CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

ETEC DA ZONA LESTE

Mtec Desenvolvimento de Sistemas AMS

Enzo Costa Paz

Felipe Vieira de Oliveira

Gustavo de Souza Morais

Sakiri Moon Payao Cestari

AGRO-G.E.S.F: Sistema Inteligente de Monitoramento e Prevenção de Pragas em Cultivos em Linha

São Paulo

2025

Enzo Costa Paz

Felipe Vieira de Oliveira

Gustavo de Souza Morais

Sakhir Moon Payao Cestari

AGRO-G.E.S.F: Sistema Inteligente de Monitoramento e Prevenção de Pragas em Cultivos em Linha

Trabalho de conclusão de curso apresentado ao Curso Técnico em Desenvolvimento de Sistemas da ETEC Zona Leste, orientado pelo Prof. Esp. Jeferson Roberto de Lima, como requisito parcial para obtenção do título de técnico em Desenvolvimento de Sistemas.

AGRO-G.E.S.SF

Sistema Inteligente de Monitoramento e Prevenção de Pragas em Cultivos em Linha

Enzo Costa Paz

Felipe Vieira de Oliveira

Gustavo de Souza Morais

Sakiri Moon Payao Cestari

Aprovada em __/__/____.

BANCA EXAMINADORA

Prof. Esp. Jeferson Roberto de Lima

Universidade do Jeferson

Prof. (Professor avaliador) Universidade do Avaliador Prof. (Professor avaliador) Universidade do Avaliador

São Paulo

2025

“A natureza é o único
livro que oferece um
conteúdo valioso em
todas as suas folhas”

Johann Goethe

RESUMO

O atual trabalho propõe se ao desenvolvimento de um sistema embarcado para o monitoramento de possíveis sinais de pragas em plantações no formato de linha, contendo em si a documentação escrita de tal. Esse sistema contará com uma câmera, bem como a utilização de uma SBC (Single Board Computer) para o processamento local da imagem. O dispositivo visa ajudar os pequenos agricultores que vêm sofrendo com a identificação tardia de pragas e doenças em suas plantações rasteiras, perdendo uma grande quantidade de mercadoria, e forçando o uso excessivo de agrotóxicos, que afetam a saúde do produtor, do produto e do consumidor final. Todo o projeto foi desenvolvido com base na metodologia qualitativa, onde se realiza uma análise mais detalhada e ouve depoimentos dos usuários e compara números. O projeto acredita que com o desenvolvimento do dispositivo, os pequenos agricultores poderão fazer uma identificação mais rápida, reduzindo a perda de produto, o que reduz o uso de agrotóxicos e que consequentemente melhora a saúde do pequeno agricultor, da plantação e do produto.

Palavras-chave: Python; Visão computacional; Machine Learning; Praga.

ABSTRACT

The current work proposes the development of an embedded system for monitoring possible signs of pests in row crops, including written documentation of such. This system will include a camera, as well as the use of an SBC (Single Board Computer) for local image processing. The device aims to help small farmers who have been suffering from the late identification of pests and diseases in their low-growing crops, losing a large quantity of goods, and forcing the excessive use of pesticides, which affect the health of the producer, the product, and the final consumer. The entire project was developed based on qualitative methodology, where a more detailed analysis is conducted, user testimonials are heard, and numbers are compared. The project believes that with the development of the device, small farmers will be able to make a quicker identification, reducing product loss, which in turn reduces the use of pesticides and consequently improves the health of the small farmer, the plantation, and the product.

Keywords: Python; Computer vision; Machine Learning; Prague.

AGRADECIMENTOS

Gostaríamos de agradecer ao professor Carlos Alberto Pereira da Silva, que nos ajudou a modelar toda a ideia desse projeto e entender quais ideias poderiam ou não ser realizadas para este ano.

Também queremos agradecer o professor Salomão Santana do Nascimento que nos ajudou durante o desenvolviemento do projeto, especialmente analisando os nossos diagramas sobre o projeto e dando dicas para melhorar o desenvolvimento da ideia.

Da mesma forma, agradeçemos a professora Edna Rodrigues Fernades Pittner por suas aulas excelentes sobre banco de dados que foram essenciais para o desenvolver do projeto.

Igualmente, precisamos agradecer a professora Andreza Maria de Souza Rocha, que nos ajudou referente a qualquer caracteristica em inglês que essa monografia possui no momento, além de ajudar a treinar tanto na fala e escrita em inglês, através de atividades e apresentações.

E por fim, queremos agradecer o professor Marlon Marques da Silva, por suas aulas de história e sociologia, aulas essas que muitas vezes eram as mais aguardas no dia por possuirem um nivel de qualidade extremo.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

CNN - Convolutional Neural Network

UI – User Interface

UX – User Experience

UML - Unified Modeling Language

PIP – Package Installer for Python

OpenCV – Open-Source Computer Vision

IoT – Internet of Things

TCL - Tool Command Language

TK – Toolkit

JVM – Java Virtual Machine

RNC – Rede Neural Convolucional

SiLU – Sigmoid Linear Unit

SE – Squeeze-and-Excitation

Lista de tabelas

GLOSSÁRIO

Overfitting – Situação em que o algoritmo se adapta excessivamente a base de dados do treino, ou até mesmo de forma precisa, o que resulta em um modelo que não é capaz de fazer predições com dados que estejam fora da base de dados.

Sumário

1 INTRODUÇÃO

O Sistema Inteligente de Monitoramento e Prevenção de Pragas em Cultivo de Linha é um dispositivo capaz de percorrer uma lavoura de plantas rasteiras em todos os cenários possíveis, identificando possíveis sinais de pragas e doenças nas folhas. Algumas das pragas e doenças identificáveis são: Vaquinha-Verde-Amarela, Requeima e Pinta-Preta. Ao identificar os sinais, passa as informações para o nosso dashboard em uma aplicação desktop via cabo USB tipo C.O objetivo é identificar de forma precoce, em cultivos de formação em linha, possíveis sinais de doenças e pragas, e permitir que o pequeno agricultor tenha uma forma de fazer o monitoramento de sua lavoura, como também, tem uma noção do histórico das pragas e doenças em suas plantações.

O principal problema é que o agricultor acaba por muitas vezes, fazendo a identificação de forma tardia da doença ou da praga, o que acaba resultando em uma perca elevada da cultura. Em complemente, isto acaba por exigir do agricultor, a aplicação de agrotóxicos de forma mais densa e agressiva, o que põe em risco os trabalhadores que o aplicam, a própria plantação e o consumidor final, a partir do momento que, ao aplicar-se com determinada ocorrência e força tais agrotóxicos, aumenta a possibilidade de erros na ajustar o equipamento de dispersão, a praga e doença presente podem consequentemente gerar uma resistência maior ao agrotóxico, dificultando futuras passagens do químico e aumentando a presença do mesmo no produto final para o consumidor.

Inicialmente, acredita-se que a principal causa para tais atos, é a falta de educação sobre o uso dos agrotóxicos nas lavouras, mais, a cultura presente em nosso país da aplicação tardia no campo e próxima a colheita, para extinguir qualquer praga e doença próxima a colheita e tentar extrair o máximo possível do produto plantado.

A sua importância se faz presente partir do momento em que se propõe a diminuir o uso de agrotóxico no meio agrícola por meio da identificação precoce das possíveis pragas e doenças, trazendo assim para o meio agrícola de menor porte, um meio para proteger suas lavouras e diminuir seus gastos. Em complemento, tal projeto, permite que o pequeno agricultor possa fazer parte do crescimento da agricultura de precisão no Brasil.

A abordagem desta pesquisa é de caráter qualitativo, pois baseia-se na análise de informações oriundas de artigos e autores que abordam a temática (Lakatos, Marconi, 2017) da presença de pragas e doenças dentro de lavouras, em destaque, as consequências oriundas de tal fato e a eficácia da utilização da linguagem Python neste meio.

O sistema se delimita inicialmente, a produtores de pequeno porte, principalmente, os quais possuam plantações de porte rasteiro e não possuam a capacidade de se equiparar a grandes produtores com alto poder aquisitivo.

A presente pesquisa tem como principais autores MARTINS; FONTES; FORNAZIER, 2013, para falar sobre manejo de pragas e a ocorrência que tais possuem no meio agrícola, BRITO, GOMIDE e CÂMARA para falar sobre intoxicação dos trabalhadores, os quais discorrem sobre a importância do manejo e cuidado na área onde foi dispersado o químico, para abordar o assunto da ausência de formação dos agricultores sobre o uso indiscriminado de agrotóxicos e uso inadequado, utilizaremos os autores Rigotto, Vasconcelos e Rocha; para desenvolver tal meio para o monitoramento da lavoura, utilizaremos a linguagem de programação Python, que é destacada por MENENZES, para poder o aprendizado de máquina para possuir uma forma de identificar as pragas e doenças, se utilizará o PyTorch como aponta Jeronimo.

O trabalho está dividido em 4 capítulos: o 1º fala sobre a importância do uso correto dos agrotóxicos, bem como sobre o manejo adequado; o 2º, é o Referencial teórico; o 3º capítulo consta do “Desenvolvimento”, no qual será realizada a análise e discussão dos resultados; e o 4º, em que faremos as considerações apontando as limitações e possíveis melhorias no futuro.

2 REFERENCIAL TEORICO

Na atual seção são apresentadas os principais conceitos e tecnologias para a construção da base teórica para o desenvolvimento do Agro-G.E.S.F.

2.1 Pragas e Agrotóxicos

Pragas, conforme estabelece a Embrapa (2024) são qualquer espécie, raça ou biotipo de um animal ou agente patogênico que venha a causar algum tipo de dano a plantas ou culturas. Em auxílio, o Senar (2017) complementa que pragas são usualmente caracterizadas como ácaros, cochonilhas e nematoides, enquanto fungos, vírus e bactérias são usualmente coligados a doenças. Complementarmente, Martins, Fontes e Fornazier (2013 apud FERRÃO et al,2017) informam que as pragas podem ocorrer de forma esporádica ou sistemática. Aquelas que ocorrem de forma esporádica e causam danos significativos, são conhecidas como pragas secundárias, já aquelas que ocorrem de forma sistemática toda vez que se implementa a cultura na lavoura e causam danos quantitativos e/ou qualitativos, são conhecidas como pragas-chaves.

Barros et al. (2019), apontam que pragas e doenças, em casos como da cultura de soja, podem causar uma perca de até 30% de produção agrícola nacional, caso não haja o devido controle.

A agricultura, os modos de produção agrícola, a situação cultural e até social, estão inseridas em diferentes contextos de forma histórica, política e econômica, de forma que, Nogueira, Szwarchald e Damacena (2020) mostram que isso tende a afetar a vida do residente rural, de modo que, a combinação dessas variáveis tende a gerar níveis de exposição maior a tais agricultores à agrotóxicos. No que tange à exposição ocupacional, Brito, Gomide e Câmara (2009) destacam que os trabalhadores rurais enfrentam riscos significativos durante a manipulação de agrotóxicos, desde o preparo das caldas até a pulverização nas lavouras.

Entre 2000 e 2012, o mercado mundial de agrotóxicos cresceu em 93%, enquanto o mercado brasileiro cresceu 190%. Em 2008, o Brasil ultrapassou os Estados Unidos e assumiu como o maior utilizador de agrotóxicos no mundo, segundo CARNEIRO et al. (2015). Rigotto, Vasconcelos e Rocha (2014) sustentam que os agrotóxicos são um problema de saúde pública, visto que muita da população é exposta aos químicos em fábricas, na agricultura, no combate às endemias e outros setores perto de plantações. Não obstante, Lopes e Albuquerque (2018) afirmam que

os agrotóxicos podem contaminar reservatórios de água, rios e outras fontes de água, interferindo nos organismos que os utilizam.

2.2 Internet das coisas (IoT)

Oliveira (2017) afirma que a internet das coisas (IoT) é um conceito que está se tornando cada vez mais presente na vida cotidiana. Nas últimas duas décadas, a IoT tem auxiliado a vivência humana em diversas áreas por meio da automação de tarefas, ou em objetos como crachás eletrônicos e veículos. A IoT consiste em qualquer objeto capaz de manipular dados em uma rede, visando auxiliar e simplificar ações complexas (MAGRANI, 2018).

2.3 Raspberry Pi 4

Conforme Dobbin (2022) indica, o Raspberry pi é um computador de placa única (SBC), montado em uma placa de circuito impresso (PCB). Se caracteriza por poder realizar as mesmas funções de um computador doméstico comum, como acessar a internet, pesquisar sites e assistir vídeos, tendo somente o adendo de levar tempo levemente maior para realizar tais funções devido a seu tamanho.

Como apontam Santos, Lopes e Dias (2024), o Raspberry pi 4 apresenta um nível de processamento mediano devido a seu processador de maior capacidade em atividades multitarefas.

Em complemento Lauxen (2023) afirma que o Raspberry pi 4 pode ser alocado dentro de um sistema como o cérebro do projeto, de forma que permita fazer a captura de imagens e controle de elementos ligados a ele.

2.4 Câmera

A Webcam Brio 100 consegue capturar imagens em Full HD com a resolução 1920x1080 pixels. Ela possui equilíbrio automático de luz, aumentando o brilho em até 50% e reduzindo sombras quando necessário (LOGITECH, 2023).

Figura 1 - Logitech Webcam Brio 100

Fonte: Logitech, 2025

2.5 Python

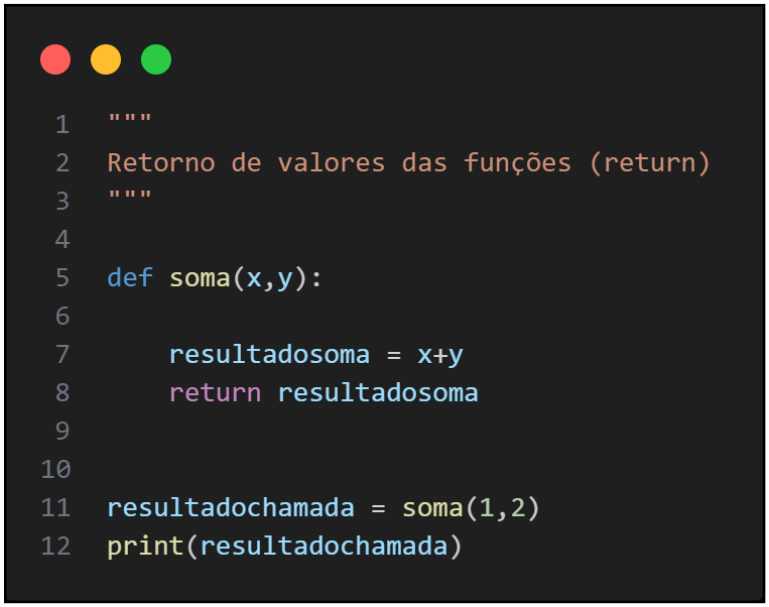
Segundo Borges (2014), a linguagem Python foi criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda (CWI), tendo como foco, o auxílio para profissionais como físicos e engenheiros.

Em conformidade a Paiva *et al.* (2020), Python é uma linguagem com características interessantes e de simples aprendizado. Inicialmente, possuía a meta inicial de ser uma linguagem de código mais enxuto e menos verboso. Em respaldo à está programação simples, Python permite o uso de diversos módulos de extensão, que fornecem ao Python, uma forte presença e poder.

Consoante a Menezes (2014), Python vem se tornando uma forte linguagem de programação em diversas áreas da computação, como inteligência artificial, banco de dados, biotecnologia e até mesmo em aplicações web. Em suporte, Python é uma linguagem conhecida por ter “batteries included”, ou seja, possui baterias inclusas, uma linguagem de programação pronta para ser usada. Em auxílio a tudo isto, Python

é uma linguagem de programação para se obter resultados em pouco tempo, devido a sua simples complexidade e alta densidade de módulos para importação.

Figura 2 - Exemplo de código Python para soma



```python
"""
Retorno de valores das funções (return)
"""

def soma(x,y):

    resultadosoma = x+y
    return resultadosoma


resultadochamada = soma(1,2)
print(resultadochamada)
```

Fonte: Autoria própria,2025

O exemplo de código acima, é um código em Python que serve para fazer a soma de dois números, estes quais são inseridos pelo próprio usuário previamente dentro do próprio código. Abaixo, segue uma breve explicação sobre o código e seu funcionamento:

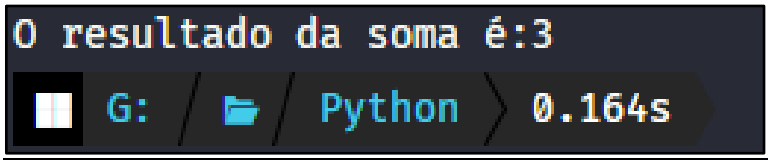
Linha 1 a 3: Cria uma “Docstring” (tipo de string), no qual no contexto atual, serve como uma forma de criar comentários, neste caso, o Python irá passar por estas linhas e irá ignorá-las;

Linha 5: Cria a função “soma”, está qual serve para modularizar o código e o deixar mais prático, permitindo que o programador não precise ficar repetindo as mesmas linhas de código, e, chamando a função somente quando necessária. Nesta aplicação, a função soma recebe dois parâmetros, dois itens, e os salva em variáveis (variáveis são formas de salvar valores específicos dentro da memória do computador) denominadas como “x” e “y”;

Linha 7 a 8: Na linha 7, é criada uma variável para receber o valor da soma das duas variáveis declaradas inicialmente, logo em seguida na linha 8, o valor dessa variável é devolvido ao meio que lhe chamou;

Linha 11 a 12: É criada uma variável chamada “resultadochamada”, à qual chama a função soma, enviando como parâmetros os números 1 e 2, ao mesmo tempo que recebe o valor do retorno dado pela função. Na linha 12, por meio da função “print()”, é exibido ao usuário por meio do prompt de comando integrado da IDE, o valor da variável “resultadochamada”.

Figura 3 - Resultado do código em Python



Fonte: Autoria Própria, 2025

Como consequência da execução do código antes ilustrado, na imagem acima há o retorno do código Python, no caso, foram passados o número um e dois, e o retorno da soma de tais números foi o número três.

2.5.1 PIP

Em conforme com a documentação oficial do Python (2025), o PIP é o administrador de pacotes, sendo utilizado majoritariamente e objetivamente para a instalação e desinstalação de pacotes.

No exemplo abaixo, há o funcionamento do comando PIP, sendo utilizado para instalar a biblioteca pytorch.

Figura 4 - Exemplo de utilização do PIP

Fonte: Autoria Própria,2025

2.5.2 Tkinter

Segundo Labaki (2004), o Tkinter é uma biblioteca padrão que vem junto da linguagem Python, essa biblioteca oferece um kit de ferramentas GUI, ou *Graphical User Interface,* que se traduz para Interface Gráfica de Usuário, as interfaces gráficas

de usuários são as áreas de interação dentro de um *software*, dentro dessas interfaces temos os *widget* que é usado para referenciar componentes dentro de GUIs, sendo elas, um botão, um menu ou uma caixa de texto.

Continuando com Menezes (2024), o Tkinter é baseado em uma biblioteca gráfica chamada TK, que foi desenvolvida para uma linguagem chamada TCL. A junção de TCL e TK foi muito utilizada em sistemas operacionais e muito popular quando Python foi criado, com o passar dos anos, essa biblioteca foi modernizada e teve os mesmos controles com aparência nativa.

Para Reitz e Schlusser (2017) o Tkinter é uma das bibliotecas mais convenientes e compatíveis dentre todas que existem por já estarem junto ao Python quando instalado e está disponível na maioria dos sistemas operacionais, como Windows, Unix e macOS X. Segue abaixo um exemplo de uma interface gráfica usando Tkinter.

Figura 5 - Exemplo de interface com Tkinter

Fonte: Autoria Própria, 2025

Este, é um exemplo de uma interface simples utilizando Tkinter, que exibe uma tela que pede para digitar o seu nome em uma caixa de texto, ao digitar, abaixo haverá um botão dizendo “Dizer Olá”, ao clicar, aparecerá um texto abaixo do botão que exibirá um cumprimento ao usuário e dizendo o nome que foi entregue na caixa de texto. Explicamos mais sobre o código abaixo:

Linha 1: é feita a importação da biblioteca Tkinter e estamos definindo um nome a essa importação para um melhor uso no futuro.

Linha 3 a 5: Definimos a função junto a um nome a ela que será executada quando o botão ser pressionado e exibirá o nome do usuário junto a um rótulo.

Linha 8 a 10: Criamos a tela principal que será exibida e damos um nome junto de um tamanho para essa tela que será de 300 pixels por 150 pixels de altura.

Linha 12 a 13: É criado um rótulo com uma mensagem pedindo para digitar o nome do usuário e exibimos na tela junto de um espaçamento vertical.

Linha 15 a 16: É definido um campo de texto para o usuário digitar seu nome e é exibido o campo na janela com um espaçamento.

Linha 18 a 19: É adicionado o botão com um texto, ao clicar no botão a função que definimos no início que exibe o nome do usuário junto com uma saudação

Linha 21 a 22: É necessário criarmos um rótulo que não terá texto para podermos mostrar a saudação

Linha 24: É iniciado o loop da aplicação, essa linha mantém a janela em funcionamento esperando pelas ações do usuário.

Figura 6 - Resultado do código Tkinter

Fonte: Autoria Própria, 2025

A imagem acima é o resultado do código explicado sendo exibida em uma tela de computador utilizando Tkinter.

2.5.3 Customtkinter

Segundo Frota, Ruver e Figueiredo (2024), o CustomTkinter é uma biblioteca de criação de interfaces gráficas inspirada no Tkinter, a biblioteca apresenta um visual mais moderno e customizável quanto o Tkinter, mas mantendo a simplicidade da biblioteca inspirada, o CustomTkinter é ótimo para aprendizes e especialistas que queiram criar interfaces elegantes sem uma dificuldade exagerada.

O código abaixo se utiliza do exemplo que foi apresentado no capítulo do Tkinter, neste exemplo vamos implementar a estilização do CustomTkinter.

Figura 7 - Exemplo de código com CustomTkinter

```python
import customtkinter as ctk

ctk.set_appearance_mode("System")
ctk.set_default_color_theme("blue")

def dizer_ola():
    nome = entrada.get()
    mensagem = f"Olá, {nome}!"
    rotulo_resultado.configure(text=mensagem)

# Janela principal
janela = ctk.CTk()
janela.title("Exemplo CustomTkinter")
janela.geometry("300x180")

# Widgets
rotulo = ctk.CTkLabel(janela, text="Digite seu nome:")
rotulo.pack(pady=5)

entrada = ctk.CTkEntry(janela)
entrada.pack(pady=5)

botao = ctk.CTkButton(janela, text="Dizer Olá", command=dizer_ola)
botao.pack(pady=5)

rotulo_resultado = ctk.CTkLabel(janela, text="")
rotulo_resultado.pack(pady=5)

# Loop da interface
janela.mainloop()
```

Fonte: Autoria Própria, 2025

Este é o mesmo exemplo que utilizamos no Tkinter, porém agora escrito utilizando o CustomTkinter, o que será exibido é o mesmo, uma interface que irá exibir uma mensagem pedindo para inserir seu nome em uma caixa de texto, ao inserir seu nome e clicar no botão abaixo da entrada da área de texto, será exibido um cumprimento ao usuário e dizendo o nome que foi apresentado no campo, abaixo segue o que está acontecendo no código:

Linha 1: é feito a importação da biblioteca CustomTkinter e definindo um nome para essa importação e ter um melhor uso.

Linha 2 a 3: Definimos qual será o modo visual dependendo do sistema operacional do usuário.

Linha 6 a 9: Criamos a função que irá guardar o texto que digitado no campo e criamos um rótulo que irá apresentar o cumprimento e o nome digitado.

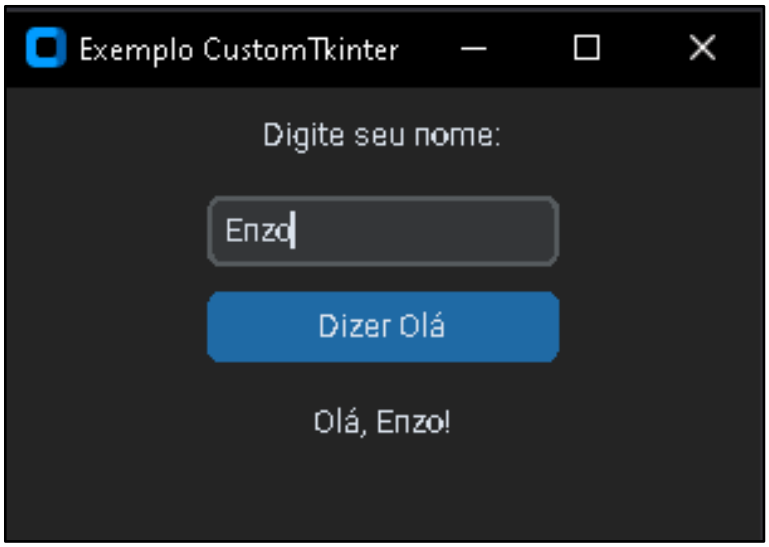
Linha 11 a 14: adicionamos a janela principal que conterá um nome e um tamanho específico.

Linha 17 a 21: Criamos um rótulo que está pedindo ao usuário para digitar o seu nome e criamos o campo de texto para o usuário poder digitar, é definido um espaçamento vertical para ambos os elementos.

Linha 23 a 27: Criamos o botão junto com um texto dentro que ao ser clicado irá chamar a função que exibirá a saudação e criamos um rótulo sem texto para uma melhor apresentação final, e novamente definimos um espaçamento vertical para os elementos.

Linha 30: é feito o loop da estrutura, essa linha permite que a tela seja exibida enquanto o código estiver ativo.

Figura 8 - Resultado do código com Custom Tkinter



Fonte: Autoria Própria, 2025

A imagem que está acima exibe o código em funcionamento e vemos que ele está personalizado com as funcionalidades do CustomTkinter.

2.5.4 Matplot lib

Desenvolvido em 2002 por John Hunter, o Matplotlib é uma biblioteca para Python para esboçar gráficos de todos os tipos. O Matplotlib consegue suportar diversos banco de dados de interfaces gráficas, pode funcionar em vários sistemas operacionais, além de poder exportar suas figuras em vários formatos de arquivos possíveis. (McKinney, 2023).

Consoante a Grus (2021), o Matplotlib oferece uma grande variedade de gráficos, sendo eles de níveis simples até níveis mais complexos de figuras, esses mapas podem ser interativos e com modelos otimizados podendo ser personalizado da forma que seja desejado.

Para Vasiliev (2022), o Matplotlib é uma das principais bibliotecas python quando o assunto é sobre criar gráficos e figuras, o Matplotlib foi responsável por inspirar diversas outras bibliotecas que também têm a funcionalidade de criar gráficos que se baseiam na estrutura do Matplotlib Vamos mostrar abaixo um exemplo de código que exibe um gráfico utilizando Matplotlib:

Figura 9 - Exemplo de código usando Matplotlib

Matplot.py > ...

```python
import matplotlib.pyplot as plt

# Dados para o gráfico
x = [5,10,20,30,40 ]
y = [7, 13, 21, 35, 38]

# Criar o gráfico
plt.plot(x, y, marker='o', linestyle='-', color='b', label='Valores')

# Adicionar título e rótulos
plt.title('Gráfico de Linha Simples')
plt.xlabel('Eixo X')
plt.ylabel('Eixo Y')
plt.legend()

# Exibir o gráfico
plt.show()
```

Fonte: Autoria Própria, 2025

Neste código, será exibido uma imagem de um gráfico com valores específicos junto de um título para o gráfico, vamos explicar melhor sobre o código abaixo:

Linha 1: Faz a importação da biblioteca matplotlib e definimos um nome a essa biblioteca.

