

CSY3025

**Artificial Intelligence
Techniques**

Assignment 2

Joshua Luke Perry (21427656)

Table of Contents

1 Introduction.....	6
2 Dataset Selection.....	6
2.1 Dataset Summary.....	6
2.2 Suitability for Image Classification.....	6
2.3 Real World Application.....	7
3 Model Selection.....	7
4 Preprocessing.....	8
5 Model Implementation.....	8
5.1 Input Layer.....	8
5.2 Augmentation Layer.....	8
5.3 Base Model.....	9
5.4 Globabl Average Layer.....	9
5.5 Dropout Layer.....	9
5.6 Output Layer.....	9
5.7 Compilation.....	10
5.7.1 Class Name Hot Encoding.....	10
5.7.2 Optimizer Function.....	10
5.7.3 Loss Function.....	10
5.7.4 Accuracy Metric.....	10
6 Training.....	11
6.1 Model Iterations.....	11
6.1.1 Initial Model.....	11
6.1.2 Increase Batch Size.....	11
6.1.3 Increase Epochs.....	12
6.1.3.1 Learning Rate Callback.....	12
6.1.3.2 Early Stopping Callback.....	12
6.2 Results.....	13
6.2.1 Validation Evaluation.....	13
6.2.1.1 Initial Model.....	13
6.2.1.2 Increase Batch Size.....	13

6.2.1.3 Increase Epochs.....	13
6.2.1.4 Increase Test Data Size.....	14
6.2.2 Testing Evaluation.....	14
6.2.2.1 Initial Model.....	14
6.2.2.2 Increase Batch Size.....	15
6.2.2.3 Increase Epochs.....	15
6.2.2.4 Increase Test Data Size.....	15
7 Conclusion.....	15
References.....	16
Appendix 1 – Categories.....	17
1.1 Crops and their diseases.....	17
2.1 Class names.....	18

Index of Tables

Table 1: The hyperparameters used in the initial attempt at model training	11
Table 2: The hyperparameters used in the second attempt at model training.....	11
Table 3: The hyperparameters used in the final attempt at model training	12
Table 4: The crops and their diseases that are represented in the dataset	18

Table of Figures

Figure 1: Results from the validation evaluation for the initial model.....	13
Figure 2: Results from the validation evaluation for the increased batch size model.....	13
Figure 3: Results from the validation evaluation for the increase epochs model.....	13
Figure 4: Results from the validation evaluation for the increase test size model.....	14
Figure 5: Results from the test evaluation for the initial model.....	14
Figure 6: Results from the test evaluation for the increase batch size model.....	15
Figure 7: Results from the test evaluation for the increase epochs model	15
Figure 8: Results from the test evaluation for the increase test size model	15

1 Introduction

2 Dataset Selection

To find a dataset, the website Kaggle was used. Kaggle is a repository of datasets posted by the companies and the public. It also provides a usability score that can be an indication of how suitable the dataset is to machine learning tasks.

The dataset that was selected was the New Plant Disease Dataset (Bhattarai, n.d.). Originally gathered by github user spMohanty by scraping the PlantVillage Crop Library website. PlantVillage is “the largest open access library on crop health in the world” (Anon, n.d.)

2.1 Dataset Summary

The dataset contains 87,000 RGB images of both healthy and diseased crops. All images are sized at 256x256. The dataset is split into 38 classes (See Appendix 2), each of which has approximately 2000 images.

2.2 Suitability for Image Classification

The dataset is relevant to the task. All class names clearly state the crop and its state of health. These names are also relevant to the images in the dataset which show the crops in a diverse range of contexts.

The large size of the dataset increases model's ability to learn detailed features, helps model's ability to generalize better. Comparable to benchmark datasets in size (Citation Needed).

2.3 Real World Application

This model could be applied to the agricultural industry to improve crop management. The model could help with early detection of disease to help

reduce the overall loss for each harvest. Furthermore, identification of the specific disease can help with the treatment of these crops.

The model could also be used by hobbyists and small farms to reduce the barrier of entry for the industry. The model achieves this by allowing users to make informed decisions surrounding their crops.

3 Model Selection

MobileNetV3 is a CNN, developed by Google, that is designed to be used on mobile and embedded hardware (Citation Needed). CNNs are particularly well suited for image classification tasks (Citation Needed). For this dataset, the ability for layers to share weights across different spatial locations is particularly helpful as it will allow for feature detection regardless of its position in the image (Citation Needed). Since the diseases on the crops can show anywhere on the crop, this should help with disease detection.

MobileNetV3 is able to run on resource limited hardware due to the use of several techniques such as: separable convolutions; inverted residuals; and linear bottlenecks. These techniques all reduce the computational cost required (Citation Needed).

