Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Visão computacional e *deep learning* aplicados à classificação de doenças foliares do milho

Gustavo Kermaunar Volobueff

Prof. Dr. Jorge Marques Prates (orientador)
Profa. Dra. Raquel Marcia Müller (coorientadora)

Dourados - MS
2025

Visão computacional e *deep learning* aplicados à classificação de doenças foliares do milho

Gustavo Kermaunar Volobueff

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Gustavo Kermaunar Volobueff e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Dourados, 19 de novembro de 2025

Prof. Dr. Jorge Marques Prates (orientador)
Profa. Dra. Raquel Marcia Müller (coorientadora)

V895v Volobueff, Gustavo Kermaunar

Visão computacional e deep learning aplicados à classificação de doenças foliares do milho / Gustavo Kermaunar Volobueff. – Dourados, MS: UEMS, 2025.
86 p.

Monografia (Graduação) – Ciência da Computação – Universidade Estadual de Mato Grosso do Sul, 2025.
Orientador: Prof. Dr. Jorge Marques Prates
Coorientadora: Profa. Dra. Raquel Marcia Müller

1. Aprendizado profundo 2. Redes neurais convolucionais 3.Visão Computacional 4. Doenças do milho 5. Agricultura de precisão I. Prates, Jorge Marques II. Müller, Raquel Marcia III. Título

CDD 23. ed. - 006.32

Ficha Catalográfica elaborada pela bibliotecária da Universidade Estadual de Mato Grosso do Sul (UEMS)
Elaine Freire Lessa – CRB-1 / 3721

Curso de Ciência da Computação
Universidade Estadual de Mato Grosso do Sul

Visão computacional e *deep learning* aplicados à classificação de doenças foliares do milho

Gustavo Kermaunar Volobueff

Novembro de 2025

Banca Examinadora:

Prof. Dr. Jorge Marques Prates (Orientador)
Área de Computação - UEMS

Profa. Dra. Raquel Marcia Müller (Coorientadora)
Área de Computação - UEMS

Prof. Dr. Diogo Fernando Trevisan
Área de Computação - UEMS

Prof. Dr. Ricardo Luís Lachi
Área de Computação - UEMS

Dedico este trabalho primeiramente aos meus pais, por todo o amor, incentivo e apoio incondicional que foram a base para cada passo desta jornada acadêmica.

À minha namorada, pela paciência, carinho e por ter caminhado ao meu lado desde o início da graduação, tornando tudo mais leve.

E ao meu orientador, pela orientação e dedicação indispensável à realização deste projeto.

Agradecimentos

Aos meus pais, Ronaldo e Érica, dedico a mais profunda gratidão. Pelo amor, sacrifício e apoio incondicional que foram o alicerce de toda a minha jornada acadêmica. Esta conquista é a materialização de um sonho que nasceu primeiro no coração dos senhores, e poder proporcionar-lhes este orgulho é minha maior felicidade.

À minha namorada, Railla, a quem tive a alegria de conhecer neste ambiente acadêmico. Agradeço por ser a inspiração e a força nos momentos de maior desafio. Sua presença tornou esta caminhada mais leve, e desejo que sua própria jornada seja sempre próspera e repleta de sucesso.

Ao meu orientador, Prof. Dr. Jorge, pela orientação segura, pela paciência em sanar todas as dúvidas e pelo incentivo constante, mesmo diante das adversidades o qual infelizmente o acometeram. Sua conduta profissional e humana é uma grande inspiração que poderei levar durante toda minha carreira profissional e vida.

À minha coorientadora, Profª. Drª. Raquel, pela valiosa colaboração, pela pronta disponibilidade em me auxiliar durante todo o desenvolvimento deste trabalho e pela honra de presidir a banca examinadora. Sua competência e dedicação são igualmente inspiradoras.

A todos os mencionados, e àqueles que de alguma forma contribuíram, meu sincero e profundo agradecimento.

Resumo

As doenças foliares em culturas agrícolas de grande importância, como o milho, representam um desafio crítico que impacta negativamente a produtividade, a economia do agronegócio e a segurança alimentar. O diagnóstico tradicional, baseado na inspeção visual, é frequentemente subjetivo e ineficaz em larga escala, criando a necessidade de ferramentas automáticas e precisas. Para endereçar esta lacuna, este trabalho propõe o desenvolvimento e a validação de um sistema de classificação de alta performance para patologias em folhas de milho, utilizando técnicas de aprendizado profundo e visão computacional. Para tal, foram comparadas três arquiteturas de Redes Neurais Convolucionais, sendo elas a **ResNet101**, **MobileNetV3-Large** e **EfficientNetV2M**, treinadas em dois conjuntos de dados públicos (**PlantVillage + PlantDoc** e **CD&S**) através de uma robusta *pipeline* de pré-processamento, com balanceamento de classes e aplicação de técnicas de regularização. Dentre os modelos avaliados, a arquitetura **EfficientNetV2M** apresentou um desempenho superior, alcançando uma acurácia de validação de 99.72% no *dataset* CD&S com imagens de *480x480 pixels*, um desempenho que supera *benchmarks* recentes na literatura. Conclui-se, portanto, que a abordagem proposta é robusta e altamente eficaz, validando o potencial das CNNs como uma ferramenta de suporte à decisão de alta precisão para agricultores, cuja viabilidade prática foi demonstrada pela implementação do modelo em um protótipo de aplicação *Web* interativa para inferência em tempo real.

Palavras-chave: Deep Learning, Redes Neurais Convolucionais, Visão Computacional, doenças do milho, agricultura de precisão.

Abstract

Foliar diseases in critical agricultural crops, such as corn (maize), represent a significant challenge that negatively impacts productivity, the agribusiness economy, and food security. Traditional diagnosis, based on visual inspection, is often subjective and ineffective at scale, creating a need for automated and accurate tools. To address this gap, this work proposes the development and validation of a high-performance classification system for corn leaf pathologies using deep learning and computer vision techniques. To this end, three Convolutional Neural Network (CNN) architectures, **ResNet101**, **MobileNetV3-Large**, and **EfficientNetV2M**, were trained and compared on two public datasets (**PlantVillage + PlantDoc** and **CD&S**) through a robust preprocessing pipeline with class balancing and the application of regularization techniques. Among the evaluated models, the **EfficientNetV2M** architecture showed superior performance, achieving a validation accuracy of **99.72%** on the CD&S dataset with 480x480 pixel images, a performance that surpasses recent benchmarks in the literature. It is therefore concluded that the proposed approach is robust and highly effective, validating the potential of CNNs as a high-precision decision support tool for farmers, whose practical feasibility was demonstrated by the implementation of the model in an interactive web application prototype for real-time inference.

Keywords: Deep Learning, Convolutional Neural Networks, Computer Vision, corn diseases, precision agriculture.

Sumário

Lista de Siglas

IA	Inteligência Artificial
CNN	Convolutional Neural Networks
DDGS	Dried Distiller's Grains with Solubles
ML	Machine Learning
TLU	Threshold Logic Unit
DL	Deep Learning
RNC	Redes Neurais Convolucionais
RPN	Region Proposal Networks
ROI	Region of Interest
PSOA-NFIS	Adaptive Network based Fuzzy Inference System optimized with Particle Swarm Optimization
SVM	Support Vector Machines
SE	Squeeze and Excitation
API	Application Programming Interface

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A agricultura, pilar da economia global e da segurança alimentar, enfrenta o desafio contínuo de otimizar a produção de forma sustentável. Neste contexto, a cultura do milho, uma das mais importantes do mundo, é constantemente ameaçada por doenças foliares que podem causar perdas significativas de produtividade. O diagnóstico preciso e ágil dessas patologias é, portanto, uma etapa crítica para um manejo agrícola eficaz. Este trabalho aborda este problema por meio da aplicação de técnicas de *Inteligência Artificial* (IA), propondo o desenvolvimento e a validação de um sistema de classificação automática de doenças do milho baseado em *Redes Neurais Convolucionais* (CNNs).

1.1 Contextualização e Problemática

O Brasil destaca-se como uma potência agrícola global, e a cultura do milho (*Zea mays L.*) é um de seus ativos mais valiosos. A produtividade nacional já demonstrou um potencial superior a *16t ha-1* (toneladas por hectare) CRUZ et al. (2010). Em escala mundial, o milho consolidou-se como a maior cultura agrícola, superando a marca de 1 bilhão de toneladas produzidas anualmente, com uma versatilidade que abrange cerca de 3.500 aplicações distintas, desde a alimentação humana e animal até a produção de biocombustíveis e polímeros (CONTINI et al., 2019).

Contudo, este potencial produtivo é constantemente ameaçado pela incidência de doenças foliares. Essas patologias, cuja prevalência varia conforme a região, o clima e a época do ano, impactam diretamente a capacidade fotossintética da planta[1], resultando em perdas de produtividade e prejuízos econômicos (CONTINI et al., 2019). O desafio é agravado pela constante mutação dos patógenos[2], que podem desenvolver resistência a defensivos agrícolas,

[1] Capacidade da planta realizar fotossíntese.

[2] Organismo, bactéria, vírus ou fungos capaz de causar doenças num hospedeiro, sendo este humano, animal ou

exigindo um monitoramento cada vez mais sofisticado.

O método tradicional de diagnóstico, adotado pela maioria dos produtores, baseia-se na inspeção visual manual da lavoura. Este processo é inerentemente problemático, especialmente em grandes extensões de terra, por ser:

- **Subjetivo e Impreciso:** depende da experiência do inspetor e é suscetível a erros humanos;
- **Custo Elevado:** demanda tempo, mão de obra e, frequentemente, a consulta a especialistas e agrônomos;
- **Pouco Escalável:** a inspeção manual de grandes áreas é lenta e, muitas vezes, inviável, retardando a detecção e a tomada de decisão.

Para superar essas limitações, a recente evolução do *Aprendizado Profundo* (*Deep Learning*) e, em particular, das Redes Neurais Convolucionais (CNNs), abriu novas fronteiras para a automação de tarefas baseadas em reconhecimento de padrões visuais. A Inteligência Artificial (IA) tem se consolidado como uma tecnologia transformadora na agricultura de precisão, atuando como o motor analítico que conecta dados coletados por sensores, drones e satélites a ações agronômicas otimizadas (BASSOI et al., 2024).

No campo da fitopatologia[3], a *Visão Computacional* se destaca como uma ferramenta de alto potencial. Modelos de CNN são capazes de aprender a identificar características visuais sutis em imagens, como as texturas, cores e formas das lesões causadas por diferentes doenças, com um nível de precisão que pode se igualar ou até superar o de especialistas humanos (BASSOI et al., 2024). Esta capacidade permite o desenvolvimento de sistemas de diagnóstico automático, oferecendo uma alternativa precisa, objetiva e escalável aos métodos tradicionais.

1.2 Justificativa e Proposta de Solução

A lacuna entre a necessidade de um monitoramento fitossanitário[4] ágil e as limitações dos métodos manuais justifica a busca por soluções tecnológicas. A detecção precoce de doenças é crucial, pois permite a aplicação de defensivos de forma direcionada e no momento correto, uma prática que não apenas aumenta a eficácia do tratamento, mas também reduz custos e o impacto ambiental. A aplicação de agrotóxicos em áreas específicas, guiada por um diagnóstico preciso, é um dos pilares da agricultura sustentável.

Diante deste contexto, este trabalho propõe o desenvolvimento e a validação de um *pipeline* de aprendizado profundo para a classificação de doenças foliares do milho. A hipótese

planta.

[3]Ramo da ciência que estuda as doenças das plantas, incluindo suas causas (agentes bióticos como fungos, bactérias, vírus, e abióticos como deficiências nutricionais e problemas climáticos), sua diagnose, evolução, e controle.

[4]São os produtos, processos e tecnologias destinados à sanidade vegetal, ou seja, ao controle de pragas e doenças.

central é que, ao empregar arquiteturas de CNNs modernas, combinadas com um conjunto robusto de técnicas de pré-processamento, aumento de dados e estratégias de treinamento, é possível criar um modelo classificador com um nível de acurácia superior aos métodos existentes na literatura, validando sua aplicabilidade prática.

1.3 Objetivos

O objetivo geral deste trabalho é desenvolver, treinar e comparar três arquiteturas de CNNs para a tarefa de classificação de doenças foliares em milho, culminando na implementação do modelo de melhor desempenho em uma aplicação *Web* interativa como prova de conceito.

1.3.1 Objetivos Específicos

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- Realizar a coleta, a organização e o pré-processamento de dois conjuntos de dados públicos de imagens de folhas de milho;
- Implementar e treinar três arquiteturas de CNN de alto desempenho: **ResNet101**, **MobileNetV3-Large** e **EfficientNetV2M**;
- Avaliar experimentalmente o impacto de diferentes resoluções de imagem de entrada no desempenho do modelo mais promissor;
- Aplicar um conjunto de técnicas de regularização e uma estratégia de ajuste fino (*fine-tuning*) para otimizar a performance e a capacidade de generalização dos modelos;
- Avaliar quantitativa e qualitativamente os modelos treinados e comparar os resultados obtidos com os de trabalhos correlatos na literatura;
- Desenvolver uma aplicação *Web* funcional para permitir a inferência em tempo real com o modelo de melhor desempenho.

1.4 Estrutura do Trabalho

Este documento está organizado da seguinte forma: o **Capítulo 2** apresenta o referencial teórico que fundamenta os conceitos de redes neurais e visão computacional aplicados. O **Capítulo 3** descreve em detalhes a metodologia empregada, desde a preparação dos dados até a arquitetura dos modelos. O **Capítulo 4** apresenta e discute os resultados obtidos nos experimentos. Finalmente, o **Capítulo 5** apresenta a conclusão deste trabalho, sumarizando os achados e propondo direções para pesquisas futuras.

CAPÍTULO 2

Revisão da Literatura

Este capítulo apresenta a fundamentação teórica que alicerça esta pesquisa. A revisão inicia-se pelo domínio de aplicação, o cultivo do milho, abordando sua relevância, os atuais avanços tecnológicos na agricultura e as dificuldades da Agricultura 4.0. O foco do problema é então delimitado na seção sobre doenças na cultura do milho.

Em seguida, o capítulo introduz as bases teóricas da solução tecnológica, começando pela Inteligência Artificial e aprofundando-se nas técnicas de *Machine Learning*. Esta segunda parte detalha os conceitos centrais para o desenvolvimento deste trabalho, como a Aprendizagem Supervisionada, os métodos de Classificação e Regressão, e a Matriz de Confusão como ferramenta de avaliação.

2.1 Cultivo do Milho

O **milho** destaca-se como um dos principais cereais cultivados pelo agronegócio brasileiro, contribuindo significativamente para a consolidação do Brasil como um dos maiores produtores e exportadores mundiais do grão, tendo a China como um dos principais mercados de destino. Na safrinha [1] ocorrida entre os anos de 2023 e 2024, a estimativa de colheita atingiu **111,6 milhões** de toneladas, com uma área cultivada de aproximadamente 20,6 milhões de hectares e uma produtividade média de 5,415 kg por hectare (GUIMARÃES, 2024). Segundo CRUZ et al. (2010), no Brasil já foi obtido uma produtividade superior a *16t ha-1* (toneladas por hectare), registrados em concursos de produtividade de milho realizados por órgãos de assistência técnica e extensão rural e empresas no qual produzem sementes.

[1] A safra refere-se ao período principal de cultivo e colheita agrícola do ano, geralmente realizado durante a estação chuvosa. Já a safrinha, também chamada de segunda safra, é o cultivo realizado logo após a safra principal, aproveitando o mesmo solo e resíduos, sendo comum no cultivo do milho em regiões como o centro-oeste.

2.1.1 Relevância do Milho

O milho é um cereal de ampla distribuição no território brasileiro, desempenhando papel estratégico na matriz agropecuária nacional. Trata-se de uma cultura essencial tanto para a alimentação humana quanto para a nutrição animal, estando presente em mais de dois milhões de estabelecimentos rurais (CONTINI et al., 2019). Sua aplicação é diversificada, abrangendo os setores alimentício, de bebidas, de biopolímeros, de biocombustíveis e de rações, o que o torna um insumo de elevada relevância para diferentes cadeias produtivas e comerciais.

Levando em consideração a produção agrícola no estado de Mato Grosso do Sul, dentre todos os cereais cultivados, o milho ganha destaque entre os produtores, incentivando-os à realizar maiores investimentos no cereal após a implantação de duas usinas de etanol extraído do milho no estado (LAMAS, 2022). Elas estão localizadas em Dourados e Maracaju, tendo a capacidade de produção de aproximadamente 3 milhões de toneladas de milho por ano.

Além da agricultura, o estado de Mato Grosso do Sul é um grande polo de criação de aves e suínos, no qual estas espécies possuem como base de alimentação o grão do milho, tanto puro quanto em misturas de rações animais. Além disso este cereal pode ser transformado em proteína animal, etanol, óleo e DDGS (*Dried Distiller's Grains with Solubles* - Grãos Secos com Solúveis de Destilaria de Milho), sendo uma grande porta de entrada para novas oportunidades e para o crescimento comercial local. Além disso, o estado é um grande exportador do grão (LAMAS, 2022).

2.1.2 Avanços Tecnológicos na Agricultura

Em um contexto de acelerado crescimento populacional, há um aumento proporcional na demanda por alimentos e produtos derivados de grãos, o que torna imprescindível a adoção de novas tecnologias voltadas ao aprimoramento, à otimização e à rentabilidade da produtividade agrícola. Nesse cenário, a gestão eficiente dos recursos disponíveis assume papel central (SILVA; CAVICHIOLI, 2020). Para tanto, são coletados dados diretamente do campo, onde ocorrem variações significativas em fatores como clima, tipo de solo e características da lavoura, resultando em operações que envolvem elevada heterogeneidade de cenários e condições.

Portanto, observa-se nos últimos anos um alto crescimento na utilização de tecnologias da informação no contexto da agricultura (SILVA; CAVICHIOLI, 2020). Esse fenômeno é comumente denominado *Agricultura 4.0*[2]. A adoção dessas inovações tecnológicas visa, primordialmente, à redução de custos operacionais, ao incremento da produtividade (tanto na produção de grãos quanto na cadeia alimentícia em geral), à racionalização do uso de recursos naturais e à minimização de perdas ao longo do processo produtivo.

[2]Refere-se a um conjunto de tecnologias voltadas à otimização da produção e da gestão no setor agrícola, por meio do aprimoramento do controle, do monitoramento e dos métodos de trabalho, utilizando-se de *softwares* e outros sistemas digitais.

2.1.3 Dificuldades da Agricultura 4.0

Um dos principais problemas para a efetiva implementação da *Agricultura 4.0* no território brasileiro é a carência de conectividade. O país ainda não dispõe de uma infraestrutura suficientemente robusta para suportar grandes volumes de transmissão de dados, o que compromete a cobertura integral das áreas rurais e impede a plena mobilidade tecnológica no campo (SILVA; CAVICHIOLI, 2020). Apesar de o Brasil ser reconhecido internacionalmente por sua elevada produtividade agrícola, sendo frequentemente referido como o "celeiro do mundo", ainda enfrenta limitações significativas, como escassez de oportunidades e baixo índice de inovações no setor do agronegócio. Nesse contexto, torna-se fundamental que os produtores rurais adotem tecnologias ao longo de toda a cadeia produtiva, com o objetivo de aprimorar e otimizar seus processos.

Para assegurar o desenvolvimento, a competitividade e a sustentabilidade das atividades agrícolas mediante o uso de novas tecnologias, é imprescindível o fortalecimento das telecomunicações, da automação, das transmissões e da análise de dados no ambiente rural (SILVA; CAVICHIOLI, 2020). No entanto, o desafio inicial reside na democratização do acesso a essas tecnologias, o que demanda estratégias voltadas à expansão da cobertura digital no campo. Tais estratégias incluem a criação e implementação de novas empresas fornecedoras de soluções tecnológicas, a ampliação da conectividade rural e a instalação de redes públicas de acesso à internet.

2.1.4 Doenças na Cultura do Milho

Contudo, uma dificuldade que afeta o cultivo do milho em todo território nacional são as doenças que afetam a planta (CASELA; FERREIRA; PINTO, 2006). Diversos fatores bióticos e abióticos[3] podem influenciar negativamente a produtividade, como a ocorrência de pragas e doenças, que, em alguns casos, são responsáveis por perdas superiores a *40%* na produção (GOMES et al., 2024). Assim, é necessário realizar o reconhecimento destes agentes prejudiciais a cultura, bem como ambientes favoráveis para seus desenvolvimentos, a fim de realizar um melhor manejo da planta garantindo uma produtividade maior.

O monitoramento de doenças realizados pela renomada Embrapa Milho e Sorgo Embrapa (2020) juntamente com empresas privadas demonstram que a **ferrugem polissora**, **ferrugem tropical**, **ferrugem comum**, **mancha branca**, **cercosporiose**, **helmintosporiose** e os **enfezamentos pálido e vermelho** são as principais enfermidades no qual afetam a cultura do milho atualmente, atingindo de formas diferentes cada região em diferentes estações do ano as produções de milho em território brasileiro. Além das doenças citadas, a **antracnose foliar** têm sido um desafio constante nos últimos anos, com o aumento da sua severidade nas culturas. Além disso, os fungos *Stenocarpella maydis* e *Stenocarpella macrospora* têm levado a acarretar podridões ao milho, no qual antes eram mais comuns na região Sul do país e em algumas

[3] Biótico refere-se a seres vivos do ambiente e abiótico a seres não vivos do ambiente, como luz e água.

áreas do Centro-Oeste, agora podem ser encontradas em outras regiões (CASELA; FERREIRA; PINTO, 2006).

Embora o artigo mencionado anteriormente, CASELA, FERREIRA e PINTO (2006), possua certo tempo desde sua divulgação, as doenças descritas neste artigo ainda persistem na atualidade, sofrendo mutações ao longo dos anos. De acordo com o artigo publicado pela Bayer em 18 de setembro de 2024, empresa amplamente consolidada no setor agrícola, algumas doenças continuam causando perdas significativas caso não sejam tratadas de forma adequada CropScience (2024), entre elas, destacam-se: **mancha branca**, **cercosporiose**, **antracnose**, **ferrugem do milho** e **enfezamento**. É possível também citar a **helmintosporiose**, causada pelo fungo *Exserohilum turcicum* (AGRO, 2022). Essa doença foliar afeta em grande frequência as culturas de milho safrinha, onde, em casos severos, pode levar a perca de aproximadamente 50% do potencial produtivo das terras.

2.2 Inteligência Artificial

O desafio descrito de reconhecer e diferenciar com precisão um vasto conjunto de doenças (como a ferrugem polissora, a cercosporiose e os enfezamentos) a partir de seus sintomas visuais é uma tarefa de alta complexidade. A subjetividade, o tempo e o custo da análise humana especializada, discutidos anteriormente, são os principais gargalos para um manejo eficiente. É exatamente para automatizar tarefas complexas de reconhecimento de padrões e tomada de decisão que se volta a área da Inteligência Artificial.

A área da **Inteligência Artificial**, sendo comumente conhecida somente por IA, tenta não somente compreender as coisas, mas também construir *agentes inteligentes*, sendo estes *softwares* no qual tem a capacidade de identificar e computar diversas situações diferentes, com obstáculos e novas informações, agindo de um modo seguro e eficaz (RUSSELL; NORVIG, 2010).

2.3 *Machine Learning*

A questão central, então, é: como um "agente inteligente"aprende a agir de forma eficaz sem ser explicitamente programado para cada cenário? A resposta está na principal subárea da IA dedicada a isso: o *Machine Learning* (ML).

A área de Machine Learning, também chamado de **Aprendizagem de Máquina**, é composto por algoritmos que conseguem aprender a partir de dados, sendo a principal técnica por trás da automação (SILVA et al., 2023). Vale ressaltar a importância da estatística neste ramo, sendo aproveitada por conta de suas ferramentas que são essenciais para o processo de construção destes modelos.

