INSTITUTO FEDERAL DE SANTA CATARINA

VINICIUS FIGUEIRÓ TONINI

Desenvolvimento de um sistema de detecção de pragas em soja utilizando técnicas de aprendizado de máquina

São José - SC

Julho/2025

DESENVOLVIMENTO DE UM SISTEMA DE DETECÇÃO DE PRAGAS EM SOJA UTILIZANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA

Monografia apresentada ao Curso de Engenharia de Telecomunicações do campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheiro de Telecomunicações.

Orientador: Prof. Ramon Mayor Martins, Dr.

Coorientador: Prof. Mario de Noronha Neto, Dr.

São José - SC

Julho/2025

Vinicius Figueiró Tonini

Desenvolvimento de um sistema de detecção de pragas em soja utilizando técnicas de aprendizado de máquina

Este trabalho foi julgado adequado para obtenção do título de Engenheiro de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, 28 de julho de 2025:

Documento assinado digitalmente
RAMON MAYOR MARTINS
Data: 20/08/2025 09:17:14-0300
CPF: ***.866.756-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Ramon Mayor Martins, Dr.
Orientador
Instituto Federal de Santa Catarina

Documento assinado digitalmente
MARIO DE NORONHA NETO
Data: 02/09/2025 06:29:17-0300
CPF: ***.859.519-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Mario de Noronha Neto, Dr.
Coorientador
Instituto Federal de Santa Catarina

Documento assinado digitalmente
ARLIONES STEVERT HOELLER JUNIOR
Data: 20/08/2025 10:15:23-0300
CPF: ***.699.639-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Arliones Stevert Hoeller Junior, Dr.
Instituto Federal de Santa Catarina

Documento assinado digitalmente
DIEGO DA SILVA DE MEDEIROS
Data: 20/08/2025 14:49:20-0300
CPF: ***.242.729-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Diego da Silva de Medeiros, Dr.
Instituto Federal de Santa Catarina

À minha esposa e família e a todos que acreditaram em mim,
com empenho, fé e perseverança, é possível transformar ideias em realizações.

AGRADECIMENTOS

Agradeço, com todo meu carinho à minha esposa Brenda, por seu companheirismo incondicional. Obrigado por estar ao meu lado em todos os momentos, por me motivar, acreditar em mim e me ajudar ao longo desta caminhada.

À minha mãe, Cleonice, sou profundamente grato por todo o apoio, dedicação e incentivo ao longo da graduação — foi graças à sua força que consegui chegar até aqui. À minha irmã, Karen, agradeço por estar sempre ao meu lado, me apoiando nos momentos difíceis e me encorajando a seguir em frente.

Ao meu orientador, professor Ramom, agradeço por aceitar o desafio de conduzir este trabalho mesmo com o projeto já em andamento. Sua disponibilidade e comprometimento foram fundamentais para que eu pudesse concluir esta etapa com confiança de um bom trabalho realizado.

Agradeço especialmente ao professor Mario Noronha, que esteve presente desde o início, contribuindo decisivamente para o amadurecimento da proposta. Mesmo precisando se afastar posteriormente, sua orientação foi essencial para dar forma e clareza ao trabalho como ele é hoje.

Sou muito grato aos meus amigos Matheus e Mike, companheiros desde o início da graduação. Por todas as tardes e noites estudando, trocando ideias e enfrentando os desafios juntos — meu sincero obrigado por cada aprendizado compartilhado.

Ao meu amigo João, agradeço pelas contribuições valiosas, pelas sugestões de melhoria e revisões criteriosas que tanto acrescentaram neste trabalho. Aprendi muito contigo.

Aos colegas do Inova, agradeço pelas conversas, pelas risadas e pelas trocas de conhecimento ao longo desses anos.

Aos meus professores, deixo meu reconhecimento e gratidão, especialmente aos professores Roberto e Emerson, que me acompanharam no projeto de pesquisa e com quem pude aprender profundamente. Tenho certeza de que fui privilegiado com um ensino de alta qualidade.

Por fim, agradeço à minha família e a todos que, de alguma forma, contribuíram para que este trabalho se tornasse possível.

What we want is a machine that can learn from experience.
Alan Turing

RESUMO

O agronegócio no Brasil é fundamental para o crescimento econômico e movimenta trilhões de reais a cada ano. Esse avanço é impulsionado pela exportação da soja e pela adoção de soluções inovadoras que aumentam a produtividade e a eficiência no campo. O Brasil se destaca como líder na exportação da soja, atendendo a uma demanda global crescente por esse grão. Nesse contexto, este trabalho teve como objetivo avaliar a viabilidade técnica de uma solução baseada em Inteligência Artificial (IA) para auxiliar na identificação de pragas em lavouras de soja. Para isso, foi desenvolvido um protótipo funcional composto por um modelo de aprendizado de máquina treinado com imagens de plantas de soja afetadas por pragas e exemplares saudáveis, integrado a um aplicativo móvel que permite que usuários enviem fotos e recebam previsões em tempo real. A proposta demonstra, em ambiente acadêmico, o potencial de integração entre IA e dispositivos móveis no monitoramento agrícola. Como resultado, obteve-se um sistema funcional que pode servir de base para estudos futuros e validação de tecnologias semelhantes em aplicações reais.

Palavras-chave: Inteligência Artificial. Agronegócio. Detecção de pragas. Soja.

ABSTRACT

Agribusiness plays a central role in Brazil's economic development, with soybean cultivation being one of its most significant pillars. This growth is driven by high global demand for soybeans and by the adoption of innovative technologies that enhance productivity and optimize farming practices. In this context, this work proposes and evaluates the technical feasibility of a solution based on Artificial Intelligence (AI) to support the identification of pests in soybean crops. A functional prototype was developed, consisting of a machine learning model trained on a dataset composed of images of both healthy and pest-affected soybean plants. This model was integrated into a mobile application, allowing users to submit images and receive real-time predictions. The system showcases, within an academic setting, the potential of integrating AI with mobile technologies for agricultural monitoring. The resulting application offers a functional baseline that may inform future research and the validation of similar technologies in real-world scenarios.

Keywords: Artificial Intelligence. Agribusiness. Pest detection. Soybeans.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface.*

AutoML *Automated Machine Learning.*

AWS *Amazon Web Services.*

CNN *Convolutional Neural Network.*

CORS *Cross-Origin Resource Sharing.*

EC2 *Amazon Elastic Compute Cloud.*

GPUs *Graphics Processing Unit.*

IA inteligência artificial.

JSON *JavaScript Object Notation.*

JWT *JSON Web Token.*

LLMs *Large Language Models.*

ResNet *Residual Neural Network.*

Rest *Representational State Transfer.*

RNAs Redes Neurais Artificiais.

SDK *Software Development Kit.*

XML *Extensible Markup Language.*

SUMÁRIO

1 INTRODUÇÃO

A agricultura brasileira é um setor econômico que contribui significativamente para o desenvolvimento do país. Segundo a Embrapa (2020), representa cerca de 21% do PIB nacional, configurando-se como uma das maiores contribuições econômicas, por englobar a soma de todas as riquezas produzidas. Atualmente, é responsável por um quinto dos empregos e por 43,2% das exportações, além de ser um dos poucos segmentos da economia brasileira que vêm mantendo crescimento positivo nos últimos anos. Dessa forma, a Embrapa (2020) afirma que, segundo modelos matemáticos, a produção de grãos no Brasil pode superar 318 milhões de toneladas até 2030, representando um aumento de 68 milhões em relação à produção atual, o que reforça a competitividade do país nesse segmento.

O crescimento econômico observado no agronegócio deve-se à ampla modernização do campo, que possibilita o aumento da eficiência dos processos na lavoura. Segundo Aires (2023), é por meio das inovações tecnológicas que se torna possível reduzir custos, aumentar a produtividade e garantir maior rentabilidade para o negócio rural. A utilização de tecnologias capazes de fornecer dados sobre clima, solo, entre outros, auxilia significativamente na tomada de decisão do agricultor e na redução de desperdícios. Portanto, com o emprego de dispositivos de inteligência artificial no monitoramento das plantas, a tendência é proporcionar uma visão mais clara da saúde da lavoura e, consequentemente, aumentar a produtividade da plantação.

Os avanços tecnológicos e o crescimento econômico têm como destaque a soja. As exportações desse grão geram uma receita que supera os dez bilhões de dólares, representando cerca de 8% do total exportado pelo país. Estima-se que um em cada quatro dólares provenientes do agronegócio brasileiro esteja relacionado à soja (DALL'AGNOL et al., 2021). No Brasil moderno, a soja pode ser comparada aos ciclos econômicos da cana-de-açúcar, da borracha e do café, que, em diferentes períodos dos séculos XVII a XX, dominaram o comércio exterior do país (DALL'AGNOL et al., 2021). De acordo com Dall'Agnol et al. (2021), há uma crescente demanda mundial por soja, e o Brasil, como principal exportador, tende a manter a liderança nas exportações globais por muitos anos — se é que será superado algum dia — tendo em vista o seu potencial de se expandir nesse setor.

No contexto do agronegócio brasileiro, este projeto busca introduzir inovações relevantes em um setor de grande importância econômica. A presença de pragas ao longo de todo o ciclo produtivo da soja — do plantio à colheita — é uma das principais preocupações, pois pode comprometer a sanidade das lavouras, reduzir a produtividade e causar

prejuízos econômicos significativos (FERREIRA; CAMPO; GóMEZ, 2014). Diante desse cenário, desenvolveu-se uma solução baseada em inteligência artificial (IA), com o objetivo de detectar pragas em plantações por meio da análise de imagens utilizando técnicas de visão computacional. Este trabalho apresenta o desenvolvimento de um protótipo funcional capaz de realizar essa detecção com aprendizado de máquina, integrado a uma aplicação móvel que permite testes e validações em contexto acadêmico.

1.1 Objetivo geral

O objetivo geral deste trabalho foi desenvolver um sistema de detecção de pragas em sojas utilizando uma abordagem centrada em dados e técnicas de aprendizado de máquina.

1.2 Objetivos específicos

Através do objetivo geral, foram definidos os seguintes objetivos específicos.

- Selecionar e organizar uma base de dados representativa, contendo imagens de plantas saudáveis e com presença de pragas.
- Realizar o pré-processamento das imagens para otimizar o desempenho do modelo de aprendizado de máquina.
- Treinar modelos de classificação utilizando a biblioteca FastAI, com foco em acurácia e generalização.
- Desenvolver uma aplicação móvel funcional capaz de capturar imagens e consultar o modelo em um servidor remoto.
- Integrar os componentes do sistema (modelo, *backend* e aplicativo) em um fluxo completo de inferência.
- Validar o funcionamento do protótipo por meio de testes práticos de envio, inferência e retorno de resultados.

1.3 Organização do texto

Este trabalho está organizado da seguinte forma: no Capítulo 2 aborda a base teórica necessária para o entendimento e desenvolvimento deste trabalho. No Capítulo 3 é apresentado o desenvolvimento deste projeto. No Capítulo 4 são apresentados os resultados obtidos. Por fim, no Capítulo 5 descreve as conclusões a respeito do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo são apresentadas as revisões bibliográficas dos principais conceitos que serão abordados, assim como estão relacionados ao desenvolvimento do projeto descrito no Capítulo 3.

2.1 Inteligência Artificial

A inteligência artificial (IA) refere-se à capacidade dos computadores de realizarem tarefas que normalmente requerem inteligência humana, como raciocínio, tomada de decisão, reconhecimento de padrões e aprendizado com a experiência (MILLINGTON, 2006; HONAVAR, 2016). Diferentemente dos seres humanos, os computadores podem executar essas tarefas com uma velocidade significativamente maior e com alta eficiência (MORETI et al., 2021). Essa combinação de velocidade, precisão e capacidade de aprendizado torna a IA uma ferramenta poderosa para resolver problemas complexos em diversos domínios.

De acordo com Norvig e Russell (2013), a IA não é um conceito recente. Ainda na década de 1950, Alan Turing propôs o chamado Teste de Turing, o qual avalia se uma máquina pode ser considerada inteligente com base na sua capacidade de fornecer respostas indistinguíveis das de um ser humano. Esse marco estabeleceu as bases para o desenvolvimento de áreas como o processamento de linguagem natural, representação de conhecimento, raciocínio automatizado e aprendizado de máquina. Embora a IA seja um ramo da computação existente há várias décadas, seu crescimento se intensificou nos últimos anos com o avanço de tecnologias como a computação em nuvem e o *Big Data* (MORETI et al., 2021).

A aplicação da inteligência artificial na agricultura tem se concentrado principalmente na automação de processos de diagnóstico e análise de dados. Para a detecção de pragas em culturas, as redes neurais convolucionais oferecem uma abordagem promissora ao automatizar a identificação visual de insetos e sintomas, reduzindo a dependência de diagnósticos manuais especializados (PINHEIRO et al., 2021).

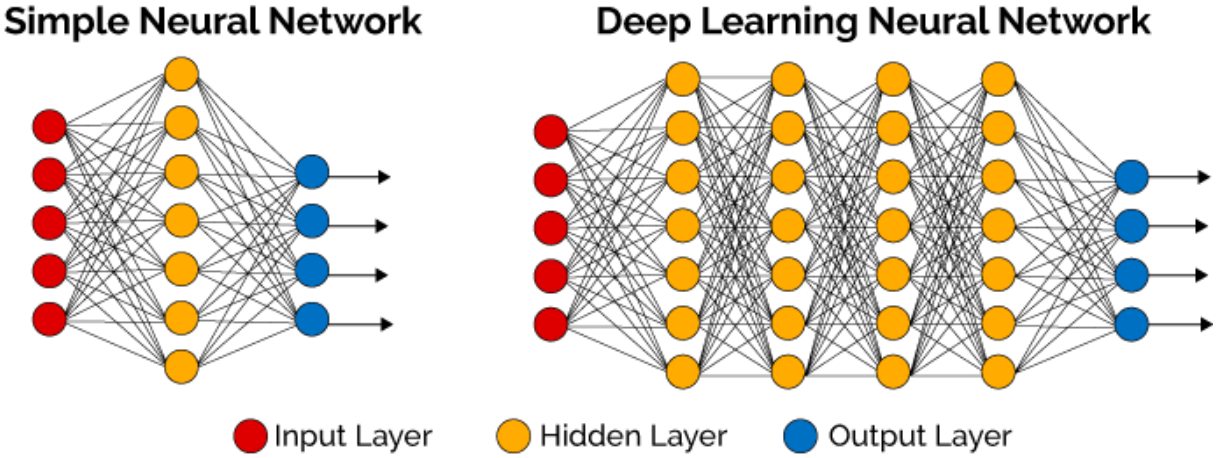
Redes neurais profundas

O aprendizado profundo, ou *deep learning*, é uma subárea do aprendizado de máquina (*machine learning*) baseada em redes neurais artificiais com múltiplas camadas ocultas e grande número de parâmetros (SHINDE; SHAH, 2018). De acordo com Patterson e Gibson (2017), os avanços recentes em *deep learning* foram impulsionados principalmente pela evolução do *hardware*, como o aumento do poder de processamento das *Graphics Processing Unit* (GPUs), o que possibilitou a criação de redes neurais mais profundas

— também conhecidas como *Deep Neural Networks (DNNs)* — e, consequentemente, a obtenção de resultados significativamente superiores em tarefas complexas, como o reconhecimento de imagens e o processamento de linguagem natural.

No geral, o aprendizado profundo utiliza uma cascata de unidades de processamento organizadas em múltiplas camadas. As camadas mais próximas da entrada de dados são responsáveis por aprender representações simples, como bordas e formas básicas, enquanto as camadas mais profundas aprendem características cada vez mais complexas, construídas a partir das representações das camadas anteriores. Essa arquitetura em camadas possibilita a formação de uma representação hierárquica e robusta dos dados, permitindo a extração de conhecimento útil mesmo em grandes volumes de informação (SHINDE; SHAH, 2018).

Figura 1 – Rede Neural Simples e Rede Neural Profunda (*Deep Learning)*



Fonte: (ACADEMY, 2022b).

Na Figura 1, apresenta-se uma representação esquemática de duas arquiteturas de Redes Neurais Artificiais (RNAs): uma rede neural simples, à esquerda, e uma rede neural profunda, à direita. Os círculos vermelhos indicam as entradas da rede, enquanto os azuis representam as saídas. Já os círculos amarelos representam os neurônios que compõem as camadas ocultas.

2.1.1 Arquiteturas de redes neurais consolidadas

Esta seção apresenta algumas das arquiteturas de redes neurais consolidadas encontradas na literatura, com ênfase naquelas relevantes para o trabalho aqui desenvolvido. Para uma apresentação mais abrangente, recomenda-se a consulta às páginas do *The Asimov Institute* ("The Neural Network Zoo") (LEIJNEN STEFAN, 2019) e da *Data Science Academy* ("Deep Learning Book") (ACADEMY, 2022a), onde uma variedade maior de arquiteturas é descrita. As arquiteturas consideradas mais adequadas a este trabalho são: *Convolutional Neural Network* (CNN) e *Residual Neural Network* (ResNet).

