



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Application of Machine Learning

Student Name: Suhaana Kapali

Student ID: 2538421

MSc Data and Artificial Intelligence

Module: L7 Application of Machine Learning

Module Code: ACCA7025

Lecturer: Dr. Aliyu Sambo

Table of Contents

1. Introduction.....	4
1.1 Background of Artificial Neural Networks	4
1.2 Convolutional Neural Networks and Recurrent Neural Networks	4
1.3 Plant Pathology Dataset.....	5
1.4 Aims and Objective.....	5
2. Literature Review.....	5
2.1 Artificial Neural Networks for Image Classification.....	5
2.2 Convolutional Neural Networks for Plant Disease Detection	6
2.3 Transfer Learning and Model Optimization	7
2.4 Research Gap.....	7
3. Data Description and Data Preparation	8
3.1 Dataset Overview	8
3.2 Data Preprocessing.....	9
3.3 Data Augmentation	9
4. Methodology and Implementation.....	10
4.1 Model Development.....	10
4.2 Model Architecture	11
4.3 Model Training	12
4.4 Evaluation Metrics	14
5. Results and Evaluation	14
5.1 Training Performance.....	14
5.2 Test Performance	15
5.3 Confusion Matrix.....	15
5.4 Learning Curves	16
6. Discussion	17
6.1 Interpretation of Results	17
6.2 Strengths and Limitations	17
6.3 Recommendations and Future Work	17
7. Conclusion	18
References.....	19

Abstract

Early detection of plant diseases is important for the improvement of crop health and minimization of agricultural losses. Deep learning has been shown to be a realistic approach for separate plant disease recognition using image classification algorithms. In this work, we developed and evaluated a transfer learning model utilizing EfficientNet-B0 for classifying 12 categories of apple leaf diseases on the Plant Pathology 2021 dataset. Image preprocessing, data augmentation, and transfer learning were applied to improve model performance and reduce overfitting. The model was trained and tested on different parts of the dataset, and the performance was evaluated on a different test set, utilizing classification accuracy and confusion matrix analysis. The suggested model achieved 66.29% test accuracy which indicates the potential of the model to classify apple leaf diseases with reasonable accuracy and computational efficiency. Although EfficientNet-B0 showed the potential of automated plant disease classification, several visually similar disease categories were still difficult to distinguish. Future works should be based on larger datasets and complex deep learning architectures to achieve considerable improvements in classification accuracy and generalization of the model.

Keywords: Plant pathology 2021, Artificial neural networks, Transfer learning, Image classification, Efficientnet-b0.1.

1. Introduction

1.1 Background of Artificial Neural Networks

Artificial Neural Networks ANNs are a class of machine learning algorithms that are based on the structure and function of biological neurons in the human brain. They contain of connected artificial neurons organized in various layers that process information and learn patterns from the input data. The training of the network involves adjusting the weights of its connections in a way that allows it to learn to recognize complicated relationships and to increase the accuracy of its predictions. Unlike standard machine learning methods, which rely on handcrafted features, ANNs can acquire meaningful representations straight from raw data. These have made them quite useful in solving classification, prediction, and pattern recognition challenges [1].

Recent breakthroughs in computer power, graphics processing units (GPUs) and the availability of massive datasets have led to dramatic improvements in the performance of deep neural networks. Thus, ANNs have been one of the most utilized approaches in the domains of computer vision, healthcare, agriculture, autonomous systems, and natural language processing [2].

1.2 Convolutional Neural Networks and Recurrent Neural Networks

To visualize prediction outcomes for all five disease categories, a confusion matrix was created. The correctly classified pictures are the diagonal elements, and the misclassified samples are the off-diagonal elements. A greater concentration along the diagonal implies that the classification task was performed well. At the same time, the matrix gives the most common pairwise misjudgments of the class labels [3].

Since visual symptoms of many plant diseases are quite alike, the model is also likely to make classification errors from time to time. The analysis of the prediction behavior shows valuable information on the characteristics and shortcomings of the designed CNN model and the possibilities for development [4].

1.3 Plant Pathology Dataset

The Plant Pathology collection shows thousands of captioned images of apple leaves in a variety of environmental settings [5]. In this study, twelve disease categories were used to train and evaluate the EfficientNet-B0 model. These categories include healthy leaves, individual disease classes, and complex disease combinations, providing a more comprehensive and challenging classification task. Before model training, all images experienced preprocessing including resizing, normalization, and data augmentation, to improve model generalization, reduce overfitting, and ensure compatibility with the EfficientNet-B0 architecture [6].

```
... Found 8691 files belonging to 12 classes.  
Found 1862 files belonging to 12 classes.  
Found 1863 files belonging to 12 classes.  
['complex', 'frog_eye_leaf_spot', 'frog_eye_leaf_spot complex', 'healthy', 'powdery_mildew', 'powdery_mildew complex', 'rust', 'rust  
Number of Classes = 12
```

1.4 Aims and Objective

The objective of this project is to design, deploy, and assess a Convolutional Neural Network for precise classification of apple leaf diseases based on the Plant Pathology 2021 dataset. The aims of this project are to review the existing literature on neural networks for image classification, discuss the differences between CNNs and RNNs, prepare the dataset using appropriate preprocessing techniques, build and optimize a CNN model using TensorFlow, evaluate its performance using standard classification metrics, and critically discuss the effectiveness and limitations of the proposed approach in plant disease recognition.

