Bachelor of Science in Computer Science
June 2023

Improving the Accuracy of Plant Leaf Disease Detection and Classification in Images of Plant Leaves:

By Exploring Various Techniques with the MobileNetV2 Model

Susanthika Sadhu
Veera Venkata Sai Kashyap Kaligotla

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science. The thesis is equivalent to 10 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Sai Kashyap Kaligotla
E-mail: veka22@student.bth.se

Susanthika Sadhu
E-mail: susd22@student.bth.se

University advisor:
Suejb Memeti,PhD
Department of Computer Science

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background:
In recent years, there has been a consistent increase in utilizing deep learning models to identify and categorize plant leaf diseases. But there is still a gap in the research study regarding the particular techniques that can enhance the performance of these models. Some techniques for enhancing accuracy like adjusting the learning rate, data augmentation operations, and adding additional layers. These techniques help to increase the accuracy of the final model, which is used in plant leaf disease prevention, agricultural productivity, and disease control.
Objectives:
The goal of this thesis is to build and train the MobileNetV2 model for plant leaf images to detect diseases and classify them with efficient techniques to increase the accuracy of the MobileNetV2 model.
Methods:
The research methodology entails running an experiment to evaluate various techniques for improving the performance of the MobileNetV2 model to detect diseases and classify them in plant leaf images. The techniques used are adjusting the learning rate, adding additional layers to the model architecture, and implementing various data augmentation operations. To determine the most optimized model, the model is trained using the MobileNetV2 architecture and evaluated using metrics such as accuracy, precision, recall, and F1-score.
Results:
The study's results demonstrated the efficiency of various implementation strategies in improving the efficiency of the plant leaf disease detection and classification model. Particularly, altering the learning rate, adding extra layers, and applying data augmentation were discovered to have considerable favorable effects. The results and graphs are presented in tabular format for better understanding and visibility.
Conclusions:
Finally, our thesis demonstrates the effectiveness of incorporating random rotation and crop data augmentation techniques in the MobileNetV2 model for plant leaf disease recognition and classification in plant leaf images. By employing these techniques, we achieved remarkable evaluation metric scores of 94% for accuracy, 91% for precision, 96% for recall, and 95% for F1-score. As a result, this enhanced model allows for plant leaf disease classification and identification of plant leaf images.

Keywords: Additional layers, data augmentations, learning rate adjustment, MobileNetV2 model, plant leaf disease detection.

Acknowledgments

We would like to express our gratitude to our supervisor, Suejb Memeti, for his outstanding supervision, guidance, and encouragement. Our sincere thanks are expressed for his active involvement in leading, helpful suggestions, friendly support, and advice during our thesis study.

We are incredibly grateful to our examiner Prashant Goswami for his excellent advice and continuous support for our project proposal. His comprehensive research lectures greatly aided us in better comprehending the bachelor thesis.

Authors
Susanthika Sadhu
Veera Venkata Sai Kashyap Kaligotla

Contents

List of Figures

List of Tables

List of Acronyms

DL Deep Learning

TL Transfer Learning

NN Neural Networks

ANN Artificial Neural Network

CNN Convolutional Neural Network

DCNN Deep Convolutional Neural Network

conv Convolutional layer

Dwise Depth wise convolutional layer

NLP Natural Language Processing

tp True positive

tn True negative

fp False positive

fn False negative

ML Machine Learning

SVM Support Vector Machine

CSV Comma Separated Values

VGG Visual Geometry Group

MSE Mean Square Error

IoT Internet of Things

IDE Integrated Development Environment

OS Operating System

RNN Recurrent neural network

MLP Muilti Layer perceptron

sklearn Scikit-learn

API Application Program Interface

GPU Graphic Processing Unit

JPEG Joint Photographic Experts Group

PNG Portable Network Graphics

BMP Bitmap Image file

TIFF Tagged Image File Format

ReLU Rectified linear activation unit

Chapter 1

Introduction

Agriculture is the process of cultivating crops using basic factors like land, water, labor, seeds, tools, and technology. We can't imagine human existence without agriculture as it is providing the necessities like food for leading a healthy life. It is the foundation of every society on earth [25]. All around the world, agriculture helps in providing employment opportunities, nutrition, clothes, and healthcare. Any nation's economic development depends greatly on farming [31]. Humans also benefit from food security. Due to the rising population, there is a growing demand for food, hence agricultural technology advancements are necessary to supply the requirement [38].

Crop health is essential for optimum productivity, hence crop monitoring must be done regularly using a highly technological manner. Crop monitoring is essential to farming for proactive control of disease and pest management, effective resource utilization, yield optimization, and data-driven decision-making. It assists farmers in detecting and addressing difficulties quickly, making educated decisions, and achieving sustainable and productive farming systems. The presence of plant leaf diseases can adversely affect agricultural productivity and food quality [18]. Early identification of such diseases can help farmers save time and labor costs and promote plant growth. However, identifying plant leaf diseases reliably can be challenging due to the vast array of diseases and pests that affect plants.

Farmers or specialists can recognize and diagnose plant leaf diseases in a generic fashion. However, this approach could be time-consuming, costly, require specialist knowledge and inaccurate [31]. Recent advancements in computer vision, Machine Learning (ML), and Deep Learning (DL) have made it easier to detect plant illnesses [19]. These modern techniques involve analyzing plant leaf images and utilizing algorithms to identify patterns and diseases. Because many plant leaf diseases look quite similar, detecting diseases from images is an important area of research. Therefore, combination of image processing, ML, and computer vision techniques is necessary to identify and classify diseases accurately in plant leaves [34].

DL has advanced significantly in recent years, especially in image classification for detecting and classifying plant leaf disease images with precision and speed. However, DL models come with risks and challenges, including the need for large amounts of data for training. Transfer Learning (TL) addresses various drawbacks of DL models. TL is a powerful DL technique that improves learning by using knowledge from a previously learned task [14]. While TL is frequently linked with small dataset sizes, it can be used in a variety of circumstances, including big dataset sizes. TL

reduces overfitting by transferring learned representations and knowledge from the pre-trained model. It improves interpretability by relying on well-studied architectures. TL makes use of existing expertise, lowering the knowledge barrier for practitioners. It also provides the opportunity to remove data biases and enhance robustness by employing transferable representations. Training time and computing expenses are reduced since TL models are lightweight and compatible with low-end graphics processing processors [24]. Furthermore, TL reduces generalization error and allows DL models to perform effectively on fresh and previously unknown data. Overall, TL improves performance, efficiency, and generalization when creating DL models [26].

We have chosen to use a TL model based on the MobileNetV2 architecture for our thesis due to its exceptional performance in image classification tasks and compatibility on mobile and embedded devices, making it suitable for use on any device with low computational power [23, 52]. The use of the MobileNetV2 model in plant leaf disease identification and classification images on mobile or edge devices provided benefits such as outstanding performance, effectiveness, and instantaneous evaluation, allowing for efficient choice-making and intervention in farming environments [52]. This deep architecture enables the model to learn complex characteristics and patterns from images, which is required to accurately classify images. Our main motivation for using the MobileNetV2 model in our thesis is its ability to achieve high performance with less computation power when compared to other TL models and its usefulness for resource-constrained situations like mobile devices and edge computing systems.

For our thesis, we plan to develop a DL model using the TL approach to detect and classify plant leaf diseases accurately. We recognize that traditional methods are time-consuming, but with our model, the identification process will be automated. We will be using the New Plant Disease dataset [7] from Kaggle, which includes 87,000 RGB images of healthy and diseased plant leaves.

The techniques used for the thesis are adding additional layers, adjusting the learning rate, and various data augmentation operations. These techniques are chosen for this thesis because they solve the challenges like capturing complex features, optimizing the training process, and addressing the limited availability of training data [33, 40]. We are going to experiment with three distinct techniques and identify the most efficient technique for the MobileNetV2 model, which will increase the model's accuracy.

This thesis aims to enhance the performance of the MobileNetV2 model in plant leaf images for disease identification and classification. The focus is on determining the most efficient technique for enhancing the performance of the model. This benefits in disease prevention, increased productivity, and disease control. We believe that our technique will make a significant contribution to plant pathology and agriculture.

1.1 Aim and objectives

1.1.1 Aim

The main aim of this thesis is to improve MobileNetV2 DL's performance for identifying and classifying plant diseases in images of plant leaves by experimenting with various techniques and analysing each technique against the model.

1.1.2 Objectives

1. To split the dataset of plant leaf images into train and validation data.
2. To build and train the MobileNetV2 model DL with Keras on the training dataset of plant leaf images.
3. Experimenting with techniques like learning rate, adding additional layers, and various data augmentation operations on the model.
4. To evaluate each model's performance based on relevant metrics, compare each model's performance and select the most appropriate technique among those techniques to improve model accuracy.

1.2 Ethical, societal and sustainability aspects

We make sure that the results of this thesis are used to assist farmers since the main goal of this thesis is to improve the performance of the MobileNetV2 model in the plant leaf disease detection task. This implies that the model accurately predicts the diseases in plant leaves. We make sure that the results are used in a way that is advantageous to farmers and does not harm them. Instead of harming plants or spreading diseases.

Our primary goal is to enhance the MobileNetV2 model performance with the most efficient technique, which will enable the model to predict the outcomes very accurately. This model allows users to find fast and easily infected plants to minimize their crop damage. Early identification of plant dis- ease can avoid using pesticides and other chemicals on plants that impact negatively on plants.

1.3 Research Question

RQ1: Which would be the most efficient technique among adjusting the learning rate, adding additional layers, and implementing various data augmentation approaches in optimizing the MobileNetV2 DL model for plant leaf disease image classification tasks in images?

Motivation: The motivation for RQ1 is to improve the accuracy and efficiency of the model in detecting and classifying plant leaf diseases in images. We can improve the model's performance by choosing the most efficient technique among them, which also helps in providing the best results for plant leaf diseases.

1.4 Scope of this thesis

This thesis employs a MobileNetV2 model to improve plant leaf disease identification and categorization in images. The goal is to identify the most efficient technique for improving the model's performance. The research will help to avoid disease and control disease in agriculture. This study has important implications for plant pathology and the agricultural sector, benefitting crop health and food production.

1.5 Outline

Our thesis is divided into seven different chapters, each chapter has its emphasis and purpose. The overview of each chapter is described below:

Chapter-1:
This chapter 1 describes the summary of the thesis, aim, and objectives, and outlines the structure of the thesis.

Chapter-2:
This chapter 2 gives background information on the primary subject of study and its implications for practice.

Chapter-3:
This chapter 3 provides a thorough review of earlier and related works on the same topic, highlighting the research need that this thesis attempts to fill.

Chapter-4:
In this chapter 4, we outline the study technique we used, including the experiment's implementation and details on the various methods used.

Chapter-5:
In this chapter 5, the results of the experiment are presented in an understandable and clear fashion.

Chapter-6
This chapter 6 examines the research study's findings and discusses the thesis's consequences.

Chapter-7
This chapter 7 summarizes the thesis and suggests future study directions.

Supplement Information
This Appendix A describes our working environment, the tools that are used in our work, and other relevant information, which is described in the chapter 2.

Chapter 2

Background

This background chapter describes to give readers a clear understanding of the topics used throughout our thesis and explains how these key terms are important to our thesis. Initially, we started with Neural Networks (NN), how NN played a big role in classifying objects by employing computer vision techniques, and how it has revolutionized various industries. We then delved into the concept of DL, which is a subset of NN, and how it has enabled machines to perform complex tasks with human-like efficiency. Next, we discussed how TL with pre-trained weights is used to detect and classify more accurately. After, we discussed adjusting the learning rate, adding layers to the architecture, and various data augmentation operations to the dataset, as well as how these techniques can be added to the model to improve its accuracy. Lastly, we discussed which evaluation metrics are used in our model. Some of the key terms related to the background chapter are explained in the supplement information chapter. This Appendix A provides additional context and details on important concepts that are related to this chapter 2.

2.1 Object detection using computer vision

Object detection is a technology that allows computers to recognize and locate specific objects in digital images and videos. It employs computer vision and image processing techniques to detect instances of class-based objects from a specific class, for example, cars, bikes, people, etc. This technology is useful in a variety of applications, including surveillance and autonomous driving, as well as medical imaging and robotics. Object identification algorithms may efficiently analyze visual data, extract key features, and properly recognize and localize items of interest by utilizing advanced algorithms and deep learning models. These models can be taught to attain astonishing levels of accuracy with the help of training datasets which include diverse instances of objects from various classes. Object detection, which combines the capabilities of computer vision with image processing, enables computers to understand and interact with the visual environment, opening the door to a variety of opportunities for creativity and problem-solving [10].

2.2 Neural Networks (NN)

An NN is a type of network that is intended to mimic how the brain functions. Artificial Neural Network (ANN) have at least two layers: an input layer, at least one hidden layer, and an output layer.

