Lucas Duarte Sobreira
Tarek Campos Saleh

Sistema de inspeção visual para detecção de Sarna em Folhas de Macieiras

Goiânia
2024

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as):

Lucas Duarte Sobreira (201802734) e Tarek Campos Saleh (201802747)

Título do trabalho:

SISTEMA DE INSPEÇÃO VISUAL PARA DETECÇÃO DE SARNA EM FOLHAS DE MACIEIRAS

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Cassio Dener Noronha Vinhal**, **Professor do Magistério Superior**, em 22/12/2024, às 13:15, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Lucas Duarte Sobreira**, **Discente**, em 22/12/2024, às 13:26, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Tarek Campos Saleh**, **Discente**, em 22/12/2024, às 14:18, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5060543** e o código CRC **8BE5A88F**.

Referência: Processo nº 23070.043763/2024-01 SEI nº 5060543

Lucas Duarte Sobreira
Tarek Campos Saleh

Sistema de inspeção visual para detecção de Sarna em Folhas de Macieiras

Trabalho de conclusão de curso apresentado na Escola de Engenharia Elétrica, Mecânica e de Computação como requisito para a conclusão do curso de Engenharia de Computação e obtenção do título de bacharel em Engenharia de Computação

Orientador: Prof. Dr. Cássio Dener Noronha Vinhal

Universidade Federal de Goiás - UFG

Escola de Engenharia Elétrica, Mecânica e de Computação (EMC)

Goiânia

2024

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Sobreira, Lucas Duarte
Sistema de inspeção visual para detecção de Sarna em Folhas de Macieiras [manuscrito] / Lucas Duarte Sobreira, Tarek Campos Saleh. - 2024.
21 f.

Orientador: Prof. Dr. Cássio Dener Noronha Vinhal.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia da Computação, Goiânia, 2024.
Bibliografia. Apêndice.
Inclui siglas, gráfico, tabelas, lista de figuras.

1. Sarna da macieira. 2. CNNs. 3. aprendizado profundo. 4. MobileNet V3. 5. diagnóstico agrícola. I. Saleh, Tarek Campos. II. Vinhal, Cássio Dener Noronha, orient. III. Título.

CDU 004

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Aos **dezenove dias do mês de dezembro do ano de 2024, 17h00** , iniciou-se a sessão pública de defesa do Projeto Final de Curso II - Trabalho de Conclusão de Curso - TCC intitulado **"SISTEMA DE INSPEÇÃO VISUAL PARA DETECÇÃO DE SARNA EM FOLHAS DE MACIEIRAS"** , de autoria de **Lucas Duarte Sobreira (201802734) e Tarek Campos Saleh (201802747)**, do curso de Engenharia de Computação, da Escola de Engenharia Elétrica, Mecânica e Computação (EMC) da UFG. Os trabalhos foram instalados pelo(a) **Prof. Dr. Cássio Dener Noronha Vinhal (EMC/UFG)**, presidente, com a participação dos demais membros da Banca Examinadora: **Prof. Dr. Marco Antonio Assfalk de Oliveira (EMC/UFG) e Prof. Dr. Gelson da Cruz Júnior (EMC/UFG)** . Após a apresentação, a banca examinadora realizou a arguição dos estudantes. Posteriormente, de forma reservada, a Banca Examinadora atribuiu a nota final de **9,0 (nove)**, tendo sido o TCC considerado **aprovado**.

Proclamados os resultados, os trabalhos foram encerrados e, para constar, lavrou-se a presente Ata que segue assinada pelos Membros da Banca Examinadora.

Documento assinado eletronicamente por **Marco Antonio Assfalk De Oliveira**, **Professor do Magistério Superior**, em 19/12/2024, às 19:05, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Gelson Da Cruz Junior**, **Professor do Magistério Superior**, em 19/12/2024, às 19:05, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Cassio Dener Noronha Vinhal**, **Professor do Magistério Superior**, em 19/12/2024, às 19:07, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5056383** e o código CRC **738231FE**.

Referência: Processo nº 23070.043763/2024-01 SEI nº 5056383

Sistema de inspeção visual para detecção de Sarna em Folhas de Macieiras

Lucas Duarte Sobreira e Tarek Campos Saleh
Escola de Engenharia Elétrica, Mecânica e da Computação
Universidade Federal de Goiás
Goiânia, Goiás, Brasil
Email: {lucasds123 & tarekcamposaleh}@discente.ufg.br

Dr. Cássio Dener Noronha Vinhal
Escola de Engenharia Elétrica, Mecânica e da Computação
Universidade Federal de Goiás
Goiânia, Goiás, Brasil
Email: vinhal@ufg.br

***Abstract*—Apple scab, caused by the fungus Venturia inaequalis, poses a substantial threat to apple production, particularly for small-scale farmers in resource-constrained environments. Traditional visual inspection methods are often ineffective, necessitate technical expertise, and are impractical on a large scale. This work proposes an economical and offline-capable solution for detecting apple scab, employing lightweight Convolutional Neural Networks (CNNs) and an intuitive web application. Images were collected from three datasets—ATLDSD, eScab, and AppleScabLDs—and class imbalance was mitigated through data augmentation. Various models were evaluated, including MobileNet V3, ShuffleNet V2, and RegNet, with ShuffleNet V2 achieving the highest accuracy (92.83%). The system integrates a web application based on Next.js and React.js, hosted on Vercel, with local inference utilizing ONNXRuntime-web and Jimp for image preprocessing. This solution provides farmers with an accessible, efficient, and offline diagnostic tool, contributing to enhanced agricultural productivity and improved disease management.**

***Index Terms*—Apple scab, CNNs, deep learning, MobileNet V3, agricultural diagnostics.**

RESUMO

A sarna-da-macieira, causada pelo fungo *Venturia inaequalis*, representa uma ameaça significativa à produção de maçãs, especialmente para pequenos agricultores em ambientes com recursos limitados. Os métodos tradicionais de inspeção visual frequentemente se mostram ineficazes, exigem expertise técnica e são impraticáveis em larga escala. Este trabalho propõe uma solução econômica e compatível com funcionamento offline para a detecção da sarna-da-macieira, utilizando Redes Neurais Convolucionais (CNNs) leves e uma aplicação web intuitiva. As imagens foram coletadas a partir de três conjuntos de dados — ATLDSD, eScab e AppleScabLDs [1] —, e o desequilíbrio de classes foi mitigado por meio de aumento de dados (data augmentation). Diversos modelos foram avaliados, incluindo MobileNet V3, ShuffleNet V2 e RegNet, sendo que o ShuffleNet V2 alcançou a maior precisão (92,83%). O sistema integra uma aplicação web baseada em Next.js e React.js, hospedada no Vercel, com inferência local utilizando ONNXRuntime-web e Jimp para pré-processamento de imagens. Esta solução oferece aos agricultores uma ferramenta diagnóstica acessível, eficiente e offline, contribuindo para o aumento da produtividade agrícola e para uma gestão mais eficaz de doenças.

[1]*Apple Tree Leaf Diseases Segmentation Dataset* (ATLDSD), *Early Scab* (eScab) e *Apple Scab Leaf Disease*(AppleScabLDs)

Palavras-chave

Sarna da macieira, CNNs, aprendizado profundo, MobileNet V3, diagnóstico agrícola.

I. INTRODUÇÃO

A sarna da macieira é uma doença fúngica que afeta severamente a produção de maçãs, causando deformações nas folhas e frutos, resultando em perdas de qualidade e quantidade. Essa doença, causada pelo fungo *Venturia inaequalis*, reduz a área fotossintética das folhas, comprometendo o desenvolvimento dos frutos e impactando negativamente a produtividade agrícola. Para pequenos agricultores e agrônomos, especialmente aqueles localizados em regiões com recursos limitados, o diagnóstico precoce é essencial para mitigar as perdas econômicas e evitar a disseminação da doença [1].

Os métodos tradicionais de detecção baseiam-se, em sua maioria, na inspeção visual das plantas, que é um processo trabalhoso, sujeito a erros humanos e dependente de especialistas com conhecimento técnico avançado. Além disso, esses métodos frequentemente falham em identificar a doença em seus estágios iniciais, quando intervenções rápidas poderiam conter a infecção e evitar danos severos. Outras soluções mais modernas, como dispositivos dedicados de análise laboratorial ou equipamentos baseados em sensores de alta tecnologia, possuem altos custos de aquisição e manutenção, tornando-os inviáveis para pequenos produtores rurais [2].

Neste contexto, o uso de inteligência artificial, particularmente CNNs, surge como uma alternativa promissora para automatizar e democratizar o diagnóstico de doenças agrícolas. As CNNs têm demonstrado eficácia em tarefas complexas de classificação de imagens, como a identificação de padrões específicos em folhas infectadas, possibilitando a detecção precoce de doenças de forma rápida e precisa. Comparada aos métodos tradicionais, a solução proposta oferece maior rapidez, redução de custos operacionais e independência de conhecimento especializado, enquanto fornece resultados com maior confiabilidade e consistência.

O presente trabalho propõe uma abordagem prática para a detecção da sarna da macieira, integrando modelos de CNN com uma aplicação web de uso offline. As imagens utilizadas foram extraídas de três conjuntos de dados amplamente reconhecidos: ATLDSD, eScab e AppleScabLDs. Para corrigir o desbalanceamento entre as classes saudáveis e doentes, aplicou-se técnicas de aumento de dados, garantindo a robustez do modelo treinado.

O treinamento do modelo foi conduzido utilizando técnicas modernas de aprendizado profundo, como transferência de aprendizado. Diversos modelos compactos, incluindo MobileNet V3, ShuffleNet V2 e EfficientNet B0, foram avaliados, sendo o MobileNet V3 selecionado como modelo final devido ao seu equilíbrio entre alta precisão (92,01%) e baixo consumo de recursos computacionais. Comparativamente, essa abordagem supera soluções tradicionais pela sua capacidade de operar em dispositivos de baixo custo, mantendo alta acurácia e eficiência, características fundamentais para aplicações em ambientes rurais.

Além disso, foi desenvolvida uma aplicação web utilizando Next.js e React.js, hospedada na plataforma Vercel. O sistema permite a execução de inferências diretamente no navegador, eliminando a necessidade de conectividade constante com servidores. Para isso, utiliza-se o *onnxruntime-web* para execução dos modelos em formato *Open Neural Network Exchange* (ONNX) e a biblioteca *JavaScript Image Manipulation Program* (Jimp) para pré-processamento de imagens. Essa abordagem resolve limitações de conectividade e reduz custos operacionais, tornando o sistema acessível para pequenos agrônomos que precisam diagnosticar rapidamente folhas potencialmente infectadas em locais remotos.

Desta forma, este artigo apresenta uma solução completa, alinhada às necessidades de pequenos agricultores, abrangendo desde a coleta e tratamento de dados até a validação e aplicação prática do modelo. Em comparação com métodos existentes, a abordagem proposta oferece uma solução economicamente viável, escalável e de alta acurácia, contribuindo para democratizar o uso de inteligência artificial no diagnóstico de doenças agrícolas. Nos tópicos subsequentes, serão abordados os conceitos teóricos, a metodologia empregada, os resultados obtidos e as contribuições desta pesquisa para o estado da arte no diagnóstico automatizado de doenças agrícolas.

II. Revisão Bibliográfica

A. Redes Neurais Artificiais

Redes neurais artificiais (RNAs) são modelos computacionais inspirados no funcionamento biológico do cérebro humano, estruturadas em camadas de neurônios artificiais que processam informações através de conexões ponderadas [4]. Cada neurônio recebe entradas, aplica uma função de ativação e transmite a saída para os neurônios subsequentes. Esses modelos são amplamente utilizados em tarefas que requerem aprendizado não linear, como classificação, regressão e detecção de padrões [3].

B. Redes Neurais Profundas

Redes neurais profundas (DNNs, do inglês *Deep Neural Networks*) são uma extensão das RNAs tradicionais, caracterizadas por possuírem múltiplas camadas intermediárias, conhecidas como camadas ocultas. Essa estrutura em profundidade permite que as DNNs aprendam representações hierárquicas de dados, extraindo características mais complexas e abstratas em tarefas como visão computacional e processamento de linguagem natural. A eficiência das DNNs está associada a avanços computacionais e ao aumento de dados disponíveis para treinamento [5].

C. Algoritmo de Backpropagation

O *backpropagation*, ou retropropagação, é o algoritmo mais comum para o treinamento de redes neurais, baseando-se na minimização de uma função de custo através do cálculo do gradiente do erro. O processo ocorre em duas fases: uma propagação direta para calcular o erro e uma propagação reversa para ajustar os pesos utilizando o método do gradiente descendente. Esse algoritmo é fundamental para o aprendizado supervisionado em redes neurais [3].

D. Redes Neurais Convolucionais

As CNNs são uma classe especializada de DNNs, particularmente eficientes em tarefas de visão computacional. CNNs utilizam camadas convolucionais para extrair características espaciais e contextuais de dados de entrada, como imagens. Essas camadas aplicam filtros, ou *kernels*, que destacam padrões locais, como bordas e texturas, sendo seguidas por operações de *pooling* para reduzir a dimensionalidade e aumentar a robustez às variações dos dados [6].