2. Literature Review

2.1 Artificial Neural Networks for Image Classification

Artificial Neural Networks are one of the most important advances in modern artificial intelligence because they can learn complex correlations directly from data. ANNs, compared to traditional machine learning approaches depending on manually extracted features, acquire important patterns during training through interconnected layers of neurons. Deep learning has enhanced this capability by introducing several hidden layers

that teach low and high level information iteratively. This has led to huge gains in performance for identifying images, object identification, medical diagnostics, and autonomous systems [7] [8].

Deep learning has been quite good at classifying images. In image classification, neural networks look at the pixel information to recognize visual traits that differentiate one category from another. The network will learn more complex attributes with each layer as it analyzes the photographs. This allows the network to correctly classify photos even when the lighting, orientation, or background conditions are different. Deep neural networks have proved to consistently outperform standard image processing approaches, as they can automatically learn hierarchical visual representations [9].

2.2 Convolutional Neural Networks for Plant Disease Detection

Convolutional neural networks are the most suitable deep learning architecture for picture classification. CNNs use convolutional layers to learn important spatial information such as edges, textures, colors, and shapes while reducing the number of parameters to be learned. The completely linked layers are used to do final classification based on the acquired features. Pooling layers lower the computational complexity by keeping only the most critical information [10].

Studies have demonstrated that CNNs are useful in agricultural applications, especially in the detection of plant disease. Deep convolutional networks were used to detect illnesses in plants with excellent accuracy from leaf pictures of the Plant Village dataset. Their results showed that CNNs have the potential to reduce the requirement for manual feature engineering of images and to improve the automatic diagnosis of diseases [11]. Similarly, Too et al. studied several pre trained CNN architectures such as VGG16, ResNet50, DenseNet121, InceptionV3 and EfficientNet and found that transfer learning may greatly boost classification performance, as well as shorten training time and processing needs [12].

CNN based algorithms normally demonstrate decent classification accuracy, although there are reports of performance decrease in the case of datasets with visually identical

clinical symptoms or imbalanced classes. These limitations demonstrate that the performance of a model depends not only on the network design but also on the quality and distribution of training data [12].

2.3 Transfer Learning and Model Optimization

Training deep neural networks from scratch involves big, labeled datasets and huge computer resources. Transfer learning challenges this problem by leveraging models that have acquired useful visual features from huge datasets such as ImageNet. The pretrained model is used as a feature extractor instead of learning each feature from scratch, and only the last classification layers are retrained for the target job.

Transfer learning has various advantages. It minimizes the training time, reduces computing cost, and often leads to improved generalization, especially when the supplied dataset is very small. Many low level picture elements, including edges, textures, and color gradients, are everywhere across diverse image domains, which has made transfer learning useful in medical imaging, industrial inspection, and agricultural disease identification [12].

An important aspect is also the model optimization to improve the performance of neural networks. The hyperparameters such as learning rate, batch size, dropout rate, choice of optimizer, and number of training epochs all have direct impact on the convergence and generalization capacity of the model. Common techniques to reduce overfitting and preserve good classification accuracy include data augmentation, early halting, dropout regularization and learning rate scheduling. Careful tweaking of these parameters can help CNN models to be more stable on data they have not seen before [6].

2.4 Research Gap

Deep learning based plant disease categorization has made fast progress, but there are still many challenges to overcome. Most published research focuses on overall high accuracy with limited exploration on class specific performance, confusion among comparable illnesses, and impact of dataset imbalance. Furthermore, several research

studies evaluate their models on various datasets, which makes comparing the methods difficult [4].

The Plant Pathology dataset presents new problems. Many diseases share the same visual signs, which means a greater chance of misdiagnosis. Therefore, it is very important to analyze precision, recall, F1 score, and confusion matrix, in addition to total accuracy, to understand the performance of the model for each class [4].

This project aims to overcome these difficulties by creating a convolutional neural network using the Plant Pathology dataset as described in the evaluation brief. The model performance is assessed using several performance measures and the strengths and flaws of the model are critically reviewed. The results give a better knowledge of CNN-based image classification for automatic plant disease diagnosis and propose possible improvements for future research.

3. Data Description and Data Preparation

3.1 Dataset Overview

To train and evaluate the EfficientNet-B0 for apple leaf disease classification the Plant Pathology dataset was used. The dataset consists of labelled RGB images of apple leaves collected under a range of environmental conditions [13]. In this study, 12 disease categories were used, including healthy leaves, individual disease classes, and complex disease combinations, providing a comprehensive multi-class classification task.

To support model development and evaluation the dataset was divided into training, validation, and test subsets. Then the training set was used to train and fine tune the EfficientNet-B0 model, the validation set was used to monitor model performance and optimize hyperparameters, and the independent test set was used to evaluate the final model on previously unseen images.

