Desenvolvimento de uma Aplicação Móvel Aplicada à Identificação de Doenças Foliares em Culturas Agrícolas

Marcos Vinícius Oliveira Silva
ORCID: 0009-0009-6793-1110
Instituto Federal da Bahia (IFBA)
Vitória da Conquista-BA, Brasil
marcosoliveirasilva7@gmail.com

Crescencio Lima
ORCID: 0000-0002-0286-2056
Instituto Federal da Bahia (IFBA)
Vitória da Conquista-BA, Brasil
crescencio@gmail.com

***Resumo*—A crescente necessidade de combater as perdas agrícolas causadas por pragas e doenças, aliada ao aumento da demanda por alimentos, impulsiona a busca por novas tecnologias que possam auxiliar os agricultores. Motivado pela necessidade de apoiar pequenos agricultores na identificação rápida e precisa de doenças foliares em plantas, foi desenvolvido um aplicativo, denominado `KAWARI`, que integra Visão Computacional e Redes Neurais Artificiais para diagnosticar doenças foliares a partir de imagens capturadas pelo usuário. Além do diagnóstico, o aplicativo fornece informações sobre a doença identificada, pontos de foco próximos, produtos para o tratamento e fornecedores locais. Para o treinamento do modelo, foi utilizado o *dataset PlantVillage*, com imagens de folhas de macieira e videira. Técnicas de aumento de dados — como ajustes de brilho, contraste, rotação e zoom — foram aplicadas exclusivamente ao conjunto de treinamento, após a separação dos subconjuntos de treino, validação e teste, evitando vazamento de dados. Em paralelo, foi desenvolvida uma API em *Node.js*, responsável pela comunicação entre o aplicativo e a RNA, e a aplicação móvel em *React Native*. A contribuição central deste trabalho reside na integração de um sistema de classificação de doenças foliares com funcionalidades complementares — mapa de focos, previsão do tempo e indicação de insumos de fornecedores locais —, que ampliam o valor prático da solução para além de um simples classificador de imagens. O modelo obteve uma acurácia de 98,63% no conjunto de teste. Os resultados demonstram o potencial da abordagem proposta em contexto experimental; a validação em campo com culturas locais constitui etapa necessária para ampliar a generalização e o impacto social da ferramenta.**

***Palavras-chave*—Doenças Foliares em Plantas, Redes Neurais Artificiais, Visão Computacional, APIs, Aplicações Mobile.**

I. Introdução

Segundo dados da FAO (Organização das Nações Unidas para a Alimentação e a Agricultura), estima-se que, por ano, entre 20% e 40% da produção agrícola global é perdida devido às pragas e doenças, resultando em um prejuízo de aproximadamente 220 bilhões de dólares [1]. No Brasil, esta perda corresponde a cerca de 7,7% da produção agrícola, com um prejuízo econômico de aproximadamente 17,7 bilhões de dólares [2].

Aliada a esta perda, a FAO também indica que o aumento da população mundial até 2032 irá pressionar a demanda global por alimentos em aproximadamente 13% [3], o que irá pôr em risco a segurança alimentar de diversas comunidades.

Com o intuito de diminuir o impacto das pragas e doenças na produção de grãos, o agronegócio evoluiu ao longo das décadas através da implantação de novas tecnologias que não só se mostraram mais eficientes no combate a estas pragas como também aumentaram o potencial produtivo das lavouras. Todo este avanço foi potencializado pelo surgimento da Agricultura 4.0 que defende uma maior integração do homem e das tecnologias disponíveis na cadeia produtiva com o intuito de trazer maior eficiência, segurança e rentabilidade [4].

Tal revolução agrotecnológica deriva-se da chamada quarta revolução industrial [5], ou Indústria 4.0, e refere-se ao uso de tecnologias inovadoras na produção de alimentos [6]. Neste contexto, o avanço tecnológico passa a ser determinante na otimização do uso de insumos e de recursos naturais, assim como no aumento da rentabilidade, eficiência e competitividade de mercado.

Dentre as inúmeras inovações apresentadas na Indústria 4.0, uma das que mais se destacam é a utilização da Visão Computacional aliada ao uso de *Machine Learning* e *Deep Learning* para resolução de problemas [7]. Assim é possível realizar o reconhecimento, identificação, detecção, reconstrução e restauração de imagens. Aplicando este conhecimento às práticas agrícolas, podemos utilizar a Visão Computacional na identificação de pragas, doenças, busca por anomalias em uma cultura, contagem de frutos, entre outros [8].

Mediante o cenário apresentado, este trabalho apresenta o desenvolvimento de uma aplicação *mobile* que utilize Visão Computacional e seja capaz de identificar doenças foliares presentes em espécimes vegetais de forma rápida, precisa e eficiente a partir de imagens e fornecer informações sobre o tratamento adequado da planta. Junto a isto, a aplicação apresenta uma lista de insumos agrícolas para tratamento das doenças e seus fornecedores na região.

Através da identificação das pragas e da localização geográfica destas, foi criado um mapa registrando os pontos de foco de determinadas doenças para que agricultores possam

monitorar suas plantações e se preparar para combater o avanço destas.

Para executar tal tarefa, o trabalho proposto realiza um estudo acerca da Visão Computacional embasada em Redes Neurais Artificiais, expondo sua forma de funcionamento e integrando os conhecimentos em programação na área de Inteligência Artificial.

A. Objetivos

Esta seção apresenta o objetivo geral e específico que deverão ser alcançados ao fim do trabalho.

1) Objetivo Geral: O objetivo geral é fornecer a pequenos agricultores uma ferramenta capaz de identificar de forma rápida e assertiva qual a doença presente na planta, além disso, prover um conjunto de informações sobre a mesma, como: dados sobre a doença, registros de doenças agrícolas próximas a sua plantação, produtos indicados para o tratamento e seus fornecedores na região.

2) Objetivos Específicos:

- Realizar um estudo acerca da Visão Computacional embasada em Redes Neurais Artificiais expondo sua forma de funcionamento;
- Integrar os conhecimentos em programação à área de Inteligência Artificial através da implementação de uma Rede Neural Convolucional;
- Desenvolver um aplicativo *mobile* que utilize Visão Computacional para identificar doenças em plantas a partir de imagens dela obtidas;

O restante deste artigo está estruturado da seguinte forma: Seções II e III apresentam, respectivamente, o referencial teórico da pesquisa e os trabalhos relacionados a esta. Seção IV explica a metodologia utilizada no desenvolvimento do trabalho. Seção V expõe as tecnologias adotadas. Seções VI, VII, VIII apresentam, respectivamente, o desenvolvimento da rede neural, API e aplicação *mobile*. Seção IX apresenta as dificuldades encontradas no desenvolvimento do aplicativo. Seção X apresenta as conclusões obtidas ao fim do trabalho e expõe melhorias que podem ser abordadas em projetos futuros;

II. Fundamentação Teórica

O método proposto neste trabalho para a detecção de doenças em espécimes vegetais faz uso de conceitos básicos da literatura de aprendizado de máquina, visão computacional e processamento de imagens. Esta seção apresenta a fundamentação teórica destes conceitos.

A. Aprendizado de Máquina

Aprendizado de Máquina é um ramo da Inteligência Artificial utilizado para definir uma classe de algoritmos capaz de melhorar seu desempenho através do ganho de experiência, desta forma, estes podem alterar seu comportamento de modo a obter sucesso na interação com novos cenários sem que haja a necessidade de interferência humana.

> Um programa de computador é dito aprender a partir de uma experiência E, com respeito a uma classe de tarefas T e medida de desempenho P, se seu desempenho nas tarefas em T, segundo a medida P, melhora com a experiência E [9, p.2, tradução nossa].

A forma como o aprendizado é adquirido varia de acordo com o paradigma utilizado para lidar com o mesmo, podendo estes serem divididos em dois grupos: preditivos e descritivos. Segundo Faceli et al. [10], algoritmos preditivos se caracterizam pela busca de uma função, obtida através de dados de treinamento, capaz de prever um rótulo ou valor que caracteriza um conjunto de dados de entrada. Por outro lado, algoritmos descritivos buscam explorar e analisar os dados em busca de similaridades (agrupamentos) ou associações (regras de associação).

Neste contexto, um sistema de reconhecimento de padrões responsável por associar classes a objetos pode ser dividido com base no método empregado pelo mesmo.

- Aprendizado supervisionado: o algoritmo passa por um processo de treinamento no qual são inseridos dados previamente catalogados para que a partir do modelo observado o classificador possa gerar regras para identificação de novos objetos;
- Aprendizado não supervisionado: o algoritmo não possui informações prévias sobre as classes a que os objetos pertencem. Neste caso, busca-se encontrar a classe mais adequada através de um processo de descoberta através do qual são identificados relacionamentos entre os objetos;
- Aprendizado por reforço: o algoritmo interage com o ambiente de forma dinâmica recebendo uma resposta baseada em uma função de avaliação que lhe permitirá ajustar seus pesos para que o algoritmo possa se aprimorar a cada interação.

Desta forma, o método a ser utilizado para detecção de padrões está diretamente relacionado ao sistema de extração de características utilizado para representar os objetos, logo, a qualidade e quantidade de características obtidas determinará o quão robusto deve ser o reconhecimento de padrões.

1) Redes Neurais Artificiais: As Redes Neurais Artificiais (RNAs) vêm sendo estudadas desde o final da década de 80 quando esta foi redescoberta como paradigma para reconhecimento de padrões. Estas estão contidas dentro da área conhecida como Sistemas Inteligentes [11] e sua implementação utiliza modelos computacionais baseados no sistema nervoso dos seres humanos.

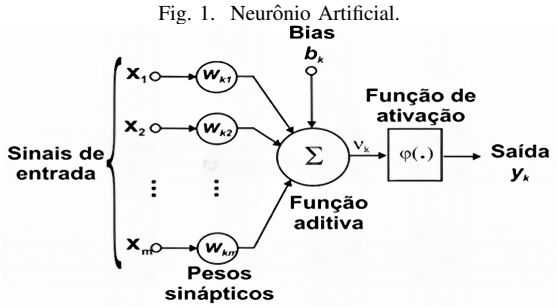
O estudo desta área foi motivado pela percepção de que o cérebro processa informações de forma completamente diferente de um computador digital. O seu sistema de processamento não linear, paralelo e densamente conectado permite que este execute processamentos altamente complexos, como reconhecimentos de padrões e controle do sistema motor, de forma mais rápida e precisa que um computador digital [12].

Uma RNA é composta por uma combinação de processadores simples, representando os neurônios naturais, onde cada processador realiza funções simples como coletar os

sinais existentes em suas entradas, agregá-los e, através de uma função, produzir uma saída. A representação mais simples de um neurônio artificial, e mais utilizada atualmente, foi proposta por McCulloch e Pitts [13]. Este contém as principais características de um neurônio biológico e pode ser observado na Figura 1.

Os sinais de entrada advindos do meio externo são representados pelas entradas x1, x2, x3, ..., xn. Estes são análogos aos neurônios naturais onde a comunicação é realizada através de sinapses.

