Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA ANÁLISE DOS ESTÁGIOS DE DESENVOLVIMENTO DA SOJA

Gabriel Chiodi Ibanez

Dr. Evandro Cesar Bracht (Orientador)

Dourados – MS
2025

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA ANÁLISE DOS ESTÁGIOS DE DESENVOLVIMENTO DA SOJA

Gabriel Chiodi Ibanez

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Gabriel Chiodi Ibanez e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, 16 de Outubro de 2025

Prof. Dr. Evandro Cesar Bracht (Orietador)

I21d Ibanez, Gabriel Chiodi

Desenvolvimento de uma aplicação para análise dos estágios de desenvolvimento da soja / Gabriel Chiodi Ibanez. – Dourados, MS: UEMS, 2025.
71 p.

Monografia (Graduação) – Sistemas de Informação – Universidade Estadual de Mato Grosso do Sul, 2025.
Orientador: Prof. Dr. Evandro Cesar Bracht.

1. Processamento de imagens 2. Redes neurais convolucionais 3. *Deep learning* 4. Soja - Estágios de crescimento I. Bracht, Evandro Cesar II.Título
CDD 23. ed. - 006.31

Ficha Catalográfica elaborada pela bibliotecária da Universidade Estadual de Mato Grosso do Sul (UEMS)
Bruna Peruffo Vieira – CRB 1/2959

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul

DESENVOLVIMENTO DE UMA APLICAÇÃO PARA ANÁLISE DOS ESTÁGIOS DE DESENVOLVIMENTO DA SOJA

Gabriel Chiodi Ibanez

Novembro de 2025

Banca Examinadora:

Prof. Dr. Evandro Cesar Bracht (Orientador)
Área de Computação – UEMS

Prof. Dr. Cleber Valgas Gomes Mira
Área de Computação – UEMS

Profa. Dra. Mercedes Rocío Gonzales Márquez
Área de Computação - UEMS

Dedico este trabalho a Deus, aos meus pais por todo amor e apoio, ao meu irmão, pelo importante papel no projeto, e à minha futura esposa, Gabriela.

“A verdade só está a serviço de seus escravos.”
(Antonin Gilbert Sertillanges)

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida, por todo amor e sabedoria a mim entregue, por todo caminho que trilhei, todos os frutos que colhi e todo aprendizado que adquiri.

Agradeço aos meus pais, Ademir Ferreira Ibanez e Sirlei Chiodi Ibanez, pela excelente criação e pelos valores a mim transmitidos, por todo amor, apoio e confiança, que foram e são fatores imprescindíveis, além de todo o suporte financeiro.

Agradeço ao meu irmão, por todo carinho e parceria ao longo da vida e também por seu papel importante na execução deste trabalho, dedicando seu tempo à coleta de imagens da plantação de soja durante todo o ciclo da planta.

Agradeço à minha futura esposa, Gabriela Regina Grance Franco, pelo amor, fidelidade, companheirismo e paz, pela tranquilidade que permite minha alta produtividade, por me motivar quando parto e me acolher quando retorno; sem ela, eu não teria conhecido este curso.

Agradeço, por fim, aos meus colegas, que me acompanharam do início ao fim do curso, aos doutores docentes do curso de Sistemas de Informação, por serem excelentes profissionais, em especial ao meu orientador, Professor Doutor Evandro Cesar Bracht, que muito colaborou com minha formação.

RESUMO

O agronegócio é um dos maiores setores produtivos brasileiro e o cenário no Mato Grosso do Sul tem se tornado cada vez mais relevante no âmbito nacional pela sua produção anual, principalmente na cultura da soja. A utilização da agricultura de precisão e suas ferramentas são indispensáveis para otimizar a produção. O desenvolvimento de um software para analisar os estágios de desenvolvimento da soja poderia colaborar para a tomada de decisão, diminuindo os custos de produção e tornando o processo menos burocrático e mais eficiente. Este trabalho visa desenvolver uma ferramenta que utilize redes neurais convolucionais para identificar padrões em imagens a fim de detectar os estágios de desenvolvimento da planta.

PALAVRAS-CHAVE: Soja, Processamento de imagens, Redes Neurais Convolucionais, *Deep Learning*, Estágios de crescimento da planta.

ABSTRACT

Agribusiness is one of the largest productive sectors in Brazil, and the scenario in Mato Grosso do Sul has become increasingly relevant on the national stage due to its annual production, especially in soybean cultivation. The use of precision agriculture and its tools is essential to optimize production. Developing software to analyze the developmental stages of soybeans would aid in decision-making, reducing production costs and making the process less bureaucratic and more efficient. This work aims to develop a tool that uses convolutional neural networks to identify patterns in images in order to detect the developmental stages of the plant.

KEYWORDS: Soybean, Image Processing, Convolutional Neural Networks, Deep Learning, Plant Growth Stages.

SUMÁRIO

1. INTRODUÇÃO

O Brasil vem se destacando mundialmente por ser uma potência no agronegócio. Atualmente é o maior produtor de soja no mundo, com uma produção estimada de 153.000 toneladas, na segunda posição aparecem os Estados Unidos com 113.000 toneladas produzidas, em terceiro lugar fica a Argentina que produziu 48.000 toneladas, seguidas pela China, com cerca de 20.000 toneladas e pela Índia com 11.000 toneladas, dados aproximados conforme levantamento pelo Departamento de Agricultura dos Estados Unidos (USDA, 2024). Com destaque para as cidades de Maracaju e Dourados, o Mato Grosso do Sul ocupa o 5° lugar no *ranking* nacional de produção. Segundo dados da Companhia Nacional de Abastecimento (CONAB, 2024), a produção total do estado na safra 2023/2024 foi 19.966,2 toneladas, com a soja representando 56,67% desse montante.

O desenvolvimento de tecnologias para auxiliar na produção da soja é uma demanda crescente no país, pois com o aumento no número de produtores, tende a aumentar a competitividade no mercado, tornando necessária a busca por qualquer auxílio tecnológico a fim de otimizar os processos no campo.

A agricultura de precisão (AP) surge como metodologia indispensável em um cenário de alta competitividade. “É um conjunto de tecnologias destinadas ao manejo de solos, culturas e insumos, que visa um melhor e mais detalhado gerenciamento do sistema de produção agrícola em todas as etapas, desde a semeadura até a colheita” (Inamasu et al., 2011, p. 32). Ela não só objetiva o lucro, ao minimizar os gastos de aplicação de insumos através da tomada de decisão advinda da análise detalhada dos dados levantados, como também aumenta a sustentabilidade ambiental, econômica e social, por evitar o excesso da aplicação de defensivos agrícolas, corretivos de solo, entre outros.

“A agricultura de precisão tem três componentes: captação de dados em uma escala e frequência adequada, interpretação e análise desses dados, gestão e implementação de uma resposta a uma escala espacial e de tempo adequada.” (Inamasu et al., 2011, p. 16).

O primeiro componente descrito refere-se à etapa de captação dos dados, que envolve as tecnologias para análise e levantamento das informações espaciais, com a utilização de *hardwares*, como os diversos tipos de sensores embarcados em máquinas, posicionadas em torres ou utilizando o meio aéreo com helicópteros ou VANTs (Veículos aéreos não tripulados).

No segundo componente, onde é realizado a análise dos dados, uma gama de *softwares* são utilizados, que consideram as informações obtidas das diversas fontes descritas, processam-nas, realizam o tratamento adequado e fazem a transmissão.

Por fim, no terceiro componente, as informações transmitidas na etapa de análise e tratamento, são utilizadas de forma precisa com maquinários para implementação em tempo real no campo, na aplicação de insumos de acordo com a variação necessária.

Um sistema com o objetivo de deduzir o estágio de crescimento de uma cultura faz parte de uma ferramenta para a AP, sendo um componente para captação e análise dos dados. Essa ferramenta pode auxiliar o agricultor na tomada de decisão, pois é capaz de identificar o estágio exato do crescimento da planta, em uma determinada área, em um determinado momento, o que possibilita determinar quais insumos aplicar, evitando desperdícios, maximizando lucros e contribuindo com o meio ambiente.

A fim de identificar o estágio da soja, o trabalho de Zhang e outros obteve êxito utilizando uma abordagem mais simples, como uma rede neural artificial treinada apenas com dados numéricos da data do plantio e maturação da planta em cada etapa (Zhang et al., 2009). Chamara e outros utilizaram uma rede neural convolucional pré-treinada para determinar o estágio de desenvolvimento da planta, e obtiveram êxito apenas com 116 imagens coletadas no campo (A.H.M et al., 2021).

Para desenvolver tal sistema, é necessário treinar uma IA com o uso de *deep learning* (aprendizado profundo) utilizando imagens de plantações dos diversos estágios como entrada. Uma rede neural convolucional é um algoritmo de *deep learning* focado no processamento de imagens, sendo o mais eficiente modelo para reconhecer padrões em dados de imagens.

Este trabalho propõe o desenvolvimento de um *software* que possa processar os dados de imagens utilizando uma rede neural convolucional, que possa deduzir o estágio de desenvolvimento da soja, e que apresente os resultados de forma intuitiva em uma interface para o usuário, possibilitando a tomada de decisões.

1.1 Justificativa

O grande potencial produtivo nas cidades de Maracaju, Ponta Porã, Sidrolândia, Dourados, Rio Brilhante, dentre outras no estado do Mato Grosso do Sul demandam tecnologias de ponta para aumentar a produtividade (kg/ha), o lucro e a sustentabilidade. Na contrapartida, embora tenha apresentado uma evolução nos últimos anos, a agricultura de precisão ainda tem baixa adesão pelos agricultores, seja por desconhecimento, por ainda ser

um investimento relativamente alto, ou até pela falta de mão de obra qualificada para lidar com os instrumentos tecnológicos.

O estudo e desenvolvimento de uma ferramenta para a AP, que demande baixo custo de investimento e que possa ser utilizada com facilidade por pessoas ou lida por outros softwares, contribuirá com a difusão de conhecimento acerca do tema, divulgação da AP e com o meio ambiente.

1.2 Objetivos

O objetivo deste trabalho é desenvolver uma ferramenta que, com base em imagens da soja, deduza qual o estágio de desenvolvimento a planta se encontra, que não necessite de muita instrução de manuseio, contribuindo com a aquisição de conhecimentos pessoais, a transmissão de informações e tecnologias, e corroboração com a divulgação da agricultura de precisão.

Objetivos específicos:

- Realizar uma ampla revisão bibliográfica em livros, artigos e publicações acerca do tema proposto;
- Estudar a linguagem de programação Java;
- Estudar as técnicas de aprendizagem de máquinas, redes neurais e inteligência artificial;
- Contribuir com a divulgação e acessibilidade da agricultura de precisão;

2. REVISÃO LITERÁRIA

2.1 A soja

O primeiro teste de plantação de soja no Brasil, ocorreu em 1882 na região da Bahia, e apesar de não obter êxito naquele primeiro momento, os estudos para viabilização continuaram nas décadas seguintes. “A primeira referência de produção comercial de soja no Brasil data de 1941 (área cultivada de 640 ha, produção de 450 toneladas e rendimento de 700 kg/ha)” (Dall’Agnol, 2011). Nos anos 1950 e 1960, houve um crescimento da produção de soja nos 3 estados da região sul brasileira, devido às condições climáticas mais favoráveis ao grão, tornando a cultura importante no país. Na década de 1970, a produção já chegava a 1500 toneladas, e no final da década, esse valor era 10 vezes maior e se tornava a principal lavoura no Brasil. Nesse mesmo período, depois de pesquisas para adequação da cultura em outras regiões, foram alteradas algumas características genéticas da planta para localizações mais tropicais de baixa latitude, possibilitando a produção em todo o território nacional.

A produção de soja trouxe diversos benefícios para o Brasil, como a aceleração da mecanização das lavouras, modernização dos sistemas de transporte, profissionalização do comércio internacional e descentralização da agroindústria que era concentrada no sul e sudeste, ajudando a povoar o interior no centro-oeste brasileiro.

> “A revolução socioeconômica e tecnológica protagonizada pela soja no Brasil Moderno, pode ser comparada aos fenômenos ocorridos com os ciclos da cana de açúcar, da borracha, do cacau e do café, que, em distintos períodos dos séculos XVII a XX, comandaram o comércio exterior do Brasil.” (Dall’Agnol, 2011).

A soja, de nome científico *Glycine max (L.) Merrill,* é uma das culturas mais importantes do mundo na atualidade, e seu óleo é o mais consumido pela população mundial para preparação de alimentos. O farelo da soja é amplamente utilizado para ração de animais, o que colabora também para alimentação da população indiretamente. O biodiesel é outro subproduto, cuja demanda vem aumentando, assim como a procura do grão pela indústria de cosméticos. Outros subprodutos da soja, incluem o sabão, tintas e solventes.

Nos últimos 5 anos, o Brasil tornou-se o maior produtor de soja do mundo, quando assumiu esta posição na safra 2019/2020 ao produzir aproximadamente 30 milhões a mais que o segundo lugar, os EUA, que vinha ocupando o topo até então. Segundos os dados apontados pelo (USDA, 2024), na safra 2023/2024, a produção brasileira representou 39% da produção mundial com cerca de 153 milhões de toneladas, e quase 58% das exportações mundiais partem do Brasil. A Figura 1 demonstra a posição dos 5 maiores produtores mundiais.

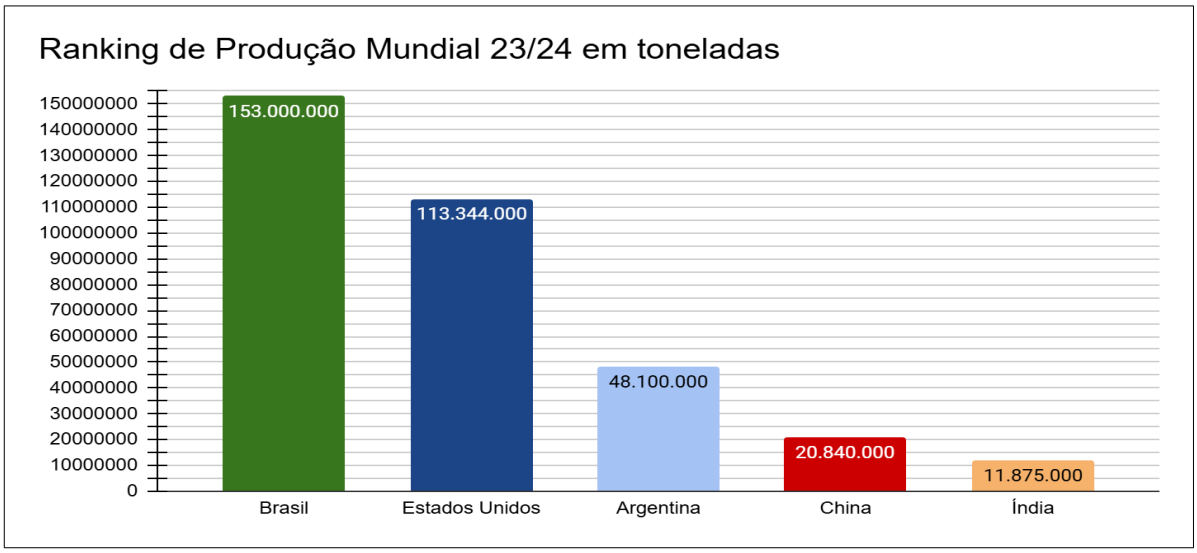


Figura 1 – Quantidade de soja produzida pelos 5 maiores produtores de soja do mundo. Fonte: (USDA, 2024).