Partition	Image Count
Training	8,691
Validation	1,862
Test	1,863
Total	12,416

3.2 Data Preprocessing

Before training, all images were preprocessed to ensure compatibility with the EfficientNet-B0 architecture. The images were resized to the required input dimensions of 224×224 pixels and pixel values were normalised using TensorFlow's preprocessing functions. The dataset was loaded using TensorFlow's image data pipeline, which generated labels based on the folder structure and organized the images into mini batches for efficient model training. These preprocessing steps ensured consistent input data and improved training efficiency [14].

3.3 Data Augmentation

Data augmentation techniques are one way of introducing variations of images to the training dataset in such a way that the images are still similar to the originals but not identical. In this study, several transformations were used to augment the image data for training, including random horizontal flipping of the images and small rotations and translations as well as zooming operations. All these changes have the effect of making the training dataset much more diverse without the need to collect any new images to add for the training set. The validation and test datasets were kept untouched to ensure the evaluation results reflect the model's actual performance with real world images that the model has not yet seen during training.

```
data_augmentation = tf.keras.Sequential([  
    layers.RandomFlip("horizontal"),  
    layers.RandomRotation(0.2),  
    layers.RandomZoom(0.2),  
    layers.RandomContrast(0.2),  
    layers.RandomTranslation(0.1,0.1),  
])
```

4. Methodology and Implementation

4.1 Model Development

In this study, an EfficientNet-B0 pre-trained convolutional neural network was used for the classification of apple leaf diseases. We selected EfficientNet-B0 due to its good tradeoff between classification accuracy and processing economy through a combination of scaling technique that simultaneously enhances network depth, width, and image resolution [15]. Instead of constructing a model from scratch, transfer learning was used, i.e. weights pre-trained on the ImageNet dataset were used. This strategy allows the model to leverage previously learned visual elements and adjust them in the plant disease classification challenge [16].

The EfficientNet-B0 convolutional base was loaded without its original classification layer (`include_top=False`) and initialised with ImageNet weights. In the first step of training, the convolutional layers were frozen to preserve the learnt features and only the newly added classification layers were trained on the Plant Pathology dataset.

```
base_model = EfficientNetB0(  
    include_top=False,  
    weights="imagenet",  
    input_shape=(224,224,3)  
)  
  
base_model.trainable = False  
  
print("EfficientNetB0 Loaded Successfully")
```

4.2 Model Architecture

The model accepted RGB images resized to 224×224 pixels, which is the required input dimension for EfficientNet-B0. The input images were first passed through a data augmentation pipeline and then preprocessed using TensorFlow's EfficientNet preprocessing function before feature extraction [15].

Dimensioning of the feature extraction stage was achieved using a global average pooling layer. Through this step, not only are the dimensions of the feature maps reduced, but the essence of the feature maps is kept. For the stabilization of the training process and the enhancement of model convergence, a batch normalization layer was added. A sequence of fully connected layers with 512 and 256 neurons that employed the ReLU activation function to learn the domain specific features was used. Dropout layers with rates of 0.5 and 0.3 respectively were introduced to fight the risk of overfitting, with deactivating the random neurons during the training phase. This way enhancing the model generalization. Finally, a SoftMax output layer generated the probability distribution across the 12 apple leaf disease categories, enabling the model to predict the most likely class for each input image [17].

```
inputs = tf.keras.Input(shape=(224,224,3))
x = data_augmentation(inputs)
x = tf.keras.applications.efficientnet.preprocess_input(x)
x = base_model(x, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.BatchNormalization()(x)
x = layers.Dense(512, activation="relu")(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.3)(x)
outputs = layers.Dense(len(class_names), activation="softmax")(x)
model = Model(inputs, outputs)
model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
sequential (Sequential)	(None, 224, 224, 3)	0
efficientnetb0 (Functional)	(None, 7, 7, 1280)	4,049,571
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1280)	0
batch_normalization (BatchNormalization)	(None, 1280)	5,120
dense (Dense)	(None, 512)	655,872
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 12)	3,084

Total params: 4,844,975 (18.48 MB)
 Trainable params: 792,844 (3.02 MB)
 Non-trainable params: 4,052,131 (15.46 MB)

4.3 Model Training

The model was trained with Adam optimiser with learning rate of 0.0001. The labels in the dataset are integer values, so Sparse Categorical Cross Entropy loss was utilized. The training was done with a batch size of 32 and validation was performed after each epoch.

Some callback procedures were implemented to improve training efficiency. EarlyStopping monitored on validation loss and best model restored if no improvement over 5 consecutive epochs. ReduceLROnPlateau would automatically lower the learning rate when the validation loss plateaued, allowing for finer optimisation in later training rounds. ModelCheckpoint saved the model with the highest validation accuracy; hence, we saved the best performing weights.

Following the initial training phase, the model was recompiled with a lower learning rate to further optimize the training prior to testing on the independent test dataset.

```

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

```

Stage 1 Training

During the first stage of training, the EfficientNet-B0 backbone remained frozen while only the newly added classification layers were trained using the Plant Pathology 2021 dataset. This enabled the model to learn task-specific features without modifying the pre-trained feature extractor. The Adam optimiser, together with EarlyStopping and ReduceLROnPlateau callbacks, helped the model converge efficiently while reducing the likelihood of overfitting. The epoch-by-epoch training log demonstrates the gradual improvement in training and validation performance as the classification head adapted to the dataset.