Fig. 1. Neurônio Artificial.



Fonte: [12].

No neurônio artificial, cada sinapse possui um peso próprio que, diferentemente do cérebro humano, varia entre valores positivos e negativos. Após a recepção do sinal xl pelo neurônio k, este é multiplicado pelo seu respectivo peso sináptico wkl. Após esta etapa é realizada a soma ponderada das entradas, tornando-se possível verificar a saída do neurônio artificial expresso por uk (Equação 1).

$$u_k = \sum_{l=1}^{m} w_{kl}\, x_l \tag{1}$$

O neurônio disposto na Figura 1 também possui um escalar bias adicionado ao produto da junção aditiva. Este possui um valor constante cujo intuito é aumentar ou diminuir a entrada da função de ativação. A Equação 2 expressa a relação entre o bias b_k e a saída u_k.

$$v_k = u_k + b_k \tag{2}$$

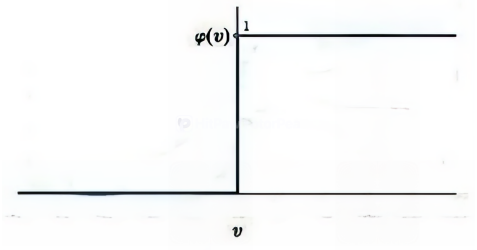
Através do modelo de neurônio artificial proposto por McCulloch e Pitts [13], foram formuladas diversas funções de ativação. As Figuras 2, 3, e 4 expressam tipos básicos dessas funções. Estas são representadas por $\varphi(v)$, definindo assim a saída do neurônio.

A função de limiar, representada graficamente através da Figura 2, assumirá o valor 1 quando o potencial de ativação for maior ou igual a zero, caso contrário, o valor assumido será zero conforme o observado na Equação 3.

$$\varphi(v) = \{1\, se\, v \geq 0\ 0\, se\, v > 0\} \tag{3}$$

A função linear por partes, representada graficamente na Figura 3, assumirá os mesmos valores dos potenciais de

Fig. 2. Função de Limiar.

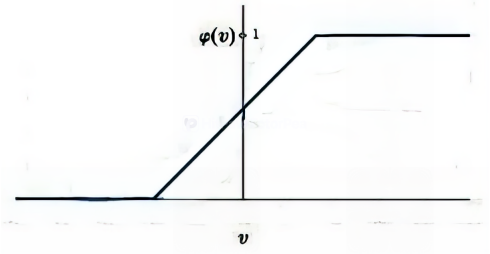


Fonte: Adaptado de [14].

ativação para valores correspondentes ao intervalo a, -a [14] conforme a Equação 4.

$$\varphi(v) = \{a,\, se\, v > a\, v, se\, -a \leq v \leq a\, a,\, se\, v < a\} \tag{4}$$

Fig. 3. Função Linear por Partes

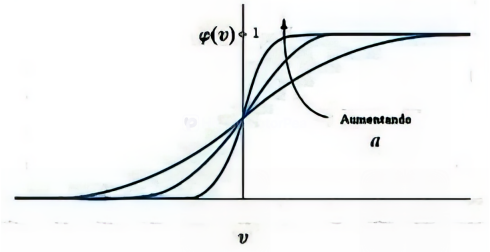


Fonte: Adaptado de [14].

A função sigmoide é a mais utilizada na construção de RNAs. Um exemplo de função sigmoide é a função tangente hiperbólica. A sua saída é definida através da Equação 5, onde o resultado sempre variará entre 0 e1. O fator de inclinação da função é definido através do parâmetro a, gerando assim diferentes inclinações na função sigmoide, conforme Figura 4.

$$\varphi(v) = \frac{1}{1 + e^{av}} \tag{5}$$

Fig. 4. Função sigmoide com parâmetro de inclinação a variável.



Fonte: Adaptado de [14].

B. Visão Computacional

Para os seres humanos, a percepção visual dos elementos que o cercam é uma tarefa relativamente simples. Distinguir cores, objetos, texturas entre outros, é uma tarefa simples.

ISSN:2596-2701

Contudo, a maior habilidade do ser humano, com respeito ao sistema visual, é a capacidade de extrair informações dos elementos que compõem aquela cena. Esta atividade realizada de forma tão trivial é, na verdade, extremamente complexa e difícil de ser reproduzida por computadores.

Segundo Shapiro e Stockman [15], o objetivo da visão computacional é tomar decisões acerca de objetos físicos reais a partir de cenas baseadas em imagens captadas. Ao buscar solucionar tal problema, cientistas estão fazendo uso de diversas técnicas para melhorar o desempenho das aplicações voltadas para a visão computacional, como uso de algoritmos matemáticos para recuperar a forma tridimensional de objetos, rastreio de objetos em cena, entre outros.

1) Técnicas Modernas para Detecção de Doenças em Plantas: No contexto da detecção de doenças foliares em plantas, destacam-se abordagens baseadas em Redes Neurais Convolucionais (CNNs) que aprendem representações hierárquicas diretamente das imagens, dispensando a extração manual de características. Modelos como *MobileNet* [16], *ResNet* [17] e *EfficientNet* [18] têm sido amplamente utilizados nesse domínio por meio de *transfer learning*, aproveitando pesos pré-treinados em grandes bases de dados e adaptando-os à classificação de doenças agrícolas [19].

O uso de *transfer learning* é especialmente relevante quando o conjunto de dados disponível é limitado. Técnicas de *data augmentation* são amplamente empregadas para aumentar a diversidade do conjunto de treinamento e reduzir o *overfitting* [20]. O presente trabalho adota uma arquitetura CNN própria treinada a partir do zero, explorando a capacidade das camadas convolucionais de aprender características específicas das doenças selecionadas.

C. Processamento de Imagens

Ao ser digitalizada, uma imagem é convertida em um *array* de dados onde cada ponto da matriz representa um *pixel*. Neste formato, a imagem pode ser manipulada pelo computador, permitindo a este realizar operações sobre a mesma.

Para que seja possível aplicar operações de visão computacional sobre uma figura, é necessário, geralmente, realizar processos de tratamento da mesma para que o algoritmo aplicado possa extrair o máximo de informações possíveis.

O primeiro passo no processo de tratamento da imagem refere-se à redução de ruído da mesma e a detecção de arestas. Estas operações são intituladas de baixo nível e se caracterizam por sua capacidade de suavizar imagens ainda que este processo não tenha consciência dos elementos que constituem a cena [21]. Este também pode ser aplicado apenas em uma área restrita da imagem.

As operações de nível intermediário estão ligadas a segmentação da imagem – particionamento da mesma em regiões – ou classificação dos objetos em cena. Por fim, as tarefas de alto nível estão relacionadas à visão humana, buscando assim dar sentido aos objetos que compõem a imagem.

III. Trabalhos Relacionados

A agricultura tem vivenciado um grande avanço tecnológico nos últimos anos. Dentro deste cenário, abordagens focadas na identificação de doenças foliares em plantas através do desenvolvimento de aplicativos móveis e visão computacional tem se demonstrado uma abordagem eficiente e inovadora. Desta forma, este capítulo possui como objetivo apresentar trabalhos correlatos que fundamentam o uso de aplicações móveis aliadas a técnicas avançadas de IA para diagnosticar doenças em plantas.

O trabalho desenvolvido por Rosa [22] teve como objetivo desenvolver um aplicativo *Android*, chamado GEFAI, capaz de identificar doenças em diferentes culturas, como soja, milho, feijão, tomate e videiras. A fim de realizar esta tarefa, foram utilizadas imagens provenientes das plataformas *PlantVillage* e *Digipathos* para realizar o treinamento da rede neural. Foram utilizados modelos pré-treinados, *Mobilenet V2* e *Inception V3*, para classificar as imagens. Embora ambos os modelos tenham apresentado bons resultados, o *Mobilenet V2* se destacou com uma acurácia de validação que variou entre 80.04% e 99.69%, a depender da cultura.

Na sua dissertação de mestrado, Leite [23] comparou modelos de detecção de objetos de um e dois estágios para a identificação de doenças na folha da macieira. No primeiro modelo utilizou-se apenas a rede neural *Yolo V3* para detecção e classificação da doença, enquanto no segundo a rede *Yolo V3* será utilizada apenas detectar e segmentar a parte da imagem referente a doença enquanto uma segunda rede neural fará a identificação desta. Ao realizar testes apresentando imagens novas aos modelos treinados, a abordagem em dois estágios se mostrou mais eficiente com uma média de 67% de acerto contra 59.99%.

Em [24], a *FiberNet* é apresentada como uma alternativa aos modelos pré-treinados comumente utilizados na construção de RNAs. Ela combina alta precisão com menor consumo de recursos computacionais, tornando-a uma alternativa extremamente interessante. Seu principal diferencial é a chamada camada *Defiber*, que reduz o tamanho da imagem antes de cada camada de convolução, simplificando o processamento dos dados e facilitando a classificação. O treinamento da rede alcançou uma precisão de 96,25%, e o conjunto de dados utilizado foi constituído por imagens tiradas pelo próprio autor.

Utilizando imagens oriundas da base de dados *PlantVillage*, o trabalho de Carvalho e Zoby [25] analisou 4485 imagens divididas entre quatro doenças distintas da cultura da videira. O conjunto de dados foi dividido da seguinte forma: 80% para treinamento, 15% para validação e 5% para testes. Foi desenvolvida uma rede neural própria e esta alcançou uma acurácia de 97%. Após esta etapa, foram utilizados outros quatro modelos pré-treinados para realizar a classificação das imagens, sendo estes a *MobileNet V2*, *Inception V3*, *ResNet V2* e *NASNet A*. Dentre os novos modelos treinados, o que obteve o melhor desempenho foi o *MobileNet V2* com uma acurácia de 95%.

Funck [26], em seu TCC em Ciência da Computação,

desenvolveu um aplicativo *Android* capaz de identificar a ferrugem asiática na folha da soja. Foi utilizado o modelo *Inception V3* para o treinamento junto ao conjunto de dados advindos da plataforma *Plant Village* com um total de 2770 imagens. O aplicativo conseguiu alcançar uma precisão de 84,61% na detecção da ferrugem asiática.

Silva e Schimiguel [27], através de seu trabalho, explora o uso das tecnologias para auxiliar pequenos agricultores. Este desenvolve um *app Android* que utiliza RNAs e Visão Computacional para identificar doenças em plantas. Foi criada e treinada uma rede neural própria que contou com 39.155 imagens divididas em 43 categorias que foram obtidas através da plataforma da EMBRAPA. A rede foi treinada variando o número de épocas, aumentando-as em 25 a cada novo ciclo, e focou apenas nas doenças, sem levar em consideração as espécies das plantas. Os resultados indicaram que, a partir de 75 épocas, a rede atingia uma média de 85% de acurácia, que pouco variava em ciclos subsequentes.

Os estudos apresentados neste capítulo revelam o avanço na aplicação de tecnologias como a inteligência artificial e visão computacional no diagnóstico de doenças foliares em plantas. A Tabela I demonstra de forma resumida as tecnologias e dados utilizados, além dos resultados obtidos.