A Figura 1 ilustra o fluxo básico de processamento de uma CNN, desde a entrada da imagem até a avaliação probabilística. Inicialmente, a imagem da folha é processada pelas camadas convolucionais, que geram mapas de características (*feature maps*) ao identificar padrões relevantes, como manchas ou texturas associadas à sarna da macieira. Em seguida, essas características passam por uma camada totalmente conectada (*fully connected layer*) que classifica as imagens com base nos padrões aprendidos. Por fim, o modelo gera uma avaliação probabilística, atribuindo uma probabilidade a cada classe (ex.: saudável ou infectada).

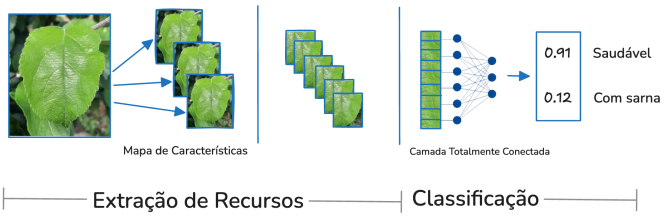


Figura 1: Fluxo de processamento em uma CNN, desde a entrada da imagem até a classificação.

Esse fluxo de processamento evidencia a capacidade das CNNs de extrair recursos hierárquicos, o que as torna uma ferramenta poderosa para o diagnóstico automatizado de doenças agrícolas, como a sarna da macieira.

E. Modelos Finais e Armazenamento

Após o treinamento, os modelos de aprendizado de máquina são frequentemente armazenados em formatos otimizados para inferência. No presente trabalho, utilizamos o formato ONNX, que permite a interoperabilidade entre diferentes plataformas e *frameworks*. O ONNX facilita a execução de modelos em dispositivos com recursos limitados, uma vez que é compatível com bibliotecas de execução eficientes como *onnxruntime-web* [7].

F. Kernels em CNNs

Um *kernel*, também conhecido como filtro, é uma matriz de pesos utilizada em operações convolucionais para extrair características locais de uma imagem ou de outros dados espaciais. Em CNNs, o *kernel* desliza pela entrada (como uma imagem), multiplicando seus valores pelos valores correspondentes na matriz de entrada, e somando os resultados para gerar um mapa de ativação. Essa operação permite detectar padrões, como bordas, texturas ou formas, que são cruciais para tarefas de classificação e segmentação. O tamanho do *kernel* (por exemplo, 3×3 ou 5×5) influencia diretamente a granularidade dos padrões extraídos [6].

G. Stride

O *stride* refere-se ao número de pixels pelos quais o kernel se desloca a cada passo durante a convolução. Um stride padrão de 1 implica que o kernel é movido um pixel por vez, enquanto um stride maior (por exemplo, 2 ou 3) reduz a sobreposição entre as regiões convoluídas e diminui a resolução do mapa de ativação resultante. O uso de *strides* maiores é uma forma eficiente de reduzir dimensionalidade, diminuindo o custo computacional sem a necessidade de técnicas adicionais de redução de dados, como *pooling* [8].

H. Max Pooling

O *max pooling* é uma operação de amostragem que reduz a dimensionalidade dos mapas de ativação em uma CNN, retendo as informações mais importantes. Ele funciona dividindo o mapa de entrada em regiões não sobrepostas e selecionando o valor máximo dentro de cada região. Por exemplo, em uma operação de *max pooling* com filtro 2×2 e stride 2, o mapa de ativação é reduzido pela metade em ambas as dimensões. Essa técnica não só reduz o custo computacional, mas também torna o modelo mais robusto a deslocamentos e distorções nos dados [9]. Além disso, operações de pooling podem ajudar a evitar o *overfitting* ao eliminar informações redundantes.

I. Métricas de Avaliação dos Modelos

A avaliação de modelos de aprendizado de máquina em tarefas de classificação é baseada em métricas que quantificam o desempenho em diferentes aspectos. Entre as métricas mais utilizadas neste trabalho estão:

1) Acurácia: A acurácia (*accuracy*) mede a proporção de predições corretas em relação ao total de amostras avaliadas. É calculada como:

$$\textit{Acurácia} = \frac{\textit{VP} + \textit{VN}}{\textit{Total de Amostras}} \tag{1}$$

onde VP e VN são, respectivamente, os verdadeiros positivos e os verdadeiros negativos. Embora amplamente utilizada, a acurácia pode ser influenciada por desbalanceamentos nas classes, não sendo ideal como única métrica em cenários onde uma classe predomina.

2) Precisão Média: A precisão (*precision*) é a proporção de verdadeiros positivos em relação ao total de predições positivas. Sua fórmula é:

$$\textit{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{2}$$

onde FP representa os falsos positivos. A precisão média é a média ponderada da precisão para todas as classes, sendo uma métrica relevante em contextos onde é importante minimizar alarmes falsos.

3) Revocação Média: A revocação (*recall*) indica a proporção de verdadeiros positivos em relação ao total de amostras positivas reais:

$$\textit{Revocação} = \frac{\textit{VP}}{\textit{VP} + \textit{FN}} \tag{3}$$

onde FN são os falsos negativos. A revocação média é calculada considerando todas as classes, sendo uma métrica essencial em aplicações que priorizam a detecção de casos positivos, mesmo ao custo de alguns falsos positivos.

4) F1-Score Médio: O F1-Score combina precisão e revocação em uma única métrica harmônica, sendo útil para balancear esses dois aspectos em problemas de classificação:

$$F1 = 2 \cdot \frac{\textit{Precisão} \cdot \textit{Revocação}}{\textit{Precisão} + \textit{Revocação}} \tag{4}$$

O F1-Score médio considera a média ponderada dos F1-Scores por classe, sendo particularmente relevante em conjuntos de dados desbalanceados.

5) Matriz de Confusão: A matriz de confusão é uma representação tabular das predições feitas pelo modelo em relação aos valores reais. É estruturada em termos de verdadeiros positivos (VP), verdadeiros negativos (VN), falsos positivos (FP) e falsos negativos (FN), conforme mostrado abaixo:

$$\begin{bmatrix} \text{VP} & \text{FP} \\ \text{FN} & \text{VN} \end{bmatrix}$$

Essa matriz fornece uma visão detalhada do desempenho do modelo, permitindo identificar padrões de erro específicos.

J. Componentes de Arquitetura das CNNs

1) Conv2d: A camada *Conv2d* realiza convoluções bidimensionais em dados de entrada, como imagens. Ela aplica filtros (*kernels*) para extrair características locais, como bordas e texturas. Seus hiperparâmetros incluem o tamanho do kernel, o *stride*, o preenchimento (*padding*) e o número de filtros [10].

2) BatchNormalization: A camada *BatchNormalization* normaliza os valores de ativação em uma camada para acelerar o treinamento e melhorar a estabilidade do modelo. Essa normalização reduz problemas como covariância interna, ajustando os valores para uma distribuição com média zero e variância unitária [8], [11].

3) ReLU: A função de ativação *Rectified Linear Unit* (ReLU) é amplamente utilizada em CNNs devido à sua simplicidade e eficiência computacional. Ela é definida como:

$$f(x) = \max(0, x) \tag{5}$$

onde x é a entrada. A *ReLU* introduz não linearidade no modelo e ajuda a mitigar o problema de gradientes desaparecendo [12].

4) AdaptativeAvgPool2d: A camada *AdaptativeAvgPool2d* reduz a dimensionalidade calculando a média de valores em regiões definidas da entrada, ajustando a saída para um tamanho pré-determinado. Essa camada é usada para criar representações compactas dos mapas de ativação antes da etapa de classificação [13].

5) Flatten: A camada *Flatten* transforma dados multidimensionais (como mapas de ativação) em um vetor unidimensional. Essa etapa é essencial para conectar as saídas de camadas convolucionais às camadas totalmente conectadas (*fully connected*) [14].

6) Linear: A camada *Linear*, também conhecida como camada totalmente conectada, realiza uma transformação linear dos dados de entrada, calculada como:

$$y = Wx + b \tag{6}$$

onde W representa os pesos, x é o vetor de entrada e b é o viés. Essa camada é crucial para a classificação final, transformando as representações aprendidas em probabilidades para cada classe [15].

K. Modelos Utilizados

Neste trabalho, foram explorados diferentes modelos de CNNs, com foco em arquiteturas leves e eficientes para implementação em dispositivos com recursos limitados. A seguir, cada modelo é descrito:

1) MobileNet V3: O MobileNet V3 foi projetado para dispositivos móveis e aplicações de baixa potência. Ele combina técnicas como convoluções separáveis em profundidade (*depthwise separable convolutions*) e blocos SandGlass para reduzir o número de parâmetros e operações computacionais, mantendo alta precisão. Além disso, utiliza o *MobileNetV3 Search Space*, que automatiza a otimização da arquitetura para diferentes contextos [16]. Esse modelo se destacou no trabalho por apresentar o melhor desempenho geral.

2) MNASNet: O MNASNet (*Mobile Neural Architecture Search Network*) é uma arquitetura otimizada automaticamente para dispositivos móveis utilizando aprendizado por reforço. Ele equilibra precisão e eficiência computacional ao explorar diversas combinações de camadas convolucionais e operações de *pooling*. Essa abordagem permite encontrar arquiteturas que maximizem o desempenho sob restrições específicas de hardware [17].

3) ShuffleNet V2: O ShuffleNet V2 é uma extensão do ShuffleNet original, projetado para melhorar a eficiência computacional em dispositivos de baixa potência. Ele introduz um mecanismo de "embaralhamento"de canais (*channel shuffle*) para aumentar a conectividade entre canais, além de utilizar convoluções ponto a ponto (*pointwise convolutions*) otimizadas. O modelo também é conhecido por sua baixa latência em inferências [18].

4) RegNet Y 400MF e RegNet Y 800MF: Os modelos RegNet (*Designing Network Design Spaces*) são CNNs projetadas para escalabilidade e simplicidade. No RegNet Y, os blocos convolucionais utilizam um número crescente de canais e parâmetros uniformemente distribuídos, permitindo maior eficiência em diferentes tamanhos de redes. Os modelos RegNet Y 400MF e 800MF representam versões otimizadas com diferentes números de parâmetros e custos computacionais, adequadas para tarefas como classificação de imagens [19].

5) EfficientNet B0 e EfficientNet B1: A família EfficientNet introduz um método de escalonamento composto (*compound scaling*) que otimiza simultaneamente a largura, profundidade e resolução das redes neurais. O EfficientNet B0 é o modelo base, que utiliza blocos MBConv e funções de ativação Swish para melhorar o desempenho e reduzir o consumo computacional. O EfficientNet B1 é uma versão ampliada, com maior capacidade de aprendizado, mas ainda eficiente em termos de parâmetros [20]. Esses modelos são amplamente reconhecidos por sua alta precisão em benchmarks de classificação de imagens, mantendo baixo custo computacional.

III. Metodologia

Houveram dois processos conduzidos para o desenvolvimento deste projeto. O primeiro, realizado para o desenvolvimento do modelo; o segundo, para o desenvolvimento do sistema. Estes são representados nas Figuras 2 e 3 respectivamente.

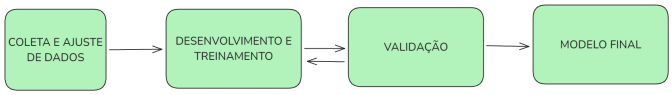


Figura 2: Visão global do processo de desenvolvimento do modelo

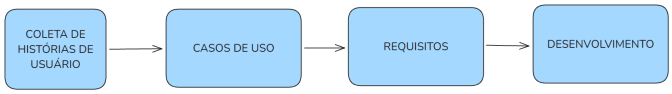


Figura 3: Visão global do processo de desenvolvimento do sistema

A sequência desta seção está organizada em subseções que cobrem, de forma sequencial, os aspectos metodológicos do projeto:

1) (III-A) Preparação do Ambiente de Trabalho, descrevendo as estações de trabalho e os programas utilizados;

2) (III-B) Coleta e Ajustes dos Dados, que detalha as origens e o tratamento dos dados utilizados;
3) (III-C) Desenvolvimento dos Modelos, onde são discutidas as abordagens de construção e treinamento das redes neurais; e
4) (III-D) Desenvolvimento da Aplicação, explicando as etapas da construção da aplicação web e os mecanismos de funcionamento offline;

A. Preparação do Ambiente de Trabalho

Para a realização do projeto, foi utilizada as duas estações de trabalho indicadas na Tabela I

	Estação Um	Estação Dois
GPU	NVIDIA GTX 1060 (3GB)	NVIDIA GTX 1060 (6GB)
RAM	16 GB DDR4 3600MHz	16 GB DDR4 3600MHz
CPU	AMD Ryzen 5700X	AMD Ryzen 5800X3D

Tabela I: Estações de trabalho

Para o desenvolvimento dos processos, as duas estações de trabalho contam com os programas e bibliotecas descritos na Tabela II.

Programa	Versão
Python	3.12
JupyterLab	4.2.0
PyTorch com CUDA [21]	2.51
Imgaug	0.4.0
Google Colaboratory	-
Neovim	0.10.2
Yarn	1.22.22
Node.js	22.12.0

Tabela II: Programas e Bibliotecas utilizadas

B. Coleta e Ajustes dos Dados

O processo de coleta e ajustes dos dados para a realização do desenvolvimento do modelo é descrito de modo geral na Figura 4 e detalhado nesta subseção.

