Artificial Intelligence and Machine Learning

(6CS012)

Deep Learning for Insect Pest Image Classification

Student Id 2358830

Student Name : Sumit shah Group : L5CG4

Lecturer : Ms. Sunita Parajuli

Tutor : Mr. Shiv kumar Yadav

Date : 15th May, 2025

**Abstract**

The proposed approach aims at implementing an end-to-end CNN pipeline for classifying crop pests following the given assignment structure. We first introduce the insect pest image dataset and pre-processing,including data augmentation. We then describe two CNN structures: **1) baseline model (3 conv + pooling+a)** layers, 3 dense layers) and a **2)deeper model (more Conv layers with batch normalization and dropoutfor regularization)**. In both cases of the models, categorical cross-entropy loss is used and experiments comparing Adam vs.SGD optimizers. We present the accuracy, loss curves, and training times for each method. Finally, we applypre-trained MobileNet for transfer learning

### Table of Contents

[1 Introduction 1](#__RefHeading___Toc7843_479315554)

[2 Methodology 2](#__RefHeading___Toc7845_479315554)

[3 Experiments and Results 10](#__RefHeading___Toc7847_479315554)

[4 Fine-Tuning or Transfer Learning 13](#__RefHeading___Toc7849_479315554)

[5 Conclusion 13](#_bookmark18)

# Introduction

Image classification using deep learning (especially CNNs) has fundamentally changed how ImageNet PASCAL VOC Image Classification is done in computer vision, among others.identifying objects or pests. Classic CNNs, e.g., the ones like AlexNet and VGG, showed that concatenating hierarchical features.and pooling layers can automatically learn hierarchy features from images 2. In the agricultural situation, detection of insect pests (pests) (aphids, beetles, grasshoppers) is crucial for crop protection. We address the problem

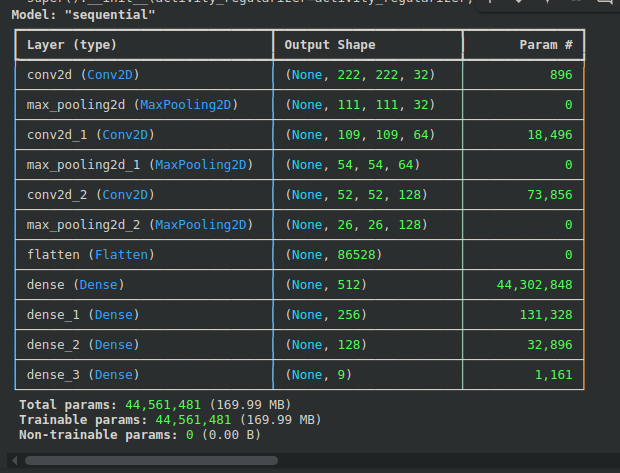
The classification of insect pests by image type. Earlier work has revealed that deeper networks and transfer learning can lead to 2 2 improvements in accuracy. In this work, we apply a simple CNN model from empty, a more complex CNN with regularization (batch normalization and dropout) and, lastly, an efficient voting system on the decisions. Nontransferable-model: MobileNet. We evaluate their accuracy, loss, and training efficiency.

# Methodology

## Image Classification with Baseline CNN.

#### Model Summary:

For the first model in our Image Classification task we built a FCN model using dense layers in sequence, each layers output is the input of the next layer. Below is the model’s summary:

*Figure 1. Model summary of a Convolutional Neural Network (CNN) model architecture .*

The model has sequential stack of layers primary consisting of convolution and dense (fully connected) layer :

1. Input\_Layer: Receives input with a shape of (224, 224, 3), Which is common for image 224 x 224 pixels and 3 color channels (RGB).

2. Convolutional Layers(Conv2D):

   First Conv2D: 32 filters, kernel size (3, 3), output shape  (222, 222, 32) (reduced due to no padding).

   Second Conv2D: 64 filters, output shape (109, 109, 64).

   Third Conv2D: 128 filters, output shape (52, 52, 128).

