CHAPTER 4
ANALYSIS AND DESIGN

4.1. Analysis

This chapter provides a detailed explanation and discusses one by one about the methods of solving problems that have been mentioned in several points in chapter 1. The main purpose of creating this project is how to predict potato and tomato diseases according to the dataset used so that the predictions are correct and produce a fairly high level of accuracy.

From the formulation of the problem that has been outlined, the first is how to predict disease in plants according to the dataset used. This case explains what stages are taken to build a system in order to predict correctly. As explained below:

1. The first input dataset obtained through *kaggle.com* is the PlantVillage dataset which contains images of potato and tomato leaves that have been divided into 3 different classes.
2. Pre-processing of the collected dataset.
3. Splits the data into training and testing sets using train_test_split.
4. Data augmentation on the training data using ImageDataGenerator API by Keras.
5. Uses K-Fold Cross Validation to further divide the training data into k subsets.
6. Builds CNN (Convolutional Neural Network) Model (MobileNet) to predict plant diseases.
7. Developed model will be deployed on Jupyter Notebook.

In the next problem formulation mentions about the prediction results. Plant disease detection can be performed using different classifiers and a multitude of techniques have been used in the past for this purpose. In this thesis, the classifiers that were used for performing the detection was the Convolution Neural Network (CNN). The dataset chosen for this paper contains 6,652 images of tomato and potato leaves with 3 different classes. Each classes contains the different number of images. The images of plant diseases are of the size 224x224 and the dataset does not contain any missing images. The dataset is commonly referred to as the PlantVillage Dataset.

Last problem formulation mentions about the resulting accuracy can be maximized. To improve accuracy, MobileNet is modelled after the classic VGG architecture. This involves building a network by stacking convolution layers. However, if there are too many convolution layers in a stack, gradient vanishing becomes an issue. ResNet's residual block facilitates interlayer communication by, among other things, allowing for feature reuse during forward propagation and reducing gradient vanishing during back propagation.

Mobile Net makes use of an inverted residual and a linear bottleneck within a depth separable convolution block. The depthwise convolutional layer's downsampling parameter is tweaked, and a 1x1 convolution layer is stacked on top of the depthwise convolutional layer. As an alternative to a nonlinear activation function, a linear activation is employed. The network consists of 19 layers, the middle of which is responsible for feature extraction and the lowest for classification. MobileNetV1's primary structure, depthwise separable convolution, has the effects of decreasing the network parameters and increasing the network speed. Although depthwise separable convolution produces the same output dimension as regular convolution, it splits regular convolution into a 3 x 3 depthwise convolution and an 1 x 1 pointwise convolution.

By combining the information from several channels into a single one, depth-wise convolution can drastically cut down on computation time and the number of parameters needed to describe an image. However, the final output data will not be related to any of the input channels because of the convolutional method's poor channel-to-channel information transmission. By applying a pointwise convolution, a special type of 1 x 1 convolution, to the result of a depthwise convolution, a linear combination can be generated. It is typical practise to use pointwise convolution to adjust the feature dimension of the output channel, as demonstrated in Figure 3, which can be viewed here. When compared to depthwise and group convolution, pointwise convolution is analogous to mixing information between channels. This can efficiently handle the issue of poor flow of information between channels, which is caused by convolution methods such as depthwise and group convolution.

Table 4.1. Parameters used in the CNN model

Parameter	Setting	Description
Epochs	25	Defines the number of iterations that the whole training dataset goes through the network
Batch Size	32	Refers to the number of training data that is used during each epoch before weights in the network are updated
Learning Rate	0.001	This parameter determines how much the weights have to change based on the error observed

The architecture and the parameters of the CNN discussed above that produced the best results were chosen at last. The reason the learning rate was set to 0.001 is that for complex problems such as image classification smaller learning rates are often good at producing higher generalization accuracies even though they produce larger training times it is worth the wait as they are known to improve the overall accuracy of the model. The batch size was set to 32 and model produced a test accuracy of 93.93%. The study found that increasing the batch size provided great robustness to the noise in the dataset.

4.2. Design

The use of flowchart aims to know the processes or procedures of a program that makes it easier to understand the program to be built. Flowchart system can be described as next:

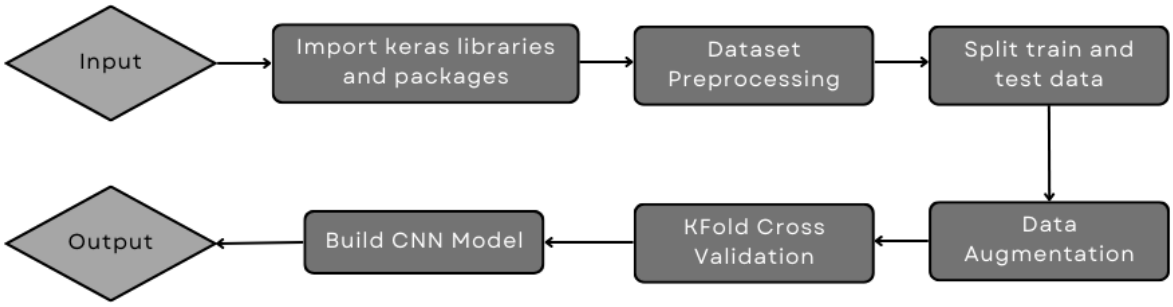

Figure 4.1 : Flowchart Design of the Program

Flowchart 4.1 is used as an overview or flow of the system itself. Starting from collecting dataset, then import keras libraries and packages as needed, then the dataset that has been prepared will go into the pre-processing stage where the dataset will be split into train and test data, next step is data augmentation which is technique for increasing the number of images in a database. By augmenting the dataset and adding distortion to the images, overfitting can be reduced during the training period. The Keras ImageDataGenerator class implements inplace data augmentation or on-the-fly data augmentation. Then, the next process is define KFold to improving model prediction. Last step is build the model and then compile the program until show the predict result. According to the description below:

1. **Input** : collecting dataset from PlantVillage Dataset.
2. **Import keras libraries and packages** : setting up libraries and packages as needed in a program built using keras.
3. **Dataset pre-processing** : The pre-processing step for any machine learning model is of great importance and ideally shapes the performance and results of the models chosen.
4. **Split train and test data** : the plant village dataset is split into two different sets, namely, train and test set with a 80:20 ratio respectively.
5. **Data augmentation** : This parameter holds the image that you want to return explanations for. The name of the setting depends on the directory in which the image is present in.
6. **K-Fold Cross Validation** : The data sample is split into ‘k’ number of smaller samples that means the sample data is being split into five and ten smaller sample respectively.

7. **Build CNN model** : Build CNN model and save the model for predict the image for plant disease detection.
8. **Output** : The resulting output is in the form of accuracy levels, accuracy graphs and loss functions, as well as prediction results from datasets that have been imported.