Na corrida interna de produção, o estado que lidera o ranking é o Mato Grosso. O Mato Grosso do Sul vem apresentando um potencial crescente nestes últimos anos, com 11.315 milhões de toneladas de soja produzidas, quantidade comparável com a Índia, o estado já ocupa a quinta posição do país na produção deste grão (CONAB, 2024). A cidade de Maracaju é a maior produtora do estado, sendo a única cidade a apresentar uma taxa de produção acima de 1 milhão de toneladas, sempre se posicionando entre os 10 municípios que mais produzem soja no Brasil. Segundo o boletim técnico Casa Rural apresentado pela Federação da Agricultura e Pecuária Mato Grosso do Sul (FAMASUL, 2024), os municípios de Maracaju, Dourados, Sidrolândia, Ponta Porã e Rio Brilhante são os principais produtores de soja do estado. A Figura 2 representa a classificação dos estados por toneladas de soja produzidas.

Figura 2 – Quantidade de soja produzida pelos 5 maiores produtores de soja do Brasil. Fonte: (CONAB, 2024).

2.2 Agricultura de precisão

Ao contrário do senso comum de que a agricultura de precisão esteja 100% relacionada à utilização de tecnologias de automação e computadorização no meio agrícola, na verdade, o conceito da AP está mais relacionado com a gestão dos processos no campo, podendo ser aplicada tanto nas lavouras como nas culturas animais. “A AP tem como foco a gestão de sistema produtivo agrícola considerando a variabilidade espacial e temporal visando minimizar efeito negativo ao meio ambiente e maximizar retorno econômico.” (Inamasu et al., 2011). A AP pode ser realizada até de forma totalmente manual, com caneta e papel, desde que tenha um controle amplo, como por exemplo, saber o estágio de crescimento atual da planta, em qual área da plantação é necessário aplicar determinado insumo e quando é o momento correto para aplicação.

No contexto atual, de grande demanda externa de *commodities*, como a soja, o número de áreas plantadas do grão quase que dobrou nos últimos dez anos apenas no estado do Mato Grosso do Sul, passando de 2,100 milhões para 4,214 milhões de hectares (FAMASUL, 2024), o que torna muito caro e prolongado para coletar dados de forma manual, sendo necessário a utilização de ferramentas para a AP, para otimizar os processos e facilitar a gestão pelo agricultor.

As ferramentas para AP são as diversas formas existentes para coletar os dados, fazer a leitura e tratamento de forma adequada, e fazer a aplicação correta de insumos com base nas

informações obtidas. Um *software* com a capacidade de processar imagens obtidas do campo e indicar qual é o estágio da planta é uma ferramenta para AP, pois ajuda o produtor a decidir qual insumo aplicar naquele determinado momento.

Nos últimos 15 anos, vem havendo grandes esforços do MAPA (Ministério da Agricultura, Pecuária e Abastecimento) junto a empresas públicas e de pesquisas como a EMBRAPA, que vem sendo uma grande impulsionadora, e outras entidades privadas para promover a AP, criando órgãos como a CBPA (Comissão Brasileira de Agricultura de Precisão), fomentando eventos, como os que ocorrem em Não-Me-Toque – Rio Grande do Sul, e incentivando a educação, porém sua adoção ainda vem ocorrendo em ritmo inferior à expansão da produção (Bernardi; Inamasu, 2014). Apesar da necessidade e importância da utilização desse meio de gestão, ela é amplamente utilizada apenas nos pilotos automáticos em maquinários e para aplicação de insumos. Em uma pesquisa em 2014 da adoção da AP no Brasil, pesquisadores da EMBRAPA concluem “o perfil dos proprietários e administradores de propriedades que adotam a AP é jovem, instruído, propenso a utilizar mais tecnologias e informática e cultivam grandes extensões de terras, e mesmo dentre esses, os equipamentos de AP são subutilizados.” (Bernardi; Inamasu, 2014, p. 17). Em visitas realizadas por técnicos da (FAMASUL, 2024) a 447 produtores no estado do MS, em maio de 2024, foi constatado que 70% fazem o uso da AP, mas destes, 39% usam apenas para fertilização, 15% usam para plantio, fertilização e aplicação, 11% para fertilização e plantio, e 4% utilizam apenas no plantio, os 30% restantes não utilizam a AP. Também foi relatado que 72% destes produtores não utilizam softwares para gestão da lavoura.

Um dos principais entraves para a adesão ao uso da AP no Brasil é a falta de mão de obra qualificada para trabalhar com as ferramentas. Outros aspectos negativos são a fama de possuir um custo elevado e o desconhecimento das inovações atuais. Silva e Silva-Mann pontuaram que a complexidade de manuseio dos equipamentos de coleta, processamento e apresentação dos dados sobre as colheitas realizadas pelos maquinários da AP, e o alto custo e desempenho exigidos pelos sistemas de processamento da época colaboraram para o atraso da adoção das práticas (Silva; Silva-Mann, 2020).

2.3 Alguns trabalhos semelhantes

Dentro do campo de estudo da utilização de imagens de plantações e redes neurais para analisar e detectar diversas variáveis, como o crescimento, doenças e agentes

patogênicos, insetos, ervas daninhas, etc., a detecção do crescimento é uma abordagem mais direta, envolvendo menos fatores a considerar, tornando a análise relativamente mais simples.

2.3.1 Predição do crescimento com redes neurais e estatística

Zhang e outros propuseram um estudo visando a simplicidade e a efetividade para prever as datas dos estágios de crescimento fenológicos da soja utilizando 3 métodos distintos para comparar o desempenho: rede neural com retro propagação, regressão por passo e interpolação. Em sua proposta, buscaram utilizar poucas variáveis, para que o projeto fosse acessível aos agricultores, demandando apenas dados numéricos como entrada. No estudo, os estágios de crescimento foram separados em 2 períodos: vegetativo e reprodutivo. Os 10 primeiros estágios de crescimento foram atribuídos ao vegetativo utilizando apenas a data de plantação e o valor médio normalizado entre a data de plantação e o número de dias até chegar no determinado estágio como parâmetros significativos. Os 8 demais foram definidos como reprodutivo, nesse foi acrescentado o parâmetro de maturidade da planta (tempo que leva do plantio até a maturidade de determinada planta). Os autores concluíram então que a utilização de redes neurais artificiais oferecem um alto grau de precisão e portanto podem ser aplicados de forma eficaz na modelagem de culturas (Zhang et al., 2009).

2.3.2 Metodologias para análise de imagens com aprendizado profundo

Ahmad e outros organizaram um estudo mais complexo analisando 70 trabalhos sobre a utilização de redes neurais na agricultura para identificação de doenças, obtendo conclusões importantes para nortear um projeto com o uso de redes neurais e sensores para coleta de imagens. Dentre os diversos temas abordados, os tópicos sensores de imagens, aprendizado profundo e a comparação entre aprendizado profundo e precisão humana são relevantes para o presente trabalho.

Para os autores, a utilização de CNN (*Convolutional neural network* - Redes neurais convolucionais) obteve um resultado satisfatório para identificação de cores e texturas para detectar doenças nas plantas, e ainda obteve um resultado semelhante mesmo retirando 75% dos parâmetros iniciais. Ele ressaltou a importância do estudo adequado para entender as técnicas, como extrair as informações e as métricas de avaliação. Dentre as ferramentas utilizadas com CNN, a classificação de imagem se destacou pela alta acurácia, e por poder ser utilizada com uma técnica de aprendizado por transferência, que consiste em adaptar e refinar

um modelo pré treinado em um grande conjunto de imagens, o que é ideal para quando existem poucos dados disponíveis, ou o prazo de projeto é curto.

> "Dos 70 estudos, 43 utilizaram classificação de imagens. Entre os estudos revisados, a classificação de imagens foi a técnica de *deep learning* mais comumente utilizada para a identificação de doenças em plantas, e o aprendizado por transferência foi empregado na maioria dos estudos." (Ahmad; Saraswat; El Gamal, 2023).

No que tange à comparação com o desempenho humano para levantamento dos dados de doenças, os autores chegaram à conclusão que na maioria dos casos, as técnicas de aprendizado profundo obtiveram um resultado superior a 94 %, que é a acurácia humana para o mesmo serviço. Ainda assim, consideráveis casos performaram de forma inferior à classificação manual humana, porém, foi destacado que nestes casos, os algoritmos consideravam um conjunto de dados envolvendo um grupo abrangente de culturas ou que as classes do conjunto de dados estavam desbalanceadas, ou seja, não possuíam a mesma quantidade de dados de imagens em cada classe, relatando a importância de focar em poucas culturas, e de manter o conjunto de dados de análise balanceado.

Dentre os sensores utilizados, destacam-se os sensores RGB (Red, Green, Blue – Vermelho, Verde, Azul), e os sensores hiperespectrais. Os sensores RGB foram amplamente utilizados pelos pesquisadores para detectar doenças em plantas, por extrair características importantes no reconhecimento de doenças e por sua acessibilidade, tendo baixo custo de aquisição, o que faz ser utilizado em larga escala, principalmente com o advento dos *smartphones*. Os sensores hiperespectrais, por outro lado, apresenta um custo elevado, sendo comumente utilizado nos VANTs, porém sua tecnologia permite a detecção de infravermelho próximo e bordas vermelhas, sendo capazes de apontar características importantes, como as doenças, estresse hídrico, informações nutricionais, ervas daninhas, entre outros (Ahmad; Saraswat; El Gamal, 2023).

2.3.3 Utilização de redes neurais convolucionais para monitoramento da soja

Em sua pesquisa, Chamara e outros estudaram a identificação de estágios da planta utilizando ferramentas baratas, como o *Raspberry PI*, um pequeno hardware computador, com uma câmera RGB acoplada, embarcado em um veículo aéreo não especificado, para captura das imagens em alta definição, armazenadas em um cartão SD. Os autores também utilizaram técnicas de CNN, para análise das imagens.

Os autores obtiveram no total 119 imagens rotuladas para o estudo e utilizaram técnicas de aprendizado por transferência, e uma CNN pré treinada com o extenso conjunto de imagens rotuladas do ImageNET, para remover ruídos das imagens, diminuindo a resolução das imagens para 512x512. Com taxa de 94% de precisão, foi utilizada a técnica de segmentação semântica para analisar cada pixel das imagens, a fim de detectar o dossel das plantas, isso é, a estrutura visível da soja, como caule, ramos e folhas. A técnica de segmentação possibilitou obter aspectos importantes de coloração da planta, melhorando a identificação do estágio do crescimento com base em índices vegetativos. Uma técnica de detecção de objeto com CNN foi utilizada para contar a quantidade de folhas, mas nesse caso foi atingido uma precisão de apenas 36%, os autores concluíram que essa técnica é promissora, mas necessita de uma quantidade muito maior de dados, e imagens com resoluções maiores (A.H.M et al., 2021).

2.4 Inteligência artificial

Quando iniciou os debates da utilização de máquinas com inteligência artificial, Alan Turing propôs o famoso “teste de Turing”, e a partir deste ponto, os estudos para IA (Inteligência artificial) ganharam atenção na comunidade acadêmica. O teste de Turing consiste em um avaliador humano conversar com outro humano e com uma máquina, e quando não for possível distinguir quem é o humano, a máquina passa no teste. Desde que iniciou a abordagem do tema, a humanidade conseguiu grandes avanços, principalmente nos últimos anos, com o surgimento de tecnologias para processamento de texto em linguagem natural, para conversa e interação com humanos como Chat GPT, plataformas para processamento de áudio e reconhecimento de voz, criação de imagens, vídeos, etc…

No início dos estudos, nas décadas de 1950 e 1960, as pesquisas eram concentradas todas na academia, pois havia medo por parte da população da substituição do trabalho humano por robôs com IA, provocando incertezas no mercado quanto a investimentos na área, assim todo investimento provinha de instituições governamentais e de fomento a pesquisa. Alguns trabalhos importantes foram realizados na época e surgiram alguns programas buscando alcançar um nível de IA forte, como o Student, em 1965, que podia resolver problemas de álgebra de nível de ensino médio. Em 1965 surgira o primeiro protótipo de chat que conversava com humanos, chamado de Elisa, que respondia perguntas com conselhos. No ano seguinte, foi desenvolvido o primeiro programa de visão computacional por um

estudante, onde foi possível identificar padrões em imagens reproduzidas por uma câmera ligada a um computador.

Em 1957, Frank Rosemblat criou o Mark 1 Perceptron, primeiro programa de inteligência artificial baseado em redes neurais, que diferenciou duas imagens 20x20, e era treinado utilizando dados com ponderações aleatórias. O programa recebia uma entrada e produzia uma saída dos perceptrons (neurônios), se a saída não fosse correta, e deveria ter sido 0, mas foi 1, o peso de 1 era decrementado ou se a saída não fosse correta, e deveria ter sido 1, mas foi 0, o peso de 1 era incrementado. Assim o processo era repetido até que o programa respondesse com precisão.

Nas décadas seguintes, conhecidas como Inverno da IA, houve um grande desânimo por parte das instituições investidoras, pois os resultados até então não atendiam às grandes expectativas geradas. Porém na academia, alguns ainda acreditavam e continuavam seus estudos, trazendo resultados importantes que serviriam de base para utilização com hardwares atuais.

Dentro do campo de estudo de inteligência artificial, estão os conceitos de *machine learning* (aprendizado de máquina) e *deep learning*. O *machine learning* consiste em um processo de identificação de padrões em um grande conjunto de dados, podendo utilizar técnicas como redes neurais, regressão linear, entre outras. O *deep learning*, é um tipo *machine learning* que utiliza redes neurais profundas com múltiplas camadas ocultas para resolver problemas mais complexos. Diversos autores costumam usar o diagrama de Venn, conforme a Figura 3, para exemplificar a relação dos conceitos.

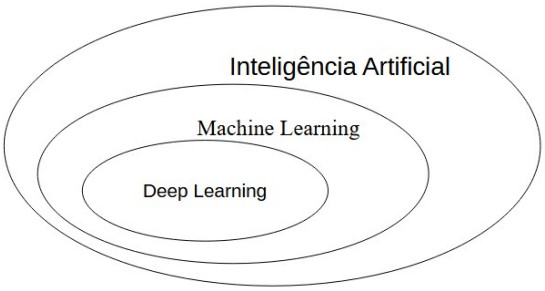


Figura 3 – Ilustração dos campos de estudo dos conceitos de IA. Fonte: Autor.