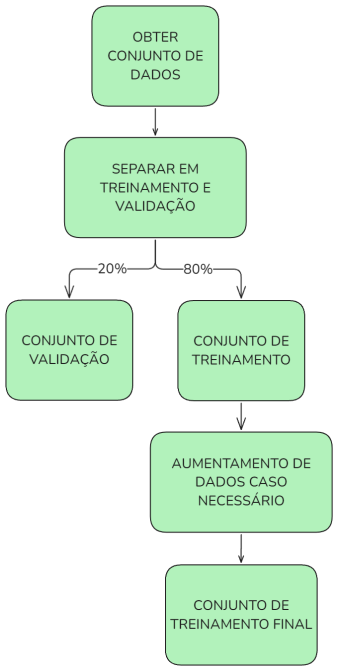


Figura 4: Visão geral do processo de ajuste de dados

1) Apresentação dos Conjuntos de Dados: Para a construção do conjunto de dados final, foram utilizados três conjuntos de dados principais:

- **ATLDSD** [22]: Contém 409 imagens de folhas saudáveis e 1232 imagens de folhas não saudáveis com diversas doenças, mas não inclui casos de sarna da macieira.
- **AppleScabLDS** [23]: Inclui 983 imagens de folhas de macieira com sarna e 207 imagens saudáveis.
- **eScabs** [24]: Composto por 700 imagens de folhas de macieira com sarna em estágio inicial e 768 imagens saudáveis capturadas em zoom.

2) Composição do Conjunto de Dados Final: O conjunto de dados final foi composto pelas imagens saudáveis do **ATLDSD**, **AppleScabLDs** e do **eScabs**, e pelas imagens com sarna do **AppleScabLDS** e do **eScabs**. A distribuição inicial das imagens no conjunto de dados final está apresentada na Tabela III.

Classe	Quantidade de Imagens
Saudáveis	1384
Com Sarna	1683

Tabela III: Distribuição inicial das classes no conjunto de dados final.

Exemplos do Conjunto de Dados: Para melhor ilustrar a composição do conjunto de dados final, a Figura 5 apresenta exemplos de imagens utilizadas. As imagens estão divididas em dois grupos: (a) folhas saudáveis e (b) folhas infectadas com sarna. Essa representação visual evidencia as principais diferenças entre as classes, como a presença de manchas e alterações texturais nas folhas infectadas, características importantes para a tarefa de classificação.

(a) Imagens de folhas saudáveis.

(b) Imagens de folhas infectadas com sarna.

Figura 5: Exemplos de imagens extraídas do conjunto de dados final. As imagens (a) representam folhas saudáveis, enquanto as imagens (b) mostram folhas infectadas com sarna.

3) Processamento e Balanceamento dos Dados: Para corrigir a distribuição de classes no conjunto de dados, foi utilizado o processo de aumentamento de dados. Devido a este processo realizado em imagens gerar alguns artefatos como bordas, para evitar um possível enviesamento do modelo, optou-se por também realizar o aumentamento na classe dominante, diferente do previsto no *Synthetic Minority Over-sampling Technique* (SMOTE), em que aplica-se apenas na classe minoritária [25]. Este processo consistiu em separar as imagens em dois grupos aleatoriamente, 80% de treinamento e 20% de validação, mantendo-se a distribuição original de classes do conjunto de dados completo, e em definir um número alvo de imagens em cada classe para realizar o aumentamento apenas nos dados de treinamento, calculado como:

$$\text{Número Alvo} = \text{Número de imagens da classe dominante} \cdot X$$

Com $X = 2$ e utilizando-se da biblioteca de apoio *'imgaug'*, foram geradas novas imagens para cada classe a partir da aplicação das transformações da Tabela IV em sequência aleatória até atingir um número próximo e superior ao alvo.

Transformação	Parâmetros Utilizados
Inversão Horizontal	50%
Rotação	[-45º, 45º]
Escala	[70%, 130%]
Translação Vertical	1%
Translação Horizontal	1%
Cisalhamento	[-20º, 20º]
Brilho	[50%, 150%]
Saturação	[50%, 150%]

Tabela IV: Transformações aplicadas no processo de aumentamento de dados.

4) Distribuição Final Corrigida: Após o balanceamento no treinamento, a distribuição final das classes nos dois conjuntos são apresentadas nas Tabelas V e VI.

Classe	Quantidade de Imagens
Saudáveis	2707
Com Sarna	2722

Tabela V: Distribuição final das classes no conjunto de dados de treinamento corrigido.

Classe	Quantidade de Imagens
Saudáveis	277
Com Sarna	337

Tabela VI: Distribuição final das classes no conjunto de dados de validação.

O processo garantiu uma base de dados balanceada, essencial para o treinamento eficiente dos modelos, além de garantir que não houvesse vazamento de dados entre os dois conjuntos.

C. Desenvolvimento e Treinamento dos Modelos

O processo de desenvolvimento e treinamento dos modelos é descrito de modo geral pela Figura 6 e detalhado nesta subseção.

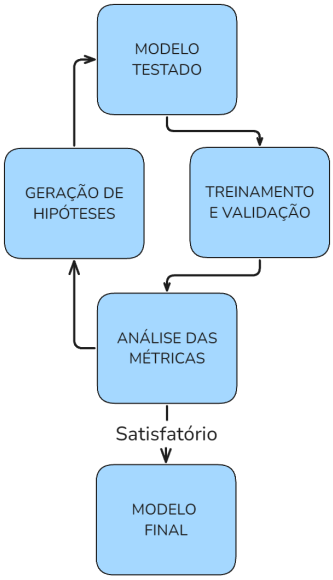


Figura 6: Visão geral do processo de desenvolvimento do modelo

Foram testados modelos compactos, incluindo MobileNet V3, MNASNET, ShuffleNet V2, RegNet Y 400/800MF, EfficientNet B0/B1, além de modelos próprios. As métricas avaliadas dos modelos foram Acurácia, Precisão Média, Revocação Média, *F1-Score* Médio e a Matriz de Confusão.

O pré-processamento de cada imagem foi ajustado conforme o modelo utilizado, exposto na Tabela VII, em que todos os

Tabela VII: Pré-processamento de Imagens por Modelo

Modelo	Redimensão (px)	Recorte (px)	Normalização	
			[0,1]	Média e Desvio Padrão
Próprio	256 x 256	256 x 256	Sim	Não
MobileNet V3	232 x 232	224 x 224	Sim	Sim
MNASNET	232 x 232	224 x 224	Sim	Sim
ShuffleNet V2	232 x 232	224 x 224	Sim	Sim
RegNet Y 400MF	256 x 256	224 x 224	Sim	Sim
RegNet Y 800MF	232 x 232	224 x 224	Sim	Sim
EfficientNet B0	256 x 256	224 x 224	Sim	Sim
EfficientNet B1	255 x 255	240 x 240	Sim	Sim

modelos baseados em transferência de aprendizado utilizaram seus respectivos pré-processamentos utilizados em seus treinamentos e especificados na documentação do PyTorch.

Já no modelo próprio, foi utilizado apenas o redimensionamento, seguido da normalização dos valores para o intervalo [0-1]. O redimensionamento é utilizado para manter um padrão na entrada, sem a necessidade de adicionar uma camada extra de compatibilidade inicialmente. A normalização para o intervalo [0-1] é feita de modo a reduzir o intervalo inicial de cada canal [0-255], assim permitindo o aprendizado em um número pequeno de épocas, evitando problemas como *grokking* [26].

1) Desenvolvimento do Modelo Próprio: Devido à diversidade de camadas de redes neurais disponíveis na biblioteca PyTorch, o número de possibilidades para a construção de redes neurais é virtualmente infinito. Para limitar o escopo, utilizamos blocos básicos de construção para as camadas convolucionais e de classificação. As camadas convolucionais consistem em: *Conv2d*, *BatchNormalization* e *ReLU*. Já as camadas de classificação foram divididas em dois tipos principais:

- *AdaptativeAvgPool2d*, *Flatten*, *Linear* com saída de dois neurônios para classificação.
- *AdaptativeAvgPool2d*, *Flatten*, *Linear*, *ReLU*, *Dropout*, e *Linear* com saída de dois neurônios para classificação.

Os hiperparâmetros das redes foram definidos como: o número de camadas, a quantidade de filtros por camada e o tamanho do filtro para as camadas convolucionais, e o número de neurônios na camada interna foi ajustado para as camadas de classificação.

O modelo inicial foi estruturado utilizando camadas convolucionais com *MaxPool* entre os blocos e uma camada de classificação ao final. A Tabela VIII apresenta a configuração detalhada do modelo inicial.

Canais de Entrada	Canais de Saída	Tamanho do Kernel	*Stride*
3	64	3	1
64	128	5	1
128	256	5	1

Tabela VIII: Configuração do modelo inicial.

Ajustes e Otimização: Com o objetivo de melhorar os resultados, reduzir o custo computacional do *forward pass* e atender às restrições de memória, foi utilizado o processo de modificação do modelo, treinamento do modelo com *early stop*, avaliação das métricas e geração de hipótese para evoluir o modelo até sua forma final. As estratégias de modificação do modelo incluíram a adição de novas camadas, redução do número de canais nas camadas iniciais e o uso intensivo de técnicas de redução de dimensionalidade, como *MaxPool* e *Conv2d* com *stride* maior que 1, sempre testando separadamente entre os dois tipos de camadas de classificação já mostrados.

Para realizar o treinamento, foi utilizada a regularização *L2* disponível por meio do otimizador *Adam*.

2) Seleção e Desenvolvimento de Modelos de Transferência de Aprendizado: A seleção de modelos de transferência de aprendizado foi realizada com base em dois critérios principais: o menor número de parâmetros e a maior acurácia *acc@5* no conjunto de dados *ImageNet-1K*. Para isso, foram analisados diversos modelos disponíveis na biblioteca PyTorch, e os melhores ranqueados foram submetidos a testes adicionais para determinar o peso total de memória necessário para realizar a classificação. Apenas os modelos que apresentaram custos computacionais de acordo com os requisitos foram mantidos para as etapas subsequentes.

O processo de adaptação dos modelos selecionados envolveu o congelamento completo das camadas convolucionais, preservando os pesos pré-treinados. A camada de classificação foi substituída pelo bloco de classificação descrito anteriormente no modelo próprio: *AdaptativeAvgPool2d*, *Flatten*, *Linear*, *ReLU*, *Dropout*, e *Linear* com saída de dois neurônios para classificação. Para ajustar os parâmetros ao modelo transferido, o número de neurônios na camada interna foi configurado para 8192.

Esse procedimento permitiu integrar as vantagens do aprendizado prévio dos modelos com a flexibilidade do bloco de classificação customizado, resultando em arquiteturas otimizadas para a tarefa proposta, enquanto respeitavam as limitações de memória das GPUs utilizadas.

3) Configuração de Treinamento: O treinamento seguiu os parâmetros descritos na Tabela IX utilizando o *Adam* como otimizador.

Realizou-se um treinamento sobre os modelos por transferência de aprendizado com apenas 10 épocas, os melhores foram selecionados para a segunda etapa do treinamento que consiste em treinar o modelo por um número indeterminado

Modelo	Tamanho do Lote	Early-Stop	Taxa de Aprendizado
MobileNet V3	20	20	0.001
MNASNET	20	20	0.001
ShuffleNet V2	8	10	0.001
RegNet Y 400MF	20	20	0.001
RegNet Y 800MF	16	15	0.001
EfficientNet B0	12	-	0.001
EfficientNet B1	4	-	0.001

Tabela IX: Configuração de Treinamento

de épocas para obter o melhor resultado antes de parar após o número de épocas *'early stop'* consecutivas sem melhora na perda de validação.

A primeira parte do treinamento foi realizada na estação de trabalho um, já o treinamento completo na estação de trabalho dois, indicadas na tabela I.

Os modelos que gastam mais memória para realizar a classificação receberam lotes de treinamento menores, mantendo-se o mais próximo possível do limite de memória da placa de vídeo da estação utilizada. O *'early stop'* foi definido conforme o tamanho do lote, tal que para lotes menores, o *'early stop'* também será menor.

Conversão para ONNX: Após o treinamento, o modelo com melhor desempenho foi convertido de PyTorch para ONNX, utilizando *'torch.onnx.export()'* e validado com *'onnxruntime-web'* para garantir sua compatibilidade.

Validação do Modelo: A validação foi conduzida no conjunto de validação reservado, utilizando métricas como acurácia, precisão média, revocação média, *F1-Score* médio e matriz de confusão, priorizando modelos com menor taxa de falsos negativos, para reduzir o risco de diagnósticos incorretos.

D. Desenvolvimento da Aplicação Web

O desenvolvimento da aplicação iniciou-se com a coleta de histórias de usuário pelos integrantes do projeto, buscando compreender as necessidades e expectativas dos potenciais usuários. A partir dessas histórias, foram elaborados casos de uso que serviram como base para a geração dos requisitos do sistema. Esses requisitos definiram as funcionalidades esperadas e auxiliaram na concepção das telas do sistema e do fluxo de dados da aplicação.

