros três retângulos que representam possíveis tratamentos para a doença identificada.

Figura 21: Tela Inicial do Aplicativo

Fonte: autoria própria

2.2.2.2 Tela do Mapa

Para a tela do mapa, foram utilizados dois webviews para abrir um mapa gerado pela biblioteca Folium. Na versão desktop do aplicativo, foi utilizado o webview da biblioteca Pywebview, enquanto na versão Android, foi utilizado o webview do repositório Android-for-Python.

A escolha de dois webviews distintos deve-se a questões de versionamento e compatibilidade. Após muitos testes, o Pywebview mostrou-se compatível com versões mais recentes do Python e com o KivyMD, mas não é compatível com Android. Por isso, foi necessário utilizar um webview específico para Android.

2.3. Conexão com o Banco de dados

Para o envio de informações de latitude, longitude e o tipo de doença detectada, foi utilizado um banco de dados online chamado Firebase. A conexão com o banco é feita no mesmo script que captura as últimas coordenadas do drone, enviando assim a latitude e longitude para o banco de dados.

2.3.1. Criação do Banco de dados

Para criar o banco de dados no Firebase, inicialmente é necessário configurar um projeto no console do Firebase. Após a criação do projeto, deve-se configurar o banco de dados para o Firebase Realtime Database. No Firebase Realtime Database, a estrutura de dados é baseada em JSON, permitindo armazenar e sincronizar dados em tempo real.

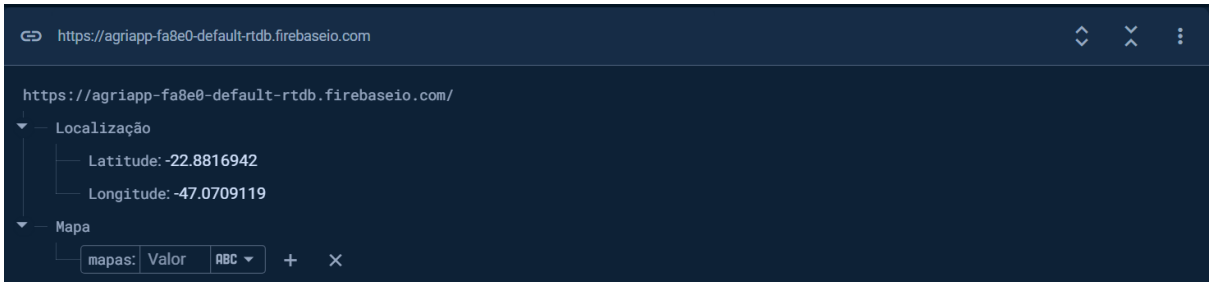
Depois de configurar o banco de dados, é preciso integrar o Firebase ao aplicativo. Isso envolve adicionar o SDK do Firebase ao projeto e configurar as credenciais de acesso. Com essa integração, o aplicativo poderá se conectar ao banco de dados e realizar operações como leitura, escrita e atualização dos dados conforme necessário.

Figura 22: Código de conexão com o banco de dados

```
#Conexão com o Banco de Dados
firebaseConfig = {
    "apiKey": "AIzaSyAaNamwwtR0ZMMAw6vd0OSF8P9sO9zQ4qU",
    "authDomain": "agriapp-fa8e0.firebaseapp.com",
    "databaseURL": "https://agriapp-fa8e0-default-rtdb.firebaseio.com",
    "projectId": "agriapp-fa8e0",
    "storageBucket": "agriapp-fa8e0.appspot.com",
    "messagingSenderId": "985642513958",
    "appId": "1:985642513958:web:7b37fcf22b46c237a8d8e7",
    "measurementId": "G-QYRPP2NJ9S"
  }
```

Fonte: autoria própria

Figura 23: Estrutura do banco de dados



Fonte: autoria própria

2.4. Criação do Mapa

A criação do mapa, utilizado para visualizar a localização onde a doença foi detectada pelo drone, foi realizada com a biblioteca Folium. O processo de criação de mapas é relativamente simples, exigindo apenas a inserção das coordenadas de latitude e longitude para gerar o mapa. No entanto, o mapa é gerado no formato HTML, o que requer o uso de um visualizador web para ser incorporado ao aplicativo.

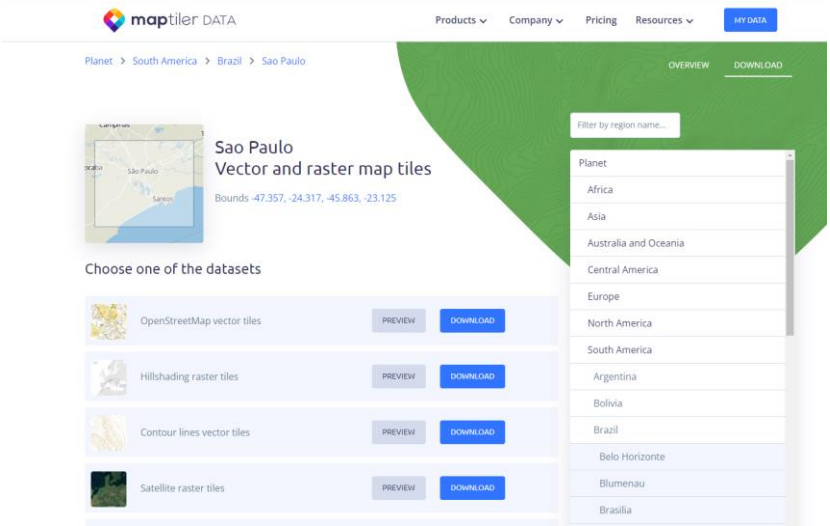
Outro aspecto relevante é que a biblioteca Folium conta com um plug-in que permite obter a localização exata do usuário, facilitando a criação de um mapa da região de cultivo do agricultor. Para a criação desses mapas, é necessário acesso à internet, pois o Folium utiliza provedores como o OpenStreetMap, que hospeda os tiles e demais dados dos mapas na web. Esses serviços precisam ser acessados online para fornecer a visualização. Contudo, um problema comum no meio rural é a falta de acesso à internet devido à infraestrutura limitada em determinadas regiões. Para resolver essa questão no protótipo, foi utilizado o mapa do Estado de São Paulo, demonstrando que é possível criar e acessar mapas com Folium sem conexão à internet. Embora o protótipo tenha utilizado o mapa de São Paulo, a mesma solução pode ser aplicada para qualquer estado brasileiro ou para o Brasil inteiro.