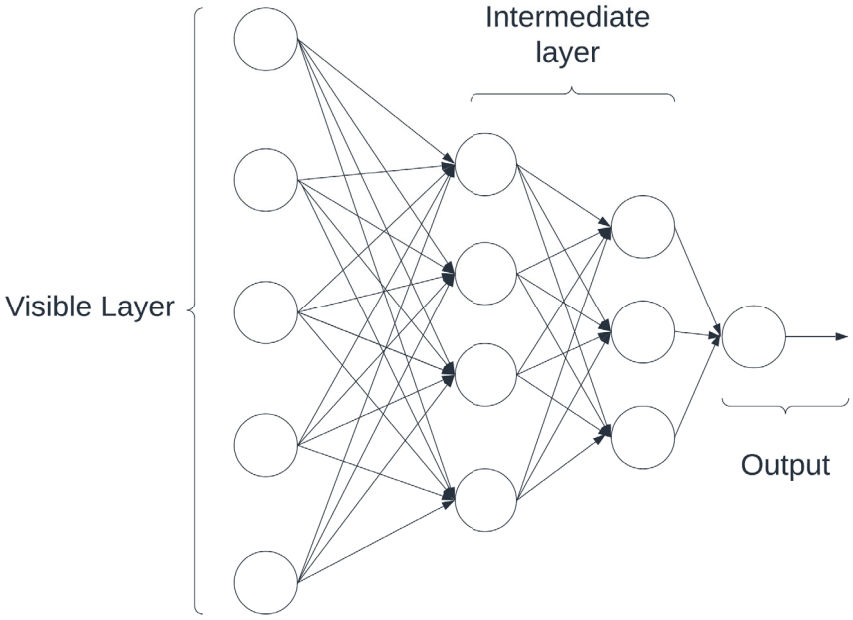

Figure 2.1: Multi-layer feed-forward network [2]

Each layer contains one or more neurons, and each neuron is linked to all the neurons in the next layer. The weights of the connections between neurons are used to calculate the input to the next layer. Each neuron's output is then calculated and passed on to the next layer. Each neural connection in a layer adds up the inputs and weights and applies an activation function to that value, along with the neuron's own bias, to generate output. This output is then sent as input to the next layer. Once at the output layer, the output is passed through another activation function, and the resulting value is the network's final output [1].

2.3 Deep Learning (DL)

DL is a type of NN composed of multiple layers, each of which has a large number of individual nodes. Due to their deeper architecture, DL networks can handle complex tasks more efficiently than traditional NN. However, due to its higher complexity, training a NN can take longer than training traditional NN. DL networks are designed to operate similarly to the human brain, enabling them to process data and make decisions in the same way as our brains do. This is accomplished by allowing the system to learn from a wide range of patterns and data kinds [49].

2.4 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN)is a type of NN that is commonly used for image recognition tasks [3,11,24]. CNNs are especially efficient because they drastically reduce computation time by using filters to progressively reduce the size of the input image. [3]. A CNN is an extension of a regular NN that is specifically designed to handle images. It typically consists of multiple blocks of convolutional and pooling

layers, which end in one or more fully-connected layers that are connected to the output layer. The convolutional layer contains several feature maps of equal size, each constructed using different filters, that are applied to the input image. After the image has been processed by the filters, it is ready to be sent to the next layer. The process of applying filters to the input image is a crucial step in the functioning of a CNN. The filters are responsible for recognizing and highlighting specific patterns or features within the image, such as edges or shapes. This feature extraction process helps CNN identify and classify images accurately. While it's possible to directly send the processed image to the fully connected layer, the computation time would suffer greatly because the image is almost fully scaled and multiplied by the number of filters used. Therefore, it's more efficient to include one or more pooling layers to further reduce the size of the image before sending it to the fully-connected layer. Overall, CNNs complex architecture is what enables it to achieve exceptional performance in image-related tasks such as object recognition and segmentation [4].

2.5 Transfer Learning (TL)

TL is an approach in which knowledge obtained from training on one type of problem is transferred to training on other related tasks or domains [35]. It is crucial in Deep Convolutional Neural Network (DCNN) since DL algorithms require a large amount of labeled data to train the models, and gathering a large labeled dataset in a domain is typically difficult. This approach is used to build high-performance classification networks for given data by using a pre-trained network on a large dataset [57]. It has the advantage of reducing training time, generalization error, and the computational cost of developing a DL model [8].

2.5.1 ImageNet Dataset

ImageNet dataset is a large visual database that is specially designed for the use of deep neural networks for image recognition. To improve the accuracy of image classification tasks, DL models are pre-trained on the ImageNet dataset, which contains millions of images from thousands of categories. This pre-training helps the models learn general features that can be useful for a wide range of image recognition tasks. TL is then used to adapt these pre-trained models to a new task, fine-tuning them on a new dataset. By leveraging the knowledge gained from pre-training on ImageNet, TL allows for faster and more accurate training on new datasets [56].

However, in our thesis, we are using the Imagenet dataset to transfer knowledge of the pre-trained weights of plant leaf diseases while creating the MobileNetV2 model step.

2.5.2 MobileNetV2 model

MobileNetV2 is a highly adaptable CNN model that is widely utilized in computer vision applications. Its lightweight architecture and efficient operations make it ideal for devices with limited resources such as mobile phones and embedded systems. It

is extensively used in picture classification, object detection, semantic segmentation, and TL applications. Its amazing blend of efficiency and accuracy enables accurate item recognition, identifying plant leaf diseases, and facial expression analysis. MobileNetV2 supports TL by exploiting pre-trained weights on large-scale datasets like ImageNet, decreasing training time and improving performance on given datasets. As a whole, its versatility, compatibility with varied workloads, and capacity to run effectively on mobile and embedded devices have solidified it as a powerful option for DL and mobile vision applications [43].

MobileNetV2 is a CNN architecture that is optimized for use on limited computational resources. This network employs an inverted residual topology, implying that residual connections exist between bottleneck levels. As its input, the network uses a compressed representation of the input image, allowing it to operate with fewer computations. The resulting image is then expanded to a higher dimension before being filtered with a compact depthwise convolution. This reduces the number of parameters that the network must calculate. Finally, using a linear convolution, the features are projected back into a low-dimensional form. This enables the network to process images more efficiently [42].

MobileNetV2 architecture

The whole MobileNetV2 model architecture consists of 53 layers (17 of these building blocks in a row. This is followed by a regular 1×1 convolution, a global average pooling layer, and a classification layer) [21]. .

1. **Building Block:** The building blocks are the basic units of the MobileNetV2 architecture. They are made up of a series of steps that are performed several times. These blocks contain depthwise separable convolutions, which are effective convolutions that factorize the traditional convolution. The building blocks allow the model to acquire more advanced interpretations as data moves through the layers by capturing and refining characteristics hierarchically. There are two types of blocks used in this architecture:

   (a) **Stride=1 block:** A residual block with a stride of 1 is a block in MobileNetV2 that keeps the feature map size the same as the input size. This block is made up of two types of convolutions, and information is added directly back to the input. This helps to keep the image's essential details and makes the network more efficient.

   (b) **Stride=2 block:** A downsizing block with a stride of 2 is a type of block used in MobileNetV2 to cut the size of the feature map in half. This is accomplished through the use of depthwise convolution, which reduces the size of the feature map. The output of the depthwise convolution is then transformed using a pointwise convolution. This block does not skip the connection but instead connects the pointwise convolution output directly to the next block.

2. **Convolutional layer (conv) :** In MobileNetV2 architecture, the 1x1 convolution is used in the bottleneck block to reduce the number of channels in

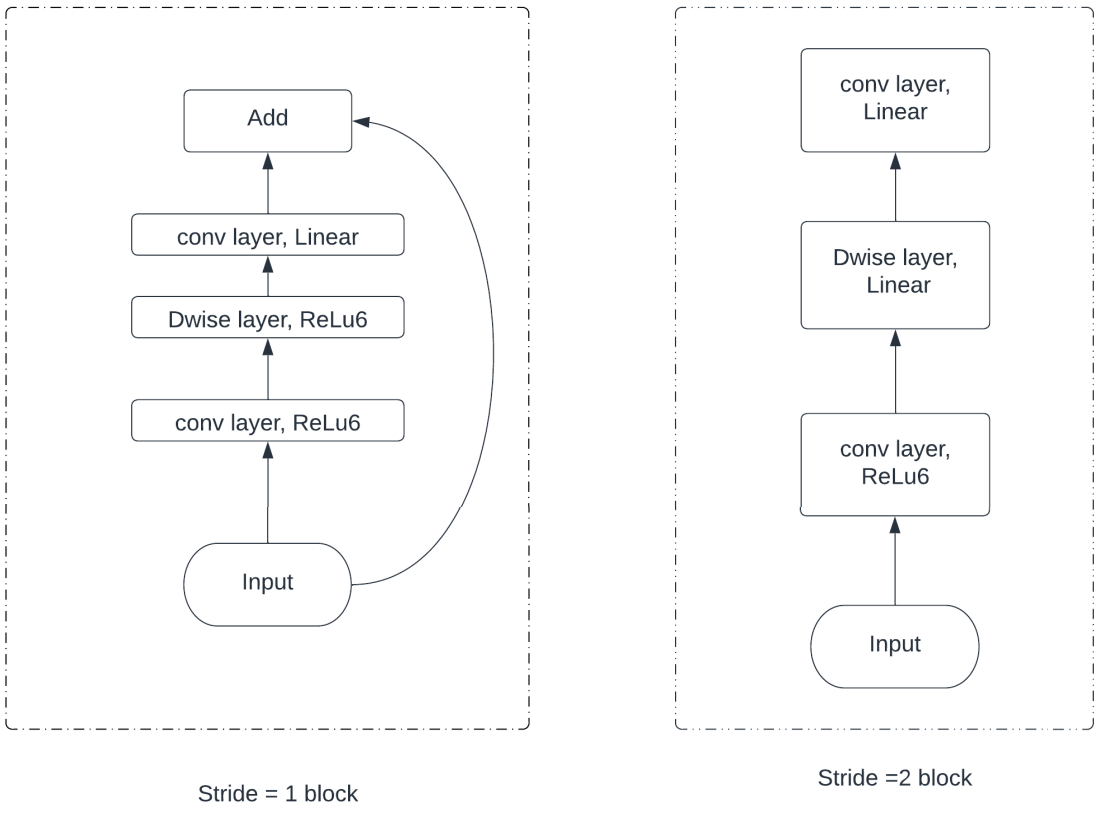

Figure 2.2: MobileNetV2 architecture, which was influenced by this article [42]

the feature map, which reduces the number of parameters and computation required to process the input feature map. A 1x1 conv is a type of convolutional layer in a NN that uses a 1x1 filter to process the input feature map. The 1x1 convolution is a powerful tool in designing NN, as it can perform several different operations on the input feature map.

3. **Depth wise convolutional layer (Dwise) :** In MobileNetV2 architecture, the second layer in each block is a depthwise convolution. Depthwise convolution applies a single filter to each input channel separately, which is different from traditional conv that applies a different filter to each input channel. By doing so, the Dwise reduces the number of parameters and computations needed in the model, making it more lightweight and computationally efficient.

4. **1X1 convolution:** After the repeated building blocks, a conventional 1X1 convolution layer is applied. Convolution is performed by moving a 1x1 kernel across the input feature maps and determining a weighted total of values within each receptive field. The goal of this 1X1 convolution is to further improve the learned characteristics by mixing and modifying data from other channels.

5. **Global Average Pooling:** Following the 1X1 convolution layer, a global average pooling layer is used. This pooling procedure calculates the mean value of each feature map over its spatial dimensions (width and height). Global average

pooling generates a compact representation of the characteristics by decreasing the spatial dimensions of the feature maps to a set of sizes independent of the input image size.

6. **Classification Layer:** The global average pooling layer's output is subsequently delivered to the classification layer. This layer associates newly discovered features with certain classifications or groups. In the classification layer, a softmax activation function is usually utilized to generate end probabilities for each class, showing how likely it is that the input image belongs to each class.

2.6 Hyper parameter tuning

Hyperparameter tuning is an important phase in the development of DL models since it has a large impact on the model's performance. The hyperparameters are the settings that govern the model's behavior, like the learning rate, batch size, and the number of hidden layers. In order to achieve optimal performance, it is crucial to carefully select the hyperparameters that best suit the specific problem. Tuning these hyperparameters entails determining the most suitable combination of values that will lead to the best possible results [6].

2.6.1 Adjusting learning rate

Adjusting the learning rate is a key approach used in NN training to improve its performance. The learning rate is one of the hyperparameters that governs how often the model weights are changed during training. By adjusting the learning rate, the model may be better optimized, and accuracy or convergence can be improved [6].

