INTRODUCTION

The old and classical approach for detection and recognition of plant diseases is based on naked eye observation, which is very slow method and also gives less accuracy. In some countries, consulting experts to find out plant disease is expensive and time consuming due to unavailability of expert. Irregular check up of plant results in growing of various diseases on plant which requires more chemicals to cure it also these chemicals are toxic to other animals, insects and birds which are helpful for agriculture. Automatic detection of plant diseases is essential to detect the symptoms of diseases in early stages when they appear on the growing leaf of plant.

1.1 Objective

Our project aim is to build mobile application to determine type of disease the leaf has been affected by, using deep learning methods to obtain accurate results **.**

1.2 Problem Statement

The manual observation of farmers is the main approach adopted in practice for detection and identification of plant diseases.Early detection is critical. But this approach requires continuous monitoring of plants which might be expensive and difficult especially in large farms.Many Solutions are proposed using MATLAB but these solutions are economically not feasible.

1.3 Existing system

The naked eye observation of experts is the main approach adopted in practice for detection and identification of plant diseases. But, this requires continuous monitoring of experts which might be prohibitively expensive in large farms. Further, in some developing countries, farmers may have to go long distances to contact experts, this makes consulting experts too expensive and time consuming and moreover farmers are unaware of non-native diseases.

It is envisaged to make available relevant information and services to the farming community and private sector or through the use of information and communication technologies, to supplement the existing delivery channels provided for by the department. Farmers‟ Portal is an endeavor in this direction to create one stop shop for meeting all information needs relating to agriculture of an Indian farmer. Once in the Farmers “Portal, a farmer will be able to get all relevant information on specific subjects around his village/block/district or state. Using the farmers‟ portal the farmer calls this service center and clears all his queries. The people present at the service search for relevant information or they are trained with all information to help farmers. This also has a disadvantage as the people at the service center cannot see the exact problem the farmer is facing. They cannot imagine the severity of the disease completely and hence at times it may result in wrong disease detection. This may also lead in destruction of the crop.

1.4 Proposed System

In this project,we have implemented an automated system for disease detection of plants using Mobilenet and integrated this system with a mobile application for Android phones to serve the farmers where they get benefitted by identifying the diseases correctly and taking measures accordingly.At first, an image of the defected plant has been taken with the camera of the mobile phone. The farmers then just have to send the image to our server by selecting the options provided on the application. The image has been analyzed in the server at the final step, the classification result along with the necessary control measurement has been sent server back to the user through the application on the phone.

SYSTEM REQUIREMENTS

2.1 Software requirements

- 64 bit Windows operating system
- Python
- Tensorflow
- Android Studio

2.1.1 Windows Operating System

64-bit.Update it to the final version, it has many updated software compared to previous version.Microsoft Windows is a group of several graphical operating system families, all of which are developed, marketed and sold by Microsoft. Each family caters to a certain sector of the computing industry. Active Windows families include Windows NT and Windows Embedded these may encompass subfamilies, e.g. Windows Embedded Compact (Windows CE) or Windows Server.

2.1.2 Installation of Python

To install Python and other scientific computing and machine learning packages simultaneously one should install Anaconda distribution. It is a Python implementation for Linux, Windows and OSX, and comprises various machine learning packages like numpy, scikit-learn, and matplotlib.

It also includes Jupyter Notebook, an interactive Python environment. Install Python 3.x version as per requirement.

To download the free Anaconda Python distribution from Continuum Analytics, one can do the following:

Visit the official site of Continuum Analytics and its download page. Note that the installation process may take 15-20 minutes as the installer contains Python, associated

packages, a code editor, and some other files. Depending on operating system, choose the installation process as explained here.

For Windows − Select the Anaconda for Windows section and look in the column with Python 2.7 or 3.x. one can find that there are two versions of the installer, one for 32-bit Windows, and one for 64-bit Windows. Choose the relevant one. Ensure that Anaconda's Python distribution installs into a single directory, and does not affect other Python installations, if any, on system.

Now, check if installation is successful. For this, go to the command line and type in the following command –

$ python

Python 3.5.2 |Anaconda custom (32-bit)|

Python Language

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast.

Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is

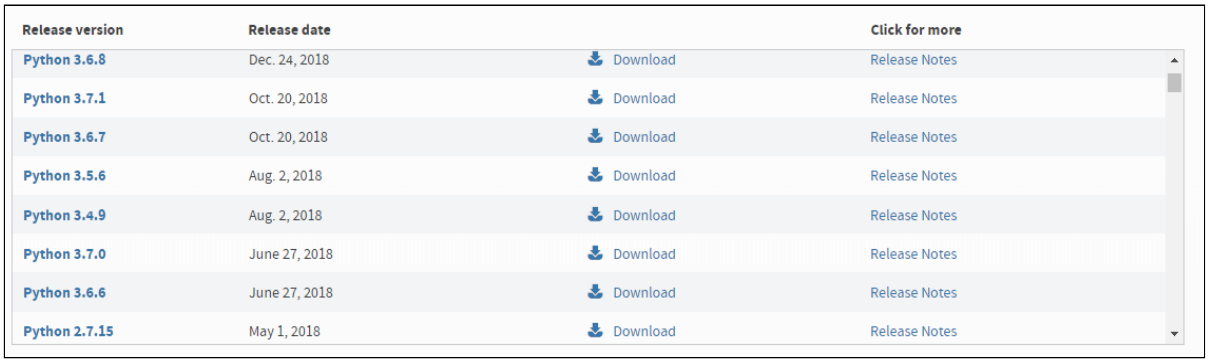
written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source.

Downloading Python

So to get started, here's how you can download the latest 64-bit Python 3.5.x if you have an older version or if you simply don't have it.

Step 1: Head over to Python 3.5.x from python.org

Step 2: Go to the Downloads page and Select the 3.5.2 download.



Release version	Release date		Click for more
Python 3.6.8	Dec. 24, 2018	Download	Release Notes
Python 3.7.1	Oct. 20, 2018	Download	Release Notes
Python 3.6.7	Oct. 20, 2018	Download	Release Notes
Python 3.5.6	Aug. 2, 2018	Download	Release Notes
Python 3.4.9	Aug. 2, 2018	Download	Release Notes
Python 3.7.0	June 27, 2018	Download	Release Notes
Python 3.6.6	June 27, 2018	Download	Release Notes
Python 2.7.15	May 1, 2018	Download	Release Notes

Fig 2.1: Python Downloads page

Step 3: After that you will be brought to another page, where you will need to select either the x86-64 or amd64 installer.

The one I specifically recommend for now is the Windows x86-64 executable installer.

Step 4: choose to Add Python 3.5 to PATH.

Step 5: Now you should be able to see a message saying Setup was successful. A way to confirm that it has installed successfully is to open your Command Prompt and check the version.

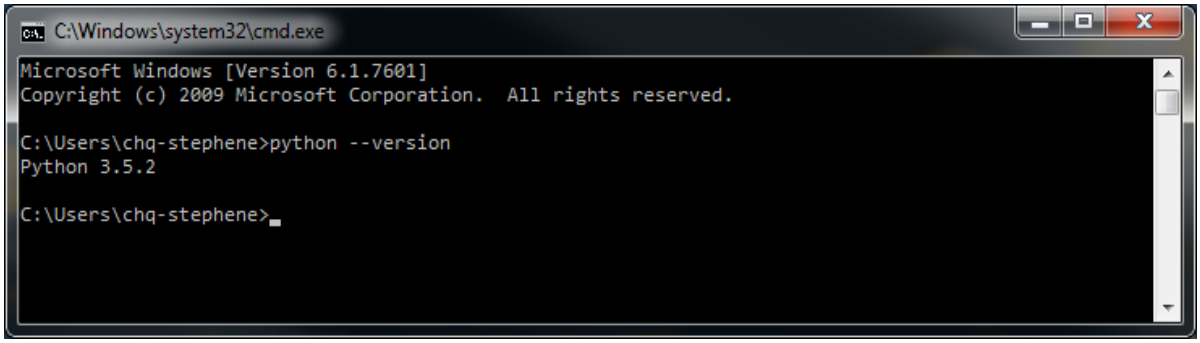


Figure 2.2 Python version check

2.1.3 Tensorflow Installation

TensorFlow is an open-source machine learning library for research and production. TensorFlow offers APIs for beginners and experts to develop for desktop, mobile, web, and cloud. See the sections below to get started.