2.4.1. Arquivos do Mapa

Para viabilizar o uso offline do mapa, foi feito o download do dataset "maptiler-osm-2024-09-30-v3.15-south-america_brazil.mbtiles" no site MapTiler *DATA*. Esse arquivo, no formato de banco de dados (.mbtiles), é utilizado para armazenar tiles de

mapas de forma compacta e eficiente. Os arquivos contêm blocos de mapas (tiles), que são pequenas imagens ou dados vetoriais usados para formar um mapa interativo. O mapa do OpenStreetMap vector tiles referente ao Estado de São Paulo foi escolhido para demonstrar a viabilidade de criar e acessar mapas offline. O *MapTiler DATA* disponibiliza mapas globais, incluindo todos os estados do Brasil. Para integrar o arquivo mbtiles com o *Folium*, foi necessária a conversão do banco de dados para uma estrutura de pastas com as imagens dos tiles.

Figura 24: Site MapTiler



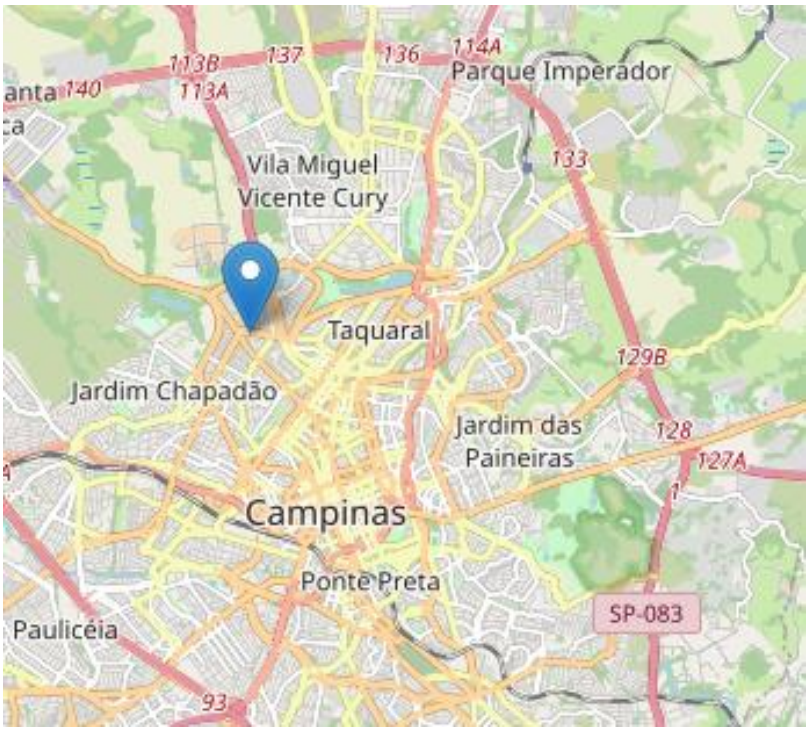
Fonte: maptiler.com

2.4.2. Criação e acesso do mapa sem Internet

A conversão do arquivo mbtiles para uma estrutura de pastas com imagens em formato PNG foi realizada por meio do script "mbtilesToPngs.py". Esse script, ao receber o caminho do arquivo mbtiles, converte o conteúdo para uma estrutura de pastas, organizando as camadas de acordo com o nível de zoom do mapa.

Para acessar o mapa de maneira offline, foi necessário adaptar o código do Folium de modo que o script passasse a utilizar os arquivos de tiles armazenados localmente. Dessa forma, em vez de carregar os tiles diretamente da internet, o mapa acessa os arquivos já convertidos e organizados nas pastas, conforme o usuário ajusta o zoom.

Figura 25: Criação de Mapa sem Internet



Fonte: autoria própria

2.5.1. Extração das coordenadas do drone

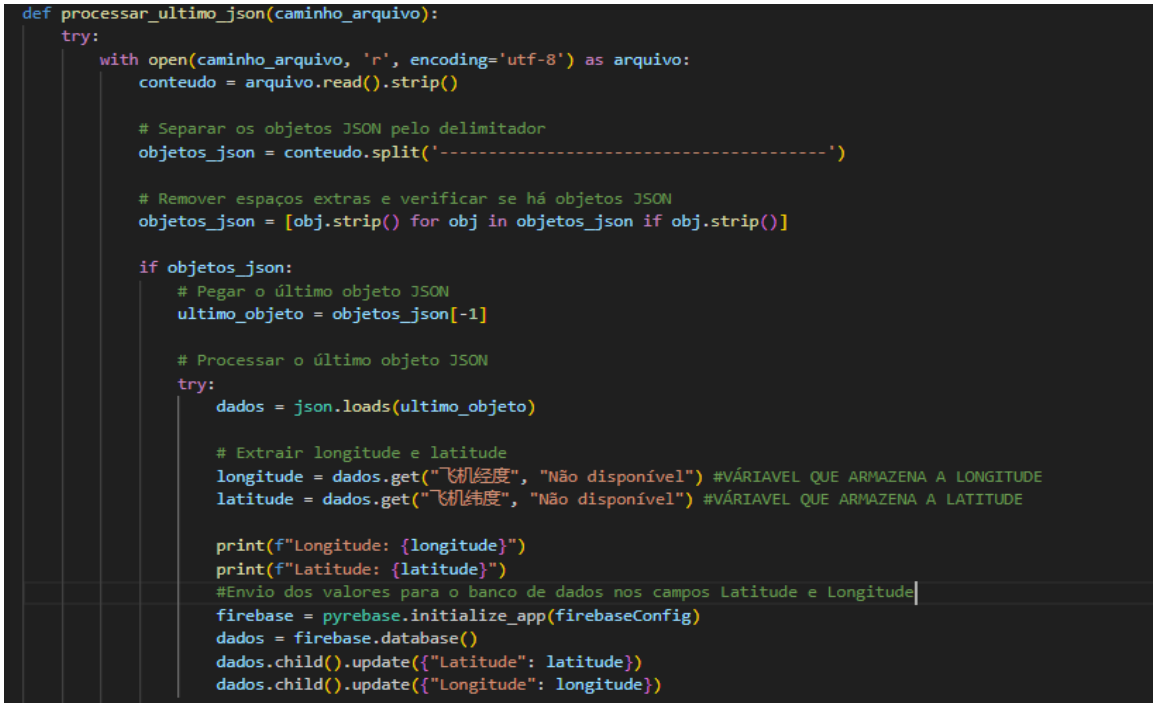
Para a extração das coordenadas do drone, foi desenvolvido um algoritmo que lê um arquivo chamado "FlightLog". Esse arquivo é gerado pelo aplicativo do drone e contém informações como data, distância, hora, latitude e longitude. O script percorre o arquivo, localiza a latitude e a longitude, salva essas informações em uma variável e envia os valores para o banco de dados.

