UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Artificial Intelligence and Machine Learning.

6CS012

An Image Classification with Convolutional Neural Network

University Id	: 2358277
Student Name	: Himani Shrestha
Group	: L6CG22
Tutor	: Mr. Aatiz Ghimire

Abstract

This project is about classifying pest images using deep learning. The goal was to create a model that can correctly identify different types of pests from pictures. Two types of Convolutional Neural Networks (CNNs) were used: one built from scratch (a basic version and a deeper version) and one using a pre-trained model called MobileNetV2 through transfer learning.

The basic CNN model quickly learned the training data but didn’t perform well on new images, showing signs of overfitting. The deeper model was slightly better at generalizing but still didn’t achieve high accuracy. On the other hand, using transfer learning with MobileNetV2 gave much better results. MobileNetV2 reached around 90% validation accuracy and performed well even with fewer training steps and less resources.

These results show that using pre-trained models like MobileNetV2 is a great choice when working with smaller datasets. They save time, improve accuracy, and help reduce overfitting. In the future, the model could be improved by using a bigger and more varied dataset, adding more data augmentation, or trying out other advanced models. This project demonstrates that transfer learning is a powerful and efficient method for image classification, especially when there are limited resources.

Tables of Contents

Table of Figures:

1. Introduction

In the process of improving computer vision, image classification plays an important role by allowing computers to automatically recognize and group based in their content. This ability to understand images has become a key strength in developing both commercial and non-commercial technologies. However, Image Classification is not just about comparing random images. In today’s world, Convolutional Neural Networks (CNNs) are widely used for this task as it can automatically learn important features like edges, textures, shapes directly from the raw images, without manual feature design.

In this project, we are trying to classify different types of pests using deep learning. The main aim is to create a model that can correctly recognize pests by looking at their pictures. This kind of system can be useful in real life, as identifying pests manually is time-consuming and can lead to delays in managing crop damage.

Here, We applied a Convolutional Neural Network (CNN) model to classify images for pest classification. We built and evaluated both a basic CNN model and a deeper CNN architecture to analyze their performance. Additionally, we implemented transfer learning by using the pre-trained MobileNet model to improve results with fewer training resources. Finally, we compared all the models in terms of accuracy, training efficiency, and overall performance to determine the most effective approach for pest image classification.

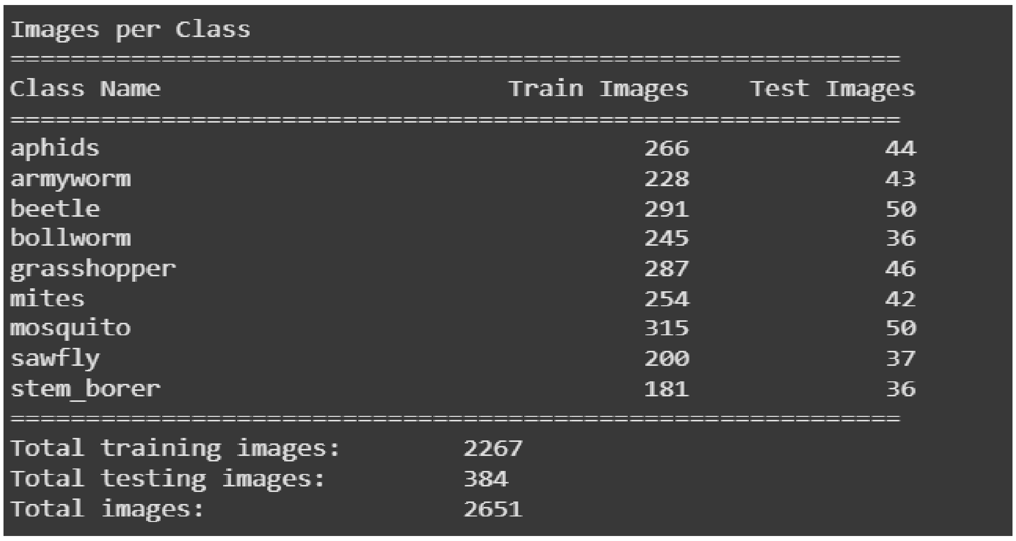
2. Dataset

2.1. Source

The dataset used in this project, InsectV2 Dataset was taken from Kaggle and was originally uploaded by **Yash Dogra** and authored by multiple authors (Imrus Salehin, Mahbubur Rahman Khan, Ummya Habiba, Nazmul Huda, Badhon Nazmun, Nessa Moon). It contains labelled images of pests and consists of 9 classes:

- aphids
- armyworm
- beetle
- bollworm
- grasshopper
- mites
- mosquito
- sawfly
- stem borer

There are about 2,651 images in total, after splitting, the training set contains 2,267 images and the testing set contains 384 images. All images were resized to 150x150 pixels in RGB format for consistency. Preprocessing involved normalizing pixel values to the range [0,1] and applying data augmentation techniques such as rotation, flipping, zooming, shifting, and shearing to increase data variability. We set aside 20% of the training data for validation. Although the dataset was imbalanced (for example, 236 mosquito images compared to 145 stem borer images), class weights were used to address the imbalance and minimize bias during training. No corrupted images were found during preprocessing.



```
Images per Class
============================================================
Class Name                        Train Images    Test Images
============================================================
aphids                                     266             44
armyworm                                   228             43
beetle                                     291             50
bollworm                                   245             36
grasshopper                                287             46
mites                                      254             42
mosquito                                   315             50
sawfly                                     200             37
stem_borer                                 181             36
============================================================
Total training images:         2267
Total testing images:          384
Total images:                  2651
```