Advantages of adjusting the learning rate [60]:

(a) **Faster convergence:** By modifying the learning rate, the model can arrive at the best solution faster, reducing overall training time.

(b) **Better optimization:** A well-tuned learning rate can assist the model in better optimizing the loss function and improving its accuracy.

(c) **Enhanced generalization:** By employing a lower learning rate, the model is less likely to overfit the training data and can better generalize to new data.

(d) **Tuning Model performance:** Model performance can be fine-tuned by iteratively modifying the learning rate and observing the influence on the model's performance.

Although adjusting the learning rate in a model is challenging, the main goal is to reduce the loss of function. However, by recursively adjusting the learning rate, there can be a minimum loss of function. On the other hand, if the learning rate is set too low, the model could become stuck at a local minimum and fail to converge to the global minimum. This could result in a suboptimal

model with poorer performance than a model trained at a higher learning rate.

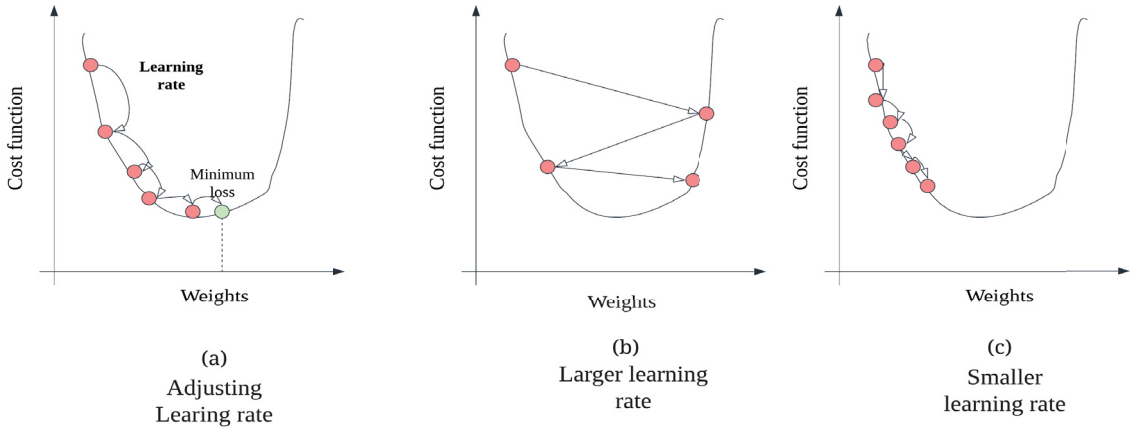

Figure 2.3: Figure 2.3 (a) depicts adjusting the learning rate to find the minimum loss function. When there is a higher rate, as shown in Figure 2.3 (b), and similarly for a lower learning rate, as shown in Figure 2.3 (c) [27].

2.7 Adding Additional Layers

Adding additional layers to NN is one of the common methods for improving the performance of models. With additional layers, the model can acquire advanced representations of the input data, resulting in more accurate predictions [48]. This technique can be helpful in the prevention of overfitting, which happens if the model becomes too complicated and starts capturing the noise in the data rather than the underlying patterns. Convolutional layers, pooling layers, dropout layers, and normalizing layers are some of the layers that can be added to NN. Every layer has a distinct purpose like decreasing the spatial dimension of the input data or injecting randomization to avoid overfitting [51].

There are so many layers in DL and NN in a model architecture, but with careful design and optimization, Some of the famous layers are the convolutional layer, dense layer, and pooling layer. Each layer is described briefly in the following points:

(a) **Convolutional layer (conv):** A convolutional layer is the first layer in CNN architecture, and it is the core building block of CNN architecture that performs convolutional operations on the input data. The convolutional layer is followed by the polling layer or fully connected layer. By adding a convolutional layer to the CNN architecture, it increases its complexity, capturing complex patterns in images. In the convolutional layer, a convolutional operation is applied to the input, and afterwards, the result is passed on to the next layer. This operation converts all the pixels in its field into one-pixel value [29].

(b) **Dense Layer :** A dense layer connects every neuron from the previous layer to every neuron in the current layer, enabling a comprehensive analysis of the input data. It is often used in the final layers of a model for predictions. Matrix-vector multiplication is performed by the dense layer, where matrix values are trainable parameters updated through backpropagation. The dense layer output is a vector of size 'm', which can be used to modify the vector's dimensionality. The number of neurons in the dense layer determines the output dimensionality [46].

(c) **Pooling layer :** This layer is used to decrease the spatial dimension of the input data by downsampling it. It is applied to each feature map of the input volume independently, and the volume depth is always preserved during the pooling operation. The most common type of pooling layer is max pooling, which takes the maximum value within a certain window of the input data. This helps reduce the number of parameters in the model and prevent overfitting [37].

2.8 Data Augumentation

Data augmentation is a technique for creating new training data from existing training data. It is an effective approach that is commonly utilized in computer vision and Natural Language Processing (NLP) tasks [47]. Data augmentation helps in reducing overfitting problems and improving the generalization of NN [36]. By producing variations of the original data, the dataset grows in size, which aids in preventing overfitting by giving a broad set of instances for the model to learn from [47].

There are many data augmentation operations that can be applied to a training data set. Some of the most well-known data augmentation operations are rotation, horizontal flip, vertical flip, colour jitter, and Random Selection of a Single Image from the training dataset.

(a) **Random rotation :** Random rotation is a common data augmentation operation that involves rotating the input data by a random angle. This helps the model learn different orientations of objects in the training dataset and can improve its ability to generalize to new data. By applying various data augmentation techniques, we can increase the variation of the training data and improve the robustness of the model [47].

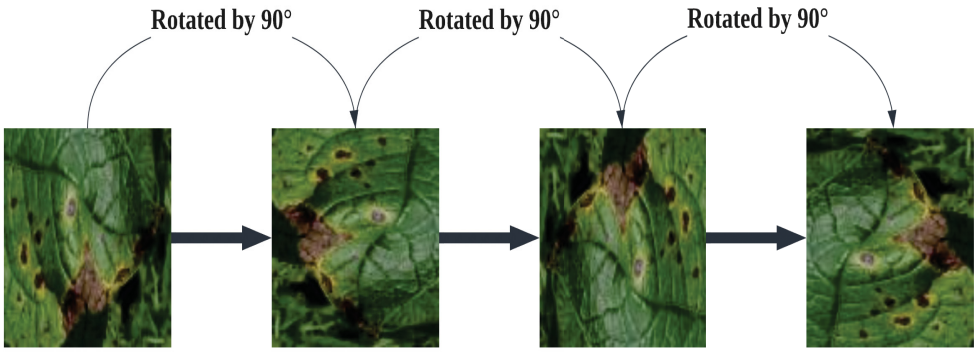

Figure 2.4: Data augmentation techniques were used to generate a random rotation of a plant leaf image from our training dataset.

(b) **Horizontal flip and Vertical flip :** Horizontal and vertical flips are also common data augmentation operations that involve flipping the input data horizontally or vertically. This helps the model learn different orientations of objects in the training dataset and can also improve its ability to generalize to new data [47].

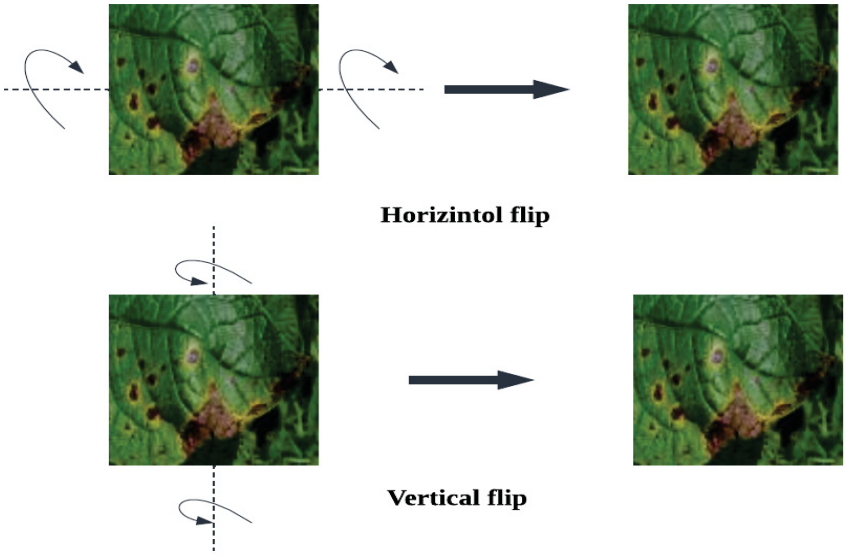

Figure 2.5: A plant leaf image from our training dataset after undergoing horizontal and vertical flipping data augmentation operations.

(c) **Random Selection of a Single Image from the training dataset :** The process of randomly selecting one image from the training dataset is referred to as the random selection of a single image. This selection is frequently used to feed a model during the training phase. During each training iteration or epoch, the image is typically chosen at random. It helps in the diversification of the data presented to the model, preventing it from memorizing or overfitting specific examples. The model is exposed

to different instances of the data by randomly selecting images, which improves its generalization capability [47].

(d) **Colour Jitter :** Another technique that can be used to augment training data is color jitter. This involves randomly adjusting the brightness, contrast, saturation, and hue of an image to create variations of the same image. This technique can help the model learn to recognize objects under different lighting conditions and colour variations, making it more robust and accurate in real-world scenarios [47].

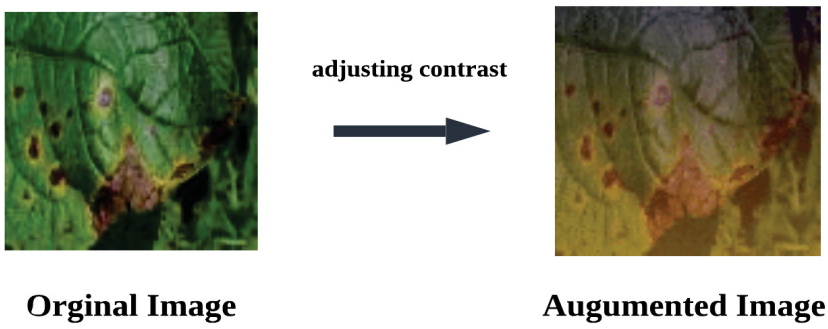

Figure 2.6: In the training dataset, a plant leaf image is used to adjust the image's contrast.

2.9 Evaluation Metrics

The evaluation metrics are used to measure the performance of a model. These measures are useful to determine how well the model predicts the results and to find which one performs most efficiently [22].

The symbols below are helpful for the formulae of different evaluation metrics:

(a) True positive (tp) and True negative (tn) denote the number of positive and negative instances that are true classified.

(b) False positive (fp) and False negative (fn) denote the number of positive and negative instances that are falsely classified.

2.9.1 Accuracy

Accuracy metric measures the proportion of true classified instances to the total number of instances [22]. The formula for the calculation of the accuracy metric:

$$Accuracy(A) = \frac{tp + tn}{tp + tn + fp + fn} \tag{2.1}$$

2.9.2 Precision

Precision metric measures the proportion of the number of true positively classified instances to the total number of positively classified instances [22]. The formula for the precision metric:

$$Precision(P) = \frac{tp}{tp + fp} \tag{2.2}$$

2.9.3 Recall

Recall measures the proportion of positive instances that are correctly classified to the total number of correctly classified instances [22]. The formula for the recall:

$$Recall(R) = \frac{tp}{tp + tn} \tag{2.3}$$

2.9.4 F1-Score

F-Score measures the harmonic mean of precision and recall [22]. The formula for the F1-Score:

$$F1 - Score(F1) = F = 2 * \frac{P * R}{P + R} \tag{2.4}$$

Chapter 3

Related Work

3.1 Related Work

Rout et al. [39] utilized computer vision techniques for disease diagnosis and classification in plant leaves. By utilizing computer vision techniques, they successfully detected powdery mildew, tomato yellow leaf curl virus, citrus greening disease, and soybean rust diseases in plant leaves. Despite identifying diseases in plant leaves, they faced some challenges, like obtaining high-quality images that accurately captured the characteristics of healthy and diseased leaves. This requires specialized equipment and expertise in image acquisition, and another challenge is developing algorithms that can accurately detect a wide range of diseases across different plant species. This requires extensive training data and a careful selection of features that are relevant across different types of diseases. Therefore, effectively to identify and classify objects, it is essential to combine image processing, ML, and computer vision techniques, as this approach helps to overcome the challenges of image segmentation, feature extraction, and classification.