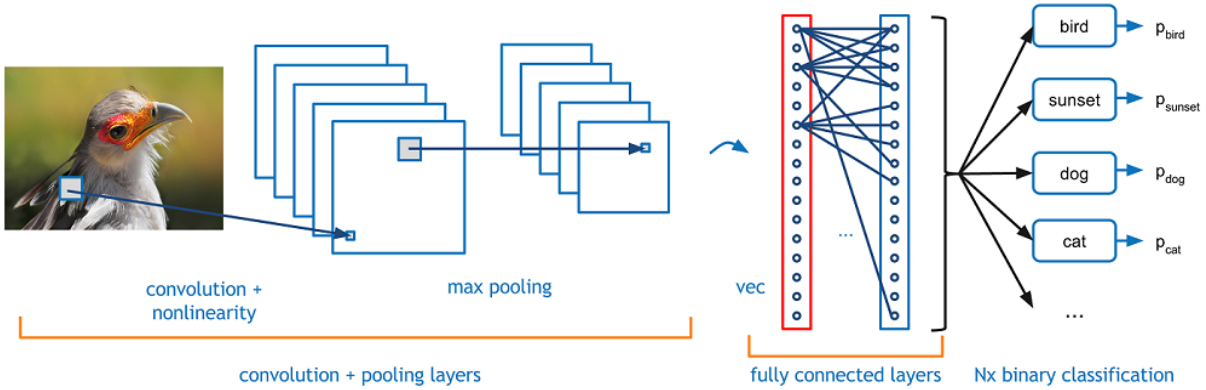
2.1.1.1 Rede Neural Convolucional (CNN)

A arquitetura de rede neural convolucional (CNN) apresenta alto desempenho em áreas relacionadas à visão computacional, como o reconhecimento de imagens e a detecção de objetos (PANG et al., 2018). Elas são organizadas em camadas sucessivas de convolução e *pooling*, onde a convolução aplica filtros para extrair características da imagem, enquanto o *pooling* reduz a dimensionalidade dos dados preservando informações relevantes.

Segundo Pang et al. (2018), CNNs são mais fáceis de treinar que outras redes profundas por utilizarem o algoritmo de retropropagação para ajuste dos pesos e apresentarem conectividade dispersa, na qual cada neurônio conecta-se apenas a uma pequena região da entrada. O compartilhamento de pesos entre diferentes regiões da imagem reduz significativamente a quantidade de parâmetros a serem aprendidos e melhora a capacidade de generalização do modelo.

A Figura 2 ilustra o funcionamento completo de uma CNN para classificação de imagens. A imagem de entrada passa por múltiplas camadas de convolução que detectam características como bordas e texturas, seguidas por camadas de *max pooling* que selecionam valores máximos de cada região, reduzindo progressivamente as dimensões espaciais. A transformação indicada pela seta "vec"converte os mapas bidimensionais em um vetor unidimensional que alimenta as camadas totalmente conectadas, destacadas pelo retângulo vermelho. Finalmente, a camada de saída realiza a classificação múltipla, atribuindo probabilidades a cada classe possível, sendo a maior probabilidade considerada a predição final (MURPHY, 2016).

Figura 2 – Exemplo de *Convolutional Neural Network* (CNN)



Fonte: (ACADEMY, 2022a).

2.1.1.2 Rede Neural Residual (ResNet)

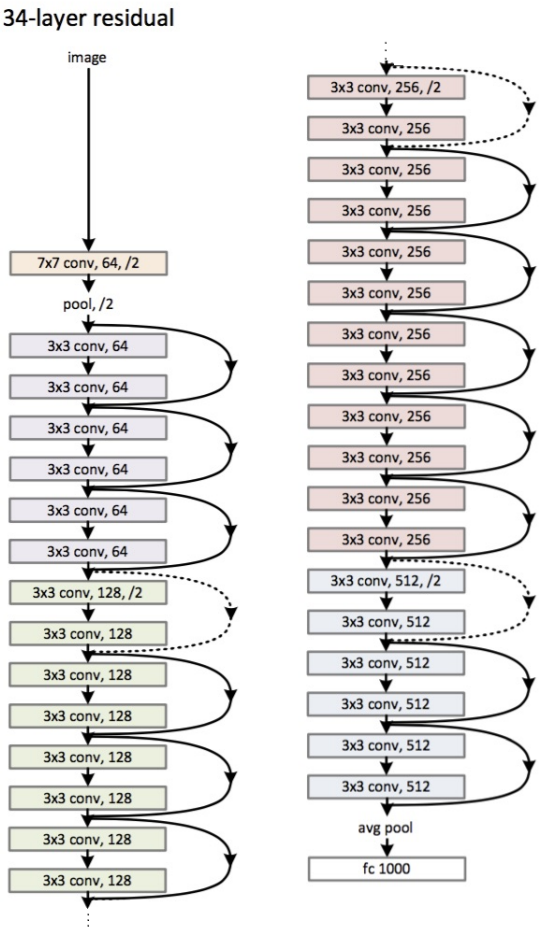
A arquitetura ResNet, baseada em redes neurais convolucionais, foi projetada para tarefas de visão computacional como classificação de imagens e detecção de objetos. Intro-

duzida por He et al. (2016), tornou-se o modelo mais utilizado na área de processamento de imagens segundo Howard e Gugger (2020). Sua principal característica é a composição por diversas camadas profundas, permitindo maior capacidade de extração de características complexas.

O diferencial da ResNet está nas conexões residuais ou *skip connections*, que permitem o fluxo direto de informações através da rede, contornando algumas camadas intermediárias. Essa inovação resolve o problema do desvanecimento do gradiente, fenômeno que dificulta o treinamento de redes muito profundas ao enfraquecer progressivamente os sinais de erro durante a retropropagação (HOWARD; GUGGER, 2020).

A Figura 3 apresenta a arquitetura ResNet com 34 camadas, onde as setas curvas ilustram as conexões residuais que caracterizam essa arquitetura. Essas conexões permitem que a informação contorne camadas intermediárias, facilitando o treinamento de redes profundas e melhorando significativamente o desempenho. Para o desenvolvimento do modelo de IA com a biblioteca FastAI, essa arquitetura será adotada devido à sua eficiência comprovada.

Figura 3 – Arquitetura da *Residual Neural Network* (ResNet) com 34 camadas



Fonte: Adaptado de (HE et al., 2016).

2.1.2 Aprendizado por transferência

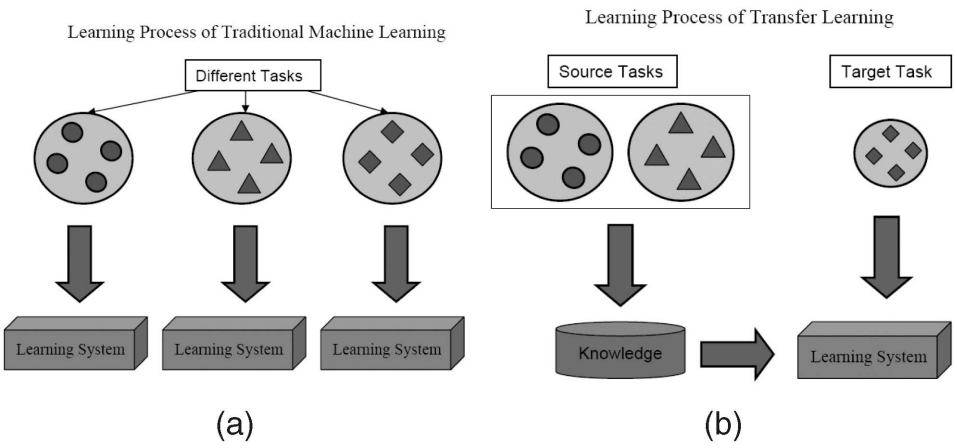
O aprendizado de máquina vem sendo aplicado com sucesso em diversas áreas e tem despertado o interesse tanto da indústria quanto da academia. Seu principal objetivo é extrair e aprender padrões a partir dos dados, a fim de estimar resultados futuros. No entanto, as abordagens tradicionais de aprendizado de máquina frequentemente enfrentam dificuldades ao lidar com conjuntos de dados rotulados e não rotulados que compartilham o mesmo espaço de características (NGUYEN et al., 2022).

As técnicas convencionais de aprendizado de máquina apresentam desempenho limitado em tarefas de previsão e classificação quando há escassez de dados de treinamento. Em muitos casos, a coleta desses dados pode ser extremamente complexa ou até inviável (NGUYEN et al., 2022).

Uma solução para esses problemas é o aprendizado por transferência. Diferentemente do aprendizado de máquina tradicional, seu principal fundamento é a transferência de conhecimento de uma tarefa fonte para uma tarefa alvo relacionada, a fim de melhorar o desempenho do aprendizado (NGUYEN et al., 2022). A motivação por trás dessa abordagem está na possibilidade de resolver novos problemas de forma mais rápida e eficiente, utilizando conhecimentos previamente adquiridos. Por exemplo, indivíduos que já aprenderam a tocar violino tendem a aprender violão com maior facilidade do que aqueles sem experiência prévia com instrumentos musicais (NGUYEN et al., 2022; WANGENHEIM, 2018).

Na Figura 4, observa-se a diferença entre os processos de aprendizado de máquina tradicional e o aprendizado por transferência. Neste último, são utilizados modelos previamente treinados para auxiliar no aprendizado de novas tarefas, aproveitando o conhecimento já adquirido (PAN; YANG, 2010).

Figura 4 – Diferença entre os processos de (a) aprendizado de máquina tradicional e (b) aprendizado de máquina por transferência



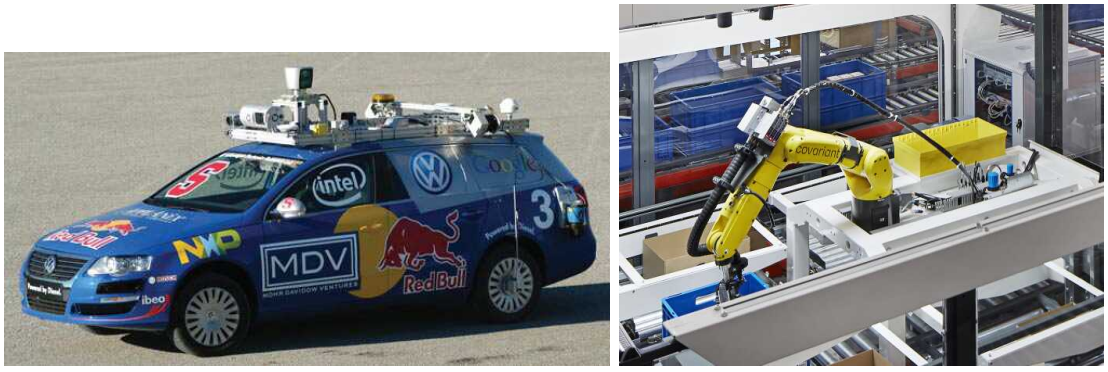
Fonte: (PAN; YANG, 2010).

2.1.3 Visão computacional

A visão computacional é o processo de replicação da visão humana por meio de software e hardware (MARENGONI; STRINGHINI, 2009). Esse processo geralmente requer etapas de pré-processamento das imagens, que precisam ser convertidas para formatos ou tamanhos específicos, além de passarem por filtragens para a remoção de ruídos originados durante a aquisição. As fontes desses ruídos são diversas, incluindo o tipo de sensor utilizado, a iluminação do ambiente e as condições climáticas no momento da captura (MARENGONI; STRINGHINI, 2009).

Na Figura 5, são apresentados alguns exemplos do uso da visão computacional no cotidiano, como veículos autônomos e robôs industriais. Além dessas aplicações, a tecnologia também é empregada na identificação de doenças por meio de exames de raio-X, na leitura de sinais de trânsito e no reconhecimento de pedestres (SZELISKI, 2022).

Figura 5 – Algumas aplicações industriais de visão computacional



(a) Veículo autônomo (b) Sistema robótico de separação em armazém

Fonte: (SZELISKI, 2022).

Até poucos anos atrás, a criação e operação de redes neurais artificiais era impraticável devido à elevada demanda computacional. No entanto, com os avanços no processamento paralelo, no uso de *Big Data* e na aplicação de CNN, houve um aumento significativo na taxa de precisão em tarefas de visão computacional. Como resultado, surgiram sistemas com níveis de reconhecimento e classificação de imagens comparáveis aos realizados por seres humanos (ACADEMY, 2022c).

Embora ainda haja muito a ser feito para que os algoritmos processem a visão da mesma forma que os seres humanos, o rápido avanço das soluções em visão computacional indica que esta é uma das áreas mais promissoras da IA (ACADEMY, 2022c).

2.2 Fast AI

A biblioteca fastAI, desenvolvida para a linguagem *Python*, tem como objetivos principais a facilidade de uso, flexibilidade e produtividade, sendo altamente configurável. Sua estrutura é baseada em uma hierarquia de APIs modulares, onde as APIs de baixo nível funcionam como blocos de construção reutilizáveis. Isso permite que usuários avancem além da interface de alto nível, personalizando ou estendendo o comportamento padrão de acordo com as necessidades específicas do projeto (HOWARD; GUGGER, 2020).

A fastAI oferece uma interface de alto nível sobre o PyTorch que permite a criação rápida de modelos complexos com poucas linhas de código, mantendo a flexibilidade para ajustes mais finos conforme necessário (HOWARD; GUGGER, 2020). A seguir, apresenta-se no Código 2.1 um trecho condensado do código utilizado para treinar um modelo com imagens da cultura da soja, classificando-as entre as categorias "praga" e "saudável". O código completo encontra-se disponível no Apêndice C.

O processo tem início na preparação dos dados por meio da função `ImageDataLoaders.from_name_func`, responsável por criar um conjunto de dados rotulado a partir dos nomes dos arquivos. Em seguida, o modelo é construído com a função `vision_learner`, utilizando a arquitetura `resnet34` e métricas como `accuracy` e `error_rate`. O treinamento ocorre com a função `fine_tune`, que aplica aprendizado por transferência sobre um modelo previamente treinado.

Após essa etapa, a função `learn.predict` permite a inferência da classe de novas imagens, enquanto `show_results` e `plot_confusion_matrix` possibilitam visualizar o desempenho do modelo, como os acertos e erros de classificação.

As imagens resultantes do processo de treinamento e validação, incluindo as matrizes de confusão, curvas de aprendizado e exemplos de classificações corretas e incorretas, serão apresentadas e analisadas detalhadamente no Capítulo 3 e Capítulo 4.

Essas funções refletem o propósito central da FastAI: fornecer uma biblioteca acessível, porém robusta, voltada à produtividade, reutilização de modelos e desenvolvimento eficiente de aplicações em aprendizado profundo (HOWARD; GUGGER, 2020).

Código 2.1 – Código *Python* utilizado para criação e treino do modelo com a biblioteca FastAi

```
1 from fastai.vision.all import *
2
3 # Carregando as imagens e organizando em DataLoader
4 path = Path("./images")
5 files = get_image_files(path)
6
7 dls = ImageDataLoaders.from_name_func(
8     path, files,
```

```
    valid_pct=0.18,
    label_func=lambda x: 'praga' if 'praga' in x.name else 'saudavel',
    bs=9,
    item_tfms=Resize(460)
)

# Construção do modelo com a arquitetura ResNet34
learn = vision_learner(dls, resnet34, metrics=[error_rate, accuracy])

# Treinamento do modelo
learn.fine_tune(20)

# Resultados
learn.show_results(max_n=9)

# Predição de uma imagem
learn.predict(files[1])

# Exemplo condesando de uso da função
interp.plot_confusion_matrix(figsize=(7,7))
```

2.3 Aplicação *mobile*

Atualmente, os sistemas operacionais móveis mais utilizados são o Android, da Google, e o iOS, da Apple (HARAKAWA; PEREIRA, 2021). O desenvolvimento de aplicativos para essas plataformas envolve a consideração de diversos requisitos, os quais influenciam diretamente na escolha das tecnologias mais adequadas para garantir desempenho, compatibilidade e uma melhor experiência do usuário.

O aplicativo móvel desenvolvido neste trabalho depende da comunicação com uma *Application Programming Interface* (API), responsável por retornar os resultados da inferência, que são então exibidos de forma amigável na interface do dispositivo. Para seu desenvolvimento, optou-se pela linguagem de programação *JavaScript*, tendo em vista sua ampla adoção na construção de aplicações *frontend* e *backend*. Segundo a pesquisa anual do *Stack Overflow*, o *JavaScript* foi apontado como a linguagem mais utilizada entre desenvolvedores, com 63,61% das respostas em um total de 87.585 participantes (OVERFLOW, 2023).

No desenvolvimento de aplicativos móveis, há uma variedade de *frameworks* disponíveis. Para aplicações escritas em *JavaScript*, destaca-se o uso do *React Native*, que permite a criação de aplicativos nativos para as plataformas Android e iOS a partir de uma única base de código. Esse *framework* é amplamente adotado por grandes empresas, como o *Facebook*, e oferece alto desempenho e flexibilidade no desenvolvimento (EISENMAN,

2017).

O *React Native* é um *framework* JavaScript baseado no React, biblioteca desenvolvida pelo *Facebook* para construção de interfaces web (EISENMAN, 2017). Antes de sua adoção, era necessário dominar linguagens e ferramentas distintas — como *Java/-Kotlin* para Android e *Objective-C/Swift* para iOS — o que tornava o desenvolvimento mais complexo e oneroso (ESCUDELARIO; PINHO, 2020). O *React Native* oferece uma solução eficiente ao viabilizar o desenvolvimento multiplataforma e permitir o acesso a funcionalidades nativas dos dispositivos, como câmera, GPS e armazenamento, por meio de interfaces escritas em JavaScript (EISENMAN, 2017).

