INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Bahia

Campus
Vitória da Conquista

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

MARCOS VINÍCIUS OLIVEIRA SILVA

KAWARI - DESENVOLVIMENTO DE UMA APLICAÇÃO MÓVEL APLICADA A IDENTIFICAÇÃO DE DOENÇAS FOLIARES EM CULTURAS AGRÍCOLAS

Vitória da Conquista-BA
2024

MARCOS VINÍCIUS OLIVEIRA SILVA

KAWARI - DESENVOLVIMENTO DE UMA APLICAÇÃO MÓVEL APLICADA A IDENTIFICAÇÃO DE DOENÇAS FOLIARES EM CULTURAS AGRÍCOLAS

Monografia apresentada ao Instituto Federal de Educação, Ciência e Tecnologia da Bahia – IFBA campus Vitória da Conquista, curso de Bacharelado em Sistemas de Informação como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Crescencio R. Lima Neto

Vitória da Conquista-BA
2024

FICHA CATALOGRÁFICA ELABORADA PELO SISTEMA DE BIBLIOTECAS DO IFBA, COM OS DADOS FORNECIDOS PELO(A) AUTOR(A)

S586 Silva, Marcos Vinicius Oliveira

KAWARI: DESENVOLVIMENTO DE UMA APLICAÇÃO MÓVEL APLICADA A IDENTIFICAÇÃO DE DOENÇAS FOLIARES EM CULTURAS AGRÍCOLAS / Marcos Vinicius Oliveira Silva; orientador Crescencio Rodrigues Lima Neto -- Vitória da Conquista : IFBA, 2024.

61 p.

Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) -- Instituto Federal da Bahia, 2024.

1. Doenças Foliares em Plantas. 2. Redes Neurais Artificiais. 3. Visão Computacional. 4. Aplicações Mobile. 5. API. I. Lima Neto, Crescencio Rodrigues, orient. II. TÍTULO.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA
Av. Sérgio Vieira de Mello, 3150 - Bairro Zabelê - CEP 45078-900 - Vitória da Conquista - BA - www.portal.ifba.edu.br

DECLARAÇÃO - VDC/CSI.VDC

A banca examinadora, aprova o Trabalho de Conclusão de Curso “KAWARI - DESENVOLVIMENTO DE UMA APLICAÇÃO MÓVEL APLICADA A IDENTIFICAÇÃO DE DOENÇAS FOLIARES EM CULTURAS AGRÍCOLAS” elaborado por “MARCOS VINÍCIUS OLIVEIRA SILVA” como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação, pelo Instituto Federal de Educação, Ciência e Tecnologia da Bahia.

Documento assinado eletronicamente por **CRESCENCIO RODRIGUES LIMA NETO**, **Professor(a) do Ensino Básico, Técnico e Tecnológico - EBTT**, em 02/09/2024, às 18:48, conforme decreto nº 8.539/2015.

Documento assinado eletronicamente por **MARCELA ALVES PEREIRA**, **Professor(a) do Ensino Básico, Técnico e Tecnológico - EBTT**, em 02/09/2024, às 22:10, conforme decreto nº 8.539/2015.

Documento assinado eletronicamente por **PABLO FREIRE MATOS**, **Professor Efetivo**, em 03/09/2024, às 12:03, conforme decreto nº 8.539/2015.

A autenticidade do documento pode ser conferida no site http://sei.ifba.edu.br/sei/controlador_externo.php?acao=documento_conferir&acao_origem=documento_conferir&id_orgao_acesso_externo=0 informando o código verificador **3703168** e o código CRC **B2CBBFF0**.

Dedico este trabalho à minha mãe e ao meu pai, in memoriam.

Agradecimentos

Primeiramente, gostaria de expressar minha mais profunda gratidão à minha esposa, Arlete. Seu apoio incondicional, paciência e compreensão foram fundamentais durante toda a minha caminhada e, principalmente, durante o processo de elaboração deste trabalho. Sua presença constante e incentivo foram uma fonte de força e motivação para mim.

A toda a minha família, mas sobretudo aos meus pais, Patriciano e Maria Selma, não tenho palavras suficientes para agradecer pelo amor ao longo da minha vida. Agradeço também aos meus irmãos pelo carinho e apoio constante, que foram essenciais em momentos difíceis.

Sou imensamente grato aos meus professores pelo esforço, dedicação, zelo e empenho em compartilhar o seu saber, oferendo-me conselhos e críticas construtivas que me permitiram ser uma pessoa e profissional melhor. Especialmente, agradeço ao professor Pablo F. Matos por ser uma referência para mim, ao professor Crescêncio R. Lima Neto por me orientar nesta que é uma fase crucial de minha formação e ao professor Camilo A. Carvalho por me dar a força que precisava para retornar a esta instituição e encerrar mais um ciclo de minha vida.

Aos meus amigos, em especial a Caio e Lucas, agradeço pelo apoio moral e pela companhia, que tornaram essa jornada mais leve e agradável. Suas palavras de encorajamento e compreensão foram vitais durante os momentos mais intensos do processo.

E, finalmente, a todos aqueles que, de alguma forma, contribuíram para que eu pudesse estar aqui neste momento. Cada gesto de apoio, cada conselho e cada contribuição foram fundamentais para que eu pudesse alcançar este objetivo.

A todos que fizeram parte desta trajetória, meus mais sinceros agradecimentos.

Marcos Vinícius Oliveira Silva

"A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original".

(Albert Einstein)

KAWARI - DESENVOLVIMENTO DE UMA APLICAÇÃO MÓVEL APLICADA A IDENTIFICAÇÃO DE DOENÇAS FOLIARES EM CULTURAS AGRÍCOLAS

Resumo

A crescente necessidade de combater as percas agrícolas causadas por pragas e doenças, aliado ao aumento da demanda por alimentos, gera a necessidade pela busca por novas técnicas e tecnologias que possam auxiliar os agricultores. Motivado pela necessidade de auxiliar pequenos agricultores no processo de identificação rápida e precisa de doenças foliares em plantas, foi desenvolvido um aplicativo, chamado KAWARI, utilizando Visão Computacional e Redes Neurais Artificiais para diagnosticar doenças foliares em plantas. O aplicativo utiliza Visão Computacional e Redes Neurais Artificiais para analisar imagens capturadas pelo usuário e fornecer um diagnóstico, além de informações adicionais sobre a doença, pontos de focos próximos, produtos para o tratamento e fornecedores locais. Para o treinamento do modelo, foi utilizado o *dataset* do *PlantVillage* contendo imagens de folhas doentes da macieira, videira, cerejeira e morangueiro. Para dar maior robustez a este, foram utilizadas técnicas de pré-processamento de imagens, como ajustes de brilho e contraste, rotação e zoom, para aumento do conjunto de dados. Em paralelo, foi desenvolvida uma API em *Node.js*, responsável pela comunicação entre a aplicativo e a RNA, e a aplicação mobile em *React Native*. A comunicação entre aplicativo, API e rede neural se mostrou eficiente, onde esta apresentou uma eficiência de 98% no diagnóstico das imagens. O trabalho expõe o potencial da Visão Computacional como ferramenta de apoio a agricultura, permitindo a detecção precisa e rápida de doenças em plantas.

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

KAWARI - DEVELOPMENT OF A MOBILE APPLICATION APPLIED TO THE IDENTIFICATION OF FOLIAR DISEASES IN AGRICULTURAL CROPS

Abstract

The growing need to address agricultural losses caused by pests and diseases, combined with the increasing demand for food, drives the search for new techniques and technologies to assist farmers. Motivated by the need to help small farmers with the rapid and accurate identification of foliar diseases in plants, an application named KAWARI was developed using Computer Vision and Artificial Neural Networks to diagnose leaf diseases in plants. The app uses Computer Vision and Artificial Neural Networks to analyze images captured by the user and provide a diagnosis, along with additional information about the disease, nearby focus points, treatment products, and local suppliers. For model training, the PlantVillage dataset was used, containing images of diseased leaves from apple trees, grapevines, cherry trees, and strawberry plants. To enhance the model's robustness, image preprocessing techniques such as brightness and contrast adjustments, rotation, and zoom were applied to augment the dataset. Additionally, a Node.js API was developed to handle communication between the application and the neural network, alongside a mobile application built with React Native. The communication between the application, API, and neural network proved efficient, with the system achieving a 98% accuracy rate in image diagnosis. This work demonstrates the potential of Computer Vision as a tool to support agriculture, enabling precise and rapid detection of plant diseases.

Keywords: Leaf Diseases in Plants, Artificial Neural Networks, Computer Vision, APIs, Mobile Applications.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

RNA	Rede Neural Artificial
API	*Application Programming Interface*
SQL	*Structured Query Language*
SGBD	Sistema de Gerenciamento de Banco de Dados
CNN	*Convolutional Neural Network*
MVC	*Model View Controller*
ID	*Identity*
HTTP	*Hypertext Transfer Protocol*
JWT	*JSON Web Token*
URL	*Uniform Resource Locator*
GPU	*Graphics Processing Unit*
TCC	Trabalho de Conclusão de Curso

Sumário

1 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 (FREAR..., 2015). No Brasil, esta perda corresponde a cerca de 7,7% da produção agrícola, com um prejuízo econômico de aproximadamente 17,7 bilhões de dólares (OLIVEIRA *et al.*, 2014).

Aliado 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% (OCDE-FAO..., 2023), 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 as 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 (FONSECA *et al.*, 2023).

Tal revolução agrotecnológica deriva-se da chamada quarta revolução industrial (SCHWAB, 2018), ou Industria 4.0, e referem-se ao uso de tecnologias inovadoras na produção de alimentos (ROSE; CHILVERS, 2018). 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 (SACOMANO *et al.*, 2018). 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 (BORTH *et al.*, 2014).

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.

O restante deste documento está estruturado da seguinte forma:

i) Capítulos 2 e 3 apresentam o referencial teórico da pesquisa e os trabalhos relacionados a esta;

ii) Capítulo 4 explica as tecnologias adotadas;

iii) Capítulos 5, 6, 7 apresentam, respectivamente, o desenvolvimento da rede neural, API e aplicação *mobile*;

iv) Capítulo 8 apresenta as dificuldades encontradas no desenvolvimento do aplicativo.

v) Capítulo 9 apresenta as conclusões obtidas ao fim do trabalho e expões melhorias que podem ser abordadas em projetos futuros;

1.1 Objetivos

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

1.1.1 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.

