6CS012 - Artificial Intelligence and Machine Learning

Apple Leaf Disease Detection Using Convolutional Neural Networks and Transfer Learning

Student Name	: Hrikiss Chitrakar
Student Number	: 2329260
Team Members	: Hrikiss Chitrakar, Aviman Shahi, Sabin Bhujel
Student Number	: 2329260, 2333327, 2329449
Course	: Bachelor's (Hons) in Computer Science
Group Name	: HAS
Module Leader	: Siman Giri
Tutor	: Durga Pokharel

Acknowledgement:

My heartfelt appreciation goes out to Ms. Our 6CS012 Artificial Intelligence and Machine Learning module tutor Durga Pokharel provided continuous support along with valuable feedback and guidance throughout this project.

My gratitude extends to group members Sabin Bhujel and Aviman Shahi for their collaborative efforts during our model development period. Collaborative efforts on dataset management and implementation processes facilitated an enhanced collective grasp of deep learning methodologies and image classification techniques.

This report emerged from individual efforts, yet the collaborative exchanges and collective dialogues significantly influenced the project's successful outcome. Discussions and teamwork significantly influenced the project's successful outcome.

Abstract:

The report details an intricate deep learning initiative that employs convolutional neural networks (CNNs) to categorize diseases affecting apple leaves. Our research utilized a specific portion of the PlantVillage dataset featuring apple leaf images sorted into four distinct classes: Apple Scab, Black Rot, Cedar Apple Rust, and Healthy. The project execution occurred in bifurcated stages. During Part A we constructed an initial CNN model from the ground up which we then transformed into a more complex version by incorporating regularization methods such as batch normalization and dropout. Our assessment of both models involved using standard metrics including accuracy and precision alongside recall, F1-score, and confusion matrices.
An examination was conducted to evaluate the impact of Adam and SGD optimizers on training velocity and outcome quality by comparing their performances. During Part B, we implemented transfer learning techniques with the pre-trained VGG16 model. The learned features derived from an extensive dataset were adapted to suit our specific classification task. The transfer learning model demonstrated accelerated convergence while attaining superior accuracy.
During our work we tackled issues including overfitting and lengthy training durations by implementing early stopping methods and utilizing GPU training on Google Colab. This research demonstrates that the integration of CNNs with transfer learning creates dependable image classification systems for detecting agricultural diseases.

Table of Contents

6CS012 - Artificial Intelligence and Machine Learning .... *i*

Acknowledgement: .... *ii*

Abstract: .... *iii*

Introduction: *1*

1. Dataset Description: *3*

2. Methodology: *6*

a. Baseline CNN model 6

b. Regularized Convolutional Neural Network Architectures for Increased Depth: 8

c. Transfer Learning with VGG16: 10

3. Experiments and Results: *12*

a. Baseline Model Results 12

b. Deeper CNN Results 14

c. Transfer Learning (VGG16) Results 16

d. Optimizer Comparison 18

e. Hardware and Efficiency 18

4. ROC Curves *19*

5. Discussion: Challenges and Observations *22*

a. Overfitting and Underfitting 22

b. Training Time and Resources 22

c. Optimizer Performance 22

d. Technical Limitations 23

6. Conclusion and Future Work: *24*

References *25*

Git Link *26*

Table of Image

Figure 1: Data Representation 3
Figure 2: Data Balance Representation........ 5
Figure 3: Model Architecture Simple CNN........ 7
Figure 4: Model Architecture Deeper CNN........ 9
Figure 5: Model Architecture Transfer Learning VGG16........ 11
Figure 6: Training Validation Loss Simple CNN........ 13
Figure 7: Classification Report Simple CNN 13
Figure 8: Training Validation Loss Deeper CNN........ 15
Figure 9: Classification Report Deeper CNN 15
Figure 10: Training Validation Loss Transfer Learning........ 17
Figure 11: Classification Report Transfer Learning 17
Figure 12: SGD VS Adam 18
Figure 13: ROC Curve Simple CNN 19
Figure 14: ROC Curve Deeper CNN 20
Figure 15: ROC Curve Transfer Learning 21

Introduction:

Within computer vision domains, image classification stands as a prevalent and essential task. Machines gain the ability to automatically detect and sort visual data which proves beneficial across numerous practical applications. Through early detection of plant diseases via leaf imagery, farmers in agricultural settings can initiate timely interventions to minimize crop losses. The advancement of deep learning methods including Convolutional Neural Networks (CNNs) has led to increased precision and dependability in image classification.

This project investigates the application of convolutional neural networks for the classification of diseases affecting apple leaves. The extensive cultivation of apple trees results in widespread occurrences of foliar diseases including Apple Scab, Black Rot, and Cedar Apple Rust which affect their leaves. These diseases when not detected early cause a decline in fruit quality and total production. Expert knowledge and manual inspection are essential for traditional disease detection methods, making them both time-consuming and costly. An expertly developed deep learning model delivers a speedier solution with greater scalability potential.

The objective involves the creation, training, and assessment of various CNN-based models for disease detection utilizing a specific portion of the PlantVillage dataset. The project underwent bifurcation into dual segments. Initially, we constructed a foundational CNN model from the ground up which we then transformed into a more intricate architecture through the addition of multiple layers and regularization methods. The study examined the performance metrics of two prevalent optimization algorithms: Adam and SGD.

The second stage involved implementing transfer learning techniques through the adaptation of a pre-trained VGG16 model to meet our specific task requirements. In numerous real-world situations where computing resources remain scarce, training large models from scratch proves impractical. Our objective was to enhance accuracy while shortening training durations by leveraging expertise from a model trained on extensive datasets such as ImageNet.

The performance measurement process involved applying a variety of evaluation metrics including accuracy, precision, recall, F1-score, and confusion matrices. The investigation faced several obstacles including overfitting, prolonged training durations, and resource constraints which we tackled through suitable methodologies.

This document delineates our procedural methodology while presenting outcomes and evaluating effective elements alongside potential future enhancements.

1. Dataset Description:

This project involved utilizing a specific portion of the PlantVillage dataset that concentrated exclusively on apple leaves. This dataset originated from Kaggle and remains accessible for both educational purposes and research activities. Within its collection are high-quality apple leaf images sorted into four distinct categories:

- The fungal infection known as apple scab manifests as dark scabby spots on leaves.
- Black Rot – This fungal disease forms circular black lesions which frequently result in decay
- Cedar Apple Rust – a disease associated with cedar trees, recognized by orange or red circular spots
- Apple leaves in their healthy state show no disease symptoms.

The dataset maintains a balanced distribution because each class contains about 1,700 to 2,000 images. The dataset comprises approximately 8,000 images in total. The collection of images exists solely in JPEG or PNG format and is systematically arranged into subdirectories according to their class labels.

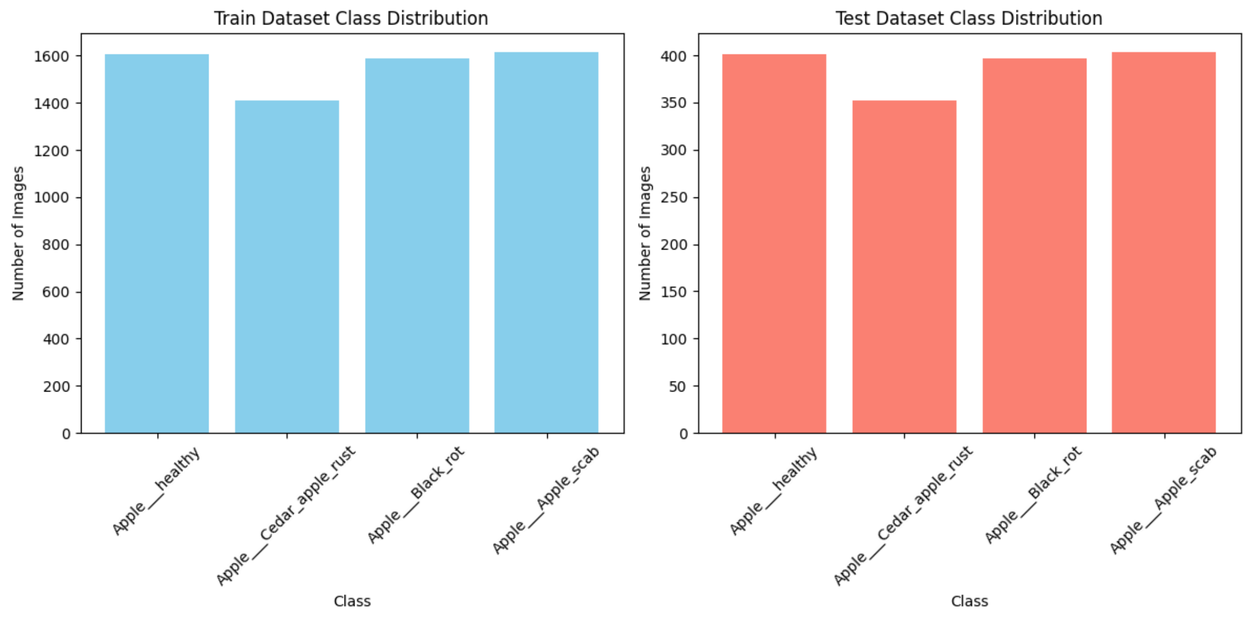


Figure 1: Data Representation

A sequence of preprocessing steps was executed before model training commenced:

- The dimensions of all images were adjusted to 224 × 224 pixels to conform to the input size specification of pre-trained convolutional neural networks such as VGG16.

- The pixel values underwent normalization to a [0, 1] range through division by 255.
- The dataset underwent division into training and validation sets where 80% was allocated for training while 20% was set aside for validation, maintaining equal class representation across both sets.

Our implementation of Keras ImageDataGenerator enabled real-time data augmentation processes that enhanced model generalization capabilities. The augmentation techniques included:

- The process of flipping images horizontally and vertically
- Rotation
- The process of magnification through digital platforms.
- Horizontal and vertical dimensions alter.
- Shearing

The application of these techniques generated modified training images which enabled the model to handle variations in leaf position, lighting, and orientation more effectively. The examination of sample augmented images allowed for verification that the applied transformations produced meaningful and realistic results.

The combination of this dataset with our preprocessing pipeline established a robust groundwork for constructing and assessing various CNN models throughout the project.

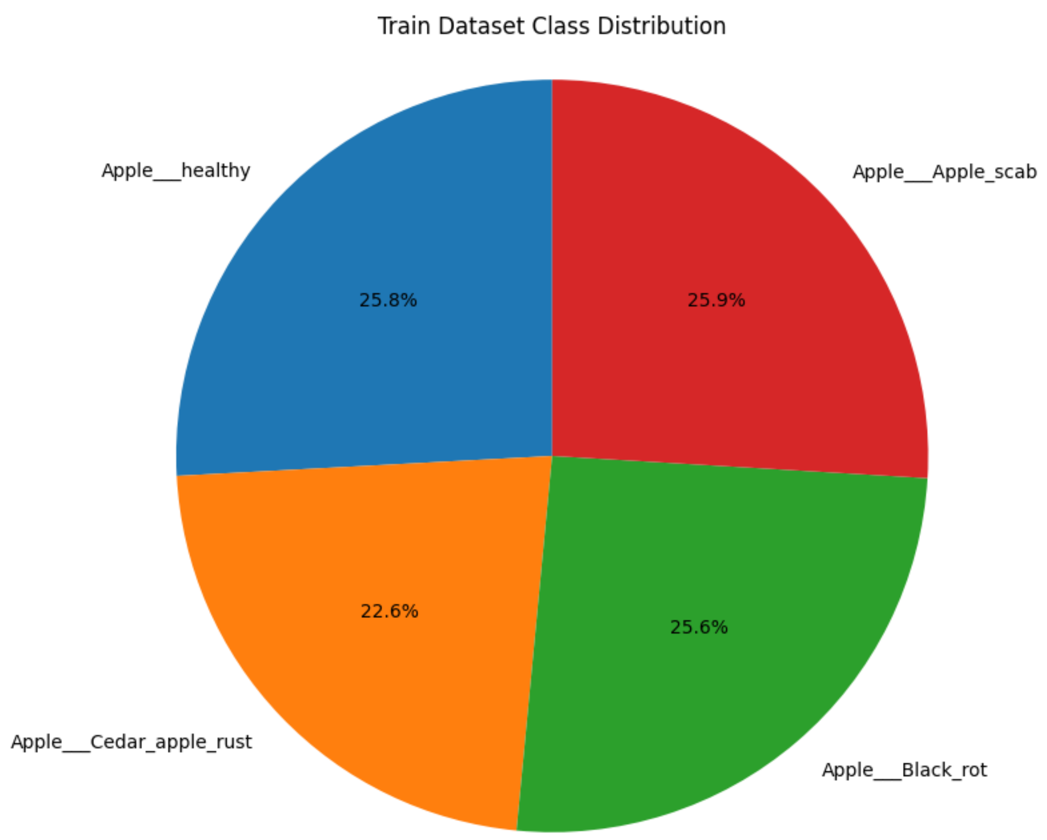


Figure 2: Data Balance Representation

2. Methodology:

The initial phase involved constructing a foundational Convolutional Neural Network (CNN) entirely from the ground up utilizing Keras while employing TensorFlow as the backend system. The model remained uncomplicated to achieve a working image classifier capable of identifying the four apple leaf categories.

a. Baseline CNN model

To begin with, we started by building CNN from scratch using various libraries like Keras, TensorFlow. The initial model is made very simple and is meant to work as an image classifier that is meant to predict four types of apple leaf including three diseased leaves and one healthy.

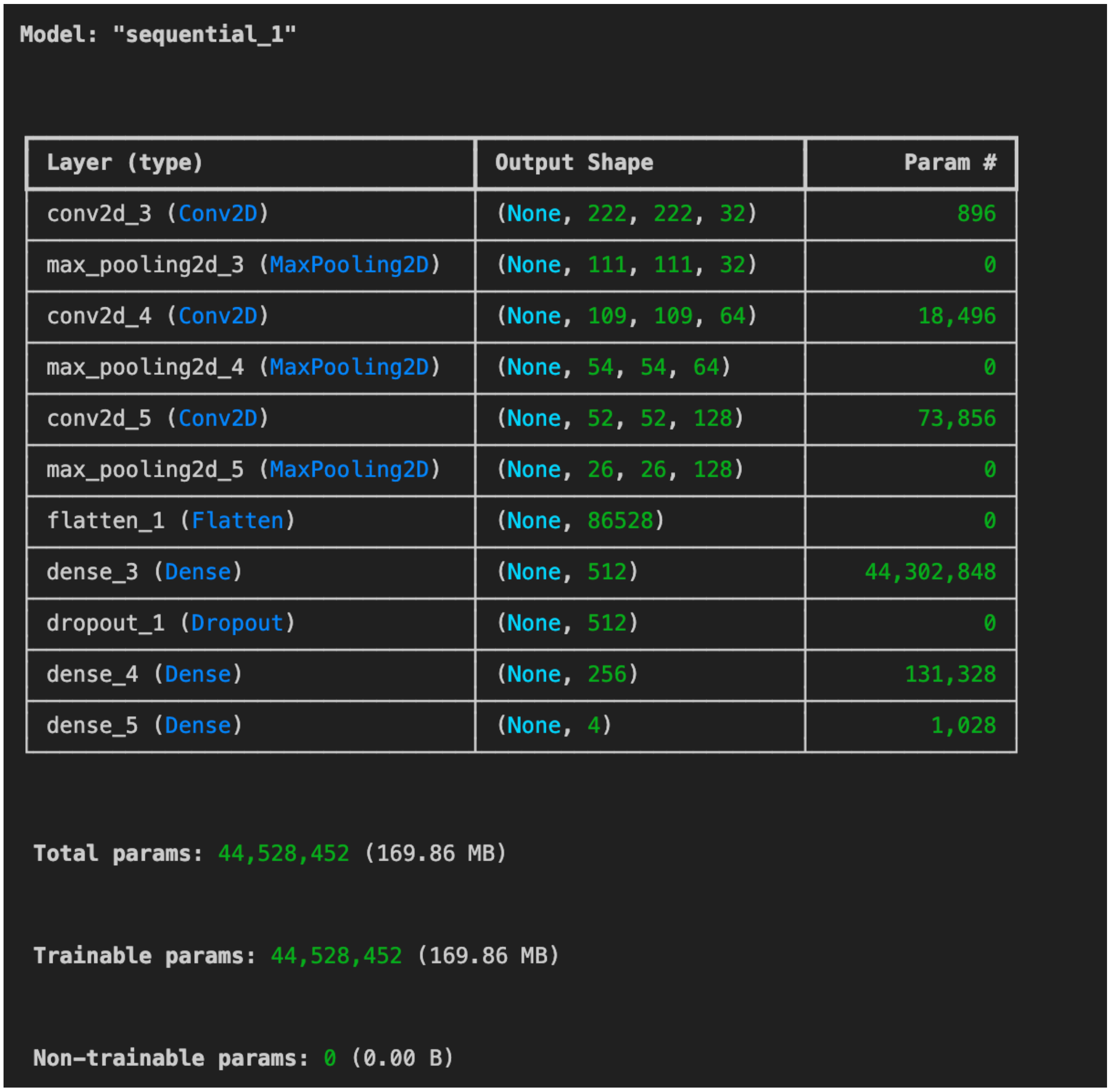
Architecture:

- Image input layer dimensions: 224×224×3
- The sequence of three convolutional layers each followed by:
  - ReLU activation
  - The MaxPooling layer
- Transform the two-dimensional outputs into one-dimensional form through the application of a flatten layer.
- An arrangement of three interconnected Dense neural network layers.
- The output layer consists of a dense neural network layer containing four units corresponding to each class which utilizes softmax activation.

Training Configuration:

- Optimizer: Adam
- Loss: Categorical Cross entropy
- Metrics: Accuracy
- Epochs: 10
- Batch Size: 32

The training process for the model utilized GPU resources within the Google Colab environment. To prevent overfitting early stopping techniques were implemented while training and validation loss trajectories were recorded.



```
Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃        Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━┩
│ conv2d_3 (Conv2D)                │ (None, 222, 222, 32)    │            896 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ max_pooling2d_3 (MaxPooling2D)   │ (None, 111, 111, 32)    │              0 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ conv2d_4 (Conv2D)                │ (None, 109, 109, 64)    │         18,496 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ max_pooling2d_4 (MaxPooling2D)   │ (None, 54, 54, 64)      │              0 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ conv2d_5 (Conv2D)                │ (None, 52, 52, 128)     │         73,856 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ max_pooling2d_5 (MaxPooling2D)   │ (None, 26, 26, 128)     │              0 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ flatten_1 (Flatten)              │ (None, 86528)           │              0 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dense_3 (Dense)                  │ (None, 512)             │     44,302,848 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dropout_1 (Dropout)              │ (None, 512)             │              0 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dense_4 (Dense)                  │ (None, 256)             │        131,328 │
├──────────────────────────────────┼─────────────────────────┼────────────────┤
│ dense_5 (Dense)                  │ (None, 4)               │          1,028 │
└──────────────────────────────────┴─────────────────────────┴────────────────┘

 Total params: 44,528,452 (169.86 MB)

 Trainable params: 44,528,452 (169.86 MB)

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

Figure 3: Model Architecture Simple CNN

b. Regularized Convolutional Neural Network Architectures for Increased Depth:

In our development process we constructed an advanced CNN model by expanding convolutional layers while incorporating regularization methods to enhance performance and mitigate overfitting.

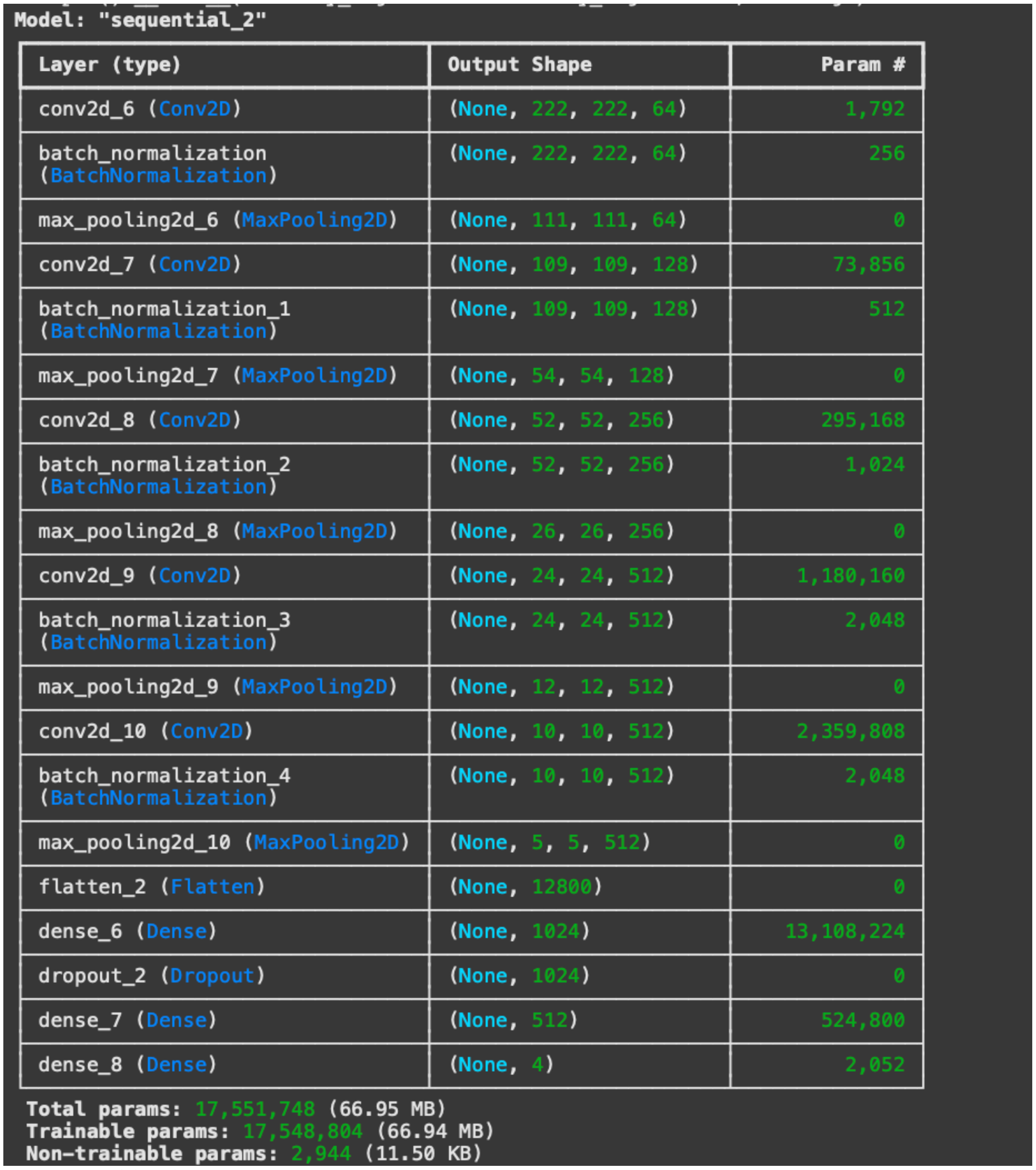
The changes made:

- The convolutional layer count experienced a twofold increase.
- The integration of Batch Normalization post-convolutional layers serves to stabilize the training process.
- Following dense layers, dropout layers with a range of 0.3 to 0.5 were implemented to mitigate overfitting.
- The fundamental framework expanded both in depth and breadth beyond natural proportions.

Training Setup:

- The dataset partitioning and augmentation processes remain identical to previous methods.
- The testing process included both Adam and SGD optimizers.
- The advanced neural architecture underwent training for 15–20 epochs with early termination protocols.

The collection of training and validation metrics enabled performance comparison against the baseline model.



