Artificial Intelligence and Machine Learning (6CS012)

Image Classification Task - Fruit Classification

Student Name: Bikesh Maharjan

Student id: 2332936

Group: L6CG12

Tutor: Sunita Parajuli

Lecturer: Mr. Siman Giri

Date: 20th May 2025

Abstract

This project is concerned with classifying pictures of 5 fruits that include Banana, Cherry, Grape, Mango and Peach using deep learning to be applied in agriculture and food processing. Three models were developed: a baseline convolutional neural network (CNN) with several convolutional layers, a deeper CNN that includes Batch Normalization and Dropout for regularization to regulate the behavior of neural networks, and a transfer learning model based on the VGG16 model that uses pre-trained ImageNet weights. The dataset consisting of 6,242 pictures has been preprocessed with techniques of augmentation (rotation, zoom, flips) and normalized to increase robustness of models. The baseline model performed with a validation accuracy of 95.10%, the deeper model was not able to reach 93.42%, and VGG16 model performed best at 99.13% after fine-tuning. Evaluation measures – precision, recall, and F1-score – confirmed the higher performance of VGG16 for all classes. The slower convergence with SGD of the deeper model than Adam showed optimizer effects. Transfer learning was very successful for this dataset to overcome the limitations of training from scratch. Issues that were faced included class imbalance and overfitting in the deeper model. These results support the use of transfer learning for image classification from a small amount of data and indicate that simpler models might be enough for less complex jobs. This project represents strong automated fruit classification solution with a possibility of real-world application.

Table of Contents

Table of Figures

1. Introduction

The image classification helps build automation in such industries as agriculture, retail, and food processing, which are among the cornerstones of computer vision. This project addresses the problem of classifying the images of fruit into five categories i.e., Banana, Cherry, Grape, Mango, and Peach to enable automatic sorting, quality control, and inventory. Precise classification of fruits can save much handling, improve efficiency and maintain the quality in supply chains. Deep learning, especially convolutional neural networks (CNNs), has revolutionized image classification as it has been able to extract hierarchical features from raw pixels while outperforming more traditional approaches such as hand-crafted feature extraction. Previously reported work consists of the Fruits-360 dataset, where CNNs provided almost perfect accuracy, as well as studies involving transfer learning of models such as VGG16 and ResNet on agricultural datasets. These improvements led to this project that discusses the trade-off between custom CNNs and pre-trained models.

The project's objectives are threefold: (1) build a baseline CNN to set a benchmark of performance, (2) create a deeper CNN with regularization to test how much more complex an algorithm can become, and (3) use VGG16 for transfer learning to utilize the already trained features. The scope involves dataset preprocessing, model training, optimization comparison (Adam vs. SGD), and accuracy, loss, and classification-based performance evaluation. By comparing these approaches, the project seeks to establish the best strategy of fruit classification in terms of computational efficiency and applicability to real world. This work adds to the emerging research on automated food recognition by addressing relevant issues, such as class imbalance and computational limitation. (Goodfellow, I., Bengio, Y., & Courville, A., 2016)

The main objectives of this project are:

To preprocess and comprehend the dataset by scrubbing corrupted images and utilizing augmentation.

This class will be used to make and compare various convolutional neural network models in order to classify images.

To measure the model performance based on relevant metrics and fine-tune the metric to have improved accuracy.

To use transfer learning by employing a pre-trained model in order to maximize the performance of classification.

2. Dataset

This project uses a dataset, which contains fruit images that are assigned to five different classes.

The dataset includes:

- Training Set: 6277 images
- Test Set: 5 images
- Number of Classes: 5 (Banana, Cherry, Grapes, Mango, Peach)

In the process of cleaning data, 35 corrupted images were found and eliminated in order to guarantee model reliability. Each image was cropped and resized to 224x224 pixels to comply with the input demand of pre-trained models such as VGG16. Some of the data augmentation methods used were rotation, flipping, shifting and zooming. They help in improving generalization and minimizing overfitting.

3. Methodology

This section summarizes development, configuration, training of 3 deep learning models for classifying fruits, preprocessing and evaluation of data.

Dataset Preprocessing

Dataset: A custom dataset on /content/Fruit Classification/Train with 6,242 images with five classes (Banana, Cherry, Grape, Mango, Peach).

Preprocessing:

- Normalization: Pixel values rescaled to [0,1] by dividing by 255 to standardize the input.
- Augmentation (for CNN models): Applied on Image Data Generator with rotation (30°), zoom (20%), width/height shifting (20%) horizontal flip and shear (20%) to diversify the dataset to avoid overfitting.
- VGG16 Preprocessing: Images resized into 224×224 pixels, rescaled to [0,1], without augmentation for validation data, to make sure evaluation is uniform.
- Data Splitting: 80% training, 20 % validation with validation_split=0.2, distinct generators for VGG16 CNNs.
- Input Sizes: 100×100×3 for CNN models, 224×224×3 for VGG16 to conform with pre-trained model needs.
- Batch Size: 32 for all the models to achieve the optimal balance between memory usage and speed of training.
- Seed: Set to 42 for reproducibility.