Figure 1: Images Per Class

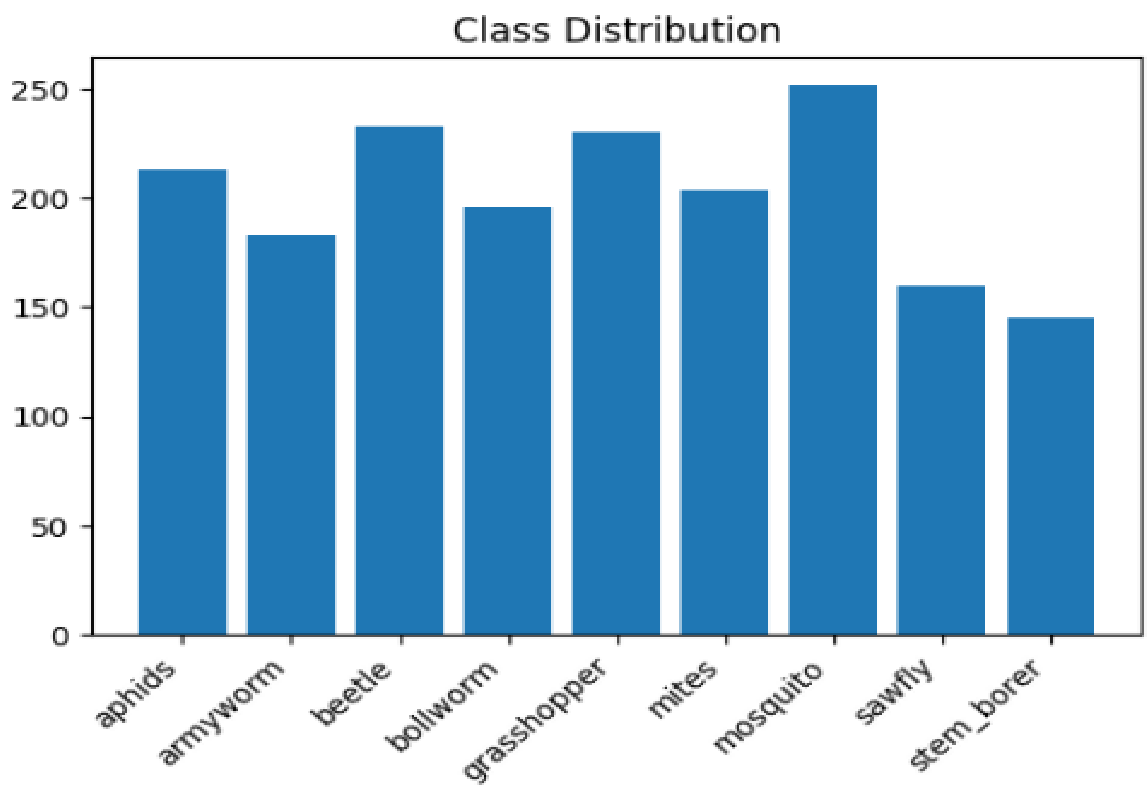


Figure 2: Class Distribution

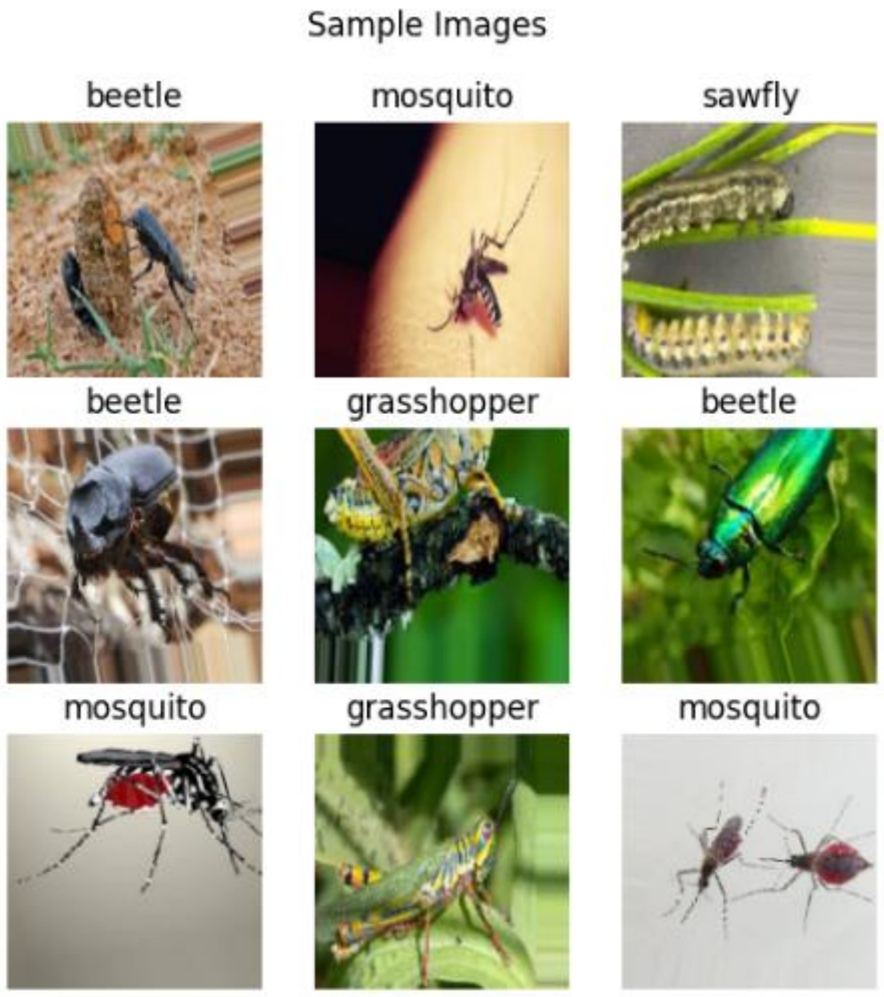


Figure 3: Sample Images

2.2. Pre-processing

Before training the model, some preprocessing steps were done on the dataset.

1. First, the pixel values of the images were changed from the original range of 0–255 to a new range between 0 and 1. This is called normalization and helps the model learn faster and more accurately.
2. Then, data augmentation was used to make the training data more varied. This included flipping, rotating, or slightly shifting the images. It helps the model perform better and prevents overfitting by giving it more examples to learn from.

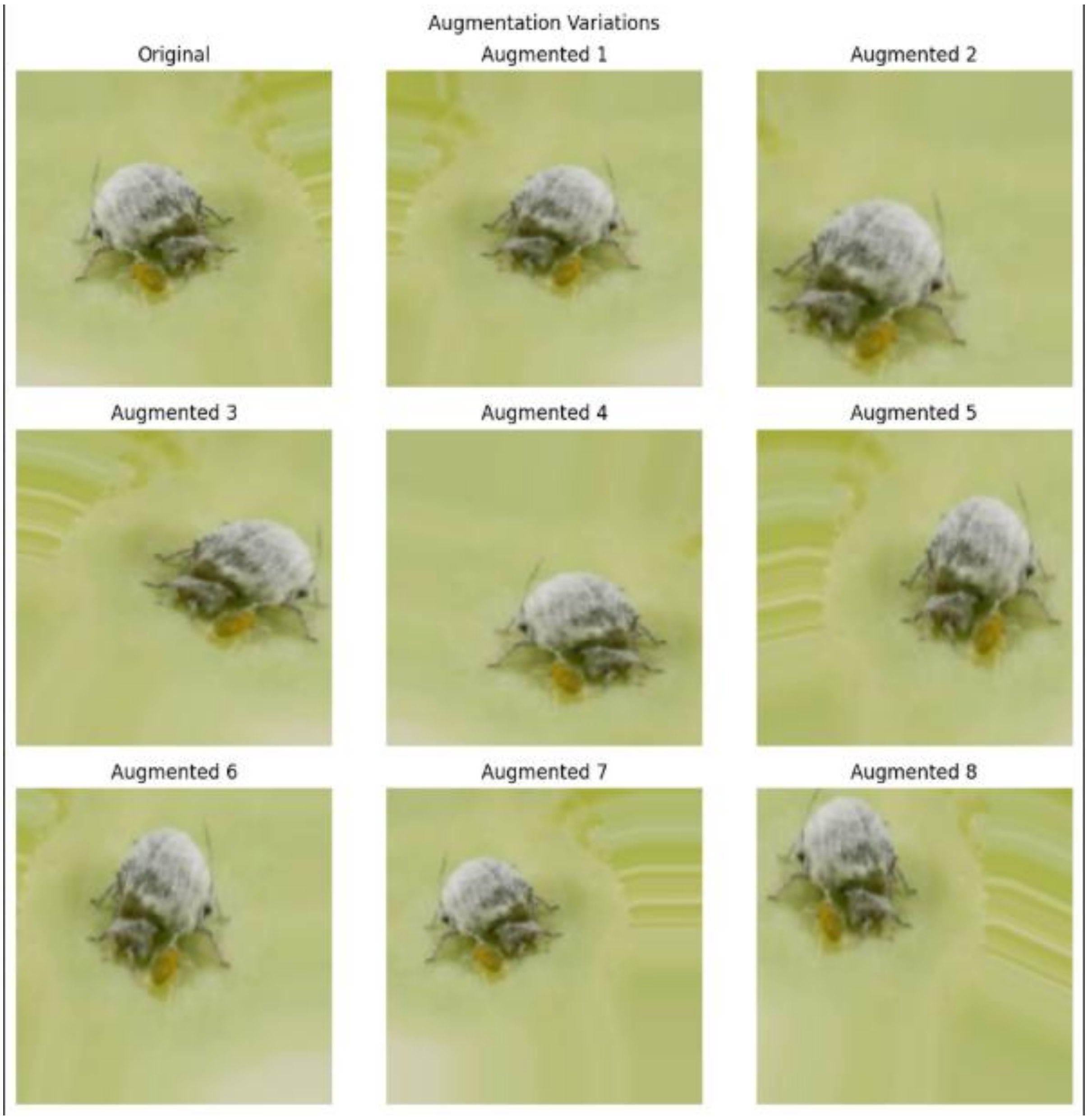


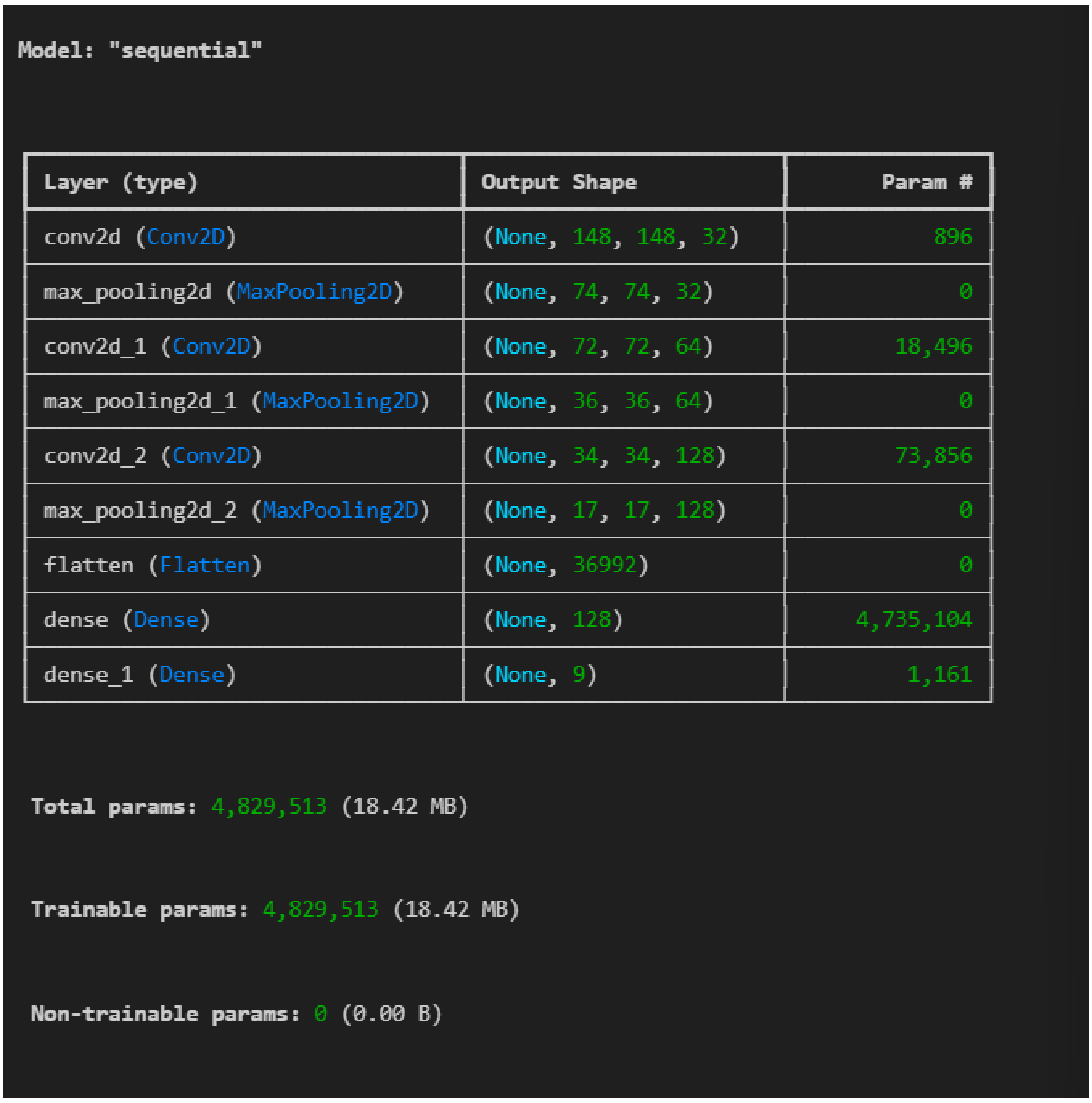
Figure 4: Data Augmentation

The dataset was organized into two main folders i.e. one for training and one for testing. The training folder was further split into two parts — 80% of the images were used for training and 20% for validation. This helped us check how well the model was learning during training, without touching the test data.