```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_6 (Conv2D)               │ (None, 222, 222, 64)   │         1,792 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization             │ (None, 222, 222, 64)   │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_6 (MaxPooling2D)  │ (None, 111, 111, 64)   │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_7 (Conv2D)               │ (None, 109, 109, 128)  │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_1           │ (None, 109, 109, 128)  │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_7 (MaxPooling2D)  │ (None, 54, 54, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_8 (Conv2D)               │ (None, 52, 52, 256)    │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_2           │ (None, 52, 52, 256)    │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_8 (MaxPooling2D)  │ (None, 26, 26, 256)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_9 (Conv2D)               │ (None, 24, 24, 512)    │     1,180,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_3           │ (None, 24, 24, 512)    │         2,048 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_9 (MaxPooling2D)  │ (None, 12, 12, 512)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_10 (Conv2D)              │ (None, 10, 10, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_4           │ (None, 10, 10, 512)    │         2,048 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_10 (MaxPooling2D) │ (None, 5, 5, 512)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_2 (Flatten)             │ (None, 12800)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_6 (Dense)                 │ (None, 1024)           │    13,108,224 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_2 (Dropout)             │ (None, 1024)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_7 (Dense)                 │ (None, 512)            │       524,800 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_8 (Dense)                 │ (None, 4)              │         2,052 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 17,551,748 (66.95 MB)
 Trainable params: 17,548,804 (66.94 MB)
 Non-trainable params: 2,944 (11.50 KB)
```

Figure 4: Model Architecture Deeper CNN

c. Transfer Learning with VGG16:

The concluding section involved implementing transfer learning techniques with the VGG16 model which had been pre-trained on the ImageNet dataset. Through this mechanism we extracted advantages from features acquired across an extensive and varied dataset.

Steps followed:

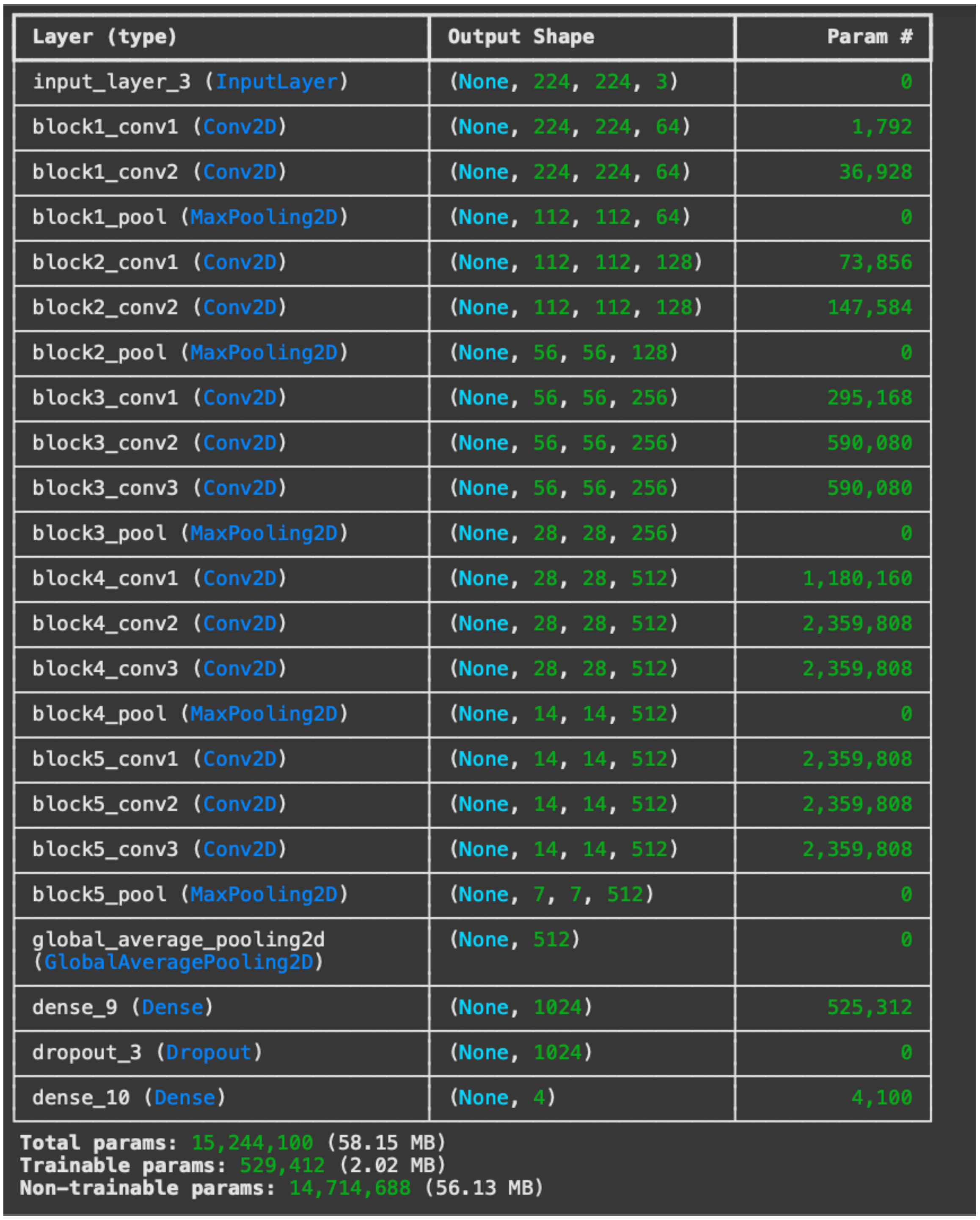
- The VGG16 model initializes with include_top set to False to remove its default classification layer.
- The convolutional base was immobilized to preserve its acquired features.
- The implementation of specialized Dense layers designed for our 4-class classification task has been completed.
- Applied Global Average Pooling prior to Dense layers.
- Terminal Dense layer configured with four units employing SoftMax activation function.

Training Configuration:

- Through feature extraction methods: solely the custom layers atop were trained.
- Optimizer: Adam
- Image input size: resized to 224×224
- Epochs: 10

The training process for this model demanded a reduced number of epochs while it reached high accuracy levels and maintained a minimal overfitting risk.

The examination of these three models enabled us to understand fully how various CNN architectures and techniques impact image classification results. The subsequent section will serve as the platform where we proceed to examine and juxtapose their outcomes.



Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense_9 (Dense)	(None, 1024)	525,312
dropout_3 (Dropout)	(None, 1024)	0
dense_10 (Dense)	(None, 4)	4,100