1.1.2 Objetivos Específicos

- Realizar um estudo acerca da Visão Computacional embasada em Redes Neurais Artificiais expondo sua forma de funcionamento;
- Integrar os conhecimentos em programação a 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 obtidas da mesma;

1.2 Hipótese

A utilização de uma aplicação *mobile* capaz de identificar doenças em plantas a partir de fotos capturadas pelo *smartphone* proverá a pequenos agricultores uma ferramenta prática na identificação de doenças que afetem suas culturas. Promovendo um tratamento precoce e, por sua vez, tornando suas colheitas mais produtivas e eficientes, garantindo assim uma maior produção de alimentos.

1.3 Metodologia

O presente trabalho, do ponto de vista da natureza, pode ser classificado como uma pesquisa exploratória aplicada (GIL, 2008). 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 visa seguir uma sequência logica de etapas. A primeira etapa consiste na aquisição dos dados, onde deverão ser obtidas e catalogadas imagens referentes às doenças foliares em espécimes vegetais.

O passo seguinte consiste no tratamento das imagens. Para isto, serão utilizadas técnicas de processamento de imagens com o intuito de realizar a extração das informações de interesse. Após esta etapa, os dados derivados das imagens pré-processadas deverão ser utilizadas para o treinamento da rede neural artificial a fim de que esta possa aprender a identificar doenças foliares em imagens não catalogadas.

A avaliação da eficiência do modelo gerado será realizada através da análise das Métricas de Classificação deste. Em paralelo as etapas supracitadas, será iniciado o desenvolvimento do *software mobile* responsável por ser a interface entre o usuário e o sistema de identificação de doenças em plantas.

2 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.

2.1 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 (MITCHELL, 1997, 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.* (2011), 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.

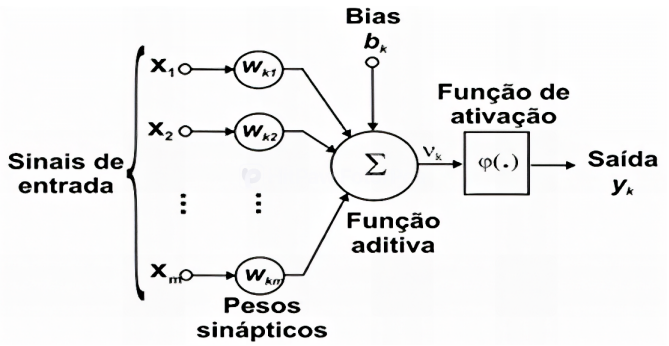
2.1.1 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNAs) vem 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 (JANG *et al.*, 1997) 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 a 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 (HAYKIN, 2001).

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 (1943). Este contém as principais características de um neurônio biológico e pode ser observado na Figura 1.

Figura 1 – Neurônio Artificial.



Fonte: (HAYKIN, 2001).

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.

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.

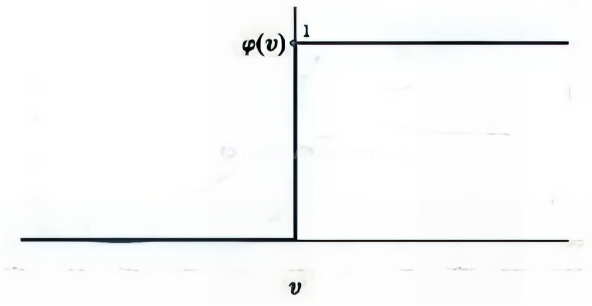
$$v_k = u_k + b_k \tag{2}$$

Através do modelo de neurônio artificial proposto por MCCULLOCH e PITTS (1943), foram formuladas diversas funções de ativação. As Figuras 2, 3, e 4 expressam tipos básicos de funções de ativação. Estas funções 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.

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

Figura 2 – Função de Limiar.

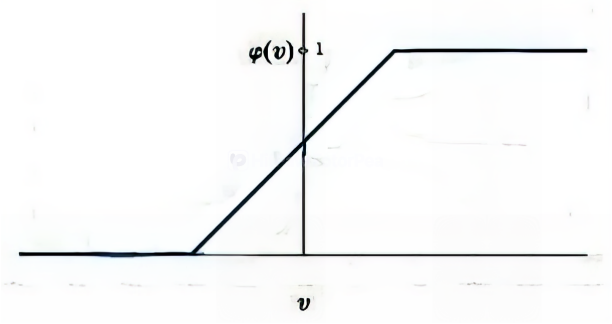


Fonte: Adaptado de Silva *et al.* (2010).

A função linear por partes, representada graficamente na Figura 3, assumirá os mesmos valores dos potenciais de ativação para valores correspondentes ao intervalo a, -a (SILVA *et al.*, 2010) conforme a Equação 4.

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

Figura 3 – Função Linear por Partes

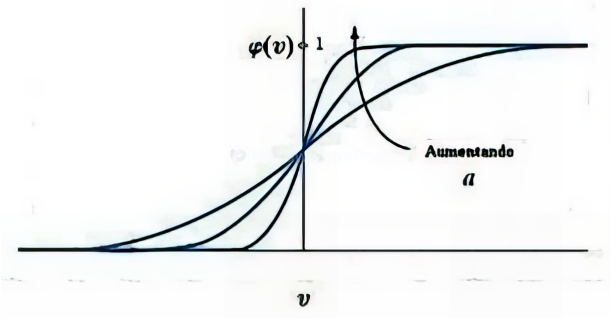


Fonte: Adaptado de Silva *et al.* (2010).

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.

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

Figura 4 – Função sigmoide com parâmetro de inclinação a variável.



Fonte: Adaptado de Silva *et al.* (2010).

2.2 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. 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 (2001), 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.

2.3 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 (RUSSELL; NORVIG, 2004). 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.

2.3.1 Detecção de Arestas

Um dos maiores problemas na detecção de objetos em imagens é a definição de bordas. Esta tarefa consiste em identificar regiões da imagem onde há uma mudança súbita no brilho (RUSSELL; NORVIG, 2004).

A detecção de bordas não é uma tarefa simples, este processo baseia-se na descontinuidade da imagem, entretanto, uma imagem digital – por ser uma representação discreta de um modelo físico – não apresenta uma função natural do que seja descontinuidade (RIOS, 2011).

Uma mudança brusca no brilho de uma imagem pode ter vários significados, isso irá variar de acordo com a iluminação, ângulo, propriedades do material, reflexão e refração da luz sobre o mesmo, cor dos objetos em cena, entre outros.

A Figura 5(a) mostra a imagem de uma cena contendo um corredor prestes a dar a largada em uma corrida, e (b) mostra a saída de um algoritmo de detecção de bordas aplicados sobre (a). É possível observar, na Figura 5(b), que nem todas as arestas estão alinhadas, também há intervalos não aparece nenhuma aresta. Há locais na imagem compostos por arestas que não possuem significado algum, pois estes são derivados de ruídos.

Figura 5 – Reconhecimento de borda.

(a) (b)

Fonte: (RIOS, 2011).

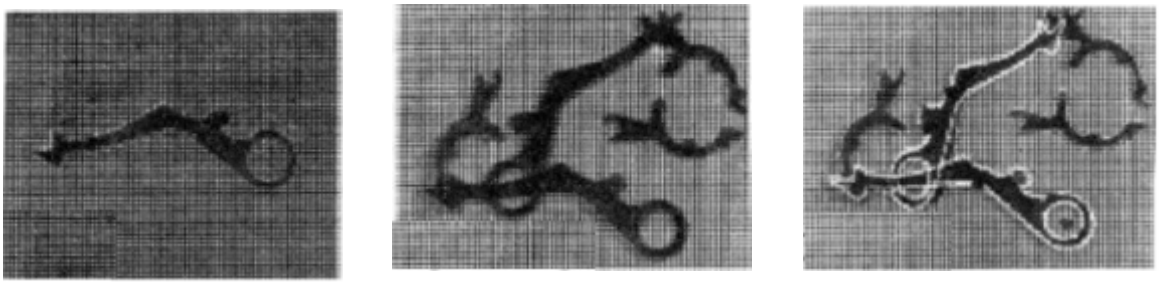
2.3.2 Segmentação de Imagens

Segundo RUSSELL e NORVIG (2004), "a segmentação é o processo de desmembrar uma imagem em grupos, com base em semelhanças entre os *pixels*". Os elementos que compõem uma imagem possuem textura, cor, brilho entre outras. A segmentação busca identificar a descontinuidade dos *pixels* para que, a partir disso, este possa dividir uma imagem em unidades significativas, ou seja, nos objetos de interesse que a compõem (FILHO; NETO, 1999).

A segmentação não supervisionada embasada apenas em atributos de baixo nível, como cor e brilho, tende a falhar (RUSSELL; NORVIG, 2004). Como citado no capítulo anterior, estes atributos sofrem influência direta da iluminação, ângulo da imagem, entre outros fatores. Uma forma de solucionar este problema seria incorporando técnicas de alto nível como métodos baseados em aparência para comparar as bordas do objeto requerido com as da figura que se espera encontrar (RIOS, 2011).

A Figura 6 (a) expressa o modelo que será utilizado para comparação, enquanto (b) expressa à cena da qual deverá ser extraída as informações acerca dos objetos que a compõem. Em (c) pode-se observar as figuras extraídas de (b) através do modelo (a).

Figura 6 – Reconhecimento de objetos com base em características.



(a) (b) (c)

Fonte: Adaptado de (RIOS, 2011).

2.3.3 Reconhecimento de Objetos

Após definir a região de interesse da imagem, por meio do processo de segmentação, é necessário descrever e representar este agrupamento de *pixels*. Para isso, utilizam-se as regiões obtidas como entrada de um classificador com o intuito de rotular os objetos (RUSSELL; NORVIG, 2004).

A representação de um objeto é definida a partir dos elementos que serão levados em consideração no momento da avaliação. Segundo GONZALES e WOODS (2000), esta representação pode ser realizada através de elementos internos ou externos.

> Geralmente, uma representação externa é escolhida quando a atenção primária estiver voltada para características de forma. Por outro lado, uma representação interna é selecionada quando a atenção estiver voltada para propriedades como cor ou textura. (GONZALES; WOODS, 2000, p. 345).

2.3.3.1 Reconhecimento Baseado em Brilho

Após a segmentação da imagem é extraído uma região de interesse. Esta nada mais é do que um subconjunto de *pixels* da imagem original, do qual as características podem ser definidas como os próprios valores brutos de brilho do pixel.