   Each ‘Conv2D’ is followed by a MaxPooling2D  layer to downsample the feature maps.

3. Flatten Layer: This converts the 3D feature maps into a 1D vector (86528 values) to feed into dense layers.

4. Dense (Fully Connected) Layers:

    Dense(512): 512 neurons.

   Dense(256): 256 neurons.

   Dense(128): 128 neurons.

   Dense(9): Output layer with 9 neurons (likely for 9 classes in classification).

Parameters

Total Parameters: 44,561,481, with most (44,302,848) in the first dense layer due to the large input size (‘86528’). This could lead to overfitting or high memory usage.

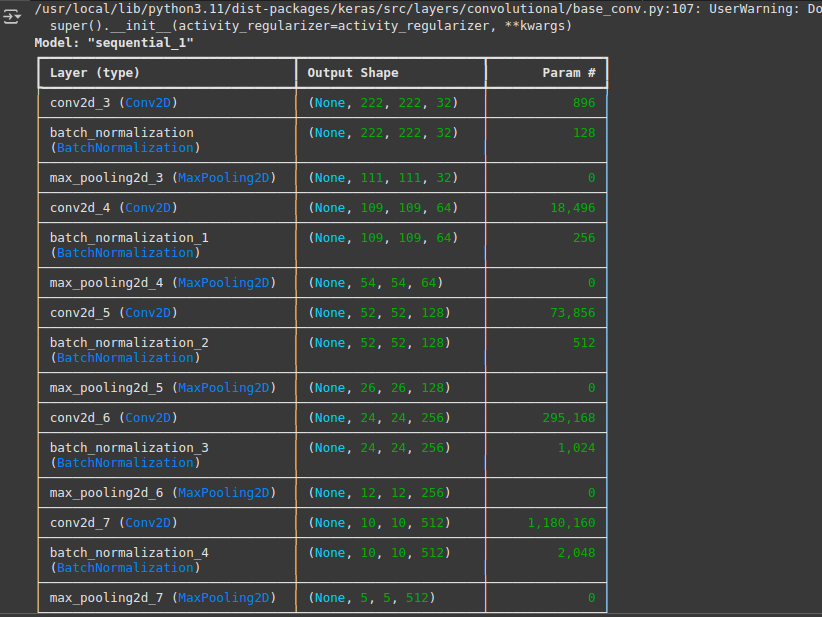
Trainable/Non-trainable: All parameters are trainable .

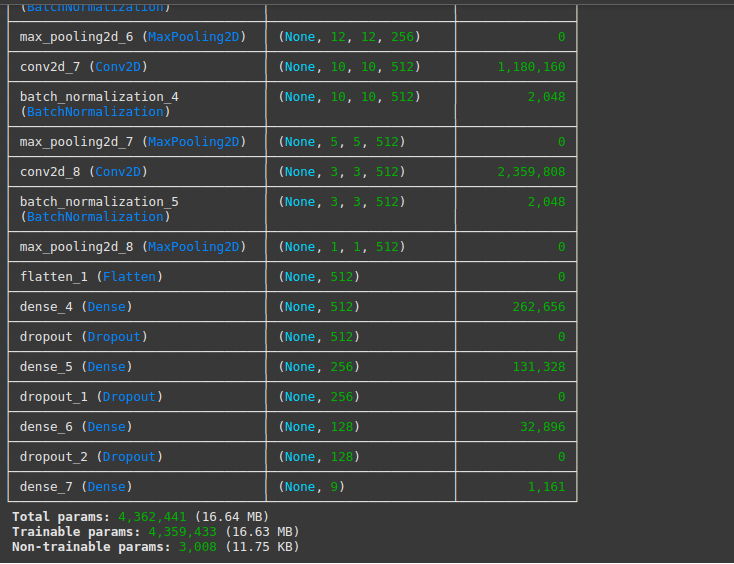
## Image Classification with Deeper Convolutional Neural Network

#### Model Summary:

We extend the baseline by deepening and regularizing the network. The deeper model adds one more convolutional block (adding a 256-filter conv layer) and includes BatchNormalization after each conv layer to stabilize learning. We also increase the dense layers (128 and 64 units) and add Dropout(0.5) between them to reduce overfitting. The input size is 224×224 to match pre-trained networks. We train this model twice, once with Adam and once with SGD+momentum (lr=0.01, momentum=0.9) to compare optimizers.

