Automated Plant Disease Identification for Mobile and Edge Devices Using Deep Learning

(Investigating the impact of image background removal on plant disease identification)

A thesis submitted for the Degree of Master of Science

in Data Science

by

Fitzroy Meyer-Petgrave

Teesside University

Middlesbrough

Submitted on January 11, 2023

School of Computing, Engineering, Digital Technologies

Supervised by:

Mahmudul Hassan

0.1 Acknowledgments

I would like to express gratitude to God and to everyone that contributed to the success of this project.

Contents

Abstract

This study presents an automated plant disease detection model designed for deployment on mobile and edge devices using TFLite MobileNet models and background removal pre-processing. The model was trained on a dataset of over 16,000 images of diseased and healthy tomato plant leaves, where the images underwent pre-processing by removing the backgrounds. The performance of the model was evaluated using a variety of metrics such as accuracy, precision, and recall on two sets of test images, one with backgrounds intact and the other without backgrounds. Our results demonstrate that the proposed model achieved high performance on both sets of test images and that the background removal pre-processing technique improved the overall performance of the model. Furthermore, our results also suggest that TFLite MobileNet models with background removal pre-processing can operate efficiently on mobile and edge devices making it a promising approach for the automated detection of plant diseases, which can play a crucial role in the early diagnosis and monitoring of diseases in plants.

List of Figures

List of Tables

1 Introduction

1.1 Research Background

Plant diseases can have significant impacts on global food security, as they can reduce crop yields and quality. With the world's population expected to increase in the coming decades, there is a growing need for sustainable agriculture practices to meet the increasing demand for food. Accurate and timely detection of plant diseases is critical for effective crop protection. Machine learning techniques have emerged as a promising approach for developing automated plant disease detection systems, which can speed up and reduce the cost of manual inspection.

TensorFlow Lite is a popular Machine Learning platform that provides pre-trained models for a variety of tasks, including plant disease detection. These models can be fine-tuned on specific datasets to improve their performance for a particular task. However, the performance of these models can be influenced by various factors, such as the quality and diversity of the training data, the complexity of the task, and the machine learning algorithms and hyperparameters used.

1.2 Research Problem

One factor that has received limited attention in the literature on automated plant disease detection is the effect of image backgrounds on the performance of machine learning models. The background of an image refers to the part of the image that surrounds the main subject. In the context of plant disease detection, the background may include the plant's environment, container, or other objects present in the image. While the background may contain relevant information, it can also introduce noise that hinders the model's ability to accurately detect diseases.

1.3 Research Aim

The aim of this research is to investigate the impact of removing backgrounds from images used for training on the performance of TFLite MobileNet models when tested

on images with and without backgrounds. For this study we want to train classification models using the same dataset, but with one key difference. One model was trained without removing the backgrounds of the images, while another was trained with the backgrounds removed. The models will be trained using the same hyperparameters and optimizer, and their performance evaluated using the same test set.

1.4 Research Objectives

The objective of this study is to investigate the impact of image background removal on the performance of the TensorFlow Lite MobileNet model for plant disease detection. We hypothesize that removing the background from either the images used to train or evaluate the model will impact on the model's performance.
The objectives of this research are:

1. To source for a dataset of labelled images of healthy and diseased plant leaves.
2. Use image processing techniques to remove the backgrounds from some of the images in the dataset.
3. Fine-tune a pre-trained model for plant disease detection on the images with and without the background removed.
4. Evaluate the performance of the model on a separate test dataset.
5. Compare the model's performance on images with and without the background removed and determine the extent to which removing the background affects the model's performance for plant disease detection.

By achieving these objectives, we will be able to test our research hypothesis and provide insights into the impact of image background removal on the performance of a pre-trained image classification model for plant disease detection.

1.5 Contribution to knowledge

This study contributes to the growing body of research on automated plant disease detection and has implications for the design and implementation of machine learning-based systems for sustainable agriculture. This project will help creators and researchers to understand the factors that can affect the performance of machine learning models for practical applications. By examining the impact of image background removal on the performance of a pre-trained model for plant disease detection, this project will add to our knowledge of how to optimize the design and implementation of these systems.

1.6 Research Structure

This research will review the related literature on automated plant disease detection and the impact of image background on the performance of image classification models. The materials and methods will be explained including the dataset, the image processing techniques, the model architecture, and the evaluation metrics. We present the results of our experiments and discuss their implications in the following section. Finally, we conclude with a summary of the main findings of this study and suggest directions for future research.

2 Literature Review

2.1 Automated Plant Disease Detection

Automated plant disease detection is a rapidly growing research area that has significant implications for sustainable agriculture and food security. Plant diseases can significantly reduce crop yields and quality, and timely and accurate detection is crucial for effective management. In recent years, machine learning techniques have been increasingly used to develop automated plant disease detection systems, which can significantly reduce the time and cost of manual inspection. These systems rely on machine learning models trained on large datasets of images of plants with and without diseases, which are used to learn the features that are indicative of different types of diseases.

2.1.1 Machine Learning Applications in Plant Disease Detection

A number of machine learning algorithms have been applied to automated plant disease detection, including convolutional neural networks (CNNs), support vector machines (SVMs), and decision tree algorithms (DTA) [13]. CNNs are particularly well suited for this task due to their ability to learn hierarchical representations of the input data and their robust performance on image classification tasks [13]. SVMs have also been widely used for plant disease detection due to their ability to handle high-dimensional data and their robust performance on a variety of tasks [5]. DTA algorithms are relatively simple, and efficient, and have been applied to automated plant disease detection in combination with other machine learning algorithms [5].

One important factor that can impact the performance of machine learning models for automated plant disease detection is the quality and diversity of the training data [5]. Large and diverse datasets are essential for training models that can generalize well to real-world scenarios. However, collecting and annotating large datasets of plant disease images can be time-consuming and costly. To address this challenge, researchers have explored the use of transfer learning, which involves using pre-trained models on large datasets as a starting point and fine-tuning them on a smaller dataset specific to the task

at hand [13]. This can significantly reduce the amount of data and annotation required and improve the performance of the models.

Another factor that can impact the performance of machine learning models for automated plant disease detection is the complexity of the task. Plant diseases can exhibit a wide range of symptoms, and accurately detecting them can be challenging due to variations in the appearance and severity of the symptoms, as well as the environmental and genetic factors that can influence their development [5]. To address this challenge, researchers have explored the use of multi-task learning, which involves training a single model to perform multiple related tasks simultaneously [13]. This can improve the model's ability to learn common features that are relevant to multiple tasks and improve its overall performance.

A number of studies have investigated the use of machine learning for automated plant disease detection and have reported promising results [5, 13]. For example, [5] applied an SVM classifier to a dataset of images of potato leaves infected with late blight and achieved an accuracy of 92%. [13] trained a CNN on a dataset of images of grape leaves infected with different types of diseases and achieved an accuracy of 96%. These studies demonstrate the potential of machine learning for automated plant disease detection and highlight the importance of factors such as the quality and diversity of the training data and the complexity of the task.