O principal problema enfrentado na aplicação desta técnica deriva da redundância intrínseca aos dados. *Pixels* que, por exemplo, estejam em um ponto central característico do objeto tenderão a possuir uma alta correlação devido a luminosidade, geometria entre outros (RUSSELL; NORVIG, 2004). Logo, a taxa de processamento será demasiadamente elevada para *pixels* que tenderão a fornecer os mesmos resultados.

Para solucionar tal problema, atualmente tem se incrementado técnicas de redução de dados ao reconhecimento baseado em brilho para que o processamento das imagens possa ser realizado com maior velocidade.

2.3.3.2 Reconhecimento Baseado em Características

Esta técnica se caracteriza pelo uso de regiões contornos para identificar a forma. Este método possui duas grandes vantagens quanto ao reconhecimento baseado no brilho: a primeira delas refere-se à quantidade de dados processados enquanto a segunda se caracteriza pela invariância na iluminação (RUSSELL; NORVIG, 2004).

A quantidade de dados processados refere-se justamente à natureza dos mesmos. Devido à quantidade de arestas presente na região de interesse ser muito menor que a quantidade de *pixels*, o custo de processamento torna-se menor, fazendo com que o tempo de resposta do algoritmo seja mais rápido.

A invariância na iluminação é uma característica que permite a este método identificar o objeto em cena sem a necessidade de uma configuração de iluminação precisa. Desde que a imagem possua um índice de contraste adequado, as arestas serão detectadas nas mesmas posições da imagem apresentada.

Como os contornos são característicos de um objeto, o uso dos mesmos torna possível detectar qual o objeto presente na imagem. Para isto, dada às arestas encontradas, basta comparar o segmento com uma biblioteca de visualizações através de um classificador de vizinhos mais próximos.

3 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 (2022) 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 (2021) 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%.

Utilizando imagens oriundas da base de dados *PlantVillage*, o trabalho de Carvalho e Zoby (2018) 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 (2019), 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 (2020), 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 1 demonstra de forma resumida as tecnologias e dados utilizados, além dos resultados obtidos.

Tabela 1 – Resumo dos trabalhos de identificação de doenças em plantas baseadas em *deep learning*.

Trabalho	**Modelo**	**Dataset**	**Descrição**	**Acurácia**
Rosa (2022)	Mobilenet_V2 Inception_V3	PlantVillage Digipathos (52.869 imagens)	60 doenças e 6 espécies	80.04% - 99.69%
Leite (2021)	YoloV3	Kaggle (1.820 imagens)	2 doenças e 1 espécie	67%
Carvalho e Zoby (2018)	RNA Própria MobileNet_V2 Inception_V3 ResNet_V2 NASNet	PlantVillage (4.485 imagens)	4 doenças e 1 espécie	95%
Funck (2019)	Inception_V3	PlantVillage (2.770 imagens)	5 doenças e 1 espécie	84.61%
Silva e Schimiguel (2020)	RNA Própria	EMBRAPA (39.155 imagens)	43 doenças	85%

Fonte: De autoria própria.

Em resumo, os trabalhos apresentados por Carvalho e Zoby (2018), Rosa (2022) e Funck (2019) demonstraram a eficácia na utilização de modelos pré-treinados, como *Mobilenet V2* e *Inception V3*, na classificação correta das doenças apresentadas. O trabalho de Silva e Schimiguel (2020) demonstraram eficiência nos resultados ao concentrar-se apenas na doença, desconsiderando a espécie da planta e, por fim, pode-se falar de Leite (2021) que 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.

O presente trabalho buscou diferenciar-se dos mencionados anteriormente ao ofe-

recer 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.

4 Infraestrutura e Tecnologias

Neste capítulo, serão apresentadas as tecnologias e infraestrutura utilizadas no desenvolvimento da aplicação móvel KAWARI, da API e da Rede Neural Artificial. Serão detalhadas as linguagens de programação, *frameworks* e bibliotecas utilizadas no projeto. Serão abordados os motivos que levaram a seleção destes componentes, tendo como ênfase os benefícios e vantagens proporcionados ao projeto.

4.1 *Visual Studio Code*

O *Visual Studio Code*[1] é um editor de código-fonte desenvolvido pela *Microsoft*. Ele roda nos sistemas operacionais *Windows*, *macOS* e *Linux* e possui suporte integrado para *JavaScript*, *TypeScipt* e *Node.js*. O mesmo ainda possui suporte para depuração, realce de sintaxe, refatoração de código, *snippets* e controle de versionamento *Git* incorporado. Por tudo o que já foi citado e por ser leve, rápido, altamente personalizável, possuir ferramentas de depuração integradas, além de possuir um extenso número de extensões disponíveis, este foi escolhido para a codificação do projeto.

4.2 Linguagens de Programação

Para o desenvolvimento da aplicação e construção da rede neural artificial foram utilizadas as seguintes linguagens de programação.

- ***Python*:** linguagem de programação amplamente utilizada para o desenvolvimento de softwares, ciência de dados e *machine learning*. Por sua grande versatilidade e facilidade de uso, acabou se tornando umas das linguagens mais comuns atualmente. Esta linguagem foi selecionada para codificação da RNA por contar com bibliotecas robustas e amplamente utilizadas para aprendizado de máquinas, simplificando a construção e treinamento dos modelos;
- ***TypeScript*:** desenvolvido pela *Microsft*, esta é uma linguagem de programação de código aberto que adiciona ao *JavaScript* a tipagem estática opcional e recursos avançados que não estavam presentes de maneira nativa nesta. Esta foi utilizada na contrução do *Front* e *Back-End* por oferecer maior segurança, robustez e ajudar na prevenção de erros de programação;
- ***HTML*:** linguagem de marcação utilizada na construção de páginas estáticas que podem ser interpretadas por navegadores, sendo principalmente utilizada na cons-

[1] https://code.visualstudio.com/

trução de aplicações *web*, sendo utilizada para a construção da estrtura da páginas da aplicação *mobile*;

- ***CSS*:** traduzido para o português, sua sigla significa Folha de Estilo em Cascatas. Facilmente utilizado com linguagens de marcação, esta foi responsável por adicionar estilos as páginas da aplicação;

4.3 *Jupyter Notebook*

Jupyter Notebook[2] é uma ferramenta de código aberto baseada em navegador para desenvolvimento de projetos de ciência de dados. Cada *notebook* é formado por uma coleção de células, estas contém código executável geralmente em linguagem *Python*[3] ou R. As "células de código" podem ser executadas individualmente e a saída, seja ela gráfica ou textual, é apresentada no documento imediatamente abaixo da célula. A simplicidade na execução dos códigos em bloco e visualização dos resultados tornou esta a escolha ideal para o desenvolvimento e treinamento da RNA.

4.4 API

Do inglês *application programming interface* ou interface de programação de aplicações, esta pode ser definida como um conjunto de serviços implementados em um *software* que podem ser utilizados por outros programas ou aplicativos. A comunicação entre as partes é possível pois estas utilizam um conjunto de definições e protocolos que regem a forma como esta é realizada, isto simplifica o desenvolvimento de aplicações uma vez que não é necessário conhecer a lógica interna da API. Todos estes pontos, somados a facilidade que esta oferece na manutenção, modularidade e escalabilidade do sistema, fizeram desta a escolha ideal para implantação neste projeto.

4.5 *SQLite*

O *SQLite*[4] é um dos mais conhecidos bancos de dados relacional, isto se dá pelo fato deste ser mais prático e acessível. Este foi escolhido por funcionar como um servidor próprio e independente, fornecendo a API acesso à um banco de dados SQL sem a execução de um SGBD (Sistema Gerenciador de Banco de Dados) separado.

[2] https://www.jupyter.org/
[3] https://www.python.org/
[4] https://www.sqlite.org/

4.6 *Node.js*

O *Node.js*[5] é um ambiente de execução em uma máquina virtual baseado no interpretador V8 do *Google* e que permite a execução de códigos *JavaScript* fora de um navegador *web*. Este consiste em um *software* de código aberto e multiplataforma utilizado na construção de aplicações rápidas e escalonáveis no lado do servidor e em rede. Este foi escolhido para a construção da API por possuir um vasto ecossistema de pacotes no npm que facilitam e acelaram o processo de desenvolvimento, além de permitir utilizar a mesma linguagem de programação (*TypeScript*) utilizada no lado do cliente.

4.7 *Express*

Express.js[6] é um *framework* de código aberto desenvolvido para a construção de aplicações *web* e APIs. Por ser um *framework web* rápido, flexível e minimalista, este se tornou um dos mais populares para o desenvolvimento de aplicativos em *Node.*js. Por simplificar a comunicação entre a API e outras aplicações, fornecendo um série de funcionalidade para gerenciamento de rotas HTTP e criação de *middlewares*, esta foi escolhida para integrar o seu desenvolvimento.

4.8 *React Native*

O *React Native*[7] é um *framework* de código aberto para o desenvolvimento de aplicativos móveis utilizando *JavaScript* e *React*. Este *framework* foi escolhido para o desenvolvimento da aplicação mobile por permitir criar aplicações nativas iOS e *Android*, que se integram com as características nativas de cada plataforma, com uma única base de código, acelerando esta etapa e reduzindo custos.

4.9 *Axios*

Axios[8] é uma biblioteca JavaScript utilizada para realizar requisições HTTP. Esta foi escolhida para ser utilizada no *Front-End* por oferecer suporte a requisições e respostas, manipulação de erros e por ser capaz de lidar com requisições assíncronas. Além disto, esta é de simples implementação e possui vasta documentação.

[5] https://nodejs.org/pt
[6] https://expressjs.com/
[7] https://reactnative.dev/
[8] https://axios-http.com/

4.10 Recursos de Produção

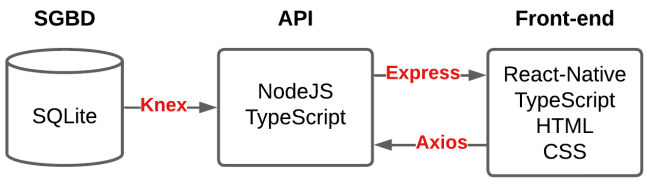
A Figura 7 apresenta, em forma de diagrama, os recursos utilizados no projeto e suas relações. Para armazenamento dos dados, foi utilizado o SQLite (Seção 4.5) por ser um sistema gerenciador de banco de dados (SGBD) com um banco de dados relacional leve e embutido. A sua integração com a API foi realizada através do *Knex*. Esta é uma ferramenta desenvolvida para o *Node.js* e permite a execução de comandos SQL em linguagem *JavaScript* e *TypeScript*.