O grande objetivo destes sistemas autônomos é desenvolver formas de aprender por si próprios, sendo por experiências e comportamentos já realizados, caracterizado por aprendizagem *não supervisionada*; ou com a entrada de *Datasets*[4], sendo este aprendizagem *supervisio-*

[4]Um conjunto de dados, ou uma coleção de exemplos (como imagens, tabelas ou textos).

nada; e a interação com um ambiente, denominada aprendizagem *por reforço* Russell e Norvig (2010), por exemplo, ao jogar um *video game*.

2.3.1 Classificação e Regressão

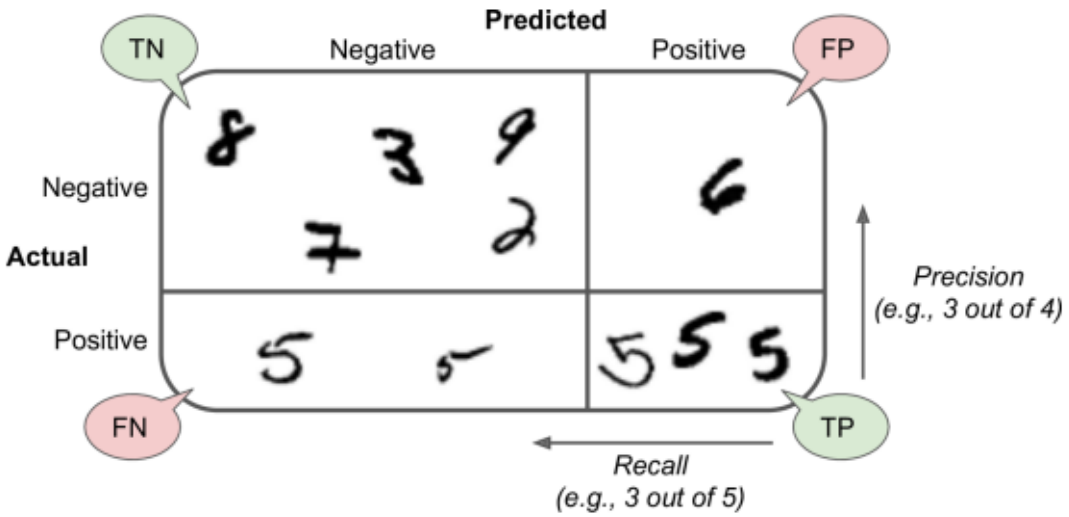
Dentre as abordagens de aprendizado citadas, a *aprendizagem supervisionada* é o foco principal. Este tipo de aprendizado, que utiliza *Datasets* rotulados, é tipicamente aplicado para resolver duas categorias de problemas: *classificação* e *regressão*. A distinção entre os dois é fundamental e baseia-se na natureza da saída que o agente deve prever.

Quando a saída entregue por um agente for um conjunto finito de valores, por exemplo, uma letra do alfabeto (a, b, c, ..., z), ou também uma resposta binária, como "sim"ou "não", este problema de aprendizagem será denominado *classificação*. Caso a saída seja um valor numérico, por exemplo, o valor de um produto ou a altura de uma pessoa, este será dito como problema de aprendizagem por *regressão* (RUSSELL; NORVIG, 2010).

2.3.2 Matriz de Confusão

Um instrumento muito utilizado em problemas de classificação é chamado de **Matriz de Confusão**, sendo este uma tabela bidimensional que realiza incrementos de quantas vezes uma categoria dentro das propostas ao agente foi classificada corretamente ou classificada incorretamente (RUSSELL; NORVIG, 2010). Geralmente, usa-se as classificações de *verdadeiro positivo*, *verdadeiro negativo*, *falso positivo* ou *falso negativo*. A **Figura 2.1** ilustra um exemplo da matriz de confusão resultante de um algoritmo classificador de diferentes imagens de números.

Figura 2.1: matriz de confusão ilustrada.



Fonte: Géron (2019, p. 92)

2.3.3 Aprendizagem Supervisionada

O **Aprendizado Supervisionado** é realizado quando um agente observa pares de entrada e saída, aprendendo uma função que mapeia os valores entre elas. Estas saídas são denominadas *rótulos*, no qual dizem o que é a entrada (RUSSELL; NORVIG, 2010). Um exemplo disso é uma entrada sendo a imagem de um grão, acompanhada de um rótulo dizendo qual é o grão da imagem, sendo ele milho, arroz ou soja. Com isso, o agente irá aprender uma função que consegue prever corretamente, ou não, os rótulos de um conjunto de entradas.

Formalmente podemos definir o aprendizado supervisionado como:

Dado um *conjunto de treinamento* contendo N pares de exemplos de entrada e saída, representados por

$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N),$$

em que cada par foi gerado por uma função desconhecida $y = f(x)$, descobrir uma função h que se aproxime da função verdadeira f (RUSSELL; NORVIG, 2010).

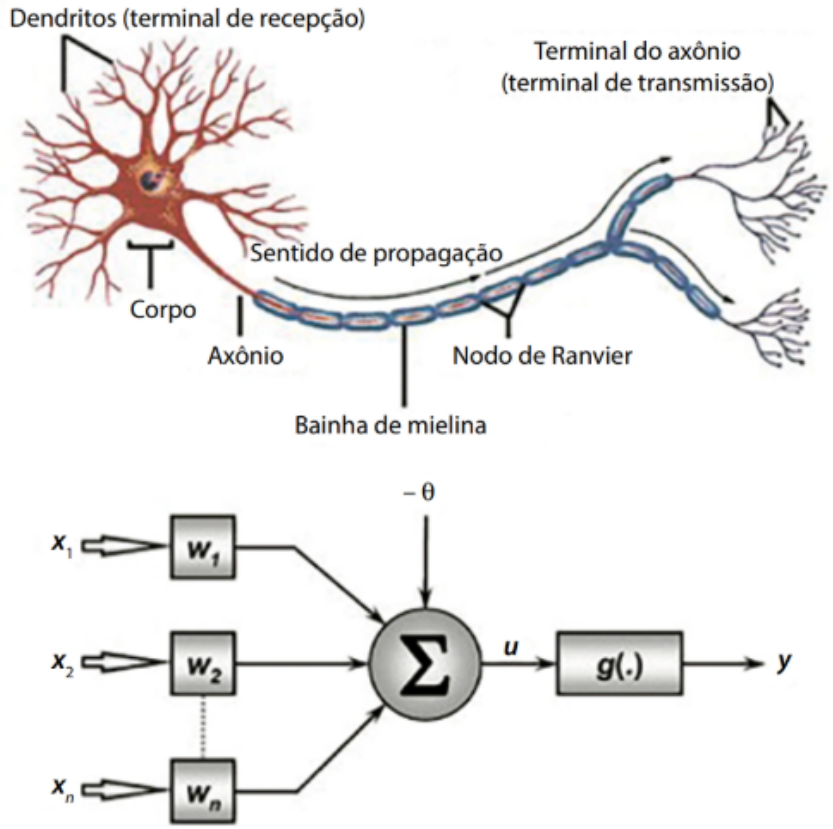
2.3.4 Aprendizagem Não Supervisionada

Diferentemente do aprendizado supervisionado, o **Aprendizado Não Supervisionado** aprende padrões nos conjuntos de entrada, sem haver nenhum tipo de rótulo associado a eles. Uma estratégia comum para realizar este aprendizado é o *agrupamento*, onde o agente realiza a separação de grupos que são possivelmente considerados úteis (RUSSELL; NORVIG, 2010). Um exemplo disso é entregar imagens de automóveis para um agente, a partir das quais ele irá realizar a separação de cada tipo de imagem semelhante, que pode ser considerado, por exemplo, um caminhão, carro ou motocicleta.

2.4 Redes Neurais Artificiais

Com estudos focados na simulação do processamento mental humano e na resolução de problemas por meio de modelos computacionais, o estudo das **Redes Neurais Artificiais** começou a tomar força, sendo uma subárea da inteligência artificial que imita o funcionamento do cérebro humano com relação ao processamento e aprendizagem de dados. As redes neurais artificiais simulam as atividades dos neurônios mapeando-os em modelos matemáticos (SILVA et al., 2023). Este conceito está ilustrado na **Figura 2.2**.

Figura 2.2: representação de um neurônio natural e um artificial.

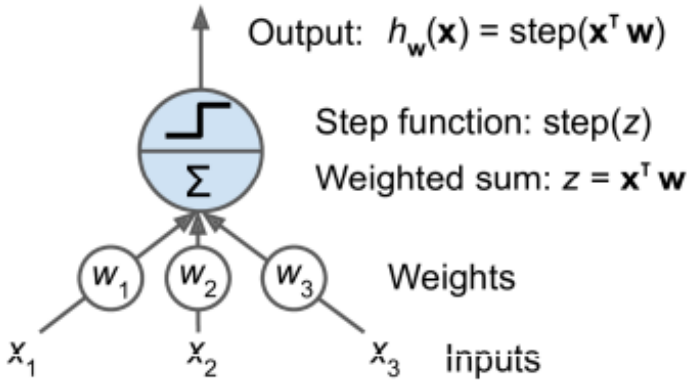


Fonte: Silva et al. (2023, p. 112).

2.4.1 *Perceptron*

O ***Perceptron*** é considerado o modelo mais simples da arquitetura de redes neurais artificiais. Descrito primeiramente por Frank Rosenblatt Rosenblatt (1957), o *perceptron* é inspirado por um neurônio artificial dito *threshold logic unit* (TLU), classificado como uma unidade de processamento para números com n entradas $x_1, x_2, x_3, ..., x_n$ e uma saída y, ou $h_w(\mathbf{x})$, como pode ser observado na **Figura 2.3**. Cada entrada x_i é associada com um peso w_i, onde uma multiplicação entre peso e entrada é realizada, e após isso uma função $step$ é aplicada a esta soma: $h_w(\mathbf{x}) = step(\mathbf{x}^T\mathbf{w})$ (GÉRON, 2019).

Figura 2.3: *threshold logic unit*.

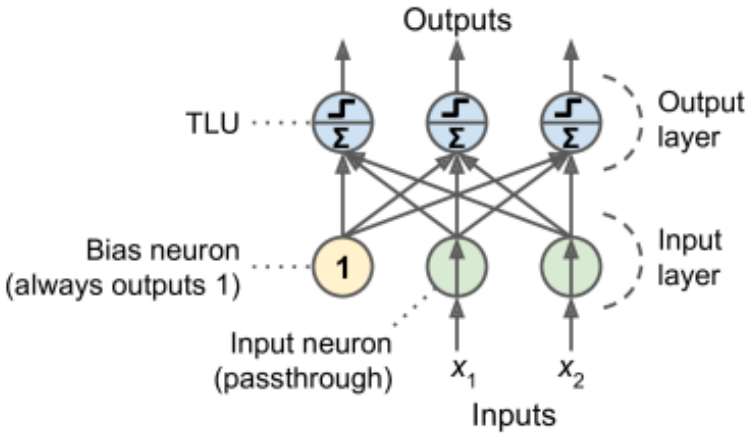


Fonte: Géron (2019, p. 284).

Simplesmente um *perceptron* é composto por uma única camada de TLUs, onde cada TLU está conectado a todas as entradas e esta camada de TLUs produzem uma saída. As entradas do *perceptron* são fornecidas a neurônios especiais, chamados de neurônios de entrada, no qual fornecem aos TLUs exatamente o que receberam como entrada. Estes neurônios de entrada em conjunto são chamados de *camada de entrada*. Além disso também pode ser adicionado uma característica a essas entradas denominada viés, no qual sempre produzem como saída o valor 1 ($b = 1$), sendo geralmente representada por um neurônio especial chamado de *neurônio de viés* (GÉRON, 2019). Com isso, a saída de um *perceptron* é simplesmente uma soma ponderada das entradas juntamente com o viés extra. Quando esta soma é passada por uma função de ativação, uma saída é gerada.

Um *perceptron* que possui a característica de classificar instâncias simultâneas em três classes binárias diferentes, sendo este um *classificador de múltiplas saídas* está representado na **Figura 2.4**.

Figura 2.4: arquitetura de um *Perceptron* com dois neurônios de entrada, um neurônio de viés, e três neurônio de saída.

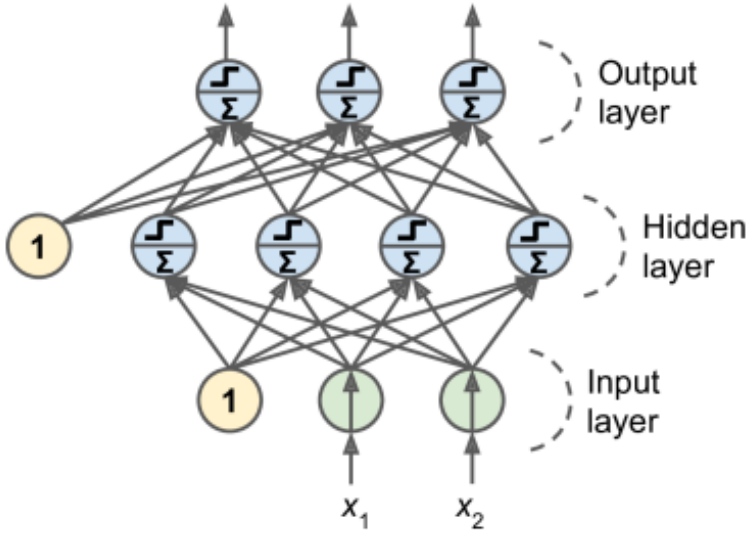


Fonte: Géron (2019, p. 286).

2.4.2 *Multilayer Perceptron*

Um ***Multilayer Perceptron*** consiste em uma camada de entrada, de passagem direta, e uma ou mais camadas de TLUs, sendo estas denominadas *hidden layers*, ou camadas ocultas. Há também uma camada final chamada camada de saída, sendo esta a final composta por TLUs (GÉRON, 2019). Na **Figura 2.5** podemos observar um *multilayer perceptron* composto por duas entradas (x_1 e x_2), uma camada oculta composta por quatro neurônios e posteriormente três neurônios de saída. Os neurônios em coloração amarelada (neurônios de viés) geralmente não são representados, diferentemente deste caso.

Figura 2.5: arquitetura de um *Multilayer Perceptron*.



Fonte: Géron (2019, p. 289).

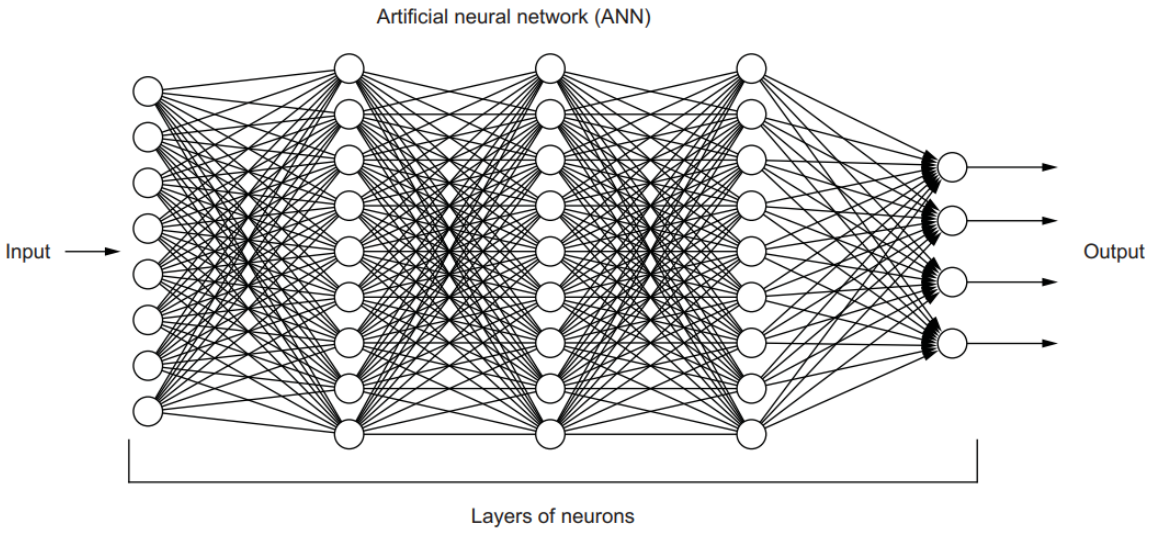
2.5 *Deep Learning*

O **Aprendizado Profundo**, popularmente conhecido por ***Deep Learning*** (DL), é um grande conjunto de técnicas onde hipóteses se transformam em circuitos algébricos complexos. Este grande conjunto possui circuitos no qual normalmente são compostos por muitas *camadas*, significando que o caminho para o processamento de uma entrada pode ser realizado em várias etapas, se aproximando ainda mais do funcionamento de um cérebro orgânico (RUSSELL; NORVIG, 2010). A base disso são as redes neurais artificiais vistas anteriormente, no qual constroem os circuitos de DL.

O aprendizado profundo inciou-se na tentativa de modelar redes de neurônios do cérebro utilizando circuitos de computadores (MCCULLOCH; PITTS, 1943). Por conta deste estudo, as redes no qual foram treinadas utilizando métodos de aprendizado profundo geralmente são classificadas como **redes neurais artificiais**, mesmo que a semelhança entre neurônios orgânicos e computacionais seja superficial (RUSSELL; NORVIG, 2010).

Como citado anteriormente, o *deep learning* consiste em uma rede neural artificial de diversas camadas, sendo semelhante ao *multilayer perceptron*. Na **Figura 2.6** podemos observar um exemplo de rede neural artificial de múltiplas camadas.

Figura 2.6: uma rede neural artificial consistindo de camadas de nós ("neurônios") conectados por arestas.



Fonte: Elgendy (2020, p. 38).

2.5.1 Pesos e Camadas

As **Redes Neurais Profundas** são grafos computacionais do tipo *feedforward*, o que significa que eles possuem uma estrutura de camadas, onde os dados fluem em uma única direção, seguindo da camada de entrada à camada de saída. Estes grafos são compostos por milhares de nós, que são unidades simples interconectadas que realizam somas ponderadas de suas entradas, sendo:

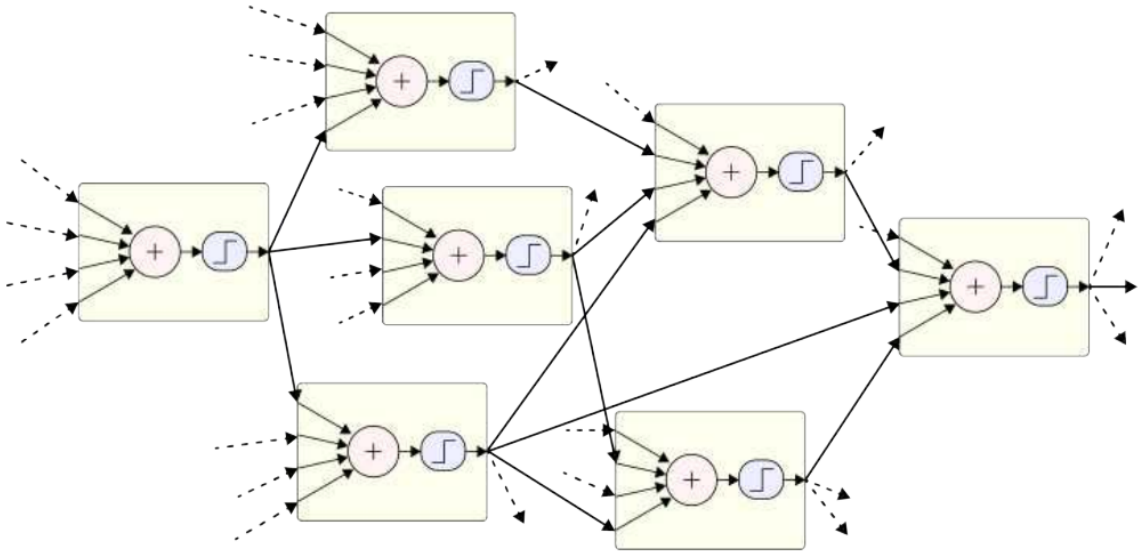
$$s_i = \mathbf{w}_i^T x_i + b_i$$

no qual segue uma *função de ativação* de remapeamento não linear

$$y_i = h(s_i),$$

Os x_i são as entradas, w_i representa seus pesos e b_i seus viéses ajustáveis, s_i é a saída da soma ponderada e y_i é a saída final após s_i ter sido calculada na função de ativação $h(s_i)$. As saídas de cada estágio que geralmente são denominadas de *activations* são alimentadas nas unidades dos estágios posteriores (SZELISKI, 2010), assim como ilustrado na **Figura 2.7**.

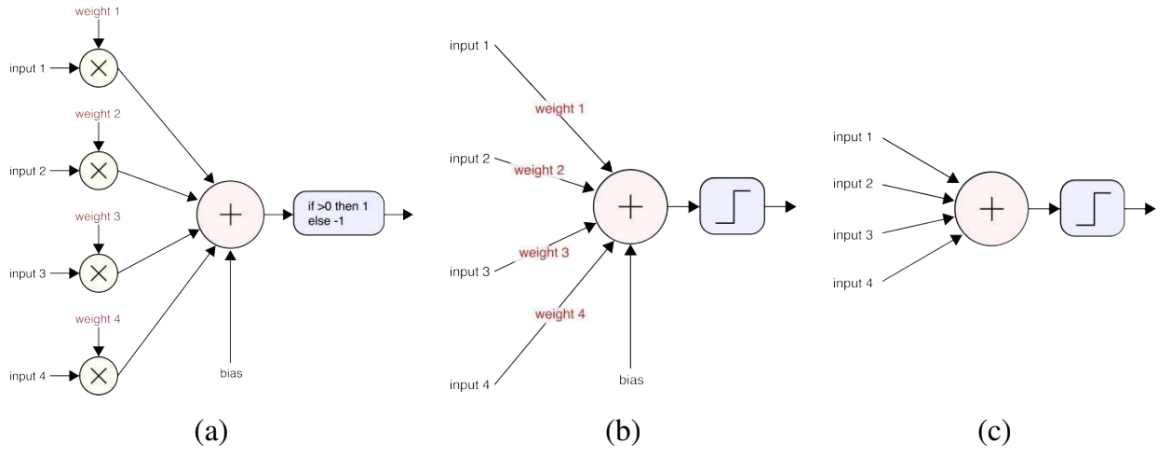
Figura 2.7: uma rede multi-camadas, mostrando como as saídas de uma unidade são alimentadas em unidades adicionais.



Fonte: Szeliski (2010, p. 271).

A **Figura 2.8** ilustra uma unidade *perceptron* (a) mostrando os pesos sendo multiplicados pelas entradas; (b) com os pesos indicados nas conexões de entrada; e (c) a forma mais comum, com os pesos e o viés não sendo representados. Uma *função de ativação não linear* segue a soma ponderada.

Figura 2.8: representação de pesos em Redes Neurais.



Fonte: Szeliski (2010, p. 270).

2.5.2 Funções de Ativação

Segundo os estudos propostos por McCulloch e Pitts McCulloch e Pitts (1943), uma unidade (nó de uma rede) faz o cálculo da soma ponderada das entradas dos nós anteriores à ele, aplicando uma função não linear, resultando assim em sua saída. Tendo a_j a saída de uma unidade j e seja $w_{i,j}$ o peso referente à ligação de uma unidade i relacionada a outra unidade j, tendo assim

$$a_j = g_j\left(\sum\nolimits_i w_{i,j}a_i\right) \equiv g_j(in_j),$$

onde g_j é uma **Função de Ativação** no qual está associada ao nó j e in_i refere-se à soma ponderada das entradas da unidade j (RUSSELL; NORVIG, 2010).

Há diversas funções de ativação diferentes, dentre as quais, as mais usuais são:

- A função logística, também chamada de ***sigmoide***:

$$\sigma(x) = 1/(1 + e^{-x}).$$

- A função ***ReLU***, que significa *rectified linear unit* (unidade linear retificada):

$$ReLU(x) = max(0, x).$$

- A função ***softplus***, uma versão adaptada da função *ReLU*:

$$softplus(x) = log(1 + e^x).$$

- A função ***tanh***:

$$tahn(x) = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

2.6 Visão Computacional

A visão é o canal sensorial que permite a observação do mundo. A maioria dos agentes visuais opera por sensoriamento passivo, o que significa que eles apenas recebem a luz ambiente, em vez de emitir luz própria para obter um estímulo. Do outro lado há o sensoriamento ativo, que consiste no envio de um sinal, por exemplo, de um radar, que envia uma onda que se propaga ao redor de certa superfície, fazendo assim que se sinta o reflexo desta onda (RUSSELL; NORVIG, 2010).