Para a implementação da aplicação, foram utilizadas as seguintes tecnologias:

- *Jimp*: Para manipulação e processamento de imagens.
- *onnxruntime-web*: Para carregamento e execução do modelo treinado no navegador.
- *Next.js* e *React.js*: Para construção da interface do usuário e gerenciamento de rotas.
- *Vercel*: Para hospedagem e disponibilização da aplicação de forma rápida e escalável.
- *Auth0*: Para realizar a autenticação do usuário.

IV. Resultados e Discussões

Nesta seção, são apresentados os principais resultados obtidos ao longo do desenvolvimento do projeto e discutidos em relação aos objetivos estabelecidos. A análise é dividida em quatro partes principais: os resultados dos modelos baseados em transferência de aprendizado (IV-A), os resultados dos modelos próprios desenvolvidos para a detecção de sarna de macieira (IV-B), e os artefatos gerados durante o desenvolvimento da aplicação web (IV-C).

Inicialmente, os resultados de transferência de aprendizado são detalhados na subseção IV-A, começando com o desempenho nas 10 épocas iniciais, utilizadas para selecionar os modelos mais promissores, e culminando nos resultados finais após o treinamento completo. Em seguida, são apresentados os resultados do modelo próprio na subseção IV-B, destacando as métricas de desempenho e comparando-os aos modelos de transferência de aprendizado.

Além disso, os artefatos relacionados ao desenvolvimento da aplicação web, como histórias de usuário, casos de uso e fluxos de dados, bem como a arquitetura geral do sistema, são apresentados na subseção IV-C.

A. Resultados Transferência de Aprendizado

Nesta subseção, são apresentados os experimentos conduzidos com diferentes modelos baseados em transferência de aprendizado, detalhando o desempenho de cada arquitetura ao longo do treinamento inicial.

1) Custo computacional: A tabela X apresenta o custo computacional de cada modelo, onde o custo do *Forward Pass* é relacionado ao armazenamento do estado de cada camada da rede neural, o peso dos parâmetros é o custo de memória necessário apenas para inicializar os parâmetros em memória e o custo estimado é a soma do *Forward Pass* com o Peso dos Parâmetros.

Modelo	*Forward Pass (MB)*	Peso dos Parâmetros (MB)	Custo estimado (MB)
MobileNet V3	116,79	41,43	158,84
MNASNet	184,62	59,17	244,41
ShuffleNet V2	220,00	**84,48**	307,82
RegNet Y 400MF	98,75	28,73	128,05
RegNet Y 800MF	138,14	46,14	184,85
EfficientNet B0	173,83	55,38	229,79
EfficientNet B1	**285,80**	64,94	**351,40**

Tabela X: Custo computacional de cada modelo

Em uma analise inicial sobre os modelos testados com transferência de aprendizado, os melhores modelos em relação a custo computacional, medido em o quanto de memória é necessário para realizar a previsão, são RegNet Y 400MF e MobileNet V3 com 128,05MB e 158,84 MB de custo

respectivamente. Os piores modelos são o ShuffleNet V2 e EfficientNet B1 com 351,40MB e 307,82MB respectivamente.

2) Resultados das 10 Épocas de Treinamento: Durante o treinamento de 10 épocas, foram avaliados múltiplos modelos com arquiteturas amplamente utilizadas na literatura. A análise foi feita considerando as curvas de perda e acurácia em conjunto com as métricas obtidas ao final de cada treinamento.

Modelo	Acurácia (%)	F1-Score (%)	Precisão (%)	Revocação (%)
MobileNet V3	**88,64**	**88,62**	**88,70**	**88,60**
MNASNet	78,73	78,32	83,68	78,66
ShuffleNet V2	86,85	86,84	87,50	86,81
RegNet Y 400MF	87,66	87,59	87,63	87,62
RegNet Y 800MF	87,01	87,02	87,49	87,13
EfficientNet B0	81,01	80,95	82,46	80,94
EfficientNet B1	75,57	75,55	77,20	75,57

Tabela XI: Métricas de validação ao final das 10 primeiras épocas de treinamento.

As figuras 7 e 8 apresentam as curvas de perda e acurácia de todos os modelos ao longo das 10 épocas de treinamento.

As métricas de performance de cada modelo após as 10 épocas de treinamento indicam que os melhores modelos são MobileNet V3, ShuffleNet V2, RegNet Y 400MF e RegNet Y 800MF, atingindo métricas de acurácia, F1-Score, precisão e revocação acima de 86%, enquanto os modelos MNASNet, EfficientNet B0 e EfficientNet B1 não obtiveram performance semelhante, terminando próximos de 80%, e podendo vir a ser desconsiderados para o treinamento completo, caso a análise das curvas de perda e acurácia não apresentem fatos que corroboram para o contrário.

Os modelos MobileNet V3, MNASNet, ShuffleNet V2, RegNet Y 400MF e RegNet Y 800MF apresentaram curvas de perda semelhantes, seguindo o padrão de redução da perda de treinamento, com a de validação abaixo ou próxima da de treinamento em sua maior parte, também tendendo a uma redução, indicando o aprendizado do modelo e generalização para casos variados dentro destas 10 épocas iniciais. Já os dois modelos EfficientNet B0 e EfficientNet B1 apresentaram reduções na curva de perda de treinamento, mas a curva de validação não seguiu o mesmo padrão, indicando que não houve generalização.

Os modelos MobileNet V3, MNASNet, ShuffleNet V2, RegNet Y 400MF e RegNet Y 800MF apresentaram curvas de acurácia de treinamento e validação crescentes, corroborando com o resultado da curva de perda. Os dois modelos EfficientNet B0 e B1 apresentaram curvas de acurácia de treinamento e validação que corroboram a análise da curva de perdas.

Como observado, o MNASNet apresentou um resultado nas métricas de validação na última época desta etapa do treinamento abaixo dos melhores modelos, entretanto foi considerado um resultado não condizente com a performance geral do modelo quando considerando as curvas de perda e acurácia durante todo o treinamento, portanto este resultado na última época foi considerado anormal e desconsiderado na análise final sobre quais modelos passarão para a próxima etapa do treinamento.

Deste modo, os modelos MobileNet V3, ShuffleNet V2, MNASNet, RegNet Y 400MF e RegNet Y 800MF foram considerados os melhores nos quesitos de capacidade de generalização e aprendizado, sendo selecionados para o treinamento completo. Os modelos EfficientNet B0 e B1 apresentaram dificuldades de aprendizado e foram desconsiderados para o treinamento completo.

3) Resultados do Treinamento Completo: Nesta seção são apresentados os resultados detalhados dos modelos selecionados após o treinamento completo, incluindo o desempenho em termos de métricas de avaliação, curvas de aprendizado e análise qualitativa dos resultados. O objetivo foi verificar a capacidade desses modelos de generalizar para o problema de detecção de sarna de macieira após um treinamento mais extenso.

As figuras 9 e 10 e Apêndice A, figura 12 mostram as curvas de perda e acurácia dos modelos durante o treinamento completo, destacando a evolução das métricas ao longo das épocas de treinamento.

A Tabela XII apresenta as métricas obtidas da melhor época do treinamento completo, incluindo acurácia, precisão, revocação e *F1-score* para cada modelo.

Modelo	Acurácia (%)	F1-Score (%)	Precisão (%)	Revocação (%)
MobileNet V3	92,01	92,02	92,02	92,02
MNASNet	89,77	89,75	89,83	89,74
ShuffleNet V2	**92,75**	**92,83**	**92,83**	**92,83**
RegNet Y 400MF	90,74	90,67	90,85	90,72
RegNet Y 800MF	91,24	91,31	91,67	91,37

Tabela XII: Métricas finais dos modelos no conjunto de validação.

Por análise das métricas da melhor época de cada modelo, o ShuffleNet V2 é o que apresentou melhor performance, seguido dos modelos MobileNet V3, RegNet Y 800MF, RegNet Y 400MF e MNASNet, respectivamente. A diferença de desempenho entre o ShuffleNet V2 e o MobileNet V3, em média de aproximadamente 0,75%, possibilita a seleção de um desses para ser utilizado na aplicação web, podendo ser selecionado aquele que apresenta melhor performance.

O modo de treinamento com *early stop* garante que encontraremos o ponto de começo do sobreajuste, visto que um dos

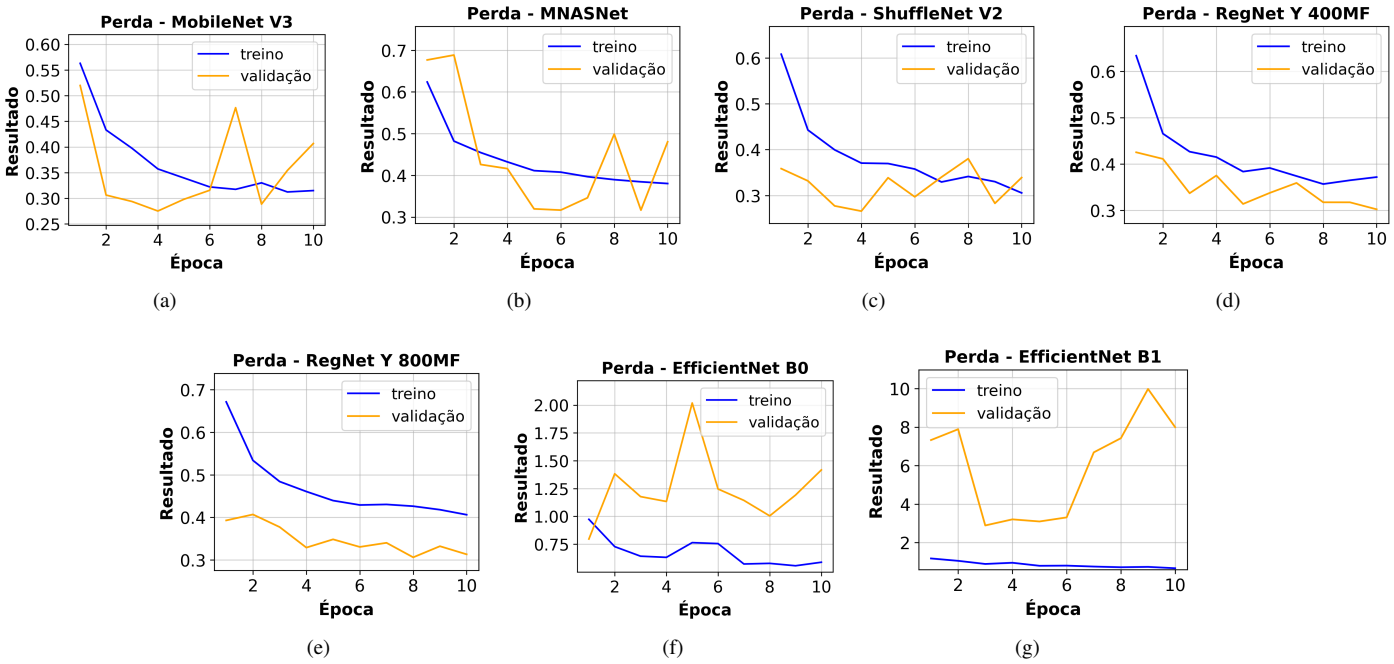


Figura 7: Curvas de perda durante o treinamento inicial para os modelos (a) MobileNet V3 (b) MNASNET (c) ShuffleNet V2 (d) RegNet Y 400MF (e) RegNet Y 800MF (f) EfficientNet B0 e (g) EfficientNet B1.

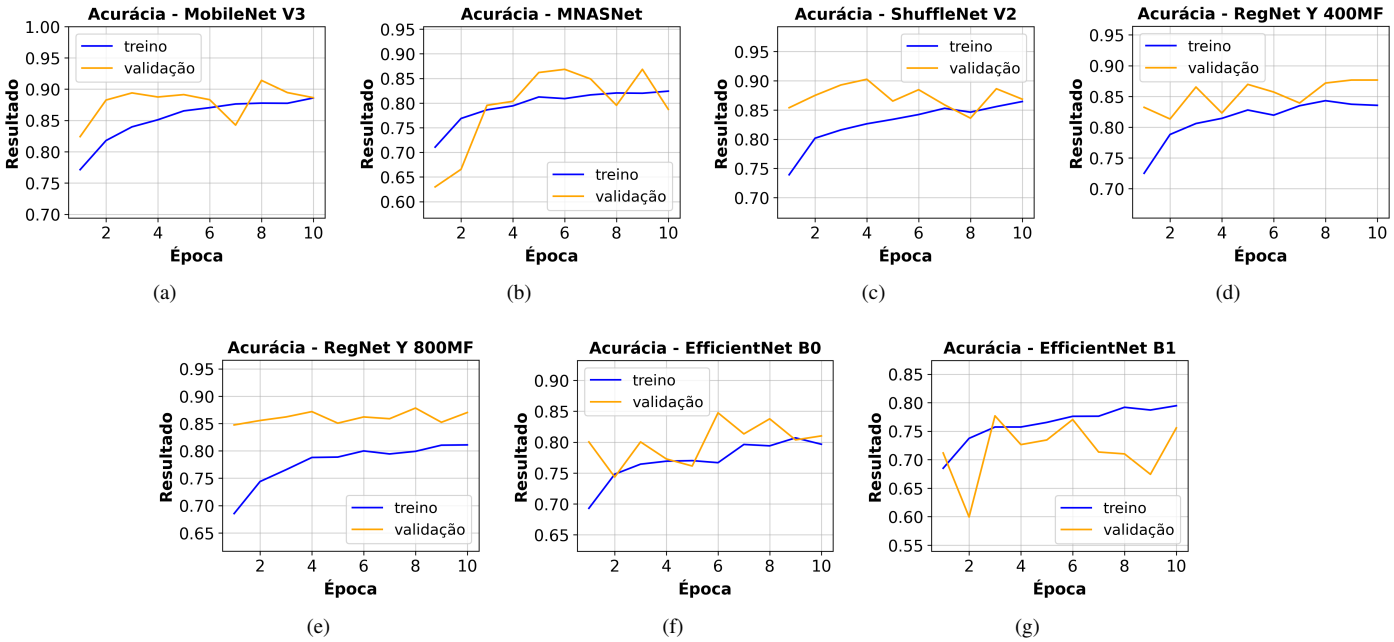


Figura 8: Curvas de acurácia durante o treinamento inicial para os modelos (a) MobileNet V3, (b) MNASNET (c) ShuffleNet V2 (d) RegNet Y 400MF (e) RegNet Y 800MF (f) EfficientNet B0 e (g) EfficientNet B1.

