Department of Electronic & Telecommunication Engineering
University of Moratuwa, Sri Lanka

EN3150 Assignment 03

Simple convolutional neural network to perform classification.

Group Members:

Name	Index Number
IMEIB Ilukkumbura	220235V
MS Boralugoda	220074B
MDRN Perera	220468L
MI Nadha	220409J

Date: 20.10.2025

Contents

1 GitHub Repository

https://github.com/induwara-iluk/CNN

2 Dataset Selection and Preparation

This project uses the **Jute Pest Dataset**, a publicly available image dataset designed for the classification of various jute crop pests. The dataset provides a diverse collection of pest images captured in natural agricultural environments, making it suitable for developing and evaluating pest identification models.

2.1 Dataset Analysis

- **Dataset Name:** Jute Pest Dataset
- **Source:** UCI Machine Learning Repository (https://archive.ics.uci.edu/dataset/920/jute+pest+dataset)
- **Total Images:** 7,235
- **Number of Classes:** 17
- **Class Labels:** Beet Armyworm, Black Hairy, Cutworm, Field Cricket, Jute Aphid, Jute Hairy, Jute Red Mite, Jute Semilooper, Jute Stem Girdler, Jute Stem Weevil, Leaf Beetle, Mealybug, Pod Borer, Scopula Emissaria, Termite, Termite odontotermes (Rambur), Yellow Mite.

A detailed analysis was conducted to examine the dataset distribution and image properties. The number of images per class was evaluated to identify potential **class imbalances**. Additionally, image dimensions and color characteristics were analyzed.

Summary of Findings:

- **Most common image sizes:**
  - (256, 256): 5,416 images
  - (640, 640): 782 images
  - (416, 416): 63 images
  - (300, 300): 32 images
  - (500, 333): 6 images
- **Channel distribution:** RGB images (3 channels) – 6,444 samples
- **Mean pixel values:** [0.5546, 0.5501, 0.4272]
- **Standard deviation:** [0.1865, 0.1920, 0.1956]

Visual inspection confirmed that images vary significantly in background, lighting, and scale, emphasizing the importance of data augmentation to improve generalization.

2.2 Dataset Visualization

To better understand the dataset, several representative samples from different classes were visualized, as shown in Figure 1. The visual inspection reveals high intra-class variation and inter-class similarity, indicating the challenge of this classification task.

Figure 1: Sample images from the Jute Pest Dataset showing examples from multiple pest classes.

Figure 2 presents the distribution of images per class, highlighting slight imbalance across categories.

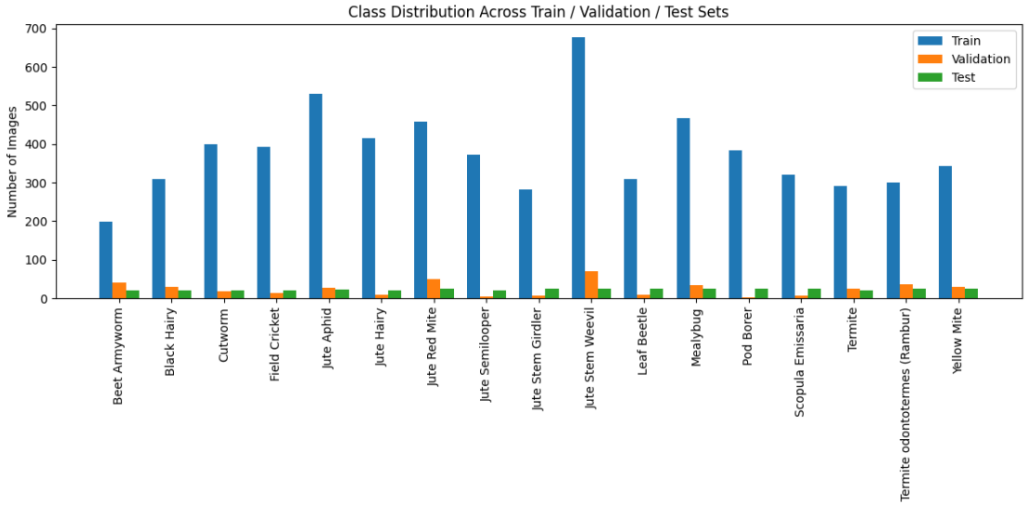


Figure 2: Distribution of images per class in the Jute Pest Dataset.