Although Chen et al. [9] developed an ML model for disease diagnosis, they encountered some limitations. These included the model's inability to handle small amounts of data as well as the need for segmentation and feature extraction. Some existing research focused on using traditional ML techniques for plant leaf disease detection in images, which provide less accurate results [53]. Wang et al. [55] conducted a comparison study between traditional ML and DL algorithms for image classification. The researchers used four different datasets to analyze and compare the accuracy and time performance of the two algorithms based on sample size and picture type. Through their analysis, they found that while traditional ML image recognition models have their advantages, they also have deficiencies that can be improved upon by utilizing DL models. Thus, they propose the use of DL models to enhance the accuracy of image recognition.

For the detection and classification of plant leaf images, DL techniques could indeed model and resolve large-scale data challenges [5,33]. Among several DL approaches, DCNN have been utilized extensively for image classification [30].

But they consist of many parameters and also require a lot of computation. So, training with TL gives better results [20].

Past studies on plant leaf disease identification in plant leaf images have looked at both classic ML methods such as Support Vector Machine (SVM) and decision trees, as well as DL approaches such as CNN and TL. These approaches, however, have limits. In this research, Geetha Ramani et al. [16] proposed a unique DCNN model that was trained using data augmentation approaches and achieved a high classification accuracy of 96.46%. The model outperforms classic ML and TL methods and exhibits consistency as well as dependability, making it a significant addition to the area.

Verma et al. [54] compared the performance of four different models, MobileNetV2, MobileNet, Inception, and DenseNet, for image classification. They evaluated the accuracy and training time of each model. The results showed that MobileNetV2 achieved the highest accuracy of 99.57% with a training time of 700.23 seconds, while DenseNet had the lowest accuracy of 98.58% with a training time of 6,789.12 seconds. Although the Inception model had slightly better accuracy, its performance was not significantly better than MobileNetV2. Based on the results, the authors concluded that MobileNetV2 outperformed the other models in terms of accuracy and training time for image classification.

These are the primarily related works for plant leaf disease detection in plant leaf images using the MobileNetV2 model. Mahesh et al. [32] used the plant village dataset, which contains 54,303 healthy and unhealthy leaves, in their work to detect plant diseases in plant leaf images. They also used the MobileNetV2 model for this purpose. Despite his comparison of the MobileV2Net, ResNet50V2, and InceptionV3 models, this study concluded that the MobileV2Net model obtained an average model accuracy of 95%. Another study utilizing the MobileNetV2 model to detect plant diseases focused more specifically on tomato leaf plant leaves and used 4,671 images from the plant village dataset. Siti Zulaikha et al. [58] in order to detect three different tomato diseases, additionally employed a variety of fine-tuning techniques for the MobileNetV2 model. The MobileNetV2 model can detect the disease with greater than 90% accuracy by adjusting the learning rate, optimization method, and batch size techniques through their experimentation.

After training the model, the performance may be insufficient to classify and detect accurately. To increase the model performance there are several techniques such as hyperparameter tuning, an ensemble of algorithms, feature selection, adding additional layers, adding more datasets, changing image size, increasing epochs [12, 15]. For our project, we chose the techniques like adjusting learning rate parameter tuning, adding additional layers, and applying data augmentation techniques. These techniques have proven to be effective in deep learning tasks, particularly in image classification [17]. Adjusting the learning rate technique helps the model to converge faster and minimize the

risk of overfitting while adding additional layers increases the model's capacity to extract features from the input data [12] Data augmentation techniques improves the model's generalization ability by generating new training data points and reducing overfitting [44]. Although there are various data operations like rotation, horizontal flip, vertical flip, random crop, color jitter, and random rotation and crop. Though the combination of these operations may create more images from a small dataset, in some cases, not all combinations of these operations are equally effective in improving the performance of the model. Out of these operations, random rotation and crop combination have proven to be effective in boosting the performance of the model because they help increase the diversity of the training dataset and reduce over lifting [50]. By comparing these techniques we aim to build a model with the highest accuracy to detect plant leaf disease and classification of plant leaf images.

Research Gap

The related work shows a gap in research investigating specific techniques like adjusting the learning rate, data augmentation, and adding additional layers for plant disease identification. The purpose of using these techniques is to improve the performance of the MobileNetV2 model for plant leaf disease identification and classification in plant images. So, our thesis is to identify an efficient technique for improving the MobileNetV2 accuracy in plant leaf disease identification and classification in plant images. The successful identification of plant diseases helps prevent disease, improve crop production, and control disease.

Chapter 4

Method

For our thesis, we utilized an experimental research methodology in order to achieve the aim and answer the research question. An experiment was done on the CNN-based MobileNetV2 architecture with three different techniques tested. In order to train the model, we used the new plant leaf diseases dataset from Kaggle. This experiment was conducted after initially trained model with no technique using three different techniques and comparing each model's performance. Based on the comparison of three different models, determine the most efficient method to increase the accuracy of the model.

The procedure for this experimentation methodology is described below:

Step-1: The input for the method is the collection of plant leaf images with corresponding annotations file. The necessary data is extracted from the parsed and preprocessed data, which includes image paths, bounding box coordinates, and class labels.

Step-2: The MobileNetV2 object detection base model is built with pre-trained weights from the ImageNet dataset. Additional layers are built on top of the model's architecture to adapt it for plant leaf disease detection and classification in plant leaf images.

Step-3: The model is prepared for training by compiling it with the Adam optimizer and a mean squared error loss function.

Step-4: During the training phase, the model is trained using a custom generator that generates batches of images and their corresponding labels. The training process involves iterating across several epochs, where each epoch consists of feeding batches of images to the model, computing the loss, and adjusting the model's weights based on the gradients.

Step-5: After training the model, it is saved as a file for future use. The saved model is loaded and the test dataset is processed to evaluate the model's performance. The test dataset contains images from the test dataset as well as the annotations that go with them. The model predicts labels based on the test data, and the predicted labels are compared to the true labels to calculate evaluation metrics such as loss, accuracy, precision, recall, and F1 score.

4.1 Experimentation

We conduct experiments in our thesis to investigate various techniques for improving a plant leaf disease detection and classification model in plant leaf images. We look at how changing the learning rate, adding layers to the MobileNetV2 architecture, and using data augmentation operations on the training dataset can improve the model's performance. By doing so, we gained insights into the best strategies for improving the accuracy and reliability of our model by analyzing the results of these experiments. Through experimentation, we improved the performance of a plant leaf disease detection and classification model in plant leaf images.

Experiment - 1 : To adjust the learning rate, we experiment during the training process, which occurs between the model creation and training processes (i.e., between steps 4 and 5). and then the model is evaluated using relevant metrics.

Experiment - 2 : Adding an additional layer to the model architecture is defined during the model creation process (i.e., in step 2), after which we will repeat steps 3-5 to build this model. Once three different layers are added to the model architecture, the model will be evaluated using relevant metrics.

Experiment - 3 : Similarly, we experiment with various data augmented operations to training dataset in order to experiment with various data-augmented operations (i.e., in step 1). We build the model again by repeating steps 2 through 4. Once various data-augmented operations are done, then the model is evaluated using relevant metrics.

The steps we follow in the process are represented in the following flow diagram:

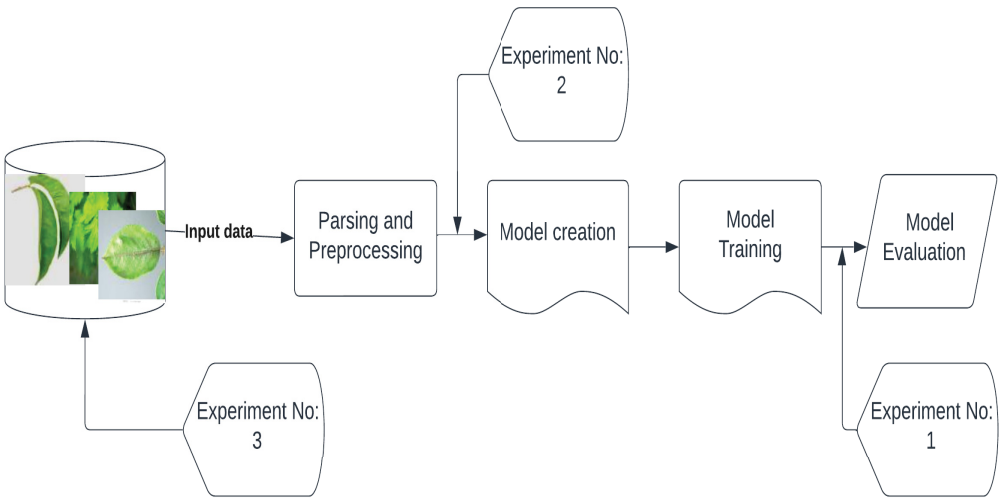

Figure 4.1: Implementation of steps

4.2 Dataset Overview

The New plant diseases dataset [7] is used for this work, and we have done experiments on it. This dataset contains approximately 87,000 RGB photos of healthy and diseased crop leaves that have been classified into 38 different classifications. The complete dataset is already partitioned into an 80/20 ratio of training and testing datasets while maintaining the directory structure.

We decided to resample the given dataset due to a large number of images and limited computational resources for the training process. The dataset was resampled to a total of 7,600 RGB images, with 200 images in each class. We reduce the size of the dataset while maintaining a balanced distribution across classes using this downsampling method. Now we can train the model within the given time constraint and optimize computational efficiency by reducing the number of images.

4.3 Data Preparation:

The data preparation step in training a model involves getting the dataset into a suitable format for the training process. It includes various tasks such as data cleaning, data transformation, and data splitting. For getting a suitable format of data, we started by creating an annotation file. In this file, each image has its own details like image_paths, box_coordinates, Next, we loaded the training dataset and its associated annotations by defining a function that returns the dataset in a suitable form for the training process.

To create an annotation file for our dataset, we decided on the Comma Separated Values (CSV) annotation format. This format will be useful in creating annotations for each image in our dataset, which can include image information such as *image_paths*, *box_coordinates(x_min, x_max, y_min and y_max)*, and *class labels*, primarily defining our annotation schema. We gathered the images we wanted to annotate and made sure they were organized. We followed by employing the Visual Geometry Group (VGG) Image Annotator tool to manually annotate the images according to our annotation schema and saved the annotation file in CSV format. Lastly, we reviewed and validate the annotations after they had been created to ensure accuracy and consistency.

Load the training dataset containing images and associated annotations

This step provides the necessary data for training the model. The training dataset contains the collection of images along with an annotation file.

For Load the Training dataset of images we defined a function *parse_annotation_csv(file_path)*

```
def parse_annotation_csv(file_path)
```

The function *parse_annotation_csv(file_path)* considers a file path as input which is basically a CSV file located at that path. The CSV file contains annotations of images, with each row presenting an annotation for a specific image. This function returns a dictionary of annotations.

4.3.1 Experimenting with various data augmentation operations to training dataset

In our thesis, we want to see how different data augmentation operations affect the performance of our model for plant leaf disease detection and classification in plant leaf images. We will test various augmentation techniques such as **rotation**, **horizontal flip**, **vertical flip**, **random crop**, **colour jitter**, and **random rotation and crop**. These techniques are commonly used to improve model accuracy and generalization by increasing the robustness of the training data. The code below shows how we experiment with various data augmentation operations for our model.

To experiment with various data augmentation operations in Keras, we followed the steps described below.

(a) We define data augmentation operations to be performed in the model by assigning each operation to a different variable.

```
rotation=30
h_flip=True
v_flip=True
```

(b) After initialising variables, we created a **datagen** object of the Instance of *ImageDataGenerator()*

(c) We generated augmented images by using **datagen** object

```
augmented_images = datagen.flow(x_train,y_train,
batch_size=32)
```

Once the augmented images are generated, these images are combined with the training dataset to create an augmented training dataset. This dataset includes both original and enhanced images. This augmented dataset is treated as the original training dataset, which is again used for training the model (which is basically step-3).

4.4 Model Creation

After the data preparation step is done, the MobileNetV2 model with pre-trained weights from ImageNet will be created by importing the necessary libraries, loading the MobileNetV2 architecture, and integrating the pre-trained

weights into the architecture. Additional layers can be added to this model by specifying them.

Define the MobileNetV2 model architecture for the detection and classification of plant leaf diseases in digital plant leaf images

The architecture of the MobileNetV2 model for plant disease identification and classification can be defined using the Keras Application Program Interface (API). Defining a function *create_ mobile netv2_ model(num_ classes).*

```
def create_mobilenetv2_model(num_classes)
```

This model is initialized with pre-trained weights from the ImageNet dataset, which aids to leverage the knowledge learned from a large-scale image classification task.

The need for this function is to define the MobileNetV2 model architecture for plant leaf disease identification and classification in plant leaf images. By using a pre-trained model as the base, we benefit from its ability to obtain meaningful features from images. The additional layers added on top of the base model allow us to adapt the model for specific classification tasks. The softmax activation in the final layer enables us to extract class probabilities, providing a prediction for each class. This architecture sets the foundation for training and fine-tuning the model to accurately classify plant leaf diseases based on input images.