Para agilizar o desenvolvimento do aplicativo, utilizou-se o Expo, uma plataforma para React Native que elimina a necessidade de configurações nativas complexas. O Expo fornece um *Software Development Kit* (SDK) com APIs JavaScript para acesso a recursos nativos do dispositivo (câmera, localização, notificações) e permite testar o aplicativo diretamente em dispositivos físicos através do aplicativo Expo Go, acelerando o ciclo de desenvolvimento e simplificando a distribuição (Expo, 2025).

No contexto deste trabalho, o aplicativo desenvolvido com React Native e Expo não realiza nenhum tipo de processamento ou inferência local. Sua principal função é permitir que o usuário selecione ou capture uma imagem da lavoura, envie essa imagem para uma API hospedada remotamente e, em seguida, exiba o resultado da classificação retornado pelo modelo de inteligência artificial. A interface foi estruturada para tratar a resposta da API e apresentá-la de forma clara e responsiva por meio de um componente visual intuitivo.

3 DESENVOLVIMENTO

Este capítulo apresenta a metodologia de implementação do sistema de detecção de pragas em plantas de soja, abordando a arquitetura do projeto, os conjuntos de dados utilizados, os modelos de inteligência artificial desenvolvidos, o aplicativo *mobile* e a integração entre os componentes.

3.1 Metodologia

Este trabalho foi dividido em quatro etapas principais. A primeira etapa consistiu em encontrar bases de dados representativas de soja e realizar o pré-processamento dos dados. A segunda etapa envolveu o treinamento e a criação do modelo, além da execução de testes de desempenho. Na terceira etapa, foi desenvolvida uma *Application Programming Interface* (API) *Representational State Transfer* (Rest) e um aplicativo móvel. Por fim, na quarta etapa, foi realizada a integração do aplicativo com a API.

Na primeira etapa, realizou-se a seleção de bases de dados representativas, compostas por imagens de plantas de soja acometidas por diferentes pragas, além de exemplares saudáveis. Em seguida, foi aplicado o pré-processamento dessas imagens, incluindo rotulagem e padronização de tamanho, com o objetivo de preparar um conjunto de dados consistente e adequado ao treinamento do modelo.

A segunda etapa concentrou-se no treinamento do modelo de classificação, utilizando a biblioteca FastAI e a linguagem Python. Foram realizados experimentos para ajustar os parâmetros e validar o desempenho do modelo, buscando maximizar a assertividade na identificação das diferentes classes de pragas e plantas saudáveis.

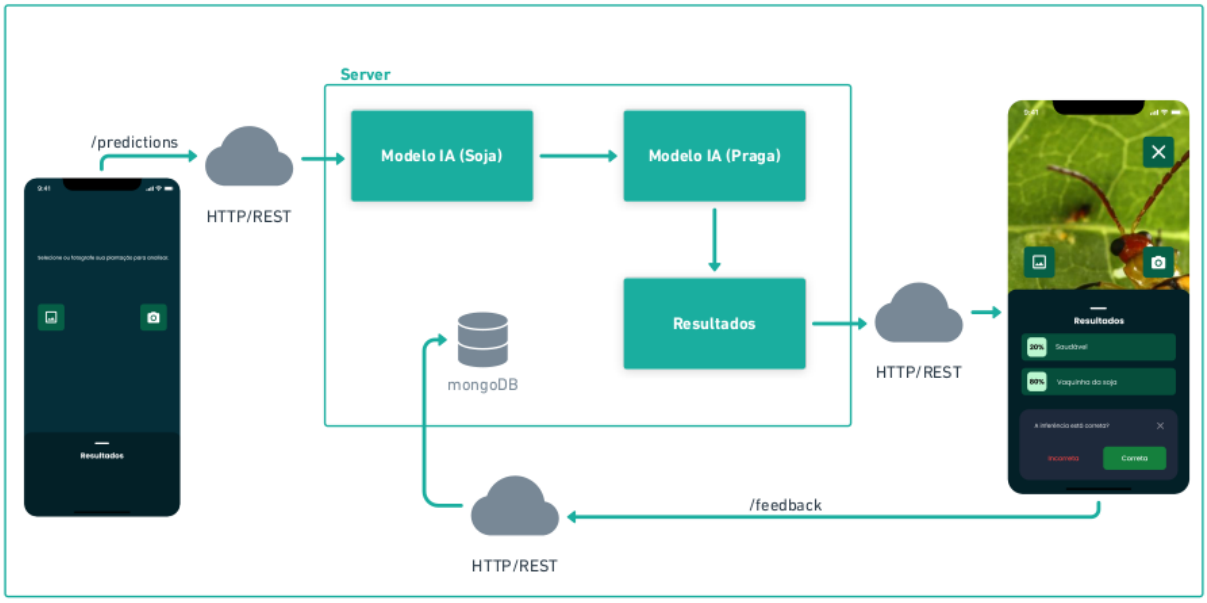
Na terceira etapa, foi desenvolvida uma API Rest, responsável por disponibilizar uma rota de comuniação capaz de receber imagens e retornar a identificação das pragas. Paralelamente, foi criado um aplicativo móvel, projetado para enviar imagens capturadas em campo diretamente para a API e exibir os resultados ao usuário de forma intuitiva e rápida.

Por fim, a quarta etapa consistiu na integração entre o aplicativo e a API, assegurando que todo o fluxo — desde o envio da imagem até o recebimento do diagnóstico — ocorresse de maneira eficiente, confiável e transparente para o usuário final.

3.2 Arquitetura do Projeto

A Figura 6 apresenta uma visão geral da arquitetura do projeto, representando o objetivo final do sistema desenvolvido. O projeto é composto por um aplicativo móvel, que representa a interface do usuário final, e um servidor que processa as imagens enviadas pelo aplicativo. Internamente, o servidor decodifica a imagem recebida, carrega os modelos de IA e retorna o resultado da análise ao aplicativo.

Figura 6 – Diagrama da arquitetura do projeto



Fonte: Elaborada pelo autor.

Quando o servidor recebe uma imagem do aplicativo, por meio de uma comunicação Rest no *endpoint* **/predictions**, a imagem trafega codificada em base64. No servidor, é carregado inicialmente um modelo pré-treinado para verificar se a imagem, após a decodificação, corresponde a uma planta de soja. Em caso afirmativo, a análise avança para o segundo modelo, treinado especificamente para detectar pragas. Por fim, o servidor retorna o resultado da inferência ao usuário final no aplicativo. Essa resposta consiste em um arquivo *JavaScript Object Notation* (JSON) contendo as porcentagens de probabilidade para cada tipo de praga que o modelo conhece, indicando como resultado final aquela com maior porcentagem.

O sistema também implementa um mecanismo de *feedback*, permitindo que o usuário envie correções caso considere a inferência incorreta. Nesse processo, o novo rótulo e a imagem em base64 são enviados ao servidor através do *endpoint* **/feedback**, onde são persistidos em um banco de dados MongoDB. Esses dados ficam armazenados para futuros retreinamentos do modelo, implementando assim um ciclo de melhoria contínua baseado nas interações dos usuários.

3.3 Definição do Dataset

Durante o desenvolvimento deste trabalho, foram utilizadas três bases de dados distintas. A primeira, intitulada Detecção de Praga - Soja e disponibilizada na plataforma Kaggle[1], é voltada para a classificação binária, contendo apenas 69 imagens de plantas de soja, divididas entre exemplares saudáveis e com pragas. Essa base foi empregada no treinamento inicial do modelo e também serviu como apoio na integração com as demais, contribuindo para o aumento do volume total de dados disponíveis.

A segunda base de dados é composta por doze classes de pragas, totalizando 1.800 imagens. No entanto, para a construção do modelo final, foram utilizadas apenas quatro dessas classes, devido à insuficiência de exemplos nas demais categorias.

Por fim, a terceira base de dados é formada por imagens de plantas de soja saudáveis, sendo utilizada para compor o dataset final e garantir maior representatividade de exemplos negativos no processo de treinamento.

A partir da integração dessas três bases, resultou um dataset final com 801 imagens, distribuídas entre cinco categorias: quatro espécies de pragas da soja e uma classe de plantas saudáveis. A seleção dessas espécies específicas baseou-se na sua relevância em campos de soja brasileiros e na disponibilidade de amostras suficientes para o treinamento adequado do modelo. A Tabela 1 apresenta a distribuição quantitativa das imagens por categoria

Tabela 1 – Número total de imagens por espécie e categoria

Espécies	Quantidade
Lagarta da soja (*Anticarsia gemmatalis*)	135
Percevejo da soja (*Nezara viridula*)	133
Vaquinha verde-amarela (*Diabrotica speciosa*)	146
Lagarta das vagens (*Spodoptera Albula*)	235
Soja saudável	152
Total	801

Fonte: Elaborada pelo autor.

3.3.1 INSECT12C

O dataset INSECT12C[2] representou a principal fonte de imagens de pragas para este trabalho. Composto por 1800 imagens em alta resolução (3024x4032 pixels), este conjunto de dados retrata pragas encontradas em plantações de soja brasileiras. As imagens foram capturadas em condições reais de campo, na fazenda experimental da Universidade Federal da Grande Dourados (UFGD), em Dourados, Mato Grosso do Sul. Um

[1] https://www.kaggle.com/datasets/neuronlab/deteco-de-pragas-soja
[2] https://github.com/EvertonTetila/INSECT12C-Dataset

aspecto importante deste dataset é que, durante o período de coleta, a área de 2 hectares não recebeu aplicação de agrotóxicos, permitindo o registro natural das pragas em seu habitat.

A quantidade de dados deste conjunto de dados não foi um fator principal para a escolha, mas a diversidade de condições de captura. As imagens foram registradas em diferentes momentos do dia que resultaram em variadas condições de iluminação, diferentes estágios de desenvolvimento das pragas e o fundo das imagens apresentam um ambiente agrícola real.

Embora o INSECT12C contenha doze espécies de pragas desfolhadoras, optou-se por utilizar apenas quatro destas espécies, complementadas por uma classe de plantas saudáveis. Esta seleção foi motivada pela necessidade de garantir um número suficiente de exemplos por categoria para favorecer o aprendizado do modelo. Testes preliminares realizados com todas as doze classes resultaram em baixa acurácia devido ao desbalanceamento e insuficiência de amostras em algumas categorias.

Durante a etapa de extração e rotulagem das imagens, foi identificada uma distribuição desigual entre as classes. Esse processo foi automatizado por meio de um *script* em *Python*, detalhado no Apêndice B, que realizou a leitura das anotações originais em formato *Extensible Markup Language* (XML). A análise revelou que diversas espécies possuíam quantidade insuficiente de amostras, inviabilizando o treinamento eficaz de classes específicas. Consequentemente, optou-se por focar o estudo nas quatro classes de pragas com maior representatividade, garantindo um balanceamento adequado do conjunto de dados final.

Os criadores do dataset utilizaram o conjunto completo das doze espécies para desenvolver um modelo de detecção em tempo real baseado na arquitetura YOLO, focando na identificação automática de pragas em campo (TETILA et al., 2024). Diferentemente da abordagem adotada neste trabalho, que priorizou um conjunto de classes reduzido, o estudo original explorou a capacidade de detecção abrangente do modelo em todas as espécies catalogadas.

3.3.2 SoyNet

Para complementar o conjunto de dados e garantir uma representação adequada de plantas saudáveis, foram adicionadas ao projeto 152 imagens de folhas de soja sem pragas vindas do dataset SoyNet[3], disponível publicamente no repositório Mendeley Data. Este conjunto de imagens contribuiu para equilibrar o dataset final, permitindo que o modelo aprendesse a distinguir entre plantas afetadas por pragas e exemplares saudáveis.

O SoyNet é um conjunto de dados abrangente que reúne mais de 9.000 imagens

[3] https://data.mendeley.com/datasets/w2r855hpx8/2

de alta qualidade de folhas de soja, tanto saudáveis quanto afetadas por doenças, capturadas em condições reais de cultivo. As imagens foram obtidas com câmeras digitais de alta resolução, sob diferentes condições de iluminação, e apresentam tamanhos variados. Embora o dataset original contenha uma grande quantidade de imagens, foram selecionados 152 exemplares de folhas saudáveis que apresentavam características visuais claras e representativas, complementando as classes de pragas obtidas do INSECT12C.

A inclusão destas imagens de plantas saudáveis contribuiu para o desenvolvimento de um modelo capaz de realizar não apenas a classificação entre diferentes tipos de pragas, mas também de distinguir com precisão entre plantas afetadas e não afetadas. Esta distinção é particularmente importante no contexto agrícola, onde a identificação precoce de plantas saudáveis versus plantas com pragas pode orientar decisões sobre intervenções no campo.

3.3.3 Pre-processamentos Realizados

O pré-processamento das imagens foi uma etapa fundamental para preparar os dados utilizados no treinamento do modelo de forma adequada. Inicialmente, foi realizada a rotulagem manual de cada imagem, classificando-as como pertencentes a uma das classes de pragas selecionadas ou à classe de soja saudável. Essa rotulagem foi importante para orientar o modelo a distinguir com maior precisão entre plantas sadias e aquelas afetadas por diferentes tipos de pragas.

No que diz respeito ao pré-processamento propriamente dito, todas as imagens foram redimensionadas para 450x450 pixels. Essa padronização de tamanho não só facilita o processamento pelo modelo, como também contribui para a uniformidade dos dados, reduzindo possíveis vieses causados por variações de resolução.

Além do redimensionamento, foram realizados experimentos com técnicas de extração de contornos utilizando vetores gradientes, com o objetivo de destacar regiões de interesse, como cortes nas folhas e a presença de insetos desfolhadores. Estes experimentos foram conduzidos inicialmente em um conjunto de dados menor, para fins de teste. No entanto, essa abordagem não apresentou ganhos significativos de acurácia que justificassem sua implementação no modelo final e aumentaria a complexidade da rota de predição. Uma possível abordagem alternativa seria utilizar as imagens com contornos destacados como forma de aumento de dados, expandindo assim o conjunto de treinamento. Por fim, optou-se por adotar apenas o redimensionamento das imagens e a rotulagem por classes como etapas principais de pré-processamento. No Apêndice A estão disponíveis algumas imagens com contorno e o código utilizado para aplicação do vetor gradiente.

3.3.4 *Data Augmentation*

O *data augmentation*, ou aumento de dados, é uma técnica utilizada para ampliar artificialmente a quantidade e a diversidade de exemplos em um conjunto de dados de imagens. Essa abordagem consiste em aplicar transformações variadas às imagens originais, como rotações, inversões, desfoque, alterações de brilho, entre outras. O objetivo é gerar novas versões das imagens, simulando diferentes condições de captura, iluminação e perspectiva, sem a necessidade de coletar mais dados reais. Essa técnica é especialmente útil para reduzir o risco de sobreajuste e melhorar a capacidade de generalização do modelo (GOODFELLOW; BENGIO; COURVILLE, 2016).

As técnicas de *data augmentation* foram aplicadas especificamente ao modelo de classificação binária, cujo conjunto de dados apresentava menor quantidade de exemplos. Foram utilizadas duas estratégias principais disponibilizadas nativamente pela biblioteca FastAI: a inversão horizontal das imagens e a aplicação de desfoque. A inversão horizontal gera uma versão espelhada da imagem original, permitindo que o modelo aprenda a reconhecer pragas independentemente da orientação da folha ou do inseto. Já o desfoque simula situações de baixa resolução ou de imagens desfocadas, tornando o modelo mais robusto a variações na qualidade das fotos capturadas em campo. Estas transformações são aplicadas aleatoriamente durante o treinamento através da classe `aug_transforms` da FastAI, que permite configurar a probabilidade e intensidade de cada transformação, aumentando efetivamente o tamanho do conjunto de treinamento sem necessidade de coleta adicional de imagens.

Para o modelo multiclasse, não foi necessário aplicar essas mesmas técnicas de aumento de dados, uma vez que os conjuntos INSECT12C e SoyNet já apresentavam, de forma nativa, uma ampla diversidade de imagens em diferentes condições. O *dataset* INSECT12C, por exemplo, inclui imagens com variações de ângulo (aéreo, frontal, lateral), casos de oclusão parcial, diferentes condições de iluminação e reflexão, além de variados níveis de nitidez, proporcionando um conjunto de treinamento naturalmente mais diversificado.

3.4 Modelos Treinados

A arquitetura ResNet34 foi adotada como base para os modelos de classificação de imagens deste trabalho. Trata-se de uma rede neural convolucional profunda, composta por 34 camadas, que é consolidada na literatura por sua notável eficiência em tarefas de visão computacional (HE et al., 2016). Para este projeto, utilizou-se uma versão desta arquitetura pré-treinada no vasto dataset ImageNet (DENG et al., 2009), que contém mais de 14 milhões de imagens distribuídas em milhares de classes. Essa abordagem, conhecida como aprendizado por transferência, permite que o modelo aproveite o conhecimento

adquirido em uma tarefa de classificação genérica.

No contexto deste trabalho, a ResNet34 foi utilizada através do *framework* FastAI, que oferece uma interface prática para aplicar técnicas de aprendizado profundo. Utilizou-se o método de ajuste fino (*fine-tuning*), que consiste em ajustar os pesos do modelo pré-treinado para o conjunto de dados específico do projeto. Esse processo permite que o modelo aprenda as características particulares das classes de interesse.

