CENTRO UNIVERSITÁRIO FEI

JOAQUIM JONKEL MAGALHÃES MELO JUNIOR

Relatório do Projeto de Visão Computacional:

Classificação de pragas em plantações de algodão.

São Bernardo do Campo

2025

JOAQUIM JONKEL MAGALHÃES MELO JUNIOR

Relatório do Projeto de Visão Computacional:

Classificação de pragas em plantações de algodão.

Relatório da disciplina **CCI210 - VISÃO COMPUTACIONAL**. Orientado pelo Prof. Isaac Jesus da Silva.

São Bernardo do Campo

2025

Sumário

Indice de ilustrações

Indice de tabelas

1. Descrição do projeto:

Esse projeto foi desenvolvido em Python com a biblioteca OpenCV para identificar potenciais ameaças à produtividade agrícola, como lagartas e manchas bacterianas (pragas), de forma automática e precisa. Esse algoritmo permitirá a classificação e identificação precoce de vetores de doenças o que viabiliza tratamentos localizados, a redução do uso de pesticidas e o aumento da eficiência geral do manejo. Tendo em vista que, com base na análise das informações, será possível acionar ações automáticas, como o tratamento de plantas doentes com o controle químico (aplicação de fungicidas sistêmicos e protetores), o controle biológico e até geração de alertas para os agricultores como aplicações futuras.

2. Revisão do estado da arte:

A detecção automática de pragas e doenças agrícolas utilizando técnicas de visão computacional tem se destacado como uma das aplicações mais promissoras da Inteligência Artificial no agronegócio. O avanço de modelos de Deep Learning, especialmente as Redes Neurais Convolucionais (CNNs), permitiu superar métodos tradicionais baseados em extração manual de características, elevando significativamente a precisão e a robustez dos sistemas de diagnóstico.

Ferentinos et al. (2018) demonstrou que redes CNN podem alcançar acurácias superiores a 99% na classificação de doenças foliares em diversas culturas agrícolas, evidenciando que essa abordagem é altamente eficaz para capturar texturas, padrões de cor e formas características de infecções. Os resultados obtidos pelo autor reforçam o potencial das CNNs para uso prático no campo, sendo capazes de lidar com variações ambientais como mudanças de iluminação e ruídos de fundo.

Em estudos complementares, Fuentes et al. (2017) apresentou um sistema capaz de identificar pragas e doenças em plantas de tomate utilizando modelos convolucionais profundos e técnicas de detecção em tempo real. Seu trabalho mostrou que, mesmo sem ambientes controlados, redes profundas treinadas adequadamente são capazes de generalizar bem o suficiente para monitoramento contínuo em lavouras, apontando para aplicações em drones, robôs agrícolas e sistemas embarcados.

Nas técnicas de otimização para treinamento de redes, Kingma e Ba (2014) introduziram o otimizador Adam, que se tornou o padrão em tarefas de visão computacional devido à sua capacidade de ajustar dinamicamente os passos de atualização dos pesos por meio dos momentos de primeira e segunda ordem do gradiente. Posteriormente, Loshchilov e Hutter (2017) propuseram o AdamW, que corrige uma limitação do Adam tradicional ao separar corretamente o *weight decay* da atualização adaptativa. Essa melhoria torna o treinamento mais estável e favorece a generalização, aspecto especialmente crítico em datasets agrícolas, que frequentemente são pequenos e apresentam grande variabilidade visual.

Outra técnica essencial discutida na literatura é o *Data Augmentation*. Shorten e Khoshgoftaar (2019) consolidaram um amplo levantamento das técnicas de aumento artificial de dados, concluindo que transformações como rotação, inversão,

deslocamentos e zoom são fundamentais para reduzir *overfitting* e melhorar a robustez dos modelos. Isso é especialmente relevante para aplicações agrícolas, onde folhas e pragas podem ser registradas em diferentes orientações, distâncias e condições de iluminação.

Por fim, arquiteturas avançadas como VGG, ResNet, Inception, Xception, EfficientNet e ConvNeXt têm sido amplamente aplicadas via *Transfer Learning* em problemas agrícolas. Esses modelos pré-treinados no ImageNet fornecem poderosas representações visuais, mas sua efetividade depende fortemente do alinhamento entre o domínio das imagens e as classes do problema. Em tarefas de diagnóstico específico, como diferenciar pragas particulares do algodão, modelos treinados do zero muitas vezes superam os pré-treinados se estes não passarem por *fine-tuning* adequado.

3. Desenvolvimento do projeto:

Antes de iniciar a codificação, foram seguidas algumas etapas para o desenvolvimento do projeto, como escolher uma commodity agrícola de grande produção e importância para o Brasil, identificar o tipo de praga mais danosa a esse produto primário do agronegócio brasileiro, para então, buscar um banco de dados para realizar o treinamento do modelo.

Tendo essas etapas como base, foi definido que a commodity agrícola foi o algodão, a praga alvo: lagartas do algodão, com foco principal na lagarta-das-folhas (*Spodoptera eridania*). E como base de dados foi utilizado o dataset “Cotton Crop Disease Detection” (JAIN, 2025) retirado da plataforma Kaggle.

3.1. Algumas Imagens dos testes realizados:

Após o treinamento do modelo foi realizado um teste do modelo com três imagens externas sendo uma de uma lagarta do algodão ("cotton_leaf_worm.png"), uma folha do algodão doente ("cotton_leaf_disease.jpg") e uma folha do algodão saudável ("Healthy_cotton_leaf.png") para testar a acurácia do modelo.

Figura 1 - Imagens externas para o teste do modelo

Fonte: Autores.

A seguir estão representados os resultados do teste de predição para as três imagens acima:

Figura 2 – Resultado do teste de predição com as imagens externas

```
print("Generating test predictions...")
predict_x=model.predict(data_cotton)
print(np.around(predict_x, 2))

# make class predictions
predictions = (predict_x > 0.5).astype(int)

# Army_worm  | Bacterial_Blight | Healthy
```

```
Generating test predictions...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step
[[1.   0.   0.  ]
 [0.96 0.04 0.  ]
 [0.   0.   1.  ]]
```

Fonte: Autores.

Adicionalmente, o modelo também foi testado com imagens de pragas de outro tipo de matéria-prima de grande importância econômica, a soja. Para isso foram usadas três imagens externas sendo a lagarta do soja (“soybean_leaf_worm.jpg”), uma folha da soja doente (“soybean_leaf_disease.jpeg”) e uma folha da soja saudável (“Healthy_soybean_leaf.jpg”) para testar a acurácia do modelo.

Figura 3 - Imagens externas para o teste do modelo com a soja

Fonte: Autores.