Before you go on with the steps, make sure that your computer meets the requirements in order for TensorFlow to work on your computer.

The following are the necessary requirements:

- TensorFlow only supports 64-bit Python 3.5.x or Python 3.6.x on Windows.
- When you download the Python 3.5.x version, it comes with the pip3 package manager (which is the program that you are going to need in order for you use to install TensorFlow on Windows).

Now once you have downloaded the latest Python, you can now put up your finishing touches by installing your TensorFlow.

Step 1: To install TensorFlow, start a terminal. Make sure that you run the cmd as an administrator.

Here's how you can run your cmd as an administrator:

Open up the Start menu, search for cmd and then right click on it and Run as an administrator.

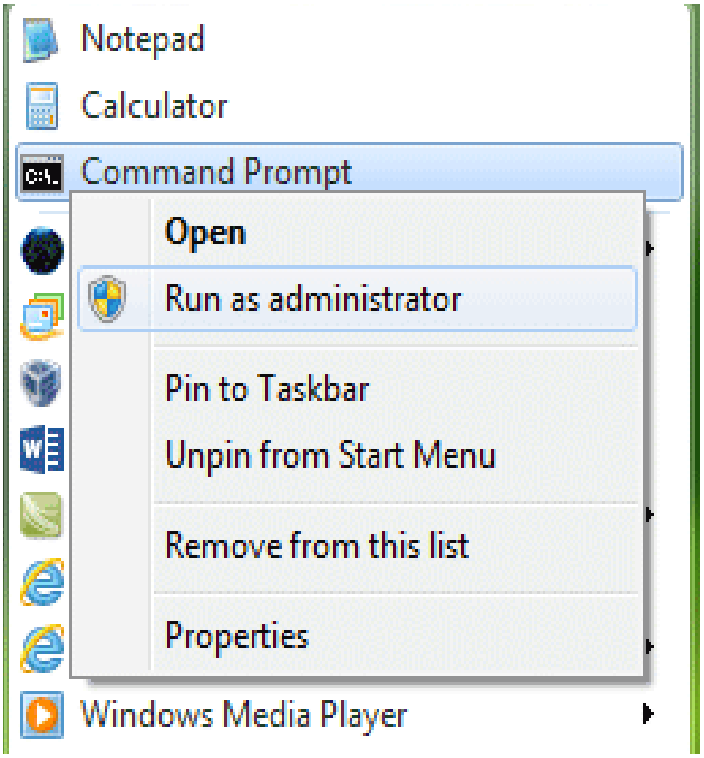


Fig 2.3: Running Command Prompt as Administrator

Step 2: Once you're done with that, now all you have to do is give just one simple little command for you to finish installing Tensorflow onto your Windows.

Just enter this command:

Create a environment named tensorflow by invoking the following command:

conda create -n tensorflow python=3.5

Activate the conda environment by issuing the following command:

activate tensorflow

Install TensorFlow:

conda install -c conda-forge tensorflow

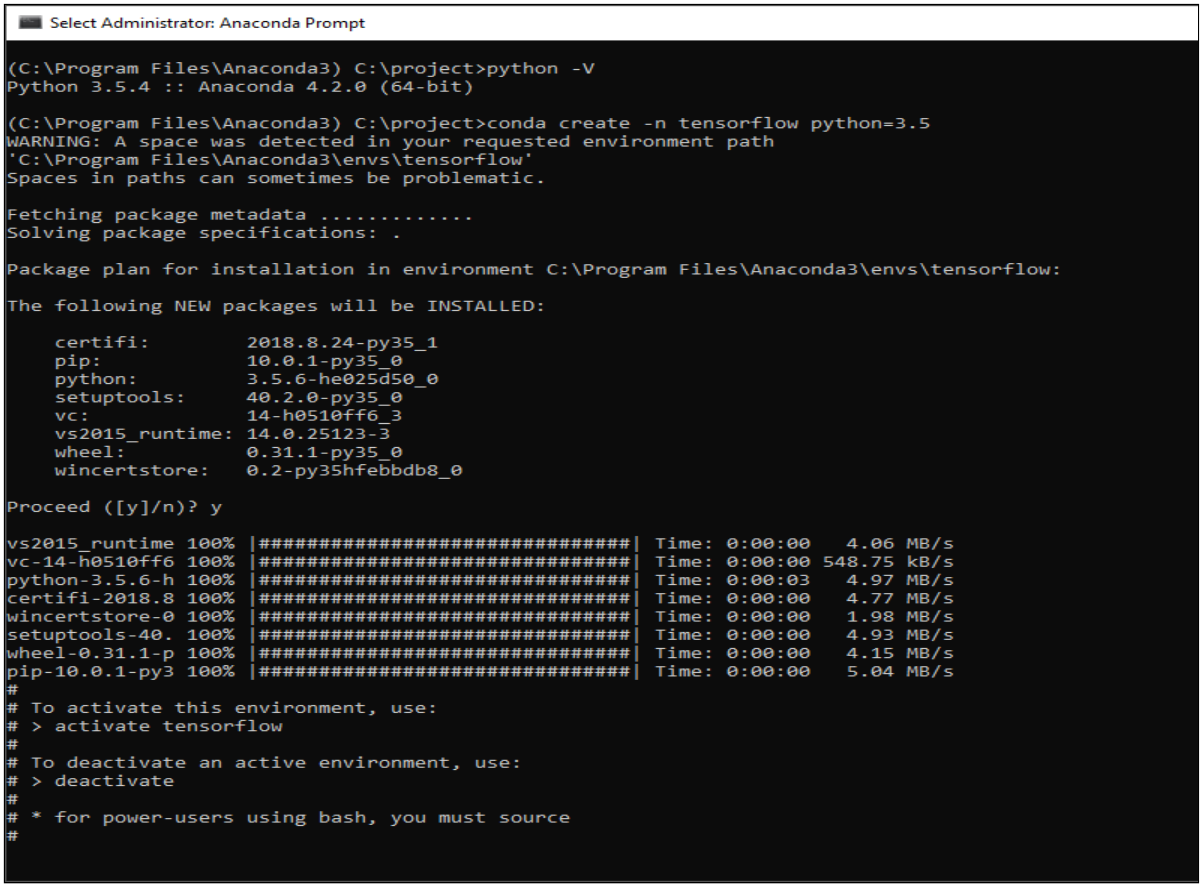


Fig 2.4: Tensorflow installation

After that we can check if all is correct by invoking python and trying the next program:

```
import tensorflow as tf
hail = tf.constant('Hello World')
session = tf.Session()
print(session.run(hail))
```

2.1.4 Android studio Installation

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems.It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development

Create an Android project:

Fig 2.5 : Android Project Window

In the Welcome to Android Studio window, click Start a new Android Studio project.

Or if you have a project opened, select File > New Project.

In the Create New Project window, enter the following values:

Application Name: "My First App"

Company Domain: "example.com"

You might want to change the project location. Also, if you want to write a Kotlin app, check the Include Kotlin support checkbox. Leave the other options as they are.

Click Next.

In the Target Android Devices screen, keep the default values and click Next.

In the Add an Activity to Mobile screen, select Empty Activity and click Next.

In the Configure Activity screen, keep the default values and click Finish.

After some processing, Android Studio opens the IDE.