```
Total params: 15,244,100 (58.15 MB)
Trainable params: 529,412 (2.02 MB)
Non-trainable params: 14,714,688 (56.13 MB)
```

Figure 5: Model Architecture Transfer Learning VGG16

3. Experiments and Results:

Upon completing the construction and training phases for all three models, we subjected them to evaluation through multiple metrics such as accuracy, precision, recall, F1-score, and confusion matrices. Our investigation included an examination of training duration alongside loss trajectories and optimizer effectiveness to discern model behavior during training phases and its ability to generalize across novel datasets.

a. Baseline Model Results

Through an unnatural training process, the baseline CNN model underwent 10 epochs of development using the Adam optimizer. It achieved:

- Training Accuracy: ~98%
- The recorded validation accuracy stands at approximately 96%.

The model demonstrated effective generalization while starting to exhibit minor overfitting tendencies after several epochs. The confusion matrix indicated a high rate of correct predictions while showing sporadic misclassifications among closely related disease categories. The loss curves for t raining and validation remained mostly stable while displaying a minor gap which suggests the model performed acceptably.

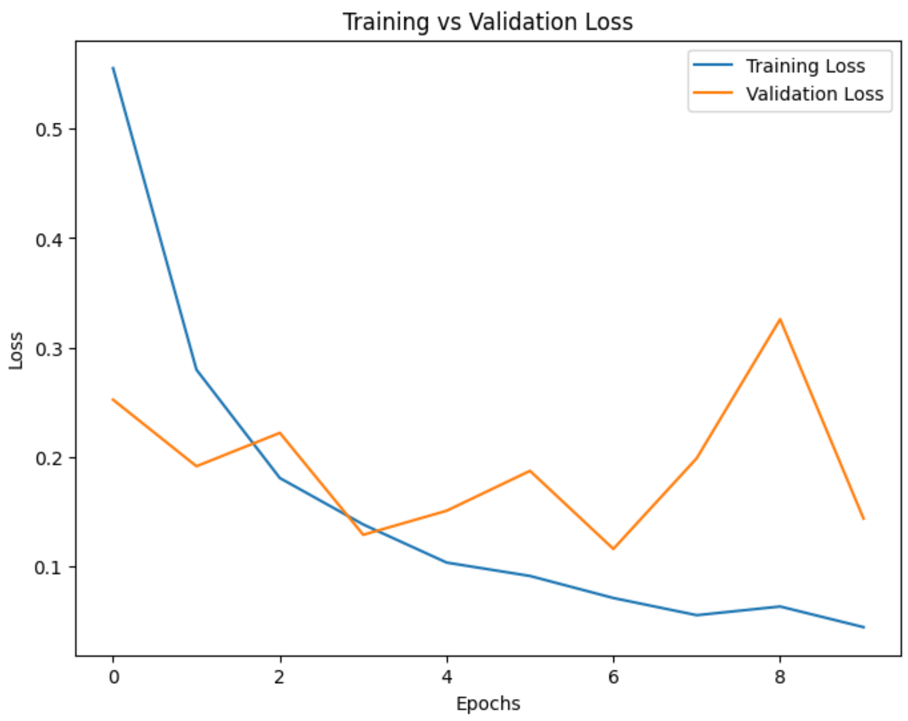
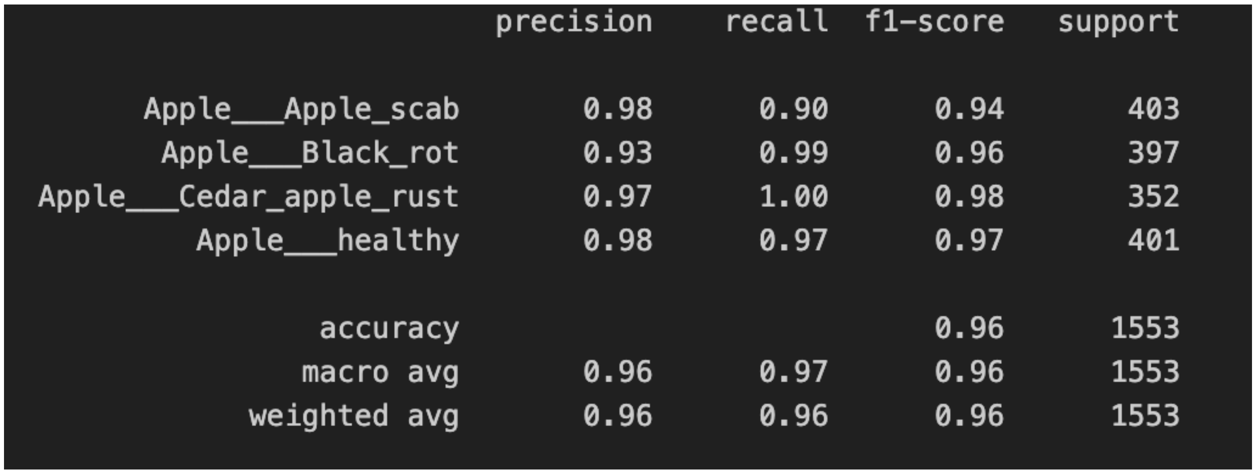


Figure 6: Training Validation Loss Simple CNN



```
                          precision    recall  f1-score   support

       Apple___Apple_scab       0.98      0.90      0.94       403
        Apple___Black_rot       0.93      0.99      0.96       397
 Apple___Cedar_apple_rust       0.97      1.00      0.98       352
          Apple___healthy       0.98      0.97      0.97       401

                 accuracy                           0.96      1553
                macro avg       0.96      0.97      0.96      1553
             weighted avg       0.96      0.96      0.96      1553
```

Figure 7: Classification Report Simple CNN

b. Deeper CNN Results

The intricate CNN architecture underwent training sessions using both Adam and SGD optimizers across 15–20 epochs. The design incorporated extra convolutional layers along with dropout techniques and batch normalization to enhance performance while mitigating overfitting.

Through the application of Adam optimizer:

- The recorded validation accuracy reached approximately 86%
- The recorded training accuracy rate exceeds ninety-six percent.

Through the employment of the SGD optimizer:

- Validation Accuracy: slightly higher (98%)
- Training processes exhibits slowed convergence rates while maintaining increased stability.

The more intricate model demonstrated enhanced performance compared to the baseline across accuracy and F1-score metrics. The implementation of regularization techniques succeeded in mitigating overfitting while batch normalization enabled accelerated and stable training processes. The training process required extended durations beyond the baseline because of increased complexity.

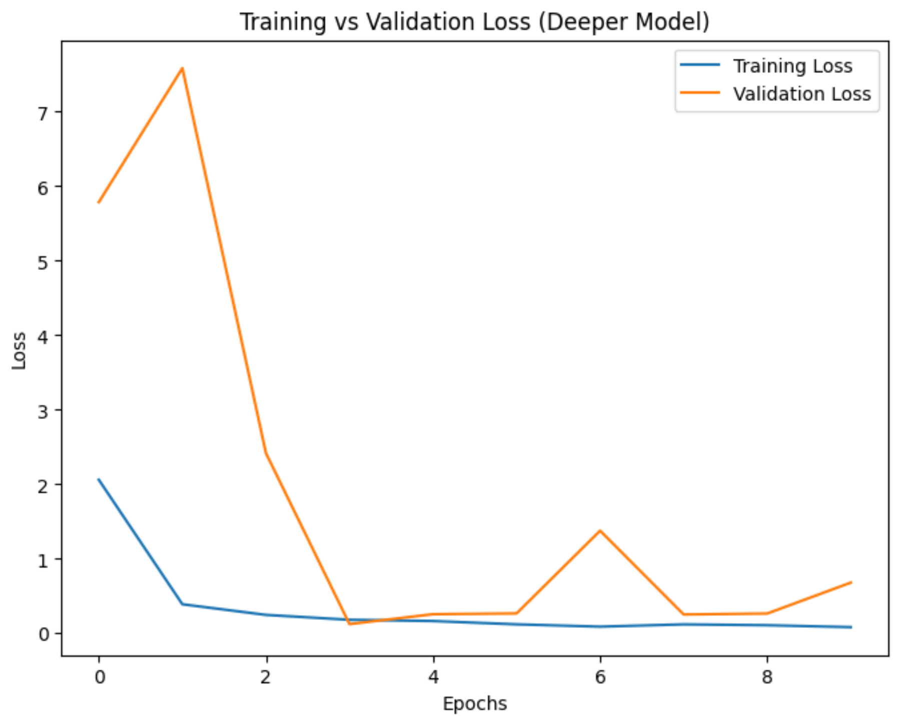
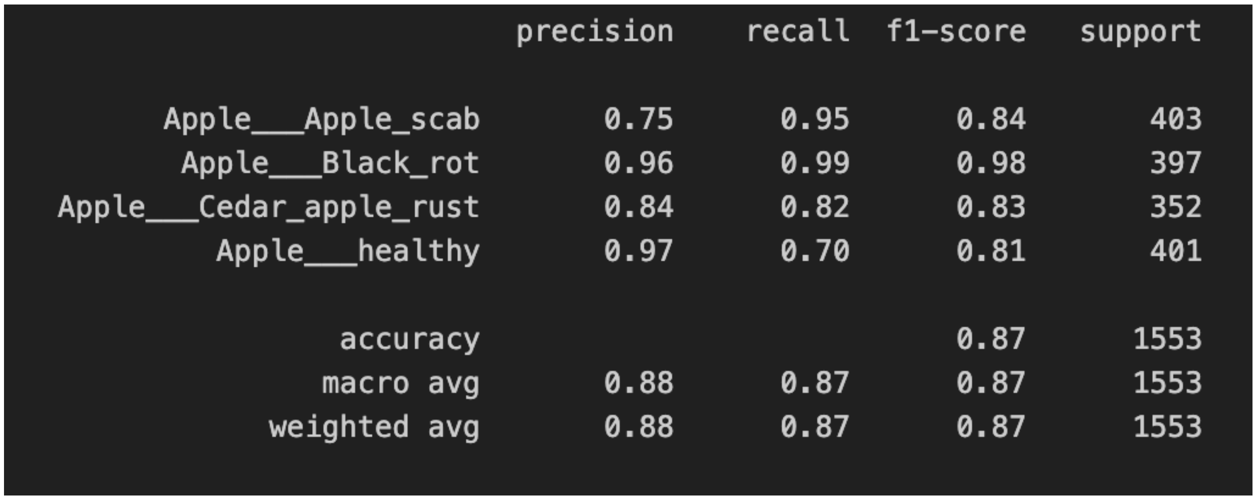


Figure 8: Training Validation Loss Deeper CNN