E a seguir estão representados os resultados do teste de predição para as três imagens da soja acima:

Figura 4 - Resultado do teste de predição com as imagens externas da soja

```
print("Generating test predictions...")
predict_x=loaded_model.predict(data_soybean)
print(np.around(predict_x, 2))

# make class predictions
predictions = np.argmax(predict_x, axis=1)

# Army_worm  | Bacterial_Blight | Healthy
```

```
Generating test predictions...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
[[0.   0.09 0.91]
 [0.02 0.96 0.02]
 [0.   1.   0.  ]]
```

Fonte: Autores.

3.2. Os algoritmos utilizados e a justificativa:

A arquitetura Convolutional Neural Network (CNN) foi adotada neste trabalho por sua comprovada eficácia na extração automática de características discriminativas em imagens agrícolas. Ferentinos (2018) demonstrou que CNNs são capazes de alcançar acurácias superiores a 99% em tarefas de identificação de doenças e pragas em folhas, superando métodos convencionais e redes totalmente conectadas (MLPs) devido à capacidade dos filtros convolucionais em capturar texturas, bordas e padrões complexos presentes nas plantas. De forma complementar, Fuentes et al. (2017) reforça a adequação das CNNs para visão computacional aplicada à agricultura ao propor detecções robustas de pragas em campo usando deep learning, evidenciando que essas arquiteturas apresentam forte generalização mesmo em ambientes não controlados.

Para otimizar o processo de treinamento diante da complexidade visual e do número limitado de amostras do dataset, optou-se pelo uso do otimizador Adam (Adaptive Moment Estimation**)**, proposto por Kingma & Ba (2014). O Adam é amplamente utilizado em tarefas de visão computacional por ajustar automaticamente a taxa de aprendizado de cada parâmetro da rede, combinando momentum de primeira e segunda ordem, o que torna o treinamento mais rápido, estável e menos dependente de ajustes manuais de hiperparâmetros. Essas propriedades são especialmente relevantes quando se deseja obter bons resultados iniciais com poucas iterações e sem necessidade de extensa busca de hiperparâmetros, como ocorre neste projeto.

Além disso, adotou-se a variante AdamW, proposta por Loshchilov & Hutter (2017), que separa adequadamente o termo de decaimento de peso (*weight decay*) da atualização adaptativa dos gradientes. Essa abordagem corrige limitações presentes no Adam tradicional e melhora a capacidade de generalização do modelo, evitando *overfitting* — fator crítico em tarefas com conjuntos de dados agrícolas geralmente reduzidos e heterogêneos.

Outra técnica incorporada ao pipeline foi o *Data Augmentation*, essencial para aumentar a variabilidade do conjunto de treinamento e melhorar o desempenho da rede em cenários reais. Shorten & Khoshgoftaar (2019) demonstram que a aplicação de transformações como rotações, deslocamentos, reflexões e zoom pode aumentar significativamente a robustez dos modelos de deep learning, especialmente em domínios com imagens sujeitas a variações de iluminação, escala e posição — condições comuns em fotografias de folhas e pragas.

A métrica-alvo adotada foi o *Recall*, uma vez que, no contexto agrícola, o custo de não detectar uma praga quando ela de fato existe (falso negativo, FN) é elevado. Assim, priorizamos minimizar FN (Maximizar *Recall*), ainda que isso possa aumentar falsos positivos (FP).

$$Recall = \frac{TP}{TP + FN}$$

Por fim, para comparação de desempenho, também foram utilizados modelos convolucionais pré-treinados do Keras, como VGG19, ResNet50, ResNet152, InceptionV3, Xception, EfficientNetV2-L e ConvNeXt-XLarge. Esses modelos, amplamente descritos na literatura e treinados em grandes bases de dados como o

ImageNet, permitem avaliar o impacto de arquiteturas profundas e pré-treinadas em relação ao modelo desenvolvido do zero, garantindo uma análise comparativa mais abrangente do estado da arte.

3.3. Descrição detalhada e explicação do modelo de IA utilizado

O problema tratado neste projeto consiste na classificação automática de imagens de folhas de algodão em três classes: *Army_worm* (lagarta), *Bacterial_Blight* (doença bacteriana) e *Healthy* (folha saudável. Para isso, foi adotada uma abordagem de Aprendizado Profundo baseada em Redes Neurais Convolucionais (Convolutional Neural Networks – CNNs), por sua capacidade comprovada de extrair automaticamente características discriminativas de imagens agrícolas.

A seguir, é apresentada a descrição do pipeline completo, da arquitetura da rede, dos hiperparâmetros e das boas práticas adotadas, bem como a justificativa técnica de cada decisão.

3.3.1. Pipeline do modelo: da imagem bruta à predição

O fluxo de processamento pode ser organizado nas seguintes etapas principais:

3.3.1.1. Aquisição e organização do dataset

- Utilizou-se o dataset "Cotton Crop Disease Detection", contendo imagens rotuladas nas três classes de interesse.
- As imagens foram organizadas em subconjuntos de **treinamento**, **validação** e **teste**, buscando manter **proporções semelhantes de amostras por classe** em cada partição, de forma a evitar viés de treinamento em favor de uma classe específica. No relatório de classificação final, o suporte por classe no conjunto de teste é idêntico (1880 amostras), evidenciando um **dataset balanceado** na avaliação.

3.3.1.2. Pré-processamento das imagens

- **Redimensionamento** para (128, 128) pixels, padronizando a entrada da rede e reduzindo a complexidade computacional.
- **Conversão para array NumPy** e reorganização no formato (altura, largura, canais), compatível com o TensorFlow/Keras.
- **Normalização** da intensidade dos pixels para o intervalo [0, 1], dividindo os valores (0–255) por 255. Essa normalização evita que valores de entrada muito grandes dificultem a convergência do treinamento e favorece a estabilidade numérica.
- **Codificação dos rótulos** em formato numérico (*one-hot encoding*) para permitir o uso da função de perda de classificação multiclasse (*categorical cross-entropy*).

3.3.1.3. Aumento de dados (*Data Augmentation*)

- Sobre o conjunto de treinamento, aplicaram-se transformações como rotações, deslocamentos horizontais e verticais, espelhamento horizontal e zoom leve.
- O objetivo é expor a rede a variações de posição, orientação e escala das pragas e doenças, **aumentando a robustez** e reduzindo ***overfitting*** mesmo com um número relativamente limitado de imagens originais.

3.3.1.4. Treinamento da CNN

- As imagens pré-processadas são alimentadas na rede convolucional.
- A cada *batch*, é calculada a função de perda e os gradientes são propagados para trás (*backpropagation*), atualizando os pesos de acordo com o otimizador escolhido (Adam ou AdamW).
- Em cada época, o desempenho é avaliado no conjunto de validação para monitorar a evolução da **acurácia**, da **perda** e de métricas como ***Recall*** e **F1-score**.