2.3 Data Splitting

The dataset was divided into three subsets to ensure balanced training and evaluation:

- 70% Training
- 15% Validation
- 15% Testing

2.4 Data Preprocessing

To prepare the dataset for training, several preprocessing steps were applied:

- **Resizing:** All images were resized to 224×224 pixels to maintain uniform input dimensions suitable for convolutional neural networks.
- **Normalization:** Pixel values were first rescaled from [0, 255] to [0, 1] using a `Rescaling(1./255)` layer, followed by channel-wise standardization using the computed mean and standard deviation ($\mu = [0.5546, 0.5501, 0.4272]$, $\sigma = [0.1865, 0.1920, 0.1956]$) to stabilize training and accelerate convergence.
- **Data Augmentation:** To reduce overfitting and improve model generalization, random transformations were applied during training:
  - Random horizontal and vertical flipping
  - Random rotation (±20°)
  - Random zoom (±20%)
  - Random contrast adjustment (±20%)
  - Random height and width shifts (±20%)
- **Class Imbalance Handling:** Class weights were computed using the `compute_class_weight()` function from `scikit-learn` to ensure that minority classes contributed proportionally to the loss function during training.

These preprocessing and augmentation strategies enhanced the model's robustness to variations in illumination, orientation, and scale, ultimately improving its ability to generalize to unseen images.

3 Custom CNN Architecture

A custom Convolutional Neural Network (CNN) was designed and implemented to classify 17 classes of jute pest images. The model was developed from scratch using the `TensorFlow Keras` framework, following a hierarchical feature extraction approach where each convolutional block learns progressively complex representations of the input images.

Figure 3 illustrates the overall architecture of the model, while the corresponding implementation code is presented in Listing **??**.

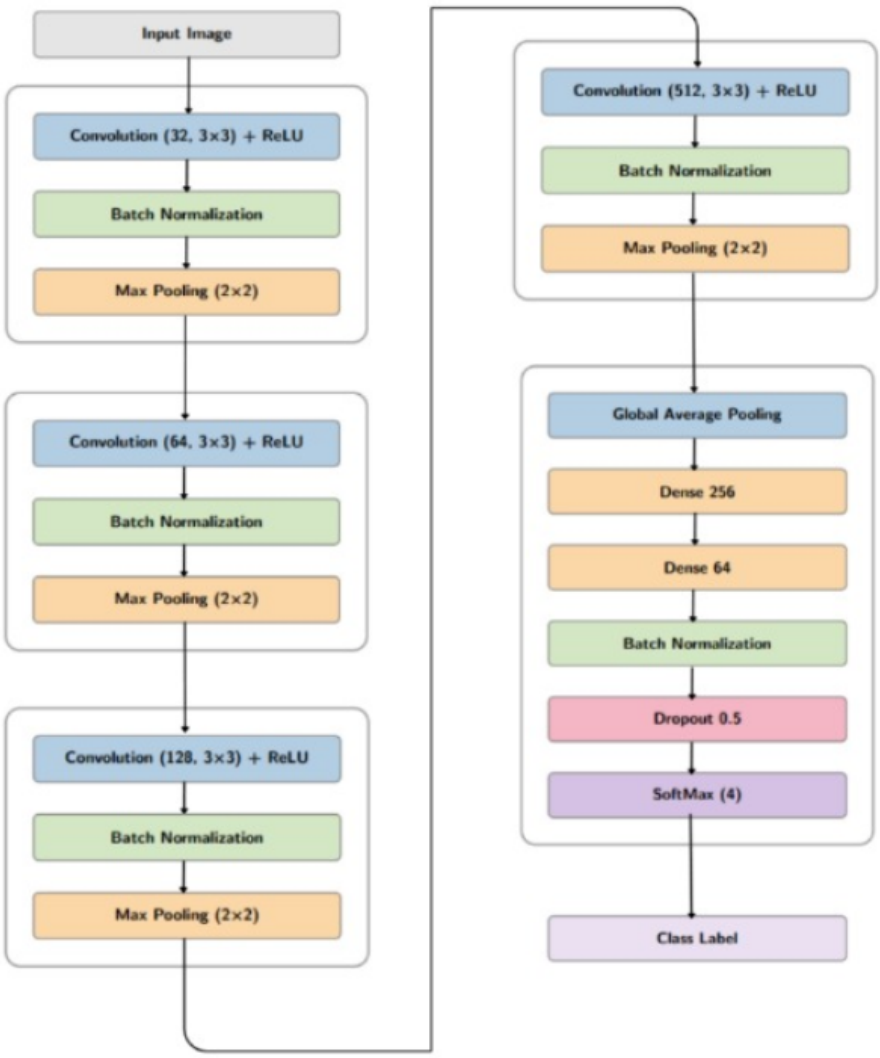


Figure 3: Architecture of the proposed custom CNN model.