```
Epoch 1/20
272/272 — 0s 71ms/step - accuracy: 0.2874 - loss: 2.3246
Epoch 1: val_accuracy improved from None to 0.61547, saving model to best_model.keras

Epoch 1: finished saving model to best_model.keras
272/272 — 76s 125ms/step - accuracy: 0.3654 - loss: 1.9830 - val_accuracy: 0.6155 - val_loss: 1.3903 - learning_rate: 1.0000e-04
Epoch 2/20
272/272 — 0s 72ms/step - accuracy: 0.4797 - loss: 1.5926
Epoch 2: val_accuracy improved from 0.61547 to 0.65789, saving model to best_model.keras

Epoch 2: finished saving model to best_model.keras
272/272 — 24s 88ms/step - accuracy: 0.4922 - loss: 1.5441 - val_accuracy: 0.6579 - val_loss: 1.1322 - learning_rate: 1.0000e-04
Epoch 3/20
272/272 — 0s 73ms/step - accuracy: 0.5205 - loss: 1.4525
Epoch 3: val_accuracy improved from 0.65789 to 0.67562, saving model to best_model.keras

Epoch 3: finished saving model to best_model.keras
272/272 — 25s 90ms/step - accuracy: 0.5349 - loss: 1.4088 - val_accuracy: 0.6756 - val_loss: 1.0664 - learning_rate: 1.0000e-04
Epoch 4/20
272/272 — 0s 73ms/step - accuracy: 0.5571 - loss: 1.3691
Epoch 4: val_accuracy improved from 0.67562 to 0.68314, saving model to best_model.keras

Epoch 4: finished saving model to best_model.keras
272/272 — 25s 90ms/step - accuracy: 0.5658 - loss: 1.3445 - val_accuracy: 0.6831 - val_loss: 1.0309 - learning_rate: 1.0000e-04
Epoch 5/20
272/272 — 0s 73ms/step - accuracy: 0.5795 - loss: 1.3014
Epoch 5: val_accuracy improved from 0.68314 to 0.69119, saving model to best_model.keras

Epoch 5: finished saving model to best_model.keras
272/272 — 25s 92ms/step - accuracy: 0.5881 - loss: 1.2571 - val_accuracy: 0.6912 - val_loss: 0.9970 - learning_rate: 1.0000e-04
Epoch 6/20
272/272 — 0s 75ms/step - accuracy: 0.6006 - loss: 1.2256
Epoch 6: val_accuracy did not improve from 0.69119
272/272 — 25s 91ms/step - accuracy: 0.6003 - loss: 1.2381 - val_accuracy: 0.6912 - val_loss: 0.9814 - learning_rate: 1.0000e-04
Epoch 7/20
272/272 — 0s 76ms/step - accuracy: 0.6052 - loss: 1.2144
Epoch 7: val_accuracy improved from 0.69119 to 0.69764, saving model to best_model.keras

Epoch 7: finished saving model to best_model.keras
272/272 — 26s 94ms/step - accuracy: 0.6101 - loss: 1.1915 - val_accuracy: 0.6976 - val_loss: 0.9608 - learning_rate: 1.0000e-04
Epoch 8/20
272/272 — 0s 77ms/step - accuracy: 0.6136 - loss: 1.1557
Epoch 8: val_accuracy improved from 0.69764 to 0.70623, saving model to best_model.keras

Epoch 8: finished saving model to best_model.keras
272/272 — 26s 95ms/step - accuracy: 0.6128 - loss: 1.1609 - val_accuracy: 0.7062 - val_loss: 0.9527 - learning_rate: 1.0000e-04
Epoch 9/20
272/272 — 0s 80ms/step - accuracy: 0.6168 - loss: 1.1312
Epoch 9: val_accuracy did not improve from 0.70623
272/272 — 26s 96ms/step - accuracy: 0.6218 - loss: 1.1413 - val_accuracy: 0.7035 - val_loss: 0.9415 - learning_rate: 1.0000e-04
Epoch 10/20
272/272 — 0s 77ms/step - accuracy: 0.6304 - loss: 1.1112
Epoch 10: val_accuracy did not improve from 0.70623
272/272 — 25s 92ms/step - accuracy: 0.6304 - loss: 1.1220 - val_accuracy: 0.6998 - val_loss: 0.9440 - learning_rate: 1.0000e-04
Epoch 11/20
272/272 — 0s 77ms/step - accuracy: 0.6398 - loss: 1.1013
Epoch 11: ReduceLROnPlateau reducing learning rate to 1.9999999494757503e-05.
```

Stage 2 Fine-Tuning

After the pre-training, a few of EfficientNet-B0 base model's layers were reenabled for fine-tuning. With new weights and a learning rate of 0.00001, the model was recompiled, and the pre-training weights were slowly adjusted to not interfere with the learned low-level visual features. Through fine-tuning, the model was able to gain higher-level and more accurate representations of apple leaf diseases, and, at the same time, it learned how to adapt to new images, which had not been shown in the training phase.

```

Epoch 1/20
272/272 — 0s 87ms/step - accuracy: 0.4938 - loss: 1.4681
Epoch 1: val_accuracy did not improve from 0.71590
272/272 — 45s 116ms/step - accuracy: 0.5080 - loss: 1.4415 - val_accuracy: 0.6697 - val_loss: 1.0822 - learning_rate: 1.0000e-05
Epoch 2/20
272/272 — 0s 90ms/step - accuracy: 0.5469 - loss: 1.3538
Epoch 2: ReduceLROnPlateau reducing learning rate to 1.9999999494757505e-06.