3.3.1.5. Avaliação e inferência

- Após o treinamento, o modelo é avaliado no conjunto de teste e em imagens externas (folhas de algodão e de soja) para verificar a capacidade de generalização.
- Na inferência, a rede produz um vetor de probabilidades para cada classe, e a predição final corresponde à classe com maior probabilidade (saída da camada *SoftMax*).

3.3.2. Arquitetura da Rede Neural Convolucional

A arquitetura utilizada é uma **CNN de profundidade moderada**, projetada para ser suficientemente expressiva para o problema, porém com complexidade compatível com o tamanho do dataset disponível.

A rede pode ser descrita em dois blocos principais:

3.3.2.1. Bloco convolucional (extração de características)

O bloco convolucional é composto por **três camadas convolucionais** seguidas de operações de normalização e redução de dimensionalidade:

3.3.2.1.1. Conv2D_1 + BatchNorm + ReLU + MaxPooling

- **Filtros:** 64
- **Tamanho do kernel:** 3×3
- ***Stride*:** (1, 1)
- ***Padding*:** *same* (ou equivalente), preservando as dimensões espaciais antes do *pooling*.

- Função de ativação ReLU após a convolução.
- ***Max-pooling* 2×2** para reduzir pela metade a altura e a largura do mapa de características.

3.3.2.1.2. Conv2D_2 + BatchNorm + ReLU + MaxPooling

- Filtros**:** 64
- Demais parâmetros idênticos (kernel 3×3, *Stride* 1, *Padding same*).
- Novamente seguida de *Max-pooling* 2×2.

3.3.2.1.3. Conv2D_3 + BatchNorm + ReLU + MaxPooling

- Filtros**:** 128
- Kernel 3×3, *Stride* 1, *Padding same*.
- *Max-pooling* 2×2 para redução final da dimensão espacial.

Justificativa do tamanho do kernel (3×3): Kernels 3×3 são padrão em CNNs modernas, pois oferecem um bom compromisso entre **capacidade de captura de padrões locais** (bordas, texturas de manchas, contornos de lagartas) e **custo computacional**. Vários kernels 3×3 empilhados são capazes de simular campos receptivos grandes (por exemplo, duas camadas 3×3 equivalem a uma 5×5 em campo receptivo efetivo), mantendo menos parâmetros do que um único kernel grande, o que favorece **generalização** e **reduz *overfitting***.

Justificativa para o uso de *Stride 1* e *Padding same*:

- ***Stride 1*** garante que não haja perda de informação entre pixels adjacentes; cada filtro "varre" todos os possíveis alinhamentos da textura na imagem.
- ***Padding same*** preserva a dimensão espacial antes do *pooling*, o que facilita o design da arquitetura e assegura que as bordas da imagem (onde podem existir sinais de pragas) não sejam descartadas prematuramente.

Justificativa do *Max-pooling* 2×2: O *Max-pooling* 2×2 reduz a dimensão dos mapas de características em um fator de 2 em cada eixo, diminuindo o custo computacional e o risco de *overfitting*, além de trazer **invariância a pequenas translações**: se o padrão (por exemplo, um buraco na folha ou o corpo da lagarta) deslocar-se levemente, ainda assim o valor máximo da região será preservado. O tamanho 2×2 é suficiente para essa invariância sem tornar a resolução muito grosseira.

Uso de Batch Normalization: A **batch normalization** é aplicada após as convoluções para:

- Normalizar a distribuição das ativações em cada mini-lote;
- Permitir o uso de taxas de aprendizado mais altas;
- Reduzir sensibilidade à inicialização e atuar como regularizador leve, mitigando *overfitting*.

3.3.2.2. Bloco totalmente conectado (decisão de classe)

Após as camadas convolucionais e de *pooling*, os mapas de características são:

3.3.2.2.1. Flatten

- Transforma o tensor 3D (altura × largura × canais) em um vetor 1D.
- Esse vetor resume as características extraídas pelo bloco convolucional.

3.3.2.2.2. Camada densa intermediária (Dense_1)

- **128 neurônios**, com ativação ReLU.
- Atua como um nível de abstração mais alto, combinando características locais (texturas, bordas) em padrões mais globais (padrões típicos de manchas bacterianas, silhueta de lagartas etc.).
- O número 128 foi escolhido como um meio-termo entre capacidade expressiva e risco de *overfitting*: valores muito altos (como 512 ou 1024) tenderiam a memorizar o conjunto de treino, enquanto valores muito baixos poderiam ser insuficientes para separar as três classes com boa margem.

3.3.2.2.3. Camada de saída (Dense_2)

- **3 neurônios**, um para cada classe: *Army_worm*, *Bacterial_Blight* e *Healthy*.
- Função de ativação ***SoftMax***, que produz um vetor de probabilidades somando 1.
- A classe predita é aquela com maior probabilidade.

3.3.2.3. Critérios para definição do número de camadas

A escolha de **três camadas convolucionais** e **uma camada densa intermediária** não foi arbitrária, mas baseada em critérios estruturados:

3.3.2.3.1. Complexidade do problema

- O problema envolve **padrões visuais relativamente locais** (manchas, furos, textura da superfície da folha), para os quais redes muito profundas (como ResNet152) tendem a ser desnecessárias se treinadas do zero em datasets pequenos.
- Uma profundidade moderada permite que a rede construa uma hierarquia de representações:
  - Primeira camada: bordas e contrastes simples;
  - Segunda camada: pequenas texturas e manchas combinadas;
  - Terceira camada: padrões mais complexos e específicos das doenças/pragas.

3.3.2.3.2. Tamanho do dataset e risco de *overfitting*

- Com poucas milhares de imagens mesmo após *augmentation*, redes muito profundas com milhões de parâmetros tenderiam a **memorizar** o conjunto de treino.
- A arquitetura proposta apresenta **número de parâmetros compatível com o tamanho dos dados**, equilibrando capacidade de modelagem e generalização.

3.3.2.3.3. Eficiência computacional e implantação futura

- Um modelo relativamente leve facilita a **implantação em sistemas embarcados** ou dispositivos de menor capacidade computacional (por exemplo, robôs de campo ou drones).
- Esse fator prático justifica evitar arquiteturas excessivamente complexas.

Assim, a dupla decisão de **3 camadas convolucionais + 1 densa intermediária** busca Maximizar a eficiência de aprendizado para o problema específico, minimizando o risco de *overfitting* e mantendo viabilidade computacional.

3.3.3. Estratégias de regularização e controle de *overfitting*

Para combater *overfitting* e melhorar a capacidade de generalização, foram adotadas as seguintes estratégias formais:

3.3.3.1. *Data Augmentation*

- Gera variações realistas das imagens de treinamento sem necessidade de novas coletas em campo.
- Aumenta o número efetivo de exemplos e permite que o modelo aprenda a ignorar variações irrelevantes (pequenas rotações, zoom, espelhamentos).