No treinamento, o *framework* simplificou a criação dos *data loaders*. Por padrão, a função de carregamento de dados divide automaticamente o conjunto de imagens em 80% para treino e 20% para validação. Definimos o parâmetro *batch size* como 9, o que apresentou o melhor desempenho após testes, incluindo os valores 12 e 16 para tamanho de *batch*. Essa escolha se justifica pelo tamanho reduzido do *dataset*, que contém apenas 801 imagens. Para *Batches* maiores, em conjuntos de dados pequenas, há o risco de comprometer a generalização do modelo, o que pode reduzir a acurácia. Com aproximadamente 641 imagens no conjunto de treino (80% de 801), um *batch size* de 12 geraria cerca de 54 atualizações de pesos por época, enquanto um batch size de 9 proporcionaria aproximadamente 72 atualizações, permitindo um aprendizado mais refinado das particularidades dos dados.

A relação entre o número de amostras no conjunto de treino e o tamanho do *batch* para determinar as iterações por época é formalizada pela seguinte expressão:

$$\text{Atualizações de Pesos por Época (Iterações)} = \frac{\text{Número de Amostras no Conjunto de Treino}}{\text{Tamanho do } \textit{Batch}} \tag{3.1}$$

Vale ressaltar que, como o número de iterações deve ser um número inteiro, o valor calculado será arredondado para o inteiro mais próximo.

Por fim, todos os experimentos foram realizados no Google Colab, um serviço que permite executar códigos *Python* por meio de um navegador e fornece acesso gratuito a recursos de GPU. A utilização dessa plataforma foi fundamental, pois o treinamento dos modelos com maior número de épocas em um *notebook* pessoal estava demandando tempo excessivo e não produziu resultados satisfatórios de treinamento devido às limitações de hardware.

3.4.1 Detecção Binária de Pragas

O primeiro modelo treinado foi desenvolvido com o objetivo de validar a integração entre os diferentes componentes do sistema e realizar os ajustes necessários no fluxo de processamento das imagens. Este modelo binário teve como finalidade identificar se a imagem fornecida apresentava ou não a presença de pragas. Durante os testes, observou-se que, ao submeter imagens que não correspondiam a plantas de soja, o modelo apresentava

resultados inconsistentes, o que motivou a incorporar uma etapa de verificação prévia para garantir que apenas imagens adequadas fossem analisadas nos estágios seguintes.

Para atender a essa demanda, foi utilizado um modelo pré-treinado com YOLO chamado *plant-leaf-detection-and-classification*[4], capaz de reconhecer folhas de soja na imagem. Dessa forma, apenas as imagens validadas como soja avançavam para o modelo específico de detecção de pragas, reduzindo inconsistências e aumentando a precisão do sistema.

Esta etapa do desenvolvimento concentrou-se na prototipação com modelos binários e teve um duplo objetivo: primeiramente, explorar as funcionalidades da biblioteca FastAI e avaliar sua viabilidade para os requisitos do projeto; em segundo lugar, validar o fluxo de integração entre o aplicativo e o servidor.

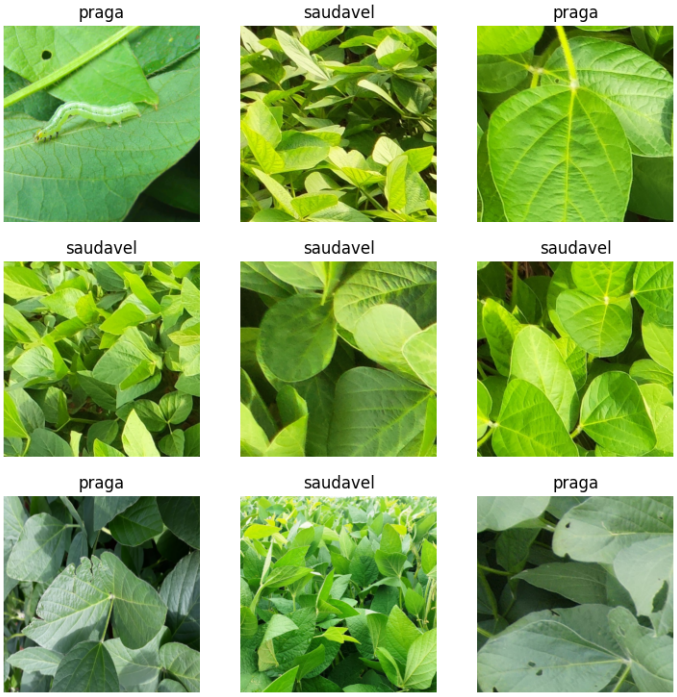
Para isso, foram realizados experimentos com um conjunto de dados inicial, que era reduzido e não balanceado. A criação de alguns modelos binários possibilitou um aprendizado prático sobre a ferramenta, mas também confirmou que a abordagem era promissora. Uma vez validada a tecnologia e o fluxo de dados, o desenvolvimento prosseguiu para o modelo multiclasse final, utilizando um dataset mais robusto e balanceado.

A Figura 7 apresenta a visualização do *DataLoader* do FastAI, exibindo nove imagens que correspondem ao tamanho de *batch* definido para o treinamento. O *batch* representa o lote de imagens processadas simultaneamente pelo modelo durante cada iteração. Cada imagem está rotulada como **praga** ou **saudável**, demonstrando a categorização dos dados. Esta visualização confirma o funcionamento correto do *DataLoader*, essencial para garantir que o modelo receba os dados adequadamente processados.

A Figura 8 apresenta exemplos de predições realizadas pelo modelo, onde são indicadas a classe real e a classe predita. Esta representação visual permite uma avaliação qualitativa do comportamento do modelo. Uma análise detalhada dos padrões de acerto e erro observados nestas predições será abordada no Capítulo 4 com os resultado obtidos no projeto.

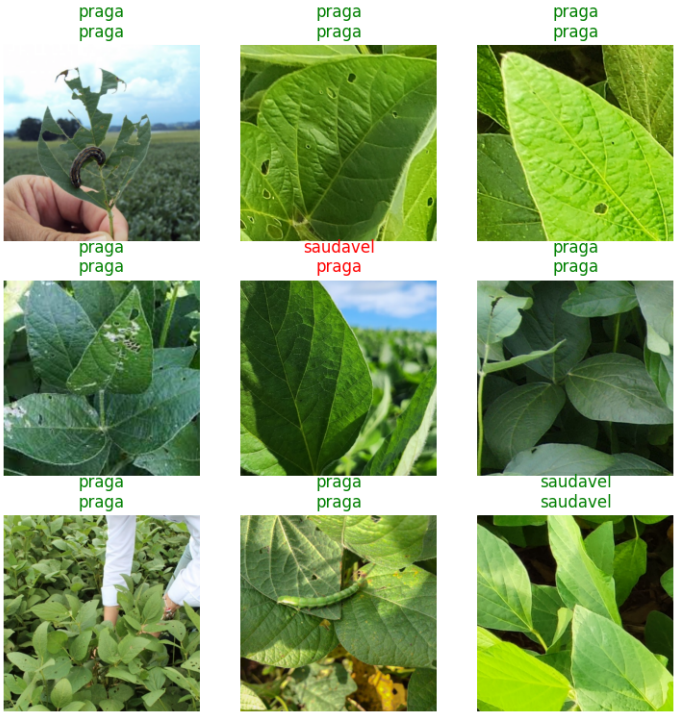
[4] https://huggingface.co/foduucom/plant-leaf-detection-and-classification

Figura 7 – Visualização do DataLoader do FastAI



Fonte: Elaborada pelo autor.

Figura 8 – Resultados do modelo binário



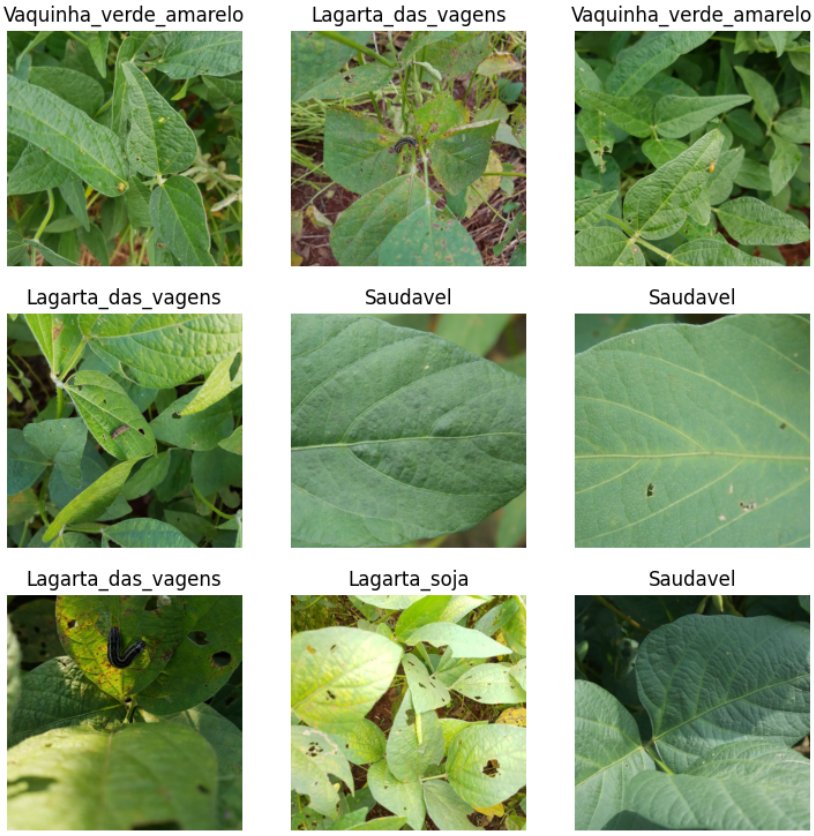
Fonte: Elaborada pelo autor.

3.4.2 Detecção Multiclasse de Pragas

A detecção multiclasse é uma abordagem no aprendizado de máquina na qual o modelo é treinado para identificar e classificar imagens em diversas categorias. No contexto deste trabalho, o desafio foi desenvolver um modelo capaz de reconhecer quatro espécies diferentes de pragas em plantações de soja, além de identificar plantas saudáveis. Embora o volume de dados utilizado para treinamento seja reduzido para padrões de aprendizado profundo, mostrou-se suficiente quando combinado ao uso de *transfer learning* com a arquitetura ResNet34 pré-treinada.

A Figura 9 apresenta a visualização do *DataLoader* do FastAI, exibindo nove imagens que correspondem ao tamanho de *batch* (bs=9) definido para o treinamento. Cada imagem está rotulada com as classes definidas, demonstrando como os dados são apresentados ao modelo durante o treinamento. Assim como no modelo binário, esta visualização confirma o funcionamento correto do *DataLoader*, que carrega as imagens rotuladas conforme esperado.

Figura 9 – Modelo multi classe data load



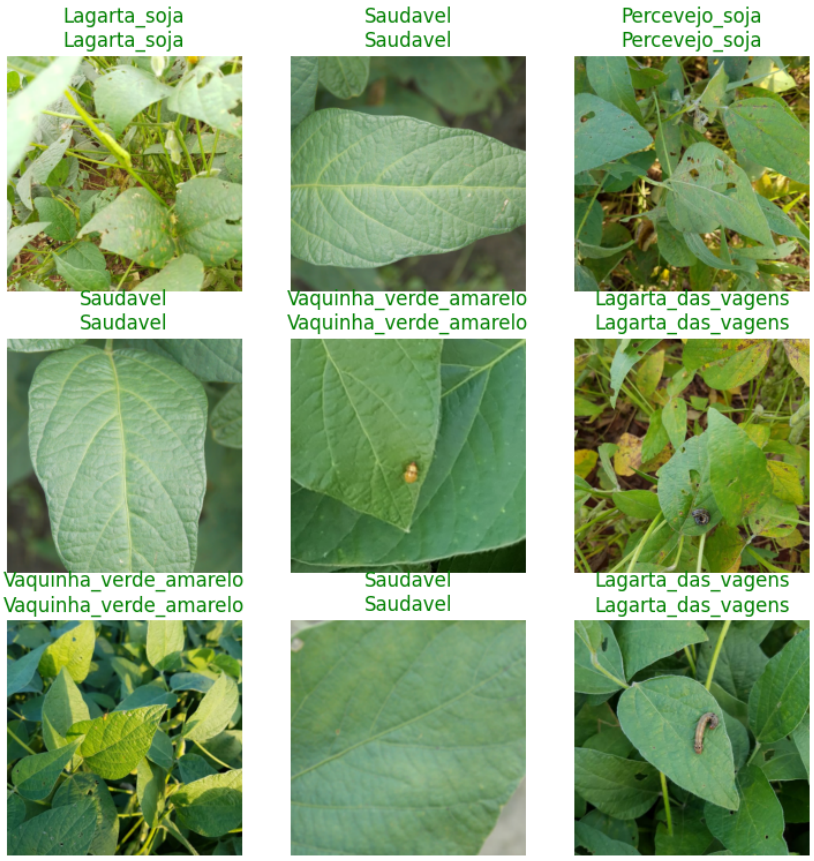
Fonte: Elaborada pelo autor.

Durante o desenvolvimento do modelo, foi adotada a técnica de *dropout* de nodos, com taxa de 0,25, ou seja, 25% das conexões entre neurônios foram desativadas aleatoriamente durante o treinamento. Essa estratégia contribui para evitar o *overfitting* - um

fenômeno onde o modelo se ajusta excessivamente aos dados de treinamento - e perde a capacidade de generalizar para novas imagens. O *overfitting* ocorre, geralmente, quando o modelo é muito complexo em relação à quantidade de dados disponíveis, memorizando padrões específicos em vez de aprender características gerais.

A Figura 10 exibe exemplos de predições realizadas pelo modelo multiclasse, onde é possível observar a capacidade do modelo em identificar corretamente as diferentes espécies de pragas e plantas saudáveis em variadas condições de iluminação e ângulos. As imagens demonstram que o modelo treinado consegue distinguir características sutis entre as espécies, como a diferença entre a Lagarta das vagens e a Lagarta da soja, ou entre a Vaquinha verde-amarelo e outras pragas. Esta visualização complementa a matriz de confusão, e possibilita uma avaliação qualitativa do desempenho do modelo em casos reais.

Figura 10 – Resultados do modelo multi classe



Fonte: Elaborada pelo autor.

O processo de treinamento do modelo foi configurado para 50 épocas, sendo monitorado por métricas essenciais, como a perda de treino (*train loss*), perda de validação (*valid loss*), taxa de erro (*error rate*) e acurácia. A análise das curvas de perda de treino e validação foi fundamental para avaliar a capacidade de generalização do modelo. Conforme destacado por (GOODFELLOW; BENGIO; COURVILLE, 2016), a observação

dessas métricas ao longo do treinamento é crucial para detectar falhas na aprendizagem e otimizar o desempenho do modelo. As curvas geradas durante esse processo podem ser visualizadas na Capítulo 4, onde também são discutidos os respectivos valores obtidos.

O acompanhamento da evolução dessas métricas ao longo das épocas permitiu avaliar o desempenho do modelo e identificar possíveis indícios de sobreajuste (*overfitting*), caracterizado por uma divergência significativa entre as curvas de treino e validação. Essa estratégia de monitoramento possibilitou a seleção do modelo final com base em um ponto de convergência considerado ideal, cujos resultados quantitativos e qualitativos são apresentados no capítulo seguinte.

3.5 Aplicativo Móvel SojaInspect

O aplicativo móvel, denominado **SojaInspect**, cujo código-fonte está disponível no GitHub[5], foi desenvolvido utilizando o *framework React Native* em conjunto com a plataforma *Expo*. A escolha dessas tecnologias foi motivada, principalmente, pela facilidade no desenvolvimento multiplataforma e pela agilidade proporcionada na criação de um protótipo funcional, aspectos essenciais para os objetivos deste trabalho.

As principais características e vantagens dessas tecnologias que fundamentaram sua escolha para este projeto são:

- **Desenvolvimento Multiplataforma:** O *React Native* permite criar aplicativos para iOS e Android a partir de uma única base de código, otimizando o tempo de desenvolvimento.
- **Ecossistema Robusto:** O ecossistema *JavaScript* oferece uma vasta biblioteca de componentes e APIs prontos para uso, especialmente para manipulação de imagens e comunicação com servidores Rest, elementos centrais deste projeto;
- **Abstração e Simplicidade:** O *Expo* simplifica o processo de desenvolvimento ao fornecer ferramentas integradas para compilação, testes e distribuição, eliminando a necessidade de configurações complexas de ambiente nativo;
- **Manutenção e Escalabilidade:** A arquitetura baseada em componentes do *React Native*, facilita a manutenção e escalabilidade do código, permitindo que novos recursos sejam implementados de forma modular sem comprometer a estabilidade do aplicativo existente.