TABELA I
RESUMO DOS TRABALHOS DE IDENTIFICAÇÃO DE DOENÇAS EM PLANTAS BASEADAS EM *deep learning*.

Trabalho	Modelo	Dataset	Descrição	Acurácia
[22]	Mobilenet V2 Inception V3	PlantVillage Digipathos (52.869 imagens)	60 doenças e 6 espécies	80.04% - 99.69%
[23]	YoloV3	Kaggle (1.820 imagens)	2 doenças e 1 espécie	67%
[24]	FiberNet	Próprio (826 imagens)	2 doenças e 1 espécie	96.25%
[25]	RNA Própria MobileNet V2 Inception V3 ResNet V2 NASNet	PlantVillage (4.485 imagens)	4 doenças e 1 espécie	95%
[26]	Inception V3	PlantVillage (2.770 imagens)	5 doenças e 1 espécie	84.61%
[27]	RNA Própria	EMBRAPA (39.155 imagens)	43 doenças	85%

Fonte: De autoria própria.

Em resumo, os trabalhos apresentados por Carvalho e Zoby [25], Rosa [22] e Funck [26] demonstraram a eficácia na utilização de modelos pré-treinados, como *Mobilenet V2* e *Inception V3*, na classificação correta das doenças apresentadas. Ferreira e Canuto [24] se destaca por apresentar um modelo pré-treinado, *FiberNet*, com alta precisão e baixo consumo de poder computacional. O trabalho de Silva e Schimiguel [27] demonstrou eficiência nos resultados ao concentrar-se apenas na doença, desconsiderando a espécie da planta e, por fim, Leite [23] apresentou uma abordagem distinta das demais ao levar em consideração a utilização de dois estágios para detecção e classificação da doença, ressaltando uma significativa melhora ao utilizar esta abordagem.

Uma limitação comum aos trabalhos correlatos é a ausência de integração entre o módulo de classificação de doenças e funcionalidades complementares voltadas ao contexto do agricultor. Os trabalhos de Rosa [22] e Funck [26], embora apresentem boas acurácias, restringem-se ao diagnóstico da doença sem oferecer informações sobre tratamento, fornecedores ou registro geográfico de ocorrências. O presente trabalho avança nesse aspecto ao propor um sistema integrado que, além do diagnóstico, fornece ao agricultor dados climáticos, mapa de focos de doenças próximos e indicação de insumos e fornecedores locais. Essa integração representa a principal diferenciação em relação às soluções existentes na literatura.

O presente trabalho buscou diferenciar-se dos mencionados anteriormente ao oferecer não apenas uma ferramenta que detecta de forma precisa doenças apresentadas nas folhas das plantas, mas também uma aplicação que proporcione um conjunto mais amplo de informações, como dados climáticos e registro de pragas em plantações próximas. Isso permite que o produtor não apenas trate, mas também previna que sua plantação seja acometida por tais doenças. Além disso, ao criar um mecanismo que integre produtores e fornecedores de produtos agrícolas locais, facilita-se o acesso aos insumos necessários para o tratamento das doenças identificadas e promove um ecossistema que beneficia o comércio local desses produtos.

Estes estudos demonstram a capacidade destas tecnologias em fornecer ferramentas práticas e acessíveis a pequenos e grandes agricultores. A evolução continua destas técnicas promete facilitar o suporte ao manejo de culturas, promovendo uma agricultura mais sustentável e eficiente.

IV. METODOLOGIA

O presente trabalho, do ponto de vista da natureza, pode ser classificado como uma pesquisa exploratória aplicada [28]. As metodologias utilizadas consistem em conduzir um estudo sobre Redes Neurais Artificiais aplicadas a Visão Computacional, com o objetivo de desenvolver uma ferramenta capaz de identificar doenças em plantas a partir de imagens.

A fim de realizar tal tarefa, o presente estudo segue uma sequência lógica de etapas, descritas a seguir e ilustradas de forma resumida no diagrama de fluxo da metodologia adotada:

1) **Aquisição dos dados:** obtenção e catalogação de imagens referentes às doenças foliares em espécimes vegetais, por meio do *dataset* público *PlantVillage*;
2) **Pré-processamento e aumento de dados:** aplicação de técnicas de processamento de imagens (*RandomRotation*, *RandomContrast*, *RandomBrightness* e *RandomZoom*) **exclusivamente sobre o conjunto de treinamento**, de forma a evitar vazamento de dados (*data leakage*) entre os conjuntos de treino, validação e teste;
3) **Divisão do conjunto de dados:** as imagens originais foram divididas em três subconjuntos mutuamente exclusivos — 80% para treinamento, 15% para teste e 5% para validação — **antes** da aplicação das técnicas de aumento. As imagens aumentadas foram adicionadas

apenas ao subconjunto de treinamento, garantindo que nenhuma imagem derivada de uma mesma imagem original apareça simultaneamente em conjuntos distintos;

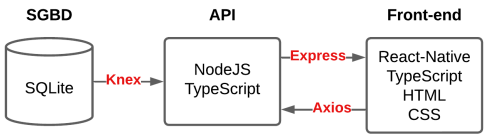
4) **Treinamento da rede neural:** utilização dos dados pré-processados para o treinamento da RNA com aprendizado supervisionado;
5) **Avaliação do modelo:** análise das métricas de classificação (acurácia, precisão, revocação e F1-Score) no conjunto de teste, bem como acompanhamento das curvas de acurácia e perda nos conjuntos de treino e validação ao longo das épocas;
6) **Desenvolvimento do *software*:** em paralelo às etapas anteriores, desenvolvimento da API em *Node.js* e do aplicativo móvel em *React Native*, responsáveis pela interface entre o usuário e o sistema de identificação de doenças.

Adota-se como critério de avaliação a acurácia obtida no conjunto de teste, que representa dados não vistos pela rede em nenhuma etapa anterior ao treinamento. Essa escolha metodológica busca assegurar a validade interna dos resultados reportados e a reprodutibilidade do experimento.

V. Recursos de Produção

A Figura 5 apresenta, em forma de diagrama, os recursos utilizados no projeto e suas relações. Para armazenamento dos dados, foi utilizado o SQLite por ser um sistema gerenciador de banco de dados (SGBD) com um banco de dados relacional leve e embutido. A sua integração com a API foi realizada através do *Knex*. Esta é uma ferramenta desenvolvida para o *Node.js* e permite a execução de comandos SQL em linguagem *JavaScript* e *TypeScript*.

Fig. 5. Diagrama dos recursos de produção.



Fonte: De autoria própria.

A API é responsável por conter as regras de negócio da aplicação, fornecendo os dados necessários ao aplicativo, sendo a ponte entre o SGBD, a RNA e o *Front-End*. Foi utilizado o Node.js em conjunto com a linguagem de programação *TypeScript* para a construção da API.

A comunicação entre *Front* e *Back-End* é gerenciada pelo *Axios* e *Express*. O *Express* opera no lado da API e é responsável pelo gerenciamento de rotas, *middlewares* e interações HTTP. Por outro lado, o *Axios* é uma biblioteca *JavaScript* utilizada no *Front-End* para realizar requisições HTTP.

O desenvolvimento do *Front-End* utilizou o React Native, TypeScript, HTML e CSS para construir um aplicativo moderno, eficaz e robusto. O React-Native permitiu a criação de uma aplicação móvel nativa utilizando uma abordagem baseada em componentes. Para a codificação desta, foi utilizada a linguagem de programação TypeScript, que adiciona tipagem estática ao JavaScript, oferecendo maior segurança ao código evitando erros comuns durante esta etapa. O HTML e CSS foram fundamentais na criação da interface com usuário, estes conferiram estrutura e estilo às telas desenvolvidas.

VI. Rede Neural Artificial

Com base no objetivo da Visão Computacional de tomar decisões sobre objetos físicos a partir de figuras, a rede neural artificial (RNA) utiliza algoritmos matemáticos para analisar e interpretar imagens. O intuito é reconhecer padrões, formas e características, permitindo a tomada de decisões.

O treinamento da rede demonstra como algoritmos complexos podem extrair informações relevantes de imagens. Para isso, a RNA utiliza o conjunto de dados *PlantVillage*, que contém imagens de folhas, para aprender a identificar padrões visuais e realizar diagnósticos de doenças.

Desta forma, este tópico abordará os métodos utilizados para a construção da RNA e treinamento do classificador utilizado para detecção de doenças em plantas. Também serão apresentadas informações sobre como a rede fornece resultados ao usuário de forma precisa. O código fonte produzido nesta etapa está disponível de forma *on-line* e pode ser acessado através do *Google Colab*[1].

A. Ambiente Computacional

O treinamento do modelo de Rede Neural Convolucional (CNN) desenvolvido neste trabalho, sua avaliação e testes de inferência foram realizados através do ambiente de desenvolvimento *Jupyter Notebook*. O ambiente de execução utilizado foi um computador pessoal que possui os seguintes recursos: processador Intel Core i5 de 7º geração, 16 GB de memória RAM DDR4, 500 GB de armazenamento em SSD NVME e Placa de Vídeo NVidia 920MX com 2 GB de memória dedicada.

O *framework* utilizado foi o *TensorFlow*[2], que é um *framework* de código aberto para computação numérica e aprendizado de máquina, na versão 2.12.0 e com a utilização também do *framework Keras*[3], que é uma biblioteca de *deep learning*, aprendizado de máquina, conhecida por sua simplicidade. Algumas outras dependências são as bibliotecas *pandas*[4] versão 1.5.3, *sklearn*[5] versão 1.2.2, *seaborn*[6] versão 0.12.2, *numpy*[7] versão 1.22.4.

Para a emulação do aplicativo em ambiente nativo, foi utilizado o *software Android Studio*. A *API level* utilizada foi a *TiramisuPrivacySandbox* x86_64 com resolução de 1080x2400 *pixels*.

[1] https://l1nk.dev/wpAdR
[2] https://www.tensorflow.org
[3] https://keras.io/
[4] https://pandas.pydata.org/
[5] https://scikit-learn.org/stable/
[6] https://seaborn.pydata.org/
[7] https://numpy.org/

B. Banco de Imagem e Pipeline de Dados

A base de imagens utilizada para o treinamento da rede neural foi o *PlantVillage Dataset*, um banco de imagens públicas disponibilizado em 17 de abril de 2019 e acessível através da plataforma *Kaggle*[8]. Esta consta com 54.305 arquivos de imagens, com 256x256 *pixels*, de doenças foliares em plantas separadas em 38 categorias. Para o desenvolvimento deste trabalho, foram selecionadas apenas 8 classes e duas espécies vegetais distintas, conforme apresentado na Tabela II.

TABELA II
Dataset INICIAL.

Classe	Número de instâncias
maca_saudavel	1.645
maca_ferrugem	275
maca_podridao	621
maca_sarna	630
uva_saudavel	453
uva_mancha_isariopsis	1.076
uva_podridao_negra	1.180
uva_sarampo_negro(esca)	1.383

Fonte: De autoria própria.