Epoch 2: val_accuracy did not improve from 0.71590
272/272 — 29s 105ms/step - accuracy: 0.5536 - loss: 1.3416 - val_accuracy: 0.6729 - val_loss: 1.0562 - learning_rate: 1.0000e-05
Epoch 3/20
272/272 — 0s 87ms/step - accuracy: 0.5838 - loss: 1.2905
Epoch 3: val_accuracy did not improve from 0.71590
272/272 — 28s 102ms/step - accuracy: 0.5751 - loss: 1.2876 - val_accuracy: 0.6713 - val_loss: 1.0571 - learning_rate: 2.0000e-06
Epoch 4/20
272/272 — 0s 87ms/step - accuracy: 0.5677 - loss: 1.2946
Epoch 4: ReduceLROnPlateau reducing learning rate to 1e-06.

Epoch 4: val_accuracy did not improve from 0.71590
272/272 — 28s 102ms/step - accuracy: 0.5780 - loss: 1.2771 - val_accuracy: 0.6735 - val_loss: 1.0449 - learning_rate: 2.0000e-06
Epoch 5/20
272/272 — 0s 88ms/step - accuracy: 0.5911 - loss: 1.2355
Epoch 5: val_accuracy did not improve from 0.71590
272/272 — 28s 104ms/step - accuracy: 0.5834 - loss: 1.2553 - val_accuracy: 0.6729 - val_loss: 1.0393 - learning_rate: 1.0000e-06
Epoch 5: early stopping
Restoring model weights from the end of the best epoch: 1.
59/59 — 6s 99ms/step - accuracy: 0.6629 - loss: 1.0459
Test Accuracy: 0.6629092693328857

```

4.4 Evaluation Metrics

The model performance was assessed using both overall and class level metrics. Overall accuracy is the proportion of pictures correctly classified in the test set. A more extensive analysis was performed by calculating the precision, recall and F1-score for each disease class.

A confusion matrix was also built to show the prediction performance for the twelve disease categories. The diagonal numbers are the correctly classified photographs and the off diagonal values are the misclassifications between disease categories. These evaluation metrics provide a comprehensive evaluation of the classification power of a model and its ability to discriminate across visually same plant diseases.

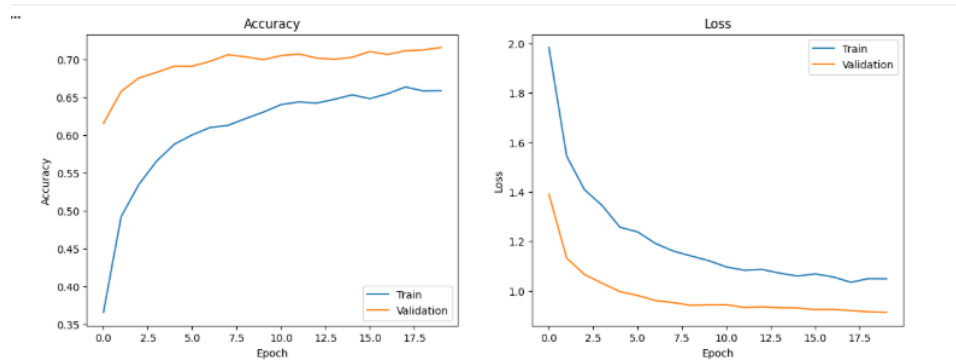
5. Results and Evaluation

5.1 Training Performance

The Plant Pathology dataset was preprocessed and augmented as previously reported, and the EfficientNet-B0 model was trained. During training, the model gradually increased training and validation accuracy, which indicated that the model had learnt important visual elements from the apple leaf images. The training and validation loss were correspondingly decreased, which proved the consistent convergence of the model in the learning process.

To increase the training efficiency, EarlyStopping, ReduceLROnPlateau and ModelCheckpoint callbacks were included. EarlyStopping stopped needless training

when validation performance stopped increasing, and ReduceLROnPlateau automatically decreased the learning rate when validation loss stopped improving. ModelCheckpoint preserved the model with best validation accuracy for examination later.



5.2 Test Performance

After training, the final model was evaluated using the independent test dataset containing 1,863 unseen images. This dataset was not used during training or validation, ensuring an unbiased assessment of the model's classification performance.

The EfficientNet-B0 model achieved a test accuracy of 66.29%, demonstrating its ability to classify the twelve apple leaf disease categories using transfer learning.

Metric	Result
Test Accuracy	66.29
Test Image Size	1,863

5.3 Confusion Matrix

A confusion matrix was generated to evaluate the classification performance across the 12 apple leaf disease categories. Diagonal values represent correctly classified images, whereas the off diagonal values indicate misclassifications between classes. Overall, the confusion matrix demonstrates that the EfficientNet-B0 model correctly classified a large proportion of the test images. However, some misclassifications occurred between disease categories with similar visual characteristics, particularly those involving

overlapping symptoms or multiple disease combinations. These results suggest that although the model learned meaningful discriminative features, distinguishing between visually similar classes remains challenging. Nevertheless, the confusion matrix provides valuable insights into class level performance and highlights opportunities for further improvements through additional training data, enhanced augmentation techniques, or more advanced deep learning architectures.



5.4 Learning Curves

The learning curves indicate the changing state of the model during each training step. The performance of the model on training and validation datasets increased, and the loss decreased over time, a clear sign that the model is effectively learning from the training data. A small difference between training and validation curves suggests that the combination of transfer learning, data augmentation, and regularization techniques had limited the effect of overfitting and enhanced the model's ability to generalize.

6. Discussion

6.1 Interpretation of Results