```
                          precision    recall  f1-score   support

       Apple___Apple_scab      0.75      0.95      0.84       403
        Apple___Black_rot      0.96      0.99      0.98       397
 Apple___Cedar_apple_rust      0.84      0.82      0.83       352
          Apple___healthy      0.97      0.70      0.81       401

                 accuracy                          0.87      1553
                macro avg      0.88      0.87      0.87      1553
             weighted avg      0.88      0.87      0.87      1553
```

Figure 9: Classification Report Deeper CNN

c. Transfer Learning (VGG16) Results

Utilizing feature extraction by freezing base layers while training custom dense layers formed the VGG16-based model. Among the trio of models examined, this one demonstrated superior performance metrics.

- The recorded validation accuracy reaches approximately 96%
- Loss: minimal at approximately 0.1.

The confusion matrix demonstrated a negligible number of misclassified instances. Thanks to its pre-trained weights, the model necessitated a reduced number of training epochs while achieving faster convergence. The study demonstrates how transfer learning provides advantages for projects with restricted data availability and computational power.

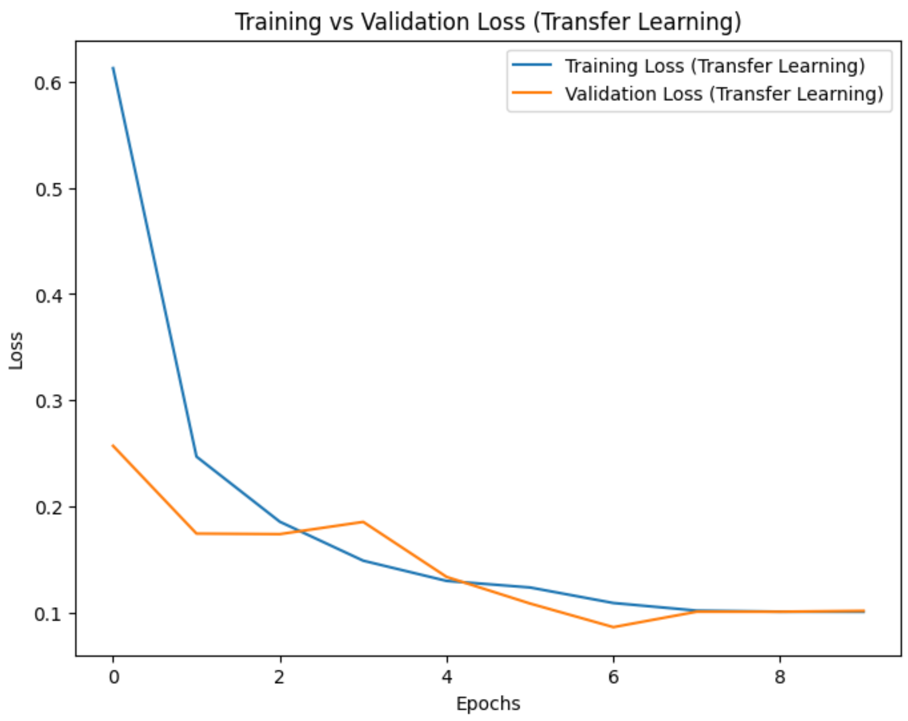
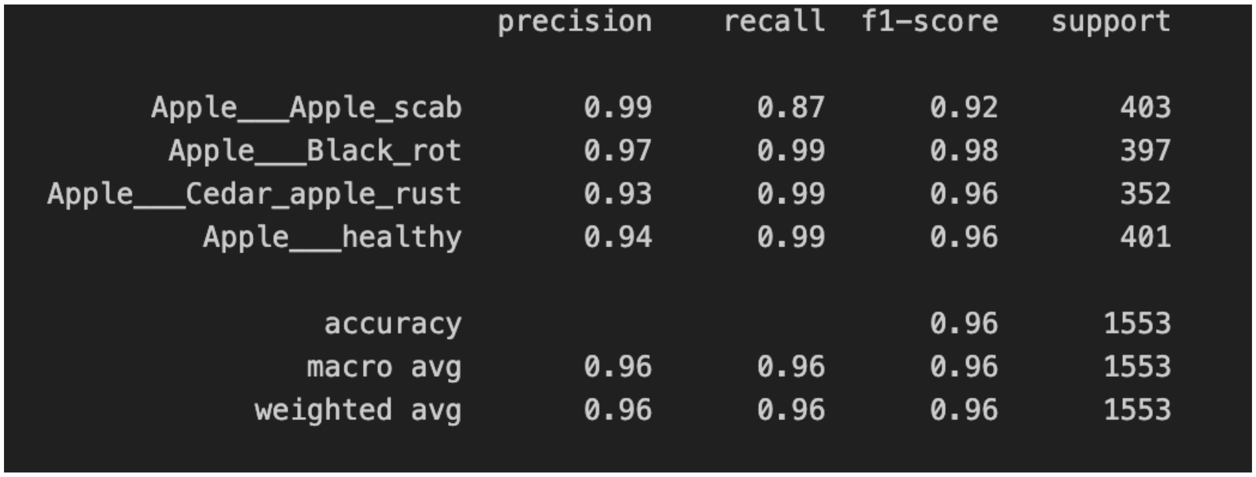


Figure 10: Training Validation Loss Transfer Learning



```
                        precision    recall  f1-score   support

      Apple___Apple_scab       0.99      0.87      0.92       403
       Apple___Black_rot       0.97      0.99      0.98       397
Apple___Cedar_apple_rust       0.93      0.99      0.96       352
         Apple___healthy       0.94      0.99      0.96       401