Figura 26: Arquivo "FlightLog"

```
{
  "日期": "2024-05-29 11:43:03",  // Data: "2024-05-29 11:43:03"
  "飞机经度": 0.0,  // Longitude do avião: 0.0
  "飞机纬度": 0.0,  // Latitude do avião: 0.0
  "水平速度(m/s)": 0.0,  // Velocidade horizontal (m/s): 0.0
  "垂直速度(m/s)": 0.0,  // Velocidade vertical (m/s): 0.0
  "飞机高度(m)": 0.0,  // Altura do avião (m): 0.0
  "飞机距离(m)": 0.0,  // Distância do avião (m): 0.0
  "卫星数": 0,  // Número de satélites: 0
  "飞机电压(V)": 58.0,  // Voltagem do avião (V): 58.0
  "飞机电量(%)": 59,  // Carga do avião (%): 59
  "定点模式": "光流定点模式",  // Modo de ponto fixo: "Modo de fluxo óptico"
  "飞行模式": "手动操作",  // Modo de voo: "Operação manual"
  "偏航角(度)": 229.35,  // Ângulo de guinada (graus): 229.35
  "横滚角(度)": -0.18,  // Ângulo de rolo (graus): -0.18
  "俯仰角(度)": -0.28,  // Ângulo de inclinação (graus): -0.28
  "地磁环境干扰量": 0,  // Interferência do ambiente magnético terrestre: 0
  "加速度计校准": "1(校准结束) ",  // Calibração do acelerômetro: "1 (Calibração concluída)"
  "地磁水平校准": "1(校准结束) ",  // Calibração horizontal magnética: "1 (Calibração concluída)"
  "地磁垂直校准": "1(校准结束) ",  // Calibração vertical magnética: "1 (Calibração concluída)"
  "低电报警": "正常",  // Alarme de baixa voltagem: "Normal"
  "堵转报警": "正常",  // Alarme de travamento do motor: "Normal"
  "锁定状态态": "上锁"  // Estado de bloqueio: "Bloqueado"
```

Fonte: autoria própria

Figura 27: Código de extração de coordenadas do arquivo FlightLog



```python
def processar_ultimo_json(caminho_arquivo):
    try:
        with open(caminho_arquivo, 'r', encoding='utf-8') as arquivo:
            conteudo = arquivo.read().strip()

            # Separar os objetos JSON pelo delimitador
            objetos_json = conteudo.split('---------------------------------------')

            # Remover espaços extras e verificar se há objetos JSON
            objetos_json = [obj.strip() for obj in objetos_json if obj.strip()]

            if objetos_json:
                # Pegar o último objeto JSON
                ultimo_objeto = objetos_json[-1]

                # Processar o último objeto JSON
                try:
                    dados = json.loads(ultimo_objeto)

                    # Extrair longitude e latitude
                    longitude = dados.get("飞机经度", "Não disponível") #VÁRIAVEL QUE ARMAZENA A LONGITUDE
                    latitude = dados.get("飞机纬度", "Não disponível") #VÁRIAVEL QUE ARMAZENA A LATITUDE

                    print(f"Longitude: {longitude}")
                    print(f"Latitude: {latitude}")
                    #Envio dos valores para o banco de dados nos campos Latitude e Longitude
                    firebase = pyrebase.initialize_app(firebaseConfig)
                    dados = firebase.database()
                    dados.child().update({"Latitude": latitude})
                    dados.child().update({"Longitude": longitude})
```