The transfer learning model based on EfficientNet-B0 achieved good performance on the classification of 12 types of apple leaf diseases. A mixture of transfer learning, data augmentation, and fine tuning was used to learn important visual features and to minimize the risk of overfitting [18]. The training and validation findings revealed that a consistent increase was achieved during the learning process, and the final model reached a test accuracy of 66.29%, suggesting that it generalized reasonably well to unseen images.

The categorization assignment was more challenging, as 12 illness categories were included with many classes of disease combinations. Thus, several superficially identical disease categories were sometimes misclassified, as shown in the confusion matrix. Thus, overlapping symptoms between various illnesses make them more difficult to differentiate, especially when many diseases occur on the same leaf.

To summarize, the data indicate that EfficientNet-B0 is a robust and computationally efficient method for automated plant disease classification while also identifying areas for further enhancement.

6.2 Strengths and Limitations

A major strength of this study is the use of EfficientNet-B0 transfer learning, which lowered training time while still achieving high classification performance. Data augmentation and dropout also aided in improving generalization and reducing over fitting.

The model still confused visually comparable diseases, which led to overall accuracy. Furthermore, the dataset was limited in size and diversity, suggesting the model may not generalize as well to photographs taken in diverse environmental situations.

6.3 Recommendations and Future Work

In future studies, the dataset could be enlarged with more photographs taken in different environmental conditions to make the model more robust. We could also improve classification accuracy by looking at larger EfficientNet variations, by modifying hyper-parameters or by investigating different architectures such as Vision Transformers [19].

Furthermore, the trained model can be deployed as a mobile or web application to help farmers and agricultural experts in real plant disease diagnosis.

7. Conclusion

The present study developed and evaluated an EfficientNet-B0 transfer learning model for the classification of 12 apple leaf disease categories using the Plant Pathology dataset. Image preprocessing, data augmentation, transfer learning, and fine tuning contributed to improved model performance and computational efficiency, resulting in a test accuracy of 66.29%.

Although some visually similar disease categories, particularly disease combination classes, remained challenging to distinguish, the overall results demonstrate that EfficientNet-B0 is a promising approach for automated plant disease classification. Future research should explore larger and more diverse datasets, advanced deep learning architectures, and further hyperparameter optimization to improve classification accuracy and model generalization.

References

- [1] I. Goodfellow, Y. Bengio and A. Courville, "Deep Learning Book," MIT Press, 18 November 2016. [Online]. Available: <https://www.deeplearningbook.org>. [Accessed 12 July 2026].
- [2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, 2015.
- [3] L. Y. Edna Chebet Too, "A comparative study of fine-tuning deep learning models for plant disease identification," *Computers and Electronics in Agriculture*, vol. 161, 2019.
- [4] Y.-S. K. Mohammad Mahmudur Rahman, "Transfer learning with deep convolutional neural networks for plant disease classification," *Computers and Electronics in Agriculture*, vol. 208, 2023.
- [5] B. G. Colomer, "Plant Pathology 2021 FGVC8: A Dataset for Apple Foliar Disease Classification," *Pattern Recognition Letters*, vol. 161, pp. 161-169, 2022.
- [6] C. Shorten, "A Survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.
- [7] Y. B. Yann LeCun, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.
- [8] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85-117, 2015.
- [9] I. S. G. H. Alex Krizhevsky, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017.
- [10] Y. B. Y. & H. G. LeCun, "Deep Learning," *Nature*, 2015.
- [11] S. P. H. D. P. & S. M. Mohanty, "Using Deep Learning for Image-Based Plant Disease Detection," *Frontiers in Plant Science*, 2016.
- [12] E. C. L. Y. N. S. & L. Y. Too, "A Comparative Study of Fine-Tuning Deep Learning Models for Plant Disease Identification," *Computers and Electronics in Agriculture*, 2019.
- [13] Y. B. Ian J Goodfellow, "Deep Learning," *MIT Press*, 2016.

- [14] A. A. P. B. Martín Abadi, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv*, p. 1–19, 2016.
- [15] Q. V. L. Mingxing Tan, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, p. 6105–6114, 2019.
- [16] Q. Y. Sinno Jialin Pan, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, p. 1345–1359, 2010.
- [17] C. S. Sergey Ioffe, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, p. 448–456, 2015.
- [18] S. G. Weifeng Chen, "When Vision Transformers Outperform ResNets without Pretraining or Strong Data Augmentations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 7, p. 4210–4224, 2024.
- [19] L. B. A. K. Alexey Dosovitskiy, "An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale," *International Conference on Learning Representations (ICLR)*, p. 1–21, 2021.

Appendix

Python Source Code