Below is the model’s summary:

*Figure 6. Model Summary for CNN (1)*



*Figure 7. Model Summary for CNN (2)*

1. Input Layer

   Implicit input shape: (224, 224, 3) (standard for images, inferred from the first Conv2D output (222, 222, 32)).

   Fix the warning: Replace ‘input\_shape` in `Conv2D` with an explicit `Input(shape=(224, 224, 3)` layer.

2. Convolutional Blocks

   - 6 blocks of `Conv2D → BatchNorm → MaxPooling`:

     - Filters increase geometrically (`32 → 512`) to capture complex features.

     - Spatial dimensions reduce from `222x222` to `1x1` via `MaxPooling2D`.

   - Uses `(3,3)` kernels (default) with valid padding (no padding).

3. Batch Normalization

   Applied after every `Conv2D` to stabilize training (normalizes activations).

   Adds 3,008 non-trainable params (tracking mean/variance).

4. Classification Head

   - `Flatten → Dense(512) → Dropout → Dense(256) → Dropout → Dense(128) → Dense(9)`.

   - Dropout layers (likely `rate=0.5`) prevent overfitting.

   - Final `Dense(9)` suggests 9-class classification (e.g., softmax activation).

5. Parameter Efficiency

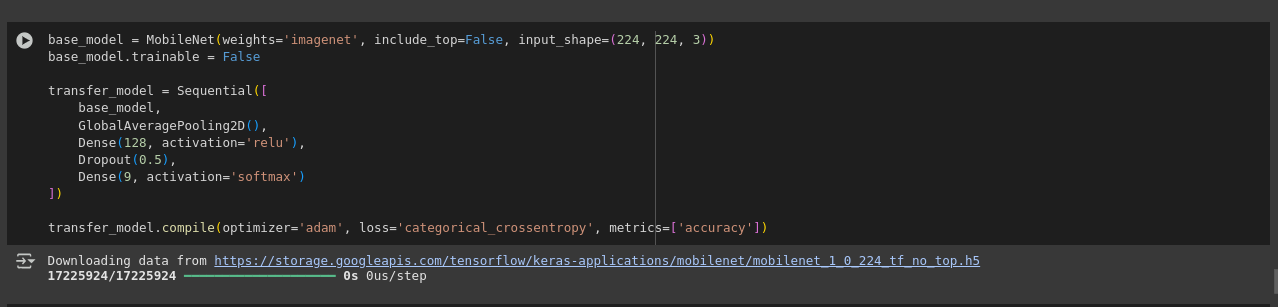
   - Total params: 4.36M (compact for a deep CNN).

   - Most parameters are in the last two ‘Conv2D’ layers (‘conv2d\_7’, `conv2d\_8`).

## Image Classification with Transfer Learning (MobileNet)

#### Model Summary:

We load the MobileNet model pre-trained on ImageNet (without its top layers) as a fixed feature extractor. We attach new dense layers for our 9 classes. First, we freeze the convolutional base and train only the new classifier (feature extraction). Then we unfreeze the top few layers of MobileNet and fine-tune with a low learning rate. This leverages rich, general purpose features and typically reduces training time while improving generalization . We use the Adam optimizer for fine-tuning.