3. Methodology

For this project, two different Convolutional Neural Network (CNN) models were created and trained to classify fruit images. The first one is a basic model (called the baseline model), and the second one is a deeper model with more layers. Both models were built to take fruit images in RGB format with a size of 128 × 128 × 3 (width, height, and colour channels). Their main task was to correctly identify and classify each image into one of the ten fruit categories.

3.1. Baseline Model



```
Model: "sequential"

┌──────────────────────────────────┬───────────────────────┬──────────────┐
│ Layer (type)                     │ Output Shape          │      Param # │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ conv2d (Conv2D)                  │ (None, 148, 148, 32)  │          896 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 74, 74, 32)    │            0 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ conv2d_1 (Conv2D)                │ (None, 72, 72, 64)    │       18,496 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 36, 36, 64)    │            0 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ conv2d_2 (Conv2D)                │ (None, 34, 34, 128)   │       73,856 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 17, 17, 128)   │            0 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ flatten (Flatten)                │ (None, 36992)         │            0 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ dense (Dense)                    │ (None, 128)           │    4,735,104 │
├──────────────────────────────────┼───────────────────────┼──────────────┤
│ dense_1 (Dense)                  │ (None, 9)             │        1,161 │
└──────────────────────────────────┴───────────────────────┴──────────────┘

 Total params: 4,829,513 (18.42 MB)

 Trainable params: 4,829,513 (18.42 MB)

 Non-trainable params: 0 (0.00 B)
```

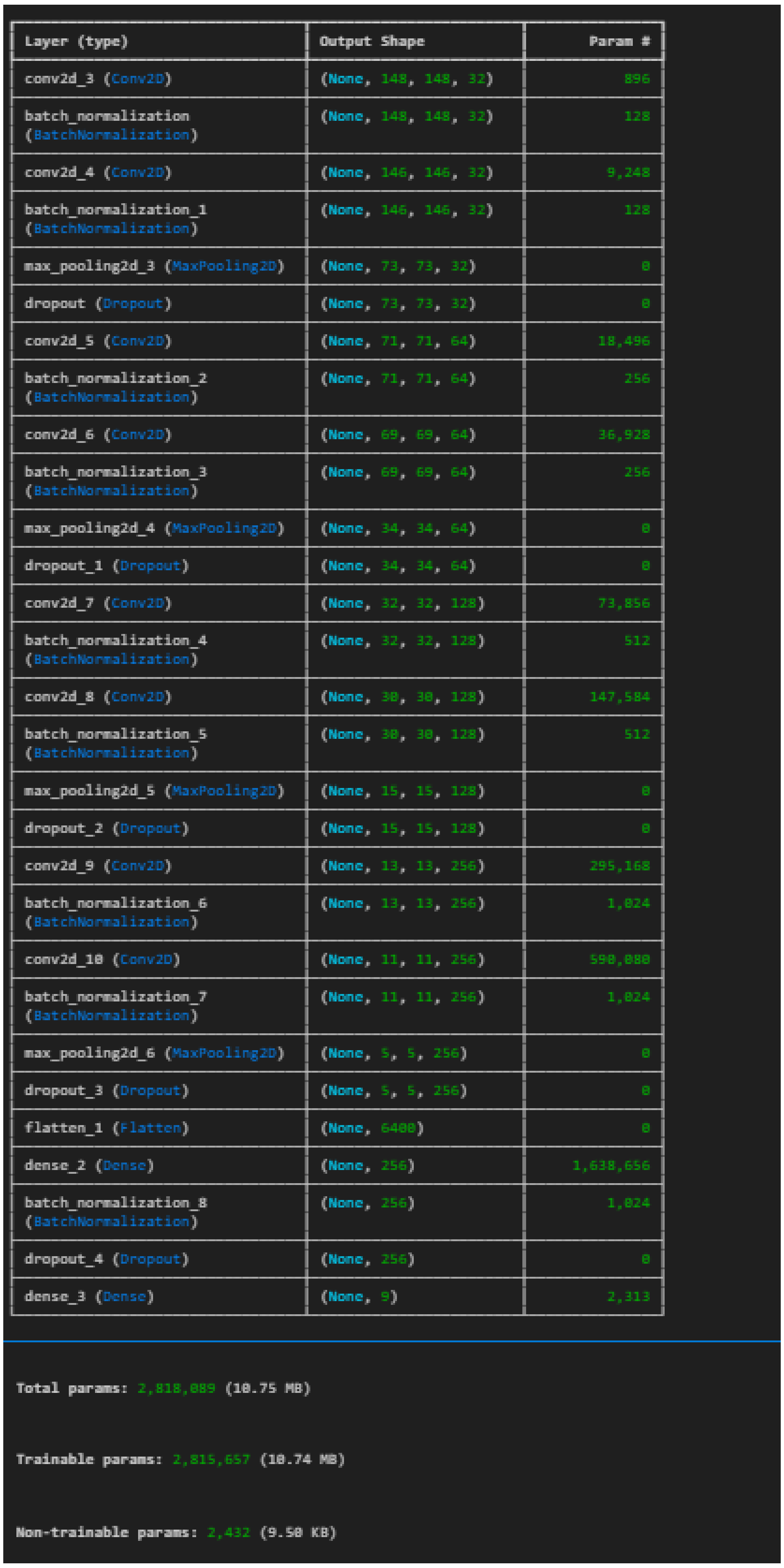
Figure 5:Model Summary of Baseline

1. The basic model had three convolutional layers:
   - The number of filters increased in each layer: first 32, then 64, and finally 128.
   - Each filter used a kernel size of 3 × 3.

- ReLU was used as the activation function after each convolution.
- A max pooling layer of size 2 × 2 was added after each convolution to reduce the size of the feature maps.
- The stride was 1 for the convolution filters and 2 for the pooling layers.

2. After the convolutional part, the model had two fully connected (dense) layers:
   - The first dense layer had 128 units with ReLU activation.
   - The second dense layer had 64 units, also with ReLU activation.
3. Finally, the output layer had:
   - 9 units (one for each pest class)
   - A softmax activation function to give the final prediction.
4. The total number of parameters in this model was 4,829,513.

3.2. Deeper Model



```
Layer (type)                     Output Shape                 Param #

conv2d_3 (Conv2D)                (None, 148, 148, 32)             896

batch_normalization              (None, 148, 148, 32)             128
(BatchNormalization)

conv2d_4 (Conv2D)                (None, 146, 146, 32)           9,248

batch_normalization_1            (None, 146, 146, 32)             128
(BatchNormalization)

max_pooling2d_3 (MaxPooling2D)   (None, 73, 73, 32)                 0

dropout (Dropout)                (None, 73, 73, 32)                 0

conv2d_5 (Conv2D)                (None, 71, 71, 64)            18,496

batch_normalization_2            (None, 71, 71, 64)               256
(BatchNormalization)

conv2d_6 (Conv2D)                (None, 69, 69, 64)            36,928

batch_normalization_3            (None, 69, 69, 64)               256
(BatchNormalization)

max_pooling2d_4 (MaxPooling2D)   (None, 34, 34, 64)                 0

dropout_1 (Dropout)              (None, 34, 34, 64)                 0

conv2d_7 (Conv2D)                (None, 32, 32, 128)           73,856

batch_normalization_4            (None, 32, 32, 128)              512
(BatchNormalization)

conv2d_8 (Conv2D)                (None, 30, 30, 128)          147,584

batch_normalization_5            (None, 30, 30, 128)              512
(BatchNormalization)

max_pooling2d_5 (MaxPooling2D)   (None, 15, 15, 128)                0

dropout_2 (Dropout)              (None, 15, 15, 128)                0

conv2d_9 (Conv2D)                (None, 13, 13, 256)          295,168

batch_normalization_6            (None, 13, 13, 256)            1,024
(BatchNormalization)

conv2d_10 (Conv2D)               (None, 11, 11, 256)          590,080

batch_normalization_7            (None, 11, 11, 256)            1,024
(BatchNormalization)

max_pooling2d_6 (MaxPooling2D)   (None, 5, 5, 256)                  0

dropout_3 (Dropout)              (None, 5, 5, 256)                  0

flatten_1 (Flatten)              (None, 6400)                       0

dense_2 (Dense)                  (None, 256)                1,638,656

batch_normalization_8            (None, 256)                    1,024
(BatchNormalization)

dropout_4 (Dropout)              (None, 256)                        0

dense_3 (Dense)                  (None, 9)                      2,313


Total params: 2,818,089 (10.75 MB)

Trainable params: 2,815,657 (10.74 MB)

Non-trainable params: 2,432 (9.50 KB)
```

Figure 6: Model Summary of Deeper

a. Enhanced CNN Architecture:

1. Convolutional Blocks:

- The model consists of four convolutional blocks, each with:
  - Two convolutional layers using a 3×3 kernel and ReLU activation.
  - Batch Normalization applied after each convolution to stabilize learning.
  - MaxPooling2D (2×2) for spatial downsampling.
  - Dropout layers added after pooling for regularization, with increasing rates:
    - 0.2 → 0.25 → 0.3 → 0.35.
- The number of filters increases progressively:
  - 32 → 64 → 128 → 256, improving the model's capacity to learn complex patterns.