O total de imagens utilizadas neste trabalho foi de 7.263 e, infelizmente, não foram escolhidas espécies que tivessem maior impacto na economia local, como café ou manga, por estas não comporem o conjunto de dados.

C. Aumento de Dados

Para ampliar o número de imagens, tornando o modelo mais robusto e adaptável à variações e condições ambientais, foram utilizadas diversas técnicas de pré-processamento dos dados.

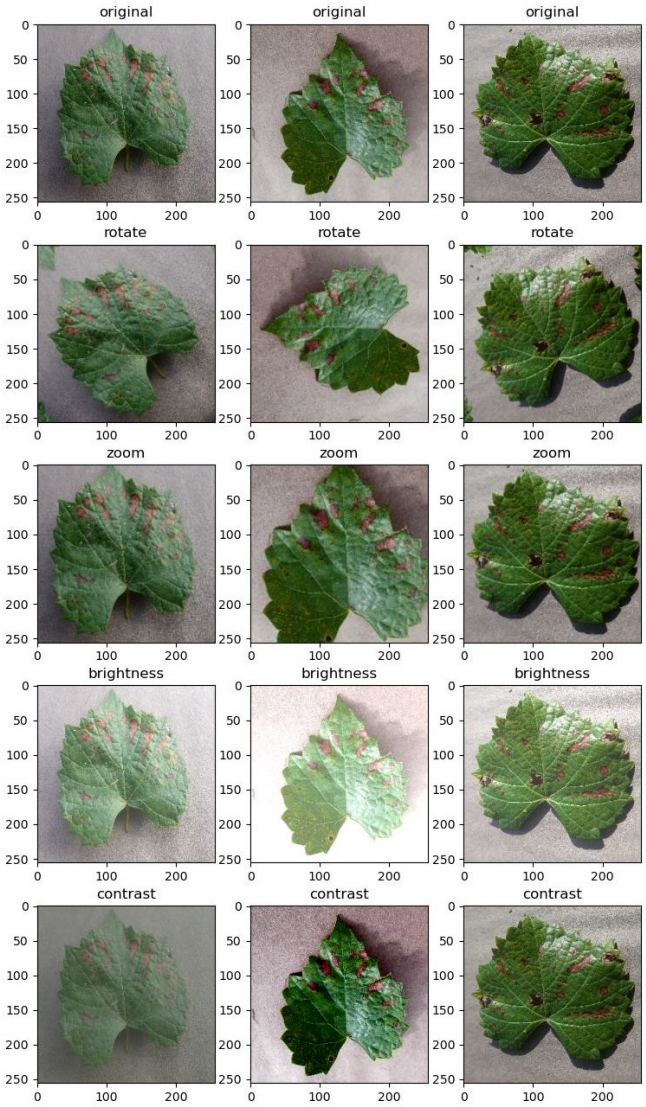
É importante ressaltar que as técnicas de aumento de dados foram aplicadas **exclusivamente** sobre o subconjunto de treinamento. A separação entre treino, validação e teste foi realizada **antes** da aplicação dessas transformações, de forma a garantir que imagens derivadas de uma mesma imagem original não apareçam simultaneamente em conjuntos distintos, evitando o vazamento de dados (*data leakage*) e assegurando a validade da avaliação do modelo.

As operações foram realizadas de forma randômica nas instâncias de treinamento através do *Keras*[9], esta é uma API de alto nível fornecido pelo *TensorFlow* para criar e treinar modelos de *deep learning*. As funções utilizadas foram:

- ***RandomRotation:*** Gira aleatoriamente as imagens com uma rotação no sentido horário e anti-horário no intervalo de 25% e 30%, respectivamente.
- ***RandomContrast:*** Ajusta aleatoriamente o contraste das imagens com uma variação de 50%, para mais ou para menos.
- ***RandomBrightness:*** Ajusta aleatoriamente o brilho das imagens. O fator de ajuste superior utilizado foi de 45%, já o inferior foi de 20% para que as imagens não ficassem demasiadamente escuras.
- ***RandomZoom:*** Ajusta aleatoriamente o zoom das imagens com variação de distância entre mais e menos 30%.

O resultado destas transformações podem ser observados na Figura 6, onde são apresentadas três figuras aleatórias em seu formato original e o resultado das mesmas após aplicação dos filtros. Ao fim deste processo, o número de imagens no subconjunto de treinamento, que inicialmente era de aproximadamente 5.810 (80% de 7.263), foi ampliado para 43.968 por meio das técnicas de aumento aplicadas.

Fig. 6. Transformações



Fonte: De autoria própria.

A Tabela III expressa a distribuição das imagens aumentadas em seus respectivos grupos. A coluna "Classe" expressa a espécie da planta e a doença analisada, enquanto "Número de instâncias" diz respeito ao número total de imagens pertencentes à classe específica após o aumento.

A divisão final do conjunto de dados, após o aumento de dados no subconjunto de treinamento, manteve os subconjuntos

[8]https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset/data
[9]https://www.tensorflow.org/guide/keras

de validação e teste com as proporções originais (5% e 15%, respectivamente) sem nenhuma transformação adicional:

- Treinamento (com aumento de dados): 43.968 imagens;
- Teste (sem aumento de dados, 15% das imagens originais): aproximadamente 1.089 imagens;
- Validação (sem aumento de dados, 5% das imagens originais): aproximadamente 363 imagens.

TABELA III
Dataset DE TREINAMENTO APÓS AUMENTO DE DADOS.

Classe	Número de instâncias
maca_saudavel	9.870
maca_ferrugem	1.722
maca_podridao	3.756
maca_sarna	3.903
uva_saudavel	2.718
uva_mancha_isariopsis	6.512
uva_podridao_negra	7.135
uva_sarampo_negro(esca)	8.352

Fonte: De autoria própria.

D. Arquitetura da Rede

Através da classe *Sequential* fornecida pela biblioteca *Keras*, foi construída uma pilha linear de camadas, sendo esta essencial para a criação da CNN. A primeira camada é responsável pelo pré-processamento das imagens, realizando uma padronização dos valores de entrada para que o processo de aprendizagem obtenha maior eficiência. Para a rede proposta neste trabalho, as imagens foram redimensionadas para a escala de 255x255 e foi realizada uma divisão dos valores de cada *pixel* por 255, resultando em imagens com *pixels* normalizados entre 0 e 1.

Após a etapa de pré-processamento, o processo sequencial da rede é composto por uma Camada Convolucional 2D seguida por uma Camada *BatchNormalization* e depois uma Camada 2D *MaxPooling*. Por fim, é inserida uma Camada de *Dropout* para evitar *overfitting*. Estas camadas se repetem nesta sequência por mais sete vezes até chegarmos às camadas de saída.

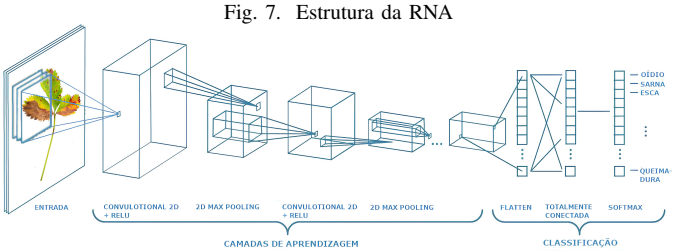
A Camada Convolucional 2D é composta por 8 *kernels* de tamanho 3x3 e utiliza a função de ativação ReLU, que retorna zero para entradas negativas e o próprio valor para entradas positivas, introduzindo não-linearidade ao modelo e acelerando a convergência durante o treinamento [29]. À medida que as camadas se repetem, a quantidade de *kernels* é dobrada e seu tamanho alterna entre 3x3 e 5x5 para que a rede capture recursos em múltiplas escalas.

Na sequência, os dados passam pela Camada *BatchNormalization* que aplica uma normalização aos dados utilizando a média e o desvio padrão do lote, ajudando a evitar problemas de convergência lenta ou divergência do modelo.

A próxima etapa refere-se à Camada *MaxPooling 2D* onde o objetivo é reduzir a amostragem de uma representação de entrada através da extração e agrupamento das características contidas nas sub-regiões da imagem. Por fim, é aplicada uma camada de *Dropout* onde será definida uma fração dos dados de entrada como 0 durante o treinamento, prevenindo assim o *overfitting*. À medida que as camadas se repetem, o *Dropout* terá seu valor alterado, iniciando em 10% até o valor máximo de 30%.

Ao fim, a camada de achatamento (*Flatten*) transforma um tensor multidimensional em um tensor unidimensional, permitindo a passagem dos dados a uma rede totalmente conectada. Esta é formada por 2 camadas densas, a primeira com 2048 neurônios e ativação ReLU, já a segunda contém 8 neurônios e ativação *Softmax*, que converte os valores de saída em uma distribuição de probabilidades sobre as 8 classes, selecionando a classe com maior probabilidade como predição final. A estrutura da RNA pode ser observada através da Figura 7.

Fig. 7. Estrutura da RNA



Fonte: Adaptado de [30].

Ressaltando que a quantidade de neurônios na última camada deve ser igual à quantidade de classes possíveis. Esta rede gerou um total de 19.044.810 parâmetros treináveis.

E. Treinamento e Avaliação do Modelo

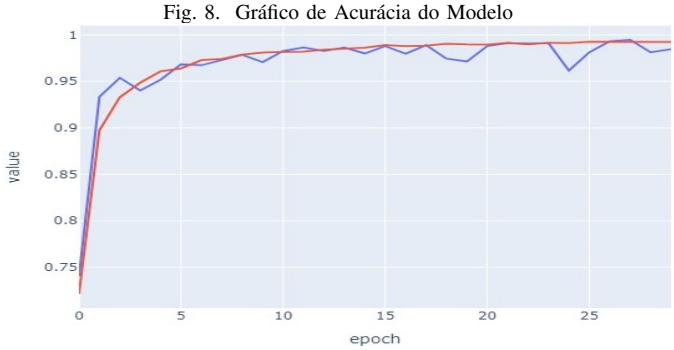
O paradigma de aprendizado adotado no desenvolvimento da rede neural artificial (RNA) foi o preditivo, com o objetivo de permitir que a rede identifique e preveja a doença presente nas folhas a partir das imagens fornecidas. Para o treinamento da rede, foi utilizado o método de aprendizado supervisionado. Nesse processo, as imagens usadas eram previamente rotuladas com as classes correspondentes, permitindo que a rede aprendesse a associar cada imagem à sua respectiva doença.

O processo de aprendizagem, passando pelas camadas informadas na Seção VI-D, foi repetido por 30 épocas (*epochs*). Durante cada iteração, os pesos sinápticos dos neurônios foram ajustados para aprimorar o desempenho da rede. Após cada treinamento, o sistema avaliou o novo modelo gerado e, se sua acurácia fosse superior à do modelo anterior, realizou a substituição.

Ao final do treinamento, foram gerados dois gráficos: um para acurácia (*accuracy*) e outro para perda (*loss*), podendo estes serem visualizados nas Figuras 8 e 9, respectivamente. Os gráficos são compostos por uma linha vermelha, que exibe a evolução do treinamento da rede, e outra azul, que apresenta o resultado obtido após aplicação do novo modelo ao conjunto de dados de validação.