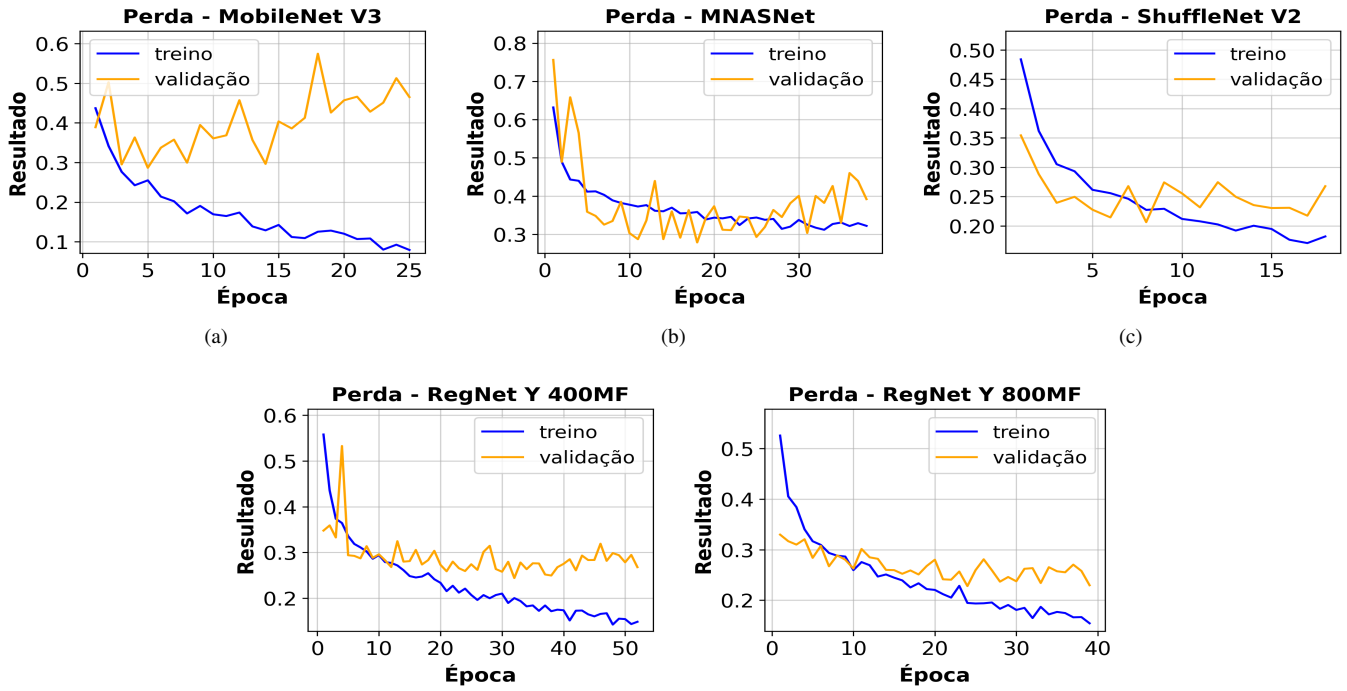


Figura 9: Curvas de perda durante o treinamento completo para os modelos (a) MobileNet V3 (b) MNASNET (c) ShuffleNet V2 (d) RegNet Y 400MF e (e) RegNet Y 800MF.

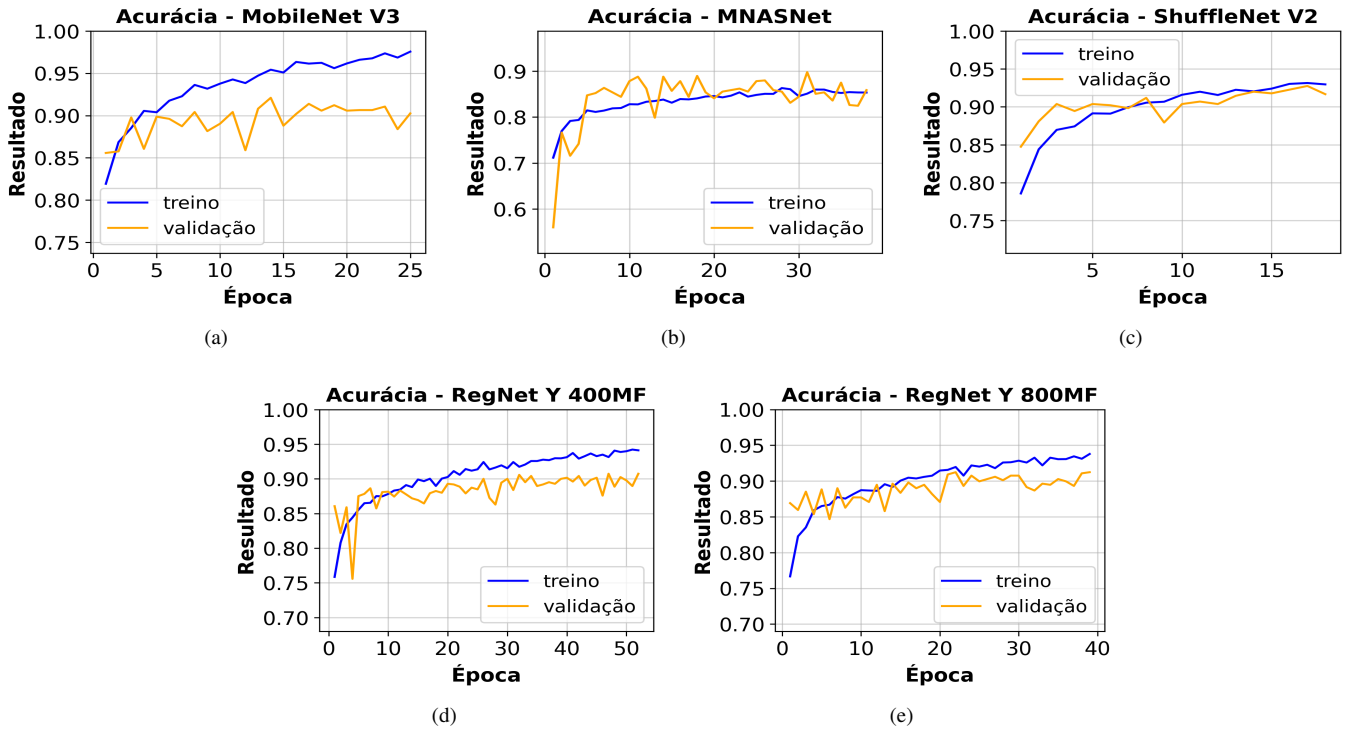


Figura 10: Curvas de acurácia durante o treinamento completo para os modelos (a) MobileNet V3 (b) MNASNET (c) ShuffleNet V2 (d) RegNet Y 400MF (e) RegNet Y 800MF.

sintomas do sobreajuste é a redução da perda de treinamento com um aumento da perda de validação. Assim, pode-se afirmar que não haverá melhora significativa nas métricas de validação após as épocas em que o modelo foi treinado.

As curvas de acurácia corroboram com a análise feita sobre a curva de perda, onde houve melhora significativa nas métricas de cada modelo até o ponto mínimo de perda. Após este ponto, houve uma redução do incremento encontrado na acurácia de validação, com aumento na acurácia de treinamento, indicando que o modelo está aprendendo as características do conjunto de dados de treinamento e não generalizando.

A tendência vista na curva de acurácia é repetida na curva do *F1 Score* apresentada no Apêndice A, figura 12. As métricas obtidas para precisão e revocação são próximas, indicando que os modelos acertam e erram cada classe com a mesma proporção, resultado que é refletido na métrica *F1 Score* de cada modelo.

Por definição da *F1 Score*, no caso de a precisão e a revocação estarem próximas, indicam que os modelos são equilibrados, não havendo diferença na utilização da acurácia ou *F1 Score* para método de comparação.

Devido aos requisitos computacionais e performance semelhante nas métricas, o modelo MobileNet V3 foi escolhido em favor do ShuffleNet V2. No decorrer desta seção, será comentado apenas sobre o MobileNet V3.

a) Matriz de Confusão: A matriz de confusão ilustra a distribuição das previsões corretas e incorretas, fornecendo uma visão mais granular sobre o comportamento do modelo em relação às classes do problema.

Classificação Real	**Com Sarna**	**Sem Sarna**
Com Sarna (Predito)	312 (VP)	24 (FP)
Sem Sarna (Predito)	25 (FN)	253 (VN)

Tabela XIII: Matriz de Confusão do MobileNet V3

A matriz de confusão mostra o número de amostras classificadas corretamente (valores na diagonal principal) e os erros de classificação (valores fora da diagonal). Observa-se que a maior parte das previsões foram corretas (565), com um número limitado de erros (49), em que a maior parte dos erros são de previsões onde as folhas estão com Sarna e que foram preditas Sem Sarna.

b) Erros de Classificação: Além da análise quantitativa fornecida pela matriz de confusão e as outras métricas, é importante visualizar os erros cometidos pelo melhor modelo treinado de transferência de aprendizado. Esses erros podem revelar padrões, identificar limitações do modelo e sugerir possíveis aprimoramentos. No Apêndice A, figura 13 foram apresentados exemplos de imagens do conjunto de teste classificadas incorretamente pelo modelo, com suas respectivas classes reais e predições:

Esses erros indicam que o modelo pode enfrentar desafios em situações como:

- Diferenças sutis entre classes (por exemplo, folhas saudáveis em estágios iniciais de doença).
- Casos onde o contexto visual não fornece informações claras para a classificação.

Portanto, observou-se que os modelos foram capazes de lidar bem com as imagens de teste, mesmo com variações no conjunto de dados, indicando uma boa capacidade de generalização. Além disso, a eficiência computacional dos modelos será discutida, considerando o uso pretendido na aplicação web.

B. Resultados Modelo Próprio

Nesta subseção, são apresentados os resultados obtidos com o modelo próprio desenvolvido ao longo do projeto. Serão detalhados o desempenho geral durante o treinamento e os testes, incluindo métricas como acurácia, perda, precisão e recall. Além disso, destacaremos o modelo que apresentou os melhores resultados e discutiremos seu desempenho em comparação com os modelos baseados em transferência de aprendizado. O modelo que alcançou o melhor desempenho foi definido pela arquitetura da Tabela XIV, obtendo um equilíbrio entre acurácia e custo computacional XV.

Canais de Entrada	**Canais de Saída**	**Tamanho do Kernel**	***Stride***
3	16	3	1
16	16	3	3
16	64	3	2
64	256	3	2
256	512	3	2
512	768	3	1
768	768	3	1
768	1280	3	1
1280	1280	3	1
1280	2	-	-

Tabela XIV: Configuração do modelo final.