4.4.1 Experimenting by additional layers

We are conducting experiments in our thesis to improve the performance of our plant leaf disease detection and classification model by adding additional layers. We examine the impact of adding a dense layer, a pooling layer, and a convolutional layer to the existing model architecture. We believe to find out how these extra layers contribute to the model's accuracy and effectiveness in identifying and categorizing plant leaf diseases through these experiments.

Before experimenting with adding various layers, we had to download the Dense layer, Convolutional layer, and Polling layer from **tensorflow.keras.layers** module into our working environment, After downloading we add layers to architecture by **model.add()** method. The code below shows how we add layers to our model by using the function **add()**.

```
model.add(Dense(64, activation='relu'))
# model.add(MaxPooling2D((2, 2)))
# model.add(Conv2D(64, (3, 3), activation='relu'))
```

4.5 Training Data Generation

Training data generation is the process of creating the training dataset that will be employed to train a DL model. To do this, We define a function called *custom_generator()*. This function takes in the parameters like *image_paths*, *annotations*, *batch_size*, *num_classes*, and *target_size* and generates batches of training data.

Generating batches of training data is a common process in most DL processes for training the model. By dividing the images in the dataset into subsets of batches, we can improve the efficiency of training, enhance model generalization by reducing overlifting for each individual class, and also improve computational efficiency. Moreover, dividing the dataset into batches also allows for easier management of memory usage during training. This is particularly important when working with large datasets that cannot be loaded entirely into memory at once. We integrated the batch of 32 sets into our method and will use it in a custom generator function for generating batches of images.

```
def custom_generator(image_paths, annotations,
batch_size ,num_classes, target_size=(224,224))
```

4.6 Model Training

Model training is the process of demonstrating a DL model to a labelled training dataset in order to train it to recognize patterns and make accurate predictions. During training, the model learns to adjust its internal parameters, primarily weights, and biases, based on the input data and the desired output, which are referred to as labels.

It is essential in object detection tasks because it allows the model to learn the characteristics and visual representations of various objects in images. The model can learn to detect and classify objects accurately by training on a labelled dataset which includes input images and their corresponding class labels.

An optimizer is a method for adjusting the parameters of a DL model during the training process. Its primary goal is to minimize the specified loss function by iteratively updating the model's parameters based on training data slopes. It is specified as an argument in the **modele.compile()** method in Keras. We chose the Adam optimizer because it is a popular optimization algorithm that adapts the learning rate for each parameter based on the gradient's first and second moments. For using the loss parameter, we specify the loss function to be used during training as Mean Square Error (MSE).

```
    model.compile(optimizer=Adam(lr=0.001),
loss='mse')
```

The model is then fit to the training dataset after it has been compiled. Fitting the model involves training it on the provided training dataset, after which it learns to generalize patterns and make predictions. To fit the model, we use the **model.fit()** method, which starts the training process. The model will be exposed to training data in batches during training, and the internal parameters will be updated iteratively using an optimizer to minimize the specified loss function.

```
history = model.fit(train_generator, epochs=50,
                    validation_data=val_generator)
```

4.6.1 Experimenting by adjusting various learning rates

One of the key aspects we investigate is the impact of varying learning rates on the model's accuracy and effectiveness. We can determine the optimal settings for improved detection and classification by fine-tuning the learning rates. To investigate the accuracy and performance of the model, typically used training and validation datasets are used. The training dataset is used to optimize the model parameters during the training process, while the validation dataset is used by the model to evaluate the model's performance. For the given experiment, the model is trained using the training dataset, and the learning rate is adjusted for each parameter. Afterwards, the model's performance is evaluated on the validation dataset.

To do so,

(a) We set the values for learning rates and epochs by using a list.

```
l_r =[0.1,0.01,0.001]
epochs_list =[10,20,30,40,50]
```

(b) For each Learning rate in *l_r* and for each number in the epochs in the *epochs_list* we created an instance of the model and compile the model using the Adam optimizer

```
obj = keras.optimizers.Adam(learning_rate=l_r)
model.compile(optimizer=obj)
```

(c) To train the model using training and validation dataset for given epochs list we used **model.fit()** method.

4.7 Model Evaluation and Saving the Model

After compiling and fitting the model, it needs to be evaluated by training data and calculated for accuracy, precision, recall, and the f1-score metric scores. To do so, first, we need to predict the trained model on the testing dataset, in which the model takes input data from the testing dataset and generates predictions

of its corresponding targeting variables. Next, we convert the predicted probabilities into class labels for a more intuitive understanding and analysis of the model's predictions. This predict label is used to calculate accuracy, precision, recall, and f1-score in Keras by passing this parameter to the imported function.

Evaluate the trained model on the test dataset

After compiling the model, it is essential to evaluate the model by testing the dataset to validate its performance.

To do so,

(a) We evaluate the model on the test dataset by calling the **model.evaluate()** method and passing the test dataset file path and test dataset labels as arguments and storing the resulting loss value in a variable named **loss**.

```
loss, accuracy = model.evaluate(test_data,
test_labels)
```

(b) We made predictions on the test dataset by calling the **model.predict()** method and storing predicted values in a variable.

```
y_pred = model.predict(test_data)
```

(c) Next, we converted the predicted probabilities to class labels by calling the **np.argmax()** function from the NumPy library.

```
y_pred_labels = np.argmax(y_pred, axis=1)
```

Calculate metrics such as loss, accuracy, precision, recall, and F1-score

After training the model, it is important to evaluate the trained model by calculating accuracy, precision, recall, and F1 scores, in order to ensure that the model has learned the desired behaviour and is able to accurately predict the labels for new data.

To calculate,

(a) We imported *precision_recall_fscore_support()* function from the **sklearn.metrics** module.

(b) We Calculate the precision, recall,accuracy and F1-score by calling the *precision_recall_fscore_support()* function for testdataset and stored in respective variables.

However, calculating the accuracy metric score while already done by evaluating the model by the test dataset step.

Save the trained model

After calculating the metrics of the model, we saved the model because enables us to reload the model later without the need for retraining, making it convenient for deployment, inference, or further fine-tuning. To save the model we call **model.save('model.h5')**, where 'model.h5' is the chosen file to save our model and '.h5' indicating a Hierarchical Data Format file.

Chapter 5

Results and Analysis

5.1 Presentation of results

This thesis aimed to develop a plant leaf disease classification and detection model and improve its performance using three different techniques. Various experiments were conducted, including exploring different learning rates, adding additional layers, and implementing data augmentation operations. The objective was to evaluate the performance of the model in terms of accuracy, precision, recall, and F1 score.

5.1.1 Evaluation results after adjusting various learning rates

In this thesis, an experiment was conducted to analyze the impact of different learning rates on the performance of a plant leaf disease classification model. Multiple learning rates were tested, ranging from 0.1 to 0.001, across different epochs. The objective was to identify the optimal learning rate that maximizes the model's accuracy, precision, recall, and F1 score. The results of this experimentation provide valuable insights into the effect of learning rates on the model's learning dynamics and overall performance.

For lr=0.1 with different epochs

We present the evaluation results for the model trained with a learning rate of 0.1 and varying epochs. The following Table 5.1 shows the performance metrics for each epoch.

Epoch	Accuracy	Precision	Recall	F1-Score
10	0.87	0.85	0.88	0.86
20	0.88	0.87	0.87	0.87
30	0.87	0.88	0.86	0.87
40	0.87	0.89	0.84	0.86
50	0.87	0.90	0.82	0.85

Table 5.1: Evaluation results for learning rate = 0.1

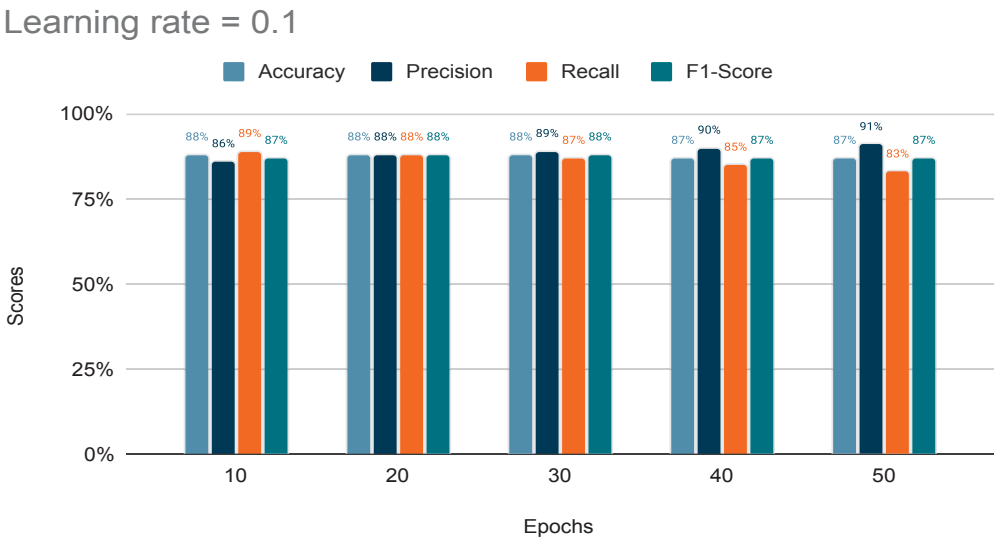

Figure 5.1: This figure depicts the comparison of accuracy, precision, recall, and the f1-score metric scores when setting the learning rate equal to 0.1 with corresponding epochs from 10 to 50.

For lr=0.01 with different epochs

We present the evaluation results for the model trained with a learning rate of 0.01 and varying epochs. The following table 5.2 shows the performance metrics for each epoch.

Epoch	Accuracy	Precision	Recall	F1-Score
10	0.88	0.86	0.89	0.87
20	0.88	0.88	0.88	0.88
30	0.88	0.89	0.87	0.88
40	0.87	0.90	0.85	0.87
50	0.87	0.91	0.83	0.87

Table 5.2: Evaluation results for learning rate = 0.01

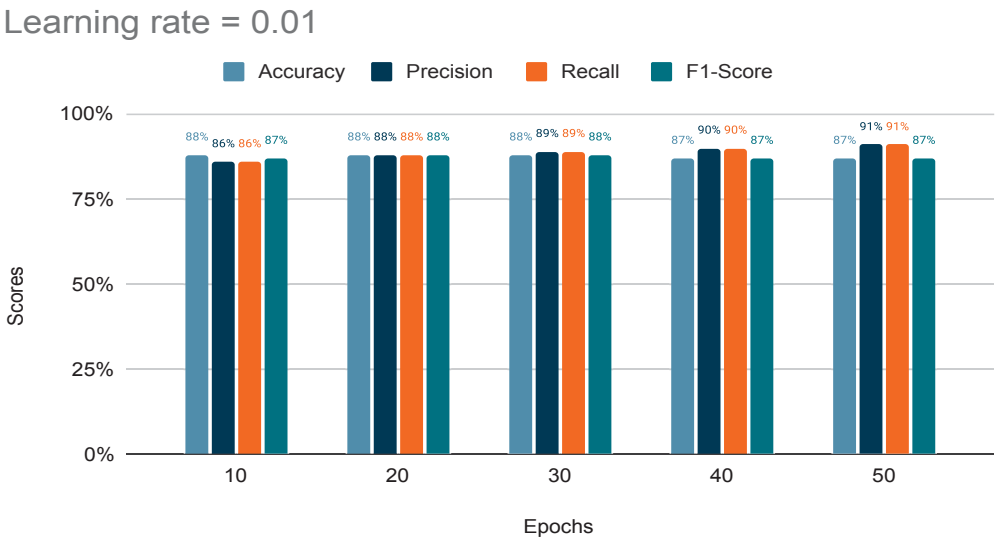

Figure 5.2: This figure depicts the comparison of accuracy, precision, recall, and the f1-score metric scores when setting the learning rate equal to 0.01 with corresponding epochs from 10 to 50.

For lr=0.001 with different epochs

We present the evaluation results for the model trained with a learning rate of 0.001 and varying epochs. The following table 5.3 shows the performance metrics for each epoch.