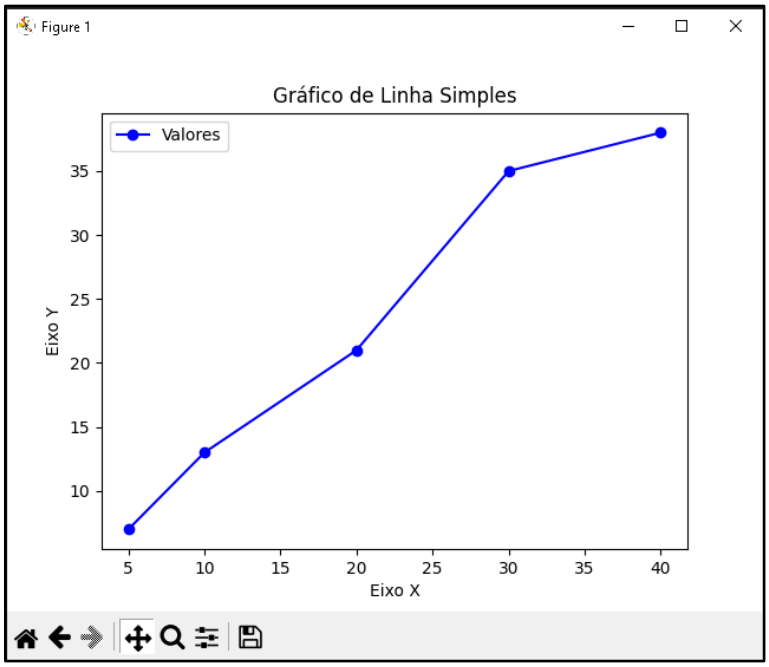
Linha 4 a 5: Definimos os valores do eixo X e os valores do eixo Y, cada valor que está sendo exibido é coordenada dentro do gráfico.

Linha 8: Cria o gráfico junto com os eixos que definimos nas linhas 4 e 5, adicionamos algumas estilizações como o tipo de marcador, qual o formato da linha e a sua cor e por fim criamos um rótulo que irá nomear a linha em específico.

Linha 11 a 14: adicionamos um título para o gráfico, um título para o eixo X e o eixo Y e colocamos um comando para exibir a legenda ao passar sobre a linha.

Linha 17: Exibe o gráfico em uma janela quando o código ser executado

Figura 10 - Resultado do código utilizando Matplotlib



Fonte: Autoria Própria, 2025

Como podemos ver na imagem acima, está sendo exibido o gráfico conforme definimos no código e utilizando o Matplotlib.

2.5.5 NumPy

Conforme aponta Lins e Souza (2023) o pacote NumPy (Numerical Python) é essencial para a computação cientifica. Em consonância com Grus (2021), o NumPy se torna viável e interessante devido a presença de seu tipo de arrays, que possuem um desempenho superior ao se comparar as listas, vetores e matrizes comuns do Python.

2.5.6 Pillow

Pillow, assim como afirma De Paula (2024) é uma biblioteca popular do Python que tem como objetivo a manipulação de imagens. Em apoio ao que foi escrito, Moreira, Marmontel, Chamorro (2023) explicam que a biblioteca Pillow é uma evolução da biblioteca PIL do próprio python, ademais, uma das principais vantagens do Pillow é sua simplicidade, o que permite o uso de tal tanto por iniciantes quanto por programadores avançados, em complemento, ela também se destaca por ser otimizada para o trabalho em imagens de alta resolução, o que a torna útil em aplicações com a necessidade de processamento em lote.

2.6 Machine learning

O aprendizado de máquina, ou mais comumente conhecido como “Machine Learning”, é, segundo Paixão *et al*. (2022), uma subárea da ciência da computação que trabalha com a união de técnica matemáticas e estatísticas aplicadas à algoritmos computacionais. Parafraseando Escovedo e Koshiyama (2020), o conceito de Machine Learning se foca na descoberta de padrões por meio dessa união de técnicas, independentemente do grau de utilidade de tais descobertas, ademais, a utilização de tais padrões para a automatização de determinadas tarefas, que antes, se poderiam se mostrar como difíceis para serem realizadas por humanos.

Quando se trata do aprendizado de máquina, se prova por meio de Ludermir (2021), que é necessário um grande volume de dados para que o algoritmo possa aprender de forma automática. Em auxílio, a acurácia de um modelo treinado pode variar conforme a quantidade de dados fornecidos ao modelo, juntamente a qualidade de tais dados, como diz Tsunoda *et al.* (2020). Em acréscimo, destaca-se o fato de que os modelos treinados têm se destacado dentro do cenário da agricultura de precisão, estes quais são sustentados por grandes volumes de dados e poder de

processamento, como também, são amplamente utilizados para a detecção de doenças e ervas daninhas em culturas, previsão de produtividade e gestão da saúde do solo, conforme demonstra Santos (2022).

2.6.1 Data Augmentation

Perante a Peinado *et al* (2019, apud Brownlee 2023), a técnica de Data Augmentation (Aumento de Dados) tem como intuito a criação de dados artificiais, para a alimentação de dados para algoritmos de aprendizado de máquina a partir de dados já existentes, como, a aplicação de mudança de ângulos da imagem, como a diminuição ou aumento de brilho e de saturação. Tal técnica tem uma devida importância como informa Shorten et al (2021, apud Oliveira et al. 2024), por poder auxiliar a minimizar o impacto da ausência de dados para determinados treinamentos de máquina.

O código a seguir, mostra que, por meio de determinadas configurações, se é possível aumentar de forma artificial a quantidade de imagens:

Figura 11 - Código de Data Augmentation

```python
import os
import matplotlib.pyplot as plt
from PIL import Image
import torch
from torchvision import transforms

transform_aug = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225]),
])

transform_original = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])
```

Fonte: Autoria própria,2025

Linha 1 a 5: Ocorre a importação das bibliotecas necessárias para o funcionamento do código;

Linha 7: É definida uma variável que irá guardar uma sequência de transformações para aplicar a técnica de Data Augmentation nas imagens;

Linha 8: Corta aleatoriamente uma região da imagem e redimensiona para 224x224 pixels. Desta forma, o modelo não fica preso à posição ou escala fixa da imagem original;

Linha 9: Aplica um espelhamento horizontal na imagem com probabilidade de 50%, o que ajuda o modelo a generalizar imagens invertidas;

Linha 10: Modifica de forma aleatória o brilho, contraste, saturação e matiz(cor) da imagem dentro dos parâmetros dados, tornando o modelo menos sensível a variações de iluminação e cor;

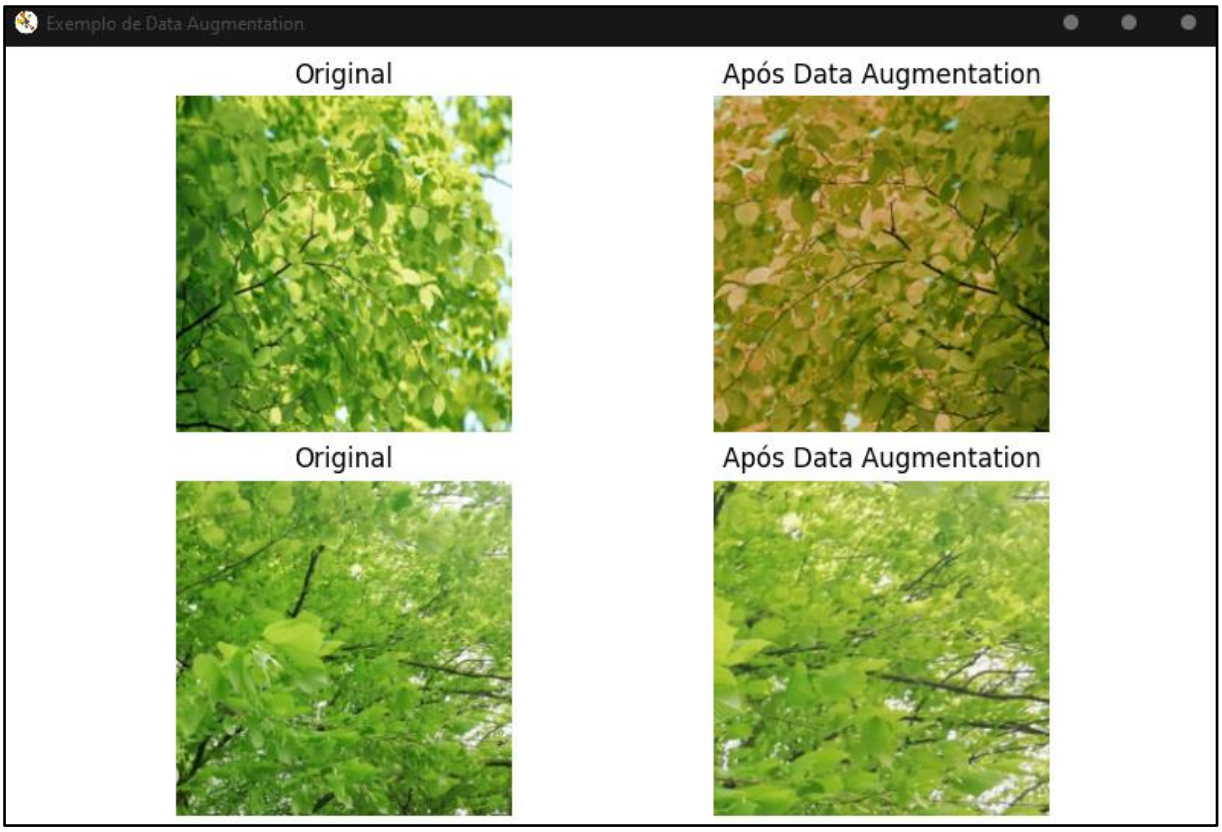
Linha 11: Transforma a imagem, para um tensor PyTorch, normalizando os valores dentro da faixa de 0 e 1;

Linha 12 e 13: Normaliza o tensor usando uma média e desvio padrão, centralizando os dados para facilitar o aprendizado de máquina;

Linha 14: É fechada a chave e os parênteses da variável, assim passando para a variável inicial, todos parâmetros que ela deve guardar;

Como resultado, a imagem abaixo demonstra o resultado após aplicar tais filtros a duas imagens de folhas de plantas.

Figura 12 – Exemplo de Data Augmentation



Fonte: Autoria própria, 2025

2.6.2 Deep Learning

Parafraseando Hosaki (2021), o Deep Learning ou aprendizado profundo, é um conjunto de técnicas de Machine Learning que se utilizam de redes neurais profundas, estas quais tiveram sua utilização inicial no ano de 2006, e atualmente estas redes neurais são compostas por diversos níveis, onde cada nível possui seu nível de abstração. O Deep Learning possui em sua essência uma similaridade com a rede neural do ser humano, o que permite uma análise de informações sem a necessidade de um processamento prévio ou a necessidade de um supervisionamento.

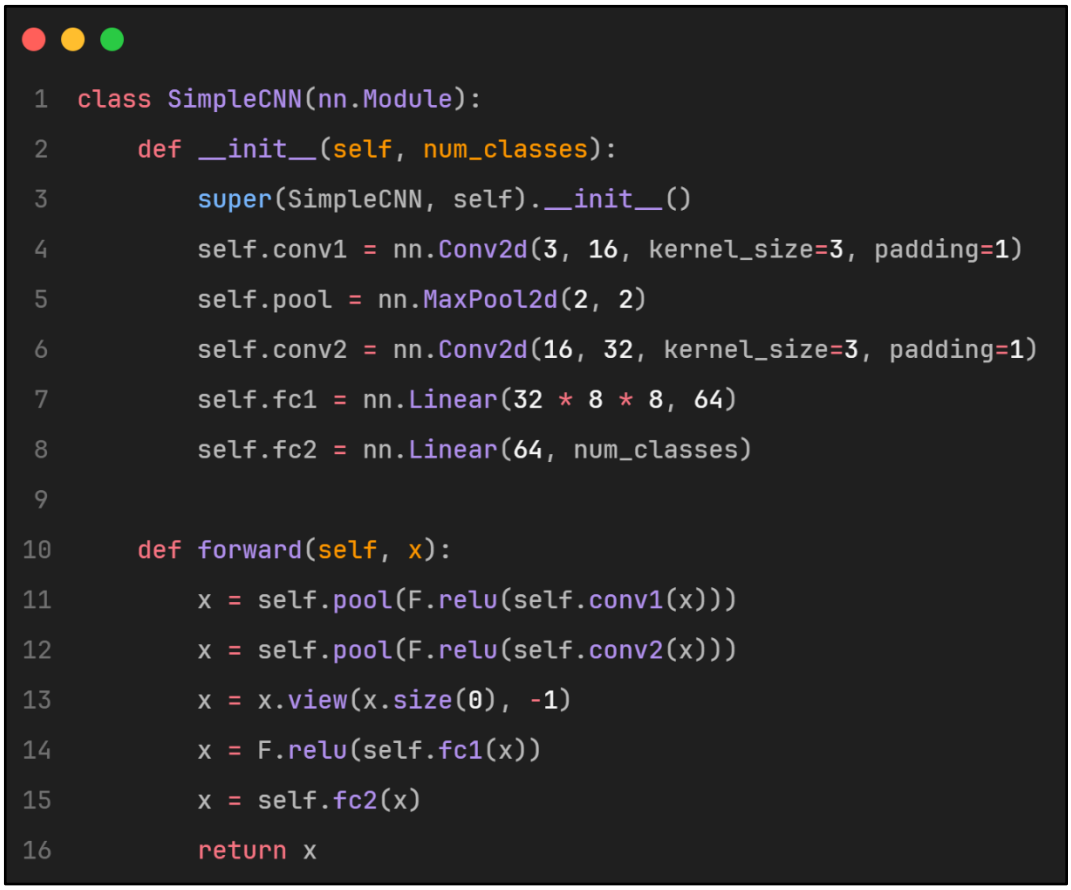
2.6.3 Convolutional Neural Network

Uma Convolutional Neural Network (CNN), em português Rede Neural Convolucional, é perante a Carneiro *et al* (2025) um tipo de rede neural artificial que é usualmente empregada para a classificação de padrões bidimensionais, e é composta por três camada, onde cada uma fica responsável por funções matemáticas especificas, para que no final, haja uma precisão maior.

Em complemento, Vargas, Paes e Vasconcelos (2016), este tipo de rede neural vem sendo amplamente utilizado em áreas como detecção, classificação de reconhecimento de imagens e vídeos.

A imagem abaixo demonstra o código de criação de uma CNN básica, possuindo apenas duas camadas.

Figura 13 - Exemplo de código de CNN



```python
class SimpleCNN(nn.Module):
    def __init__(self, num_classes):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(32 * 8 * 8, 64)
        self.fc2 = nn.Linear(64, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Fonte: Autoria própria, 2025

Logo abaixo, há a explicação prévia do que cada linha corresponde dentro da CNN:

Linha 1: Se declara o nome da classe da rede neural, que é “SimpleCNN”. Similar a uma receita, é ela que vai descrever o que é necessário e quais os passos para fazer a rede. O “nn.Module” é uma ferramenta do PyTorch que ajuda a organizar as camadas da rede;

Linha 2: É onde se inicia a configuração da rede, e o num_classes, é o número que corresponde a quantos tipos de itens a rede pode identificar;

Linha 3: Mais um detalhe técnico, mas que basicamente “conecta” a rede ao PyTorch, dizendo que ela segue as regras de um nn.Module;

Linha 4: É onde se cria a primeira camada convolucional, se assimila a seguinte ideia, seria como um filtro que observa a imagem e encontra padrões simples, como bordas ou cores;

Linha 5: A camada de pooling é responsável por reduzir o tamanho da imagem para poder simplificar o trabalho, onda ela divide a imagem em 2x2 pixels e escolhe o valor mais importante para cada quadrado;

Linha 6: Similar a primeira camada convolucional, esta segunda camada se diferencia por estar em um nível mais profundo, trabalhando com mais mapas que a primeira camada e produzindo mais mapas. É por meio destes mapas que se encontram padrões mais complexos, como formas e texturas;

Linha 7: É aqui onde se inicia a parte de classificação. A camada fc1 pega todos os padrões encontrados pelas camadas anteriores e organiza-os em números. Depois das camadas anteriores, os dados das imagens foram diminuídos para 32 mapas, onde cada um possui 8x8 pixels, o que resulta em 2048 números. Destes 2048 números, eles são transformados em uma lista menor de 64 números, que resumem as informações das imagens;

Linha 8: Nesta camada, se toma a lista de 64 número e é feita a resposta final, trazendo em que categoria a imagem se encontra;

Linha 9: Correspondente a dizer como a rede deve funcionar, o forward é como o passo a passo, descrevendo a ordem em que as camadas devem ser usadas. O x é a imagem de entrada;

Linha 10: A imagem passa pela primeira camada convolucional, que encontra padrões. Em seguida, se aplica a função F.relu(), a qual funciona como um filtro, mantendo somente os valores positivos e ajudando a rede a aprender melhor. E no final, a camada de pooling, a qual reduz o tamanho da imagem;

Linha 11: Logo mais, a imagem entra novamente em outra camada, mas agora na segunda camada, e novamente ocorre o mesmo processo que ocorreu na camada anterior, contudo, já identificando mais padrões;

Linha 12: Aqui, os dados que ainda estão na forma de mapa, são “achatados” em uma lista de números. O x.size(0) faz que com que se mantenha o número de imagens no lote;

Linha 13: Os 2048 números passam pela primeira camada, que transforma eles em 64 números, e em seguida, a função F.relu age novamente para manter apenas os valores positivos;

Linha 14: Os 64 números passam pela segunda camada, a qual dá uma pontuação para cada categoria possível para aquela imagem;

Linha 15: A rede devolve o resultado, que é a previsão da qual categoria a imagem pertence;

2.7 Visão computacional

A visão computacional, segundo Barelli (2018) em 1982, Ballard e Brown definiram a visão computacional em sua obra chamada Computer Vision, como a ciência que que estuda e desenvolve meios para permitir que computadores possam por meio de sensores, enxergar e extrair informações do que enxergam do mundo. Conforme estabelece Silva (2020), a visão computacional é comumente associada a coleta, análise e processamento dos dados visuais por meio de computadores para diversos objetivos, indo desde a identificação de rostos até a identificação de objetos.

2.7.1 Open-Source Computer Vision Library (OpenCV)

Segundo Marengoni e Stringhini (2010), OpenCV (Open-Source Computer Vision) é uma biblioteca open source criada pela Intel Corporation com mais de 500 funções. O intuito inicial da empresa, era permitir que usuários e programadores tivessem uma forma mais acessível de se utilizar a visão computacional. A biblioteca é dividida em cinco partições: Processamento de imagens; Análise Estrutural; Análise de movimentos; Reconhecimento de padrões e calibração de câmera e reconstrução 3D.

Em consonância com Neto (2020), OpenCV é utilizada pelas principais big techs do mercado, como exemplo, a Microsoft, IBM e Google. A OpenCV possui diversas funções internas para o processamento da imagem digital, mas vale-se ressaltar a identificação de bordas e a transformação de espaço de cores.

Como indicado por Antonello (2018), o OpenCV possui uma maleabilidade maior para a aplicação com visão computacional a partir do momento que permite separar e visualizar os três canais RGB (Red, Green e Blue).

Figura 14 - Exemplo de código utilizando OpenCV

```python
import cv2
import numpy as np

imagem = cv2.imread("exemplo.jpg")
imagem = cv2.resize(imagem,(1000,1000))

imagem_hsv = cv2.cvtColor(imagem,cv2.COLOR_BGR2HSV)

verde_baixo = np.array([35,40,40])
verde_alto = np.array([85,255,255])

mask = cv2.inRange(imagem_hsv,verde_baixo,verde_alto)

resultado = cv2.bitwise_and(imagem,imagem, mask = mask)

cv2.imshow("Original",imagem)
cv2.imshow("Máscara",mask)
cv2.imshow("Resultado",resultado)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Fonte: Autoria própria, 2025

Linha 1 a 2: Ocorre a importação das bibliotecas para o funcionamento do código

Linha 4: Lê uma imagem chamada de “exemplo.jpg” e é armazenada na variável imagem.

Linha 5: Redimensiona a imagem para um tamanho fixo de 1000x1000 pixels

Linha 7: Converte a imagem do espaço de cores BGR (padrão do OpenCV) para HSV, o qual é mais adequado para uma segmentação de cores. A imagem convertida, é armazenada dentro da variável *imagem_hsv.*

Linha 9 e10: Definem os limites da cor verde dentro do espaço HSV, a variável *verde_baixo* define um limite inferior (tons de verde mais escuros) e a variável *verde_alto* define um limite superior (tons de verde mais claros). Estes valores são usados para isolar apenas os pixels da imagem que se encaixam nessa faixa.

Linha 12: Cria uma máscara binária (preto e branco) onde os pixels dentro da faixa de verde são brancos (255) e os demais são pretos (0). Essa máscara é utilizada para identificar as áreas com verde na imagem.

Linha 14: Aplica a máscara sobre a imagem original usando a função *bitwise_and().* Isso faz com que apenas os pixels dentro da faixa de cor verde apareçam, e o resto seja escurecido. O resultado é salvo na variável *resultado*.

Linha 16 a 18: Exibem as janelas com as imagens, o primeiro comando "*cv2.imshow()"* mostra a imagem original, logo mais, a segunda vez que chama a função, mostra a máscara binária aplicada na imagem, e na terceira e última chamada da função, é exibida a imagem filtrada com apenas a cor verde visível.

Linha 20: Espera o usuário apertar alguma tecla.

Linha 21: Fecha todas as janelas abertas do OpenCV após uma tecla ser pressionada

Figura 15 - Imagem antes do processamento usando OpenCV

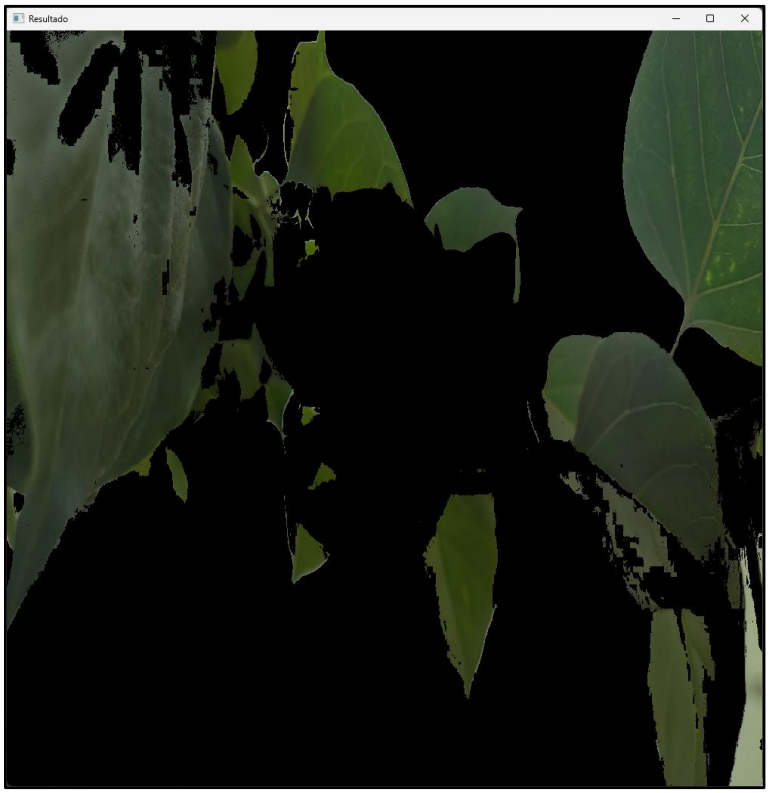
Fonte: Autoria própria,2025

Figura 16 - Aplicação do filtro binário na imagem

Fonte: Autoria própria,2025

A imagem acima mostra a aplicação da máscara binária na imagem original, onde se encontra da cor preta, é a parte da foto a qual o programa não encontrou a cor verde, e branca a parte dentro da imagem a qual o programa identificou como verde.

Figura 17 - Resultado do processamento da imagem



Fonte: Autoria própria,2025

Como resultado, a imagem acima mostra como o programa recorta toda parte que não esteja no espectro verde anteriormente definido e mantém somente aquelas que se caracterizam dentro deste espectro.

2.8 PyTorch

Em conformidade com Jeronymo (2019), PyTorch é uma biblioteca open-source para python, que busca oferecer uma flexibilidade e eficiência no treinamento de redes neurais profundas. Isto é possível, pois um dos métodos que permite isto, é uma programação prévia de funções comuns para tal proposito.

Outrossim, como informa Saavedra et al. (2022) o PyTorch se mostra como uma boa biblioteca para o uso no aprendizado de máquina, devido ao uso de tensores, estes quais, são extensão lógica de matrizes multidimensionais capazes de organizar e gerenciar os dados a serem utilizados para treinamento.

O PyTorch demonstrou uma vantagem de desempenho ao utilizar GPU's embarcadas em relação a CPU's, superior a 80 vezes, como indica Crestani *et al.* (2018)

Figura 18 - Exemplo de código utilizando PyTorch

```python
import torch
from torchvision import models, transforms
from PIL import Image

imagem = Image.open("flor.jpg")

preprocessador = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

img_tens = preprocessador(imagem).unsqueeze(0)

modelo = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
modelo.eval()

with torch.no_grad():
    saida = modelo(img_tens)
    probabilidade = torch.nn.functional.softmax(saida[0], dim=0)
    classe = probabilidade.argmax().item()

import json
import urllib.request
with urllib.request.urlopen("https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt") as f:
    labels = [line.strip() for line in f.readlines()]

print(f"Classe prevista: {labels[classe]} (probabilidade: {probabilidade[classe]*100:.2f}%)")
```

Fonte: Autoria própria

O exemplo de código ilustrado é um código em Python que se utiliza da biblioteca PyTorch juntamento do modelo pré-treinado ResNet18 para a realizar a classificação de uma imagem. A imagem é pré-processada, classificada, e o resultado é exibido junto da janela em uma imagem.

Abaixo, segue uma breve explicação sobre o código e seu funcionamento:

Linha 1 a 4: Nestas linhas são realizadas as importações de cada biblioteca necessária para rodar o programa

Linha 5: A imagem chamada "exemplo.jpg" é carregada usando a biblioteca PIL e armazenada na variável imagem.

Linha 7 a 15: Antes de usar o modelo para fazer a classificação, o programa a prepara:

- Redimensiona a imagem para um tamanho padrão;
- Recorta o centro da imagem;
- Converte a imagem para um formato que o programa consiga compreender;
- Ajusta as cores, de forma que fique similar com os tons parecidos aos usados no Machine Learning.

Linha 17: Aplica toda essa configuração na imagem e a transforma em algo que possa ser uado pelo modelo pré-treinado.

Linhas 19 a 20: O programa carrega o modelo chamado ResNet18. Este modelo já foi treinado anteriormente para fazer o reconhecimento de milhares de tipos diferentes de imagens. Em seguida, o modelo é colocado em um modo de previsão, que é um modo de uso e não para treinamento.

Linhas 22 a 25: Nesta parte, é onde o processamento bruto é feito:

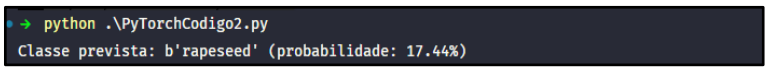
- A imagem é enviada para o modelo por meio da variável “imagem”;
- O modelo análise a imagem e retorna várias possibilidades possíveis do que pode ser aquela imagem e associa uma “nota de confiança” a cada possiblidade;
- O programa escolher a opção com a maior nota de confiança, ou seja, à opção que o modelo acha que é mais provável de ser.

Linhas 27 a 30: O programa acessa um site que possui uma lista de nomes das categorias que o modelo tem conhecimento. Por meio desta lista, é traduzida o número (por exemplo, “183”) da classe em um nome (como “cachorro vira-lata”).

