VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Image-based plant disease detection

By
Tran Duc Khoa

A thesis submitted to the School of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Computer Science

Ho Chi Minh City, Vietnam
Year 2021

Image-based plant disease detection

APPROVED BY:

_____,
Dr. Chan Thanh Tung, Chair

_____
Dr. Nguyen Van Sinh

_____
Dr. Mai Hoang Bao An

_____
Dr. Nguyen Thi Thanh Sang

_____
Assoc. Prof. Nguyen Thi Thuy Loan

THESIS COMMITTEE

ACKNOWLEGMENTS

It is with deep gratitude and appreciation that I acknowledge the professional guidance of Assoc. Prof. Vo Thi Luu Phuong. Her constant motivation and support helped me to achieve my goal.

My gratitude goes to my collage, Mr. Le Xuan Hieu. His technical help and guidance made me have a great learning experience. I am grateful to the School of Computer Science faculty of the Ho Chi Minh City International University, especially Dr. Nguyen Van Sinh, for his support since the beginning of my studies and my gratitude to the reading members' examination committee.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Agriculture productivity is one of the major sectors of the Vietnam economy, which contributes roughly 20 percent to the GDP and food to fulfill 95.54 million Vietnamese people [1]. For thousands of years, the Vietnamese are practicing agriculture. However, some small farmers use the outdated traditional agriculture method, which is highly sensitive to climate change and weather extremes such as the increase of severe drought episodes and salinity intrusion, which will pose challenges for farmers. Nowadays, the number of high-quality laborers in the agriculture, forestry, fishery sector is decreasing, and the negative effect of climate change on agriculture creates many kinds of crop disease, which constitute a significant threat to crop plan on the bigger picture causes serious implication for Vietnam economic development. Applying the new computer technology and advanced equipment to develop the automatic leaf disease detection technique helps and provides useful information for the farmer to identify diseases. This thesis presents image processing techniques such as image segmentation, image classification combined with the smartphone's advanced HD camera to propose a new way for smartphone-assisted plant disease determination.

CHAPTER 1

INTRODUCTION

1.1. Background

Agriculture in Vietnam can provide enough food to meet the requirement of more than 95.54 million Vietnamese people [1]. At present, because of Vietnam's geographical location, it is likely one of the most vulnerable countries to climate change and weather extremes, but its agriculture remained underdeveloped and lacks the resources to use modern methods to deal with the severe implications climate change. As a result of climate change, plant diseases have become more dangerous to agriculture for Vietnam and create harmful issues for farmers who mainly depend on agriculture productivity.

1.2.Scope and Objectives

The project assists farmers who usually recognize plant diseases manually, which requires a lot of experience and knowledge to identify the disease type. By emphasizes majorly, deep learning technology implements large convolutional neural networks model for automatic detection of plant diseases, which will take fewer efforts, less time, and become more accurate than the traditional way. Remarkably, the primary objective is to promote a new mobile application that allows users to capture plant leaf via HD camera and process the image locally to find, analyze, and categorize which kind of plant diseases or healthy. The thesis's primary objectives: Study machine learning and deep learning technologies and strengthen analyzing skills in choosing, training, and evaluating different suitable models for plant disease detection problem. Experimenting with different generative networks, generating more plant leaf image data, and implementing a segmentation pipeline to avoid misclassification due to unwanted input before feeding to the model. Finally, Research for understanding the fundamental of AI-based mobile application development.

1.3.Assumption and Solution

There are some committees needed to be defined before the project comes to presentation. The application for plant disease detection is established as an implementation for the thesis topic. It must carry essential functions like capturing, pre-processing an image before classifying and identifying. Nevertheless, the mobile application will not ensure to have an excellent UI because it is not the main objective. Furthermore, the accuracy of results cannot be completely satisfied in some situations (depending on the device that operates the applications and the input image).

1.4.Structure of thesis

This thesis consists of six sections: (1) Introduction, (2) Literature Review, (3) Methodology and Implementation, (4) Experimental result, and (5) Conclusions and Future Work. to start with, Introduction, include the structure of this thesis and explains the research problems and the solution for these like essential assumptions, objectives, and background are placed in this part. Next, the Literature Review sections include the academic knowledge of the implemented technology obtained from the Pre-Thesis to the thesis, which mostly is deep learning and image processing, including a universal idea about machine learning, deep learning, the in-depth comparison between model architecture. Such fundamentals provide handy tools for perception and promoting the final work. The methodology and Implementation section illustrate the prerequisite for setting up before developing the final product. The section also provides the frameworks, libraries, tools, and techniques used in those steps. Next, the Evaluation and Work Done section displays the experiment progress through concepts of the product and model visualization to make the decision. Finally, the Conclusions and Future Work sum-up and self-reflects the thesis and indicates the future opportunity of the work.

CHAPTER 2

LITURATURE REVIEW/RELATED WORK

2.1. Related Work

Researchers have proposed many plant disease detection approaches using both Machine Learning techniques, including classical algorithms like Support Vector Machines (SVM), K-means clustering, and K-Nearest Neighbours (KNN). Also, Deep Learning includes various ConvNet models such as VGGNet, InceptionNet, ResNet, etc.

Phadikar and Sil [2] works on input disease images of rice plants and applied K-means algorithms to detect infected parts of the plants, then this part has been utilized for classification purposes using a Deep Neural Network (DNN).

Gavhale [3] transform RGB color to another color format. Next, he enhances and segments the image region using K-mean clustering to discover the injury and hardness areas of leaves, then SVM is used for feature extraction and classification.

Deshpande [4] begins by obtaining input images that are prepared for augmentation. Then archive target areas (disease spots in image) using image segmentation techniques if a yellow margin border of the diseased spot on a leaf can conclude that it is infected by bacterial blight.

Reyes [5] uses 1.8 million images as a dataset to propose transfer learning with a pre-trained convolutional neural network and a fine-tuning strategy to benefit from increasing the training performance of a neural network model and producing lower wrong prediction.

Elangovan and Nalini [6] use the combination of Support Vector Machines (SVM) and image segmentation approach, including Otsu's method, k-means clustering, to classify various plant leaf.

Muhammad Hammad Saleem, Johan Potgieter, and Khalid Mahmood Arif [7] introduced the deep learning-based relative evaluation to classify plant disease. Firstly, the best

ConvNet was achieved by handling a similar experiment among famous ConvNet and revised and cascaded/hybrid variants of some of the DL models introduced in modern study.

Muhammad Mohsin Kabir, Abu Quwsar Ohi, and M. F. Mridha [8] implement various common ConvNet architectures. The results prove that the DenseNet and Xception achieve more reliable in multi-label plant disease detection tasks.

Hossian [9] worked on plant disease detection and recognition on the Tea plant leaves. The detection process uses the support vector machine classifier to classify 11 features for the images, then are analyzed and uploaded into the SVM database. This process shows it takes less computation time with high accuracy and improves detection and recognition efficiency.

Singh [10] introduced leaf disease detection by using soft computing methods and image segmentation. This approach automatically analyzes and classifies the diseases in the leaf, and then he conducts surveys of different diseases related to plants is also used in the detection, which is done thanks to a Genetic algorithm, which optimizes the variables efficiently.

Inspired by these works, this thesis uses several pre-trained models to experiment to produce the best results in plant disease detection tasks.

2.2. Technical Background

2.2.1 Machine Learning

Machine learning is a part of artificial intelligence to provide the computer skills to learn on its own and feed it several examples. Machine learning uses algorithms and neural network models to support computer systems in progressively enhancing performance. It also builds a model using sample data to make decisions without the need for additional human programming. Machine Learning has been broadly using in natural language processing, search engines, computer vision, and much more.

2.2.2. Deep Learning

Deep learning is simulates based on the human brain in processing data to generate patterns [11]. Deep learning can be supervised, unsupervised, or semi-supervised, with networks that can learn unsupervised from unstructured or unlabeled data. While Machine learning algorithms or systems are used when the dataset is relatively small, Deep learning can deal with a massive scale of data, namely, big data. There are server key benefits of deep learning, such as the capacity to deliver high-quality outcomes, exclusion of additional losses, and much more.

Deep Learning is a developing area with tremendous applications that are applied in several use cases. It is essential when new to this field to acknowledge and recognize the different types of Supervised models used in Deep Learning, which are described below:

- Classic Neural Network, which has unique, allows it to accommodate primary binary patterns through a sequence of information, resembling a human brain's patterns. A Multilayer Perceptron [12] is considered the standard neural network model consisting of more than two layers.
- Recurrent Neural Networks (RNN) was produced for predicting and forecasting sequences. Long short-term memory [13] (LSTM) and new approach transformers are conventional RNN algorithms with several use cases: image classification, captioning, sentiment analysis, video classification, and much more.
- Convolutional Neural Network (ConvNet) is created for image data, managing, and computing a more significant amount of complexity before pre-processing. ConvNet is considered the most practical and flexible model for both image classification and detection problems. ConvNet frequently uses image input to analyze OCR documents, transform the 2D field to 1D for more accelerated processing, and supervise significant complexity in measuring the output.

2.2.3. Fundamental Architecture of Convolutional Neural Network (CNNs)

Computer Vision and Deep Learning have been assembled and become more advance in recent years. Image classification or object detection is responsible for inputting an image date and outputting a predicted class name or a potential value of one or many classes that suitably represents the image when choosing an image as an input, assigning the weight of multiple features in the input image, and can distinguish each another. The ConvNet required a considerably lower pre-processing time than other types of classification algorithms. ConvNet typically has three main layers: The Convolutional layer, then the pooling layer, and the fully-connected layer.

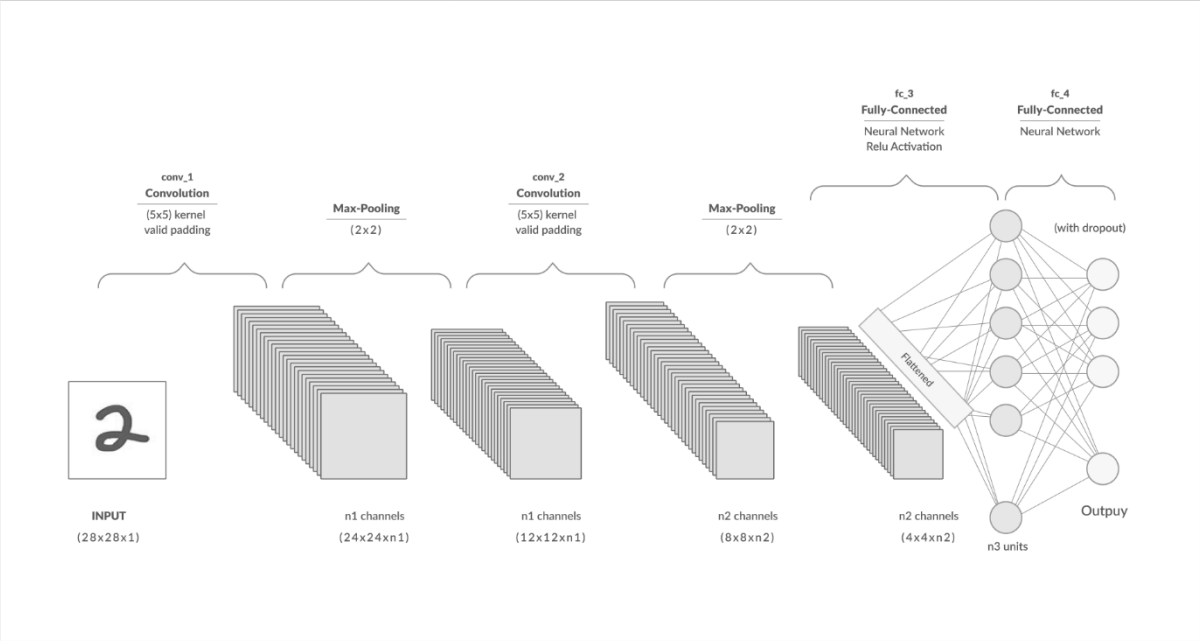


Figure 1. is an illustration showing the Convolution Neural Network to an image of hand-writing digits [14].

Convolutional layer: is applied to obtain features from the input such as color, edges, and corners. It can recognize more complex features when moving deeper inside the network, such as ash shapes, digits, and face parts. The convolutional layer tries to produce a dot product of two filters (or two feature kernels), which is the set of various parameters that can be learned. In terms of spatial dimensionality, those kernels are generally poor but cast

along the whole of the input's depth [15]. When the input goes to a convolutional layer, the layer slide to every filter over the input to create a 2-dimension activation map. The Convolutional layers can optimize their output, significantly reducing the model's complexity thanks to three hyperparameters: the stride, the depth, and zero-padding.

- The **stride** is the depth around the input's spatial dimensionality, which sets on the receptive field. When the stride is equal to one, which means at a time, it moves the filter one pixel. When it equal to two, then the stride jumps two pixels at a time, producing fewer output amounts spatially.
- The **depth** is a hyperparameter that corresponds to the number of kernels (filters) required, each learning to scan for some unusual thing in the input.
- **Zero-padding** is a hyperparameter, a powerful technique to give additional controls to the output volumes' dimensionality.

Parameter Sharing works on the assumption that if a region feature is beneficial to calculate at a set spatial point, it can be useful to calculate another point. Parameter distribution essentially uses in Convolutional Layer to manage the parameters' quantities. Here is the representation of a convolutional layer. The kernel center is set over the input vector computed and displaced with the sum of weighted and nearby pixels.

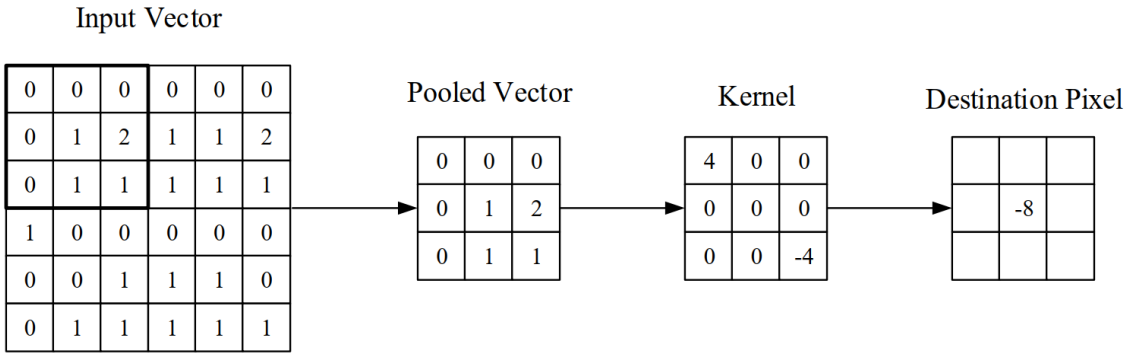


Figure 2. A visual represent of a simple convolutional layer.