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*. Sua forma de funcionamento pode ser melhor observada na Seção 4.4. Foi utilizado o Node.js (Seção 4.6) em conjunto com a linguagem de programação *TypeScript* (Seção 4.2) para a contrução da API.

A comunicação entre *Front* e *Back-End* é gerenciada pelo *Axios* e *Express*, conforme descrito nas Seções 4.9 e 4.7. O *Express* opera no lado da API e é responsável pelo gerenciamento de rotas, textitmiddlewares 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 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 crição da interface com usuário, estes conferiram estrutura e estilo as telas desenvolvidas. Todas estas tecnologias podem ser analisadas de forma individual nas Seções 4.8 e 4.2.

Figura 7 – Diagrama dos recursos de produção.



Fonte: De autoria própria.

4.11 *Android SDK Build Tools*

O *Android SDK Build Tools*[9] é uma ferramenta de emulação de dispositivos baseada no QEMU, um emulador e virtualizador de máquina genérico e de código aberto. Este oferece ao desenvolvedores uma ferramenta para compilarem e testarem seus aplicativos em ambiente real de execução *Android*. Este foi utilizado para emular um dispositivo *Android* que pudesse executar a aplicação *mobile*

[9] https://developer.android.com/

5 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 *link* presente no Apêndice A.

5.1 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*[1], 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*[2], que é uma biblioteca de *deep learning*, aprendizado de máquina, conhecida por sua simplicidade. Algumas outras dependências são as bibliotecas *pandas*[3] versão 1.5.3, *sklearn*[4] versão 1.2.2, *seaborn*[5] versão 0.12.2, *numpy*[6] 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://www.tensorflow.org
[2] https://keras.io/
[3] https://pandas.pydata.org/
[4] https://scikit-learn.org/stable/
[5] https://seaborn.pydata.org/
[6] https://numpy.org/

5.2 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*[7]. 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 categorias de doenças e quatro especies vegetais distintas, conforme apresentado na Tabela 2.

Tabela 2 – *Dataset* inicial.

Classe	**Número de instâncias**
cereja_oidio	1.052
maca_ferrugem	275
maca_podridao	621
maca_sarna	630
morango_queimadura_folha	1109
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 7.326 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.

5.3 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.

As operações foram realizadas de forma randômica nas instâncias de treinamento através do *Keras*[8], 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.

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

- ***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 no Apêndice B, 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, que inicialmente era de 7.326, foi para 44.468.

A Tabela 3 expressa a divisão destas em seus respectivos grupos. A coluna "Classe" expressa a espécie da planta e a doença analisada, enquanto "Número de instância" diz respeito ao número total de imagens pertencentes a classe específica.

Tabela 3 – *Dataset* final.

Classe	**Número de instâncias**
cereja_oidio	6.367
maca_ferrugem	1.722
maca_podridao	3.756
maca_sarna	3.903
morango_queimadura_folha	6.721
uva_mancha_isariopsis	6.512
uva_podridao_negra	7.135
uva_sarampo_negro(esca)	8.352

Fonte: De autoria própria.

Por fim, para garantir um treinamento com resultados satisfatórios, as imagens foram divididas em três grupos com as seguintes proporções:

- 80% das imagens reservadas para o treinamento da rede neural;
- 15% das imagens reservadas para teste do aprendizado da RNA;
- 5% das imagens reservadas para validação do modelo gerado.

5.4 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 composta 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 setes vezes até chegarmos as camadas de saída.

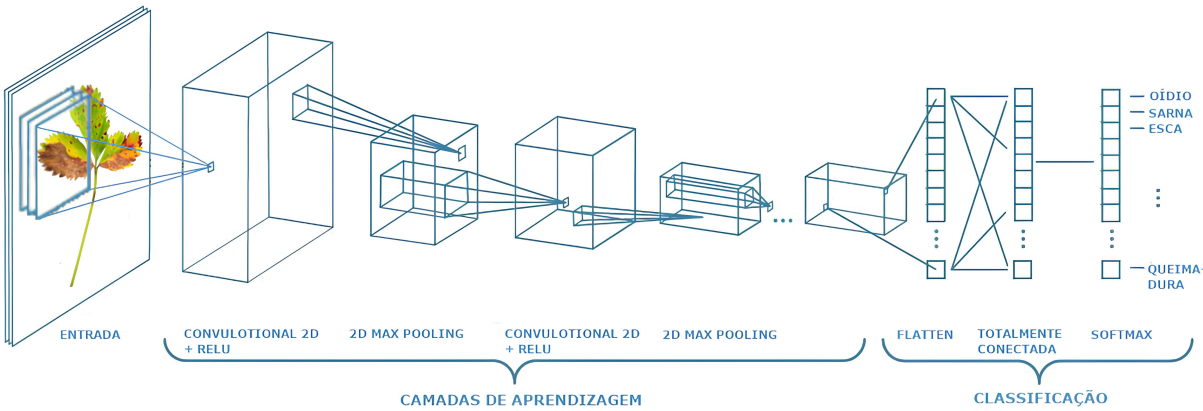
A Camada Convolucional 2D é composta por 8 *kernels* de tamanho 3x3 e utiliza a função de ativação ReLu. A medida que as camadas se repetem, a quantidade de *kernels* é dobrada e seu tamanho alterna entre 3x3 e 5x5 para que a rede capture recursos em múltiplas escalas.

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 a Camada *MaxPooling 2D* onde o objetivo é reduzir a amostragem de uma representação de entrada através da extração e agrupamento das caraterísticas contidas nas sub-regiões da imagem. Por fim, é aplicada uma camada de *Dropout* onde será definido uma fração dos dados de entrada como 0 durante o treinamento, prevenindo assim o overfitting. A 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*. A estrutura da RNA pode ser observada através da Figura 8.

Figura 8 – Estrutura da RNA



Fonte: Adaptado de Raghav (2018).

Ressaltando que a quantidade de neurônios na última camada deve ser igual a quantidade de classes possíveis. Esta rede gerou um total de 19.044.810 parâmetros treináveis. Um resumo das camadas do modelo de treinamento pode ser observado no Apêndice C.

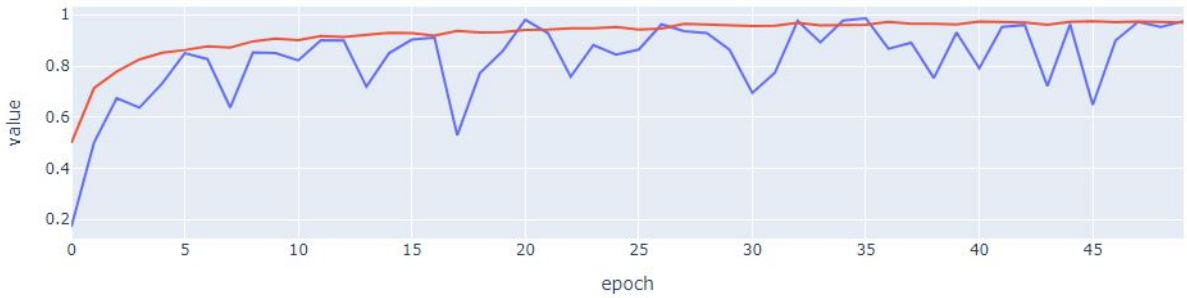
5.5 Treinamento e Avaliação do Modelo

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 5.4, foi repetido por 50 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: uma para acurácia (*accuracy*) e outro para perca (*loss*), podendo estes serem visualizados nas Figuras 9 e 10, 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 teste.

Figura 9 – Gráfico Precisão Modelo



Fonte: De autoria própria.

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

Figura 10 – Gráfico Perca do Modelo

Fonte: De autoria própria.

Com o passar das épocas, a perca diminuiu e a precisão 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 apresentado na fase de teste do modelo foi registrado na época 35, sendo este de 0.0488. Nesta mesma época, o modelo também obteve a maior precisão, alcançando 0.9863, ou 98,63%. A acurácia na fase de teste foi o critério utilizado para selecionar o modelo, e por isso, este foi o escolhido. Os valores das métricas de acurácia (*Acc.*) e perda (*Loss*) das bases de treino e validação para a época 35 podem ser observados na Tabela 4.

Tabela 4 – Valores de métricas na época 35.

Época	**Acc. Treino**	**Loss Treino**	**Acc. Validação**	**Loss Validação**
35	0.9609	0.1400	0.9863	0.0488

Fonte: De autoria própria.

5.6 Métricas de Classificação do Modelo

O relatório de classificação da rede neural, presente na Figura 11, exibe algumas informações que podem ser analisadas para obter um maior entendimento do modelo gerado. Essas métricas são fundamentais para avaliar o desempenho da rede em tarefas de classificação, equilíbrio entre classes e acurácia geral.

A primeira métrica avaliada foi a precisão (*Precision*). Pode-se definir esta como a proporção entre verdadeiros positivos (VP) em relação a soma dos verdadeiros e falsos positivos (VP + FP), a sua fórmula pode ser observada na 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 0.96, ou 96%, pelas classes: maca_sarna, uva_mancha_isariopsis e uva_podridao_negra.

Figura 11 – Relatório de classificação

	precision	recall	f1-score	support
cereja_oidio	0.99	0.99	0.99	163
maca_ferrugem	1.00	0.88	0.94	43
maca_podridao_negra	0.99	0.99	0.99	83
maca_sarna	0.96	0.98	0.97	84
morango_queimadura_folha	1.00	0.99	0.99	190
uva_mancha_isariopsis	0.96	1.00	0.98	180
uva_podridao_negra	0.96	0.97	0.96	186
uva_sarampo_negro(esca)	0.98	0.97	0.98	243
accuracy			0.98	1172
macro avg	0.98	0.97	0.98	1172
weighted avg	0.98	0.98	0.98	1172

Fonte: De autoria própria.

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

Em seguida analisou-se a revocação (*Recall*) (Eq. 7), que mede a proporção entre verdadeiros positivos identificados de forma correta dentre todas os valores de classe positiva esperados. Assim, de forma inversa a precisão (*Precision*), esta é relevante quando considerado os falsos negativos mais prejudiciais que os falsos positivos. Para este parâmetro, o menor valor apresentado pelo modelo de 0.88, ou 88%, sendo esta classe a da maca_ferrugem.

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

Por fim, a última métrica avaliada é a *F1-Score* e sua 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 0.94 e 0.99, 94% e 99% respectivamente, sendo o menor valor obtido o da classe maca_ferrugem.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{8}$$

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 no Apêndice D.

5.7 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[9] 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 as originais para o realização do treinamento.