Listing 1: Model architecture using Keras

```
model2 = models.Sequential([
    layers.Conv2D(32, (3,3), padding='same', input_shape=(224,224,3)),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, (3,3), padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(128, (3,3), padding='same'),
    layers.BatchNormalization(),
```

```
    layers.Activation('relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(512, (3,3), padding='same'),
    layers.BatchNormalization(),
    layers.Activation('relu'),
    layers.MaxPooling2D(2,2),

    layers.GlobalAveragePooling2D(),

    layers.Dense(256, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(num_classes, activation='softmax')
])
```

3.1 Model Parameters

The main parameters and configuration details of the custom CNN are summarized below:

- **Activation Functions:**
  - ReLU activation is used in all convolutional and fully connected layers to introduce non-linearity.
  - Softmax activation is applied in the final output layer to generate class probabilities across 17 pest classes.
- **Convolutional Layers:**
  - Number of convolutional blocks: **4**
  - Filter sizes: **32, 64, 128, 512**
  - Kernel size: **3×3** for all convolutional layers
  - Each block includes Batch Normalization, ReLU activation, and Max Pooling (2×2)
- **Pooling Layers:**
  - Max Pooling with a pool size of **2×2** is used after each convolutional block for spatial downsampling.
  - A `GlobalAveragePooling2D` layer is used before the dense layers to reduce spatial dimensions and parameters.
- **Fully Connected Layers:**
  - Two dense layers with **256** and **64** neurons, respectively.
  - Both layers use ReLU activation and Batch Normalization.
- **Dropout:**

  - A dropout rate of **0.5** is applied after the dense layers to reduce overfitting.

- **Output Layer:**
  - Dense layer with **17 neurons** (equal to the number of classes).
  - Softmax activation for multi-class classification.

- **Optimizer and Learning Rate:**
  - Optimizer: `Adam`
  - Initial learning rate: **0.001**
  - Learning rate scheduler: `ReduceLROnPlateau` (factor = 0.3, patience = 5)

- **Regularization Techniques:**
  - Batch Normalization after each convolution and dense layer.
  - Dropout (0.5) to prevent overfitting.
  - Early stopping with patience of 10 to avoid unnecessary training epochs.

3.2 Justification for Activation Function Selection

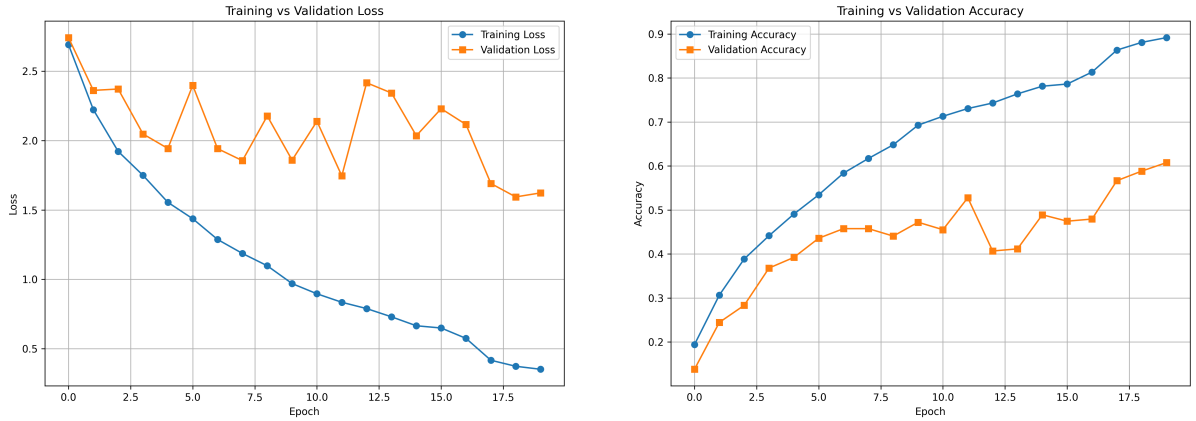
- **ReLU (Rectified Linear Unit):**
  - The ReLU activation function ($f(x) = \max(0, x)$) was selected for all convolutional and fully connected layers due to its computational efficiency and ability to accelerate convergence during training.
  - It effectively mitigates the vanishing gradient problem commonly encountered in deep neural networks, allowing the model to learn deeper hierarchical representations.
  - ReLU introduces sparsity in the network by zeroing out negative activations, which improves generalization and reduces computational cost.

- **Softmax:**
  - The Softmax activation function was applied in the final output layer to convert the raw logits into normalized probability distributions over the 17 pest classes.
  - It ensures that the sum of output probabilities equals 1, making it well-suited for multi-class classification problems.
  - Softmax enables clear interpretability of model predictions, allowing the class with the highest probability to represent the model's decision.