Fonte: autoria própria

2.5.2. Criação dos pontos no Mapa

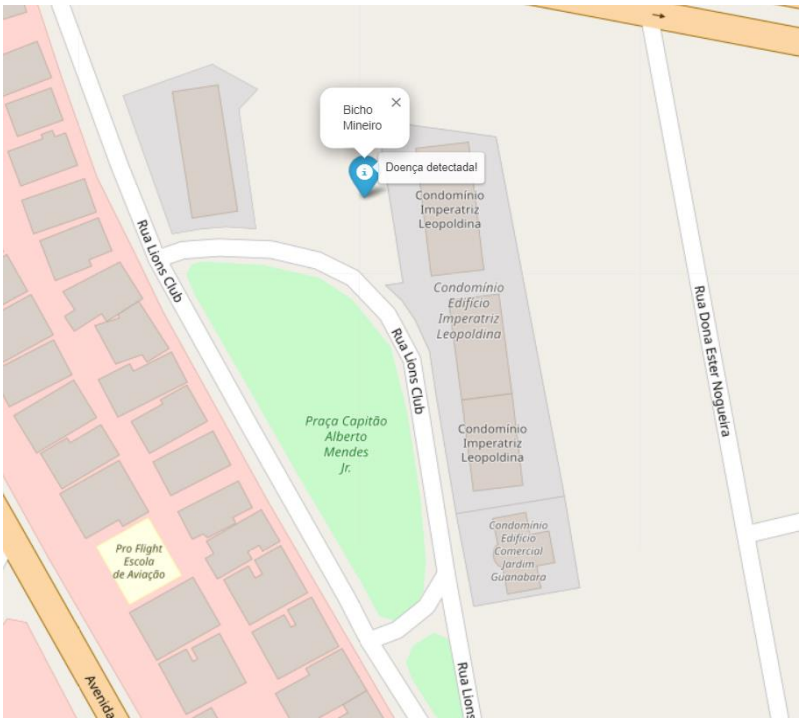
Para criar um ponto no mapa com base nessas coordenadas, o sistema faz com que, quando a IA detecta algo, o script de extração seja acionado. Assim, as últimas coordenadas do drone, onde algo foi detectado, são registradas no banco de dados. No código de criação do mapa, as variáveis latitude e longitude são usadas para recuperar os valores armazenados no banco de dados e criar um ponto no mapa. O mapa gerado também vem com o zoom máximo, facilitando a visualização de ruas e avenidas.

Figura 28: Código de criação dos pontos da doença

```python
# Inicializar o Firebase
firebase = pyrebase.initialize_app(firebaseConfig)
db = firebase.database()

# Ler dados do Realtime Database
data = db.child("/Localização").get()  # Substitua "your_path" pelo caminho correto
print(data.val())

# Verificar o tipo de dados retornados e acessar os valores
data_dict = data.val()  # Obtém os dados como um dicionário

if data_dict:
    latitude = data_dict.get("Latitude")
    longitude = data_dict.get("Longitude")

    if latitude is not None and longitude is not None:
        print(f"Latitude: {latitude}")
        print(f"Longitude: {longitude}")
    else:
        print("Latitude ou Longitude não encontrados.")
else:
    print("No data found at the specified path.")

m = folium.Map(location=[latitude, longitude], zoom_start=18)
folium.Marker(
    location=[latitude, longitude],
    tooltip="Doença detectada!",
    popup="Bicho Mineiro",
    icon=folium.Icon(color="blue"),
).add_to(m)

m.save("mapa.html")
```

Fonte: autoria própria

Figura 29: Visualização do mapa gerado com ponto da doença



Fonte: autoria própria

3. FUNDAMENTAÇÃO TEÓRICA

3.1. Agricultura Brasileira

O avanço da tecnologia no agronegócio no Brasil está se tornando uma necessidade cada vez maior para sustentar o mercado, e os drones são essenciais na melhoria das lavouras. Equipados com câmeras de alta resolução e sensores avançados, esses dispositivos não tripulados permitem que os agricultores monitorem cada hectare com um nível de detalhe e praticidade que não seria possível de maneira manual. Eles são capazes de mapear áreas, identificar focos de incêndio, monitorar sistemas de irrigação e até mesmo aplicar defensivos agrícolas de forma mais precisa e eficiente.

Nos últimos anos, o uso de drones no agronegócio brasileiro aumento consideravelmente. Em 2023, o número de drones registrados na Agência Nacional de Aviação Civil (ANAC) chegou a 2.287, um aumento de 266% em relação ao ano anterior. Além disso o agronegócio já responde por 25% do faturamento da indústria de drones do Brasil. A tendência é que essa participação continue crescendo, à medida que mais produtores rurais adotam essa tecnologia.

O aumento dos drones também é reflexo de uma mudança no campo. Uma pesquisa realizada pela Embrapa, em parceria com o Sebrae e o Inpe, mostrou que 84% dos agricultores brasileiros já utilizam algum tipo de tecnologia digital em suas atividades. Além dos drones, muitos produtores estão recorrendo à internet e às redes sociais para buscar informações, negociar insumos e vender sua produção. Essas ferramentas digitais estão facilitando o dia a dia no campo, permitindo uma comunicação mais ágil e o acesso a novos mercados.

Vendo de maneira global, de acordo com estimativas, o mercado de drones voltado para a agricultura deve quadruplicar nos próximos anos, passando de US$ 1,2 bilhão em 2019 para US$ 4,8 bilhões em 2024. Esse crescimento é impulsionado pela crescente demanda por alimentos e pela necessidade de tecnologias que aumentem a eficiência e a sustentabilidade das atividades agrícolas.

No Brasil, o mercado de drones é o maior da América Latina e o segundo maior das Américas, atrás apenas dos Estados Unidos. Com um faturamento anual de US$

373 milhões, sendo destacado como um dos países principais no setor de drones. A Embrapa e outras instituições têm incentivado a utilização dessas tecnologias, pois elas não só aumentam a produtividade, mas também reduzem os impactos ambientais e melhoram a administração das áreas rurais.

Para muitos agricultores, os drones deixaram de ser apenas uma tecnologia que apenas ajudaria e se tornaram ferramentas essenciais para o sucesso no campo. Eles ajudam a tomar decisões mais informadas e precisas, seja na escolha da melhor área de plantio ou no monitoramento da saúde das plantas. Com isso, os drones estão não apenas transformando o agronegócio brasileiro, mas também abrindo novas oportunidades para a inovação do setor.

3.1.1. Agricultura familiar

O Brasil se consolidou como um dos maiores produtores e exportadores agrícolas do mundo, tendo o setor como umas das principais atividades econômicas desenvolvidas no país.

Em grande maioria, os agricultores de grande porte focam nas exportações internacionais, que geral mais lucro, enquanto os agricultores familiares vendem seus cultivos internamente, sendo os responsáveis pela sustentação da maioria do país.