```
import tensorflow as tf
```

```
import matplotlib.pyplot as plt
```

```
import zipfile
```

```
import os
```

```
from tensorflow.keras import layers
```

```
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.applications import EfficientNetB0
```

```
import os
```

```
for root, dirs, files in os.walk("/content"):
```

```
    for file in files:
```

```
        if file.endswith(".zip"):
```

```
            print(os.path.join(root, file))
```

```
import zipfile
```

```
import os
```

```
import shutil # Import shutil for rmtree

zip_file = "/content/PlantPathology_Split.zip"
extract_path = "/content/PlantPathology_Split"

# Ensure clean extraction: remove existing directory if it exists
if os.path.exists(extract_path):
    print(f"Removing existing directory: {extract_path}")
    shutil.rmtree(extract_path)

print(f"Extracting {zip_file} to {extract_path}...")
with zipfile.ZipFile(zip_file, "r") as zip_ref:
    zip_ref.extractall(extract_path)

print("Extraction completed!")

import os

base_dir = None
```

```
for root, dirs, files in os.walk(extract_path):
```

```
    if "train" in dirs and "valid" in dirs and "test" in dirs:
```

```
        base_dir = root
```

```
        break
```

```
print("Dataset Path:")
```

```
print(base_dir)
```

```
IMG_SIZE = (224,224)
```

```
BATCH_SIZE = 32
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
```

```
    os.path.join(base_dir,"train"),
```

```
    image_size=IMG_SIZE,
```

```
    batch_size=BATCH_SIZE,
```

```
label_mode="int"

)

valid_ds = tf.keras.utils.image_dataset_from_directory(

    os.path.join(base_dir,"valid"),

    image_size=IMG_SIZE,

    batch_size=BATCH_SIZE,

    label_mode="int"

)

test_ds = tf.keras.utils.image_dataset_from_directory(

    os.path.join(base_dir,"test"),
```

```
image_size=IMG_SIZE,  
  
batch_size=BATCH_SIZE,  
  
label_mode="int",  
  
shuffle=False  
  
)  
  
class_names = train_ds.class_names  
  
print(class_names)  
  
print("Number of Classes =",len(class_names))  
  
AUTOTUNE = tf.data.AUTOTUNE  
  
train_ds = train_ds.cache().shuffle(1000).prefetch(AUTOTUNE)
```

```
valid_ds = valid_ds.cache().prefetch(AUTOTUNE)
```

```
test_ds = test_ds.cache().prefetch(AUTOTUNE)
```

```
data_augmentation = tf.keras.Sequential([
```

```
    layers.RandomFlip("horizontal"),
```

```
    layers.RandomRotation(0.2),
```

```
    layers.RandomZoom(0.2),
```

```
    layers.RandomContrast(0.2),
```

```
    layers.RandomTranslation(0.1,0.1),
```

```
])
```

```
base_model = EfficientNetB0(
```

```
include_top=False,  
  
weights="imagenet",  
  
input_shape=(224,224,3)  
  
)  
  
base_model.trainable = False  
  
print("EfficientNetB0 Loaded Successfully")  
  
inputs = tf.keras.Input(shape=(224,224,3))  
  
x = data_augmentation(inputs)  
  
x = tf.keras.applications.efficientnet.preprocess_input(x)  
  
x = base_model(x, training=False)
```

```
x = layers.GlobalAveragePooling2D()(x)
```

```
x = layers.BatchNormalization()(x)
```

```
x = layers.Dense(512, activation="relu")(x)
```

```
x = layers.Dropout(0.5)(x)
```

```
x = layers.Dense(256, activation="relu")(x)
```

```
x = layers.Dropout(0.3)(x)
```

```
outputs = layers.Dense(len(class_names), activation="softmax")(x)
```

```
model = Model(inputs, outputs)
```

```
model.summary()
```

```
model.compile(
```

```
optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),

loss="sparse_categorical_crossentropy",

metrics=["accuracy"]

)

early_stop = tf.keras.callbacks.EarlyStopping(

    monitor="val_loss",

    patience=5,

    restore_best_weights=True,

    verbose=1

)

reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(

    monitor="val_loss",

    factor=0.2,

    patience=2,
```

```
min_lr=1e-6,  
verbose=1  
)  
  
checkpoint = tf.keras.callbacks.ModelCheckpoint(  
    "best_model.keras",  
    monitor="val_accuracy",  
    save_best_only=True,  
    verbose=1  
)  
  
history1 = model.fit(  
    train_ds,  
    validation_data=valid_ds,  
    epochs=20,  
    callbacks=[  
        early_stop,  
        reduce_lr,  
        checkpoint  
    ]
```

```
)
```

```
base_model.trainable = True
```

```
for layer in base_model.layers[:-20]:
```

```
    layer.trainable = False
```

```
print("Fine-tuning last 20 layers...")
```

```
model.compile(
```

```
    optimizer=tf.keras.optimizers.Adam(
```

```
        learning_rate=1e-5
```

```
    ),
```

```
    loss="sparse_categorical_crossentropy",
```

```
    metrics=["accuracy"]
```

```
)
```

```
history2 = model.fit(
```

```
    train_ds,
```

```
    validation_data=valid_ds,
```

```
epochs=20,  
callbacks=[  
    early_stop,  
    reduce_lr,  
    checkpoint  
]  
)  
  
test_loss, test_accuracy = model.evaluate(test_ds)  
  
print("Test Accuracy:", test_accuracy)  
  
import numpy as np  
from sklearn.metrics import confusion_matrix  
import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Get predictions on the test dataset  
predictions = model.predict(test_ds)  
predicted_classes = np.argmax(predictions, axis=1)
```