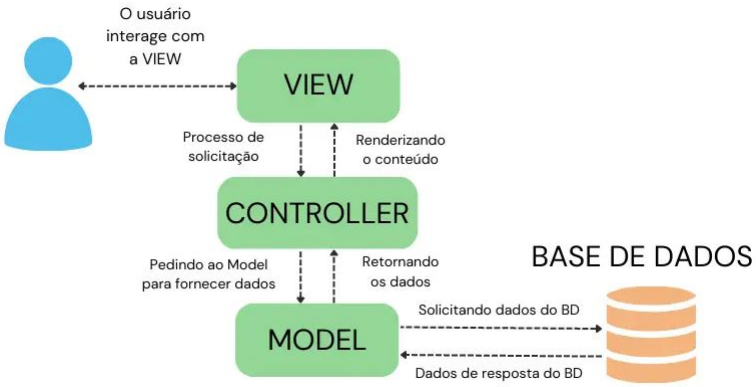
[9] https://www.tensorflow.org/js/guide/conversion

6 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*[1]. Este pode ser acessado através do *link* disponível no Apêndice A. 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 dependência entre estas através da divisão de responsabilidades.

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 (O. . . , 2023). A Figura 12 exibe, de forma ilustrada, a sua forma de funcionamento.

Figura 12 – Arquitetura MVC.



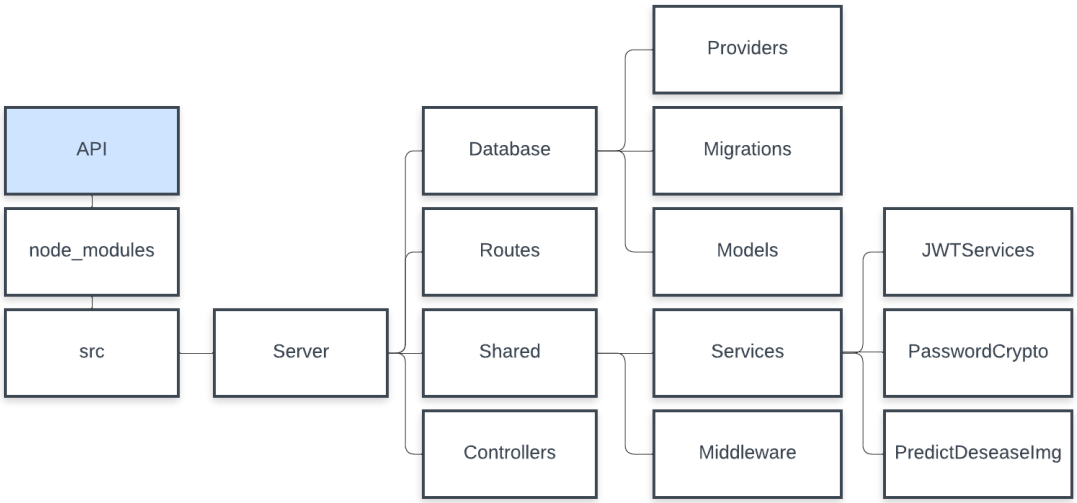
Fonte: (O. . . , 2023).

6.1 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 13 apresenta um diagrama com a estrutura do projeto, onde é possível identificar os módulos da aplicação.

[1] https://github.com

Figura 13 – Estrutura API.



Fonte: De autoria própria.

6.1.1 *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 o versionamento do código. A estrutura deste, criada através das *Migrations*, podem ser observadas através do Diagrama Entidade Relacionamento (DER) presente no Apêndice E.

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*[2].

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] https://knexjs.org/

6.1.2 *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.

6.1.2.1 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 6.1. Em resumo, este código integra as bibliotecas *TensorFlow*.js[3] e *Jimp*[4] para construir uma função que realiza predições de imagens usando um modelo pré-treinado, retornando resultados baseados nos IDs das classes preditas. Esta pose 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, a imagem é convertida para um `TensorFlow` e realizada

[3] https://www.*TensorFlow*.org/
[4] https://www.npmjs.com/package/*Jimp*

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.

```
1  interface PredictRequest extends Request {
2    file: { buffer: Buffer; }
3  }
4
5  const CLASS_ID = [1, 2, 3, 4, 5, 6, 7, 8];
6
7  let model: tf.LayersModel;
8
9  (async () => { model = await loadModel(); }
10
11 const predictDeseaseImg = async (imageBuffer: Buffer):
12 Promise<number | Error > => {
13   try {
14     const image = await \textit{Jimp}.read(imageBuffer);
15     image.resize(256,256).normalize();
16
17     const ima\textit{GET}ensor = tf.tidy(() => {
18       const buffer = image.bitmap.data;
19       const imgTensor = tf.tensor3d(new Uint8Array(buffer),
20                         [image.bitmape.height, image.bitmap.width, 4]);
21       const resized = imgTensor.slice([0, 0, 0], [256, 256, 3]).
22                         div(tf.scalar(255));
23       return resized.expandDims(0);
24     });
25
26   const predictions = model.predict(ima\textit{GET}ensor) as tf.Tensor;
27   const predictClassId = CLASS_ID[predictions.argMax(1).dataSync()[0]];
28
29   return predctedClassId;
30   } catch(error) {
31     return new Error('Erro durante a predicao da imagem');
32   }
33 }
```

Código 6.1 – Função PredictDeseaseImg.

6.1.2.2 *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[5], 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.

[5] https://www.npmjs.com/package/jsonwebtoken

7 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* e pode ser acessado através do *link* disponível no Apêndice A. 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.

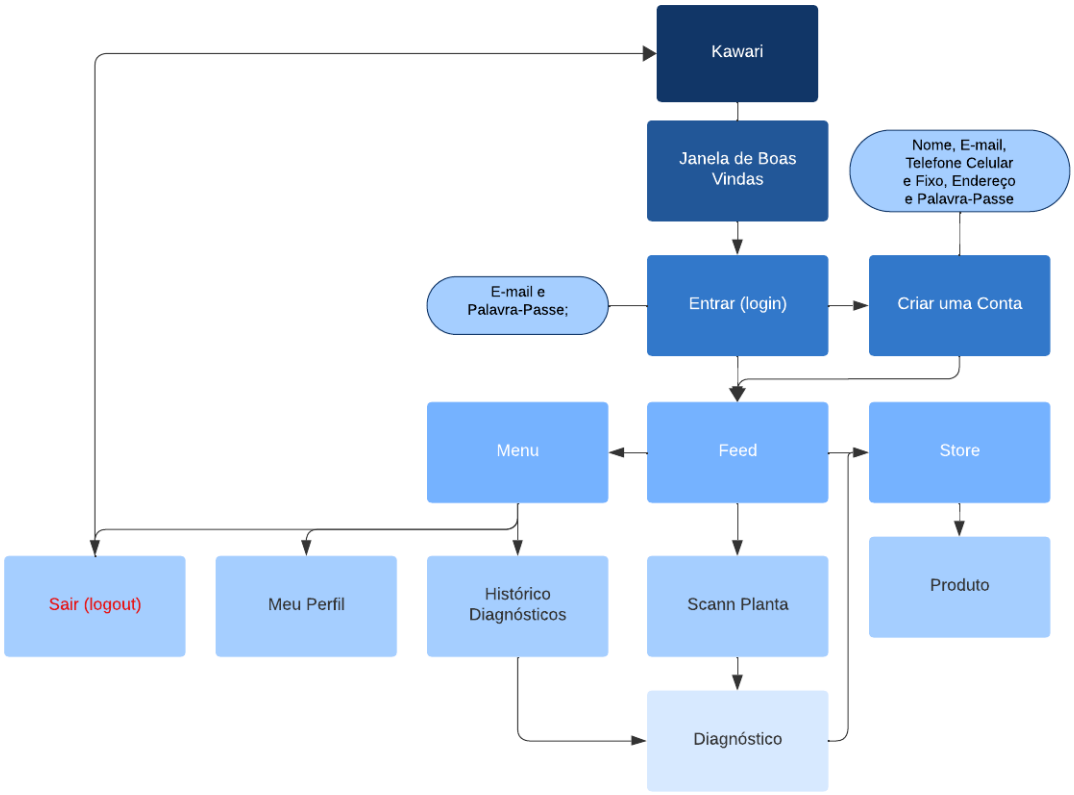
Utilizando o *token* JWT, a aplicação agora pode acessar *endpoints* protegidos pelo *backend*. Através da implementação da biblioteca *Axios*[1], é possível realizar requisições *GET, POST, PUT, DELETE,* entre outras, para buscar, enviar, atualizar ou executar outras operações necessárias para a aplicação. Para manter um estado global coeso, foi utilizado contextos para gerenciar as informações compartilhadas entre componentes.

7.1 Fluxo da Aplicação

De um modo geral, a organização lógica da aplicação ficou distribuída entre três eixos: entrada através do *login*, avaliação e diagnostico 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.

[1] https://www.npmjs.com/package/axios

Figura 14 – Fluxo da Aplicação *Kawari*.



Fonte: De autoria própria.

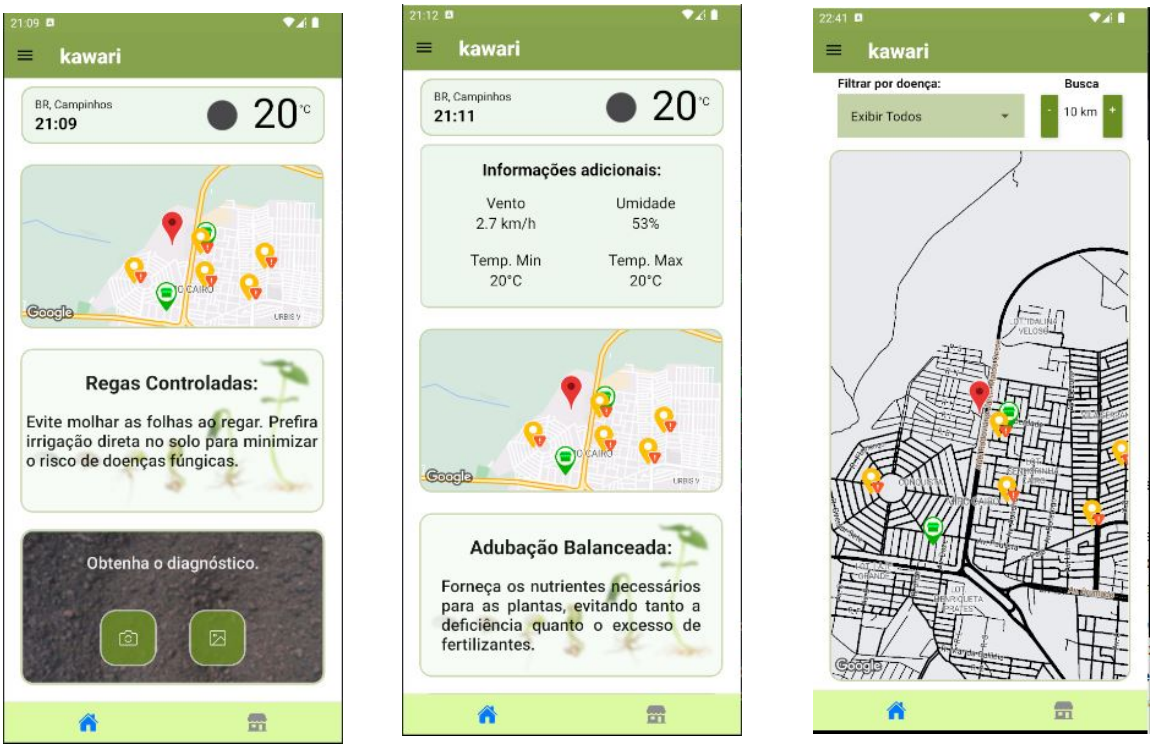
7.2 Página *Home*

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 exibi 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.