Epoch	Accuracy	Precision	Recall	F1-Score
10	0.87	0.85	0.90	0.87
20	0.88	0.87	0.88	0.88
30	0.88	0.88	0.87	0.88
40	0.87	0.89	0.85	0.87
50	0.87	0.90	0.83	0.87

Table 5.3: Evaluation results for learning rate = 0.001

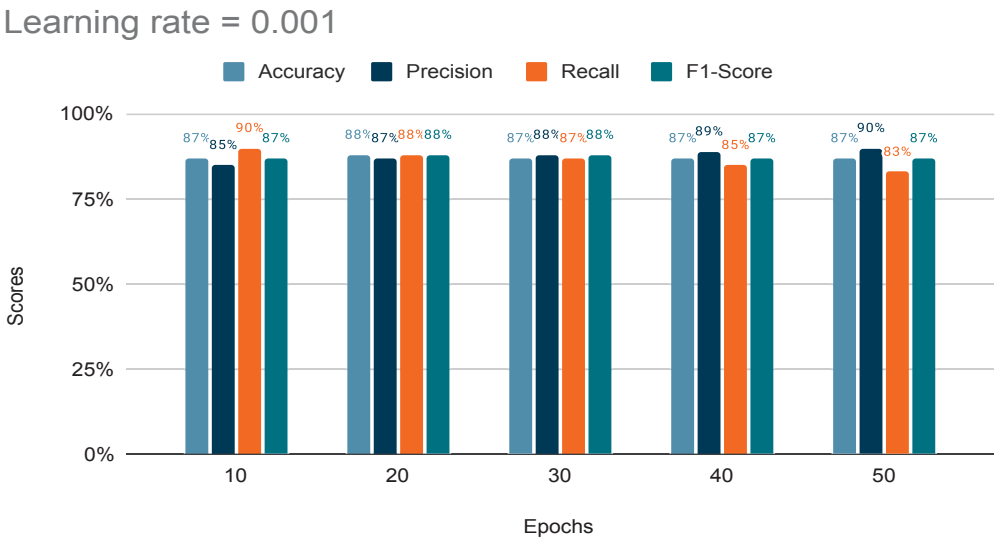

Figure 5.3: This figure depicts the comparison of accuracy, precision, recall, and the f1-score metric scores when setting the learning rate equal to 0.001 with corresponding epochs from 10 to 50.

5.1.2 Evaluation results after adding additional layers

Next, the experiment was done with the technique of adding additional layers. For this technique, we examined adding different layers like the pooling layer convolutional layer, and dense layer. The goal was to find the efficient layer that provides maximum results for the provided metrics by adding it. The results obtained with different layers are shown below in table 5.4:

Adding layers to the MobileNetV2	**Accuracy**	**Precision**	**Recall**	**F1-Score**
No additional layer	0.89	0.83	0.82	0.82
Adding dense layer	0.91	0.87	0.85	0.85
Adding pooling layer	0.92	0.85	0.83	0.83
Adding convolutional layer	0.93	0.89	0.87	0.87

Table 5.4: Evaluation results for adding additional layers

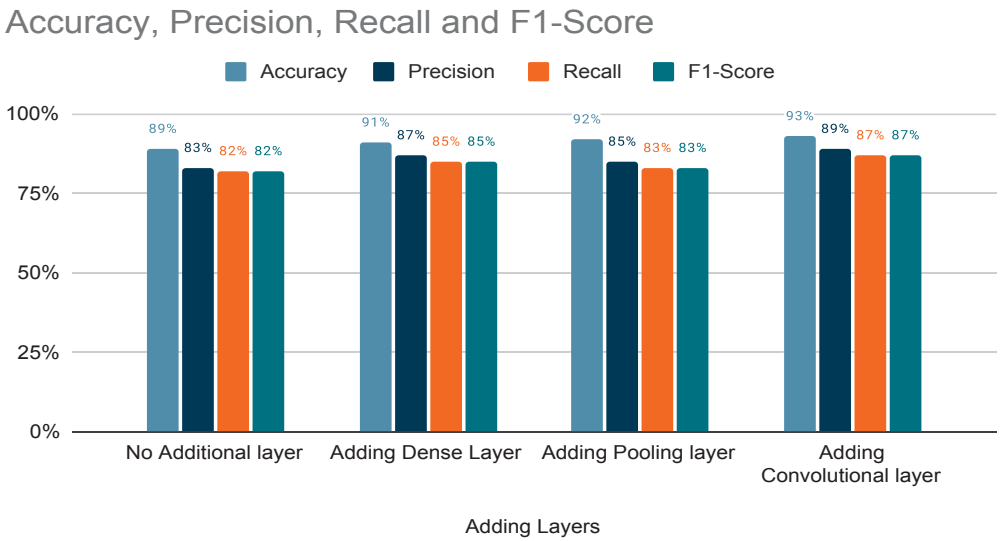

Figure 5.4: This vertical bar graph compares the accuracy, precision, recall, and f1-score metric scores when different layers are added to MobileNetV2 architecture.

5.1.3 Evaluation results after various data augmentation operations

To enhance the performance of our model, we applied various data augmentation techniques to our training dataset. The following table5.5 displays the evaluation results of implementing Technique 3, which includes rotation, horizontal flip, vertical flip, random crop, color jitter, and random rotation and crop:

Data Augmentation operations applied to the validation set of the MobileNetV2	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Rotation	0.88	0.92	0.94	0.93
Horizontal Flip	0.85	0.89	0.91	0.90
Vertical Flip	0.84	0.88	0.90	0.89
Random Crop	0.89	0.93	0.95	0.94
Color Jitter	0.87	0.90	0.92	0.91
Random rotation and crop	0.94	0.91	0.96	0.95

Table 5.5: Evaluation results for experimenting with various data augmentation operations

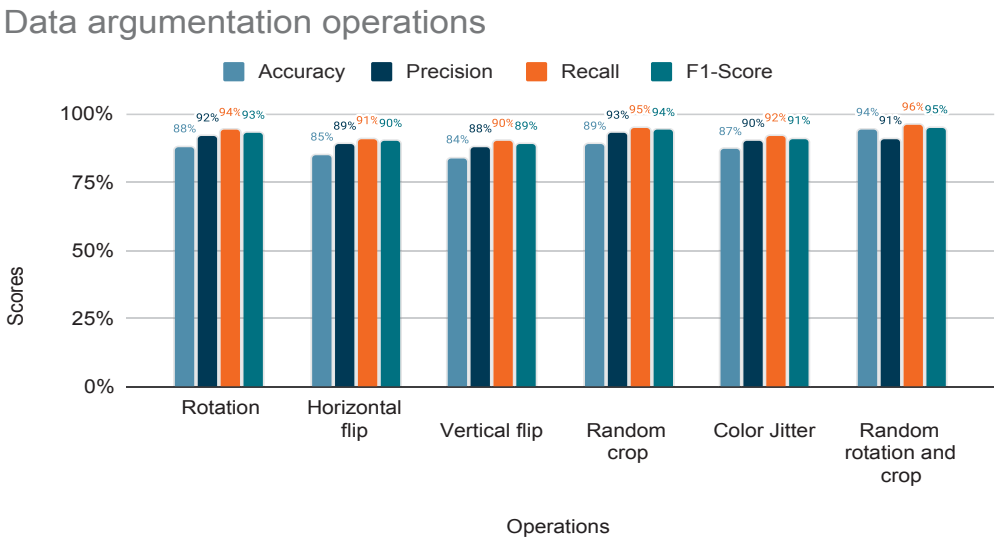

Figure 5.5: This vertical bar graph compares the metric scores of accuracy, precision, recall, and f1-score when various data augmentation operations are performed on a training dataset.

5.2 Analysis and Interpretation

From the obtained results, the following can be observed:

(a) **Varying learning rates and epochs:** The choice of learning rate and the number of epochs while training the model can significantly affect model performance. Greater learning rates enhance precision but may lead to a lower recall. Similarly, increasing the number of epochs can improve accuracy up to a certain point, after which it might also result in overfitting.

   i. With lr=0.1, the best result is obtained after 20 epochs with an accuracy of 0.88 and an F1-score of 0.87. Raising the number of epochs above 20 reduces performance.

   ii. For lr=0.01, the best result is obtained after 30 epochs with an accuracy of 0.88 and an F1-score of 0.88. Raising the number of epochs above 30 has no apparent impact on performance.

   iii. With lr=0.001, the best result is obtained after 20 epochs having an accuracy of 0.88 and an F1-score of 0.88. Increasing the number of epochs beyond 20 reduces performance.

   Overall, lr=0.01 yields the best results, and the model needs to be trained for about 30 epochs. When the learning rate was set to 0.001, the validation accuracy steadily increased but remained relatively low, indicating that the model was learning slowly and likely becoming stuck in local minima. When the learning rate was set to 0.1, the validation accuracy was inconsistent and unstable, indicating that the model was learning too quickly and likely overshooting the optimal answer. In contrast, when the

learning rate was set to 0.01, the validation accuracy gradually grew and the overall performance was significantly better. As a result, a learning rate of 0.01 is proposed for MobileNetV2. So, lr=0.01 with 30 epochs is considered to be one of the best parameter configurations.

(b) **Adding additional layers:** Adding extra layers to the model can help it perform better compared to the base model. This is because the additional layers aid in feature extraction and categorization, resulting in greater accuracy, precision, and recall.

i. The overall performance of the base MobileNetV2 model with no additional layer is not very good, with a maximum accuracy of only 0.89. This shows that the model may be too simple for capturing all of the information required for accurate categorization.

ii. Adding a dense layer to the base model enhances its performance greatly, achieving a maximum accuracy of 0.91. This implies that increasing the number of trainable parameters in the model can help it capture the required features for accurate classification.

iii. The addition of a pooling layer to the base model enhances overall performance, but not as significantly as the addition of a dense layer. The highest possible accuracy is 0.92, indicating that pooling can help the model capture specific aspects more accurately, but it may not be as useful as adding additional trainable parameters.

iv. Adding a convolutional layer to the base model boosts overall performance even more, with a maximum accuracy of 0.93. This implies that increasing the number of convolutional layers may assist the model to capture more complicated features and enhance its overall performance.

Therefore, the addition of a convolutional layer to the base MobileNetV2 model can significantly improve its performance. So, it can be one of the best parameter configurations.

(c) **Various data augmentation operations:** Implementing different data augmentation techniques, like rotation, flipping, cropping, and color modifications, results in significant improvements in precision, accuracy, recall, and F1-score. The augmentation operations help the model learn robust representations by increasing the diversity and variability in the training data, leading to enhanced generalization abilities and improved performance in classifying plant leaf diseases.

i. Applying a rotation augmentation leads to the best overall performance, with an average accuracy of 0.92. This shows that including rotation augmentation enhances the model's generalization and performance on test data.

ii. Adding horizontal and vertical flip augmentations improves model performance, but not as significantly as rotation augmentation. This implies that flip augmentations may assist with the model generalize better.

iii. Applying a random crop augmentation increases the model's performance as well, with an average accuracy of 0.93. This shows that

random cropping can help the model learn better representations of the input data.

iv. Applying color jitter augmentation improves performance slightly, having an average accuracy of 0.90. This implies that color jitter can assist the model in learning more accurate color representations of the input data.

v. Lastly, combining random rotation and crop augmentation yields the best overall performance, with an average accuracy of 0.95. This shows that adding several augmentations may assist the model to learn and adapt better representations of the input data.

Overall, the random rotation and crop augmentation apply to the validation dataset of the MobileNetV2 provided great results. So, it can also be one of the best parameter configuration techniques.

Chapter 6

Discussion

We carefully designed experiments and evaluated the performance of our proposed CNN architecture throughout the research process. We obtained results that provide insights and answers to the research questions posed by training the model on the new plant diseases dataset.

Research Question:

Which would be the most efficient technique for adjusting the learning rate, adding additional layers, and implementing various data augmentation approaches to optimize the MobileNetV2 model in plant leaf disease classification tasks in images?

Answer: We systematically explored various techniques based on the experimental results presented in the chapter5 by modifying the parameters in our model. To evaluate the model's performance, we recorded the corresponding metric scores for each technique. We identified the best parameter configuration after analyzing and comparing the results, which can be seen in the table below6.1.