2.4.1 Machine learning

Para Andriy Burkov “*Machine learning* é um subcampo da ciência da computação que se concentra em desenvolver algoritmos, cuja usabilidade depende do conjunto de exemplos

de algum fenômeno" (Burkov, 2019, p. 7). O *machine learning* é um processo de reconhecimento de padrões com base em análises probabilísticas, utilizando também outras ferramentas matemáticas, como álgebra linear. Sua proposta é aprender e melhorar a cada passo analisando dados, sem ter de ser explicitamente programado. Tom Taulli resume, "Em essência, trata-se de um processo de adoção de dados rotulados (etiquetados) e busca de relacionamentos." (Taulli, 2019). Dentre os conceitos estatísticos utilizados, podemos destacar o teorema de Bayes, desvio padrão e distribuição normal.

O Teorema de Bayes é utilizado para equilibrar os dados conhecidos com os novos resultados obtidos, atualizando as estimativas de probabilidades com os novos resultados. O desvio padrão tem a função de detectar os casos que estão muito dispersos da média para ignorá-los ou tomar alguma tratativa. A distribuição normal descreve como os dados tendem a diminuir a frequência quanto mais se distanciam da média de ocorrência.

Conforme descreve (Goodfellow; Courville; Bengio, 2016, p. 98) "A maioria dos algoritmos de aprendizado de máquina podem ser divididos nas categorias de aprendizado supervisionado e aprendizado não supervisionado." O aprendizado supervisionado necessita que todo o conjunto de dados estejam rotulados, já com o aprendizado não supervisionado, esse trabalho de classificação é realizado pelos próprios algoritmos, utilizando o *deep learning* para detectar padrões.

Um dos principais problemas de algoritmos de *machine learning* tradicionais, consiste na rotulação dos dados, isto é, colocar etiquetas descritivas para classificação desses dados, dependendo de trabalho externo. Por exemplo, para identificação de uma imagem, um ser humano pode fornecer alguns atributos, mas não vai conseguir reconhecer todas as características relevantes da imagem, o que torna o processo ineficiente, principalmente quando lidar com um conjunto grande de dados, pois torna a tarefa ainda mais trabalhosa. Em alguns casos, foram abordadas algumas alternativas para melhorar este trabalho, como exemplifica (Taulli, 2019), que através de financiamento coletivo (*crowd funding*) construiu-se o sistema ImageNET, um enorme conjunto de dados estruturados e rotulados por humanos, trazendo grandes avanços para treinamento de modelos de IA.

Embora o problema descrito, os algoritmos clássicos de *machine learning* ainda são muito eficientes. A escolha de sua adoção depende de uma boa análise dos requisitos e entendimento do problema, por exemplo, para detecção de spam em e-mails, algoritmos como SVM (*Support vector machines* – Máquinas de vetores de suporte) ou RF (*Random forest* – Floresta aleatória) podem ser a melhor escolha, pela alta eficiência apresentada.

Dentre os algoritmos mais famosos de aprendizagem supervisionada, estão Naive Bayes, o SVM, e o KNN(*K nearest neighbor* – K vizinhos próximos).

O método *Naive* Bayes utiliza o teorema de Bayes para calcular a probabilidade de um objeto pertencer a uma determinada classe, com base em suas variáveis. A palavra *naive* do inglês significa ingênuo, chamado assim pois o algoritmo assume que as variáveis são independentes. Este algoritmo é comumente utilizado para análise de texto e de sentimento de mercado.

O algoritmo SVM cria um espaço multidimensional dividido por uma linha (2D) ou plano (3D) chamados de hiperplano, separando os dados em duas classes. O algoritmo utiliza os vetores de suporte, baseados nos pontos que estão mais próximos do hiperplano, para criar a margem. A margem é a distância entre o hiperplano e os pontos mais próximos, sem que haja pontos nela. O objetivo é encontrar o hiperplano, em que a margem seja a maior possível. Este algoritmo é bastante utilizado para detecção de spam e para reconhecimento de escrita à mão.

O KNN é um algoritmo simples que utiliza a premissa de que os dados semelhantes estejam próximos. Ao inserir um novo ponto de dados, o algoritmo calcula a distância entre ele e os demais pontos e define a quantidade de vizinhos, com base na variável K pré-determinadas. Dois exemplos de uso são a classificação em categorias, com base nas características, e a detecção de anomalias, com base em um vizinho muito distante dos demais.

2.4.2 Deep learning

Deep learning é um subcampo do *machine learning*, que utiliza múltiplas camadas internas para processar os dados, encontrando especificações impossíveis para um ser humano perceber. O funcionamento do *deep learning* é baseado no cérebro humano, por exemplo, uma rede neural artificial é uma função que inclui unidades (perceptrons ou neurônios), e cada uma possui um valor e um peso, que segue para a camada oculta, onde é realizada o cálculo da função, com uma constante chamado viés, obtendo a saída como resultado. A trajetória do algoritmo é basicamente da camada de entrada para a camada oculta e depois para a camada de saída. "*Deep learning* refere-se ao treinamento de uma rede neural com mais de duas camadas não relacionadas à saída" (Burkov, 2019, p. 77).

Apesar de muitos desses algoritmos terem sido formulados na década de 80, começaram a ganhar ênfase apenas nessa última década. Uma das principais motivações foi a

evolução das tecnologias de hardwares, como os processadores menores, possuindo mais núcleos e maior capacidade de processamento, a maior velocidade e capacidade de armazenamento das memórias RAM (*Random access memory* – Memória de acesso aleatório), as inovações na memória de "disco", como o caso dos SSDs (*Solid state drive* – Unidade de estado sólido) que utiliza circuitos integrados para armazenar os dados. O principal destaque no que tange aos hardwares, são as GPUs (Graphics processing unit – Unidade de processamento gráfico) dedicadas, que apesar de sua criação objetivar atender o mercado de jogos a priori, tem sido muito utilizada para treinamento e execução de redes neurais, pois permite maior velocidade de processamento em paralelo. Outro fator determinante à ascensão do *deep learning* foi a crescente disposição de dados na internet para análise, como por exemplo o já mencionado ImageNET.

Alguns dos algoritmos de redes neurais são RNN (*Recurrent neural network* – Redes neurais recorrentes) (Hochreiter; Schmidhuber, 1997), CNN e GAN (*Generative adversarial network* – Rede adversária generativa) (Goodfellow et al., 2014).

As RNNs são utilizadas para rotular, classificar ou gerar sequências. Seu modo de trabalho consiste em gerar uma matriz (sequência), onde cada linha representa um vetor de características cuja ordem é importante. Quando uma sequência foi rotulada, significa que foi prevista uma classe para cada vetor de características, e quando uma sequência foi classificada, indica que foi prevista uma classe para uma sequência inteira, e então uma nova sequência é gerada. Cada unidade de uma camada recorrente possui um valor de estado, e recebe duas entradas, um vetor de saídas e um vetor de estado da camada anterior. As RNNs são comumente utilizadas no processamento de falas, para processar texto e sugerir palavras.

As CNNs foram criadas pensando no trabalho com análise de imagens. Em uma CNN, a entrada é uma imagem, que passa pela etapa de convolução, onde são criados filtros de matrizes 3x3, seguido pela etapa agrupamento e ativação, sendo realizado repetidamente até a camada totalmente conectada onde a saída é gerada.

As GANs são redes neurais que competem entre si em um *loop* de *feedbacks*. Seu modo trabalho funciona com o gerador, discriminatório e modo de ajustes, no gerador são criadas as imagens, por exemplo, no discriminatório as criações são analisadas para verificar a integridade, e no modo de ajustes, as imagens são alteradas por meio de várias iterações a fim de ficar o mais realista possível. GANs são utilizadas para geração de imagens e vídeos.

2.4.3 Redes neurais convolucionais

As CNNs são responsáveis por avanços importantes na pesquisa de IA, sendo um dos primeiros modelos a obter sucesso ao tentar reproduzir uma rede neural baseado em um conceito biológico, no caso, a resposta do cérebro para estímulos visuais. Também foram uma das primeiras redes neurais a resolver um problema comercial. Nos anos 90, uma CNN foi desenvolvida para leitura de cheques, e no fim da década, esse sistema processava mais de 10% dos cheques nos EUA (Goodfellow; Courville; Bengio, 2016). Este modelo de *deep learning* ganhou muita visibilidade e adesão quando Alex Krizhevsky apresentou sua versão, o AlexNet no torneio do ImageNET em 2012 ILSVRC-2012 (ImageNet Large Scale Visual Recognition Challenge – Desafio de reconhecimento visual de larga escala do ImageNET) que ganhou demonstrando uma taxa de erro de 10,9% menor que o segundo colocado (Krizhevsky; Sutskever; Hinton, 2017).

O funcionamento das CNNs são baseadas no córtex visual primário dos gatos, tomada a partir das descobertas dos neurofisiologistas David Hubel e Torsten Wiesel. Sua grande descoberta foi que os neurônios no sistema visual inicial respondiam mais fortemente a padrões muito específicos de luz, como barras orientadas com precisão, começando o processo de filtragem de informações (Goodfellow; Courville; Bengio, 2016). Há 3 processos em que as CNNs foram inspiradas no sistema visual, os filtros, os mapas de características e o agrupamento. O córtex visual primário é a região do cérebro onde recebe informações da retina. Na retina contém células mais simples que detectam apenas barras de luz orientadas, é semelhante a forma de detecção de padrões mais escuros dos filtros. Uma luz que é capturada pela parte inferior da retina afeta uma parte correspondente do córtex, que é a forma como um filtro trabalha com um mapa de características em uma CNN. No córtex, há células mais complexas que respondem às células mais simples que recebem informações da retina desconsiderando variações nas posições, assim como acontece na etapa de agrupamento das CNNs, que desconsideram posições espaciais exatas das características detectadas.

A primeira camada de uma CNN é chamada de camada de entrada, onde ocorre a leitura da imagem e criação de uma matriz para representar os valores dos pixels, representada pela Figura 4. Se a imagem de entrada for 16x16 pixel, uma matriz 16x16 é criada, se a imagem for em preto e branco, a matriz será 2D (16 de largura x 16 de comprimento) e é atribuído um valor de 0 (preto) a 255 (branco) para cada pixel em seu respectivo campo da matriz, quando a imagem é colorida, a matriz será 3D (16 de largura x 16 de comprimento x 3 de profundidade) sendo em cada plano de profundidade definido um valor de 0 a 255 RGB,

ou seja, um plano para cor vermelha, um para cor verde e um para cor azul. O campo da matriz que representa o pixel da imagem, é chamado de neurônio, então o processo ficaria da seguinte forma, em uma imagem colorida, o pixel na linha 1 e coluna 1 é analisado, e então na matriz, linha 1, coluna 1, na primeira profundidade, é definido um valor para o neurônio de 0 (preto) a 255 (vermelho), para o neurônio da profundidade 2, o valor é entre 0 (preto) a 255 (verde), e no neurônio da profundidade 3, o valor varia entre 0 (preto) a 255 (azul), assim feito, segue para a linha 1, coluna 2, e o processo é repetido até que toda a imagem seja transcrita na matriz (Nielsen, 2015).

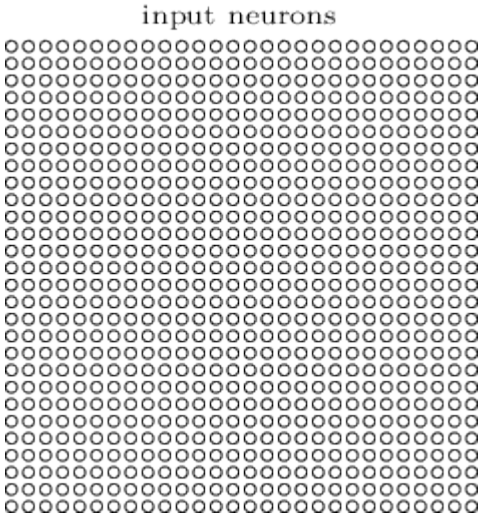


Figura 4 – Ilustra os “neurônios” de uma CNN após a leitura de uma imagem. Fonte: (Nielsen, 2015).

Na camada oculta, é onde acontece todo o trabalho da rede neural, possuindo várias camadas convolucionais e de *pooling* (agrupamento). Ao entrar em uma camada convolucional, é criado um filtro e um mapa de características para a matriz de entrada. Um filtro (também chamado de *kernel* ou mapa de recursos), Figura 5, é na verdade, uma pequena matriz de tamanho 3x3 ou 5x5, no caso de uma matriz 3D, o filtro será 3D, 5x5x3. Um filtro 5x5 possui 25 neurônios onde em cada um possui um peso e valor de viés, e esse filtro vai corresponder a um neurônio do mapa de característica, Figura 6. O objetivo primário dos filtros é identificar linhas, bordas, texturas e formas, os pesos vão se adaptando para responder a quantidade de luz do pixel, um ponto mais escuro, terá um peso maior, em um mais claro, o peso será menor. Por exemplo, esse filtro percorre a matriz de entrada (por isso é chamado convolução) multiplicando o valor dos pixels pelos pesos, e realizando um somatório ponderado, em seguida o valor do viés é aplicado para ajudar na função de ativação, e por último, a função de ativação é aplicada, definindo se um neurônio será ativado ou não,

guardando o valor obtido no mapa de características bidimensional. O mapa de características também será bidimensional para uma matriz 3D, pois a multiplicação dos pesos com os pixeis é realizada para os valores das 3 camadas RGB simultaneamente, e acumulado no somatório.

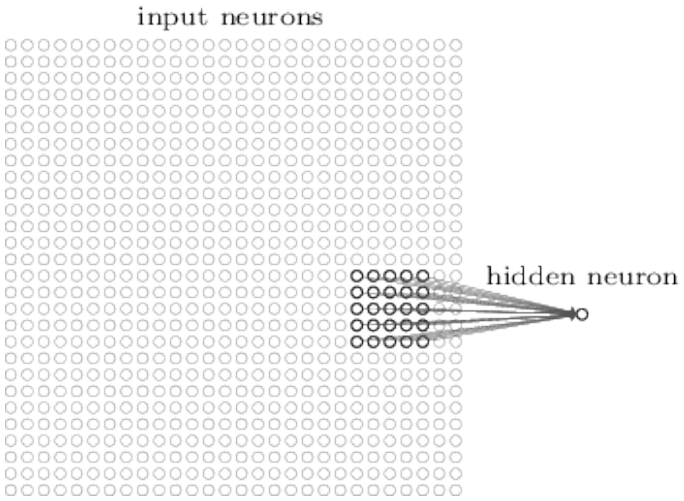


Figura 5 – Representação de um filtro percorrendo a matriz resultante da leitura de uma imagem. Fonte: (Nielsen, 2015).