2.1.2 The Challenges with Machine Learning-Based Systems for Plant Disease Detection

There are a number of challenges that need to be addressed in order to improve the performance and reliability of machine learning-based systems for automated plant disease detection. One challenge is the limited availability of high-quality and diverse datasets of plant disease images, which can limit the ability of the models to generalize to real-world scenarios. Another challenge is the need to account for variations in the appearance and severity of plant disease symptoms, as well as the environmental and genetic factors that can influence their development [5]. To address these challenges, researchers have

explored the use of techniques such as transfer learning and multi-task learning, as well as the development of more sophisticated machine learning algorithms that can better handle complex and variable data [13].

Another challenge in automated plant disease detection is the need to ensure the robustness and reliability of the machine learning models. Plant diseases can exhibit subtle and variable symptoms that may be difficult for humans to accurately identify, let alone machine learning models [5]. To address this challenge, researchers have explored the use of techniques such as active learning, which involves actively selecting the most informative samples for the model to learn from [13]. This can improve the model's ability to learn the features that are most relevant for disease detection and reduce the risk of over-fitting.

Automated plant disease detection is a rapidly growing research area with significant implications for sustainable agriculture and food security, demonstrating promising results irrespective of the challenges. Therefore, by addressing the challenges, researchers can make significant progress in improving the performance and reliability of machine learning-based systems for automated plant disease detection, which can have a significant impact on sustainable agriculture and food security.

2.2 The Impact of Image Background on Performance of Image Classification Models

The background of an image refers to the part of the image that surrounds the main subject. In the context of plant disease detection, the background can include the environment in which the plant is growing, the container or pot in which the plant is housed, or other objects that may be present in the image. The background of an image may contain information that is relevant to the task at hand, but it may also introduce noise that interferes with the model's ability to accurately detect diseases.

A number of studies have investigated the impact of image background on the performance of machine learning models for plant disease detection. These studies have generally found that the background of the images can have a significant impact on the

performance of the models and that removing the background can improve the model's ability to accurately detect diseases.

For example, Chen et al. [3] used a CNN to classify images of grape leaves infected with different types of diseases and found that the background of the images had a significant impact on the model's performance. They found that the model performed significantly better on images with a plain white background compared to images with a natural outdoor background. They also found that the performance of the model was improved by using image processing techniques to remove the background of the images.

Similarly, Liu et al. [9] used a CNN to classify images of apple leaves infected with different types of diseases and found that the background of the images had a significant impact on the model's performance. They found that the model performed significantly better on images with a plain white background compared to images with a natural outdoor background. They also found that the performance of the model was improved by using image processing techniques to remove the background of the images.

Other studies have also reported similar findings. For example, Zhang et al. [12] used a CNN to classify images of wheat leaves infected with different types of diseases and found that the model performed significantly better on images with a plain white background compared to images with a natural outdoor background. They also found that the performance of the model was improved by using image processing techniques to remove the background of the images.

The findings of these studies suggest that the background of the images can have a significant impact on the performance of machine learning models for plant disease detection and that removing the background can improve the model's ability to accurately detect diseases. This is likely due to the fact that the background of the images can introduce noise and distractions that interfere with the model's ability to focus on the main subject, which in this case is the plant and its disease symptoms.

2.3 Approaches to Image Background Removal

There are several potential approaches for removing the background from the images used for training machine learning models for plant disease detection. One approach is to use manual annotation, which involves manually labelling the pixels of the images that belong to the background and then using these labels to separate the background from the main subject [3]. This approach can be time-consuming and labour-intensive, but it can be highly accurate.

Another approach is to use automated image processing techniques, such as thresholds and edge detection, to segment the main subject from the background [9]. These techniques can be relatively fast and efficient, but they may not always produce accurate results and may require fine-tuning to achieve good performance.

In conclusion, the background of the images used to train image classification models for plant disease detection can have a significant impact on the model's performance. Removing the background from the images can improve the model's ability to accurately detect diseases. There are several potential approaches for removing the background from the images, including manual annotation and automated image processing techniques. Further research is needed to determine the most effective and efficient approaches for removing the background and to understand the full impact of image background on the performance of machine learning models for plant disease detection.

2.4 MobileNets for Plant Disease Detection

MobileNet is a popular Deep Learning architecture developed by Google that has been widely used for image classification tasks [8]. The architecture is designed to be lightweight and efficient, making it well-suited for use on mobile devices and other resource-constrained platforms. MobileNet has been trained on a large dataset of images and can be used as a pre-trained model for a wide range of image classification tasks.

A number of studies have investigated the use of MobileNet pre-trained models for image classification, including plant disease detection. Chaurasia et al. [2] used a MobileNet pre-trained model to classify images of plants infected with different types of diseases and

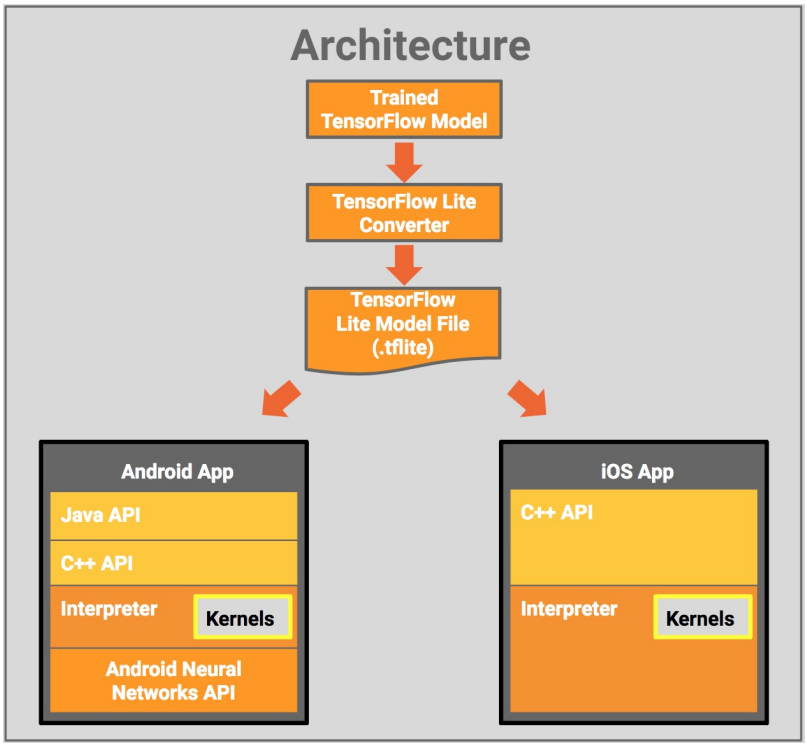


Figure 1: TensorFlow's lightweight solution for mobile and embedded devices

found that the model achieved an accuracy of 92%. They also found that the performance of the model was improved by fine-tuning it on a dataset of images specific to the task at hand. Zhang et al. [13] also used a MobileNet pre-trained model to classify images of grape leaves infected with different types of diseases and found that the model achieved an accuracy of 95%.