Figura 15 – Tela Home.



(a) (b) (c)

Fonte: De autoria própria.

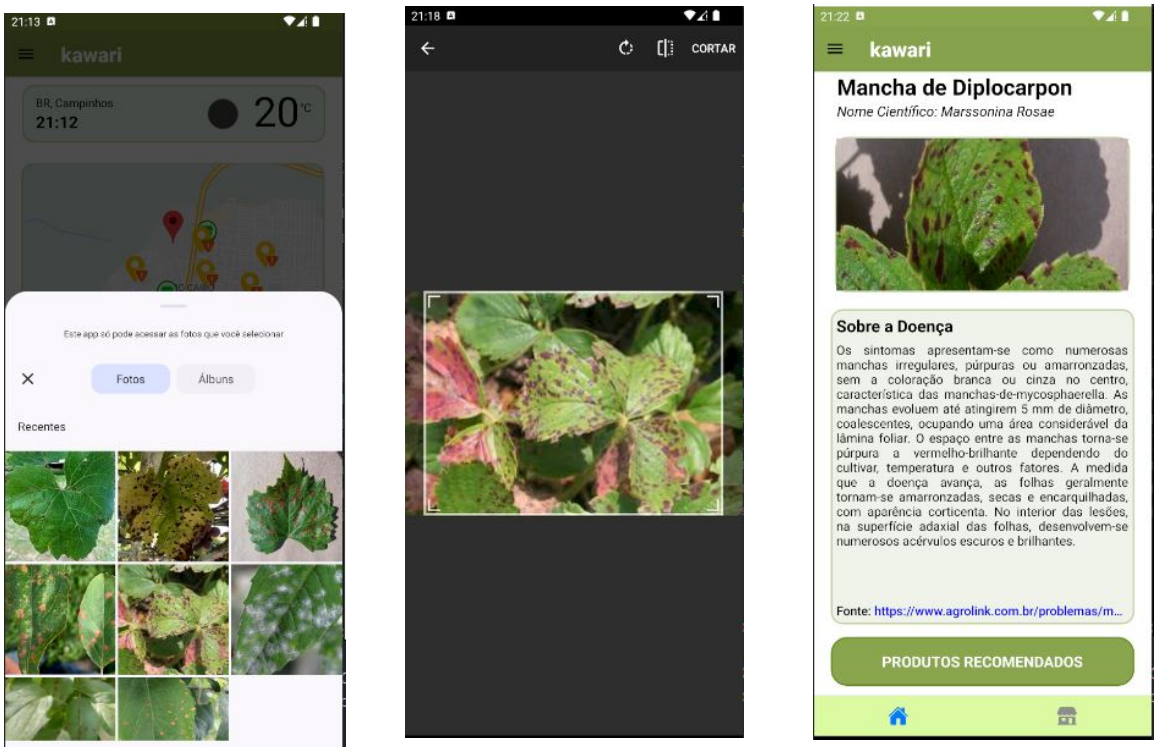
7.3 Tela de Diagnóstico

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).

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 Diagnostico, conforme apresentado na Figura 16 (c), onde será exibido 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.

Figura 16 – Diagnóstico de Doenças em Plantas.



(a) (b) (c)

Fonte: De autoria própria.

7.4 *Store*

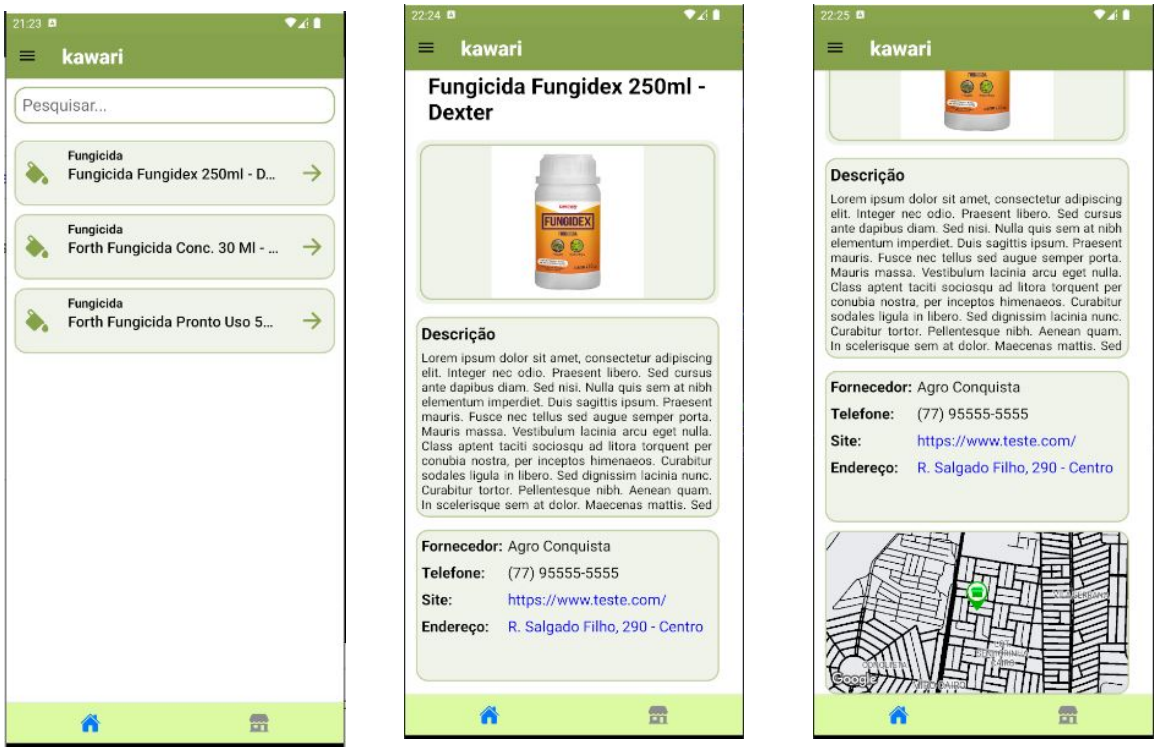
A tela da loja (*Store*) é responsável por exibir os produtos oferecidos pelos fornecedores de produtos agrícolas parceiros. Ela pode ser acessada de duas maneiras distintas na aplicação. A primeira forma é por meio da *Tab Bar*, situada na parte inferior da tela, através do ícone de loja. A segunda 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.

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.

Figura 17 – Loja da Aplicação.



(a) (b) (c)

Fonte: De autoria própria.

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, conforme ilustrado na Figura 17 (c).

8 Ameaças a Validade

Foi necessário muita dedicação e trabalho para desenvolver um aplicativo móvel capaz de diagnosticar doenças foliares em plantas utilizando redes neurais. Um dos inúmeros desafios encontrados ao longo deste projeto foi o treinamento da RNA. Devido à velocidade das máquinas disponibilizadas pelo *Google Colab*[1] e o custo para utilização de GPUs, visto que é necessário pagar para fazer uso deste recurso, o aprendizado foi realizado em ambiente local. Assim, devido a limitação do poder computacional, houve um aumento significativo no tempo necessário para realizar o treinamento. Devido a isto, tornou-se necessário a otimização do código e paciência para a obtenção de um modelo eficiente.

Além disso, disponibilizar a API em uma plataforma online apresentou alguns desafios. A infraestrutura necessária para disponibilizá-la em um ambiente web se mostrou muito cara, inviabilizando sua operação em servidores externos. Dessa forma, a API foi executada em ambiente local, o que não impactou o desenvolvimento do projeto, mas acabou limitando a capacidade de acesso e escalabilidade do sistema.

Outro ponto a ser abordado foi a dificuldade em encontrar uma API que fornecesse, de modo gratuito, dados meteorológicos, visto que estes são essenciais para aplicação, uma vez que as condições climáticas possuem impacto direto no manejo e saúde das plantações. Após muita pesquisa, foi escolhida a API *OpenWeather*[2] por oferecer dados confiáveis e atender aos critérios pré-estabelecidos.

A construção deste aplicativo exigiu uma abordagem técnica, com boa gestão dos recursos e expectativas. Os inúmeros obstáculos enfrentados durante este projeto foram superados com estudo, dedicação e metodologia, resultando em uma ferramenta capaz de auxiliar pequenos agricultores a diagnosticar doenças foliares em plantas.

[1] https://colab.research.google.com/
[2] https://openweathermap.org/api

9 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 do TCC.

9.1 Trabalhos Futuros

Devido à indisponibilidade de tempo, algumas implementações que poderiam ter agregado significativamente a qualidade de projeto não foram implementadas. Contudo, nesta seção abordaremos algumas destas para que em projetos e trabalhos futuros estas possam ser executadas.

- **Divisão do Aprendizado:** o treinamento da rede neural foi realizado utilizando imagens contendo doenças e espécies de plantas distintas. Desta forma, 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, uma sugestão de melhoria seria a divisão do treinamento da rede neural por espécie para que esta obtenha uma maior acurácia;
- **Combinação de Resultados:** a identificação das doenças foram feitas utilizando como base apenas as imagens das folhas doentes, assim, doenças que possuem características semelhantes podem ser mais difíceis de serem reconhecidas pela rede neural. Para contornar este problema e obter resultados mais precisos, o ideal seria

realizar o treinamento das folhas, frutos e caule das plantas. Tal treinamento deve ser realizado de forma separada e o resultado final deve refletir a combinação da predição apresentado por estes, trazendo assim mais segurança e precisão quanto ao diagnóstico;

- **Perfil do Fornecedor:** neste projeto não foi possível realizar a implementação do perfil de fornecedor, entretanto, por ser uma parte vital para utilização da aplicação, esta deve ser implementada futuramente para que os fornecedores possam cadastrar os produtos agrícolas.