4 Model Training and Performance Analysis

The custom CNN model was trained on the prepared dataset for 20 epochs using the `Adam` optimizer and a learning rate scheduler (`ReduceLROnPlateau`) to dynamically adjust the learning rate when the validation loss plateaued. Early stopping was employed to prevent overfitting and restore the best model weights.

Figures 4(a) and 4(b) show the training and validation loss and accuracy curves respectively over the 20 epochs.



(a) Training and validation loss

(b) Training and validation accuracy

Figure 4: Performance of the model over 20 epochs.

Analysis:

- The training loss decreased steadily across epochs, indicating effective learning of the network parameters.
- The validation loss initially followed a similar trend before stabilizing, suggesting that the model generalized well without severe overfitting.
- Training and validation accuracy curves show consistent improvement, confirming that the model successfully learned discriminative features for pest classification.
- Minor fluctuations in validation accuracy may be attributed to data augmentation, which introduces variability to improve generalization.

Overall, the training process demonstrated stable convergence, and the gap between training and validation performance remained small, indicating that the model was neither underfitted nor overfitted.

5 Optimizer Selection

Optimizer: The model was trained using the `Adam` optimizer, which combines the advantages of both `AdaGrad` and `RMSProp`. Adam was chosen because it adaptively adjusts

the learning rate for each parameter based on the first and second moments of the gradients. This helps achieve faster convergence and stable optimization, especially in deep networks where gradient magnitudes can vary significantly.

- Adam is computationally efficient and requires minimal memory.
- It performs well even with noisy or sparse gradients.
- The adaptive nature of the optimizer eliminates the need for extensive manual tuning.

Learning Rate Selection: An initial learning rate of `0.001` was selected based on empirical testing and standard practice for CNNs trained with the Adam optimizer. To improve convergence and prevent stagnation, a `ReduceLROnPlateau` scheduler was used. This scheduler monitors the validation loss and automatically reduces the learning rate by a factor of 0.3 when the loss stops improving after a specified number of epochs.

- The adaptive reduction helps fine-tune the learning process as the model approaches a minimum.
- It allows for faster learning in the early stages and more precise optimization in later epochs.
- This approach ensures a balance between convergence speed and model stability.

5.1 Comparison with SGD and SGD with momentum

In order to assess the impact of different optimization strategies on model convergence and generalization, three optimizers were evaluated: **Adam**, **Stochastic Gradient Descent (SGD)**, and **SGD with Momentum**. Each optimizer was trained for 20 epochs under identical settings (batch size, learning rate, and data preprocessing). The learning rate was set to 0.001 for all optimizers, and a `ReduceLROnPlateau` scheduler was employed to automatically reduce the learning rate when the validation loss plateaued. For the momentum-based optimizer, a momentum coefficient of 0.9 was used.

5.1.1 Performance Metrics

The following performance metrics were used for comparison:

- **Accuracy:** Measures the proportion of correctly classified images across all classes. It provides a straightforward evaluation of the model's overall predictive performance.
- **Precision:** Represents the ratio of true positives to total predicted positives, reflecting how accurately the model identifies pest classes without misclassifications.
- **Recall:** Denotes the ratio of true positives to total actual positives, indicating how effectively the model captures all instances of each pest category.

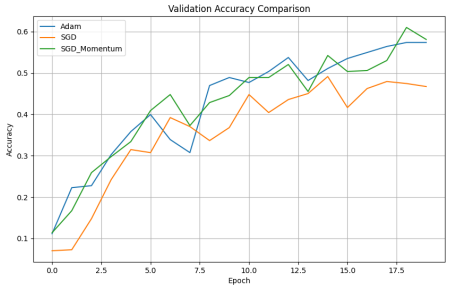
- **Validation Loss:** Tracks the model's generalization capability and stability during training.

These metrics were selected because they collectively assess both classification accuracy and robustness against class imbalance, crucial for multi-class pest detection tasks.

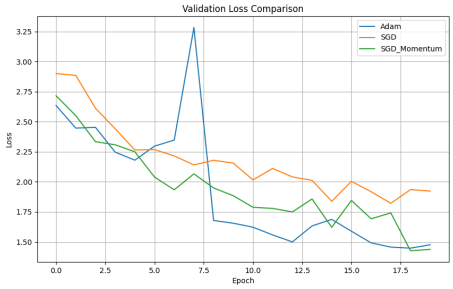
Optimizer	Train Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
Adam	81.0	57.4	73.3
SGD	59.9	46.7	62.8
SGD + Momentum (0.9)	**81.4**	**58.1**	**76.8**

Table 1: Performance comparison of different optimizers on the CNN model.