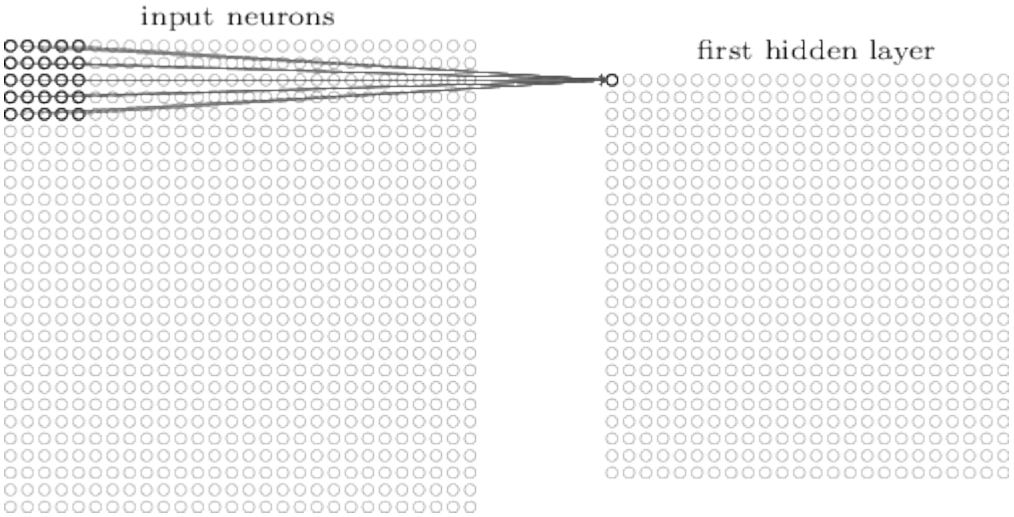


Figura 6 – Ilustração da geração de um mapa de características a partir da leitura de uma matriz por um filtro. Fonte: (Nielsen, 2015).

O tamanho do mapa de características é determinado pelo deslocamento do filtro dentro da matriz de entrada, por exemplo, se o deslocamento for definido como 1, para um filtro 5x5 em uma matriz de entrada 16x16, a quantidade de colunas que o filtro irá andar para atravessar a matriz será 12, e a quantidade para as linhas será o mesmo, resultando em um mapa de características de tamanho 12x12, caso o deslocamento fosse de 2 em 2 casas, o

tamanho resultante do mapa seria 6x6. O viés e os pesos são os mesmos para um determinado filtro que é compartilhado enquanto percorre a imagem. Cada filtro possui o seu próprio viés e peso e vai resultar em um mapa de características diferente, portanto, se existir 50 filtros para analisar uma matriz de imagem, cada um com seu próprio viés e peso, irá resultar em 50 mapas de características.

O processo de uma camada convolucional se torna muito eficiente, pois consegue obter o mesmo desempenho que uma primeira camada totalmente conectada, reduzindo drasticamente a quantidade de parâmetros (Nielsen, 2015). Em uma camada totalmente conectada, cada neurônio se conecta com todos os neurônios da matriz de entrada, suponha-se que a matriz de entrada seja 16x16, temos então 256 neurônios, então suponha-se modestamente uma matriz com 16 neurônios e 16 pesos conectados a cada neurônio da entrada, teríamos 16x256 = 4096 + vieses 4112 parâmetros, e para entradas muito grandes, como uma imagem em alta resolução, esse valor seria muito maior. Já numa camada convolucional, temos apenas os parâmetros do filtro vezes os neurônios de entrada 5x5 = 25 + 1 viés = 26 parâmetros, e se quisermos aplicar 20 mapas de características, teríamos 520 parâmetros.

Na camada de *pooling* é realizado o agrupamento, para cada mapa de características recebidos como entrada no *pooling*, é criado um mapa de características condensado, onde uma região de neurônios 2x2 do mapa de entrada é resumido para 1 neurônio do mapa condensado, Figura 7. Essa camada visa reduzir a dimensionalidade do mapa de características, usando técnicas como o *max-pooling* (agrupamento máximo), que usa a ativação máxima, guardando no novo mapa condensado apenas o valor máximo da região onde a matriz 2x2 está percorrendo. Dessa forma, descarta a posição do maior valor encontrado na sub-região, sinalizando apenas que algo foi encontrado. Esse processo reduz ainda mais a complexidade do algoritmo, guardando apenas os valores mais importantes, resumindo as informações para a camada de saída, tornando-o ainda mais eficiente.

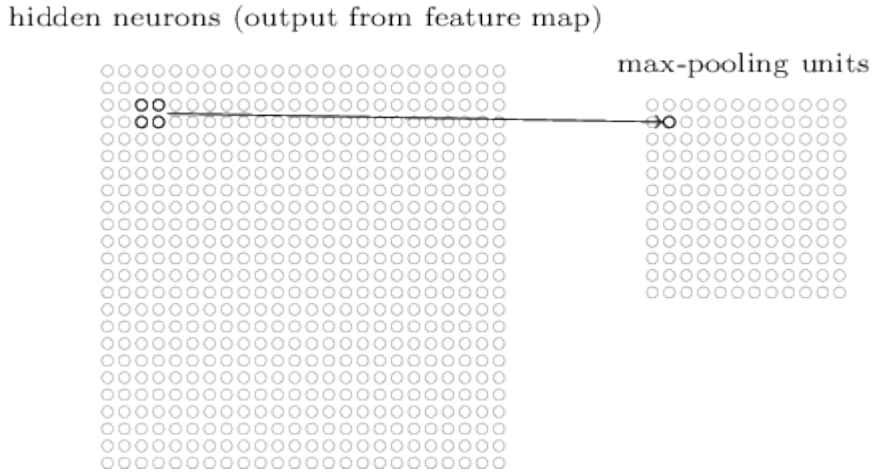


Figura 7 – Representação do agrupamento realizada por uma matriz quadrada. Fonte: (Nielsen, 2015).

Na camada de saída é formada uma rede convolucional totalmente conectada, onde cada neurônio de saída é conectado a todos os neurônios da camada de *pooling*, para realizar a dedução. A quantidade de neurônios finais depende da quantidade de classes definida para predição, por exemplo, para identificar em quais dos 10 estágios uma planta de soja está, é criado um vetor com 10 neurônios, onde uma função de ativação é aplicada para definir a probabilidade de a planta pertencer a cada classe, Figura 8. A função de ativação também dependerá de qual é o objetivo do algoritmo.

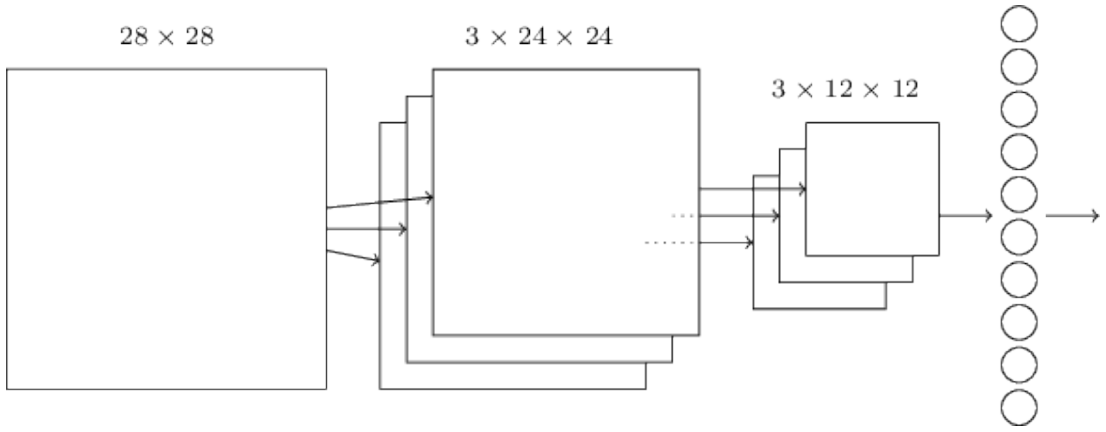


Figura 8 – Representação de uma CNN totalmente conectada. Fonte: (Nielsen, 2015).

A fim de ilustrar melhor o funcionamento de uma CNN, será utilizada a LeNet, uma CNN desenvolvida para reconhecimento de caracteres manuscritos, especificamente para identificação de dígitos em cheques bancários (Lecun et al., 1998).

A LeNet é constituída de 7 camadas além da entrada, possuindo dois pares de camadas de convolução seguidas por camadas de *pooling,* além de uma terceira camada de convolução, seguida por uma camada totalmente conectada e, por fim, pela camada de saída. A três

camadas de convolução são responsáveis por extrair as características das imagens. As duas camadas de *pooling* realizam a condensação dos mapas de características, reduzindo a dimensionalidade e diminuindo a quantidade de parâmetros da rede. A camada totalmente conectada realiza a classificação preliminar, e na camada de saída é aplicada a função de ativação final, que retorna a probabilidade de cada classe.

A Figura 9 representa o digito 8, utilizado para reconhecimento pelo modelo.

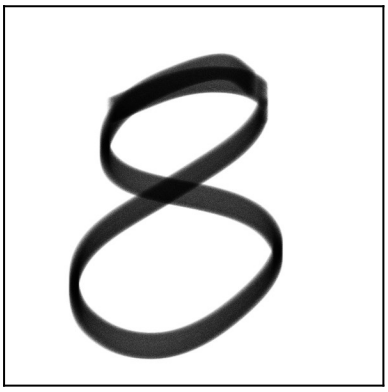


Figura 9 – Dígito “8” manuscrito, utilizado como entrada na LeNet. Fonte: Autor.

Em seguida, a Figura 10 mostra imagem de entrada após a leitura pelo modelo, em uma matriz de 28 pixels de largura por 28 pixels de altura.

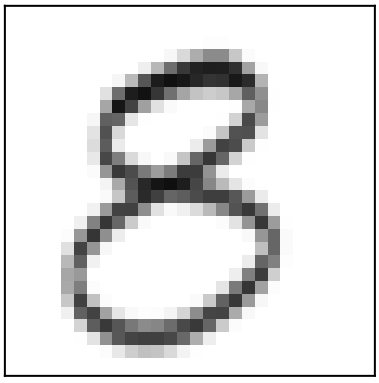


Figura 10 – Valor na matriz de entrada, após leitura pelo modelo. Fonte: Autor.

A Figura 11 a seguir representa os 6 mapas de características resultantes da primeira convolução, realizada por 6 filtros 5x5 diferentes, cada um com seu próprio peso e viés.

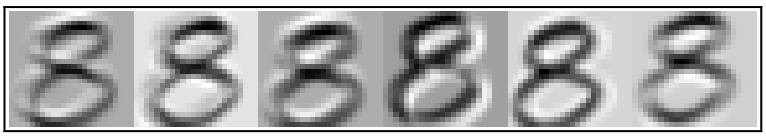


Figura 11 – Mapas de características após primeira convolução. Fonte: Autor.

Logo abaixo, a Figura 12 mostra os 6 mapas condensados após a aplicação da primeira camada de *pooling*. realizados por 6 filtros 2x2.

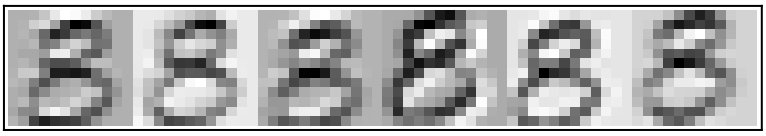


Figura 12 – Mapas condensados após primeiro *pooling*. Fonte: Autor.

Na figura 13, observam-se 16 mapas de características resultantes da segunda convolução, realizada por 16 filtros 5x5. A Figura 14 apresenta os 16 mapas condensados após aplicação da segunda etapa de *pooling*.

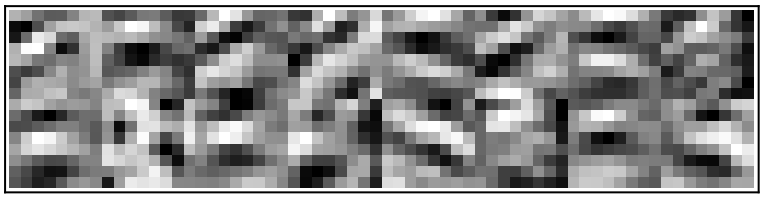


Figura 13 – Mapas de características após segunda convolução. Fonte: Autor.

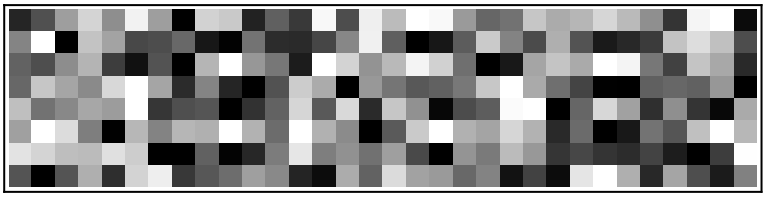


Figura 14 – Mapas condensados após segundo *pooling*. Fonte: Autor.

A Figura 15 mostra o vetor resultante da terceira camada de convolução, onde as ativações são reorganizadas para serem conectadas à primeira camada totalmente conectada. A Figura 16 apresenta o vetor resultante da primeira camada totalmente conectada, na qual cada neurônio recebe entradas de todos os neurônios da camada anterior (Figura 15).

Figura 15 – Vetor resultante após terceira convolução. Fonte: Autor.

Figura 16 – Vetor resultante após ativação, representando a primeira camada totalmente conectada. Fonte: Autor.

Na Figura 17, visualizam-se 10 linhas verticais, representando as classes de dígitos de 0 a 9. Essa figura corresponde à camada de saída, e quanto mais claro o feixe, maior a probabilidade atribuída à classe correspondente. Nota-se que a nona linha é a mais luminosa, indicando o dígito “8” como previsão do modelo.

Figura 17 – Vetor final representando as classes de saída. Fonte: Autor.

Por fim, a Figura 18 apresenta uma visão ilustrativa geral do funcionamento da LeNet.

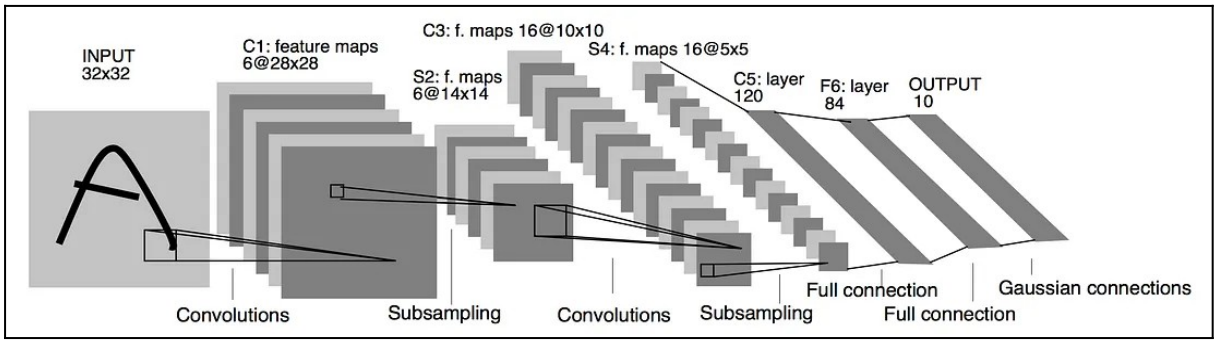


Figura 18 – Representação da LeNet-5, uma CNN utilizada para reconhecimento de dígitos. Fonte: (Lecun et al., 1998).