The pooling layer: is the layer that reduces the computational cost needed to process the input by both the number of parameters and computational complexity of the model. In this layer, it tries to obtain the principal features from an insufficient amount area, which means is to take the highest values in the pooling region and to move the pooling area each time when handling a different part of the matrix. Specifically, there are two kinds of pooling: MAX pooling [16][17] and AVERAGE pooling [18]. The principal point to distinguish between two kinds is, AVERAGE-pooling takes the mean of all the pooling area values, but in MAX-pooling takes the highest values inside the pooling area. Consequently, a matrix that carries the image's principal features is obtained, and this matrix has even lesser dimensions after going through with the pooling layer.

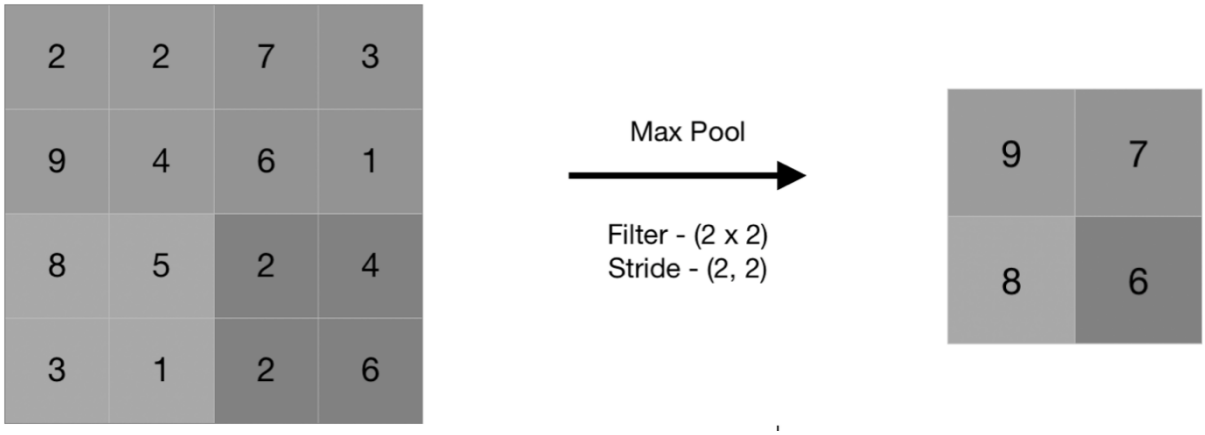


Figure 3. A visual represent of MAX-pooling technique.

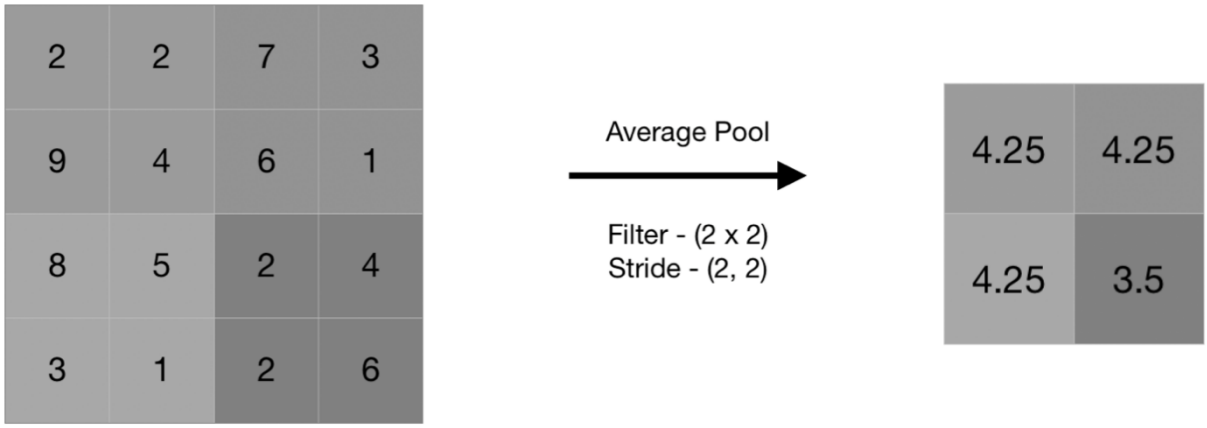


Figure 4. A visual represent of AVERAGE-pooling technique.

Fully connected layer: this layer highlights some features in an image and significantly reduces the image's dimensions. It takes the convolution processor pooling process's results and uses them to classify the image into a label. The fully connected part of the ConvNet determines the most accurate weight, and then each neuron receives weight, which prioritizes the most appropriate label to make the classification decision. The visual below represents how the input value goes through inside the neurons' initial layer, then multiplied by a weight, then it passes into the activation function to the output layer, in which every neuron represents a classification label.

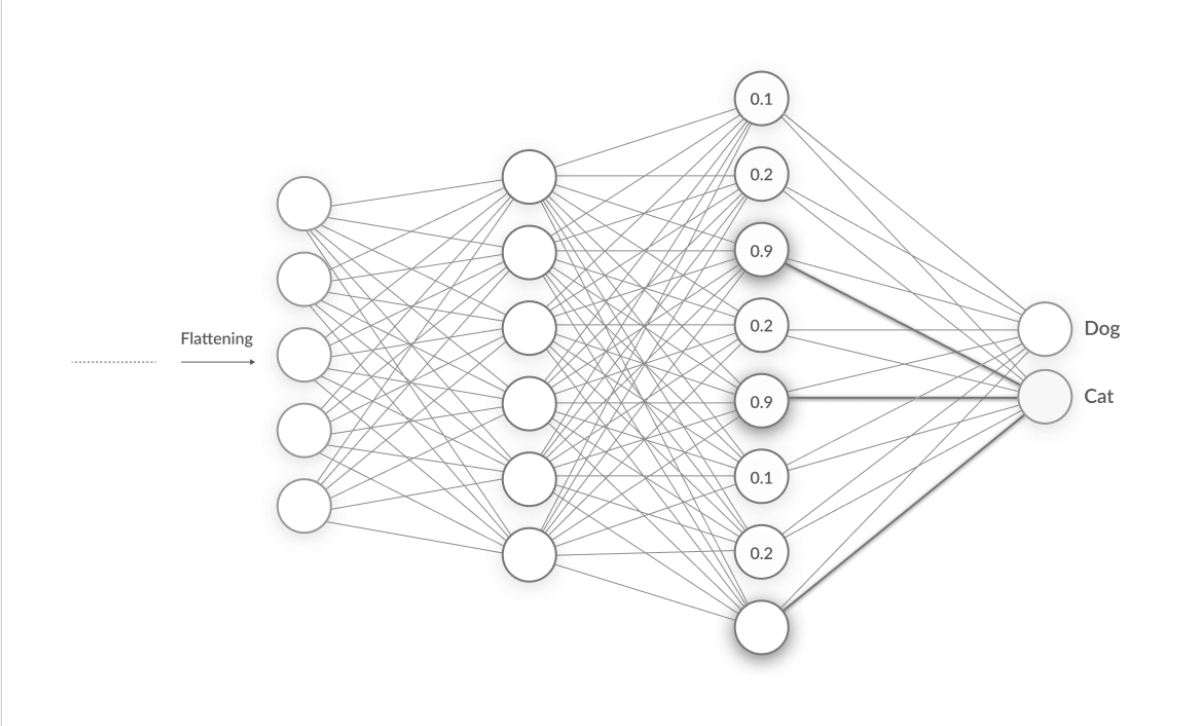


Figure 5. A visual represent the role of a fully connected layer in a CNN architecture [14]

2.2.4. Some well-famous CNNs Architectures

LeNet-5 [19] is the famous ConvNet architecture since it was built by Yann LeCun and has been broadly used to recognize the handwritten digit (MNIST Dataset). It is formed of several layers illustrated in figure. Its architecture comprises seven layers of a composition consisting of three convolutional layers, two subsampling layers, and two fully connected layers.

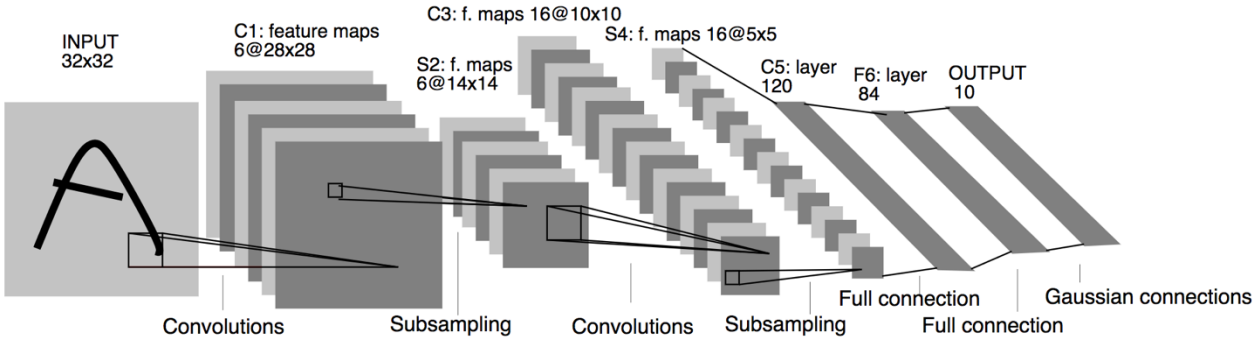


Figure 6. A visual represent the Architecture of LeNet-5[20].

AlexNet [21]: ConvNet architecture was mainly built by Alex Krizhevsky. When the second-best reached only 26%, AlexNet achieved a top-five error rate of 17%. As a fantastic result, it won the ImageNet ILSVRC challenge in 2012. Its architecture is illustrated in figure below comparable to LeNet, only broader and more profound. Instead of stacking a pooling layer on each other convolutional layer's head, AlexNet stacks convolutional layers straight on one another's head.

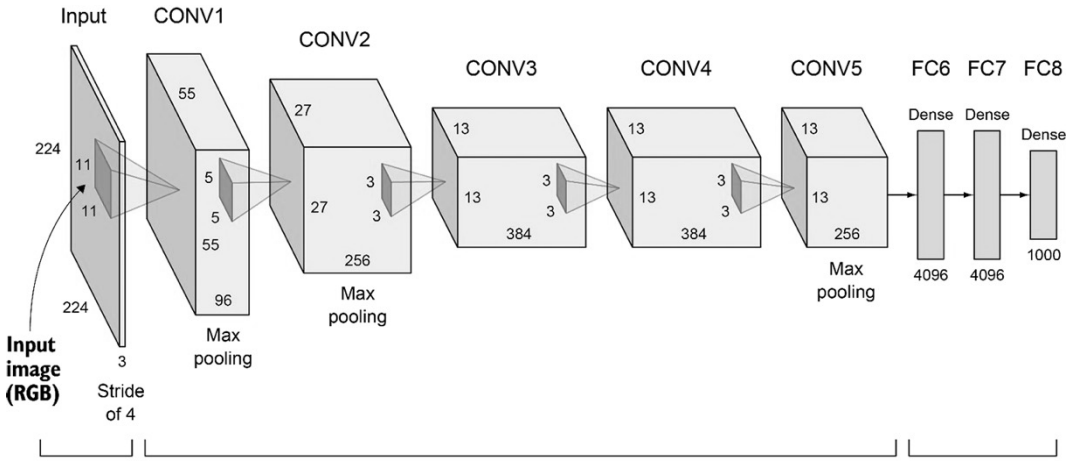


Figure 7. AlexNet's Architecture from Left to Right [22].

GoogLeNet [13]: is a ConvNet architecture used for image classification developed by Google, and it also is the winner of the ILSVRS. GoogLeNet is also called Inception-v1, which includes 1x1 convolution at the center of the network as a dimension compression model to diminish the computation cost and bottleneck, but width and depth can be improved. When performing 5x5 convolution without using 1x1 convolution, it will take more operations than the use of 1x1 convolution, which reduces the model size and somehow helps decrease the overfitting problem. Unlike AlexNet, which uses fully connected layers at the top of the layer, GoogLeNet applied global-average pooling nearly at the top of the network to increase accuracy. Various inception models joined together to reach deeper in the figure below make GoogLenet an intense model with 22 layers.

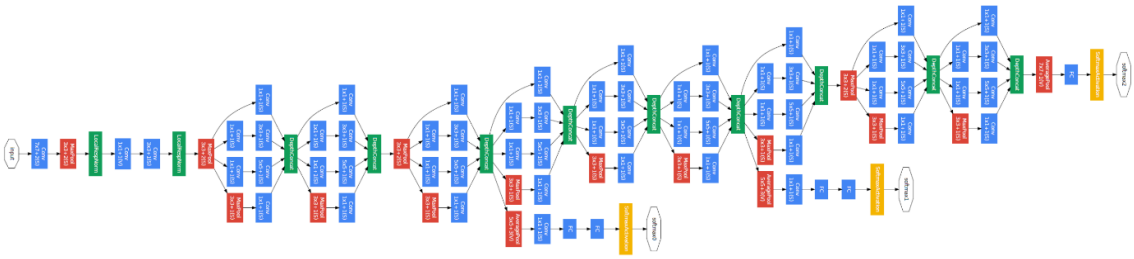


Figure 8. GoogLeNet's Architecture [24].

Inception-v3 [25]: basically, is GoogLeNet combine with factorization ideas to decrease the number of connections or parameters without reducing the network performance. In Inception-v3, the auxiliary classifier is used to have a deeper network, which was already suggested in GoogLeNet is used as a regularizer and has only one last 17×17 layer applied with auxiliary classifier. Since having efficient grid size reduction, the less expensive and still efficient network is easy to achieve in Inception-v3. Also, with 42 layers, it produces much more efficient than AlexNet in terms of computation cost and only about 2.5 higher than GoogLeNet. With the architecture shown in the figure below, In ILSVRC 2015, Inception-V3 obtained 1st for image classification with the lowest top-5 error rate of 3.58%, proving that Inception-V3 produces a better result than other ConvNet Architecture.