Since the model can run on this type of hardware it makes it perfect for vision applications for IoT and instances where running the tasks remotely is not an option.

To achieve the required classification task, this model will be used in conjunction with a classifier head achieved through transfer learning. This approach has been chosen as it will allow the utilization of the model's key features (Such as resource efficiency, generalisation, and reduced overfitting) while also specialising for the specific task required (Citation Needed). Furthermore, since fewer parameters are required to be trained, it will reduce the overall training time.

4 Preprocessing

The MobileNetV3 model integrates the preprocessing into the model as a rescaling layer. This normalises the pixels to values in the range of -1 to 1 (Citation Needed).

Due to the convolutional nature of the base model, the images do not need to be resized even though the input for the model is smaller (Citation Needed).

5 Model Implementation

This section outlines how the model was implemented layer by layer (excluding the layers within the base model). This section also covers the functions used during compilation.

5.1 Input Layer

This layer is the entry point to the model, it accepts a 160x160 image with 3 color channels.

5.2 Augmentation Layer

This layer augments the data to increase the robustness of the model. The data is augmented by passing through augmentation layers sequentially. The first layer is a RandomFlip layer. This layer has a 50% chance to flip each image across the horizontal axis. After this, each image is passed through a RandomRotation layer. This layer rotates each image a random amount in the range of ± 0.2 radians. This augmentation process will increase the variation in the data, which will help reduce overfitting by allowing the model to learn more diverse patterns (Citation Needed).

5.3 Base Model

The augmented data is then fed into the MobileNetV3Large model. The model is initialised without the classification layers and with the pre-trained weights for

the ImageNet dataset. The classification layers are not needed because they will be replaced with the classifier head created for this dataset.

5.4 Global Average Layer

This layer is used as it has several features that can help in image classification. Firstly, this layer produces a fixed-length feature vector. This makes the model invariant to spatial translations or variations in the image input size (Citation Needed). Secondly, this layer can reduce the number of parameters in subsequent layers while retaining important feature information, helping to reduce overfitting (Citation Needed). Finally, this layer allows the existing base model used to adapt to new datasets during transfer learning (Citation Needed).

5.5 Dropout Layer

This layer is used to help prevent overfitting. It achieves this by randomly setting some of the input neurons to 0. The probability of this is 20%. This helps to stop the model from relying too much on specific features which can help improve generalisation (Citation Needed).

5.6 Output Layer

The output layer for the classifier is a Dense layer. These are commonly used as output layers due to their ability to adapt to different types of tasks. For multi-class classification, a Dense layer can have an output neuron for each class name. By applying the softmax activation function, the output score for each class is converted into the probability the input fits that class. This is useful in multi-class classification because it allows interpretation of the uncertainty of the predictions.

5.7 Compilation

This section describes the functions and metrics used for the compilation of the model.

5.7.1 Class Name Hot Encoding

The tensorflow library normally hot encodes the class names during the import process. However, this did not happen during the compilation of this model. Therefore, the class names were hot encoded using a lambda function. Hot encoding converts non-numerical data into a numerical format. This is important as it allows the model to more easily process the information. Furthermore, loss functions normally rely on numerical information to be accurate (Citation Needed).

5.7.2 Optimizer Function

The model was compiled using the Adam optimizer function. This function combines the advantages of both the AdaGrad and RMSProp algorithms allowing it to offer both performance and efficiency.

5.7.3 Loss Function

The loss function used for this model was CategoricalCrossEntropy. This function is specifically designed for multi-class classification tasks. Furthermore, when used in conjunction with the softmax activation in the final layer, this ensures that the sum of the outputs is 1 (Citation Needed).

5.7.4 Accuracy Metric

To measure accuracy the CategoricalAccuracy function was used. This function is particularly effective for datasets with balanced class. This function was also chosen for its ease of interpretability; the higher the accuracy, the better the model is at predicting the class of an input. This is simply calculate as the ratio of correct prediction to total predictions.

6 Training

The model was trained using a GPU to decrease training time. An autotuned prefetching routine was used to prevent I/O blocking.

6.1 Model Iterations

The training for the model was modified several times during development to maximise the accuracy of the model.

6.1.1 Initial Model

Hyperparameter	Value
Batch Size	32
Image Size	160x160
Learning Rate	0.0001
Epochs	10

Table 1: The hyperparameters used in the initial attempt at model training

6.1.2 Increase Batch Size

Hyperparameter	Value
Batch Size	64
Image Size	160x160
Learning Rate	0.0001
Epochs	10

Table 2: The hyperparameters used in the second attempt at model training

6.1.3 Increase Epochs

Hyperparameter	Value
Batch Size	64
Image Size	160x160
Learning Rate	0.0001
Epochs	50

Table 3: The hyperparameters used in the final attempt at model training

Alongside the alterations to the hyperparameters, callback functions were employed to enhance the training process.