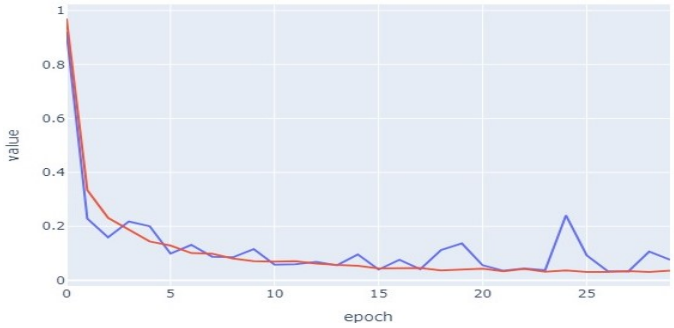
Observa-se que a acurácia do modelo aumentou continuamente ao longo das épocas, enquanto a perda diminuiu progressivamente, aproximando-se de zero. Isto ocorre porque a cada novo ciclo completo de treinamento, os pesos sinápticos

Fig. 8. Gráfico de Acurácia do Modelo



Fonte: De autoria própria.

Fig. 9. Gráfico de Perda do Modelo



Fonte: De autoria própria.

da rede neural foram corrigidos, permitindo uma melhor generalização. Em outras palavras, a rede se adapta não somente aos dados de treinamento, mas também se ajusta de forma eficaz a novos dados não vistos.

Com o passar das épocas, a perda diminuiu e a acurácia aumentou de forma consistente. As métricas de treinamento se estabilizaram rapidamente após cinco iterações, ao passo que as métricas de validação apresentaram uma maior volatilidade. Este comportamento já era esperado, visto que os dados de validação são mais difíceis de prever, uma vez que a RNA não está familiarizada com estas imagens.

O menor valor de perda no conjunto de **teste** foi registrado na época 27, sendo este de 0,0137. Nesta mesma época, o modelo também obteve a maior acurácia no conjunto de teste, alcançando 0,9863, ou 98,63%. A acurácia no conjunto de teste foi o critério utilizado para a seleção do modelo final. Para fins de transparência, os valores das métricas de acurácia (*Acc.*) e perda (*Loss*) dos conjuntos de treino e de validação monitorados na época 27 são apresentados na Tabela IV. Ressalta-se que esses valores dizem respeito ao monitoramento interno durante o treinamento e não ao resultado final de avaliação do modelo, que é dado pelo conjunto de teste.

F. Métricas de Classificação do Modelo

O relatório de classificação da rede neural, presente na Figura 10, exibe algumas informações que podem ser analisadas para obter um maior entendimento do modelo gerado. Essas métricas são fundamentais para avaliar o desempenho

TABELA IV
VALORES DE MÉTRICAS DE TREINO E VALIDAÇÃO NA ÉPOCA 27.

Época	Acc. Treino	Loss Treino	Acc. Validação	Loss Validação
27	0,9863	0,0137	0,9888	0,0112

Fonte: De autoria própria.

da rede em tarefas de classificação, equilíbrio entre classes e acurácia geral.

Fig. 10. Relatório de classificação

	precision	recall	f1-score	support
maca_ferrugem	1.00	1.00	1.00	84
maca_podridao	1.00	0.96	0.98	186
maca_sarna	0.99	0.99	0.99	186
maca_saudavel	0.99	1.00	0.99	492
uva_mancha_isariopsis	0.99	0.98	0.99	324
uva_podridao_negra	0.97	0.99	0.98	354
uva_sarampo_negro(esca)	1.00	0.99	0.99	414
uva_saudavel	0.99	1.00	0.99	168
accuracy			0.99	2208
macro avg	0.99	0.99	0.99	2208
weighted avg	0.99	0.99	0.99	2208

Fonte: De autoria própria.

A primeira métrica avaliada foi a precisão (*Precision*). Pode-se definir esta como a proporção entre verdadeiros positivos (VP) em relação à soma dos verdadeiros e falsos positivos (VP + FP), conforme a Equação 6. Este parâmetro é relevante quando falsos positivos são considerados mais danosos do que falsos negativos. No modelo gerado pela RNA, o menor índice apresentado foi de 0,97, ou 97%, pela classe uva_podridao_negra.

$$Precision = \frac{VP}{VP + FP} \tag{6}$$

Em seguida analisou-se a revocação (*Recall*) (Eq. 7), que mede a proporção entre verdadeiros positivos identificados de forma correta dentre todos os valores de classe positiva esperados. Assim, de forma inversa à precisão (*Precision*), esta é relevante quando considerados os falsos negativos mais prejudiciais que os falsos positivos. Para este parâmetro, o menor valor apresentado pelo modelo foi de 0,96, ou 96%, sendo esta classe a da maca_podridao.

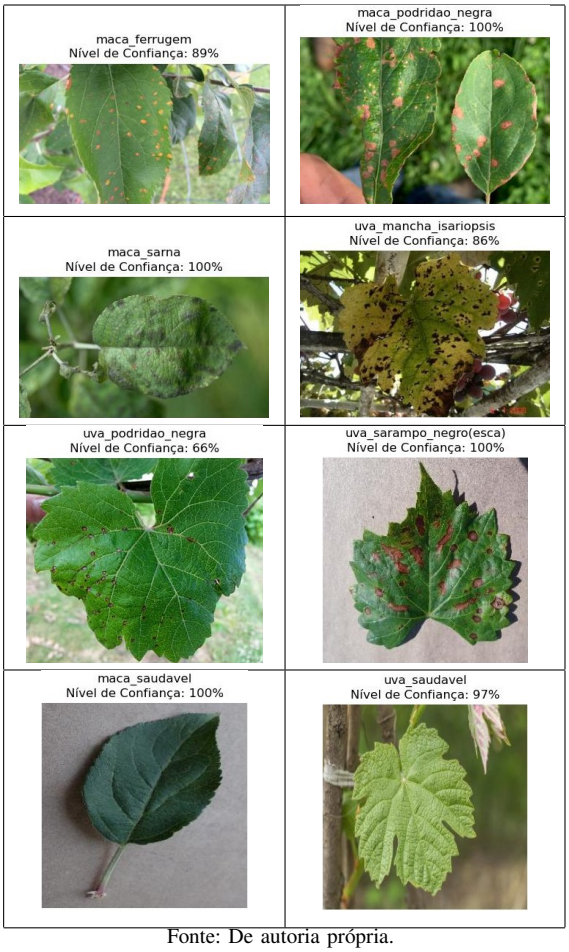
$$Recall = \frac{VP}{VP + FN} \tag{7}$$

Por fim, a última métrica avaliada é o *F1-Score*, cuja fórmula pode ser observada na Equação 8. Esta é representada pela média harmônica entre precisão (*Precision*) e revocação (*Recall*). O *F1-Score* oferece uma avaliação mais equilibrada entre o resultado de falsos negativos e positivos. O modelo treinado apresentou valores que variaram entre 0,98 e 1,00, representando entre 98% e 100% de desempenho respectivamente, sendo o menor valor obtido o das classes maca_podridao e uva_podridao_negra.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (8)$$

Quando avaliamos os valores mínimos obtidos por cada métrica supracitada, pode-se dizer que a rede neural obteve uma boa performance em sua classificação. Ao fim do treinamento, o modelo gerado foi testado sobre imagens que até então não haviam sido utilizadas em nenhuma etapa anterior. Este classificou-as corretamente e o resultado pode ser observado na Tabela V.

TABELA V
PREDIÇÃO DE DOENÇAS FOLIARES.



Fonte: De autoria própria.

G. Conversão do Modelo

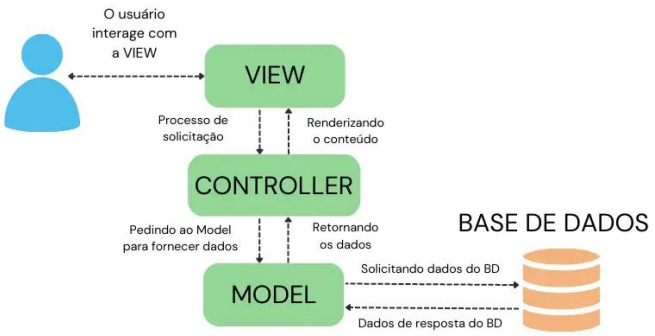
Ao fim do treinamento da RNA foi gerado um modelo no formato h5, entretanto este formato não pode ser lido pela API que foi construída em *TypeScript* com o *NodeJS*. Desta forma, foi necessário utilizar a biblioteca tensorflowjs_converter[10] para converter o modelo para o formato *json*.

Nesta fase do projeto houve alguns problemas: a conversão ocorreu de forma correta, entretanto, por não haver bibliotecas em *JavaScript* compatíveis com as utilizadas na fase de aumento de dados, o modelo não pode ser lido pela aplicação. Para contornar esta situação, ao invés de realizar esta etapa no momento do treinamento da rede, as imagens passaram previamente por esta etapa e seu resultado foi armazenado em formato de novas imagens que se juntaram às originais para a realização do treinamento.

VII. DESENVOLVIMENTO DA API

A API foi desenvolvida em *NodeJS* utilizando *TypeScript* como linguagem de programação e todo o seu código foi versionado utilizando a ferramenta *GitHub*[11]. A arquitetura adotada foi a *MVC* (*Model-View-Controller*), sua escolha deve-se ao fato desta ser um padrão comumente utilizado no desenvolvimento de *softwares* e por permitir uma divisão mais clara e organizada entre os componentes da aplicação. A divisão de camadas, aqui proposta, reduz as dependências entre estas através da divisão de responsabilidades. A Figura 11 exibe, de forma ilustrada, a sua forma de funcionamento.

Fig. 11. Arquitetura MVC.



Fonte: [31].

Nesta arquitetura, o usuário comunica-se diretamente com a camada de visualização (*View*), que captura sua interação e transmite a ação desejada para a camada de controle (*Controller*). Esta, por sua vez, realiza a lógica necessária para a interação com a camada de modelo (*Model*) que, por fim, realiza a manipulação dos dados. Ao fim deste processo, a camada de visualização é atualizada pelo controle com base nos dados fornecidos pelo modelo [31].

A. Estrutura da API

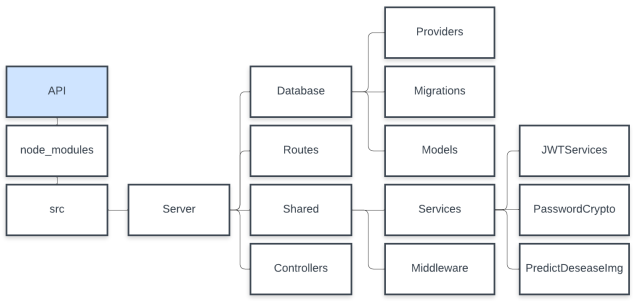
Conforme mencionado, a construção de uma API seguindo os padrões de arquitetura MVC possuem uma estrutura modularizada, desta forma o projeto foi dividido entre *Database*, *Routes*, *Shared* e *Controllers*. A Figura 12 apresenta um diagrama com a estrutura do projeto, onde é possível identificar os módulos da aplicação.