3.3.3.2. Batch Normalization

- Reduz a covariância interna das ativações, estabilizando o treinamento e atuando como regularizador leve ao introduzir ruído estatístico nas ativações de cada mini-lote.

3.3.3.3. Dropout nas camadas densas

- No segundo modelo (AdamW), foram utilizadas camadas de **Dropout** entre o *flatten* e as camadas densas.
- O dropout zera aleatoriamente uma fração das ativações durante o treinamento, forçando a rede a não depender fortemente de poucos neurônios isolados, o que **reduz o risco de memorização** do conjunto de treino e melhora a robustez.

3.3.3.4. Monitoramento de métricas adicionais (F1-score)

- Além da acurácia, o relatório de classificação apresenta ***precision*, *recall* e F1-score** para cada classe.

- O **F1-score** é definido como:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

e representa a **média harmônica** entre *precision* e *recall*.

- Essa métrica é particularmente útil quando se deseja equilibrar a capacidade do modelo de **evitar falsos negativos** (alto *recall*) sem aumentar excessivamente os **falsos positivos** (queda de *precision*).
- Ao acompanhar o F1-score no conjunto de validação, é possível identificar situações em que a acurácia global pode parecer alta, mas o modelo está desequilibrado entre classes ou cometendo erros críticos em uma delas.

3.3.3.5. Comparação entre modelos (Adam vs. AdamW)

- O uso de dois otimizadores diferentes, mantendo a mesma arquitetura, funciona também como estratégia de controle de *overfitting*:
  - O modelo com Adam apresentou alta acurácia de treino, porém acurácia de validação inferior.
  - Com AdamW, a acurácia de validação aumentou, indicando melhor **generalização**.
- Essa comparação orienta a escolha do modelo final não apenas pela acurácia de treino, mas pelo comportamento em dados não vistos.

3.3.4. Boas práticas adotadas e outras recomendações

Além dos pontos já descritos, a solução segue diversas boas práticas em Visão Computacional e Deep Learning:

- **Dataset balanceado:** garantir números semelhantes de amostras por classe nas partições de treino/validação/teste para evitar viés do modelo.
- **Separação clara de treino, validação e teste:** evita a contaminação dos dados de avaliação.
- **Uso de métricas adequadas ao contexto agrícola:** prioridade para ***Recall*** e apoio do **F1-score**, devido ao alto custo de não detectar uma praga (falso negativo).
- **Avaliação com imagens externas e de outra cultura (soja):** permite verificar se o modelo realmente generaliza ou se está apenas memorizando o conjunto de treino de algodão.
- **Arquitetura moderadamente complexa:** atende ao problema sem exageros de profundidade e número de parâmetros.

Como extensões futuras, recomenda-se a inclusão de técnicas como ***early stopping*** (interromper o treinamento quando a perda de validação não melhora após certo número de épocas) e, caso o dataset seja ampliado, avaliação de **modelos um pouco mais profundos** ou ***fine-tuning* de modelos pré-treinados** especificamente no domínio de folhas de algodão.

3.3.5. Hiperparâmetros utilizados e justificativas

A Tabela a seguir (que pode ser adicionada no relatório, se desejado) sintetizaria os principais hiperparâmetros e a justificativa de cada um. Aqui, textualizamos:

- **Dimensão de entrada:** 128×128×3
    - Compromisso entre preservação de detalhes visuais e custo computacional.
- **Número de camadas convolucionais:** 3
    - Suficiente para aprender hierarquias de características relevantes sem excesso de profundidade, dada a quantidade de dados e o problema específico.
- **Número de filtros por camada:** 64, 64 e 128
    - Aumentar o número de filtros nas camadas mais profundas permite capturar padrões mais complexos à medida que a dimensão espacial diminui (via *pooling*), mantendo o custo computacional controlado nas camadas iniciais.
- **Tamanho do kernel:** 3×3
    - Padrão moderno que equilibra capacidade de captura de padrões locais e número de parâmetros, favorecendo generalização.
- ***Stride*:** 1 nas convoluções
    - Garante exploração completa da imagem sem perda de resolução espacial antes do *pooling*.
- ***Padding*:** *same*
    - Preserva as dimensões espaciais, evitando perda de informação nas bordas e simplificando o design da rede.
- ***Max-pooling*:** janelas 2×2
    - Redução controlada da dimensionalidade, garantindo invariância a pequenas translações sem agressiva perda de resolução.
- **Camada densa intermediária:** 128 neurônios, ReLU
    - Número de neurônios suficiente para combinar características extraídas pela CNN, sem exagero de parâmetros.

- **Camada de saída:** 3 neurônios, *SoftMax*
    - Correspondente às três classes do problema, com interpretação probabilística.
- **Função de ativação nas camadas ocultas:** ReLU
    - Introduz não linearidade sem saturar gradientes, acelerando a convergência.
- **Otimizadores:** Adam e AdamW
    - Adam pela rapidez de convergência; AdamW por melhor generalização via *weight decay* desacoplado.
- **Número de épocas:**
    - Primeiro modelo (Adam): 15 épocas, adequado para um modelo inicial e para evitar treinamentos muito longos com risco de *overfitting*.
    - Segundo modelo (AdamW): 50 épocas, explorando maior tempo de treinamento, compensado pelo melhor comportamento de generalização do otimizador e pelas técnicas de regularização (dropout).
- **Tamanho do batch (batch_size):** 20 (Adam) e 35 (AdamW)
    - Batches pequenos tendem a introduzir mais ruído nas estimativas de gradiente, oferecendo leve efeito de regularização; no modelo AdamW, um batch um pouco maior equilibra estabilidade do gradiente com custo computacional.

3.3.6. Explicação matemática dos otimizadores Adam e AdamW

Os pesos da rede são atualizados a cada iteração a partir dos gradientes $g_t$da função de perda em relação aos parâmetros. O **Adam** (*Adaptive Moment Estimation*) calcula médias móveis de primeira e segunda ordem dos gradientes:

- Primeira ordem (média do gradiente):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

- Segunda ordem (média do quadrado do gradiente):

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Essas estimativas são então corrigidas para viés:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

E os pesos são atualizados por:

$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

onde αé a taxa de aprendizado e ϵé um termo pequeno para estabilidade numérica.