Linha 32: Ao final do programa, o mesmo mostra por via de linha de comando dentro do terminal integrado a IDE, qual foi a resposta do modelo e com que confiança ele deu para tal resposta

Como resultado, o modelo identifica a imagem como “rapeseed”, uma espécie de flor, mas com uma baixa nota de confiança, o que mostra a necessidade do treinamento

Figura 19 - Resultado da análise de imagem



Fonte: Autoria própria, 2025

2.8.1 TorchVision

Consoante a Zuanazzi (2024), TorchVision é uma extensão da biblioteca PyTorch que permite usufruir-se de um conjunto de dados mais populares, arquiteturas de modelos outrossim também a transformações comumente utilizadas em visão computacional. Por meio de modelos pré-treinados, o TorchVision permite que o programador se utilize de modelos com um prévio treinamento com grandes quantidades de dados para o treinamento mais breve para a execução de tarefas específicas.

Figura 20 - Exemplo de código utilizando TorchVision

```python
import torch
from torchvision import models, transforms
from PIL import Image
import cv2
import numpy as np

imagem = Image.open("image.jpg")

preprocessador = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])

img_tens = preprocessador(imagem).unsqueeze(0)

modelo = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
modelo.eval()

with torch.no_grad():
    saida = modelo(img_tens)
    probabilidade = torch.nn.functional.softmax(saida[0], dim=0)
    classe = probabilidade.argmax().item()

import json
import urllib.request
with urllib.request.urlopen("https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt") as f:
    labels = [line.strip() for line in f.readlines()]

imagem_cv = cv2.cvtColor(np.array(imagem), cv2.COLOR_RGB2BGR)
cv2.imshow("Imagem Classificada", imagem_cv)
cv2.waitKey(0)
cv2.destroyAllWindows()

print(f"Classe prevista: {labels[classe]} (probabilidade: {probabilidade[classe]*100:.2f}%)")
```

Fonte: Autoria própria,2025

Linhas 1 a 5: São importadas as bibliotecas necessárias para o funcionamento do código e utilização de modelos pré-treinados;

Linha 7: Carrega uma imagem chamada *image.jpg* usando a biblioteca *Pillow* e à armazena dentro de uma variável chamada imagem;

Linha 9 a 17: Define o pré-processamento da imagem, este qual inclui o redimensionamento da imagem para 256 pixels, cortar de forma central para 224x224pixels, converter a imagem para tensor e normalizar com a média e desvio padrão das imagens do dataset (conjunto de alto volume de imagens).

Linha 19: Aplica as transformações definidadas anteriormente à imagem, e em seguida é adcionada uma dimensão extra (batch size = 1), esta qual é necessária para o modelo;

Linha 21 e 22: Carrega o modelo pré-treinado *ResNet18* e o coloca em modo de avaliação, o qual é o modo usado para poder fazer previsões sem a necessidade de um treinamento inicial;

Linha 24 a 27: Inicia um bloco onde o PyTorch não calcula os gradientes (por economia de memória e desempenho), em seguida a imagem é passada para o modelo, gerando uma saída bruta (logits). Essa saída é convertida em probabilidade com *softmax*, e a classe mais provável é extraída com *argmax*;

Linha 29 a 32: Faz o download do arquivo *imagenet_classes.txt* diretamente do GitHub, que contém os nomes de 1000 classes do ImageNet. Cada linha do arquivo representa uma classe, e é lida e armazenada na lista *labels*;

Linha 34: Converte a imagem de *PIL* para *NumPy* e altera o formato de cor de RGB para BGR (formato esperado pelo OpenCV)

Linha 35 a 37: Exibe a imaegm original com o rótulo da classe prevista usando o OpenCV, espera que o usuário pressione uma tecla e depois fecha todas as janelas abertas ligadas ao programa;

Linha 39: Imprime no terminal a classe prevista com a probabilidade associada. O valor da probabilidade é multiplicado por 100 para exibir como porcentagem.

Figura 21 - Exibição da imagem classificada com TorchVision

Fonte: Autoria própria,2025

Como a imagem abaixo ilustra, o modelo caracterizou a imagem passada como “banana” e com uma porcentagem de confiança de 56.23%.

Figura 22 - Resultado do código utilizando TorchVision

Fonte: Autoria própria,2025

2.9 Kotlin

De acordo com Resende (2018), o Kotlin surgiu em 2016 criada pela empresa JetBrains, com o objetivo de criar uma linguagem tipada para a máquina virtual JVM, ou Java Virtual Machine (máquina virtual Java) que fosse produtiva e com o um aprendizado simples e prático.

Em consoante a Jemerov e Isakova (2017), o Kotlin é uma linguagem de programação considerada resumida, confiável e objetiva, podendo ser utilizada em todos os lugares em que o Java está sendo utilizado como em setores referente ao lado de servidores e no seu principal uso hoje em dia, aplicações para Android.

Parafraseando a Silva e Zuchi (2023), com a ascensão e crescimento do Kotlin, a linguagem foi declarada pela Google, como a linguagem de programação oficial dos sistemas Android para frente, mostrando como essa tecnologia tem grande potencial no futuro de desenvolvimento de aplicativos de celular.

2.10 Docker

Segundo Vitalino e Castro (2018), tudo se iniciou em 2008 com a fundação da dotCloud pelo CEO Solomon Hykes, contudo, o Docker nasceu somente a partir do momento em que Hykes transformou o core de sua plataforma em código aberto.

Além disso, parafraseado Gomes (2019), o Docker oferece ferramentas para poder usufruir de ambientes isolados para rodar aplicações de formas separadas, e tendo um maior poder de gerenciamento da infraestrutura da aplicação.

2.11 Banco de dados

Para Elmasri e Navathe (2011), um banco de dados de modo simples, é como uma coleção organizada de dados, designada para atender às necessidades de diversas aplicações. Toda essa estrutura é gerenciada por um Sistema de Gerenciamento de Banco de Dados (SGBD), um software que permite a definição, criação, manutenção e controle do acesso aos dados armazenados, de acordo com Date (2004). O SGBD é como um intermediário entre os usuários e o banco de dados mantendo a integridade, segurança e disponibilidade das informações, elementos essenciais para sistemas modernos, conforme ressaltam Ramakrishnan e Gehrke (2011).

2.11.1 SQLite

De acordo com Comachio (2011), o SQLite é um banco de dados open-source que permite comandos SQL e que, na prática, cria um arquivo em disco, que mantém diversas tabelas e entidades num arquivo com extensão ".db".

Como diz Silva *et al*. (2017), uma base de dados em servidor, tal como um Sistema Gerenciador de Banco de Dados (SGBD), pode acabar sendo uma opção robusta demais para uma aplicação simples, sendo normalmente utilizados em sistemas com alta manipulação de dados e muito armazenamento de entidades.

Assim como diz Beaulieu (2019), o SQL foi criado com a intenção de manejar o modelo relacional de banco de dados, que utiliza tabelas relacionadas entre si. Isso faz com que a administração de um banco de dados seja simples e eficaz, com apenas alguns comandos SQL.

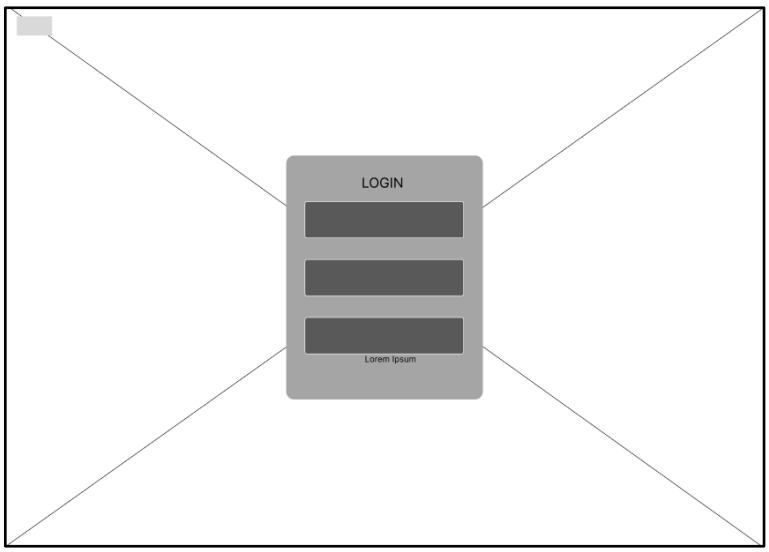
2.12 Wireframes

Wireframe, como afirma Teixeira (2014), é um rascunho, um desenho básico do produto final, para se possuir uma direção de como o desenvolver do projeto deve ocorrer.

2.12.1 Wireframe de baixa fidelidade

Os *Wireframes* de baixa fidelidade ou prototipação de baixa fidelidade, não são totalmente fiéis a ideia final do projeto, contudo são meios que permitem a apresentação simples e rápida de uma determinada ideia para os interessados sem a necessidade de uma codificação e permite diversas alterações devido a seu baixo custo, isto como afirma Rodrigues (2017).

Figura 23 - Exemplo de Wireframe de baixa fidelidade

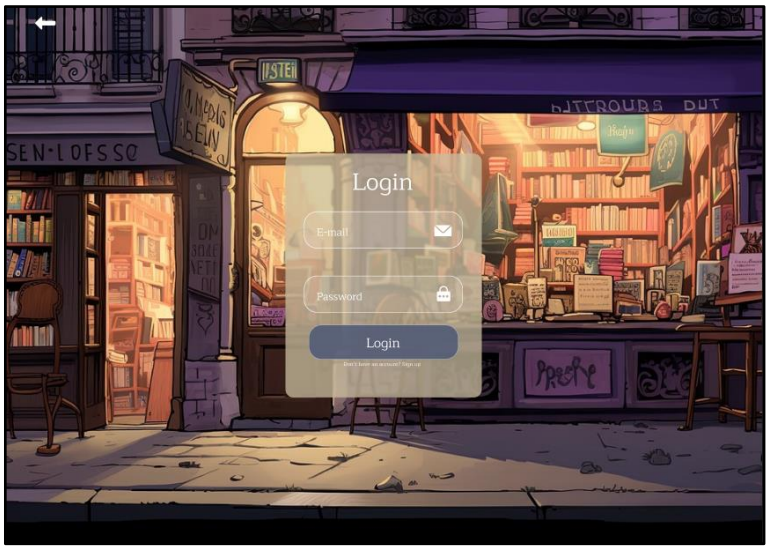


Fonte: Autoria própria,2025

2.12.2 Wireframes de alta fidelidade

Em consonância com Rodrigues (2017), a prototipação de alta fidelidade, mais conhecida como Wireframe de Alta fidelidade, utiliza-se de forma preferencial, elementos que irão estar presentes no produto final, trazendo assim um maior nível de fidelidade, além de trazer um maior desempenho para a venda do produto para a parte interessada. Contudo, este tipo de prototipação, tem como empecilho inicial o maior tempo para construção, custo mais elevado e maior dificuldade para caso haja a necessidade da alteração.

Figura 24 - Exemplo de Wireframe de alta fidelidade



Fonte: Autoria própria, 2025

2.12.3 UX (User Experience)

Segundo o Teixeira (2014), *User Exeprience*, ou UX, é um modelo de design, com o objetivo de ajustar quais ações os usuários podem realizar em um sistema, quais atividades podem ser feitas e quais serão as ordens de apresentação de cada dela e como será feito essa exposição.

2.12.4 UI (User Interface)

De acordo com Krunk (2014), o *User Interface*, traduzido para Interface Visual, ou UI, se compõe dos utensílios que o usufruidor pode interagir dentro do produto para conseguir realizar suas metas e garantir que a sua alegria seja realiza ao utilizar um produto.

Em congruência Neves (2018), durante a criação da interface visuais de sistemas, é definido como será tratado e implementado a composição dos objetos visuais que busca comunicar as interações e dados da tela, durante a manipulação dos aspectos visuais é sempre importante lembrar que o ser humano possui a capacidade de diferenciar objetos apenas olhando para a sua aparência.

2.13 UML

Criado em 1995 com a unificação de três linguagem de modelagem, a Unified Modeling Language, traduzindo para o português, Linguagem de Modelagem Unificada ou UML é uma linguagem teórica para a modelagem de softwares que seguem o modelo de orientado objeto. (Guedes,2009).

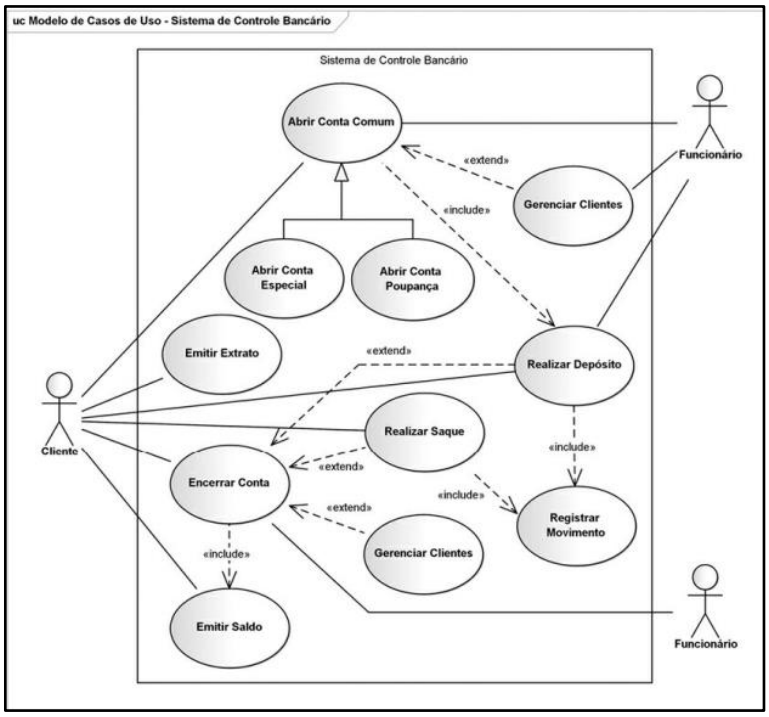
Segundo os autores Booch, Rumbaugh e Jacobson (2012), a UML é utilizada em softwares de estrutura complexa para determinar os requisitos, processos e funções de um sistema de nível simples ou avançados, além disso a UML permite definir quais serão as classes que criadas dentro do projeto e esquemas de banco de dados, a UML é o esqueleto dos projetos de software.

Consoante a Craig Larman (2005), a UML se tornou extremamente importante dentro de documentações de softwares por conta da requisição de padrões, essas normas se tornam importante no desenvolvimento criam melhores práticas para os programadores.

2.13.1 Caso de Uso

De acordo com Fowler (2005), para descrever os requisitos funcionais de um software, utilizamos a técnica do caso de uso, o caso de uso serve para apresentar quais são as interações típicas que haverá entre os usuários e o sistema, explicando em um formato de narrativa sobre as interações. Abaixo temos um exemplo de caso de uso:

Figura 25 - Exemplo de um Caso de Uso



Fonte: Guedes, 2009, p.31

O exemplo acima exemplifica o caso de uso de um sistema de controle de banco, no esquema, podemos ver que existem 3 personagens principais que no UML chamamos de atores, sendo eles, o funcionário, cliente e outro funcionário. Para um melhor entendimento vamos explicar alguns termos:

Extend: esse termo se refere a ações que ocorrem de forma opcional dentro do sistema, por exemplo, para um funcionário, é opcional que ele abra uma conta no banco para poder gerenciar os clientes

Include: Por outro lado, esse termo se refere a ações que devem acontecer caso uma outra atividade for executada, por exemplo, para um cliente poder encerrar sua conta, é obrigatório que seja emitido o saldo de sua conta.

Generalização: o termo é utilizado quando existe ações muito semelhante uma das outras, porém ainda assim são levemente diferentes e precisam ser diferenciadas, para isso, criamos um caso de uso geral e os elementos específicos serão herdados do elemento geral, por exemplo, quando o cliente escolher abrir uma conta, ele deve também escolher se ele deseja abrir uma conta especial ou uma conta poupança.

2.13.2 Documentação de Casos de Usos

De acordo com Guedes (2009), a documentação do caso de uso busca detalhar características de um caso de uso em específico, detalhes como quais são os atores que atuam sobre ele, todas as fases que devem ser realizadas tanto pelo ator e pelo sistema, quais parâmetros o caso de uso deve providos e quais restrições ou validações que deve existir neste caso de uso.

Tabela 1- Exemplo de Documentação de Caso de Uso

Nome do Caso de Uso	Fazer Login
Caso de Uso Geral	
Ator Principal	Cliente.
Atores Secundários	
Resumo	Este caso de uso descrever as etapas realizadas pelo cliente para permitir o acesso ao sistema com as suas credenciais.
Pré-Condições	O cliente deve estar cadastro no sistema.
Pós-Condições	O cliente é autenticado e redirecionado para a página inicial.
Fluxo Principal	
Ações do Ator	Ações do Sistema
1. O cliente acessa a tela de login.	
2. Informa o email e senha cadastrados.	
	3. O sistema valida as credenciais.
	4. Concede acesso ao usuário.
Restrições/Validações	1. O email fornecido deve ser um email valido.
Fluxo alternativo – Credenciais Invalidas	
Ações do Ator	Ações do Sistema
	1. O sistema percebe que os dados estão incorretos. 2. Informa ao cliente para verificar as informações fornecidas.
Fluxo de Exceção – Conta bloqueada	
Ações do Ator	Ações do Sistema

	1. O sistema visualiza que os dados fornecidos estão bloqueados no banco de dados. 2. Informa e orienta o cliente a criar uma nova conta.

Explicação da documentação acima:

Incialmente é definido qual é o nome do caso de uso que está sendo documentado, no caso o nosso caso de uso se chama “Fazer Login”. Após isso é definido qual o seu caso de uso geral, que serve quando o caso de uso é uma especialização ou descendente de um caso de uso pai, o que não é o caso nesse exemplo.

Posteriormente é exibido quem são o ator principal do caso de uso juntamente, se necessário, é colocado quem é o ator secundário do caso de uso abaixo dos atores existe uma breve descrição sobre a ação em especifica que o caso de uso está realizando

Seguidamente, é definido quais são as pré-condições, que são todas as condições que ocorrem antes do caso de uso que está sendo explicado no momento, junto de suas pós-condições, que são as ações que vão ocorrer após a ação do caso de uso acontecer.

Vale ressaltar que existe também uma lista de restrições e validações sobre o caso de uso, isso são todas as regras ou condições que impedem ou permitem a interação do caso de uso com os atores.

Seguindo o modelo do exemplo, o ator principal acessa a tela de login e informa o seu e-mail e senha que já estão cadastrado no banco de dados da aplicação, após isso o sistema consulta se os dados apresentados estão de acordo com que esta salvo no banco de dados do sistema e se estiver correto a aplicação leva para a página principal.

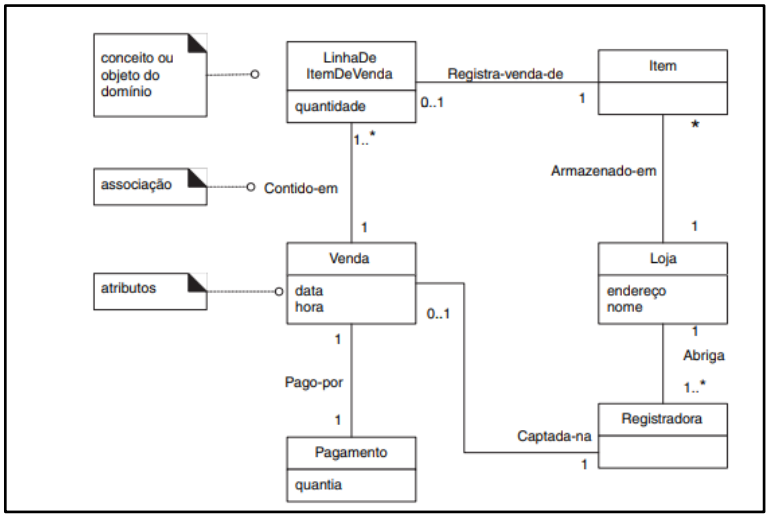
Assim sendo, existe também o fluxo alternativo, que explica uma ação variada do fluxo principal do caso de uso que não é ideal, o fluxo alternativo que está no exemplo descreve quando as credenciais estiverem incorretas.

E para finalizar, a documentação conta com o fluxo de exceção que trabalha com as chances de o sistema não conseguir continuar com a ação, devido a violação de alguma regra de negócio que a aplicação possui.

2.13.3 Diagrama de Classe

Segundo Barbosa e Sarro (2013), diagramas de classes retratam quais serão os itens, funções e como funcionará os relacionamentos entre eles e respectivamente suas restrições, fora isso, esses diagramas conseguem exemplificar as operações de uma classe no sistema. Vamos exemplificar melhor abaixo:

Figura 26 - Exemplo de um Diagrama de Classe



Fonte: Larman, (2000), p. 53

Vemos acima um exemplo de um diagrama de classe de um sistema de Pagamento e Vendas, para poder entender melhor essa estrutura, vamos explicar certos termos:

Classe: representa um modelo dentro do sistema que irá definir quais sãos os dados e métodos, por exemplo, na imagem temos a classe “Loja”.

Atributo: também chamado de variáveis, representam as propriedades de uma classe, são nos atributos que vamos guardar os dados, por exemplo, na classe “Loja”, temos os atributos “endereço” e “nome”.

Método: são as funções e ações que a classe pode fazer no sistema, na imagem acima, não temos um método, mas poderíamos ter o método “FazerPagamento ()” (os métodos são representados por parênteses) dentro da classe Pagamento.

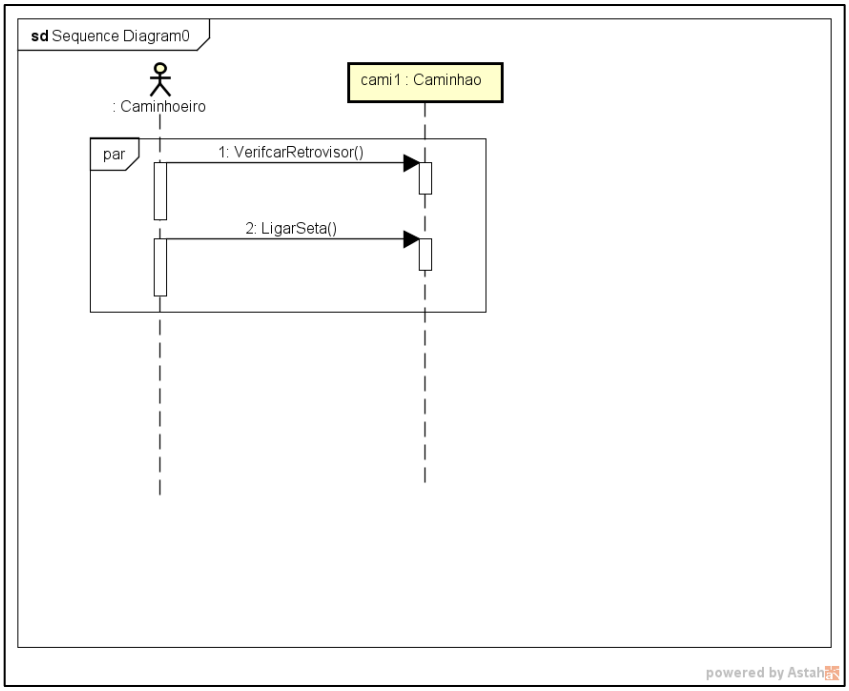
Associação: Exibe qual o relacionamento entre as classes, junto do relacionamento deve ter também a sua multiplicidade, que indica quantos elementos podem transmitidos de uma classe a outra, no exemplo acima a classe “Venda” se relaciona com a classe “Pagamento” com uma multiplicidade de 1 para 1, ou seja, pelo menos um elemento está sendo transmitido.

2.13.4 Diagrama de Sequência

Segundo Fowler (2000), um diagrama de sequência aponta o comportamento de um cenário em específico, analisando diversos exemplos de objetos e avisos que são transmitidas entre si dentro de um caso de uso.

Consoante a Larman (2000), o diagrama de sequência é um semblante que mostra quais são as ações que atores externos podem gerar, deve exibir toda a ordem e respectivos eventos entre os sistemas.

Figura 27 - Exemplo de diagrama de sequência



Fonte: Autoria Própria, 2025

Acima é possível um exemplo de diagrama de sequência que será explicado logo abaixo:

É possível observar que existe o ator “Caminhoneiro” que irá realizar duas operações que podem ser simultâneas ou não sobre que são representadas pelo

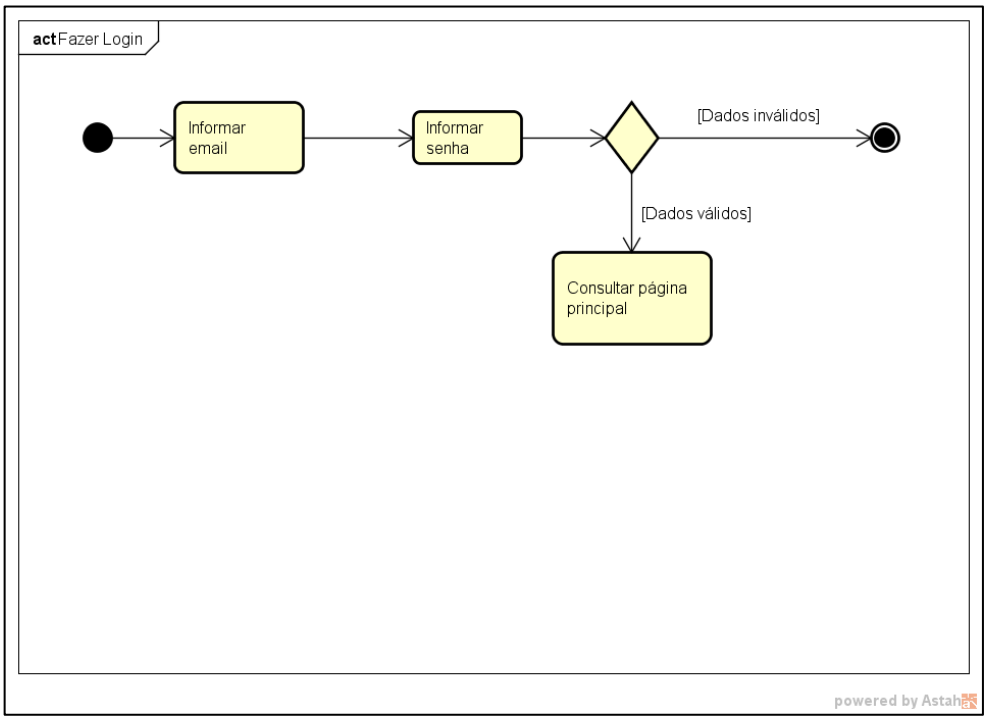
retângulo “par” sobre o objeto “cami1” que faz parte da classe “Caminhão”, as operações realizadas sobre o veículo em específico são elas “VerificarRetrovisor” e “LigarSeta”.

2.13.5 Diagrama de Atividade

Segundo Booch, Rumbaugh e Jocobson (2012), o diagrama de atividade exibe todo o fluxo de uma atividade para outra mostrando especialmente a sua concorrência e as derivações de controle dessas atividades.

Em congruência com Guedes (2009), os diagramas de atividades destacam toda a sequência e condições necessárias para controlar todos os comportamentos de baixo nível, fazendo com que dessa forma, esse diagrama em específico seja um dos mais detalhistas entre os diagramas do UML.

Figura 28 - Exemplo de um diagrama de atividade



Fonte: Autoria Própria, 2025

O diagrama acima exemplifica um diagrama de atividade através de um fluxo para fazer login em uma aplicação, abaixo será explicado mais profundamente esse diagrama:

Primeiramente o diagrama inicia aguardando que o usuário informe o email de sua conta e seguidamente apresente a sua senha, após isso, é iniciado um nó de decisão onde caso os dados informados sejam inválidos, o fluxo é encerrado ou caso os dados estejam corretos, a sequência irá levar o usuário para a página principal da aplicação.

2.14 Impressão 3D

Como explica Volpato (2017), a manufatura aditiva, ou impressão 3D, é o processo de construção de objetos por camadas. Primeiramente o objeto 3D é criado digitalmente e, em seguida, é fatiado em camadas 2D em um software próprio para isso, criando um arquivo onde a impressora interpreta e imprime camada por camada.