1) Database: A camada de *Database* foi dividida entre *Provider*, *Migrations* e *Models*. As *Migrations* trabalham com a manipulação da base de dados criando, alterando ou removendo tabelas. Estas fornecem os mecanismos para controlar as alterações no banco de dados juntamento com

[10]https://www.tensorflow.org/js/guide/conversion

[11]https://github.com/marcosoliveirasilva/API-PLANT-NODE-TS

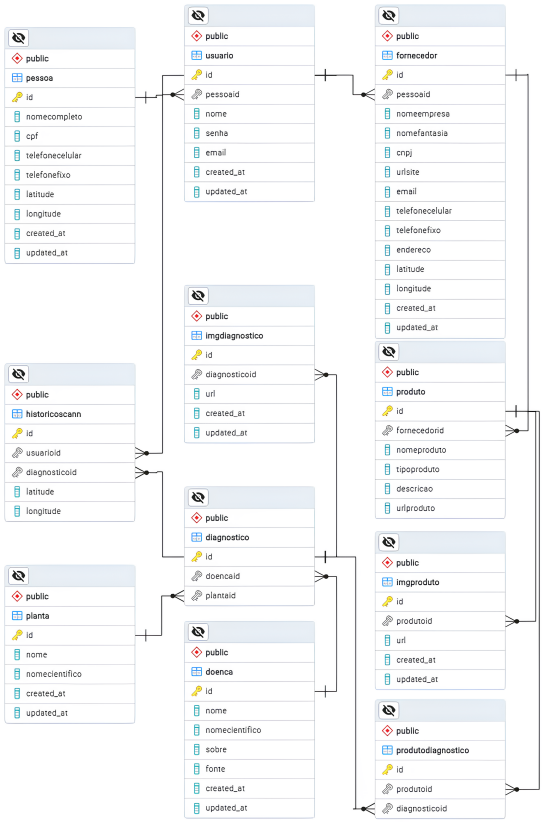
Fig. 12. Estrutura API.



Fonte: De autoria própria.

o versionamento do código. A estrutura deste, criada através das *Migrations*, podem ser observadas através do Diagrama Entidade Relacionamento (DER) presente na Figura 13.

Fig. 13. Diagrama Entidade Relacionamento.



Fonte: De autoria própria.

As *Models* fornecem as estruturas das tabelas criadas através de interfaces, estes são elementos do modelo que definem o conjunto de operações que outros elementos como classes ou componentes devem implementar.

Por fim, os *Providers* são responsáveis por manipular os dados da base, estes realizam operações como *insert*, *delete*, *update* e *query* dos dados. Todas estas operações realizadas no banco de dados, junto com a criação e manipulação das tabelas foram realizadas utilizando o *knex.js*[12].

O *knex.js* é uma ferramenta utilizada junto ao *Node.js* que permite a unificação na forma de realizar as *querys* para os bancos SQL através do *JavaScript*, possuindo também a possibilidade de utilizar o *TypeScript*. Isto trás mais liberdade ao projeto uma vez que torna-se possível alternar entre bancos distintos sem a necessidade de alterar o código já criado.

2) Controllers: O controlador (*Controller*) é responsável por executar a lógica da aplicação intermediando a comunicação entre a camada de visualização e dados. Para realizar algumas de suas tarefas esta camada irá, em alguns momentos, acessar as funções presentes em *Shared*, uma vez que esta classe é composta por serviços que podem ser requisitados por diversas funções da aplicação.

A camada de controle torna-se responsável por atender solicitações do usuário, manipulando a informação entre a requisição e inserção na base de dados. Tudo isto, obviamente, através de interações com a camada de modelo. Embora o controlador execute inúmeras tarefas, a mais importante para esta aplicação é a detecção de doenças em plantas através de imagens. Tal operação é realizada através da interação com a função *PredictDeseaseImg* que é acessada pelo controlador.

a) Função Para Predição de Doenças: A função utilizada para realizar a tarefa de predizer qual doença está presente na folha da planta exibida na imagem é a *PredictDeseaseImg*, esta pode ser observada em Código 1. Em resumo, este código integra as bibliotecas *TensorFlow*.js[13] e *Jimp*[14] para construir uma função que realiza predições de imagens usando um modelo pré-treinado, retornando resultados baseados nos IDs das classes preditas. Esta pode ser subdividida da seguinte forma:

- ***Imports* e Interfaces:** são importados os módulos necessários como `TensorFlow`, `Jimp` para processamento de imagem, além de módulos personalizados. Também é definida uma interface `PredictRequest` que estende a interface `Request` do `Express`, adicionando um campo `file` do tipo `buffer`;
- **Constantes e Variáveis:** é declarada a constante `CLASS_ID` e a variável `model`. `CLASS_ID` é um vetor que contém os *IDs* das classes de predição enquanto `model` é uma variável que armazenará o modelo `TensorFlow` carregado de forma assíncrona;
- **Carregamento do Modelo:** utiliza uma função assíncrona auto-executável para carregar o modelo `TensorFlow`. Desta forma é garantido que o modelo esteja disponível antes de lidar com as requisições de predição;
- **Função *predict*:** esta função é responsável por lidar com requisições *POST* para predição de imagens. Para isto ela inicialmente tenta ler e processar a imagem enviada na requisição. Utilizando o `Jimp`, ela normaliza e redimensiona a imagem para 256x256 *pixels*. Após isto,

[12]https://knexjs.org/
[13]https://www.*TensorFlow*.org/
[14]https://www.npmjs.com/package/*Jimp*

a imagem é convertida para um `TensorFlow` e realizada a predição utilizando o modelo carregado. Por fim, é obtido o `ID` da classe predita com maior probabilidade a partir do tensor de predições que será utilizado para buscar um diagnóstico correspondente na base de dados;
- **Tratamento de Erros:** caso ocorra um erro durante o processo de predição, o servidor retorna um erro 500 com uma mensagem apropriada;
- **Respostas da API:** se tudo ocorrer de modo satisfatório, a função retorna o `ID` da classe para que o diagnóstico possa ser obtido.

```
interface PredictRequest extends Request {
  file: { buffer: Buffer; }
}

const CLASS_ID = [1, 2, 3, 4, 5, 6, 7, 8];

let model: tf.LayersModel;

(async () => { model = await loadModel(); })();

const predictDeseaseImg = async (imageBuffer:
    Buffer):
Promise<number | Error > => {
  try {
    const image = await \textit{Jimp}.read(
        imageBuffer);
    image.resize(256,256).normalize();

    const ima\textit{GET}ensor = tf.tidy(() => {
      const buffer = image.bitmap.data;
      const imgTensor = tf.tensor3d(new
          Uint8Array(buffer),
                        [image.bitmap.height,
                            image.bitmap.width,
                            4]);
      const resized = imgTensor.slice([0, 0, 0],
          [256, 256, 3]).
                        div(tf.scalar(255));
      return resized.expandDims(0);
    });

  const predictions = model.predict(ima\textit{
      GET}ensor) as tf.Tensor;
  const predictClassId = CLASS_ID[predictions.
      argMax(1).dataSync()[0]];

  return predictClassId;
  } catch(error) {
    return new Error('Erro durante a predicao da
        imagem');
  }
}
```

Código 1. Função PredictDeseaseImg.

b) Routes: As rotas são mecanismos de comunicação entre o sistema e outras aplicações, estes fornecem caminhos em uma aplicação *web* que especificam como os recursos podem ser acessados e manipulados através do protocolo HTTP. Elas servem como pontes entre o cliente e a API, direcionando solicitações para funções ou controladores específicos que lidam com lógica de negócios, processamento de dados e interações com o banco de dados.

Para lidar com as rotas, *middlewares*, *templates* e interações HTTP de maneira simples e eficiente, foi utilizado o *framework Express.js*. Através deste é possível utilizar diferentes métodos HTTP (`GET`, `POST`, `PUT`, `DELETE`, etc.), além de permitir a inclusão de parâmetros para uma manipulação mais precisa dos dados.

Para realizar a troca de informações entre servidor e aplicação de forma segura foi implementado o processo de autenticação através do *middleware*. Este processo visa verificar a identidade de um usuário ou aplicação antes deste obter acesso a recursos restritos da API.

O processo de autenticação envolve a verificação de credenciais do usuário, neste caso seria o *e-mail* e a senha. Caso a identidade seja confirmada, é fornecido um *token*, através do JWT[15], que concede acesso seguro as funcionalidades especificas ou dados protegidos.

A utilização do *Express.js* junto ao JWT fornece uma abordagem eficaz na implantação de um processo de autenticação em APIs. Enquanto o *Express* fornece a estrutura adequada para gerenciar rotas e *middlewares*, o JWT facilita a geração, validação e transmissão de forma segura de *tokens* entre cliente e servidor.

VIII. Desenvolvimento da Aplicação *Mobile*

Nesta seção serão abordados pontos importantes no desenvolvimento da interface da aplicação, assim como suas funcionalidades. O nome escolhido para o sistema foi `KAWARI` por significar ”planta doente” em língua guarani. O código fonte está disponível no *GitHub*[16] e o vídeo, exibindo seu funcionamento, está disponível no *Google Drive*[17]. Para o seu desenvolvimento foi utilizado o *framework React Native*. Esta escolha se deu devido a dois principais fatores:

- **Desenvolvimento Multiplataforma:** o *React Native* permite o desenvolvimento de aplicativos nativos para iOS e *Android* utilizando uma única base de código. Isto simplifica a programação, além de diminuir o custo e esforço em comparação com o desenvolvimento nativo de forma individual para cada plataforma;
- **Eficiência e Produtividade:** o *React Native* permite a reutilização dos componentes de interface do usuário em diferentes partes do aplicativo e entre plataformas. Facilitando a manutenção do código enquanto reduz o tempo e desenvolvimento;

Ao iniciar a aplicação, o usuário será direcionado para a tela de *login* onde deverá inserir suas credenciais (*e-mail* e senha). Estes dados são enviados a API para validação e, caso estejam corretos, será retornado um *token* JWT para ser armazenado localmente no *AsyncStorage* do *react* para autenticação futura.

Utilizando o *token* JWT, a aplicação agora pode acessar *endpoints* protegidos pelo *backend*. Através da implementação da biblioteca *Axios*[18], é possível realizar requisições `GET`, `POST`, `PUT`, `DELETE`, entre outras, para buscar, enviar, atualizar ou executar outras operações necessárias para a

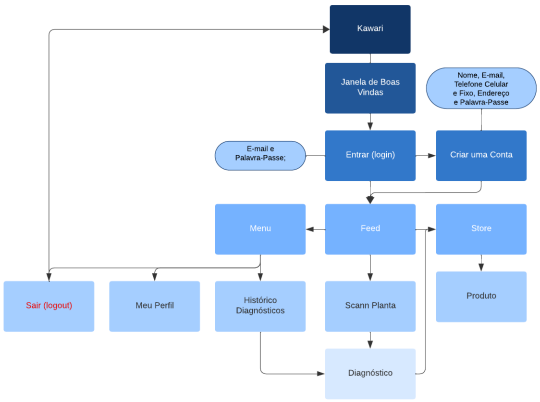
[15]https://www.npmjs.com/package/jsonwebtoken
[16]https://github.com/marcosoliveirasilva/FRONT-PLANT-REACT-EXPO
[17]https://acesse.one/uyY01
[18]https://www.npmjs.com/package/axios

aplicação. Para manter um estado global coeso, foi utilizado contextos para gerenciar as informações compartilhadas entre componentes.