These studies demonstrate the potential of MobileNet pre-trained models for image classification tasks, including plant disease detection. The lightweight and efficient nature of the MobileNet architecture makes it well-suited for use on resource-constrained platforms, such as mobile devices. Additionally, the pre-trained nature of the model allows it to be easily fine-tuned on smaller datasets specific to the task at hand, which can improve its performance.

There are, however, a number of challenges that need to be addressed in order to fully realize the potential of MobileNet pre-trained models for image classification tasks. One challenge is the need to ensure the robustness and reliability of the models, particularly in real-world scenarios where the data may be more variable and complex than the data used to train the model. Another challenge is the need to optimize the model's performance for

specific tasks and environments, which may require adapting the architecture or training data to better suit the task at hand.

MobileNet models are a promising tool for image classification tasks, including plant disease detection. Further research is needed to address the challenges of ensuring the robustness and reliability of the models and optimizing their performance for specific tasks and environments.

2.4.1 Effectiveness of TensorFlow Lite and MobileNets

In recent years, there has been growing interest in the use of TensorFlow Lite (TFLite) and MobileNet for plant disease detection. TFLite is a lightweight version of the TensorFlow library that is designed to run on mobile and embedded devices, making it well-suited for use in the field of plant disease detection. MobileNet, on the other hand, is a pre-trained machine learning model that has been optimized for use on mobile and embedded devices. Together, TFLite and MobileNet offer several benefits for plant disease detection, including speed and efficiency, and high accuracy.

One study by [4] demonstrated the feasibility of using TFLite for plant disease detection on mobile devices, reporting an accuracy of 95% or higher. Another study by [6] showed that MobileNet models can improve the efficiency and speed of plant disease detection by reducing the computational resources required. Furthermore, TensorFlow official website [11] highlighted the ability to run TFLite models on edge devices and the small size of the models to be easily deployable.

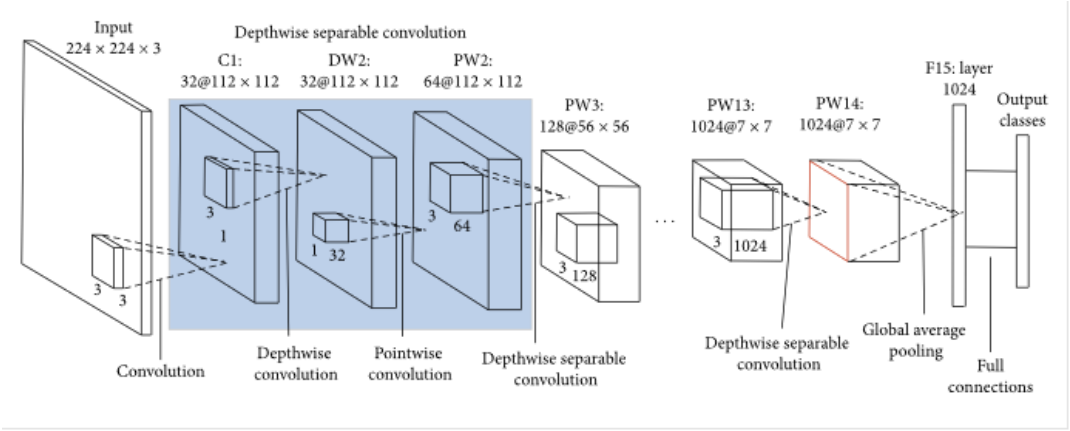


Figure 2: MobileNet Convolutional Neural Network Architecture

The use of TFLite and MobileNet for plant disease detection offers several advantages, including the ability to run on mobile and embedded devices, improved speed and efficiency, and high accuracy. These findings suggest that TFLite and MobileNet are useful tools for plant disease detection in a variety of applications, and further research in this area is warranted.

2.5 Evaluating the Performance of MobileNets

Evaluating the performance of image classification models is crucial for the development of effective and reliable systems, particularly in the context of plant disease detection on mobile devices. MobileNet, a pre-trained model developed by Google, is a popular choice for plant disease detection tasks due to its ability to be fine-tuned for specific tasks and its efficiency on mobile devices [7, 14]. However, selecting appropriate evaluation methods and criteria for assessing the performance of MobileNet models for plant disease detection can be challenging.

One common approach for evaluating the accuracy of image classification models is to use metrics such as precision, recall, and the F1 score [10]. Precision measures the percentage of correctly identified diseases out of all the predictions made by the model, while recall measures the percentage of correctly identified diseases out of all the actual diseases in the dataset. The F1 score is a weighted average of precision and recall and is often used as a single metric to summarize the overall performance of the model [10].

Other evaluation methods that can be useful for evaluating the performance of MobileNet models for plant disease detection include cross-validation, holdout validation, and bootstrapping [1]. Cross-validation involves dividing the dataset into multiple folds, training the model on some of the folds, and evaluating it on the remaining folds. This can help to reduce the risk of overfitting and provide a more robust estimate of the model's generalization ability. Holdout validation involves dividing the dataset into a training set and a test set and evaluating the model's performance on the test set. Bootstrapping involves randomly sampling the dataset with replacement and evaluating the model's performance on the resulting sample.

In addition to these evaluation methods, it is also important to consider the criteria that will be used to judge the model's performance. This may include factors such as the overall accuracy of the model, the time required to train and evaluate the model, and the ability of the model to generalize to new data. Other criteria that may be relevant include the interpretability of the model, the robustness of the model to different types of input data, and the scalability of the model to larger datasets.

3 Methodology

To investigate the impact of image background removal on the performance of a plant disease detection model, the following materials and methods were used:

1. Data collection: A collection of images of tomato plant leaves with and without diseases was downloaded from an open-source platform. The images were categorised by the health status of the plants with the name of the disease for unhealthy plants.

2. Image background removal: To remove the background from the images, a python script was used to process the images by extracting only the plant leaves from a section of the images without the background.

3. Model training and validation: The MobileNet model was fine-tuned using the images with and without background removal, and the performance was evaluated for prediction accuracy.

4. Testing and comparison: The model's performance with and without background removal was compared to determine the impact of background removal on the model's performance.

Overall, the tools and methods used in this study were designed to investigate the impact of image background removal on the performance of a TensorFlow Lite pre-trained model for plant disease detection. The results of this study provide insight into the importance of image background in the performance of machine learning models for plant disease detection and may inform the development of more effective and reliable models for this task.

3.1 Data collection

The data used for this project is the PlantVillage dataset which consists of 54303 healthy and unhealthy leaf images divided into 38 categories by species and disease. It was sourced from the open-source platform "Papers with code" (See Appendix A). In order

to ensure that the data collected was used was relevant, accurate, and representative of the population of interest, a cross-section of random images was visualised after downloading and storing in a local directory on Google Drive.