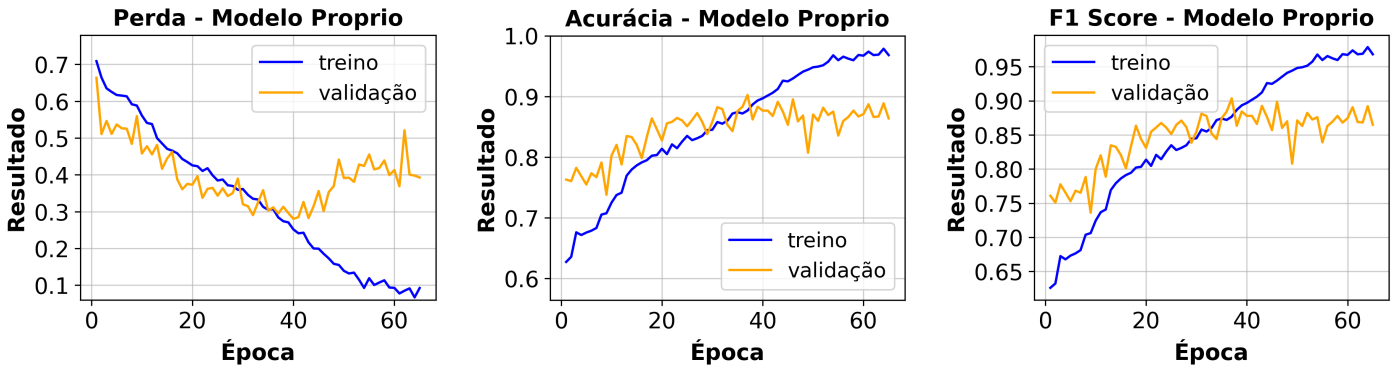
Forward Pass (MB)	32,78
Peso dos Parâmetros (MB)	128,93
Custo estimado (MB)	162,46

Tabela XV: Custo computacional do modelo final

O custo computacional demonstra que o modelo próprio é otimizado para operação em *batch* visto que a maior parte do seu custo estimado está no peso dos parâmetros, enquanto o *Forward Pass* é pequeno em comparação com o peso, resultando em um pequeno aumento do custo computacional ao classificar múltiplas imagens ao mesmo tempo.

a) Gráficos de Acurácia e Perda Durante o Treinamento: A Figura 11 apresenta os gráficos da perda, acurácia e *F1 Score* ao longo das épocas de treinamento para o modelo próprio selecionado.

A Tabela XVI resume as principais métricas obtidas na melhor época do treinamento, escolhida com base no desempenho no conjunto de validação.



(a) (b) (c)

Figura 11: Gráficos de perda (a), acurácia (b) e *F1 Score* (c) durante o treinamento do modelo próprio.

Métrica	Valor (%)
Acurácia	89,57
F1-Score	89,58
Precisão	89,61
Revocação	89,58

Tabela XVI: Métricas finais do modelo próprio.

b) Matriz de Confusão: A matriz de confusão referente ao melhor modelo próprio no conjunto de teste é apresentada na Tabela XVII.

Classificação Real	Com Sarna	Sem Sarna
Com Sarna (Predito)	302 (VP)	29 (FP)
Sem Sarna (Predito)	35 (FN)	248 (VN)

Tabela XVII: Matriz de Confusão do Modelo Próprio

Os resultados obtidos pelo modelo próprio demonstram que, apesar de apresentar desempenho levemente inferior em relação ao melhor modelo de *transferência de aprendizado* MobileNet V3, o modelo próprio conseguiu atender às principais demandas do projeto. Sua arquitetura personalizada permitiu uma solução eficiente e otimizada para uso offline, conciliando precisão e desempenho computacional.

O modelo apresenta equilíbrio em relação aos erros, onde pela matriz de confusão XVII observa-se 29 Falsos Positivos e 35 Falsos Negativos. Equilíbrio também observado no *F1 Score*.

Os resultados confirmam que o modelo próprio é uma alternativa viável, entretanto, devido à diferença de desempenho nas métricas e por a diferença de custo computacional para uma imagem não ser grande, o modelo MobileNet V3 foi selecionado como modelo final para realização das classificações na aplicação web devido a restrições de memória e que não haverá múltiplas imagens sendo processadas ao mesmo tempo.

C. Resultados da Aplicação Web

Esta subseção apresenta os resultados obtidos no desenvolvimento da aplicação web para utilização do modelo de detecção de sarna de macieira. Inicialmente, são detalhadas as histórias do usuário, os casos de uso e o fluxo de dados projetado para a aplicação. Em seguida, é exibido o resultado final da aplicação, com capturas de tela das principais funcionalidades: tela de login, tela principal, histórico de classificações e exibição de detalhes. Por fim, é discutida a implementação técnica, destacando os desafios encontrados e as soluções aplicadas.

1) Histórias do Usuário: As histórias do usuário foram coletadas para guiar o desenvolvimento da aplicação e garantir que atendesse às necessidades dos usuários finais. As principais histórias estão listadas abaixo:

- Como usuário, quero realizar login de maneira simples para acessar a aplicação.
- Como usuário, quero realizar o *upload* de imagens para análise de sarna de macieira.
- Como usuário, quero visualizar o histórico de análises realizadas.
- Como usuário, quero acessar detalhes das análises anteriores para acompanhar os resultados.

2) Casos de Uso: Os casos de uso derivados das histórias do usuário foram definidos para estruturar o fluxo de interações com a aplicação. A tabela XVIII apresenta os principais casos de uso identificados.

ID	Descrição
CU01	Realizar login na aplicação.
CU02	Fazer *upload* de imagens para análise.
CU03	Consultar o histórico de análises realizadas.
CU04	Visualizar detalhes de uma análise específica.

Tabela XVIII: Casos de Uso da Aplicação Web.

3) Requisitos do Sistema: Para o desenvolvimento e funcionamento adequado da aplicação web, os seguintes requisitos de sistema foram definidos:

- **Requisitos Funcionais:**
  - RF01: O sistema deve permitir que o usuário realize login em até dois cliques.
  - RF02: O sistema deve possibilitar o *upload* de imagens para análise de sarna de macieira em até dois cliques.
  - RF03: O sistema deve permitir o usuário um visualizar o histórico de análises realizadas na seção em até um clique.
  - RF04: O sistema deve exibir informações detalhadas sobre uma análise específica em até um clique.
  - RF05: O sistema deve permitir o usuário realizar
- **Requisitos Não Funcionais:**
  - RNF01: O sistema deve ser responsivo, adaptando-se a diferentes dispositivos e tamanhos de tela.
  - RNF02: O processamento do modelo de machine learning deve ser realizado no cliente para minimizar a dependência do servidor.
  - RNF03: A aplicação deve ser hospedada em um ambiente que ofereça escalabilidade e facilidade de deploy.
  - RNF04: O sistema deve garantir um tempo de resposta inferior a 2 segundos para exibir resultados após o início da analise de uma imagem.
  - RNF05: O histórico e os detalhes das análises devem ser acessíveis mesmo em ambientes com conectividade limitada.
  - RNF06: O sistema deve tentar manter o modelo salvo no dispositivo do cliente.
  - RNF07: O sistema deve ser possível de realizar analises mesmo em ambientes com conectividade limitada.

4) Fluxo de Dados: O fluxo de dados da aplicação descreve as interações entre os diferentes componentes do sistema, desde o *upload* de imagens até a exibição dos resultados. No Apêndice B, Figura 14 apresenta o diagrama do fluxo de dados.

5) Resultado Final da Aplicação: A aplicação foi desenvolvida para ser simples, eficiente e acessível. As principais telas estão descritas abaixo:

- **Tela de Login:** interface simples e segura para autenticação do usuário (Apêndice B, Figura 15).
- **Tela Principal:** permite o *upload* de imagens e exibe os resultados recentes (Apêndice B, Figuras 16 e 17).
- **Histórico:** apresenta uma lista das análises anteriores (Apêndice B, Figura 18).
- **Exibição de Detalhes:** oferece informações detalhadas sobre uma análise específica (Apêndice B, Figura 19).

6) Discussão Técnica: A implementação técnica da aplicação cumpriu os requisitos apresentados, em especial, os requisitos de análise em ambientes de conectividade restrita. A implementação da análise em situações de conexão limitada há limitações, sendo necessário ter visitado o site, realizado a autenticação para obtenção do modelo, além de manter a página aberta enquanto não tiver conexão com a internet. Caso haja suporte pelo navegador, este será armazenado juntamente ao *IndexedDB*, e reutilizado nas futuras seções, assim evitando requisições desnecessárias para reobter o modelo.

O processo de classificação das imagens descrito em sua maioria no Apêndice B, figura 14 impôs limitações na implementação, sendo necessário um cuidado extra para não atingir os limites de memória, em que é necessário realizar o processamento de cada imagem uma por vez para reduzir a possibilidade de erros por falta de memória, seja para a realização do processamento das imagens e o subsequente armazenamento dos tensores ou a classificação pelo modelo.

A escolha da biblioteca *onnxruntime-web* foi essencial para possibilitar a utilização do modelo em modo de conexão limitada, assim como a utilização da biblioteca *Jimp*, devido à eficiência computacional e tamanho reduzido, permitindo replicar o processo de processamento da imagem e classificação utilizado no treinamento e validação com *PyTorch* em um ambiente com recursos limitados do cliente.

O desenvolvimento da aplicação com *Next.js*, *React.js* e hospedando na *Vercel* demonstrou-se correta, ao possibilitar um rápido desenvolvimento da página principal, já possuir bibliotecas para fácil integração com o serviço de autenticação *Auth0* e por sua integração com o ambiente da *Vercel* permitindo hospedar o modelo treinado no formato *ONNX* com alta disponibilidade banda em sua versão grátis.

V. Conclusão

Este trabalho apresentou uma solução acessível e eficiente para a detecção da sarna da macieira, uma doença de grande impacto na produção agrícola, especialmente para pequenos agricultores em ambientes com recursos limitados. A abordagem integrou CNNs leves com uma aplicação web capaz de operar offline, atendendo às necessidades de diagnósticos rápidos e precisos em regiões remotas.

Foram explorados diversos modelos, incluindo MobileNet V3, ShuffleNet V2 e RegNet Y, sendo o MobileNet V3 escolhido devido ao seu equilíbrio entre precisão (92,01%) e eficiência computacional. Além disso, a aplicação web desenvolvida com *Next.js* e *React.js*, utilizando *ONNXRuntime-web* e *Jimp*, mostrou-se uma solução robusta para a execução local das inferências, dispensando a necessidade de conexão constante à internet.

Os resultados alcançados demonstram a viabilidade de combinar aprendizado profundo e tecnologias acessíveis para democratizar o diagnóstico agrícola. Contudo, há várias possibilidades de aprimoramento e expansão deste trabalho em estudos futuros. Primeiramente, explorar a generalização do modelo para outras doenças foliares além da sarna da macieira, incluindo doenças como manchas bacterianas, ferrugem e oídio. Isso pode ser viabilizado através da ampliação e diversificação do conjunto de dados, bem como da aplicação de técnicas avançadas de aprendizado multitarefa, permitindo

que o modelo reconheça múltiplas condições de saúde das folhas em um único framework.

Outra linha promissora é o desenvolvimento de um sistema de diagnóstico mais detalhado, capaz de não apenas identificar a doença presente na folha, mas também fornecer recomendações específicas para manejo e tratamento. Isso incluiria a integração de sistemas de recomendação baseados em IA, capazes de sugerir práticas agrícolas apropriadas para cada tipo de problema detectado, promovendo intervenções mais rápidas e eficazes.

Além disso, a possibilidade de embarcar o modelo em dispositivos de hardware leve, como microcontroladores e sistemas embarcados com suporte a inferência de aprendizado profundo (ex.: NVIDIA Jetson Nano ou Google Coral [21]), abre novas oportunidades para diagnósticos em tempo real diretamente no campo. Essa abordagem eliminaria a necessidade de utilizar a aplicação web, oferecendo uma solução ainda mais portátil e de baixa latência, ideal para agricultores em áreas remotas e com conectividade limitada. Esse sistema poderia ser integrado a dispositivos de captura de imagem, como drones ou câmeras portáteis, para monitoramento contínuo e automatizado.

Por fim, explorar técnicas de compressão de modelos, como poda de redes e quantização, pode ajudar a reduzir ainda mais os requisitos de memória e energia, tornando o sistema embarcado mais eficiente e acessível.

Este trabalho destaca o potencial da inteligência artificial em promover soluções práticas e de baixo custo para desafios no setor agrícola, contribuindo para a sustentabilidade e produtividade no campo. As propostas de trabalhos futuros visam expandir essa contribuição, levando tecnologias acessíveis a um número ainda maior de produtores rurais e adaptando o sistema para atender a uma variedade mais ampla de aplicações agrícolas.

AGRADECIMENTOS

Aos nossos colegas de classe, registramos nossa profunda gratidão pela parceria, apoio e colaboração ao longo desses anos. Cada um de vocês foi fundamental para tornar essa jornada acadêmica mais rica e significativa.

Aos professores que, com dedicação, paciência e compromisso, compartilharam seus conhecimentos e orientações nossos sinceros agradecimentos. Suas contribuições foram essenciais para nosso crescimento pessoal e profissional.

Estendemos também nossa gratidão a todos os profissionais da Universidade Federal de Goiás, cujas diversas áreas e esforços tornam possível a formação de tantos alunos, incluindo a nossa. Obrigado por sua dedicação e trabalho contínuo.

Ao Professor Cássio Dener Noronha Vinhal, expressamos nosso mais sincero agradecimento por nos proporcionar a oportunidade de realizar este estudo.

REFERÊNCIAS

[1] "Sarna da Maçã," Agrolink. Disponível em: https://www.agrolink.com.br/problemas/sarna-da-maca-1692.html. Acesso em: 20 dez. 2024.

[2] "Com safra de mais de 1 milhão de toneladas, Brasil é um dos 10 maiores produtores mundiais de maçã," Globo Rural, 2021. Disponível em: https://globorural.globo.com/Noticias/Agricultura/Hortifruti/noticia/2021/12/com-safra-de-mais-de-1-milhao-de-toneladas-brasil-e-um-dos-10-maiores-produtores-mundiais-de-maca.html. Acesso em: 20 dez. 2024.