A. *Fluxo da Aplicação*

De um modo geral, a organização lógica da aplicação ficou distribuída entre três eixos: entrada através do *login*, avaliação e diagnóstico das doenças e a loja (*store*) com os produtos adequados para o tratamento destas. A Figura 14 exibe de forma mais detalhada todo o fluxo descrito.

Fig. 14. Fluxo da Aplicação `KAWARI`.



Fonte: De autoria própria.

B. *Página Home*

Ao realizar *login*, o usuário é direcionado para a página principal da aplicação que é composta por um *widget* de previsão do tempo, um mapa com a localização da plantação, dicas sobre plantio e a ferramenta para detecção de doenças nas folhas das plantas. Através da Figura 15 (a) é possível observar a disposição de tais elementos em tela.

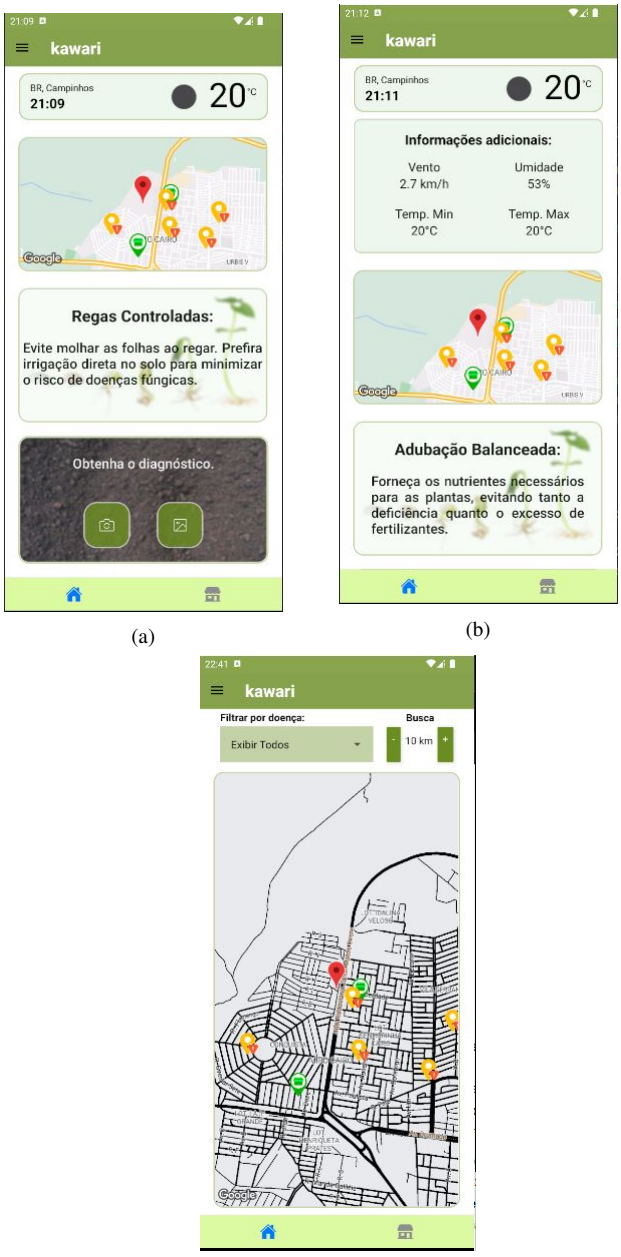
O *widget* de previsão do tempo exibe informações como localização, horário, temperatura e um ícone que reflete o clima atual. Clicando sobre este é exibido um conjunto maior de informações para que o usuário possa controlar melhor sua plantação. As informações adicionais podem ser observadas na Figura 15 (b) e são elas: velocidade do vento, umidade relativa do ar e temperaturas máximas e mínimas.

Além de exibir a localização da plantação indicada pelo usuário, o mapa exibe informações sobre focos de doenças próximas e lojas de produtos agrícolas parceiras. Clicando sobre o mapa, este se expande exibindo filtros que permitem realizar buscas por doenças específicas e por raio de busca, permitindo o acesso a informações mais precisas. A Figura 15 (c) exibe o mapa expandido e seus filtros.

C. *Tela de Diagnóstico*

O *widget* de identificação de doenças, na tela *Home*, contém dois botões. Um deles aciona a câmera e outro a galeria do dispositivo móvel. Caso o usuário selecione a galeria, esta irá exibir as imagens presentes na memória do *smartphone*, assim como pode ser observado na Figura 16 (a).

Fig. 15. Tela Home.



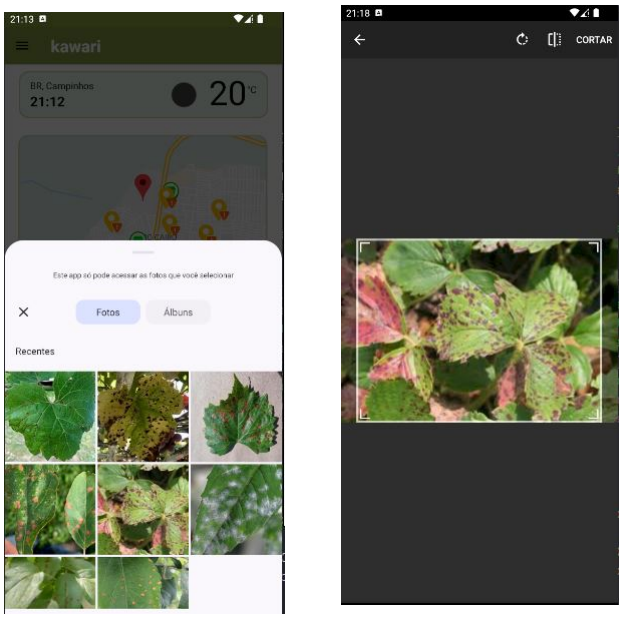
Fonte: De autoria própria.

Caso a imagem selecionada seja muito grande, a aplicação solicitará o seu redimensionamento para que esta se adéque ao padrão de 256x256 *pixels* conforme mostrado na Figura 16 (b). Após isto a imagem é enviada a API para que esta identifique a doença e forneça o diagnóstico ao cliente.

Após realização da predição, o usuário é direcionado a tela de Diagnóstico, conforme apresentado na Figura 16 (c), onde serão exibidas as seguintes informações: nome científico e popular da doença, figuras de outras plantas com o mesmo diagnóstico, informações gerais da doença (com o *link* da fonte) e o botão “Produtos Recomendado” que redireciona

para a tela da loja (*Store*), exibindo apenas os produtos adequados para o tratamento da doença em questão.

Fig. 16. Diagnóstico de Doenças em Plantas.



(a) (b)

(c)

Fonte: De autoria própria.

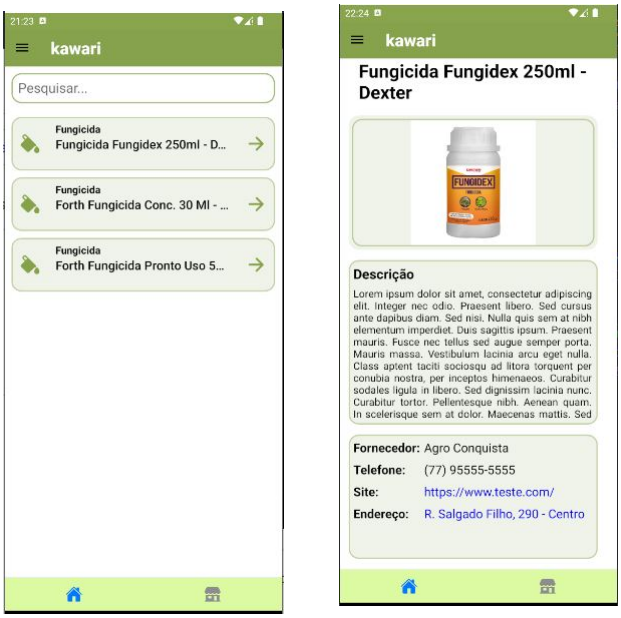
D. *Store*

A tela da loja (*Store*) é responsável por exibir os produtos oferecidos pelos fornecedores de produtos agrícolas parceiros. Ela pode ser acessada de duas maneiras distintas na aplicação. A primeira forma é por meio da *Tab Bar*, situada na parte inferior da tela, através do ícone de loja. A segunda opção é clicando no botão "Produtos Recomendado" disponível na tela de diagnósticos

Caso o usuário acesse a loja através da *Tab Bar*, serão exibidos todos os produtos disponíveis de forma indiscriminada. No entanto, se o acesso for realizado a partir de um diagnóstico específico, serão exibidos apenas os produtos recomendados para o tratamento para a doença específica.

A tela de listagem de produtos, ilustrada na Figura 17 (a), apresenta uma barra de pesquisa que permite ao usuário procurar produtos específicos de forma eficiente. Essa funcionalidade facilita a localização de itens desejados, proporcionando uma experiência de navegação mais intuitiva e direta.

Fig. 17. Loja da Aplicação.



(a) (b)

Fonte: De autoria própria.

Ao clicar sobre produto desejado, o usuário é direcionado para uma tela que exibe informações detalhadas sobre o item, como mostrado na Figura 17 (b). Esta tela inclui o nome do produto, uma imagem ilustrativa e sua descrição completa.

Além dessas informações, a tela também fornece dados sobre o fornecedor, como nome, telefone, URL do produto e endereço físico da loja. Ao clicar na URL, o usuário é redirecionado para a página do produto no navegador padrão do dispositivo. Se o usuário clicar no endereço, o mapa se expande para mostrar a localização da loja.

IX. *Ameaças à Validade*

Esta seção discute as principais ameaças à validade interna, externa, de construto e de conclusão do presente trabalho, seguindo a taxonomia proposta por [32].

A. *Validade Interna*

A principal ameaça à validade interna refere-se ao processo de aumento de dados (*data augmentation*). O aumento foi aplicado exclusivamente sobre o conjunto de treinamento, antes da

separação dos subconjuntos de validação e teste. Isso garante que imagens derivadas de uma mesma imagem original não apareçam simultaneamente em conjuntos distintos, mitigando o risco de vazamento de dados (*data leakage*). A acurácia de 98,63% reportada corresponde ao desempenho do modelo no conjunto de teste, cujas imagens nunca foram utilizadas em nenhuma etapa de treinamento ou validação.