3. DESENVOLVIMENTO

Neste capítulo, serão apresentadas a bibliografia adicional que complementará a etapa de desenvolvimento do aplicativo proposto, os passos de elaboração, desenvolvimento e aplicação do projeto, além da demonstração de utilização do programa.

3.1 Levantamento bibliográfico

Com a finalidade de desenvolver o programa, foi necessário o estudo de novas metodologias e ferramentas.

3.1.1 PyTorch

O PyTorch é um *framework* de código aberto, desenvolvido em Python, que possibilita a utilização de modelos de aprendizado de máquina em pesquisas e no desenvolvimento de produtos. Sua facilidade de escalonamento para produção o tornou amplamente utilizado pela indústria. Este *framework* foi otimizado para treinar modelos tanto em GPU quanto em CPU (*Central processing unit* - Unidade central de processamento), e possui suporte para diversas marcas de *hardware* disponíveis no mercado ("PyTorch documentation — PyTorch 2.8 documentation", *[S.d.]*).

3.1.2 ONNX

O ONNX (*Open Neural Network Exchange* – Troca aberta de redes neurais) é um formato aberto para a representação de modelos de *machine learning* e de *deep learning*. Esse formato permite a interoperabilidade entre *frameworks* de diferentes linguagens. Em outras palavras, um modelo de aprendizado profundo treinado em um *framework* da linguagem Python, como o PyTorch, pode ser salvo em formato ".onnx", e posteriormente aberto e utilizado por um *framework* da linguagem JavaScript, por exemplo. Para possibilitar a execução de modelos nesse formato, foi desenvolvido o ONNX Runtime, um ambiente multiplataforma responsável pela leitura, otimização e execução de arquivos ONNX ("ONNX 1.20.0 documentation", *[S.d.]*).

3.1.3 Transfer learning

O *transfer learning* (aprendizado por transferência) é um campo de estudo dentro do *machine learning* que tem como foco a utilização do conhecimento adquirido em um treinamento para aprender uma nova habilidade. "O objetivo do *transfer learning* é solucionar

um novo problema a partir da identificação e utilização de similaridades entre dados, modelos ou tarefas de um problema antigo e um novo, possibilitando a transferência de conhecimento" (Wang; Chen, 2023, p. 5).

Wang e Chen descrevem os principais motivos para o uso dessa tecnologia. A primeira delas é a dificuldade em obter grandes quantidades de dados rotulados na prática. A cada segundo, uma enorme quantidade de dados é gerada em diversas modalidades, caracterizando a era do *big data* (grande volume de dados). No entanto, a maior parte desses dados é bruta, desorganizada e não está rotulada, ou seja, inadequada para treinamento. O processo de tratamento e rotulagem desses dados exige grande esforço, problema que pode ser evitado usando o *transfer learning*.

A principal dificuldade em treinar um modelo de *machine learning* do zero está no poder computacional necessário para processar grandes volumes de dados. À medida que o modelo de treinamento cresce, é necessário cada vez mais dados organizados e maior capacidade de processamento para continuar seu aprimoramento, recursos que apenas grandes corporações possuem. Para contornar essa limitação, pesquisadores e estudantes recorrem então à utilização de modelos pré-treinados.

Outro benefício que o *transfer learning* pode prover é a capacidade de generalização. Treinar um modelo de *machine learning* com uma pequena quantidade de dados o tornará "cego", isto é, incapaz de predizer corretamente sobre tipos específicos de dados que não foram utilizados no treinamento.

O propósito do *machine learning* é criar um modelo generalizado que atenda às demandas de diferentes usuários, dispositivos e ambientes, pois desenvolver um modelo grande e eficiente para cada categoria de dados seria inviável. Assim, cada usuário pode aproveitar o modelo geral por meio do *transfer learning* e refina-lo conforme sua área ou necessidade específica (Wang; Chen, 2023).

3.1.4 Redes residuais

Uma ResNet50 (He et al., 2015) se trata de uma ResNet (*Residual networks* – Redes residuais), é uma arquitetura de CNN composta por 50 camadas de profundidade. Sua principal inovação consiste na criação de "curto-circuitos", conexões de atalhos que tornam o aprendizado mais eficiente.

Esses atalhos evitam o desaparecimento do gradiente, comum em redes muito profundas, e tornam o treinamento mais estável. A arquitetura utiliza blocos do tipo *bottleneck*

(gargalo), que reduzem a dimensionalidade antes da convolução principal, diminuindo o custo computacional e mantendo o desempenho. Essa estrutura permite que a ResNet50 aprenda representações visuais complexas com eficiência.

3.1.5 Utilizando transfer learning em redes residuais pré-treinadas

A ResNet50 utilizada no *transfer learning* já vem pré-treinada no ImageNet, conjunto com mais de 15 milhões de imagens rotuladas e cerca de 22 mil categorias (Krizhevsky; Sutskever; Hinton, 2017). O treinamento da ResNet50 é feito sobre um subconjunto padronizado de mil classes, onde cada classe possui mil imagens, o que fornece à rede conhecimento visual amplo, como detecção de bordas, texturas, formas e padrões comuns em imagens naturais (He et al., 2015).

Para utilizar o transfer learning com a ResNet50, a arquitetura pode ser compreendida como dividida em duas partes: o extrator de características e o classificador. O extrator de características corresponde às camadas convolucionais e de agrupamento da ResNet50, já pré-treinadas no ImageNet, responsáveis por identificar padrões visuais gerais. Essas camadas devem ser congeladas para preservar os pesos aprendidos e evitar que o novo treinamento os altere.

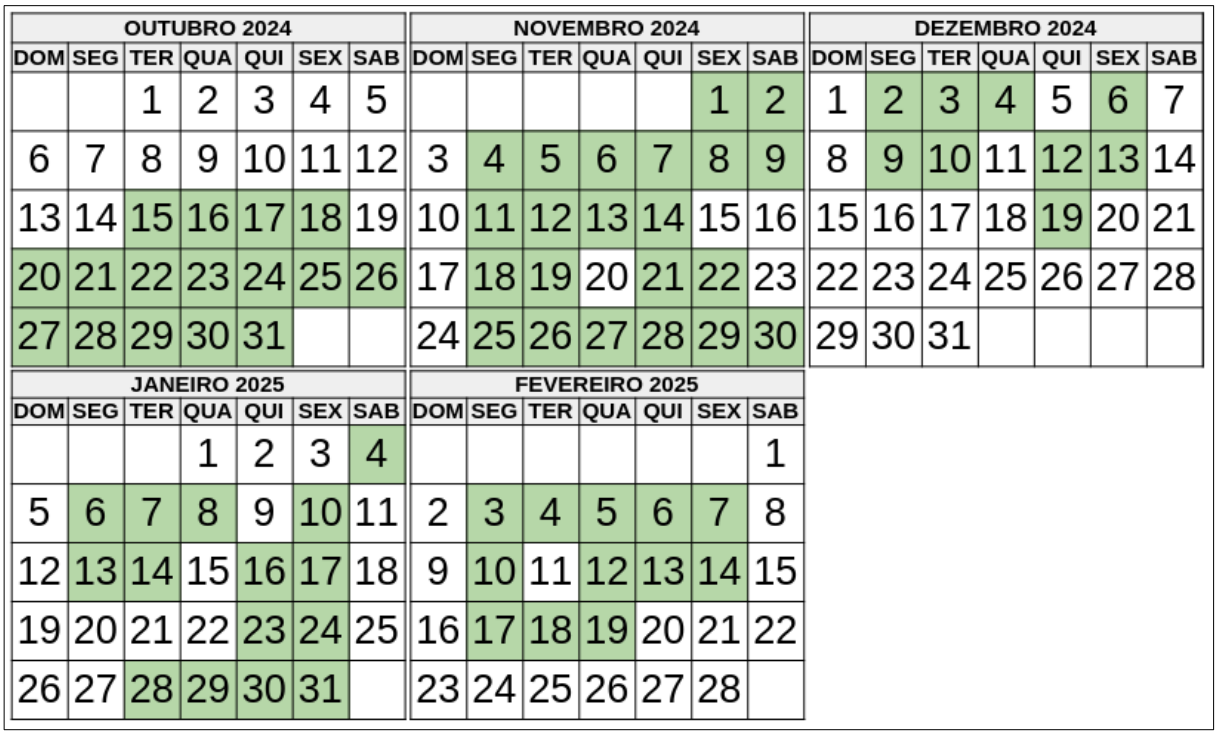
O classificador corresponde às camadas finais adicionadas ao modelo. As mil classes originais da ResNet50 são removidas e substituídas por uma camada de pooling global seguida de uma camada densa ajustada ao número de classes do novo conjunto de dados, com função de ativação apropriada para fornecer as probabilidades de cada categoria. É nessa parte final que ocorre o ajuste fino durante o treinamento, permitindo que o modelo adapte suas representações às características específicas das novas imagens.

3.2 Preparação de material para desenvolvimento

3.2.1 Coleta das imagens

A coleta das imagens foi realizada em uma plantação de soja, utilizando um aparelho celular Iphone X. As imagens foram capturadas no formato HEIC, com resolução de 4032 x 3024, por uma câmera de 12 megapixels. Em cada dia de captura, foram obtidas duas imagens: uma a aproximadamente 50 centímetros acima do solo, para registrar uma visão frontal próxima da planta, e outra a cerca de 1,70 metro acima do solo, com um ângulo aproximado de 45°, buscando representar uma captura natural humana de seu celular. Ao

todo, foram reunidas 146 imagens ao longo de 73 dias de coleta, cobrindo todo o ciclo da soja desde o plantio, no dia 15/10/2024, até a colheita, em 19/02/2025. A Figura 19 demonstra as datas em que foram realizadas as coletas de imagens.



OUTUBRO 2024						
DOM	SEG	TER	QUA	QUI	SEX	SAB
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

NOVEMBRO 2024						
DOM	SEG	TER	QUA	QUI	SEX	SAB
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

DEZEMBRO 2024						
DOM	SEG	TER	QUA	QUI	SEX	SAB
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

JANEIRO 2025						
DOM	SEG	TER	QUA	QUI	SEX	SAB
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

FEVEREIRO 2025						
DOM	SEG	TER	QUA	QUI	SEX	SAB
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	

Figura 19 – Demarcação dos dias de coleta das imagens, representados pela cor de fundo verde. Fonte: Autor.

3.2.2 Organização da imagens

Para utilizar as imagens em um modelo de predição, é necessário organizá-las em classes, que serão usadas para definir o vetor de saída do modelo. Os neurônios totalmente conectados à camada de *pooling* realizam a ativação final, determinando as probabilidades de cada imagem pertencer a cada classe.

As imagens foram classificadas pelo autor com base em um artigo da EMBRAPA que descreve os estágios de desenvolvimento da soja (Seixas et al., 2020). Para isso, foram criadas 11 pastas para organizar as imagens de acordo com os estágios vegetativos e reprodutivos.

Os estágios vegetativos da soja, identificados pela letra V, descrevem o desenvolvimento inicial da planta. No VE, ocorre a emergência, quando os cotilédones surgem acima do solo e formam ângulo de 90° com o hipocótilo. No VC, os cotilédones estão totalmente abertos e as folhas unifolioladas deixam de se tocar. A partir daí, cada novo estádio (V1, V2, V3, V4, V5, V6...Vn) representa o surgimento e abertura completa de uma nova folha trifoliolada no nó seguinte, indicando o avanço progressivo do crescimento vegetativo.

Já os estágios reprodutivos, identificados pela letra R e numerados de um a oito, compreendem o ciclo entre florescimento e maturação. Eles se dividem em quatro fases: florescimento (R1 e R2), desenvolvimento da vagem (R3 e R4), desenvolvimento do grão (R5 e R6) e maturação da planta (R7 e R8), que marcam o encerramento do ciclo produtivo da soja (Seixas et al., 2020).

A Tabela 1 apresenta os estágios vegetativos da soja organizados em colunas que contêm o estágio, a denominação, a descrição e a figura à qual cada estágio referencia.

Estágio	Denominação	Descrição	Figura
01-VE	Emergência	Cotilédones acima da superfície do solo	Figura 20
02-VC	Cotilédone	Cotilédones completamente abertos	Figura 21
03-V1	Primeiro nó	Folhas unifolioladas completamente desenvolvidas	
03-V2	Segundo nó	Primeira folha trifoliolada completamente desenvolvida	
03-Vn	Enésimo nó	Ante-enésima folha trifoliolada completamente desenvolvida	Figura 22

Tabela 1 – Descrição dos estágios vegetativos da soja. Fonte: (Seixas et al., 2020).

Figura 20 – Soja no estágio VE coletada em campo. Fonte: Autor.

Figura 21 – Soja no estágio VC coletada em campo. Fonte: Autor.

Figura 22 – Soja no estágio Vn coletada em campo. Fonte: Autor.

A Tabela 2 apresenta os estágios reprodutivos da soja organizada em colunas que contêm o estágio, a denominação, a descrição e a figura à qual cada estágio se refere.

Estágio	Denominação	Descrição	Figura
04-R1	Início do florescimento	Uma flor aberta em qualquer nó do caule (haste principal)	Figura 23
05-R2	Florescimento pleno	Uma flor aberta num dos 2 últimos nós do caule com folha completamente desenvolvida	Figura 24
06-R3	Início da formação da vagem	Vagem com 5 mm de comprimento num dos 4 últimos nós do caule com folha completamente desenvolvida	Figura 25
07-R4	Vagem completamente desenvolvida	Vagem com 2 cm de comprimento num dos 4 últimos nós do caule com folha completamente desenvolvida	Figura 26
08-R5	Início do enchimento do grão	Grão com 3 mm de comprimento em vagem num dos 4 últimos nós do caule, com folha completamente desenvolvida	Figura 27
09-R6	Grão cheio ou completo	Vagem contendo grãos verdes preenchendo as cavidades da vagem de um dos 4 últimos nós do caule, com folha completamente desenvolvida	Figura 28
10-R7	Início da maturação	Uma vagem normal no caule com coloração de madura	Figura 29
11-R8	Maturação plena	95% das vagens com coloração de madura	Figura 30

Tabela 2 – Descrição dos estágios reprodutivos da soja. Fonte: (Seixas et al., 2020).

Figura 23 – Soja no estágio R1 coletada em campo. Fonte: Autor.