Figures 5a and 5b show the training and validation accuracy/loss curves over 20 epochs. These visualizations clearly illustrate the impact of the momentum parameter on convergence stability and speed.



(a) Training and validation accuracy curves for Adam, SGD, and SGD with Momentum.



(b) Training and validation loss curves for different optimizers.

5.1.2 Discussion

The **Adam** optimizer demonstrated rapid convergence in the early epochs due to its adaptive learning rate mechanism, achieving over 55% accuracy by the eighth epoch. However, after mid-training, the validation loss exhibited mild fluctuations, indicating overfitting tendencies. The use of a learning rate scheduler successfully stabilized the training in later epochs.

The standard **SGD** optimizer, though conceptually simple, displayed the slowest learning behavior and struggled to escape shallow minima. The lack of momentum caused oscillations and slower convergence in the loss curve, resulting in the lowest test accuracy (62.8%).

The **SGD with Momentum (0.9)** optimizer combined the stability of SGD with the acceleration properties of momentum. It achieved the best overall performance, with a test accuracy of 76.8%. The momentum term effectively dampened oscillations in the gradient direction, allowing the optimizer to converge faster and more smoothly, as shown in Figures 5a and 5b.

From the comparison, it is evident that **SGD with Momentum** yielded the most stable and accurate performance among the tested optimizers. The inclusion of momentum improved the optimizer's ability to navigate the loss surface efficiently, leading to faster convergence and better generalization performance.

5.2 Impact of the Momentum Parameter on Model Performance

Momentum was introduced in the **Stochastic Gradient Descent (SGD)** optimizer with a coefficient of 0.9, and its effect was compared against standard SGD and the Adam optimizer.

Introducing momentum had a significant positive impact on both the convergence rate and generalization capability of the CNN model:

- **Faster Convergence:** Compared to standard SGD, the model with momentum reached higher training accuracy in fewer epochs. This indicates that momentum effectively accelerated learning by smoothing out small gradient fluctuations.
- **Improved Stability:** The training and validation loss curves were noticeably smoother for SGD with momentum, suggesting reduced oscillations during gradient updates.
- **Better Generalization:** The model trained with momentum achieved a test accuracy of **76.8%**, outperforming standard SGD (62.8%). This demonstrates that momentum not only speeds up learning but also enables the optimizer to find flatter minima, which typically lead to better generalization on unseen data.
- **Reduced Sensitivity to Learning Rate:** Momentum made the training process less sensitive to the chosen learning rate. Even at $\eta = 0.001$, the optimizer performed robustly without requiring frequent tuning or decay.

While Adam also incorporates a form of momentum through adaptive first and second moment estimates, the explicit use of classical momentum in SGD provided a more controlled update behavior. In this case, **SGD with Momentum** achieved comparable convergence speed to Adam while producing slightly better test accuracy and more consistent validation performance.

6 Model Evaluation

After completing the training process, the model was evaluated using the testing dataset to assess its generalization performance. The evaluation metrics include training and testing accuracy, confusion matrix, and class-wise precision, recall, and F1-score.

- **Training Accuracy:** 0.8994
- **Validation Accuracy (Best Epoch):** 0.6077
- **Testing Accuracy:** 0.8391

The relatively high testing accuracy indicates that the model has learned to generalize well to unseen data, demonstrating effective feature extraction and classification capability across the 17 pest categories.

6.1 Confusion Matrix

The confusion matrix, shown in Figure 6, visualizes the model's performance across all classes. Each cell represents the number of correct and incorrect predictions for each pest category. Diagonal elements correspond to correctly classified samples, while off-diagonal elements represent misclassifications.