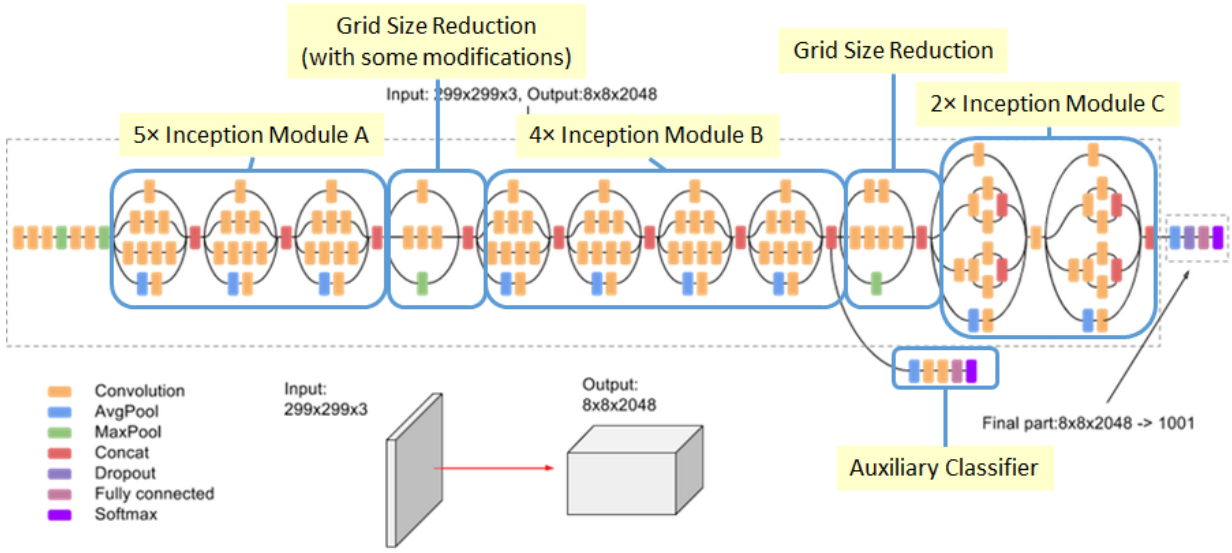


Figure 9. Inception-V3 Architecture [26].

VGGNet [27]: architecture, built by the Visual Geometry Group research lab at Oxford University, got the first rank in the ILSVRC challenge in 2014. Despite having a classical and straightforward architecture, VGGNet has the repetition of only two to three convolutional layers and a pooling layer until reaching a sum of sixteen or nineteen convolutional layers plus the output layer and a last dense network with two hidden layers. VGGNet used only 3×3 filters. However, it has numerous filters. The only difference between other VGG variant is the number of convolutional layers.

Figure 10. VGGNet-16 Architecture with 16 layers [28].

ResNet's [29] illustrated in Figure 11. It is straightforward. It begins and ends precisely similar GoogLeNet the only difference is a dropout layer. Nevertheless, it Won the

ILSVRC challenge in applying a Residual Network that delivered a phenomenal top-five error rate under 3.6%. After that, it established the widespread trend: models are getting deeper to increase accuracy connections) to be capable of training a deep network.

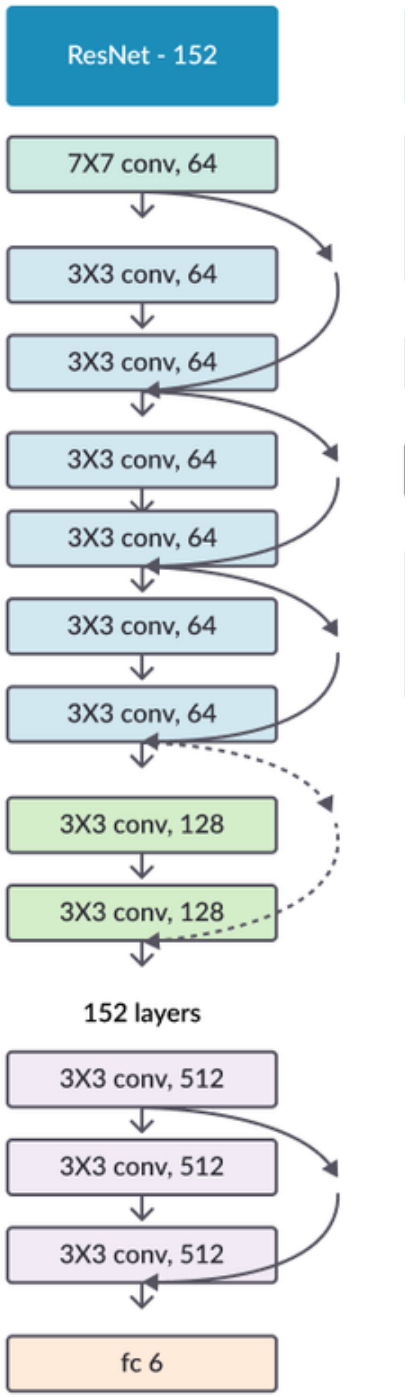


Figure 11. ResNet-152 architecture with 152 layer and skip connection [30].

2.2.4. EfficientNet

Many researchers have been experimenting and developing more efficient architectures to enhance accuracy on different tasks in recent years. However, most of them are not decisive in processing load, and larger numbers represent the total number of blocks (layers) to go deeper, enhancing ConvNet's accuracy and increasing the number of parameters, leading to extended training time and cost more resource budget. In 2019, Quoc V. Le and Mingxing Tan introduced new approaches concentrate on improving accuracy and provide superior performance compared to another counterpart by proposing a new scaling approach that consistently scales focus on a combination of depth, width, and resolution using an extremely convincing compound coefficient. Their implementation was similar to the MNasNet approach [31], which uses neural architecture search to design a new baseline network to obtain a mobile-size network, called EfficientNets [32].

2.2.4.1. Compound Scaling

The scaling method is commonly performed to enhance the accuracy of a particular task. Before 2019, the most popular approach to scale up ConvNets was either by one of three dimensions - depth (number of layers), width (number of channels), or image resolution (image size). EfficientNets, on the other hand, perform Compound Scaling, which scales all three dimensions while maintaining a balance between all dimensions of the network. In other words, it controls the constraints on the scaling coefficients when require to increase or decrease the depth, height, and resolution of a specific network. (see the figure below will define what scaling indicates across different dimensions).

Depth Scaling:

Depth is able to scaled-down or scaled-up by removing or adding layers. For instance, ResNets can be scaled-down from ResNet-1000 to ResNet-101 and scaled-up from ResNet-101 to ResNet-1000. A deeper network is able to obtain more valuable and intricate features to generalize new tasks properly so that the network performance should significantly

enhance with more layers. However, realistically it does not follow and difficult to train due to the vanishing gradient difficulty. Even if avoiding the gradients to vanish and using several methods to smooth the training, adding extra layers does not regularly provide a good result. For instance, ResNet-101 has similar accuracy as ResNet-1000.

Width Scaling:

Scaling the number of channels is commonly used when the researcher wants to obtain a mobile-size network like MobileNets [33] and MNasNet. Wider networks are more likely to be capable of obtaining more relevant features. Furthermore, more miniature models are more comfortable to train, but the difficulty is that even though it can create the network greatly wide, as a result, the accuracy saturates quickly with a larger width with shallow models.

Resolution Scaling:

Intuitively, the features of a high-quality image maybe provide more information; therefore, high-quality pictures should work better. However, similar in some Object detection scenarios, using image resolutions that do not scale linearly like 300x300, or 512x512, or 600x600..., which more likely to impact the model's accuracy to gain diminishes rapidly. For instance, enhancing resolution from 490x490 to 550x550 does not yield tremendous improvements.

Compound Scaling:

The earlier three cases guide the initial observation: Larger networks tend to produce higher accuracy when they have greater width, depth, or resolution. However, once the accuracy increase reaches about 80%, it is rapidly saturated, which indicates the limitation of one-dimensional scaling.

The authors discovered that the network's depth and width should be raised when the images' resolution rises. As the depth rises, enhancing receptive fields captures comparable features that carry extra pixels in a picture. Additionally, the width is extended, which enables to capture of more relevant features. There were several experiments with various scaling values

per dimension to validate this intuition. For instance, as shown in the figure… below, all dimensions of depth, width, and resolution must have to be scaled together with a specific coefficient, and there is an optimal balance for each dimension relative to the others that help the model performs much better efficiency under the equivalent FLOPS.

Consequently, it is crucial to match the whole dimensions of a network, including width, depth, and resolution. In other words, to scale up the ConvNet, the depth, width, and resolution of layers should rise by 20%, 10%, and 15%, respectively, to keep things as useful as desirable while extending the implementation and achieving the ConvNet accuracy.

2.2.4.2. **EfficientNet Architecture**

The authors apply the Neural Architecture Search approach, which similar to the MNasNet, to obtain EfficientNet base model (B0) by using a reinforcement learning-based approach to leverage a multi-objective search optimizing both accuracy and FLOPS. To understand how the EfficientNet architecture developed, first look into the MnasNet Architecture, developing based on neural architecture search which framework consists of three main components (as a figure below):

- a controller base on a recurrent neural network (RNN)
- a trainer to achieve the model accuracy
- a mobile phone-based inference engine for estimating the latency

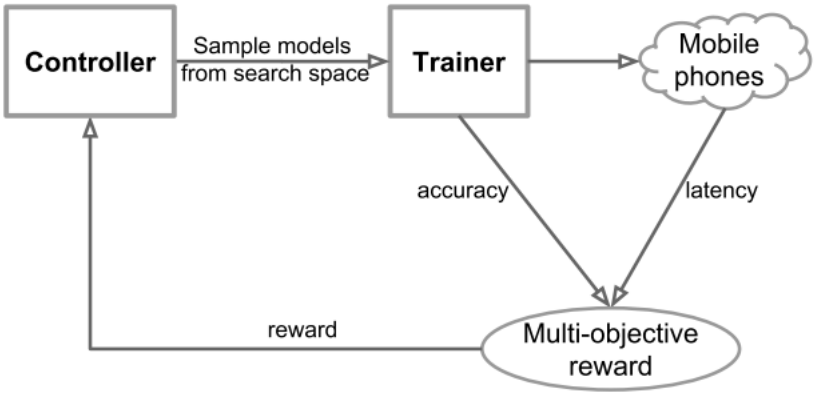


Figure 12. An overview of MNasNet Approach.

For the MNasNet approach, the RNN controller has applied to find a model architecture and then train on the ImageNet dataset to archive accuracy and latency values. The reward function is then calculated, and feedback is sent back to the controller to repeat this process a few times until the best architecture is obtained such that its accuracy is maximum given latency is lower than a particular specified value.

The researcher obtained the MNasNet architecture that reached 75.2% top-1 accuracy and 78ms latency. Similarly, the EfficientNet discovers an optimal neural network architecture that maximizes $ACC(m) \times \left[\frac{LAT}{T}\right]^{W}$. However, FLoating-point Operations Per Second (FLOPS) was applied as a standard measure of computer performance rather than latency in the objective function instead of particular hardware as opposed to MNasNet architecture. Consequently, EfficientNet-B0 is lightly larger because of a more significant FLOPS purpose, and the network layers/blocks are as shown below figure:

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	28 × 28	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Figure 13. An overview of EfficienNet-B0 Architecture.

The MBConv block is an Inverted Residual bottleneck with an Excite and Squeeze block included. As in the Bottleneck layers introduced in the InceptionV2 architecture [34], the

integral approach uses a 1x1 convolution to downsample the number of channels then use the 3x3, 5x5 convolution operation to the decreased number of channels to archive output features. Finally, use another 1x1 convolution operation again to increase the number of channels to the initial value.

As in MBConv, the inverted bottleneck does the reverse - instead of reducing the number of channels, and the first 1x1 convolution layer increases the number of channels to 3 times the initial.

Adopting a standard convolution operation here would be costly, so a Depthwise Convolution is applied to obtain the output feature map. Finally, the second 1x1 convolution layer brings down the number of channels to the initial value, illustrated in the figure below.

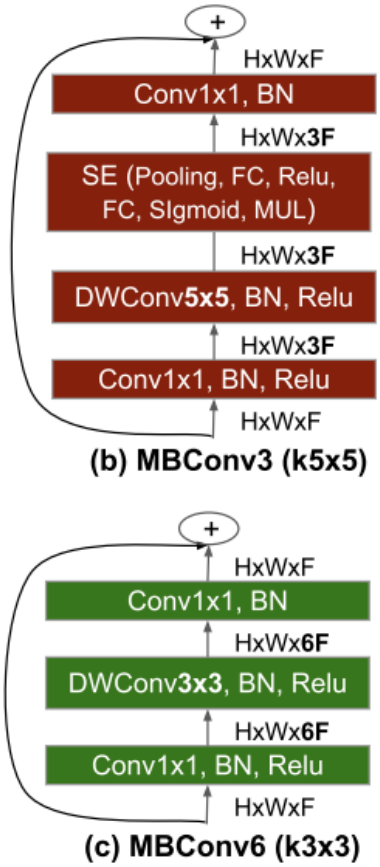


Figure 14. A demonstration of MBconv layer.

In other words, after archive a new efficient base model, compound scaling is applied to scale up the EfficientNet from B0-B7 (increasing all three dimensions include depth,

width, resolution) that lead EfficientNet accuracy to compare to most other CNN's on ImageNet dataset is significantly better. However, the models generated by arbitrarily choosing must follow scaling portion in as equation, which was mention above, and the selection ratio of the resolution, depth, and width are also restrained by several circumstances as listed:

- Resolution: need not be divided by eight, which generate zero-padding near edges of several layers, draining computational resources, mostly uses to smaller modifications of the model. Therefore, the input image resolution for B0 and B1 should be 224 and 240.
- Depth and width: EfficientNet's construction blocks require channel size have to be multiples of eight.
- Resource limit: growing depth and (maybe or) width and remain resolution ratio is able to enhance efficiency and more likely to prevent memory limitation from resolution bottleneck.

Consequently, the depth, width, and resolution values of the EfficientNet models' family are chosen manually and verified to perform excellent outcomes. Therefore, the author's implementation only provides these eight models from B0 to B7 and their expected input resolution, instead of allowing the random selection of width/depth/resolution parameters.