[3] D. Rumelhart, G. Hinton, e R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986. Disponível em: https://doi.org/10.1038/323533a0.

[4] A. de Carvalho, "Redes Neurais Artificiais," Disponível em: https://sites.icmc.usp.br/andre/research/neural/. Acesso em: 03 dez. 2024.

[5] Y. LeCun, Y. Bengio, e G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015. Disponível em: https://doi.org/10.1038/nature14539. Acesso em: 03 dez. 2024.

[6] A. Krizhevsky, I. Sutskever, e G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, Toronto:Universidade de Toronto, 2012.

[7] Microsoft, "onnxruntime-web," 2019. Disponível em: https://github.com/microsoft/onnxruntime-web. Acesso em: 10 dez. 2024.

[8] S. Ioffe e C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167*, 2015. doi: 10.48550/arXiv.1502.03167.

[9] Y. LeCun, L. Bottou, Y. Bengio, e P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998. doi: 10.1109/5.726791.

[10] PyTorch, "torch.nn.Conv2d," Disponível em: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html. Acesso em: 10 dez. 2024.

[11] PyTorch, "torch.nn.BatchNorm2d," Disponível em: https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html. Acesso em: 10 dez. 2024.

[12] PyTorch, "torch.nn.ReLU," Disponível em: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html. Acesso em: 10 dez. 2024.

[13] PyTorch, "torch.nn.AdaptiveAvgPool2d," Disponível em: https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html. Acesso em: 10 dez. 2024.

[14] PyTorch, "torch.flatten," Disponível em: https://pytorch.org/docs/stable/generated/torch.flatten.html. Acesso em: 10 dez. 2024.

[15] PyTorch, "torch.nn.Linear," Disponível em: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html. Acesso em: 10 dez. 2024.

[16] A. Howard et al., "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South), 2019, pp. 1314-1324, doi: 10.1109/ICCV.2019.00140.

[17] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, e Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," *CVPR 2019*, arXiv:1807.11626 [cs.CV], 2019. doi: 10.48550/arXiv.1807.11626.

[18] N. Ma, X. Zhang, H. Zheng, e J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[19] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, e P. Dollár, "Designing Network Design Spaces," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10425-10433. doi: 10.1109/CVPR42600.2020.01044.

[20] M. Tan e Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," May 2019. doi: 10.48550/arXiv.1905.11946.

[21] PyTorch, "PyTorch documentation," 2023. Disponível em: https://pytorch.org/. Acesso em: 29 nov. 2024.

[22] "ATLDSD," Disponível em: https://www.kaggle.com/datasets/gabrielwang01/atldsd-class. Acesso em: 29 nov. 2024.

[23] "AppleScabLDS," Disponível em: https://www.kaggle.com/datasets/projectlzp201910094/applescablds. Acesso em: 29 nov. 2024.

[24] "eScab," Disponível em: https://www.kaggle.com/datasets/projectlzp201910094/eapplescab. Acesso em: 29 nov. 2024.

[25] N. V. Chawla, K. W. Bowyer, L. O. Hall, e W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002. doi: 10.1613/jair.953.

[26] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra, "Grokking: Generalization beyond overfitting on small algorithmic

datasets," CoRR, vol. abs/2201.02177, 2022. [Online]. Available: https://arxiv.org/abs/2201.02177.

APÊNDICES

APÊNDICE A
RESULTADOS EXTRAS DOS TREINAMENTO DOS MODELOS

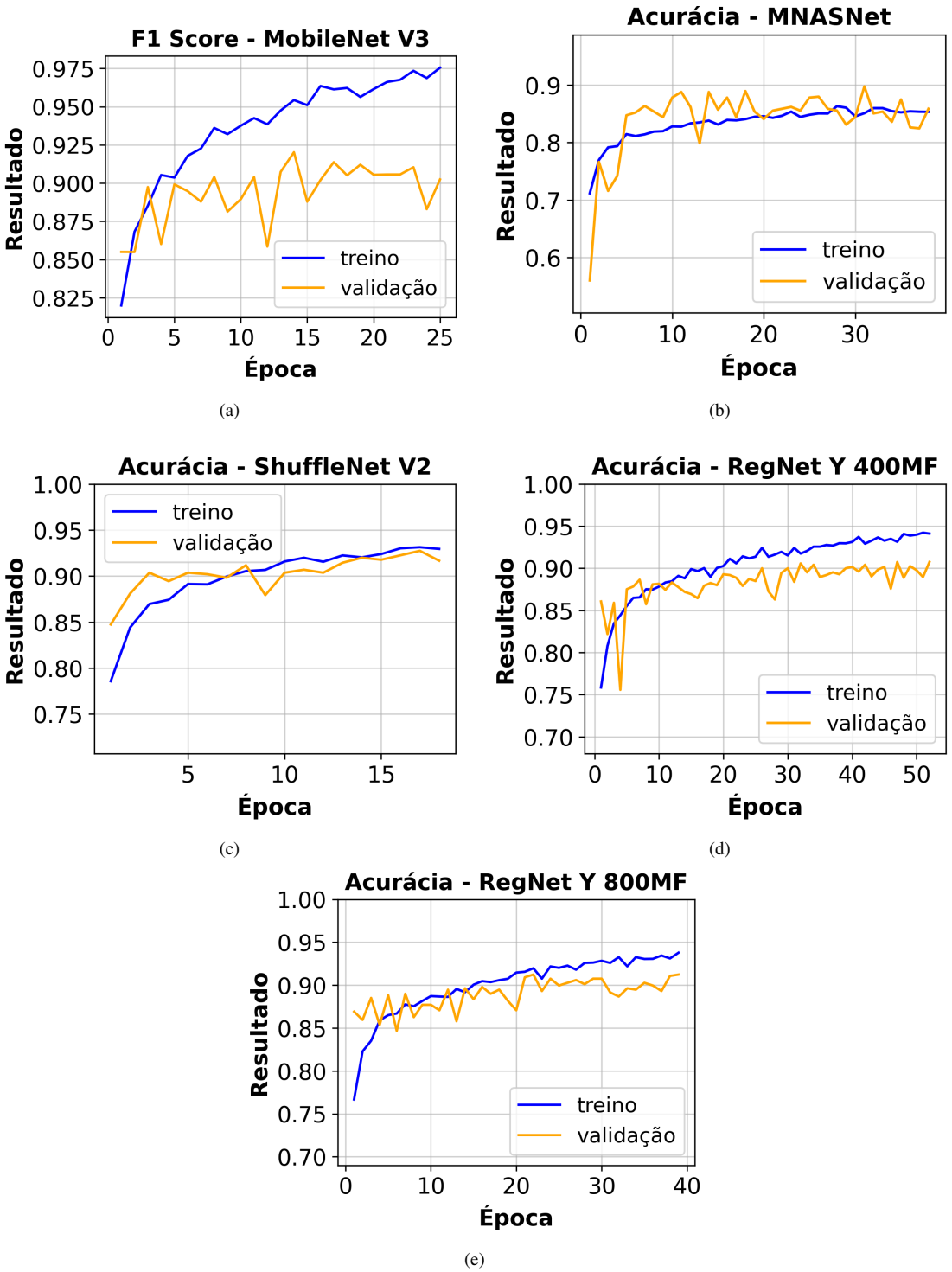


Figura 12: Curvas de acurácia durante o treinamento completo para os modelos (a) MobileNet V3 (b) MNASNET (c) ShuffleNet V2 (d) RegNet Y 400MF (e) RegNet Y 800MF.

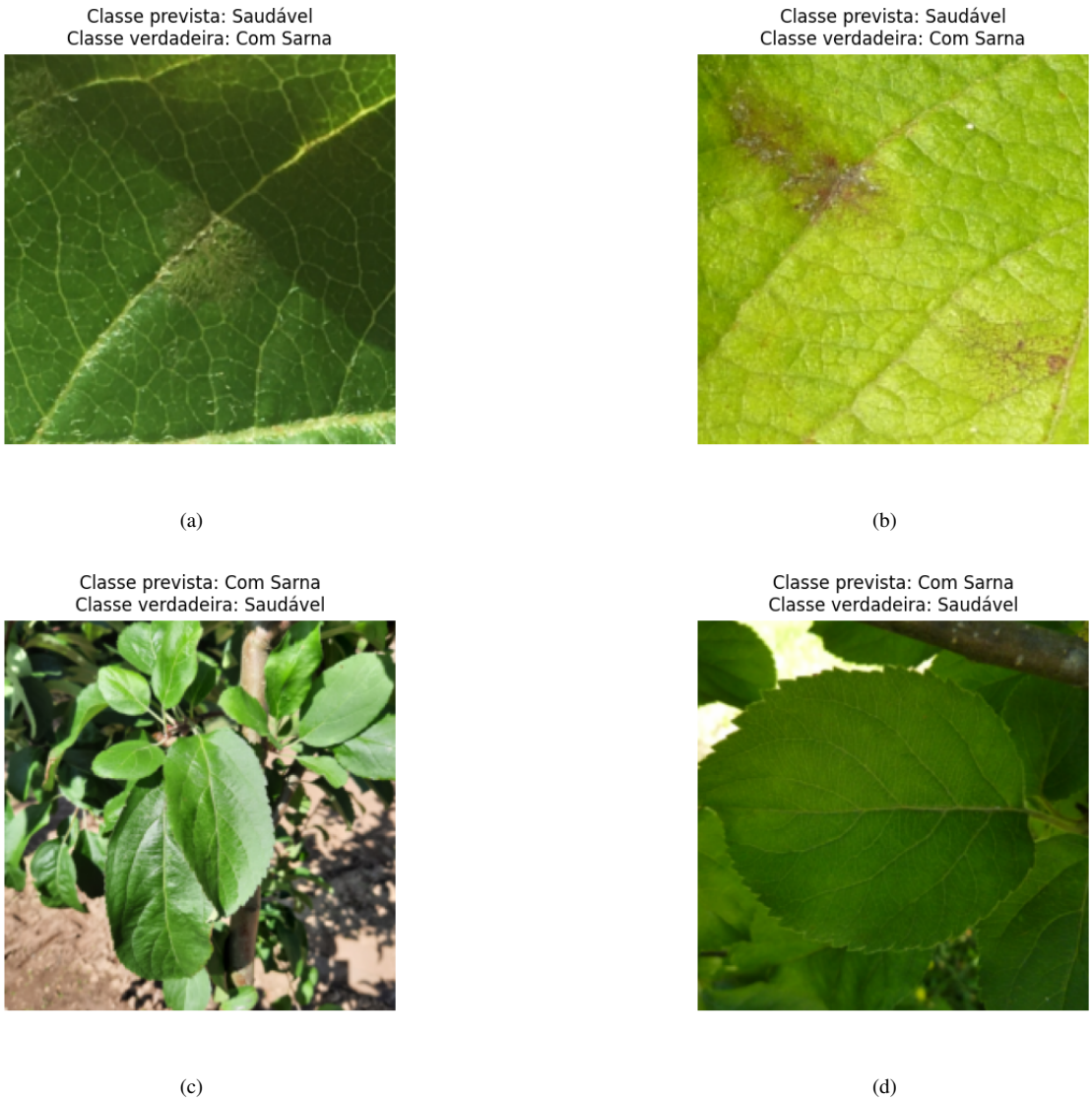


Figura 13: Exemplos de erros do modelo MobileNet V3. Cada imagem apresenta a classe real (abaixo) e a predição incorreta (acima).

APÊNDICE B
ARTEFATOS DA APLICAÇÃO

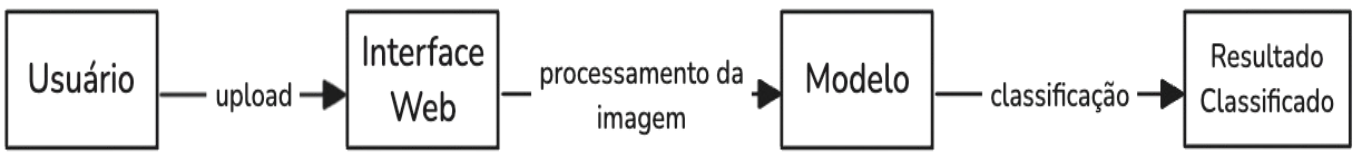


Figura 14: Diagrama do fluxo de dados da aplicação web.