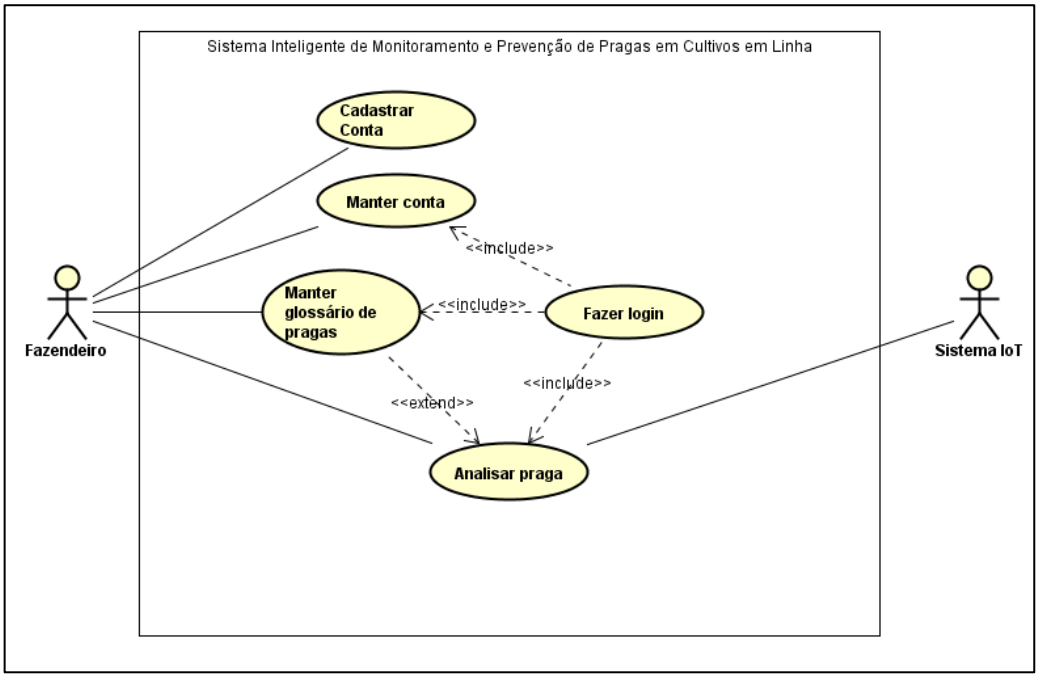
3 DESENVOLVIMENTO

O Capítulo a seguir explica os passos necessários para criar o sistema Agro G.E.S.F. A explicação seguirá a ordem de apresentar primeiro as documentações, sustentada pelo livro “UML 2” de Gilleanes T.A. Guedes, onde é exibido os diagramas de caso de uso, sequência, atividade e máquina de estado. Após isso, será explicado todo o processo de configuração do IoT Raspberry PI, implementação e a sua utilização junto da explicação de como foi o processo de modelagem da case do projeto. Seguido a isso, o texto vai explicar a metodologia usada para a criação da aplicação em desktop e as suas posteriores telas.

3.1 Diagrama de Caso de Uso

Abaixo é mostrada a representação do diagrama de caso de uso, onde é exibido os objetivos entre os atores Fazendeiro e Sistema off Home com a aplicação. Ao todo, os casos de uso que foram adicionados somados são 5 que explica as funcionalidades disponíveis no software do Agro G.E.S.F

Figura 29 - Diagrama de Caso de Uso do Agro G.E.S.F



Fonte: Autoria Própria, 2025

3.2 Documentação do Caso de Uso

A documentação dos casos de uso é necessária pois explica de uma forma mais precisa todas as etapas que são percorridas tanto pelo usuário e pelo sistema para realizar o caso de uso específico e as suas ações alternativas. Além disso, este capítulo irá abordar também os requisitos funcionais, que são qualquer funcionalidade que os devidos atores Fazendeiro e Sistema Iot podem realizar dentro do sistema, além disso, será explicado também os requisitos não funcionais, que ao contrário dos requisitos funcionais, são características mais relacionadas a desempenho, segurança e usabilidade do sistema, e por fim, será apresentado as regras de negócio que são as características que definem e diferenciam o produto e a empresa AGRO-G.E.S.F de outras companhias.

Requisitos Funcionais:

- Capturar as imagens das folhas de forma automática
- Processar localmente as imagens com CNN
- Enviar os resultados da análise para o aplicativo
- Receber os alertas das detecções das pragas/doenças
- Apresentar o feedback da confiabilidade da precisão de forma visual
- Consultar o glossário de pragas e doenças
- Manter dados da conta
- Confirmar ou rejeitar as detecções
- Visualizar o histórico de detecções
- Visualizar mapa de calor das detecções
- Armazenar as informações no banco de dados
- Funcionar em modo offline
- Cadastrar conta do usuário
- Fazer login no sistema
- Sincronização automática entre desktop e mobile

Requisitos Não Funcionais:

- Acessibilidade econômica
- Baixo consumo de energia
- Processo embarcado
- Operação offline com sincronização posterior
- Tempo de detecção em tempo real
- Exibição clara da confiança da predição
- Eficiência computacional
- Usabilidade para baixa alfabetização digital
- Robustez em ambiente rural
- Escalabilidade

Regras de Negócio:

- O sistema deve identificar apenas pragas e doenças;
- O sistema deve ser utilizado apenas em plantações de linhas com plantas de baixa altura;
- O sistema deve tirar a foto e após isso deve fazer a análise;
- O sistema deve identificar de forma semiautomática as pragas e doenças.

Segue abaixo a documentação dos casos de uso:

Tabela 2 - Documentação do Caso de Uso Cadastrar Conta

Nome do Caso de Uso	Cadastrar Conta
Ator Principal	Fazendeiro
Resumo	Este caso de uso descreve as etapas para criar uma conta no sistema, informando os dados necessários para criar o acesso
Pré-Condições	
Pós-Condições	O usuário terá acesso a todas as funcionalidades que o sistema oferece
Cenário Principal	
Ações do Ator	**Ações do Sistema**
1. O fazendeiro acessa o sistema e seleciona a opção de cadastrar conta	
	2. O sistema solicita as informações necessárias para criar a conta
3. O Fazendeiro fornece os dados	
	4. O sistema valida as informações e registra a nova conta
Restrições/validações	

Fonte: Autoria própria, 2025

Tabela 3 - Documentação do Caso de Uso Fazer Login

Nome do Caso de Uso	Fazer Login
Ator Principal	Fazendeiro
Resumo	Esse caso de uso descreve todas as etapas necessárias para permitir a autenticação no sistema
Pré-Condições	
Pós-Condições	O usuário terá acesso a todas as funcionalidades que o sistema oferece
Cenário Principal	
Ações do Ator	**Ações do Sistema**
1. O fazendeiro informa o nome de usuário e a sua senha	
	2. O sistema valida as credenciais
	3. Os dados são autenticados e acontece o redirecionamento para a tela principal
Restrições/validações	

Fonte: Autoria própria, 2025

Tabela 4 - Documentação do Caso de Uso Manter Conta

Nome do Caso de Uso	Manter Conta
Ator Principal	Fazendeiro
Resumo	Caso de uso que exemplifica as etapas necessárias que permite realizar as ações de criar, editar, visualizar e deletar uma conta
Pré-Condições	
Pós-Condições	
Cenário Principal	
Ações do Ator	**Ações do Sistema**
1. O fazendeiro acessa a página de perfil no sistema	
	2. O sistema exibe os dados que foram cadastrados
3. O fazendeiro realiza as alterações e confirma	
	4. O sistema salva as modificações
Restrições/validações	1. Verificar se foi realizado o login

Fonte: Autoria própria, 2025

Tabela 5 - Documentação do Caso de Uso Manter Glossário de Pragas

Nome do Caso de Uso	Manter Glossário de Pragas
Ator Principal	Fazendeiro
Resumo	Caso de uso feito para explicar as ações que permitem cadastrar, visualizar, alterar ou remover registros de pragas
Pré-Condições	
Pós-Condições	O usuário terá acesso a todas as funcionalidades que o sistema oferece
Cenário Principal	
Ações do Ator	**Ações do Sistema**
1. O fazendeiro informa o nome de usuário e a sua senha	
	2. O sistema valida as credenciais
	3. Os dados são autenticados e acontece o redirecionamento para a tela principal
Restrições/validações	

Fonte: Autoria própria, 2025

Tabela 6 - Documentação do Caso de Uso Analisar Praga

Nome do Caso de Uso	Analisar Praga
Ator Principal	Fazendeiro
Ator Secundário	Sistema IoT
Resumo	Este caso de uso explica todas as etapas que permite realizar a análise com base em dados coletados pelo Sistema Iot e pela visualização do Fazendeiro, que irá visualizar uma foto enviada pelo dispositivo e informar se o que foi enviado é uma praga ou doença
Pré-Condições	
Pós-Condições	
Cenário Principal	
Ações do Ator	**Ações do Sistema**
1. O fazendeiro informa o nome de usuário e a sua senha	
	2. O sistema valida as credenciais
	3. Os dados são autenticados e acontece o redirecionamento para a tela principal
Restrições/validações	

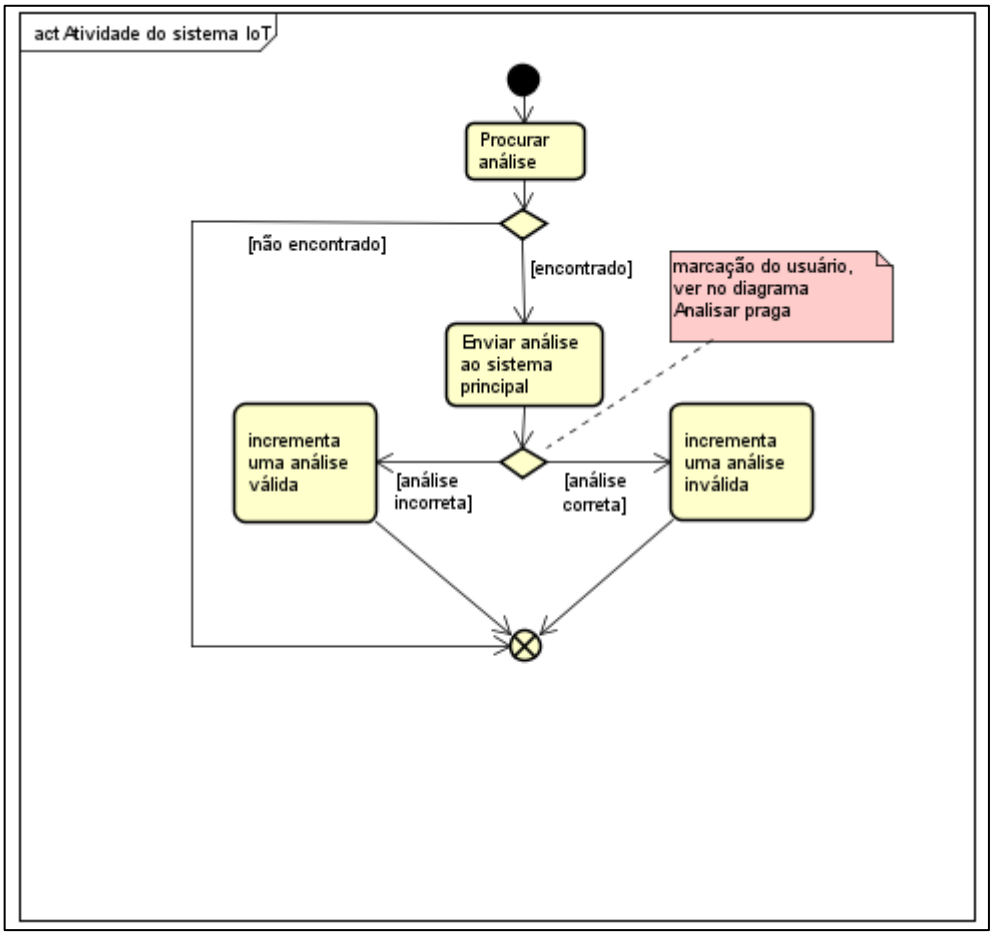
Fonte: Autoria própria, 2025

3.3 Diagrama de Atividade

Neste tópico, é mostrado os diagramas de atividade do projeto Agro G.E.S.F, onde é utilizado para moldar todas a atividades possíveis dentro do sistema em termos de nível de máquina, isso permite destacar todas as regras, etapas e requisições para a conclusão da ação específica.

Este Diagrama explica todas as atividades que o dispositivo físico pode realizar:

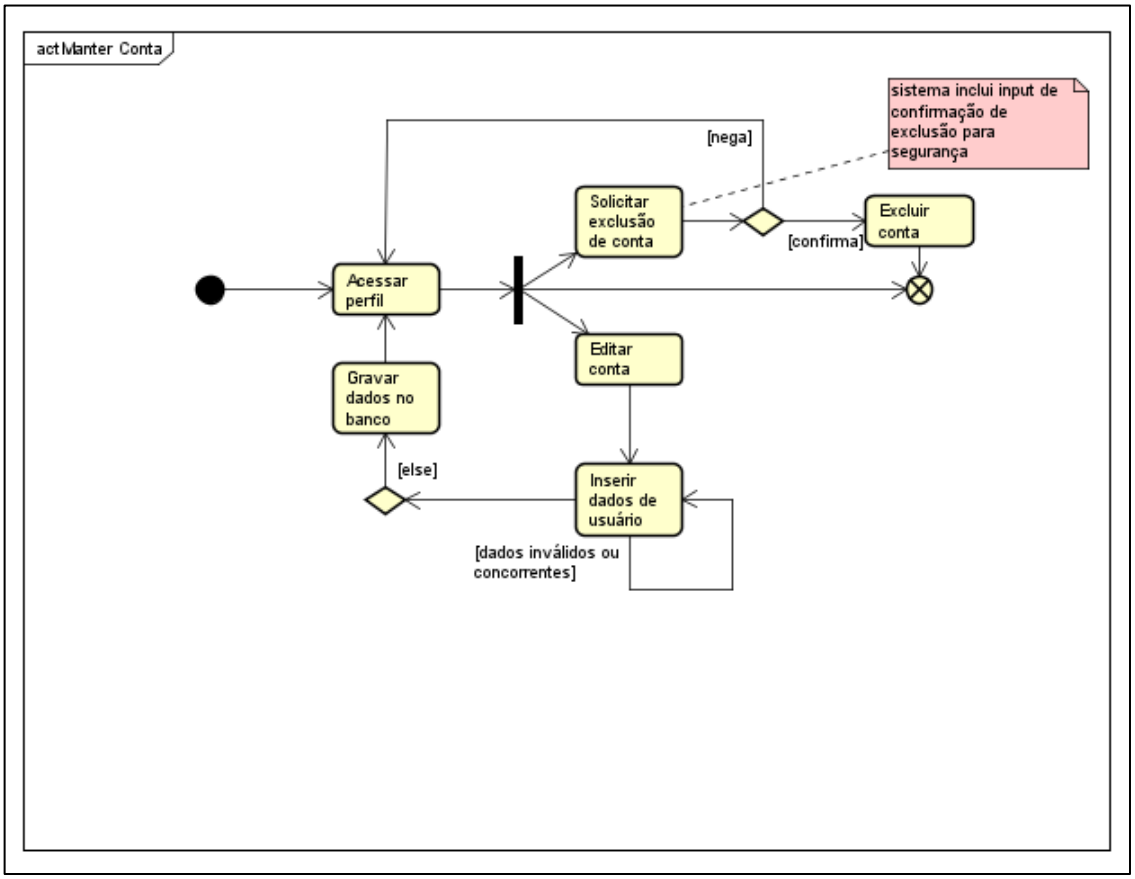
Figura 30 - Diagrama de Atividade do Sistema Iot



Fonte: Autoria Própria, 2025

Esse diagrama explica os passos necessários pelo software para poder acessar o perfil ou cadastrar uma nova conta:

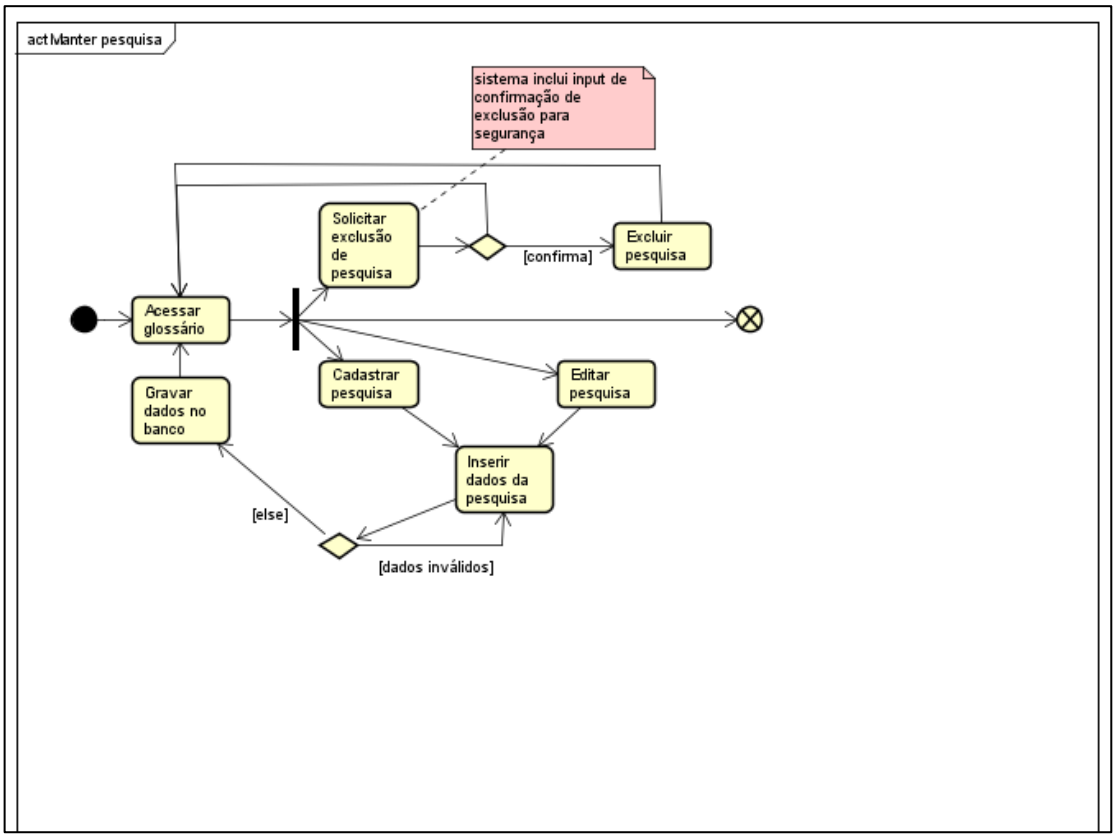
Figura 31 - Diagrama de Atividade de Manter Conta



Fonte: Autoria Própria, 2025

Esse diagrama demonstra todas as ações que o sistema pode fazer quando ele está realizando uma pesquisa sobre a praga ou doença detectada:

Figura 32 - Diagrama de Atividade de Manter Pesquisa

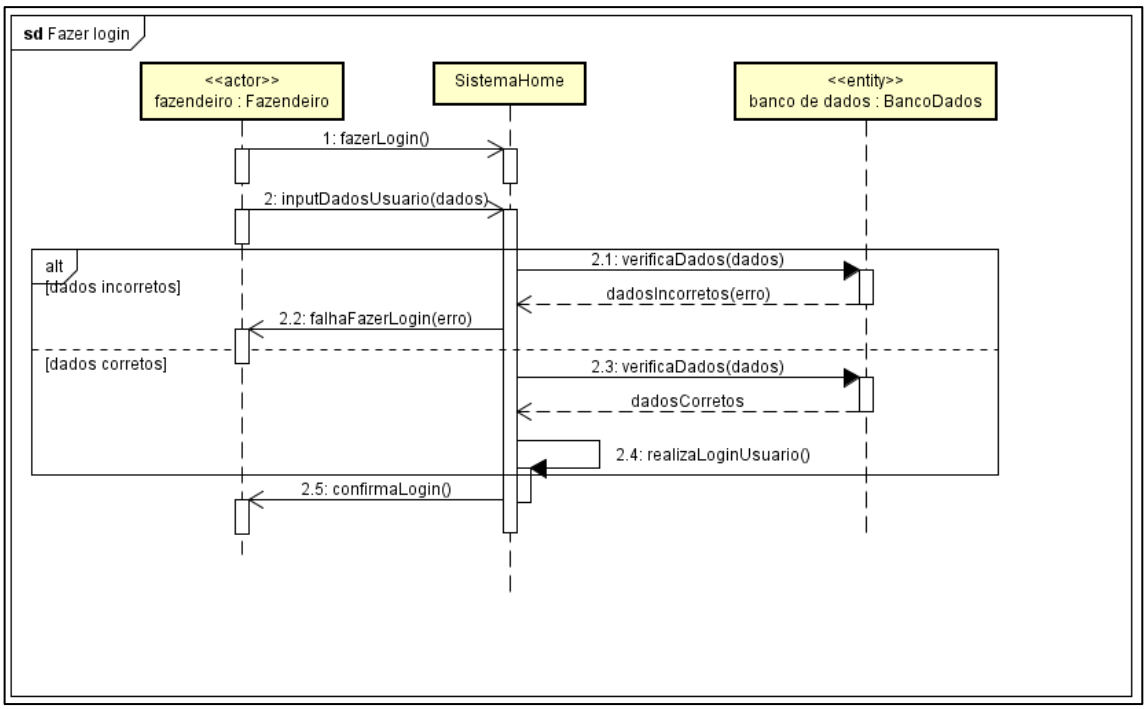


Fonte: Autoria Própria, 2025

3.4 Diagrama de Sequência

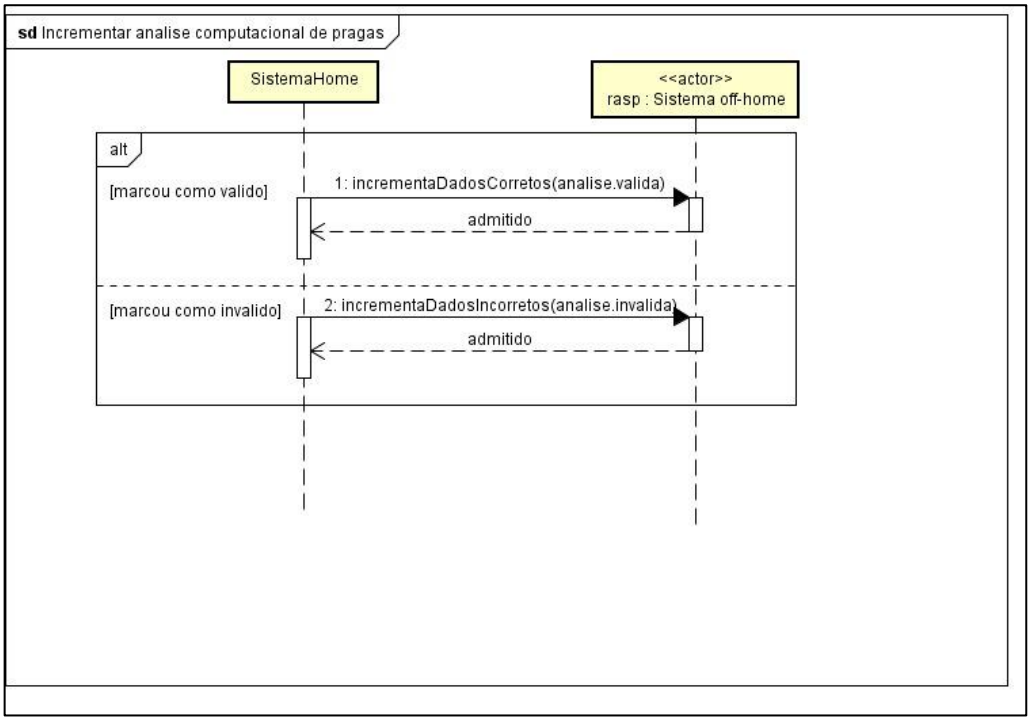
Neste módulo, é apresentado todos os diagramas de sequência do projeto, onde explicam as ações da aplicação, esta parte tem o objetivo de explicar com clareza e de forma contínua todo o percurso de dados e mensagens que serão trocadas, vale ressaltar que o diagrama de sequência tem uma forte conexão com o diagrama de caso de uso, visto que é partir dele que se pode criar o diagrama de sequência.