Thanks to the combination of compound scaling and efficient base model, which deliver impressive results compared the number of parameters and computations needed versus nearly every different ConvNet architecture (5x+ decrease while maintaining or hitting most accuracy). Perhaps more importantly, most classification task presently utilizing transfer learning to enhance the accuracy as well as reduce computation cost and training time, so the author experiment EfficientNet using pre-trained weights from the ImageNet dataset on various tasets, which is the typical method to observe how most CNN's get implemented on real products, and the results are quite remarkable.

2.2.5. Transfer Learning

2.2.5.1. Motivation

When human has a natural capacity to carry experience across tasks that help them use in the similar method to resolve relevant tasks, they do not require to learn everything from the beginning when trying to learn additional features or topic that also means humans can carry their knowledge from earlier learned fields to the newer field. In Deep Learning, a pre-trained model has been traditionally designed to solve one specific task, re-purposed in Transfer Learning to apply it to a diverse but related problem. For example, the knowledge gained while acquiring to identify motorbike can be applied to identify the car. Transfer Learning's benefit is using existing knowledge from the origin learner in the purpose task without learning from the beginning. It is a shortcut to saving time and producing more reliable performance.

2.2.5.2. Strategies

Deep Learning has gained significant advancement in the previous year, which has allowed dealing with complex difficulties like image classification. Several DNN with high efficiency improved and experimented across various fields, such as natural language processing, computer vision, and image processing. These pre-trained models form the foundation of transfer learning in the meaning of deep learning. There are two most popular strategies for deep transfer learning:

Feature Extractor: Deep learning models are layered architectures that acquire different features at various layers. These layers are attached to the top of the last layer to generate the output. This layered architecture enables a pre-trained network (such as EfficientNet) without its last layer set as a feature extractor for other responsibilities. The approach here is not to change the model layers' weights when training new data for the new specific task.

Fine Tuning: This is an approach of Transfer Learning, where it does not change the last layer and retrain some of the earlier layers make the first layers have been examined to

obtain universal features. The next layers concentrate further on the unique task. For instance, we have a dataset, and we use 80% of it in training, then train the same model with the remaining 20%. Usually, changing the learning rate to a smaller one for not significantly impacting the already adjusted weights. It is likely to fine-tune everything in the Convolutional Neural Network layers or desirable to maintain some of the previous layers set (due to overfitting) and just fine-tune several parts of the network. The observation prompts this that the previous features of a ConvNet include more universal features that beneficial to several tasks, but the following layers of the ConvNet grow more particular to the initial dataset's classes' specifications.

When and how to fine-tune: When and how to fine-tune: It depends on how the researcher decides what kind of transfer learning to use on new data. The two most crucial factors are the similarity to the original dataset and size. The ConvNet features are universal in the initial layers and more distinct in the following layers. Here are some regular habits for operating the four real situations:

- The new dataset is low and comparable to the initial dataset. It is the most practical approach to train a linear classifier on the ConvNet codes.
- The new dataset is broad and associates with the initial dataset because additional data drives to extra assurance that it will not overfit if it were to attempt to fine-tune through the entire network.
- The new dataset is insufficient and also diverse from the initial dataset. It is well-optimized to implement the support vector machine classifier from activations someplace previously in the network.
- The new dataset is enormous and also distinct from the initial dataset. Because the dataset is vast, it is likely best to train a Convolutional Neural Network from scratch, but it is worth considering initializing weights from a pre-trained model because of its benefits. There are sufficient data and assurance to fine-tune over the full network.

CHAPTER 3

METHODOLOGY

This thesis's final product requires training, evaluating, and several fine-tuning models that promote identify plant disease from input images with as high accuracy as possible. EfficientNet will be the base pre-trained model of the project because that was trained on a comprehensive dataset, low error rate, and parameter and high performance. Android is chosen to be the operation system of a prototype for this project because of its wide range and popularity of mobile devices. Additionally, android is well-backed by its massive community of open-source. Four preliminary plans need to be obtained for this work to be complete: (1) Initialize ML pipeline and Requirements, (2) Training and evaluating Models, and (3) Implementing mobile application.

3.1. ML Pipeline and Requirement

3.1.1. ML Pipeline

The principal purpose of having a conventional pipeline for every Machine Learning model is to execute administration over it. A well-organized pipeline offers the implementation more and more adaptability. Machine learning (ML) pipelines consist of several processes to train a model (see figure): Data collection, data validation, data preprocessing, model training, model evaluation, and model deploying.

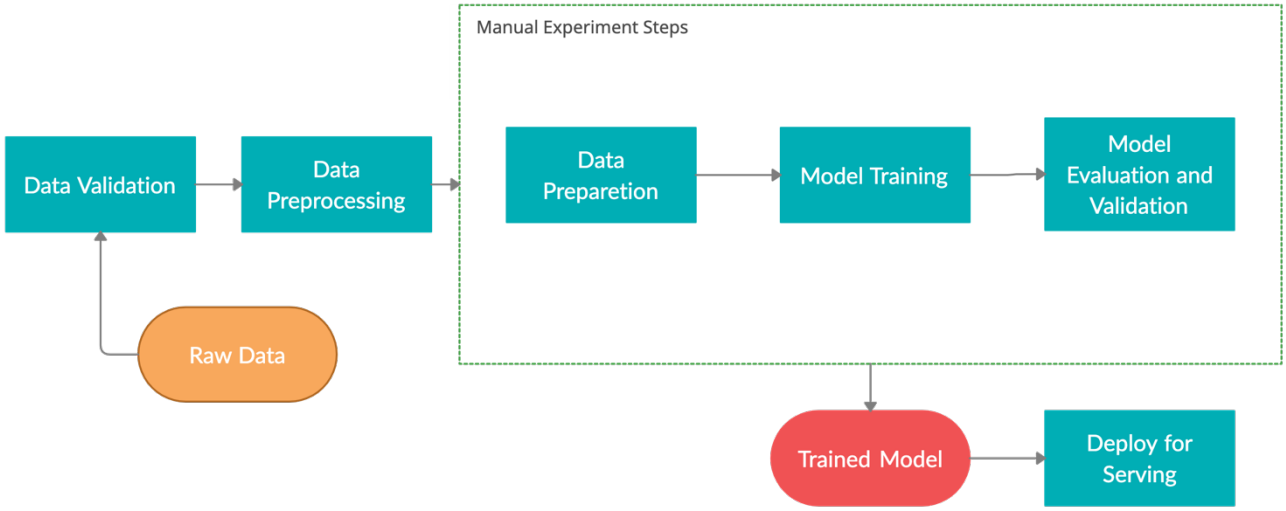


Figure 12. ML Pipeline of this thesis

3.1.2. System requirement

Since developing this application, especially for image-based plant classification, I set myself a goal of guaranteeing that my application would meet the requirements as listed below:

- Users can capture an image from rear and back camera.
- Users can choose image from gallery for classification.
- Result of the classification image is loaded automatically after users capture or load image successfully.
- The model must have high classification accuracy
- The UI must be clear enough to avoid misunderstanding usage.
- The app must be battery-saving and lite-weight.

3.1.3. Dataset analyzing

We analyze 115,404 images of plant leaves, which covered 38 class labels attached to them. Across full experiments, two distinct variants of the entire PlantVillage dataset will be examined, begin with the color version, then experiment with the leaves were segmented by eliminating whole the additional background, which more likely to increase accuracy. Due to the enormous number of images from the dataset, we split it into three parts, generally into 80% of training, 10% of the validation set, and 10% of the test set.

3.1.4. Choosing model architecture

As Deep CNNs require intensive training, networks VGGNet-16, InceptionNet-v3, ResNet-50, MobileNet-v2 and EfficientNet-B1 were evaluated as base networks. The selection of the base model for the image classification algorithms amongst these state-of-the-art models depends heavily on the speed versus performance payoff between the models. The speed and

accuracy vary heavily between the different models as well as the test set accuracy between the networks. According to our data, EfficientNet-B1 is faster than models with inception module from InceptionNet-v3, but when it comes in test set accuracy, which is also the case between ResNet-50 and InceptionNet-v3, where they are deeper meaning that there is more parameter to fit data. As a result, they have slightly higher accuracy but lower latency because of their weight. In a mobile environment, the processing time that the mobile device is able to perform computations is crucial for the mobile experience. With a too complicated and computationally expensive network, the device would struggle to run the application on low latency. According to my experience, a minimum latency below 1000ms is required for a smooth mobile experience, which limited the base network to either ResNet-50, InceptionNet-v3 or VGGNet-16 has low speed on mobile devices. Nevertheless, all mentioned architectures were evaluated and assessed.

3.2. System Design

System Architecture Diagram is conventional method to illustrate how components inside the systems communicate with each other and the system's formation.

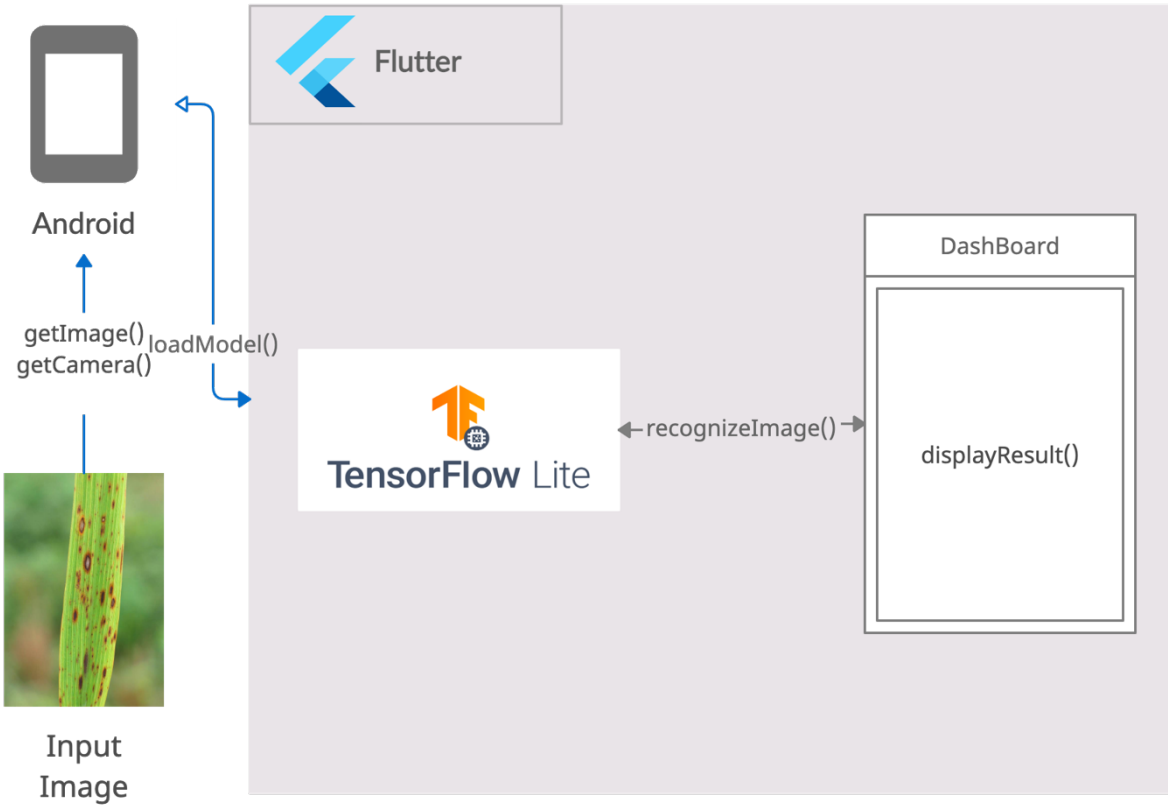


Figure 13. The system architecture diagram.

The figure above illustrates that the application is a manageable system since after archiving the best model for plant disease detection thanks to TensorFlow-lite, making it easy to integrate into the Flutter mobile application. From that figure, we can conclude that this work's principle is to the model, which plays an essential role as the heart of the application. Once the application is opened, an association between the TensorFlow-lite and Flutter allows classifying the image when the user picks from the image library or takes new pictures with the camera.

3.2.1. User Interface design

Designing of the wireframe is one of the crucial stages of the User Experience and User Interface (UX-UI) design. Wireframe essentially provides an overview of the layout, the

layout of components that helps generate the backbone and standards for the detailed design of the entire user interface.

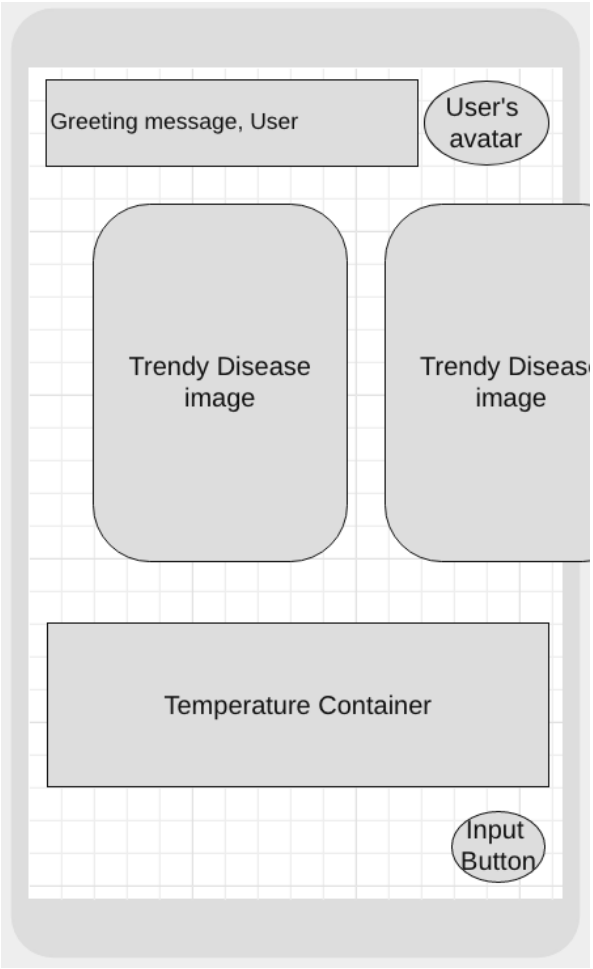


Figure 14. Wireframe of Initial App Screen.

Figure 22 gives a general look at the wireframe of the initial screen. The greeting message and user's avatar will be at the top of the screen, and the stack contains trendy diseases at the middle; the temperature container and input button will be at the bottom of the screen. This screen will show up when the user first opens the application. The user can input the image for classifying by capture from the camera or choose an image from the gallery.