                accuracy                           0.96      1553
               macro avg       0.96      0.96      0.96      1553
            weighted avg       0.96      0.96      0.96      1553
```

Figure 11: Classification Report Transfer Learning

d. Optimizer Comparison

- Adam: The process reaches stability more quickly while achieving superior ultimate precision.
- SGD: Stochastic Gradient Descent operates at reduced speeds yet occasionally achieves greater stability making it suitable for fine-tuning processes.

The training process for Adam proved to be more efficient when developing both custom CNNs and implementing transfer learning techniques. The application of SGD with reduced learning rates proved effective in diminishing overfitting.

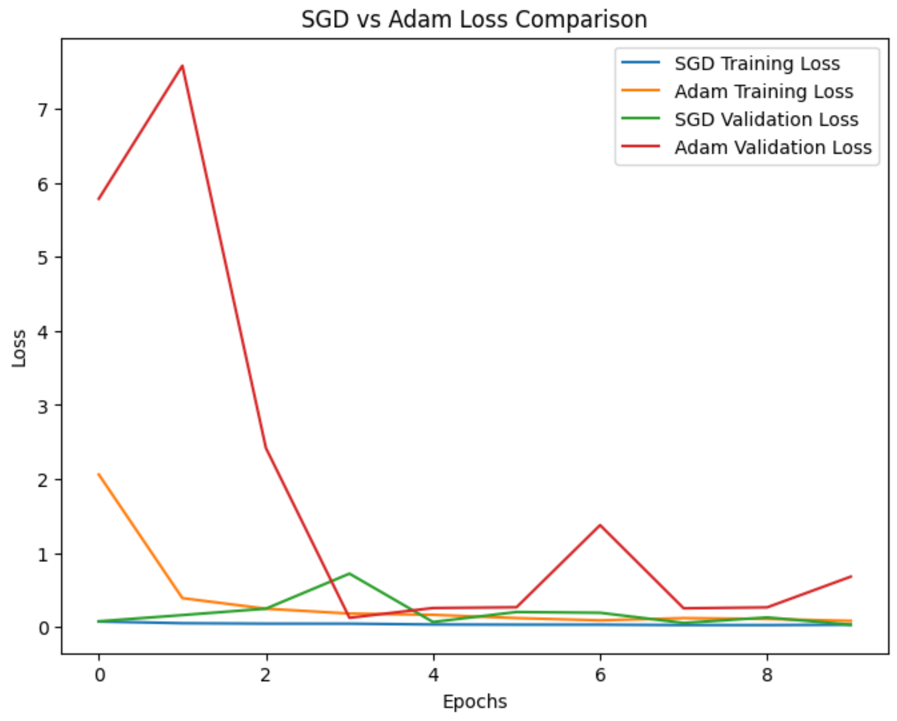


Figure 12: SGD VS Adam

e. Hardware and Efficiency

Every model underwent training on Google Colab with GPU acceleration. Training the deeper CNN model required the most time whereas the transfer learning model achieved optimal efficiency regarding both training duration and accuracy performance (SharadaP.Mohanty, 2016).

4. ROC Curves

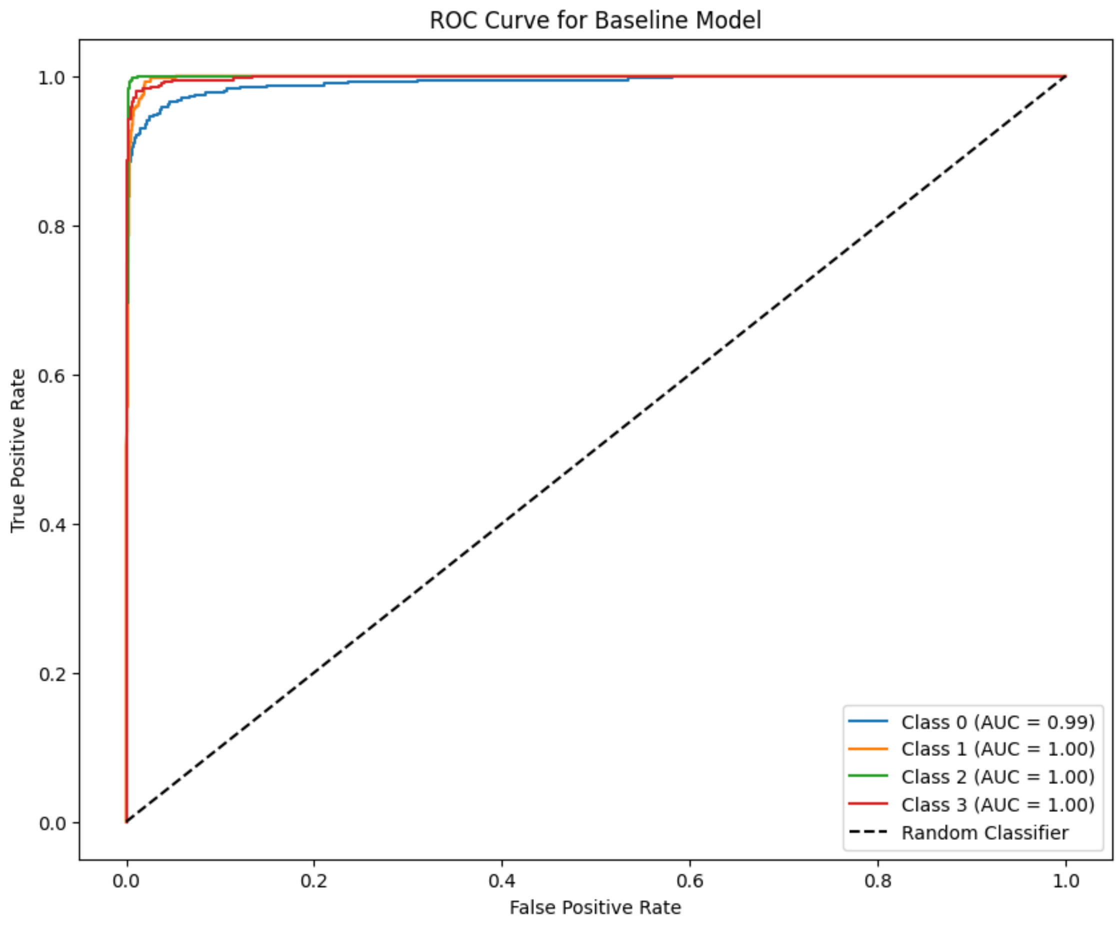


Figure 13: ROC Curve Simple CNN

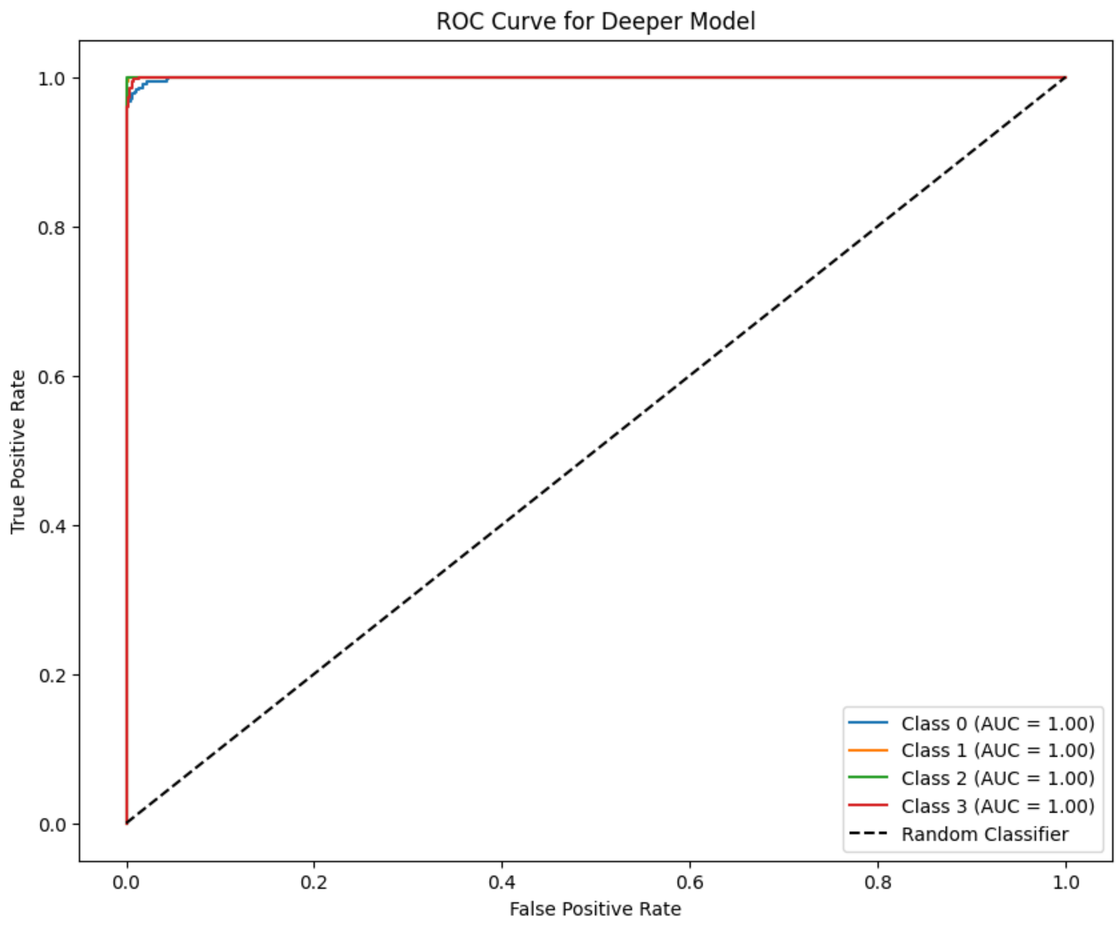


Figure 14: ROC Curve Deeper CNN

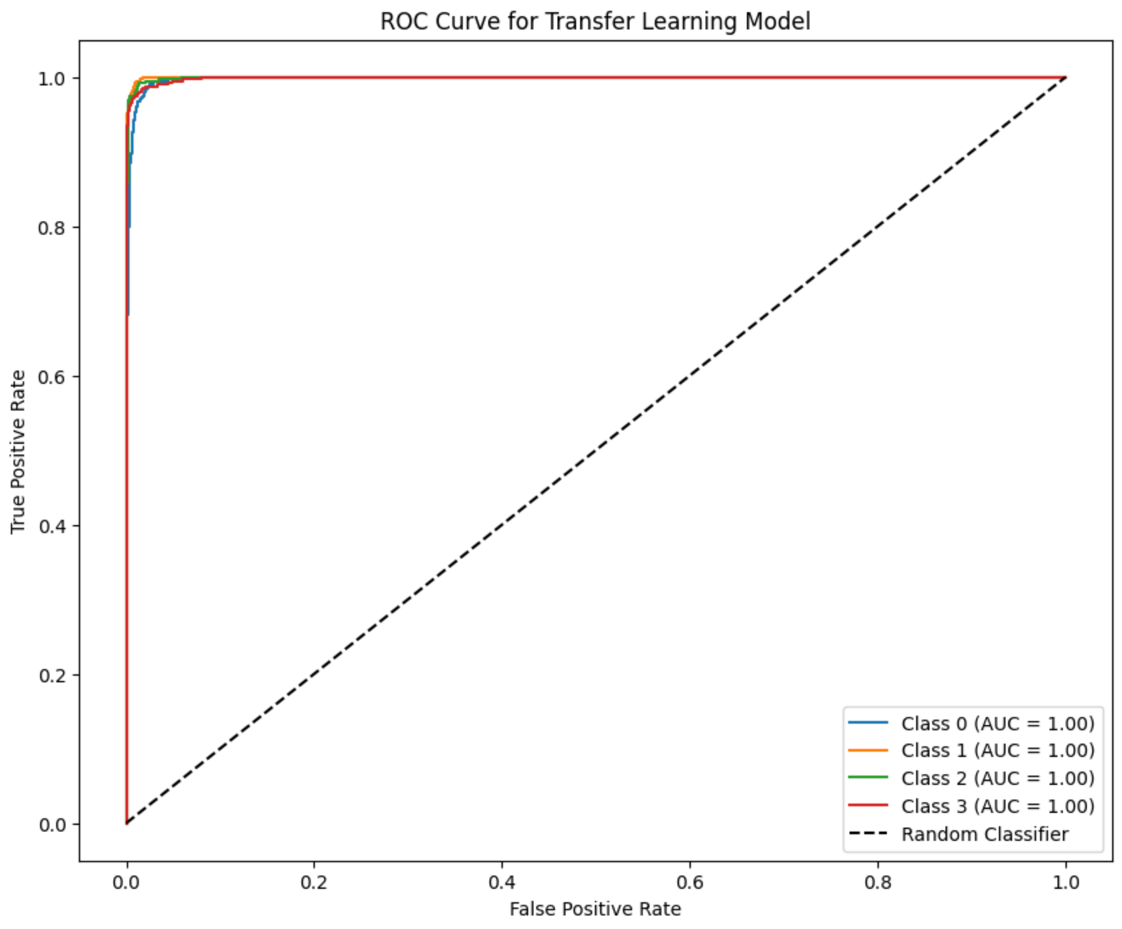


Figure 15: ROC Curve Transfer Learning

5. Discussion: Challenges and Observations

Throughout the project we faced multiple obstacles which allowed us to obtain critical understanding regarding the behavior of various CNN architectures during actual training scenarios.

a. Overfitting and Underfitting

The primary problem identified involved overfitting within both baseline and advanced CNN architectures. The training accuracy exhibited a consistent upward trend while validation accuracy reached a plateau or declined after several epochs. The models absorbed excessive information from the training data which caused difficulties in generalizing to new situations. To address this, we used Dropout layers in the deeper model and applied Early Stopping during training. The application of these methods prevented overfitting by terminating the training process when validation performance ceased to show advancements.

b. Training Time and Resources

The deeper CNN model's training duration surpassed the baseline model because of its numerous layers and parameters which extended the processing time. The management of this task would have presented significant challenges in the absence of GPU support. The deployment of GPU acceleration in the Google Colab, enabled us to achieve faster and more efficient training outcomes.

The VGG16 transfer learning model exhibited accelerated training speeds because the majority of its layers had pre-existing training from ImageNet data. The dataset's balanced nature combined with our restricted computational resources made this solution perfectly suited to our problem.

c. Optimizer Performance

Our trials included the utilization of both Adam and SGD optimizers. The deeper model exhibited enhanced convergence speed and smoothness with Adam. The process of fine-tuning pre-trained models occasionally benefited from using SGD with reduced learning rates to diminish overfitting while achieving marginally more consistent outcomes.

d. Technical Limitations

A significant constraint emerged from Colab's limited memory capacity which posed challenges for training deeper models. The careful management of batch sizes alongside augmentation techniques was necessary to prevent memory errors from occurring. The process of saving and restoring checkpoints proved beneficial for crash recovery.