Figura 33 - Diagrama de Sequência de Fazer Login



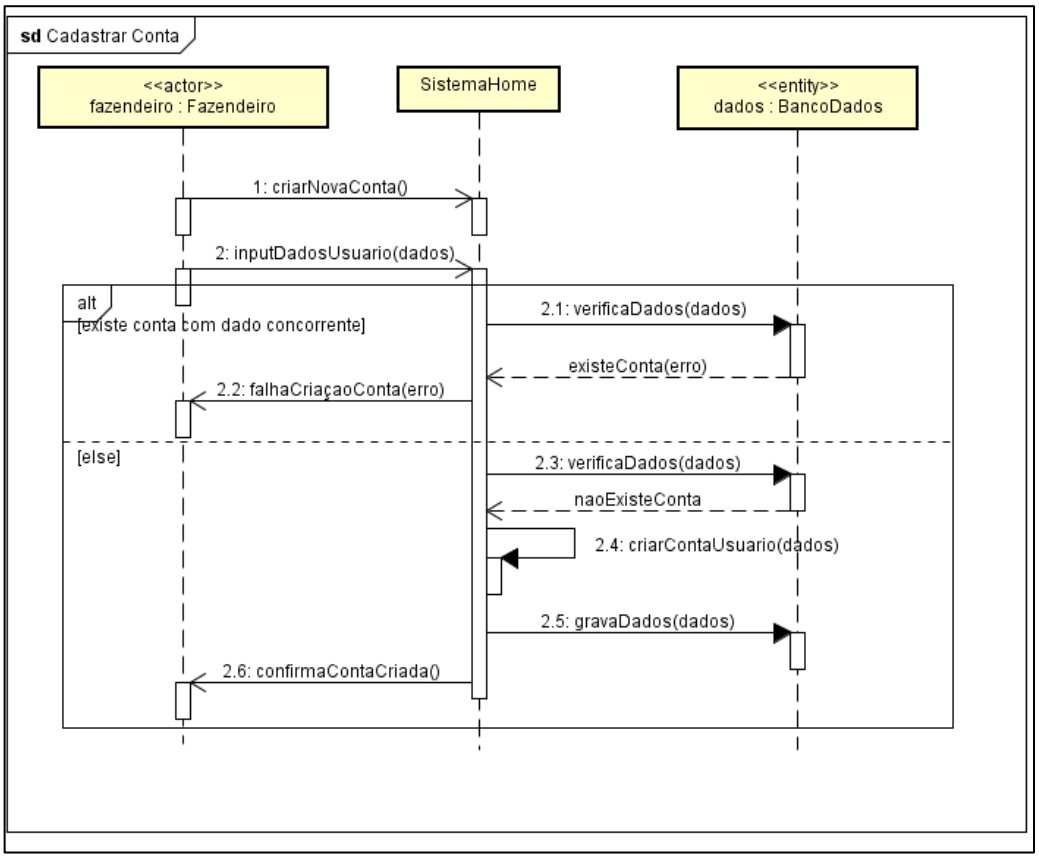
Fonte: Autoria Própria, 2025

Figura 34 - Diagrama de Sequência de Incrementar Análise Computacional



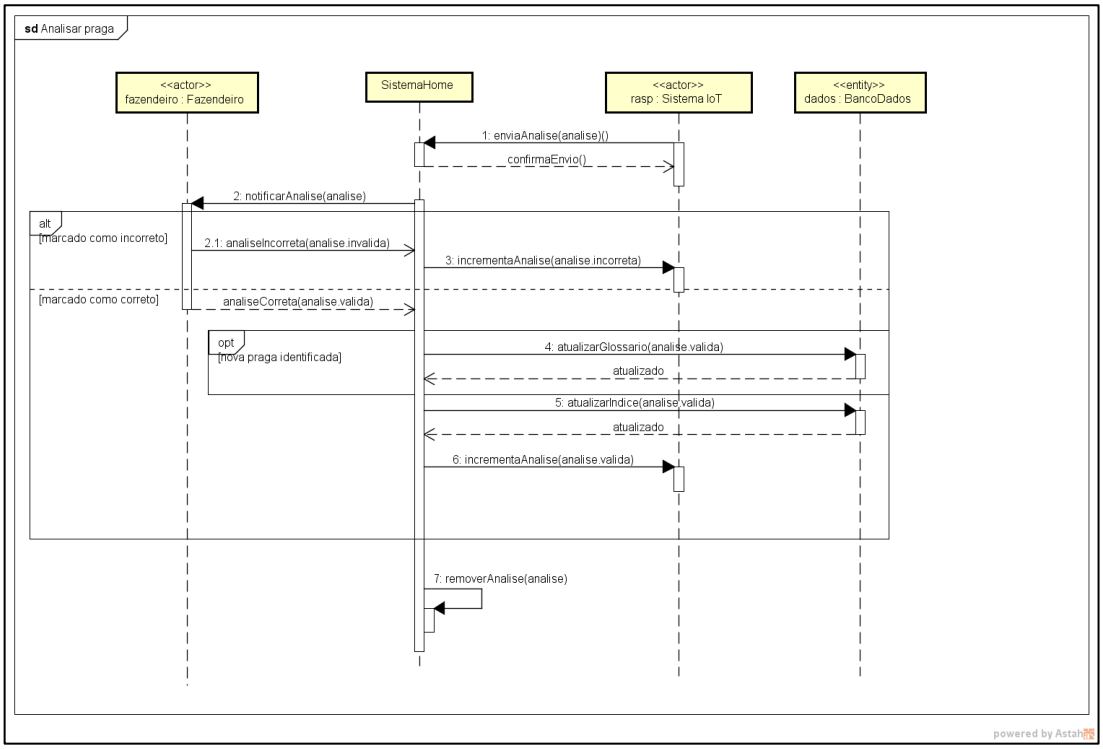
Fonte: Autoria Própria, 2025

Figura 35 - Diagrama de Sequência de Realizar Cadastro



Fonte: Autoria Própria, 2025

Figura 36 - Diagrama de Sequência de Validar Análise



Fonte: Autoria Própria, 2025

3.5 Implementação do IoT

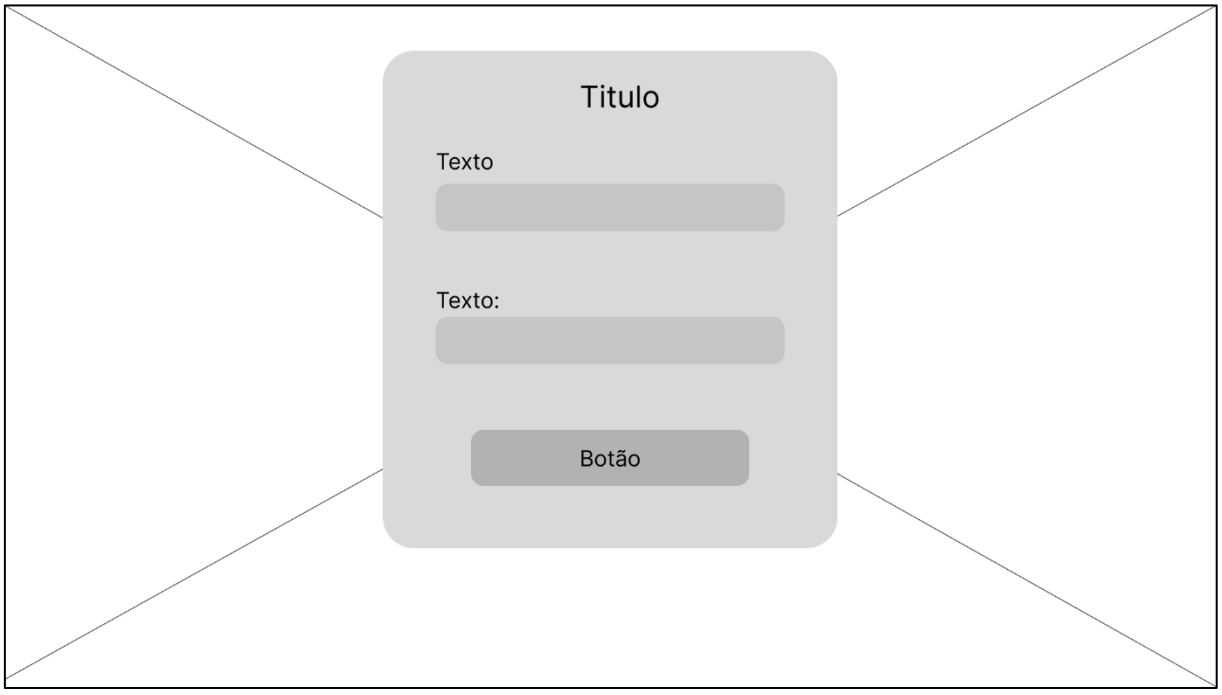
O seguinte tópico demonstrará como ocorreu a implementação do software no Raspberry PI. Utilizando-nos do Raspberry Pi Imager, instalamos a versão oficial do Raspberry Pi OS, um sistema operacional baseado em Linux o qual fornece um bom desempenho, mas sem sacrificar a interface gráfica, em um SD Card classe 10 de 64GB para obter o maior desempenho possível nessa configuração. Para configurar a comunicação entre o Raspberry e o celular, foi utilizado o pacote hostapd, dnsmasq, dhcpcd e o rfkill, estes quais são pacotes padrão do Linux focados para a comunicação de dados por meio de uma rede wireless, e em conjunto com estes pacotes, há um arquivo em python o qual transforma o Raspberry em uma espécie de servidor e permite que outros dispositivos se conectem a ele e possam receber dados. Por final, foi criado um arquivo de inicialização automática desse servidor, se utilizando de um arquivo do tipo .sh, pois assim, toda vez que o sistema ligar, o Raspberry irá rodar esse arquivo e consequentemente iniciar os demais serviços de forma automática.

3.6 Desenvolvimento das Telas para Desktop e Celular

Neste tópico, serão exibidas e explicadas as páginas, que são os elementos visuais, que o usuário terá acesso, primeiramente, será explicado as telas desenvolvidas para computador e após isso, o capítulo irá abordar sobre as páginas pensadas e desenvolvidas para celulares. Tanto as páginas de desktop e smartphones foram desenvolvidas levando em consideração os *wireframes* de baixa e alta qualidade que foram prototipados com o ambiente de desenvolvimento de design Figma, onde exibe com confiabilidade de como será cada interface do software. A aplicação possui 7 telas para computador e 6 páginas para celular, onde todas foram pensadas utilizando métodos UX e UI para melhorar a usabilidade para os pequenos agricultores que podem não possuir um alto conhecimento para navegar entre páginas.

Primeiramente o capítulo irá explicar as páginas referente a desktop e após isso será apresentado as telas para celular. A primeira interface que será mostrada, diz respeito a página que irá aparecer quando a aplicação for iniciada, que consiste em fazer o login ou criar uma conta se necessário, na área de login é necessário colocar o nome de seu usuário e sua senha, enquanto na parte de criar uma conta, é necessário informar um nome de usuário, email, senha e por uma questão de segurança, é necessário informar novamente a senha para poder ser verificado se em ambas as caixas de texto, as senhas estão iguais.

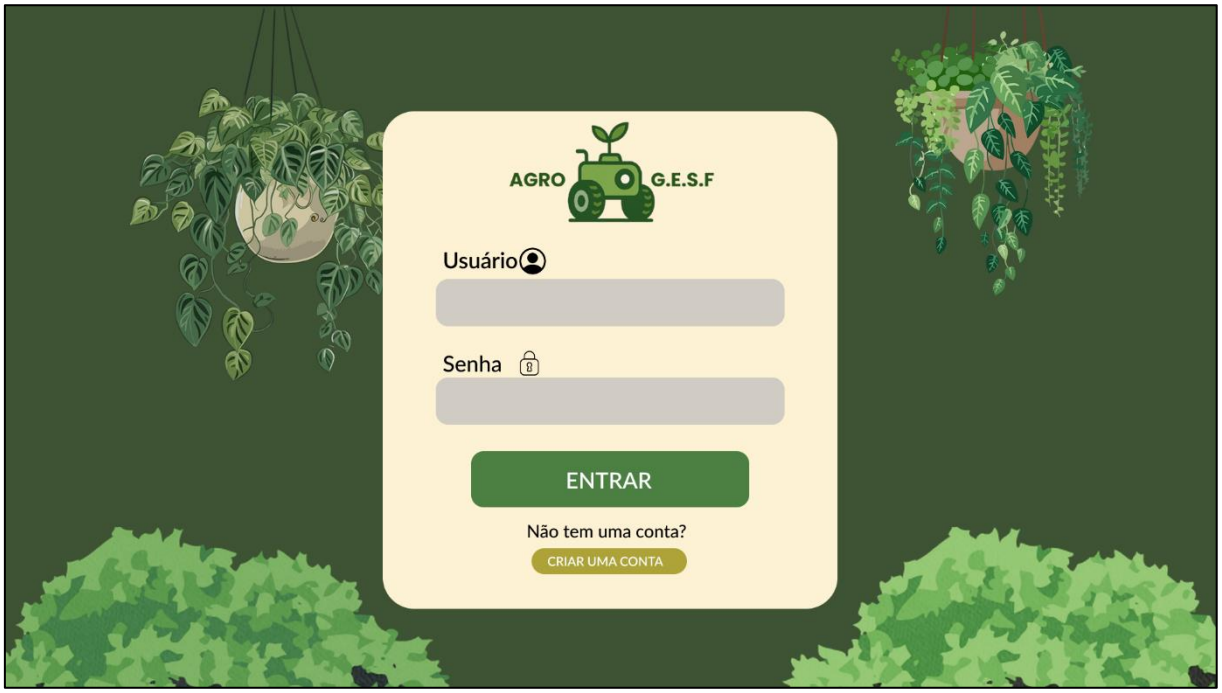
Figura 37 - Wireframe de Baixa Fidelidade: Tela de Login



Fonte: Autoria Própria, 2025

Agora, é exibido o *wireframe* de alta qualidade:

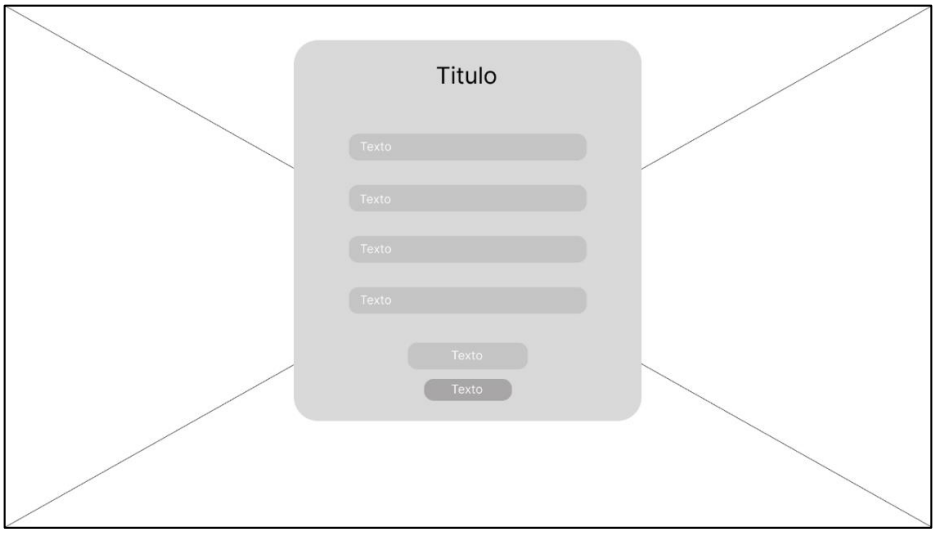
Figura 38 - Wireframe de Alta Fidelidade: Tela de Login



Fonte: Autoria Própria, 2025

O segundo *wireframe,* apresenta a tela de criar uma conta para o usuário. Nesta tela é exibido as informações necessárias para se criar uma conta, sendo elas, um nome de usuário, email, senha e por questões de segurança, é necessário que o agricultor informe a mesma senha novamente abaixo para evitar problemas como erro de digitação e garantir que o usuário realmente saiba a senha, após o cadastro, o cliente pode voltar a tela de login e entrar normalmente.

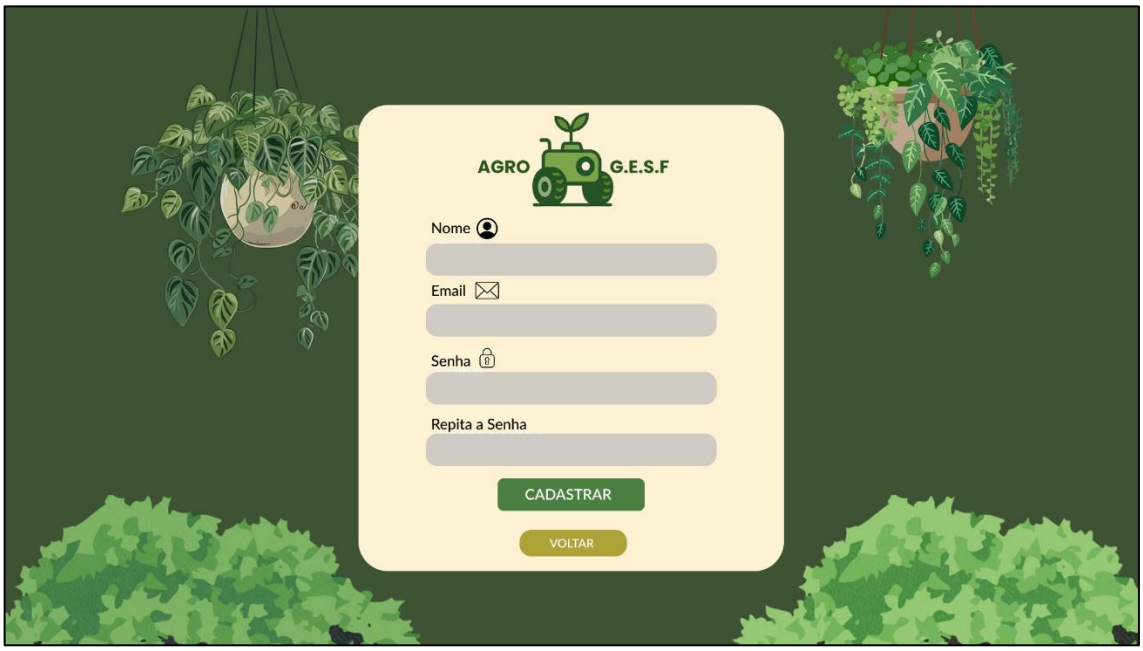
Figura 39 - Wireframe de baixa fidelidade: Tela de Cadastro



Fonte: Autoria Própria, 2025

Sequencialmente, é exibido o diagrama de alta fidelidade:

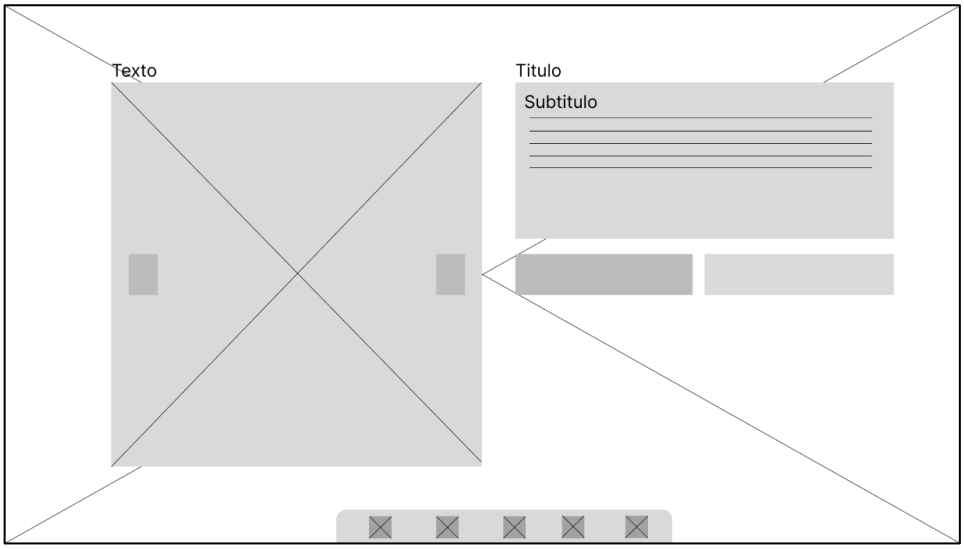
Figura 40 - Wireframe de alta fidelidade: Tela de Cadastro



Fonte: Autoria Própria, 2025

A próxima tela, sendo essa a principal de todo o software após fazer login. Esta tela é onde apresenta o momento em que um sinal de praga é identificado, no momento que algo é detectado, é exibido na tela uma foto do que foi encontrado, junto de uma descrição da doença ou infecção específica, abaixo do texto, existe dois botões onde através deles o fazendeiro pode dizer para o sistema se o que foi mostrado é uma praga ou não. Além disso, existe um menu inferior que leva para as telas de glossário, gráfico de detecções, perfil e sobre a equipe do projeto, telas essas que serão explicadas com o decorrer do capítulo.

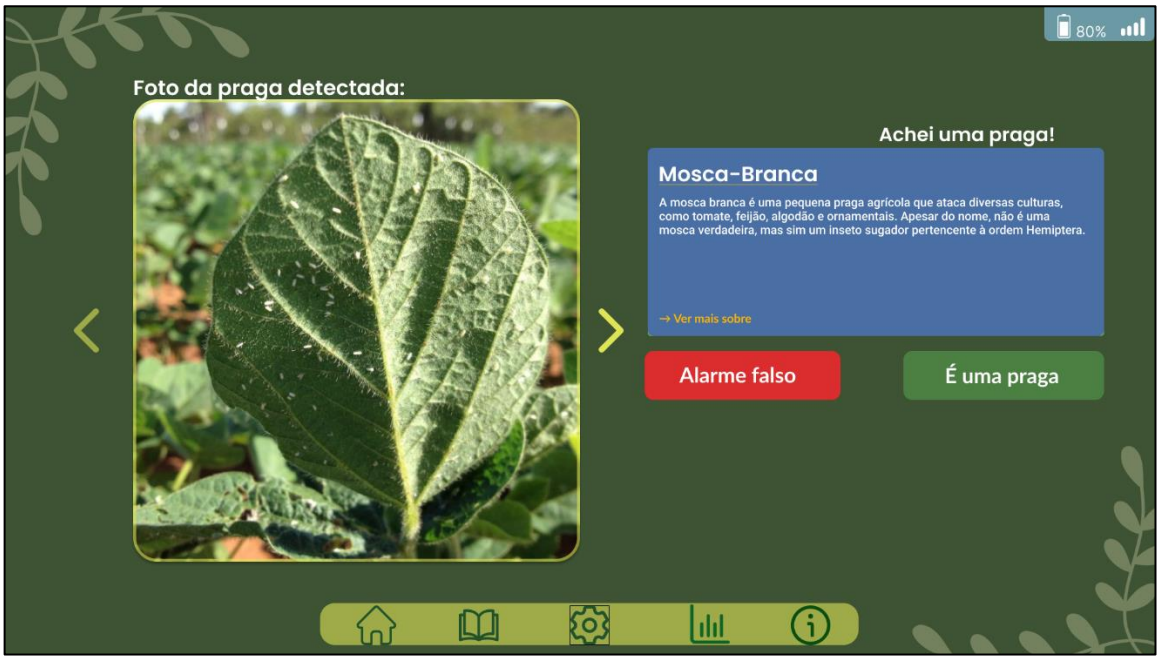
Figura 41 - Wireframe de baixa fidelidade: Tela de Dashboard



Fonte: Autoria Própria, 2025

Agora, será apresentado o *wireframe* de alta fidelidade:

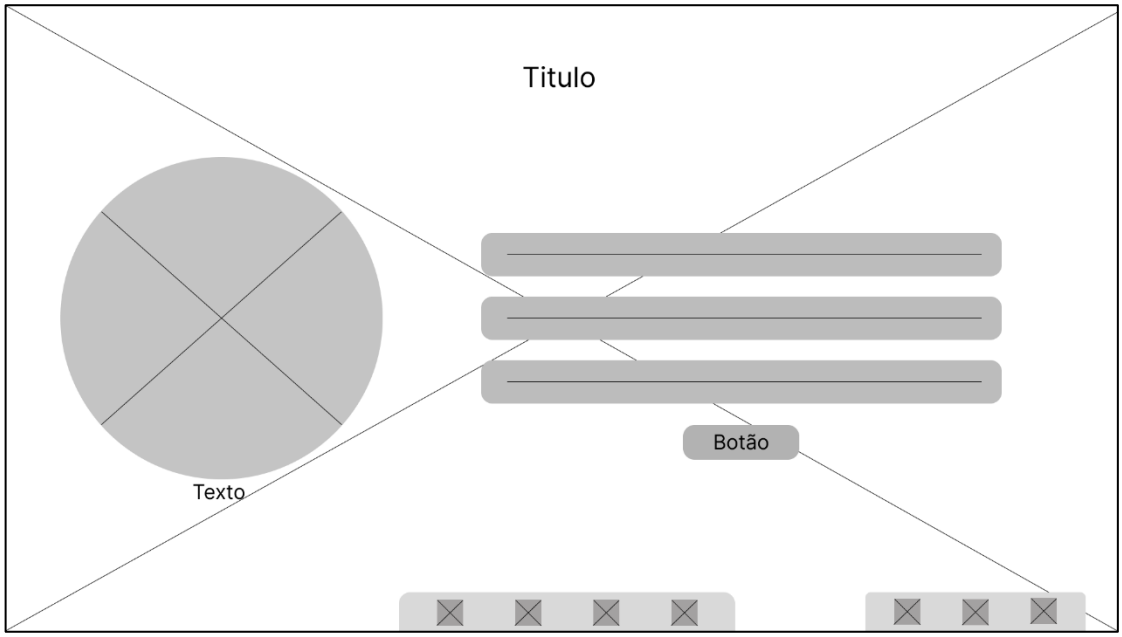
Figura 42 - Diagrama de alta fidelidade: Tela de Dashboard



Fonte: Autoria Própria, 2025

Em seguida, será explicada a tela de perfil, esta tela o usuário pode visualizar e editar os seus dados que ele inseriu na tela de cadastro, além disso também possível colocar uma foto de usuário.

Figura 43 - Wireframe de baixa de fidelidade: Tela de Perfil



Fonte: Autoria Própria, 2025

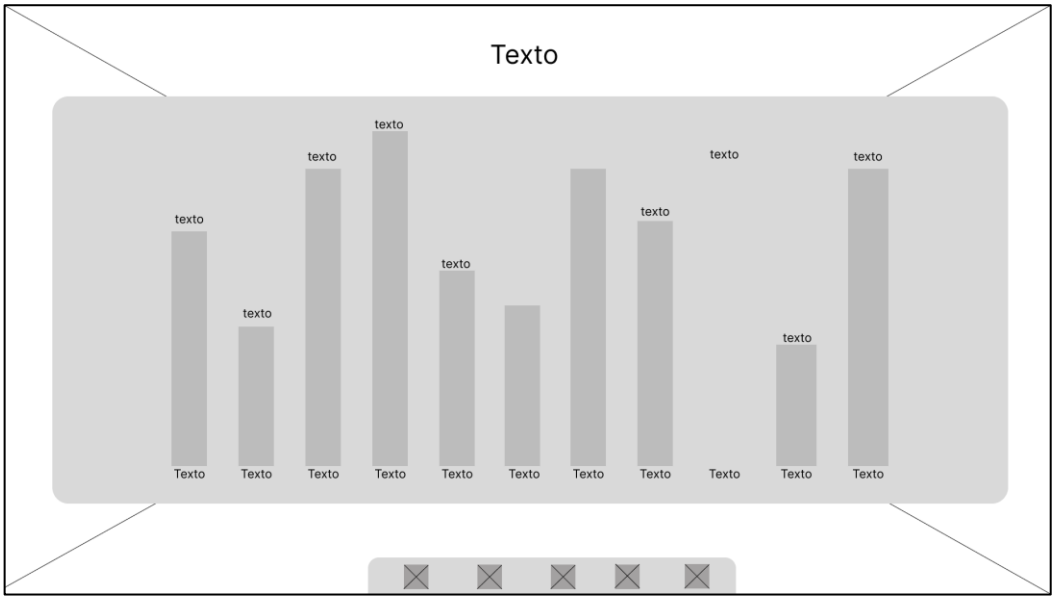
Em seguida, é exibido o *wirefeame* de alta fidelidade:

Figura 44 - Wireframe de alta fidelidade: Tela de Perfil

Fonte: Autoria Própria, 2025

A próxima tela que será explicada aqui é a tela de gráfico de pragas, nesta tela, é exibido um gráfico de todas as pragas ou doenças que já foram identificadas separados por meses.

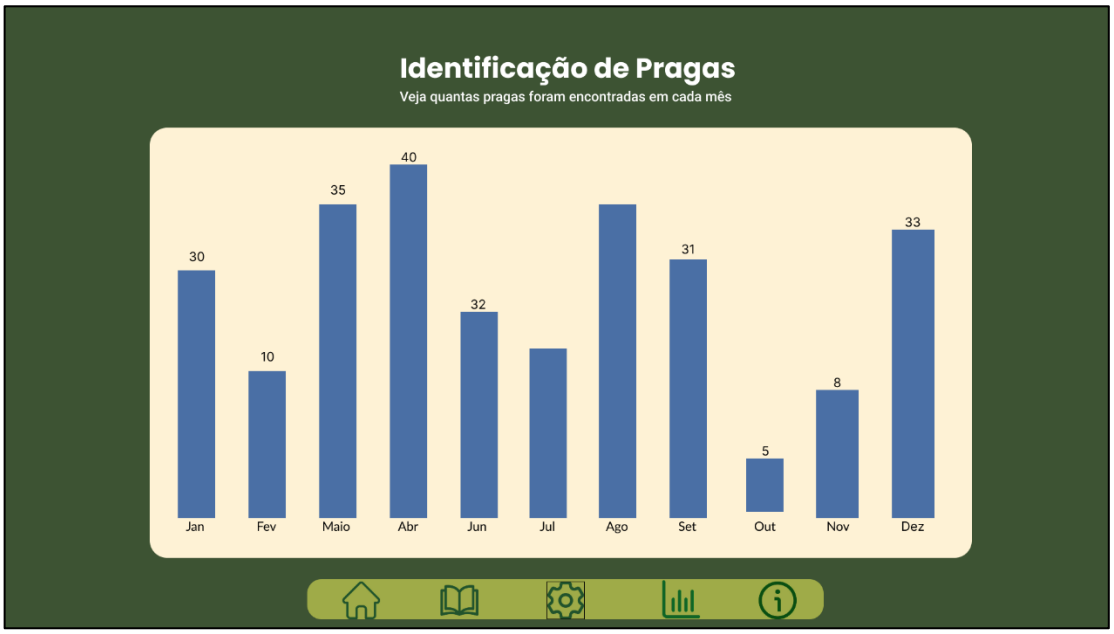
Figura 45 - Wireframe de baixa fidelidade: Tela Gráfico de Pragas



Fonte: Autoria Própria, 2025

Abaixo, é apresentado a tela com as cores, imagens e texto necessários:

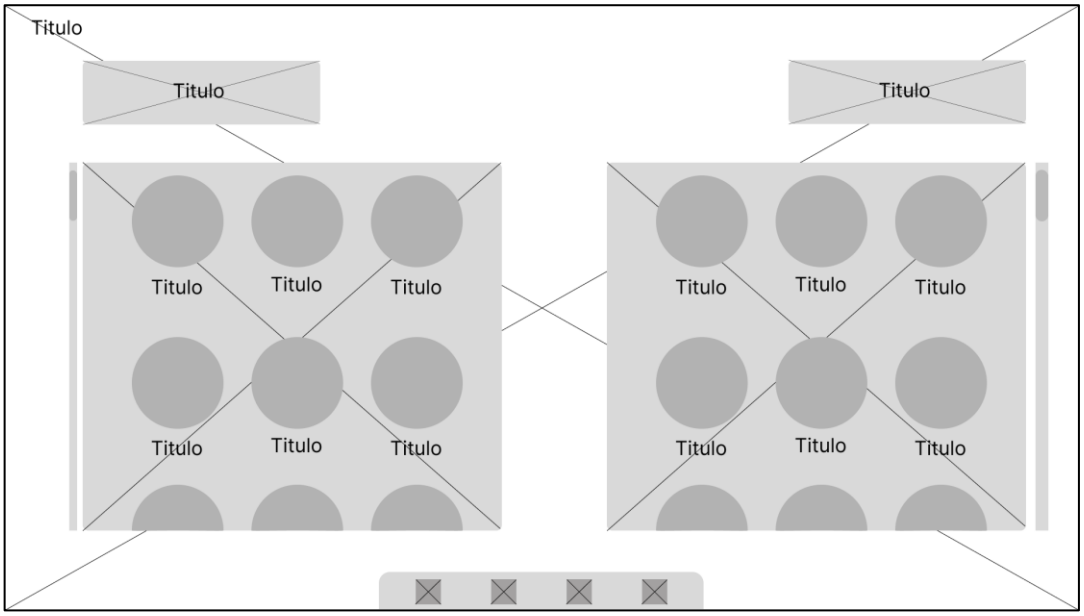
Figura 46 - Wireframe de alta fidelidade: Tela de Gráfico de Pragas



Fonte: Autoria Própria, 2025

Indo para o próximo e penúltimo *wireframe*, sendo a tela de glossário de pragas, onde nesta página, é possível visualizar quais pragas já foram detectadas pelo dispositivo, onde essas análises ficam salvas no banco de dados, e exibidas aqui para que o pequeno agricultor possa ter uma visualização de informações como nome, foto e descrição de toda praga e doença encontradas em suas plantas.