- **Ambiente Real:** a aplicação do sistema em ambiente real e com espécies vegetais que possuam relevância na agricultura local, como café e manga, permitirá obter um estudo de caso mais preciso e com maior impacto sobre a comunidade.

Referências

BORTH, M. R.; IACIA, J. C.; PISTORI, H.; RUVIARO, C. F. A visão computacional do agronegócio: Aplicações e direcionamentos. **7º Encontro Científico de Administração, Economia e Contabilidade (ECAECO)**, 2014. Disponível em: <http://www.gpec.ucdb.br/pistori/publicacoes/borth_ecaeco2014.pdf>. Acesso em: 11 abr 2024.

CARVALHO, R. G.; ZOBY, L. T. Convolutional neural networks for leaf disease classification. In: SBC. **Anais do XV Encontro Nacional de Inteligência Artificial e Computacional**. [S.l.], 2018. p. 332–342.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. L. F. **INTELIGÊNCIA ARTIFICIAL**: uma abordagem de aprendizado de máquina. Rio de Janeiro: LTC, 2011. 394 p.

FILHO, O. M.; NETO, H. V. **PROCESSAMENTO DIGITAL DE IMAGENS.** Rio de Janeiro: Brasport, 1999. 307 p.

FONSECA, S. M.; MASSRUHÁ, S.; LEITE, A. de A.; BOLFE Édson L. **AGRO 4.0**: O papel da pesquisa e perspectivas para a transformação digital na agricultura. Rio de Janeiro: Autografia, 2023. 58-77 p.

FREAR AS PRAGAS E AS DOENÇAS DAS PLANTAS: especialistas planejam medidas a nível global. **FAO**, 2015. Disponível em: <https://www.fao.org/brasil/noticias/detail-events/pt/c/293049/>. Acesso em: 11 abr 2024.

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

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

GONZALES, R. C.; WOODS, R. E. **PROCESSAMENTO DIGITAL DE IMAGENS.** São Paulo: Blucher, 2000. 528 p.

HAYKIN, S. **REDES NEURAIS**: Princípios e prática. 2. ed. São Paulo: Bookman, 2001. 900 p.

JANG, J. R.; SUN, C.; MIZUTANI, E. **NEURO-FUZZY AND SOFT COMPUTING**: A computational approach to learning and machine intelligence. Londres: Prentice-Hall, 1997. 607 p.

LEITE, T. d. M. **Deep learning em dois estágios para detecção e classificaçãode doenças em folhas de plantas com aplicação em dispositivos móveis**. Dissertação (Mestrado) — Universidade de São Paulo, 2021.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, n. 4, p. 115–133, 1943.

MITCHELL, T. M. **MACHINE LEARNING**. 1. ed. New York: McGraw-Hill Education, 1997. 432 p.

O que é MVC?: Entenda arquitetura de padrão mvc. **Usando py**, 2023. Disponível em: <https://www.usandopy.com/pt/artigo/o-que-e-mvc-entenda-arquitetura-de-padrao-mvc/>. Acesso em: 15 jul 2024.

OCDE-FAO PERSPECTIVAS AGRÍCOLAS 2023-2032. **OECD/FAO**, Paris, p. 391, 2023. Disponível em: <https://doi.org/10.1787/2ad6c3ab-es>. Acesso em: 11 abr 2024.

OLIVEIRA, C. M.; AUADA, A. M.; MENDES, S. M.; FRIZZAS, M. R. Crop losses and the economic impact of insect pests on brazilian agriculture. **Crop Protection**, v. 56, p. 50–54, 2014. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S026121941300269X>. Acesso em: 11 abr 2024.

RAGHAV, P. **Understanding of Convolutional Neural Network (CNN)**: Deep learning. 2018. Disponível em: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148>. Acesso em: 20 ago 2024.

RIOS, L. R. S. Visão computacional. **Departamento de Ciência da Computação – Universidade Federal da Bahia**, Salvador, p. 64, 2011. Disponível em: <https://bdm.ufpa.br/jspui/bitstream/prefix/1329/1/TCC_VisaoComputacionalAplicada.pdf>. Acesso em: 2024-04-09.

ROSA, E. N. Utilização de redes neurais para reconhecimento de doenças foliares em culturas agrícolas. **Universidade Federal de Santa Catarina**, Curitibanos, SC., 2022.

ROSE, D. C.; CHILVERS, J. Agriculture 4.0: Broadening responsible innovation in an era of smart farming. **Frontiers in Sustainable Food Systems**, v. 2, 2018. Disponível em: <https://doi.org/10.3389/fsufs.2018.00087>. Acesso em: 11 abr 2024.

RUSSELL, S.; NORVIG, P. **INTELIGÊNCIA ARTIFICIAL**: Tradução da segunda edição. 1. ed. Rio de Janeiro: Elsevier, 2004. 1021 p.

SACOMANO, J. B.; GONÇALVES, R. F.; SILVA, M. T. da; BONILLA, S. H.; SATYRO, V. C. **INDUSTRIA 4.0**: Conceitos e fundamentos. 1. ed. São Paulo: Edgard Blucher, 2018. 181 p.

SCHWAB, K. **A QUARTA REVOLUÇÃO INDUSTRIAL.**: uma abordagem de aprendizado de máquina. 1. ed. [S.l.]: Edipro, 2018. 160 p.

SHAPIRO, L.; STOCKMAN, G. **COMPUTER VISION**. 1. ed. Londres: Prentice Hall, 2001. 608 p.

SILVA, I. N. d.; SPATTI, D. H.; FLAUZINO, R. A. **Redes neurais artificiais para engenharia e ciências aplicadas**. São Paulo: Artliber Editora, 2010. 399 p.

SILVA, M. J.; SCHIMIGUEL, J. Identificação de doenças em plantas por meio de processamento de imagens: redes neurais convolucionais como auxílio à agricultura. **Revista de Ubiquidade**, v. 3, n. 1, p. 91–111, 2020.

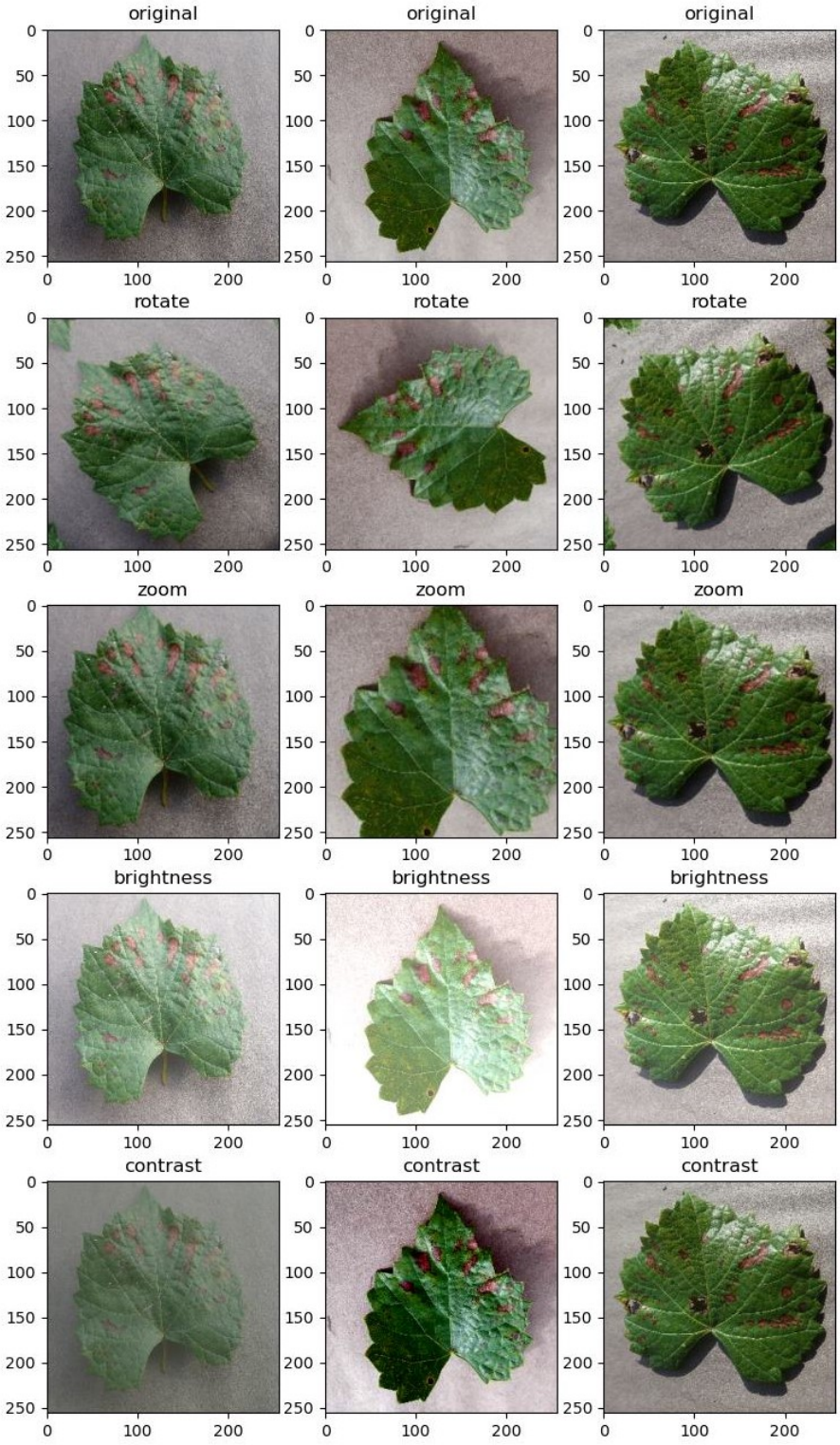
Apêndices

APÊNDICE A – Links da Aplicação.

RNA	RNA	
API	API-PLANT-NODE-TS	
Front-End	FRONT-PLANT-REACT-EXPO	
Aplicação	KAWARI	

Fonte: De autoria própria.