2. Dense Layers (Fully Connected):

- After flattening, the model has:
  - One Dense layer with 256 units, ReLU activation, and L2 regularization.
  - Followed by Batch Normalization and a Dropout of 0.4 to reduce overfitting.

3. Output Layer:

- A final Dense layer with 9 units (matching the number of classes).
- Uses Softmax activation to generate class probabilities.

4. Total Parameters:

- Total parameters: 2,818,089
- Trainable parameters: 2,815,657
- Non-trainable parameters: 2,432

5. Regularization Techniques Used:

- Batch Normalization: Accelerates convergence and reduces internal covariate shift.
- Dropout: Gradually increasing dropout rates from 0.2 to 0.4 help reduce overfitting in deeper layers.

3.3. Loss Function

Sparse Categorical Crossentropy was chosen as the loss function because it works well for multi-class classification problems, especially when the labels are given as integers instead of one-hot encoded vectors.

3.4. Optimizer

The Adam optimizer was chosen because it adjusts the learning rate during training, helping the model converge faster and more steadily compared to traditional gradient descent methods.

3.5. Hyperparameters

A learning rate of 0.001, a batch size of 32, and 20 training epochs were used. These values are commonly used for CNN training because they help keep the training fast while still giving good model performance.

4. Experiments and Results

To evaluate the performance of both models, experiments were conducted by training them on the prepared pest dataset. The training and validation performance were monitored over 20 epochs for each model.

4.1. Model Architecture

4.1.1. Baseline Model:

- Training accuracy reached around 78% by epoch 23.
- Validation accuracy stayed low, around 40-50%.
- Baseline train loss is decreasing .
- Validation loss stayed same as it is.
- This shows the model was overfitting — it learned the training data well but did not perform well on new data.

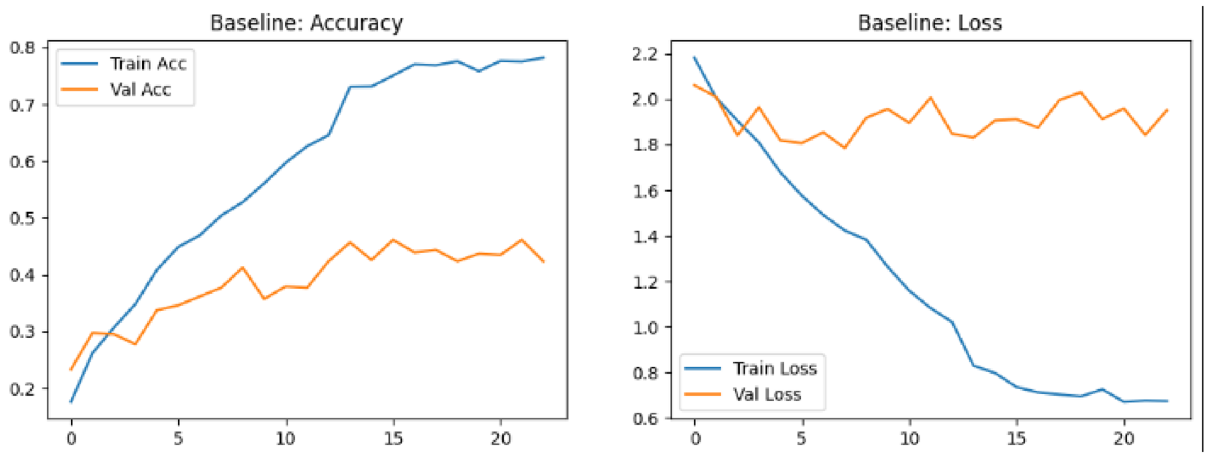
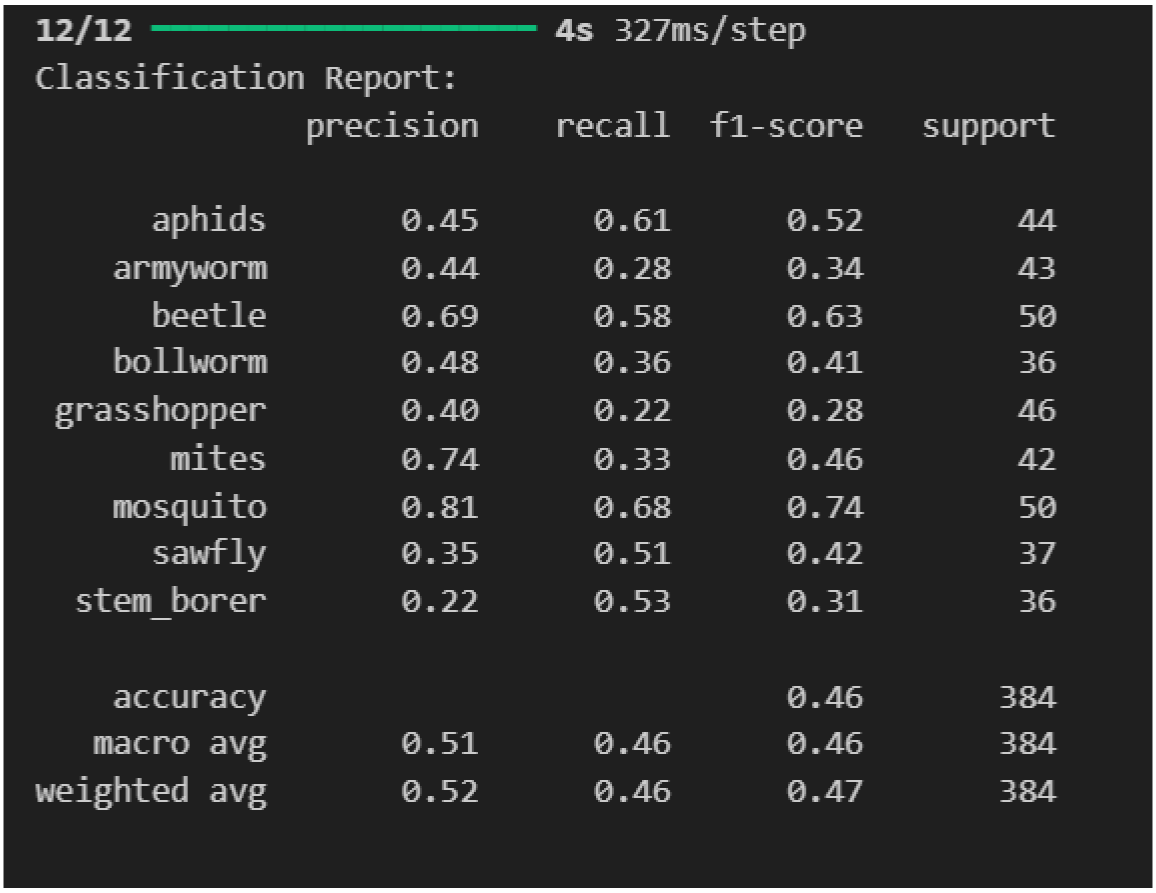


Figure 7: Accuracy Comparison of Base Model



```
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 327ms/step
Classification Report:
              precision    recall  f1-score   support

      aphids       0.45      0.61      0.52        44
    armyworm       0.44      0.28      0.34        43
      beetle       0.69      0.58      0.63        50
    bollworm       0.48      0.36      0.41        36
 grasshopper       0.40      0.22      0.28        46
       mites       0.74      0.33      0.46        42
    mosquito       0.81      0.68      0.74        50
      sawfly       0.35      0.51      0.42        37
  stem_borer       0.22      0.53      0.31        36

    accuracy                           0.46       384
   macro avg       0.51      0.46      0.46       384
weighted avg       0.52      0.46      0.47       384
```