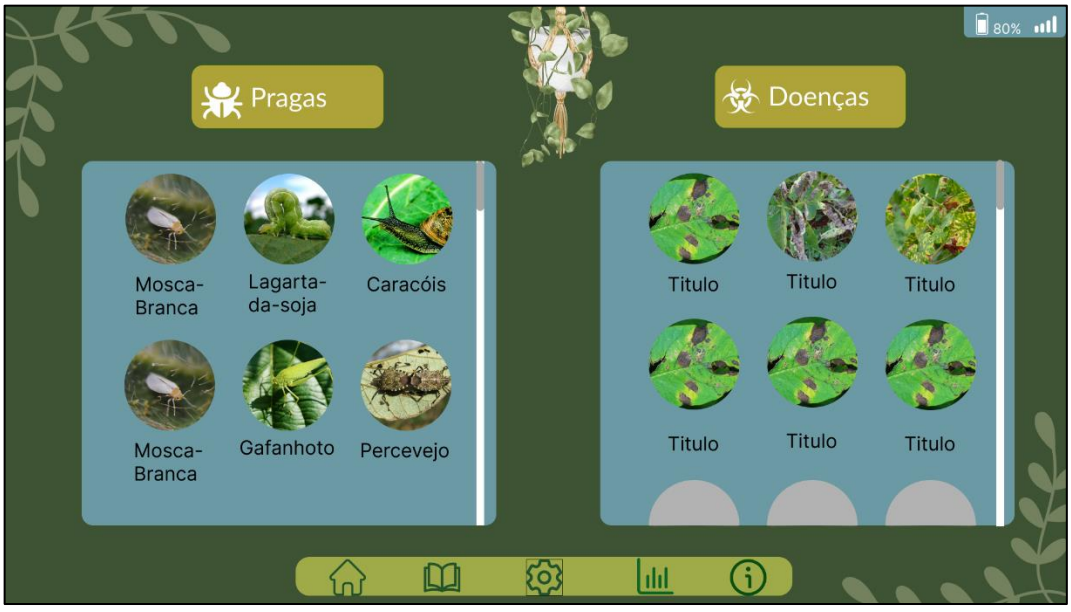
Figura 47 - Wireframe de baixa fidelidade da página de glossário



Fonte: Autoria Própria, 2025

Seguidamente, será apresentado o *wireframe* de alta fidelidade:

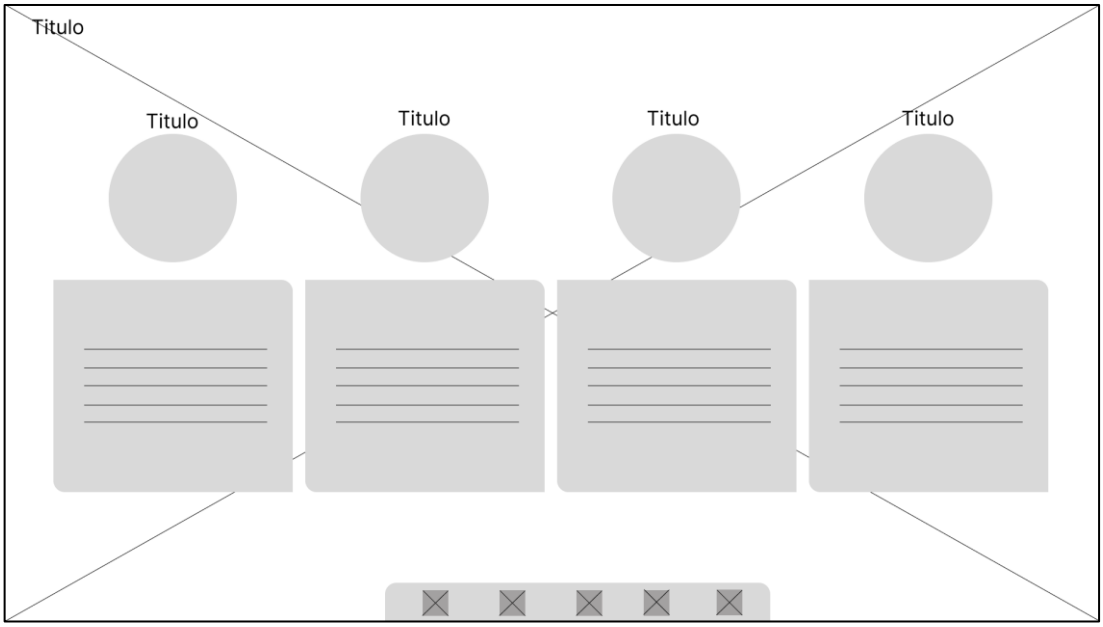
Figura 48 - Wiframe de alta fidelidade da página de glossário



Fonte: Autoria Própria, 2025

E por fim, o último *wireframe* a ser exibido é a tela sobre a equipe do projeto, nesta tela é mostrado a foto dos integrantes que desenvolveram o sistema junto de um texto que explica o que cada membro fez no projeto.

Figura 49 - Wireframe de baixa fidelidade: Tela Sobre A Equipe do Projeto



Fonte: Autoria Própria, 2025

E por fim, será apresentado o *wireframe* de alta fidelidade da página sobre a equipe do projeto:

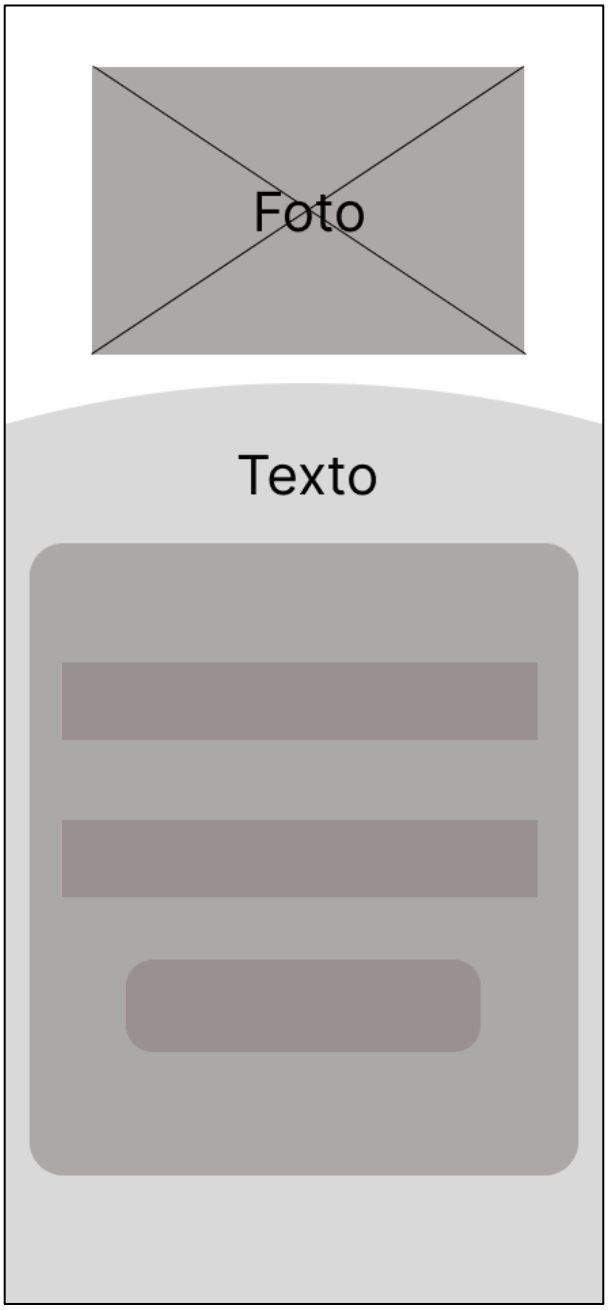
Figura 50 - Wireframe de alta fidelidade: Tela Sobre a Equipe do Projeto

Fonte: Autoria Própria, 2025

Agora que foi finalizado as telas para computador, será apresentado as telas para celulares referentes ao projeto.

A primeira tela que será explicada, sendo essa a tela de Login, onde sua função segue sendo a mesma que a tela de desktop e com os mesmos componentes, tendo uma caixa de texto para informar seu nome e outra para informar sua senha, além de dois botões sendo um para entrar na conta e outro que leva para a tela de cadastro que falaremos a seguir.

Figura 51 - Wireframe de baixa fidelidade: Página de Login para Celular



Fonte: Autoria Própria, 2025

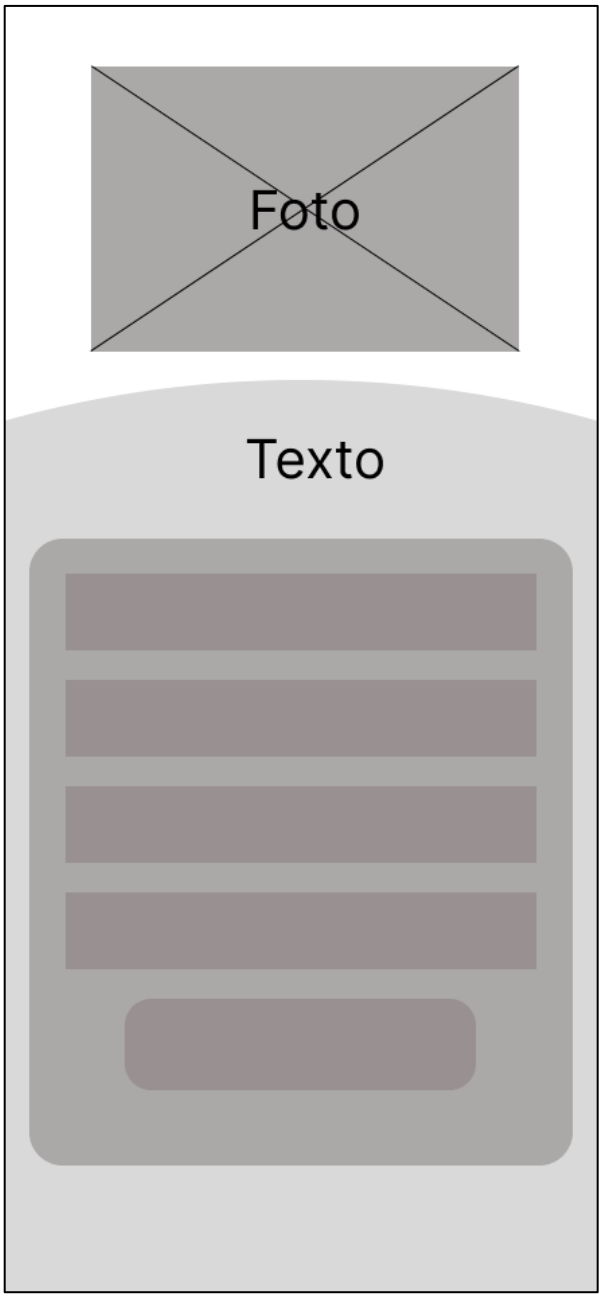
Respectivamente, é exibido o protótipo de alta fidelidade:

Figura 52 - Wireframe de alta fidelidade: Página de Login para Celular

Autoria: Fonte Própria, 2025

Respectivamente, exibindo a próxima página, sendo essa a página de cadastro, onde seu esquema segue o mesmo que a página de login, sendo composto por caixas de textos para definir o seu nome de usuário, um email, uma senha e por motivos de segurança, sendo necessário repetir a senha, evitando erros de digitação e garantindo que o usuário realmente saiba qual a sua senha.

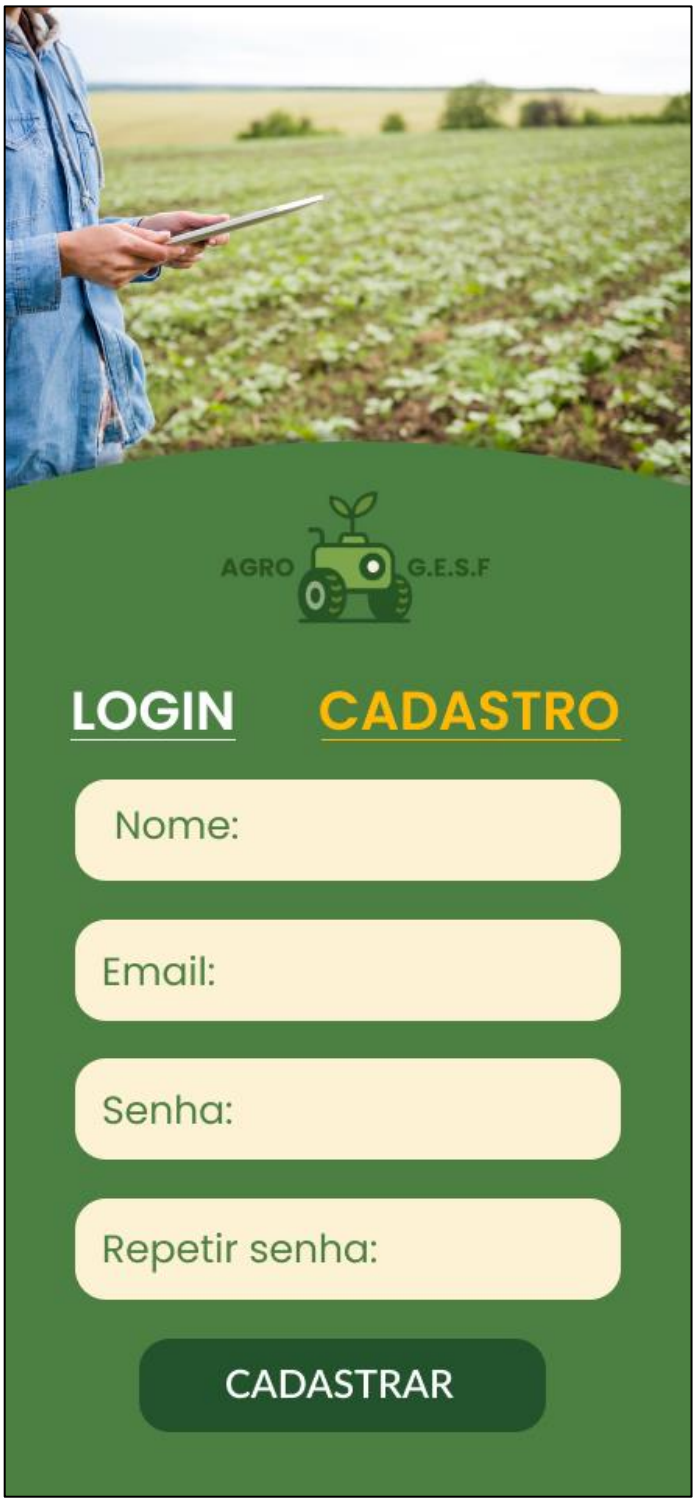
Figura 53 - Wireframe de baixa fidelidade: Página de Cadastro para Celular



Fonte: Autoria Própria, 2025

Seguidamente, é exibida a modelagem de alta fidelidade da página:

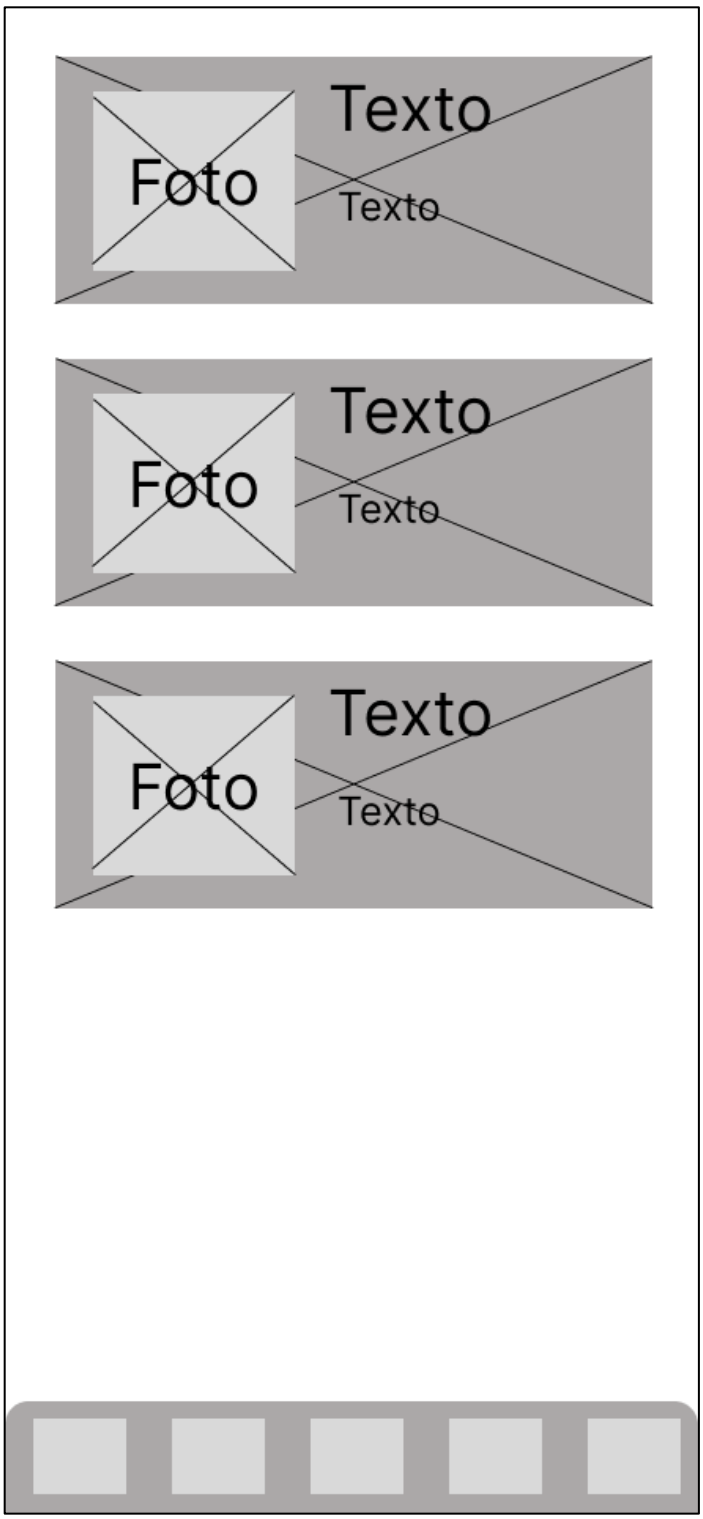
Figura 54 - Wireframe de Alta Fidelidade: Página de Cadastro para Celular



Fonte: Autoria Própria, 2025

Indo para a próxima página, sendo essa a home ou página principal e a primeira que será vista após realizar o Login, onde apareça as detecções que foram encontradas, apresentando nome, foto e a porcentagem de precisão que foi feita.

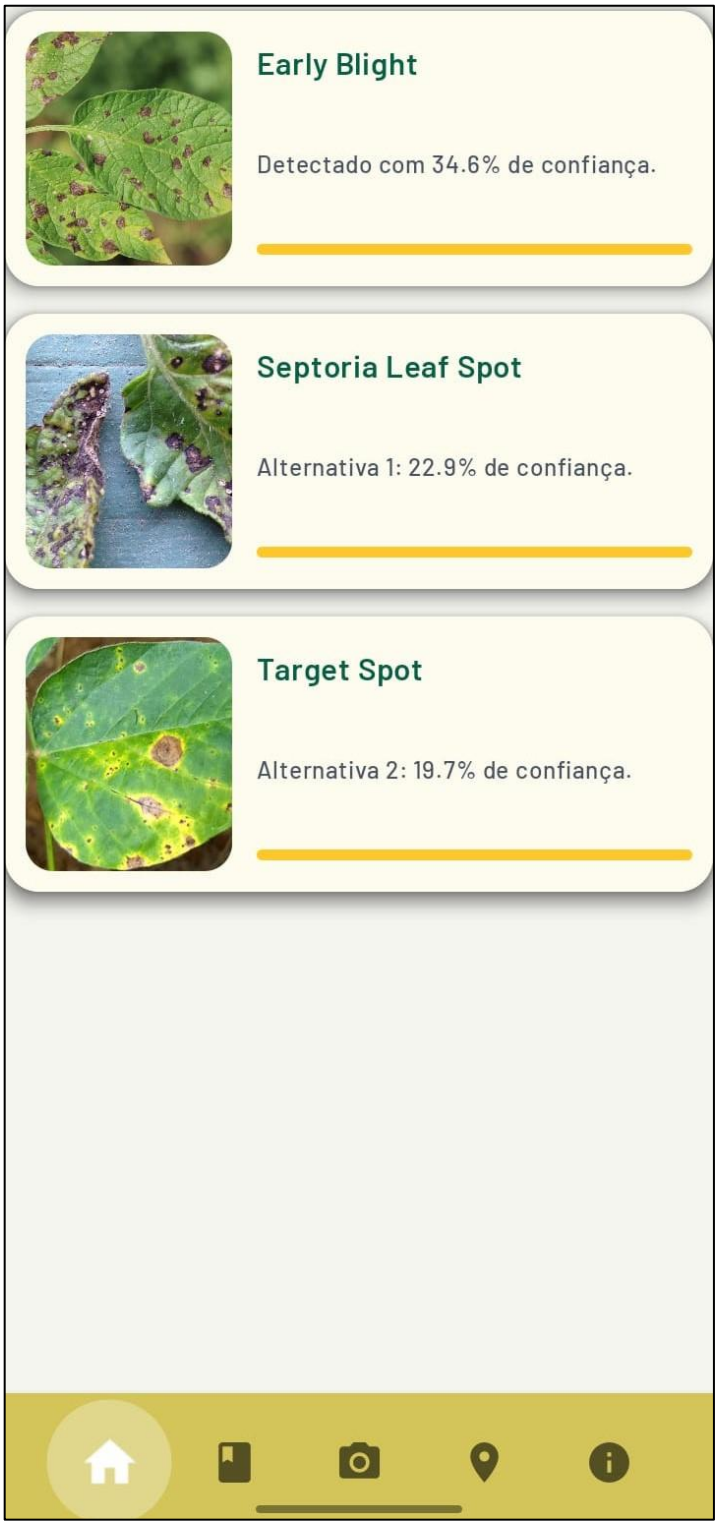
Figura 55 - Wireframe de baixa fidelidade: Página Principal



Fonte: Autoria Própria, 2025

Respectivamente, é apresentado o protótipo de alta fidelidade:

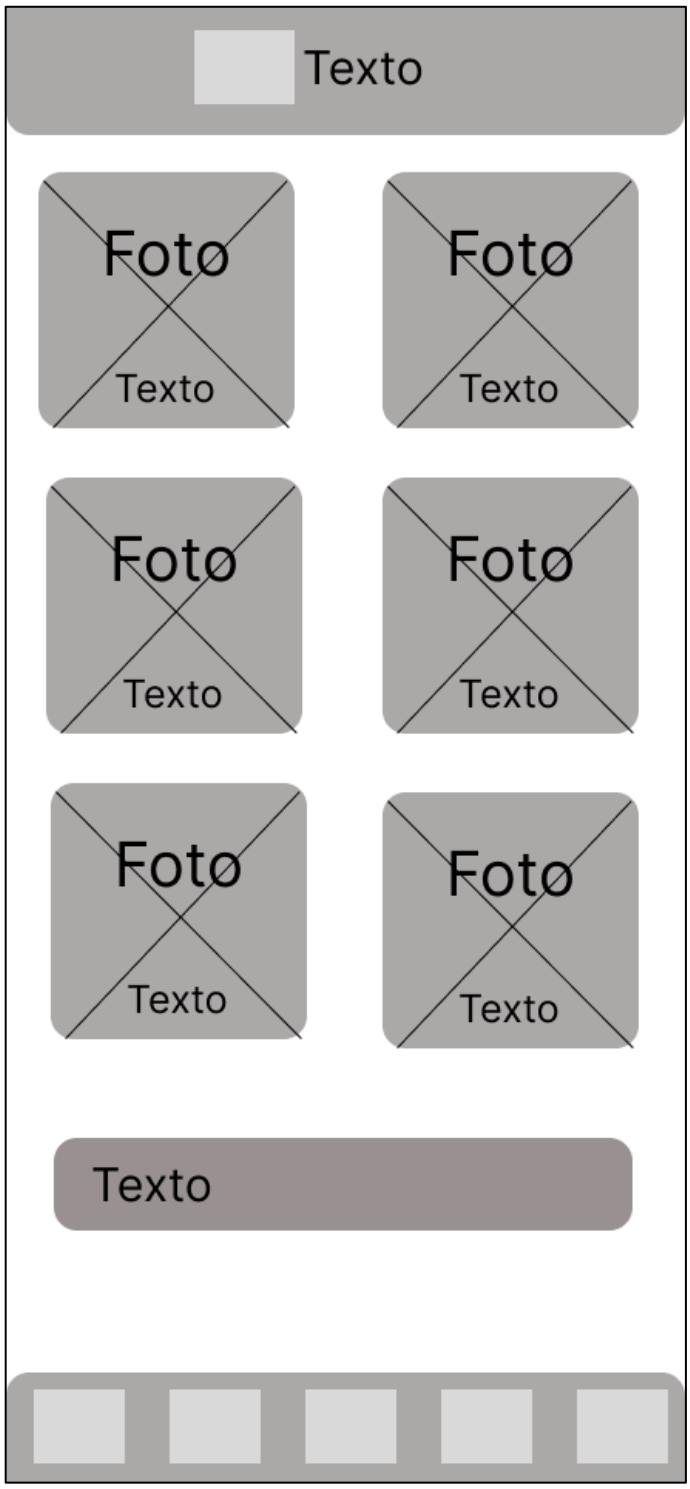
Figura 56 - Wireframe de Alta Fidelidade: Página Principal



Fonte: Autoria Própria, 2025

Explicando a próxima a tela, sendo essa a página de glossário, nesta área é exibido todas as pragas e doenças, sendo dividida em duas páginas praticamente idênticas, onde é exibido o nome, foto e descrição de cada detecção já realizada na plantação.