Tabela 1: Censo Agropecuário 2017, divulgado pelo IBGE

Censo Agro 2017						
Total, agricultura familiar	**Estabelecimentos**		**Área (ha)**		**Pessoal ocupado**	
Total	**5 073 324**	**100,0 %**	**351 289 816**	**100,0 %**	**15 105 125**	**100,0%**
NORMAS VIGENTES EM 2017						
Não é agricultura familiar	1 175 916	23,2%	270 398 732	77,0%	4 989 566	33,0%
Agricultura familiar	3 897 408	76,8%	80 891 084	23,0%	10 115 559	67,0%

Fonte: IBGE

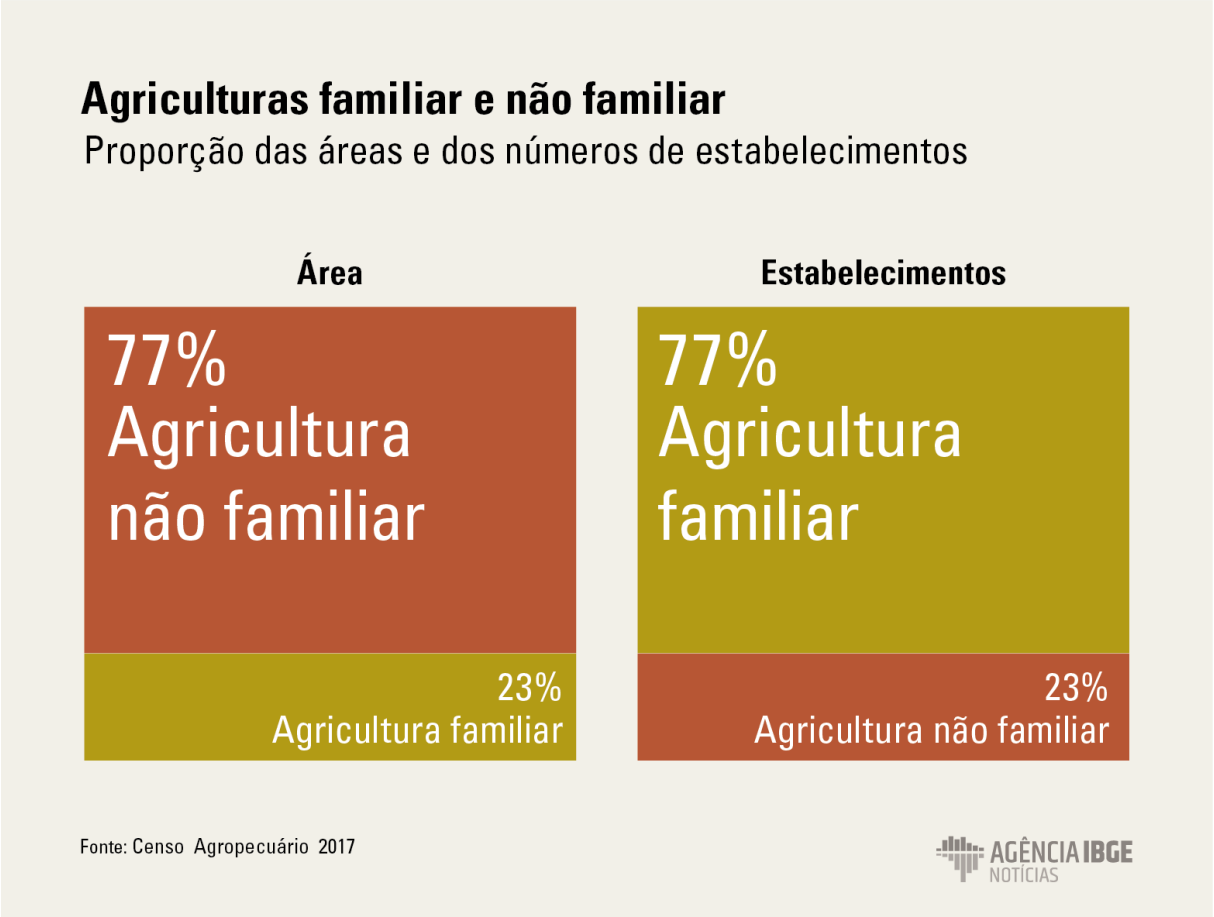
O Censo identificou 5.073.324 estabelecimentos agropecuários no Brasil, uma redução de 2,0% em relação a 2006. No entanto, a área total dos estabelecimentos cresceu 5,8%, atingindo 351.289.816 hectares.

De acordo com a tabela é perceptível ver a quantidade de produtores familiares, mesmo que os produtores familiares tenham áreas menores que agricultores de grandes portes, sua quantidade de estabelecimentos compensa seu tamanho.

Em 2017, 15,1 milhões de pessoas estavam ocupadas nos estabelecimentos agropecuários, uma queda de 1,5 milhão em comparação com 2006. Nos estabelecimentos da Agricultura Familiar, houve uma redução de 2,166 milhões de trabalhadores, enquanto nos familiares não, o número aumentou em 702,9 mil.

Dessa forma é perceptível que o crescimento das áreas não familiares enquanto a quantidade de estabelecimentos diminui e a mão de obra aumenta na agricultura familiar.

Figura 30: Censo Agropecuário 2017



Fonte: Censo Agropecuário 2017

No Censo Agropecuário de 2017, 3.897.408 estabelecimentos atenderam aos critérios e foram classificados como agricultura familiar, o que representa 77% dos estabelecimentos agropecuários levantados pelo censo. Ocupavam uma área de 80,9 milhões de hectares, ou seja, 23% da área total dos estabelecimentos agropecuários brasileiros. Em comparação aos grandes estabelecimentos, responsáveis pela produção de commodities (produtos básicos e primários que são produzidos em grande escala e têm pouco ou nenhum valor agregado) agrícolas de exportação.

3.1.2. Utilizando o café

Dentre os cultivos do Brasil, o café é um dos maiores valores de produção, segundo a IBGE (2022), sendo o Brasil o maior produtor e exportador de café do mundo. No entanto, todos os cultivos sofrem com desafios que podem provocar

problemas futuros ou maiores gastos que podem prejudicar sua produtividade, como base de estudo foi utilizado as doenças: Ferrugem do Café e o Bicho-Mineiro.