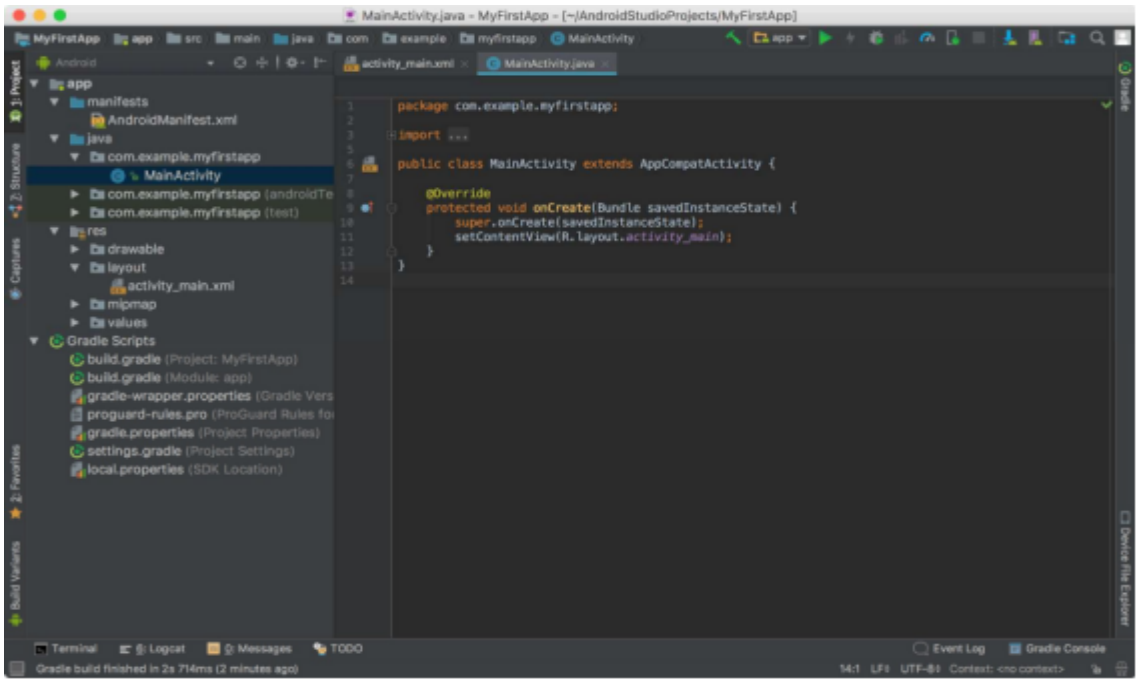


Fig 2.6: MainActivity Java Code

Now take a moment to review the most important files.

First, be sure the Project window is open (select View > Tool Windows > Project) and the Android view is selected from the drop-down list at the top of that window. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity

This is the main activity (the entry point for your app). When you build and run the app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml

This XML file defines the layout for the activity's UI. It contains a TextView element with the text "Hello world!".

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts > build.gradle

You'll see two files with this name: one for the project and one for the "app" module. Each module has its own build.gradle file, but this project currently has just one module. You'll mostly work with the module's build.gradle file to configure how the Gradle tools compile and build your app. For more information about this file, see Configure Your Build.

Run on a real device

Set up your device as follows:

- Connect your device to your development machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device.
- Enable USB debugging in the Developer options as follows.

First, you must enable the developer options:

- Open the Settings app.
- (Only on Android 8.0 or higher) Select System.
- Scroll to the bottom and select About phone.
- Scroll to the bottom and tap Build number 7 times.
- Return to the previous screen to find Developer options near the bottom.

Open Developer options, and then scroll down to find and enable USB debugging. Run the app on your device as follows:
In Android Studio, click the app module in the Project window and then select Run > Run (or click Run in the toolbar).

- In the Select Deployment Target window, select your device, and click OK.