Na área de **Visão Computacional**, a ideia principal é descrever o que pode ser visualizado no mundo observável, com uma ou mais imagens, reconstruindo suas propriedades, como formato, iluminação e cores (SZELISKI, 2010). Os humanos com milhares de anos de experiência, conseguem facilmente descrever e classificar imagens, diferentemente de algoritmos de visão computacional, no qual precisam constantemente receber dados de treinamento para reduzir a tendência ao erro.

Para que este processo seja realizado com êxito, é necessário dados de testes realistas, tanto sintéticos, para que seja usado em correções e testar a sensibilidade ao ruído, alterando imagens do mundo real, no qual também são utilizadas (SZELISKI, 2010). Se *machine learning* estiver sendo usado para este ato, torna-se ainda mais importante possuir dados de treinamento imparciais e representativos, uma quantidade geralmente alta para obter bom resultados e haver maior distinção de dados.

Para que um sistema de Visão Computacional consiga "descrever o que pode ser visualizado", ele deve ser treinado para executar diversas tarefas. A mais fundamental e primária dessas tarefas é a **Classificação de Imagens**, que busca responder à pergunta mais básica: "Qual é o objeto ou cena principal nesta imagem?". Esta é a base para tarefas mais complexas.

2.6.1 Classificação de Imagens

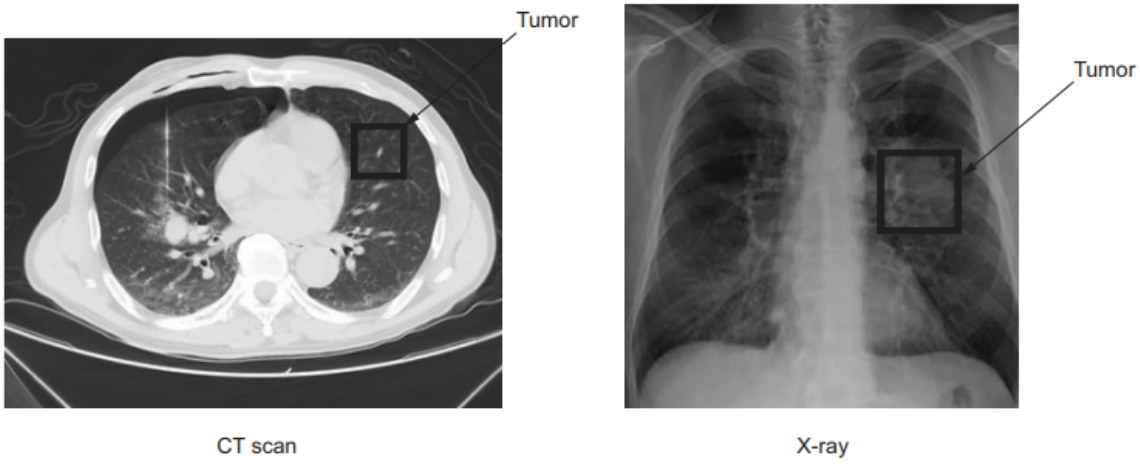
Um dos principais cenários de uso da **Classificação de Imagens** é identificar o objeto principal em uma imagem. Nesses casos, assume-se que a imagem é dominada por um único item, como em um classificador de frutas. O modelo recebe uma imagem (ex: de uma maçã inteira) e retorna um rótulo único ('maçã') que classifica a imagem inteira, sem precisar lidar com outros objetos ou fatores complexos na cena (RUSSELL; NORVIG, 2010).

O segundo caso é de uma categoria de imagens que possui diversos objetos, fazendo com que o classificador deva identificar diversos cenários, seres vivos e outros grupos (RUSSELL; NORVIG, 2010). Este classificador agora irá tratar imagens em larga escala, no qual possui a capacidade de produzir e classificar imagens com precisão, relativo ao seu contexto em geral, como uma paisagem na natureza com um cachorro no fundo, ou um escritório com uma mesa de madeira, fazendo a distinção do que é coerente estar em ambientes conexos com seu contexto.

Um exemplo de processamento e classificação de imagens é no diagnóstico de câncer, mais especificamente no pulmão, como pode ser observado na **Figura 2.9**. A imagem lado es-

querdo mostra uma tomografia computadorizada onde há a identificação de um pequeno nódulo de câncer. E da mesma forma, na imagem da direita, porém com uma imagem de Raios X.

Figura 2.9: sistema de visão com capacidade de identificar pequenos nódulos em raios X.



Fonte: Elgendy (2020, p. 11).

Os nódulos de câncer costumam iniciar com pequenos nódulos quase imperceptíveis à olho nu, no qual as imagens de exames são entregues a profissionais capacitados, no qual facilmente conseguem detectar pequenos tumores de 6 a 10 milímetros. Porém, quando a imagem contém um nódulo extremamente pequeno, como 4 milímetros, esta tarefa se torna complexa. Com o uso das redes de *deep learning*, especificamente as **Redes Neurais Convolucionais**, agora é possível que estes agentes aprendam características automaticamente a partir da entrada de uma larga escala de dados de diferentes da mesmas classes, fazendo assim com que eles aprendam e façam análises minuciosas de diversas imagens (ELGENDY, 2020).

Uma rede neural convolucional é um tipo de rede neural artificial, destacando-se no processamento e classificação de imagens de diferentes categorias e grupos (ELGENDY, 2020).

2.6.2 Redes Convolucionais

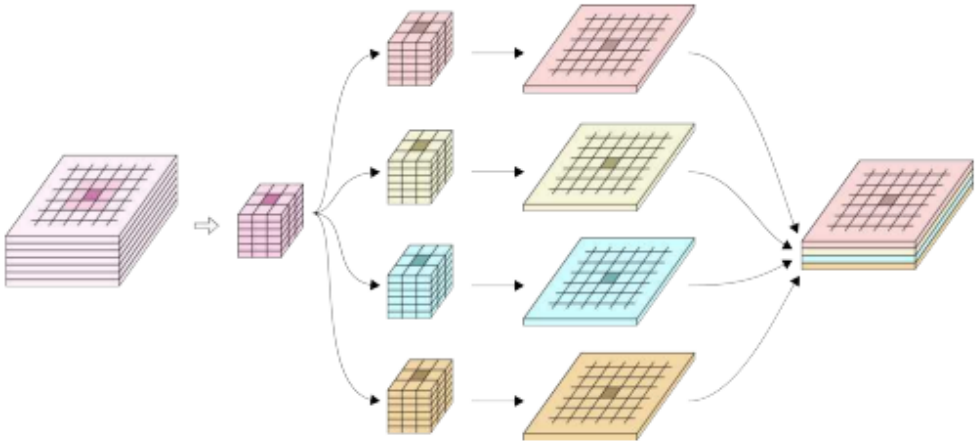
As **Redes Neurais Convolucionais** (RNC) são um tipo de rede neural artificial projetada especificamente para processar dados que possuem estrutura de grade, como imagens, onde as RNCs se destacam espetacularmente bem (RUSSELL; NORVIG, 2010). Com dados de treinamento e engenhosidade suficientes, as RNCs possuem a capacidade de classificação melhores do que qualquer outro método já produzido.

Este tipo de rede neural artificial possui conexões espacialmente locais, ao menos em camadas iniciais, tendo padrões de pesos que se repetem em nós de cada camada desta rede. Este padrão de pesos possui o nome de *kernel*, onde seu processo de aplicação aos *pixels* de uma imagem ou em unidades organizadas espacialmente em uma camada seguinte é chamada de *convolução* (RUSSELL; NORVIG, 2010).

Porém, diferentemente da convolução em imagens, no qual o mesmo filtro é aplicado a cada canal de cor, as redes neurais convolucionais geralmente aplicam a combinação linear das ativações de cada canal C_1 de entrada de uma camada anterior, utilizando núcleos diferentes de convolução para cada um dos canais C_2 de saída (SZELISKI, 2010). Um exemplo disso pode ser observado nas **Figuras 2.10 e 2.11**. A principal atividade das RNCs é construir características locais, e após isso, combinar estas características de várias formas a fim de produzir aspectos mais discriminantes e semanticamente significativos.

Na **Figura 2.10** pode-se observar uma convolução 2D com diversos canais de entrada e saída, onde cada núcleo de convolução recebe de entrada os canais C_1 da camada anterior a ele, produzindo os valores após a não-linearidade da função de ativação em um dos canais C_2 da camada subsequente. Para cada canal de saída, temos S^2xC_1 pesos do núcleo, possuindo então o número total de parâmetros ajustáveis em cada camada de convolução sendo $S^2C_1C_2$ (SZELISKI, 2010). No exemplo desta **Figura 2.10**, há $C_1 = 6$ canais de entrada e $C_2 = 4$ canais de saída, sendo a janela de convolução $S = 3$, no total, $9 * 6 * 4$ pesos que são aprendíveis.

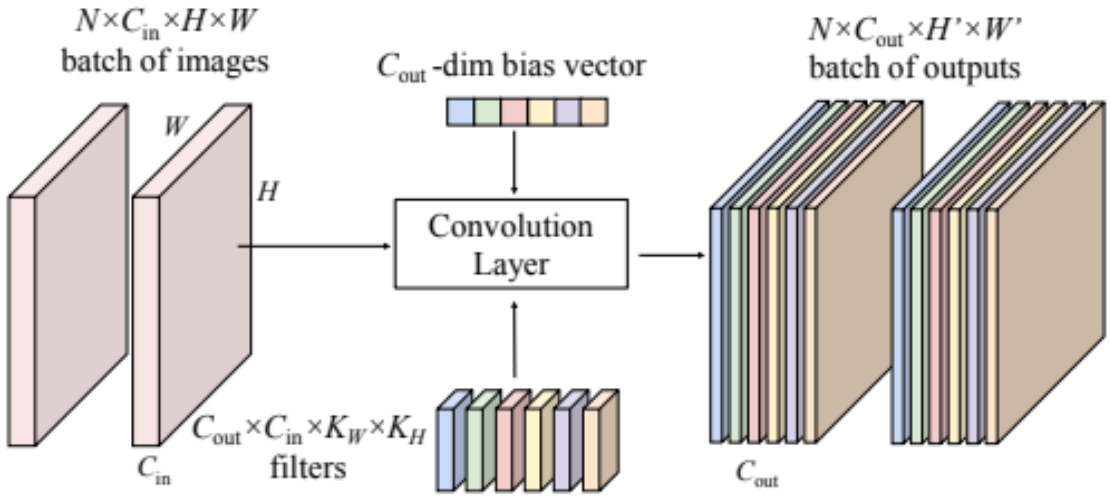
Figura 2.10: convolução 2D com múltiplos canais de entrada e saída.



Fonte: Szeliski (2010, p. 292).

Já na **Figura 2.11** é mostrada uma convolução 2D com múltiplos lotes (*batches*) de imagens, nos canais de entrada e saída. Utilizando a descida do gradiente em *mini-batches*, um lote inteiro de imagens para treinamento é carregado em uma camada convolucional, recebendo de entrada todos os C_{in} canais da camada antecedente, produzindo os valores após a não-linearidade da função de ativação em algum dos canais C_{out} da próxima camada (SZELISKI, 2010). Para cada canal de saída observa-se $K_wK_hC_{in}$ pesos do núcleo, com o total de parâmetros aprendíveis em cada camada convolucional sendo $K_wK_hC_{in}C_{out}$. Nesta **Figura 2.11**, temos $C_{in} = 3$ canais de entrada e $C_{out} = 6$ canais de saída.

Figura 2.11: convolução 2D com múltiplos lotes (batches), canais de entrada e saída.



Fonte: Szeliski (2010, p. 293).

2.6.2.1 *Pooling*

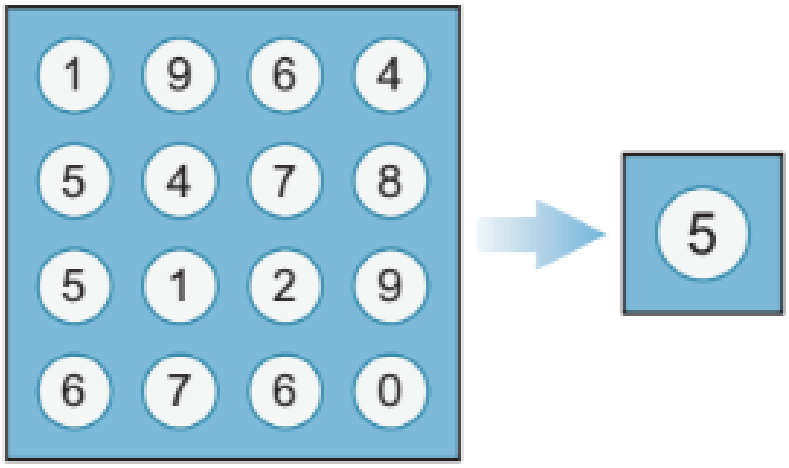
A ***Camada de Pooling*** de uma rede neural artificial tem a funcionalidade de simplificar a informação da camada anterior em um único valor. O *pooling* atua como uma camada de convolução, tendo um *kernel* de tamanho l e passada s, possuindo uma operação fixa aplicada na camada. Geralmente nenhuma função de ativação está associada à camada de *pooling* (RUSSELL; NORVIG, 2010). Existem duas formas comuns de *pooling*:

- ***Average-Pooling***: este, calcula o valor médio de suas l entradas, da mesma forma que a convolução com um vetor de *kernel* uniforme $k = [1/l, ..., 1/l]$. Tomando $l = s$, a ideia é reduzir a resolução da imagem de entrada, sendo chamado de *downsampling*, por um fator de s. Um objeto que possuía tamanho em *pixels* de, por exemplo, *10s pixels*, após o *average-pooling*, ocuparia apenas *10 pixels*. O mesmo classificador que foi capaz de aprender e identificar um objeto na resolução de *10 pixels*, sendo esta a imagem original, agora é capaz de identificar este mesmo objeto na imagem com *downsampling*, ainda que ele fosse muito grande para ser reconhecido na imagem original (RUSSELL; NORVIG, 2010). Resumindo, temos que o *average-pooling* facilita o reconhecimento multiescalar (em vários níveis). Ele possui a capacidade também de reduzir a quantidade de pesos que seriam necessários nas próximas camadas, tornando, possivelmente, o aprendizado mais veloz. A **Figura 2.12** mostra um exemplo de *average-pooling*;

- ***Max-Pooling***: este, realiza o cálculo do valor máximo de suas l entradas, podendo também ser utilizado exclusivamente para *downsampling*, porém, possuindo uma semântica diferente. Basicamente o *pooling* máximo realiza uma espécie de disjunção lógica, informando caso um fator exista em algum lugar do espaço sensorial

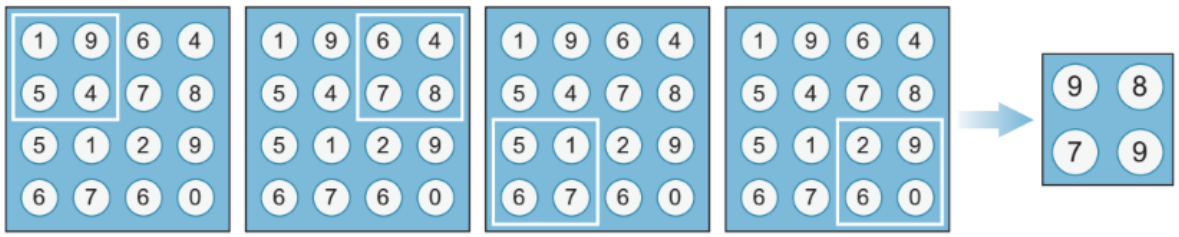
da unidade (RUSSELL; NORVIG, 2010). A **Figura 2.13** mostra um exemplo de *max-pooling*, na qual pode se observar um filtro de *pooling* com o tamanho de 2 X 2 e *strides* de 2 (o filtro "pula"2 *pixels* quando desliza sobre a imagem).

Figura 2.12: *average-pooling* calculando o valor médio de todos os *pixels* em um *feature map*.



Fonte: Elgendy (2020, p. 116).

Figura 2.13: um filtro de *pooling* 2 X 2 e *strides* de 2, reduzindo o *feature map* de 4 X 4 para 2 X 2.



Fonte: Elgendy (2020, p. 115).

As camadas convolucionais, de ativação e de *pooling*, descritas até aqui, são os blocos de construção fundamentais das Redes Neurais Convolucionais (RNCs). É a combinação estratégica dessas camadas que permite aos modelos aprender a extrair características e identificar padrões visuais complexos, exatamente como os necessários para diferenciar as lesões de doenças em folhas.

Tendo estabelecido a base teórica dessas ferramentas, a próxima seção analisará como outros pesquisadores têm aplicado essas arquiteturas em trabalhos práticos no contexto da agricultura.

2.7 Trabalhos Relacionados

Nesta seção será discutido trabalhos semelhantes no qual também fazem o uso de técnicas de Inteligência Artificial aplicadas no contexto da agricultura.

O trabalho realizado por Salvadori (2024) faz o uso de uma base de dados para treinar uma RNC com a finalidade de detectar pragas em folhas de plantas. Foi utilizado o modelo pré-treinado já consolidado, a *ResNet-18*, possuindo 18 camadas de profundidade, onde 17 destas camadas são convolucionais e uma totalmente conectada, que foi utilizada para detectar e classificar anomalias nas folhas da cultura da soja.

Neste trabalho foi selecionado um *dataset* que possui três categorias, sendo elas folhas saudáveis, folhas danificadas por lagartas e folhas danificadas por *Diabrotica speciosa* (SALVADORI, 2024). Foi também realizado um aumento de dados em cada classe de imagens para maior diversidade de treinamento do modelo.

Um artigo publicado por Masood et al. (2023) faz a apresentação de uma RNC treinada especificamente para realizar a detecção e a a classificação de doenças em folhas de milho, sendo esta a *MaizeNet*, no qual apresentou uma acurácia maior comparada a outros modelos já existentes no mercado.

O *MaizeNet* utiliza um método de RNCs melhorada de outra abordagem, sendo esta a *Faster-RCNN*, utilizando o modelo *ResNet-50* que foi treinado com *datasets* de folhas de milho no qual se apresentaram saudáveis ou com enfermidades (MASOOD et al., 2023). O conjunto de dados possui diversas imagens com ruído, embaçamento, diferentes cores ou iluminação para entregar um resultado mais próximo do real e mais suscetível a erros. O *framework MaizeNet* faz o uso de quatro passos, sendo eles: (1) extrator de características, (2) RPN (*Region Proposal Networks* - Rede de Propostas Regionais), (3) ROI *Pooling* (*Region of Interest Pooling* - Agrupamento de Regiões de Interesse) e (4) categorização para identificar e classificar várias doenças foliares no milho.

Outro artigo relevante foi desenvolvido por Ghazal, Munir e Qureshi (2024) com o intuito de realizar estudos sobre o uso de visão computacional em diferentes ambientes da agricultura de precisão, abordando diversos estágios da agricultura digital (chamada também de agricultura 4.0), como aquisição de imagem, costura de imagens, análise de imagens, entre outros.

O artigo apresentou uma exploração extensa do ciclo de vida digital de plantações com agricultura de precisão, revisando em um só artigo diversas técnicas que contribuem para o desenvolvimento de sistemas automatizados guiados por visão, apresentando também fatores que demonstram que o uso de sistemas inteligentes utilizando visão computacional ainda está em uma fase inicial, devendo ser ainda explorada e discutida devido a diversos fatores que podem levar a novas pesquisas na área, como fatores climáticos imprevisíveis, condições da terra e outras influências que afetam as plantações e abrem portas para o uso da automação na agricultura. (GHAZAL; MUNIR; QURESHI, 2024)

Um artigo publicado por Sujatha et al. (2025) propõe o uso de IA para a automação da detecção de doenças em diversas plantas, de forma que supere a dificuldade de realizar tal ato em métodos convencionais, propondo um modelo de *deep learning* para realizar a extração de características das folhas de plantas e *machine learning* para realizar o processamento.

Para isso, foram utilizados modelos de redes neurais convolucionais, como o *VGG19* e *Inception v3*, em conjunto com quatro bases de dados, sendo elas de folhas de bananeira, fruta-do-conde, figo e batata (SUJATHA et al., 2025). A integração de DL com ML mostrou-se ser uma abordagem poderosa, identificando os desafios envolvendo as variações morfológicas, diversidade de doenças e influências do ambiente. Por fim, a pesquisa realiza uma comparação com outras metodologias que não utilizaram dessa mesclagem, onde os modelos propostos se mostraram mais eficientes em comparação aos demais.

Gardezi et al. (2024) em seu artigo publicado em 2024, realizou uma pesquisa com a finalidade de endereçar problemas e oportunidades no uso de inteligência artificial na agricultura de precisão. Gardezi propõe quatro recomendações de melhorias nos sistemas que são aplicados à esta área, sendo elas: primeiro, a melhoria na descrição e transparência dos modelos, segundo, uma responsabilidade clara a ser implementada nas decisões da IA, terceiro, conscientização sobre a necessidade do uso de IAs para que haja uma melhor parceria de máquina e humano na agricultura, e por último, regulamentação dos direitos dos dados, privacidade e segurança.

O uso de IA na agricultura de precisão ajuda fazendeiros a terem práticas e resultados precisos a depender do clima e diversas variações do campo (GARDEZI et al., 2024). Com seus estudos, Gardezi chegou a conclusão de que a inteligência artificial possui a capacidade de separar ruídos de sinais verdadeiros e fazer recomendações de onde, quando, e quanto plantar, pulverizar agrotóxicos, realizar adubagem ou colher.

Por fim, um artigo publicado em 2020, produzido por Pham et al. (2020) desenvolve e compara diversos modelos de IAs, nomeados PSOA-NFIS (*Adaptive Network based Fuzzy Inference System optimized with Particle Swarm Optimization* - Sistema de Inferência *Fuzzy* baseado em Rede Adaptativa otimizado com Otimização por Enxame de Partículas), Redes Neurais Artificiais e SVMs (*Support Vector Machines* - Máquina de Vetores de Suporte) na previsão da precipitação diária em uma província do Vietnã.

Para isso, foram utilizadas variáveis meteorológicas de parâmetro, como temperatura máxima, temperatura mínima, velocidade do vento, umidade relativa e radiação solar para serem utilizadas como entrada no modelo, obtendo como resultado a precipitação diária conforme as variáveis selecionadas (PHAM et al., 2020). Durante o treinamento e testes dos modelos, todos estes apresentaram previsões razoáveis de precipitação, porém, o modelo SVM mostrou-se o melhor método para esta categoria.

CAPÍTULO 3

Metodologia

Inicialmente, faz-se necessário contextualizar o desenvolvimento e a condução da pesquisa, apresentando de forma geral os recursos empregados e as etapas executadas ao longo do trabalho. Este capítulo aborda as bases de dados utilizadas, os procedimentos de preparação das imagens, bem como os modelos de Redes Neurais Convolucionais adotados para a realização da tarefa de classificação.

3.1 Aquisição e Preparação dos Dados

O treinamento dos modelos de redes neurais convolucionais foi conduzido utilizando dois *datasets* obtidos de fontes públicas na internet. Antes da etapa de treinamento, os dados foram submetidos a um processo de pré-processamento e técnicas de *oversampling*, com o objetivo de mitigar o desbalanceamento entre as classes e, consequentemente, aprimorar o desempenho dos modelos em termos de acurácia.