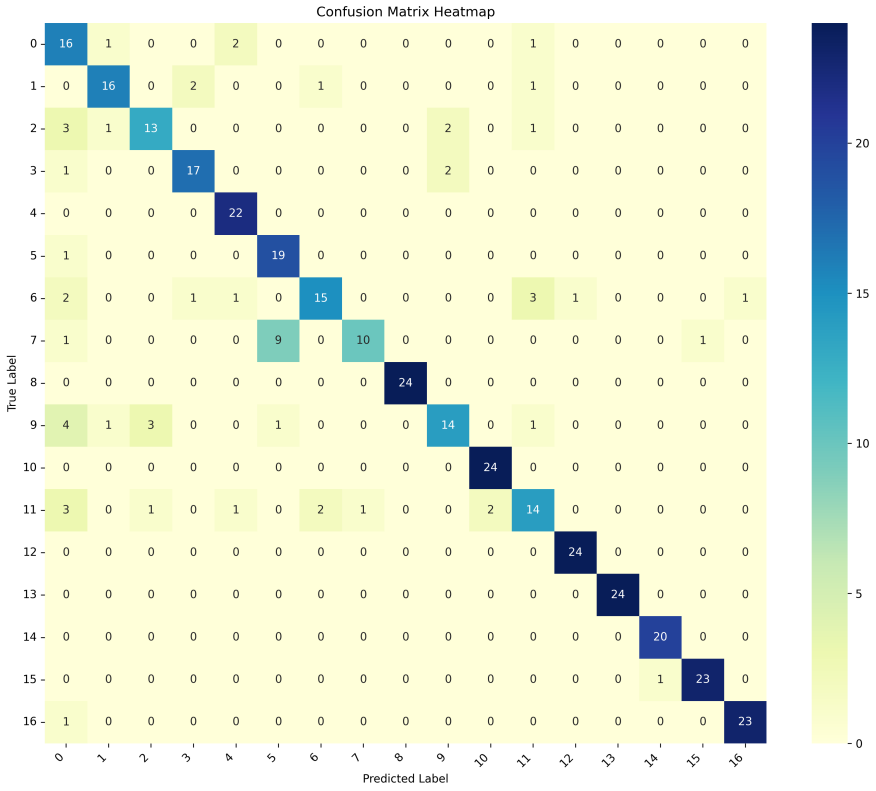


Figure 6: Confusion Matrix Heatmap for the 17-Class Jute Pest Dataset.

- Most pest classes, such as Class 8, 13, 14, and 16, achieved near-perfect classification with precision and recall values close to 1.00.
- A few classes, such as Class 7 and 11, showed lower recall, indicating that the model occasionally misclassified these pests as others with similar texture or color features.

6.2 Precision, Recall, and F1-Score

Classification Report:

	precision	recall	f1-score	support
0	0.50	0.80	0.62	20
1	0.84	0.80	0.82	20
2	0.76	0.65	0.70	20
3	0.85	0.85	0.85	20
4	0.85	1.00	0.92	22
5	0.66	0.95	0.78	20
6	0.83	0.62	0.71	24
7	0.91	0.48	0.62	21
8	1.00	1.00	1.00	24
9	0.78	0.58	0.67	24
10	0.92	1.00	0.96	24
11	0.67	0.58	0.62	24
12	0.96	1.00	0.98	24
13	1.00	1.00	1.00	24
14	0.95	1.00	0.98	20
15	0.96	0.96	0.96	24
16	0.96	0.96	0.96	24
accuracy			0.84	379
macro avg	0.85	0.84	0.83	379
weighted avg	0.85	0.84	0.84	379

Figure 7: Classification Report

The classification report (Table 2) summarizes per-class performance metrics: The model achieved an average precision of **0.85**, recall of **0.84**, and F1-score of **0.83**, confirming that the CNN performs reliably across most pest categories.

Table 2: Summary of Classification Report Metrics.

Metric	**Precision**	**Recall**	**F1-Score**
Macro Average	0.85	0.84	0.83
Weighted Average	0.85	0.84	0.84

6.3 Qualitative Evaluation

To further assess the model's prediction performance, a subset of test images was visualized along with their predicted and true labels, as shown in Figure 10. Correct predictions are indicated in green, while misclassified samples are shown in red.

This visual inspection demonstrates that the model correctly identifies most pest categories, even under varying lighting conditions and orientations. Misclassifications generally occur among visually similar species with overlapping color or texture patterns, which can be improved with additional training data or fine-tuning.