Technique of best parameter configuration	**Accuracy**	**Precision**	**Recall**	**F1-Score**
When adjusting Learning rate = 0.01 and epoch=30	0.88	0.89	0.82	0.82
When adding Convolutional layer to MobileVNetV2 architechture	0.93	0.89	0.87	0.87
When adding Random rotation and crop to validation dataset	0.94	0.91	0.96	0.95

Table 6.1: Technique with best parameter configuration vs overall metrics score

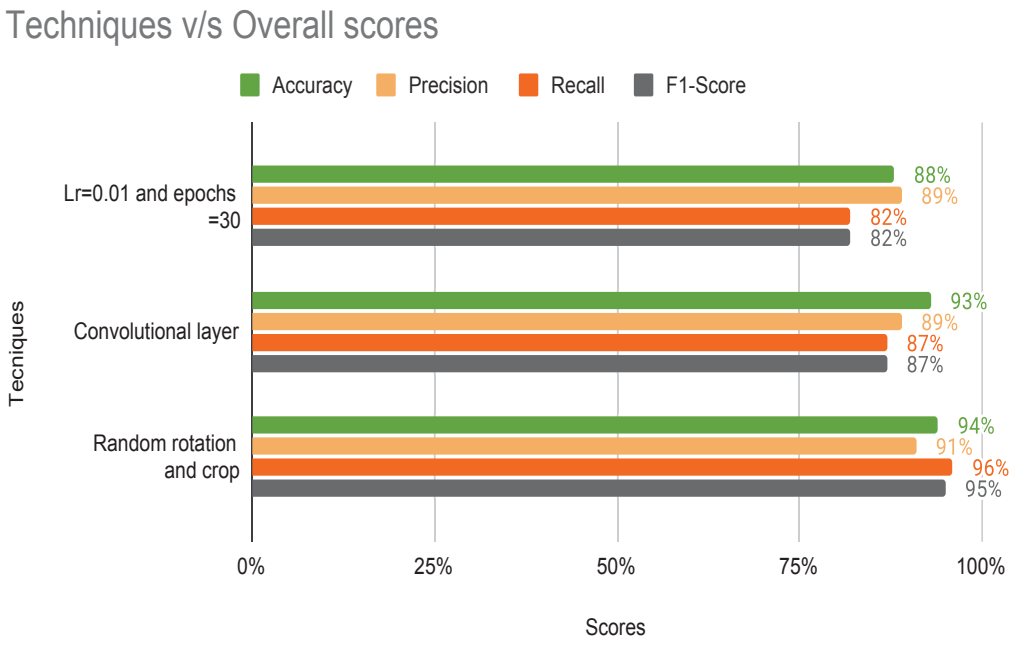

Figure 6.1: A horizontal bar graph used to visually represent the performance of each parameter configuration. This graph compares the metric scores obtained by different parameter configurations, making it easier to determine the best parameter configuration.

6.1 Discussion of Experimental results

(a) **Learning rate:**

- The model trained with a learning rate of 0.01 performed well, achieving metric scores of 87%, 89%, 82%, and 82%. These results indicate that the model performed well throughout the evaluation process.
- This suggests that the selected learning rate is suitable for this task, allowing the model to converge and generalize well on the given validation data set.

(b) **Additional Convolutional Layer:**

- The model enhanced with an additional convolutional layer showed a significant improvement in accuracy, achieving a metric score of 93%, 89%, 87%, and 87%.
- This indicates that the added layer helped in capturing more complex patterns and features within the leaf images, leading to better classification performance.
- The increased depth and capacity of the model allowed for a more refined representation of the input data, resulting in improved accuracy.

(c) **Random Rotation and Crop Data Augmentation:**

- The technique of applying random rotation and crop as a data augmentation parameter resulted in a metric score of 94%, 91%, 96%, and 95%.
- This augmentation technique introduces variations in the training data, simulating real-world scenarios where leaves may be in different orientations or partially visible.
- By exposing the model to a diverse range of leaf images, the model became more robust and better equipped to handle variations in the input data, leading to improved accuracy.

Adjusting Learning Rate: Adjusting the learning rate with 0.01 and 30 epochs did not result in major improvements in the efficiency of the model. A quicker learning rate can help the model converge faster, but it can also result in overfitting and decreased generalization capabilities.

Adding Additional Layer: Adding a dense layer or a pooling layer reduced accuracy slightly, while adding a convolutional layer improved accuracy slightly but not equally as the data augmentation technique.

Data Augmentation Operation: The method of applying random rotation and cropping (Data augmentation technique) into the training data assisted the model in performing well on new data and producing excellent outcomes on the test dataset. This is because it made the training data more diverse and unexpected, allowing the model to learn more efficiently and function well on unknown data. The model is subjected to diverse views and perspectives of the leaf photos by randomly rotating and cropping the pictures, which can assist it to acquire more robust features and patterns. Also, by adding more randomness and variability into the training data, this method helps to alleviate the problem of overfitting by preventing the model from memorizing the training data and instead pushes it to learn more generalizable characteristics.

However, changing the learning rate and adding a convolutional layer can assist to enhance the model's accuracy, the improvement may not be significant enough to make a major difference. These strategies have limits and are unable to compensate for a lack of diversity and quantity of training data. As a result, **random rotation and crop technique** continues to be an important method for enhancing model accuracy, particularly in cases where training data is restricted. The model can acquire more robust features and generalize better to unknown data by increasing the volume and diversity of the training data.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

Early detection of plant leaf diseases helps in disease prevention and increases productivity and disease control. The goal of this thesis was to determine the most effective method for recognizing and classifying plant leaf diseases employing the MobileNetV2 model. MobileNetV2 is a CNN architecture developed specifically for mobile and embedding visual applications. This was designed to meet the demand for effective and lightweight models that can function on resource-constrained devices such as smartphones, tablets, and Internet of Things (IoT) devices. The most important reason for utilizing MobileNetV2 for this thesis is its capability to establish an appropriate balance of accuracy and processing efficiency, which makes it suitable for real-time applications on mobile devices. Then, we explored three distinct strategies: adjusting the learning rate, using data augmentation techniques, and adding extra layers to the model. The main reasons for investigating these techniques are to reduce the problem of overfitting in the model, improve the model accuracy, and also they perform well in the image classification tasks.

We started by importing the new plant diseases dataset, which consists of photos of damaged and healthy plants as well as annotations for each. We then used the three strategies discussed above to train and assess the MobileNetV2 model. We discovered that adding convolutional layers to the model increased its accuracy by 2-3% when compared with the original model, while data augmentation approaches such as random rotation and crop enhanced accuracy by 5-6%. However, changing the learning rate had no significant effect on the model's accuracy.

Finally, the MobileNetV2 model with random rotation and crop proved to be the most efficient method for identifying and classifying plant leaf diseases in plant images. Our results demonstrated that this technique considerably increased the model's accuracy to 94% without raising its computational cost.

7.2 Future work

For future work, the research can be expanded in the following way:

(a) Tuning with more parameters like dropout rate, batch size, etc.

(b) Experimenting with these techniques on different datasets.

(c) At present, the experiment is done with a single technique, it can be extended with combinations of multiple techniques to observe how they impact the MobileNetV2 model performance.

(d) Instead of fixed scales of images for training and testing, it can be extended to test the model with multiple scales.

(e) The present method detects only static pictures, it can be extended by adapting this to detect in real-time on live video sources which will be more helpful to the farmers.

References

[1] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, no. 978, p. 3, 2018.

[2] K. I. Ahamed and S. Akthar, "A study on neural network architectures," *Comp. Eng. Intell. Syst*, vol. 7, pp. 1–7, 2016.

[3] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[4] R. Andersson Dickfors and N. Grannas, "Object detection using deep learning on metal chips in manufacturing," 2021.

[5] O. Asif, S. A. Haider, S. R. Naqvi, J. F. Zaki, K.-S. Kwak, and S. R. Islam, "A deep learning model for remaining useful life prediction of aircraft turbofan engine on c-mapss dataset," *IEEE Access*, vol. 10, pp. 95 425–95 440, 2022.

[6] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[7] S. BHATTARA. (2019) New plant diseases dataset. [Online]. Available: https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset

[8] D. Castillo. (2021) Transfer learning for machine learning. [Online]. Available: https://www.seldon.io/transfer-learning

[9] L. Chen and Y. Yuan, "Agricultural disease image dataset for disease identification based on machine learning," in *Big Scientific Data Management: First International Conference, BigSDM 2018, Beijing, China, November 30–December 1, 2018, Revised Selected Papers 1*. Springer, 2019, pp. 263–274.

[10] S. Dasiopoulou, V. Mezaris, I. Kompatsiaris, V.-K. Papastathis, and M. G. Strintzis, "Knowledge-assisted semantic video object detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 10, pp. 1210–1224, 2005.

[11] P. R. M. de Araujo and R. G. Lins, "Computer vision system for workpiece referencing in three-axis machining centers," *The International Journal of Advanced Manufacturing Technology*, vol. 106, pp. 2007–2020, 2020.

[12] J. Dsouza. (2021, November) How to improve the accuracy of your image recognition models. [Online]. Available: https://www.freecodecamp.org/news/improve-image-recognition-model-accuracy-with-these-hacks/

[13] H. Fan, S. Liu, M. Ferianc, H.-C. Ng, Z. Que, S. Liu, X. Niu, and W. W. C. Luk, "A real-time object detection accelerator with compressed ssdlite on fpga," *2018 International Conference on Field-Programmable Technology (FPT)*, pp. 14–21, 2018.

[14] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and electronics in agriculture*, vol. 145, pp. 311–318, 2018.

[15] R. Gandhi. (2022, January) How to increase the accuracy of a neural network. [Online]. Available: https://towardsdatascience.com/how-to-increase-the-accuracy-of-a-neural-network-9f5d1c6f407d

[16] G. Geetharamani and A. Pandian, "Identification of plant leaf diseases using a nine-layer deep convolutional neural network," *Computers & Electrical Engineering*, vol. 76, pp. 323–338, 2019.

[17] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[18] L. Haiyan, L. Xiruo, and Z. Hongjie, "Research on quality and product losses of soybean by grey leaf spot [j]," *Chinese Journal of Oil Crop Sciences*, vol. 27, no. 3, pp. 66–69, 2005.

[19] S. S. Harakannanavar, J. M. Rudagi, V. I. Puranikmath, A. Siddiqua, and R. Pramodhini, "Plant leaf disease detection using computer vision and machine learning algorithms," *Global Transitions Proceedings*, vol. 3, no. 1, pp. 305–310, 2022.

[20] S. M. Hassan, A. K. Maji, M. Jasiński, Z. Leonowicz, and E. Jasińska, "Identification of plant-leaf diseases using cnn and transfer-learning approach," *Electronics*, vol. 10, no. 12, p. 1388, 2021.

[21] M. Hollemans, "Mobilenet version 2-machinethink.net," https://machinethink.net/blog/mobilenet-v2/, [Accessed 23-Apr-2023].

[22] M. Hossin and M. N. Sulaiman, "A review on evaluation metrics for data classification evaluations," *International journal of data mining & knowledge management process*, vol. 5, no. 2, p. 1, 2015.

[23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[24] M. Hussain, J. J. Bird, and D. R. Faria, "A study on cnn transfer learning for image classification," in *Advances in Computational Intelligence Systems: Contributions Presented at the 18th UK Workshop on Computational Intelligence, September 5-7, 2018, Nottingham, UK*. Springer, 2019, pp. 191–202.

[25] M. A. Jasim and J. M. Al-Tuwaijari, "Plant leaf diseases detection and classification using image processing and deep learning techniques," in *2020 International Conference on Computer Science and Software Engineering (CSASE)*. IEEE, 2020, pp. 259–265.

[26] D. P. P. Javierto, J. D. Z. Martin, and J. F. Villaverde, "Robusta coffee leaf detection based on yolov3-mobilenetv2 model," in *2021 IEEE*

13th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM). IEEE, 2021, pp. 1–6.

[27] J. jordan. (2018) Setting the learning rate of your neural network. [Online]. Available: https://www.jeremyjordan.me/nn-learning-rate/

[28] C. Kevin. (2018, May) Feature maps. [Online]. Available: https://medium.com/@chriskevin_80184/feature-maps-ee8e11a71f9e

[29] R. Khedgaonkar, K. Singh, and M. Raghuwanshi, "Local plastic surgery-based face recognition using convolutional neural networks," in *Demystifying Big Data, Machine Learning, and Deep Learning for Healthcare Analytics*. Elsevier, 2021, pp. 215–246.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[31] A. Luckey, "Assessing youth perceptions and knowledge of agriculture: The impact of participating in an agventure program," Ph.D. dissertation, Texas A & M University, 2012.

[32] T. Mahesh, R. Sivakami, I. Manimozhi, N. Krishnamoorthy, B. Swapna *et al.*, "Early predictive model for detection of plant leaf diseases using mobilenetv2 architecture," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 2, pp. 46–54, 2023.

[33] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using deep learning for image-based plant disease detection," *Frontiers in plant science*, vol. 7, p. 1419, 2016.

[34] D. Munjal, L. Singh, M. Pandey, and S. Lakra, "A systematic review on the detection and classification of plant diseases using machine learning," *International Journal of Software Innovation (IJSI)*, vol. 11, no. 1, pp. 1–25, 2023.

[35] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[36] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[37] K. Rana. (2020) Pooling layer—short and simple. [Online]. Available: https://ai.plainenglish.io/pooling-layer-beginner-to-intermediate-fa0dbdce80eb

[38] P. Rosset, "Food sovereignty and the contemporary food crisis," *Development*, vol. 51, no. 4, pp. 460–463, 2008.

[39] R. Rout and P. Parida, *A Review on Leaf Disease Detection Using Computer Vision Approach*, 03 2020, pp. 863–871.