Segundo a EMBRAPA, a ferrugem do café é causada pelo fungo Hemileia vastatrix. Os sintomas da doença manifestam-se nas folhas da planta, onde surgem manchas amarelas, que gradualmente se tornam laranjas devido à produção de esporos do fungo. Enquanto a doença avança, as folhas infectadas caem prematuramente, reduzindo a capacidade da planta de realizar a fotossíntese e, consequentemente, a produção de grãos. O controle da ferrugem pode ser feito através das variedades de café que apresentem resistência, a utilização de fungicidas à base de cobre, fungicidas sistêmicos e ditiocarbamatos.

O bicho-mineiro do café, causado pela mariposa Leucoptera coffeella. As lagartas do inseto causam a mesma queda prematura das folhas. Como consequência, elas ficam mais fracas e sem folhas, a capacidade de fotossíntese das plantas reduz drasticamente, o que leva à queda da produtividade. A maneira de controle mais utilizada é o controle químico, mesmo que não seja muito ideal por causa de suas desvantagens. Alguns dos ingredientes mais utilizados são o Thiametoxan, Clorantraniliprole e o Cloridrato de Cartap.

4. PLANILHA DE CUSTOS DO PROJETO

Tabela 2: Planilha de custos do projeto

QTD	DESCRIÇÃO DO RECURSO	VALOR UNITÁRO (R$)	VALOR TOTAL (R$)	FONTE
01	Drone L900 PRO.	R$ 300	R$300,00	amazon.com.br
03	Técnico de informática júnior.	R$ 2.468,00	R$ 7.404,00	salario.com.br
03	IdeaPad 1 AMD Ryzen 3 7320U 4GB 256GB SSD Windows 11 Home 15,6" HD	R$ 2.023,99	R$ 6.071,97	lenovo.com
01	Google Colab Pro	R$ 58,00	R$ 58,00	colab.google
TOTAL			R$	13.833,97

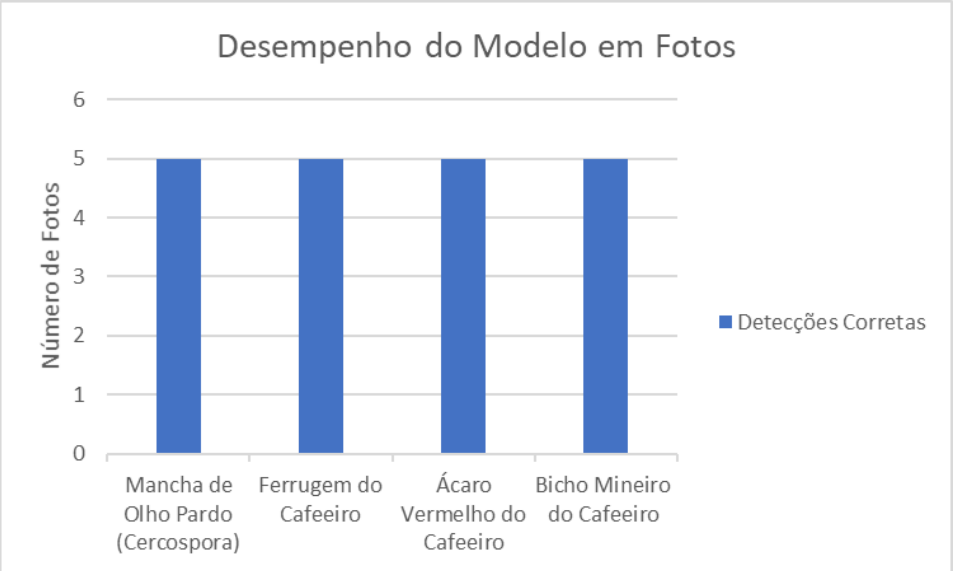
5. RESULTADOS E ANÁLISES DE DADOS

O primeiro modelo treinado neste projeto apresentou um desempenho satisfatório. Com base nos resultados deste primeiro teste, foi possível analisar o desempenho do modelo e identificar áreas que necessitam de ajustes no seu no treinamento para alcançar maior precisão e eficiência no modelo. Outro resultado obtido com êxito foi a extração das coordenadas do drone para sua marcação no mapa. As coordenadas são extraídas de um arquivo JSON gerado pelo aplicativo do drone, que atualiza a posição a cada 5 segundos.

5.1 Modelo em Foto

O modelo treinado com imagens apresentou alta eficiência. Para o treinamento, foram utilizadas 25 fotos, sendo 5 para cada uma das doenças: Mancha de Olho Pardo, Minador de Folhas, Ferrugem das Folhas e Ácaro Vermelho, além de 5 fotos de plantas de café saudáveis. O desempenho do modelo foi excelente nas imagens, com a detecção correta de todas as doenças e a ausência de falsas identificações nas plantas saudáveis. Ressalta-se que a detecção em fotos teve um ótimo desempenho devido à maior resolução das imagens e à ausência de fluxo contínuo de vídeo, que também contribui para a melhoria dos resultados.