Figure 8: Classification Report of Base Model

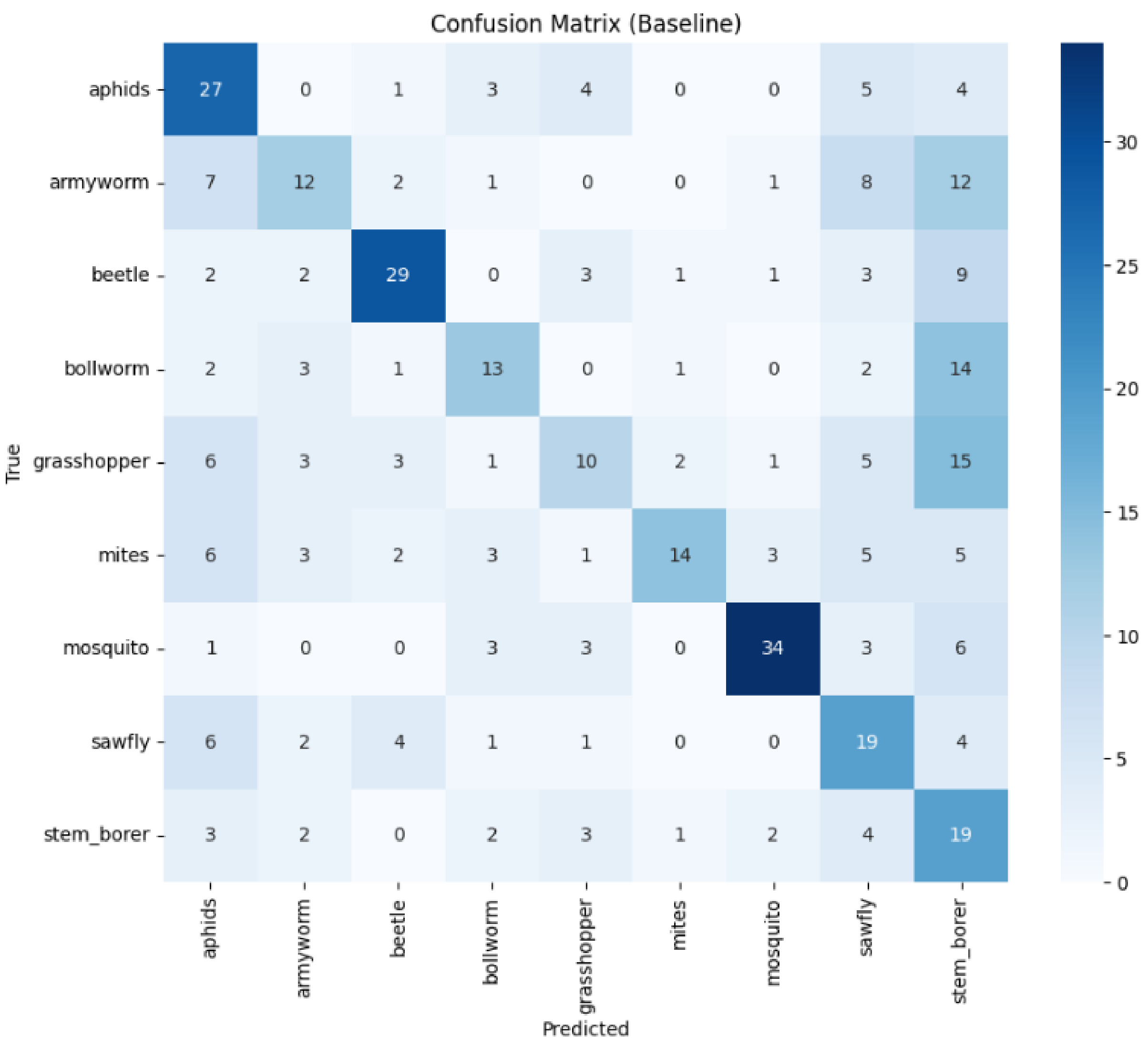
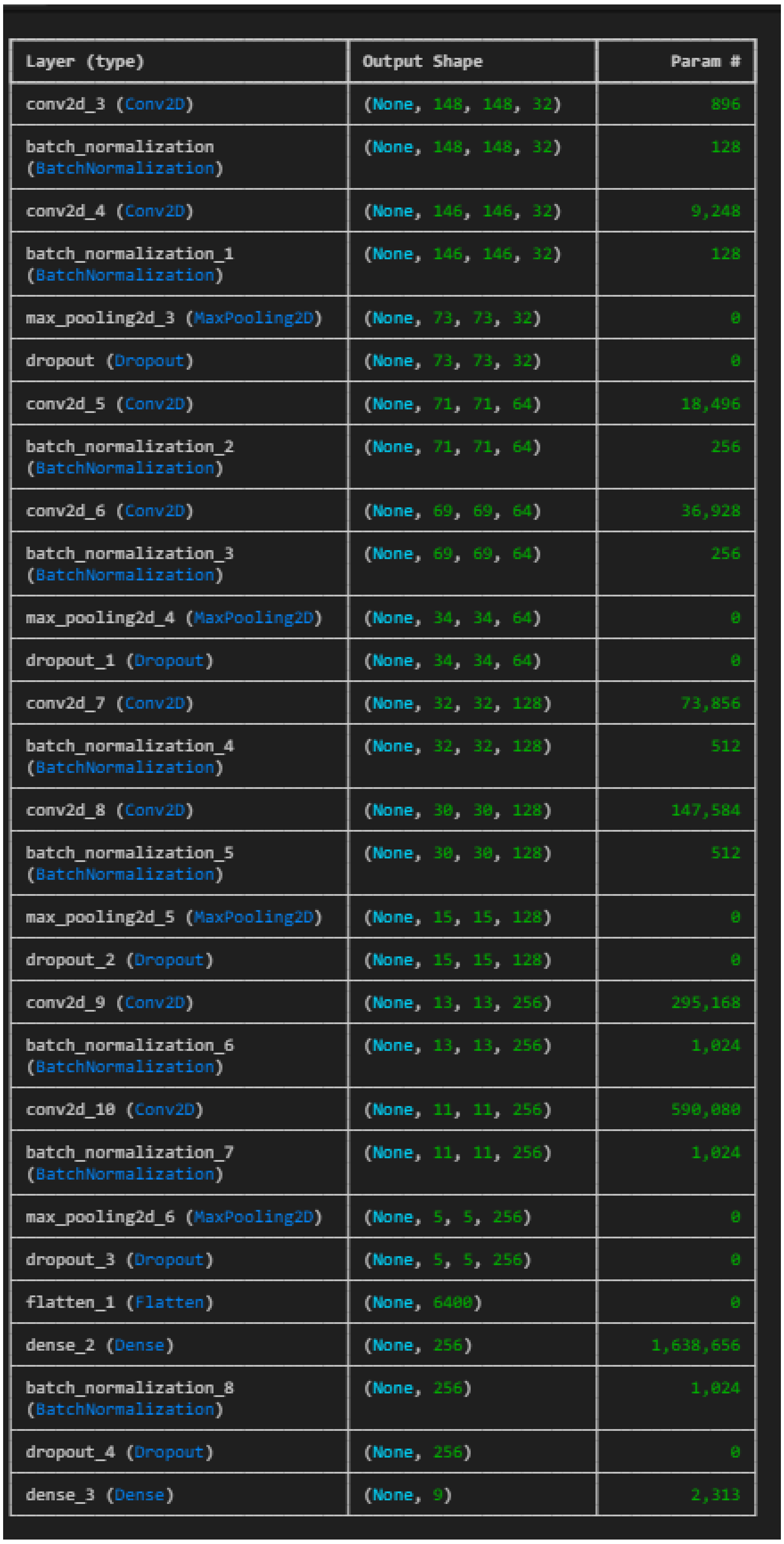


Figure 9: Confusion Matrix

4.1.2 Deeper Model:



```
┌────────────────────────────────────┬──────────────────────────┬───────────────┐
│ Layer (type)                       │ Output Shape             │       Param # │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)                  │ (None, 148, 148, 32)     │           896 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization                │ (None, 148, 148, 32)     │           128 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_4 (Conv2D)                  │ (None, 146, 146, 32)     │         9,248 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_1              │ (None, 146, 146, 32)     │           128 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)     │ (None, 73, 73, 32)       │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ dropout (Dropout)                  │ (None, 73, 73, 32)       │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_5 (Conv2D)                  │ (None, 71, 71, 64)       │        18,496 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_2              │ (None, 71, 71, 64)       │           256 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_6 (Conv2D)                  │ (None, 69, 69, 64)       │        36,928 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_3              │ (None, 69, 69, 64)       │           256 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d_4 (MaxPooling2D)     │ (None, 34, 34, 64)       │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ dropout_1 (Dropout)                │ (None, 34, 34, 64)       │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_7 (Conv2D)                  │ (None, 32, 32, 128)      │        73,856 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_4              │ (None, 32, 32, 128)      │           512 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_8 (Conv2D)                  │ (None, 30, 30, 128)      │       147,584 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_5              │ (None, 30, 30, 128)      │           512 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d_5 (MaxPooling2D)     │ (None, 15, 15, 128)      │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ dropout_2 (Dropout)                │ (None, 15, 15, 128)      │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_9 (Conv2D)                  │ (None, 13, 13, 256)      │       295,168 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_6              │ (None, 13, 13, 256)      │         1,024 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_10 (Conv2D)                 │ (None, 11, 11, 256)      │       590,080 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_7              │ (None, 11, 11, 256)      │         1,024 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d_6 (MaxPooling2D)     │ (None, 5, 5, 256)        │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ dropout_3 (Dropout)                │ (None, 5, 5, 256)        │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ flatten_1 (Flatten)                │ (None, 6400)             │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ dense_2 (Dense)                    │ (None, 256)              │     1,638,656 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ batch_normalization_8              │ (None, 256)              │         1,024 │
│ (BatchNormalization)               │                          │               │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ dropout_4 (Dropout)                │ (None, 256)              │             0 │
├────────────────────────────────────┼──────────────────────────┼───────────────┤
│ dense_3 (Dense)                    │ (None, 9)                │         2,313 │
└────────────────────────────────────┴──────────────────────────┴───────────────┘
```