3.1.1 Fonte dos Dados

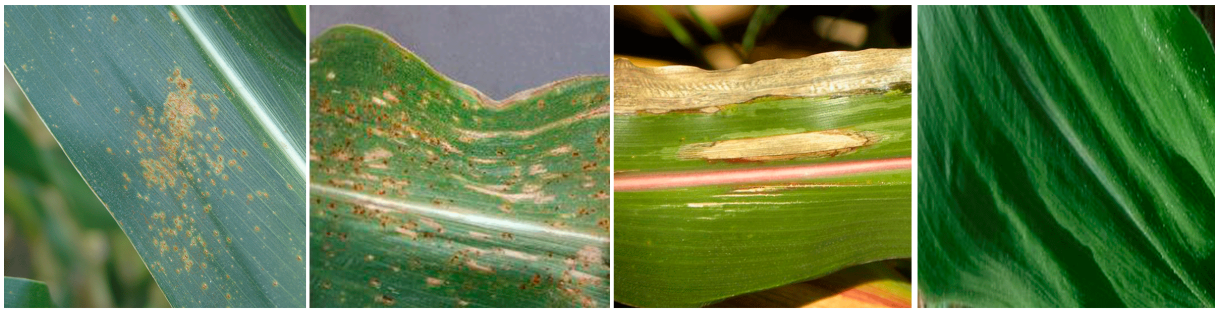
Neste trabalho, foram utilizadas duas bases de dados compostas por imagens de folhas de milho acometidas por diferentes doenças foliares. Com relação a primeira base de dados, as imagens desta foram extraídas a partir da junção de dois conjuntos de dados: **PlantDoc** (SINGH et al., 2020) + **PlantVillage** (HUGHES; SALATHÉ , 2015). Ambas as bases contêm imagens de diversas espécies vegetais, no entanto, para os fins desta pesquisa, foram selecionadas apenas as classes correspondentes à cultura do milho (*Zea mays*[1]).

A seleção resultou em um total de 4.188 imagens distribuídas entre quatro classes: 1.306 imagens de folhas infectadas com ferrugem comum (***Common Rust***), 574 com cercosporiose (***Gray Leaf Spot***), 1.146 com helmintosporiose (***Blight***), e 1.162 imagens de folhas saudáveis

[1] Nome científico do milho.

(***Healthy***). Considerando o evidente desbalanceamento entre as classes, em especial a classe *Gray Leaf Spot*, foi aplicada posteriormente uma técnica de *oversampling*, cuja metodologia será detalhada nas seções subsequentes, com o objetivo de mitigar impactos negativos no desempenho dos modelos durante o treinamento. Na **Figura 3.1** pode-se observar amostras de cada classe dentro da base.

Figura 3.1: (a) *Common Rust* (b) *Gray Leaf Spot* (c) *Blight* (d) *Healthy*.



Fonte: Singh et al. (2020) e Hughes e Salathé (2015)

A segunda base de dados utilizada neste trabalho corresponde ao conjunto disponibilizado por Ahmad et al. (2021) sendo ela a **CD&S**, composto por um total de 1.597 imagens de folhas de milho distribuídas em três classes patológicas: 511 imagens correspondentes à helmintosporiose (***Northern Leaf Blight***), 524 à cercosporiose (***Gray Leaf Spot***) e 562 à mancha de *Bipolaris* (***Northern Leaf Spot***).

Considerando a semelhança visual entre as classes e a possível sensibilidade dos modelos à variação intra e interclasse, foi realizada a aplicação de uma técnica de *oversampling* em todas as classes, com o intuito de aumentar o número de amostras e promover um treinamento mais robusto e balanceado. O procedimento adotado para esse tratamento será detalhado em seção específica posteriormente neste capítulo. Na **Figura 3.2** pode-se observar uma amostra das imagens de cada classe da base de dados.

Figura 3.2: (a) *Northern Leaf Blight* (b) *Gray Leaf Spot* (c) *Northern Leaf Spot*.

(a) (b) (c)

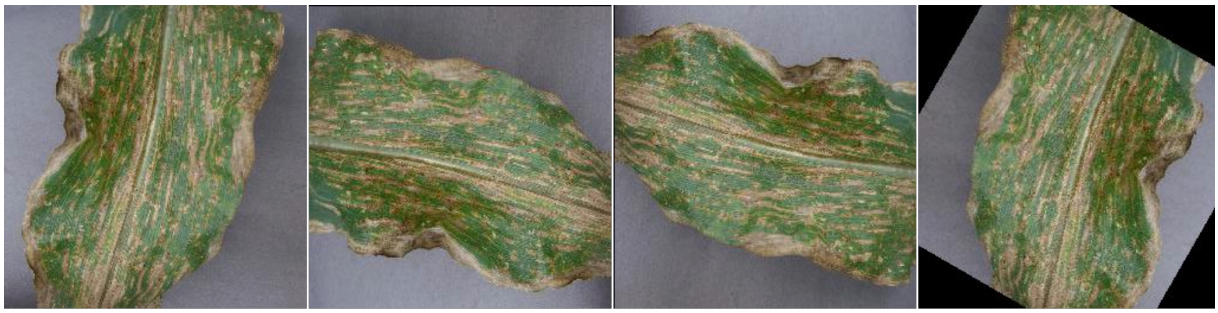
Fonte: Ahmad et al. (2021)

3.1.2 *Oversampling*

Durante a análise inicial das bases de dados, observou-se, na junção dos conjuntos **PlantVillage + PlantDoc**, uma desproporção significativa no número de amostras da classe *Gray Leaf Spot*. Com o intuito de mitigar tal desbalanceamento, foi aplicada uma técnica de *oversampling* direcionada especificamente a essa classe.

O procedimento consistiu na ampliação do número de amostras a partir das 574 imagens originais, que foram submetidas a transformações geométricas, como rotações aleatórias, resultando em múltiplas variações artificiais de cada imagem. Cada amostra original foi transformada, de forma aleatória, em outras três ou quatro imagens adicionais, como pode ser observado na **Figura 3.3**. Como resultado, o total de imagens da classe *Gray Leaf Spot* foi expandido para **1.642** amostras, compondo um novo conjunto de dados com **5.256** imagens no total.

Figura 3.3: rotações realizadas em uma imagem (a) do *dataset* original.



(a)

Fonte: elaborada pelo autor

Já na base de dados **CD&S**, constatou-se uma quantidade relativamente reduzida de imagens nas três classes consideradas, o que poderia impactar negativamente o desempenho

dos modelos de classificação, especialmente no que se refere à acurácia. No entanto, ao explorar a fonte original da base, identificou-se a disponibilidade de um subconjunto adicional de imagens para cada classe, nas quais as folhas de milho foram segmentadas e apresentadas sobre um fundo uniformemente preto, como pode ser observado na **Figura 3.4**, facilitando a análise visual e o aprendizado dos modelos. Optou-se por utilizar este subconjunto com fundo preto em vez de replicar a técnica de *oversample* usada no **PlantDoc + PlantVillage**. Essa decisão baseou-se em uma vantagem estratégica: o fundo preto já isola a folha de ruídos (**Figura 3.4**), permitindo avaliar o modelo em dados idealmente segmentados e, ao mesmo tempo, diversificando as abordagens de tratamento de dados entre os dois conjuntos.

Figura 3.4: amostras do subconjunto adicional do CD&S com o fundo uniformemente preto.

Fonte: Ahmad et al. (2021)

Essas imagens suplementares contribuíram com mais 228 amostras para a classe helmintosporiose (*Northern Leaf Blight*), totalizando **739** imagens; 262 para a classe cercosporiose (*Gray Leaf Spot*), totalizando **786** imagens; e 265 para a classe mancha de *Bipolaris* (*Northern Leaf Spot*), totalizando **827** imagens. Com isso, o conjunto final do **CD&S** utilizado para o treinamento dos modelos totalizou **2.352** imagens, distribuídas de forma mais equilibrada entre as classes.

3.1.3 Divisão dos Dados

Para a realização do treinamento e validação dos modelos, os dados foram divididos de forma estratificada, mantendo a proporção entre as classes em ambas as etapas. Em ambos os *datasets*, adotou-se a proporção de **70%** das imagens para o conjunto de treinamento e **30%** para o conjunto de validação, assegurando uma distribuição representativa das classes em cada subconjunto.

A fim de garantir que a divisão dos dados ocorresse de maneira aleatória, porém reprodutível, foi aplicado o embaralhamento (*shuffling*) dos dados antes da separação, utilizando uma semente de aleatoriedade fixa com valor igual a **123**. Essa abordagem assegura que os

resultados obtidos sejam consistentes em diferentes execuções do processo, além de evitar que possíveis padrões na ordenação original das imagens influenciem negativamente no aprendizado dos modelos (DUTTA, 2024).

É possível observar estas definições no **Código-Fonte 3.1**.

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.3,
    subset='training',
    seed=123,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='int'
)
val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.3,
    subset='validation',
    seed=123,
    image_size=img_size,
    batch_size=batch_size,
    label_mode='int'
)
```

Código-Fonte 3.1: funções para definir os *datasets* de treino e validação.

3.1.3.1 Pré-Processamento das Imagens

Inicialmente, todas as imagens dos conjuntos de treinamento e validação passaram por uma função de pré-processamento específica para a arquitetura de cada modelo, disponibilizada, respectivamente, da seguinte forma: `applications.mobilenet_v3.preprocess_input`, `applications.resnet.preprocess_input` e `applications.efficientnet_v2.preprocess_input`. Estas funções ajustam o escalonamento dos *pixels* das imagens, convertendo-os do intervalo original [0, 255] para os formatos exatos que cada modelo espera, como o escalonamento para [0, 1] (utilizado pelo **EfficientNetV2**) ou para [-1, 1] (utilizado pelo **MobileNetV3** e **ResNet101**), uma condição essencial para o sucesso da aprendizagem. Simultaneamente, os rótulos de cada imagem foram transformados para o formato ***One-Hot Encoded***, sendo este o processo de transformar um único rótulo categórico em um vetor binário (composto apenas por *0s* e *1s*), onde apenas um único bit é "quente"(com o valor 1), enquanto todos os outros são "frios"(com o valor 0) (SAMUELS, 2024). Este *encoding* é requisito da função de perda *Categorical Cross-Entropy*, o qual será citada posteriormente.

Em seguida, uma técnica de *data augmentation* chamada ***MixUp*** foi aplicada exclusivamente ao conjunto de treinamento. Segundo Liu (2024), o *MixUp* gera novos exemplos sintéticos ao criar combinações lineares convexas entre pares de imagens e seus respectivos ró-

tulos. Essa estratégia funciona como um regularizador, incentivando o modelo a desenvolver uma fronteira de decisão mais suave e melhorando sua capacidade de generalização.

Por fim, a pipeline foi otimizada para máxima eficiência durante o treinamento. O método *.cache()* foi utilizado para armazenar os dados em memória após o primeiro carregamento, evitando gargalos de *I/O* nas épocas subsequentes. Para o conjunto de treinamento, aplicou-se o método *.shuffle()* para embaralhar os dados a cada época, garantindo a aleatoriedade dos lotes. Ambos os conjuntos de dados (treinamento e validação) utilizaram o método *.prefetch()*, que permite que a CPU prepare o próximo lote de dados enquanto a GPU está processando o lote atual. A configuração *tf.data.AUTOTUNE* foi empregada em todas as etapas para alocar recursos computacionais de forma dinâmica e otimizada.

Como exemplo, o **Código-Fonte 3.2** mostra a implementação do pré-processamento realizado no modelo *ResNet-101*. Os demais modelos seguiram o mesmo padrão.

```
def one_hot(image, label, num_classes):
    return image, tf.one_hot(label, num_classes)

def mixup(image, label, alpha=MIXUP_ALPHA):
    batch_size = tf.shape(image)[0]
    idx = tf.random.shuffle(tf.range(batch_size))
    lam = tf.random.uniform(shape=[], minval=0.0, maxval=alpha)
    mixed_image = (1 - lam) * image + lam * tf.gather(image, idx)
    mixed_label = (1 - lam) * label + lam * tf.gather(label, idx)
    return mixed_image, mixed_label

def preprocess(image, label):
    image = applications.resnet.preprocess_input(image)
    return one_hot(image, label, num_classes)

    train_ds = train_ds.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
    val_ds = val_ds.map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
    train_ds = train_ds.map(mixup, num_parallel_calls=tf.data.AUTOTUNE)
    train_ds = train_ds.cache().shuffle(1024).prefetch(tf.data.AUTOTUNE)
    val_ds = val_ds.cache().prefetch(tf.data.AUTOTUNE)
```

Código-Fonte 3.2: pré-processamento das imagens.

3.2 Aumento de Dados

Com o intuito de aumentar a diversidade de amostras apresentadas aos modelos durante o treinamento e, consequentemente, mitigar o risco de *overfitting*, foi implementada uma estratégia de ***Data Augmentation*** aplicada de forma uniforme a todos os modelos e para ambas as bases de dados. Essa técnica visa gerar variações artificiais das imagens originais, permitindo que a rede aprenda padrões mais robustos e generalizáveis, reduzindo a probabilidade de memorizar características específicas de amostras individuais.

O *pipeline* de aumento de dados adotado neste trabalho compreendeu as seguintes operações:

- ***Random Flip*** (espelhamento) nas direções horizontal e vertical;
- **Rotação aleatória** com fator 0.1, utilizando preenchimento de cor nos *pixels* extrapolados da imagem;
- **Zoom aleatório** com fator 0.3, também com preenchimento de cor nos *pixels* extrapolados;
- **Ajuste aleatório de contraste** com variação de 30%.

A função responsável pela aplicação desse *pipeline* encontra-se no **Código-Fonte 3.3**. É importante ressaltar que essa etapa foi aplicada exclusivamente ao conjunto de treinamento, sendo desnecessária sua utilização no conjunto de validação, de modo a preservar a avaliação imparcial do desempenho dos modelos.

```
def get_data_augmentation():
    return models.Sequential([
        layers.RandomFlip('horizontal_and_vertical'),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.3),
        layers.RandomContrast(0.3),
    ])
```

Código-Fonte 3.3: função para definir o aumento de dados do *dataset*.

3.3 Modelagem

Nesta seção são apresentadas as arquiteturas de RNC utilizadas no desenvolvimento deste trabalho, detalhando-se as configurações adotadas para cada modelo, bem como as estratégias de treinamento empregadas. Serão descritos os parâmetros de inicialização, as técnicas de *fine-tuning* e regularização, além das escolhas relacionadas à função de perda, otimizadores e métricas de avaliação.

3.3.1 Modelos Convolucionais Utilizados

O treinamento foi conduzido com três modelos distintos de redes neurais convolucionais, com o objetivo de identificar aquele que apresentasse o melhor desempenho em termos de acurácia para ambas as bases de dados utilizadas.

3.3.1.1 *MobileNetV3-Large*

Considerando a elevada semelhança visual entre as imagens das classes de doenças foliares, muitas vezes com diferenças sutis e de difícil percepção a olho nu, optou-se pela seleção da

arquitetura ***MobileNetV3-Large*** no treinamento dos modelos. Essa escolha justifica-se pelo fato de a *MobileNetV3* apresentar uma estrutura convolucional profunda e, ao mesmo tempo, otimizada para eficiência computacional, o que a torna particularmente adequada tanto para ambientes de alto desempenho quanto para potenciais aplicações em dispositivos móveis, alinhando-se a possíveis cenários futuros de uso desta pesquisa.

A *MobileNetV3*, proposta por Elaziz et al. (2023), resulta da combinação de avanços introduzidos em arquiteturas anteriores, como a *MobileNetV2* e a *EfficientNet*. A versão *Large* possui aproximadamente 118 camadas convolucionais, totalizando cerca de 5,4 milhões de parâmetros treináveis, e recebe como entrada imagens no formato **224x224x3**.

Entre suas principais inovações, destaca-se a utilização do bloco SE (*Squeeze-and-Excitation*), que implementa um mecanismo de atenção de canais capaz de identificar e reforçar os mapas de características mais relevantes em diferentes estágios da rede. Outra contribuição importante é a adoção da função de ativação *Hard-Swish*, uma aproximação eficiente da função *Swish*, porém com menor custo computacional, favorecendo sua execução em ambientes com restrição de recursos. Além disso, a arquitetura faz uso de convoluções com profundidade separável (*Depthwise Separable Convolutions*), reduzindo a complexidade do modelo sem comprometer significativamente a capacidade de extração de características.

A arquitetura *MobileNetV3*, cujo diagrama é apresentado na **Figura 3.5**, apresenta seu bloco de construção fundamental, que constitui o *backbone*[2] da rede, é a **Convolução Separável em Profundidade** (*Depthwise Separable Convolution*). Esta técnica otimiza a convolução padrão ao fatorizá-la em duas operações sequenciais:

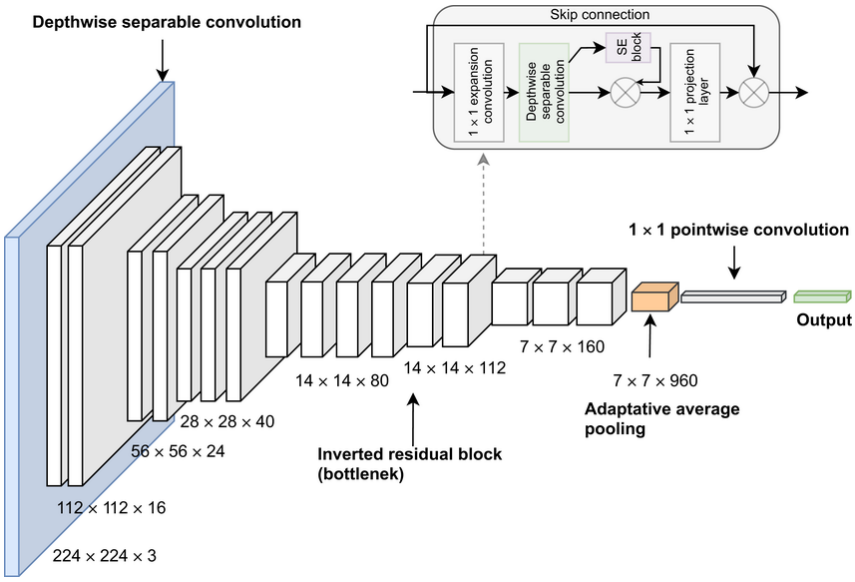
1. **Convolução em Profundidade (*Depthwise Convolution*):** onde um único filtro convolucional é aplicado a cada canal de entrada de forma independente, realizando a filtragem espacial;

2. **Convolução Ponto a Ponto (*Pointwise Convolution*):** uma convolução de dimensão *1x1* que projeta e combina as saídas da etapa anterior, realizando a filtragem de canais.

Esses blocos de construção são organizados em estruturas conhecidas como **Blocos de Resíduo Invertido** (*Inverted Residual Blocks*), que funcionam como *bottlenecks* (gargalos) para reduzir a complexidade e o número de parâmetros, garantindo a alta eficiência do modelo sem comprometer significativamente a acurácia.

[2] O *backbone* é responsável por extrair e codificar representações relevantes dos dados de entrada (SHROFF, 2023). Ele atua como extrator principal de características, capturando informações de baixo e alto nível a partir das imagens.

Figura 3.5: diagrama da arquitetura do modelo *MobileNetV3*.



Fonte: Elaziz et al. (2023)

3.3.1.2 *ResNet101*

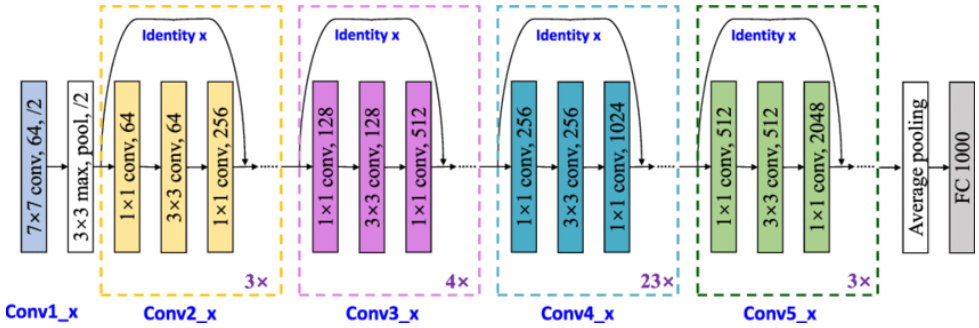
Em contraste com a *MobileNetV3*, a ***ResNet-101***, descrita por He et al. (2015), é uma arquitetura profunda e de propósito geral, projetada para alcançar alta acurácia em tarefas de classificação de imagens em larga escala. O modelo possui 101 camadas treináveis, aproximadamente 44,5 milhões de parâmetros e recebe como entrada imagens no formato $\mathbf{224}\times\mathbf{224}\times\mathbf{3}$.

O principal diferencial da família *ResNet* reside no uso de conexões residuais (*Skip-Connections*), que somam a ativação de uma camada às saídas de camadas posteriores, facilitando o fluxo de gradientes durante o treinamento e mitigando o problema do *vanishing gradient*. O *vanishing gradient* ocorre quando os gradientes usados para atualizar os pesos tornam-se muito pequenos durante a retropropagação, especialmente nas camadas iniciais (HE et al., 2015).

Para redes profundas, a *ResNet-101* adota blocos do tipo *bottleneck*, cuja estrutura típica é 1×1 (redução de dimensões) $\rightarrow 3 \times 3$ (processamento espacial) $\rightarrow 1 \times 1$ (expansão de dimensões). Essa organização reduz o custo computacional preservando a capacidade de extração de características. A arquitetura também emprega normalização em lotes (*Batch Normalization*) e não linearidades do tipo *ReLU* após as convoluções, além de *shortcuts* de identidade ou projeção (quando há mudança de dimensão).

A **Figura 3.6** resume a arquitetura em forma de diagrama do *ResNet-101*.

Figura 3.6: diagrama da arquitetura do modelo *ResNet101*.



Fonte: Chen et al. (2021)

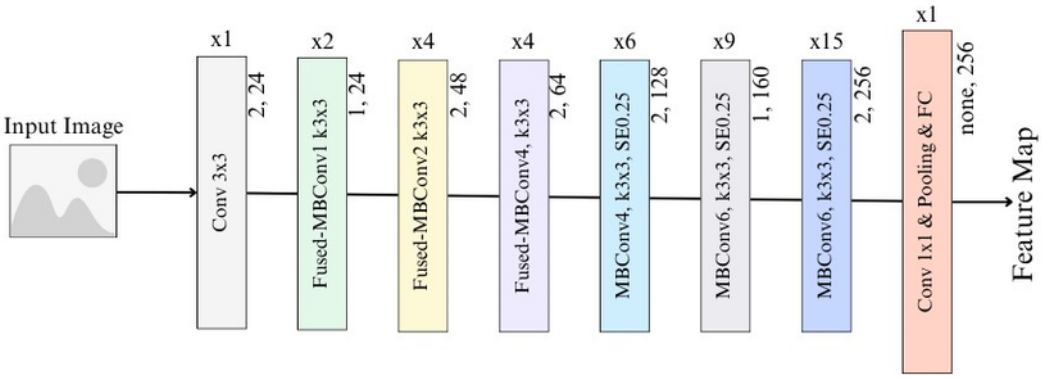
3.3.1.3 *EfficientNetV2-M*

O último modelo empregado pertence à família ***EfficientNetV2***, tendo sido selecionada a variante de porte médio (***M***) por equilibrar acurácia e eficiência computacional, conforme apresentado por Tan e Le (2021). Essa arquitetura, mais recente que as demais avaliadas, foi concebida para acelerar o treinamento em altas resoluções e escalonar de forma eficaz para *datasets* mais complexos. Na configuração canônica, a *EfficientNetV2-M* utiliza imagens de entrada no formato $\mathbf{480}\times\mathbf{480}\times\mathbf{3}$ e possui da ordem de dezenas de milhões de parâmetros (cerca de 54 milhões), com profundidade efetiva elevada decorrente da repetição de blocos ao longo dos estágios.

A *EfficientNet* original (2019) introduziu o *Compound Scaling*, que realiza a expansão coordenada de profundidade, largura e resolução. A *EfficientNetV2*, por sua vez, aprimora esse desenho ao incorporar: Blocos ***Fused-MBConv*** nas camadas iniciais, substituindo a decomposição *Depthwise* + *Pointwise* por uma sequência *conv* 3×3 seguida de *conv* 1×1, sendo uma solução mais eficiente em GPUs modernas; versões otimizadas dos blocos ***MBConv*** com SE nas camadas mais profundas; e estratégias de treinamento progressivo e *Regularization* ajustadas ao regime de grande escala, reduzindo custo e tempo de convergência sem sacrificar desempenho.

A **Figura 3.7** resume em forma de diagrama a arquitetura em estágios da *EfficientNetV2*.

Figura 3.7: diagrama da arquitetura do modelo *EfficientNetV2*.



Fonte: Aldakhil e Almutairi (2024)

3.3.2 Configuração dos Modelos

Os três modelos treinados partem de uma configuração-base idêntica, incluindo tamanho de entrada, estratégia de *fine-tuning* e outros hiperparâmetros. Portanto, as subseções a seguir descrevem essa configuração comum, sendo válidas para todos os modelos.

3.3.2.1 Tamanho de Entrada

Cada arquitetura de rede neural convolucional possui dimensões de entrada específicas recomendadas para seu melhor desempenho. Dessa forma, as imagens utilizadas neste trabalho foram automaticamente redimensionadas no código-fonte, de modo a atender ao tamanho adequado exigido por cada modelo.

Os valores da variável `IMG_SIZE`, que representa a dimensão das imagens de entrada, foram definidos da seguinte forma: para o *MobileNetV3-Large*, **224x224 *pixels***; para o *ResNet101*, igualmente **224x224 *pixels***.

No caso do *EfficientNetV2M*, optou-se por realizar experimentos adicionais a fim de avaliar o impacto da variação do tamanho de entrada no desempenho do modelo. Para isso, foram utilizados quatro tamanhos diferentes, incluindo o padrão recomendado de **480x480 *pixels***. Os tamanhos testados foram: **224x224 *pixels***, **300x300 *pixels***, **380x380 *pixels*** e **480x480 *pixels***.

No **Código-Fonte 3.1** pode ser observado o processo de preparação do *dataset* para redimensionamento das imagens, e no **Código-Fonte 3.4** é apresentado o trecho responsável pela construção do modelo *MobileNetV3-Large*, especificando a dimensão de entrada por meio do parâmetro `input_shape = (*img_size, channels)`, onde `img_size = (224,224)`.