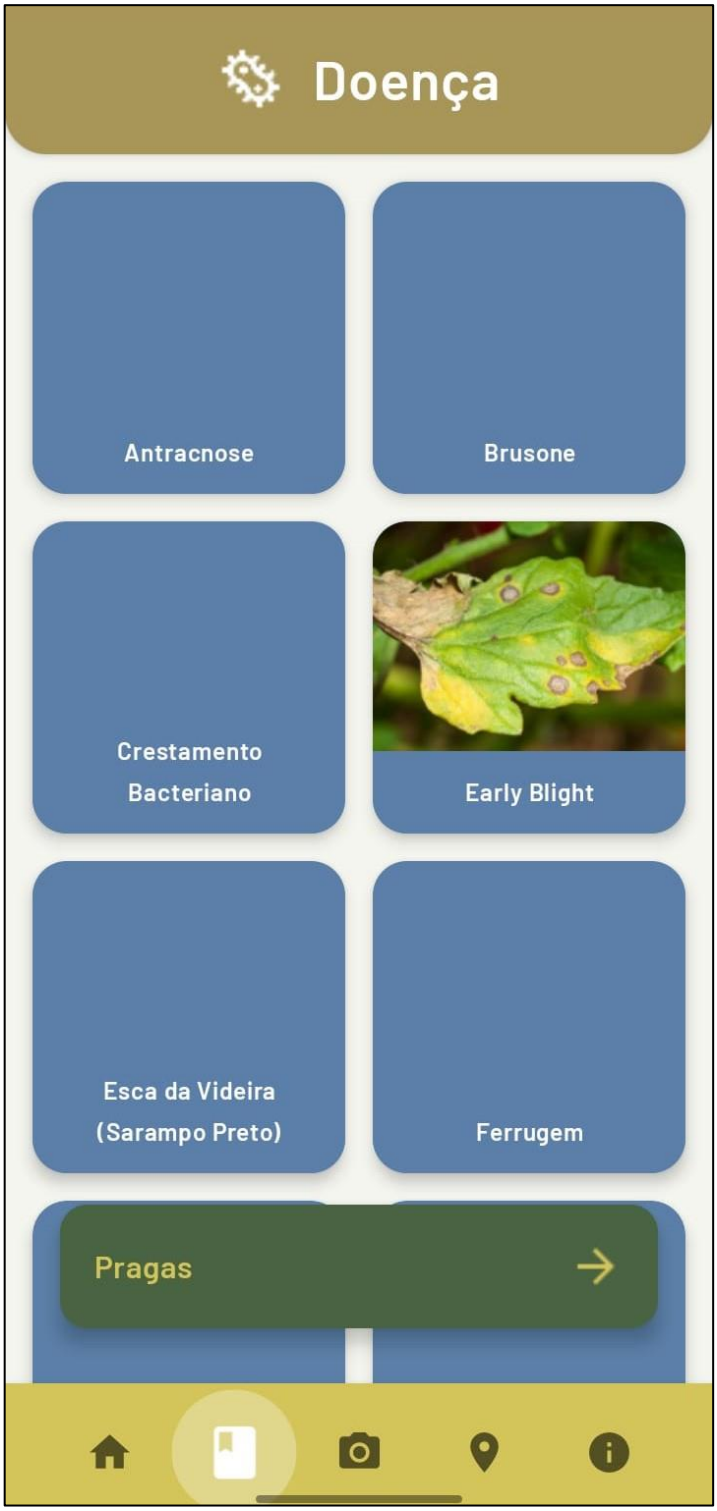
Figura 57 - Wireframe de baixa fidelidade: Glossário de pragas e doenças



Fonte: Autoria Própria, 2025

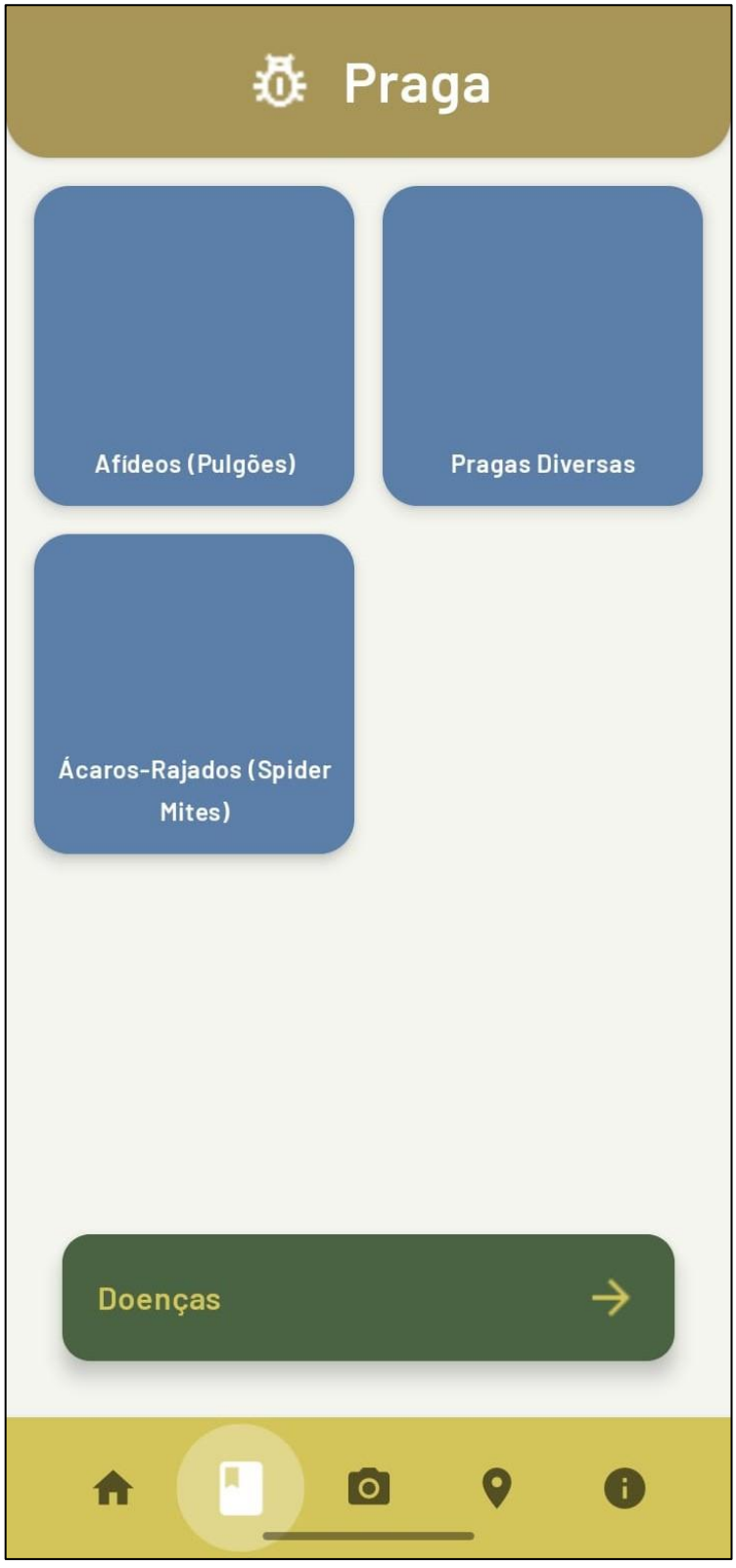
Seguidamente é exibido os wireframes de alta fidelidade tanta da tela de glossário de pragas e doenças:

Figura 58 - Wireframe de alta fidelidade: Glossário de Doença



Fonte: Autoria Própria, 2025

Figura 59 - Wireframe de alta fidelidade: Glossário de Pragas



Fonte: Autoria Própria, 2025

A próxima página é sobre o GPS, nesta tela, é exibido um gráfico de calor, que é criado a partir das informações geográficas de onde o dispositivo encontrou uma detecção, permitindo que o fazendo caminhe até o local preciso de onde foi encontrado a praga ou doença.

Figura 60 - Wireframe de baixa fidelidade: Mapa de Calor

Fonte: Autoria Própria, 2025

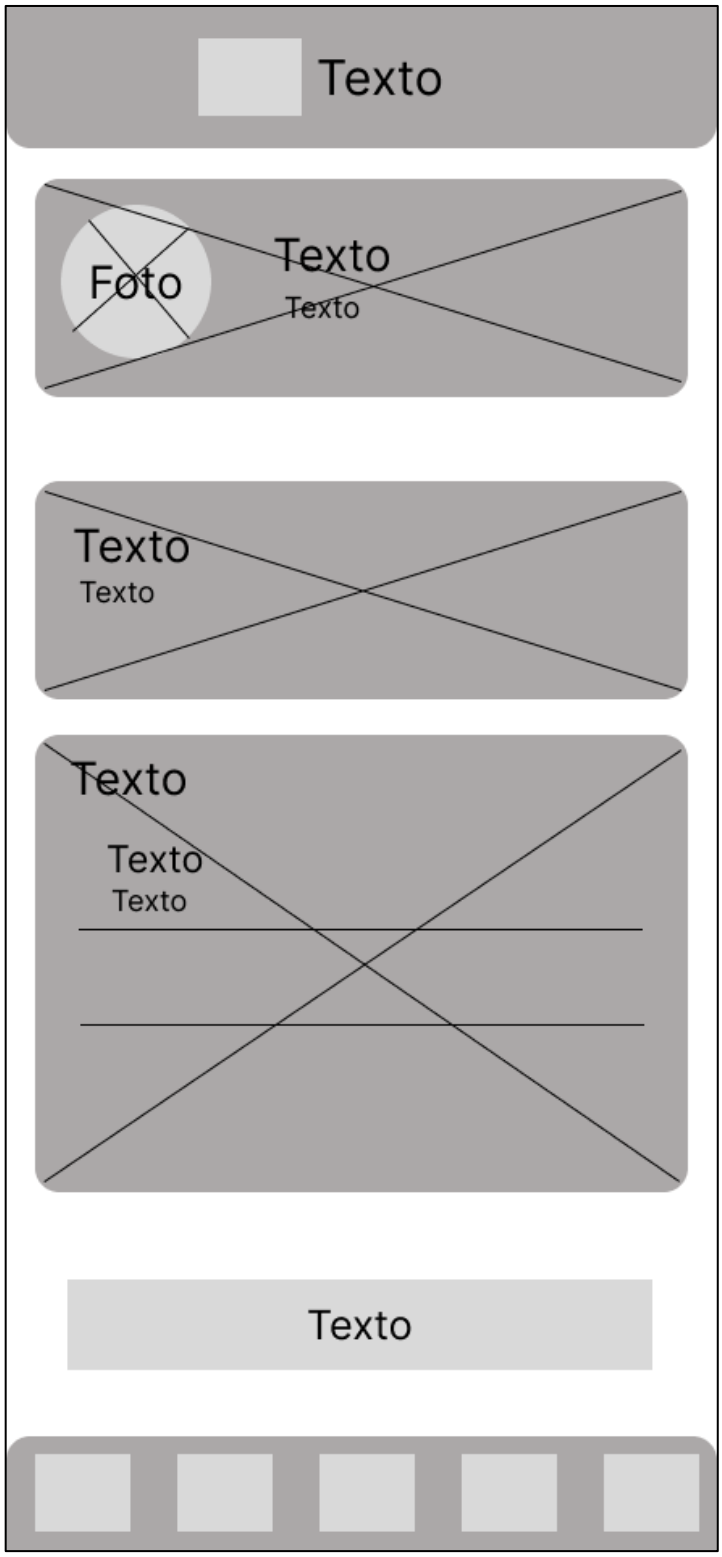
Agora será apresentado a página de alta fidelidade:

Figura 61 - Wiframe de alta fidelidade: Mapa de Calor

Fonte: Autoria Própria, 2025

Indo para a última página, que exibe as informações do usuário cadastrado e certas informações gerais sobre o aplicativo, além de permitir que seja possível sair da conta, levando para a página de login.

Figura 62 - Wireframe de baixa fidelidade: Informações do Usuário



Fonte: Autoria Própria, 2025

E para finalizar, será apresentado o wireframe de alta fidelidade da tela para celular:

Figura 63 - Wireframe de alta fidelidade: Informações do Usuário

Fonte: Autoria Própria, 2025

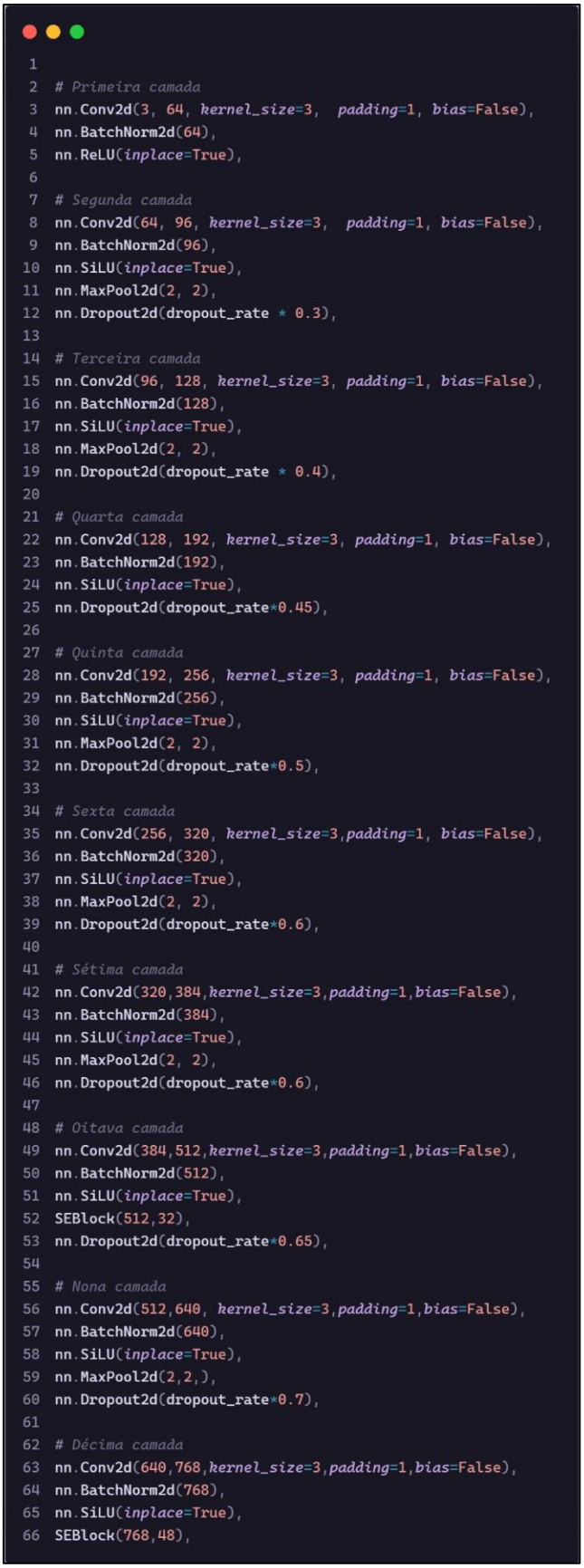
3.7 Rede Neural Convolucional

No presente tópico, será apresentado os conceitos necessários e aplicados para a construção e treinamento da Rede Neural Convolucional (RNC) utilizada neste trabalho. Neste capítulo serão descritos a arquitetura adotada, os motivos das escolhas estruturais e o processo de treinamento e ajuste fino da rede. Em auxílio, serão juntamente apresentados, as técnicas para reduzir overfitting e os procedimentos para validar o desempenho do modelo no problema de detecção de pregas e doenças foliares.

A arquitetura desenvolvida neste trabalho foi projetada especificamente para poder lidar com a grande variedade de classes presentes no dataset de treino, visando manter um bom equilíbrio entre desempenho e custo computacional. A rede é composta por múltiplos blocos convolucionais organizados de forma progressiva, aumentando a profundidade e quantidade de filtros a cada etapa, de maneira a se obter um aprofundamento maior nos detalhes mais específicos de cada classe. Cada bloco é acompanhado de camadas de normalização, funções de ativação SiLU e diferentes níveis de Dropout para reduzir overfitting. Além disso, foi incorporado ao esqueleto da RNC módulos SE (Squeeze-and-Excitation) para permitir que a rede aprenda a dar pesos diferentes para cada canal de feature map, melhorando a capacidade de discriminação da RNC.

Logo abaixo, há o código que representa o esqueleto de tal rede neural, e a logo após, a explicação destrinchando seus principais conceitos.

Figura 64 - Código de estrutura da RNC

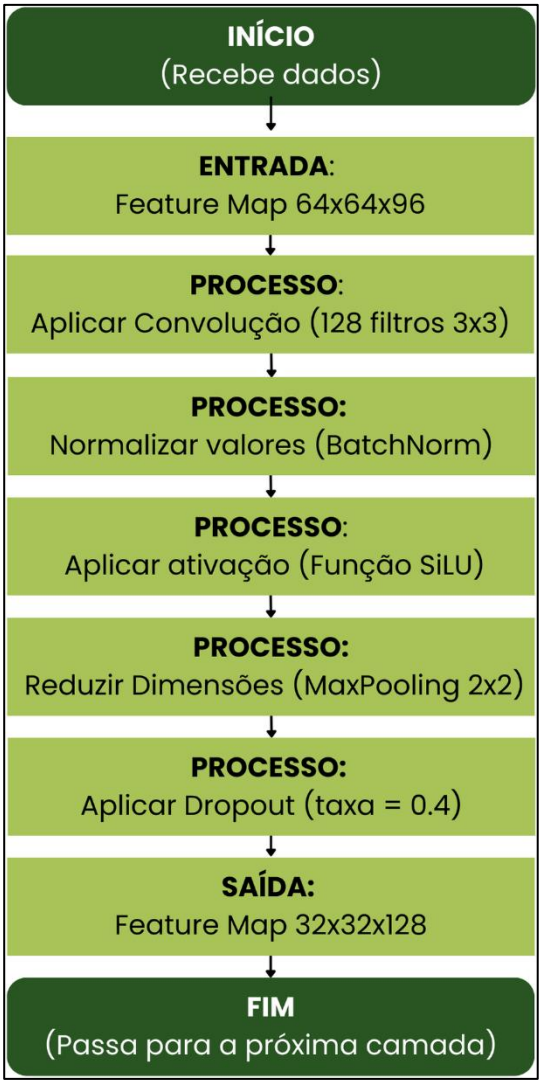


```
1
2  # Primeira camada
3  nn.Conv2d(3, 64, kernel_size=3,  padding=1, bias=False),
4  nn.BatchNorm2d(64),
5  nn.ReLU(inplace=True),
6
7  # Segunda camada
8  nn.Conv2d(64, 96, kernel_size=3,  padding=1, bias=False),
9  nn.BatchNorm2d(96),
10 nn.SiLU(inplace=True),
11 nn.MaxPool2d(2, 2),
12 nn.Dropout2d(dropout_rate * 0.3),
13
14 # Terceira camada
15 nn.Conv2d(96, 128, kernel_size=3, padding=1, bias=False),
16 nn.BatchNorm2d(128),
17 nn.SiLU(inplace=True),
18 nn.MaxPool2d(2, 2),
19 nn.Dropout2d(dropout_rate * 0.4),
20
21 # Quarta camada
22 nn.Conv2d(128, 192, kernel_size=3, padding=1, bias=False),
23 nn.BatchNorm2d(192),
24 nn.SiLU(inplace=True),
25 nn.Dropout2d(dropout_rate*0.45),
26
27 # Quinta camada
28 nn.Conv2d(192, 256, kernel_size=3, padding=1, bias=False),
29 nn.BatchNorm2d(256),
30 nn.SiLU(inplace=True),
31 nn.MaxPool2d(2, 2),
32 nn.Dropout2d(dropout_rate*0.5),
33
34 # Sexta camada
35 nn.Conv2d(256, 320, kernel_size=3,padding=1, bias=False),
36 nn.BatchNorm2d(320),
37 nn.SiLU(inplace=True),
38 nn.MaxPool2d(2, 2),
39 nn.Dropout2d(dropout_rate*0.6),
40
41 # Sétima camada
42 nn.Conv2d(320,384,kernel_size=3,padding=1,bias=False),
43 nn.BatchNorm2d(384),
44 nn.SiLU(inplace=True),
45 nn.MaxPool2d(2, 2),
46 nn.Dropout2d(dropout_rate*0.6),
47
48 # Oitava camada
49 nn.Conv2d(384,512,kernel_size=3,padding=1,bias=False),
50 nn.BatchNorm2d(512),
51 nn.SiLU(inplace=True),
52 SEBlock(512,32),
53 nn.Dropout2d(dropout_rate*0.65),
54
55 # Nona camada
56 nn.Conv2d(512,640, kernel_size=3,padding=1,bias=False),
57 nn.BatchNorm2d(640),
58 nn.SiLU(inplace=True),
59 nn.MaxPool2d(2,2,),
60 nn.Dropout2d(dropout_rate*0.7),
61
62 # Décima camada
63 nn.Conv2d(640,768,kernel_size=3,padding=1,bias=False),
64 nn.BatchNorm2d(768),
65 nn.SiLU(inplace=True),
66 SEBlock(768,48),
```

Fonte: Autoria Própria, 2025

Uma camada de uma rede neural pode ser definida como a peça fundamental de processamento a qual recebe os dados de entrada, se utiliza de transformações matemáticas específicas sobre esses dados e produz uma saída que vai ser utilizada como entrada para a camada seguinte. Em RNC's, cada camada atua como uma extratora de características, sendo responsável por identificar padrões específicos nas imagens processadas. Abaixo há um pequeno fluxo de como seria de forma simplificada, o funcionamento de uma camada, juntamente com o significado de cada termo técnico.

Figura 65 - Fluxo de funcionamento de camada



Fonte: Autoria Própria, 2025

Feature Map: Um feature map (mapa de características) é a representação dos dados em uma determinada etapa da rede neural, nesse caso, os números 64x64x96 indicam as dimensões desse mapa – 64 pixels de largura, 64 pixels de altura e 96 canais de profundidade. Cada canal contém informações sobre diferentes

características que foram detectadas na imagem, como bordas verticais, horizontais, texturas ou gradientes de cor. É como se houvesse 96 imagens sobrepostas de 64x64 pixels, e cada uma destaca um aspecto diferente da imagem original;

Convolução: Consiste em deslizar pequenos filtros sobre toda a imagem de entrada, calculando em cada posição uma multiplicação entre os valores do filtro e os valores da imagem. Cada filtro é treinado para detectar um padrão específico, como uma borda, uma linha, uma textura ou qualquer característica que seja relevante para o treino. Quando o filtro encontra o padrão que procura, ele produz um valor alto naquela região. Aplicando múltiplos filtros de forma simultânea, a rede consegue detectar diversões padrões diferentes ao mesmo tempo;

Normalizar Valores (BatchNorm): A normalização por lote é uma técnica que padroniza os valores que foram gerados pela convolução. Após a convolução, os valores podem ter uma alta variação em escala, onde alguns são extremamente grandes e outros muito pequenos, o que dificulta a rede a aprender. A normalização calcula a média e o desvio padrão de todos os valores no lote atual e ajusta esses valores para que fiquem em uma escala mais uniforme;

SiLU(Sigmoid Linear Unit): A função de ativação SiLU é responsável por introduzir não-linearidade na rede neural. Ela recebe cada valor calculado e o transforma aplicando a fórmula SiLU (x) = x × σ (x). Na prática, isso significa que valores positivos são mantidos ou até ampliados, enquanto valores negativos são suavizados, para se aproximarem do número zero. Essa característica permite que a rede aprenda relações mais complexas e sutis nos dados.

MaxPooling: O Max Pooling é uma operação de redução dimensional que divide o feature map em pequenas regiões e seleciona apenas o valore máximo de cada região. Por exemplo, se um bloco contém os valores 3,7,2 e 5, apena o valor 7 é mantido. Isso reduz pela metade tanto largura quanto à altura do feature map, diminuindo significativamente a quantidade de dados a serem processados na camada seguinte. Além de reduzir o custo computacional, o Max Pooling também ajuda a rede a se tornar mais robusta a pequenas variações na posição dos padrões detectados.

Dropout: O Dropout é uma técnica de regularização que previne o overfitting (quando a rede decora os dados de treino em vez de aprender padrões generalizáveis). Durante o treinamento o Dropout desativa aleatoriamente uma porcentagem de neurônios em cada iteração;

Feature Map 32x32x128: É o resultado após todas as operações da camada. Comprando com a entrada de 64x64x96 podemos observar que as dimensões foram reduzidas pela metade devido ao Max Pooling, enquanto a profundidade aumentou devido a aplicação de mais filtros na convolução. Isso significa que a rede está extraindo mais características diferentes, porém, de uma imagem menor e mais compacta. Assim, cada camada subsequente tende a seguir este padrão, diminuindo

as dimensões espaciais enquanto aumenta a profundidade, concentrando de forma progressiva mais informações semânticas em representações mais abstratas

Após a definição da estrutura da rede, se deu início a etapa de treinamento do modelo. O objetivo era ajustar os pesos internos da rede utilizando o conjunto de imagens disponíveis no dataset, permitindo que o modelo aprendesse a reconhecer padrões mais básicos como texturas, bordas e gradientes de cor, obtendo assim, um backbone sólido. Nesta etapa, a rede foi treinada utilizando a técnica de aprendizagem supervisionada, a qual cada imagem possui um rótulo correspondente à classe correta. Após isso, nos utilizamos deste modelo pré treinado e o qual já possuía um breve conhecimento sobre padrões mais generalistas, para criar um modelo mais refinado a partir deste, contudo, agora utilizando especificamente dados voltados para o cenário agrícola

Mas antecedendo isto, primeiro tivemos que organizar o dataset em uma estrutura 70/20/10, onde, 70% das imagens seriam voltadas para o treino da rede, 20% das imagens seriam utilizadas para a validação do treino do modelo, para poder medir o desempenho do modelo durante o treino e evitar overfitting, e 10% das imagens eram utilizadas somente no momento final para avaliar o real desempenho do modelo.

Por meio desta divisão, nós conseguimos garantir que houvesse uma quantia decente de imagens para o ajuste de pesos da rede, mas também uma quantia suficiente para a validação do modelo e para medição de seu desempenho, e assim iniciamos o treino do modelo utilizando o código abaixo.