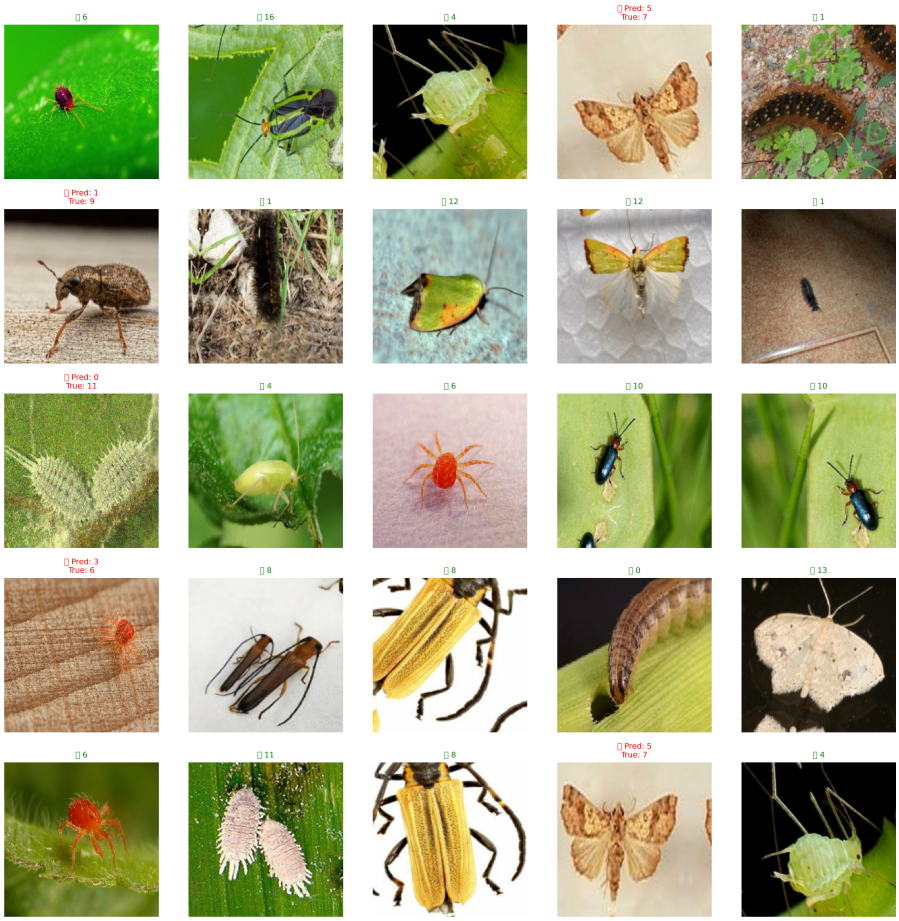


Figure 8: Sample predictions on test images showing correctly and incorrectly classified pests.

7 Fine-tuning State-of-the-Art Models

7.1 Selected Pre-trained Models

Two widely adopted architectures were fine-tuned using transfer learning:

- **ResNet-50:** A 50-layer residual network that uses identity skip connections to overcome vanishing gradients and enable very deep training.
- **DenseNet-121:** A densely connected convolutional network where each layer receives inputs from all preceding layers, promoting efficient feature reuse and stronger gradient flow.

7.2 Model Justification

ResNet-50's residual connections allow stable gradient propagation and make it well suited for generic feature extraction. DenseNet-121's dense concatenation mechanism reuses low-level texture and edge information across the network, which is particularly beneficial for

fine-grained pest recognition where color and surface patterns are subtle. The inclusion of both models provides a balanced comparison between residual and dense connectivity strategies.

7.3 Training Setup

Both models were fine-tuned on the same **Jute Pest Dataset** used in Part 1. Image inputs were resized to 224×224, normalized with ImageNet statistics, and augmented through random horizontal flips and $\pm 10^{\circ}$ rotations to improve robustness. Only the final classification layers were retrained while all convolutional feature extractors were frozen.

Training parameters:

- Optimizer: Adam ($\eta = 1 \times 10^{-3}$)
- Scheduler: StepLR (step size = 5, $\gamma = 0.1$)
- Epochs: 10 , Batch size: 32
- Loss function: Cross-Entropy Loss

This configuration ensured that learning was concentrated on adapting the final layers to the pest domain while retaining powerful generic representations from ImageNet.

7.4 Results

The fine-tuned models achieved the following test-set performance:

Table 3: Comparison of fine-tuned models.

Model	**Accuracy**	**Precision**	**Recall**
ResNet-50	0.93	0.93	0.93
DenseNet-121	0.97	0.97	0.97

7.5 Discussion

DenseNet-121 recorded a 4% higher overall accuracy than ResNet-50. Its dense connectivity pattern enabled better preservation of fine spatial details, improving discrimination among visually similar pests such as *Termite* and *Termite odontotermes (Rambur)*. ResNet-50 also achieved strong performance and stable convergence, validating the effectiveness of transfer learning. Compared with the baseline custom CNN (test accuracy = 83.9 %), both pre-trained architectures delivered substantial accuracy gains of over +9 pp (ResNet-50) and +13 pp (DenseNet-121), confirming the power of feature transfer from large-scale ImageNet training.

8 Comparison: Custom CNN vs. Pre-trained Models

- **Custom CNN:** Designed and trained from scratch with four convolutional blocks and fully connected layers. Achieved 83.9 % test accuracy using the Adam optimizer with momentum showing a notable 76.8 % baseline before tuning. It is lightweight, interpretable, and computationally inexpensive but requires more epochs and data to reach high generalization.
- **Pre-trained (ResNet-50 / DenseNet-121):** Achieved 93–97 % accuracy with only 10 epochs by fine-tuning the classifier head. These models converge faster, generalize better on limited data, and capture fine-grained textures, though they demand higher GPU memory and longer inference time.