```
base_model = applications.MobileNetV3Large(
    include_top=False,
    weights='imagenet',
    input_shape=(*img_size, channels),
    pooling='avg'
)
```

Código-Fonte 3.4: função de construção do modelo *MobileNetV3-Large* com a definição do parâmetro *img_size*.

3.3.2.2 Tamanho do Lote

O tamanho do lote (***Batch Size***), define o número de amostras de dados que são processadas pelo modelo antes que seus pesos internos sejam atualizados. Para este trabalho, foi fixado um *batch size* de 16 para os conjuntos de treinamento e validação.

A utilização de um valor que é uma potência de dois ($16 = 2^4$) segue uma prática comum que visa otimizar a eficiência computacional na GPU. A aplicação deste hiperparâmetro na criação dos conjuntos de dados pode ser observada no **Código-Fonte 3.1**.

3.3.2.3 Estratégias de *Fine-Tuning*

O processo de treinamento dos modelos foi dividido em duas etapas distintas: *pré-treinamento* e *fine-tuning*. No *pré-treinamento*, apenas a camada de classificação final foi ajustada, mantendo todas as camadas convolucionais das arquiteturas congeladas
`base_model.trainable = False`.

Após essa etapa inicial, procedeu-se ao *fine-tuning* dos modelos. Nesse momento, as arquiteturas base foram parcialmente descongeladas para permitir um ajuste mais refinado dos pesos em níveis mais profundos da rede. Essa liberação ocorreu de forma controlada, respeitando as particularidades de cada arquitetura: no *MobileNetV3-Large*, apenas as últimas **30 camadas** foram liberadas para ajuste, assim como demonstrado no **Código-Fonte 3.5**; no *ResNet-101*, apenas as últimas **60 camadas**; e, por fim, no *EfficientNetV2M*, também foram liberadas as últimas **60 camadas**, assim como demonstrado no **Código-Fonte 3.6** enquanto as demais permaneceram congeladas.

Essa estratégia tem como objetivo ajustar seletivamente as representações de alto nível da rede, responsáveis pela identificação de padrões mais específicos das classes do conjunto de dados de folhas de milho, sem comprometer as representações de baixo e médio nível (como bordas, texturas e formas gerais), que já haviam sido aprendidas de forma robusta.

O congelamento parcial das camadas reduz significativamente o risco de *overfitting*, especialmente em cenários de *datasets* com distribuição desbalanceada ou número limitado de amostras, além de mitigar o custo computacional que ocorreria ao treinar todas as camadas simultaneamente.

```
    base_model.trainable = True
    fine_tune_at = len(base_model.layers) - 30
    for layer in base_model.layers[:fine_tune_at]:
        layer.trainable = False
```

Código-Fonte 3.5: *Fine-tuning* referente ao *MobileNetV3-Large*.

```
base_model.trainable = True
fine_tune_at = len(base_model.layers) - 60
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

Código-Fonte 3.6: *Fine-tuning* referente ao *ResNet-101* e *EfficientNetV2M*.

3.3.2.4 Camadas Adicionais

Com o objetivo de adaptar os modelos pré-treinados às particularidades do conjunto de dados empregado, foi projetada uma **cabeça de classificação customizada** acoplada ao *backbone* de todos os modelos.

As camadas adicionais visam tanto melhorar a capacidade de generalização quanto reduzir riscos de sobreajuste (*overfitting*), garantindo que o modelo aprenda padrões robustos e não apenas memorizações específicas do conjunto de treinamento. A arquitetura proposta pode ser observada no **Código-Fonte 3.7** e é composta pelos seguintes componentes:

- ***Data Augmentation***: aplicado diretamente no *pipeline* do modelo, amplia artificialmente a diversidade das amostras de treino por meio de transformações (rotações, inversões, variações de brilho, entre outras), promovendo maior robustez contra variações intra-classes. A função que implementa esses aumentos encontra-se no **Código-Fonte 3.3**;
- ***Batch Normalization***: normaliza as ativações intermediárias, estabilizando e acelerando o processo de otimização. Essa técnica também atua como uma forma de regularização, reduzindo a dependência do modelo em inicializações específicas e ajudando na convergência;
- ***Dropout***: durante o treinamento, desativa aleatoriamente 30% das ativações, prevenindo a coadaptação excessiva entre neurônios. Essa estratégia de regularização aumenta a capacidade do modelo em generalizar para dados não vistos;
- **Camada Densa de Saída (*Softmax*)**: responsável pela classificação final das amostras. A ativação *softmax* converte os valores da camada anterior em probabilidades normalizadas, adequadas para problemas de classificação multiclasse. Além disso, foi aplicado um *Kernel Regularizer* do tipo **L2** ($\lambda = 1e{-}4$), penalizando pesos excessivamente grandes e, consequentemente, contribuindo para maior controle do sobreajuste.

```
x = get_data_augmentation()(inputs)
x = base_model(x, training=False)
x = layers.BatchNormalization()(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(
    num_classes,
    activation='softmax',
    kernel_regularizer=tf.keras.regularizers.l2(1e-4))(x)
```

Código-Fonte 3.7: camadas adicionais após o *backbone*.

3.3.3 Hiperparâmetros

Hiperparâmetros são valores definidos antes do treinamento e que controlam o comportamento do algoritmo de aprendizado (NYUYTIYMBIY, 2020). Eles têm grande impacto na velocidade de convergência, qualidade da generalização e desempenho final do modelo.

3.3.3.1 Otimizador

Para o processo de otimização dos parâmetros nos três modelos de RNC, foi selecionado o algoritmo ***AdamW***[3]. A escolha fundamenta-se na sua arquitetura, que combina as vantagens do otimizador *Adam* (*Adaptive Moment Estimation*) com uma correção crucial no mecanismo de decaimento de peso (*Weight Decay*).

O otimizador *Adam*, conforme descrito por Kingma e Ba (2017), é um método de descida de gradiente estocástico baseado na estimativa adaptativa de momentos de primeira e segunda ordem. Suas principais vantagens são a eficiência computacional, a baixa exigência de memória e a adequação para problemas com grande volume de dados ou parâmetros.

Contudo, a implementação tradicional do *Adam* trata o decaimento de peso de forma acoplada à atualização dos gradientes, o que pode levar a uma regularização ineficaz. O ***AdamW*** soluciona essa limitação ao desacoplar o decaimento de peso do cálculo dos gradientes. Desta forma, o algoritmo primeiro realiza o decaimento diretamente sobre os pesos do modelo e, subsequentemente, aplica a atualização baseada nos gradientes adaptativos. A configuração do otimizador ***AdamW*** empregada neste trabalho é detalhada no **Código-Fonte 3.8**.

[3]Otimizadores são algoritmos ou métodos utilizados para ajustar os atributos de uma rede neural, como pesos e taxas de aprendizado, com o objetivo de minimizar a função de perda e, consequentemente, otimizar o seu desempenho.

```
model.compile(
    optimizer=tf.keras.optimizers.AdamW(weight_decay=1e-4),
    loss=tf.keras.losses.CategoricalCrossentropy(
        label_smoothing=LABEL_SMOOTHING),
    metrics=['accuracy']
)
```

Código-Fonte 3.8: configuração do treinamento pré *fine-tuning* dos modelos.

3.3.3.2 Função de Perda

A função de perda (*Loss Function*) é responsável por quantificar a discrepância entre os valores previstos pelo modelo e os rótulos verdadeiros de referência. Conforme aponta Bergmann e Stryker (2024), uma perda elevada indica previsões imprecisas, enquanto uma perda reduzida sinaliza um bom ajuste do modelo aos dados. O objetivo do treinamento é, portanto, minimizar o valor desta função.

Para os três modelos desenvolvidos neste trabalho, foi empregada a função de perda **Entropia Cruzada Categórica** (*Categorical Cross-Entropy*), uma escolha padrão para problemas de classificação multiclasse. Essa função mensura a divergência entre a distribuição de probabilidade gerada na saída do modelo e a distribuição alvo, que representa a classe correta.

A distribuição alvo é tipicamente definida por um vetor *One-Hot Encoded*, onde a posição correspondente à classe correta contém o valor 1 e as demais posições contêm 0. Essa codificação foi realizada na etapa de pré-processamento dos dados. No entanto, para aprimorar a robustez do modelo, a função de perda foi combinada com a técnica de regularização **Suavização de Rótulos** (*Label Smoothing*), utilizando um fator de **0.1**.

O *Label Smoothing* mitiga o risco de excesso de confiança (*overconfidence*) do modelo, ao transformar os rótulos "duros"(ex: **[0, 1, 0]**) em rótulos "suaves". Isso é feito subtraindo-se uma pequena porção de probabilidade da classe correta e distribuindo-a uniformemente entre todas as classes. A configuração desta função de perda pode ser observada na implementação apresentada anteriormente no **Código-Fonte 3.8**.

3.3.3.3 Taxa de Aprendizado

A **Taxa de Aprendizado** (*Learning Rate*) é um hiperparâmetro que determina a magnitude dos ajustes nos pesos do modelo durante o processo de otimização, buscando minimizar a função de perda (BELCIC; STRYKER, 2024). Em vez de um valor fixo, este trabalho empregou um agendador dinâmico para gerenciar este hiperparâmetro de forma adaptativa ao longo do treinamento.

A estratégia adotada é conhecida como **Aquecimento com Decaimento Cosseno** (*Warmup with Cosine Decay*) segundo Kalra e Barkeshli (2024), sendo esta implementada por meio de um *callback* que ajusta a taxa de aprendizado a cada passo de treinamento. Essa abordagem é dividida em duas fases distintas:

- **Fase de Aquecimento**: durante as **5 épocas** iniciais de treinamento, a taxa de apren-

dizado aumenta linearmente de um valor próximo a zero até atingir um máximo pré-definido de $1e-3$. Este aquecimento gradual permite que o modelo se ajuste suavemente no início do treinamento, evitando a desestabilização dos pesos herdados do pré-treinamento, o que poderia ocorrer com uma taxa de aprendizado inicialmente alta;

- **Fase de Decaimento Cosseno**: concluída a fase de aquecimento, a taxa de aprendizado inicia um processo de decaimento suave, seguindo a forma de uma função cosseno. Ela parte do valor máximo de $1e-3$ e é progressivamente reduzida até um valor mínimo de $1e-6$ ao final das épocas de treinamento. À medida que o modelo se aproxima de uma região de erro mínimo, passos menores são necessários para realizar um ajuste fino e garantir uma convergência estável. A implementação deste agendador dinâmico, que foi aplicada de forma idêntica a todos os modelos, é detalhada no **Código-Fonte 3.9**.

```python
class WarmupCosineDecay(keras.callbacks.Callback):
    def __init__(self, total_steps, warmup_steps, initial_lr, final_lr):
        super().__init__()
        self.total_steps = total_steps
        self.warmup_steps = warmup_steps
        self.initial_lr = initial_lr
        self.final_lr = final_lr
        self.global_step = 0

    def on_train_batch_begin(self, batch, logs=None):
        self.global_step += 1
        if self.global_step < self.warmup_steps:
            lr = self.initial_lr * (self.global_step / self.warmup_steps)
        else:
            cosine_decay = 0.5 * (1 + math.cos(math.pi *
            (self.global_step - self.warmup_steps) /
            (self.total_steps - self.warmup_steps)))

            lr = self.final_lr + (self.initial_lr - self.final_lr) *
            cosine_decay

        self.model.optimizer.learning_rate.assign(lr)
```

Código-Fonte 3.9: implementação do *callback* para o agendador de taxa de aprendizado.

3.3.3.4 Duração do Treinamento

O treinamento de cada modelo foi estruturado em uma estratégia de duas fases, uma prática comum e eficaz em aprendizagem por transferência (*Transfer Learning*).

A primeira fase consistiu no treinamento da camada de classificação, com o modelo base (pré-treinado) congelado. Esta etapa teve uma duração fixa de **20 épocas**. O objetivo foi

adaptar os pesos da nova camada classificadora às características extraídas pelo modelo base, sem alterar seus pesos originais.

A segunda fase foi a de ajuste fino (*fine-tuning*), onde as camadas superiores do modelo base foram descongeladas para serem treinadas juntamente com a camada de classificação. Para esta fase, foi estipulado um máximo de **30 épocas** adicionais.

A implementação desta estratégia de treinamento é demonstrada no **Código-Fonte 3.10**.

```

# Constantes
EPOCHS_PRE_TRAIN = 20
EPOCHS_FINE_TUNE = 30
TOTAL_EPOCHS = EPOCHS_PRE_TRAIN + EPOCHS_FINE_TUNE

# Fase 1: Treinamento com o modelo base congelado
history_pre = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=EPOCHS_PRE_TRAIN,
    callbacks=[lr_scheduler_cb]
)

# Fase 2: Ajuste fino (fine-tuning)
history_fine = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=TOTAL_EPOCHS,
    initial_epoch=history_pre.epoch[-1] + 1,
    callbacks=[
        lr_scheduler_cb,
        keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=6,
        restore_best_weights=True),

        keras.callbacks.ModelCheckpoint('caminho/do/modelo.keras',
        monitor='val_accuracy',
        save_best_only=True, mode='max')
    ]
)
```

Código-Fonte 3.10: estrutura de treinamento em duas fases com *callbacks* de controle.

3.3.4 Estratégias de Controle de Treinamento

Para garantir a robustez do processo de treinamento e a qualidade do modelo final, foram empregadas duas estratégias de controle implementadas como *callbacks*. O objetivo principal foi aplicar uma forma de regularização para mitigar o *overfitting* e persistir em disco a versão mais performática do modelo.

A primeira estratégia foi a **parada antecipada** (*Early Stopping*), uma técnica de regularização que monitora o desempenho do modelo em dados não vistos. Foi configurada para observar a acurácia no conjunto de validação ao final de cada época. Caso a métrica não apresentasse melhora por um período de **6 épocas** consecutivas, sendo este um hiperparâmetro conhecido como paciência (*patience*), o treinamento seria automaticamente interrompido. Esta abordagem previne que o modelo continue a treinar desnecessariamente e comece a memorizar os dados de treinamento, prejudicando sua capacidade de generalização.

A segunda estratégia foi o **ponto de verificação do modelo** (*Model Checkpoint*). Este *callback* foi configurado para salvar em disco, de forma autônoma, apenas a versão do modelo que atingisse o maior valor de acurácia de validação ao longo de todo o treinamento. Dessa forma, garante-se que o artefato final seja o modelo de melhor desempenho, independentemente de o treinamento ter sido interrompido pela parada antecipada ou ter concluído todas as épocas.

Ambos os *callbacks* foram ativados apenas na segunda fase do treinamento (ajuste fino) como pode ser observado no **Código-Fonte 3.10**. A justificativa para essa decisão é que a primeira fase, com o modelo base congelado, é uma etapa de aprendizado mais estável. O risco de sobreajuste torna-se mais acentuado durante o ajuste fino, quando camadas mais profundas são treinadas, tornando o monitoramento e o controle mais críticos nesta etapa.

3.4 Avaliação dos Modelos

Para aferir o desempenho e a capacidade de generalização dos modelos treinados, foi conduzida uma avaliação quantitativa. Foram utilizadas métricas específicas para analisar a dinâmica do aprendizado, diagnosticar anomalias como sobreajuste (*overfitting*) ou subajuste (*underfitting*), e mensurar a performance de classificação em detalhes.

3.4.1 Métricas Utilizadas

A avaliação foi fundamentada em três pilares: a análise das curvas de aprendizado (**Acurácia e Perda**), a inspeção visual dos erros de classificação (**Matriz de Confusão**) e um relatório quantitativo de desempenho por classe (***Classification Report***).

As curvas de **Acurácia** (*Accuracy*) e **Perda** (*Loss*) foram geradas para cada modelo, plotando-se as métricas de treinamento e de validação em função das épocas. Esta análise comparativa é fundamental para visualizar a convergência do modelo e diagnosticar problemas. Uma grande lacuna entre as curvas de treinamento e validação, por exemplo, é um claro indicador de sobreajuste.

A **Matriz de Confusão** serviu como uma ferramenta diagnóstica essencial para a análise qualitativa dos erros. Através dela, foi possível identificar padrões de classificação incorreta entre as classes. Um exemplo notório ocorreu antes da aplicação de técnicas de balanceamento, onde a matriz revelou que os modelos confundiam sistematicamente as classes *Gray Leaf Spot* e *Blight*. Esta observação foi crucial para a decisão de aprimorar o conjunto de dados.

Por fim, o ***Classification Report*** forneceu um resumo quantitativo detalhado do desem-

penho. Este relatório inclui as seguintes métricas por classe:

- **Precisão (*Precision*):** de todas as previsões positivas para uma classe, qual a porcentagem de acerto;
- **Revocação (*Recall*):** de todas as amostras que realmente pertencem a uma classe, qual a porcentagem que o modelo identificou corretamente;
- ***F1-Score*:** A média harmônica entre precisão e revocação, útil para balancear ambas as métricas;
- **Suporte (*Support*):** o número real de ocorrências de cada classe no conjunto de dados.

As funções utilizadas para gerar essas avaliações são apresentadas no **Código-Fonte 3.11** e **Código-Fonte 3.12**.

```python
def plot_history(history, title_prefix):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs_range = range(len(acc))
    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Acurácia do Treinamento')
    plt.plot(epochs_range, val_acc, label='Acurácia da Validação')
    plt.legend(loc='lower right')
    plt.title(f'{title_prefix} - Acurácia')
    plt.xlabel('Épocas')
    plt.ylabel('Acurácia')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Loss do Treinamento')
    plt.plot(epochs_range, val_loss, label='Loss da Validação')
    plt.legend(loc='upper right')
    plt.title(f'{title_prefix} - Loss)')
    plt.xlabel('Épocas')
    plt.ylabel('Loss')

    plt.suptitle(title_prefix, fontsize=16)
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()
```

Código-Fonte 3.11: função *plot_history* que exibe os gráficos de acurácia e *loss* para treinamento e validação.

```python
def evaluate_model(model, val_ds, class_names):
    y_true, y_pred = [], []
    for images, labels in val_ds:
        preds = model.predict(images)
        y_true.extend(np.argmax(labels.numpy(), axis=1))
        y_pred.extend(np.argmax(preds, axis=1))]

    print(classification_report(y_true, y_pred,
        target_names=class_names,
        digits=4))
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(cm, display_labels=class_names)
    disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
    plt.title("Matriz de Confusão")
    plt.tight_layout()
    plt.show()
```

Código-Fonte 3.12: função *evaluate_model* que exibe a matriz de confusão e o *classification report*.

3.4.2 Controle de Sobreajuste e Subajuste

Um dos principais desafios no treinamento de redes neurais profundas é encontrar um equilíbrio entre o sobreajuste (*overfitting*) e o subajuste (*underfitting*). As técnicas para realizar este controle foram desenvolvidas em tópicos anteriores.

Para mitigar o sobreajuste, fenômeno no qual o modelo apresenta alto desempenho nos dados de treinamento mas falha em generalizar para novos dados, foram aplicadas as seguintes técnicas de regularização:

- ***Data Augmentation***;
- ***Dropout***;
- ***Early Stopping***;
- **Regularização L2**;
- ***Oversampling***.

Por outro lado, para mitigar o subajuste, que ocorre quando o modelo é incapaz de capturar a complexidade dos dados, as seguintes abordagens foram adotadas:

- **Uso de Modelos de Alta Capacidade**;
- **Duração Extensiva do Treinamento**;
- **Agendador de Taxa de Aprendizado**.

3.5 Ambiente e Ferramentas de Desenvolvimento

Esta seção detalha o conjunto de tecnologias, bibliotecas e a infraestrutura de hardware utilizadas para a implementação, treinamento e disponibilização dos modelos de classificação de imagens desenvolvidos neste trabalho, além disso será apresentado a aplicação *Web* implementada para complementar o treinamento.

3.5.1 Ferramentas e Tecnologias Utilizadas

O projeto foi desenvolvido sobre um ecossistema de software robusto, centrado na linguagem de programação **Python**, cuja escolha se justifica por sua vasta comunidade e pela maturidade de suas bibliotecas para aprendizado de máquina e análise de dados.

O núcleo do desenvolvimento de aprendizado profundo foi sustentado pelas seguintes bibliotecas:

- **TensorFlow (com Keras):** utilizado como o *framework* principal para a definição, treinamento e inferência dos modelos de redes neurais convolucionais;
- **Scikit-Learn:** empregado para a geração de métricas de avaliação de desempenho, como o relatório de classificação e a matriz de confusão;
- **NumPy:** utilizado para a manipulação eficiente de arranjos numéricos multidimensionais (tensores);
- **Matplotlib:** utilizado para a visualização de dados, incluindo a plotagem das curvas de aprendizado e da matriz de confusão.

O treinamento dos modelos foi executado em ambiente de computação em nuvem, utilizando a plataforma **Google Colab Pro**. A infraestrutura de hardware disponibilizada consistiu em uma GPU **NVIDIA L4**, equipada com **22,5 GB** de memória GDDR6 e baseada na arquitetura Ada Lovelace (XIAOTENG; LIU; HALIM, 2024). Este acelerador foi crucial para o projeto, pois seus **núcleos tensores de 4ª geração** aceleram massivamente os cálculos de matrizes, e sua ampla memória de vídeo (VRAM) foi essencial para acomodar os modelos profundos e as imagens de alta resolução utilizadas. O ambiente também contava com **53 GB** de memória RAM do sistema e **235,7 GB** de armazenamento.

Adicionalmente, foi desenvolvida uma aplicação *Web* para permitir a interação do usuário com o modelo treinado. A arquitetura da aplicação foi composta por: **React** e **TailwindCSS** para a interface do usuário (*frontend*), realizando requisições ao *backend* com a biblioteca **Axios** e **Python** com o *micro-framework* **FastAPI** para a construção da API de inferência (*backend*). O controle de versão do código-fonte foi gerenciado com **Git** e hospedado remotamente na plataforma **GitHub**[4].