APÊNDICE B – Aplicação de transformações aleatórias a imagens



Fonte: De autoria própria.

APÊNDICE C – Rede Neural.

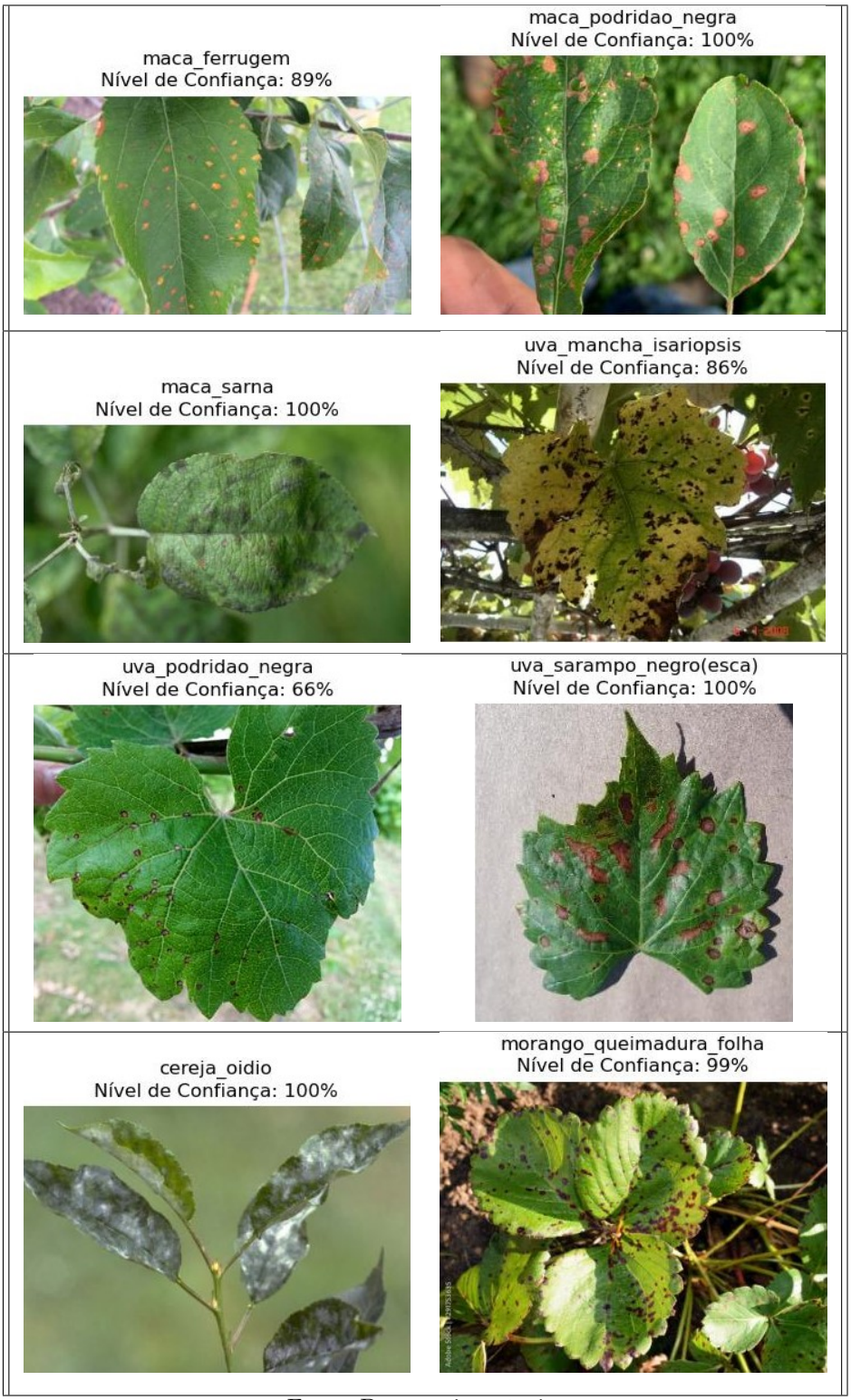
Model: "sequential_2"

Layer (type)	Output Shape	Param #
sequential_3 (Sequential)	(None, 256, 256, 3)	0
conv2d_7 (Conv2D)	(None, 256, 256, 8)	224
batch_normalization_7 (BatchNormalization)	(None, 256, 256, 8)	32
max_pooling2d_7 (MaxPooling2D)	(None, 128, 128, 8)	0
dropout_8 (Dropout)	(None, 128, 128, 8)	0
conv2d_8 (Conv2D)	(None, 128, 128, 16)	3,216
batch_normalization_8 (BatchNormalization)	(None, 128, 128, 16)	64
max_pooling2d_8 (MaxPooling2D)	(None, 64, 64, 16)	0
dropout_9 (Dropout)	(None, 64, 64, 16)	0
conv2d_9 (Conv2D)	(None, 64, 64, 32)	4,640
batch_normalization_9 (BatchNormalization)	(None, 64, 64, 32)	128
max_pooling2d_9 (MaxPooling2D)	(None, 32, 32, 32)	0
dropout_10 (Dropout)	(None, 32, 32, 32)	0
conv2d_10 (Conv2D)	(None, 32, 32, 64)	51,264
batch_normalization_10 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_10 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout_11 (Dropout)	(None, 16, 16, 64)	0
conv2d_11 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_11 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_11 (MaxPooling2D)	(None, 8, 8, 128)	0
dropout_12 (Dropout)	(None, 8, 8, 128)	0
conv2d_12 (Conv2D)	(None, 8, 8, 256)	819,456
batch_normalization_12 (BatchNormalization)	(None, 8, 8, 256)	1,024
max_pooling2d_12 (MaxPooling2D)	(None, 4, 4, 256)	0
dropout_13 (Dropout)	(None, 4, 4, 256)	0
conv2d_13 (Conv2D)	(None, 4, 4, 512)	1,180,160
batch_normalization_13 (BatchNormalization)	(None, 4, 4, 512)	2,048
max_pooling2d_13 (MaxPooling2D)	(None, 2, 2, 512)	0
dropout_14 (Dropout)	(None, 2, 2, 512)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_2 (Dense)	(None, 2048)	4,196,352
dropout_15 (Dropout)	(None, 2048)	0
dense_3 (Dense)	(None, 8)	16,392

Total params: 19,044,810 (72.65 MB)
Trainable params: 6,347,592 (24.21 MB)
Non-trainable params: 2,032 (7.94 KB)
Optimizer params: 12,695,186 (48.43 MB)

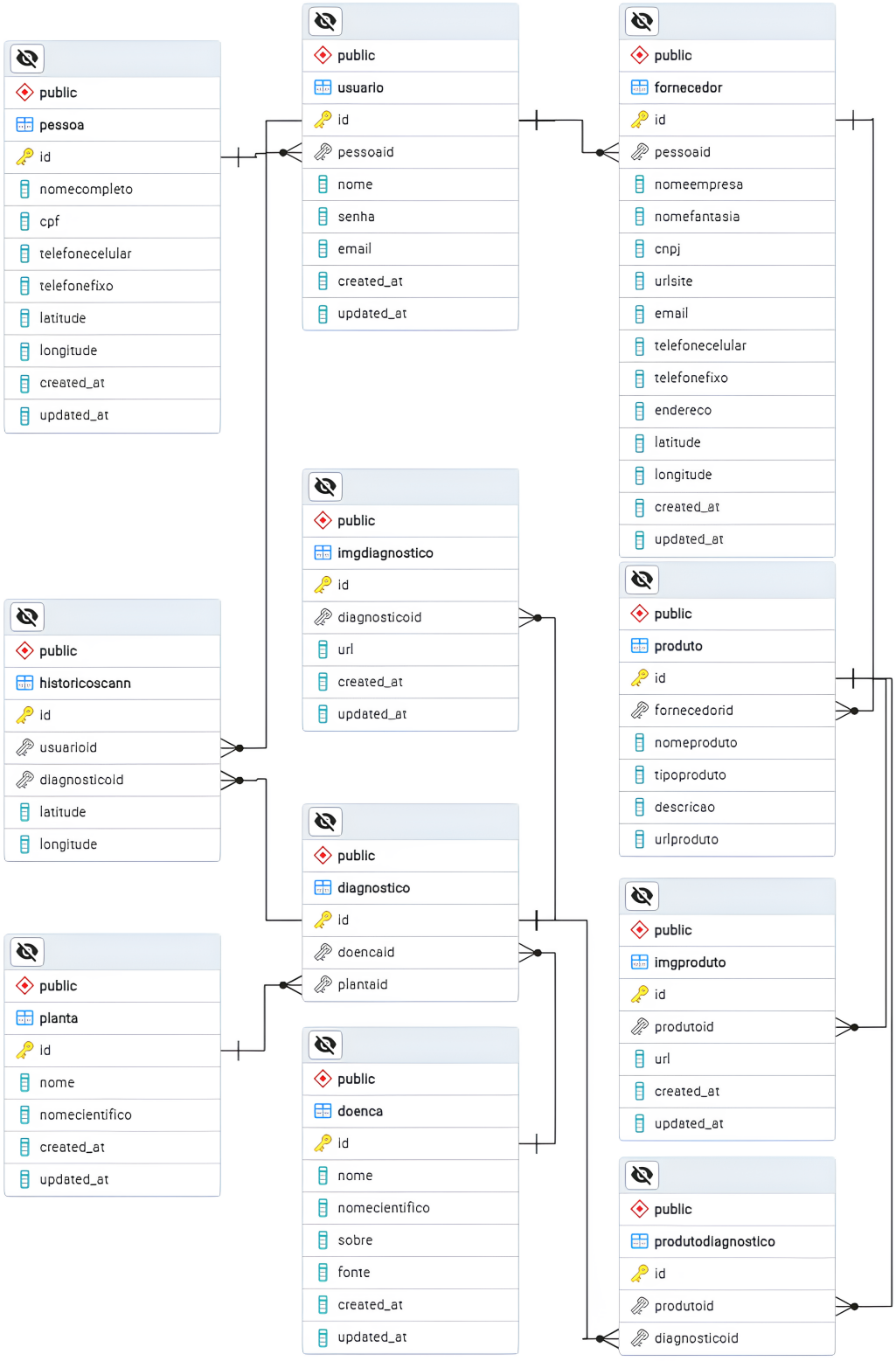
Fonte: De autoria própria.

APÊNDICE D – Predição de Imagens.



Fonte: De autoria própria.

APÊNDICE E – Diagrama Entidade Relacionamento (DER).



Fonte: De autoria própria.