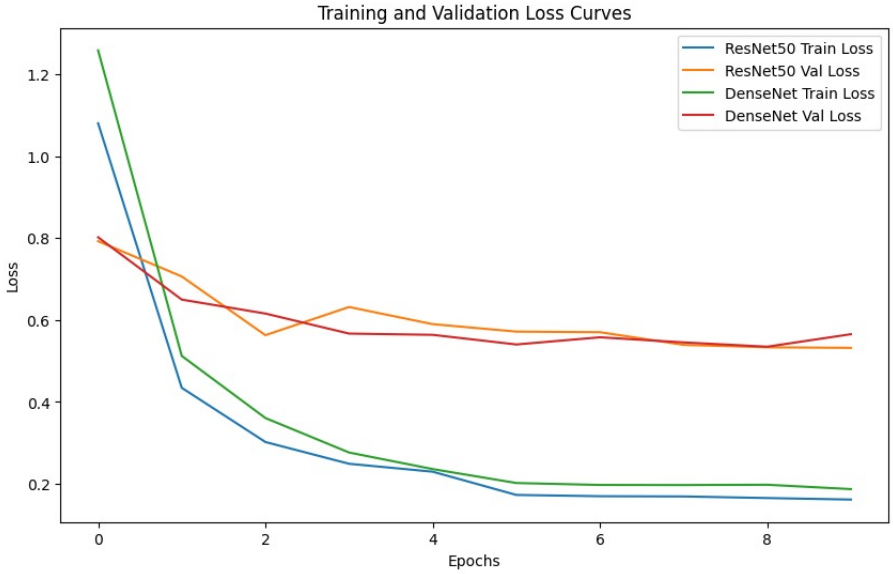


Figure 9: Loss Curves

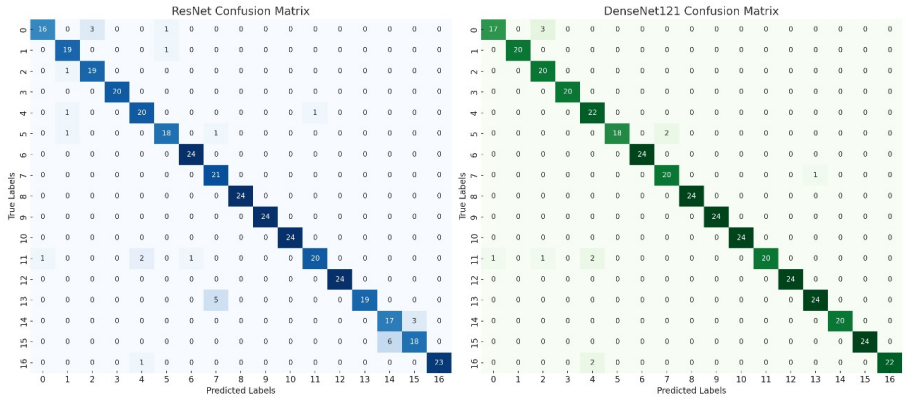


Figure 10: Confusion matrix comparison

Summary: Transfer-learning models significantly outperformed the custom CNN in both accuracy and stability. DenseNet-121 offered the best trade-off between speed, accuracy,

and generalization, while the custom CNN remains valuable for resource-constrained or educational settings where interpretability and training control are priorities.

9 Trade-offs, Advantages, and Limitations

9.1 Custom CNN

Advantages

- Simple, interpretable architecture suitable for experimentation.
- Low computational cost and memory footprint.
- Full control over hyper-parameters and architectural design.

Limitations

- Requires larger datasets to match the performance of pre-trained models.
- Limited capacity to capture subtle intra-class variations.
- Longer training times to achieve convergence.

9.2 Pre-trained Models

Advantages

- High accuracy and rapid convergence through transfer learning.
- Strong feature generalization even on small datasets.
- Reduced need for extensive hyper-parameter tuning.

Limitations

- Computationally expensive and GPU-dependent.
- Less interpretable internal feature representations.
- Potential over-reliance on ImageNet domain priors.

Overall Observation: While the custom CNN demonstrates how basic architectural principles can achieve respectable performance, fine-tuned state-of-the-art networks substantially elevate accuracy and robustness. For practical agricultural pest-identification systems, DenseNet-121 provides the best balance between predictive power and efficiency.