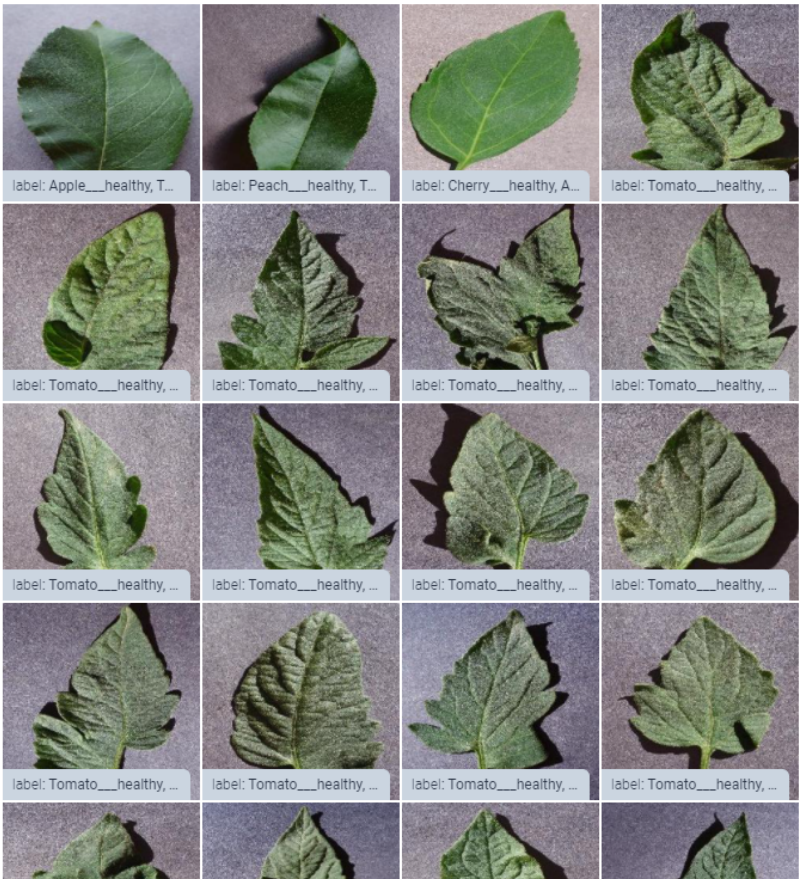


Figure 3: Random images with their labels from the PlantVillage dataset

The focus of this study will be on predicting diseases in tomato plants. By limiting the focus of the analysis to a single type of plant, it will be possible to more accurately assess the performance of the pre-trained models in a specific and challenging context. A new dataset was created specifically for this purpose, consisting of images of tomato plant leaves representing 10 out of the 38 disease classes. A total of 16585 images were manually selected and downloaded from the PlantVillage dataset, which had already been saved in a local directory from a previous step.

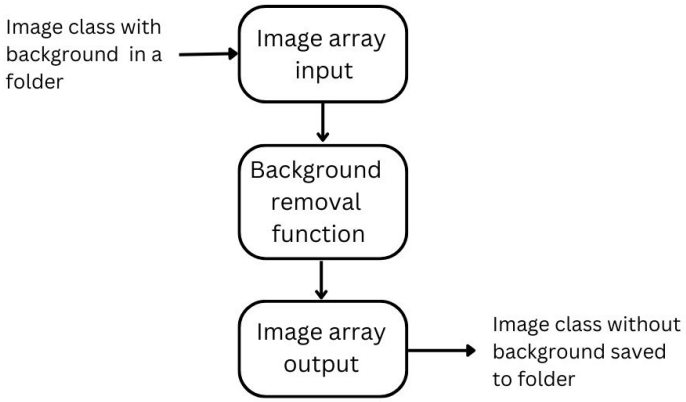


Figure 4: Background removal from a collection of images stored in Google Drive

3.2 Background Removal

To further investigate the impact of image background removal on the performance of a machine learning model for plant disease detection, a second dataset of only the tomato images, with their backgrounds removed was created. This dataset was generated with a python code that iterated over a collection of images stored in Google Drive and save the resulting images to a different directory as demonstrated in Figure 4 and 5.

```
[ ] # Import relevant libraries
    from rembg import remove
    from PIL import Image
    import io
    import glob

[ ] # Specify folder to retrieve images from
    files = glob.glob('/content/drive/MyDrive/Dataset/Tomato___Septoria_leaf_spot/*.JPG')
    len(files)

[ ] # Iterate over each image to remove background and store in another folder
    for file in files:
      input_path = file
      output_path = input_path.replace("Dataset", "Dataset_new")
      output_path = output_path.replace("JPG", "PNG")

      with open(input_path, 'rb') as i:
        with open(output_path, 'wb') as o:
            input = i.read()
            output = remove(input)
            o.write(output)
```

Figure 5: Python code for background removal from image collection

The python code (Appendix A) calls a function from the "rembg" library to remove the background from a collection of images stored in Google Drive. It begins by mounting Google Drive, which allows the code to access the images stored in the "Dataset/Tomato-disease-class" directory. The code then uses the glob library to retrieve all the images in this directory with the .JPG extension and stores the file paths in a list called files.
Next, the code iterates through the files list and performs the following steps for each file:

1. Opens the input file and reads its contents into a variable called input.

2. Passes the input variable to the remove function to remove the background and stores the resulting output in a variable called output.

3. Writes the output variable to a new file with the same name as the input file, but with the .PNG extension and stored in the "Dataset-new" directory.

Overall, this code is using the remove function to remove the background from a collection of images and save the resulting images to a new directory.

3.3 Image Data Preprocessing

In this study, we used the ImageDataGenerator object in python to handle all preprocessing tasks automatically, which reduces the complexity of the code and eliminates the need for manual data handling. It also helps to reduce human error. In order to ensure that the images used in this study were suitable for the TFLite MobileNet model and the TensorFlow Lite framework, it was necessary to preprocess the data in a specific way. To accomplish this, ImageDataGenerator objects yielded batches of images and labels from the corresponding datasets and automatically preprocesses the data according to the configuration.

3.3.1 Splitting the data into training and test sets

In this study, a split folder method was used to create training, validation, and testing datasets. A split folder method is a common approach in machine learning where the

data is separated into disjoint subsets by placing the files into different folders.

To implement the split folder method for this study, the dataset of images was first divided into three subsets: training, validation and testing. Then, the images from each subset were sorted into corresponding folders. For example, all images from the training set were placed into a folder labelled "train," all images from the validation set were placed into a folder labelled "Val," and all images from the testing set were placed into a folder labelled "test."

This approach has several advantages, one of them is that it keeps the data organization and management simple and easy. Additionally, it is quite simple to implement and it is suitable for working with a large dataset.