O processo de criação do aplicativo começou com a elaboração de um protótipo da interface em uma ferramenta de design online[6], onde foram definidos o *layout*, as cores

[5] https://github.com/viniciusft81/tcc-detector-praga-app
[6] https://www.figma.com/design/pgXN3EkuG1my7AkS1eJgRT/SojaInspect-App?node-id=4008-64

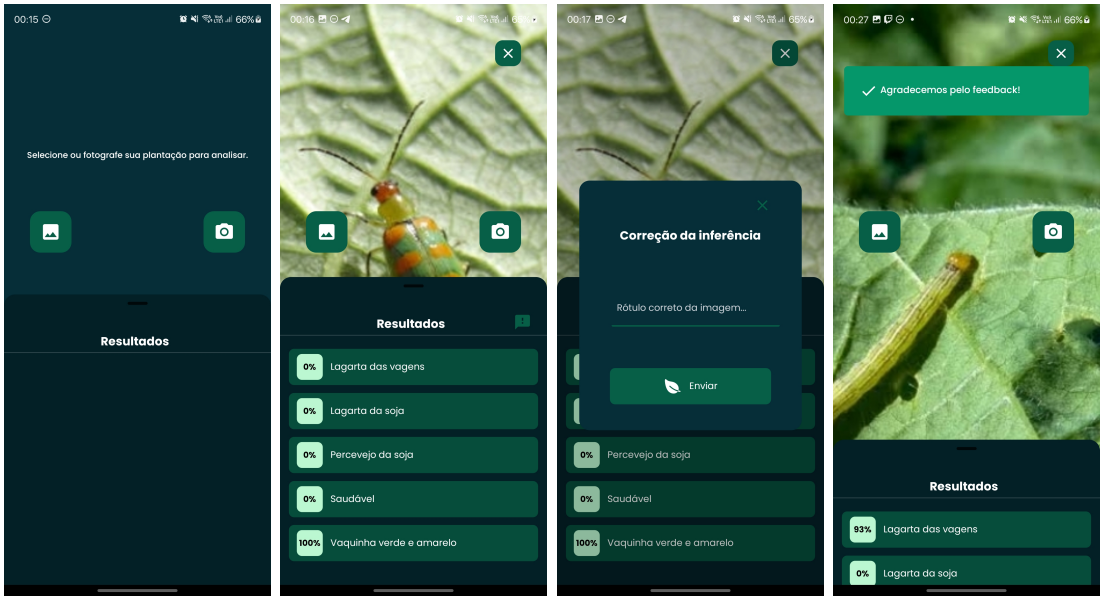
e a experiência do usuário. Essa etapa permitiu validar as ideias de design antes do início da programação.

O aplicativo oferece funcionalidades essenciais para o fluxo do projeto como: permitir que o usuário selecione imagens da galeria do dispositivo ou capturar novas fotos diretamente pela câmera. Após a seleção, a imagem é codificada em base64 e enviada para o servidor por meio de uma requisição Rest. O servidor, por sua vez, processa a imagem e retorna as probabilidades de cada classe de praga detectada, apresentando os resultados ao usuário de forma intuitiva.

As porcentagens exibidas para cada classe são provenientes da saída do modelo de detecção de pragas, que utiliza tensores para calcular a probabilidade de cada categoria. Dessa forma, o usuário pode visualizar não apenas a classe mais provável, mas também o grau de confiança do modelo na predição.

A Figura 11 apresenta diferentes telas do aplicativo, evidenciando a interface inicial, a visualização dos resultados e o mecanismo de envio de *feedback*, incluindo a notificação que confirma o envio da resposta pelo usuário.

Figura 11 – *Interface* do aplicativo: envio de imagem, exibição dos resultados e sistema de *feedback*.



(a) Tela inicial (b) Tela de resultados (c) Tela de *feedback* (d) Notificação de confirmação

Além de exibir os resultados com base nas probabilidades retornadas, o aplicativo permite que o usuário envie *feedback* sobre a precisão da predição, contribuindo para a melhoria contínua do sistema. Por decisão de projeto, a maior parte da lógica e do processamento foi mantida no servidor, tornando o aplicativo mais leve e eficiente. Em desenvolvimentos futuros, com otimizações do modelo, será possível considerar a inclusão de funcionalidades mais avançadas diretamente no aplicativo, como o carregamento do

modelo e a validação local no dispositivo, reduzindo a dependência de conexão com o servidor — o que aumentaria significativamente a autonomia da aplicação em ambientes de campo com acesso limitado à internet.

3.6 Integração Entre Servidor e Aplicativo

A integração entre o servidor e o aplicativo foi realizada com a implementação do *backend* em *Python*, utilizando o *framework Flask* para o carregamento do modelo de detecção binária e a disponibilização das rotas. Para garantir o correto funcionamento, foi necessário implementar e ajustar a comunicação entre as duas partes. Durante o desenvolvimento, configurou-se o *Cross-Origin Resource Sharing* (CORS), permitindo que o aplicativo realizasse requisições ao servidor mesmo quando originadas de ambientes diferentes.

No primeiro teste de integração, a API ainda não possuía um modelo para detectar se a imagem era de soja; assim, ao enviar imagens de qualquer objeto ou pessoa, eram retornados resultados inconsistentes.

Para contornar essa situação, tornou-se necessário incluir um modelo adicional para validar se a imagem enviada realmente correspondia a uma planta de soja. Essa etapa foi importante para evitar erros de classificação e garantir respostas coerentes do sistema. Portanto, quando uma imagem é enviada ao servidor, é realizado o carregamento do modelo mencionado na subseção 3.4.1, que assegura que a imagem é de soja antes de avançar para a classificação de pragas.

A integração também evidenciou pontos de melhoria no *backend*, como o tratamento adequado de erros e exceções, tornando o sistema mais estável e confiável. O formato do corpo das requisições enviadas pelo aplicativo foi ajustado conforme as validações implementadas no servidor, o que exigiu adaptações tanto no aplicativo quanto no servidor.

3.6.1 Publicação do Servidor

A primeira tentativa de publicação do servidor ocorreu na plataforma Render[7], um serviço de hospedagem para aplicações *backend*. No entanto, a tentativa não foi bem-sucedida devido à limitação de memória do plano gratuito, já que a instalação das bibliotecas *Python* e o carregamento do modelo de *machine learning* exigem recursos acima do que a plataforma oferece sem custos.

Como alternativa, foi realizado um teste de publicação utilizando um servidor de *reverse proxy Nginx* em uma instância *Amazon Elastic Compute Cloud* (EC2) da *Amazon*

[7] https://render.com/

Web Services (AWS). Essa abordagem funcionou tecnicamente, mas, devido à preocupação com possíveis cobranças futuras na nuvem, a instância foi desativada após os testes.

Devido às dificuldades com a publicação, não foi possível disponibilizar o aplicativo nas lojas oficiais (*Play Store* e *Apple Store*), pois o funcionamento do aplicativo depende do backend estar disponível de forma contínua.

Apesar dos desafios, após ajustes no formato das requisições e na validação das imagens, a integração foi concluída com sucesso. O aplicativo conseguiu se comunicar com o servidor, enviar imagens e receber os resultados das predições, validando assim a arquitetura proposta.

3.6.2 Api Rest

A API foi desenvolvida em *Python*, com o *framework Flask*, cujo código fonte está disponível no GitHub[8]. Funciona como intermediário entre o aplicativo e os modelos de inteligência artificial. Para atender às necessidades do sistema, a API disponibiliza duas rotas principais:

- **/predictions**: rota responsável por receber imagens e retornar análises de pragas;
- **/feedback**: rota que permite o envio de correções e armazena no banco de dados a correção da inferência fornecida pelo usuário para aprimoramento do modelo.

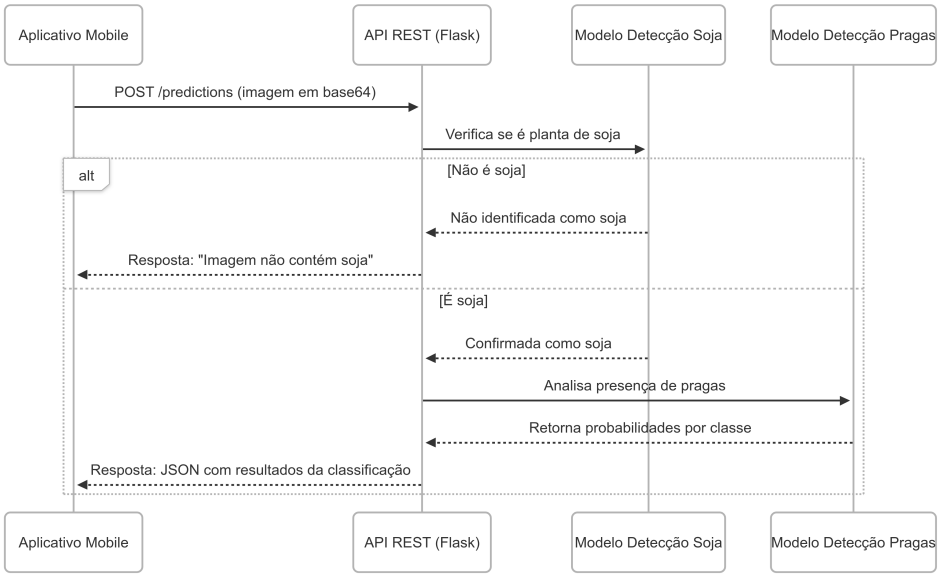
É importante ressaltar que esta implementação de prova de conceito não inclui autenticação nas rotas da API, visando facilitar os testes de validação dentro do escopo acadêmico do projeto. Em um ambiente de produção, seriam necessários mecanismos de segurança como autenticação *JSON Web Token* (JWT), limitação de requisições e criptografia para proteger os dados. Estas medidas estão previstas como sugestões para implementações futuras.

Na rota `/predictions`, o aplicativo envia uma imagem codificada em base64 para a API. Inicialmente, utiliza-se o modelo de detecção de folhas de plantas *(plant-leaf-detection-and-classification)* para verificar se a imagem corresponde a uma planta de soja. Confirmada essa condição, a imagem segue para um segundo modelo, treinado com a biblioteca FastAI, responsável por realizar a detecção das pragas. Ao final do processamento, é retornado ao usuário as probabilidades associadas a cada classe identificada.

A Figura 12 ilustra o fluxo completo de processamento da rota de predições, desde o envio da imagem pelo aplicativo até o retorno dos resultados, destacando as verificações sequenciais realizadas pelos modelos.

[8] https://github.com/viniciusft81/tcc-detector-praga-api

Figura 12 – Diagrama de sequência do fluxo de predição de pragas

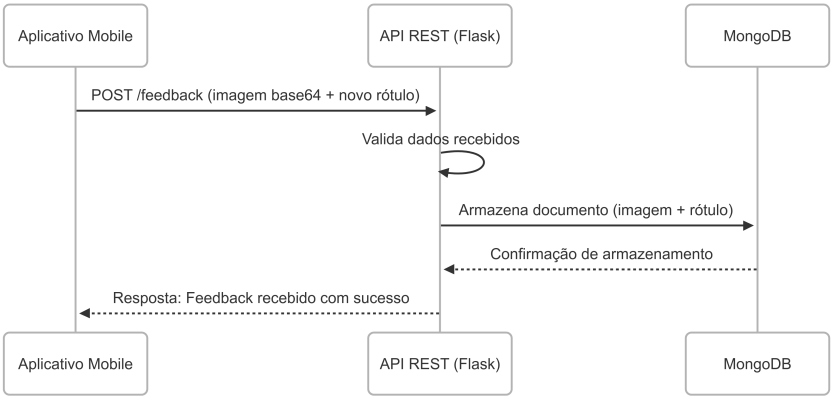


Fonte: Elaborada pelo autor.

A rota `/feedback` permite que especialistas revisem e corrijam as predições feitas pelo sistema. Cada *feedback* enviado é armazenado em um banco de dados **MongoDB**, contendo a imagem e o novo rótulo definido pelo usuário. Esses dados são fundamentais para o aprimoramento contínuo do sistema, pois possibilitam o retreinamento do modelo com exemplos corrigidos, aumentando a precisão das futuras predições.

De acordo com o diagrama de sequência da Figura 13, o processo de feedback estabelece um mecanismo de aprendizado contínuo, baseado nas interações dos usuários, onde as correções são sistematicamente incorporadas ao banco de dados para futuros aprimoramentos do modelo.

Figura 13 – Diagrama de sequência do processo de feedback e armazenamento de correções



Fonte: Elaborada pelo autor.

4 RESULTADOS OBTIDOS

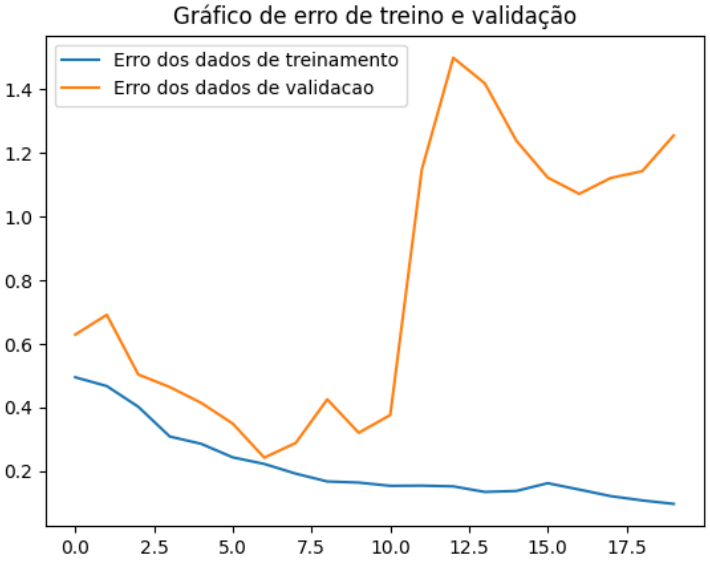
Este capítulo apresenta os resultados obtidos no desenvolvimento e a avaliação dos modelos de detecção de pragas em plantas de soja. São analisados dois modelos principais: o modelo binário, que permitiu validar a arquitetura e identificar problemas de *overfitting*, e o modelo multiclasse, que representa a evolução do trabalho com capacidade para classificar diferentes espécies de pragas. Para cada modelo, serão examinadas as métricas de desempenho, incluindo acurácia, perda de validação, matrizes de confusão e curvas de aprendizado, demonstrando a progressão técnica e as decisões que fundamentaram o desenvolvimento da solução final.

4.1 Validação da Arquitetura e Análise de *Overfitting* com Modelos binário

Antes do desenvolvimento do modelo multiclasse, foram implementados modelos de classificação binária para detectar a presença ou ausência de pragas.

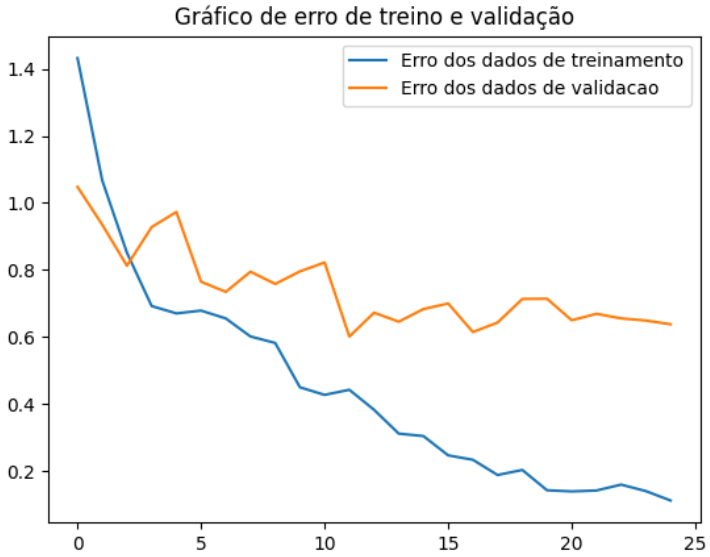
Dois treinamentos distintos foram realizados para gerar os modelos, o primeiro com 20 épocas e o segundo com 25 épocas. Ambos os treinamentos foram monitorados por métricas de desempenho como perda de treino, perda de validação e acurácia. A Figura 14 mostra o comportamento das curvas de erro durante as 20 épocas, enquanto a Figura 15 apresenta a curva de erro estendida até a última época.

Figura 14 – Curvas de erro de treino e validação para o modelo binário (20 épocas)



Fonte: Elaborada pelo autor.

Figura 15 – Curvas de erro de treino e validação para o modelo binário (25 épocas)



Fonte: Elaborada pelo autor.

A análise dos gráficos revela um padrão característico de *overfitting* nos dois modelos. Porém, no modelo com 20 épocas, a perda de validação começou a divergir significativamente da perda de treino a partir da época 6 e consequentemente é o modelo que teve maior *overfitting*.

A biblioteca FastAI, por padrão, salva apenas o modelo resultante da última época de treinamento, não implementando automaticamente o mecanismo de *early stopping*. Para utilizar essa técnica, é necessário empregar uma função de *callback* específica, recurso não utilizado nos modelos binários. Apesar de o melhor desempenho no conjunto de validação ter ocorrido na época 6, com perda de validação de 0,241 e acurácia de 87,5%, esse modelo não foi preservado, pois não havia configuração para salvá-lo automaticamente. Como consequência, apenas o modelo final, da última época, foi armazenado. A partir da época 6, observou-se aumento expressivo na perda de validação, atingindo 1,499 na época 12, caracterizando um caso de *overfitting* severo.

A Tabela 2 resume as métricas mais relevantes do modelo de 20 épocas. As tabelas completas, com todas as épocas dos dois experimentos, estão disponíveis no Apêndice C.

O segundo experimento com o modelo binário foi conduzido por 25 épocas e apresentou resultados mais estáveis em comparação ao primeiro. A perda de validação oscilou menos drasticamente e não indicou sinais claros de *overfitting* severo, mesmo nas épocas finais. A diferença crucial para essa melhoria, foi utilizar a técnica de aumento de dados, onde foi utilizado um número maior de imagens ajustada artificialmente.

O melhor modelo foi o da época 11, com uma perda de validação de 0,601 e acurácia de 80,72%, conforme ilustrado na Tabela 3. A partir da época 16, observou-se

Tabela 2 – Métricas do treinamento do modelo binário em épocas selecionadas

Época	valid_loss	Acurácia	Observação
0	0.628923	0.750000	Início do treinamento
6	**0.241891**	**0.875000**	**Melhor modelo**
8	0.425176	0.812500	Início do *overfitting*
12	1.499742	0.750000	*Overfitting* severo
19	1.255381	0.812500	Modelo salvo

Fonte: Elaborada pelo autor.

uma estabilização da acurácia em valores superiores a 83%, resultando na melhor marca registrada de 84,75% na última época, que foi o modelo salvo. Esses resultados indicam um comportamento de aprendizado mais consistente em relação ao modelo anterior, com ganhos progressivos de desempenho e menor risco de *overfitting*, justificando sua utilização na etapa de integração com o aplicativo móvel.