[4]Link para o repositório: https://github.com/gustavokv/disease-detection

3.5.2 Aplicação *Web*

Para demonstrar a aplicabilidade prática dos modelos treinados, foi desenvolvida uma aplicação *Web* que serve como uma interface interativa para o usuário. A aplicação permite a classificação de novas imagens, fornecendo um meio de validar visualmente o desempenho do modelo com dados arbitrários.

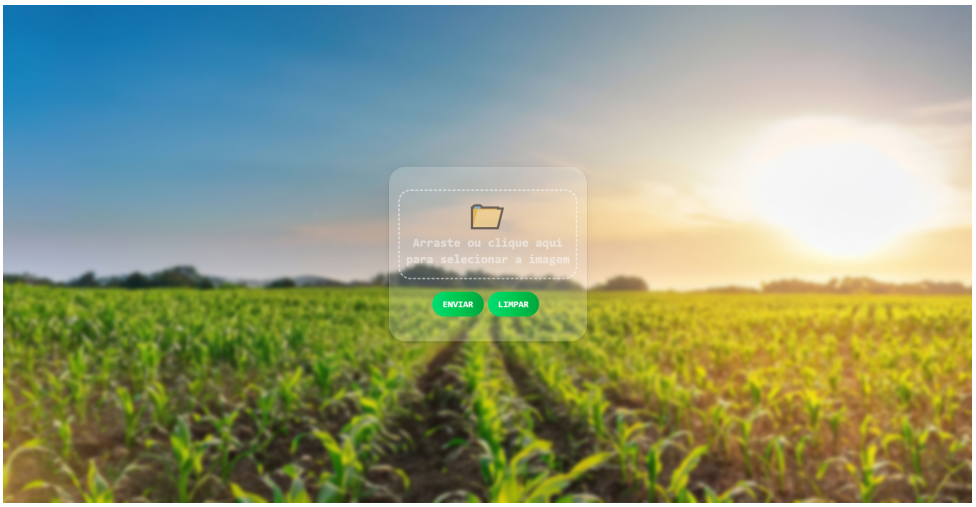
3.5.2.1 Arquitetura da Aplicação

A aplicação foi projetada como uma **Aplicação de Página Única** (*Single-Page Application - SPA*), com uma clara separação entre a interface do usuário (*frontend*) e a lógica de inferência do modelo (*backend*).

A interface, desenvolvida com a biblioteca **React**, proporciona uma experiência de uso fluida. Conforme ilustrado na **Figura 3.8**, o sistema oferece ao usuário a capacidade de submeter uma imagem para análise por meio de seleção de arquivo ou da funcionalidade de arrastar-e-soltar (*drag-and-drop*). Uma vez que a imagem é carregada (**Figura 3.9**), o usuário pode acionar o processo de classificação ou limpar a seleção atual.

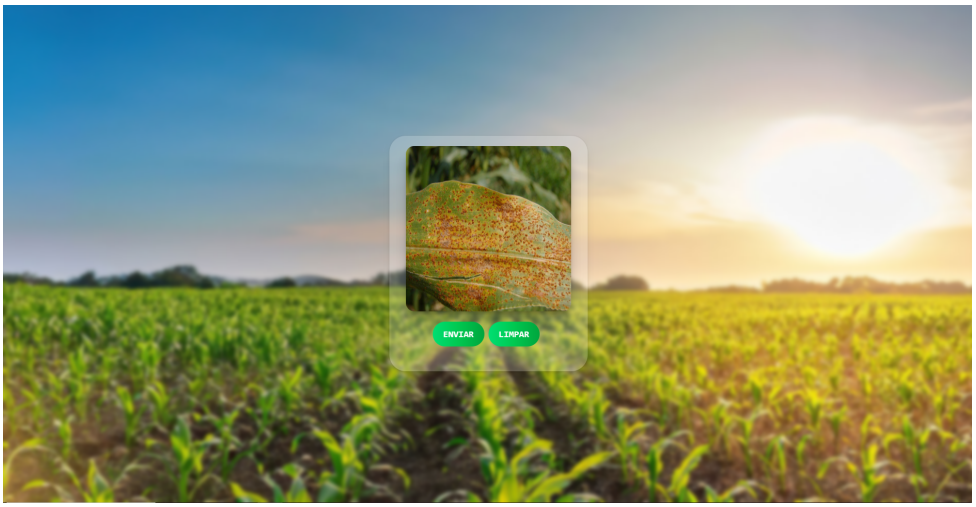
A comunicação com o servidor é realizada por uma requisição HTTP assíncrona, gerenciada pela biblioteca **Axios**. O servidor, implementado em **Python** com o *framework* **FastAPI**, recebe a imagem, executa a inferência com o modelo treinado (como pode ser observado no **Código-Fonte 3.13**) e retorna a **Classe** predita e a respectiva probabilidade de **Confiança**. O resultado é então exibido dinamicamente na interface para o usuário.

Figura 3.8: interface da aplicação *Web* em seu estado inicial.



Fonte: elaborado pelo autor.

Figura 3.9: interface da aplicação após o carregamento de uma imagem pelo usuário.



Fonte: elaborado pelo autor.

```
MODEL = tf.keras.models.load_model('../saved_models/
        efficientnetv2m_db1_corn_classifier.keras')

CLASS_NAMES = ["Blight", "Common_Rust", "Gray_Leaf_Spot", "Healthy"]

def read_file_as_image(data) -> np.ndarray:
    img = np.array(Image.open(BytesIO(data)))
    return img

@app.post('/predict')
async def predict(file: UploadFile = File(...)):
    img = read_file_as_image(await file.read())
    img_resized = resize(img, [480,480])
    img_batch = np.expand_dims(img_resized, 0)

    predictions = MODEL.predict(img_batch)
    predicted_class = CLASS_NAMES[np.argmax(predictions[0])]
    confidence = np.max(predictions[0])

    print(predicted_class)

    return {
        'class': predicted_class,
        'confidence': float(confidence)
    }
```

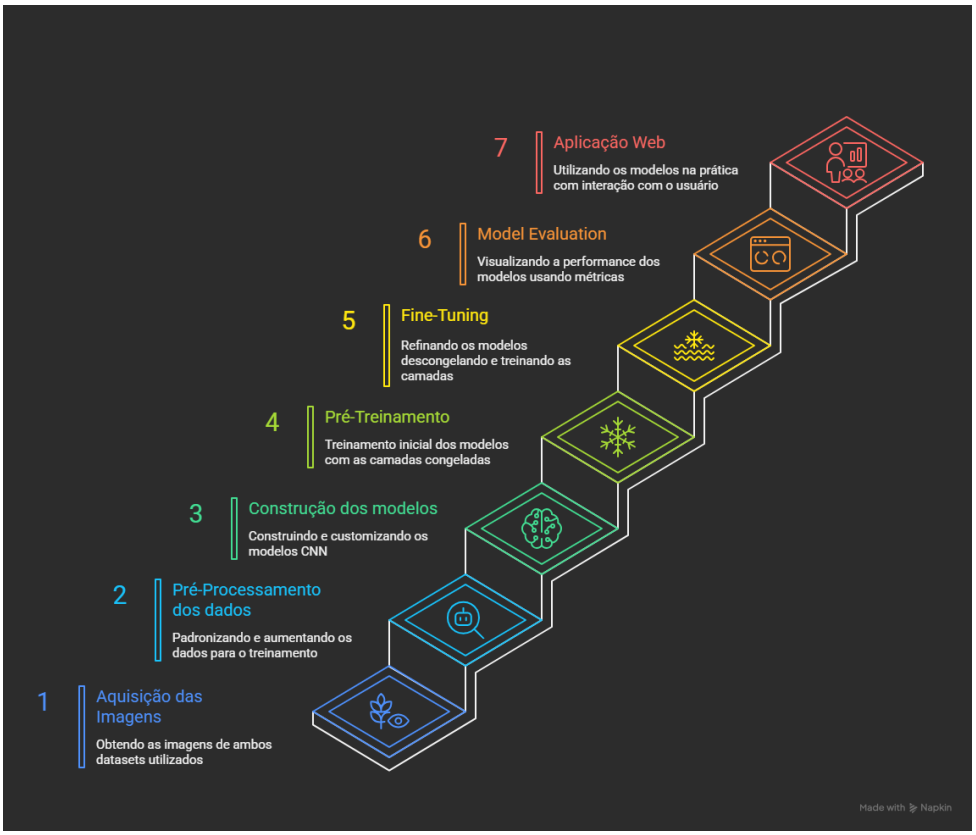
Código-Fonte 3.13: função *end-point* com *FastAPI* para receber a requisição da imagem e retornar a inferência.

3.6 *Pipeline* Integrada do Projeto

Esta seção consolida as etapas metodológicas descritas anteriormente, apresentando um resumo gráfico que oferece uma visão holística do fluxo de trabalho do projeto. O *pipeline* ilustra o processo completo, desde a aquisição e o pré-processamento dos dados até o treinamento, a avaliação e, finalmente, a implantação dos modelos em uma aplicação *Web* interativa.

Para garantir a comparabilidade dos resultados, foi estabelecido um processo de treinamento unificado, aplicado de forma consistente aos três modelos de RNCs avaliados. O fluxograma geral que representa este *pipeline* padronizado é apresentado na **Figura 3.10**.

Figura 3.10: visão geral do *pipeline* do projeto, da aquisição de dados à inferência na aplicação *Web*.



Fonte: elaborado pelo autor.

CAPÍTULO 4

Resultados

Este capítulo é dedicado à apresentação e à análise comparativa dos resultados obtidos a partir dos experimentos com os três modelos de Redes RNC. O desempenho aqui documentado reflete diretamente a aplicação das estratégias metodológicas definidas anteriormente, incluindo a otimização de hiperparâmetros, o balanceamento de classes por sobreamostragem (*oversampling*) e a implementação de um conjunto de técnicas de regularização para mitigar os efeitos de sobreajuste (*overfitting*) e subajuste (*underfitting*).

4.1 Resultados do Treinamento dos Modelos

Nesta seção, são apresentados e analisados os resultados quantitativos obtidos por cada um dos três modelos de RNCs. A avaliação foca na comparação de desempenho entre as arquiteturas e na influência dos diferentes conjuntos de dados utilizados no treinamento.

4.1.1 Acurácia e Perda

A análise quantitativa do desempenho inicia-se com as métricas primárias de Acurácia e Perda, que indicam a capacidade de generalização do modelo e a sua convergência durante o treinamento. Os resultados finais de cada modelo no conjunto de validação são consolidados nas tabelas a seguir, permitindo uma comparação direta.

4.1.1.1 *MobileNetV3-Large* e *ResNet101*

Os modelos **MobileNetV3-Large** e **ResNet101** foram treinados sob as mesmas condições para ambos os conjuntos de dados. A **Tabela 4.1** sumariza os valores de acurácia e perda obtidos ao final do treinamento.

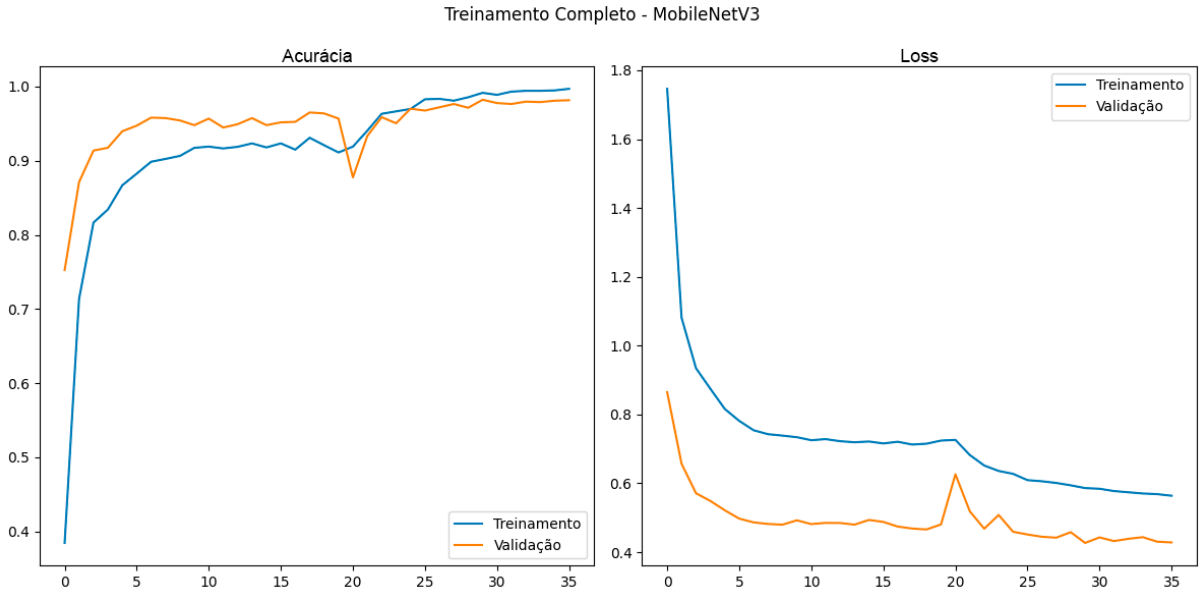
Tabela 4.1: resultados finais de Acurácia e Perda para os modelos MobileNetV3 e ResNet101.

Modelo	**Base de Dados**	**Treinamento**		**Validação**	
		Acurácia	Perda	Acurácia	Perda
MobileNetV3-Large	PlantDoc + PlantVillage	0,99	0,57	0,9822	0,43
	CD&S	0,99	0,55	0,9546	0,41
ResNet101	PlantDoc + PlantVillage	0,98	0,71	0,9740	0,41
	CD&S	0,96	0,16	0,9475	0,15

Fonte: elaborada pelo autor.

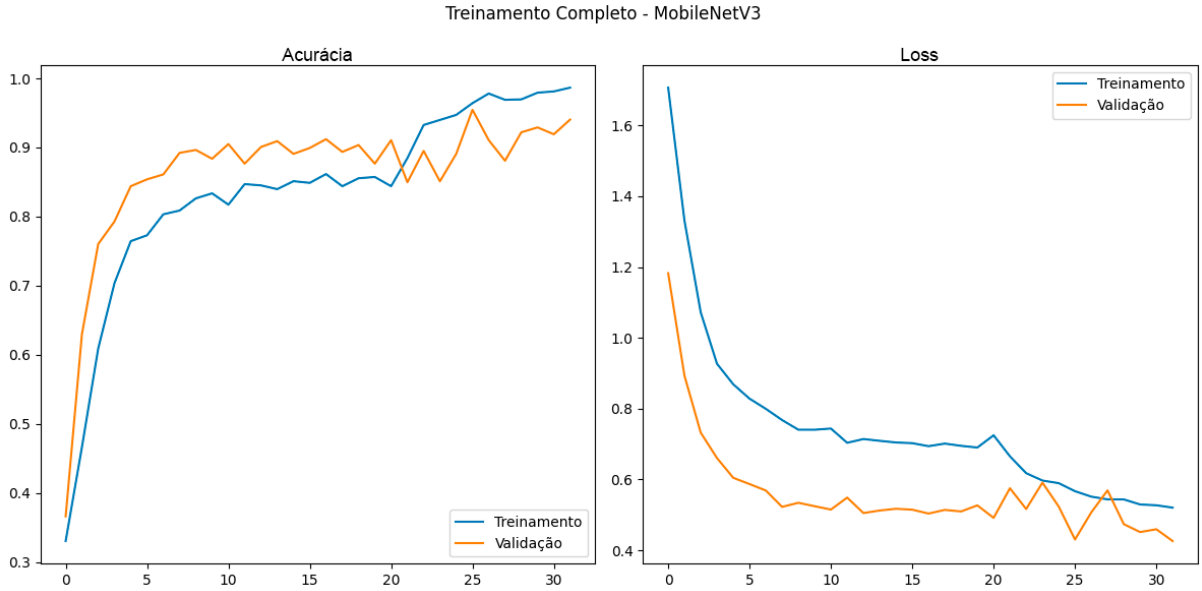
Observa-se que ambos os modelos atingiram altos índices de acurácia, validando a eficácia das estratégias de pré-processamento e regularização empregadas. O *MobileNetV3-Large* obteve a maior acurácia de validação (**98,22%**) com a base de dados combinada, enquanto o *ResNet101* demonstrou uma perda de validação excepcionalmente baixa (**0,15**) no *dataset* CD&S. As curvas de aprendizado que ilustram a evolução dessas métricas ao longo das épocas são apresentadas nas **Figuras** 4.1 a 4.4.

Figura 4.1: curvas de Acurácia e Perda (MobileNetV3-Large) na base *PlantDoc + PlantVillage*.



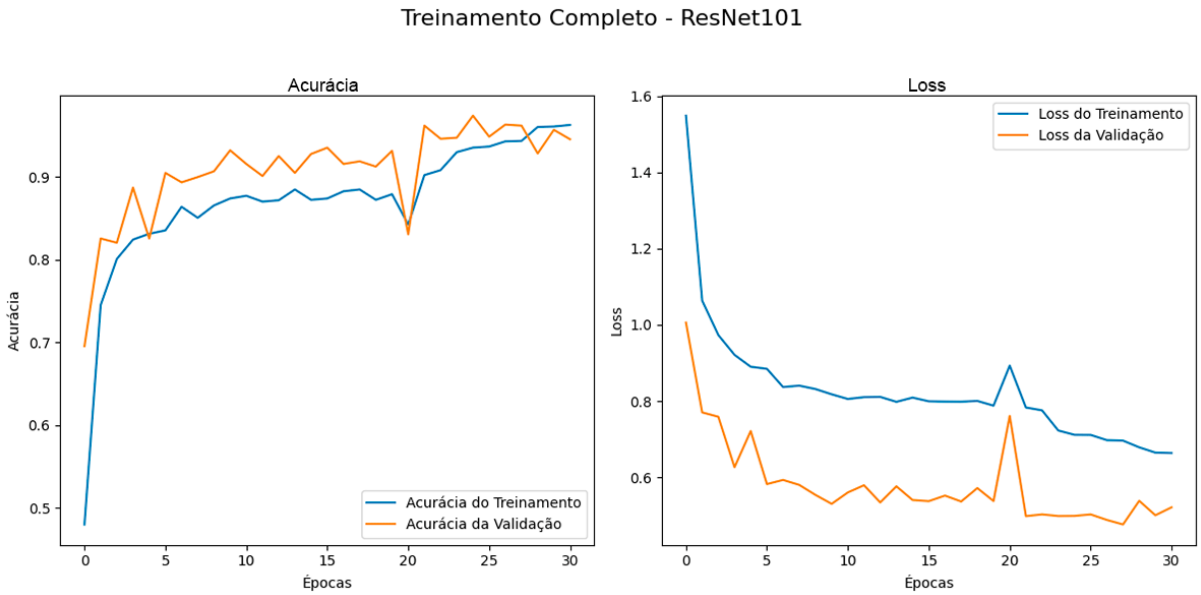
Fonte: elaborada pelo autor.

Figura 4.2: curvas de Acurácia e Perda (MobileNetV3-Large) na base CD&S.



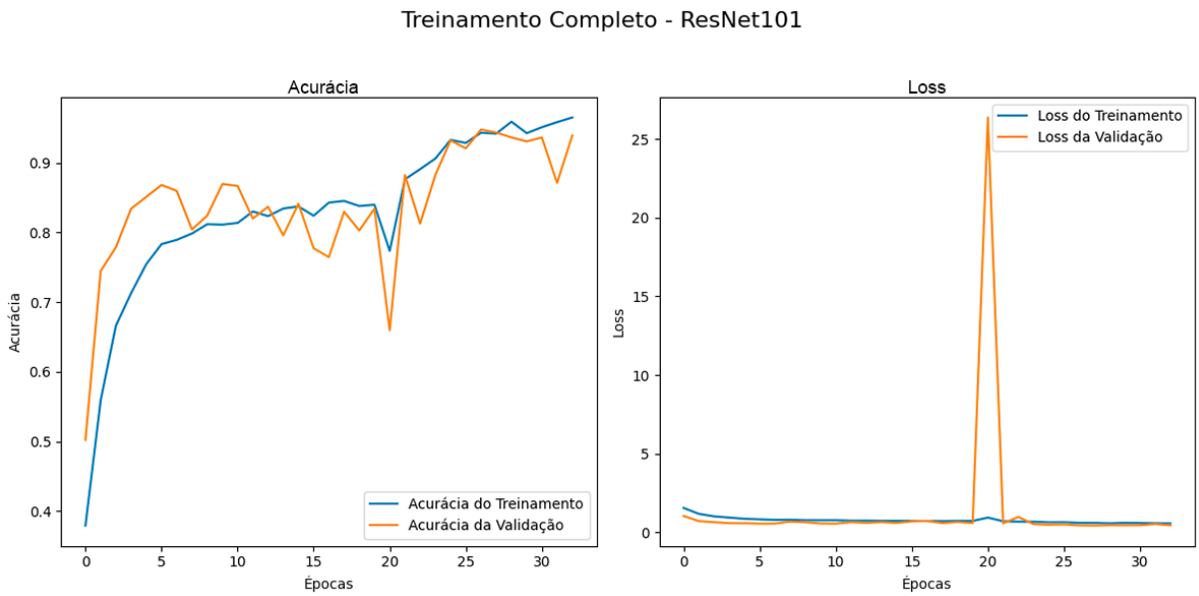
Fonte: elaborada pelo autor.

Figura 4.3: curvas de Acurácia e Perda (ResNet101) na base *PlantDoc* + *PlantVillage*.



Fonte: elaborada pelo autor.

Figura 4.4: curvas de Acurácia e Perda (ResNet101) na base CD&S.



Fonte: elaborada pelo autor.

4.1.1.2 *EfficientNetV2M*

Para a arquitetura **EfficientNetV2M**, foi conduzido um experimento adicional para investigar o impacto da resolução da imagem de entrada (*IMG_SIZE*) no desempenho do modelo. A **Tabela 4.2** detalha os resultados obtidos para quatro diferentes resoluções em ambos os conjuntos de dados.

Tabela 4.2: resultados de Acurácia e Perda para o EfficientNetV2M com diferentes resoluções de imagem.

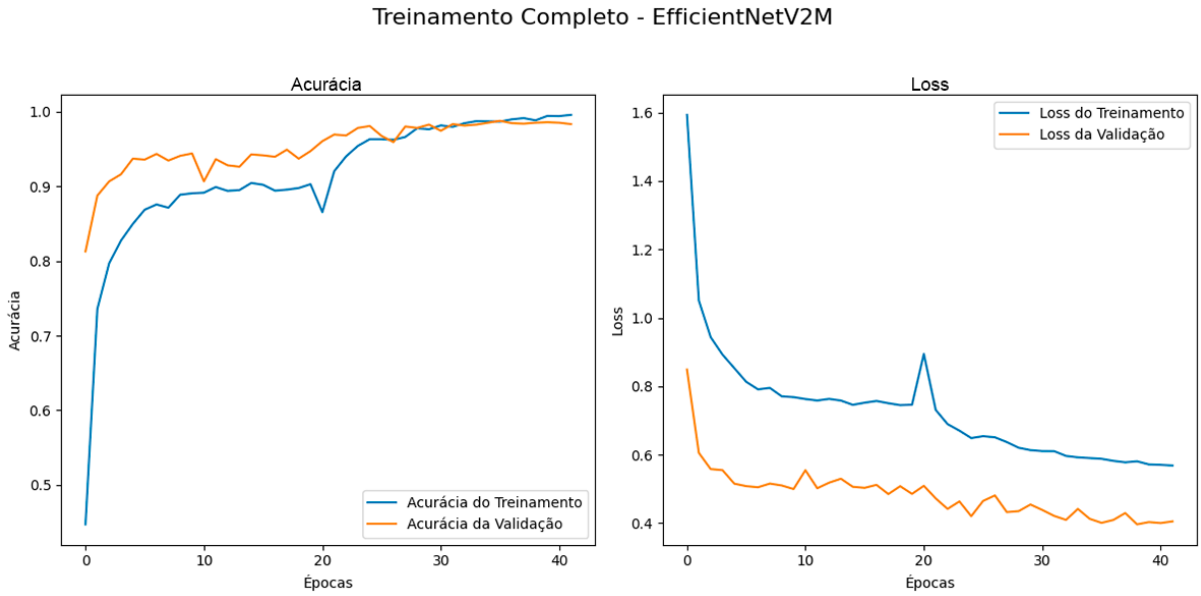
Base de Dados	**Tamanho da Imagem**	**Treinamento**		**Validação**	
	(`IMG_SIZE`)	Acurácia	Perda	Acurácia	Perda
PlantDoc + PlantVillage	224x224	0,99	0,60	0,9791	0,41
	300x300	0,99	0,60	0,9791	0,41
	380x380	0,99	0,63	0,9860	0,42
	480x480 (Ideal)	0,99	0,62	**0,9879**	0,42
CD&S	224x224	0,98	0,58	0,9617	0,41
	300x300	0,99	0,61	0,9901	0,40
	380x380	0,99	0,60	0,9943	0,38
	480x480 (Ideal)	0,99	0,58	**0,9972**	**0,36**

Fonte: elaborada pelo autor.