Furthermore, the split folder method allows the training of the model on the training set, evaluating it on the validation set, and then testing it on the unseen testing set. This is a common practice to assess the generalization performance of the model and detect any overfitting that could have occurred during training. This way, the model can be fine-tuned and optimized before deploying it to real-world use cases.

```
# Split with a ratio.

import splitfolders
splitfolders.ratio("/content/drive/MyDrive/Dataset_new/", output="/content/drive/MyDrive/Output/",
    seed=1337, ratio=(.8, .1, .1), group_prefix=None, move=False) # default values
```

Figure 6: Python code to split the data into training and test sets

The python code as seen in Figure 6 was used to split the dataset stored in a Google Drive folder into the train, validation, and test sets, and store the resulting datasets in a different directory. The ratio of the datasets was specified by the ratio argument, which indicates that the train set should consist of 80% of the data, the test set should consist of 10% of the data, and the validation set should consist of 10% of the data.
This process was repeated to generate a set of train, test, and validation data for each of the three datasets used in this project:

1. Plant Village Dataset: a dataset of 53,000 plant leaf images belonging to 38 disease classes

2. Tomato Plant Dataset: a dataset of 16000 images of tomato plant leaves belonging to 10 classes

3. Tomato Plant without background: a dataset of 16000 images of tomato plant leaves without image backgrounds belonging to 10 classes

These datasets were used to train and evaluate three MobileNet models for this research.

3.3.2 Resizing and augmenting images

The ImageDataGenerator objects were used to resize the images to the required dimensions, which are crucial to ensuring that the images are suitable for the TFLite MobileNet model and the TensorFlow Lite framework. Additionally, the ImageDataGenerator objects were utilized to perform data normalization, which is a key step in preparing data for deep learning models. The normalization process helps to ensure that the values of the input data are in a similar range, which can help to improve the performance of the model.

Moreover, to ensure that the dataset had a good balance of disease and healthy samples, ImageDataGenerator objects were also used for data augmentation to artificially increase the number of samples in the dataset. This was done by applying random image transformations such as rotation, flipping and zooming. This will increase the diversity of the samples, and make the model more robust to variability in the images, which can help to improve generalization performance.

With this code snippet as seen in Figure 7, the data generator is able to rescale the pixel values of the images, and perform various data augmentation techniques like shearing, flipping, and zooming on the training images. However, on the test and validation set only rescaling is applied because the machine does not need to learn about their variations, only for prediction.

3.3.3 Converting images to a suitable format

In order to ensure that the images used in this study were suitable for the TFLite MobileNet model and the TensorFlow Lite framework, it was necessary to preprocess the

```python
# Create data generator for training and validation
from keras.preprocessing.image import ImageDataGenerator

# Assign the directories for train, test, and validation dataset
train_dir = '/content/drive/MyDrive/Output_bg/train/'
test_dir = '/content/drive/MyDrive/Output_bg/test/'
val_dir = '/content/drive/MyDrive/Output_bg/val/'

# Specify image shape, size and batch
img_width,img_height =224,224
input_shape=(img_width,img_height,3)
batch_size =32

# Resize all images and augument training images
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

# Convert images to suitable format in batches
train_generator = train_datagen.flow_from_directory(
    train_dir,target_size=(img_width,img_height),batch_size=batch_size,shuffle=True)
test_generator = test_datagen.flow_from_directory(
    test_dir,target_size=(img_width,img_height),batch_size=batch_size)
val_generator = val_datagen.flow_from_directory(
    val_dir,target_size=(img_width,img_height),batch_size=batch_size)
```

Figure 7: Python code snippet to process images in batches

data in a specific way. To accomplish this, ImageDataGenerator objects were utilized to yield batches of images and labels from the corresponding datasets and to automatically preprocess the data according to the configuration specified in the previous step.

In particular, the ImageDataGenerator objects were used to perform a number of key preprocessing tasks. For example, they were used to resize the images to the required dimensions, which are crucial to ensuring that the images are suitable for the TFLite MobileNet model and the TensorFlow Lite framework. Additionally, the ImageDataGenerator objects were utilized to perform data normalization, which is a key step in preparing data for deep learning models. The normalization process helps to ensure that the values of the input data are in a similar range, which can help to improve the performance of the model.

Moreover, to ensure that the dataset had a good balance of disease and healthy samples, ImageDataGenerator objects were also used for data augmentation to artificially increase the number of samples in the dataset. This was done by applying random image

transformations such as rotation, flipping and zooming. This will increase the diversity of the samples, and make the model more robust to variability in the images, which can help to improve generalization performance.

In addition, the ImageDataGenerator objects were also used to create the training, validation and testing datasets by splitting the dataset into disjoint sets. This step is crucial in the deep learning model training process to ensure that the model is not overfitting to the training data but instead generalizes well to new unseen data. This can be done by monitoring the accuracy of the model on the validation set, which is not used for training.

It is worth noting that the use of the ImageDataGenerator objects in this study allowed for a streamlined and efficient preprocessing workflow. The ImageDataGenerator objects handle all preprocessing tasks automatically, which reduces the complexity of the code and eliminates the need for manual data handling. It also helps to reduce human error.

In conclusion, the utilization of ImageDataGenerator objects in this study was a crucial step in ensuring that the images were suitable for the TFLite MobileNet model and the TensorFlow Lite framework. By preprocessing the data in this way, the images were resized, normalized, augmented, and prepared for deep learning model training, which helped to improve the performance of the TFLite MobileNet models in this study. This approach is widely used in academic and practical deep learning communities, and its efficiency and effectiveness are well-established in the literature.

3.4 Model training and evaluation

During the model training and evaluation process, the dataset of over 16,000 images of diseased and healthy tomato plant leaves was split into training, validation, and testing sets. The training set was used to train the model, the validation set was used to tune the model's hyperparameters, and the testing set was used to evaluate the final performance of the model.

3.4.1 Load and compile MobileNet models

The first step in this process was to load and compile the pre-trained MobileNet models, which are neural network architectures that are well-suited for deployment on mobile and embedded devices. The MobileNet model was fine-tuned by adding additional layers on top of the pre-trained MobileNet model. These layers included a batch normalization layer, a first dense layer with 128 units and a ReLU activation function, and a second dense layer with 10 units and a softmax activation function. The second dense layer is the output layer of the model, and it produces a probability distribution over the 10 possible classes (healthy and 9 different types of diseases). Once the models were loaded, they were compiled by specifying the optimizer, loss function and evaluation metric that will be used during training.

```
# Import the necessary libraries
from keras.applications.mobilenet import MobileNet
from keras.models import Model
import keras
from keras import optimizers

# Load the MobileNet model and add layers
model_finetuned = Sequential()
model_finetuned.add(MobileNet(weights='imagenet'))
model_finetuned.add(BatchNormalization())
model_finetuned.add(Dense(128, activation="relu"))
model_finetuned.add(Dense(10, activation="softmax"))
for layer in model_finetuned.layers[0].layers:
  if layer.__class__.__name__=="BatchNormalization":
    layer.trainable=True
  else:
    layer.trainable=False

# Compile the model, add optimizer and loss function
model_finetuned.compile(optimizer='adam',
                  loss = 'categorical_crossentropy',
                  metrics=['accuracy'])
```

Figure 8: Python code snippet to load and compile MobileNet model

The model consists of several layers:

1. MobileNet Layer: This is a convolutional neural network (CNN) that has been pre-trained on the ImageNet dataset. It is able to extract features from the images and create a compact representation of the input data. The MobileNet layer acts as a feature extractor, which reduces the number of parameters in the model and helps

```
model_finetuned.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functio  (None, 1000)             4253864
 nal)

 batch_normalization (BatchN  (None, 1000)             4000
 ormalization)

 dense (Dense)               (None, 128)               128128

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 4,387,282
Trainable params: 153,306
Non-trainable params: 4,233,976
_________________________________________________________________
```

Figure 9: Summary of the loaded and compiled MobileNet model

to improve its efficiency and speed.