Figura 24 – Soja no estágio R2 coletada em campo. Fonte: Autor.

Figura 25 – Soja no estágio R3 coletada em campo. Fonte: Autor.

Figura 26 – Soja no estágio R4 coletada em campo. Fonte: Autor.

Figura 27 – Soja no estágio R5 coletada em campo. Fonte: Autor.

Figura 28 – Soja no estágio R6 coletada em campo. Fonte: Autor.

Figura 29 – Soja no estágio R7 coletada em campo. Fonte: Autor.

Figura 30 – Soja no estágio R8 coletada em campo. Fonte: Autor.

A classificação das imagens da soja de acordo com seus estágios de crescimento, está apresentada na Tabela 3.

Estágio	01-VE	02-VC	03-Vn	04-R1	05-R2	06-R3	07-R4	08-R5	09-R6	10-R7	11-R8
Qtd. de imagens	16	8	34	18	16	2	2	6	12	20	10
Período	15/10/24 - 23/10/24	24/10/24 - 27/10/24	28/10/24 - 14/11/24	19/11/24 - 30/11/24	02/12/24 - 13/12/24	19/12/24	06/01/25	07/01/25 - 10/01/02	13/01/25 - 24/01/25	28/01/25 - 10/02/25	12/02/25 - 18/02/25

Tabela 3 – Período e quantidade de imagens em cada classe que representa o estágio de desenvolvimento da soja. Fonte: Autor.

Além da separação dos dados em pastas de acordo com suas classes, é importante separar os dados de treinamento dos dados de validação. Os dados de validação são usados para testar a acurácia do modelo final, e para isso, faz necessário que eles sejam distintos dos dados de treinamento, mas pertencentes à mesma classe, pois o contrário pode resultar na contaminação do modelo e em avaliações incorretas. A Tabela 4 demonstra a organização final dos dados, prontos para início do treinamento e dos testes.

Estágio	01-VE	02-VC	03-Vn	04-R1	05-R2	06-R3	07-R4	08-R5	09-R6	10-R7	11-R8
Qtd. de imagens para treinamento	10	4	24	12	10	1	1	4	8	14	6
Qtd. de imagens para validação	6	4	10	6	6	1	1	2	4	6	4

Tabela 4 – Demonstração da separação dos dados entre o conjunto de treinamento e validação. Fonte: Autor.

3.2.3 Treinamento do modelo

Com o objetivo de treinar o modelo, foi utilizado o *framework* PyTorch da linguagem de programação Python, devido sua compatibilidade com diferentes *hardwares*. Para o refinamento com o banco de dados de imagens de soja, empregou-se o modelo de rede neural convolucional ResNet50, pré-treinado com o enorme conjunto de dados do ImageNET, para utilização do aprendizado por transferência.

Na Figura 31 é demonstrado o processo de treinamento na linguagem Python, realizado pelo autor. Nas linhas 2 a 6 do código, são importadas as bibliotecas torch, que representa o núcleo matemático do PyTorch, e torchvision, voltada à visão computacional. Além dessas, são importadas, a partir dessas bibliotecas, ferramentas essenciais, tais como “nn”, que contém os métodos para trabalho com redes neurais; “transforms” utilizado para pré-processamento e aumentações de imagens, como redimensionamento, cortes, rotações e normalização; “ImageFolder”, que permite tornar cada sub-diretório em uma classe; e outros utilitários como o “DataLoader”.

A variável “num_classes” é declarada para armazenar a quantidade de classes, baseada nos subdiretórios existentes, enquanto a variável “dispostivo” define se será utilizado um *hardware* de GPU compatível ou CPU. Em seguida, na linha 12, é definido o objeto “tfms_treino” da classe “Compose”, no qual são aplicadas as transformações de aumento de dados, como “RandomResizedCrop”, que gera um corte aleatório da imagem com tamanho de 224 pixels; “RandomHorizontalFlip”, que realiza inversão vertical aleatória; e “RandomRotation”, que aplica um giro aleatório de até 15°. Nesse mesmo objeto, “ToTensor” converte a imagem em um tensor pytorch e “Normalize” realiza a normalização com base nos padrões das imagens do ImageNET utilizadas no treinamento da ResNet50.

```
1   # pip install torch torchvision
2   import torch, torchvision as tv
3   from torch import nn
4   from torchvision import transforms as T
5   from torchvision.datasets import ImageFolder
6   from torch.utils.data import DataLoader
7
8   num_classes = None
9   dispositivo = "cuda" if torch.cuda.is_available() else "cpu"
10
11  # transformações de treino
12  tfms_treino = T.Compose([
13      T.RandomResizedCrop(224, scale=(0.8,1.0)),
14      T.RandomHorizontalFlip(),
15      T.RandomRotation(15),
16      T.ToTensor(),
17      T.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]),
18  ])
```

Figura 31 – Parte um do código de treinamento do modelo. Fonte: Autor.

Na Figura 32 abaixo, o objeto “ds_treino” recebe uma instância da classe “ImageFolder” definindo cada subdiretório da pasta “plantio_14_10_2024_treino” como uma classe para o modelo de predição. O parâmetro “transforma=tfms_treino” indica que, para cada imagem encontrada no diretório, serão aplicadas as transformações de aumento definidas no objeto “tfms_treino”. A variável “num_classes” armazena a quantidade de classes obtida de “ds_treino”.

O objeto “dl_treino” utiliza “ds_treino” para criar lotes de 32 imagens (batch_size=32) e, com “shuffle=true”, garante que a ordem das imagens seja aleatória a cada época, evitando que o modelo memorize a sequência de entrada. O parâmetro “num_workers” dedica quatro subprocessos paralelos para o carregamento dos dados durante o treinamento.

O objeto “modelo” carrega uma instância da rede ResNet50, com todos os pesos pré-treinados pelo ImageNET (linha 28), substitui a última camada totalmente conectada pela quantidade de classes contida em “num_classes” (linha 29), e define o *hardware* no qual o novo modelo será treinado (linha 30).

A variável “otimização” configura à todos os pesos, a taxa de aprendizado e a regularização, enquanto “criterio” utiliza a função de perda “CrossEntropyloss” para medir a diferença entre as predições do modelo e os rótulos verdadeiros das imagens.

```
20  # dataset de treino
21  ds_treino = ImageFolder("plantio_14_10_2024_treino", transform=tfms_treino)
22  num_classes = len(ds_treino.classes)
23
24  # dataloader de treino
25  dl_treino = DataLoader(ds_treino, batch_size=32, shuffle=True, num_workers=4)
26
27  # modelo
28  modelo = tv.models.resnet50(weights=tv.models.ResNet50_Weights.IMAGENET1K_V2)
29  modelo.fc = nn.Linear(modelo.fc.in_features, num_classes)
30  modelo = modelo.to(dispositivo)
31
32  # otimizador e critério (função de perda)
33  otimizador = torch.optim.AdamW(modelo.parameters(), lr=1e-4, weight_decay=1e-4)
34  criterio = nn.CrossEntropyLoss()
```

Figura 32 – Parte dois do código de treinamento do modelo. Fonte: Autor.

Na Figura 33, é iniciado um laço no qual o processo de treinamento do modelo é repetido 10 vezes, sendo cada repetição denominada época. Em cada época, segue-se o seguinte processo: na linha 38 o modelo é colocado em modo de treinamento; na linha 39, são carregados os lotes de imagens (entradas) e os rótulos das classes (rotulos) a partir do "dl_treino". Em seguida, "entradas" e "rotulos" são direcionados ao *hardware* definido em "dispositivo". A variável "perda" (linha 42) armazena a taxa de perda do modelo, utilizada na retro propagação de erros (linha 43). Por fim, na linha 44 é aplicado o otimizador para atualizar os pesos da rede.

```
36  # loop de treino
37  for epoca in range(10):
38      modelo.train()
39      for entradas, rotulos in dl_treino:
40          entradas, rotulos = entradas.to(dispositivo), rotulos.to(dispositivo)
41          otimizador.zero_grad()
42          perda = criterio(modelo(entradas), rotulos)
43          perda.backward()
44          otimizador.step()
45
46  # EXPORTAR DEPOIS (com try/except)
47  try:
48      entrada_dummy = torch.randn(1,3,224,224, device=dispositivo)
49      torch.onnx.export(
50          modelo, entrada_dummy, "resnet50_soja.onnx",
51          input_names=["input"], output_names=["prob"],
52          dynamic_axes={"input":{0:"batch"}, "prob":{0:"batch"}},
53          opset_version=13
54      )
55  except Exception as e:
56      print(f"[AVISO] Falha na exportação ONNX: {e}")
```

Figura 33 – Parte três do código de treinamento do modelo. Fonte: Autor.

Na Figura 34 a seguir, é apresentado o código-fonte completo para o treinamento do modelo na linguagem Python.

```python
# pip install torch torchvision
import torch, torchvision as tv
from torch import nn
from torchvision import transforms as T
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader

num_classes = None
dispositivo = "cuda" if torch.cuda.is_available() else "cpu"

# transformações de treino
tfms_treino = T.Compose([
    T.RandomResizedCrop(224, scale=(0.8,1.0)),
    T.RandomHorizontalFlip(),
    T.RandomRotation(15),
    T.ToTensor(),
    T.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]),
])

# dataset de treino
ds_treino = ImageFolder("plantio_14_10_2024_treino", transform=tfms_treino)
num_classes = len(ds_treino.classes)

# dataloader de treino
dl_treino = DataLoader(ds_treino, batch_size=32, shuffle=True, num_workers=4)

# modelo
modelo = tv.models.resnet50(weights=tv.models.ResNet50_Weights.IMAGENET1K_V2)
modelo.fc = nn.Linear(modelo.fc.in_features, num_classes)
modelo = modelo.to(dispositivo)

# otimizador e critério (função de perda)
otimizador = torch.optim.AdamW(modelo.parameters(), lr=1e-4, weight_decay=1e-4)
criterio = nn.CrossEntropyLoss()

# loop de treino
for epoca in range(10):
    modelo.train()
    for entradas, rotulos in dl_treino:
        entradas, rotulos = entradas.to(dispositivo), rotulos.to(dispositivo)
        otimizador.zero_grad()
        perda = criterio(modelo(entradas), rotulos)
        perda.backward()
        otimizador.step()

# EXPORTAR DEPOIS (com try/except)
try:
    entrada_dummy = torch.randn(1,3,224,224, device=dispositivo)
    torch.onnx.export(
        modelo, entrada_dummy, "resnet50_soja.onnx",
        input_names=["input"], output_names=["prob"],
        dynamic_axes={"input":{0:"batch"}, "prob":{0:"batch"}},
        opset_version=13
    )
except Exception as e:
    print(f"[AVISO] Falha na exportação ONNX: {e}")
```

Figura 34 – Código completo utilizado para o treinamento de uma CNN por meio de *transfer learning*. Fonte: Autor.

3.3 Desenvolvimento do sistema

Com o modelo já refinado para identificar os estágios de desenvolvimento da soja e salvo em formato ONNX, o passo seguinte é desenvolver a interface do programa para o usuário final. Esse processo será realizado na linguagem de programação Java.

3.3.1 Requisitos do sistema

A fim de tornar o programa amigável e intuitivo para o usuário, a proposta de projeto é simples, considerando que o objetivo geral do trabalho, na parte de interação com o sistema, também não é complexo.

De forma direta, o programa deve ser capaz de abrir uma pequena interface gráfica com o *layout* padrão de janelas, apresentando a barra de título com o nome do sistema e contendo os botões de controle no canto superior direito para minimizar, maximizar/restaurar e fechar. Na área de conteúdo, deve haver um botão que permita ao usuário buscar no computador e selecionar, por meio do explorador de arquivos, a imagem a ser analisada. Após a seleção, a imagem deve ser exibida na tela, juntamente com o resultado da análise realizada pelo modelo, ou seja, o estágio de crescimento que a soja se encontra. O sistema também deve permitir que o usuário selecione novas imagens para análise.

3.3.2 Estrutura do sistema

A Figura 35 apresenta um diagrama de classes UML (*Unified modeling language* – linguagem de modelagem unificada), cujo objetivo é demonstrar a estrutura do software, ou seja, as classes, atributos e métodos utilizados na implementação do sistema. A Tabela 5 a seguir fornece uma breve descrição da finalidade de cada método pertencentes às classes contidas no programa.

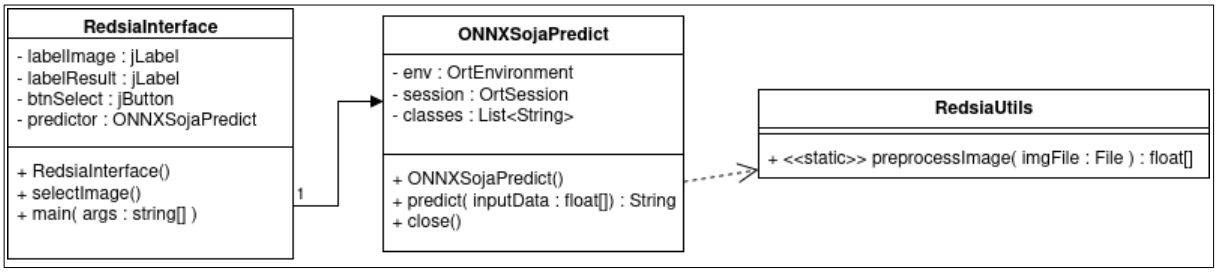


Figura 35 – Diagrama de classes UML que representa a estrutura de classes do sistema. Fonte: Autor.

Classes	Métodos	Objetivo
RedsiaInterface	redsiaInterface()	Cria a janela, define layout e tamanho, inicializa o modelo ONNX, cria os componentes gráficos e exibe a interface.
RedsiaInterface	selectImage()	Abre o seletor de arquivos, carrega a imagem escolhida, exibe na tela, pré-processa com RedsiaUtils, envia ao modelo e mostra o resultado.
RedsiaInterface	main()	Inicia a aplicação Swing e cria uma instância de RedsiaInterface.
ONNXSojaPredict	ONNXSojaPredict()	Cria o ambiente ONNX, carrega o modelo .onnx, inicia a sessão de inferência e lê os metadados com as classes.
ONNXSojaPredict	predict()	Cria tensor de entrada, roda o modelo, aplica softmax, identifica a classe com maior probabilidade e retorna o nome e a confiança.
ONNXSojaPredict	close()	Fecha a sessão e o ambiente ONNX.
RedsiaUtils	preProcessImage()	Lê a imagem, faz corte central, redimensiona para 224x224, normaliza os canais RGB e retorna os dados prontos para o modelo.

Tabela 5 – Descrição do comportamento dos métodos pertencentes às classes do sistema. Fonte: Autor.

3.3.3 Interface do sistema

A interface do sistema apresenta uma tela simples ao usuário, com o botão “Selecionar Imagem”, conforme mostrado na Figura 36, cujo *layout* é baseado em programas comuns de pequenos ajustes em imagens, como aqueles que permitem ao usuário abrir arquivos para remasterização ou remoção de fundos.