CHAPTER 4

IMPLEMENTATION

4.1. Setting up Google Colaboratory

In building the Deep Learning model, it is essential to have a Jupyter notebook with well-configuration and offers to the developers using GPU for training models. There are many emulators with different pros and cons. In many cases, Google Colaboratory is considered a primary notebook environment to implement, train, and fine-tune model because it runs entirely in the cloud and supports many popular and high-level machine learning libraries that can be quickly loaded. The networks were trained on a computational server (Google Colab) with the following hardware specifications:

• GPU: Nvidia Tesla P100 (16GB)

• CPU: Intel(R) Xeon(R) CPU @ 2.30GHz

• RAM: 12GB 1600MHz

4.1.1. Cloning Dataset repositories in Google Colaboratory

Since our dataset is not too large, we use GitHub to store our datasets on the cloud because it is designed to collaborate on coding projects and a potentially great resource for researchers to make our data publicly available. We clone a GitHub repo inside Google Colaboratory by going to the GitHub repository and copy the clone link of the repository:

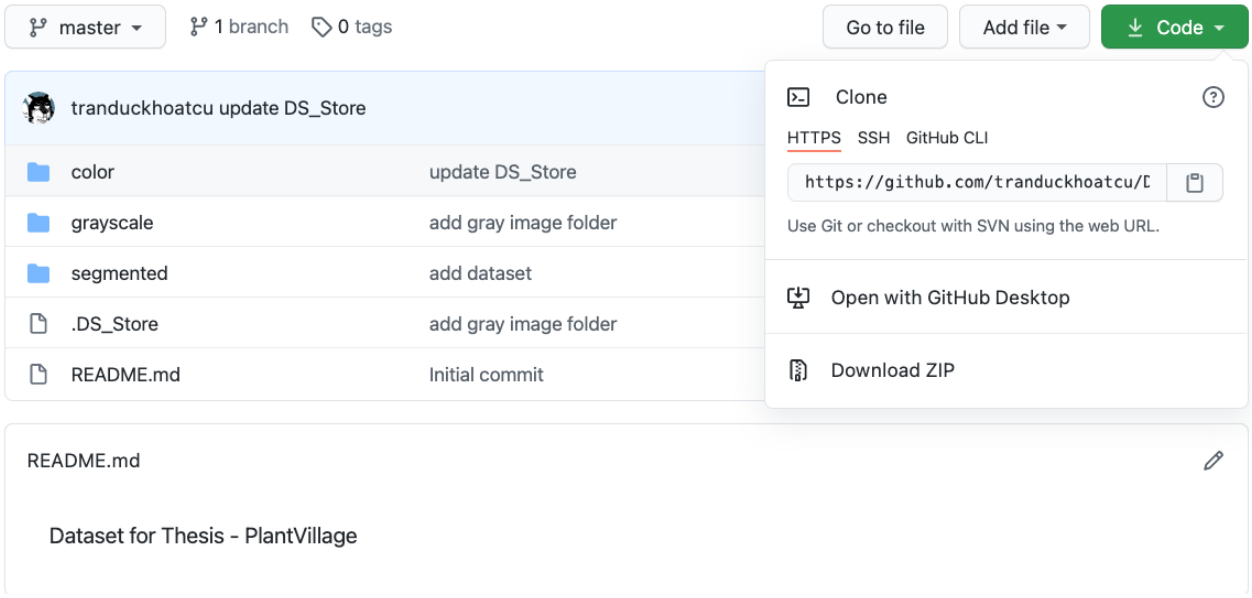


Figure 15. The GitHub repository stored dataset for this thesis.

4.2. Setting up Comet.ml

Comet is the best alternative to the standard Tensorboard enables data scientists and teams to track, compare, explain, and optimize experiments and models, which also supports various functions such as logging of experiment metrics, plots, images, sound, gradients, model weights, and an online dashboard to store in the cloud (figure 27).

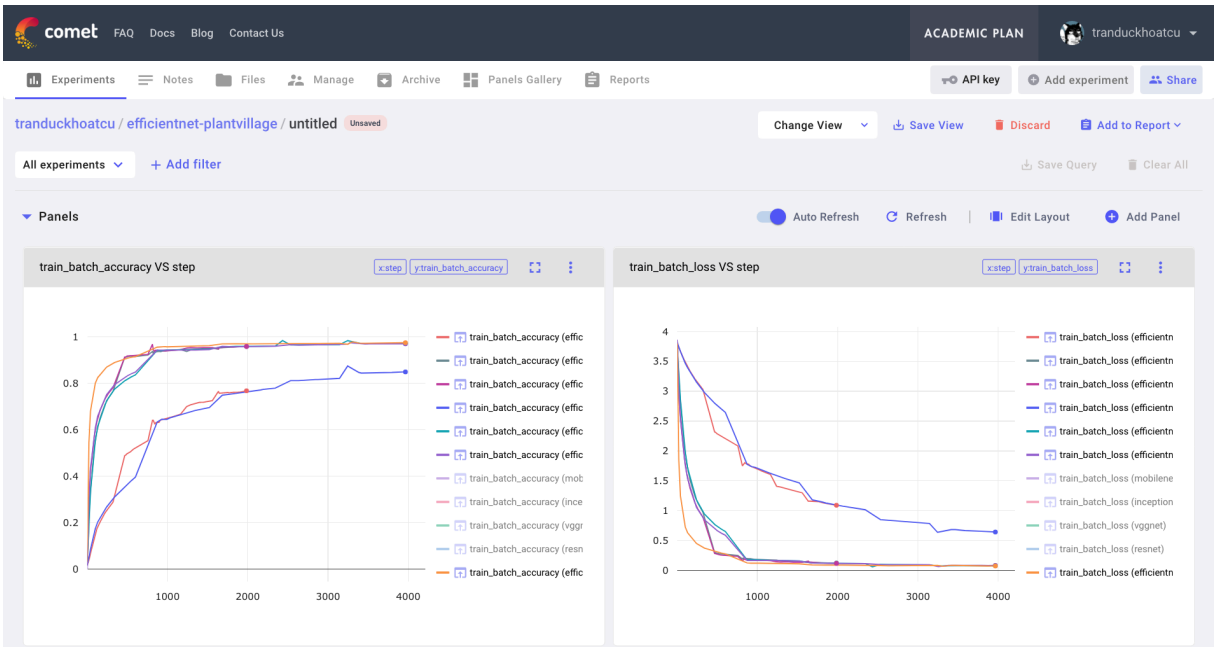


Figure 16. The main dashboard of our thesis in Comet.ml.

That service provides a dashboard that brings together the code of our experiments and results. Besides, the Commet.ml also allows us to optimize our models by tweaking and monitoring each experiment's hyperparameters. To be specific, it not only tracks the results and provides a graph of results, but it also trails the code changes and imports them to compare later all the different aspects of the various versions of experiments.

4.3. Setting up Android Studio

In building an Android application process, it is essential to have an emulator that will install and run the application in debug mode before releasing it as an apk file to run on Android devices. There are many emulators with different pros and cons. In that case, I choose Android Studio as my primary emulator to run my application in the developing process due to its stability and the variety of features supported.

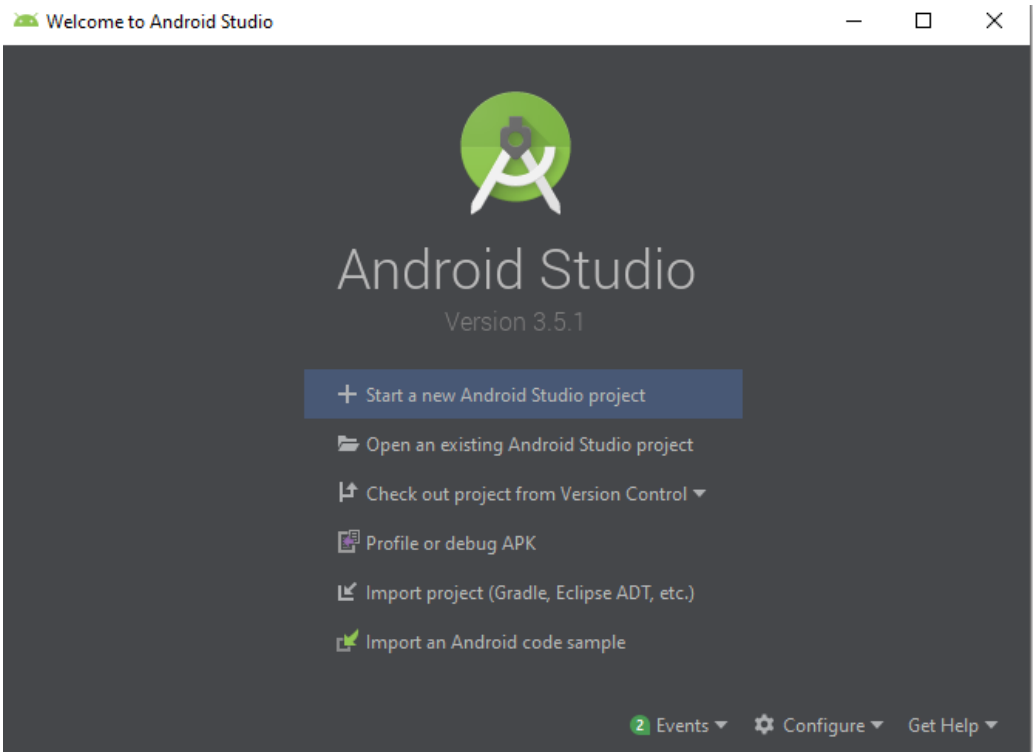


Figure 17. The initial of Android Studio.

Figure 28 shows the first screen when we start Android Studio. Software Development Kit (SDK) is required to run any emulators. In Android Studio, the SDK manager

can easily be found in the Configure menu (in the bottom right corner of the screen). All available SDKs will show up when we click Configure and choose the SDK Manager.

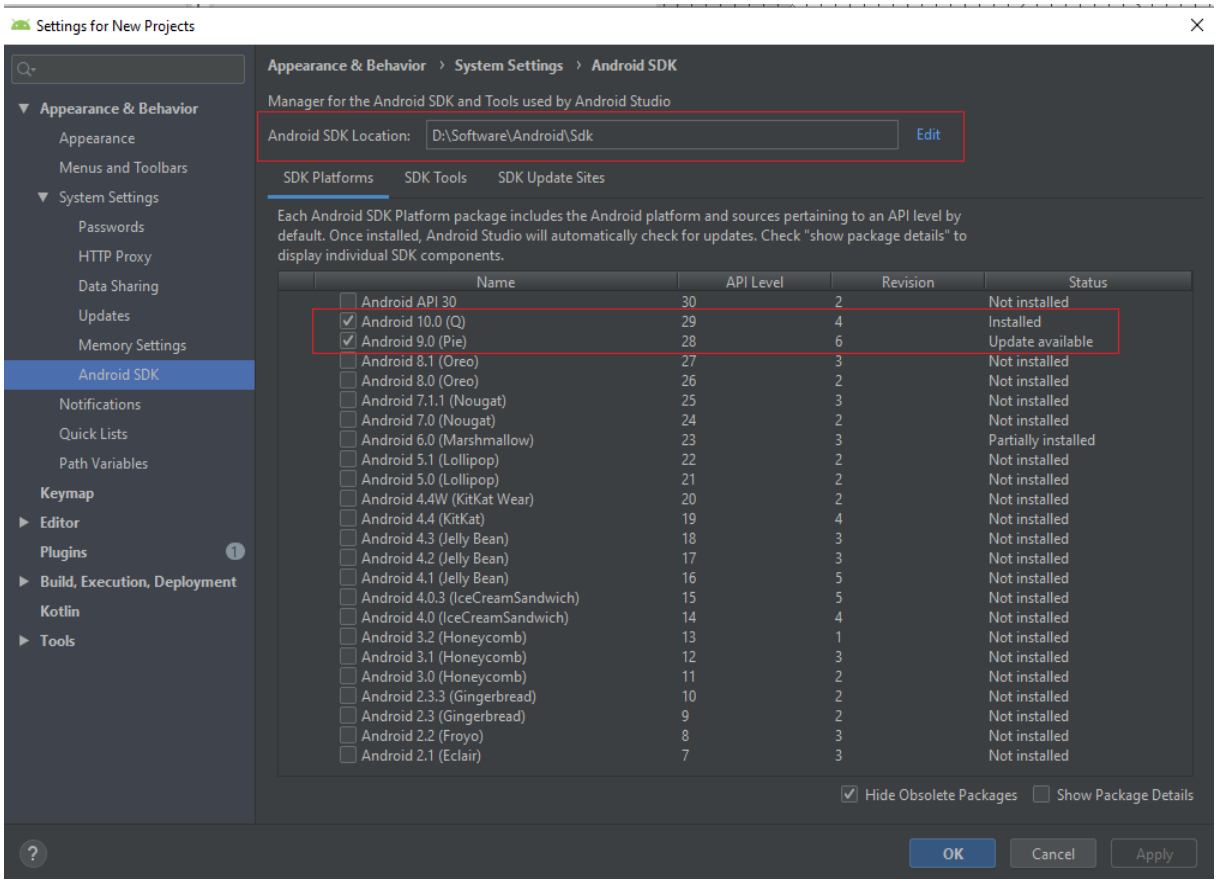


Figure 18. The SDK manager menu.

Figure 29 illustrates the UI for managing SDK. The full detailed information of SDK is available here, including Android SDK Location and installed SDK platforms. I have installed Android 10.0 (Q) and Android 9.0 (P) for easy maintenance and well-support. Within the time of doing this thesis, most Android devices have been updated to Android 9. Built on Android 9, my application can run on most Android devices to do this thesis.

After installing all the required SDK platforms, the next step is to create an Android Virtual Device (AVD). Choosing AVD Manager in Configure Menu jumps right into the AVD Manager Menu.

CHAPTER 5

EXPERIMENTAL RESULT

5.1. Dataset Description

To improve the performance of image classifiers for plant disease detection, an enormous verified dataset of images of the infected and regular leaf was demanded for a long time until PlantVillage [35], the name of an online platform committed to crop diseases and healthy sponsored by many laboratory experimentation stations affiliated with Land Grant Universities in the US, which contributes the dataset by collect thousands of images in many conditions. The Plant Village's dataset records contain 54,305 images of healthy and 26 diseases from 14 crop species: Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Bell. Pepper, Potato, Raspberry, Soybean, Squash, Strawberry, Tomato. Furthermore, studies on the crucial role of rice plants in the Vietnamese agricultural economy, several researchers take a colossal effort to conduct a brand-new Rice dataset [36], which consisted of 3,355 images from 4 types of rice diseases, has been added to enhance our dataset's diversity to identify plant disease significantly.