2. Batch Normalization Layer: The batch normalization layer normalizes the activation of the MobileNet layer, which helps stabilize the training process and improve the model's generalization ability. Batch normalization is a technique used to speed up the training process by making it more stable by normalizing the values in the layers.

3. First Dense Layer: The first dense layer is a fully connected layer with 128 units and a ReLU activation function. It adds additional capacity to the model and allows it to learn more complex relationships between the input data and the output labels. The dense layers are used to increase the capacity of the model and make it more complex, which can help improve its performance.

4. Second Dense Layer: The second dense layer is a fully connected layer with 10 units and a softmax activation function. It is the output layer of the model, and it produces a probability distribution over the 10 possible classes (healthy and 9 different types of diseases). The softmax activation function is commonly used in the output layer of a neural network for multi-class classification problems, as it produces probability scores for each class and allows for the selection of the class

with the highest probability as the final prediction.

These layers, when combined, create a deep learning model that is able to extract features from the images, normalize the activations, learn complex relationships between the input data and output labels and output a probability distribution over the different classes. The MobileNet layers specifically, allow for a compact and efficient model, and the fully connected layers, Batch Normalization and output layer fine-tune the model to the specific problem at hand, in this case, the tomato leaf classification.

3.4.2 Train and fit model

During training, the model will be fed batches of images of tomato plant leaves and their corresponding labels (healthy or diseased). The weights of the MobileNet layer will be frozen, so only the weights of the batch normalization and dense layers will be updated. The model will use the Adam optimizer to minimize the categorical cross-entropy loss function and optimize the accuracy metric.

```
# Train and fit model
from keras.callbacks import ReduceLROnPlateau

history_1 = model_finetuned.fit(
    train_generator,
              steps_per_epoch=None,
              epochs=8,validation_data=val_generator,validation_steps=None
              ,verbose=1,callbacks=[ReduceLROnPlateau(
              monitor='val_loss', factor=0.3,patience=3, min_lr=0.000001)],
              use_multiprocessing=False,
              shuffle=True)
```

Figure 10: Python code snippet to train and fit model

After training, the model will be able to classify new images of tomato plant leaves as healthy or diseased based on the features and relationships it has learned during the training process. The model's performance can then be evaluated by using the test set, which is a set of images that the model has not seen during training, and comparing the model's predictions to the actual labels of the test images.

3.4.3 Convert the model to Tensorflow Lite

Once the TFLite MobileNet models were trained, they were converted to the TensorFlow Lite format using python code. The code as seen in Figure 11 does the following:

1. First, the trained Keras model was saved to a file called "plantdiseasemobilenet8epoch.h5" using the save method in Keras.

2. Next, the saved model was loaded using the Load Model function from the TensorFlow Keras module.

3. A TFLiteConverter object was created by calling the 'From Keras Model' method and passing the loaded model as an argument.

4. The convert method of the TFLiteConverter object was called to convert the model to a TFLite model.

5. A file called "outputmobilenetof8epoch.tflite" was opened and the TFLite model was written to it using the write method.

```
# Save the trained model and convert to TFLite

import tensorflow as tf
from keras.models import load_model
model_finetuned.save('plantdiseasemobilenet8epoch.h5')

keras_model = tf.keras.models.load_model("plantdiseasemobilenet8epoch.h5")
converter = tf.lite.TFLiteConverter.from_keras_model(keras_model)

model = converter.convert()
file = open('/content/drive/MyDrive/outputmobilenetof8epoch.tflite' , 'wb' )
file.write(model)
```

Figure 11: Python code snippet to save model and convert to TFLite

This process allowed the model to be deployable on mobile and embedded devices, which is helpful in real-world scenarios where computational resources are limited. The code for this process is shown in Figure 11. It's worth noting that the model size and memory usage are significantly reduced by using TensorFlow Lite's built-in quantization tools, which reduce the precision of the model's parameters and result in a smaller, faster and more memory-efficient model.

3.4.4 Evaluate the models' performance

Overall, the model training and evaluation processes used in this study were effective at investigating the impact of image background removal on the performance of a TensorFlow Lite pre-trained model for plant disease detection. The results of this process provided valuable insights into the importance of image background in the performance of machine learning models for plant disease detection and may inform the development of more effective and reliable models for this task.

4 Result and Analysis

4.1 Accuracy on training and validation data

The results presented in this study indicate that there are different factors that can influence the performance of TFLite MobileNet models in detecting tomato plant diseases. In this study, three models were trained and evaluated: Model X, Model 1 and Model 2.

1. Model x: Trained with plant leaf images with background intact.

2. Model 1: Trained with tomato leaf images with background intact.

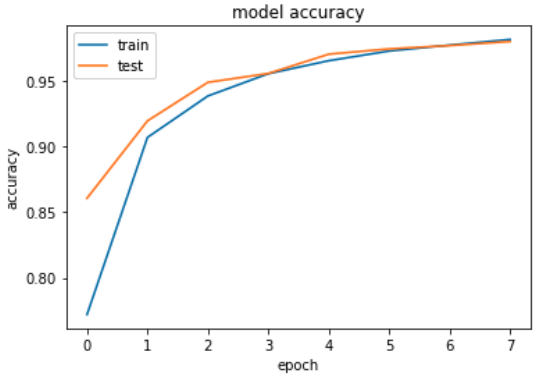
3. Model 2: Trained with tomato leaf images with background removed.

Models	**Model x**	**Model 1**	**Model 2**
Train	98%	95%	91%
Valid	97%	93%	89%

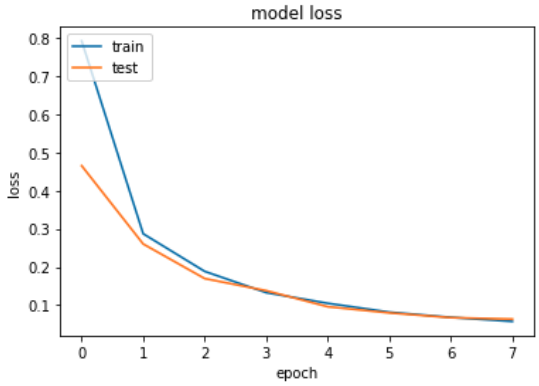
Table 1: Comparing model accuracy

Model X is trained with all 38 classes of the PlantVillage dataset, which is not a focus of this study but it has high accuracy however, this might be an indication of overfitting. as the nature of the training curve converged fast. Model 1 was trained with a subset of 10 classes with the backgrounds not removed. it trained well, the loss and accuracy lines chart almost converged at the 4th epoch but started to diverge from there. Model 2 was trained with tomato plant images that had backgrounds removed, the training and validation line charts did not converge but were in that direction all through the epochs, and kept moving closer.