The project enabled us to explore and comprehend the intricate trade-offs among model complexity, training duration, and accuracy performance. The study underscored the necessity of both regularization techniques and optimizer adjustments during CNN training sessions.

6. Conclusion and Future Work:

Through this project we investigated various image classification methodologies utilizing Convolutional Neural Networks (CNNs). Through experimentation with an actual apple leaf image dataset, we developed and tested three distinct models: a basic CNN, an advanced CNN with regularization techniques, and a transfer learning model utilizing VGG16.

The initial model served as an effective foundation yet demonstrated overfitting tendencies alongside restricted generalization capabilities. The advanced model featuring dropout and batch normalization achieved better performance and stability yet demanded increased training time and computational resources. The transfer learning model incorporating VGG16 delivered superior performance by attaining high accuracy while minimizing training duration. The use of pre-trained networks for specialized tasks became evident as beneficial particularly in situations where computational resources or large datasets remained scarce.

Our investigation revealed that optimizer selection produced substantial variations in model performance. Adam demonstrated effective performance for rapid convergence whereas SGD provided greater control during the fine-tuning of large models.

Numerous potential enhancements exist for this project when considering future development opportunities. Exploring additional pre-trained models such as ResNet and EfficientNet could potentially deliver superior performance outcomes. The exploration of varied learning rates alongside data augmentation techniques and ensembling methods offers potential performance enhancement. The transformation of this model into a real-time web application or mobile tool for farmers would render the research practically impactful.

Through our work on this project we gained practical experience with deep learning pipelines from preprocessing to evaluation stages while observing how various model architectures impact real-world image classification performance.

References

Timilsina, S., Gautam, M. & Bhattarai, B., 2022. *NepBERTa: Nepali Language Model Trained in a Large Corpus.* s.l., Association for Computational Linguistics.
SharadaP.Mohanty, 2016. Using Deep Learning for Image-Based Plant Disease Detection. *frontiers in Plant Science,* Volume 7, p. 10.

Mohanty, S.P., Hughes, D.P. and Salathé, M., 2016. *Using deep learning for image-based plant disease detection*. Frontiers in Plant Science, 7, p.1419. https://doi.org/10.3389/fpls.2016.01419

Simonyan, K. and Zisserman, A., 2015. *Very deep convolutional networks for large-scale image recognition*. In *International Conference on Learning Representations (ICLR)*. Available at: https://arxiv.org/abs/1409.1556

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. *ImageNet classification with deep convolutional neural networks*. In *Advances in Neural Information Processing Systems*, 25, pp.1097-1105.

Ferentinos, K.P., 2018. *Deep learning models for plant disease detection and diagnosis*. Computers and Electronics in Agriculture, 145, pp.311-318. https://doi.org/10.1016/j.compag.2018.01.009

Shorten, C. and Khoshgoftaar, T.M., 2019. *A survey on image data augmentation for deep learning*. Journal of Big Data, 6(1), p.60. https://doi.org/10.1186/s40537-019-0197-0

Pan, S.J. and Yang, Q., 2010. *A survey on transfer learning*. IEEE Transactions on Knowledge and Data Engineering, 22(10), pp.1345–1359. https://doi.org/10.1109/TKDE.2009.191

Git Link

Click here to load the Git repo containing the code of Assessment One.