

Gabriel Augusto de Melo Poli

**Técnicas de *Machine Learning*
Aplicadas à Classificação de Pragas na
Cultura de Soja**

UBERLÂNDIA

2024

Gabriel Augusto de Melo Poli

Técnicas de *Machine Learning* Aplicadas à Classificação de Pragas na Cultura de Soja

Trabalho de Conclusão de Curso da Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Câmpus Santa Mônica, como requisito para a obtenção do título de Bacharel em Engenharia de Controle e Automação

Universidade Federal de Uberlândia – UFU

Faculdade de Engenharia Elétrica

Orientadora: Profa. Dra. Gabriela Viera Lima

UBERLÂNDIA

2024

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

P766 2024	Poli, Gabriel Augusto de Melo, 2002- Técnicas de Machine Learning Aplicadas à Classificação de Pragas na Cultura de Soja [recurso eletrônico] / Gabriel Augusto de Melo Poli. - 2024. Orientadora: Gabriela Vieira Lima. Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Uberlândia, Graduação em Engenharia de Controle e Automação. Modo de acesso: Internet. Inclui bibliografia. Inclui ilustrações. 1. Processos de fabricação - Automação. I. Lima, Gabriela Vieira, 1990-, (Orient.). II. Universidade Federal de Uberlândia. Graduação em Engenharia de Controle e Automação. III. Título. CDU: 67.02:681.3
--------------	--

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

Agradecimentos

Primeiramente a Deus, pelo dom de minha vida, e por me conduzir e amparar em todos os momentos de dificuldade durante toda minha trajetória;

Aos meus pais, Luciana e César, que me incentivaram a manter-me constante em meus estudos mesmo diante de minha ausência, e forneceram todo suporte necessário para que eu pudesse estar aqui hoje;

A todos meus familiares, em especial minha avó, Maria Zuila, meu irmão João Paulo, minha cunhada Lhais e meu sobrinho João Manoel pelo constante apoio e compreensão;

Aos amigos que fiz durante a graduação e que se tornaram uma parte importante de minha vida;

Aos meus professores, por todos os ensinamentos e correções que me tornaram uma pessoa melhor e me prepararam para a vida profissional;

À minha orientadora, prof. Dra. Gabriela Viera Lima, por me receber e por todo ensinamento, dedicação e compreensão.

Resumo

A evolução da agricultura ao longo dos séculos culminou na Agricultura 4.0, caracterizada pela incorporação de conectividade e automação ao setor. Nesse contexto, os Veículos Aéreos Não Tripulados (VANTs), conhecidos como drones, surgem como ferramentas valiosas para a Agricultura 4.0.

O uso de drones permite o mapeamento eficiente de propriedades, monitoramento remoto de plantações e detecção precoce de problemas. Por meio de técnicas *Machine Learning*, os drones podem identificar pragas e doenças com alta precisão. Essa abordagem oferece uma maneira inovadora de otimizar a produção e reduzir custos.

O objetivo deste trabalho é desenvolver um modelo de identificação baseado em Redes Neurais Artificiais (RNA) capaz de detectar pragas específicas, como lagartas e a vaquinha verde-amarela (*Diabrotica speciosa*), em plantas de soja. O modelo visa auxiliar na tomada de decisão quanto à aplicação de defensivos agrícolas, indicando áreas afetadas e identificando o inseto causador do dano.

Palavras-chave: Agricultura 4.0, Drones, *Machine Learning*, Classificação de Pragas.

Abstract

The evolution of agriculture over the centuries has culminated in Agriculture 4.0, characterized by the integration of connectivity and automation into the sector. In this context, Unmanned Aerial Vehicles (UAVs), commonly known as drones, emerge as valuable tools for Agriculture 4.0.

Drones enable efficient property mapping, remote monitoring of crops, and early detection of issues. Leveraging Machine Learning techniques, drones can accurately identify pests and diseases. This innovative approach optimizes production and reduces costs.

The objective of this study is to develop an Artificial Neural Network (ANN) based identification model capable of detecting specific pests, such as caterpillars and the green-yellow beetle (*Diabrotica speciosa*), in soybean plants. The model aims to assist decision-making regarding the application of agricultural pesticides by identifying affected areas and pinpointing the insect responsible for the damage.

Keywords: Agriculture 4.0, Drones, Machine Learning, Pest Classification.

Lista de Ilustrações

Figura 1: Ilustração da operação de convolução.....	19
Figura 2: Estrutura de uma rede neural convolucional.	20
Figura 3: Convolução entre imagem de entrada 6x6 e filtro 3x3, passo 1.	21
Figura 4: Um mapa de ativação 4x4 é gerado.....	21
Figura 5: Aplicação do <i>zero padding</i>	22
Figura 6: Filtros aplicados ao dígito 4.	22
Figura 7: Convolução de uma imagem colorida.	23
Figura 8: Função ReLU.....	24
Figura 9: Aplicação da função em um mapa de características.	24
Figura 10: Camada de <i>Max-Pooling</i>	25
Figura 11: Representação das camadas completamente conectadas.	25
Figura 12: Ilustração do processo de <i>flattening</i>	26
Figura 13: Curva de generalização que apresenta a perda de treinamento e validação.	27
Figura 14: Comparação entre uma rede sem dropout e com <i>dropout</i> de 50%.	28
Figura 15: Exemplo de imagem gerada pelo algoritmo de aplicação.	37
Figura 16: Modelos com três camadas convolucionais (CC), sem <i>data augmentation</i>	40
Figura 17: Modelos com três camadas convolucionais (CC), com <i>data augmentation</i>	41
Figura 18: Modelos com quatro camadas convolucionais (CC), sem <i>data augmentation</i>	42
Figura 19: Modelos com quatro camadas convolucionais (CC), com <i>data augmentation</i>	42
Figura 20: Modelos com cinco camadas convolucionais (CC), sem <i>data augmentation</i>	43
Figura 21: Modelos com cinco camadas convolucionais (CC), com <i>data augmentation</i>	44
Figura 22: Modelo com três camadas convolucionais (CC), sem <i>data augmentation</i>	45
Figura 23: Modelos com três camadas convolucionais (CC), com <i>data augmentation</i>	45
Figura 24: Modelos com quatro camadas convolucionais (CC), sem <i>data augmentation</i>	46
Figura 25: Modelos com quatro camadas convolucionais (CC), com <i>data augmentation</i>	47
Figura 26: Modelos com cinco camadas convolucionais (CC), sem <i>data augmentation</i>	48
Figura 27: Modelos com cinco camadas convolucionais (CC), com <i>data augmentation</i>	48
Figura 28: Modelos com três camadas convolucionais (CC), com <i>data augmentation</i>	49
Figura 29: Modelos com quatro camadas convolucionais (CC), com <i>data augmentation</i>	50
Figura 30: Modelo com cinco camadas convolucionais (CC), sem <i>data augmentation</i>	51
Figura 31: Modelos com cinco camadas convolucionais (CC), com <i>data augmentation</i>	51
Figura 32: Modelo com três camadas convolucionais (CC), com <i>data augmentation</i>	52
Figura 33: Modelos com quatro camadas convolucionais (CC), com <i>data augmentation</i>	53
Figura 34: Modelos com cinco camadas convolucionais (CC), com <i>data augmentation</i>	54
Figura 35: Modelo com três camadas convolucionais (CC), com <i>data augmentation</i>	55

Figura 36: Modelos com quatro camadas convolucionais (CC), com <i>data augmentation</i>	56
Figura 37: Modelo com três camadas convolucionais (CC), com <i>data augmentation</i>	57
Figura 38: Modelo com três camadas convolucionais (CC), com <i>data augmentation</i>	58
Figura 39: Modelo com três camadas convolucionais (CC), com <i>data augmentation</i>	59
Figura 40: Interface Gráfica para classificação de imagens.	66
Figura 41: A imagem carregada para a interface gráfica e exibida.	66
Figura 42: Apresentação da Classificação e do Nível de Confiança.	67
Figura 43: Imagem rotulada.	67

Lista de Tabelas

Tabela 1: Distribuição completa das camadas genéricas.	34
Tabela 2: Relação entre camadas e modelos.....	35
Tabela 3: Acurácia dos modelos para todas as etapas de treinamento.	63
Tabela 4: Perda dos modelos para todas as etapas de treinamento.....	64

Lista de Abreviaturas e Siglas

AD	<i>Agricultura Digital</i>
AP	<i>Agricultura de Precisão</i>
API	<i>Application Programming Interface (Interface de Programação de Aplicação)</i>
CC	<i>Camada Convolucional</i>
CPU	<i>Central Processing Unit (Unidade de Processamento Central)</i>
DL	<i>Deep Learning (Aprendizagem Profunda)</i>
GPS	<i>Global Positioning System (Sistema Global de Posicionamento)</i>
GPU	<i>Graphics Processing Unit (Unidade de Processamento Gráfico)</i>
IA	<i>Inteligência Artificial</i>
ML	<i>Machine Learning (Aprendizagem de Máquina)</i>
RNA	<i>Rede Neural Artificial</i>
TF	<i>TensorFlow</i>
TPU	<i>Tensor Processing Unit (Unidade de Processamento de Tensor)</i>
VANT	<i>Veículo Aéreo Não Tripulado</i>

Sumário

1. INTRODUÇÃO	12
1.1. JUSTIFICATIVA	13
1.2. OBJETIVO.....	14
1.2.1. <i>Objetivos Específicos</i>	14
2. REFERENCIAL TEÓRICO	15
2.1. AGRICULTURA 4.0.....	15
2.2. DRONES OU VEÍCULOS AÉREOS NÃO TRIPULADOS	16
2.3. INTELIGÊNCIA ARTIFICIAL.....	17
2.3.1. <i>Machine Learning</i>	17
2.3.1.1. Redes Neurais artificiais.....	18
2.3.1.2. Convolução	18
2.3.1.3. Redes Neurais Convolucionais	19
2.3.1.3.1. Camada Convolucional.....	20
2.3.1.3.2. Função de Ativação.....	23
2.3.1.3.3. Pooling	24
2.3.1.3.4. Camadas totalmente conectadas.....	25
2.3.2. <i>Overfitting</i>	26
2.3.3. <i>Dropout</i>	28
2.3.4. <i>Learning rate</i>	28
2.3.5. <i>Função perda</i>	29
2.4. <i>PYTHON</i>	29
2.5. <i>TENSORFLOW</i>	30
2.6. <i>API KERAS</i>	30
3. METODOLOGIA.....	32
3.1. ALGORITMO DE TREINAMENTO.....	32
3.2. MODELOS DESENVOLVIDOS	34
3.3. ALGORITMO DE APLICAÇÃO	36
3.4. INTERFACE GRÁFICA.....	38
4. RESULTADOS E DISCUSSÕES	40
4.1. TREINAMENTO	40
4.1.1. <i>Divergência na curva de perda de validação</i>	59
4.1.2. <i>Oscilações nas métricas de treinamento</i>	60
4.1.3. <i>Oscilações nas métricas de validação</i>	60
4.1.4. <i>Inversão de desempenho em relação aos dados de treinamento e validação</i>	61
5. CONCLUSÃO.....	68
REFERÊNCIAS BIBLIOGRÁFICAS	70
APÊNDICE A	76

1. INTRODUÇÃO

A agricultura, no século XX, ganhou um forte impulso para evolução e melhoria. Na Agricultura 1.0, a principal característica é o uso tração animal nas atividades realizadas no campo. Já a Agricultura 2.0 foi marcada pela substituição da tração animal pelo motor à combustão. No decorrer dos anos, um marco tecnológico possibilitou a criação do *Global Positioning System* (GPS), a partir do emprego desta tecnologia no campo, surge a Agricultura 3.0. Por fim, tem-se a Agricultura 4.0, a revolução que incorporou conectividade e automação ao setor com o uso de drones, robôs etc. (ESPERIDIÃO et al., 2019).

Uma das tecnologias com crescimento acelerado de aplicação nas lavouras são os *Veículos Aéreos Não Tripulados* (VANTs), popularmente conhecidos como drones, surgindo como uma ferramenta valiosa na Agricultura 4.0. A atual fase de desenvolvimento tecnológico favorece e facilita o emprego dos VANTs na área agrícola e em missões de reconhecimento, principalmente pela redução de custos e tamanho dos equipamentos quando comparado ao maquinário convencional, além da necessidade de otimização da produção. O uso de drones como ferramenta auxilia agricultores em tarefas diárias, como mapeamento de pequenas e grandes propriedades de forma facilitada, sem tripulação e controlado remotamente (DOS SANTOS et al, 2019).

Além de facilitar atividades cotidianas, os drones podem fornecer imagens em altas resoluções de porções de terra consideráveis, evidenciando detalhes que não poderiam ser detectados a olho nu. Os VANTs podem ser capazes de monitorar a plantação à distância e de forma rápida e eficaz, delimitar área de plantio de forma a não ultrapassar o limite de área, detectar pragas e doenças, auxiliando os produtores a tomarem decisões eficazes e, conseqüentemente, obterem maior produtividade e reduzirem perdas (DA SILVA, 2022).

As técnicas modernas de detecção com uso de drones possuem alta eficiência e velocidade na identificação de doenças e pragas em plantas, pois são capazes de percorrer vastas áreas em pouco tempo, reduzindo o trabalho manual, muitas vezes limitado à programação da rota a ser percorrida pela aeronave. Para realizar este trabalho, os VANTs obtêm imagens em altas resoluções e utilizam estratégias de *Deep Learning* (DL) e *Machine Learning* (ML) para realizar o tratamento das imagens. Por meio de características extraídas de imagens, como manchas, diferenças de

pigmentação ou formato das folhas, é possível identificar doenças e pragas de forma assertiva (SALVADORI, 2024).

1.1. Justificativa

A soja (*Glycine max*) é uma oleaginosa de origem asiática e a mais produzida e comercializada no mundo. Dentre seus principais derivados estão o óleo e farelo, utilizados tanto para alimentação humana e animal, quanto para produções de biocombustível e outros químicos. O Brasil, juntamente com Estados Unidos da América e a Argentina, são os principais produtores do grão, detendo 80% do total mundial (LEMOS et al., 2017). Para a safra 2023/2024, apesar do aumento da área cultivada em 1,65 milhões de hectares em relação à safra 2022/2023, há expectativas de uma pequena redução (4,5%) na produção nacional de soja em relação à safra 2022/2023 devido ao fenômeno El Niño. Contudo, a produção por hectare ainda se mantém elevada. A estimativa atual é de 147,68 milhões de toneladas de grãos em uma área estimada em 45,73 milhões de hectares, resultando em uma produtividade média de 3.229 kg/ha. A estimativa apresentada indica que 49,98% da produção total de grãos do Brasil é composta pela soja (CONAB, 2024).

Desde o plantio até a colheita, as plantações de soja estão sujeitas a ataques biológicos de patógenos e pragas agrícolas. Conhecer o dano causado por estes agentes na lavoura é importante para realizar a aplicação de defensivos agrícolas de forma otimizada, reduzindo o alto custo do controle químico e impacto ambiental decorrente da aplicação dos defensivos (TETILA, 2019). A aplicação sistemática de agroquímicos é caracterizada por elevadas taxas de desperdício de produtos, mão de obra e energia. Estima-se que grande parte dos custos operacionais para a produção de soja estejam concentrados nos fertilizantes (27,82%), defensivos agrícolas (18,24%), seguidos pelas operações de máquinas (9,10%), sementes (7,35%) e depreciação de maquinário (6,76%) (CONAB, 2016).

Patógenos e pragas estão presentes em muitas regiões produtivas do mundo, acarretando perda de produtividade em várias culturas, como soja e milho. O monitoramento de doenças e pragas em fases iniciais de possíveis epidemias é um aspecto fundamental para proteção da lavoura. Antes da aplicação de qualquer medida defensiva, há a necessidade de realizar a correta identificação do agente causador da contaminação. Um mesmo defensor agrícola pode apresentar diferentes eficácias para espécies semelhantes. No Brasil, até o momento, foram identificadas

mais de 30 espécies de doenças e 40 espécies de insetos causadores de danos à soja (TETILA, 2019).

Programas de controle de doenças e pragas adequados não recomendam a aplicação de defensivos em toda a plantação utilizando uma taxa fixa que normalmente não seguem critérios de manejo. Aplicações em área total e de forma abusiva aumenta os custos de produção, seleção de pragas e patógenos resistentes, diminuição da população de insetos polinizadores e contaminação da água (TETILA, 2019).

1.2. Objetivo

O objetivo deste trabalho é treinar um modelo de identificação em Redes Neurais Artificiais (RNAs) capaz de identificar, através de imagens aéreas ou não, plantas (soja) afetadas por pragas, especificamente lagartas e a vaquinha verde-amarela (*Diabrotica speciosa*), e plantas saudáveis.

1.2.1. Objetivos Específicos

- Desenvolver um programa utilizando a linguagem de programação *Python*, a biblioteca *Tensor Flow* e a *Application Programming Interface* (API) *Keras* para treinar um modelo de RNA capaz de realizar as identificações propostas.
- Treinar e otimizar o modelo de RNA capaz de identificar plantas saudáveis, plantas atacadas por lagartas e plantas atacadas pela vaquinha verde-amarela de forma a atingir maiores níveis de acurácia a um menor custo computacional.
- Desenvolver um programa em *Python* que possa utilizar o modelo de identificação treinado para, de fato, realizar a classificação das imagens fornecidas.
- Desenvolver uma interface homem-máquina para facilitar o uso do programa de classificação de imagens.

2. REFERENCIAL TEÓRICO

Para auxiliar na compreensão do trabalho aqui proposto, a seguir serão apresentadas bases conceituais que englobam aspectos técnicos considerados no desenvolvimento desta pesquisa. O campo no qual a proposta está inserida apresenta uma natureza interdisciplinar. Portanto, ao esclarecer esses elementos, busca-se proporcionar uma melhor compreensão dos fundamentos teóricos do projeto que, além de contribuir na análise objetiva do trabalho proposto, auxiliará na interpretação das contribuições deste para a Agricultura 4.0.

2.1. Agricultura 4.0

O emprego de tecnologias no campo é um fator determinante para que a agricultura alcançasse a sua atual posição. A Agricultura 4.0 representa o emprego e interação de diversas áreas que englobam as tecnologias para a integração de processos agrícolas, favorecendo a redução de incertezas na cadeia produtiva. O termo Agricultura 4.0 surge a partir da Indústria 4.0, originária na Quarta Revolução Industrial. O processo evolutivo da indústria parte da Era Artesanal, passando por máquinas a vapor, máquinas elétricas e uso de computadores nas linhas de produção. E agora, diante do cenário da globalização, tem-se a Indústria 4.0, caracterizada pela fusão entre mundo virtual e o mundo real, representada pelos sistemas *ciberfísicos* e por tecnologias como robótica avançada, como os VANTs e a Inteligência Artificial (IA) (LISBINSKI et al., 2020).

Para compreender melhor a Agricultura 4.0, é necessário elucidar a Agricultura de Precisão (AP), um de seus principais pilares, e que cronologicamente antecedeu a Agricultura 4.0 ou Agricultura Digital (AD). De forma teórica, a AP é um grupo de tecnologias utilizadas de modo que as lavouras e os sistemas de produção agrícolas sejam otimizados, tendo como base o gerenciamento da variabilidade espacial da produção e dos fatores nela envolvidos. No Brasil, a AP é definida como um sistema de gestão agrícola baseado na variabilidade espacial e temporal da unidade produtiva, cujos principais objetivos são o aumento do retorno econômico, a sustentabilidade e a redução de impactos ambientais (VILLAFUERTE et al. 2018).