Figura 31: Gráfico Referente ao Desempenho do Modelo em Fotos

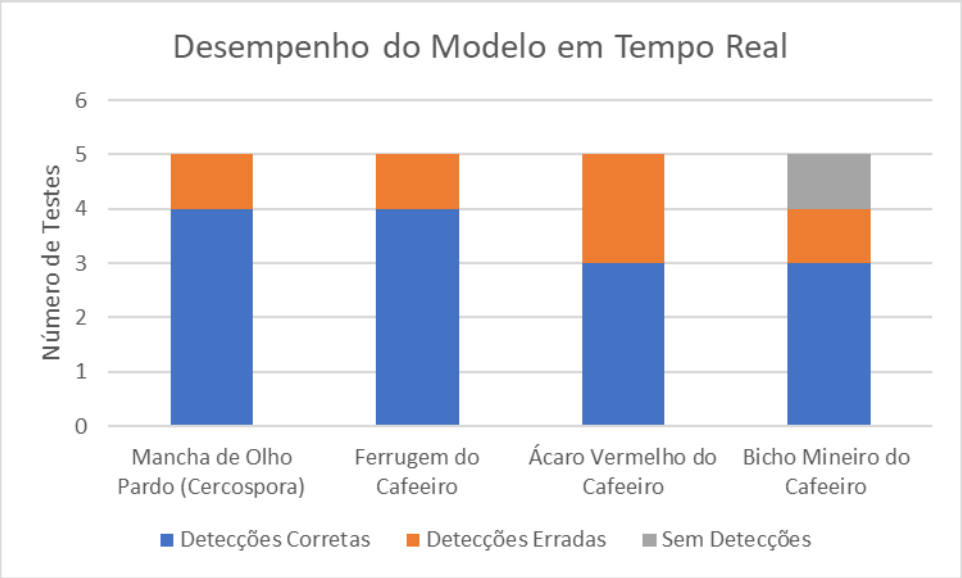


Fonte: autoria própria

5.2 Modelo em Tempo Real

O modelo em tempo real apresenta mais limitações em comparação com o modelo baseado em fotos. Seus requisitos computacionais são mais elevados e sua precisão tende a ser menor. Além disso, a distância de captura da imagem e o fluxo de vídeo são fatores que impactam significativamente o desempenho e a precisão do modelo. Todos esses aspectos contribuem para a redução da eficácia do modelo em tempo real.

Figura 32: Gráfico Referente ao Desempenho do Modelo em Tempo Real



Fonte: Autoria própria

Os testes realizados mostraram que a versão em tempo real apresentou um desempenho aceitável para um primeiro modelo. É importante destacar que a distância e o ângulo de captura influenciam a detecção. O modelo apresentou erros em algumas situações devido ao ângulo de captura e à distância, mas apresentou bons resultados quando as condições eram mais favoráveis. Melhorias no treinamento do modelo podem aumentar sua eficiência e precisão.

5.3 Mapeamento

O mapeamento está funcionando conforme o esperado. Testes realizados com a extração das coordenadas do drone possibilitam a marcação de pontos no mapa utilizando a biblioteca Folium. A precisão do GPS do drone é excelente, fornecendo coordenadas que refletem com exatidão suas posições.

Foi desenvolvido um código em Python para extrair as últimas coordenadas salvas no arquivo JSON, que são então atribuídas a variáveis em Python. Essas variáveis são, subsequentemente, inseridas no banco de dados e, a partir dele, transferidas para o mapa. Além disso, é possível personalizar o nome e a cor dos pontos no mapa, permitindo a criação de pontos distintos para cada tipo de doença, exibindo seus respectivos nomes.

6. DISCUSSÃO

O projeto visa auxiliar pequenos e médios agricultores no controle de pragas, oferecendo uma tecnologia acessível que considera suas limitações financeiras. O modelo de Inteligência Artificial foi desenvolvido para detectar quatro tipos de doenças na plantação de café e foi conectado a um drone de baixo custo. Além disso, as coordenadas do drone foram extraídas no momento da detecção, permitindo a criação de um mapa para que o agricultor visualize as áreas afetadas e o modelo de drone selecionado para este projeto possui funcionalidade de voo autônomo, possibilitada pelo seu sistema de GPS integrado.

O desenvolvimento deste sistema atingiu os objetivos propostos, incluindo a acessibilidade tecnológica, a detecção de pragas em tempo real e por meio de imagens estáticas, o voo autônomo do drone e o mapeamento das áreas afetadas. A implementação de uma tecnologia de baixo custo permitiu abordar a problemática inicial, proporcionando uma solução eficaz.

Os principais desafios enfrentados no projeto foram o poder computacional utilizado para o treinamento da Inteligência Artificial e a câmera do drone. Para melhorar o projeto, recomenda-se a escolha de um modelo de drone equipado com uma câmera de melhor qualidade e maior capacidade de zoom, o que aumentaria a precisão na captura de imagens e impactaria positivamente a eficácia da detecção. Outro aspecto relevante é o hardware utilizado para o treinamento do modelo de rede neural. O treinamento de uma Inteligência Artificial é uma tarefa que exige equipamentos de alto desempenho. A utilização de um dataset maior contribuiria para aumentar a precisão do modelo de IA reduzindo a probabilidade de falsas detecções ou erros de classificação. Contudo, à medida que o tamanho do dataset cresce, o tempo e os recursos computacionais também aumentam, tornando necessário o uso de hardware avançado para processar os dados e gerar um modelo mais preciso.

Assim, para aprimorar o projeto, recomenda-se investir em um drone com melhores especificações e um hardware mais eficiente para o treinamento do modelo de IA, o que resultaria em um sistema ainda mais preciso na detecção e controle de pragas nas plantações.

7. CONCLUSÕES

Os objetivos estabelecidos para o projeto foram atingidos com sucesso. O desenvolvimento do aplicativo e a integração da inteligência artificial com o drone possibilitaram a detecção em tempo real de doenças nas plantações. O sistema foi capaz de realizar o mapeamento das áreas afetadas, e o voo autônomo do drone foi viabilizado por meio do GPS integrado e das funcionalidades do aplicativo.

No entanto, o projeto encontrou limitações devido às características do drone utilizado. A qualidade da imagem capturada pelo drone não é alta, a duração da bateria é limitada e o drone não possui as funcionalidades avançadas presentes em modelos mais modernos. Essas limitações impactam a precisão na detecção de doenças, uma vez que uma câmera com maior resolução e um zoom mais eficiente poderiam melhorar a identificação das doenças. Além disso, drones mais modernos possuem maior capacidade de bateria, maior alcance e contém funcionalidades que permitem a conexão com a inteligência artificial de maneira mais simples e melhor.

Para aprimorar o sistema, recomenda-se a utilização de drones equipados com câmeras 4K ou Full HD e com maior duração de bateria. Esses aprimoramentos poderão aumentar a eficiência e a precisão do sistema, oferecendo melhores resultados para os agricultores.