Figure 10: Deeper Model

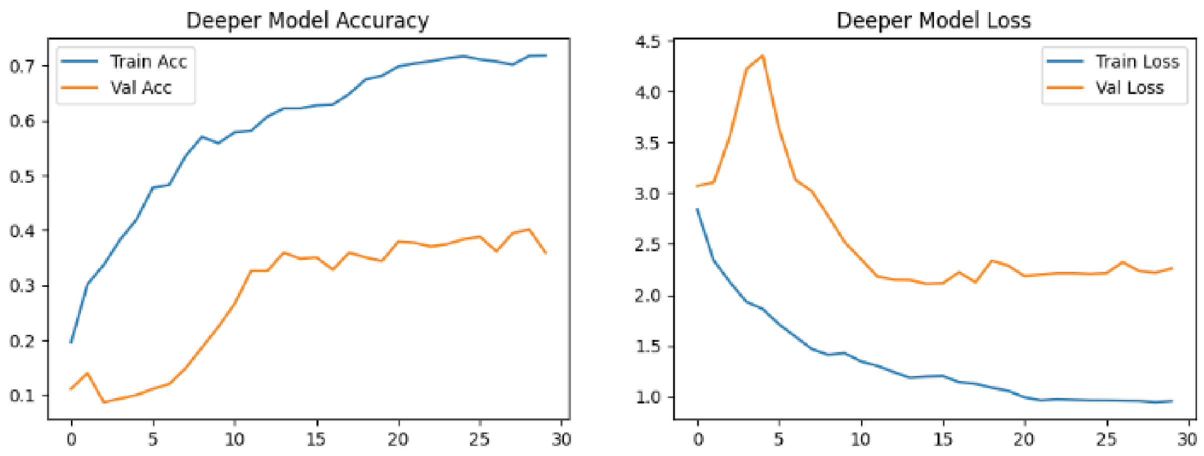
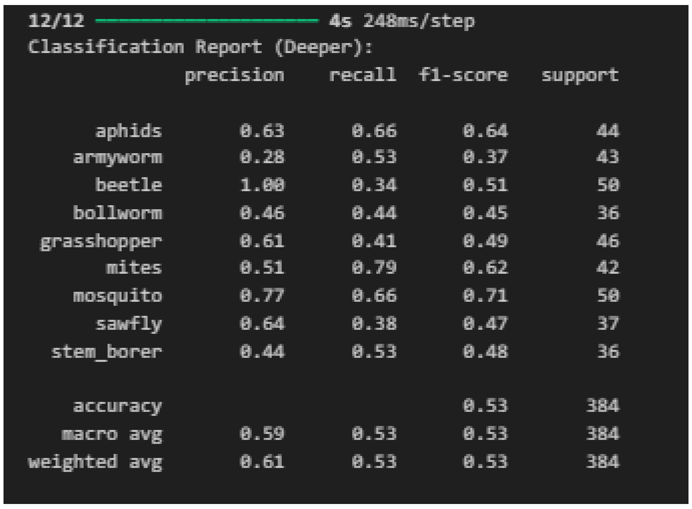


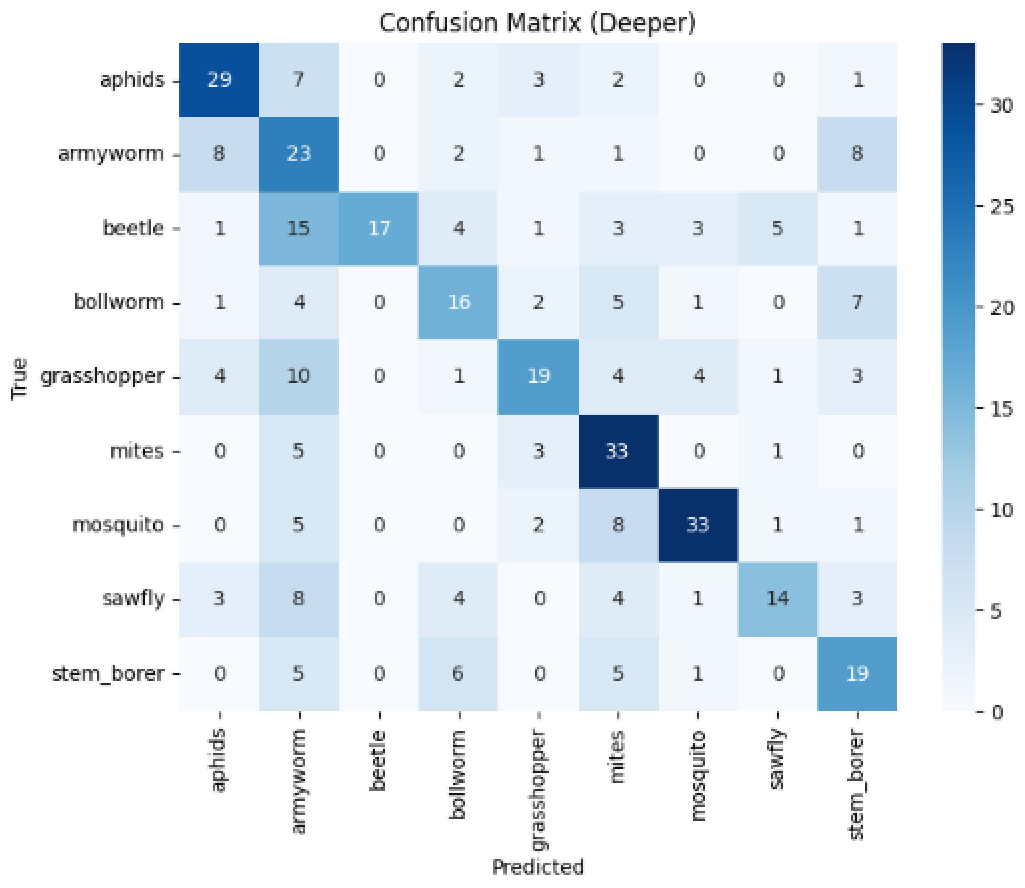
Figure 11: Deeper Model Accuracy and Loss Graph

4.1.3 Classification Report (Deeper):

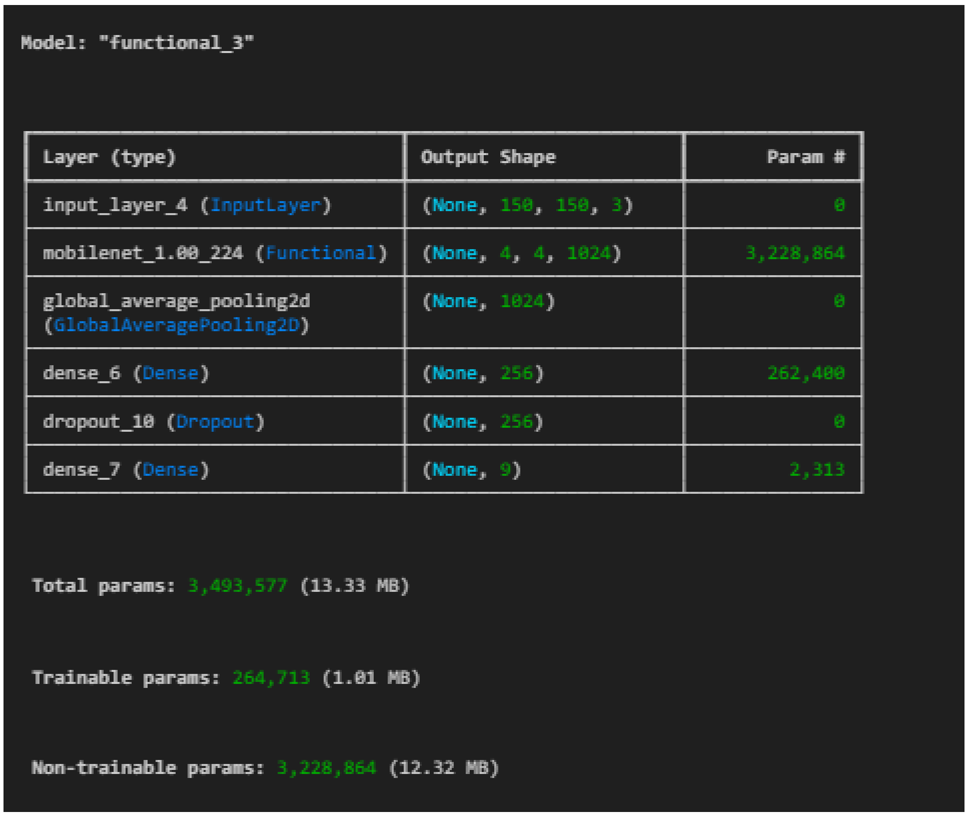


```
12/12 ━━━━━━━━━━━━━━━━━━━━ 4s 248ms/step
Classification Report (Deeper):
              precision    recall  f1-score   support

      aphids       0.63      0.66      0.64        44
    armyworm       0.28      0.53      0.37        43
      beetle       1.00      0.34      0.51        50
    bollworm       0.46      0.44      0.45        36
 grasshopper       0.61      0.41      0.49        46
       mites       0.51      0.79      0.62        42
    mosquito       0.77      0.66      0.71        50
      sawfly       0.64      0.38      0.47        37
  stem_borer       0.44      0.53      0.48        36

    accuracy                           0.53       384
   macro avg       0.59      0.53      0.53       384
weighted avg       0.61      0.53      0.53       384
```