O **AdamW** introduz uma correção importante: o termo de *weight decay* (regularização L2) é **desacoplado** da atualização adaptativa:

$$w_{t+1} = w_t - \alpha(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda w_t)$$

onde λé o coeficiente de *weight decay*. Em vez de incorporar a regularização na própria função de perda, ela é aplicada diretamente sobre os pesos, o que torna o comportamento da regularização **mais previsível** e eficaz, melhorando a capacidade de generalização e reduzindo *overfitting*, como observado na comparação entre os modelos. (KINGMA, D, 2014)

4. Experimentos e Resultados:

Antes do treinamento da rede convolucional, as imagens passaram por etapas de pré-processamento, como: redimensionamento para as imagens terem tamanhos consistentes (128,128) e para facilitar não só sua visualização e extração de características, como também o treinamento do modelo. Além disso, conversão da imagem para um formato de array Numpy (isso é necessário para o processamento de imagens com bibliotecas como o Tensorflow) e o armazenamento da classe de cada imagem em uma variável "labels". Depois a intensidade dos pixels foi dimensionada para escala [0, 1], essa normalização ajuda a aprimorar a performance do treinamento de redes neurais.

Além disso, foi utilizada a técnica de *Data Augmentation* no conjunto de treino para aumentar não só o tamanho do dataset, como também a capacidade de classificação da rede para diferentes condições, como rotação, deslocamentos horizontal e vertical, espelhamento horizontal e um pequeno zoom nas imagens. Totalizando 4320 imagens.

Figura 5 – Visualização de algumas imagens do treinamento

Fonte: Autores.

Para o primeiro modelo, foram utilizados o algoritmo "Adam" e três camadas convolucionais, sendo as duas primeiras com 64 e a terceira com 128 filtros, responsáveis pela extração de padrões de baixo nível (bordas e contrastes). Uma camada de Flatten, que converte os mapas de características em um vetor unidimensional. Duas camadas densas (*Fully Connected*), a primeira camada tem 128 neurônios, responsável por combinar padrões de alto nível para formar representações mais abstratas, e a camada de saída (dense_1) tem 3 neurônios, camada de saída para classificação multiclasse, sendo elas: *Army_worm*, *Bacterial_Blight* e *Healthy*. A função de ativação usada foi a *Rectified Linear Unit* (ReLU) que é a mais comumente utilizada em redes neurais para introduzir não linearidade nos modelos, o que é crucial para aprender padrões complexos nos dados, evitando saturação e acelerando a convergência. A técnica de normalização em lotes (*batch normalization*) foi usada para acelerar o treinamento da rede, permitir o uso de taxas de aprendizado maiores, tornar o modelo mais estável e menos sensível à inicialização dos pesos, e atuar como uma forma de regularização, reduzindo a necessidade de *Dropout* em alguns casos.

Ademais, foi utilizado o *Max-pooling* para reduzir a dimensionalidade dos mapas de características (*feature maps*) de uma camada convolucional anterior, que nesse caso de (2,2), ela pega uma janela de 2x2 pixels e seleciona o valor máximo dentro dessa janela. Essa operação é então deslizada sobre todo o mapa de características para diminuir a quantidade computacional necessária, o que ajuda a controlar o *overfitting* e torna o modelo mais robusto a pequenas variações na posição dos recursos na imagem, pois o valor máximo será detectado mesmo se o recurso se mover ligeiramente. E na última camada está a função de ativação *SoftMax* tipicamente utilizada em modelos de classificação multiclasse (como este, que classifica em 3 classes).

Tabela 1 - Model summary do algoritmo Adam

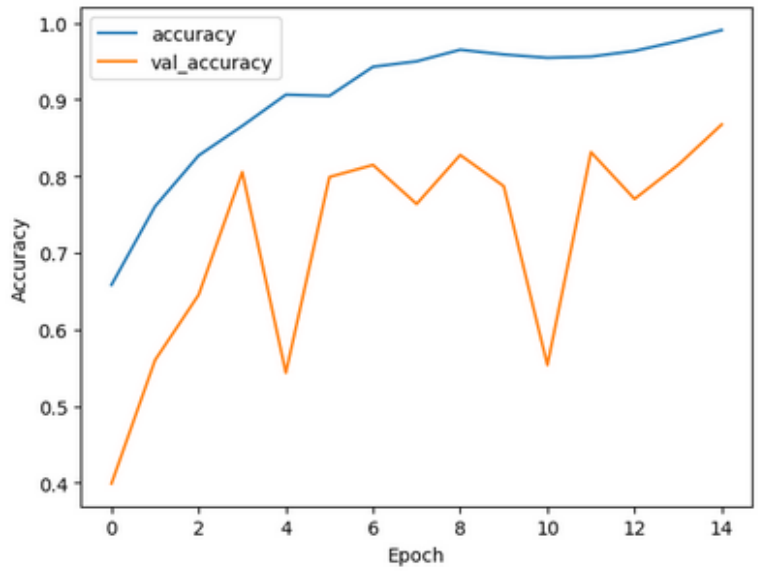
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 64)	1,792
batch_normalization (BatchNormalization)	(None, 126, 126, 64)	256
max_pooling2d (MaxPooling2D)	(None, 63, 63, 64)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	36,928
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dropout (Dropout)	(None, 25088)	0
dense (Dense)	(None, 128)	3,211,392
dense_1 (Dense)	(None, 3)	387

Total params: 3,325,379 (12.69 MB)
Trainable params: 3,324,867 (12.68 MB)
Non-trainable params: 512 (2.00 KB)

Fonte: Autores.

Na compilação do modelo, os hiperparâmetros adotados foram número de épocas igual a 15 (*epochs*), tamanho da amostra de 20 (*batch_size*) e 25% do conjunto de dados de treinamento foi separado para avaliar o modelo (*val_accuracy*) .

Figura 6 - Gráfico da acurácia e da avaliação do modelo Adam em função da época



Fonte: Autores.