A Agricultura Digital foi desenvolvida com base nos preceitos da AP, aos quais novas tecnologias foram agregadas, tendo como princípio primordial possibilitar maior conectividade, visando capturar dados do campo e gerar informações úteis para o

produtor, neste sentido, novas tecnologias como Inteligência Artificial, têm ampliado o leque de possibilidades para tornar a agricultura cada vez mais competitiva (VILLAFUERTE et al. 2018). Na Agricultura 4.0, a IA pode ser observada na visão computacional e robótica por meio do uso de VANTs, por exemplo, sendo estes considerados uma tecnologia aplicável na totalidade do processo de produção (OLIVEIRA, 2022).

2.2. Drones ou Veículos Aéreos Não Tripulados

Os Veículos Aéreos Não Tripulados (VANTs) correspondem às aeronaves que podem ser controladas sem a necessidade de um piloto a bordo. Os primeiros drones surgiram nos Estados Unidos da América após a Segunda Guerra Mundial como produto da corrida tecnológica da Guerra Fria. Um dos primeiros modelos desenvolvidos, Aquila, necessitava de 30 pessoas para controlá-lo e voou por alguns minutos, mesmo com uma previsão de autonomia de 20 horas. No Brasil, a história dos drones é marcada pelo BQM1BR, primeiro VANT registrado no país, cuja fabricação foi de responsabilidade da Companhia Brasileira de Tratores (CTB). O protótipo era movido a jatos e destinado a ser utilizado como alvo aéreo e fez seu primeiro voo em 1983. Em ambientes profissionais, os VANTs são utilizados para inúmeros fins, desde vigilância, como proteção de fronteiras, e aplicações agrícolas, incluindo o monitoramento de safras e controle de pragas (DA SILVA, 2022).

Na agricultura, muitos estudos revelam a utilização de drones pela AP, indicando que os avanços tecnológicos e o desenvolvimento de sistemas de navegação a nível global tornaram possível a aplicação de drones em tarefas como plantio, análise de solo, irrigação e monitoramento das plantações (OLIVEIRA, 2022).

A aplicação de drones na agricultura vem ganhando cada vez mais força e mercado, estima-se que 25% do faturamento mundial deste tipo de veículo seja na agricultura. Devido a capacidade de gerar imagens de alta qualidade que podem ser utilizadas em *softwares* e algoritmos, os VANTs podem proporcionar análises dos mais diversos aspectos do plantio, como retenção de água no solo, taxa de crescimento e detecção de plantas mesmo com distâncias relativamente grandes. Assim, embora haja necessidade de implementação e treinamento de *softwares* e equipamentos, os benefícios são variados, desde o aumento do retorno financeiro à segurança e comodidade para os produtores (DA SILVA, 2022).

2.3. Inteligência Artificial

Inteligência Artificial é um campo de pesquisa pertencente a duas grandes áreas, ciências da computação e engenharias. Embora seja reconhecida por uma série de aplicações, grande parte de suas características permanecem desconhecidos, mesmo entre as áreas que mais dedicam tempo ao seu estudo. O objeto da IA é a compreensão da inteligência através de modelos computacionais e o desenvolvimento de sistemas capazes de implementá-los de forma correta. Para isto, esse campo de pesquisa mantém vínculos interdisciplinares com a matemática, psicologia, neurociência, linguística e outras ciências cognitivas (MARQUIS et al., 2020).

A Inteligência Artificial aplicada à agricultura é considerada um ramo tecnológico recente. Máquinas e equipamentos com IA integrada agregam ao sistema agrícola padrões de qualidade, melhorias no desenvolvimento, colheita e comercialização. Soluções embasadas em Inteligência Artificial têm viabilizado o aumento da produção a um menor custo e maior qualidade (OLIVEIRA, 2022).

2.3.1. *Machine Learning*

Machine Learning (ML) é um subconjunto da inteligência artificial (GOOGLE.COM, 2023). O termo foi criado pelo pioneiro da IA Arthur Samuel em 1959. O conceito, segundo ele, descreve um campo de estudo que possibilita os computadores aprenderem mesmo que não tenham sido configurados para tal. Com o advento da Internet, o ML começou a ganhar forma, uma vez que, com tanta informação armazenada na *web*, foi necessário criar um meio para organizar todo o conteúdo disponível. Assim, um dos pilares do ML é justamente a análise de dados com a finalidade de reconhecer padrões (VILLAFUERTE et al. 2018).

O princípio da IA é que uma máquina seja capaz de analisar o ambiente em que está inserida e executar ações embasadas em certa racionalidade para alcançar objetivos específicos. *Machine Learning* é quando esta mesma máquina, de acordo com um conjunto pré-definido de protocolos e procedimentos, seja capaz de melhorar sua capacidade de resolver problema e atingir metas dentro desse ambiente. Quando o sistema recebe uma quantidade crescente de conjuntos similares de dados que podem ser categorizados de forma específica, a capacidade de racionalizar aumenta, possibilitando melhorar a classificação de uma gama de resultados. De forma simplificada, é a capacidade da máquina aprender a partir de dados que elas recebem

e que podem definir padrões a serem seguidos (VILLAFUERTE et al. 2018).

Para este trabalho, o conjunto de dados a ser utilizado é o banco de imagens (*dataset*) disponível em: <https://prod-dcd-datasets-cache-zipfiles.s3.eu-west-1.amazonaws.com/bycbh73438-1.zip>. O banco de imagens possui, inicialmente 6410 imagens divididas em três classes: *Healthy* (saúdavel) com 896 imagens, *Caterpillar* (lagarta), 3309 imagens, *Diabrotica speciosa* com 2205 imagens.

2.3.1.1. Redes Neurais artificiais

Uma rede neural é um modelo de aprendizado de máquina que toma decisões de maneira semelhante ao cérebro humano, utilizando processos que imitam a forma como os neurônios biológicos trabalham em conjunto para identificar fenômenos, avaliar opções e chegar a conclusões (IBM.COM, 2024).

As RNAs são compostas por camadas de neurônios artificiais ou nós:

uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. Cada nó se conecta a outros e possui seu próprio peso e limiar associados. Se a saída de qualquer nó individual estiver acima do valor de limiar especificado, esse nó é ativado, enviando dados para a próxima camada da rede. Caso contrário, nenhum dado é passado para a próxima camada (IBM.COM, 2024).

As RNAs dependem de dados para que possam ser treinadas para poderem melhorar sua precisão à medida que aprendem. Quando ajustadas para precisão, são ferramentas altamente poderosas em ciências da computação e IA que tornam possível a classificação e agrupamento de dados com alta velocidade. Tarefas de reconhecimento, seja de fala ou imagens, podem demandar horas quando realizadas de forma manual por um especialista humano. Ao realizar a mesma tarefa com auxílio das RNAs, o tempo demandado passa a ser de alguns minutos. Pode-se dizer que o algoritmo de RNA mais conhecido é o sistema de buscas do Google (IBM.COM, 2024).

2.3.1.2. Convolução

A convolução é uma operação matemática que combina duas funções, f e g , resultando em uma nova função. Essa nova função representa uma versão modificada de f , influenciada pelas características de g . No contexto de processamento de imagens, onde as imagens são representadas como funções bidimensionais, a convolução é uma ferramenta essencial para diversas tarefas, como detecção de bordas, suavização de ruídos e extração de características relevantes (FERREIRA,

2017).

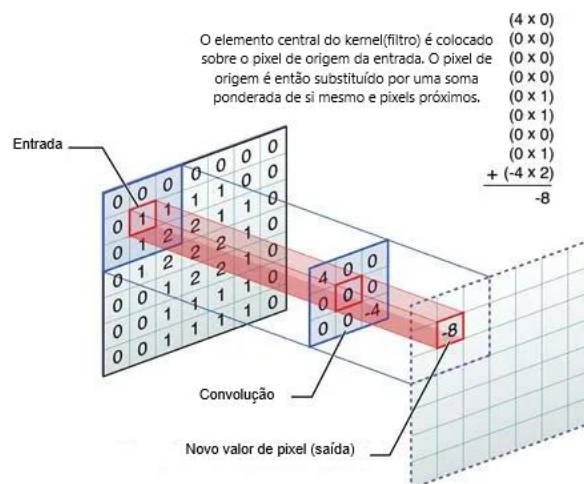
Sejam f e g funções discretas, a convolução é dada por

$$(f * g)(k) = h(k) = \sum_{i=0}^k f(i)g.(k - i)$$

Onde $*$ representa o operador de convolução.

No processamento de imagens, convolução é o processo de aplicar um filtro ou *kernel* (uma matriz de valores) a cada pixel de uma imagem. Esse filtro "desliza" sobre a imagem, calculando um novo valor para o pixel central baseado nos valores dos pixels vizinhos cobertos pelo filtro. Para os pixels da borda, um valor (geralmente zero) é adicionado à imagem para permitir o cálculo da convolução (JESUS; COSTA JR, 2015).

Figura 1: Ilustração da operação de convolução.



Fonte: Basavarajaiah (2019) – adaptado.

2.3.1.3. Redes Neurais Convolucionais

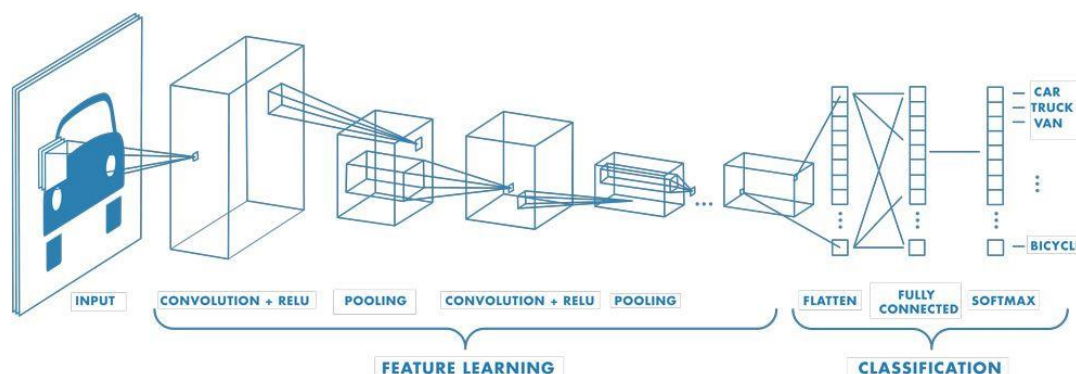
Em meados da década de 2000, o campo do aprendizado de máquina experimentou um avanço significativo com a introdução de técnicas de aprendizado não supervisionado para redes neurais. Essa inovação, liderada por pesquisadores do CIFAR - *Canadian Institute for Advanced Research*, permitiu que as redes neurais aprendessem representações complexas dos dados sem a necessidade de grandes conjuntos de dados rotulados. A aplicação inicial dessa abordagem ao reconhecimento de fala, impulsionada pelo aumento da capacidade computacional das *Graphics Processing Units* (GPUs), demonstrou o potencial das redes neurais para resolver problemas complexos de aprendizado de máquina. Concomitantemente, as Redes Neurais Convolucionais (RNCs) emergiram como uma arquitetura

particularmente bem-sucedida para tarefas de visão computacional, superando as dificuldades associadas ao treinamento de redes neurais tradicionais (FERREIRA, 2017).

Para extrair as características relevantes dos dados, as RNCs se baseiam em três componentes fundamentais: camadas convolucionais, que realizam a extração de padrões locais; camadas de *pooling*, que reduzem a dimensionalidade dos dados e aumentam a invariância a translações; e camadas totalmente conectadas, responsáveis pela classificação final. Qualquer arquitetura de CNN, independentemente de sua complexidade, é construída a partir desses blocos básicos (RODRIGUES, 2018).

A figura 2 apresenta a estrutura atual de uma RNC.

Figura 2: Estrutura de uma rede neural convolucional.

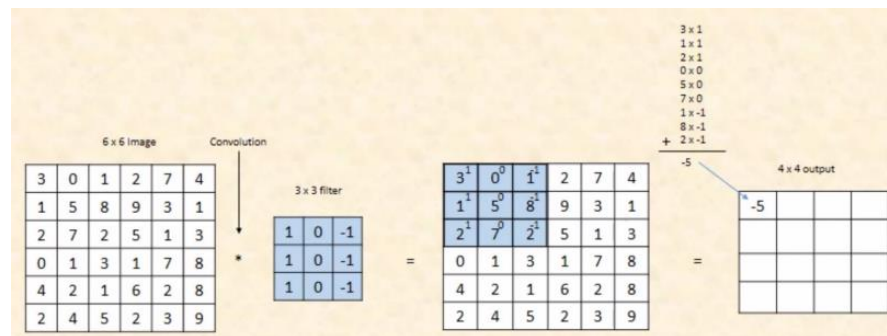


Fonte: MathWorks (2017).

2.3.1.3.1. Camada Convolucional

A camada convolucional (CC) é responsável por identificar padrões relevantes em uma imagem. Ela utiliza filtros de pequeno tamanho que se movem ao longo da imagem, realizando a convolução. Esses filtros são ajustados durante o treinamento para detectar características específicas, como bordas, texturas e cores. À medida que os dados avançam pela rede, os filtros aprendem a reconhecer padrões cada vez mais complexos. No entanto, o aumento do número de filtros e camadas convolucionais pode levar a um aumento significativo no custo computacional (RODRIGUES, 2018). A figura 3 apresenta uma convolução entre uma imagem de entrada 6x6 e filtro 3x3.

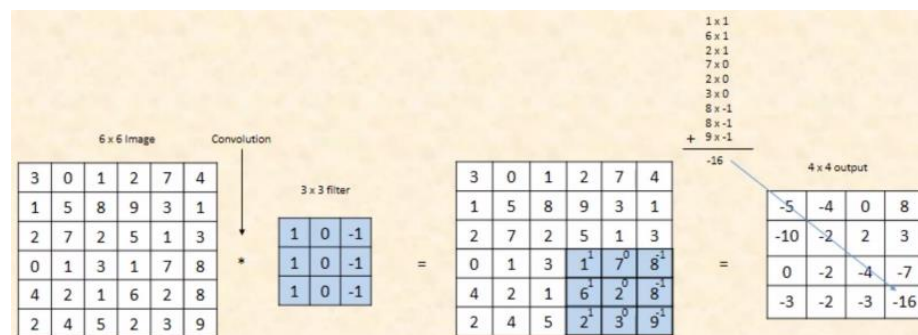
Figura 3: Convolução entre imagem de entrada 6x6 e filtro 3x3, passo 1.



Fonte: (DA CUNHA, 2020).

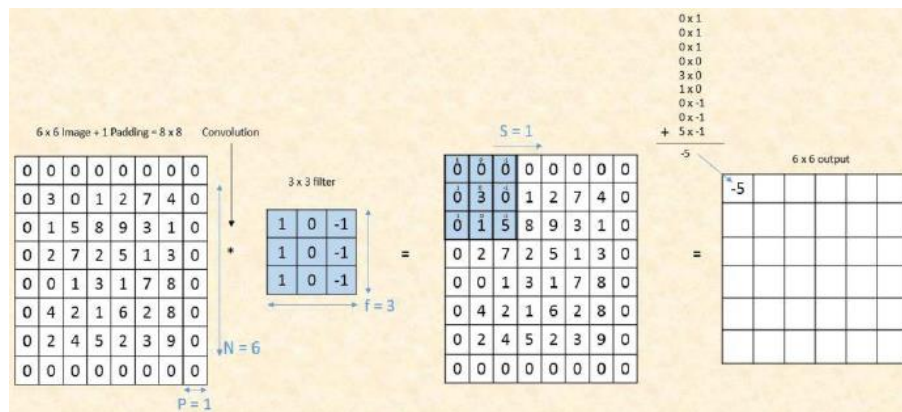
Após a conclusão do processo de convolução, um mapa de características é originado, conforme apresentado na figura 4.

Figura 4: Um mapa de ativação 4x4 é gerado.



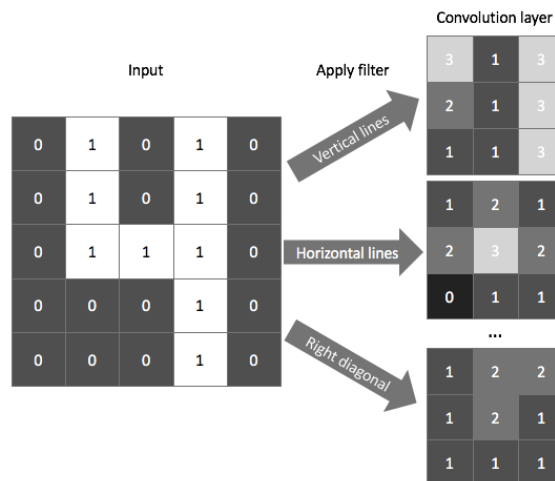
Fonte: (DA CUNHA, 2020).

No exemplo anterior, a imagem de entrada com dimensões 6x6 e filtro com tamanho 3x3, resultou em um mapa de características com tamanho 4x4. Em alguns casos, essa redução de dimensões é indesejada. Para corrigir o encolhimento, adiciona-se à imagem original uma borda preenchida com zeros, chamada *zero padding* (GOODFELLOW; BENGIO; COURVILLE, 2016). A borda *zero padding* impede o encolhimento dos dados durante a convolução, como mostra a figura 5.

Figura 5: Aplicação do zero padding.

Fonte: (DA CUNHA, 2020).

A camada convolucional produz uma imagem de saída que se concentra apenas nas regiões da imagem que exibiram a característica que estava procurando. A figura 6 apresenta filtros projetados para encontrar borda verticais e horizontais além de “diagonal direita” no dígito 4. É possível observar que ao aplicar o filtro de diagonal direita, a imagem de saída é essencialmente escura, informando que o recurso procurado não foi encontrado (JORDAN, 2017).

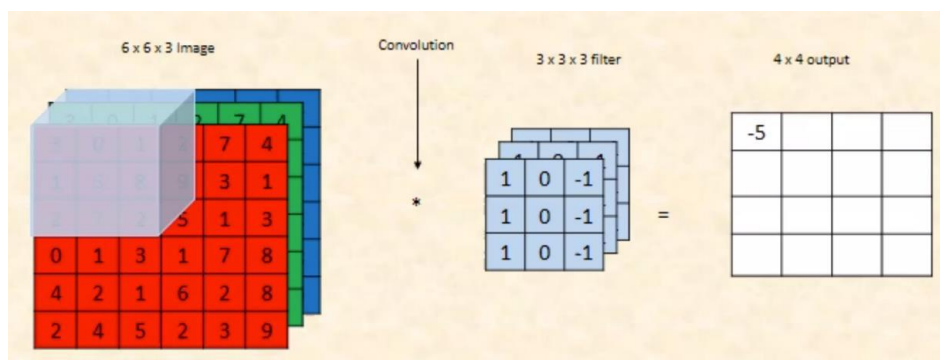
Figura 6: Filtros aplicados ao dígito 4.

Fonte: JORDAN, 2017.

Ao trabalhar com imagens coloridas, a entrada para a convolução não é mais um mapa de intensidade, mas um tensor tridimensional. A terceira dimensão representa os canais de cor (vermelho, verde e azul). Consequentemente, os filtros também possuem três canais, um para cada canal de cor da imagem. A convolução é

realizada entre o campo receptivo da imagem e cada canal do filtro, e os resultados são somados para produzir um único valor no mapa de características de saída (DA CUNHA, 2020). A figura 7 apresenta uma imagem 6x6x3 aplicada a um filtro 3x3x3, onde a última dimensão representa os canais de cor.

Figura 7: Convolução de uma imagem colorida.



Fonte: (DA CUNHA, 2020).