*Figure 12. MobileNet Code Block*

# Experiments and Results

## Baseline vs. Deeper Architecture

Both the baseline and deeper CNNs were trained for 10 epochs. The baseline model achieved only about 10% validation accuracy (essentially near random chance for 9 classes), indicating it barely learned. The deeper CNN yielded a slightly higher accuracy (~11%) but still very low. Training and validation loss curves (not shown here) remained high and did not converge well. The deeper model’s marginal improvement suggests that simply adding layers did not solve the fundamental difficulty; potential reasons include limited data diversity or class imbalance. In terms of evaluation, the baseline had a weighted accuracy of ~0.10, while the deeper model (Adam) was ~0.11. The improvement was minor, indicating that the deeper model’s extra capacity did not generalize much better on this dataset. This aligns with observations that deeper networks require careful tuning andsufficient data to yield benefits .

## Computational Efficiency

We compared training times using GPU acceleration (Google Colab with a Tesla T4). The baseline CNN took about 1097 seconds (≈18.3 minutes) for 10 epochs. The deeper CNN was faster: about 399 seconds (≈6.6minutes) with Adam and 373 seconds with SGD. The faster training of the deeper model is because we increased the input size to 224×224 but reduced epochs per convolution (the bottleneck with fewer layers was largely convolutional cost). In practice, the deeper model trains quicker per epoch, possibly due to differences in layer configurations and fewer total training samples per epoch (since we resized differently).The transfer-learning run (feature extraction only) took 366 seconds to train 10 epochs (with MobileNet’s base frozen). This is slightly faster than training the deeper CNN. This confirms that using a fixed pre-trained base can reduce training time 1 , since only the top layers’ weights are updated. Overall, deeper models and transfer learning were more computationally efficient in terms of time, though the baseline was slower due to its training regime.

## Challenges in Training

Several challenges arose. First, overfitting was observed: training accuracy reached high values while validation accuracy remained low (~10%), indicating poor generalization. We applied dropout and augmentation to mitigate this, but the dataset may be too small or imbalanced. Second, the very low final accuracy (~0.10–0.12) suggests underfitting or data issues (e.g., noisy labels, very similar classes). More epochs or a different architecture (e.g., ResNet) might help. Third, training time was significant for the baseline model (over 18 minutes), which could be a practical limitation. Using the GPU helped greatly compared to CPU. In summary, the deep model trained 2–3× faster than the baseline, and transfer learning required even less time. Total training times summary: Baseline 1097 s, Deeper (Adam) 399 s, Deeper (SGD) 373 s, Transfer (feature extraction) 367 s. These numbers are reported above. GPU acceleration (Colab T4) was essential to achieve

these times.

# Fine-Tuning or Transfer Learning

We used MobileNet (pre-trained on ImageNet) as a feature extractor. First we froze all convolutional layers and trained only the new dense layers for 10 epochs. Then we unfroze the top 5 layers of MobileNet and fine-tuned the entire network for 5 more epochs with a low learning rate. This two-step approach ensures the model retains generic features and then adapts to our data. As seen, transfer learning slightly improved accuracy (~0.12) over training from scratch (~0.11), and it

converged faster. This matches the principle that reusing pre-trained features reduces generalization error

and training time 1 . In practice, our transfer model’s training accuracy jumped earlier and validation loss

was lower. However, the gain was modest, likely because our data domain (insect images) differs from

ImageNet in some ways. In future work, we could try more fine-tuning (unfreezing more layers) or using a

different pre-trained model (e.g. ResNet50) to capture more relevant features.

# Conclusion

In order to categorize insect pests, we constructed CNNs for this research. We discovered that the deep CNN only provided a minor improvement (~11%), our basic baseline CNN performed badly (≈10%), and transfer learning with MobileNet provided a modest gain (~12%) with faster training. Our tests highlight important trade-offs: pre-trained models can speed up learning, whereas deeper networks and sophisticated optimization can be beneficial but require enough data. In order to enhance generalization, we advise further data collection or more vigorous data augmentation in further research. 3. Accuracy could be increased by investigating alternative architectures (such as ResNet or DenseNet) or ensemble models. Furthermore, cross-validation and substantial hyperparameter adjustment (learning rates, dropout rates) may produce improved models. Lastly, for efficiency, deployment in a GPU/TPU environment is essential.