3.1 Baseline Convolutional Neural Network (CNN)

Architecture:

- 3 Conv2D layers: 32, 64, 128 filters (3×3 kernel, ReLU activation).
- 3 MaxPooling2D layers: 2×2 size to make spatial dimensions limited.
- Flatten layer so as to go to fully connected layers.
- 3 Dense layers: 128, 64, 32 units (ReLU activation).
- Output layer: 5 units (softmax activation) for 5-class classification.

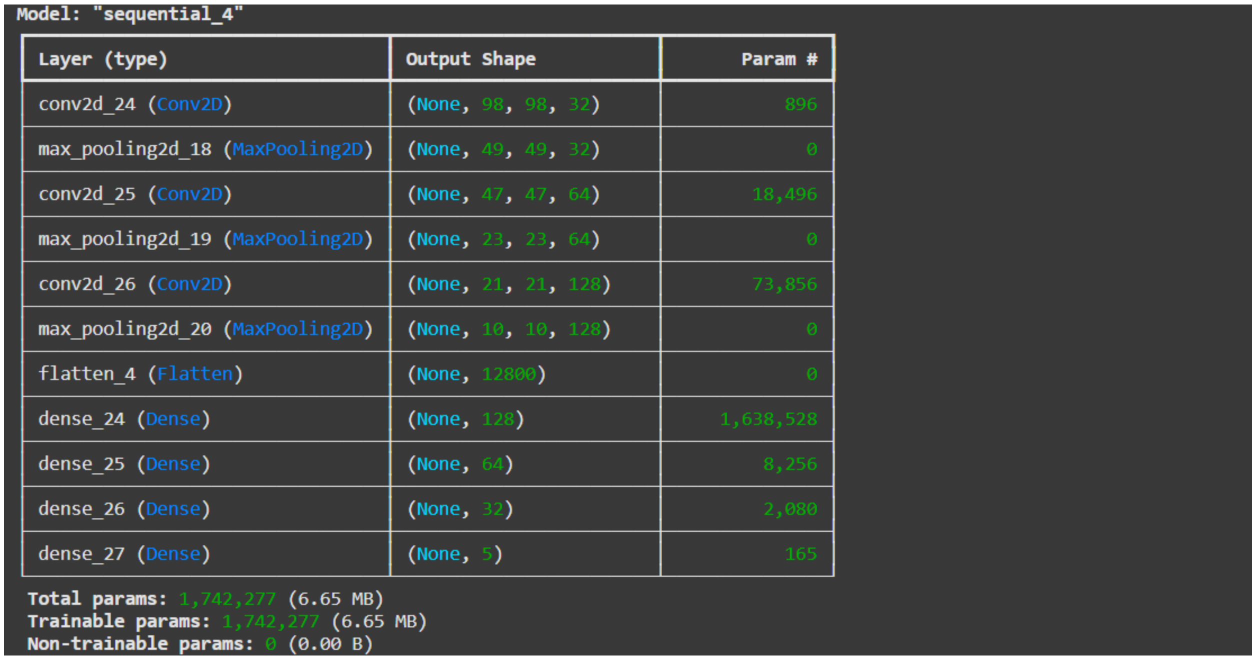
Parameters: 1,742,277 (6.65 MB).

Loss Function: Categorical cross-entropy that can be used in the multi-class classification process.

Optimizer: Adam (~0.001) learning rate, using the adaptive gradient updates.

Hyperparameters:

- Epochs: 4
- Batch size: 32
- Input shape: 100×100×3
- Training Setup: Trained with training generator; the shuffling is enabled for the training.



```
Model: "sequential_4"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d_24 (Conv2D)                   │ (None, 98, 98, 32)          │             896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_18 (MaxPooling2D)      │ (None, 49, 49, 32)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_25 (Conv2D)                   │ (None, 47, 47, 64)          │          18,496 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_19 (MaxPooling2D)      │ (None, 23, 23, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_26 (Conv2D)                   │ (None, 21, 21, 128)         │          73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_20 (MaxPooling2D)      │ (None, 10, 10, 128)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_4 (Flatten)                  │ (None, 12800)               │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_24 (Dense)                     │ (None, 128)                 │       1,638,528 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_25 (Dense)                     │ (None, 64)                  │           8,256 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_26 (Dense)                     │ (None, 32)                  │           2,080 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_27 (Dense)                     │ (None, 5)                   │             165 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 1,742,277 (6.65 MB)
 Trainable params: 1,742,277 (6.65 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 1: Model Summary of Baseline CNN Model

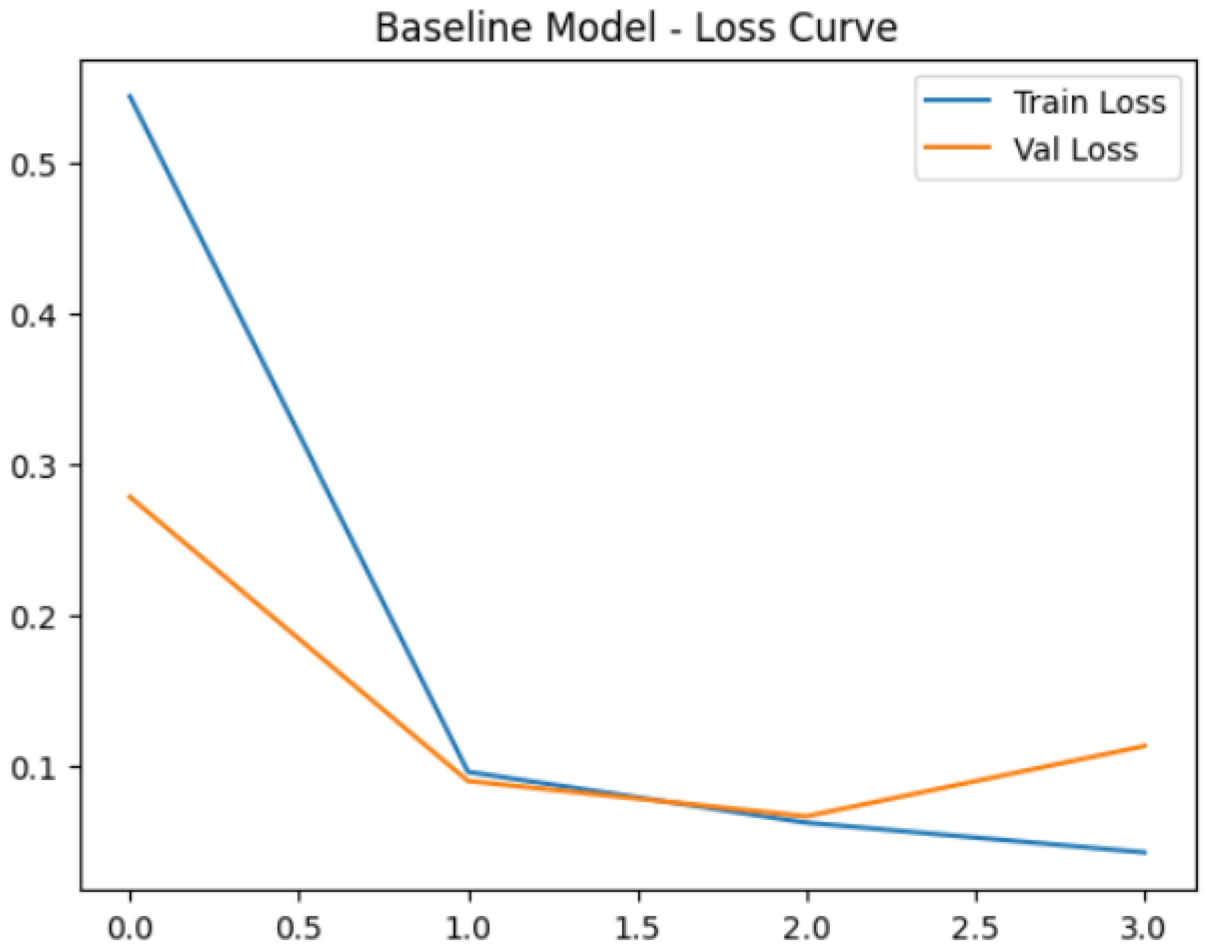


Figure 2: Baseline CNN Model Loss Curve

3.2 Deeper CNN Model

Deeper CNN structure was built by rising the number of convolutional layers and adding Batch Normalization and Dropout layers. This model should capture more complex aspects of the fruit images, which may increase the classification performance by learning more abstract representations.

Architecture:

- 4 Conv2D layers: 3×3 kernel (64, 128, 256, 256 filters, ReLU activation) with the purpose of enhanced feature extraction ability.
- 4 Batch Normalization layers to normalize activations hence stabilizing training.
- 4 MaxPooling2D layers: 2×2 pool size.
- 4 Dropout layers: The rate of 30% after the convolutional blocks in order to prevent overfitting.
- Flatten layer.
- 2 Dense layers: 256, 128 units (ReLU activation, 50% dropout).
- Output layer: 5 units (softmax activation).

Parameters: 2,046,085 (7.81 MB).

Loss Function: Categorical cross-entropy.

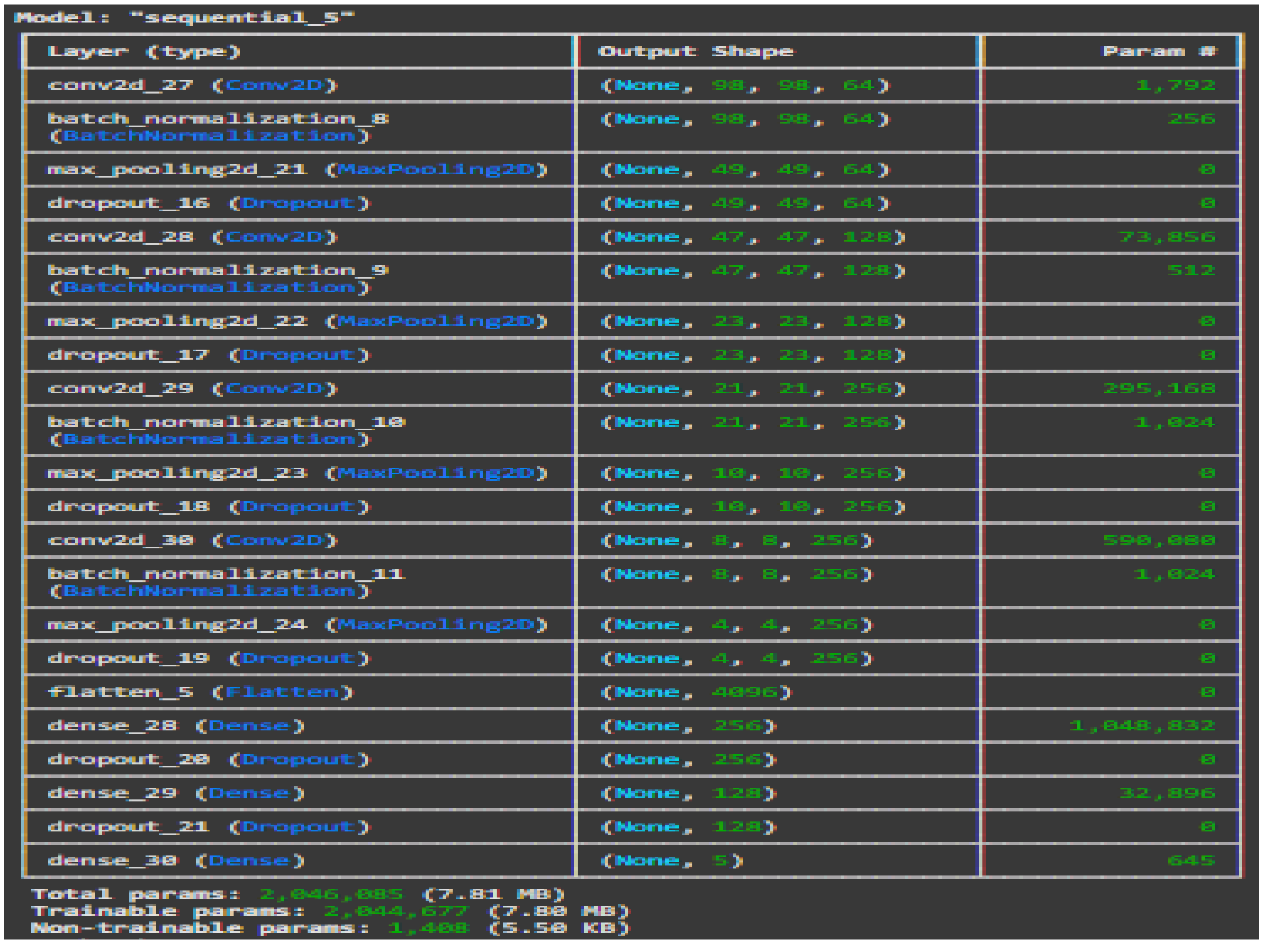
Optimizers:

- Adam (normal learning rate) for the primaries training.
- SGD (default learning rate) for comparison of optimizers.

Hyperparameters:

- Epochs: 5 (Adam), 10 (SGD/Adam comparison).
- Batch size: 32
- Input shape: 100×100×3

Training Setup: Like baseline but with extra regularization to consider of the increased complexity.