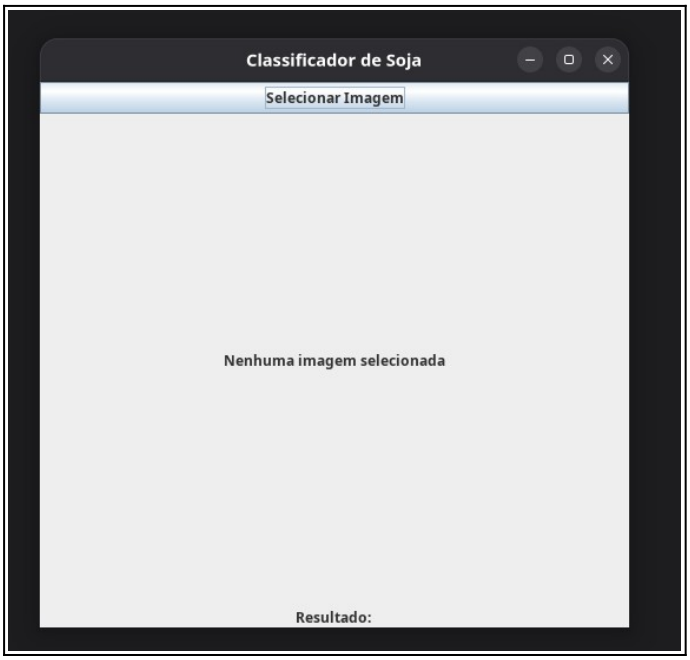


Figura 36 – Interface inicial do sistema. Fonte: Autor.

Ao pressionar o botão “Selecionar Imagem”, uma janela é aberta com o explorador do arquivos, permitindo que usuário localize, em seu computador, a imagem que será utilizada para que o sistema identifique o estágio de desenvolvimento da soja, conforme apresentado na Figura 37.

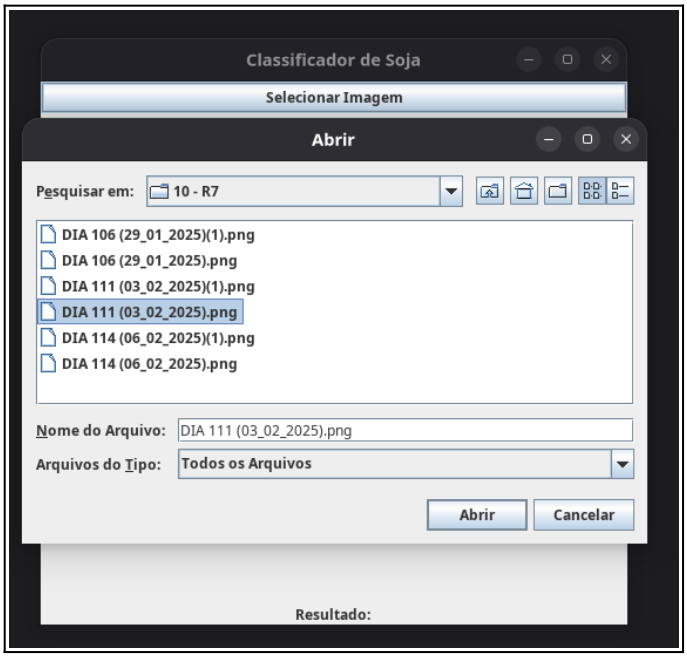


Figura 37 – Visualização do explorador de arquivos do sistema. Fonte: Autor.

Por fim, após seleção da imagem, o sistema realiza a classificação desta e atualiza a janela principal com a imagem escolhida. Logo abaixo, no rótulo “Resultado”, é exibida a

resposta do modelo, indicando a porcentagem de probabilidade de a imagem pertencer àquela classe, conforme mostrado na Figura 38.

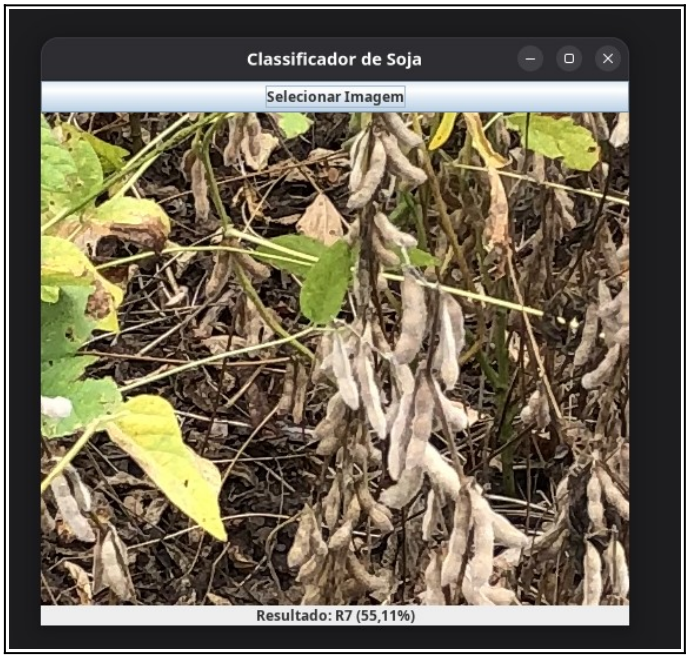


Figura 38 – Demonstração do sistema após seleção da imagem e dedução do modelo. Fonte: Autor.

4. RESULTADOS

Para apresentação da acurácia final atingida pelo modelo treinado e salvo em formato ONNX, foi criado um *script* (sequência de instruções) em python para ler as imagens do diretório de validação e apresentar o resultado da avaliação e dedução do estágio da soja. A seguir será demonstrado uma breve explicação deste *script*.

Nas linhas de código na Figura 39 abaixo são importadas as bibliotecas necessárias para execução do *script*.

```
1    import os, argparse, onnxruntime as ort
2    from torchvision import transforms as T
3    from torchvision.datasets import ImageFolder
4    from torch.utils.data import DataLoader
5    from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
6    import numpy as np
7    import matplotlib.pyplot as plt
8    from PIL import ImageFile
9    import torch
10
11   ImageFile.LOAD_TRUNCATED_IMAGES = True  # evita erro com imagens corrompidas
12
```

Figura 39 – Parte 1 do código usado para validação do modelo treinado. Fonte: Autor.

Em seguida, Figura 40, é realizado o carregamento dos rótulos das classes com base nos nomes das pastas existentes no diretório, por meio da função “carregar_rotulos”. A função “carregar_modelo_onnx” carrega o modelo treinado, salvo em formato ONNX, e inicia uma sessão do ONNX Runtime. Já a função “plotar_matriz_confusao” é utilizada para gerar um gráfico *heatmap* (mapa de calor), denominado matriz de confusão, a partir dos dados obtidos na análise de desempenho do modelo, a fim de melhorar a visualização dos resultados.

```
# Usa labels.txt se existir, senão pega do ImageFolder
def carregar_rotulos(diretorio_validacao):
    if os.path.exists("labels.txt"):
        with open("labels.txt") as f:
            return [l.strip() for l in f if l.strip()]
    return ImageFolder(diretorio_validacao).classes

# Cria sessão ONNX Runtime
def carregar_modelo_onnx(caminho_modelo="resnet50_soja.onnx"):
    return ort.InferenceSession(caminho_modelo, providers=["CPUExecutionProvider"])

# Cria um gráfico da matriz de confusão e salva em arquivo em arquivo de imagem
def plotar_matriz_confusao(matriz, rotulos, arquivo_saida="matriz_confusao.png", normalizar=True):
    if normalizar:
        matriz = matriz.astype("float") / matriz.sum(axis=1, keepdims=True)
        matriz = np.nan_to_num(matriz)

    plt.figure(figsize=(10, 8))
    plt.imshow(matriz, interpolation="nearest")
    plt.title("Matriz de Confusão" + (" (normalizada)" if normalizar else ""))
    plt.colorbar()
    pos = np.arange(len(rotulos))
    plt.xticks(pos, rotulos, rotation=45, ha="right")
    plt.yticks(pos, rotulos)
    plt.xlabel("Predito")
    plt.ylabel("Verdadeiro")
    plt.tight_layout()
    plt.savefig(arquivo_saida, dpi=150)
    plt.close()

```

Figura 40 - Parte 2 do código usado para validação do modelo treinado. Fonte: Autor.

A seguir na Figura 41, dentro da função “principal”, é configurado o objeto “parser”, que define parâmetros padrão, como o diretório dos dados, o modelo utilizado, os arquivos de saída da análise, permitindo o ajuste desses valores via terminal. Nas linhas seguintes, são carregados os rótulos e determinada a quantidade de classes. Em sequência, são definidas as transformações das imagens, apenas para adequação ao formato dos dados do modelo, sem aplicação de aumentações nesta etapa de validação. Por fim, o conjunto de validação é carregado com base nessas configurações, e o objeto “dl_validacao” é criado para realizar a leitura dos dados em lote.

```
43  def principal():
44      parser = argparse.ArgumentParser()
45      parser.add_argument("--dir_validacao", default="plantio_14_10_2024_validacao")
46      parser.add_argument("--tamanho_lote", type=int, default=64)
47      parser.add_argument("--num_workers", type=int, default=0)
48      parser.add_argument("--modelo_onnx", default="resnet50_soja.onnx")
49      parser.add_argument("--saida_relatorio", default="relatorio_classificacao.txt")
50      parser.add_argument("--saida_matriz", default="matriz_confusao.png")
51      args = parser.parse_args()
52
53      # 1) Rotulos
54      nomes_classes = carregar_rotulos(args.dir_validacao)
55      qtd_classes = len(nomes_classes)
56
57      # 2) Dataset sem aumentação
58      transformacoes = T.Compose([
59          T.Resize(256), T.CenterCrop(224),
60          T.ToTensor(),
61          T.Normalize([0.485,0.456,0.406],[0.229,0.224,0.225]),
62      ])
63      ds_validacao = ImageFolder(args.dir_validacao, transform=transformacoes)
64      dl_validacao = DataLoader(ds_validacao, batch_size=args.tamanho_lote, shuffle=False,
65                                num_workers=args.num_workers)
66
67      # 3) Modelo ONNX
68      sessao = carregar_modelo_onnx(args.modelo_onnx)
69      entrada_nome = sessao.get_inputs()[0].name
70
71      # 4) Inferência
72      y_verdadeiro, y_predito = [], []
73      for imagens, rotulos in dl_validacao:
74          entradas = imagens.numpy()
75          saidas = sessao.run(None, {entrada_nome: entradas})[0]
76          preds = np.argmax(saidas, axis=1)
77          y_verdadeiro.append(rotulos.numpy())
78          y_predito.append(preds)
79
80      y_verdadeiro = np.concatenate(y_verdadeiro)
81      y_predito = np.concatenate(y_predito)
82
```

Figura 41 - Parte 3 do código usado para validação do modelo treinado. Fonte: Autor.

Então por conseguinte, o modelo ONNX é carregado e inicializado, e são criadas listas para guardar os rótulos reais das classes e os rótulos identificados pelo modelo. Em seguida, o laço de repetição percorre as pastas do conjunto de validação, lendo as imagens, realizando as inferências e adicionando os resultados às listas correspondentes. Posteriormente, é calculada a acurácia total e exibida no terminal, assim como o relatório de precisão do sistema. Por fim é gerada a matriz de confusão, salva em arquivo de imagem, e o relatório completo é gravado em arquivo de texto, conforme Figura 42.

```
83        # 5) Métricas
84        acuracia = accuracy_score(y_verdadeiro, y_predito)
85        print(f"Acurácia (validação): {acuracia:.4f}")
86
87        relatorio = classification_report(
88            y_verdadeiro, y_predito, target_names=nomes_classes, digits=4
89        )
90        print("\nRelatório por classe:\n")
91        print(relatorio)
92
93        # 6) Matriz de confusão
94        matriz = confusion_matrix(y_verdadeiro, y_predito, labels=list(range(qtd_classes)))
95        plotar_matriz_confusao(matriz, nomes_classes, args.saida_matriz, normalizar=True)
96        print(f"Matriz de confusão salva em: {args.saida_matriz}")
97
98        # 7) Salvar relatório
99        with open(args.saida_relatorio, "w") as f:
100           f.write(f"Acurácia: {acuracia:.4f}\n\n")
101           f.write(relatorio)
102       print(f"Relatório salvo em: {args.saida_relatorio}")
103
104   if __name__ == "__main__":
105       principal()
```

Figura 42 - Parte 4 do código usado para validação do modelo treinado. Fonte: Autor.

4.1 Análise dos resultados

No geral, o programa apresentou bons resultados na maioria dos cenários. Conforme relatório exportado do *script* de testes, o modelo obteve uma acurácia total de 70% nos estágios que continham dados suficientes para a identificação.

A Tabela 6 está organizada em colunas que representam as 11 classes dos estágios de crescimento da soja e em linhas que indicam a precisão, a sensibilidade, a pontuação total e a quantidade de imagens avaliadas. A coluna Precisão mede, a partir das imagens classificadas como pertencentes a determinado estágio, o percentual das que realmente correspondem a esse estágio. A coluna Sensibilidade indica o percentual de imagens que pertencem a um estágio e que o modelo conseguiu identificar corretamente. A Pontuação Total é a média aritmética entre a precisão e a sensibilidade, visando medir o equilíbrio entre ambas. Por fim,

a última coluna, “Imagens Avaliadas/Total de imagens” mostra a razão entre o número de amostras usadas para validação de cada classe e o total de imagens pertencentes a estas classes.

Estágios	Precisão em %	Sensibilidade em %	Pontuação Total em %	Imagens Avaliadas/Total de imagens
01 – VE	1.0000	0.6667	0.8000	6/16
02 – VC	0.0000	0.0000	0.0000	4/8
03 – Vn	0.5556	1.0000	0.7143	10/34
04 – R1	1.0000	0.6667	0.8000	6/18
05 – R2	0.6000	1.0000	0.7500	6/16
06 – R3	0.0000	0.0000	0.0000	1/2
07 – R4	0.0000	0.0000	0.0000	1/2
08 – R5	0.0000	0.0000	0.0000	2/6
09 – R6	0.7500	0.7500	0.7500	4/12
10 – R7	0.7500	1.0000	0.8571	6/20
11 – R8	1.0000	0.5000	0.6667	4/10

Tabela 6 – Tabela criada com base no relatório gerado pelo *script* de validação a partir dos dados de validação. Fonte: Autor.