Os resultados demonstram uma clara correlação positiva entre o aumento da resolução da imagem e a acurácia de validação, especialmente no *dataset* CD&S. O desempenho máximo

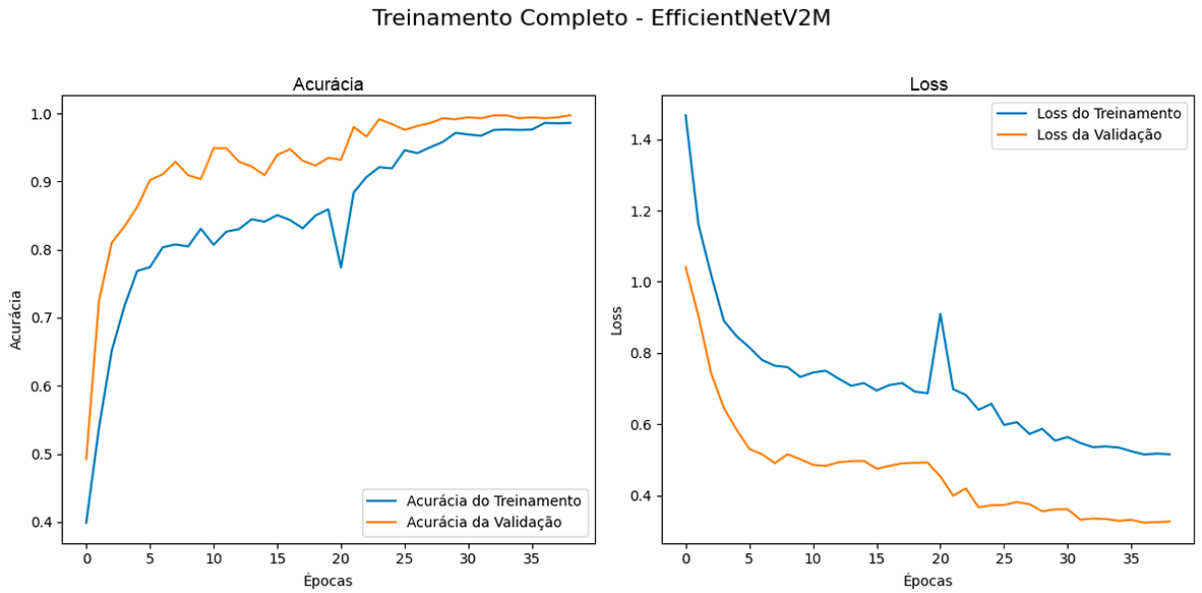
foi consistentemente alcançado com a resolução de *480x480 pixels*, validando-a como a escolha ideal para esta arquitetura. Com esta resolução, o modelo atingiu uma acurácia de validação de **99,72%** no *dataset* CD&S, o melhor resultado entre todos os experimentos conduzidos. As curvas de aprendizado para esta configuração ótima são apresentadas nas **Figuras 4.5** e **4.6**.

Figura 4.5: curvas de Acurácia e Perda (EfficientNetV2M, *480x480*) na base *PlantDoc* + *PlantVillage*.



Fonte: elaborada pelo autor.

Figura 4.6: curvas de Acurácia e Perda (EfficientNetV2M, *480x480*) na base CD&S.



Fonte: elaborada pelo autor.

4.1.2 Comparação Entre os Modelos

Dado a análise anterior, a **Tabela 4.3** consolida a melhor performance alcançada por cada uma das três arquiteturas avaliadas, considerando a maior acurácia de validação obtida nos experimentos. A análise comparativa dos resultados revela um claro destaque para o modelo **EfficientNetV2M**.

Operando com imagens de resolução *480x480* no conjunto de dados **CD&S**, o EfficientNetV2M atingiu uma acurácia de validação de **99,72%** e a menor perda de validação entre todos os testes, com **0,36**. Este resultado não apenas supera as outras arquiteturas, MobileNetV3-Large (98,22%) e ResNet101 (97,40%), mas também valida a hipótese de que a maior resolução de imagem, combinada com a eficiência da arquitetura **EfficientNet**, é a abordagem mais eficaz para o problema proposto.

Tabela 4.3: comparativo de desempenho final entre as arquiteturas de modelo, destacando a melhor performance de validação alcançada por cada uma.

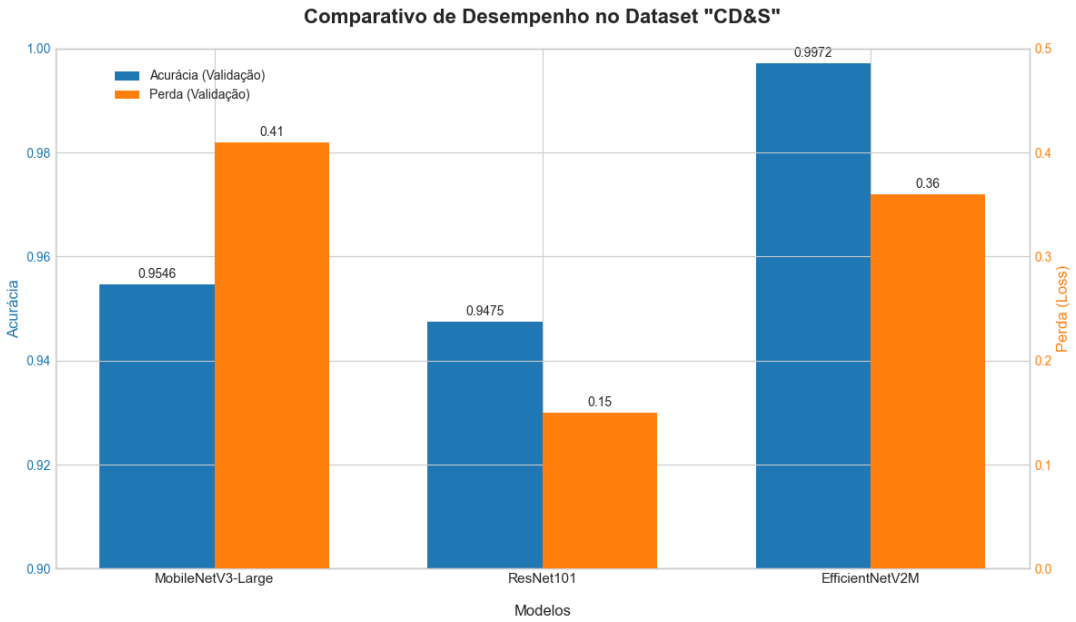
Modelo	Base de Dados	Tamanho da Imagem	Acurácia (Validação)	Perda (Validação)
MobileNetV3-Large	PlantDoc + PlantVillage	224x224	0,9822	0,43
ResNet101	PlantDoc + PlantVillage	224x224	0,9740	0,41
EfficientNetV2M	**CD&S**	**480x480**	**0,9972**	**0,36**

Fonte: elaborada pelo autor.

Para uma análise comparativa direta do desempenho final de cada arquitetura, as métricas de acurácia e perda de validação são apresentadas graficamente. A **Figura 4.7** ilustra

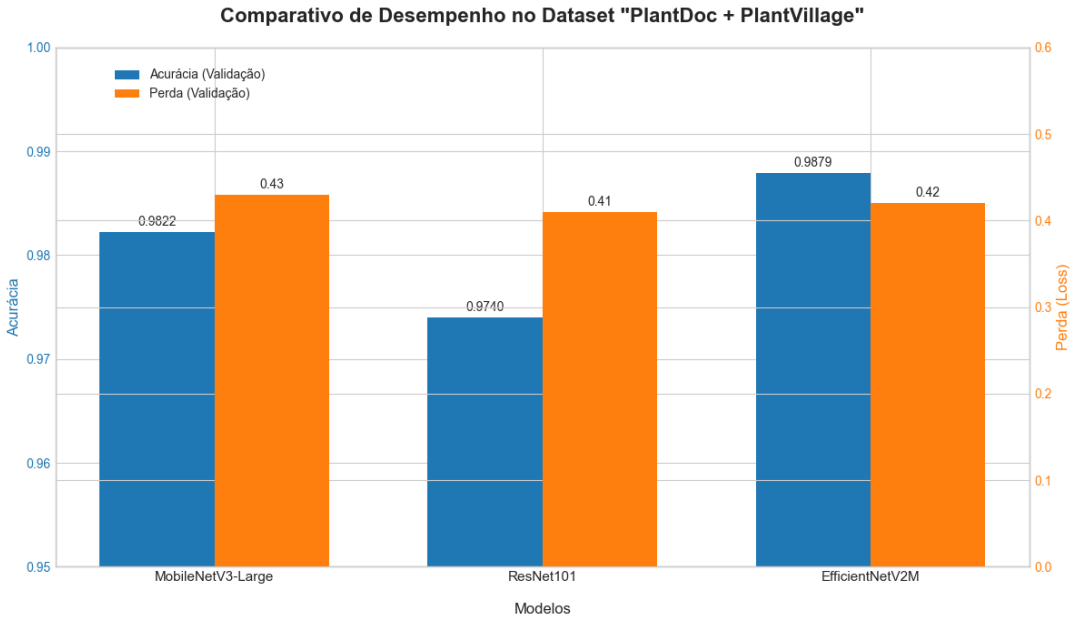
os resultados consolidados para o treinamento com o *dataset* **CD&S**, enquanto a **Figura 4.8** apresenta o desempenho obtido com o *dataset* combinado **PlantDoc + PlantVillage**.

Figura 4.7: desempenho comparativo final (Acurácia e Perda) no *dataset* CD&S.



Fonte: elaborada pelo autor.

Figura 4.8: desempenho comparativo final (Acurácia e Perda) no *dataset PlantDoc + PlantVillage*.



Fonte: elaborada pelo autor.

4.2 Avaliação Detalhada

Nesta seção será apresentado uma avaliação mais detalhada do comportamento do modelo que obteve um melhor desempenho no que se refere em acurácia e perda (EfficientNetV2M), sendo analisado como se comportou com cada classe e quais foram as dificuldades.

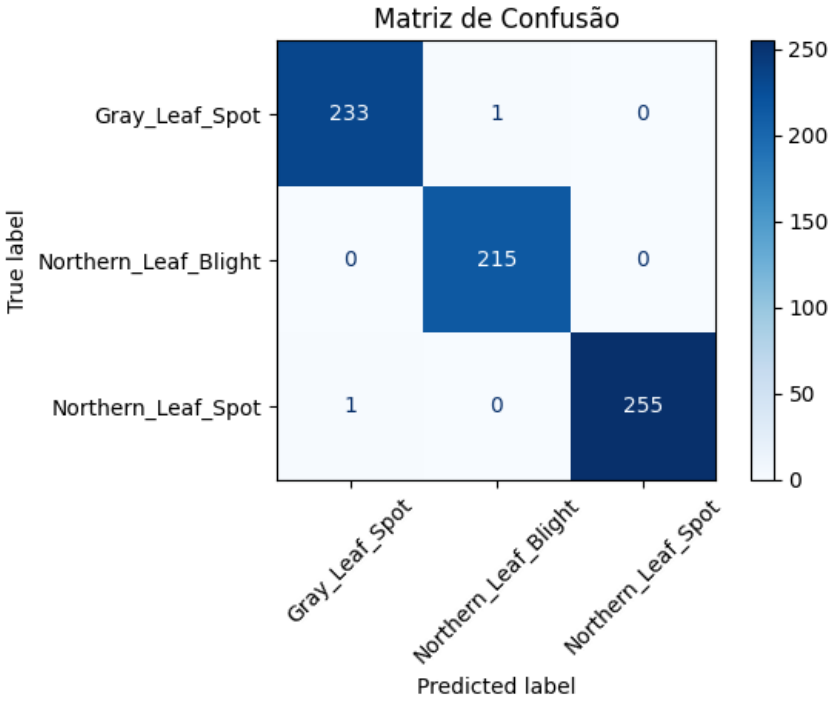
4.2.1 Análise Qualitativa de Desempenho via Matriz de Confusão

Para além das métricas quantitativas como acurácia, a matriz de confusão oferece uma análise qualitativa indispensável sobre o comportamento do modelo. Ela permite uma inspeção detalhada do desempenho em cada classe, revelando acertos, erros e, crucialmente, os padrões de confusão entre as categorias. A seguir, são analisadas as matrizes de confusão para o modelo **EfficientNetV2M** nos dois conjuntos de dados.

4.2.1.1 Desempenho no Dataset CD&S

A **Figura 4.9** apresenta a matriz de confusão para o modelo treinado com o dataset **CD&S**. A análise da matriz revela um desempenho **excepcional**, com uma performance de classificação próxima da perfeição.

Figura 4.9: matriz de confusão do modelo EfficientNetV2M no *dataset* CD&S.



Fonte: elaborada pelo autor.

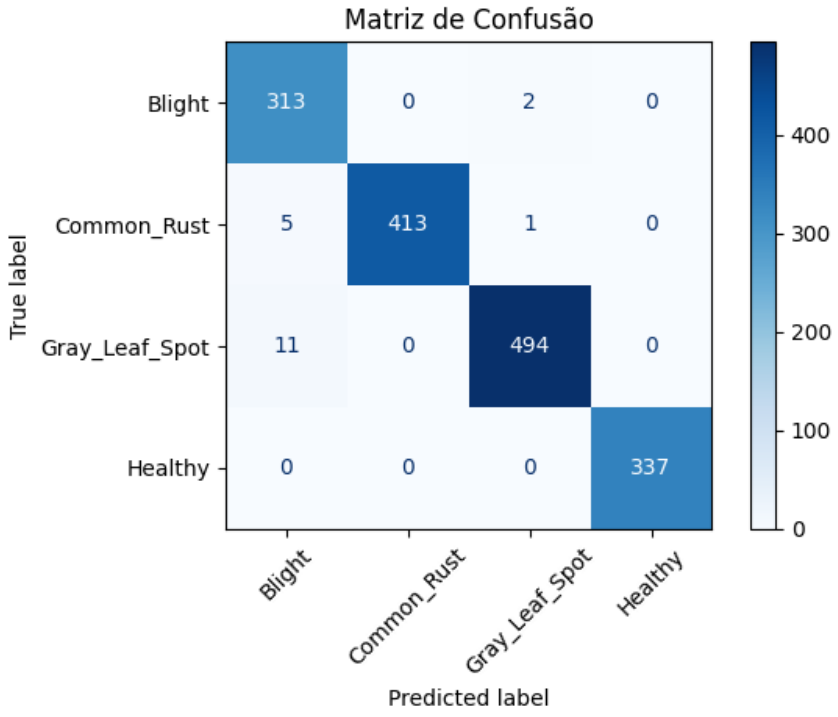
De um total de 705 amostras de validação, ocorreram apenas **duas classificações incorretas**. A classe ***Northern Leaf Blight*** foi identificada com 100% de precisão e revocação, sem nenhum erro. As duas únicas falhas foram mínimas e não indicam um viés sistemático: uma

instância de ***Gray Leaf Spot*** foi confundida com ***Northern Leaf Blight***, e uma de ***Northern Leaf Spot*** foi classificada como ***Gray Leaf Spot***. Este resultado demonstra o altíssimo poder discriminativo do modelo para distinguir as características visuais sutis entre as três doenças presentes neste *dataset*.

4.2.1.2 Desempenho no Dataset PlantDoc + PlantVillage

A **Figura 4.10** exibe a matriz de confusão para o mesmo modelo, mas treinado com o *dataset* combinado **PlantDoc + PlantVillage**. Embora o desempenho geral permaneça **extremamente robusto**, a matriz revela a maior complexidade desta tarefa de classificação, que envolve quatro classes.

Figura 4.10: matriz de confusão do modelo EfficientNetV2M no *dataset PlantDoc+ PlantVillage*.



Fonte: elaborada pelo autor.

Dois pontos de destaque emergem da análise:

- **Identificação Perfeita de Folhas Saudáveis:** o modelo classificou corretamente todas as 337 instâncias da classe **Healthy**, sem cometer nenhum erro. Este é um resultado de grande importância prática, pois indica que o modelo é extremamente confiável para identificar plantas sadias, minimizando falsos positivos de doenças;
- **Padrão de Confusão Principal:** a maior parte dos erros de classificação ocorreu entre as doenças. O padrão de confusão mais significativo foi a classificação incorreta de

11 amostras de ***Gray Leaf Spot*** como ***Blight***. Este erro sugere uma alta similaridade visual entre os sintomas dessas duas doenças neste *dataset*, representando o maior desafio para o modelo. Confusões menores, como 5 instâncias de ***Common Rust*** classificadas como ***Blight***, também foram observadas.

4.2.2 Análise Quantitativa (Precisão, Revocação e F1-Score)

Enquanto a acurácia geral oferece uma visão panorâmica do desempenho, uma análise granular por classe é essencial para compreender o comportamento do modelo. As métricas de Precisão (*Precision*), Revocação (*Recall*) e *F1-Score*, consolidadas no Relatório de Classificação (*Classification Report*), fornecem essa visão detalhada, sendo especialmente importantes em cenários com múltiplos rótulos.

4.2.2.1 Desempenho no *Dataset CD&S*

A **Tabela 4.4** apresenta os resultados quantitativos do modelo **EfficientNetV2M** no *dataset* **CD&S**. Os resultados confirmam a performance excepcional observada na matriz de confusão.

Tabela 4.4: relatório de Classificação do modelo EfficientNetV2M no *dataset* CD&S.

Classe	Precisão	Revocação	F1-Score	Suporte
Gray_Leaf_Spot	0,9957	0,9957	0,9957	234
Northern_Leaf_Blight	0,9954	1,0000	0,9977	215
Northern_Leaf_Spot	1,0000	0,9961	0,9980	256
Acurácia			0,9972	705
Média Macro	0,9970	0,9973	0,9971	705
Média Ponderada	0,9972	0,9972	0,9972	705

Fonte: elaborada pelo autor.

O desempenho do modelo foi uniformemente excelente, com valores de *F1-Score* superiores a 0,995 para todas as classes. Isso indica um equilíbrio quase perfeito entre Precisão e Revocação, sem pontos fracos evidentes. Destaca-se a **revocação perfeita (1,0000)** para a classe ***Northern Leaf Blight***, significando que o modelo identificou corretamente todas as amostras desta doença, e a **precisão perfeita (1,0000)** para ***Northern Leaf Spot***, indicando que nenhuma outra classe foi incorretamente classificada como tal. Conclui-se que o modelo não apresentou dificuldades significativas neste *dataset*.

4.2.2.2 Desempenho no *Dataset PlantDoc + PlantVillage*

A **Tabela 4.5** detalha o desempenho no *dataset* combinado **PlantDoc + PlantVillage**. Embora a performance geral continue em um patamar de excelência, a análise por classe revela nuances importantes e as dificuldades específicas do modelo.

Tabela 4.5: relatório de Classificação do modelo EfficientNetV2M no *dataset PlantDoc + PlantVillage*.

Classe	Precisão	Revocação	F1-Score	Suporte
Blight	0,9514	0,9937	0,9720	315
Common_Rust	1,0000	0,9857	0,9928	419
Gray_Leaf_Spot	0,9940	0,9782	0,9860	505
Healthy	1,0000	1,0000	1,0000	337
Acurácia			0,9879	1576
Média Macro	0,9863	0,9894	0,9877	1576
Média Ponderada	0,9883	0,9879	0,9880	1576

Fonte: elaborada pelo autor.

A análise quantitativa deste relatório corrobora os achados da matriz de confusão:

- **Excelência na Classe 'Healthy':** o modelo alcançou a perfeição, com Precisão, Revocação e *F1-Score* de **1,0000** para a classe de folhas saudáveis. Isso reforça a confiabilidade do modelo em seu caso de uso mais crítico: não diagnosticar uma doença onde não existe;
- **Dificuldade Principal – Precisão da Classe 'Blight':** a classe ***Blight*** apresentou a métrica de desempenho mais baixa, com uma **precisão de 0,9514**. Isso significa que, embora o modelo seja muito bom em encontrar quase todos os casos de ***Blight*** (revocação de 0,9937), ele tende a classificar outras doenças como ***Blight*** com mais frequência (gerando falsos positivos para esta classe). Este foi o principal ponto de dificuldade do modelo;
- **Relação com a Classe 'Gray Leaf Spot':** a dificuldade acima está diretamente ligada à revocação ligeiramente mais baixa da classe ***Gray Leaf Spot*** **(0,9782)**. Esta métrica indica que algumas amostras de ***Gray Leaf Spot*** não foram identificadas corretamente, sendo, como visto na matriz de confusão, classificadas como ***Blight***;
- **Força na Classe 'Common Rust':** similarmente à classe ***Healthy***, o modelo demonstrou uma **precisão perfeita (1,0000)** para a classe ***Common Rust***, indicando que nunca errou ao predizer esta ferrugem.

4.3 Análise de Sobreajuste e Subajuste

Uma análise crítica dos resultados envolve a avaliação de quão bem os modelos generalizam para dados não vistos, evitando os fenômenos de *underfitting* e *overfitting*. Com base nos dados apresentados nas **Tabelas** 4.1 e 4.2, é possível apontar as seguintes observações:

Primeiramente, pode-se afirmar que o **subajuste não foi um problema** em nenhum dos experimentos conduzidos. Todos os três modelos, em ambos os conjuntos de dados, alcançaram altíssimos valores de acurácia tanto no treinamento quanto na validação (superiores a 94% em todos os casos de validação). Isso indica que as arquiteturas selecionadas possuíam capacidade e complexidade suficientes para aprender e capturar os padrões intrincados presentes nos dados das folhas de milho.

O controle do **sobreajuste**, por outro lado, foi um dos principais focos das estratégias de treinamento, e os resultados demonstram sua notável eficácia, onde indicador primário de sobreajuste foi minimizado em todos os cenários. Observa-se que a acurácia de treinamento (geralmente 98-99%) e a de validação se mantiveram muito próximas, um forte indicativo de que as técnicas de regularização (como *Data Augmentation*, *Dropout*, *Early Stopping* e Regularização L2) foram bem-sucedidas.

Um comportamento particularmente interessante e positivo foi observado nos valores de perda. Em diversas instâncias, especialmente nos modelos **MobileNetV3-Large** e **EfficientNetV2M**, a perda de validação final foi **inferior à perda de treinamento**. Este fenômeno, embora contraintuitivo à primeira vista, é um sinal de um processo de regularização bem-sucedido. Ele ocorre porque técnicas como *Dropout* e *MixUp* são aplicadas exclusivamente durante o treinamento, tornando a tarefa de otimização artificialmente mais difícil para o modelo. Durante a validação, o modelo utiliza toda a sua capacidade (sem neurônios desativados ou imagens misturadas), resultando em um desempenho mais limpo e, consequentemente, uma perda menor.

Comparativamente, embora todos os modelos tenham demonstrado um excelente controle de sobreajuste, o **EfficientNetV2M** exibiu a maior robustez. No seu melhor desempenho no *dataset* CD&S, a acurácia de validação (**0,9972**) foi praticamente idêntica à de treinamento (0,99), representando um modelo excepcionalmente bem generalizado. Em contrapartida, o MobileNetV3-Large no mesmo *dataset* apresentou uma diferença ligeiramente maior (99% vs. 95,46%), sugerindo uma tendência marginalmente superior ao sobreajuste, ainda que muito bem controlada.