Figura 7 - Treinamento, acurácia de treinamento e de validação do modelo Adam

```
Epoch 1/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 10s 28ms/step - accuracy: 0.6101 - loss: 2.3774 - val_accuracy: 0.3991 - val_loss: 1.4905
Epoch 2/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - accuracy: 0.7678 - loss: 0.6073 - val_accuracy: 0.5602 - val_loss: 0.9291
Epoch 3/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - accuracy: 0.8293 - loss: 0.4247 - val_accuracy: 0.6454 - val_loss: 0.8180
Epoch 4/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.8786 - loss: 0.3479 - val_accuracy: 0.8056 - val_loss: 0.4942
Epoch 5/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - accuracy: 0.9045 - loss: 0.2451 - val_accuracy: 0.5435 - val_loss: 4.5035
Epoch 6/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step - accuracy: 0.9121 - loss: 0.2378 - val_accuracy: 0.7991 - val_loss: 0.5976
Epoch 7/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - accuracy: 0.9464 - loss: 0.1512 - val_accuracy: 0.8148 - val_loss: 0.7031
Epoch 8/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - accuracy: 0.9580 - loss: 0.1138 - val_accuracy: 0.7639 - val_loss: 0.8070
Epoch 9/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step - accuracy: 0.9679 - loss: 0.0932 - val_accuracy: 0.8278 - val_loss: 0.6889
Epoch 10/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - accuracy: 0.9701 - loss: 0.0880 - val_accuracy: 0.7870 - val_loss: 0.8993
Epoch 11/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.9568 - loss: 0.1575 - val_accuracy: 0.5537 - val_loss: 4.8244
Epoch 12/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step - accuracy: 0.9597 - loss: 0.1287 - val_accuracy: 0.8315 - val_loss: 0.6939
Epoch 13/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.9659 - loss: 0.0951 - val_accuracy: 0.7704 - val_loss: 0.9118
Epoch 14/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 22ms/step - accuracy: 0.9749 - loss: 0.0784 - val_accuracy: 0.8148 - val_loss: 0.7105
Epoch 15/15
162/162 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step - accuracy: 0.9927 - loss: 0.0274 - val_accuracy: 0.8676 - val_loss: 0.5368
```

```
print(f"Acurácia de treinamento: {history.history['accuracy'][-1]*100:.2f}%")
print(f"Acurácia de validação: {history.history['val_accuracy'][-1]*100:.2f}%")
```

```
Acurácia de treinamento: 99.07%
Acurácia de validação: 86.76%
```

Fonte: Autores.

Com base nos resultados, é possível fazer algumas conclusões, começando pela acurácia de treinamento alta, o que significa que o modelo aprendeu a identificar bem as características de cada imagem e a classificar cada uma em suas respectivas classes. Por outro lado, a acurácia de validação ter resultado em um valor abaixo dos 90% pode indicar que o modelo decorou as imagens e não possui uma grande capacidade de generalização

indicando um possível *overfitting* apesar das imagens terem passado por diferentes técnicas de pré-processamento para evitar esse problema.

Figura 8 - Relatório de classificação do modelo Adam

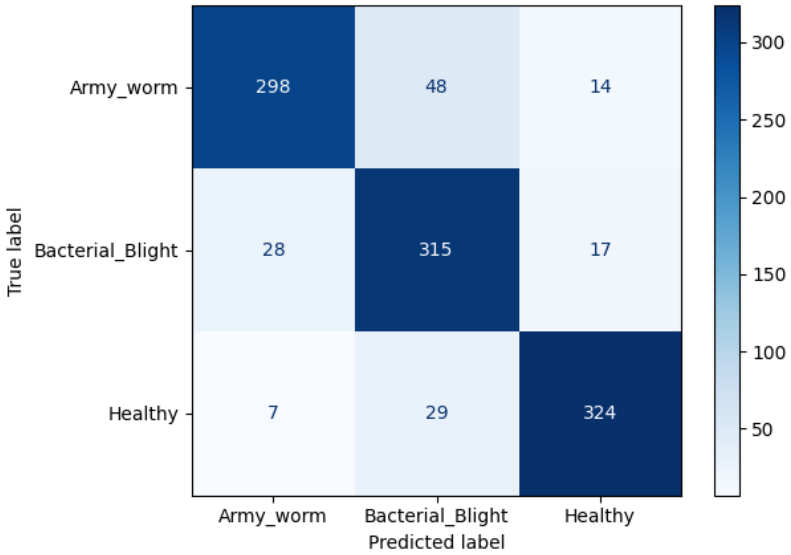
🎯 Acurácia de teste: 86.76%

📊 Classification Report:

	precision	recall	f1-score	support
Army_worm	0.89	0.83	0.86	360
Bacterial_Blight	0.80	0.88	0.84	360
Healthy	0.91	0.90	0.91	360
accuracy			0.87	1080
macro avg	0.87	0.87	0.87	1080
weighted avg	0.87	0.87	0.87	1080

Fonte: Autores.

Tabela 2 - Matriz de confusão do modelo Adam



Fonte: Autores.

No segundo modelo, utilizou-se o otimizador AdamW, mantendo a arquitetura básica de CNN, com ajustes nos hiperparâmetros (50 épocas; *batch size* = 35). Esse otimizador separa o *weight decay* da atualização adaptativa dos gradientes, reduzindo *overfitting* e favorecendo a generalização.

Tabela 3 - Model summary do algoritmo AdamW

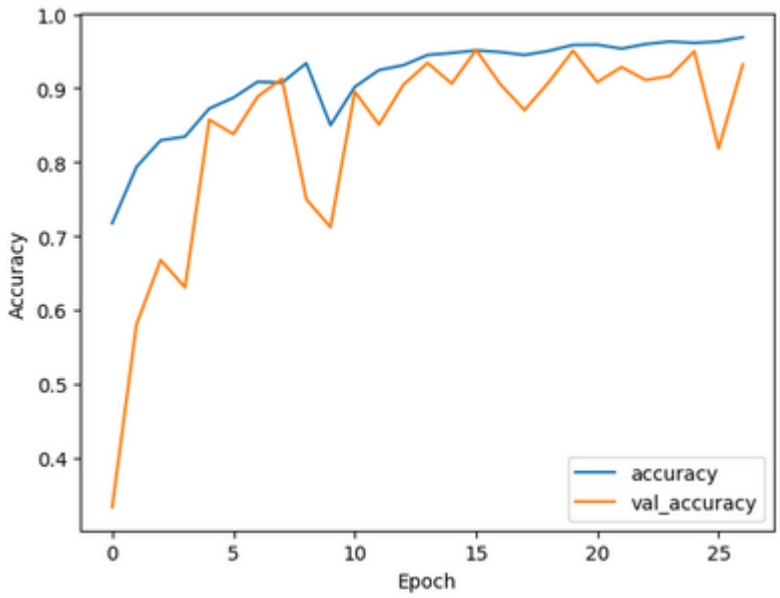
Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 128, 128, 32)	896
batch_normalization_4 (BatchNormalization)	(None, 128, 128, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_4 (Conv2D)	(None, 64, 64, 64)	18,496
batch_normalization_5 (BatchNormalization)	(None, 64, 64, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_5 (Conv2D)	(None, 32, 32, 128)	73,856
batch_normalization_6 (BatchNormalization)	(None, 32, 32, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 128)	0
flatten_1 (Flatten)	(None, 32768)	0
dropout_1 (Dropout)	(None, 32768)	0
dense_2 (Dense)	(None, 128)	4,194,432
batch_normalization_7 (BatchNormalization)	(None, 128)	512
dense_3 (Dense)	(None, 3)	387

Total params: 4,289,475 (16.36 MB)
Trainable params: 4,288,771 (16.36 MB)
Non-trainable params: 704 (2.75 KB)

Fonte: Autores.