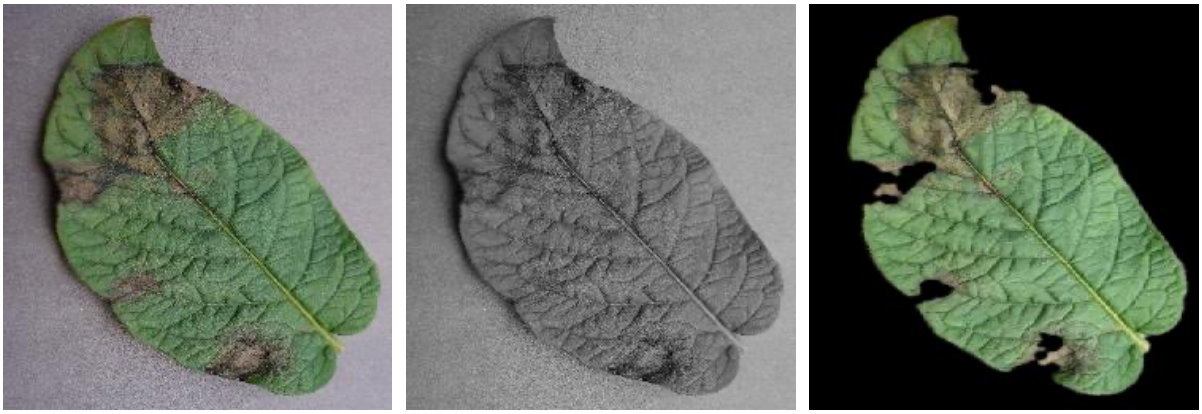
Name (images)	Status (images)
Apple (3,171)	Healthy (1,645) Apple Scab (630) Black Rot (621) Cedar Apple Rust (275)
Blueberry (1,502)	Healthy (1,502)
Cherry (1,906)	Healthy (854) Powdery Mildew (1,052)
Corn (3,852)	Healthy (1,162) Cercospora Leaf Spot Gray Leaf Spot (513) Common Rust (1,192) Northern Leaf Blight (985)
Grape (4,062)	Healthy (423)

	Black Rot (1,180) Esca Black Measles (1,383) Isariopsis Leaf Spot (1,076)
Orange (5,507)	Haunglongbing “Citrus greening” (5,507)
Peach (2657)	Healthy (360) Bacterial Spot (2,297)
Bell Pepper (2,475)	Healthy (1,478) Bacterial Spot (997)
Potato (2,152)	Healthy (152) Early Blight (1,000) Late Blight (1,000)
Raspberry (371)	Healthy (371)
Soybean (5,090)	Healthy (5,090)
Squash (1,835)	Powdery Mildew (1,835)
Strawberry (1,565)	Healthy (456) Leaf Scorch (1,109)
Tomato (18,160)	Healthy (1,591) Bacterial Spot (2,127) Early Blight (1,000) Late Blight (1,909) Leaf Mold (952) Septoria Leaf Spot (1,771) Spider Mites Two-spotted or Spider Mite (1,676) Target Spot (1,404) Tomato Mosaic Virus (373) Tomato Yellow Leaf Curl Virus (5,357)
Rice (3,355)	Leaf Blast (780) Hispa (566) Healthy (1489) BrownSpot (524)

Table 1. List of disease and species in PlantVillage and Leaf dataset.

Our dataset includes three illustrations of the image data: color image, gray-scaled version, and a segmented-variant dataset (see Figure 20). All three versions show a specific difference in achievement over the entire experiments with models. Because of the tremendous number of images, which more likely lead to overfitting issues, the dataset was separated for our classification model to train 80% of the data, validate with 10% and 10% remainder of the data to guarantee that it fits further data or predict later observations positively.

(a) Color 1 (b) Grayscale 1 (c) Segmented 1



(d) Color 2 (e) Grayscale 2 (f) Segmented 2

Figure 19. Example of three different versions of healthy and diseases leaf images used in several configurations.

5.2. Evaluation

In this thesis, we examine the performances of some ConvNet such as VGGNet-16, Inception-v3, ResNet-50, MobileNet-v2, and EfficientNet-B1 models on the phrase break prediction task. We used the TensorFlow learning library, running the Keras backend to build our models. For all models, the learning rate was fixed at 1e-3. All the ConvNet models were trained with batch sizes of 64, while the optimizer used in this thesis was Adam. All the code, as well as data used in this work, are available online.

At first, the image dataset was split into training, validation, and testing image sets with the ratio of 80%, 10%, and 10%, respectively. The augmentation for the training images is achieved by width and height shifting, flipping, and zooming the training images to enrich our dataset, which helps the model prevents overfitting issues. The Transfer Learning technique was utilized by removing the last layers of a pre-trained model and substituting them with layers that can acquire the essential features distinct to the training dataset. Transfer learning diminishes execution time and enhances accuracy when compared to models that train from scratch. the method uses five different pre-trained models, namely, VGGNet-16, Inception-v3, ResNet-50, MobileNet-v2 and EfficientNet-B1.

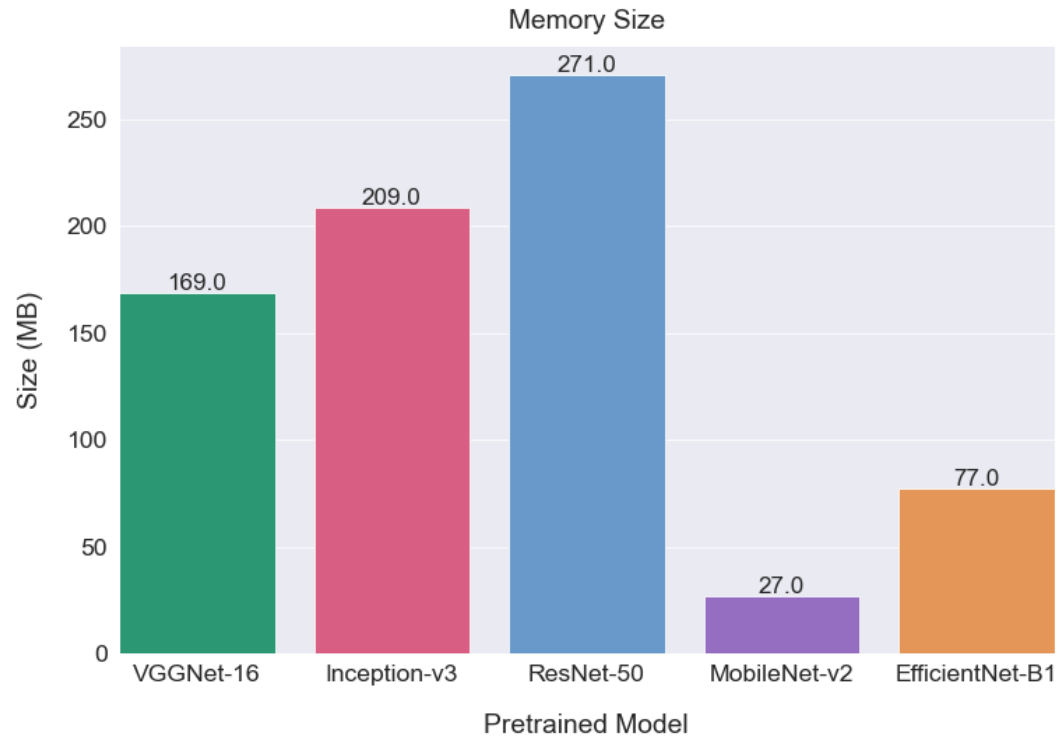


Figure 20. A visual representation of memory size of five different pre-trained models.

Depending upon each model's memory size, the pre-trained models are classified into smaller and bigger models. The smaller models use less than 30MB and therefore are firmly fit for edge apps.

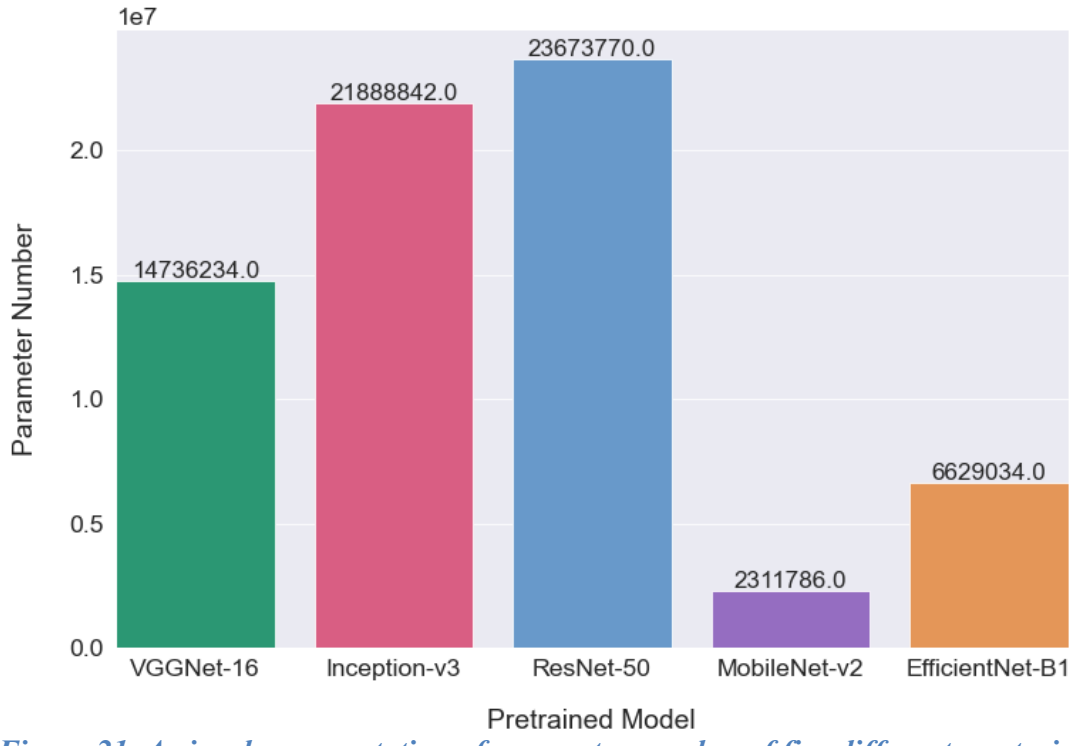


Figure 21. A visual representation of parameter number of five different pre-trained models.

Among the larger models, ResNet-50 had the most extensive parameter of approximately 23.6 million, whereas VGGNet-16 had the lowest number of approximately 14.7 million parameters. When it comes to the smaller mobile-oriented models, MobileNet-v2 had a number parameter of approximately 2.3 million compared to EfficientNet-B1, which had approximately 6.6 million parameters.

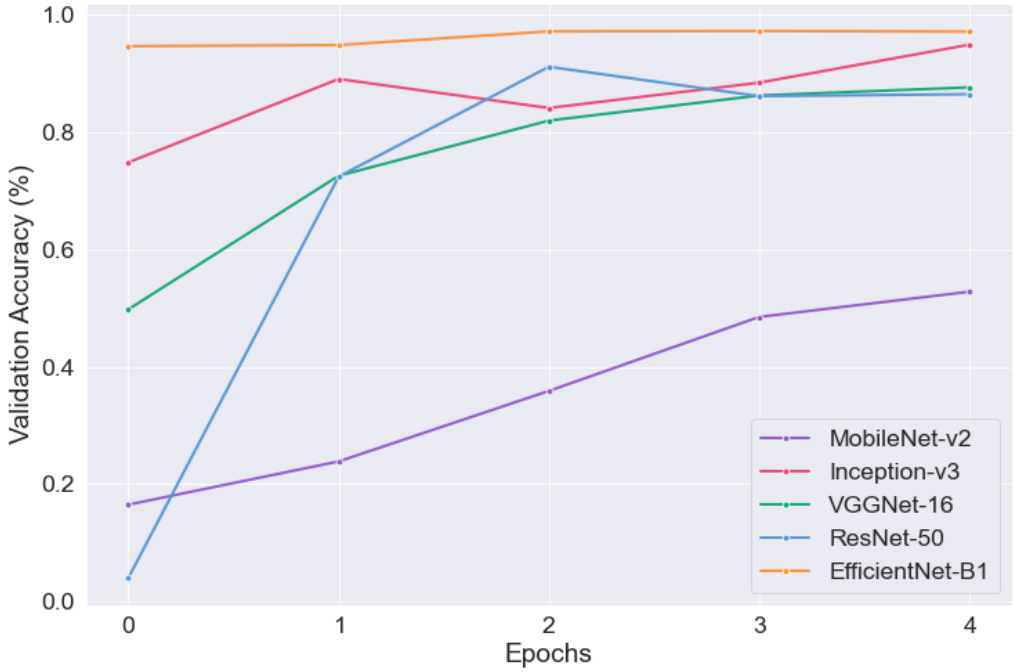


Figure 22. A visual representation of validation accuracy with five epochs of five different pre-trained models.

Amongst the larger models, EfficientNet-B1 had the highest validation accuracy after five epochs of approximately 97.2%, whereas the smaller mobile-oriented models MobileNet-v2 had only 52.8%.