2.3.1.3.2. Função de Ativação

As operações realizadas pela rede até o momento são basicamente lineares uma vez que os cálculos realizados são somas e multiplicações. Dessa forma, é convenção aplicar uma camada não-linear (ou de ativação) após as camadas de convolução, cuja função é introduzir não-linearidade ao sistema. Ao introduzir não-linearidades a um modelo, este torna-se capaz de criar associações com complexidade elevada entre entrada e saída da rede, algo essencial para execução de atividades complexas, como as que envolvem processamento de imagens (GOODFELLOW; BENGIO; COURVILLE, 2016).

Funções de ativação como tangente hiperbólica e sigmóide foram amplamente empregadas no passado para introduzir não-linearidade em redes neurais. Contudo, pesquisas mais recentes demonstraram que a função ReLU – *Rectified Linear Unit* – apresenta vantagens significativas, permitindo um treinamento mais eficiente computacionalmente e, muitas vezes, resultando em modelos com maior precisão. Essa superioridade da ReLU a tornou a função de ativação mais popular em redes neurais modernas (KRIZHEVSKY; SUTSKEVER; HINTON, 2017; DESHPANDE, 2016).

A figura 8 apresenta a função ReLU ou unidade linear retificada aplica a função $f(x) = \max(0, x)$, introduzindo um conceito conhecido como esparsidade em redes neurais. Isso ocorre porque, ao definir a saída como zero para entradas negativas, a

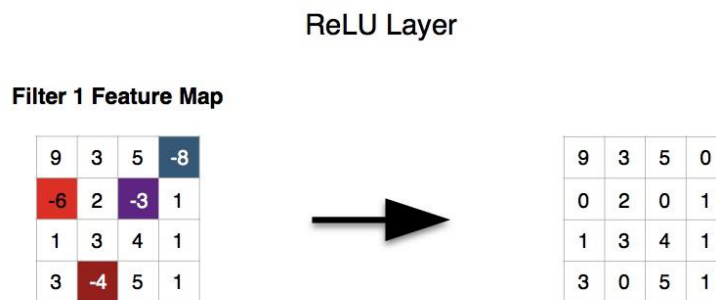
ReLU desativa uma parcela significativa dos neurônios em cada camada, como exemplifica a figura 9. Essa característica torna a rede mais eficiente, pois menos cálculos são necessários durante o treinamento e a inferência. Além disso, a esparsidade pode ajudar a regularizar o modelo, evitando o *overfitting* (Data Science Academy, 2022).

Figura 8: Função ReLU



Fonte: GÖRNER, 2018.

Figura 9: Aplicação da função em um mapa de características.



Fonte: ANIEMEKA, 2017.

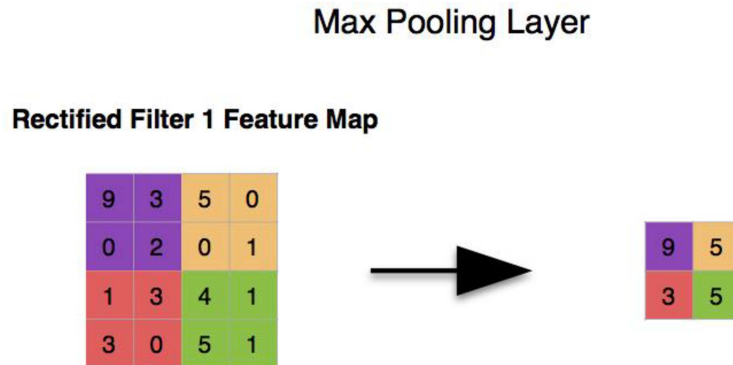
2.3.1.3.3. Pooling

Completando a arquitetura das RNC, além das camadas convolucionais, as redes neurais convolucionais utilizam as camadas de *pooling*. Elas reduzem a resolução espacial dos mapas de características, preservando apenas as informações mais relevantes. Essa redução de dimensionalidade contribui para diminuir o número de parâmetros da rede e torna o modelo mais resistente a pequenas variações na entrada (Data Science Academy, 2022).

As camadas de *pooling* atuam como um filtro que condensa as informações extraídas pelas camadas convolucionais. Cada unidade da camada de *pooling* pode resumir uma região da camada anterior. Na figura 10, cada unidade da camada de *pooling* resume uma região 2x2 neurônios (quadrados coloridos) da camada anterior. A técnica aplicada é a de *max-pooling*, que seleciona o valor máximo dentro de cada

uma dessas regiões e o passa para a próxima camada, assim, a saída produzida contém o maior valor presente na região analisada (Data Science Academy, 2022).

Figura 10: Camada de *Max-Pooling*.



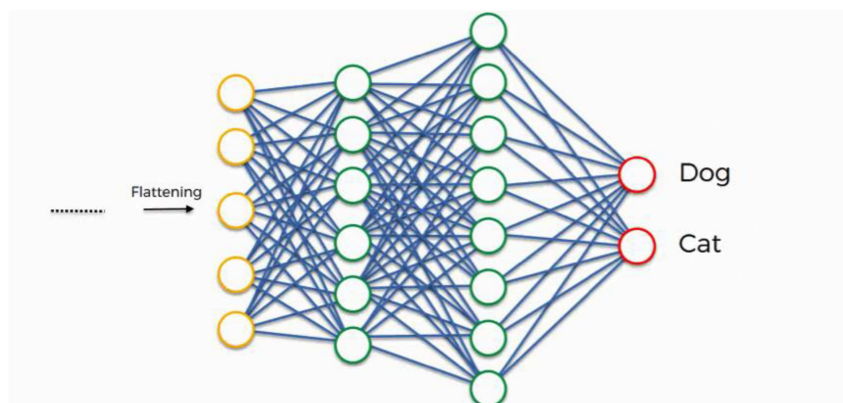
Fonte: ANIEMEKA, 2017 – adaptado.

2.3.1.3.4. Camadas totalmente conectadas

A camada totalmente conectada (TC) atua como o centro de tomada de decisão da rede neural convolucional. Enquanto as camadas anteriores extraem características como bordas, texturas e formas, a camada totalmente conectada integra essas informações para realizar a classificação final da imagem, escolhendo a classe que melhor se ajusta aos dados de entrada (DESHPANDE, 2016).

A camada totalmente conectada integra todas as características e atributos obtidos nas camadas anteriores por meio de uma estrutura neural tradicional de camadas ocultas, contudo, todos os neurônios da camada estão interconectados (DA CUNHA, 2020) como apresenta a figura 11.

Figura 11: Representação das camadas completamente conectadas.

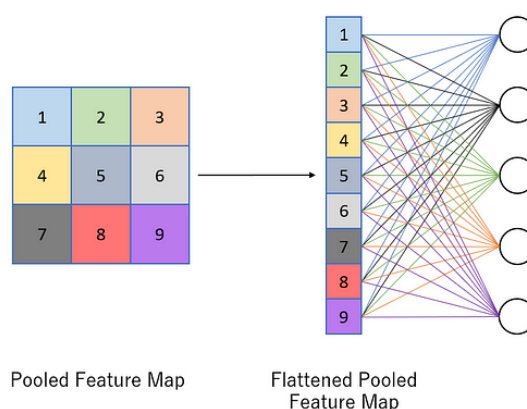


Fonte: SuperDataScience, 2018.

Para que a saída das camadas convolucionais e de *pooling* possa ser

processada pela camada totalmente conectada, é necessário um passo intermediário: o achatamento ou *flattening*. Nesse processo, as matrizes que representam os mapas de ativação são "achatadas" em um único vetor unidimensional, conforme figura 12. Essa transformação permite que todos os neurônios da camada totalmente conectada estejam conectados a todos os elementos do vetor, facilitando a integração de todas as informações extraídas da imagem (JORDAN, 2017).

Figura 12: Ilustração do processo de *flattening*.



Fonte: Aramendia, 2024.

A camada totalmente conectada realiza uma análise semântica profunda das saídas das camadas anteriores (que representam os mapas de ativações de características de alto nível de abstração) buscando padrões que correspondam às diferentes classes. A título de ilustração, se ao classificar uma imagem como cão, ela possivelmente obteve altos valores nos mapas de ativação associados a características como pata ou quatro pernas. À luz dessa perspectiva, se a rede classificou a imagem como pássaro, certamente os mapas de ativação que representam bicos ou asas obtiveram valores altos (DESHPANDE, 2016).

Por fim, para que a classificação final seja dada em porcentagem, a função *softmax* pode ser usada. Ela normaliza as saídas da rede, garantindo que a soma de todas as probabilidades seja igual a 1. Isso permite que interpretemos as saídas como uma distribuição de probabilidade categórica, onde cada valor representa a confiança da rede naquela classe (JORDAN, 2017).

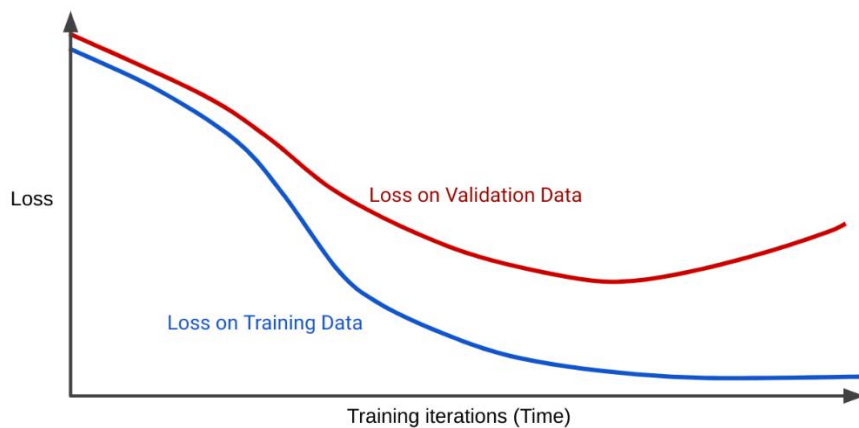
2.3.2. *Overfitting*

Overfitting é um problema em aprendizado de máquina que ocorre quando um modelo se ajusta excessivamente aos dados de treinamento, capturando até mesmo

ruídos e particularidades aleatórias. Consequentemente, o modelo perde a capacidade de generalizar para novos dados, resultando em um desempenho pobre em situações reais. É como ajustar uma curva a cada ponto de um gráfico, criando uma linha extremamente complexa que não representa a tendência geral dos dados (GOOGLE.COM, 2024).

O *overfitting* pode ser detectado por meio das curvas de perda (o quão longe a previsão está da realidade) de um modelo em relação ao número de iterações de treinamento. A figura 13 apresenta uma generalização de curva, ou seja, um gráfico que apresenta duas ou mais curvas de perda (GOOGLE.COM, 2024).

Figura 13: Curva de generalização que apresenta a perda de treinamento e validação.



Fonte: GOOGLE.COM, 2024.

Inicialmente, as duas curvas apresentam comportamento semelhante e, depois, divergem. Ao passo que, após certo número de iterações, a perda para o conjunto de treinamento diminui ou permanece estável, a perda para o conjunto de validação aumenta. Isso sugere um *overfitting*. Por outro lado, uma curva de generalização para um modelo bem ajustado mostra duas curvas de perda com comportamentos semelhantes (GOOGLE.COM, 2024).

As principais causas do *overfitting* estão relacionadas ao tamanho reduzido do conjunto de treinamento, limitando a capacidade do modelo aprender padrões gerais e robustos; presença de ruídos nos dados, levando o modelo a aprender padrões espúrios; treinamento prolongado, pois o modelo começa a memorizar os exemplos de treinamento e; alta complexidade do modelo, uma vez que podem capturar padrões complexos e irrelevantes nos dados (AMAZON.COM, 2022).

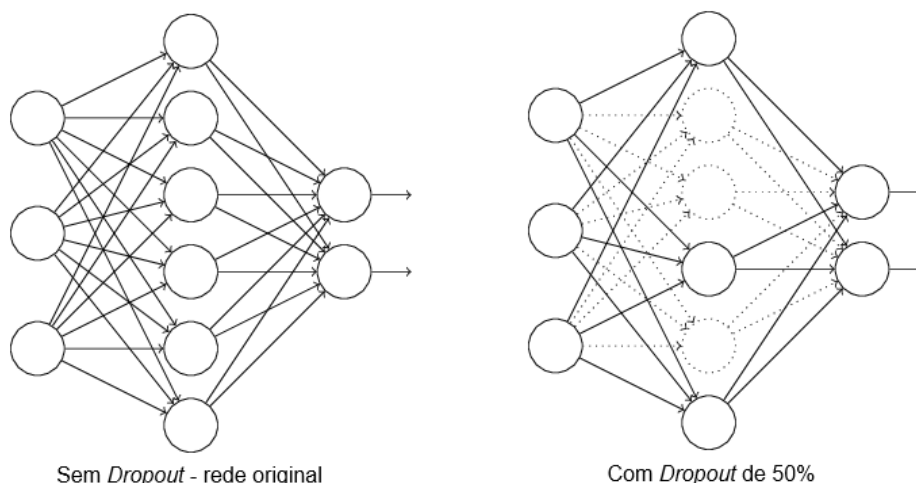
Para evitar o sobreajuste pode-se utilizar algumas estratégias, como a parada

antecipada, onde o treinamento é encerrado antes que o modelo aprenda os ruídos nos dados; a poda, que consiste na identificação dos recursos mais importantes dentro do conjunto de treinamento e elimina os irrelevantes; o aumento de dados (*Data augmentation*), que realiza pequenas alterações nos dados de amostrada a cada iteração do modelo. Se feito com moderação, o aumento de dados faz com que os conjuntos de treinamento pareçam exclusivos, impedindo que o modelo aprenda suas características. Em imagens, podem ser aplicadas rotações, translações, inversões etc. (AMAZON.COM, 2022).

2.3.3. Dropout

O *dropout* é uma técnica de regularização que consiste na desativação aleatória de uma porcentagem de neurônios em cada iteração do treinamento de uma rede neural. Ao desativar neurônios de forma aleatória, força-se a rede a aprender representações mais robustas pois diversifica o treinamento, dificultando que o modelo memorize dados, além de reduzir a dependência das representações em relação aos neurônios individuais, o que contribui para reduzir o *overfitting* e melhorar a generalização do modelo. A figura 14 esboça uma representação técnica do *dropout*.

Figura 14: Comparação entre uma rede sem dropout e com *dropout* de 50%.



Fonte: Data Science Academy, 2022 - adaptado.

2.3.4. Learning rate

Os pesos são o cerne do aprendizado de máquina. São eles que permitem que um modelo, inicialmente "cru", se transforme em uma ferramenta poderosa para tomar decisões baseadas em dados. Ao atribuir diferentes níveis de importância a cada entrada, os pesos definem a representação interna dos dados no modelo, permitindo

que ele capture padrões complexos e não lineares (CYPRIANO, 2024).

A taxa de aprendizado ou *learning rate* é um hiperparâmetro crucial em diversos algoritmos de otimização utilizados em *machine learning*. Ela determina o tamanho dos passos que um algoritmo dá na direção de minimizar a função de perda. Essa função de perda quantifica o erro entre as previsões do modelo e os valores reais. Ao ajustar os parâmetros do modelo, como os pesos, o algoritmo utiliza a taxa de aprendizado para determinar a magnitude da atualização em cada iteração. Uma taxa de aprendizado muito pequena pode levar a um treinamento lento, pois o modelo fará ajustes muito graduais. Por outro lado, uma taxa de aprendizado muito grande pode causar instabilidade, impedindo que o modelo convirja para um mínimo da função de perda (GOOGLE.COM, 2024).

2.3.5. Função perda

As funções de perda são funções matemáticas que mapeiam a distância entre as saídas previstas por um modelo e os valores reais. Ao minimizar essa função, o algoritmo de aprendizado busca os parâmetros do modelo que melhor se ajustam aos dados de treinamento. A escolha da função de perda depende do tipo de problema (regressão, classificação) e dos objetivos específicos da aplicação (GEEKSFORGEEKS, 2024). Para a aplicação aqui proposta, a função de perda utilizada é a *Categorical Cross Entropy Loss* ou Perda de Entropia Cruzada Categórica, cuja aplicação são algoritmos de classificação.

Em tarefas de classificação, os modelos de aprendizado de máquina produzem uma distribuição de probabilidades para cada exemplo, indicando a probabilidade de ele pertencer a cada classe. A entropia cruzada quantifica a divergência entre essa distribuição de probabilidades prevista e a distribuição verdadeira, que é uma distribuição categórica com valor 1 para a classe correta e 0 para as demais (GEEKSFORGEEKS, 2024).

2.4. Python

Python é uma linguagem de programação de altíssimo nível (VHLL – *Very High Level Language*). Buscando o ideal “programação de computadores para todos”, seu criador, o holandês Guido Van Rossum, desenvolveu uma linguagem de programação gratuita, de código fonte aberto e compatível com inúmeros sistemas operacionais, como *Windows*, *Linux*, *Mac* e até mesmo *smartphones* (LAMBERT, 2022).

O *Python* é uma linguagem de programação que pode ser aplicada no desenvolvimento de inúmeras soluções, científicas ou não. A linguagem possui suporte para integração com diversas linguagens e ferramentas, além de ser distribuído com uma extensa biblioteca nativa. Sua sintaxe é clara e de fácil compreensão. O uso do *Python* é frequentemente associado ao aumento de produtividade e à produção de programas de alta qualidade. Por fim, as principais características do *Python* são: multiplataforma (pode ser instalado em qualquer plataforma computacional), portabilidade (aplicações desenvolvidas em *Python* podem ser facilmente distribuídas para várias plataformas diferentes da original), *software* livre (não há limitações à distribuição gratuita ou venda de programas), extensibilidade (pode ser estendido através de módulos escritos em *Python* ou rotinas em outras linguagens), orientado a objetos, (tudo nesta linguagem é um objeto, desde funções até mesmo outros programas escritos em *Python*) (COELHO, 2007).

2.5. TensorFlow

TensorFlow (TF) é um *framework* de código aberto desenvolvido pelo *Google Brain* utilizando as linguagens *Python* e *JavaScript*. Ele auxilia no desenvolvimento de ML, podendo ser executado sobre diversas plataformas e arquiteturas, incluindo *Central Processing Unit* (CPU), GPU e, mais recente, *Tensor Processing Unit* (TPU). Atualmente, é uma das principais ferramentas disponíveis no mercado para criação de redes neurais *deep learning*, sendo possível agilizar e facilitar o processo de obtenção de dados, treinamento de modelos, realizar previsões e refinar resultados futuros (TENSORFLOW, [s.d.]).

A execução do TF ocorre sobre uma aplicação de alta performance escrita em C e C++, unindo esta característica às facilidades do *Python*. Sua arquitetura se divide em três partes principais: pré-processamento de dados, construção de modelos, treinamento e estimativas do modelo criado. O maior benefício que o TF tem a oferecer ao desenvolvimento de sistemas inteligentes é a abstração, permitindo que o desenvolvedor se concentre na lógica geral da aplicação, não se preocupando com detalhes básicos da implementação de algoritmos ou como ligar saída de uma função a entrada de outra, por exemplo (FALCÃO et al., 2019).

2.6. API Keras

Keras é uma API de alto nível (*high-level*) destinada a construção e treinamento

de modelos de aprendizagem profunda. Utilizada para prototipagem rápida, pesquisa avançada e produção a API conta com três vantagens principais: amigável, modular e combinável, fácil de estender. O *Keras* possui uma interface simples e fácil de usar e um mecanismo de *realimentação* de erros, facilitando o desenvolvimento por parte do programador. Além disso, a API provê uma variedade de recursos e métricas que podem auxiliar na otimização das RNAs, além de fornecer métricas quanto a performance e precisão dos modelos criados (FALCÃO et al., 2019).