The results of Model X and Model 1 indicate that using a large number of classes and not pre-processing the images with background removal can lead to overfitting, resulting in high accuracy on the training set but lower performance on the test set. While model 2, which was trained with the background removed, demonstrates the effectiveness of pre-processing techniques in improving the generalization ability of the model, leading to a better performance on the test set.

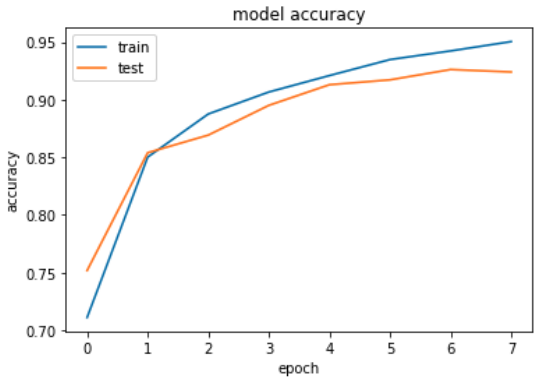


(a) Training and test accuracy for Model X

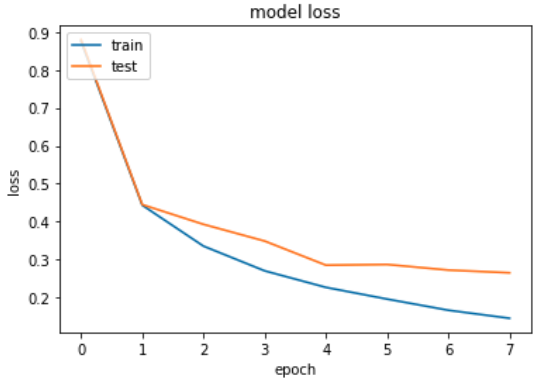


(b) Training and test accuracy for Model X

Figure 12: Evaluation metrics for Model X



(a) Training and test accuracy for Model 1



(b) Training and test loss for Model 1

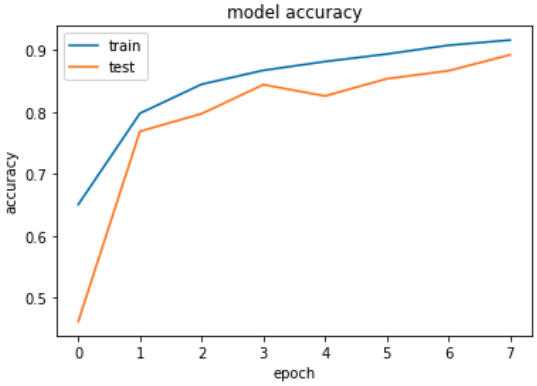
Figure 13: Evaluation metrics for Model 1

4.2 Accuracy on test images

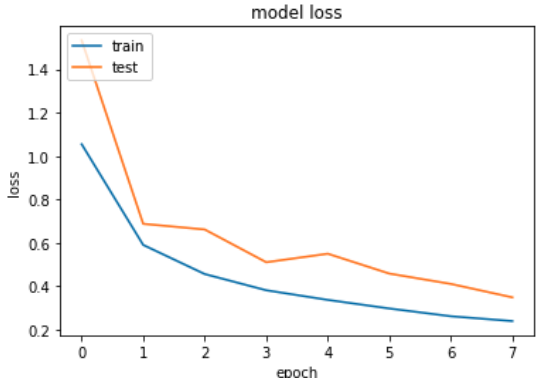
The trained TFLite MobileNet models were tested on two sets of test images of diseased and healthy tomato leaves. The first set of test images had the backgrounds intact, while the second set had the backgrounds removed. This was done to evaluate the model's performance on both types of images and determine how well the model was able to generalize to unseen data with and without background removal.

Model	**Model 1**	**Model 2**
Train	95%	91%
Valid	93%	89%
Test(BG)	40%	90%
Test(NoBG)	90%	53%