Figura 66 - Código de treino da RNC

```python

def train_model(model, dataloaders, dataset_sizes, criterion, optimizer, device, num_epochs):

    model = model.to(device)
    print(device)
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    history = {'train_loss': [], 'val_loss': [], 'train_acc': [], 'val_acc': []}

    for epoch in range(num_epochs):
        print(f'Epoch {epoch+1}/{num_epochs}')
        print('-' * 20)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
                    acc = calculate_accuracy(outputs, labels)

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += (acc * inputs.size(0))

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects / dataset_sizes[phase]

            history[f'{phase}_loss'].append(epoch_loss)
            history[f'{phase}_acc'].append(epoch_acc)

            print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

            if phase == 'val' and epoch > 0:
                if (history['val_loss'][-1] > history['val_loss'][-2] and
                    history['train_loss'][-1] < history['train_loss'][-2]):
                    print('Aviso: Possível overfitting detectado')

            # Salvar o melhor modelo
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
                save_checkpoint(model, optimizer, epoch)

        print(f'Melhor val Acc: {best_acc:.4f}')
        print()

    model.load_state_dict(best_model_wts)
    return model, history

```

Fonte: Autoria Própria, 2025

O código prévio demonstra a função fundamental para a criação da rede, o código de treino. A função se apresenta como sendo curta, mas densamente ideal para a criação de uma rede bem estruturada. Abaixo segue uma breve explicação sobre os blocos do código:

Linha 2: Define a função e os parâmetros que deve receber;

Linha 4 -9: São definidas as principais variáveis para o funcionamento do código, onde a variável “model” é inserida dentro do dispositivo de processamento e a variável best_acc fica responsável por armazenar aépoca do treino que teve uma melhor acurácia e a variável “history” salva asinformações fundamentais para o acompanhamento do treino.;

Linha 11: Se inicia o loop principal, o qual irá repetir o mesmo processo de aprendizado, a partir do valor associado a variável “num_epochs”;

Linha 15-19: Novamente, outro loop, contudo este é responsável por informar ao código se o modelo deve ser treinado naquele determinado momento ou ocorrer uma validação;

Linha 21-22: São definidas as variáveis que vão armazenar os acertos e os erros do modelo;

Linha 24: É aqui onde se inicia o loop que irá percorrer todos os batches do conjunto atual, seja ele de treino ou validação. Nesta linha, ele extrai a entrada (imagens) e os rótulos das classes de cada batch vindo do dataloader;

Linha 25-26: Nessas linhas, tanto as imagens quanto seus respectivos rótulos são enviados para o processamento. Isso garante que o modelo e os dados estejam no mesmo lugar, permitindo que o processamento ocorra de forma correta;

Linha 27: O otimizador tem seus gradientes zerados, e isto é importante pois, por padrão, o PyTorch acumula os gradientes a cada batch. Por isto, essa linha impede que os gradientes antigos interfiram no aprendizado atual;

Linha 29: É definido se o cálculo de gradientes deve ser ativado ou não. Quando o código está na fase de treino, os gradientes se fazem em necessidade, contudo, na fase de validação do modelo, eles são desabilitados, o que permite uma otimização de memória e desempenho;

Linha 30: O batch de imagens é passado pelo modelo, o qual produz uma predição para ela. É neste momento em que a rede tenta “adivinhar” a classe da imagem;

Linha 31: O erro ou perda é calculado comparando as predições do modelo com os rótulos verdadeiros. Está métrica permite guiar o aprendizado do modelo;

Linha 32: A acurácia daquele batch é calculada e isso fornece uma medida imediata de quantas imagens o modelo classificou corretamente naquele grupo específico;

Linha 34-37: Esse bloco é executado somente na fase de treino. Ele é responsável por realizar o processo de aprendizado, onde, primeiro calcula-se o gradiente com base na perda, e depois o otimizador atualiza os pesos do modelo. É aqui que o modelo efetivamente “aprende”;

Linha 39: O valor de perda de batch é acumulado para que mais tarde, seja calculada a perda média por época inteira;

Linha 40: A quantidade de acertos daquele batch é adicionada ao total acumulado, permitindo assim fazer o cálculo da acurácia média do “epoch”;

Linha 41-42: Ao final de todos os batches da fase, é calculada a média de perda e da acurácia, onde se divide os valores acumulados pelo tamanho total do conjunto;

Linha 43-44: As métricas calculadas são armazenadas na variável definida anteriormente como “history”, que armazena o histórico completo do treino;

Linha 48: Os resultados da época são exibidos no terminal, o que facilita o acompanhamento do desempenho do modelo durante o treinamento;

Linha 49-53: Este bloco é responsável por detectar sinais de overfitting, verificando se a perda de validação aumentou enquanto a perde de treino diminuiu, e caso isso ocorra, é emitido um aviso indicando tal problema;

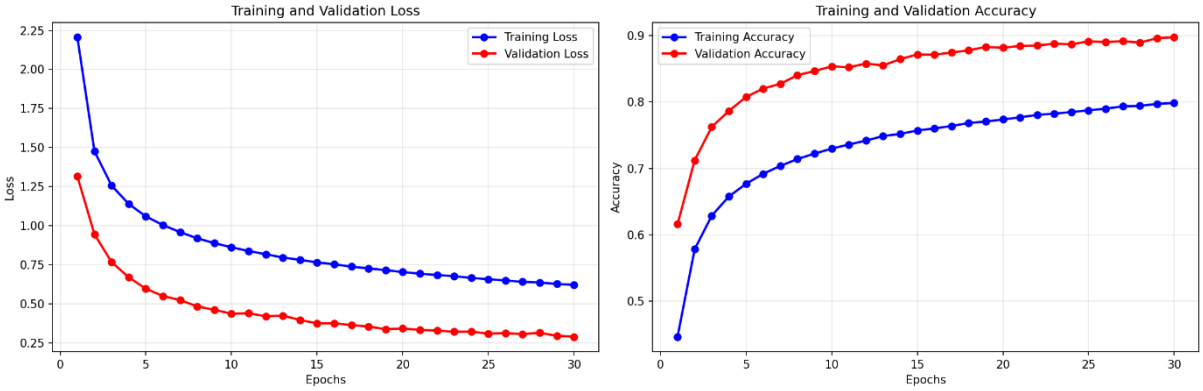
Linha 55-60: Aqui o código verifica se a atual fase é de validação e se a acurácia obtida é a melhor já registrada até então, caso seja, o modelo é salvo como o melhor encontrado. A variável “best_model_wts” armazena exatamente os pesos dessa melhor versão enquanto o “save_checkpoint” registra tudo em um arquivo e é emitido uma mensagem avisando que foi encontrado a melhor acurácia até então;

Linha 63: No final do processo de treino, o modelo é restaurado para o estado em que obteve seu melhor desempenho na validação, o que garante que o modelo usado seja o mais eficiente;

Linha 64-65: Por fim, a função retorna o modelo otimizado e o histórico completo das métricas coletadas ao longo das épocas.

Logo após o fim do treino, é gerado um arquivo no formato. pth (extensão própria para arquivos criados se utilizando do PyTorch) com todos os pesos e arquitetura do modelo, e, os gráficos correspondentes ao treino e como ocorreu o crescimento da acurácia da validação e treino, abaixo segue um dos gráficos da época 30.

Figura 67 - Gráfico de aprendizado



Fonte: Autoria Própria, 2025

4 CONSIDERAÇÕES FINAIS

O agronegócio é um forte pilar da economia do Brasil e o país e nos últimos anos tem começado a implementar meios tecnológicos para aperfeiçoar essa área, investindo no modo de agricultura de precisão para o futuro, e é de extrema importância que seja feita uma inclusão dessas tecnologias para que não só apenas produtores de larga escala, mas que também produtores de menor tamanho tenham acesso a esses avanços, pois hoje em dia, o pequeno agricultor ainda é uma peça fundamental da agricultura no Brasil, portanto, estamos muito felizes em contribuir para atender a esses pequenos agricultores, utilizando da tecnologia e da criatividade de inovar para poder permitir que os produtores de pequeno porte possam ter uma saúde melhor ,terem uma queda considerável na perda de suas produções e o mais essencial, estejam começando a implementar a agricultura de precisão em suas lavouras , é essencial lembrar que a tecnologia está avançando de forma extremamente rápida e caso seja definido quais são suas funcionalidade no futuro para a sociedade, é de extrema importância que esse desenvolvimento seja para permitir que ideias revolucionárias e beneficiarias para o mundo saiam do campo teórico apenas e se tornem ideias que sejam aplicadas no campo físico, fazendo com que a imagem de melhorar o mundo deixe de ser apenas um sonho e se torne uma realidade.

Vale salientar, que o projeto ainda pretende evoluir e ainda existem muitos obstáculos a serem superados, mas com determinação, cooperação e harmonia, esses problemas podem ser resolvidos com o decorrer do desenvolvimento, o projeto pode e irá implementar novas ideias para tornar o dispositivo mais autêntico, benéfico e eficiente. Queremos fazer com que o dispositivo IoT seja totalmente autônomo, não sendo necessário que o pequeno agricultor esteja constantemente direcionando o automóvel, pretendemos também construir nosso próprio protótipo de automovível, comprando e utilizando todas as peças necessárias para a sua criação e garantindo que ele ainda possa lidar com todos os cenários que acontecem diariamente com o pequeno produtor e por fim, planejamos melhorar o *deep learning* do nosso dispositivo, especialmente sobre a confiança no momento da predição da praga ou doença.

Uma melhor autonomia irá fazer com que uma maior quantidade de pequenos agricultores esteja sendo introduzidos ao novo modelo de agricultura, impactando diretamente de forma positiva em sua saúde, suas mercadorias e no bem-estar do consumidor final, sendo assim, o projeto está satisfeito com o resultado apresentado para este ano e acreditamos que implementamos todos os nossos conhecimentos e esforços necessários, mas ainda pretendemos fazer as melhorias que foram citadas neste durante o decorrer deste capitulo.

REFERÊNCIAS

EMBRAPA. **Pragas quarentenárias: perguntas e respostas**. 2024. Disponível em: https://www.embrapa.br/tema-pragas-quarentenarias/perguntas-e-respostas. Acesso em: 26 maio 2025.

MARCONI,Marina;LAKATOS,Eva. **Fundamentos de metodologia científica.** 8. ed. São Paulo: Atlas, 2017.

SENAR – SERVIÇO NACIONAL DE APRENDIZAGEM RURAL. **Café: controle de pragas, doenças e plantas daninhas**. Brasília: SENAR, 2017. 71 p. (Coleção SENAR, 190). ISBN 978-85-7664-151-3.

NOGUEIRA, Fernanda, SZWARCWALD, DAMACENA, Gisele. **Exposição a agrotóxicos e agravos à saúde em trabalhadores agrícolas: o que revela a literatura?**: Revista Brasileira de Saúde Ocupacional, 2020. Disponível em: https://ninho.inca.gov.br/jspui/handle/123456789/9016. Acesso em 15 nov 2025.

FERRÃO, Romário Gava; FONSECA, Aymbiré Francisco Almeida da; FERRÃO, Maria Amélia Gava; MUNER, Lúcio Herzog De (org). **Café Conilon**. 2. ed. atual. e ampl., 2. reimp. Vitória, ES: Incaper, 2017.

CENTRO DE ESTUDOS AVANÇADOS EM ECONOMIA APLICADA (CEPEA). **Efeito do não tratamento de pragas e doenças sobre preços ao consumidor de produtos da cadeia produtiva de soja**. Piracicaba: CEPEA, jul. 2019. Disponível em: www.cepea.esalq.usp.br. Acesso em: 18 set. 2025.

CARNEIRO, Fernando Ferreira; AUGUSTO, Lia Giraldo da Silva; RIGOTTO, Raquel Maria; FRIEDRICH, Karen; BÚRIGO, André Campos. Dossiê ABRASCO**: Um alerta sobre os impactos dos agrotóxicos na saúde**. Escola Politécnica de Saúde Joaquim Venâncio: Expressão Popular, 2015. Disponível em: https://abrasco.org.br/download/dossie-abrasco-um-alerta-sobre-os-impactos-dos-agrotoxicos-na saude/?wpdmdl=79148&refresh=690d0994e28cb1762462100. Acesso em: 06 novembro 2025.

BRITO, Paula Fernandes de; GOMIDE, Márcia; CÂMARA, Volney de Magalhães. **Agrotóxicos e saúde: realidade e desafios para mudança de práticas na agricultura.** Physis: Revista de Saúde Coletiva, Rio de Janeiro, v. 19, n. 1, p. 207-225, 2009.

RIGOTTO, Raquel Maria; VASCONCELOS, Dayse Paixão e; ROCHA, Mayara Melo. **Uso de agrotóxicos no Brasil e problemas para a saúde pública**. Ceará, Fortaleza, 2014. Disponível em: https://www.scielosp.org/article/csp/2014.v30n7/1360-1362/pt/#. Acesso em: 05 novembro 2025.

LOPES, Carla Vanessa A.; ALBUQUERQUE, Guilherme Souza Cavalcanti de. **Agrotóxicos e seus impactos na saúde humana e ambiental: uma revisão sistemática.** Curitiba, Paraná, 2018. Disponível em: https://www.scielo.br/j/sdeb/a/bGBYRZvVVKMrV4yzqfwwKtP/?lang=pt. Acesso em: 05 novembro 2025.

OLIVEIRA, Sérgio. **Internet das Coisas com ESP8266, Arduino e Raspberry Pi**. São Paulo: Novatec, 2017.

MAGRANI, Eduardo. **A Internet das Coisas**. Rio de Janeiro: FGV Editora, 2018.

DOBBIN, Ivan Diniz. **Desenvolvimento de software com interface gráfica em Raspberry PI 4 para controle de uma máquina de cisalhamento**. 2022. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Software) – Universidade de Brasília, Faculdade UnB Gama, Brasília, 2022.

SANTOS, P. R. S.; LOPES, A. F. G.; DIAS, W. R. A. Usando RaspBerry Pi para redução energética no IFRO. **Revista de Gestão e Secretariado**, *[S. l.]*, v. 15, n. 4, p. e3596, 2024. DOI: 10.7769/gesec.v15i4.3596. Disponível em: https://ojs.revistagesec.org.br/secretariado/article/view/3596. Acesso em: 12 ago. 2025.

LAUXEN, Rafael; LOVATTO, Andressa; SERPA DA ROSA, Ronaldo**. MicroscopePi: o desenvolvimento de um microscópio digital com Raspberry Pi 4**. In: SALÃO DE PESQUISA, EXTENSÃO E ENSINO DO IFRS, 8., 2023, Bento Gonçalves. Anais [...]. Bento Gonçalves: Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul, 2023. p. [página inicial-final]. Disponível em: https://eventos.ifrs.edu.br/index.php/salao/8salao/paper/view/2456. Acesso em: 12 ago. 2025.

LOGITECH INTERNATIONAL S.A. **BRIO 100 Webcam**. Logitech Brasil. Disponível em: https://www.logitech.com/pt-br/shop/p/brio-100-webcam.960-001586 .Acesso em: 15 nov 2025.

BORGES, Luiz. *Python para Desenvolvedores*. 3. ed. Rio de Janeiro: Novatec, 2014.

PAIVA, Fábio *et al*. **Introdução a Python com Aplicações de Sistemas Operacionais**. Natal: IFRN, 2020.

MENEZES, Nilo. **Introdução à Programação com Python**. 2. ed. São Paulo: Novatec, 2014.

PYTHON SOFTWARE FOUNDATION. **Documentação do Python***. 3. ed.* Disponível em: https://docs.python.org/pt-br/3. Acesso em: 15 set. 2025.

LABAKI, Josué. **Introdução a Python – Módulo C**. Ilha Solteira: UNESP, 2004.

MENEZES, Nilo. **Introdução à Programação com Python***. 4.* ed. São Paulo: Novatec, 2024.

LINS, Andréa e SOUZA, Lins. **Python para matemáticos**. Rio de Janeiro: Sociedade Brasileira de Matemática, 2023. eBook. ISBN 978-85-8337-216-5.

MOREIRA, João, MARMONTEL, Silvia, CHAMORRO, Luis. **Software Desktop para Conversão de Arquivos: Aplicação do Python e Uso da Inteligência Artificial para Aprendizado**. Revista Acadêmica da Faculdade Professor Sérgio Silva. Porto Alegre, v. 3, n. 1, p 1-27, fev/2025. Disponível em: https://rfaps.com/index.php/rfaps/article/view/11. Acesso em 16 nov 2025.

ESCOVEDO, Tatiana, KOSHIYAMA, Adriano. **Introdução a Data Science Algoritimos de Machine Learning e Métodos de Análise**. São Paulo: Casa do Código,2020.

LUDERMIR, Teresa. **Inteligência Artificial e Aprendizado de Máquina: Estudo Atual e Tendências**. Estudos Avançados, São Paulo: Universidade de São Paulo, v. 35, n. 101, p. 85-94 2021. Disponível em: https://revistas.usp.br/eav/article/view/185035. Acesso em 13 ago 2025.

PEINADO, Hugo, OLIVEIRA, Carolina, OTTONI, André, MELO, Roseneia, COSTA, Dayana, NOVO, Marcela. **Análise de Técnicas de *Data Augmentation* para Aperfeiçoamento da Detecção de Sistemas de Guarda-Corpo e Rodapés em Canteiros de Obras com Inteligência Artificial**. Salvador: Universidade Federal da Bahia, v.13., 2023. p. 1-11. Disponível em: https://eventos.antac.org.br/index.php/sibragec/article/view/2637. Aceso em 18 nov 2025.

GRUS, Joel. ***Data Science do Zero: Noções Fundamentais com Python*. 2. ed**. Rio de Janeiro: Alta Books, 2021. 416 p. ISBN 978-85-5081-176-5.

DE PAULA, Leonardo Scarmato Jorge. **Mineração de repositórios para avaliar a influência das mudanças de código ao longo do tempo.** 2024. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru, 2024.

PAIXÃO, Gabriela et al. **Machine learning na medicina: revisão e aplicabilidade.** *Arquivos Brasileiros de Cardiologia*, [S. l.], [s. n.], [s. v.], [s. n.], [20--]. Disponível em: https://www.scielo.br/j/abc/a/WMgVngCLbYfJrkmC65VFCkp/?format=pdf&lang=pt. Acesso em: 11 ago. 2025.

TSUNODA, D.F.; MOREIRA, P.S.C.; GUIMARÃES, A.J.R. **Machine learning e revisão sistemática de literatura automatizada: uma revisão sistemática.** Rev. Tecnol. Soc., Curitiba, v. 16, n. 45, p. 337-354, out./dez., 2020. Disponível em: https://periodicos.utfpr.edu.br/rts/article/view/12119. Acesso em: 16 nov 2025.

SANTOS, Rogério P. dos; BEKO, Marko; LEITHARDT, Valderi R. Q. **Modelo de Machine Learning em Tempo Real para Agricultura de Precisão**. *In*: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD-RS), 22 , 2022, Curitiba. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2022. p. 69-70. ISSN 2595-4164. DOI:

HOSAKI, Gabriel Yuri; RIBEIRO, Douglas Francisco. **Deep learning: ensinando a aprender**. Revista Científica da FATEC de Assis, Assis, v. 3, n. 1, 2021. ISSN 2674-6743.

VISÃO COMPUTACIONAL E RACISMO ALGORÍTMICO: BRANQUITUDE E OPACIDADE NO APRENDIZADO DE MÁQUINA. Revista da Associação Brasileira de Pesquisadores/as Negros/as (ABPN), *[S. l.]*, v. 12, n. 31, 2020. Disponível em: https://abpnrevista.org.br/site/article/view/744. Acesso em: 27 maio. 2025.

MARENGONI, M.; STRINGHINI, S. **Tutorial: introdução à visão computacional usando OpenCV**. *Revista de Informática Teórica e Aplicada*, Porto Alegre, v. 16, n. 1, p. 125–160, 2010. DOI: 10.22456/2175-2745.11477.

BARELLI, Felipe. ***Introdução à Visão Computacional: uma Abordagem prática com Python e OpenCV***. São Paulo: Casa do Código, 2018.

FERNANDES NETO, Euripedes Purcinio**. Visão computacional para identificação de cores em tempo real com OpenCV e Python**. 2020. Monografia (Graduação em Engenharia da Computação) – Faculdade de Tecnologia e Ciências Sociais Aplicadas, Centro Universitário de Brasília, Brasília, 2020.

ANTONELLO, Ricardo. **Introdução à Visão Computacional com Python e OpenCV**. Luzerna: Instituto Federal Catarinense, 2018.

JERONYMO, Victor. **Implementando Métodos de otimização para treinamento de Redes Neurais com Pytorch.** Universidade de Campinas, 2019.

SAAVEDRA, Marcos Rodrigo Mendes; SOUZA, Flávio Ramon Almeida de; ARAÚJO, Josivaldo de Souza. **Avaliação do uso do TensorFlow e do PyTorch em uma rede neural artificial utilizada para o reconhecimento facial**. In: CONTECSI – INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGY MANAGEMENT, 19., 2022, São Paulo. *19th CONTECSI 2022 - Proceedings and Abstracts*. São Paulo: TECSI – FEA USP, 2022. p. 01-11.

CRESTANI, Angelo Nery Vieira; SOUZA, Paulo Silas Severo de; MARQUES, Wagner dos Santos; KONZEN, Marcos Paulo; ROSSI, Fábio Diniz**. Avaliando o desempenho do PyTorch sobre GPUs embarcadas**. *In*: ESCOLA REGIONAL DE ALTO DESEMPENHO DA REGIÃO SUL (ERAD-RS), 2018, Porto Alegre. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2018. ISSN 2595-4164.

ZUANAZZI, T. P**. Aprendizado profundo para a segmentação de regiões teciduais em cirurgias minimamente invasivas**. 2024. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica e de Telecomunicações) – Faculdade de Engenharia, Universidade Estadual Paulista “Júlio de Mesquita Filho”, São João da Boa Vista, 2024.

JEMEROV, Dmitry; ISAKOVA, Svetlana. **Kotlin em Ação**. São Paulo: Novatec, 2017.

RESENDE, Kassiano. **Kotlin com Android Crie aplicativos de maneira fácil e divertida**. São Paulo: Casa do Código, 2018.

SILVA, Victor de Farias; ZUCHI, Jederson Donizete. **Análise dos Fundamentos do Paradigma Orientado a Objetos na Linguagem Kotlin**. *Interface Tecnológica*, Taquaritinga: FATEC Taquaritinga, v. 20, n. 2, p. 1-12, dez, 2023. DOI: 10.31510/infa.v20i2.1796. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/pt_BR/article/view/1796. Acesso em: 12 nov. 2025.

VITALINO, Jeferson, CASTRO, Marcus. **Descomplicando o Docker**. 2. Ed. São Paulo: Brasport, 2018.

PEINADO, Hugo Sefrian; OLIVEIRA, Carolina Andrade de; OTTONI, André Luiz Carvalho; MELO, Roseneia Rodrigues Santos de; COSTA, Dayana Bastos; NOVO, Marcela Silva. **Análise de técnicas de data augmentation para aperfeiçoamento da detecção de sistemas de guarda-corpo e rodapés em canteiros de obras com inteligência artificial**. In: SIMPÓSIO BRASILEIRO DE GESTÃO E ECONOMIA DA CONSTRUÇÃO, v. 13., 2023. Anais [...]. Porto Alegre: ANTAC, 2023. p. 1–10. DOI: 10.46421/sibragec.v13i00.2585. Disponível em: https://eventos.antac.org.br/index.php/sibragec/article/view/2585. Acesso em: 11 ago. 2025.

OLIVEIRA, Walisson Santos; SILVA, Alisson Souza; MELO, Roseneia Rodrigues Santos de; BRAGA, Pedro Afonso Vieira Fernandes; COSTA, Dayana Bastos. **Uso de Data Augmentation para reconhecimento automatizado de anomalias em fachada**. In: ENCONTRO NACIONAL DE TECNOLOGIA DO AMBIENTE CONSTRUÍDO, v. 20., 2024. Anais [...]. Porto Alegre: ANTAC, 2024. p. 1–13. DOI: 10.46421/entac.v20i1.5843. Disponível em: https://eventos.antac.org.br/index.php/entac/article/view/5843. Acesso em: 11 ago. 2025.

CARNEIRO, Maria Sheila; GATTO, Dacyr Dante de Oliveira; SASSI, Renato José; GASPAR, Marcos Antonio. **Rede neural convolucional aplicada na análise de sentimentos em comentários de clientes de empresa do ramo varejista**. *Navus – Revista de Gestão e Tecnologia*, v. 16, e2028, 2025. DOI: https://doi.org/10.22279/navus.v16.2028. Disponível em: https://navus.sc.senac.br/index.php/navus/article/view/2028. Acesso em: 12 ago. 2025.

VARGAS, Ana Caroline Gomes; PAES, Aline; VASCONCELOS, Cristina Nader. **Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres**. In: CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES (SIBGRAPI), 29., 2016, São José dos Campos, SP, Brazil. *Proceedings*. Porto Alegre: Sociedade Brasileira de Computação, 2016. Disponível em: http://urlib.net/ibi/8JMKD3MGPAW/3ME3L2P. Acesso em: 12 ago. 2025.

ELMASRI, Ramez; NAVATHE, Shamkant. **Sistema de Banco de Dados**. 6.ed. São Paulo: Pearson Education, 2011.

DATE, C.J. **Introdução a Sistema de Banco de Dados**. 8. Ed. Rio de Janeiro: Elsevier, 2004.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de Gerenciamento de Banco de Dados**. 3.ed. Porto Alegre: AMGH, 2011.

COMACHIO, Vanderson. **Funcionamento de banco de dados de em Android: um estudo experimental utilizando SQLite.** 2011. 68 f. Trabalho de Conclusão de Curso(graduação). Universidade Federal do Paraná, Medianeira, 2011. Disponível em: http://repositorio.utfpr.edu.br/jspui/handle/1/13401. Acesso em: 3 nov. 2025.

SILVA, Paulo; SCHANTZ, Douglas; ANTUNES, Rodrigo; SCHUCH, Regis Rodolfo. **SQLite para dispositivos móveis**. In: SEMINÁRIO INTERINSTITUCIONAL DE ENSINO, PESQUISA E EXTENSÃO, 22., 2017, Cruz Alta. Anais [...]. Cruz Alta: Universidade de Cruz Alta, 2017. p. 1-5. Disponível em: https://www.academia.edu/85791802/SQLite_para_dispositivos_móveis. Acesso em: 12 ago. 2025.

BEAULIEU, Alan. **Aprendendo SQL: dominando os fundamentos de SQL**. São Paulo: Novatec, 2010. E-book. Disponível em: https://www.amazon.com.br/Aprendendo-SQL-Alan-Beaulieu-ebook/dp/B07B4J9L1M. Acesso em: 12 ago. 2025.

REITZ, Kenneth; SCHLUSSER, Tanya. **O Guia do Mochileiro Python: melhores práticas para desenvolvimento.** São Paulo: Novatec, 2017.

FROTA, Hélder S.; RUVER, Fábio S.; FIGUEIREDO, Josiel. **Implementação de software desktop em Python.** In: ESCOLA REGIONAL DE INFORMÁTICA DE

MATO GROSSO, 13., 2024, Alto Araguaia. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2024. p. [página inicial-final]. Disponível em: https://doi.org/10.5753/eri-mt.2024.245813. Acesso em: 12 ago. 2025.

MCKINNEY, Wes. **Python para Análise de Dados: tratamento de dados com pandas, NumPy & Jupyter**. 3. ed. São Paulo: Novatec, 2023.

VASILIEV, Yuli. **Python para Ciência de Dados: uma introdução prática**. São Paulo: Novatec, 2023.

BARBOSA, Rafael; SARRO, Carlos. **Modelagem de sistemas utilizando a UML: aplicando a engenharia reversa**. 2013. 84 f. Monografia (Bacharelado em Análise de Sistemas e Tecnologia da Informação) – Americana: Faculdade de Tecnologia de Americana, Centro Estadual de Educação Tecnológica Paula Souza, 2013.

RODRIGUES, Paulo. **Aplicação do Conceito Visual *Material Design* no Desenvolvimento de um Protótipo de Interface Gráfica**. 2017. Trabalho de Conclusão de Curso (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas) – Universidade Tecnológica Federal do Paraná 2017. Disponível em: http://repositorio.utfpr.edu.br/jspui/handle/1/16810, Acesso em: 15 jul 2025.

KRUNK, Steve. **Não me Faça Pensar**. 2.ed. Rio de Janeiro: Alta Books, 2014.

YARA, Neves**. User Experience & User Interface Desenvolvimento e Concepção de Projetos Digitais**. Caldas da Rainha: Escola Superior de Artes e Desing, 2018.

GUEDES, Gilleanes. **UML 2 Uma Abordagem Prática**. 3. ed. São Paulo, 2018.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário.** Rio de Janeiro: Elsevier. 2012.

LARMAN, Craig. **Utilizando UML e Padrões.** Porto Alegre: Bookman, 2000.

FOWLER, Martin.**UML Essencial.** Porto Alegre: Bookman, 2005.

TEIXEIRA, Fabricio**. Introdução e boas práticas em UX Design**. São Paulo: Casa do Código, 2014.

VOLPATO, Neri (Org.). Manufatura aditiva: **tecnologias e aplicações da impressão 3D**. São Paulo: Blucher, 2017.