3. METODOLOGIA

A metodologia empregada neste trabalho seguiu uma abordagem estruturada em quatro fases. Na primeira fase, foi realizada a implementação de um código em *Python* com o intuito de criar um ambiente de treinamento para o algoritmo de RNC. A segunda fase envolveu o processo de treinamento propriamente dito, no qual foram ajustadas características do modelo e aplicadas técnicas de otimização para maximizar sua precisão. Na terceira fase, foi desenvolvido um código para a execução do modelo treinado em novos dados, permitindo a obtenção de previsões e a avaliação do desempenho final. Por fim, com o objetivo de tornar o sistema mais amigável, foi construída uma interface gráfica em *Python*, que possibilita a interação do usuário de forma intuitiva e eficiente.

Na sessão 3.1 será apresentada a estrutura básica do algoritmo de treinamento utilizada para treinar os modelos de RNC. A seção 3.2 explorará os modelos de redes neurais criados. A seção 3.3 abordará o algoritmo utilizado para a execução do modelo. Por fim, o tópico 3.4 explanará a estrutura básica do código de interface gráfica.

3.1. Algoritmo de treinamento

A biblioteca *TensorFlow* possui um grande ecossistema de aprendizagem em sua plataforma *online*, possuindo desde tutoriais simples a exemplos prontos disponibilizados por meio do *Google Colab* (plataforma de programação *online* desenvolvida e mantida pelo *Google*). Como ponto de partida, o exemplo *images.ipynb* (*TensorFlow*, 2020) foi utilizado.

O algoritmo adaptado constrói, treina e avalia múltiplos modelos de classificação de imagens. Ele carrega o banco de imagens disponível utilizado para treinamento e validação, divide-o em conjuntos de treinamento e validação e aplica aumento de dados para melhorar a generalização do modelo. O código também utiliza *dropout* para reduzir o *overfitting*.

O código itera sobre diferentes taxas de *dropout* e números de camadas convolucionais para criar e treinar vários modelos. Além das camadas convolucionais, os modelos possuem uma camada de achatamento, duas camadas completamente conectadas e uma camada de saída. Para cada modelo, ele registra a precisão e a perda do treinamento e da validação, salva os pesos do modelo da melhor época e

gera gráficos do desempenho do modelo.

Após o treinamento, o código carrega determinados modelos salvos e os retreina usando o mesmo conjunto de dados e configurações. Tanto para o primeiro treinamento quando para os demais, ele gera gráficos do desempenho do modelo e salva os dados em arquivos .csv.

De forma detalhada:

a) Setup e Carregamento dos Dados:

O código começa importando as bibliotecas necessárias, como *TensorFlow*, *keras*, *Matplotlib*, *NumPy* e *Pandas*. Em seguida, define a URL do conjunto de dados de imagens, realiza seu *download* e extraí-lo. Após baixar os dados, antes de dar continuidade aos treinamentos, foram removidas, de forma aleatória, seis imagens de cada classe para serem utilizados em testes futuros. O conjunto de dados é carregado e dividido em conjuntos de treinamento (80%) e validação (20%). Além disso, o tamanho dos lotes de imagens e as dimensões das imagens são definidas. Por fim, o código define nomes das classes presentes no conjunto de dados.

b) Pré-processamento de Dados:

As imagens são padronizadas para o intervalo [0, 1] usando, o que é benéfico para o treinamento de redes neurais.

c) Construção e Treinamento do Modelo:

Para modelos que utilizam *data augmentation* para reduzir o *overfitting*, algumas manipulações são aplicadas às imagens, como espelhamento horizontal, rotação, *zoom* e ajustes de contrastes aleatórios, além de translações.

Outra técnica para evitar *overfitting* é o *dropout*, que desativa aleatoriamente neurônios durante o treinamento, forçando a rede a aprender características mais robustas. O *dropout* varia de 0 a 40%.

Para a construção do modelo propriamente dita, foi desenvolvida uma função cujos parâmetros são o número de camadas convolucionais e a taxa de *dropout*. O modelo consiste em camadas convolucionais (*Conv2D*), camadas de *pooling* (*MaxPooling2D*), camada de achatamento (*Flatten*), *dropout*, camadas

densas (*Dense*) ou camadas completamente conectadas e uma camada de saída.

A compilação se dá após a definição do otimizador *adam*, a função de perda *SparseCategoricalCrossentropy* e a métrica *accuracy* utilizada para otimizar o modelo.

Para treinar e salvar o modelo, uma nova função foi criada. Ela realiza o treinamento e validação do modelo utilizando seus respectivos dados durante 20 épocas. A cada época, a função verifica a precisão de validação, caso a precisão atual seja melhor que a anterior o modelo atual é salvo. Após separar manualmente os modelos que atingiram a precisão mínima estabelecida, estes são submetidos a uma nova fase de treinamento que mantém a configuração inicial de treinamento.

d) Visualização e Salvamento dos Resultados:

Após o treinamento e retreinamento, o código gera gráficos da precisão e perda do treinamento e validação. Os gráficos são salvos como imagens .png. e seus respectivos dados são salvos em arquivos .csv usando *Pandas*.

3.2. Modelos desenvolvidos

Com o intuito de obter um modelo de rede neural com alta precisão, foram desenvolvidos 30 modelos iniciais. As variações entre modelos podem ser observadas na presença ou não de *data augmentation*, porcentagem de *dropout* e quantidade de camadas convolucionais. A tabela 1 apresenta a estrutura completa da rede neural.

Tabela 1: Distribuição completa das camadas genéricas.

Camada	Tipo	Parâmetros	Ativação	Descrição
1	<i>RandomFlip</i>	<i>horizontal</i>	-	Espelha a imagem horizontalmente
2	<i>RandomRotation</i>	0.2	-	Rotaciona a imagem aleatoriamente em 20%
3	<i>RandomZoom</i>	0.2	-	Aplica zoom aleatório de 20% na imagem
4	<i>RandomContrast</i>	0.2	-	Ajusta o contraste da imagem aleatoriamente em 20%
5	<i>RandomTranslation</i>	0.1, 0.1	-	Translada a imagem aleatoriamente em 10% na horizontal e vertical
6	<i>Rescaling</i>	1./255	-	Padroniza os valores dos pixels para o intervalo [0, 1]
7	<i>Conv2D</i>	16 filtros, <i>kernel</i> 3x3, <i>padding</i> 'same'	<i>ReLU</i>	Extraí características da imagem
8	<i>MaxPooling2D</i>	<i>pool size</i> 2x2	-	Reduz a dimensionalidade da imagem

9	Conv2D	32 filtros, kernel 3x3, padding 'same'	ReLU	Extraí características da imagem
10	MaxPooling2D	pool size 2x2	-	Reduz a dimensionalidade da imagem
11	Conv2D	64 filtros, kernel 3x3, padding 'same'	ReLU	Extraí características da imagem
12	MaxPooling2D	pool size 2x2	-	Reduz a dimensionalidade da imagem
13	Conv2D	128 filtros, kernel 3x3, padding 'same'	ReLU	Extraí características da imagem
14	MaxPooling2D	pool size 2x2	-	Reduz a dimensionalidade da imagem
15	Conv2D	256 filtros, kernel 3x3, padding 'same'	ReLU	Extraí características da imagem
16	MaxPooling2D	pool size 2x2	-	Reduz a dimensionalidade da imagem
17	Dropout	0.0 a 0.4	-	Desativa aleatoriamente neurônios durante o treinamento
18	Flatten	-	-	Transforma a matriz em um vetor
19	Dense	256 neurônios	ReLU	Camada totalmente conectada
20	Dense	128 neurônios	ReLU	Camada totalmente conectada
21	Dense	número de classes (3)	Softmax	Camada de saída com número de neurônios igual ao número de classes

Fonte: Elaborado pelo autor.

A tabela 1, além de apresentar as 21 possíveis camadas dos modelos desenvolvidos, descreve os parâmetros para cada camada, a função de ativação aplicada a cada uma e uma breve descrição. As camadas “Conv2D” necessitam do parâmetro “padding 'same'” para parametrizar a função de modo que as bordas da imagem sejam preenchidas com zeros, mantendo seu tamanho original após a operação de convolução. A tabela 2 indica quais camadas estão presentes em cada um dos 30 modelos. Na coluna da camada 13, a indicação de presença ou não da camada foi substituída pelo parâmetro que descreve a taxa de *dropout*.

Tabela 2: Relação entre camadas e modelos.

Dropout			Canada																					
			CC																					
Sem Data Augmentation	0%		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	
		1	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	0.0	x	x	x	x
		2	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	-	-	0.0	x	x	x	x
	10%	3	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	x	x	0.0	x	x	x	x
		1	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	0.1	x	x	x	x
		2	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	-	-	0.1	x	x	x	x
	20%	3	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	x	x	0.1	x	x	x	x
		1	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	0.2	x	x	x	x
		2	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	-	-	0.2	x	x	x	x
	30%	3	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	x	x	0.2	x	x	x	x
		1	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	0.3	x	x	x	x
		2	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	-	-	0.3	x	x	x	x
4	3	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	x	x	0.3	x	x	x	x	
4	1	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	0.4	x	x	x	x	

Com Data Augmentation		2	-	-	-	-	-	x	x	x	x	x	x	x	x	x	-	-	0.4	x	x	x	x
		3	-	-	-	-	-	x	x	x	x	x	x	x	x	x	x	x	0.4	x	x	x	x
	0%	1	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	0.0	x	x	x	x
		2	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	0.0	x	x	x	x
		3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0.0	x	x	x	x
	10%	1	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	0.1	x	x	x	x
		2	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	0.1	x	x	x	x
		3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0.1	x	x	x	x
	20%	1	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	0.2	x	x	x	x
		2	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	0.2	x	x	x	x
		3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0.2	x	x	x	x
	30%	1	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	0.3	x	x	x	x
		2	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	0.3	x	x	x	x
		3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0.3	x	x	x	x
	40%	1	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	-	0.4	x	x	x	x
		2	x	x	x	x	x	x	x	x	x	x	x	x	x	-	-	-	0.4	x	x	x	x
		3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0.4	x	x	x	x

Fonte: Elaborado pelo autor.

O desempenho detalhado de cada modelo será apresentado no tópico “4. Resultados”.

3.3. Algoritmo de aplicação

Após o treinamento dos modelos e a seleção daquele com melhor desempenho, é necessário criar um algoritmo que seja capaz de utilizar e testar o modelo e realizar previsões para novos dados. Dessa forma, utilizando como base o exemplo *images.ipynb* (TensorFlow, 2020), uma codificação em *Python* foi desenvolvida.

O *script* carrega um modelo de classificação de imagens treinado e o utiliza para prever a classe de novas imagens. Ele percorre as imagens em um diretório, carrega cada imagem, redimensiona e a converte em um *array*. Em seguida, o modelo faz uma previsão e a pontuação é calculada. O código então exibe a imagem original com sua classe real e a imagem com a classe prevista e a pontuação. Por fim, ele salva a figura e a exibe.

A seguir, a estrutura aprofundada da aplicação é apresentada.

a) Carregamento do Modelo e Definição de Parâmetros:

O código inicia importando as bibliotecas necessárias e em seguida, define o caminho para o modelo salvo e o carrega. Por fim, estabelece as dimensões das imagens de entrada e os nomes das classes que o modelo foi treinado para reconhecer.

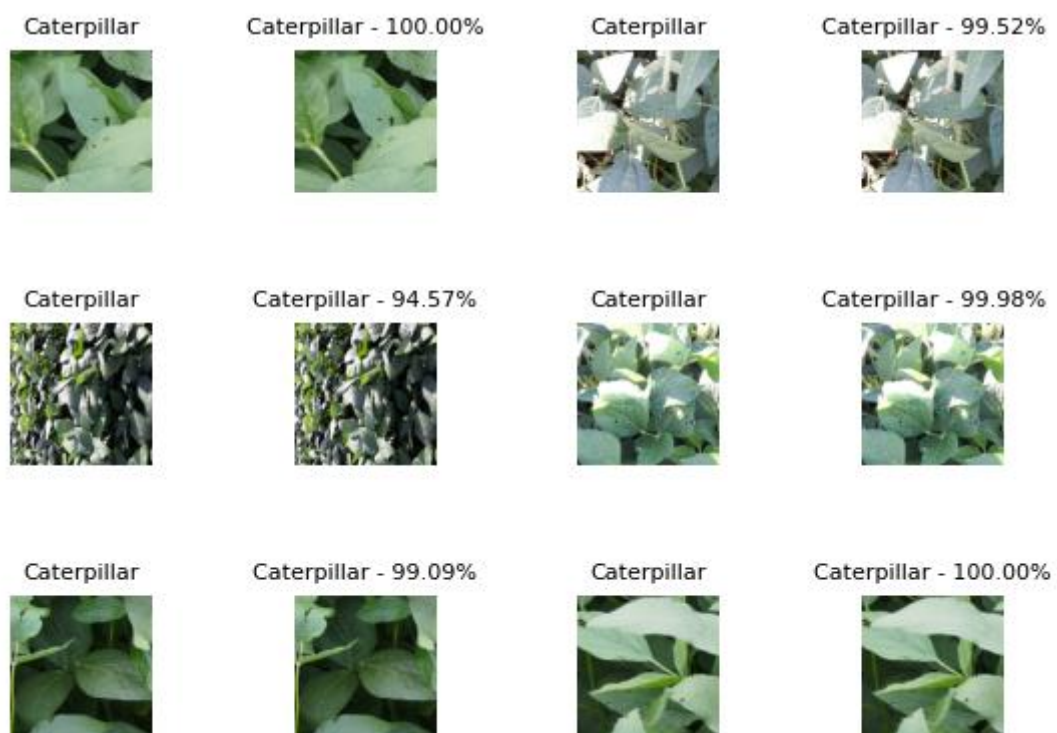
b) Carregamento e Predição das Imagens:

Após carregar o modelo, o *scrip* percorre todos os arquivos de imagem em um local especificado, extrai a classe real da imagem a partir do nome do arquivo, carrega cada imagem e a redimensiona para as dimensões definidas, converte a imagem para um *array NumPy* e, por fim, realiza a predição da classe da imagem e aplica a função *softmax* para converter as saídas do modelo em probabilidades para cada classe.

c) Visualização dos Resultados:

Neste trecho da aplicação, a imagem original e a imagem com a predição são exibidas. A imagem original recebe um título contendo a classe real e a segunda imagem recebe um título contendo a classe prevista e seu nível de confiança. Por fim, a nova imagem gerada, apresentada na figura 15, contendo a sequência de imagens reais e predições, é salva e exibida na tela. Na imagem gerada a classe real é apresentada na primeira e terceira colunas, a classificação pelo algoritmo e respectivo nível de confiança são exibidas na segunda e quarta colunas.

Figura 15: Exemplo de imagem gerada pelo algoritmo de aplicação.



Fonte: Elaborado pelo autor.

3.4. Interface gráfica

O tópico anterior mostrou o *script* utilizado para execução do modelo final de RNC. Agora, para tornar a utilização do algoritmo de classificação mais amigável, foi desenvolvida uma interface gráfica.

O código usa a biblioteca *Tkinter* para criar uma interface gráfica de usuário ou *graphical user interface* (GUI). A partir da interface, de forma oculta, o *script* retorna o caminho para a imagem a ser classificada, com a biblioteca PIL, o aplicativo permite que o usuário carregue uma imagem e a exiba na interface. Em seguida, utilizando o modelo pré-treinado, o aplicativo classifica a imagem, prevendo se ela pertence a uma das três classes: "*Caterpillar*" (lagarta), "*Diabrotica speciosa*" (besouro) e "*Healthy*" (saúdável). O resultado da classificação, juntamente com a probabilidade (confiança do modelo), é exibido na interface. O usuário também tem a opção de salvar a imagem com a classificação e a probabilidade sobrepostas.

De forma aprofundada:

a) Inicialização e Carregamento do Modelo:

O código importa as bibliotecas necessárias e carrega o modelo final. O caminho para o modelo é definido com base no local de execução do arquivo *Python*, assim, o modelo precisa apenas estar na mesma pasta que o arquivo *.py* para que a interface funcione. Em seguida, as dimensões da imagem de entrada são definidas.

b) Processamento

Inicialmente, a imagem a ser classificada é selecionada pelo usuário através de uma caixa de diálogo. A imagem é exibida na interface, redimensionada e convertida em um *array*. Este, por sua vez, é submetido à entrada do modelo de classificação que retorna a classe prevista e seu nível de confiança. Por fim, a imagem de entrada é editada, passando a exibir a classificação juntamente com o nível de confiança obtido, e salva. A classificação da imagem ocorre por meio da seleção da classe com maior probabilidade de correspondência conforme calculado pelo modelo.

c) Criação e execução da Interface Gráfica:

A interface gráfica é baseada na biblioteca *Tkinter*, que cria a janela principal e os *widgets* da interface, como botões, *labels* e *frames*, além de permitir a personalização da interface por meio da alteração de estilos, cores e fontes. Nesse trecho do código, os botões, vistos pelo usuário, que permitem a escolha da imagem e sua classificação são associados às suas respectivas funções ocultas. Finalizando, o *loop* que mantém a janela aberta e responde às interações do usuário é iniciado.

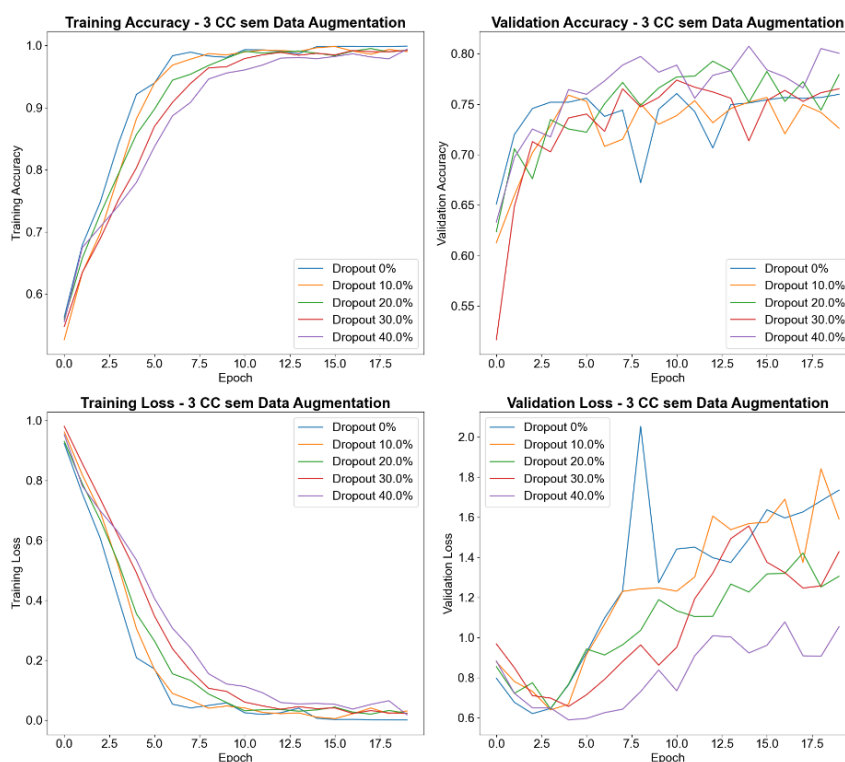
4. RESULTADOS E DISCUSSÕES

O processo de treinamento dos modelos de predição resultou em inúmeras informações pertinentes ao desenvolvimento do projeto aqui proposto. Estes dados, juntamente com a interface gráfica, serão apresentados a seguir.