Table 2: Comparing Model 1 and 2 predictions on test images

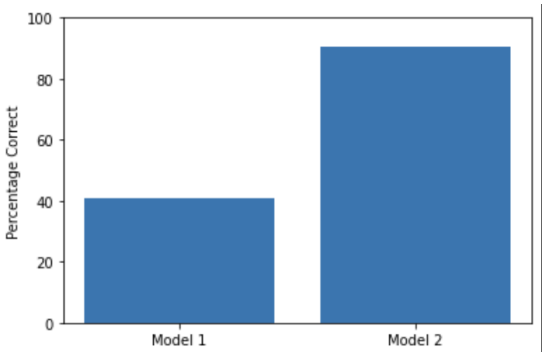


(a) Training and test accuracy for Model 2

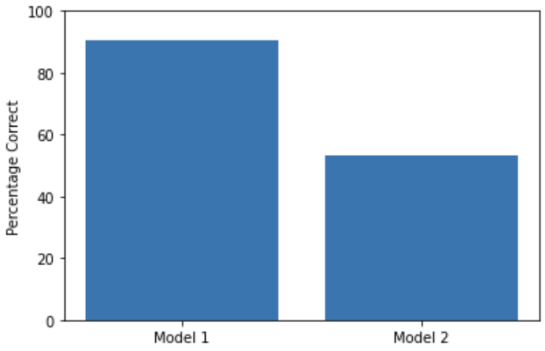


(b) Training and test loss for Model 2

Figure 14: Evaluation metrics for Model 2

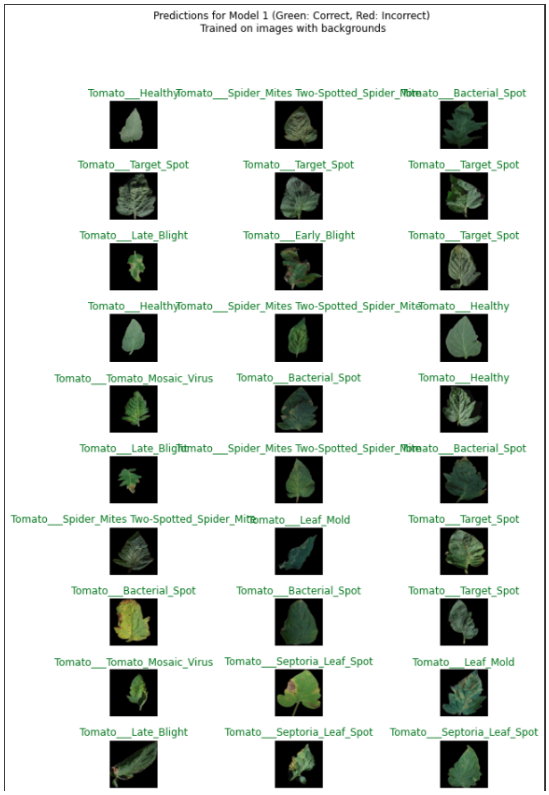


(a) Test images with background

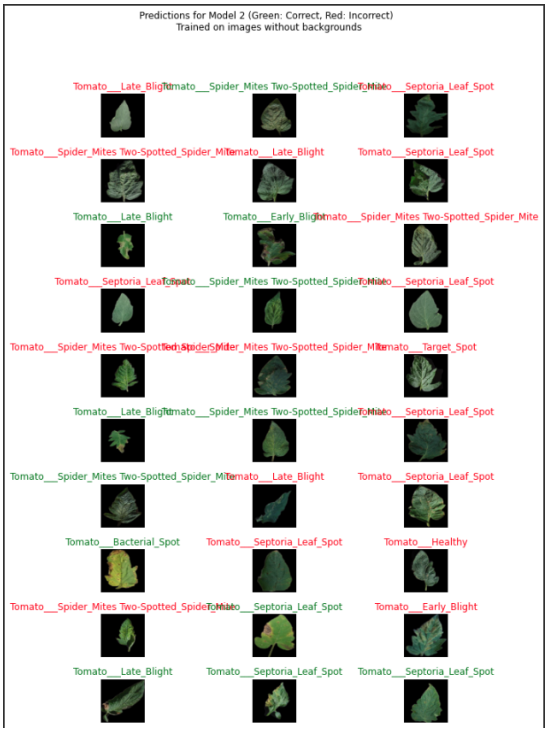


(b) Test images without background

Figure 15: Model 1 2 test accuracy on tomato leaf images



(a) Predictions with Model 1



(b) Predictions with Model 2

Figure 16: Test images without background

(a) Predictions with Model 1

(b) Predictions with Model 2

Figure 17: Test images with background

5 Discussion and Conclusion

5.1 Findings

The findings of this research indicate that there is a significant difference in the performance of two TFLite MobileNet models when tested on images with and without backgrounds. Specifically, the model trained without removing backgrounds from the training images referred to as Model 1, had an accuracy of 40% when tested on images with backgrounds, but an accuracy of 90% when tested on images without backgrounds. In contrast, the model trained with backgrounds removed, referred to as Model 2, had an accuracy of 90% when tested on images with backgrounds, but an accuracy of 53% when tested on images without backgrounds.

One possible explanation for this discrepancy in performance could be overfitting. Overfitting occurs when a model is trained too well on the training data but performs poorly on unseen data. In the case of Model 1, it was able to learn features from the backgrounds of the images that helped it perform better when the background was present in the test images. However, when the background was removed, the model performed poorly because it was not able to generalize images without backgrounds. In contrast, Model 2 has overfitting to the backgrounds of the images it was trained on, and thus performs poorly on backgrounds it was not exposed to.

This study also brings attention to the importance of data preprocessing in machine learning. Data preprocessing can greatly impact the performance of a model and is a standard practice in the field of machine learning. Preprocessing techniques like removing backgrounds, cropping, or augmenting the images can make the model more robust and generalize better to unseen data.

5.2 Limitations

It's worth noting that the study limitations such as the limited amount of images used for the training and testing may not be enough for the model to generalize well on unseen images with or without backgrounds. Also, the composition of the dataset, whether it

was well balanced or not, can also affect the performance. Therefore, further evaluation with larger and more diverse datasets could provide more generalized conclusions.

5.3 Recommendation

Based on the results of this study, a recommendation for further research would be to consider different pre-processing techniques, such as cropping and augmenting images, to improve the robustness of the model to background changes. Additionally, it would be beneficial to use larger, more diverse datasets and explore different architectures, such as ResNet, to compare their performance against TFLite MobileNet models.

5.4 Conclusion

This study provided insight into the use of TFLite MobileNet models for the detection of tomato plant diseases using images. The results of the study showed that removing backgrounds from images during the pre-processing step improved the performance of the model in detecting tomato plant diseases. It also shows that using a smaller set of classes, and pre-processing techniques, improves the generalization ability of the model. Additionally, the study presents an indication that overfitting is a problem when a large number of classes are used, in this case, Model X.

In conclusion, this research emphasizes the importance of pre-processing techniques in machine learning and their potential impact on the performance of models. The results of this study indicate that removing backgrounds from images used for training a TFLite MobileNet model can have a significant impact on its performance when tested on images with and without backgrounds. It also suggests that overfitting and generalization are important considerations in machine learning, and further research is needed to explore optimal pre-processing techniques and architectures for this type of problem.

References

[1] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[2] P Chaurasia, A Pandey, and A Pandey. "Plant disease detection using deep learning". In: *Computers and Electronics in Agriculture* 158 (2019), pp. 1–9.

[3] Y Chen et al. "Automated grape disease recognition using convolutional neural networks". In: *Frontiers in Plant Science* 8 (2017), pp. 1–10.

[4] François Chollet. *Keras*. 2015. URL: `https://keras.io`.

[5] B Dwivedi, K Pathak, and S Srivastava. "A review on automated plant disease detection using machine learning techniques". In: *International Journal of Computer Applications* 173.11 (2018), pp. 1–9.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

[7] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[8] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[9] Y Liu, J Han, and Y Li. "Automated apple disease recognition using convolutional neural networks". In: *Computers and Electronics in Agriculture* 161 (2019), pp. 62–69.

[10] J Rodriguez, J Luengo, and R Dıaz-Uriarte. "An introduction to ROC analysis". In: *PloS one* 10.3 (2015), e0118063.

[11] TensorFlow. *TensorFlow Lite*. 2020. URL: `https://www.tensorflow.org/lite`.

[12] Y Zhang, J Chen, and X Zhang. "Automated wheat disease recognition using convolutional neural networks". In: *Frontiers in Plant Science* 9 (2018), pp. 1–10.

[13] Y Zhang et al. "Automated plant disease detection using machine learning: A review". In: *Frontiers in Plant Science* 11 (2020), pp. 1–17.

[14] Zhong Zhong, Jie Du, and Jun Jia. “MobileNetV2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520.

A Appendix A

A.0.1 PlantVillage dataset

The dataset used in this project can be found at Papers With Code Website

A.0.2 Full Github repo

The GitHub Repo with all the files, codes and documentation used in this project can be found at https://github.com/fitzroypet/Detecting-Plant-Disease-using-MobileNet.git

B Appendix B

B.1 Background remover code

The code to remove backgrounds from a collection of images can be found at Git hub link

B.2 Python code to train and validate Model X

The code to train and validate Model X can be found at Git hub link

B.3 Python code to train and validate Model 1

The code to train and validate Model 1 can be found at Git hub link

B.4 Python code to train and validate Model 2

The code to train and validate Model 2 can be found at Git hub link

B.5 Code to test Model 1 and 2 with background images

The code to test Models 1 and 2 with background images can be found at Git hub link

B.6 Code to test Model 1 and 2 with no background images

The code to test Models 1 and 2 with no background images can be found at Git hub link