Tabela 3 – Métricas do treinamento do modelo binário com 25 épocas em pontos selecionados

Época	valid_loss	Acurácia	Observação
0	1.047810	0.6726	Início do treinamento
11	**0.601252**	**0.8072**	**Melhor modelo**
16	0.614762	0.8341	Estabilização da acurácia
24	**0.637959**	**0.8475**	**Maior acurácia registrada**

Fonte: Elaborada pelo autor.

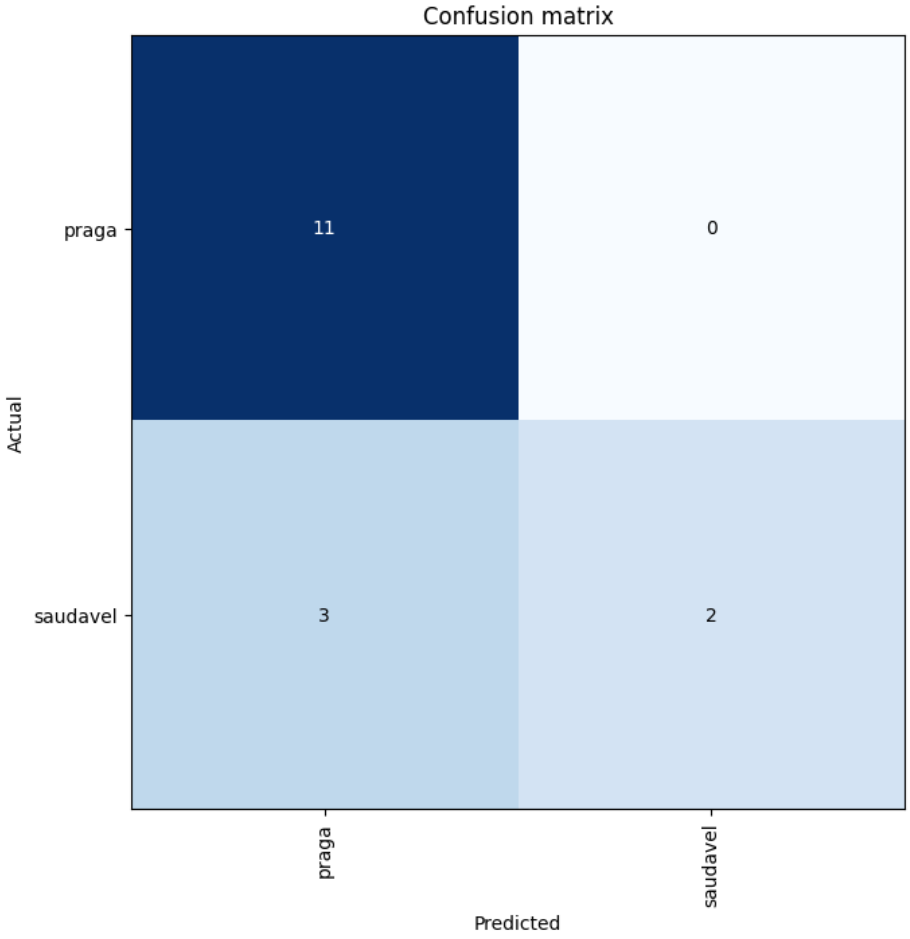
A identificação de *overfitting* nesses modelos motivou a adoção de técnicas de regularização no modelo multiclasse. Entre elas, destacou-se o uso de uma camada de *Dropout*, que desativa aleatoriamente parte dos neurônios durante o treinamento para reduzir a memorização dos dados de treino, aliada à utilização de uma base de dados balanceada.

A matriz de confusão da Figura 16 apresenta o desempenho quantitativo do modelo binário de 25 épocas na classificação entre plantas com pragas e plantas saudáveis, indicando:

- 11 verdadeiros positivos: plantas com pragas corretamente identificadas;
- Nenhum falso negativo: nenhuma planta com praga foi classificada erroneamente como saudável;
- 3 falsos positivos: plantas saudáveis foram incorretamente classificadas como com pragas;

- 2 verdadeiros negativos: plantas saudáveis corretamente identificadas.

Figura 16 – Matriz de confusão do modelo de detecção de pragas binário



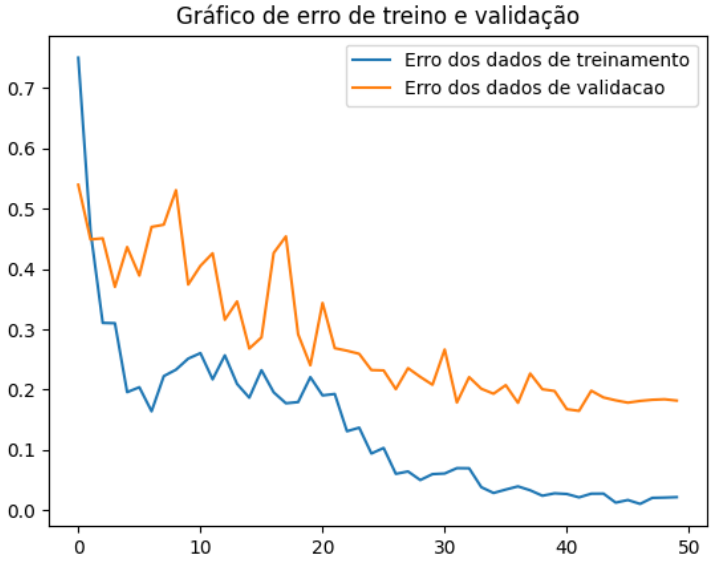
Fonte: Elaborada pelo autor.

4.2 Treinamento e Desempenho do Modelo Multiclasse

Com base nos aprendizados adquiridos com o modelos anteriores, o modelo multiclasse foi desenvolvido com a adição de técnicas de regularização incorporando o *Dropout* de *nodes* à uma taxa de 25% com o intuito de mitigar o *overfitting*. O treinamento foi realizado ao longo de 50 épocas, com monitoramento contínuo das métricas de perda (*valid_loss*) e acurácia.

A evolução do treinamento é ilustrado na Figura 17, que apresenta a convergência das curvas de perda de treinamento e validação. Observa-se um aprendizado rápido nas épocas iniciais, com a estabilização da perda de validação a partir da época 35, indicando que o modelo se aproximava de seu ponto ótimo de generalização.

Figura 17 – Curvas de erro de treino e validação para o modelo multiclasse



Fonte: Elaborada pelo autor.

A Tabela 4 apresenta os principais marcos do processo de treinamento. O melhor desempenho no conjunto de validação foi alcançado na época 41, registrando a menor perda de validação 0,1647 e uma acurácia de 96,87%.

Embora a menor perda de validação tenha ocorrido na época 41 (0,1647), o modelo não foi preservado, pois não havia sido configurado um callback de salvamento do melhor modelo. Assim, a etapa final de avaliação/integração foi conduzida com o modelo da última época, que apresentou acurácia elevada de 97,5%, ainda que com perda de validação ligeiramente superior (0,1818).

Em trabalhos futuros, recomenda-se configurar a função de *callback* `SaveModelCallback` para garantir o salvamento do melhor modelo.

A Figura 18 apresenta a matriz de confusão do modelo, que demonstra excelente desempenho na classificação das cinco categorias (quatro espécies de pragas e plantas

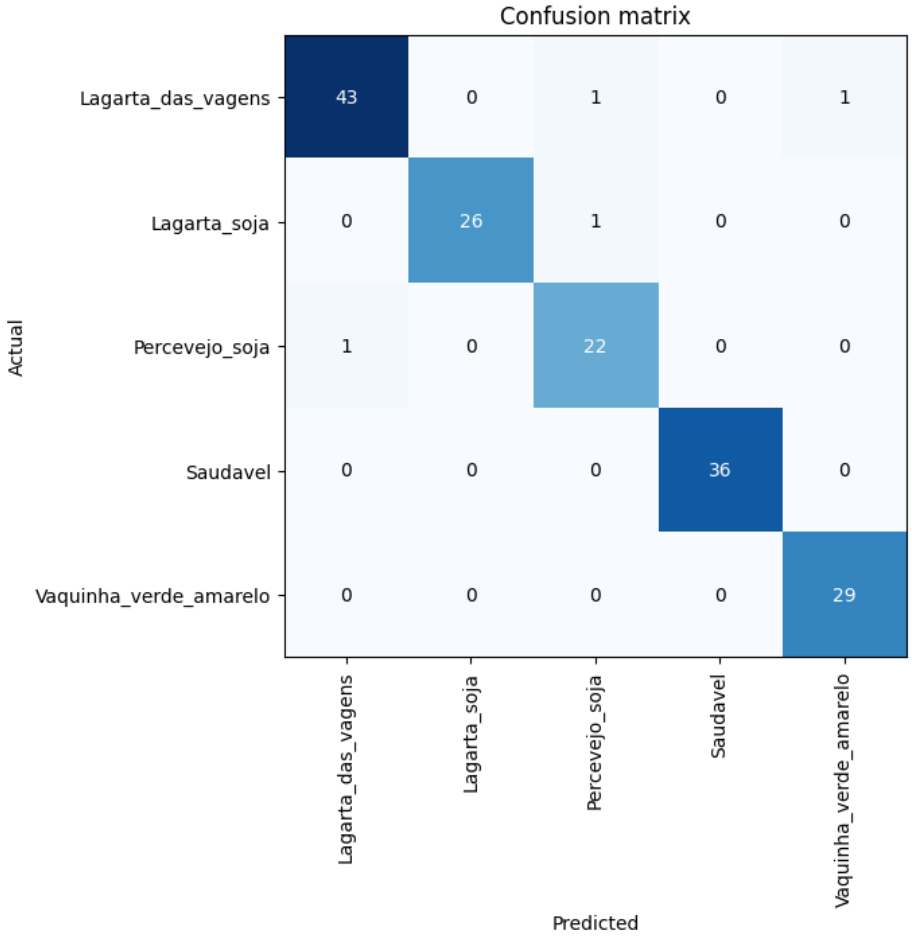
Tabela 4 – Métricas do treinamento multiclasse em épocas selecionadas

Época	Perda (Val.)	Taxa de Erro	Acurácia	Observação
0	0.539794	0.212500	0.787500	Início
14	0.268095	0.087500	0.912500	Progresso
41	**0.164701**	**0.031250**	**0.968750**	**Melhor Modelo**
49	0.181813	0.025000	0.975000	Final

Fonte: Elaborada pelo autor.

saudáveis). A diagonal principal evidencia um alto número de classificações corretas, com destaque para a Lagarta das vagens (43 acertos) e plantas saudáveis (36 acertos). Os poucos erros de classificação (valores fora da diagonal) são mínimos, indicando que o modelo alcançou alta precisão na diferenciação entre as classes, mesmo com características visuais por vezes semelhantes entre algumas pragas.

Figura 18 – Matriz de confusão do modelo de detecção de pragas multiclasse



Fonte: Elaborada pelo autor.

A tabela completa com as métricas detalhadas de todas as 50 épocas de treinamento está disponível para consulta no Apêndice C.

4.2.1 *Logs* da aplicação

Durante os testes do sistema, os registros de *log* apresentados na Figura 19 foram fundamentais para acompanhar o funcionamento da aplicação. Ao fazer o envio de imagens pelo aplicativo para a API, o *backend* realiza o processamento e retorna o resultado da predição.

O primeiro registro no *log* após o envio da imagem confirma que ela foi corretamente decodificada pela biblioteca *Pillow* (`PIL`), evidenciando que o arquivo foi recebido com sucesso e está em formato compatível com o modelo. Em seguida, o log registra o carregamento do modelo de detecção de soja, utilizado para identificar a presença de plantas de soja na imagem.

Após a etapa de detecção, a imagem segmentada é encaminhada ao modelo de classificação de pragas. O *log* apresenta a predição realizada, indicando a classe identificada (por exemplo, **Vaquinha verde amarelo** ou **Saudável**) e os tensores com os valores de confiança associados a cada classe. Também são exibidas informações técnicas do modelo *YOLOv8*, incluindo número de camadas, parâmetros e métricas de desempenho de inferência (tempo de pré-processamento, inferência e pós-processamento por imagem).

Esses registros permitiram validar o correto funcionamento de cada etapa do sistema, facilitando o diagnóstico de eventuais erros e o monitoramento da performance em diferentes cenários de entrada.

Figura 19 – Exemplo de *Logs* do servidor

```
========================================
category tensor([27.])
========================================
07/18/2025 21:47:40 - INFO - root -   Predição realizada com sucesso: Vaquinha_verde_amarelo, tensor
(4), tensor([1.21725e-04, 6.47843e-04, 3.53418e-06, 8.76713e-07, 9.99226e-01])
Imagem decodificada com sucesso: <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x500>
Ultralytics YOLOv8.0.43 🚀 Python-3.10.12 torch-2.2.2+cu121 CPU
Model summary (fused): 168 layers, 11143386 parameters, 0 gradients, 28.5 GFLOPs

0: 640x640 2 grams, 582.5ms
Speed: 2.6ms preprocess, 582.5ms inference, 1.9ms postprocess per image at shape (1, 3, 640, 640)
========================================
category tensor([38., 38.])
========================================
07/18/2025 21:48:01 - INFO - root -   Predição realizada com sucesso: Saudavel, tensor(3), tensor([0
.02019, 0.25392, 0.03253, 0.64572, 0.04764])
Imagem decodificada com sucesso: <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x500>
Ultralytics YOLOv8.0.43 🚀 Python-3.10.12 torch-2.2.2+cu121 CPU
Model summary (fused): 168 layers, 11143386 parameters, 0 gradients, 28.5 GFLOPs

0: 640x640 1 tomato, 1210.7ms
Speed: 5.2ms preprocess, 1210.7ms inference, 1.8ms postprocess per image at shape (1, 3, 640, 640)
========================================
category tensor([36.])
========================================
07/18/2025 21:48:23 - INFO - root -   Predição realizada com sucesso: Vaquinha_verde_amarelo, tensor
(4), tensor([1.26300e-03, 2.58070e-03, 9.97980e-03, 8.89461e-04, 9.85287e-01])
```

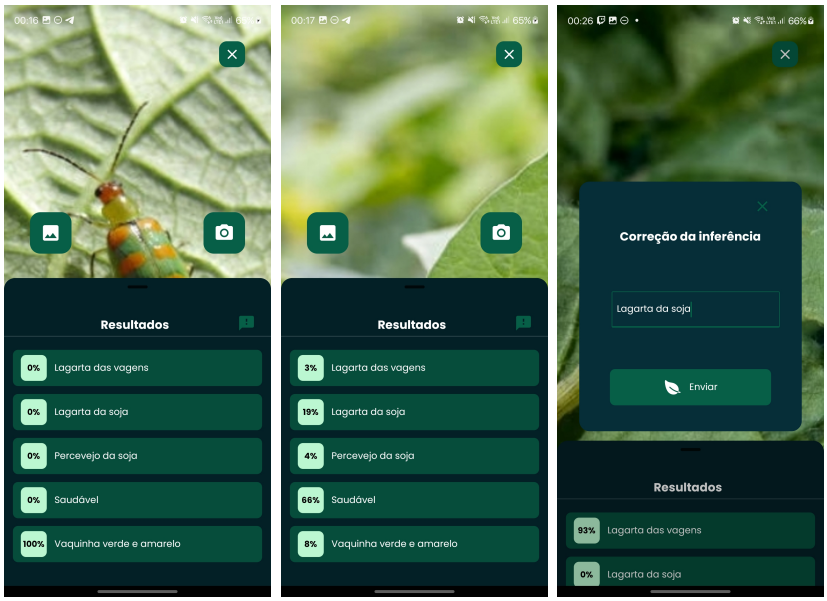
Fonte: Elaborada pelo autor.

4.2.2 Resultados da Inferência no Aplicativo

A Figura 20 apresenta os resultados obtidos no aplicativo a partir da inferência de imagens enviadas. A imagem (a) exibe a classificação correta de uma planta com praga, retornando como resultado a categoria **Vaquinha verde e amarelo** com 100% de certeza. Já a imagem (b) apresenta a identificação de uma folha **Saudável**, demonstrando que o sistema também é capaz de reconhecer corretamente casos em que não há pragas.

Contudo, como ilustrado na imagem (c), foi identificado um exemplo em que a inferência retornou incorretamente a classe **Lagarta das vagens**, quando na verdade se tratava da **Lagarta da soja**. Esse erro foi corrigido manualmente por meio do envio de um *feedback*, funcionalidade implementada na aplicação e que permite refinar o processo de validação e aprimoramento contínuo do modelo.

Figura 20 – Resultados de inferência para diferentes imagens



(a) Resultado da inferência no aplicativo para uma planta com praga

(b) Resultado da inferência no aplicativo para uma planta saudável

(c) Envio de uma correção para o *endpoint* de *feedback*

Fonte: Elaborada pelo autor.

A correção foi enviado via requisição Rest para o *endpoint* de *feedback* e armazenado no banco de dados MongoDB, conforme apresentado na Código 4.1. Os registros são armazenados com a imagem em formato base64, o rótulo enviado pelo servidor e o novo rótulo com a correção.