4.1. Treinamento

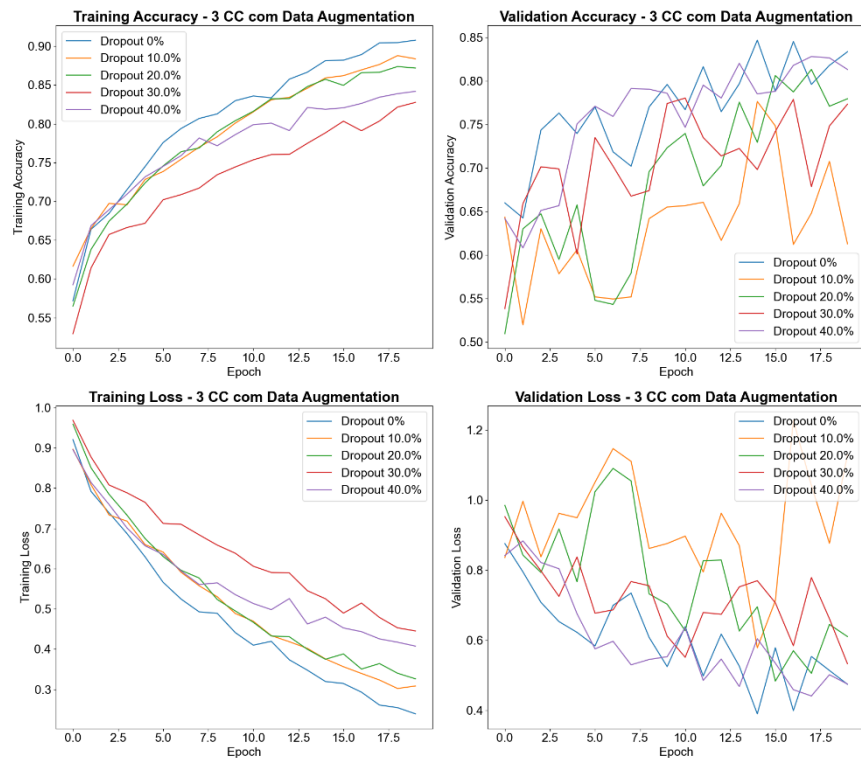
O processo de treinamento foi estruturado de forma a obter o melhor desempenho para o sistema de predição. Para isto, como mencionado anteriormente, foram desenvolvidos 30 modelos de redes neurais diferentes. Inicialmente, cada modelo foi submetido a uma rodada de treinamento com 20 épocas, onde apenas os parâmetros da iteração com maior acurácia foram salvos. Após a primeira fase de treinamento, os modelos com os melhores desempenho seguiram para a segunda fase de treinamento e assim sucessivamente, resultando em oito fases de treinamento. As figuras a seguir, apresentam o comportamento dos modelos com três camadas convolucionais ao longo das 20 épocas de treinamento. Os modelos estão divididos de acordo com a presença ou não de *data augmentation*. A primeira coluna apresenta dados de pertinentes ao treinamento e a segunda coluna diz respeito aos dados de validação.

Figura 16: Modelos com três camadas convolucionais (CC), sem *data augmentation*.



Fonte: Elaborado pelo autor.

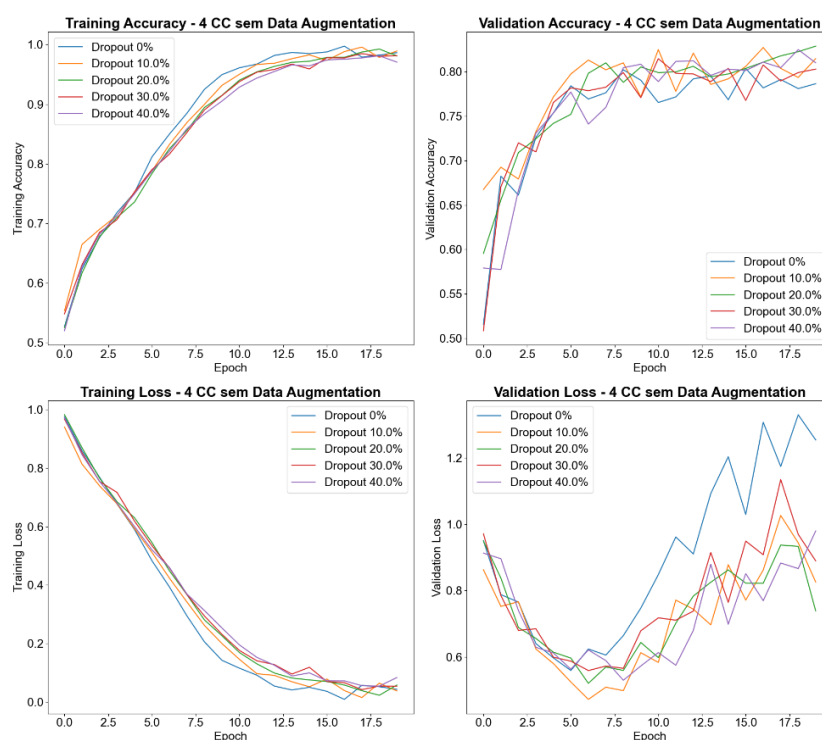
Figura 17: Modelos com três camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

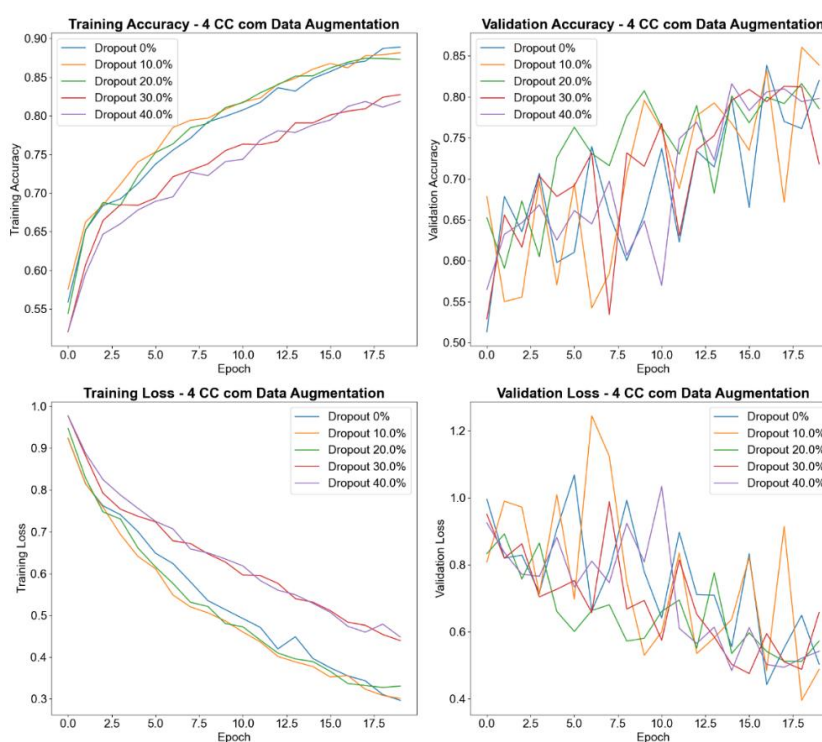
Nas figuras anteriores, é possível observar que os comportamentos das curvas de treinamento seguem o padrão esperado, onde a perda diminui à medida que a acurácia aumenta. Contudo, na fase de validação todos os modelos sem *data augmentation* e alguns modelos com aumento de dados apresentam divergência na curva de perda de validação. Além disso, ambas as curvas de validação apresentaram oscilações em todos os modelos. A seguir, são apresentados os comportamentos dos modelos com quatro camadas convolucionais.

Figura 18: Modelos com quatro camadas convolucionais (CC), sem *data augmentation*.



Fonte: Elaborado pelo autor.

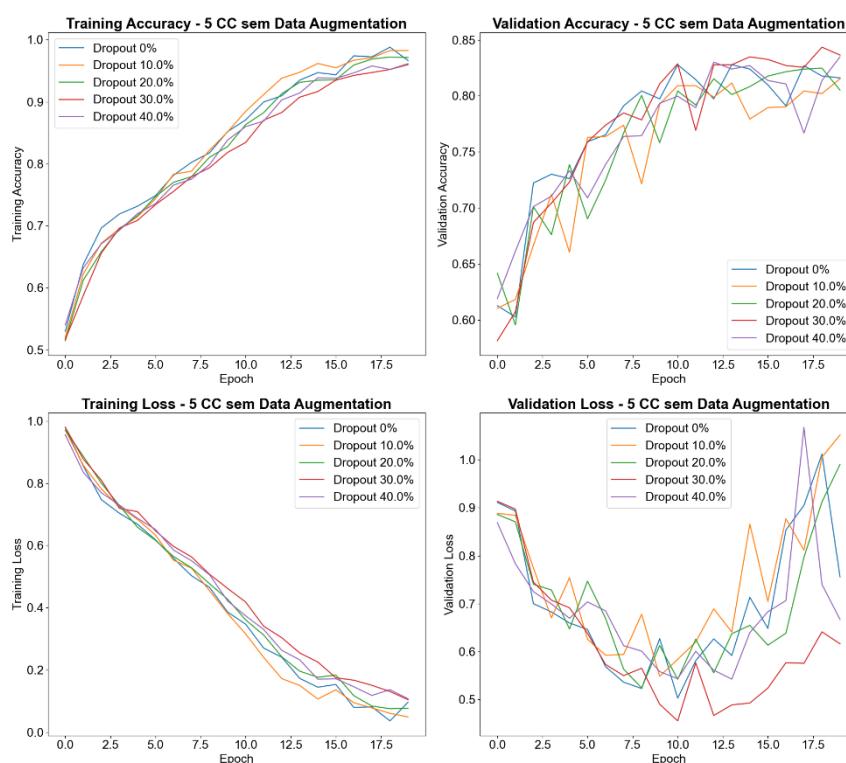
Figura 19: Modelos com quatro camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

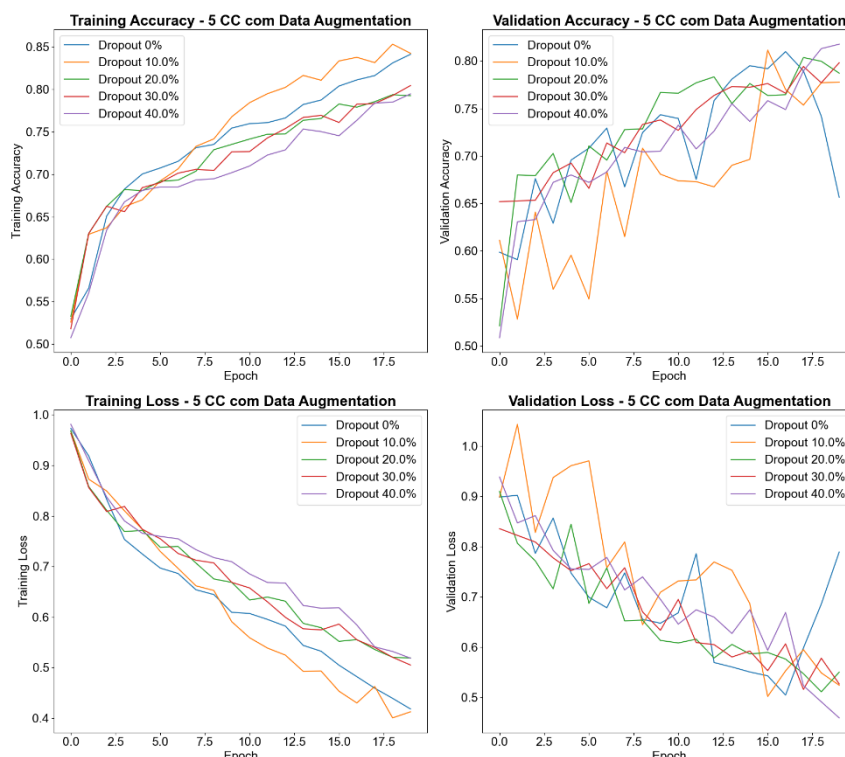
Semelhante aos modelos com três camadas convolucionais, os modelos atuais apresentaram o comportamento de treinamento esperado e todos os modelos apresentaram oscilações em ambas as curvas de validação. O modelo sem aumento de dados ainda apresentou divergência na perda de validação, contudo, a amplitude da perda foi reduzida em comparação ao modelo com três camadas convolucionais e sem *data augmentation*. O modelo com aumento de dados passou a apresentar uma curva de perda de validação que tende a convergência. As figuras 20 e 21 ilustram o comportamento dos modelos com cinco camadas convolucionais.

Figura 20: Modelos com cinco camadas convolucionais (CC), sem *data augmentation*.



Fonte: Elaborado pelo autor.

Figura 21: Modelos com cinco camadas convolucionais (CC), com *data augmentation*.

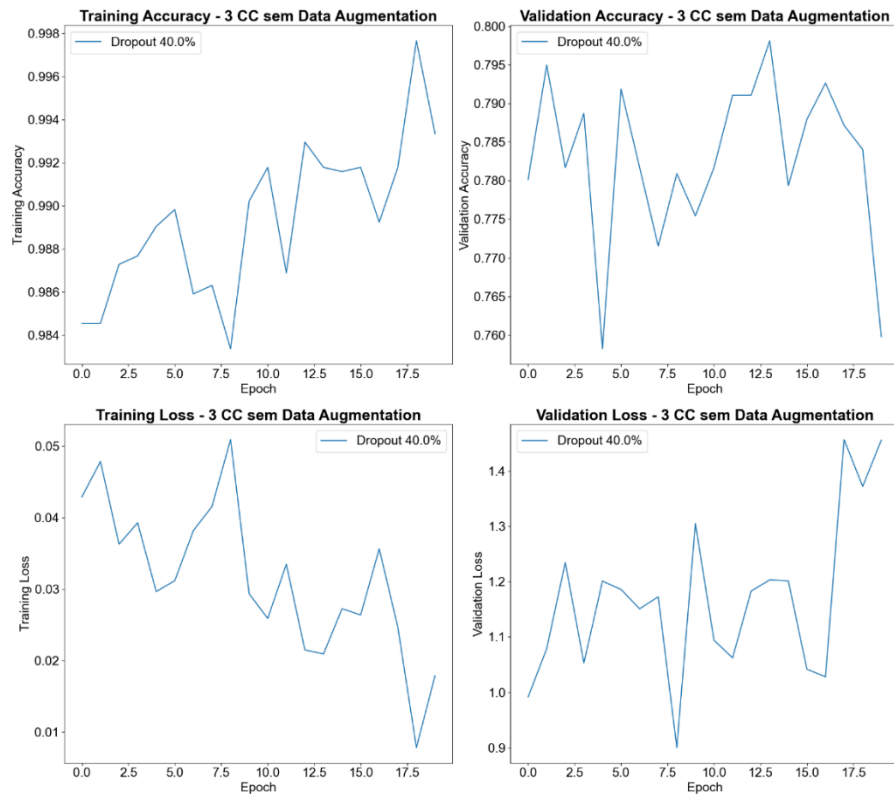


Fonte: Elaborado pelo autor.

O comportamento observado para os modelos atuais mantém o padrão observado nos modelos anteriores, incluindo a convergência da curva de perda de validação para modelos com *data augmentation* e a redução da amplitude da curva de perda de validação, que ainda tende à divergência, para modelos sem aumento de dados.

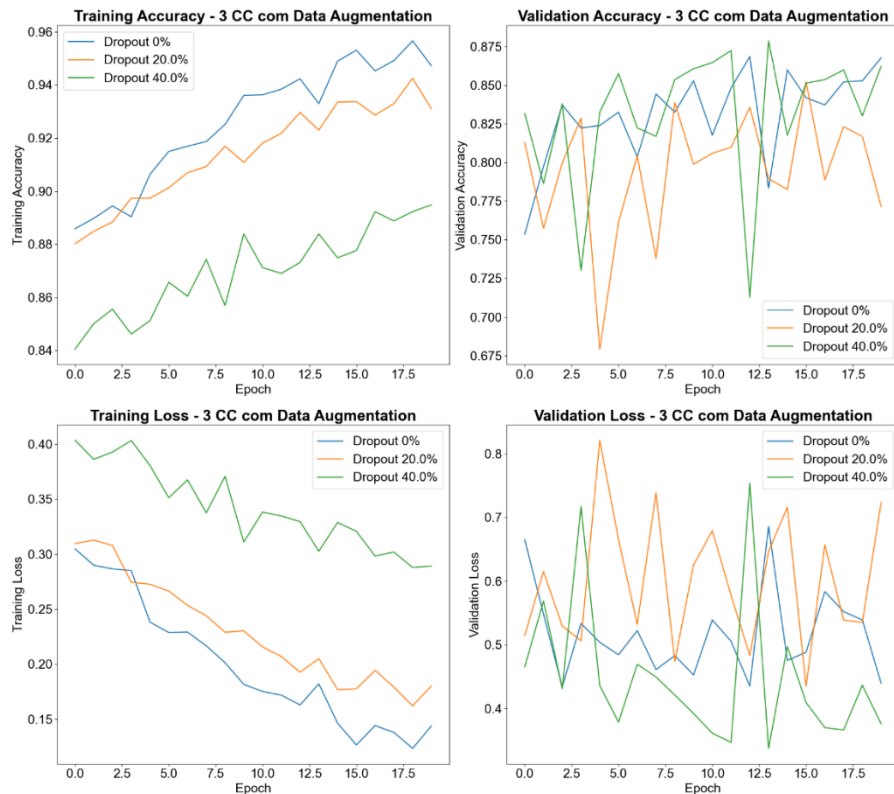
Para a segunda fase de treinamento, foram selecionados os modelos com acurácia acima de 80% e, após uma nova rodada de treinamento com 20 épocas, os modelos passaram a apresentar novos desempenhos. A seguir, são apresentados os desempenhos individuais ao longo do novo período de treinamento, iniciando com os modelos com três camadas convolucionais nas figuras 22 e 23.

Figura 22: Modelo com três camadas convolucionais (CC), sem *data augmentation*.



Fonte: Elaborado pelo autor.

Figura 23: Modelos com três camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

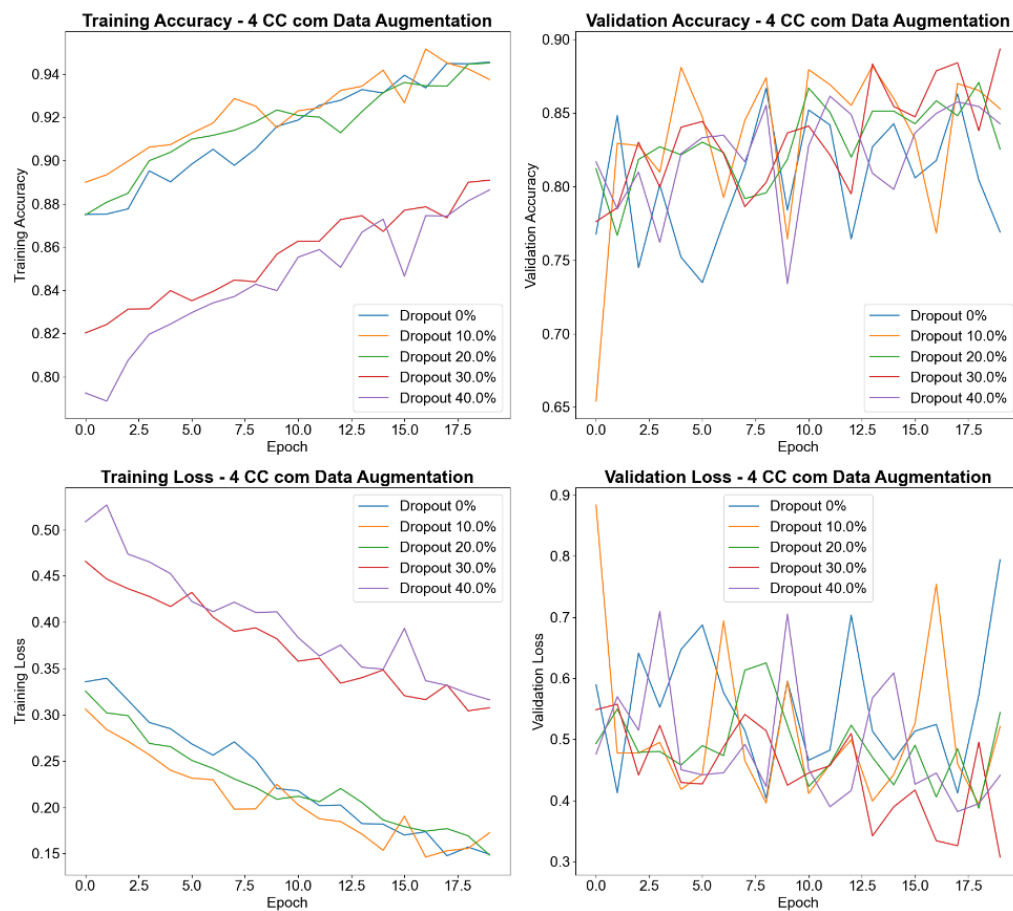
O modelo sem *data augmentation* manteve a divergência da curva de perda de validação, embora com menor amplitude, enquanto os modelos com aumento de dados seguiram a tendência de conversão, semelhante ao comportamento observado na primeira etapa. Contudo, as oscilações se tornaram mais acentuadas para os dados de validação e passaram a ser observadas nos dados de treinamento. As próximas figuras apresentam os comportamentos dos modelos com quatro camadas.