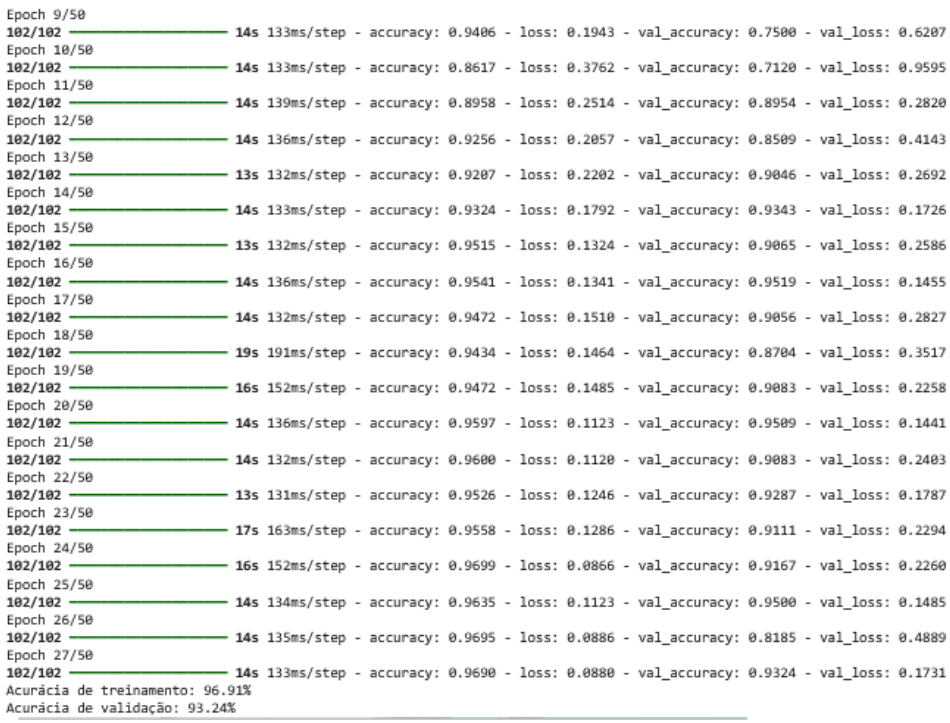
Na compilação do modelo AdamW, os hiperparâmetros adotados foram diferentes no número de épocas igual a 50 (*epochs*) e no tamanho da amostra de 35 (*batch_size*).

Figura 9 - Gráfico da acurácia e da avaliação do modelo AdamW em função da época



Fonte: Autores.

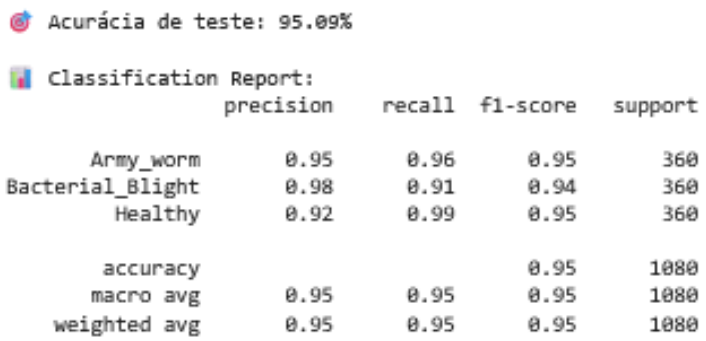
Figura 10 - Treinamento, acurácia de treinamento e de validação do modelo AdamW



Fonte: Autores.

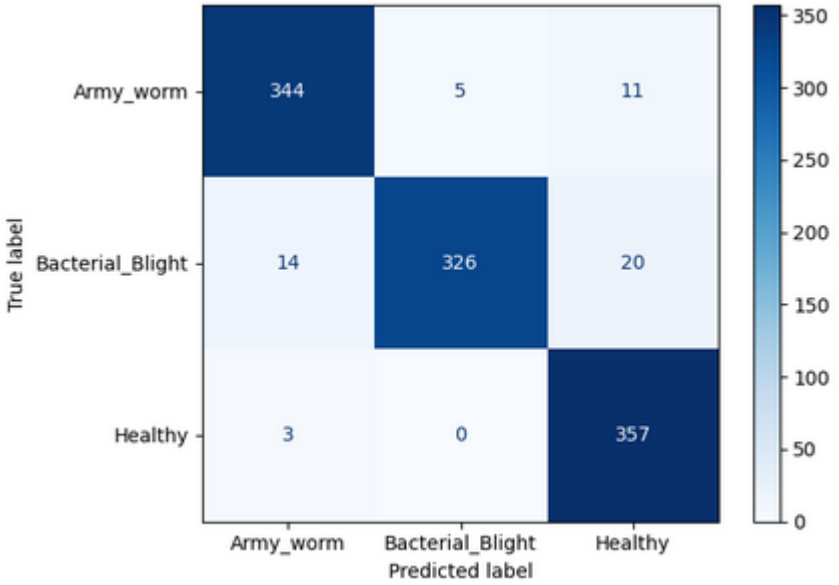
Com base nos resultados, é possível observar que, embora a acurácia de treinamento tenha diminuído cerca de 2%, a acurácia de validação subiu quase 7% o que evidencia a melhora na capacidade de generalização do novo modelo utilizando o algoritmo Adam adaptado evitando *overfitting*.

Figura 11 - Relatório de classificação do modelo AdamW



Fonte: Autores.

Tabela 4 - Matriz de confusão do modelo AdamW



Fonte: Autores.

Por fim, foi avaliada a capacidade de generalização "fora do domínio" em modelos pré-treinados do Keras (VGG19, ResNet50, ResNet152, InceptionV3, Xception, EfficientNetV2-L e ConvNeXt-XLarge), sem ajuste fino no dataset de algodão. Como esperado, sem *fine-tuning* e com rótulos não alinhados, os modelos não distinguiram corretamente o tipo de lagarta, como é possível ver nos testes abaixo:

Figura 12 - VGG19

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
Predicted:
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
35363/35363 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
('n02259212', 'leafhopper', np.float32(0.37797314))
('n02168699', 'long-horned_beetle', np.float32(0.21829747))
('n02226429', 'grasshopper', np.float32(0.119693935))
('n02169497', 'leaf_beetle', np.float32(0.09203239))
('n02264363', 'lacewing', np.float32(0.091095954))
```

Fonte: Autores.

Figura 13 - ResNet50

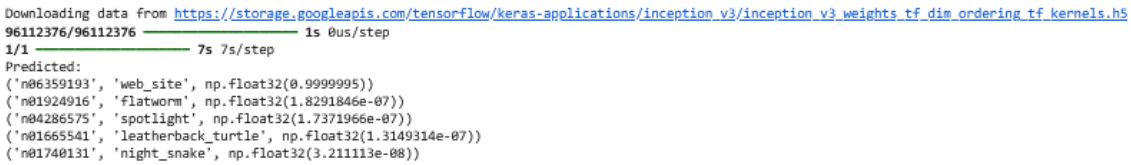
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels.h5
102967424/102967424 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Predicted:
('n02259212', 'leafhopper', np.float32(0.5249273))
('n02264363', 'lacewing', np.float32(0.29590476))
('n02226429', 'grasshopper', np.float32(0.038533915))
('n01689811', 'alligator_lizard', np.float32(0.032496378))
('n01784675', 'centipede', np.float32(0.024999736))
```

Fonte: Autores.