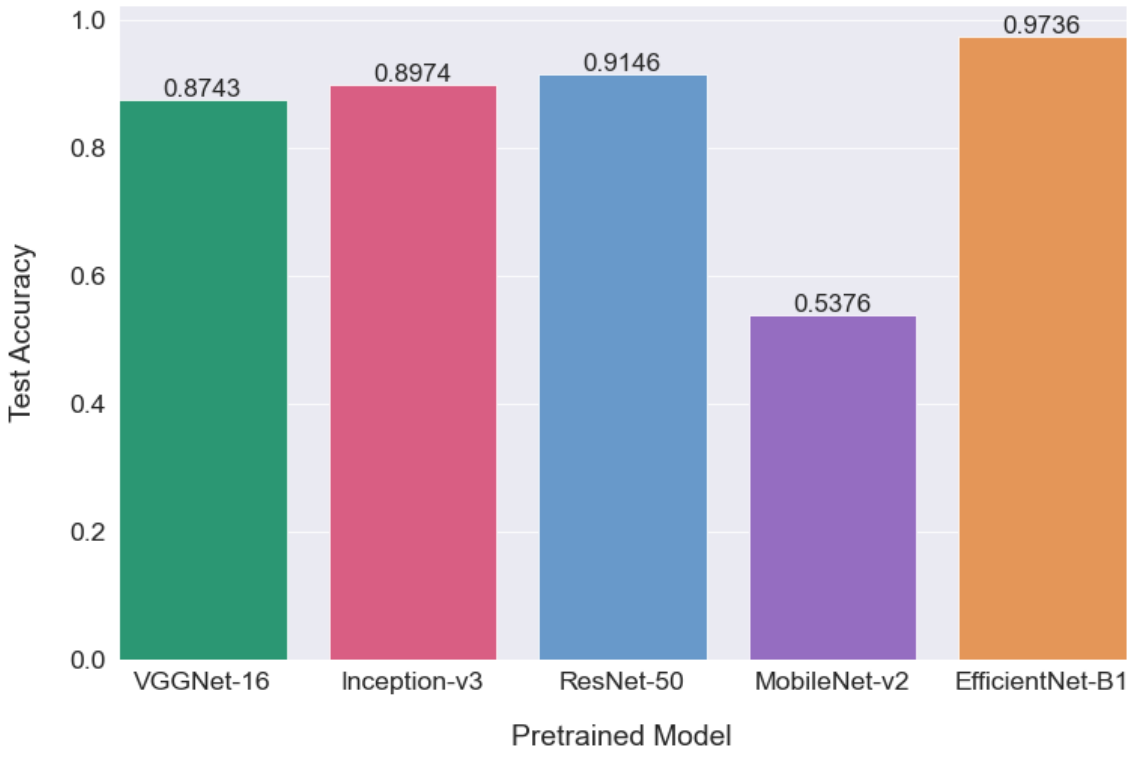


Figure 23. A visual representation of testing accuracy of five different pre-trained models.

When it comes to the test set's accuracy, EfficientNet-B1 performs gorgeously with the highest testing accuracy of nearly 97.36%, whereas the smaller models, MobileNet-v2, produced the lowest testing accuracy of only 53.76%.

In consequence, after several experiments with transfer learning techniques to classify infected and healthy leaves image, EfficientNet-B1 archived the highest testing accuracy of 97.36% and only use a decent memory size (77MB) thanks to the combination of compound scaling and efficient base model, which not only deliver impressive results in our problems but also best suited for mobile applications with memory and processing constraints.

After that, we run some experiments to fine-tune the model by changing some hyperparameter to enhance the accuracy and performance of EfficientNet when applying to plant diseases classification problem. Table 3 shows the parameter as well as hyperparameter values used to fine-tune our model.

Model	Type of Optimizer	Batch size	Learning rate callback	Epochs
EfficientNet-B1	RMSprop	64	ReduceLROnPlateau (Reduce learning rate when accuracy has stopped improving)	5
EfficientNet-B1	Adam	64		
EfficientNet-B1	SGD	64		
EfficientNet-B1	RMSprop	128		
EfficientNet-B1	Adam	128		
EfficientNet-B1	SGD	128		

Table 2. List of parameters as well as hyperparameter used in each experiment

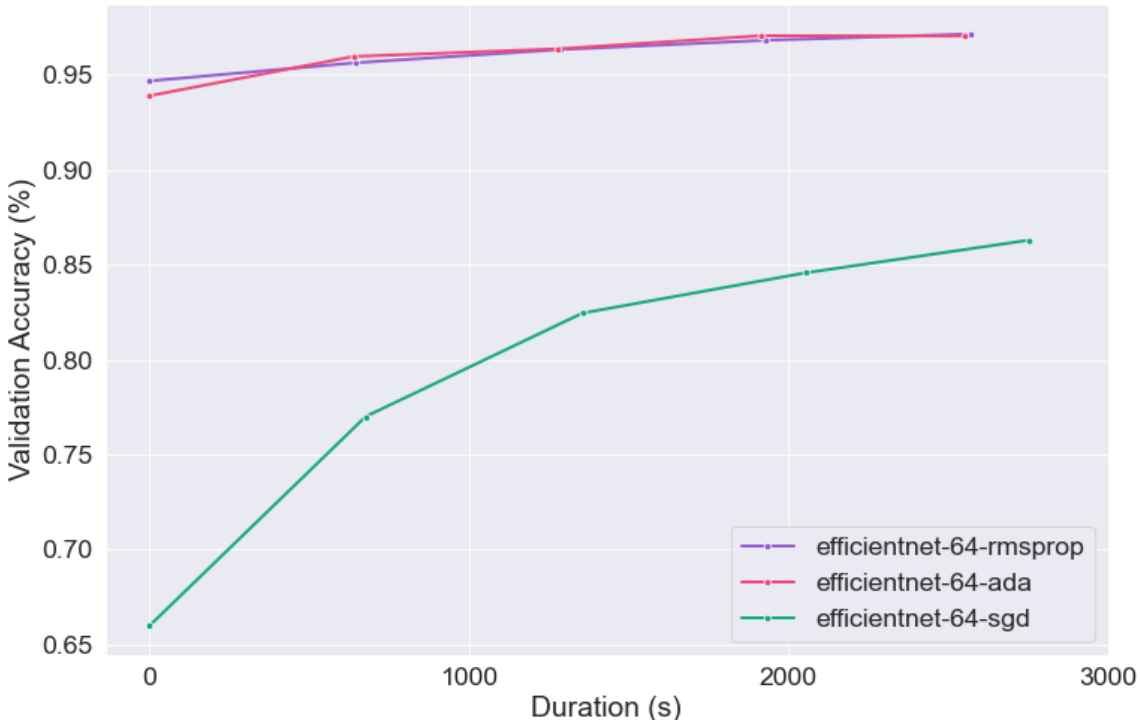


Figure 24. A visual representation of validation accuracy of first 3 experiments.

From figure 35, RMSprop performs slightly better than Adam optimizer with the highest validation accuracy of approximately 97.14% and 97.04%, respectively, which means RMSprop reduces more than 3% that model makes a wrong classification.

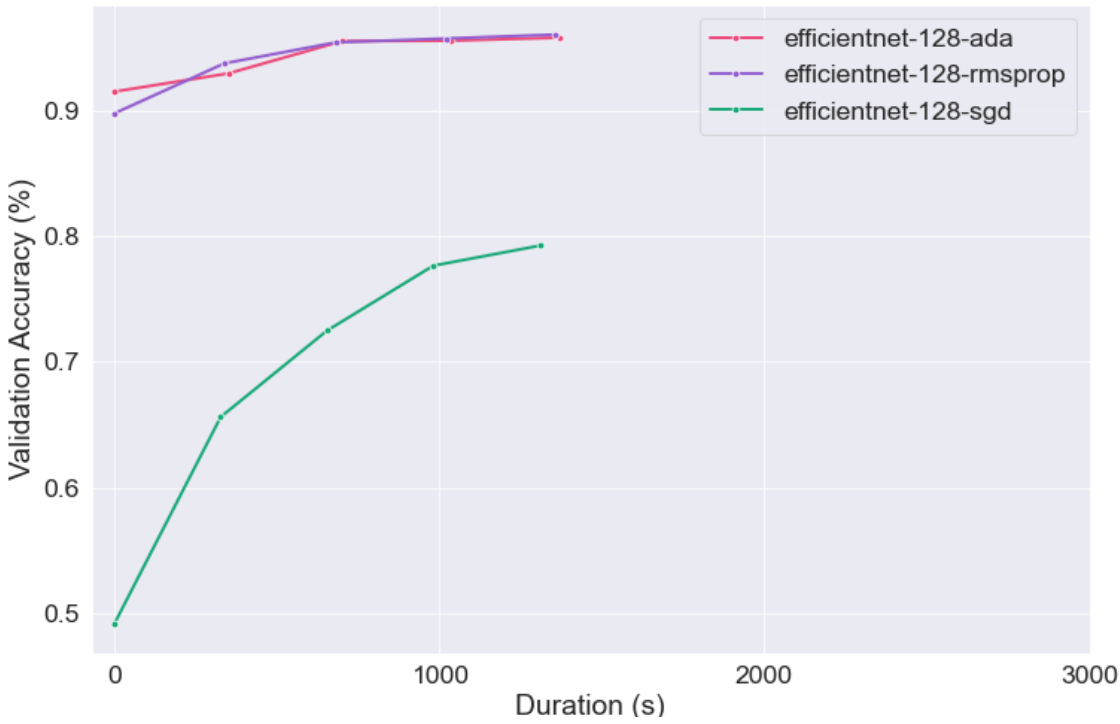


Figure 25. A visual representation of validation accuracy of next 3 experiments.

From figure 36, even though we increase the batch size to speed up training progress, RMSprop still archive better result than Adam optimizer with the validation accuracy of approximately 96.07% and 95.83%, respectively.

All in all, we use TensorFlow-Lite to convert base EfficientNet-B1 trained with RMSprop optimizer and 128 batch sizes to a new model particular format optimized for speed or storage. With this unique format, the model can be deployed on edge devices like mobiles using Android or iOS to easily make the Edge inference.

5.3. Illustration

Figures 37 show the final result of the Main Screen depending on wild frames in the User Interface Design part. The simple UI makes it easier for the user to get used to the app.

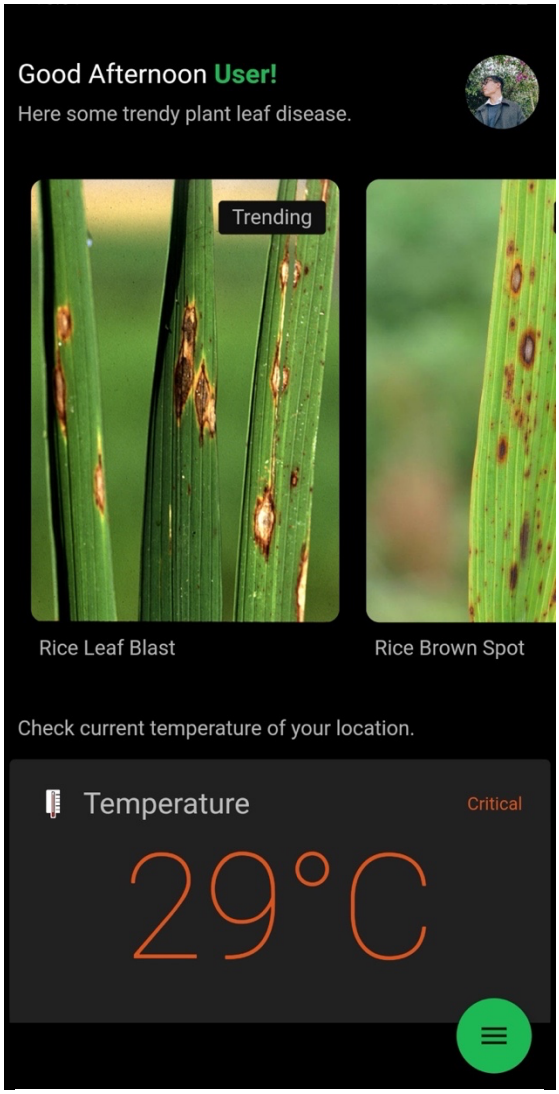


Figure 26. Main Dashboard Screen.

The main Dashboard content is shown in the figure. In the top corner, there is a container that displays the user image profile and greeting message. Next, a stack contains multiple cards of trendy diseases, which enable the user to view information about these diseases quickly. Following that is the container, which illustrates the temperature in Celsius degree of user's location.

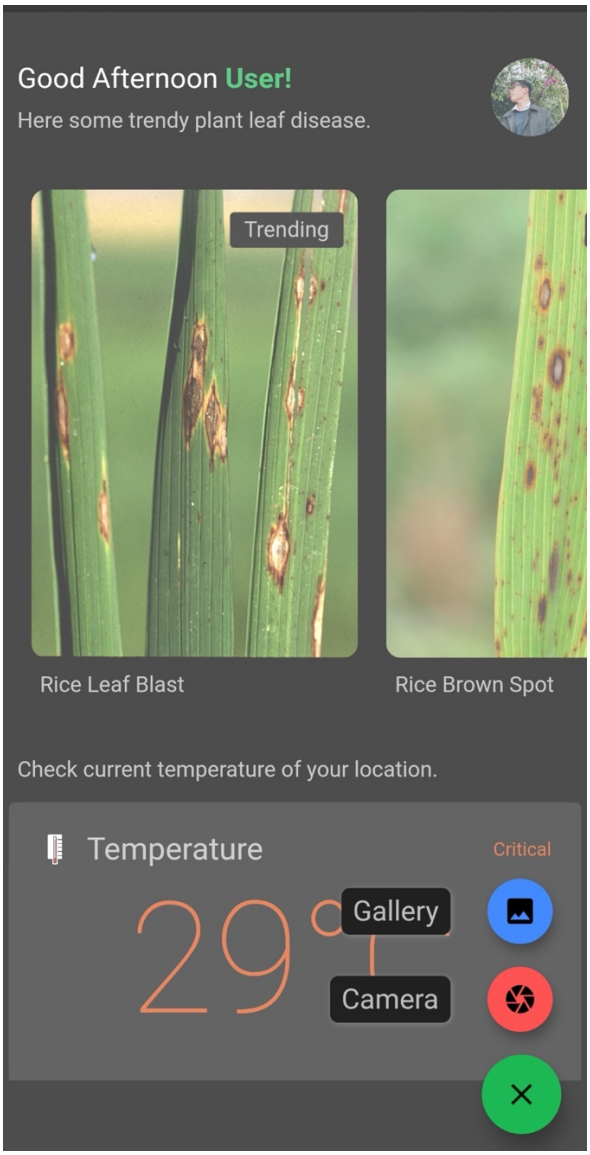


Figure 27. Screen when user tap to Input Choosing Button.

The figure 38 illustrates The Input Choosing Button allows the user to choose which type of image input for the application helps the user pick images from the image library and take new pictures with the camera effortlessly.

Our fine-tuned EfficientNet-B1, which converts to TensorFlow-Lite format to easily integrate into the application, will automatically load every time when the user launches it. When the user inputs the image, the model will take a duration to process and predict, then return the class name and the probability if the model is able to recognize it, which illustrates by the figure below.