Figura 24: Modelos com quatro camadas convolucionais (CC), sem *data augmentation*.



Fonte: Elaborado pelo autor.

Figura 25: Modelos com quatro camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

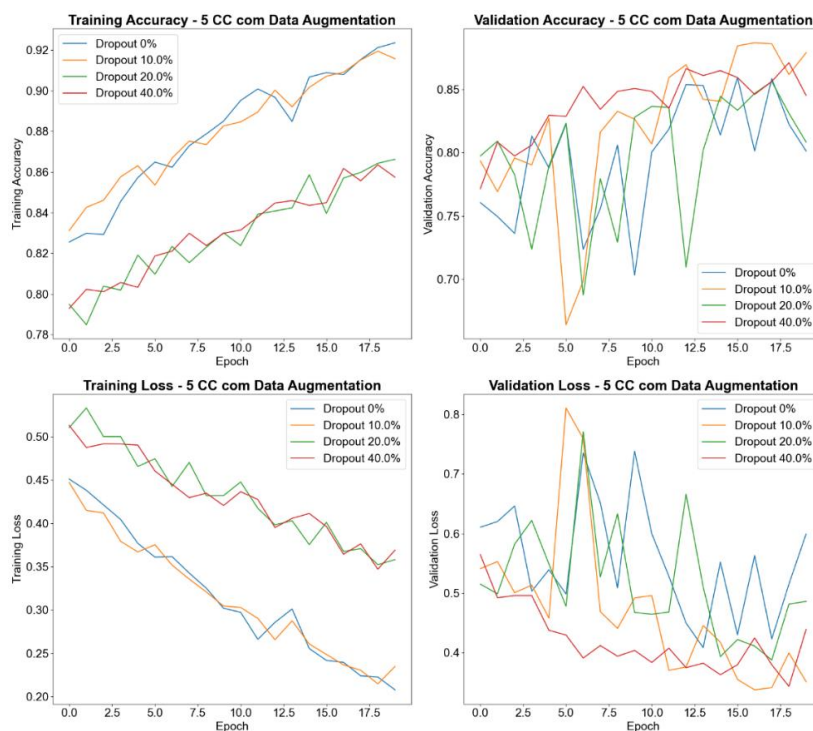
Os dados de treinamento, apesar de alcançarem um comportamento próximo ao ideal, ainda apresentaram oscilações. Além disso, os modelos sem aumento de dados, além de manterem a divergência, voltaram a apresentar elevação de perda, quebrando a tendência observada nos modelos anteriores. Finalizando a segunda etapa, os dados dos modelos com cinco camadas são apresentados.

Figura 26: Modelos com cinco camadas convolucionais (CC), sem *data augmentation*.



Fonte: Elaborado pelo autor.

Figura 27: Modelos com cinco camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

O comportamento dos dados de treinamento manteve o padrão já observado, já os dados de validação apresentaram oscilações ainda maiores. A acurácia de validação para modelos sem aumento de dados manteve-se entre 76 e 84%, com alguns pontos fora dessa margem, já a perda para esses modelos manteve o comportamento de divergência, mas voltou a apresentar redução de amplitude. Para modelos com aumento de dados, a acurácia de validação apresentou aumento leve com oscilações e a perda continuou a apresentar o comportamento de convergência observado para esses modelos.

A terceira etapa foi constituída de modelos que obtiveram acurácia acima de 85%. As figuras 28 a 31 apresentam o desempenho dos modelos selecionados, iniciando por modelos com três camadas com aumento de dados.

Figura 28: Modelos com três camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

Os modelos com três camadas com aumento de dados apresentaram um comportamento curioso na terceira fase de treinamento: todas as curvas de perda de validação apresentaram uma tendência divergente nas últimas épocas de

treinamento. Além disso, as perdas entre os modelos começaram a apresentar diferenças notáveis, sendo o modelo com *dropout* de 40% o mais estável, com menor perda e maior acurácia de treinamento. Este comportamento contradiz os dados de treinamento, os quais informam que o modelo em questão aparentou ser o menos preciso e o com maior perda.

Continuando a terceira fase, a figura 29 apresenta o comportamento dos modelos com quatro camadas convolucionais.

Figura 29: Modelos com quatro camadas convolucionais (CC), com *data augmentation*.

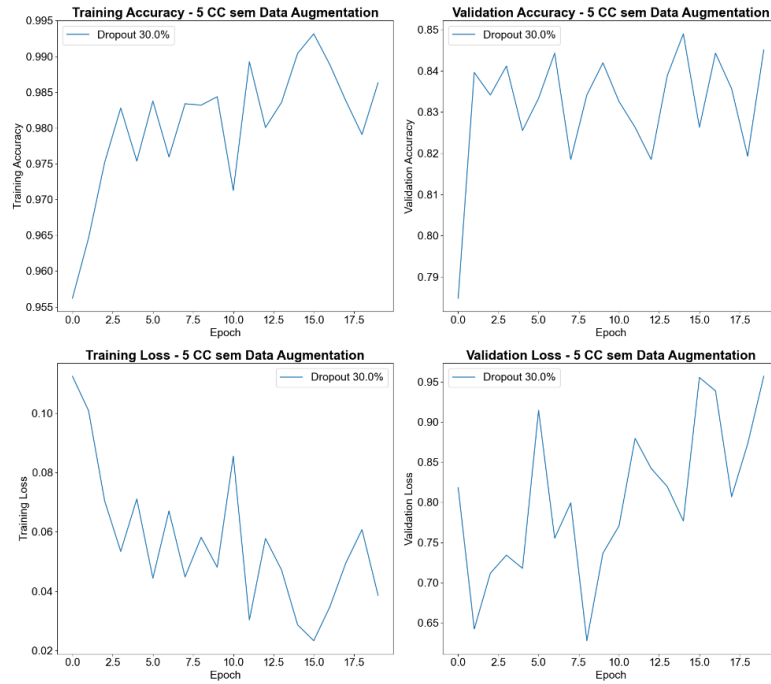


Fonte: Elaborado pelo autor.

Nos modelos da figura 29 observa-se um comportamento semelhante ao observado nas fases anteriores no que diz respeito aos dados de treinamento. A acurácia de validação manteve-se oscilante entre aproximadamente 80 e 90%; a perda de validação também se manteve oscilante, mas apresentou oscilações proporcionalmente maiores, variando entre aproximadamente 0,3 e 0,7.

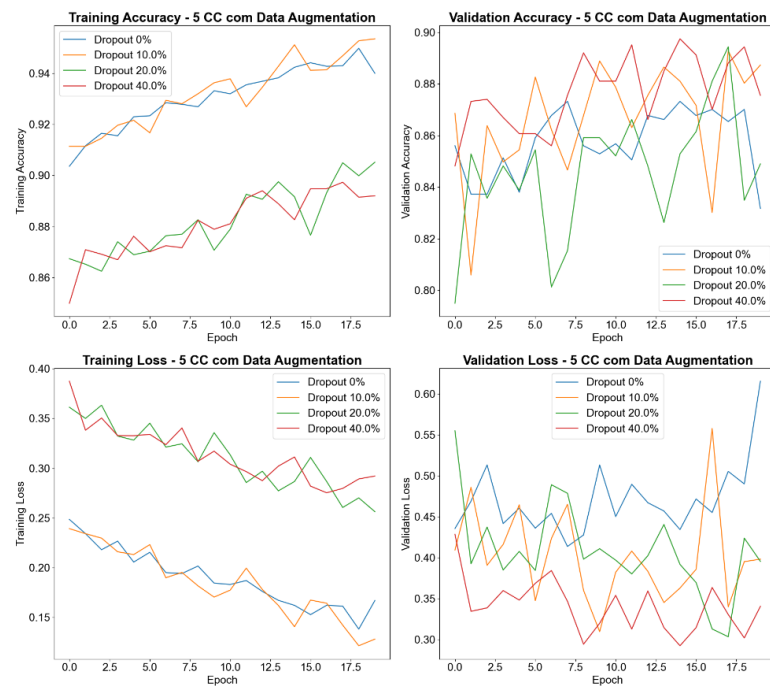
Finalizando a terceira fase, tem-se os dados para os modelos com cinco camadas convolucionais.

Figura 30: Modelo com cinco camadas convolucionais (CC), sem *data augmentation*.



Fonte: Elaborado pelo autor.

Figura 31: Modelos com cinco camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

Conforme o esperado com base nas fases anteriores, a perda de validação do modelo sem aumento de dados divergiu. Para os modelos com *data augmentation*, a tendência de estabilidade e oscilação para a validação foi mantida, a acurácia apresentou mínimo e máximo de aproximadamente 80 e 90% respectivamente. Já a perda oscilou entre 0,3 e 0,6, aproximadamente.

A quarta fase de treinamento contou com nove modelos provenientes da terceira fase de treinamento. Os modelos que avançaram para a fase atual alcançaram no mínimo 89% de acurácia. As figuras 32 a 34 apresentam o desempenho individual dos modelos que avançaram para esta etapa.

Figura 32: Modelo com três camadas convolucionais (CC), com *data augmentation*.

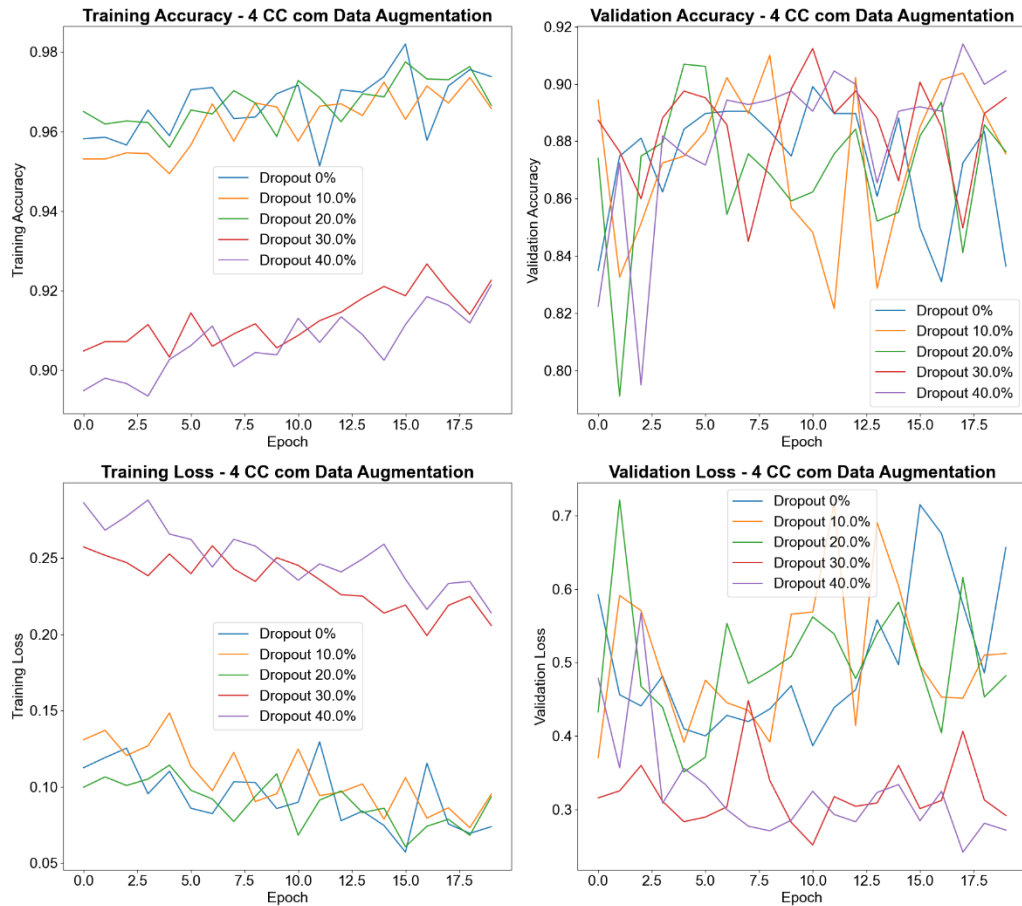


Fonte: Elaborado pelo autor.

O modelo da figura anterior mostrou diferenças significativas em relação aos modelos e fases anteriores. Agora, além da acurácia e perda de treinamento tenderem à convergência para um e zero, respectivamente, os dados de validação passaram a apresentar comportamento semelhante, indicando que o modelo pode ter se adaptado de forma satisfatória à tarefa a ser executada.

A figura 33 ilustra o comportamento dos modelos com quatro camadas convolucionais com aumento de dados.

Figura 33: Modelos com quatro camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

Nos modelos da figura 33 apresentam o mesmo comportamento contraditório observado na figura 18. Apesar do comportamento da acurácia e perda para os dados de treinamento indicaram que os modelos com *dropout* de 30 e 40% são os modelos com os piores desempenhos, os dados de validação mostraram o oposto, informando que estes mesmos modelos são os com maior acurácia e menor perda.

Finalizando a quarta fase de treinamento, tem-se os comportamentos dos modelos com cinco camadas com *data augmentation* apresentados na figura 34.

Figura 34: Modelos com cinco camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

Assim como no gráfico anterior, é possível observar uma inversão de desempenho dos modelos em relação aos dados de treinamento e validação, no entanto, a inversão se torna mais sutil devido às oscilações mais evidentes.

A quinta fase, agora com apenas 4 modelos dos 30 inicialmente propostos, reteinou modelos que alcançaram no mínimo 91% de acurácia na fase quatro. Os desempenhos individuais de cada modelo ao longo das 20 épocas de retreinamento podem ser observados nas figuras 35 e 36.

Figura 35: Modelo com três camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

O modelo acima, assim, como na etapa anterior, apresentou comportamento positivo no que diz respeito à sua adaptação a atividade proposta, mantendo tendência de convergência das curvas de acurácia de treinamento e validação a um, e convergência das curvas de perda de treinamento e validação a zero.

Os comportamentos dos últimos modelos da fase 5 são apresentados na figura 36.

Figura 36: Modelos com quatro camadas convolucionais (CC), com *data augmentation*.

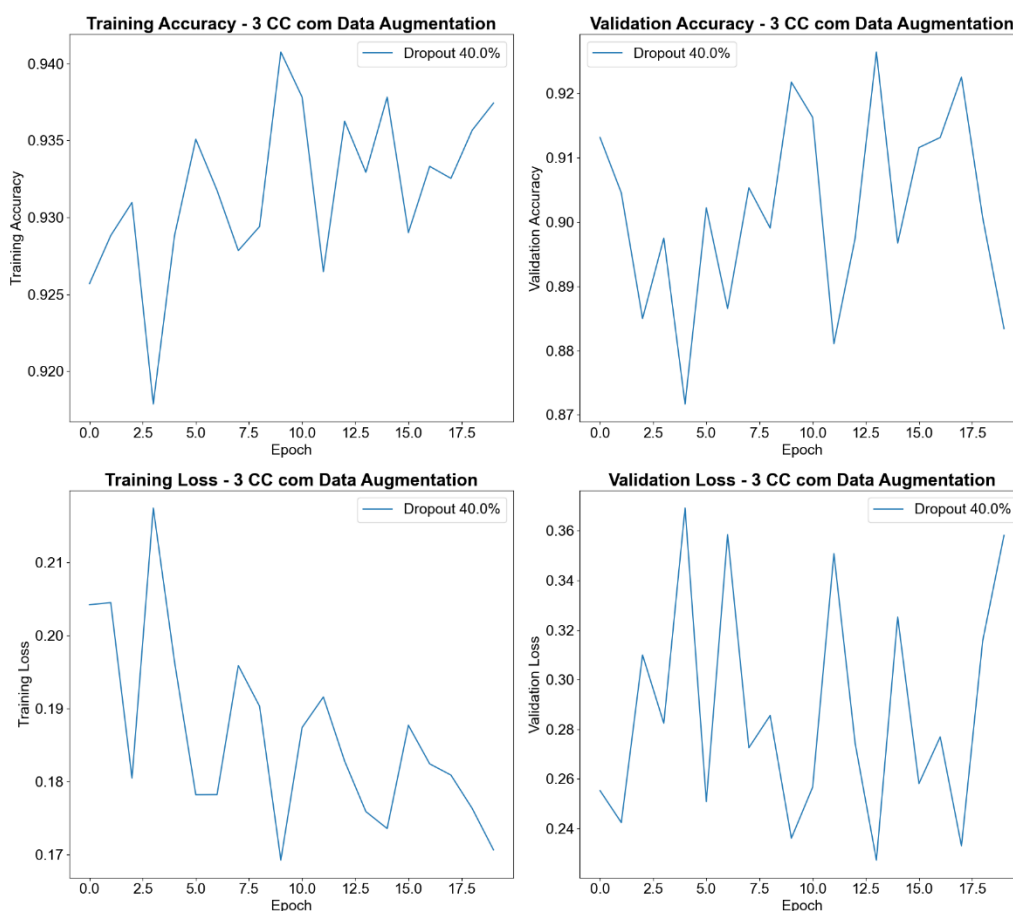


Fonte: Elaborado pelo autor.

Para os modelos com quatro camadas convolucionais foi observada a mesma inversão de desempenho em relação aos dados de treinamento e validação apresentada na etapa anterior, porém mais nítida. Além disso, observa-se um pico de perda de validação e um afundamento no desempenho na época cinco.

A partir da sexta fase, apenas um modelo passará a ser treinado uma vez que apenas modelos cuja acurácia mínima foi de 92%. O desempenho do modelo nesta fase é apresentado na figura 36.

Figura 37: Modelo com três camadas convolucionais (CC), com *data augmentation*.