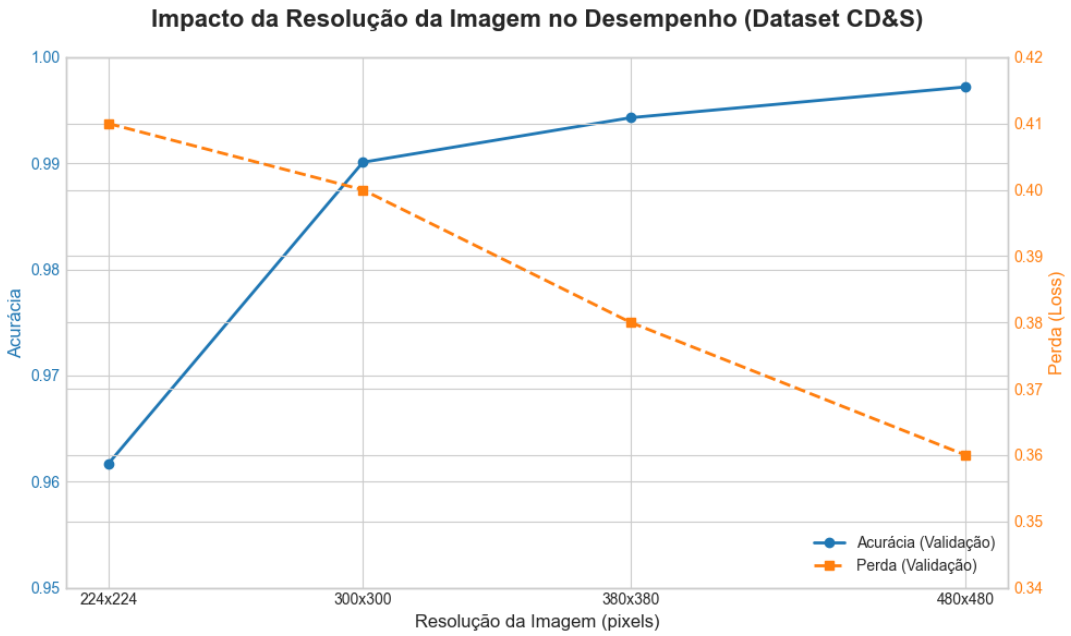
4.4 Análise do Impacto da Resolução da Imagem de Entrada

Conforme a metodologia, foram conduzidos experimentos específicos com o modelo **EfficientNetV2M** para avaliar o impacto da resolução espacial das imagens de entrada no desempenho da classificação. Quatro resoluções distintas foram testadas (*224x224*, *300x300*, *380x380* e *480x480 pixels*) com o objetivo de identificar a configuração ótima e compreender como o modelo utiliza o aumento de detalhes visuais. A análise a seguir é baseada nos resultados obtidos para cada um dos conjuntos de dados.

4.4.1 Desempenho no *Dataset* CD&S

O comportamento do modelo no *dataset* CD&S, conforme ilustrado na **Figura 4.11**, revela uma **correlação positiva e acentuada** entre o aumento da resolução da imagem e a performance de validação.

Figura 4.11: impacto da resolução da imagem no desempenho (*Dataset* CD&S).



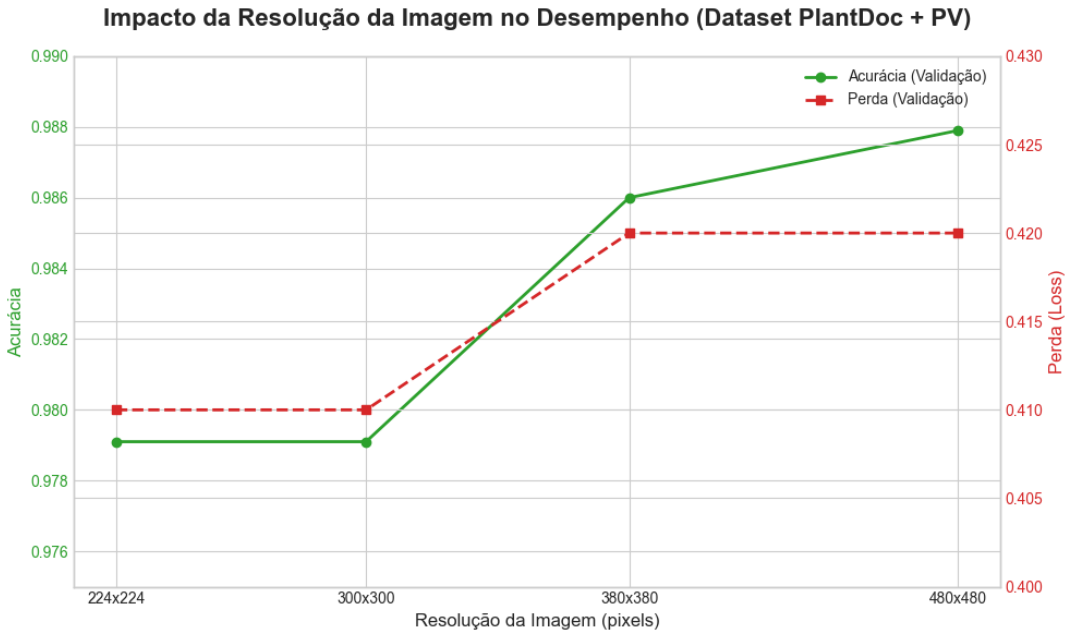
Fonte: elaborada pelo autor.

Observa-se um ganho de desempenho consistente e significativo a cada incremento na resolução. A acurácia de validação saltou de **96,17%** com imagens de *224x224* para um pico de **99,72%** com imagens de *480x480*. De forma complementar, a perda de validação apresentou uma tendência de queda contínua, diminuindo de 0,41 para 0,36. Este comportamento sugere que, para as três classes de doenças presentes neste *dataset*, os detalhes visuais adicionais fornecidos por imagens maiores continham informações discriminativas cruciais, que o modelo EfficientNetV2M foi capaz de extrair e utilizar eficientemente para aprimorar sua fronteira de decisão.

4.4.2 Desempenho no *Dataset* PlantDoc + PlantVillage

Em contrapartida, a análise do desempenho no *dataset* combinado PlantDoc + PlantVillage, apresentada na **Figura 4.12**, exibe uma relação mais complexa e sutil.

Figura 4.12: impacto da resolução da imagem no desempenho (*dataset* PlantDoc + PlantVillage).



Fonte: elaborada pelo autor.

A tendência de melhoria da acurácia com o aumento da resolução ainda é presente, com o valor subindo de **97,91%** para **98,79%**. No entanto, os ganhos são mais marginais em comparação com o *dataset* CD&S, e a performance permaneceu estagnada entre as resoluções de *224x224* e *300x300*.

O comportamento da perda de validação é particularmente notável: ela permaneceu praticamente estável, oscilando entre 0,41 e 0,42, sem uma tendência clara de queda. Este fenômeno sugere que, embora o aumento de detalhes tenha permitido ao modelo corrigir algumas classificações (aumentando a acurácia), ele não resultou em um aumento geral da confiança do modelo ou na redução da magnitude do erro para as amostras mais desafiadoras. Isso indica que, para este *dataset* mais complexo e com maior similaridade visual entre as classes (como ***Blight*** e ***Gray Leaf Spot***), a resolução da imagem é apenas um dos fatores de desempenho, e não o gargalo principal.

Com base nos ganhos de desempenho claros e consistentes observados em ambos os cenários, e especialmente na expressiva melhoria no dataset CD&S, a resolução de ***480x480 pixels*** **foi objetivamente confirmada como a configuração ótima**. Esta escolha maximizou a capacidade do modelo de extrair características relevantes e resultou na mais alta acurácia de validação em todos os experimentos conduzidos.

4.5 Seleção do Melhor Modelo

Após a análise comparativa dos resultados apresentados, esta seção tem como objetivo consolidar os achados e justificar a seleção da arquitetura de melhor desempenho para ser integrada à aplicação *Web* final. A escolha foi baseada em um conjunto de critérios objetivos, priorizando não apenas a acurácia máxima, mas também o equilíbrio entre as métricas de classificação por classe e a robustez geral do modelo.

Com base nas evidências, o modelo **EfficientNetV2M** foi inequivocamente selecionado como a arquitetura de desempenho superior. A principal justificativa quantitativa para esta escolha é o fato de ter alcançado o maior valor de acurácia de validação entre todos os experimentos: **99,72%** no *dataset* **CD&S**, utilizando imagens com resolução de *480x480 pixels*.

Além da acurácia máxima, a análise do *F1-Score* confirmou a superioridade do **EfficientNetV2M**. Com uma média ponderada de *F1-Score* de **0,9972** no *dataset* CD&S e **0,9880** no mais complexo *dataset* **PlantDoc + PlantVillage**, o modelo demonstrou um excelente equilíbrio entre precisão e revocação. Este desempenho foi consistentemente superior ao dos modelos MobileNetV3-Large e ResNet101 em ambos os cenários, indicando uma maior capacidade de generalização.

Os critérios qualitativos, obtidos a partir da análise da matriz de confusão e do relatório de classificação, reforçaram a decisão. Destacam-se dois pontos cruciais:

1. **Confiabilidade na Classe Crítica:** o modelo demonstrou uma performance perfeita na identificação da classe ***Healthy*** (saudável) no *dataset* PlantDoc + PlantVillage, com precisão e revocação de 100%. Para uma aplicação prática, a capacidade de identificar corretamente plantas sadias, minimizando falsos alarmes, é um fator de grande importância;

2. **Escalabilidade com a Resolução:** os experimentos com diferentes tamanhos de imagem mostraram que o **EfficientNetV2M** foi capaz de alavancar o aumento da resolução para melhorar progressivamente seu desempenho, um indicativo de uma arquitetura robusta que extrai eficientemente características detalhadas das imagens.

Portanto, com base na superioridade quantitativa (maior acurácia e *F1-score*) e nas evidências qualitativas de robustez (excelente desempenho na classe **'Healthy'** e escalabilidade com a resolução de imagem), o modelo **EfficientNetV2M**, treinado com a configuração de melhor desempenho, foi o escolhido como a solução final para o problema de classificação de doenças em folhas de milho proposto neste trabalho.

CAPÍTULO 5

Conclusão

Este trabalho abordou o desafio crítico das doenças foliares na cultura do milho, um problema de impacto global que afeta a produtividade agrícola em diversas escalas. O objetivo central foi desenvolver e avaliar, de forma comparativa, um sistema de classificação automática baseado em Redes Neurais Convolucionais (RNCs), visando fornecer uma ferramenta de suporte à decisão que auxilie agricultores na identificação precisa e tempestiva de patologias, possibilitando assim um manejo mais eficiente e sustentável da lavoura.

Para atingir este objetivo, foi construída uma *pipeline* metodológica abrangente. Foram utilizados dois conjuntos de dados públicos, a combinação **PlantVillage + PlantDoc** e o **CD&S**, que passaram por um rigoroso pré-processamento, incluindo o balanceamento de classes via sobreamostragem (*oversampling*). O treinamento das três arquiteturas de RNC selecionadas (**ResNet101**, **MobileNetV3-Large** e **EfficientNetV2M**) foi conduzido sob uma estratégia bifásica de transferência de aprendizado e ajuste fino (*fine-tuning*), controlada por um agendador de taxa de aprendizado dinâmico (*Warmup with Cosine Decay*). Um robusto conjunto de técnicas de regularização, como *Data Augmentation*, *Dropout*, *Early Stopping* e Regularização L2, foi empregado para garantir a generalização dos modelos.

A análise de resultados demonstrou o desempenho superior da arquitetura **EfficientNetV2M**. Este modelo alcançou o mais alto desempenho em ambos os cenários, atingindo uma acurácia de validação de **98.79%** no dataset **PlantVillage + PlantDoc** e um resultado excepcional de **99.72%** no dataset **CD&S**. Este último valor representa um avanço significativo em relação a trabalhos correlatos, como o de Masood et al. (2023), que obteve 97.89% de acurácia no mesmo conjunto de dados, validando a eficácia da pipeline proposta. Adicionalmente, a análise qualitativa revelou a notável capacidade do modelo em identificar corretamente folhas saudáveis, um fator crucial para a confiança em uma aplicação prática.

As principais contribuições deste trabalho podem ser sumarizadas em: (1) o desenvol-

vimento de um modelo de alta performance que supera resultados recentes na literatura para a classificação de doenças do milho; (2) uma análise comparativa aprofundada de três arquiteturas de RNC modernas em dois *datasets* distintos; (3) a validação empírica de que o aumento da resolução da imagem de entrada (até *480x480 pixels*) melhora progressivamente o desempenho do modelo EfficientNetV2M; e (4) a implementação de um protótipo de aplicação *Web* funcional, que demonstra a viabilidade da implantação da solução.

O principal desafio metodológico encontrado foi a limitação e o desbalanceamento dos dados disponíveis, o que tornou necessária a aplicação de técnicas de sobreamostragem. Além disso, a alta similaridade visual entre certas classes, como ***Gray Leaf Spot*** e ***Blight***, representou uma complexidade intrínseca à tarefa de classificação, exigindo um modelo com alto poder discriminativo.

Como trabalhos futuros, sugere-se a expansão dos experimentos, com a exploração de outros hiperparâmetros (como o *batch size*), a avaliação de novas arquiteturas emergentes (por exemplo, *Vision Transformers*) e a aplicação da metodologia em outras culturas agrícolas. Em uma perspectiva de aplicação, o próximo passo seria a implementação do modelo em sistemas embarcados e de borda (*edge computing*), como em drones ou dispositivos móveis, para permitir a análise *in-situ* [1] e em tempo real, notificando o produtor sobre a detecção de patógenos diretamente no campo.

Conclui-se, portanto, que a aplicação de técnicas de aprendizado profundo, quando combinada com uma metodologia robusta de tratamento de dados e treinamento, não apenas é viável para a identificação de doenças foliares do milho, mas também alcança um nível de precisão que pode representar um avanço significativo para as práticas de monitoramento agrícola. Este trabalho evidencia o potencial transformador da inteligência artificial como uma aliada estratégica para o aumento da eficiência, produtividade e sustentabilidade no agronegócio.

[1] Ensaios *in-situ* referem-se a métodos usados para medir as propriedades do solo diretamente em sua localização no solo, sem a necessidade de extrair amostras para análise laboratorial.

Referências Bibliográficas

AGRO, M. *Milho: doenças foliares geram prejuízos à produtividade*. 2022. Acesso em: 07/04/2025. Disponível em: <https://maisagro.syngenta.com.br/dia-a-dia-do-campo/milho-doencas-foliares-geram-prejuizos-na-produtividade/>.

AHMAD, A. et al. *CD&S Dataset*. 2021. Acesso em: 12/11/2025. Disponível em: <https://arxiv.org/abs/2110.12084>.

ALDAKHIL, L.; ALMUTAIRI, A. Multi-fruit classification and grading using a same-domain transfer learning approach. *IEEE Access*, PP, p. 1–1, 01 2024.

BASSOI, L. H. et al. Agricultura de precisão: Um novo olhar na era digital. *Repositório Alice*, 2024. Acesso em: 12/11/2025. Disponível em: <http://www.alice.cnptia.embrapa.br/alice/handle/doc/1169415>.

BELCIC, I.; STRYKER, C. *What is learning rate in machine learning?* 2024. Acesso em: 19/08/2025. Disponível em: <https://www.ibm.com/think/topics/learning-rate>.

BERGMANN, D.; STRYKER, C. *What is a loss function?* 2024. Acesso em: 18/08/2025. Disponível em: <https://www.ibm.com/think/topics/loss-function>.

CASELA, C.; FERREIRA, A.; PINTO, N. *Doenças na cultura do milho In. CRUZ, J.C.; KARAM, D.; MONTEIRO, M.A.R.; MAGALHÃES, P.C. A cultura do milho*. 1. ed. [S.l.]: Embrapa Milho e Sorgo, 2006. 215–257 p.

CHEN, J. et al. Quantification of water inflow in rock tunnel faces via convolutional neural network approach. *Automation in Construction*, v. 123, p. 103526, 03 2021.

CONTINI, E. et al. Milho: caracterização e desafios tecnológicos. *Embrapa Milho e Sorgo*, 2019. Acesso em: 12/11/2025. Disponível em: <https://www.bdpa.cnptia.embrapa.br/consulta/buscab=ad&id=1106547&biblioteca=vazio&busca=1106547&qFacets=1106547&sort=&paginacao=t&paginaAtual=1>.

CROPSCIENCE, B. *Conheça as principais doenças do milho*. 2024. Acesso em: 07/04/2025. Disponível em: <https://www.agro.bayer.com.br/conteudos-impulso-bayer/doencas-milho>.

CRUZ, J. et al. Cultivo do milho. *Embrapa Milho e Sorgo*, Embrapa Milho e Sorgo, 2010. Acesso em: 12/11/2025. Disponível em: <https://ainfo.cnptia.embrapa.br/digital/bitstream/item/27037/1/Plantio.pdf>.

DUTTA, S. *What is Shuffling the Data? A Guide for Students*. 2024. Acesso em: 06/08/2025. Disponível em: <https://medium.com/@sanjay_dutta/what-is-shuffling-the-data-a-guide-for-students-0f874572baf6>.

ELAZIZ, M. E. A. et al. Evolution toward intelligent communications: Impact of deep learning applications on the future of 6g technology. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, v. 14, 11 2023.

ELGENDY, M. *Deep Learning for Vision Systems*. Manning, 2020. Acesso em: 12/11/2025. ISBN 9781617296192. Disponível em: <https://books.google.com.br/books?id=97YCEAAAQBAJ>.

EMBRAPA. *Embrapa Milho e Sorgo*. Brasília, DF: Embrapa, 2020. Acesso em: 20/08/2025. Disponível em: <https://www.embrapa.br/>.

GARDEZI, M. et al. Artificial intelligence in farming: Challenges and opportunities for building trust. *Agronomy Journal*, v. 116, n. 3, p. 1217–1228, 2024. Acesso em: 12/11/2025. Disponível em: <https://acsess.onlinelibrary.wiley.com/doi/abs/10.1002/agj2.21353>.

GÉRON, A. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated, 2019. Acesso em: 12/11/2025. ISBN 9781492032649. Disponível em: <https://books.google.com.br/books?id=OCS1twEACAAJ>.

GHAZAL, S.; MUNIR, A.; QURESHI, W. S. Computer vision in smart agriculture and precision farming: Techniques and applications. *Artificial Intelligence in Agriculture*, v. 13, p. 64–83, 2024. ISSN 2589-7217. Acesso em: 12/11/2025. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2589721724000266>.

GOMES, R. et al. Abordagem sobre pragas e doenças do milho. *Contribuciones a Las Ciencias Sociales*, Contribuciones a Las Ciencias Sociales, 2024. Acesso em: 12/11/2025. Disponível em: <https://ojs.revistacontribuciones.com/ojs/index.php/clcs/article/view/5582>.

GUIMARÃES, L. J. M. Dia nacional do milho - a importância do milho para o agronegócio brasileiro. *Embrapa Milho e Sorgo*, Embrapa Milho e Sorgo, 2024. Acesso em: 12/11/2025. Disponível em: <https://www.embrapa.br/busca-de-noticias/-/noticia/89583335/artigo-dia-nacional-do-milho---a-importancia-do-milho-para-o-agronegocio-brasileiro>.

HE, K. et al. Deep residual learning for image recognition. 2015. Acesso em: 12/11/2025. Disponível em: <https://arxiv.org/abs/1512.03385>.

HUGHES, D. P.; SALATHÉ , M. Plantvillage dataset. *CoRR*, abs/1511.08060, 2015. Acesso em: 12/11/2025. Disponível em: <http://arxiv.org/abs/1511.08060>.

KALRA, D.; BARKESHLI, M. Why warmup the learning rate? underlying mechanisms and improvements. 06 2024.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *University of Amsterdam and University of Toronto*, 2017. Acesso em: 12/11/2025. Disponível em: <https://arxiv.org/abs/1412.6980>.

LAMAS, F. M. Milho: oportunidade para a agricultura de mato grosso do sul. *Embrapa Agropecuária Oeste*, 2022. Acesso em: 12/11/2025. Disponível em: <https://www.embrapa.br/busca-de-noticias/-/noticia/72854764/artigo---milho-oportunidade-para-a-agricultura-de-mato-grosso-do-sul>.

LIU, V. *Mixup: A trivial but powerful image augmentation technique*. 2024. Acesso em: 18/08/2025. Disponível em: <https://medium.com/@lhungting/mixup-a-trivial-but-powerful-image-augmentation-technique-4e2d0725b8e3>.

MASOOD, M. et al. Maizenet: A deep learning approach for effective recognition of maize plant leaf diseases. *IEEE Access*, v. 11, p. 52862–52876, 2023.

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

NYUYTIYMBIY, K. *Parameters and Hyperparameters in Machine Learning and Deep Learning*. 2020. Acesso em: 17/08/2025. Disponível em: <https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac>.

PHAM, B. T. et al. Development of advanced artificial intelligence models for daily rainfall prediction. *Atmospheric Research*, v. 237, p. 104845, 2020. ISSN 0169-8095. Acesso em: 12/11/2025. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169809519311238>.

ROSENBLATT, F. *The Perceptron, a Perceiving and Recognizing Automaton*. Cornell Aeronautical Laboratory, 1957. (Report: Cornell Aeronautical Laboratory). Acesso em: 12/11/2025. Disponível em: <https://books.google.com.br/books?id=P_XGPgAACAAJ>.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.l.]: Prentice Hall, 2010.

SALVADORI, M. F. Aplicação de machine learning para detecção de pragas em plantas. *Universidade Federal do Rio Grande do Sul*, COMGRAD-CCA, 2024.

SAMUELS, J. One-hot encoding and two-hot encoding: An introduction. *Imperial College London*, 01 2024.

SHROFF, M. *Know your Neural Network architecture more by understanding these terms*. 2023. Acesso em: 17/08/2025. Disponível em: <https://medium.com/@shroffmegha6695/know-your-neural-network-architecture-more-by-understanding-these-terms-67faf4ea0efb>.

SILVA, F. da et al. *Inteligência Artificial*. SAGAH, 2023. Acesso em: 12/11/2025. ISBN 9786556904542. Disponível em: <https://books.google.com.br/books?id=qQoF0AEACAAJ>.

SILVA, J. M. P.; CAVICHIOLI, F. A. O uso da agricultura 4.0 como perspectiva do aumento da produtividade no campo. *Revista Interface Tecnológica*, v. 17, n. 2, p. 616–629, dez. 2020. Acesso em: 12/11/2025. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/article/view/1068>.

SINGH, D. et al. Plantdoc: A dataset for visual plant disease detection. In: *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*. New York, NY, USA: Association for Computing Machinery, 2020. (CoDS COMAD 2020), p. 249–253. ISBN 9781450377386. Acesso em: 12/11/2025. Disponível em: <https://doi.org/10.1145/3371158.3371196>.

SUJATHA, R. et al. Advancing plant leaf disease detection integrating machine learning and deep learning. *Scientific Reports*, v. 15, 2025. ISSN 2045-2322. Acesso em: 12/11/2025. Disponível em: <https://www.nature.com/articles/s41598-024-72197-2#citeas>.

SZELISKI, R. *Computer Vision: Algorithms and Applications*. 1st. ed. Berlin, Heidelberg: Springer-Verlag, 2010. Acesso em: 12/11/2025. ISBN 1848829345.

TAN, M.; LE, Q. V. *EfficientNetV2: Smaller Models and Faster Training*. 2021. Acesso em: 12/11/2025. Disponível em: <https://arxiv.org/abs/2104.00298>.

XIAOTENG; LIU; HALIM, P. Understanding gemm performance and energy on nvidia ada lovelace: A machine learning-based analytical approach. *New York University*, 2024. Acesso em: 12/11/2025. Disponível em: <https://arxiv.org/abs/2411.16954>.