```
# Get true labels from the test dataset

true_classes = np.concatenate([y for x, y in test_ds], axis=0)


# Compute the confusion matrix

cm = confusion_matrix(true_classes, predicted_classes)


# Plot the confusion matrix as a heatmap

plt.figure(figsize=(12, 10))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix Heatmap')

plt.xticks(rotation=90)

plt.yticks(rotation=0)

plt.tight_layout()

plt.show()


"""### Confusion Matrix for the First 5 Classes"""
```

```
import numpy as np

from sklearn.metrics import confusion_matrix

import tensorflow as tf # Added for dataset creation

import os # Added for dataset path


# --- Ensure core variables for dataset and model are available ---

# Assume these are constants from earlier cells if not in globals

# Fallback values are hardcoded from the notebook's execution history to
increase robustness

IMG_SIZE = globals().get('IMG_SIZE', (224, 224))

BATCH_SIZE = globals().get('BATCH_SIZE', 32)

base_dir = globals().get('base_dir',
'/content/PlantPathology_Split/PlantPathology_Split')


# Ensure train_ds and test_ds are available. Recreate if missing.

if 'train_ds' not in globals() or 'test_ds' not in globals():

    print("Re-creating train_ds and test_ds as they were not found in the
current scope.")

    train_ds = tf.keras.utils.image_dataset_from_directory(

        os.path.join(base_dir, "train"),
```

```
    image_size=IMG_SIZE,  
    batch_size=BATCH_SIZE,  
    label_mode="int"  
)  
  
test_ds = tf.keras.utils.image_dataset_from_directory(  
    os.path.join(base_dir, "test"),  
    image_size=IMG_SIZE,  
    batch_size=BATCH_SIZE,  
    label_mode="int",  
    shuffle=False  
)
```

Ensure class_names is available. Re-derive from train_ds if missing.

if 'class_names' not in globals():

```
    print("Re-defining class_names from train_ds.")
```

```
    class_names = train_ds.class_names
```

Ensure true_classes and predicted_classes are available. Recalculate if missing.

if 'true_classes' not in globals() or 'predicted_classes' not in globals():

```
if 'model' in globals():

    print("Re-calculating predictions as they were not found in the current
scope.")

    predictions = globals()['model'].predict(test_ds)

    predicted_classes = np.argmax(predictions, axis=1)

    true_classes = np.concatenate([y for x, y in test_ds], axis=0)

else:

    raise NameError("Model is not defined. Please ensure model training
cells (e.g., _cgKdQoNjb7H or OIZ--l-YjcBI) have been executed to define
'model'.")

# --- Original logic for 5-class heatmap ---

# Select the first 5 class names

first_5_class_names = class_names[:5]

# Get the indices of the first 5 classes

first_5_class_indices = list(range(len(first_5_class_names)))

# Filter true and predicted classes to only include the first 5 classes

filtered_true_classes = []

filtered_predicted_classes = []
```

```
for i in range(len(true_classes)):

    if true_classes[i] in first_5_class_indices:

        filtered_true_classes.append(true_classes[i])

        filtered_predicted_classes.append(predicted_classes[i])


# Convert to numpy arrays

filtered_true_classes = np.array(filtered_true_classes)

filtered_predicted_classes = np.array(filtered_predicted_classes)


# Compute the confusion matrix for the first 5 classes

cm_5_classes = confusion_matrix(filtered_true_classes,
filtered_predicted_classes, labels=first_5_class_indices)


import matplotlib.pyplot as plt

import seaborn as sns


# Plot the confusion matrix for the first 5 classes

plt.figure(figsize=(10, 8))

sns.heatmap(cm_5_classes, annot=True, fmt='d', cmap='Blues',
```

```
        xticklabels=first_5_class_names, yticklabels=first_5_class_names)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix Heatmap for First 5 Classes')

plt.xticks(rotation=45, ha='right')

plt.yticks(rotation=0)

plt.tight_layout()

plt.show()


plt.figure(figsize=(14,5))


plt.subplot(1,2,1)

plt.plot(history1.history["accuracy"])

plt.plot(history1.history["val_accuracy"])

plt.title("Accuracy")

plt.xlabel("Epoch")

plt.ylabel("Accuracy")

plt.legend(["Train", "Validation"])


plt.subplot(1,2,2)
```

```
plt.plot(history1.history["loss"])
```

```
plt.plot(history1.history["val_loss"])
```

```
plt.title("Loss")
```

```
plt.xlabel("Epoch")
```

```
plt.ylabel("Loss")
```

```
plt.legend(["Train", "Validation"])
```

```
plt.show()
```

```
model.save("Plant_Disease_EfficientNetB0_Final.keras")
```

```
print("Model Saved Successfully!")
```