Código 4.1 – Documento MongoDb com a correção da inferência salva no banco de dados

```
1 {
2   "_id": {
3     "$oid": "687b1304c87bdfeade8f4c5d"
```

```
  },
  "data": {
    "image": {
      "base64": "/9j/4AAQSkZfP3xY2z5ULy1tvMDxSqIk7R...7p0FLUFIaHeL/1
    H649j2PYxzqfExjDQe/6HhH//2Q=="
    },
    "result": "Lagarta_das_vagens",
    "result_correction": "Lagarta da Soja"
  }
}
```

Atualmente, o sistema armazena as correções sem validação adicional do rótulo enviado, funcionando como um repositório de dados para futuras melhorias. Não há implementação para retreinamento do modelo com base nessas correções. Uma evolução natural do sistema seria a implementação de um painel administrativo que exibisse métricas sobre as correções recebidas e permitisse ao administrador iniciar o processo de retreinamento quando um número significativo de correções estivesse disponível, garantindo assim o aprimoramento contínuo do modelo.

Observou-se ainda, uma possível tendência do modelo em atribuir a categoria **Lagarta das vagens** em casos de incerteza. Esse comportamento pode indicar uma distribuição desbalanceada no conjunto de treinamento ou baixa diferenciação entre classes visualmente semelhantes, o que representa uma oportunidade clara de melhoria futura na curadoria e diversificação do dataset.

4.3 Uso de Plataformas AutoML para Prototipagem

Durante as etapas iniciais do projeto, explorou-se o uso de plataformas *Automated Machine Learning* (AutoML) como Landing AI[1] e Roboflow[2], com o objetivo de avaliar a viabilidade de construir modelos funcionais a partir de conjuntos de dados reduzidos. Essas ferramentas permitiram experimentar rapidamente fluxos completos de visão computacional com poucas imagens, evidenciando sua utilidade como ponto de partida em projetos com restrições de tempo ou conhecimento técnico.

Apesar da praticidade, optou-se por desenvolver o modelo principal com a biblioteca FastAI, uma ferramenta de código aberto que proporciona controle total sobre o processo de treinamento, ajustes finos e integração. Além da flexibilidade técnica, a FastAI não impõe custos diretos de uso, ao contrário das plataformas comerciais, que operam com modelos de crédito ou planos pagos. Assim, o uso inicial das plataformas serviu como

[1] https://landing.ai/
[2] https://roboflow.com/

apoio exploratório, mas a escolha final recaiu sobre uma abordagem programável, personalizável e economicamente viável.

5 CONCLUSÕES

O desenvolvimento do sistema de detecção de pragas em plantas de soja, baseado em técnicas de aprendizado de máquina, demonstrou a viabilidade e relevância da aplicação de inteligência artificial no agronegócio brasileiro. O trabalho abrangeu desde a seleção criteriosa de bases de dados representativas, passando pelo pré-processamento e aumento de dados, até o treinamento de modelos robustos com a arquitetura ResNet34 e a implementação de uma solução móvel funcional, voltada ao uso prático em campo.

Os resultados obtidos evidenciaram alta precisão na identificação de diferentes espécies de pragas e na distinção entre plantas saudáveis e afetadas, mesmo diante de um volume de dados relativamente reduzido. O uso de técnicas como transferência de aprendizado, *fine-tuning*, *Dropout* e *data augmentation* foi fundamental para maximizar o desempenho dos modelos, tornando-os aptos a lidar com variações reais de campo.

A arquitetura proposta, baseada em uma API REST e um aplicativo desenvolvido em *React Native*, mostrou-se eficiente tanto no processamento quanto na experiência do usuário. O sistema permite que agricultores realizem diagnósticos rápidos e recebam *feedbacks* claros, além de contribuir para o aprimoramento contínuo do modelo por meio do ciclo de *feedback*.

Apesar dos desafios enfrentados na publicação do *backend* e na publicação do aplicativo, a integração entre os componentes foi validada com sucesso, confirmando o potencial da solução para uso prático. Como perspectivas futuras, destaca-se a possibilidade de ampliar o *dataset*, incorporar novas funcionalidades no aplicativo e otimizar o processamento local, com o objetivo de reduzir a dependência de conexão com o servidor.

Em síntese, este trabalho apresenta uma prova de conceito que explora o uso de inteligência artificial na agricultura digital, evidenciando o potencial de soluções baseadas em visão computacional para a detecção automatizada de pragas em lavouras de soja. Embora ainda em estágio experimental, os resultados obtidos reforçam a viabilidade técnica da abordagem proposta e abrem caminho para desenvolvimentos voltados à aplicação prática no campo.

5.1 Trabalhos futuros

Embora o sistema proposto tenha alcançado resultados satisfatórios, diversas oportunidades de aprimoramento e expansão podem ser exploradas em trabalhos futuros.

Uma das direções mais imediatas é a ampliação do conjunto de dados, incorpo-

rando novas imagens, capturadas com distintas condições ambientais e estágios de desenvolvimento das plantas. Além disso, sugere-se a inclusão de novas classes de pragas, doenças fúngicas e deficiências nutricionais, ampliando a abrangência do modelo e sua utilidade prática no campo.

Do ponto de vista da visão computacional, é recomendável investigar técnicas de compressão e otimização de modelos — como quantização, distilação ou uso de arquiteturas leves — visando reduzir o consumo de recursos e permitir inferência diretamente no dispositivo móvel, mesmo em cenários com conectividade limitada.

O *backend*, atualmente funcional em ambiente de desenvolvimento, pode ser aprimorado para facilitar a publicação em infraestrutura escalável e eficiente, minimizando o uso de memória e melhorando a resposta a múltiplas requisições simultâneas. Isso permitiria, viabilizar a publicação do aplicativo em lojas oficiais, com distribuição mais ampla.

No âmbito da inteligência artificial generativa, vislumbra-se a adoção de *Large Language Models* (LLMs) para criação de assistentes inteligentes integrados ao aplicativo, capazes de responder dúvidas do usuário sobre diagnóstico, tratamento e manejo integrado de pragas. Outra possibilidade seria a integração com plataformas como a API do Google (*Vertex AI Agents*), explorando fluxos conversacionais personalizados ou agentes autônomos voltados à recomendações no contexto agrícola.

Por fim, propõe-se a construção de uma arquitetura modular e reusável para projetos semelhantes em outras culturas de plantas, incentivando a disseminação de tecnologias acessíveis e de código aberto no setor do agronegócio.

REFERÊNCIAS

ACADEMY, D. S. *Capítulo 10 - As 10 Principais Arquiteturas de Redes Neurais*. [s.n.], 2022. Disponível em: https://www.deeplearningbook.com.br/as-10-principais-arquiteturas-de-redes-neurais/. 17, 18

ACADEMY, D. S. *Capítulo 3 - O Que São Redes Neurais Artificiais Profundas ou Deep Learning?* [s.n.], 2022. Disponível em: https://www.deeplearningbook.com.br/o-que-sao-redes-neurais-artificiais-profundas/. 17

ACADEMY, D. S. *O Que é Visão Computacional?* 2022. Disponível em: https://blog.dsacademy.com.br/o-que-e-visao_computacional/. 21

AIRES, R. *Tecnologia no agronegócio: importância e principais tendências*. 2023. Disponível em: https://www.agriq.com.br/tecnologia-no-agronegocio/. 14

DALL'AGNOL, A. et al. *Importância socioeconômica da soja - Portal Embrapa*. 2021. Disponível em: https://www.embrapa.br/agencia-de-informacao-tecnologica/cultivos/soja/pre-producao/socioeconomia/importancia-socioeconomica-da-soja. 14

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 248–255. 30

EISENMAN, B. *Learning React Native: Building Native Mobile Apps with JavaScript*. [S.l.]: O'Reilly Media, 2017. ISBN 978-1-4919-8909-8. 24

EMBRAPA. *VII Plano Diretor da Embrapa 2020-2030*. 2020. Disponível em: https://www.embrapa.br/vii-plano-diretor. 14

ESCUDELARIO, B.; PINHO, D. *React Native: Desenvolvimento de aplicativos mobile com React*. [S.l.]: Casa do Código, 2020. ISBN 9786586110067. 24

Expo. *Expo Documentation*. 2025. Acessado em: 10 jun. 2025. Disponível em: https://docs.expo.dev/. 24

FERREIRA, B. S. C.; CAMPO, C. B. H.; GóMEZ, D. R. S. Inimigos naturais de Helicoverpa armigera em soja. 2014. Disponível em: https://ainfo.cnptia.embrapa.br/digital/bitstream/item/107296/1/Inimigos-naturais-de-Helicoverpa-armigera-em-soja.pdf. 15

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org. 30, 35

HARAKAWA, F. S.; PEREIRA, A. S. Desenvolvimento e Implementação do Aplicativo Mobile Blog de Todas. 2021. Disponível em: https://repositorio.ufsm.br/bitstream/handle/1/25618/Harakawa_Felipe_Seidi_2021_TCC.pdf?sequence=1&isAllowed=y. 23

HE, K. et al. Deep residual learning for image recognition. 2016. 19, 30

HONAVAR, V. Artificial Intelligence: An Overview. 2016. Disponível em: https://faculty.ist.psu.edu/vhonavar/Courses/ai/handout1.pdf. 16

HOWARD, J.; GUGGER, S. *Deep Learning for Coders with fastai and PyTorch*. [S.l.]: O'Reilly Media, 2020. ISBN 978-1-4920-4549-6. 19, 22

LEIJNEN STEFAN, F. v. V. *The Neural Network Zoo*. [s.n.], 2019. Disponível em: https://www.asimovinstitute.org/neural-network-zoo/. 17

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à Visão Computacional usando OpenCV. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125–160, 2009. ISSN 2175-2745. Number: 1. Disponível em: https://seer.ufrgs.br/index.php/rita/article/view/rita_v16_n1_p125. 21

MILLINGTON, I. *Artificial Intelligence for Games*. [S.l.]: Taylor & Francis, 2006. (Morgan Kaufmann series in interactive 3D technology). ISBN 978-0-12-497782-2. 16

MORETI, M. P. et al. Inteligência Artificial no Agronegócio e os Desafios para a Proteção da Propriedade Intelectual. *Cadernos de Prospecção*, v. 14, n. 1, p. 60–60, jan. 2021. ISSN 2317-0026. Number: 1. Disponível em: https://periodicos.ufba.br/index.php/nit/article/view/33098. 16

MURPHY, J. M. J. An Overview of Convolutional Neural Network Architectures for Deep Learning. 2016. Disponível em: https://www.microway.com/download/whitepaper/An_Overview_of_Convolutional_Neural_Network_Architectures_for_Deep_Learning_fall2016.pdf. 18

NGUYEN, C. T. et al. Transfer Learning for Wireless Networks: A Comprehensive Survey. *Proceedings of the IEEE*, v. 110, n. 8, p. 1073–1115, ago. 2022. ISSN 0018-9219, 1558-2256. Disponível em: https://ieeexplore.ieee.org/document/9789336/. 20

NORVIG, P.; RUSSELL, S. *Inteligência Artificial*. 3. ed. ELSEVIER EDITORA, 2013. ISBN 978-85-352-3701-6. Disponível em: https://www.cin.ufpe.br/~gtsa/Periodo/PDF/4P/SI.pdf. 16

OVERFLOW, S. *Most popular technologies*. 2023. Disponível em: https://survey.stackoverflow.co/2023/#most-popular-technologies-language. 23

PAN, S. J.; YANG, Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, v. 22, n. 10, p. 1345–1359, out. 2010. ISSN 1041-4347. Disponível em: http://ieeexplore.ieee.org/document/5288526/. 20

PANG, Y. et al. Convolution in Convolution for Network in Network. *IEEE Transactions on Neural Networks and Learning Systems*, v. 29, n. 5, p. 1587–1597, maio 2018. ISSN 2162-237X, 2162-2388. Disponível em: https://ieeexplore.ieee.org/document/7879808/. 18

PATTERSON, J.; GIBSON, A. *Deep Learning: A Practitioner's Approach*. [S.l.]: O'Reilly, 2017. ISBN 978-1-4919-1425-0. 16

PINHEIRO, R. de M. et al. Inteligência artificial na agricultura com aplicabilidade no setor sementeiro | Diversitas Journal. ago. 2021. Disponível em: https://diversitasjournal.com.br/diversitas_journal/article/view/1857. 16

SHINDE, P. P.; SHAH, S. A Review of Machine Learning and Deep Learning Applications. IEEE, Pune, India, p. 1–6, ago. 2018. Disponível em: https://ieeexplore.ieee.org/document/8697857/. 16, 17

SZELISKI, R. *Computer Vision: Algorithms and Applications*. [S.l.]: Springer International Publishing, 2022. (Texts in Computer Science). ISBN 978-3-030-34372-9. 21

TETILA, E. C. et al. Yolo performance analysis for real-time detection of soybean pests. *Smart Agricultural Technology*, Elsevier, v. 7, p. 100405, 2024. 28

WANGENHEIM, A. von. *Deep Learning::Aprendizado por Transferência e Ajuste Fino*. 2018. Disponível em: https://lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/deep-learningaprendizado-por-transferencia-e-ajuste-fino/. 20

Apêndices

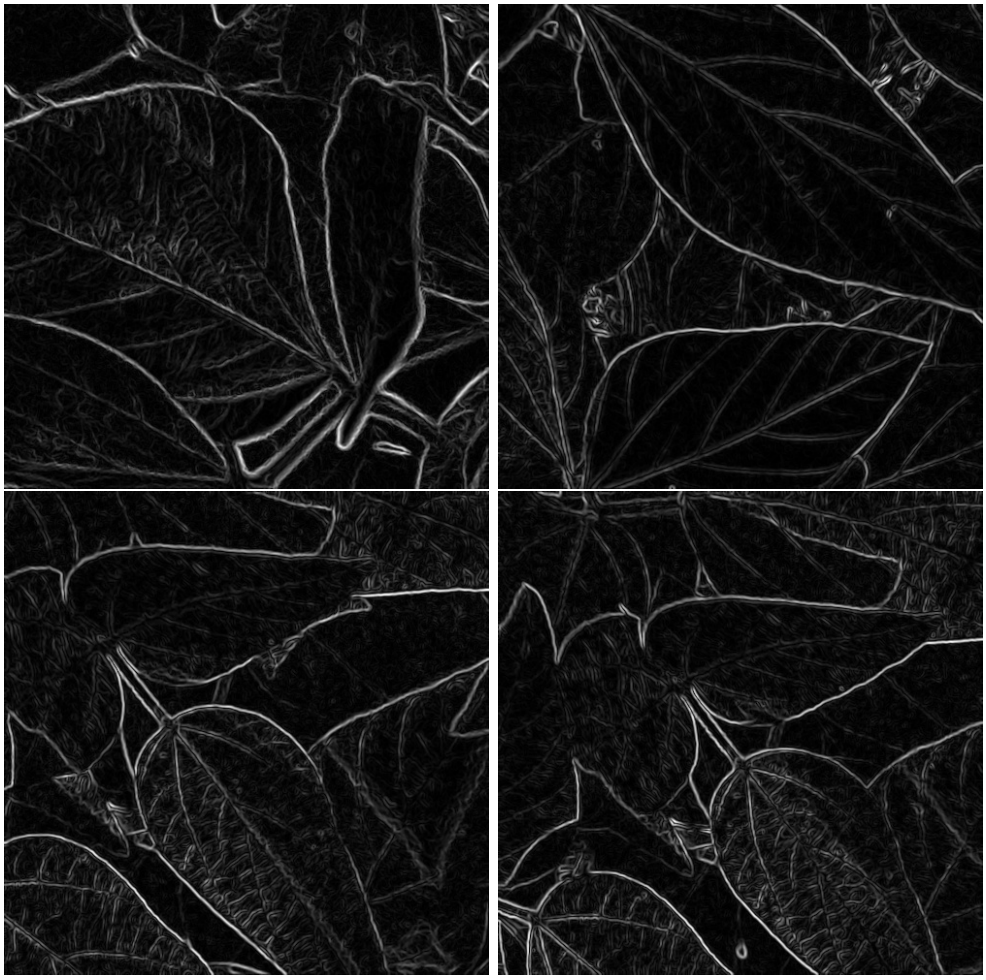
APÊNDICE A – PRÉ-PROCESSAMENTO COM VETOR GRADIENTE

Neste apêndice, são apresentadas as imagens resultantes do pré-processamento utilizando o vetor gradiente, bem como o código correspondente à sua implementação. Embora a técnica tenha gerado imagens com destaque nos contornos das folhas, não se observaram ganhos significativos no desempenho dos modelos. Além disso, sua adoção aumentou a complexidade do *endpoint* de predição. Ainda assim, este experimento é incluído como registro de um caso de teste, ilustrando a aplicação de diferentes abordagens de pré-processamento.

Figura 21 – Contorno de imagens com praga

Fonte: Elaborada pelo autor.

Figura 22 – Contorno de imagens saudáveis



Fonte: Elaborada pelo autor.