4.1.4 Confusion Matrix of Deeper Model



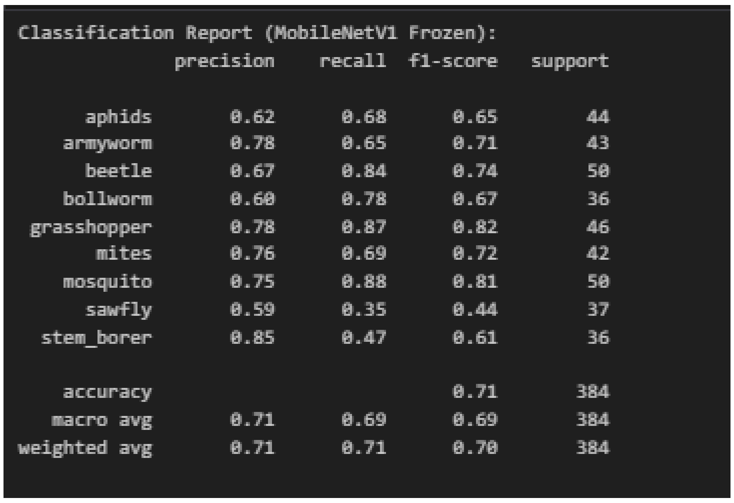
4.2 Mobilenet frozen



```
Model: "functional_3"

┌────────────────────────────────────┬──────────────────────────────┬──────────────┐
│ Layer (type)                       │ Output Shape                 │      Param # │
├────────────────────────────────────┼──────────────────────────────┼──────────────┤
│ input_layer_4 (InputLayer)         │ (None, 150, 150, 3)          │            0 │
├────────────────────────────────────┼──────────────────────────────┼──────────────┤
│ mobilenet_1.00_224 (Functional)    │ (None, 4, 4, 1024)           │    3,228,864 │
├────────────────────────────────────┼──────────────────────────────┼──────────────┤
│ global_average_pooling2d           │ (None, 1024)                 │            0 │
│ (GlobalAveragePooling2D)           │                              │              │
├────────────────────────────────────┼──────────────────────────────┼──────────────┤
│ dense_6 (Dense)                    │ (None, 256)                  │      262,400 │
├────────────────────────────────────┼──────────────────────────────┼──────────────┤
│ dropout_10 (Dropout)               │ (None, 256)                  │            0 │
├────────────────────────────────────┼──────────────────────────────┼──────────────┤
│ dense_7 (Dense)                    │ (None, 9)                    │        2,313 │
└────────────────────────────────────┴──────────────────────────────┴──────────────┘

 Total params: 3,493,577 (13.33 MB)

 Trainable params: 264,713 (1.01 MB)

 Non-trainable params: 3,228,864 (12.32 MB)
```

Figure 12: MobileNetV1 Frozen

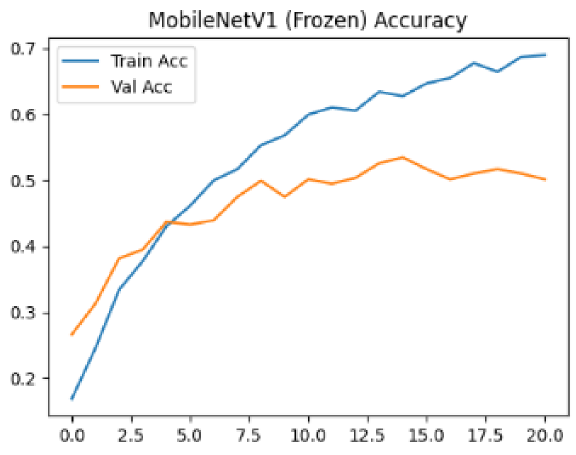
4.2.1 Accuracy of MobileNetV1 Frozen:

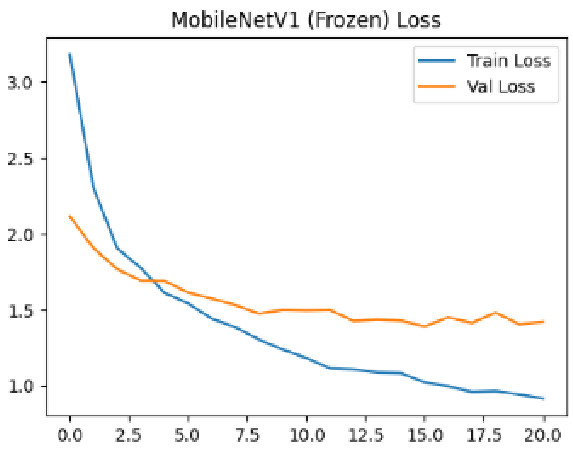


```
Classification Report (MobileNetV1 Frozen):
              precision    recall  f1-score   support

      aphids       0.62      0.68      0.65        44
    armyworm       0.78      0.65      0.71        43
      beetle       0.67      0.84      0.74        50
    bollworm       0.60      0.78      0.67        36
 grasshopper       0.78      0.87      0.82        46
       mites       0.76      0.69      0.72        42
    mosquito       0.75      0.88      0.81        50
      sawfly       0.59      0.35      0.44        37
  stem_borer       0.85      0.47      0.61        36

    accuracy                           0.71       384
   macro avg       0.71      0.69      0.69       384
weighted avg       0.71      0.71      0.70       384
```

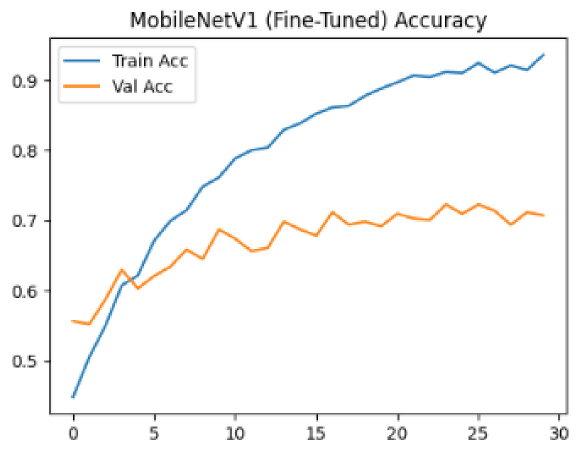
Figure 13: Classification report of MobileNet Frozen