6.1.3.1 Learning Rate Callback

The ReduceLROnPlateau callback was used to dynamically adjust the learning rate in the event the loss value plateaus. The callback reduces the learning rate by a factor of 0.1 if the loss does not improve for 5 consecutive epochs. The minimum value for the learning rate that was allowed was 0.000001.

6.1.3.2 Early Stopping Callback

This callback reduces the risk of overfitting. This stops the model training if the model's loss stops improving for 10 consecutive epochs. The callback also restores the model weights to the state they were in in the best epoch.

6.2 Results

This section contains the accuracy and loss metrics for each iteration of the model after being evaluated with the validation data and the test data.

6.2.1 Validation Evaluation

Each iteration of the model was evaluated against the validation data after being trained.

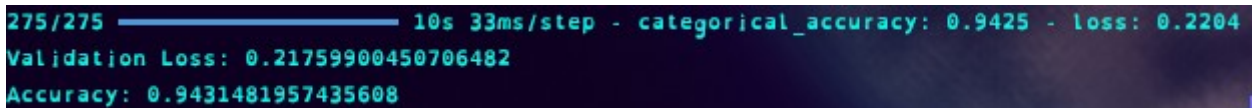
6.2.1.1 Initial Model



```
550/550 ————— 11s 18ms/step - categorical_accuracy: 0.9514 - loss: 0.1830  
Validation Loss: 0.18300175666809082  
Accuracy: 0.9504894018173218
```

Figure 1: Results from the validation evaluation for the initial model

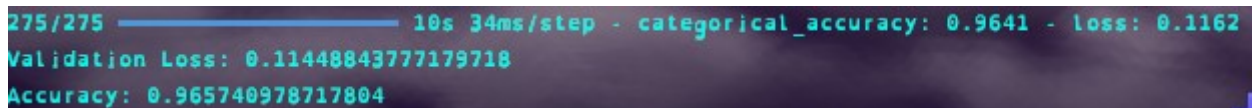
6.2.1.2 Increase Batch Size



```
275/275 ————— 10s 33ms/step - categorical_accuracy: 0.9425 - loss: 0.2204  
Validation Loss: 0.21759900450706482  
Accuracy: 0.9431481957435608
```

Figure 2: Results from the validation evaluation for the increased batch size model

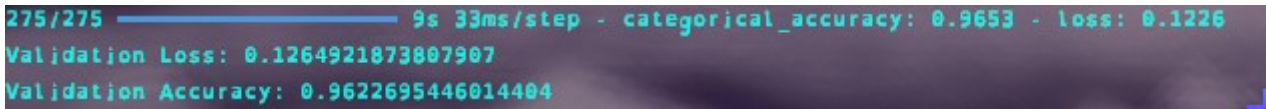
6.2.1.3 Increase Epochs



```
275/275 ————— 10s 34ms/step - categorical_accuracy: 0.9641 - loss: 0.1162  
Validation Loss: 0.11448843777179718  
Accuracy: 0.965740978717804
```

Figure 3: Results from the validation evaluation for the increase epochs model

6.2.1.4 Increase Test Data Size



```
275/275 ————— 9s 33ms/step - categorical_accuracy: 0.9653 - loss: 0.1226  
Validation Loss: 0.1264921873807907  
Validation Accuracy: 0.9622695446014404
```

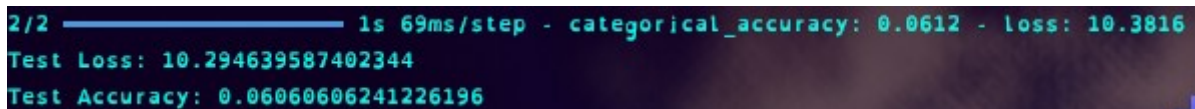
Figure 4: Results from the validation evaluation for the increase test size model

6.2.2 Testing Evaluation

Each iteration of the model was evaluated against the test data after being trained. The dataset came pre split into training, testing, and validation data. It was not until the testing phase that it was realised that the test data was insufficient for the task. The test data only consisted of 33 images. In comparison to the size of the dataset as a whole, this was not a representative sample.

Since the first 3 iterations of the model were already trained using the preexisting splits, a fourth iteration was produced a test data from the training data. The validation data was approximately 20% of the dataset. With this in mind, the size of the test data sample was set to the same size so that the data would have a 60/20/20 split. The model was then retrained with the same hyperparameters as the third iteration (See Section 6.1.3).