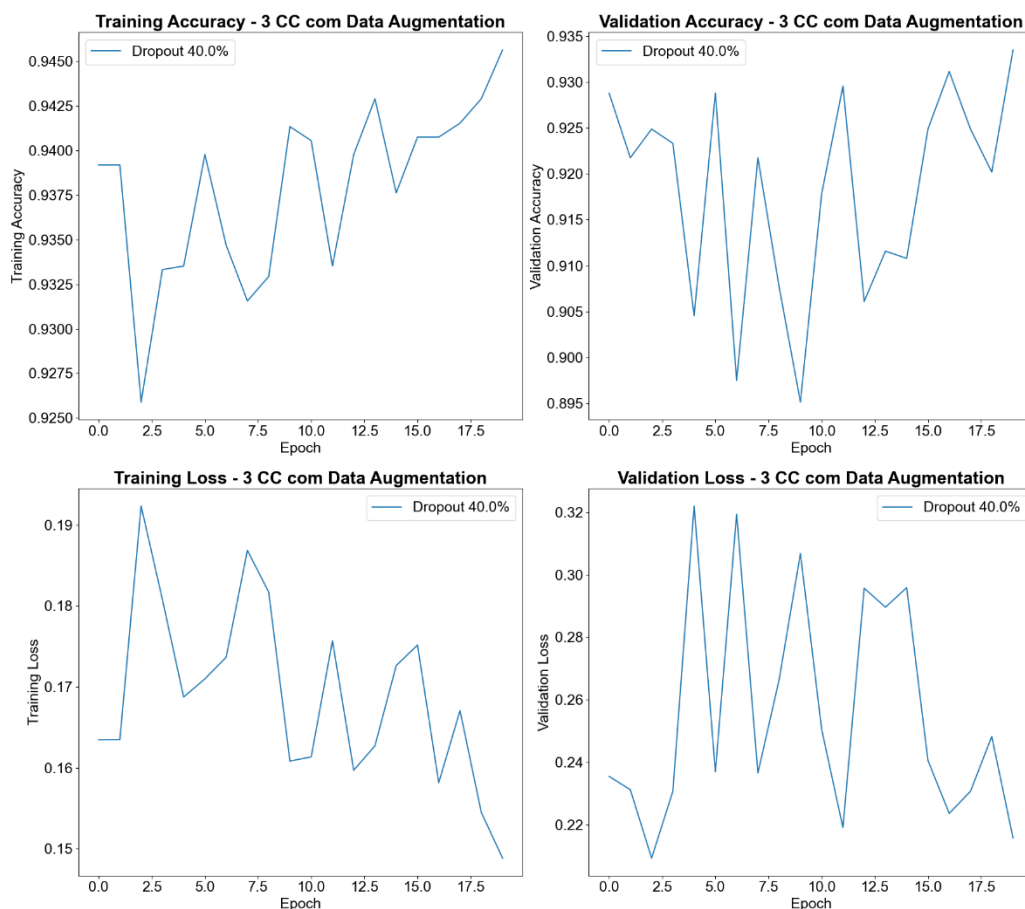


Fonte: Elaborado pelo autor.

Para a sexta fase, o modelo manteve os padrões para dos dados de treinamento vistos anteriormente, contudo, nesta fase, a acurácia de validação iniciou o processo de treinamento em queda, mas se recuperou, alcançando seu pico na época 13, sendo essa a melhor acurácia até o momento. A perda de validação manteve um comportamento predominantemente oscilatório, sem tender a convergência em zero, porém apresentou valor máximo menor que o valor máximo observado na fase anterior.

A sétima fase de treinamento reprocessou o modelo da fase anterior com o objetivo de elevar a acurácia do algoritmo. O desempenho do modelo após o retreinamento é apresentado na figura 37.

Figura 38: Modelo com três camadas convolucionais (CC), com *data augmentation*.

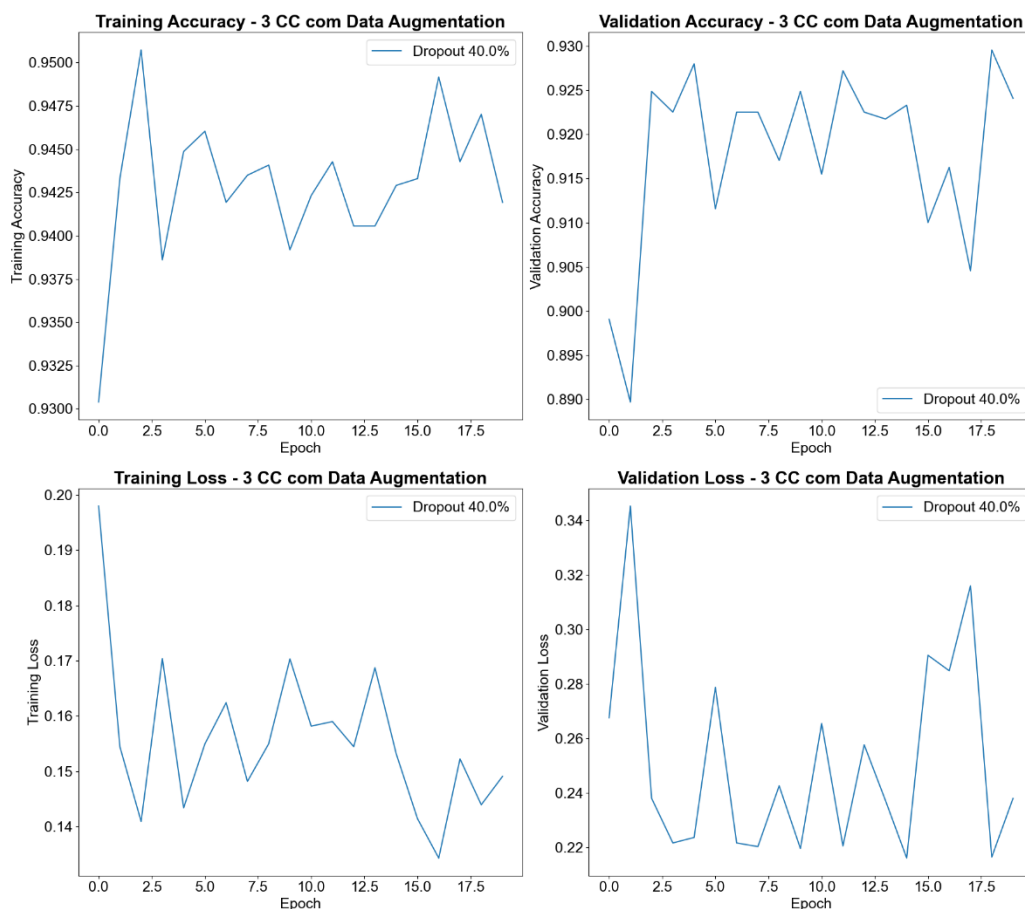


Fonte: Elaborado pelo autor.

A sétima fase manteve o comportamento observado na fase anterior, no entanto, o pico de perda de validação foi menor que o observado anteriormente, e a acurácia de validação alcançou um novo patamar, superior ao nível obtido na sexta fase.

A oitava fase, assim como a fase anterior, buscou melhorar o desempenho do algoritmo. O comportamento do modelo no decorrer das 20 épocas é apresentado na figura 38.

Figura 39: Modelo com três camadas convolucionais (CC), com *data augmentation*.



Fonte: Elaborado pelo autor.

Na última fase de treinamento, observou-se uma redução de desempenho do modelo: a acurácia de treinamento, apesar de iniciar com um pico de aproximadamente 95%, decai com o decorrer das épocas. A perda de treinamento apresenta um decaimento inicial, contudo volta a subir e se mantém variável entre aproximadamente 0,13 e 0,17. Para a validação, a acurácia apresentou um comportamento oscilante predominantemente entre 91 e 92,5%, já a perda de treinamento inicia-se com um pico de aproximadamente 0,34 que decai, mantendo-se flutuante entre 0,22 e 0,32.

4.1.1. Divergência na curva de perda de validação

Ao observar as figuras 16, 18, 20, 22, 24, 26 e 30, é possível notar que a curva de perda de validação possui um comportamento semelhante a uma parábola de coeficiente positivo, ou seja, inicialmente ela tende a convergência em zero, assim como a perda de treinamento e, após algumas épocas, diverge. Como visto, modelos

que apresentam o comportamento descrito possuem fortes indícios de *overfitting*. Os gráficos que apresentaram o sobreajuste descrevem o comportamento de modelos que não possuem *data augmentation*. Tal informação corrobora o exposto anteriormente: poucos dados de treinamento fazem com que o modelo perca capacidade de generalização devida a baixa variação de padrões. Ao realizar o aumento artificial de dados, a cada iteração o modelo se depara com dados “novos”, aumentando a variabilidade de padrões.

Alguns modelos que utilizam *data augmentation* principalmente modelos com pequenas taxas de *dropout* também apresentaram *overfitting*, validando o que foi evidenciado no Referencial Teórico: a desativação aleatória torna o modelo mais robusto por dificultar a memorização de dados o que evita o *overfitting*.

4.1.2. Oscilações nas métricas de treinamento

Após a primeira fase de treinamento, um novo comportamento é observado nas curvas de acurácia e perda relacionadas ao conjunto de treinamento: oscilações. De acordo com o *GOOGLE.COM* (2024), estas oscilações podem estar relacionadas a uma alta taxa de aprendizagem, uma vez que os ajustes realizados nos pesos do modelo durante o treinamento são grandes, ou seja, os incrementos ou decrementos realizados sobre os pesos são grandes, fazendo com que o resultado alcançado flutue entorno do resultado verdadeiro. Na primeira fase, este comportamento não é observado de forma acentuada como nas demais fases. Nela, o modelo nunca teve contato com os dados, dessa forma, os ajustes realizados são maiores para reduzir o erro inicial. A partir da segunda fase, onde o modelo já passou por um ajuste grosso dos parâmetros, a sintonia deve ser mais suave e, para isso a taxa de aprendizagem deve ser menor, contudo, como a taxa de aprendizagem foi constante para todas as fases de treinamento, o comportamento oscilatório pôde ser observado no desempenho de treinamento.

4.1.3. Oscilações nas métricas de validação

Os gráficos de acurácia e perda de validação também apresentaram oscilações. No entanto, a justificativa para esse fenômeno nos dados de validação se deve a outros motivos. Esse comportamento, conforme mencionado pelo *GOOGLE.COM* (2024), é explicado pela presença de *overfitting*. Quando um modelo perde a capacidade de generalização, as classificações realizadas em dados

diferentes dos utilizados no treinamento podem oscilar com base na proximidade entre os dados de treinamento e validação.

Uma segunda justificativa é o desequilíbrio entre as classes, onde uma classe possui mais dados que outra, o que ocorre no banco de dados utilizado, uma vez que este possui aproximadamente 6410 imagens divididas em três classes: *Healthy* (saúdavel) com 896 imagens, *Caterpillar* (lagarta), 3309 imagens, *Diabrotica speciosa* com 2205 imagens. Ao expor o modelo a dados desequilibrados, ele pode tender a alguns cenários. No primeiro, a cada iteração o modelo pode receber exemplos de diferentes classes, se a maioria dos dados for de classes majoritárias, o modelo tende a se ajustar aos padrões destas classes, podendo causar oscilações de desempenho quando o modelo encontra exemplos de classes minoritárias. Uma analogia válida é comparar o modelo a um aluno que estuda para uma prova com base em avaliações anteriores: caso ele estude provas que possuam mais questões de determinado assunto, ele pode ter melhor desempenho caso sua avaliação possua mais questões deste assunto, do contrário, seu desempenho poderá ser prejudicado.

No segundo cenário, devido à baixa variabilidade de dados das classes minoritárias, pode ocorrer *overfitting* em relação a estes dados, causando oscilações quando o modelo é submetido a novos dados. Aqui, pode-se comparar o comportamento do modelo a um professor que está preparando seus alunos para uma avaliação, mas tem um número muito limitado de exemplos de perguntas sobre um tema específico, enquanto tem muitas perguntas de outros temas. O professor ensina aos alunos exatamente as poucas questões que têm desse tema raro. No dia da prova, os alunos se saem muito bem nas questões que eram idênticas às que o professor ensinou (porque memorizaram essas poucas perguntas), mas se aparecerem novas perguntas sobre o mesmo tema, os alunos não sabem respondê-las, porque só decoraram as perguntas anteriores e não entenderam o conceito de maneira geral.

4.1.4. Inversão de desempenho em relação aos dados de treinamento e validação

Como visto nas figuras 28, 33, 34 e 36, um comportamento diferente foi observado em relação aos dados de treinamento e validação. Modelos que na fase de treinamento apresentaram as melhores métricas, obtiveram desempenhos inferiores aos demais na fase de validação.

Tal comportamento é fruto de um provável sobreajuste dos modelos que obtiveram as melhores métricas durante o treinamento. O *overfitting*, como dito anteriormente, ocorre quando o modelo perde a capacidade de generalização e passa a memorizar os dados de treinamento. Dessa forma, os modelos apresentaram alto desempenho para os dados de treinamento, contudo, ao se depararem com novos dados, não foram capazes de classificá-los de forma correta, implicando em métricas inferiores para a validação.

Uma análise mais aprofundada dos gráficos revela que modelos com maior *dropout*, embora não tenham se adaptado de forma precisa aos dados de treinamento, são os que apresentaram os melhores desempenhos na validação, mostrando novamente que o desligamento aleatório de neurônios torna o modelo mais robusto e com maior capacidade de generalização.

As tabelas 3 e 4 apresentam, respectivamente, as acurácias e perdas de validação dos modelos para todas as etapas de treinamento de forma compilada, de forma a facilitar as observações que tangem os desempenhos de treinamento.

Tabela 3: Acurácia dos modelos para todas as etapas de treinamento.

		Acurácia								
	Dropout	CC	Fase 1	Fase 2	Fase 3	Fase 4	Fase 5	Fase 6	Fase 7	Fase 8
Sem Data Augmentation	0%	3	76,06	-	-	-	-	-	-	-
		4	80,36	80,67	-	-	-	-	-	-
		5	82,79	84,51	-	-	-	-	-	-
	10%	3	75,90	-	-	-	-	-	-	-
		4	82,71	83,18	-	-	-	-	-	-
		5	81,53	83,10	-	-	-	-	-	-
	20%	3	79,26	-	-	-	-	-	-	-
		4	82,86	83,26	-	-	-	-	-	-
		5	82,47	83,88	-	-	-	-	-	-
	30%	3	77,39	-	-	-	-	-	-	-
		4	81,46	83,49	-	-	-	-	-	-
		5	84,35	85,13	84,90	-	-	-	-	-
Com Data Augmentation	0%	3	80,75	79,81	-	-	-	-	-	-
		4	82,47	83,10	-	-	-	-	-	-
		5	83,49	84,27	-	-	-	-	-	-
	10%	3	84,66	86,85	88,11	-	-	-	-	-
		4	83,88	86,70	89,05	89,91	-	-	-	-
		5	80,99	85,92	87,32	-	-	-	-	-
	20%	3	77,62	-	-	-	-	-	-	-
		4	86,07	88,18	90,22	91,00	90,77	-	-	-
		5	81,14	88,65	89,28	90,53	-	-	-	-
	30%	3	81,30	85,21	87,48	-	-	-	-	-
		4	81,61	87,09	89,05	90,69	-	-	-	-
		5	80,36	85,60	89,44	90,38	-	-	-	-
	40%	3	78,01	-	-	-	-	-	-	-
		4	81,30	89,36	89,36	91,24	91,00	-	-	-
		5	79,81	-	-	-	-	-	-	-
		3	82,79	87,87	89,51	91,08	92,33	92,64	93,35	92,96
		4	81,61	86,15	89,28	91,39	91,94	-	-	-
		5	81,77	87,09	89,75	90,85	-	-	-	-

Fonte: Elaborado pelo autor.

Tabela 4: Perda dos modelos para todas as etapas de treinamento.

		Perda								
	Dropout	CC	Fase 1	Fase 2	Fase 3	Fase 4	Fase 5	Fase 6	Fase 7	Fase 8
Sem Data Augmentation	0%	3	1,44	-	-	-	-	-	-	-
		4	1,03	1,49	-	-	-	-	-	-
		5	0,50	0,95	-	-	-	-	-	-
	10%	3	0,67	-	-	-	-	-	-	-
		4	0,86	1,16	-	-	-	-	-	-
		5	1,05	0,95	-	-	-	-	-	-
	20%	3	1,11	-	-	-	-	-	-	-
		4	0,74	1,04	-	-	-	-	-	-
		5	0,91	1,13	-	-	-	-	-	-
	30%	3	0,95	-	-	-	-	-	-	-
		4	0,72	1,11	-	-	-	-	-	-
		5	0,64	0,52	0,78	-	-	-	-	-
Com Data Augmentation	0%	3	0,92	1,20	-	-	-	-	-	-
		4	0,87	0,89	-	-	-	-	-	-
		5	0,67	0,81	-	-	-	-	-	-
	10%	3	0,39	0,43	0,43	-	-	-	-	-
		4	0,44	0,40	0,40	0,39	-	-	-	-
		5	0,51	0,43	0,41	-	-	-	-	-
	20%	3	0,58	-	-	-	-	-	-	-
		4	0,40	0,40	0,40	0,39	0,42	-	-	-
		5	0,50	0,34	0,34	0,32	-	-	-	-
	30%	3	0,51	0,44	0,41	-	-	-	-	-
		4	0,51	0,39	0,42	0,35	-	-	-	-
		5	0,54	0,39	0,30	0,28	-	-	-	-
	40%	3	0,55	-	-	-	-	-	-	-
		4	0,51	0,31	0,30	0,25	0,27	-	-	-
		5	0,53	-	-	-	-	-	-	-
		3	0,44	0,34	0,29	0,26	0,23	0,23	0,21	0,22
		4	0,48	0,39	0,30	0,24	0,25	-	-	-
		5	0,46	0,34	0,29	0,27	-	-	-	-

Fonte: Elaborado pelo autor.

Ao observar a tabela 3, é visível que a cada fase de treinamento, a maioria dos modelos elevaram suas acurácias, evolução normalmente esperada dada o aumento do número de iterações. Contudo, alguns modelos apresentaram um comportamento diferente, constituído do aumento da acurácia e posterior redução, característico de *overfitting* por excesso de treinamento.

Ainda sobre a tabela 3, o modelo com *data augmentation*, três camadas convolucionais e 40% de *dropout*, único modelo que avançou para a sexta fase, apresentou declínio de acurácia na oitava fase de treinamento. **Dessa forma, o modelo final utilizado no sistema de reconhecimento é o modelo com *data augmentation*, três camadas convolucionais e 40% de *dropout*, treinado até a sétima fase.**

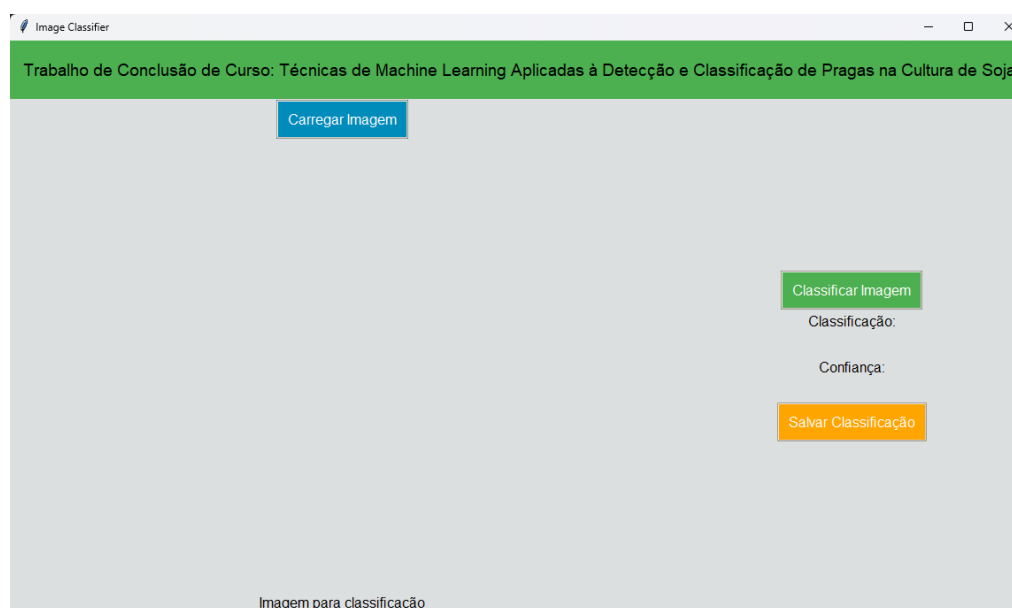
A tabela 4, que apresenta a perda dos modelos, apresenta informações que podem levar à contradição, algumas redes, principalmente aquelas sem *data augmentation*, que aumentaram sua acurácia também elevaram sua perda. Esse comportamento anômalo revela que, apesar do algoritmo ter aumentado o número de previsões corretas, as previsões erradas passaram a ficar cada vez mais distantes dos dados reais.