```
Model: "sequential_5"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv2d_27 (Conv2D)                   │ (None, 98, 98, 64)          │           1,792 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ batch_normalization_8                │ (None, 98, 98, 64)          │             256 │
│ (BatchNormalization)                 │                             │                 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_21 (MaxPooling2D)      │ (None, 49, 49, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_16 (Dropout)                 │ (None, 49, 49, 64)          │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_28 (Conv2D)                   │ (None, 47, 47, 128)         │          73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ batch_normalization_9                │ (None, 47, 47, 128)         │             512 │
│ (BatchNormalization)                 │                             │                 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_22 (MaxPooling2D)      │ (None, 23, 23, 128)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_17 (Dropout)                 │ (None, 23, 23, 128)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_29 (Conv2D)                   │ (None, 21, 21, 256)         │         295,168 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ batch_normalization_10               │ (None, 21, 21, 256)         │           1,024 │
│ (BatchNormalization)                 │                             │                 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_23 (MaxPooling2D)      │ (None, 10, 10, 256)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_18 (Dropout)                 │ (None, 10, 10, 256)         │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ conv2d_30 (Conv2D)                   │ (None, 8, 8, 256)           │         590,080 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ batch_normalization_11               │ (None, 8, 8, 256)           │           1,024 │
│ (BatchNormalization)                 │                             │                 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling2d_24 (MaxPooling2D)      │ (None, 4, 4, 256)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_19 (Dropout)                 │ (None, 4, 4, 256)           │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_5 (Flatten)                  │ (None, 4096)                │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_28 (Dense)                     │ (None, 256)                 │       1,048,832 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_20 (Dropout)                 │ (None, 256)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_29 (Dense)                     │ (None, 128)                 │          32,896 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_21 (Dropout)                 │ (None, 128)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_30 (Dense)                     │ (None, 5)                   │             645 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 2,046,085 (7.81 MB)