Outra ameaça interna diz respeito ao viés do *dataset PlantVillage*: as imagens foram coletadas em ambiente controlado, com iluminação padronizada e fundo uniforme, o que pode não refletir as condições variáveis encontradas em lavouras reais. Esse fator pode elevar artificialmente a acurácia reportada em relação ao desempenho esperado em campo.

B. Validade Externa

A validade externa é limitada pelo escopo do conjunto de dados utilizado. O modelo foi treinado apenas com imagens de macieira e videira provenientes do *PlantVillage*, o que restringe sua generalização para outras culturas, especialmente aquelas com maior relevância econômica regional, como café e manga. O próprio trabalho reconhece essa limitação e a aponta como oportunidade para estudos futuros.

Adicionalmente, a ausência de validação com imagens coletadas em campo (lavouras reais, com variação de iluminação, ângulo e condições climáticas) representa uma ameaça à generalização dos resultados. Embora o aplicativo tenha sido testado em ambiente emulado (*Android Studio*), não foram realizados testes de usabilidade formais com agricultores ou potenciais usuários finais.

C. Validade de Construto

A acurácia foi adotada como métrica principal de avaliação, complementada por precisão (*Precision*), revocação (*Recall*) e F1-Score. A consistência terminológica entre "fase de teste" e "validação" é esclarecida da seguinte forma: a acurácia de 98,63% refere-se ao conjunto de **teste** (15% das imagens), enquanto os valores de Acc. Treino = 0,9863 e Acc. Validação = 0,9888 presentes na Tabela IV dizem respeito, respectivamente, ao monitoramento do treinamento e à avaliação no subconjunto de validação (5% das imagens) durante a época 27. Essas são métricas distintas e complementares, reportadas de forma transparente para permitir a avaliação do grau de generalização do modelo.

A ausência de uma matriz de confusão detalhada por classe constitui uma limitação do presente trabalho. O relatório de classificação apresentado na Figura 10 fornece as métricas por classe, o que permite uma análise equivalente do desempenho por categoria. A inclusão de uma matriz de confusão completa é apontada como melhoria para versões futuras.

D. Validade de Conclusão

As conclusões do trabalho estão circunscritas ao contexto experimental descrito. A dependência de conectividade de rede para comunicação com a API e o modelo constitui uma limitação operacional que pode afetar o uso em regiões rurais com infraestrutura precária. Além disso, a infraestrutura de servidores externos se mostrou economicamente inviável no escopo atual, o que limita a escalabilidade e a disponibilidade do sistema para múltiplos usuários simultâneos.

Por fim, destaca-se que o presente trabalho tem caráter preliminar e experimental. Os resultados reportados devem ser interpretados como evidência inicial do potencial da abordagem proposta, e não como validação definitiva para implantação em larga escala.

X. Conclusão

O objetivo estabelecido na concepção deste trabalho foi a criação de uma aplicação *mobile* que utilize Visão Computacional para identificar doenças foliares presentes em espécimes vegetais de forma rápida, precisa e eficiente. Somado a isto, a aplicação ainda deveria fornecer dados sobre pontos de focos de doenças próximos a sua localização, informações sobre os insumos agrícolas adequados para o tratamento das doenças e seus fornecedores na região.

O processo mais complexo para o desenvolvimento desta aplicação foi a construção de uma rede neural capaz de identificar com precisão as doenças apresentadas nas imagens, sendo essencial a utilização de um banco de imagens robusto e de qualidade para executar tal tarefa. A construção do modelo foi feita utilizando a linguagem de programação *Python* e demandou significativos recursos computacionais, se mostrando complexa e demorada. Foram repetidos diversas vezes o ciclo de ajuste, treinamento, sendo esta especialmente extensa, e teste do modelo até que o resultado final apresentasse o nível de confiança esperado.

Após o desenvolvimento do sistema, foram realizados testes que constataram que os requisitos específicos e funcionais estabelecidos no início do projeto foram alcançados. Assim, atendendo as atividades de desenvolvimento da aplicação.

É importante ressaltar que o presente trabalho tem caráter preliminar e exploratório. O modelo foi treinado e avaliado com imagens do *dataset PlantVillage*, composto por imagens de macieira e videira capturadas em ambiente controlado. A acurácia de 98,63% reportada representa o desempenho no conjunto de teste e deve ser interpretada nesse contexto específico. A validação em campo, com imagens de culturas locais como café e manga e em condições reais de lavoura, constitui passo necessário para ampliar a validade e o impacto social da proposta.

A. Trabalhos Futuros

A seguir destacamos algumas sugestões para trabalhos futuros.

- **Divisão do Aprendizado:** plantas de espécies diferentes, mas que possuem forma da folha e aspecto da doença semelhantes, podem ser confundidas no momento do reconhecimento. Com base nisto, planeja-se estruturar a divisão do treinamento da rede neural por espécie em trabalhos futuros, visando mitigar essa confusão e obter uma maior acurácia;
- **Combinação de Resultados:** doenças que possuem características semelhantes podem ser mais difíceis de serem

reconhecidas pela rede neural. Para contornar este problema e obter resultados mais precisos, pretende-se expandir o modelo para abranger também o reconhecimento de frutos e caules das plantas. Tal treinamento será realizado de forma separada e o resultado final deverá refletir a combinação das predições, trazendo assim mais segurança e precisão quanto ao diagnóstico;

- **Perfil do Fornecedor:** a arquitetura atual do Kawari já integra com sucesso a visualização de insumos recomendados e os dados de contato e geográficos de fornecedores locais para o usuário final. Como evolução da plataforma, almeja-se desenvolver e implantar o "Portal do Fornecedor", para que os comércios locais cadastrem e atualizem seus catálogos de produtos de forma autônoma e dinâmica;
- **Validação em Ambiente Real:** a aplicação do sistema em ambiente real, com imagens coletadas diretamente em lavouras e com espécies vegetais que possuam relevância na agricultura local, como café e manga, constitui a principal prioridade para a continuidade desta pesquisa. A construção de um *dataset* próprio com culturas locais permitirá reduzir a lacuna entre o problema social anunciado e a validação efetivamente realizada, ampliando o impacto e a generalização do trabalho.
- **Comparação com Modelos de Referência:** como desdobramento deste estudo, planeja-se realizar uma comparação experimental direta entre a arquitetura proposta e modelos de referência amplamente utilizados, como *MobileNet*, *ResNet*, *EfficientNet* e *Inception*, utilizando o mesmo conjunto de dados. Essa comparação permitirá avaliar de forma mais rigorosa as vantagens e limitações da arquitetura desenvolvida.
- **Teste de Usabilidade:** pretende-se também realizar testes de usabilidade com agricultores e potenciais usuários finais. A aplicação de avaliações heurísticas ou testes com usuários reais servirá como forma de validar a experiência de uso do aplicativo e identificar oportunidades de melhoria prática na interface.

REFERENCES

[1] "Frear as pragas e as doenÇas das plantas," *FAO*, 2015.

[2] C. M. OLIVEIRA, A. M. AUADA, S. M. MENDES, and M. R. FRIZZAS, "Crop losses and the economic impact of insect pests on brazilian agriculture," *Crop Protection*, vol. 56, pp. 50–54, 2014.

[3] "Ocde-fao perspectivas agrÍcolas 2023-2032.," *OECD/FAO*, p. 391, 2023.

[4] S. M. FONSECA, S. MASSRUHÁ, A. de Andrade LEITE, and Édson Luis BOLFE, *AGRO 4.0*. Rio de Janeiro: Autografia, 2023.

[5] K. SCHWAB, *A QUARTA REVOLUÇÃO INDUSTRIAL*. Edipro, 1 ed., 2018.

[6] D. C. ROSE and J. CHILVERS, "Agriculture 4.0," *Frontiers in Sustainable Food Systems*, vol. 2, 2018.

[7] J. B. SACOMANO, R. F. GONÇALVES, M. T. da SILVA, S. H. BONILLA, and V. C. SATYRO, *INDUSTRIA 4.0*. São Paulo: Edgard Blucher, 1 ed., 2018.

[8] M. R. BORTH, J. C. IACIA, H. PISTORI, and C. F. RUVIARO, "A visão computacional do agronegócio," *7º Encontro Científico de Administração, Economia e Contabilidade (ECAECO)*, 2014.

[9] T. M. MITCHELL, *MACHINE LEARNING*. New York: McGraw-Hill Education, 1 ed., 1997.

[10] K. FACELI, A. C. LORENA, J. GAMA, and A. C. P. L. F. CARVALHO, *INTELIGÊNCIA ARTIFICIAL*. Rio de Janeiro: LTC, 2011.

[11] J. R. JANG, C. SUN, and E. MIZUTANI, *NEURO-FUZZY AND SOFT COMPUTING*. Londres: Prentice-Hall, 1997.

[12] S. HAYKIN, *REDES NEURAIS*. São Paulo: Bookman, 2 ed., 2001.

[13] W. S. MCCULLOCH and W. PITTS, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[14] I. N. d. Silva, D. H. Spatti, and R. A. Flauzino, *Redes neurais artificiais para engenharia e ciências aplicadas*. São Paulo: Artliber Editora, 2010.

[15] L. SHAPIRO and G. STOCKMAN, *COMPUTER VISION*. Londres: Prentice Hall, 1 ed., 2001.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[18] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning (ICML)*, pp. 6105–6114, 2019.

[19] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 2016.

[20] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[21] S. RUSSELL and P. NORVIG, *INTELIGÊNCIA ARTIFICIAL*. Rio de Janeiro: Elsevier, 1 ed., 2004.

[22] E. N. Rosa, "Utilização de redes neurais para reconhecimento de doenças foliares em culturas agrícolas," *Universidade Federal de Santa Catarina*, 2022.

[23] T. d. M. Leite, "Deep learning em dois estágios para detecção e classificaçãode doenças em folhas de plantas com aplicação em dispositivos móveis," Master's thesis, Universidade de São Paulo, 2021.

[24] V. R. Ferreira and A. M. de Paula Canuto, "Potencializando a classificação de imagens com eficiência e robustez através da fibernet.," *Revista de Ciência da Computação*, vol. 5, no. 1, pp. 7–20, 2023.

[25] R. G. Carvalho and L. T. Zoby, "Convolutional neural networks for leaf disease classification," in *Anais do XV Encontro Nacional de Inteligência Artificial e Computacional*, pp. 332–342, SBC, 2018.

[26] F. Funck, "Detectando a ferrugem asiática na folha da soja utilizando redes neurais convolucionais," *Monografia de Graduação, Universidade Tecnológica Federal do Paraná*, 2019.

[27] M. J. Silva and J. Schimiguel, "Identificação de doenças em plantas por meio de processamento de imagens: redes neurais convolucionais como auxílio à agricultura," *Revista de Ubiquidade*, vol. 3, no. 1, pp. 91–111, 2020.

[28] A. C. Gil, *Métodos e técnicas de pesquisa social*. São Paulo: 6. ed. Ediitora Atlas SA, 2008.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. http://www.deeplearningbook.org.

[30] P. Raghav, "Understanding of convolutional neural network (cnn)," 2018.

[31] "O que é mvc?," *Usando py*, 2023.

[32] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer, 2012.