[40] A. Saleh, M. Sheaves, and M. Rahimi Azghadi, "Computer vision and deep learning for fish classification in underwater habitats: A survey," *Fish and Fisheries*, vol. 23, no. 4, pp. 977–999, 2022.

[41] M. Sandler and A. Howard. (2018) Mobilenetv2: The next generation of on-device computer vision networks. [Online]. Available: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html?m=1

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[43] ——, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[44] D. Shah. (2023, March) Data augmentation guide. [Online]. Available: https://www.v7labs.com/blog/data-augmentation-guide

[45] T. Shah. (2017) About train, validation and test sets in machine learning. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7

[46] P. Sharma. (2020) Keras dense layer explained for beginners. [Online]. Available: https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/

[47] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[49] A. G. Smith, E. Han, J. Petersen, N. A. F. Olsen, C. Giese, M. Athmann, D. B. Dresbøll, and K. Thorup-Kristensen, "Rootpainter: deep learning segmentation of biological images with corrective annotation," *New Phytologist*, vol. 236, no. 2, pp. 774–791, 2022.

[50] J. Solawetz. (2020) Getting started with data augmentation in computer vision. [Online]. Available: https://blog.roboflow.com/boosting-image-detection-performance-with-data-augmentation/

[51] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[52] S.-H. Tsang. (2019) Review: Mobilenetv2-lightweightmodelimageclassification. [Online]. Available: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

[53] D. Varshney, B. Babukhanwala, J. Khan, D. Saxena, and A. kumar Singh, "Machine learning techniques for plant disease detection," in *2021 5th International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2021, pp. 1574–1581.

[54] D. Verma, D. Bordoloi, and V. Tripathi, "Plant leaf disease detection using mobilenetv2," *Webology*, vol. 18, no. 5, pp. 3241–3246, 2021.

[55] P. Wang, E. Fan, and P. Wang, "Comparative analysis of image classification algorithms based on traditional machine learning and deep learning," *Pattern Recognition Letters*, vol. 141, pp. 61–67, 2021.

[56] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet

training in minutes," in *Proceedings of the 47th International Conference on Parallel Processing*, 2018, pp. 1–10.

[57] Y. Yuan, S. Fang, and L. Chen, "Crop disease image classification based on transfer learning with dcnns," in *Pattern Recognition and Computer Vision: First Chinese Conference, PRCV 2018, Guangzhou, China, November 23-26, 2018, Proceedings, Part II 1*. Springer, 2018, pp. 457–468.

[58] S. Z. M. Zaki, M. A. Zulkifley, M. M. Stofa, N. A. M. Kamari, and N. A. Mohamed, "Classification of tomato leaf diseases using mobilenet v2," *IAES International Journal of Artificial Intelligence*, vol. 9, no. 2, p. 290, 2020.

[59] X. Zhou and A. Lerch, "Chord detection using deep learning," 01 2015.

[60] H. Zulkifli. (2018) Understanding learning rates and how it improves performance in deep learning. [Online]. Available: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059

Appendix A

Supplemental Information

A.1 Working environment

We created a suitable working environment for our research and experimentation in our thesis. We set up our working environment by doing the following:

(a) **Installation of the Visual Studio Editor :** To efficiently write and manage our code, we installed the Visual Studio Editor, a popular Integrated Development Environment (IDE).

(b) **Programming language :** Python was chosen as our primary programming language for implementation in our thesis.

(c) **Operating System :** Windows Operating System (OS) was chosen as the operating system for our experimentation and development.

(d) **Dataset Set Acquisition :** We obtained the new Plant Disease datasets from Kaggle, a popular platform for accessing and sharing datasets relevant to various domains.

(e) **Libraries Dependencies :** To make our work easier, we imported a number of necessary libraries. These included NumPy, Scikit-learn, Tensorflow, and Keras. These libraries offer functionalities for developing and training models, managing datasets, and carrying out various data manipulation and analysis tasks. They are widely used in the field of DL.

(f) **Creation of annotation file :** To create an annotation file, we used VGG Image Annotator to extract *box_corcordinates*, *class_labels* and *image_paths paths*. Primarily, we are defining the annotation schema by using this tool.

(g) **Graphs :** After calculating accuracy, precision, recall, and f1-score metric scores, we stored these scores in a Google Sheet. Google Sheets includes a feature for displaying charts from existing sheets.

A.2 Libraries and Tools

A.2.1 Visual Studio Editor

Visual Studio Code is a lightweight and adaptable source code editor designed by Microsoft that is frequently utilized by programmers for a variety of coding languages and environments. It includes syntax highlighting, automatic code completion, debugging tasks, as well as integrated version control tools. With a thriving marketplace of extensions, developers can tailor the editor's capabilities to their own requirements. Visual Studio Code is a multi-platform development environment that is accessible on Windows, macOS, and Linux. It is noted for its rapidity and extensive assistance from the community [1].

A.2.2 Python

Python is a popular general-purpose programming language that was created by Guido van Rossum and released in 1991. Since then, it has become one of the most widely used programming languages in the world. Python's popularity can be attributed to its simplicity, readability, and ease of use. It is a high-level language that is interpreted, which means that it does not need to be compiled before it can be run. This makes it easy to write and test code quickly. Python is also an object-oriented language, which means that it supports the creation of objects that have properties and methods. This makes it easy to write complex programs that are easy to maintain and extend. Additionally, Python has a large standard library that provides many useful modules for tasks such as web development, data analysis, and scientific computing [2].

A.2.3 Kaggle

Kaggle is an online community and platform for data scientists and ML practitioners. It provides a variety of tools, such as datasets, ML competitions, discussion forums, and interactive courses. Users can analyze and investigate varied datasets, participate in machine learning challenges to solve real-world problems, cooperate with each other via conversations and code sharing, and improve their abilities through interactive classes. It provides a central location for data science enthusiasts to study, cooperate, and demonstrate their skills in a collaborative and competitive atmosphere [3].

A.2.4 Keras

Keras is a Python-based open-source NN library that is easy to use. It offers a high-level API for developing and training DL models. Keras allows users

[1] https://link.springer.com/chapter/10.1007/978-1-4842-6901-5_1
[2] https://www.python.org/
[3] https://www.kaggle.com/

to easily design and test with various NN architectures such as CNNs, RNNs, and Muilti Layer perceptron (MLP). It includes pre-defined layers, activation functions, and optimizers that can be integrated simply, as well as a chance to modify and construct custom layers and loss functions. Keras interfaces effortlessly with major DL frameworks like TensorFlow and Theano, and it stresses ease and accessibility. It is frequently employed for deep learning applications because of its simplicity of usage, versatility, and compatibility [4].

A.2.5 Scikit-learn

Scikit-learn (sklearn)is a popular open-source Python ML library. It includes a complete set of tools and methods for a wide range of ML problems, like regression, clustering, classification, and dimensionality reduction. Scikit-learn provides effective implementations of major ML techniques and enables data preprocessing, feature extraction, and model evaluation through an intuitive and consistent API. It works well alongside additional Python scientific computing packages and is compatible with frameworks such as TensorFlow and PyTorch [5].

A.2.6 Tensorflow

TensorFlow is a prominent open-source deep learning framework created by Google. It provides an extensive framework for creating and implementing machine learning models. TensorFlow allows programmers to create and train neural networks using high-level API such as Keras, or they can use the lower-level TensorFlow API for further customization. The framework enables a wide range of NN topologies, like CNN, Recurrent neural network (RNN), and transformers. TensorFlow offers distributed computing features for training models over numerous Graphic Processing Unit (GPU) or workstations. It also makes model deployment easier on a variety of platforms, such as mobile and edge devices. The TensorFlow ecosystem offers pre-trained models, datasets, and libraries that enable tasks such as image classification, object identification, and natural language processing. Compatibility with major libraries and frameworks improves interoperability [6].

A.2.7 VGG Image Annotator

VGG Image Annotator is an image and video annotation tool that was built by researchers at Oxford University. It is a free open-source tool and is available on the website. Some of the key points of the VGG annotator are that it is used to define regions in images and videos, used for object detection, image segmentation, and other computer vision tasks; it supports a variety of formats, which include Joint Photographic Experts Group (JPEG), Portable Network

[4] https://doi.org/10.1007/978-1-4842-3516-4_2
[5] http://scikit-learn.sourceforge.net.
[6] https://journals.sagepub.com/doi/10.3102/1076998619872761

Graphics (PNG), Bitmap Image file (BMP), and Tagged Image File Format (TIFF); it is used to emulate the performance of computer vision tasks; it is used to label images and videos for supervised learning; it has a simple interface, making it a great choice for simple annotation projects; and it also supports circles and ellipses for polygon labelling [7].

A.3 Terms related to chapter 2

A.3.1 Linearity versus Non-Linearity in NN

Linearity refers to the property of a model in which the output is directly proportional to the input, whereas non-linearity denotes a more complex relationship between input and output that cannot be expressed as a simple linear function.

Linearity: Linear models are frequently the simplest and most effective method. A linear model, in fact, fits a straight line to the data, allowing it to predict based on a linear relationship between the input features and the output variable.

Non-Linearity: Non-linear models, on the other hand, can capture more complex relationships between the input features and the output variable. This makes them useful in situations where a linear model may not be able to accurately represent the data.

A.3.2 Feature Maps

A feature map is the output of a particular filter applied to an input image. It is also known as an activation map or feature activation. These filters are small sections of the image that represent various characteristics. One filter may detect horizontal lines, while another may detect edges. Each filter applied to the input image generates a feature map that highlights the areas of the image that contain that particular feature. When multiple filters are applied to an input image, the same number of feature maps are generated [28].
A channel is a collection of feature maps which act as a convolutional layer's output [13].

A.3.3 Bottlenecks

In the context of NN, a bottleneck refers to a layer in the network that has a smaller number of neurons than the layers before or after it. The purpose of a bottleneck layer is to make it easier for the network to process.

[7] `https://www.robots.ox.ac.uk/~vgg/software/via/`

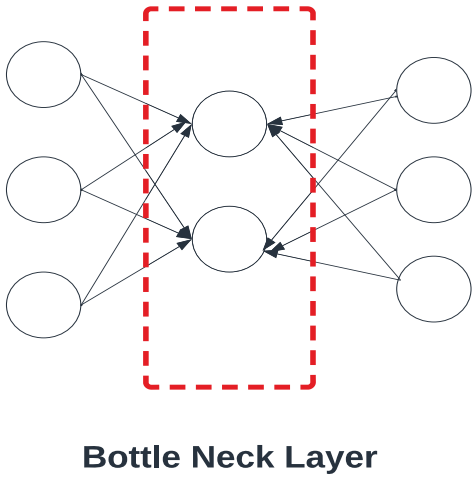

Figure A.1: Visualization of a bottleneck architecture, which was inspired from [59]

A.3.4 Activation Function

In NN, an activation function is a mathematical function that adds non-linearity to the output of the network. This function is applied to the weighted sum total of a neuron's inputs and decides whether the neuron needs to be activated or not. The purpose of the activation function is to introduce non-linearity into the network.

There are several kinds of activation functions, and each one uniquely processes data. Sigmoid, Rectified linear activation unit (ReLU), and Softmax are some examples of activation functions. Each of these functions has distinct properties and is best suited to specific tasks. The Sigmoid function, for example, is frequently used in binary classification tasks, whereas the Softmax function is commonly used in multi-class classification tasks [4].

Linear Activation Function

A linear activation function is a mathematical function that performs a simple linear transformation on the input data. Unlike other activation functions, it does not introduce any non-linearity into the network's output. As a result, it is often used in the final layer of a neural network for regression tasks, where the goal is to predict a continuous output value. In MobileNetV2, the linear activation function is used in the final layer of the network. This layer produces a vector of scores that represents the predicted probabilities of the input image belonging to different classes. These scores are calculated by multiplying the input features by a set of weights and adding a bias term. The linear activation function is then applied to this result to produce the final output scores [41].

ReLU6 (Rectified Linear Unit) Activation Function

ReLU6 is a type of activation function that is commonly used in NN. Its purpose is to prevent the activation values from becoming too large, which could slow down the learning process and cause numerical instability. The ReLU6 activation function is applied after each convolutional layer in the network. By using the ReLU6 function, non-linearity is introduced into the network, which is important for learning complex patterns in the data. In MobileNetV2, the ReLU6 function is specifically used to keep the model lightweight and computationally efficient [41].

A.3.5 Train, test and validate data sets

Train dataset: A training dataset is a set of data used to train an ML or DL model. It is a subset of the overall dataset that is used to teach the model how to recognize patterns and make predictions based on the input data [45].

Validation dataset: A validation set is used to monitor the model performance during the training process, fine-tune hyperparameters, and perform model selection [45].

Test dataset: A test set is ideally used only once at the very end of the project to evaluate the performance of the final model that is fine-tuned and selected on the training process with training and validation sets [45].

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona,
Sweden