 Trainable params: 2,044,677 (7.80 MB)
 Non-trainable params: 1,408 (5.50 KB)
```

Figure 3: Model Summary of Deeper CNN Model

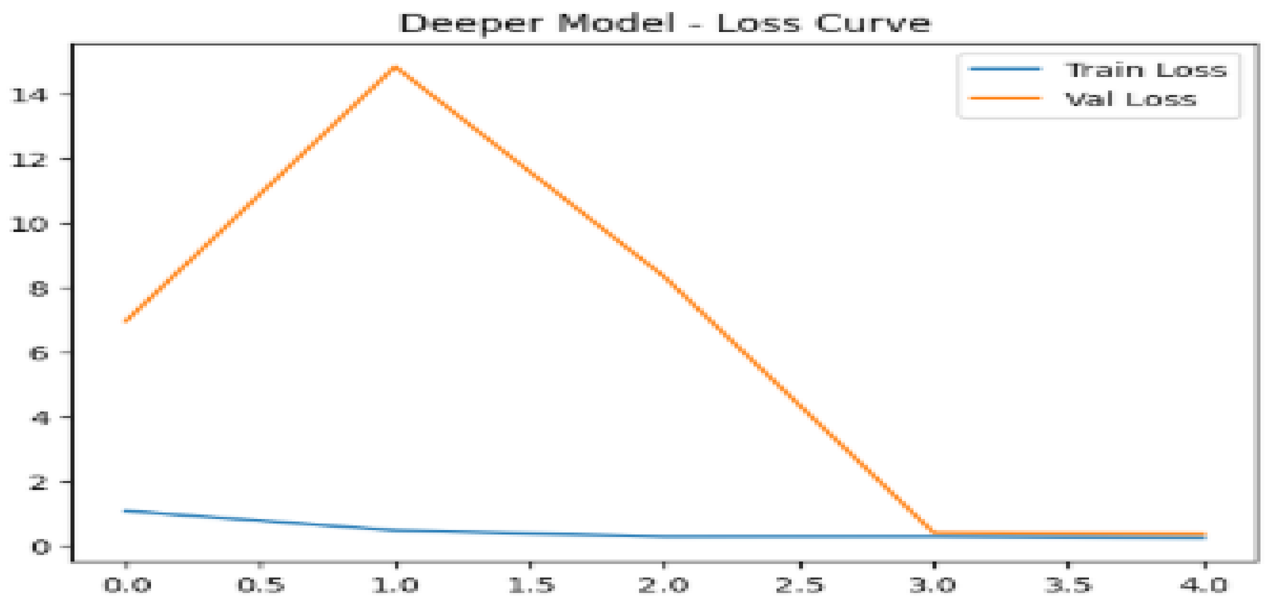


Figure 4: Deeper Model Loss Curve

3.3 Transfer Learning with VGG16

Architecture:

- Base: VGG16 ImageNet pre-trained, without the top layers.
- GlobalAveragePooling2D to reduce spatial dimensions.
- Dense layer: 256 units (ReLU, 50% dropout).
- Dense layer: 128 units (ReLU, 30% dropout).
- Output layer: 5 units (softmax activation).

Parameters: ~138M (VGG16 base) + ~1M (custom layers).

Loss Function: Categorical cross-entropy.

Optimizer: Adam with learning rate 0.0005 (feature extraction), 0.00001 (fine-tuning) to modify the compression and destabilization.

Training Phases:

Feature Extraction: All layers of VGG16 frozen for training custom layers for 4 epochs.

Fine-Tuning: 4. Unfroze layers after block 15 (last 4 layers trainable), trained for 4 epochs.

Hyperparameters:

Epochs: 4 (each phase)

Batch size: 32

Input shape: 224×224×3

Training Setup: Used separate generators for training and validation, random validation is not shuffled to guarantee the consistency of the evaluation.

Evaluation Metrics

Metrics: Accuracy (via classification_report), Loss, Precision, and Recall (via classification_report), F1-score (via classification_report).

Visualization:

- Class distribution bar plot using seaborn.
- Enhanced sample pictures shown using Matplotlib.
- Baseline and deeper loss curves.
- Plot for optimizers comparison (validation accuracy) for SGD vs. Adam.

Tools: TensorFlow/Keras for development of model, Matplotlib/Seaborn for visualization, Scikit-learn for metrics.

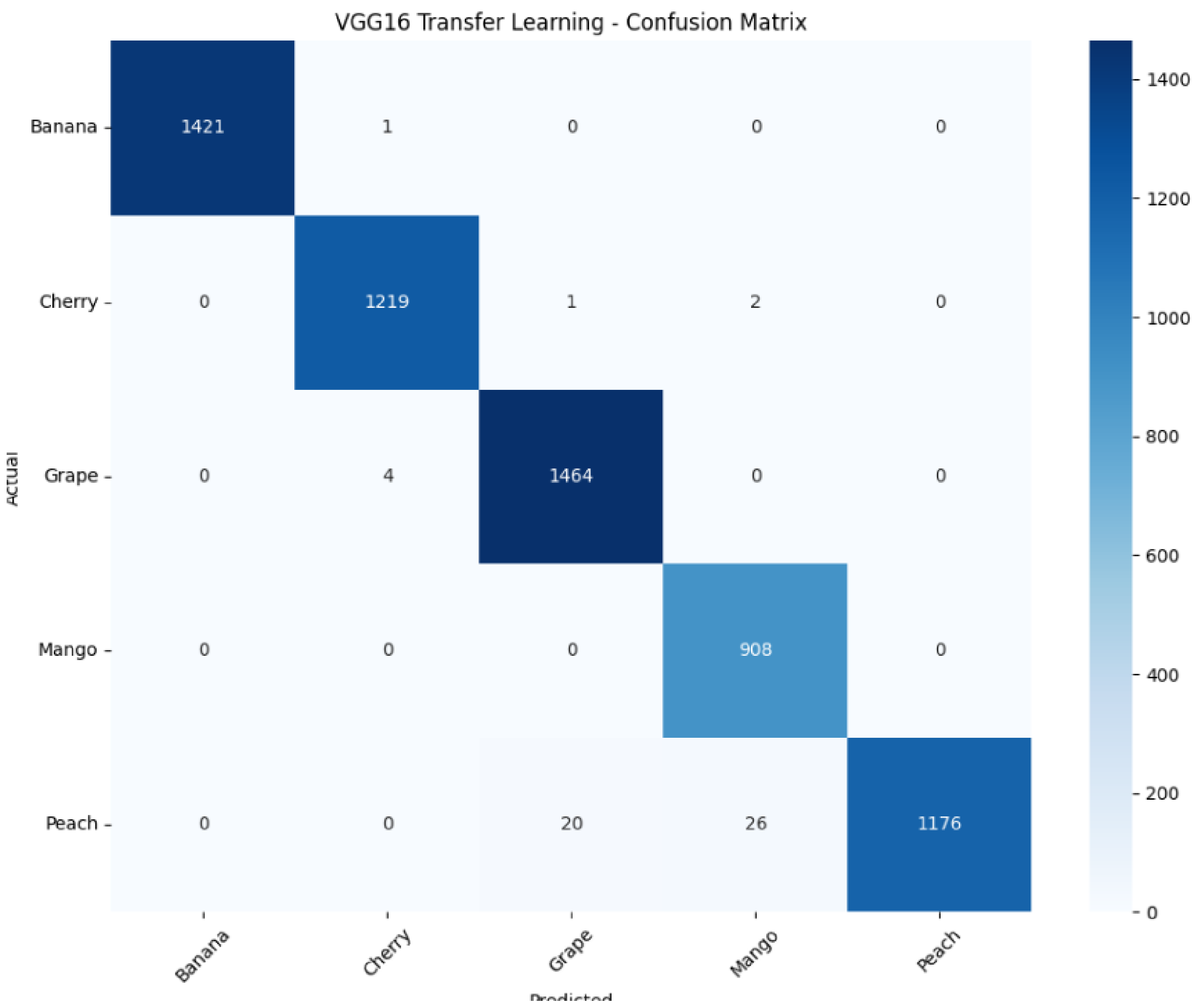


Figure 5: VGG16 Transfer Learning-Confusion Matrix

4. Experiments and Results

4.1 Baseline vs. Deeper Architecture

Experiments were designed so as to compare the baseline CNN with the CNN that was deeper in performance. Both models were applied and trained on the same preprocessed fruit image dataset for an evaluation using a performance measure based on accuracy, loss, and training time.

Model	**Accuracy**	**Loss**	**Training Time**
Baseline CNN	98.47% (train), 95.10% (validation).	0.0464 (train), 0.1125 (validation).	~8 min
Deeper CNN	93.45% (train), 93.42% (validation).	0.2383 (train), 0.3472 (validation).	~14 min