Em síntese, o protótipo desenvolvido demonstrou o funcionamento do sistema proposto e estabeleceu uma base para futuras melhorias. O avanço para equipamentos mais modernos poderá levar a um sistema ainda mais robusto e eficaz para o monitoramento de doenças nas plantações.

REFERÊNCIAS BIBLIOGRÁFICAS

ADAMI, J. M. M.; SOUZA, F. L. P. **Análise de imagens coletadas por drones para identificação da soja**, 2021. Disponível em:<https://ric.cps.sp.gov.br/handle/123456789/6473> Acesso em: 07 fev. 2024.

BITENCOURT, V. C. C. et al. **USO DE ALGORITMOS DE CLASSIFICAÇÃO PARA IDENTIFICAÇÃO DA SIGATOKA-AMARELA ATRAVÉS DE IMAGENS DE VANT**. Disponível em: <https://proceedings.science/sbsr-2019/trabalhos/uso-de-algoritmos-de-classificacao-para-identificacao-da-sigatoka-amarela-atrave?lang=pt-br>. Acesso em: 17 fev. 2024.

BUAINAIN, A. M.; CAVALCANTE, P.; CONSOLINE, L. **Estado atual da agricultura digital no Brasil: inclusão dos agricultores familiares e pequenos produtores rurais**. 2021. Disponível: <https://hdl.handle.net/11362/46958> Acesso em: 02 fev. 2024.

CASTRO, G. D. M. **Modelo para monitoramento remoto da ferrugem do cafeeiro utilizando Machine Learning.** 2022. 33 f. 2022. Disponível em: <https://locus.ufv.br//handle/123456789/30489>. Acesso em: 22 fev. 2024.

DIAS, B. G. L.; COELHO, A. D. **Inovação no agronegócio utilizando IoT**. Escola de Engenharia Mauá. São Caetano do Sul, 2019. Disponível em: <https://maua.br/files/122019/inovacao-agronegocio-utilizando-iot-261145.pdf>. Acesso em: 01 fev. 2024.

EUSTÁQUIO, J. V. F. **Distribuição produtiva e tecnológica dos estabelecimentos agropecuários de menor porte e gestão familiar no Brasil.** Disponível em: <researchgate.net/publication/263697445_Distribuicao_produtiva_e_tecnologica_dos_estabelecimentos_agropecuarios_de_menor_porte_e_gestao_familiar_no_Brasil>. Acesso em: 02 fev. 2024.

FERRAZ, J. V. D. **TREINAMENTO DE UMA REDE NEURAL PARA DETECTAR RACHADURAS.** Disponível em: <https://repositorio.unesp.br/server/api/core/bitstreams/77ebcb98-726f-489b-b344-b4e397cd47c2/content>. Acesso em: 4 ago. 2024.

LEMES, D. P.; BRESCIANI, D. G. **A agricultura familiar no município de Juína: "uma análise de caso dos produtores da APROFEJU".** Revista Científica da Ajes, v. 1, n. 2, 2010. Disponível em: <https://www.revista.ajes.edu.br/index.php/rca/article/view/57>. Acesso em: 02 fev. 2024.

LIMA, N. L. **Implementação de banco de imagens para classificação de produtos agrícolas através de Machine Learning.** Disponível em: <https://hdl.handle.net/20.500.12733/5989>. Acesso em: 17 fev. 2024.

NASCIMENTO, G. P. **Classificação de tipos de parafusos e porcas com a utilização da rede YOLOv8.** Disponível em: <https://repositorio.ifes.edu.br/handle/123456789/4460>. Acesso em: 17 ago. 2024.

NVIDIA. **What is Convolution?** Disponível em: <https://developer.nvidia.com/discover/convolution>. Acesso em: 26 ago. 2024.

OLIVEIRA, G. F. C. **Detecção de plantas daninhas em lavouras de mandioca utilizando rede neural em imagens aéreas obtidas por VANT.** Disponível em: < https://repositorio.ifgoiano.edu.br/handle/prefix/4348>. Acesso em: 27 jul. 2024

OLIVEIRA, V. L. C. de. **Estudo dos Agronegócios 4.0 – Tecnologias, desafios e benefícios nos Agronegócios.** Disponível em: <https://rsdjournal.org/index.php/rsd/article/view/35379>. Acesso em: 01 fev. 2024.

PAZOTI, M. A. **CITRUSVIS - Um sistema de visão computacional para a identificação do fungo Guignardia citricarpa, causador da mancha preta em citros. 2005.** Disponível em: <doi:10.11606/D.55.2017.tde-15122017-145610>. Acesso em: 14 fev. 2024.

SILVA, D. A. **Método de fitopatometria utilizando processamento de imagens e Inteligência Artificial para detecção e quantificação de sintomas e sinais de Ferrugem Asiática da Soja aplicado ao melhoramento genético.** Disponível em: <DOI http://doi.org/10.14393/ufu.te.2022.5043>. Acesso em: 24 fev. 2024.

SINGH, A. et al. **Machine learning for high-throughput stress phenotyping in plants.** Trends in plant science, v. 21, n. 2, p. 110-124, 2016. Disponível em: <cell.com/trends/plant-science/fulltext/S1360-1385(15)00263-0>. Acesso em: 17 fev. 2024.

TRINDADE, L. D. G**. Investigando técnicas de processamento de imagens com IA na detecção de ferrugem em folhas de soja**. Disponível em: <https://repositorio.unipampa.edu.br/jspui/handle/riu/6751.>. Acesso em: 25 fev. 2024.

ULTRALYTICS. **Ultralytics Documentation.** Disponível em: <https://docs.ultralytics.com/>. Acesso em: 10 jul. 2024.

VASCONCELOS, G. C. **Identificação da praga bicho-mineiro em plantações de café usando imagens aéreas e Deep Learning.** Disponível em: <https://repositorio.ufu.br/handle/123456789/26175>. Acesso em: 15 fev. 2024.