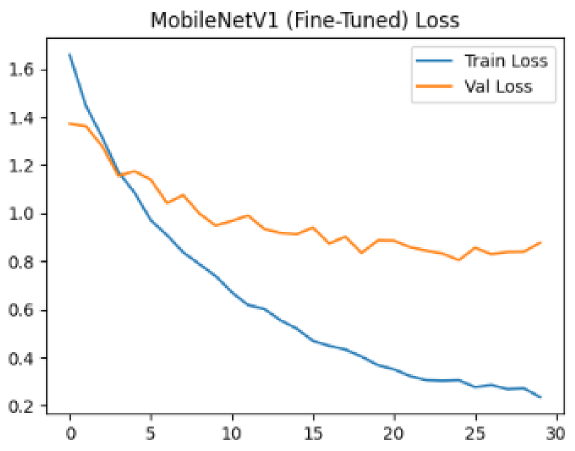
4.2.2 Graph of MobileNetV1



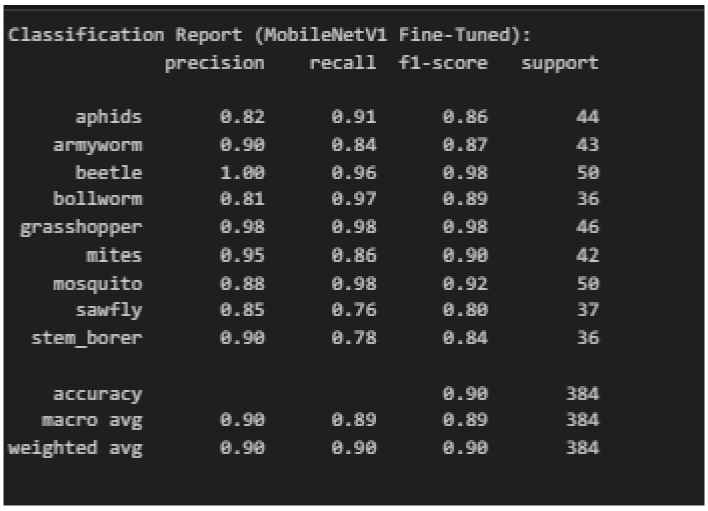


4.2.3 Graph of MobileNetV1 (Fine-Tuned)



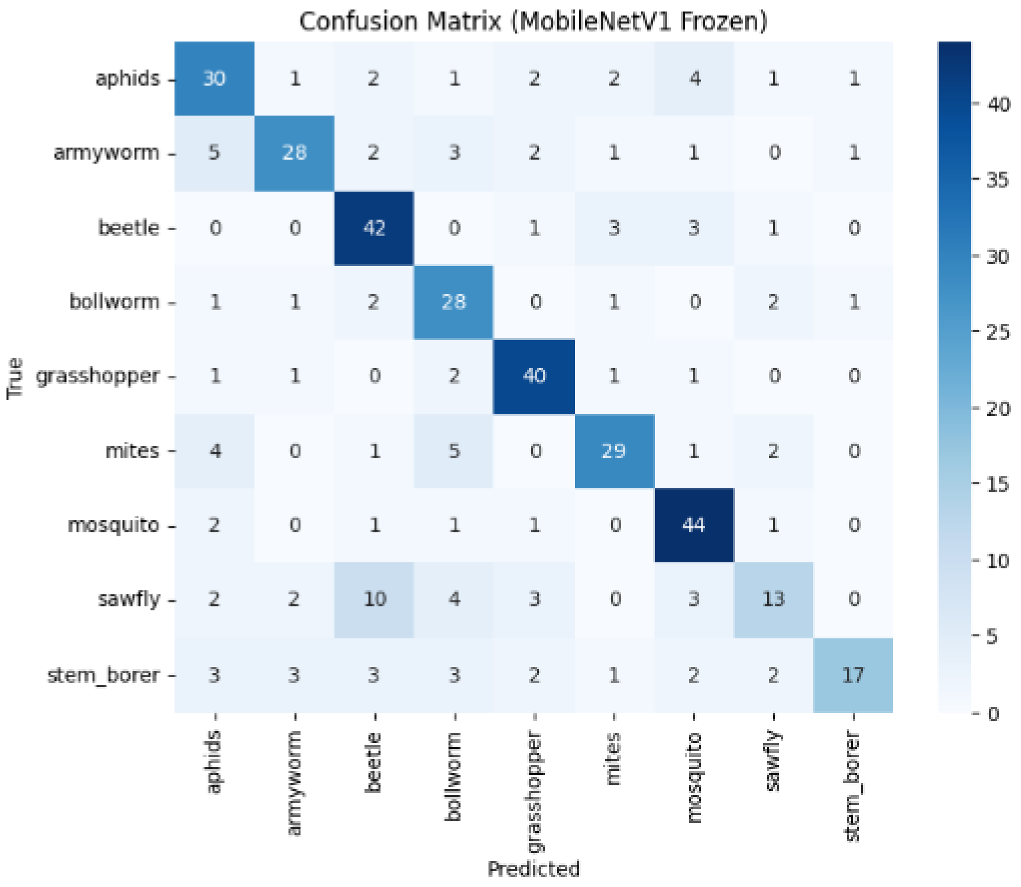


4.2.4 Accuracy of MobileNetV1 Fine-Tuned:



```
Classification Report (MobileNetV1 Fine-Tuned):
              precision    recall  f1-score   support

      aphids       0.82      0.91      0.86        44
    armyworm       0.90      0.84      0.87        43
      beetle       1.00      0.96      0.98        50
    bollworm       0.81      0.97      0.89        36
 grasshopper       0.98      0.98      0.98        46
       mites       0.95      0.86      0.90        42
    mosquito       0.88      0.98      0.92        50
      sawfly       0.85      0.76      0.80        37
  stem_borer       0.90      0.78      0.84        36

    accuracy                           0.90       384
   macro avg       0.90      0.89      0.89       384
weighted avg       0.90      0.90      0.90       384
```

4.2.4 Confusion Matrix (MobileNetV1 Frozen)



True \ Predicted	aphids	armyworm	beetle	bollworm	grasshopper	mites	mosquito	sawfly	stem_borer
aphids	30	1	2	1	2	2	4	1	1
armyworm	5	28	2	3	2	1	1	0	1
beetle	0	0	42	0	1	3	3	1	0
bollworm	1	1	2	28	0	1	0	2	1
grasshopper	1	1	0	2	40	1	1	0	0
mites	4	0	1	5	0	29	1	2	0
mosquito	2	0	1	1	1	0	44	1	0
sawfly	2	2	10	4	3	0	3	13	0
stem_borer	3	3	3	3	2	1	2	2	17

Figure 14: Confusion Matrix (Frozen)

4.3 MobileNetV1 (Fine Tuned)

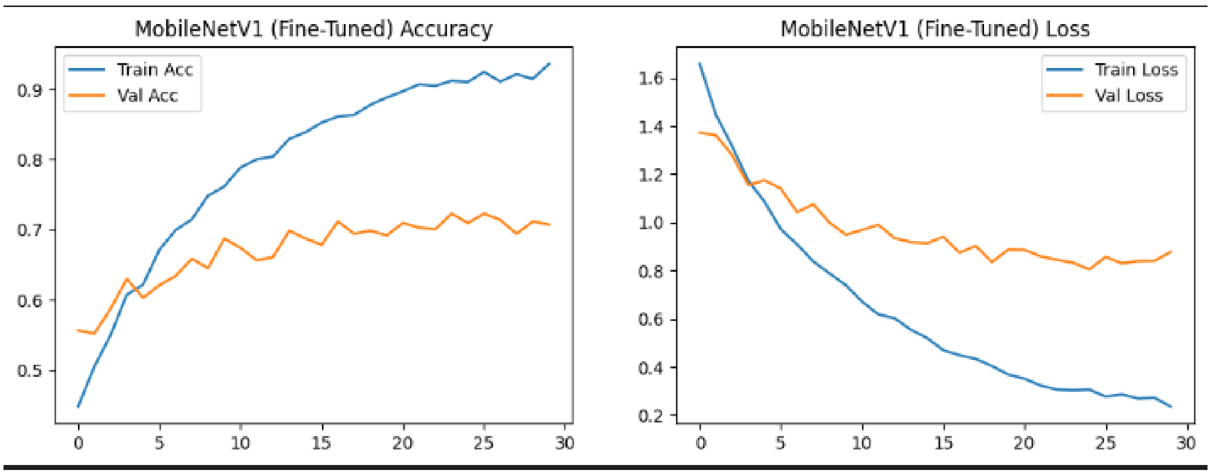


Figure 15: MobileNetV1 Finetuned graph

4.3 Confusion Matrix

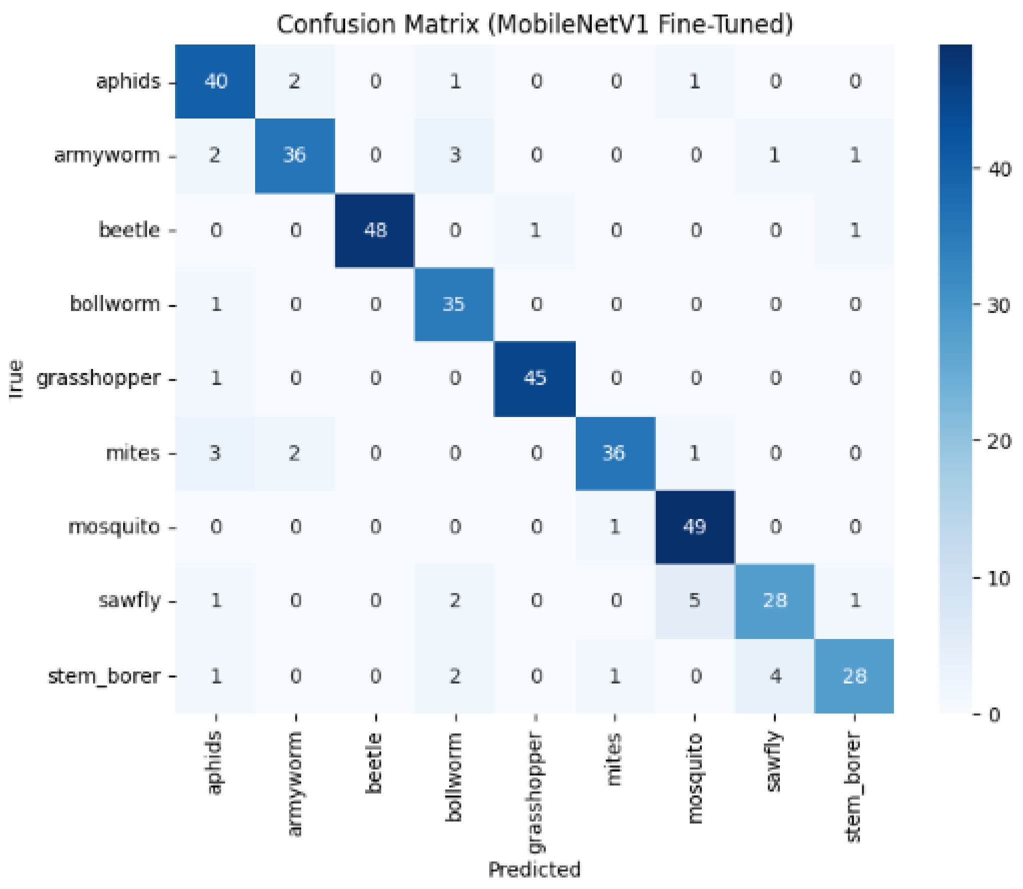
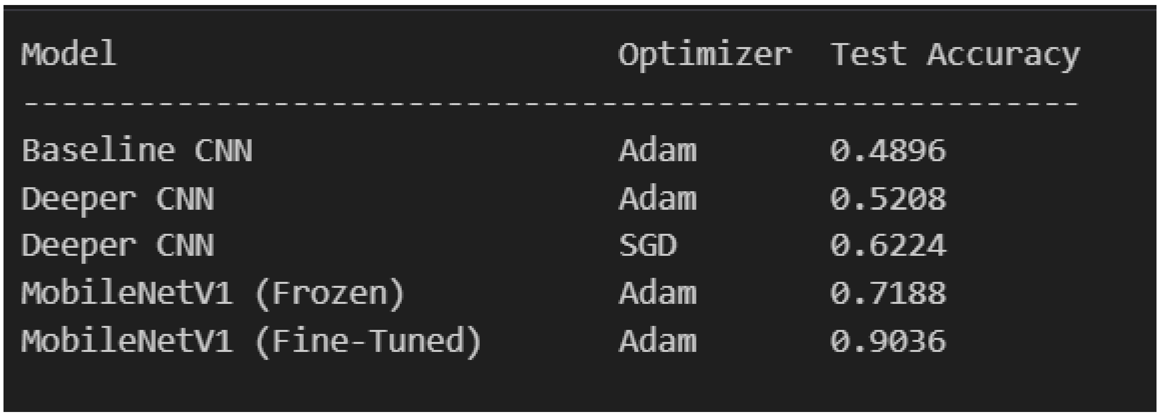


Figure 16: Confusion matrix

4.3 Overall Accuracy Report



```
Model                          Optimizer  Test Accuracy
-------------------------------------------------------
Baseline CNN                   Adam       0.4896
Deeper CNN                     Adam       0.5208
Deeper CNN                     SGD        0.6224
MobileNetV1 (Frozen)           Adam       0.7188
MobileNetV1 (Fine-Tuned)       Adam       0.9036
```

Figure 17: Overall Accuracy

This shows the MobileNetV1 (Fine Tuned) has the best accuracy .