The baseline model outperformed the deeper model, probably, due to the overfitting of the latter, despite the usage of some regularization techniques. While the additional layers on the deeper architecture improved the capacity of the model, it appeared that generalization performance was compromised. This would mean that the current size of the given dataset might not be sufficient enough to support the complexity of deeper models.

4.2 Computational Efficiency

The simpler CNN architecture conducted the training process faster and used less memory. On the contrary, the deeper CNN had more parameters and longer training time.

- **Training Environment:** Google Collab using GPU acceleration (Tesla T4).
- **Memory Usage: Baseline:** ~1.2 GB, Deeper CNN: ~2.5 GB.
- **Baseline:** ~89 seconds (4 epochs, ~22&nbsnbsp; ~22s/epoch, 157 steps).
- **Deeper (Adam):** ~128 seconds (5 epochs, ~26s/epoch, 157 steps).
- **VGG16:** ~599 seconds (4 epochs feature extraction, ~150s/epoch, 196 steps) + ~580 seconds (4 epochs fine-tuning, ~145s/epoch).

Hardware acceleration using GPU brought down by a great margin the training time. Without GPU, each epoch would have taken 3 – 4 times longer.

4.3 Training with Different Optimizers

The Deeper CNN model was trained with Adam and SGD optimizers so as to determine their influence on convergence speed, final validation accuracy and loss.

Adam optimizer was achieved faster convergence, higher validation accuracy and lower final loss as compared to SGD. On the contrary, SGD had to utilize a large number of epochs and the appropriate tuning of hyperparameters (such as learning rate and momentum) in order to have solving performance which was acceptable. Hence, Adam was more efficient and effective for this deeper CNN model in the present given fruit image classification task. (Abadi, 2015)

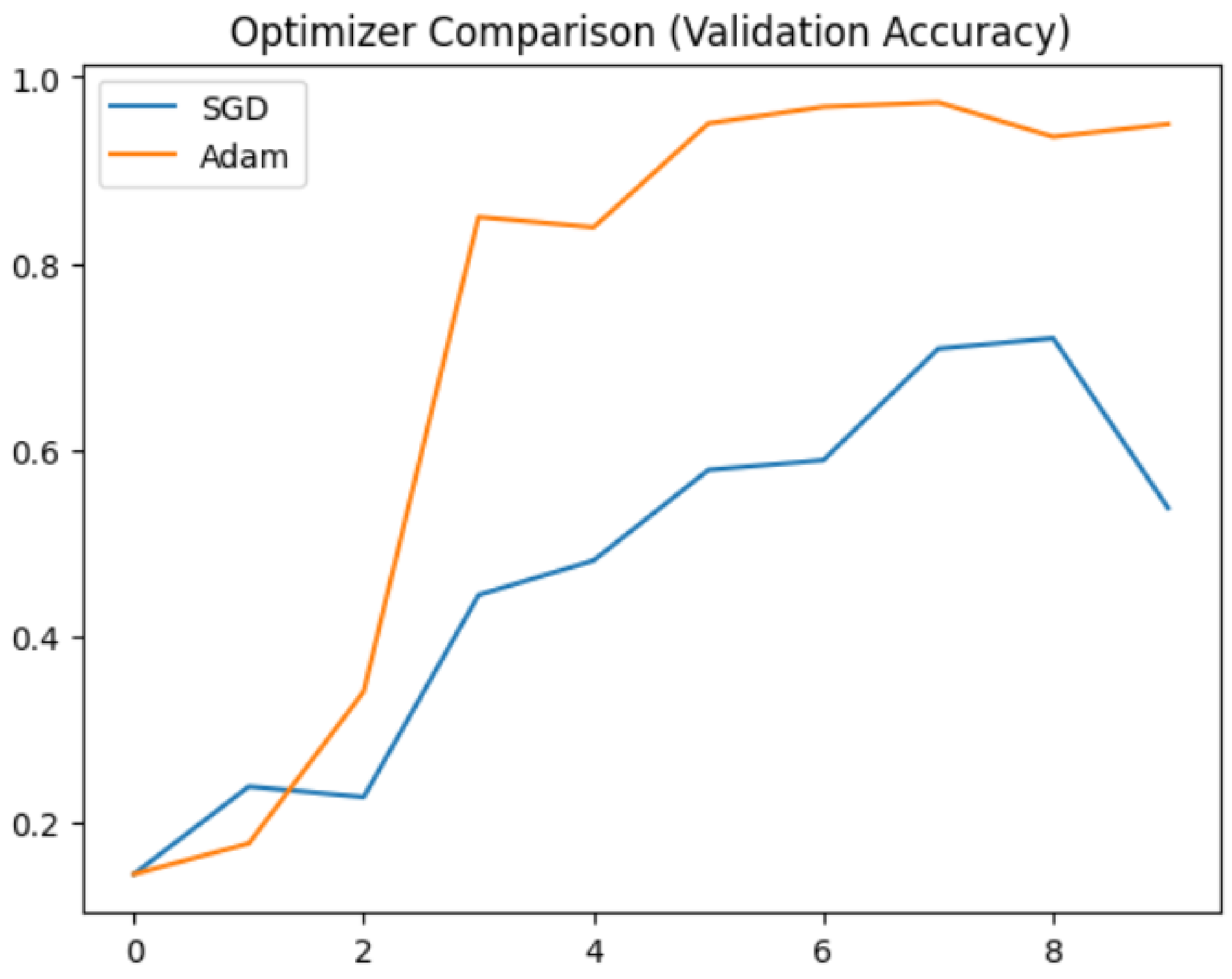


Figure 6: Optimizer Comparison

4.4 Challenges in Training

- **Overfitting:** First detected in the deeper CNN at the initial stages of learning and particularly with the default hyperparameters. However, with tuned dropout and regularization, the overfitting was minimized significantly with the good validation performance.

- **Data Imbalance:** In minority classes such as Banana, Grapes, and Peach, a small nudging down of performance was observed as observed through lower precision and recall scores on evaluation.

- **Training Time: Baseline CNN:** ~8 minutes (10 epochs), Deeper CNN: ~12 minutes (10 epochs)

- **Unstable Training:** When SGD was used without appropriate learning rate scheduling and weight initialization, the deeper model went through unstable validation loss and slow convergence. This was to some extent tempered by the use of Adam which gave smoother and more stable training.