O desbalanceamento de classes pode, novamente, ser uma justificativa para o fato observado. Como visto, o desbalanceamento pode favorecer a classe majoritária e, à medida que o treinamento avança, os modelos se adaptam cada vez mais a essa classe, aumentando o erro para as classes minoritárias. A falta do aumento artificial de dados potencializa o aumento da perda simultânea à elevação da acurácia uma vez que, ao manter apenas os dados originais, ou seja, sem variações, o modelo tende a perder a capacidade de generalização de forma mais acentuada, tornando a adaptação às classes com maior número de dados mais rápida.

4.2. Interface Gráfica

A interface gráfica de usuário foi desenvolvida com o objetivo de tornar a utilização do sistema de identificação e classificação mais amigável ao usuário, retirando a necessidade de este ter conhecimento prévio em programação. A figura 40 apresenta a tela inicial da interface.

Figura 40: Interface Gráfica para classificação de imagens.

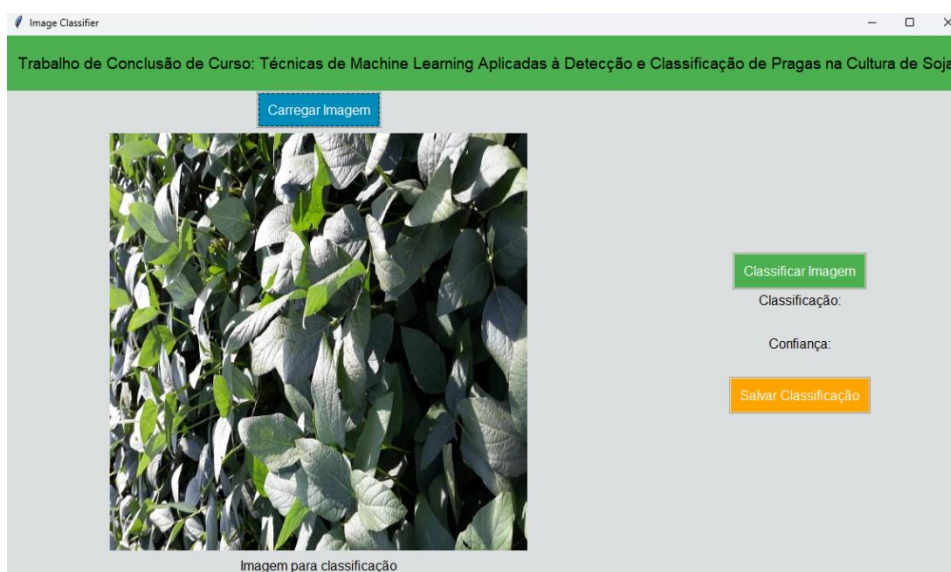


Fonte: Elaborado pelo autor.

A interface conta com três botões:

- Carregar Imagem – sua função é abrir uma nova janela que permita a escolha da imagem a ser classificada, realiza seu carregamento e a exibe na interface. As figuras 41 apresenta o comportamento esperado após o acionamento do botão e carregamento da imagem.

Figura 41: A imagem carregada para a interface gráfica e exibida.



Fonte: Elaborado pelo autor.

- Classificar Imagem – responsável por iniciar o processo de classificação. Ela realiza a classificação por meio do modelo obtido na sétima fase de treinamento e a exibe na interface, como mostra a figura 42.

Figura 42: Apresentação da Classificação e do Nível de Confiança.



Fonte: Elaborado pelo autor.

- Salvar Classificação – tem a função de salvar a imagem classificada juntamente com um título que contém a classificação obtida e o nível de confiança no mesmo local da imagem carregada para classificação. A figura 43 exibe a imagem salva com o rótulo previsto.

Figura 43: Imagem rotulada.

Caterpillar - 95.12%



Fonte: Elaborado pelo autor.

5. CONCLUSÃO

Técnicas de *Machine Learning* são alvos de estudo há muitos anos, contudo, limitações computacionais impediram seu desenvolvimento. Hoje, com o salto tecnológico representado principalmente pelo advento das GPUs, essas técnicas voltaram a se tornar objetos de investigação, alcançando patamares nunca vistos e, paralelo a isso, tem-se o desenvolvimento agrícola. Impulsionada pelas Revoluções Industriais, a agricultura tem incorporado cada vez mais tecnologia ao campo, passando a utilizar recursos de ponta, como drones, com o intuito de alcançar maior produtividade e redução de custos. A informatização do setor agroindustrial somada ao desenvolvimento e melhoria das técnicas de aprendizagem de máquina possibilitaram o desenvolvimento do tema aqui abordado – Técnicas de Machine Learning Aplicadas à Detecção e Classificação de Pragas na Cultura de Soja.

Dentre as inúmeras estruturas que possibilitam o aprendizado de máquina, a Rede Neural Convolucional foi escolhida devido ao seu ótimo desempenho ao se tratar de tarefas associadas à classificação de imagens. A partir da convolução entre uma imagem de entrada e inúmeros filtros, associada a técnicas de manipulação de imagens e estruturas básicas de redes neurais, obteve-se uma estrutura capaz de classificar imagens com acurácia superior a 93%.

Para alcançar o alto desempenho, diversos modelos de RNC foram criados e submetidos a fases de treinamento. A cada fase, a quantidade de modelos que avançavam para a próxima etapa reduzia, ao ponto de alcançar o modelo com maior otimização.

Para alcançar o objetivo final do projeto e com o intuito de tornar a utilização do algoritmo mais amigável, uma interface gráfica foi desenvolvida. A partir dela, o usuário pode selecionar a imagem a ser classificada, realizar sua classificação e salvar os resultados obtidos.

O principal desafio na execução do trabalho se deu durante seu planejamento: encontrar um *dataset* que contivesse imagens capturadas por drones e que permitissem a identificação de pragas por meio delas. Durante o processo de escrita e elaboração do projeto muitos bancos de imagens destinadas a detecção e classificação de pragas foram encontrados, no entanto, todas as imagens mostravam apenas as folhas das plantas, sem imagens a distâncias maiores, como as capturadas por drones. Assim, a escolha das pragas a serem identificadas se deu pela

disponibilidade dos dados.

A partir dos conhecimentos aqui descritos, novos eixos temáticos podem ser desenvolvidos. Dentre eles, destaca-se a aplicação da metodologia desenvolvida a outros bancos de imagens, possibilitando a ampliação de pragas e culturas aplicáveis, elevando o nível de variabilidade da aplicação. Além disso, partindo da aplicação de drones na coleta de dados, uma proposta futura poderá abordar o uso de dados adicionais capturados por estes veículos, utilizando dados de GPS para mapear áreas de maior incidência de pragas ou estabelecer possíveis relações ambientais ao surgimento dos insetos, por exemplo. Outra possível vertente é a aplicação das técnicas apresentadas ao reconhecimento de condições ambientais adversas, como estresse hídrico ou falta de nutrientes, por meio de imagens.

O trabalho desenvolvido traz contribuições relevantes para sociedade, uma vez que, ao se relacionar diretamente à agricultura, relaciona-se também às questões ambientais e à segurança alimentar. O uso de técnicas de reconhecimento avançadas permite identificar pragas de forma mais rápida e precisa, permitindo a implementação de medidas preventivas e o uso eficiente de recursos, incluindo defensivos agrícolas, possibilitando o aumento da produção e redução dos impactos ambientais, uma vez que, ao reduzir o uso de defensivos, a probabilidade de contaminação de solo e água, a seleção de pragas resistentes e o extermínio de agentes polinizadores é reduzido. Além disso, a proposta apresentada promove a inovação tecnológica no setor agrícola, aproximando-o das tendências atuais. Por fim, é válido citar as contribuições pessoais que o trabalho trouxe ao estudante que, além de melhorar suas habilidades de programação, pesquisa, experimentação e análise crítica, pôde ter contato com a resolução de um problema real e atual.

REFERÊNCIAS BIBLIOGRÁFICAS

AMAZON.COM. **AWS**, 2022. O que é overfitting? Disponível em: <https://aws.amazon.com/pt/what-is/overfitting/>. Acesso em: 02 Out. 2024. Citado 2 vezes nas páginas 26 e 27.

ANIEMEKA, Ifu. A Friendly introduction to Convolutional Neural Networks. **Hashrocket**, 2017. Disponível em: <https://hashrocket.com/blog/posts/a-friendly-introduction-to-convolutional-neural-networks>. Acesso em 02 out. 2024. Citado duas vezes nas páginas 23 e 24.

ARAMENDIA, Alejandro Ito. Convolutional Neural Networks (CNNs): A Complete Guide, 2024. Disponível em: <https://medium.com/@alejandro.itoaramendia/convolutional-neural-networks-cnns-a-complete-guide-a803534a1930>. Acesso em: 02 Out. 2024. Citado uma vez na página 25.

BASAVARAJIAH, Madhushree. 6 basic things to know about Convolution. **Medium**, 2019. Disponível em: <https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411>. Acesso em 02 out. 2024. Citado uma vez na página 18.

COELHO, F. **Computação Científica com Python**. 1. ed. Petrópolis: [s.n.], 2007. Citado 1 vez na página 29.

CONAB – COMPANHIA NACIONAL DE ABASTECIMENTO. Acompanhamento da Safra Brasileira de Grãos, Brasília, DF, v. 11, safra 2023/24, n. 8 oitavo levantamento, maio 2024. Citado uma vez na página 12.

CONAB – COMPANHIA NACIONAL DE ABASTECIMENTO. Evolução dos custos de produção de soja no Brasil. **Compêndio de Estudos Conab**, Brasília, DF, v. 2, 2016. Citado uma vez na página 12.

Convolutional Neural Networks (CNN): Step 4 – Full, SuperDataScience, 2018. Disponível em: <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection>. Acesso em: 02 Out. 2024. Citado uma vez na página 24.

CYPRIANO, Maycon, Pesos - Aquilo que define o "shape" da sua aplicação. **Linkedin**, 2024. Disponível em: <https://pt.linkedin.com/pulse/pesos-aquilo-que-define-o-shape-da-sua-aplicação-maycon-cypriano-ufhqf>. Acesso em: 13 Out. 2024. Citado uma vez na página 27.

DA CUNHA, Leandro Cardoso. **REDES NEURAIAS CONVOLUCIONAIS E SEGMENTAÇÃO DE IMAGENS – UMA REVISÃO BIBLIOGRÁFICA**. Monografia (Graduação em Engenharia de Controle e Automação) – Universidade Federal de Ouro Preto. Ouro Preto, p. 55, 2020. Citado 6 vezes nas páginas 20, 21, 22 e 24.

DA SILVA, B. F.; CAVICHIOLI, F. A. O uso de veículos aéreos não tripulados para detecção de pragas e doenças na cultura da soja. **Revista Interface Tecnológica**, v. 19, n. 1, p. 236-247, 2022. Citado 3 vezes nas páginas 11 e 15.

DATA SCIENCE ACADEMY. **Deep Learning Book**, c2022. Capítulo 23 – Como Funciona o Dropout. Disponível em: <https://www.deeplearningbook.com.br/capitulo-23-como-funciona-o-dropout/>. Acesso em: 02 Out. 2024. Citado uma vez na página 27.

DATA SCIENCE ACADEMY. **Deep Learning Book**, c2022. Capítulo 43 – Camadas de Pooling em Redes Neurais Convolucionais. Disponível em: <https://www.deeplearningbook.com.br/camadas-de-pooling-em-redes-neurais-convolucionais/>. Acesso em: 02 Out. 2024. Citado 2 vezes nas páginas 23 e 24.

DATA SCIENCE ACADEMY. **Deep Learning Book**, c2022. Capítulo 8 – Função de Ativação. Disponível em: <https://www.deeplearningbook.com.br/funcao-de-ativacao/>. Acesso em: 02 Out. 2024. Citado uma vez na página 23.

DESHPANDE, Adit. A Beginner's Guide to Understanding Convolutional Neural Networks. Adit Deshpande Blog. 2016. Disponível em: <https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>. Acesso em: 02 Out. 2024. Citado 3 vezes nas páginas 22, 24 e 25.

DOS SANTOS, A. O.; CAVIOCHIOLI, F. A.; SOARES, N. M.; CLAUDINO, T. S. AGRICULTURA 4.0: uso de drones. **SIMTEC - Simpósio de Tecnologia da Fatec Taquaritinga**, v. 6, n. 1, p. 38-47, 22 dez. 2019. Citado uma vez na página 11.

ESPERIDIÃO, T. L.; DOS SANTOS, T. C.; AMARANTE, M. DOS S. AGRICULTURA 4.0: SOFTWARE DE GERENCIAMENTO DE PRODUÇÃO. **Revista Pesquisa e Ação**, v. 5, n. 4, p. 122-131, 5 dez. 2019. Citado uma vez na página 11.

FALCÃO, João Vitor Regis et al. Redes neurais deep learning com tensorflow. **RE3C-Revista Eletrônica Científica de Ciência da Computação**, v. 14, n. 1, 2019. Citado 2 vezes nas páginas 29 e 30.

FERREIRA, Alessandro dos Santos. **Redes neurais convolucionais profundas na detecção de plantas daninhas em lavoura de soja**. Tese (Mestrado em Ciências da Computação) – Faculdade de Ciências da Computação, Universidade Federal do Mato Grosso do Sul. Campo Grande, p.70, 2017. Citado duas vezes nas páginas 17 e 19.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep Learning. MIT Press. 2016. Citado 2 vezes nas páginas 20 e 22.

GOOGLE.COM. **Google Cloud**, 2023. Inteligência artificial (IA) x machine learning (ML). Disponível em: <https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning?hl=pt-BR>. Acesso em: 15 Out. 2024. Citado uma vez na página 16.

GOOGLE.COM. **Google for Developers**, c2024. Curso intensivo de machine learning. Disponível em: <https://developers.google.com/machine-learning/crash-course?hl=pt-br>. Acesso em: 12 Out. 2024. Citado duas vezes na página 59.

GOOGLE.COM. **Google for Developers**, c2024. Overfitting. Disponível em: <https://developers.google.com/machine-learning/crash-course/overfitting/overfitting?hl=pt-br>. Acesso em: 02 Out. 2024. Citado 4 vezes na página 26.

GOOGLE.COM. **Google for Developers**, c2024. Regressão linear: hiperparâmetros. Disponível em: <https://developers.google.com/machine-learning/crash-course/linear-regression/hyperparameters?hl=pt-br>. Acesso em: 13 Out. 2024. Citado uma vez na página 28.

GÖRNER, Martin. Modern Convolutional Neural Nets. 2018. Apresentação de

slides. Disponível em: <https://github.com/GoogleCloudPlatform/tensorflow-without-a-phd>. Acesso em: 02 Out. 2024. Citado uma vez na página 23.

IBM.COM. **IBM**, 2024. O que é uma rede neural? Disponível em: <https://www.ibm.com/br-pt/topics/neural-networks>. Acesso em: 20 mai. 2024. Citado 3 vezes na página 17.

JESUS, Edison O.; COSTA JR, Roberto. A utilização de filtros gaussianos na análise de imagens digitais. **Proceeding Series of the Brazilian Society of Computational and Applied Mathematics**, v. 3, n. 1, 2015. Citado 1 vez na página 18.

JORDAN, Jeremy. Convolutional Neural Networks. **Jeremy Jordan**, 2017. Disponível em: <https://www.jeremyjordan.me/convolutional-neural-networks/>. Acesso em: 02 Out. 2024. Citado 4 vezes nas páginas 21 e 25.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*. v. 60. n. 6. maio 2017. Citado uma vez na página 22.

LAMBERT, Kenneth A. **Fundamentos de Python: primeiros programas**. São Paulo: Cengage Learning Brasil, 2022. *E-book*. ISBN 9786555584301. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9786555584301/>. Acesso em: 17 mai. 2024. Citado 1 vez na página 28.

LEMO, Mario Luiz Freitas et al. Agregação de valor na cadeia de soja. 2017. Citado uma vez na página 12.

LISBINSKI, Fernanda Cigainski et al. Perspectivas e desafios da Agricultura 4.0 para o setor agrícola. *Anais do VIII Simpósio da Ciência do Agronegócio*, 2020. Citado uma vez na página 14.

MARQUIS, P.; PARDE, H.; PAPINI, O. A Guided Tour of Artificial Intelligence Research: Volume III: Interfaces and Applications of Artificial Intelligence. [S.l.]: Springer Cham, 2020. Citado 1 vez na página 16.

ML | Common Loss Functions. **GeeksforGeeks**, 2024. Disponível em:

<https://www.geeksforgeeks.org/ml-common-loss-functions/>. Acesso em: 13 Out. 2024. Citado uma vez na página 28.

OLIVEIRA, Caio Rezende de. A Importância da Agricultura 4.0 para o aumento da produtividade. 2022. Citado 3 vezes nas páginas 15 e 16.

RODRIGUES, Diego Alves. **DEEP LEARNING E REDES NEURAIS CONVOLUCIONAIS: RECONHECIMENTO AUTOMÁTICO DE CARACTERES EM PLACAS DE LICENCIAMENTO AUTOMOTIVO**. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Centro de Informática, Universidade Federal da Paraíba. João Pessoa, p. 37, 2018. Citado 2 vezes na página 19.

SALVADORI, Marcelo Ferreira. Aplicação de machine learning para detecção de pragas em plantas. 2024. Citado uma vez na página 12.

TENSORFLOW. **TensorFlow**, [s.d.] Learnin. Disponível em: <https://www.tensorflow.org/learn?hl=pt-br>. Acesso em: 20 mai. 2024. Citado uma vez na página 29.

TENSORFLOW. **TensorFlow**, c2020. images.ipynb. Disponível em: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/load_data/images.ipynb?hl=pt-br#scrollTo=mt9dL5dlir8X. Acesso em: 05 Out. 2024. Citado duas vezes nas páginas 31 e 35.

TETILA, E. C. **Detecção e classificação de doenças e pragas da soja usando imagens de veículos aéreos não tripulados e técnicas de visão computacional**. Tese (doutorado) – Universidade Católica Dom Bosco. Campo Grande, p.104, 2019. Citado 3 vezes nas páginas 12 e 13.

VILLAFUERTE, Andrés et al. Agricultura 4.0: estudo de inovação disruptiva no agronegócio brasileiro. In: **International Symposium on Technological Innovation**. 2018. p. 150-162. Citado 4 vezes nas páginas 14, 15, 16 e 17.

What Is a Convolutional Neural Network? **MathWorks**, 2017. Disponível em: https://www.mathworks.com/discovery/convolutional-neural-network.html?s_tid=srchtitle_site_search_1_convolution%20neural%20network. Acesso em 02 out. 2024. Citado uma vez na página 19.

What Is Cross-Entropy Loss Function? **GeeksforGeeks**, 2024. Disponível em: <https://www.geeksforgeeks.org/what-is-cross-entropy-loss-function/>. Acesso em: 13 Out. 2024. Citado uma vez na página 28.

APÊNDICE A

Os algoritmos desenvolvidos para este trabalho foram publicados em <https://github.com/gabrielpoli30/TCC.git>.