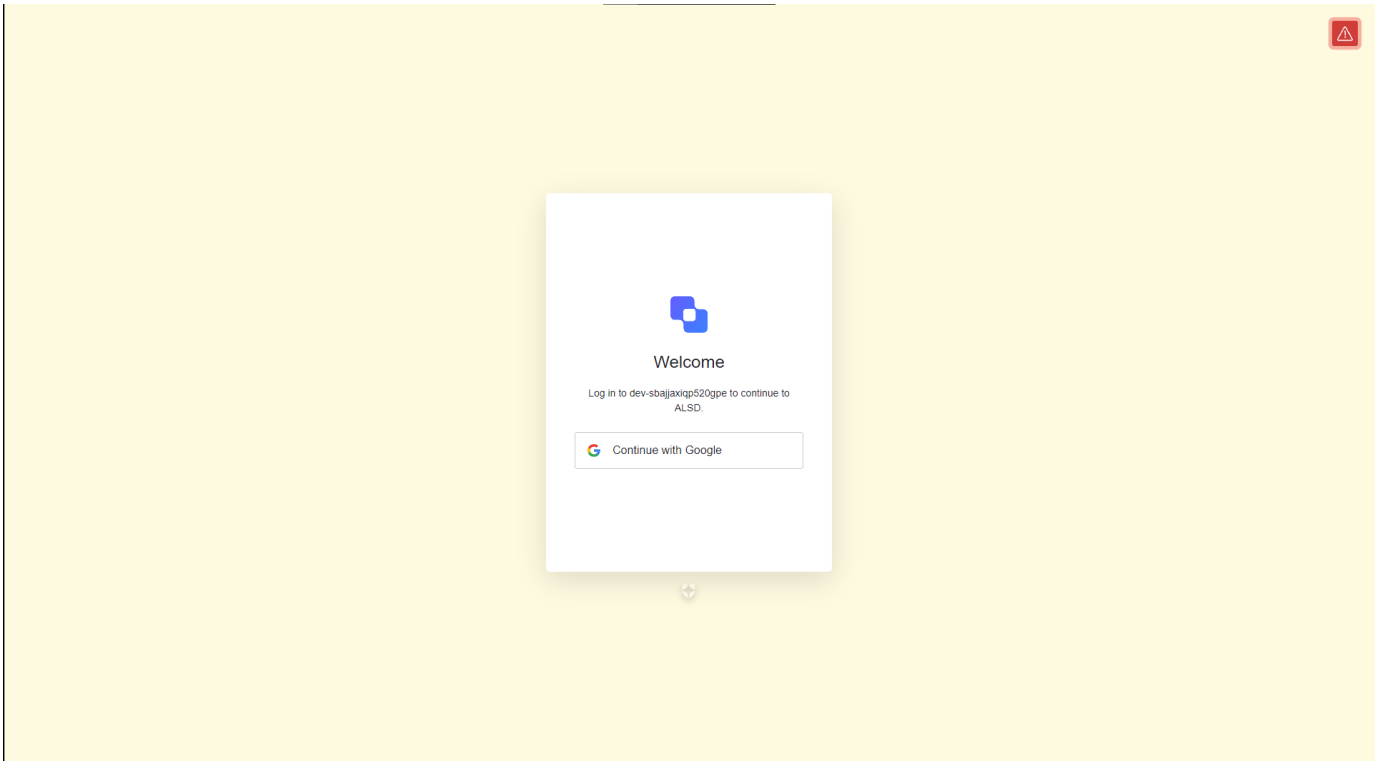


Figura 15: Tela de Login da aplicação.

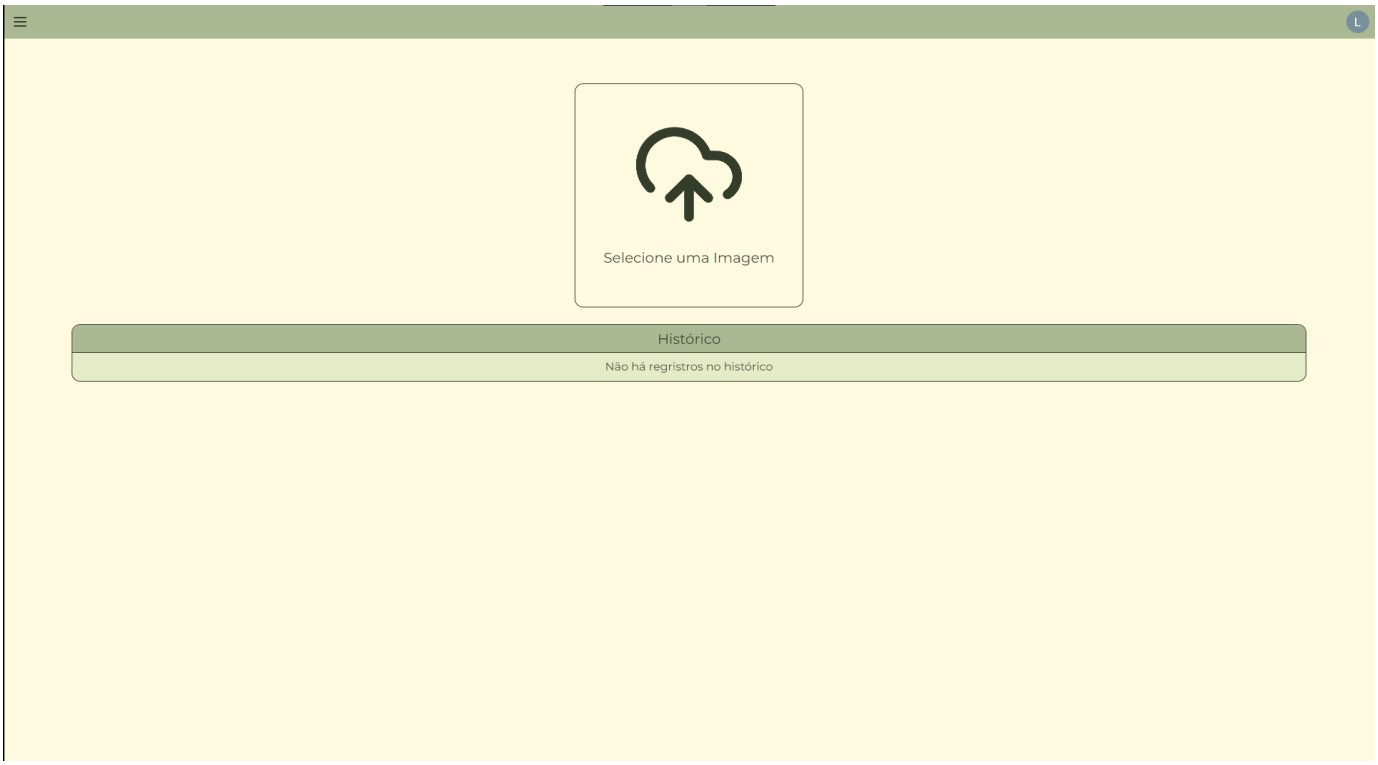


Figura 16: Tela Principal da aplicação antes de adicionar imagens para análise.

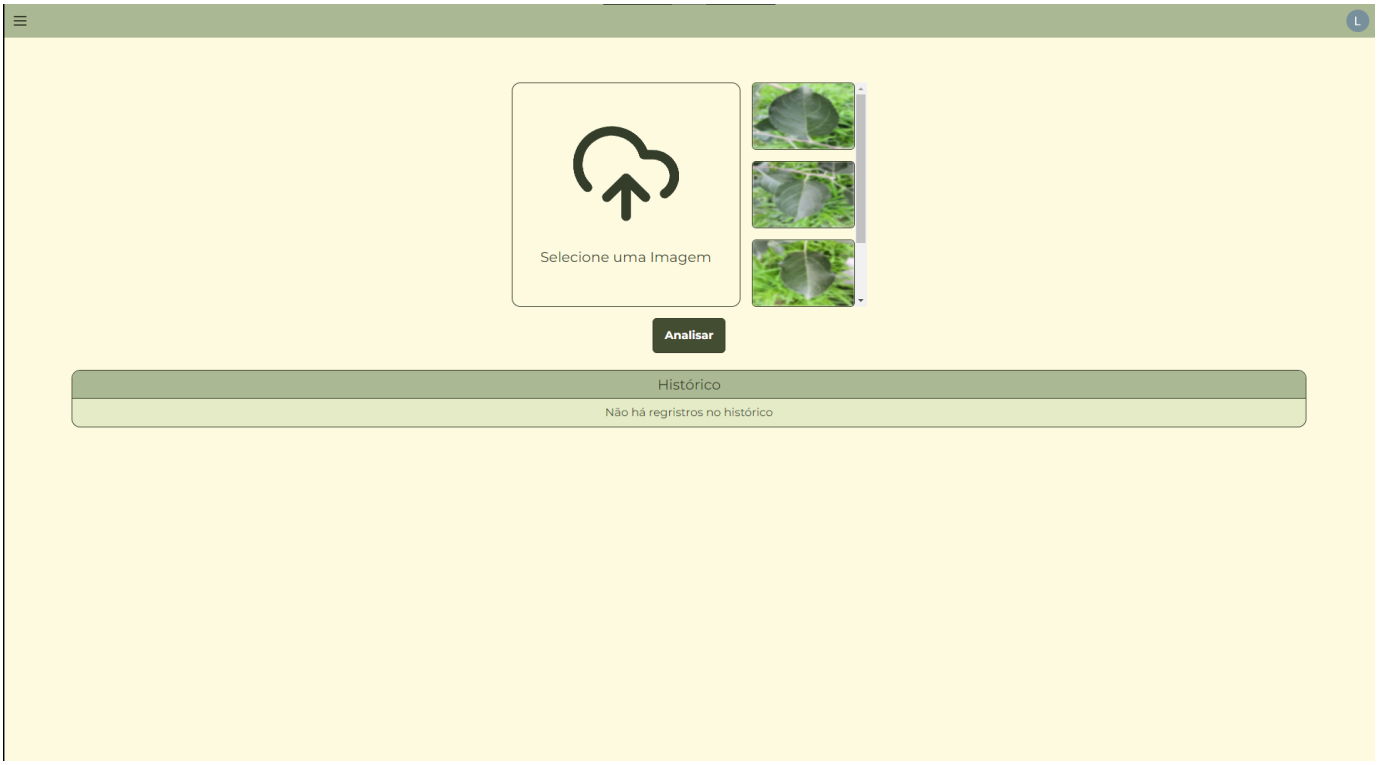


Figura 17: Tela Principal da aplicação após adicionar imagens para análise.

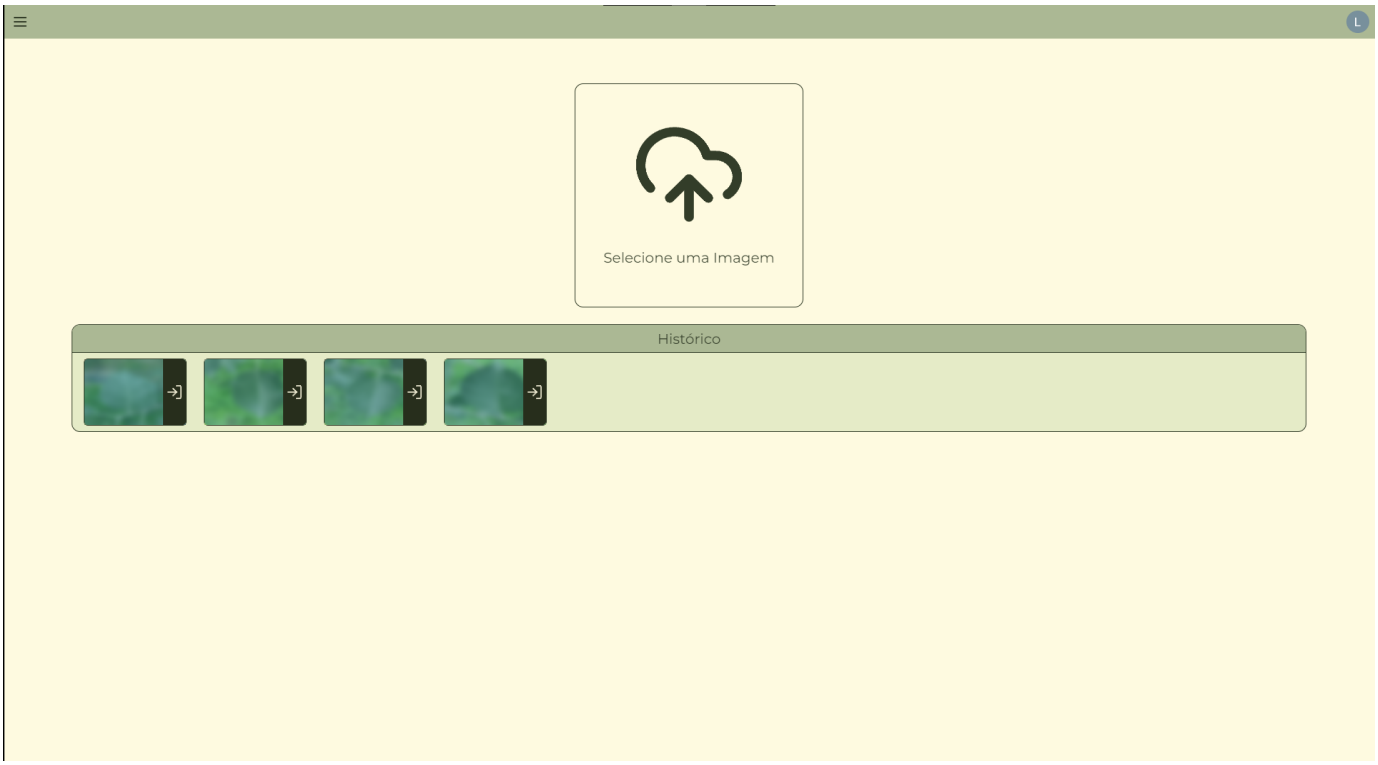


Figura 18: Tela de Histórico da aplicação mostrando análises anteriores.

Figura 19: Tela de Detalhes da aplicação com informações de uma análise específica.