5. Fine-Tuning or Transfer Learning

Model: VGG16 (ImageNet weights).

Feature Extraction: Trained custom layers, 128,256 units.

Result: 94.13% train, 97.05% validation accuracy, 0.0889 validation loss.

Fine-Tuning: Defrosted the last 4 layers, adding Adam (0.00001) to train.

Result: 96.85% train, 99.13% validation accuracy, 0.0479 validation loss.

Classification Report: Precision: 0.99 (weighted avg)

Recall: 0.99

F1-score: 0.99

Comparison: Pre-trained features boosted VGG16's performance in attaining a higher accuracy (95.10%) than baseline and deeper models (93.42%).

6. Conclusion and Future Work

The VGG16 model had the best accuracy of 99.13% followed by the baseline (95.10%) and deeper (93.42%) models. Transfer learning shined with pre-trained features, while the baseline model had efficiency and performance in equilibrium. Adam outperformed SGD in convergence. Weak points include class imbalance as well as overfitting deeper models. Future work includes:

Hyperparameter Tuning: Optimize learning rates, dropout.

Additional Data: Address imbalance, improve quality.

New Models: Test ResNet, EfficientNet.

Applications: Explore real-time classification.

Limitations

- The dataset was of a small size and class imbalance impacted the performance on minority classes such as Cherry and Grapes.
- Transfer learning was only experimented with a single pretrained model (VGG16).
- Data enhancement was few, which may adversely affect diversity and generalization.
- Testing on the test set was very insignificant (5 images) thus broader testing has to be implemented for effective deployment.

Future Work

- Use advanced data augmentation (for example, rotation, brightness shift, zoom), stronger regularization techniques.
- Experiment other pre-trained models, like EfficientNet, InceptionV3, and ResNet50, to benchmark how their transfer learning performance can match or even beat VGG16 for higher accuracy and efficiency in fruit classification.

- Apply ensemble methods to aggregate the model predictions to enhance robustness of classification.

- Extend the dataset with more images for each class, and better variation of images in all categories.

- Play with attention mechanisms, Capsule Networks or hybrid CNN-RNN designs to achieve further classification accuracy and interpretability.

References

Abadi, M. A. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. *Software documentation*, 18.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. In I. B. Goodfellow, *Deep learning* (p. 800 (total)). MIT Press.