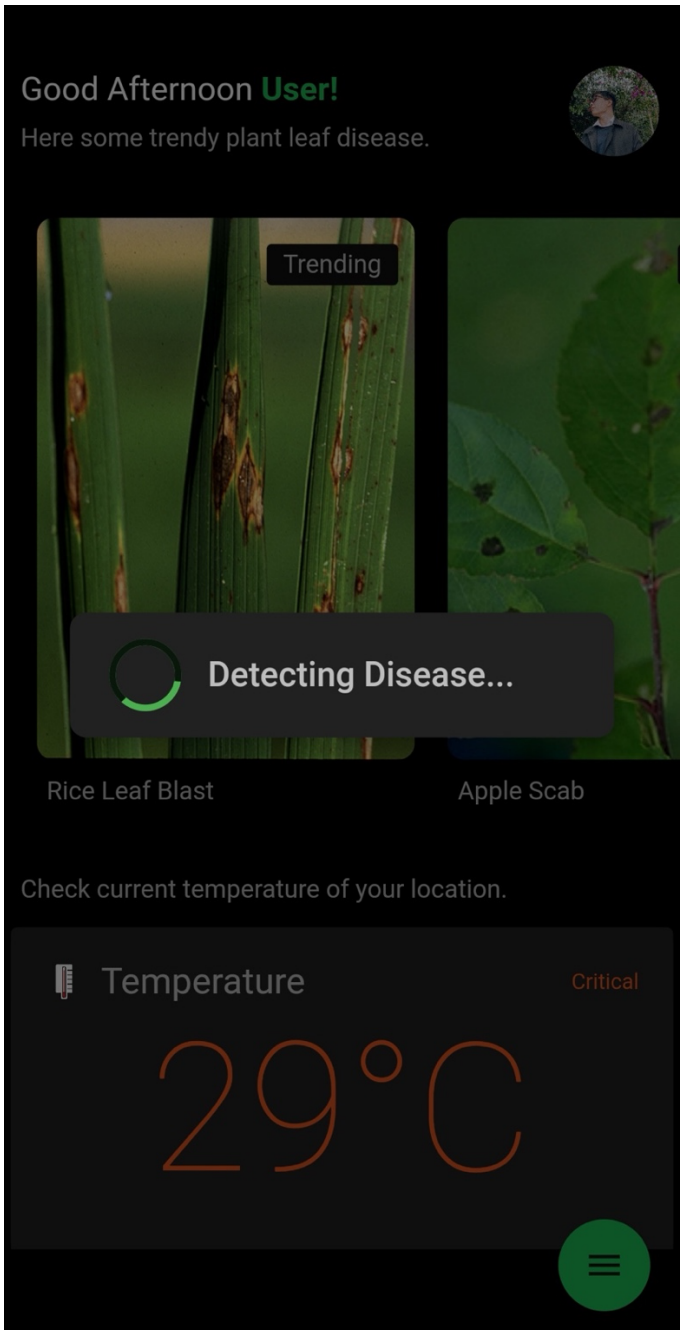


Figure 28. Screen when the application processing the input image.

When the model recognizes the image class, the application returns the Diseases screen (shown in figure 39), which carries the extra information consisting of several other similar images of this disease and the symptoms, which provide the farmer with significant advice to identify the diseases.

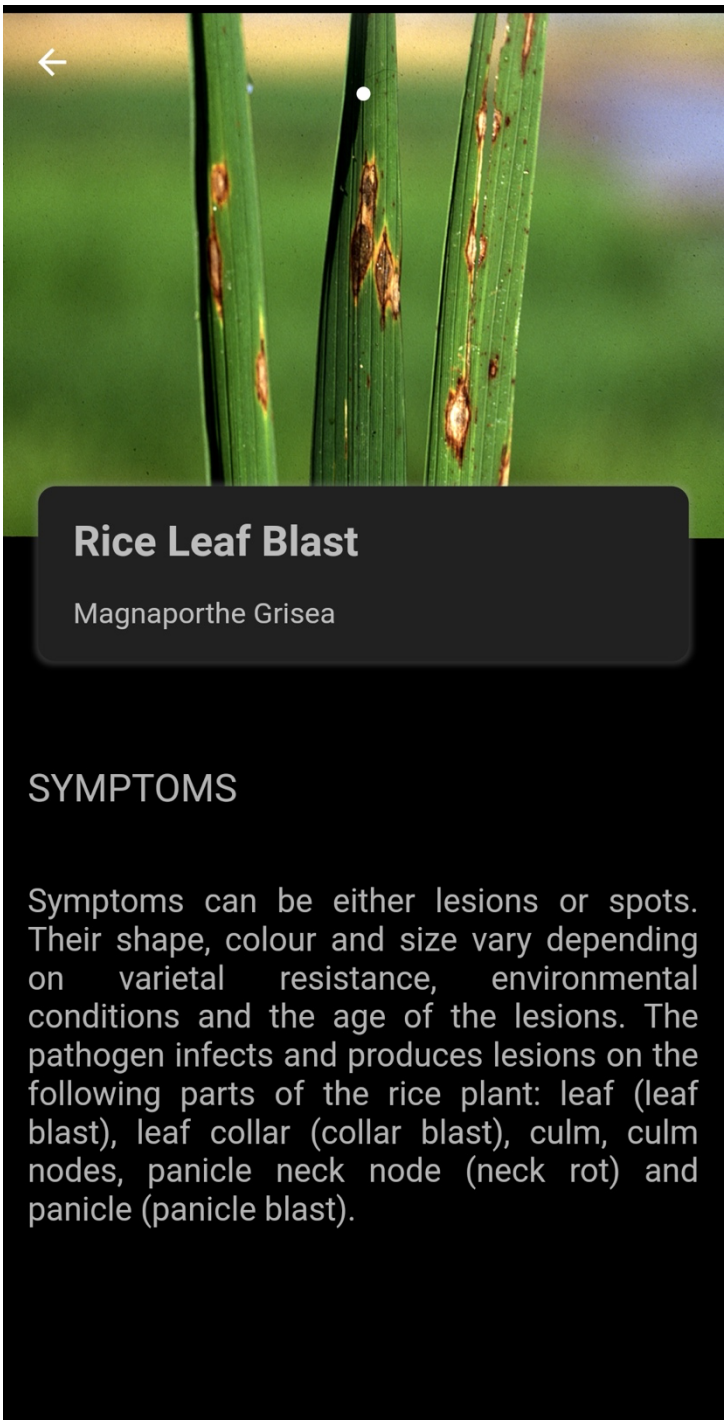


Figure 29. The Rice Leaf Blast Disease Screen.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. Conclusion

In general, the demand for mobility increases with the help of cutting-edge technologies, solutions to support the farmer to change their lives positively. This thesis's idea is not new, but it is a good practice in the Deep learning approach, mobile application and combines many technologies to build a model that integrated to mobile that can be a useful assistant in life. The model has supported many plants and diseases, but the number of plant diseases is limited due to a shortage of this specific data.

In conclusion, this thesis proposes an application as a solution to help and provides useful information for the farmer to identify diseases.

6.2. Future work

During project development, it is inevitable that the application still has many shortcomings since I concentrate on several critical features related to the classification of plant diseases. The performance of mobile applications is still not well-optimized, and the user interface UI maybe not clear enough to avoid confusion when using. Furthermore, in the future, I would add more features to promote the ability to popularly used this application:

- Cloud infrastructures: to standardize user data on one machine or in a datacenter.
- Federated Learning: to enable mobile phones to train a model on edge and inference while maintaining all the training data on the device, which is a better way to protect user privacy.
- Re-design: to enhance the user experience. The mobile app now is required a new approach, which hyper intuitive and visually incredible interface designs. Following this trend is considered a critical stage of the development process.

REFERENCES

[1] The World Bank, C. (2020, Oct 6). "Vietnam Overview". The World Bank. https://www.worldbank.org/en/country/vietnam/overview

[2] Santanu Phadikar & Jaya Sil (2008) Rice Disease Identification Using Pattern Recognition Techniques, Proceedings Of 11th International Conference On Computer And Information Technology

[3] Ms. Kiran R. Gavhale, Prof. Ujwalla Gawande, and Mr. Kamal O. Hajari, "Unhealthy Region of Citrus Leaf Detection using Image Processing Techniques," IEEE International Conference on Convergence of Technology

[4] Tejal Deshpande, Sharmila Sengupta, and K.S.Raghuvanshi, "Grading & Identification of

Disease in Pomegranate Leaf and Fruit," IJCSIT

[5] A. K. Reyes, J. C. Caicedo, and J. E. Camargo, "Fine-tuning deep convolutional networks for plant recognition," in Proceedings of the Working Notes of CLEF 2015 Conference, 2015

[6] K. Elangovan and S. Nalini, (2017), "Plant Disease Classification Using Image Segmentation and SVM Techniques", ISSN 0973-1873 Vol 13

[7] Saleem MH, Potgieter J and Arif KM, (2020), "Plant Disease Classification: A Comparative Evaluation of Convolutional Neural Networks and Deep Learning Optimizers", Plants 9(10):1319

[8] Kabir, Muhammad & Ohi, Abu & Ph. D., M.. (2020), "A Multi-Plant Disease Diagnosis Method using Convolutional Neural Network"

[9] Hossain, Md Selim, et al, (2018), "Recognition and detection of tea leaf's diseases using support vector machine." Signal Processing & Its Applications (CSPA), 2018 IEEE 14th International Colloquium on. IEEE, 2018.

[10] Singh, Jaskaran, and Harpreet Kaur, (2018), "A review on: Various techniques of plant leaf disease detection." 2018 2nd International Conference on Inventive Systems and Control (ICISC). IEEE, 2018.

[11] Marshall Hargrave, (2020, Nov 24). "Deep Learning". Investopedia. https://www.investopedia.com/terms/d/deep-learning.asp

[12] Marius-Constantin Popescu, Valentina E. Balas, Liliana Perescu-Popescu, and Nikos Mastorakis, (2009, July). "Multilayer perceptron and neural networks". WSEAS Trans. Cir. and Sys. 8, 7, 579–588.

[13] Sepp Hochreiter and Jürgen Schmidhuber, (1997), "Long Short-Term Memory," Neural Computation 9, no. 8, 1735–1780.

[14] Varshini, Chava & Hruday, Gurram & Chandu, Guguloth & Sharif, Shaik. (2020). "Sign Language Recognition". International Journal of Engineering Research and. V9. 10.17577/IJERTV9IS050781.

[15] Irhum Shafkat, (2018, Jun 2). "Intuitively Understanding Convolutions for Deep Learning" Towards Data Science. https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

[16] Yamaguchi, Kouichi; Sakamoto, Kenji; Akabane, Toshio; Fujimoto, Yoshiji (November 1990). A Neural Network for Speaker-Independent Isolated Word Recognition. First International Conference on Spoken Language Processing (ICSLP 90). Kobe, Japan.

[17] Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). "Multi-column deep neural networks for image classification". 2012 IEEE Conference on Computer Vision and Pattern Recognition. New York, NY: Institute of Electrical and Electronics Engineers (IEEE). pp. 3642–3649.

[18] Scherer, Dominik; Müller, Andreas C.; Behnke, Sven (2010). "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". Artificial Neural Networks (ICANN), 20th International Conference on. Thessaloniki, Greece: Springer. pp. 92–101.

[19] Rachit Tayal, (2020, May 26). "Deep Learning for Computer Vision". Towards Data Science. https://towardsdatascience.com/deep-learning-for-computer-vision-c4e5f191c522

[20] Yann LeCun et al, (1998). "Gradient-Based Learning Applied to Document Recognition," Proceedings of the IEEE 86, no. 11, 2278–2324.

[21] Gómez Blas, Nuria; de Mingo López, Luis F.; Arteta Albert, Alberto; Martínez Llamas, Javier. (2020). "Image Classification with Convolutional Neural Networks Using Gulf of Maine Humpback Whale Catalog" Electronics 9, no. 5: 731.

[22] Alex Krizhevsky et al, (2012), "ImageNet Classification with Deep Convolutional Neural Networks," _Proceedings of the 25th International Conference on Neural Information Processing Systems 1: 1097–1105.

[23] Pedraza, Anibal & Gallego, Jaime & Lopez, Samuel & Gonzalez, Lucia & Laurinavicius, Arvydas & Bueno, Gloria. (2017). "Glomerulus Classification with Convolutional Neural Networks". 839-849. 10.1007/978-3-319-60964-5_73.

[24] Christian Szegedy et al, (2015), "Going Deeper with Convolutions," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition: 1–9.

[25] Huy Hoang, (2018, August 19)."Overview Convolutional Neural Networks". Andpythings. https://andpythings.wordpress.com/2018/08/19/overview-convoluational-neural-networks/

[26] Szegedy, Christian & Vanhoucke, Vincent & Ioffe, Sergey & Shlens, Jon & Wojna, ZB. (2016). Rethinking the Inception Architecture for Computer Vision. 10.1109/CVPR.2016.308.

[27] Karen Simonyan and Andrew Zisserman, (2014), "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv preprint arXiv:1409.1556.

[28] Tanya Dixit, (2018, Jun 16). "Rethinking the Inception Architecture for Computer Vision — Part 1". Medium. https://medium.com/coinmonks/rethinking-the-inception-architecture-for-computer-vision-part-1-2938cc7c7872

[29] Kaiming He et al, (2015). "Deep Residual Learning for Image Recognition," arXiv preprint arXiv:1512:03385.

[30] Ložnjak, Stjepan & Kramberger, Tin & Cesar, Ivan & Kovačević, Renata. (2020). "Automobile Classification Using Transfer Learning on ResNet Neural Network Architecture". 10.19279/TVZ.PD.2020-8-1-18.

[31] Tan, Mingxing & Chen, Bo & Pang, Ruoming & Vasudevan, Vijay & Sandler, Mark & Howard, Andrew & Le, Quoc. (2019). "MnasNet: Platform-Aware Neural Architecture Search for Mobile". 2815-2823. 10.1109/CVPR.2019.00293.

[32] Tan, M. & Le, Q. V. (2019), "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", cite arxiv:1905.11946Comment: Published in ICML 2019.

[33] Howard, Andrew & Zhu, Menglong & Chen, Bo & Kalenichenko, Dmitry & Wang, Weijun & Weyand, Tobias & Andreetto, Marco & Adam, Hartwig. (2017). "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications".

[34] Hughes, David & Salathe, Marcel. (2015). "An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing".

[35] Huy Do, (2019, Aug). "Rice Diseases Image Dataset: An image dataset for rice and its diseases". Kaggle. https://www.kaggle.com/minhhuy2810/rice-diseases-image-dataset