Código A.1 – Código *Python* utilizado para fazer o pré-processamento e gerar as imagens com contorno

```python
import cv2
import numpy as np
import os

def preprocess_image(image_path):
    # Carregar a imagem
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Aplicar o operador Sobel nos eixos x e y para calcular as derivadas de intensidade
    sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
    sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)

    # Calcular a magnitude do gradiente
    magnitude_gradient = np.sqrt(sobelx**2 + sobely**2)

    # Normalizar os valores para 0-255 (8 bits)
    magnitude_gradient = cv2.normalize(magnitude_gradient, None, 0, 255, cv2.NORM_MINMAX)

    # Converter para tipo de dados uint8 (8 bits)
    magnitude_gradient = np.uint8(magnitude_gradient)

    return magnitude_gradient

input_directory = './images/'

output_directory = './images_preprocessed/'

if not os.path.exists(output_directory):
    os.makedirs(output_directory)

for filename in os.listdir(input_directory):
    if filename.endswith('.jpg') or filename.endswith('.png'):
        input_path = os.path.join(input_directory, filename)

        preprocessed_image = preprocess_image(input_path)

        output_path = os.path.join(output_directory, filename)
        cv2.imwrite(output_path, preprocessed_image)

        print(f'Imagem {filename} pré-processada e salva com sucesso em {output_path}')
```

APÊNDICE B – CÓDIGOS DE PREPARAÇÃO DO *DATASET*

Neste apêndice, são apresentados os códigos utilizados para a extração e rotulagem das imagens que compõem o *dataset*, garantindo a organização dos dados para o treinamento dos modelos.

Código B.1 – Código *Python* utilizado para extrair as imagens do dataset INSECT12C

```python
import os
import shutil
import xml.etree.ElementTree as ET

def renomear_e_mover_imagens_com_xml(diretorio_entrada, diretorio_saida):
    # Inicializar o índice como 1
    idx = 3222
    # Loop pelos arquivos no diretório de entrada
    for arquivo in os.listdir(diretorio_entrada):
        if arquivo.endswith(".xml"):
            caminho_arquivo_xml = os.path.join(diretorio_entrada, arquivo)
            # Parsing do arquivo XML
            tree = ET.parse(caminho_arquivo_xml)
            root = tree.getroot()
            # Procurar pela primeira aparição da tag <name>
            for name_tag in root.iter('name'):
                if name_tag is not None:
                    nome_novo = f"{name_tag.text}_{idx}"  # Adicionando o índice ao nome
                    idx += 1  # Incrementar o índice para o próximo arquivo
                    print(nome_novo)
                    # Verificar se existe uma imagem correspondente
                    nome_imagem = arquivo.replace(".xml", ".jpg")  # Assumindo que as imagens tenham extensão .jpg
                    caminho_imagem_origem = os.path.join(diretorio_entrada, nome_imagem)
                    if os.path.exists(caminho_imagem_origem):
                        # Mover a imagem renomeada para o diretório de saída
                        novo_nome_imagem = f"{nome_novo}.jpg"
                        caminho_imagem_destino = os.path.join(diretorio_saida, novo_nome_imagem)
                        shutil.copy(caminho_imagem_origem, caminho_imagem_destino)
                else:
                    print(f"Arquivo {arquivo} não contém a tag <name>.")
```

```
31
32 # Diretório de entrada com os arquivos XML e as imagens
33 diretorio_entrada = '../part-12-12-img-1651-1800'
34 # Diretório de saída para as imagens renomeadas
35 diretorio_saida = './novas'
36
37 renomear_e_mover_imagens_com_xml(diretorio_entrada, diretorio_saida)
```

Código B.2 – Código *Python* utilizado para renomear imagens do *Dataset* SoyNet

```
1  import os
2
3  def renomear_imagens(diretorio, nome_antigo, nome_novo):
4      # Listar todos os arquivos no diretório
5      for nome_arquivo in os.listdir(diretorio):
6          # Verificar se o arquivo contém a parte do nome que queremos modificar
7          if nome_antigo in nome_arquivo:
8              # Separar o número do resto do nome
9              parte_numero = nome_arquivo.replace(nome_antigo, "").split('.')[0]
10             extensao = nome_arquivo.split('.')[-1]
11             novo_nome_arquivo = f"{nome_novo}{parte_numero}.{extensao}"
12             caminho_antigo = os.path.join(diretorio, nome_arquivo)
13             caminho_novo = os.path.join(diretorio, novo_nome_arquivo)
14             # Renomear o arquivo
15             os.rename(caminho_antigo, caminho_novo)
16             print(f"{nome_arquivo} renomeado para {novo_nome_arquivo}")
17
18 # Diretório onde as imagens estão localizadas
19 diretorio_imagens = "./novas"
20
21 # Nomes antigos e novos
22 nome_antigo = "Soy"
23 nome_novo = "Saudavel"
24
25 # Chamar a função para renomear as imagens
26 renomear_imagens(diretorio_imagens, nome_antigo, nome_novo)
```

Código B.3 – Código *Python* utilizado para renomear as imagens e ficar no mesmo padrão

```
1  import os
2  import re
3
4  def rename_files(directory):
5      for filename in os.listdir(directory):
6          # Match the pattern "text (number).extension"
7          match = re.match(r'(.+?) \((\d+)\)(\.\w+)', filename)
8          if match:
9              new_name = f"{match.group(1)}_{match.group(2)}{match.group(3)}"
```

```
            old_path = os.path.join(directory, filename)
            new_path = os.path.join(directory, new_name)
            os.rename(old_path, new_path)
            print(f'Renamed: {filename} to {new_name}')

# Specify the directory containing the images
directory = './novas'

rename_files(directory)
```

APÊNDICE C – CÓDIGO DOS MODELOS COM FASTAI E DADOS DE TREINAMENTO

Este apêndice apresenta os códigos completos utilizados na construção, treinamento e exportação do modelo de classificação de imagens da cultura da soja, utilizando a biblioteca FastAI. O código é usado como base para os testes iniciais e está organizado em etapas: preparação dos dados, definição do modelo, treinamento, avaliação e exportação do modelo treinado.

Código C.1 – Código *Python* utilizado para realizar o treinamento do modelo binário

```
from fastai.vision.all import *

# Baixando e preparando os dados
path = Path("./images")
path.ls()

files = get_image_files(path)
len(files)

# Criando um DataLoader para os dados
dls = ImageDataLoaders.from_name_func(
    path, files,valid_pct=0.18, label_func=lambda x: 'praga' if 'praga' in x else 'saudavel', bs=9,
    item_tfms=Resize(460)
)

dls.show_batch()

# Criando o modelo
learn = vision_learner(dls, resnet34, metrics=[error_rate, accuracy])

# Treinando o modelo
learn.fine_tune(20)

learn.predict(files[1])
#('praga', tensor(0), tensor([9.9998e-01, 2.0970e-05]))

learn.show_results(max_n=16)

```

```
# Matriz de confusão
interp = ClassificationInterpretation.from_learner(learn)
losses,idxs = interp.top_losses()
len(dls.valid_ds)==len(losses)==len(idxs)
interp.plot_confusion_matrix(figsize=(7,7))


# Export do modelo treinado
import os, dill
print(os.getcwd())

temp_path = Path(os.getcwd())
learn.path = temp_path
learn.export("model_soy_v2.pkl", pickle_module=dill)
```

Tabela 5 – Métricas de Treinamento por época do modelo binário de 20 épocas

Época	train_loss	valid_loss	error_rate	accuracy	Tempo
0	0.494658	0.628923	0.250000	0.750000	00:47
1	0.467058	0.691123	0.312500	0.687500	00:49
2	0.402001	0.503400	0.187500	0.812500	00:51
3	0.308360	0.463978	0.125000	0.875000	00:51
4	0.285354	0.414078	0.125000	0.875000	00:50
5	0.242807	0.349005	0.187500	0.812500	00:51
6	0.222190	0.241891	0.125000	0.875000	00:51
7	0.191605	0.288296	0.125000	0.875000	00:51
8	0.167056	0.425176	0.187500	0.812500	00:51
9	0.163520	0.320135	0.250000	0.750000	00:50
10	0.153290	0.376019	0.187500	0.812500	00:50
11	0.153842	1.147473	0.250000	0.750000	00:51
12	0.151626	1.499742	0.250000	0.750000	00:51
13	0.134107	1.419184	0.250000	0.750000	00:50
14	0.137144	1.239196	0.187500	0.812500	00:51
15	0.161592	1.122660	0.250000	0.750000	00:52
16	0.141539	1.071537	0.250000	0.750000	00:50
17	0.120893	1.121560	0.250000	0.750000	00:52
18	0.107450	1.142910	0.250000	0.750000	00:51
19	0.096448	1.255381	0.187500	0.812500	00:51

Fonte: Elaborada pelo autor.

Tabela 6 – Métricas de treinamento por época do modelo binário de 25 épocas

Época	**train_loss**	**valid_loss**	**error_rate**	**accuracy**	**Tempo**
0	1.432177	1.047810	0.327354	0.672646	00:27
1	1.067323	0.935273	0.300448	0.699552	00:29
2	0.851556	0.812719	0.251121	0.748879	00:28
3	0.692472	0.927961	0.273543	0.726457	00:28
4	0.670447	0.973255	0.246637	0.753363	00:29
5	0.678778	0.764895	0.251121	0.748879	00:28
6	0.655417	0.734745	0.233184	0.766816	00:28
7	0.601530	0.794843	0.224215	0.775785	00:29
8	0.582233	0.758315	0.192825	0.807175	00:28
9	0.449984	0.795747	0.192825	0.807175	00:29
10	0.427303	0.822391	0.192825	0.807175	00:29
11	0.442523	0.601252	0.192825	0.807175	00:27
12	0.382882	0.672555	0.197309	0.802691	00:28
13	0.311674	0.645711	0.183856	0.816144	00:31
14	0.304318	0.683406	0.183856	0.816144	00:28
15	0.246841	0.700002	0.165919	0.834081	00:28
16	0.233597	0.614762	0.165919	0.834081	00:29
17	0.188161	0.643135	0.170404	0.829596	00:27
18	0.203307	0.713457	0.201794	0.798206	00:29
19	0.142669	0.714160	0.197309	0.802691	00:30
20	0.139220	0.649953	0.179372	0.820628	00:29
21	0.141911	0.669219	0.183856	0.816144	00:30
22	0.159482	0.655573	0.161435	0.838565	00:28
23	0.140160	0.649136	0.170404	0.829596	00:28
24	0.112131	0.637959	0.152466	0.847534	00:27

Fonte: Elaborada pelo autor.

Código C.2 – Código *Python* utilizado para realizar o treinamento do modelo multi classe

```
from fastai.vision.all import *
from google.colab import drive
drive.mount('/content/drive')

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

# pegando caminho dos dados
path = Path("/content/drive/MyDrive/Tcc-imagens/rotuladas")
path.ls()

files = get_image_files(path)
len(files)
pat = r'^(.*)_\d+.[jJ][pP][gG]$'

# Criando um DataLoader para os dados
dls = ImageDataLoaders.from_name_re(path, files,pat,bs=9,item_tfms=Resize(460))

dls.show_batch()

# Criando o modelo
# Drop nodes ps=0.25
learn = vision_learner(dls, resnet34, metrics=[error_rate, accuracy], ps=0.25)

# Treinando o modelo
learn.fine_tune(50)

interp = Interpretation.from_learner(learn)
interp.plot_top_losses(9, figsize=(15,10))

learn.show_results(max_n=9)

# Matriz de confusão
interp = ClassificationInterpretation.from_learner(learn)
losses,idxs = interp.top_losses()
len(dls.valid_ds)==len(losses)==len(idxs)
interp.plot_confusion_matrix(figsize=(7,7))

# Salva o modelo
import os
print(os.getcwd())

temp_path = Path(os.getcwd())
learn.path = temp_path
learn.export("model_soy_v4.pkl")

```

```
# carrega o modelo para testes
learn = load_learner("/content/drive/MyDrive/Tcc-imagens/modelos/model_soy_v4.pkl")

# Testar modelo com imagem diferente do dataset
img_t = Image.open("/content/Fig.-6.png")
image_rgba = img_t.convert("RGBA")
show_image(image_rgba)

pred_class, pred_idx, outputs = learn.predict(img_t)
print(pred_class)

classes = learn.dls.vocab
class_probs = {cls: prob for cls, prob in zip(classes, outputs)}

for cls, prob in class_probs.items():
  print(f"Classe: {cls}, Probabilidade: {prob}")

# Saída do teste
#Vaquinha_verde_amarelo
#Classe: Lagarta_das_vagens, Probabilidade: 0.11671234667301178
#Classe: Lagarta_soja, Probabilidade: 3.235928431877255e-07
#Classe: Percevejo_soja, Probabilidade: 0.006084074266254902
#Classe: Saudavel, Probabilidade: 0.4375137686729431
#Classe: Vaquinha_verde_amarelo, Probabilidade: 0.4396894574165344

import os
from collections import Counter
import re

def contar_nomes_base(diretorio):
    # Listar todos os arquivos no diretório
    arquivos = os.listdir(diretorio)

    # Expressão regular para extrair o nome base
    padrao = re.compile(r'^(.*)_\d+.[jJ][pP][gG]$')

    # Lista para armazenar os nomes base
    nomes_base = []

    for arquivo in arquivos:
        # Verificar se o arquivo corresponde ao padrão
        match = padrao.match(arquivo)
        if match:
            nome_base = match.group(1)  # Captura o primeiro grupo (nome base)
            nomes_base.append(nome_base)

    # Contar as ocorrências de cada nome base
```

```
    contagem = Counter(nomes_base)

    return contagem

contagem = contar_nomes_base(path)

# Exibir os resultados
for nome_base, quantidade in contagem.items():
    print(f"{nome_base}: {quantidade}")

# Distribuição das images
#Saudavel: 152
#Lagarta_soja: 135
#Percevejo_soja: 133
#Lagarta_das_vagens: 235
#Vaquinha_verde_amarelo: 146
```

Código C.3 – Código *Python* utilizado para criar os gráficos com as curvas de treinamento

```
import matplotlib.pyplot as plt
import pandas as pd

#df = pd.read_csv('dados_20epocas.csv')
df = pd.read_csv('modelo_50epocas.csv')

plt.plot(df['train_loss'], label='erro dos dados de treinamento')
plt.plot(df['valid_loss'], label='erro dos dados de validacao')
plt.title('grafico de erro de treino e validação')
plt.legend()
```

Tabela 7 – Métricas de treinamento por época para modelo multiclasse (épocas 0-24)

Época	**train_loss**	**valid_loss**	**error_rate**	**accuracy**	**Tempo**
0	0.750583	0.539794	0.212500	0.787500	00:55
1	0.462953	0.449093	0.143750	0.856250	00:48
2	0.310686	0.450971	0.125000	0.875000	00:48
3	0.310138	0.370443	0.125000	0.875000	00:52
4	0.195789	0.436479	0.137500	0.862500	00:51
5	0.203934	0.389155	0.112500	0.887500	00:49
6	0.164042	0.469876	0.106250	0.893750	00:48
7	0.222397	0.473591	0.112500	0.887500	00:48
8	0.233088	0.530806	0.112500	0.887500	00:55
9	0.251422	0.374321	0.087500	0.912500	00:48
10	0.260531	0.405281	0.118750	0.881250	00:47
11	0.217025	0.426163	0.112500	0.887500	00:49
12	0.256851	0.315940	0.087500	0.912500	00:50
13	0.209406	0.346258	0.087500	0.912500	00:51
14	0.186622	0.268095	0.087500	0.912500	00:49
15	0.232231	0.286633	0.087500	0.912500	00:48
16	0.195488	0.426706	0.100000	0.900000	00:47
17	0.177343	0.454158	0.125000	0.875000	00:51
18	0.179295	0.291695	0.068750	0.931250	00:48
19	0.220668	0.240355	0.068750	0.931250	00:47
20	0.190386	0.343831	0.075000	0.925000	00:49
21	0.192755	0.268708	0.062500	0.937500	00:49
22	0.130909	0.264512	0.068750	0.931250	00:52
23	0.136885	0.259368	0.068750	0.931250	00:49
24	0.094001	0.232440	0.056250	0.943750	00:49

Fonte: Elaborada pelo autor.

Tabela 8 – Métricas de treinamento por época para modelo multiclasse (épocas 25-49)

Época	**train_loss**	**valid_loss**	**error_rate**	**accuracy**	**Tempo**
25	0.103191	0.231849	0.068750	0.931250	00:49
26	0.060279	0.200760	0.037500	0.962500	00:47
27	0.064237	0.235601	0.068750	0.931250	00:51
28	0.049985	0.221156	0.062500	0.937500	00:49
29	0.059801	0.208035	0.068750	0.931250	00:48
30	0.060848	0.266527	0.062500	0.937500	00:48
31	0.069721	0.178564	0.050000	0.950000	00:51
32	0.069506	0.220716	0.043750	0.956250	00:49
33	0.038126	0.201411	0.043750	0.956250	00:49
34	0.028542	0.193177	0.056250	0.943750	00:48
35	0.034205	0.207414	0.025000	0.975000	00:48
36	0.039472	0.178094	0.037500	0.962500	00:50
37	0.032981	0.226685	0.050000	0.950000	00:48
38	0.023972	0.200616	0.037500	0.962500	00:48
39	0.027961	0.197650	0.050000	0.950000	00:49
40	0.026840	0.167531	0.037500	0.962500	00:51
41	0.021239	0.164701	0.031250	0.968750	00:47
42	0.027375	0.198191	0.037500	0.962500	00:48
43	0.027507	0.186982	0.037500	0.962500	00:47
44	0.012512	0.182188	0.025000	0.975000	00:49
45	0.016851	0.178335	0.037500	0.962500	00:51
46	0.010405	0.181221	0.025000	0.975000	00:47
47	0.020428	0.183129	0.043750	0.956250	00:47
48	0.020893	0.183957	0.031250	0.968750	00:46
49	0.021614	0.181813	0.025000	0.975000	00:51

Fonte: Elaborada pelo autor.