Figura 14 - ResNet152

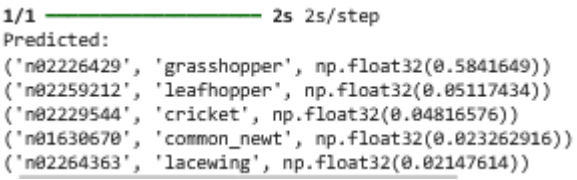
Fonte: Autores.

Figura 15 - Inception V3



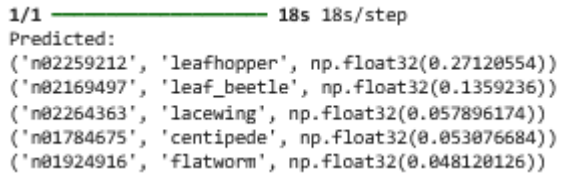
Fonte: Autores.

Figura 16 - Xception



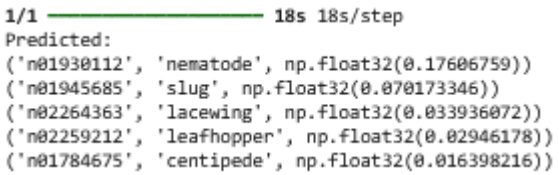
Fonte: Autores.

Figura 17 - EfficientNet V2L



Fonte: Autores.

Figura 18 - ConvNeXt XLarge



Fonte: Autores.

E a conclusão foi que nenhum dos modelos, mesmo que apresentassem datasets enormes, conseguiu identificar qual tipo de lagarta era, nem se era uma lagarta.

5. Conclusão:

O desenvolvimento deste projeto permitiu compreender de forma clara os fatores que influenciam o desempenho de um modelo de classificação de pragas em plantações de algodão utilizando técnicas de visão computacional. Ao longo das etapas de pré-processamento, treinamento e avaliação, verificou-se que a escolha da arquitetura, dos hiperparâmetros e das estratégias de regularização é determinante para alcançar resultados consistentes. Também se constatou que, mesmo com a repetição dos mesmos hiperparâmetros, o treinamento não produz resultados idênticos devido à natureza estocástica das redes neurais.

As etapas de pré-processamento mostraram-se fundamentais. O redimensionamento das imagens possibilitou que a rede captasse melhor os detalhes discriminativos das classes, enquanto o *Data Augmentation* aumentou a variabilidade do conjunto de treinamento, reduzindo a probabilidade de *overfitting*. Transformações como rotação, deslocamentos, espelhamento e zoom contribuíram para melhorar a robustez do modelo, e o embaralhamento das amostras impediu que ele aprendesse padrões artificiais relacionados à ordem das imagens.

Os experimentos com o primeiro modelo, treinado com o otimizador Adam, revelaram uma acurácia de treinamento próxima de 99%, mas uma acurácia de validação de aproximadamente 87%, indicando forte tendência ao *overfitting*. Essa suspeita foi confirmada em testes com imagens externas e com um conjunto de teste realístico, no qual o modelo atingiu apenas 57,22% de acurácia. Embora tenha identificado corretamente algumas imagens próximas ao seu domínio, apresentou dificuldades significativas com lagartas do algodão e confundiu folhas doentes com outras classes, destacando limitações em sua capacidade de generalização.

A introdução do segundo modelo, treinado com o otimizador AdamW, permitiu observar avanços importantes. Embora sua acurácia de treinamento tenha sido ligeiramente inferior, houve um aumento substancial na acurácia de validação, indicando uma melhor capacidade de generalização. Essa melhoria decorre da forma como AdamW separa o termo de *weight decay* da atualização dos gradientes, corrigindo limitações do Adam tradicional e reduzindo o risco de *overfitting*. Assim, a comparação entre ambos os modelos demonstra que o Adam é eficiente para convergência rápida, porém mais suscetível à memorização do conjunto de treinamento, enquanto o AdamW apresentou desempenho mais equilibrado, favorecendo a estabilidade e a generalização — fatores essenciais para aplicações em cenários reais.

Os testes externos reforçaram essas conclusões. Mesmo quando exposto a imagens nunca vistas, o modelo buscou enquadrá-las em uma das três classes aprendidas (*Army_worm*, *Bacterial_Blight* e *Healthy*), comportamento esperado em classificadores fechados. Contudo, persistiram dificuldades em lidar com imagens de outras culturas (como soja) e com pragas diferentes, o que evidencia a necessidade de datasets mais amplos e diversificados para aplicações agrícolas reais e mais robustas.

De modo geral, o projeto demonstrou o potencial das CNNs para a classificação automática de pragas em plantações, mas também destacou desafios como a limitação dos conjuntos de dados disponíveis, a dificuldade de generalização para condições reais

e a necessidade de mecanismos que permitam ao modelo identificar imagens fora do seu domínio. Conclui-se que, embora o sistema apresente bom desempenho em ambientes controlados, aprimoramentos relacionados ao tamanho, à diversidade do dataset e à inclusão de técnicas complementares — como métodos de rejeição ou detecção de anomalias — são essenciais para torná-lo mais confiável e aplicável ao manejo agrícola em campo.

REFERÊNCIAS:

FERENTINOS, K. P. **Deep learning models for plant disease detection and diagnosis.** ***Computers and Electronics in Agriculture***, 2018.

FUENTES, A. et al. **A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition**. *Sensors*, 2017.

JAIN, Paridhi. ***Cotton Crop Disease Detection***. Kaggle, 2021. Disponível em: https://www.kaggle.com/datasets/paridhijain02122001/cotton-crop-disease-detection. Acesso em: 21 out. 2025.

KINGMA, D. P.; BA, J. Adam: **A method for stochastic optimization**. *arXiv:1412.6980*, 2014.

LOSHCHILOV, I.; HUTTER, F. **Decoupled *weight decay* regularization**. *arXiv:1711.05101*, 2017.

IOFFE, S.; SZEGEDY, C**.** ***Batch normalization*****: accelerating deep network training by reducing internal covariate** shift. *arXiv:1502.03167*, 2015.

SHORTEN, C.; KHOSHGOFTAAR, T. M. **A survey on image *Data Augmentation* for deep learning**. *Journal of Big Data*, 2019.