A Figura 43 apresenta uma matriz de confusão, gráfico gerado a partir dos dados referente às classes verdadeiras das imagens e das classes acusadas pelo modelo. As linhas representam as classes verdadeiras, isto é, as pastas nas quais as imagens estão armazenadas para validação. As colunas, por sua vez, representam o julgamento do modelo para as imagens analisadas. Ao lado da matriz, há uma legenda de cores em formato de coluna, cujo objetivo é identificar o percentual de imagens de cada classe que o modelo inferiu pertencer a determinado estágio. As cores variam do roxo, representando o 0%, até o amarelo, que indica 100%. Uma matriz de confusão ideal para esse modelo seria uma diagonal perfeita totalmente amarela, enquanto os demais quadrantes deveriam ser roxos, pois isso indicaria que o modelo classificou corretamente todas as imagens de cada classe – por exemplo, as imagens da classe 1 na linha 1 estariam também na coluna 1, e o mesmo se aplicaria às demais classes.

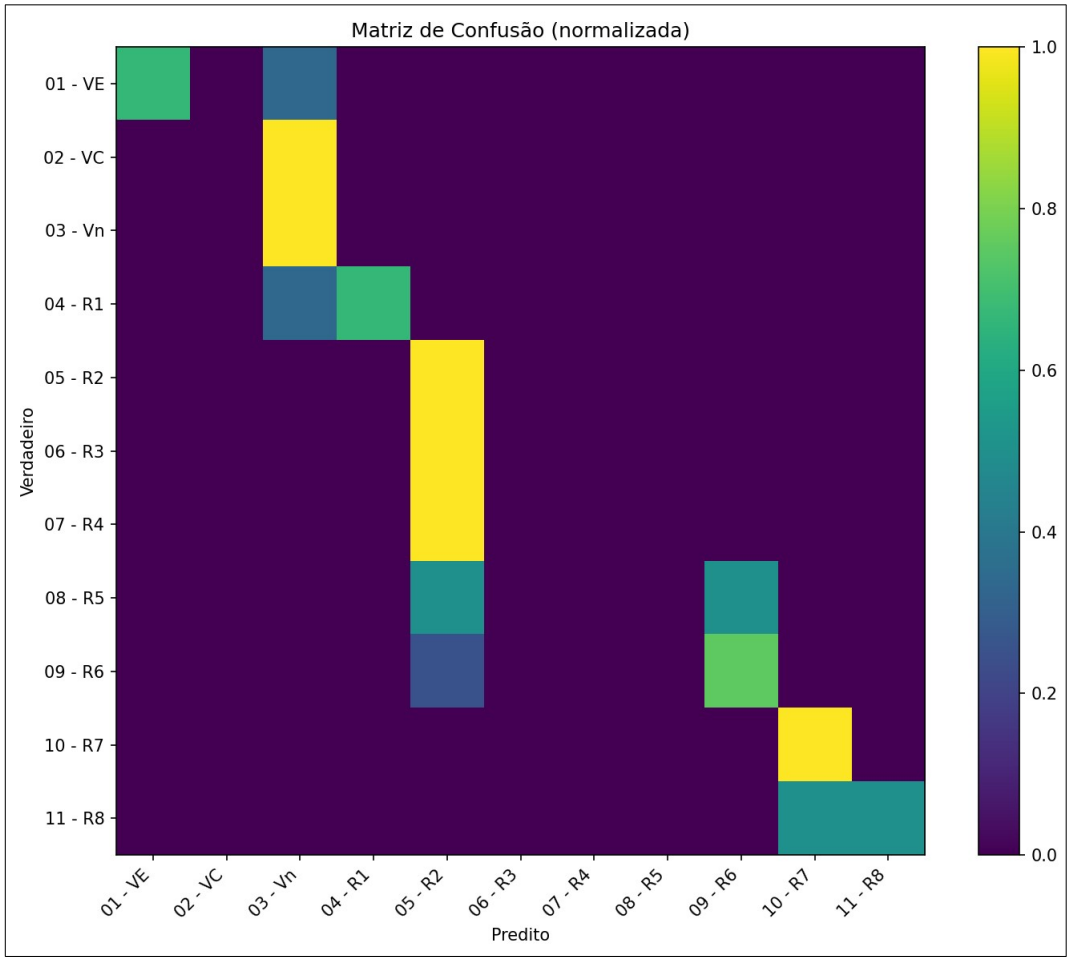


Figura 43 – Matriz de confusão gerada pelo *script* de validação a partir dos dados de validação. Fonte: Autor.

5. CONCLUSÃO

Diante da bibliografia levantada, constatou-se que o Brasil é a maior potência mundial na produção e exportação da cultura da soja. O estado do Mato Grosso do Sul está entre os cinco principais produtores do país, com destaque para municípios como Dourados e Maracaju. Apesar disso, a utilização de tecnologias voltadas à agricultura de precisão ainda apresenta uma adesão lenta, quando comparada a países desenvolvidos e outras potências agrícolas.

Nos últimos anos, o campo de estudo em inteligência artificial têm crescido drasticamente, impulsionado pelos avanços tecnológicos em hardware e pela enorme disponibilidade de dados. Uma das principais técnicas dessa área é a rede neural convolucional, considerada o método mais eficiente para aplicação de visão computacional e inteligência artificial no reconhecimento de padrões em imagens.

O programa desenvolvido para identificar o estágio da soja, apresentou resultados animadores. Com uma acurácia de 70% nas classes cujo modelo possuía dados suficientes para reconhecer padrões, demonstrou ser possível utilizar um computador com baixo poder de processamento e uma quantidade reduzida de dados para o treinamento do modelo. O sistema possui uma interface simples e intuitiva, evitando a necessidade de conhecimentos técnicos avançados para sua utilização. Além disso, por ser gratuito, pode auxiliar pequenos agricultores que não dispõem de recursos financeiros significativos, contribuindo para a aplicação correta de insumos agrícolas conforme o estágio de desenvolvimento da cultura.

5.1 Trabalhos futuros e melhorias

Apesar de o programa apresentar resultados promissores para a aplicação de inteligência artificial na análise de imagens da soja, ainda há vários pontos de melhorias a serem aplicados.

A principal melhoria necessária para uma versão final do sistema, com funcionamento pleno, é o aumento da precisão na análise realizada pelo modelo. Com base nos resultados obtidos, verificou-se uma taxa de acurácia de 70%, porém há estágios de desenvolvimento da planta que não apresentam acertos. Por outro lado, algumas classes apresentaram resultados próximos ou iguais a 100% de precisão, indicando que, com um volume de dados adequado, o modelo pode aprender de forma satisfatória. Assim, para obter um modelo mais acurado, é necessário aumentar o banco de dados de imagens de treinamento, aumentando a quantidade

de amostras em cada estágio de crescimento da planta para que a rede neural aprenda corretamente.

Outro ponto que pode ser aprimorado é a interface de usuário. O programa atingiu seu objetivo ao oferecer um *layout* simples, intuitivo e minimalista, facilitando a utilização do sistema. Contudo, a janela principal pode ser expandida para apresentar detalhes técnicos sobre cada estágio da soja, assim como os principais cuidados e insumos a serem aplicados em cada período de desenvolvimento da planta.

Em trabalhos futuros, a área de estudo pode ser ampliada para incluir outras aplicações de análise de imagem, como a identificação do estágio de desenvolvimento de diferentes culturas agrícolas. Além disso, é possível treinar novos modelos voltados ao reconhecimento de doenças e deficiências nutricionais em cultivares.

REFERÊNCIAS BIBLIOGRÁFICAS

A.H.M, Nipuna Chamara *et al.* A Deep Convolutional Neural Network Based Image Processing Framework for Monitoring the Growth of Soybean Crops. *In*: 2021 ASABE ANNUAL INTERNATIONAL VIRTUAL MEETING, JULY 12-16, 2021. **2021 ASABE Annual International Virtual Meeting, July 12-16, 2021**. American Society of Agricultural and Biological Engineers, 2021. Disponível em: <http://elibrary.asabe.org/abstract.asp?JID=5&AID=52346&CID=virt2021&T=1>. Acesso em: 27 set. 2024

AHMAD, Aanis; SARASWAT, Dharmendra; EL GAMAL, Aly. A survey on using deep learning techniques for plant disease diagnosis and recommendations for development of appropriate tools. **Smart Agricultural Technology**, v. 3, p. 100083, fev. 2023.

BERNARDI, AC de C.; INAMASU, Ricardo Y. Adoção da agricultura de precisão no Brasil. 22 dez. 2014.

BURKOV, Andriy. **The hundred-page machine learning book**. Polen: Andriy Burkov, 2019.

CONAB. **Portal de Informações Agropecuárias**. Disponível em: <https://portaldeinformacoes.conab.gov.br>. Acesso em: 12 out. 2024.

DALL'AGNOL, Amélio. A soja no Brasil: evolução, causas, impactos e perspectivas. *In*: CONGRESO DE LA SOJA DEL MERCOSUR, 5; FORO DE LA SOJA ASIA 1. **MERCOSOJA 2011**. Rosário: 23 nov. 2011.

FAMASUL. **Produtividade da soja safra 2023-2024**: Boletim semanal casa rural - agricultura. Campo Grande: FAMASUL, 22 maio 2024. Disponível em: <https://portal.sistemafamasul.com.br/sites/default/files/boletimcasapdf/560%20-%20BOLETIM%20SEMANAL%20CASA%20RURAL%20-%20AGRICULTURA%20-%20CIRCULAR%20560%20%2022.05.2024%20-%20PRODUTIVIDADE%20DA%20SOJA%20SAFRA%202023-2024_1.pdf>. Acesso em: 14 out. 2024.

GOODFELLOW, Ian; COURVILLE, Aaron; BENGIO, Yoshua. **Deep learning**. Cambridge, Massachusetts: The MIT Press, 2016.

GOODFELLOW, Ian J. *et al.* **Generative Adversarial Networks**. arXiv, , 11 jun. 2014. Disponível em: <http://arxiv.org/abs/1406.2661>. Acesso em: 12 nov. 2025

HE, Kaiming *et al.* **Deep Residual Learning for Image Recognition**. arXiv, , 11 dez. 2015. Disponível em: <http://arxiv.org/abs/1512.03385>. Acesso em: 13 nov. 2025

HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-Term Memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1 nov. 1997.

INAMASU, Ricardo Yassushi *et al.* **Agricultura de Precisão: um novo olhar.** *[S.l.]*: Embrapa Instrumentação Agropecuária, 2011.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet classification with deep convolutional neural networks. **Communications of the ACM**, v. 60, n. 6, p. 84–90, 24 maio 2017.

LECUN, Yann *et al.* Gradient-Based Learning Applied to Document Recognition. p. 46, nov. 1998.

NIELSEN, Michael A. **Neural Networks and Deep Learning**. Disponível em: <http://neuralnetworksanddeeplearning.com>. Acesso em: 1 out. 2024.

ONNX 1.20.0 documentation. Disponível em: <https://onnx.ai/onnx/index.html>. Acesso em: 20 set. 2025.

PyTorch documentation — PyTorch 2.8 documentation. Disponível em: <https://docs.pytorch.org/docs/stable/index.html>. Acesso em: 7 out. 2025.

SEIXAS, Claudine Dinali Santos *et al.* **Tecnologias de produção de soja**. Londrina: Embrapa Soja, 2020.

SILVA, Wanderson De Vasconcelos Rodrigues Da; SILVA-MANN, Renata. Agricultura de Precisão no Brasil: conjuntura atual, desafios e perspectivas. **Research, Society and Development**, v. 9, n. 11, p. e1979119603, 11 nov. 2020.

TAULLI, Tom. **Introdução à Inteligência Artificial**. *[S.l.]*: Novatec Editora, 2019.

USDA. **Soybean Explorer**. Disponível em: <https://ipad.fas.usda.gov/cropexplorer/cropview/commodityView.aspx?cropid=2222000&sel_year=2023&rankby=Production>. Acesso em: 12 out. 2024.

WANG, Jindong; CHEN, Yiqiang. **Introduction to Transfer Learning: Algorithms and Practice**. Singapore: Springer Nature Singapore, 2023.

ZHANG, Jiu-Quan *et al.* Prediction of Soybean Growth and Development Using Artificial Neural Network and Statistical Models. **Acta Agronomica Sinica**, v. 35, n. 2, p. 341–347, fev. 2009.

APÊNDICES

APÊNDICE A – Instalação e execução do programa

Para execução do programa, é necessário que se tenha o JDK (Java development kit – Kit de desenvolvimento Java) instalado no computador. É recomendada a instalação da versão 21, por ser a versão estável mais recente disponível.

1. Acesse o *link* <https://www.oracle.com/java/technologies/javase/jdk21-archive-downloads.html> e faça o download do JDK para a versão adequada do sistema operacional.
    - No Windows, garanta que as variáveis de ambiente estejam configuradas corretamente. Essas variáveis são configuradas automaticamente se a instalação for realizada pelo instalador, opção “Windows x64 installer”.
2. Acesse o repositório do GitHub do autor, no *link* <https://github.com/GabrielChiodiI/redsia>, e procure pela opção “Program Download” no arquivo “README.md”, haverá o *link* do arquivo “redsia-1.0-complete.jar” no Google Drive, devido ao tamanho ser maior do que o GitHub suporta.
3. O sistema já deverá reconhecer o arquivo .jar como um programa java, dê dois cliques no arquivo ou pressione o botão direito do mouse e clique em abrir.
    - O programa também pode ser aberto através do terminal. Abra o terminal na pasta onde está o arquivo e insira o seguinte comando: “java -jar redsia-1.0-complete.jar”.
4. A pasta de validação está disponível também no GitHub, no arquivo “README.md”. Para testes, acesse o *link* em “validation data” para baixá-la.

APÊNDICE B – Execução do script de validação

O *script* de validação também está disponível para *download* no repositório do projeto, para usá-lo é necessário possuir o Python instalado no computador.

1. Acesse o *link* <https://www.python.org/downloads/windows/>, faça o *download* da versão estável mais recente na opção “Windows installer (64 bit)” no caso do Windows e instale-o. No *link* é possível encontrar também as versões para Linux e Mac OS.

2. Acesse o repositório do autor <https://github.com/GabrielChiodiI/redsia>, baixe o *script* “teste_validacao.py” e o arquivo “resnet50_soja.onnx” na seção “.onnx version”.
3. Baixe a pasta “plantio_14_10_2024_validacao” pelo *link* “validation data” no arquivo “README.md”.
4. Instale as dependências Python necessárias para a execução do *script*, utilize o seguinte comando: “pip install torch torchvision onnxruntime numpy scikit-learn matplotlib pillow”.
5. Com todos os arquivos baixados e no mesmo diretório (itens 2 e 3) e com as devidas dependências instaladas (item 4), rode o comando: “python teste_validacao.py”.

O relatório e a matriz de confusão utilizados foram extraídos através do passo a passo acima. O *hash* MD5 do modelo utilizado, “resnet50_soja.onnx”, é o seguinte: “8d43ee47c15bf8b17c0240be1ebb4d48”.

O mesmo passo a passo pode ser utilizado para o treinamento do modelo, basta baixar os dados de treinamento em “training data” e o *script* “soja_resnet50.py” no GitHub do projeto.