6.2.2.1 Initial Model



```
2/2 ————— 1s 69ms/step - categorical_accuracy: 0.0612 - loss: 10.3816  
Test Loss: 10.294639587402344  
Test Accuracy: 0.06060606241226196
```

Figure 5: Results from the test evaluation for the initial model

6.2.2.2 Increase Batch Size

```
1/1 — 1s 758ms/step - categorical_accuracy: 0.0606 - loss: 9.4185  
Test Loss: 9.418469429016113  
Test Accuracy: 0.06060606241226196
```

Figure 6: Results from the test evaluation for the increase batch size model

6.2.2.3 Increase Epochs

```
1/1 — 1s 682ms/step - categorical_accuracy: 0.0606 - loss: 13.5540  
Test Loss: 13.554000854492188  
Test Accuracy: 0.06060606241226196
```

Figure 7: Results from the test evaluation for the increase epochs model

6.2.2.4 Increase Test Data Size

```
275/275 — 12s 33ms/step - categorical_accuracy: 0.9608 - loss: 0.1271  
Test Loss: 0.1278374344110489  
Test Accuracy: 0.9604761004447937
```

Figure 8: Results from the test evaluation for the increase test size model

7 Future Work

To improve this model in the future, finetuning could be applied to the top layers of the base model. This could further increase the accuracy of the model.

The dataset is also quite old and since the original scraping of the data the website has continued to grow it's database. This could provide opportunity for more diverse data and potentially more classes that could be learnt to be classified.

References

Anon PlantVillage. *Plant Village*. [online]. Available from: <https://plantvillage.psu.edu/projects> [Accessed May 18, 2024].

Bhattarai, S. New Plant Diseases Dataset. Kaggle. [online]. Available from: <https://www.kaggle.com/datasets/vipooooool/new-plant-diseases-dataset> [Accessed May 18, 2024].

Appendix 1 – Categories

The dataset was split into 38 categories. This consists of 14 different crops and their accompanying states of health.

1.1 Crops and their diseases

Crop	Disease (or Healthy)
Apple	Healthy
	Black Rot
	Cedar Apple Rust
Blueberry	Healthy
Cherry (including sour)	Healthy
	Powdery Mildew
Corn (Maize)	Healthy
	Cecospora Leaf Spot (Gray Leaf Spot)
	Common Rust
	Northern Leaf Blight
Grape	Healthy
	Black Rot
	Esca (Black Measles)
	Leaf Blight (Isariopsis Leaf Spot)
Orange	Haunglongbing (Citrus Greening)
Peach	Healthy
	Bacterial Spot
Bell Pepper	Healthy
	Bacterial Spot
Potato	Healthy
	Early Blight
	Late Blight
Raspberry	Healthy
Soybean	Healthy
Squash	Powdery Mildew
Strawberry	Healthy
	Leaf Scorch

Crop	Disease (or Healthy)
Tomato	Healthy
	Bacterial SPot
	Early Blight
	Late Blight
	Leaf Mold
	Septoria Leaf Spot
	Spider Mites (Two-spotted Spider Mite)
	Target Spot
	Yellow Leaf Curl Virus
	Mosaic Virus

Table 4: The crops and their diseases that are represented in the dataset

2.1 Class names

Below is a list of the class names as found in the dataset's directory structure.

Apple__Apple_scan

Apple__Black_rot

Apple__Cedar_apple_rust

Apple__healthy

Blueberry__healthy

Cherry_(including_sour)__Powdery_mildew

Cherry_(including_sour)__healthy

Corn_(maize)__Cercospora_leaf_spot Gray_leaf_spot

Corn_(maize)__Common_rust_

Corn_(maize)__Northern_Leaf_Blight

Corn_(maize)__healthy

Grape__Black_rot

Grape__Esca_(Black_Measles)

Grape__Leaf_blight_(Isariopsis_Leaf_Spot)

Grape__healthy

Orange__Haunglongbing(Citrus_greening)

Peach_Bacterial_spot

Peach_healthy

Pepper,_bell__Bacterial_spot

Pepper,_bell__healthy

Potato__Early_blight

Potato__Late_blight

Potato__healthy

Raspberry__healthy

Soybean__healthy

Squash__Powdery_mildew

Strawberry__Leaf_scorch

Strawberry__healthy

Tomato__Bacterial_spot

Tomato__Early_blight

Tomato__Late_blight

Tomato__Leaf_Mold

Tomato__Septoria_leaf_spot

Tomato__Spider_mites Two-spotted_spider_mite

Tomato__Target_Spot

Tomato__Tomato_Yellow_Leaf_Curl_Virus

Tomato__Tomato_mosaic_virus

Tomato__healthy