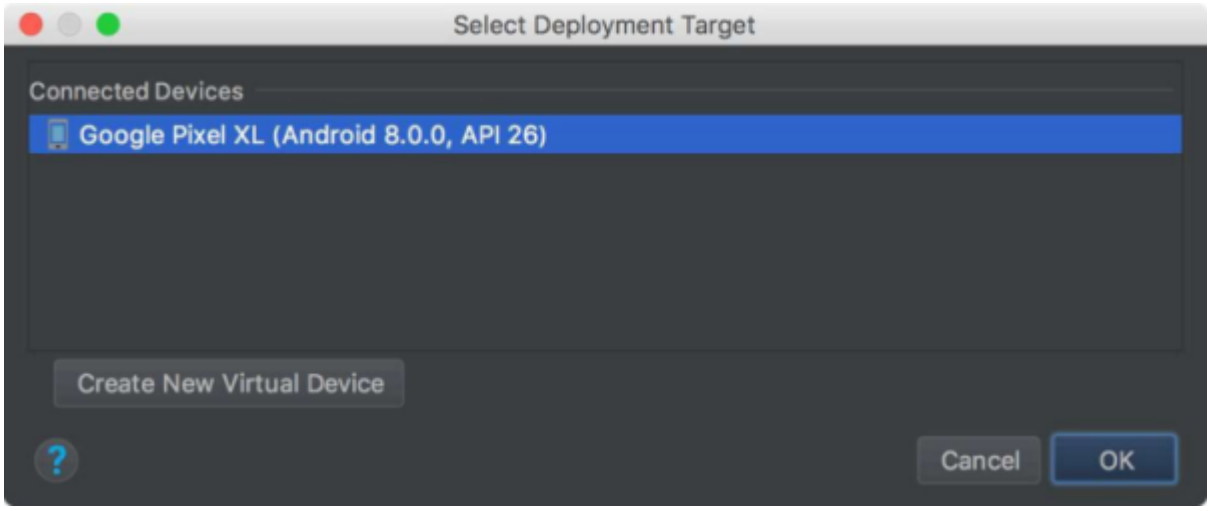


Fig 2.7 : Connected Devices

Android Studio installs the app on your connected device and starts it. You should now see "Hello World!" displayed in the app running on your device.

Run on an emulator

Run the app on an emulator as follows:

- In Android Studio, click the app module in the Project window and then select Run > Run (or click Run in the toolbar).
- In the select deployment target window, click Create New Virtual Device.

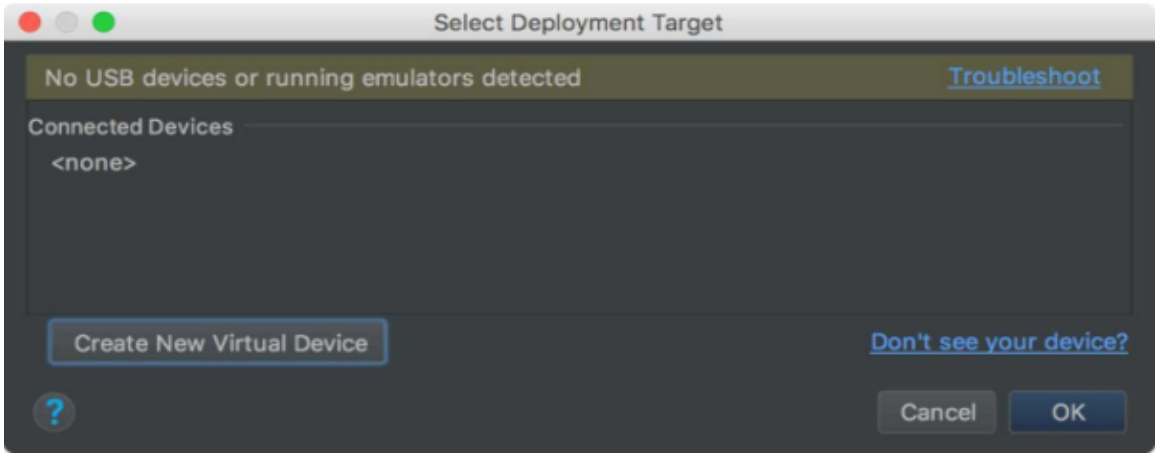


Fig 2.8 : Connected Devices

- In the Select Hardware screen, select a phone device, such as Pixel, and then click Next.
- In the System Image screen, select the version with the highest API level.
  If you don't have that version installed, a Download link is shown, so click that and complete the download.

  Click Next.
- On the Android Virtual Device (AVD) screen, leave all the settings alone and click Finish.

Back in the Select Deployment Target dialog, select the device you just created and click OK.

Android Studio installs the app on the emulator and starts it. You should now see "Hello World!" displayed in the app running on the emulator.

2.2 Hardware Requirements

- Android 5.0+
- Processor Intel i7 with NVIDIA GPU
- RAM- 8GB
- System type: 64-bit OS,X64-based process

2.3 Feasibility Study

Economical Feasibility:

In Artificial Agricultural Arborist application we are using the open source software. We are using Windows operating system.Tensorflow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. For application development purpose we are using android studio IDE which is also free to use , all the above mentioned are open source. They are economically feasible.

Technical Feasibility:

A prototype of Artificial Agricultural Arborist application was developed to verify the technical feasibility. The prototype of application is working successfully and hence the project is technically feasible.

LITERATURE SURVEY

3.1 Survey-1

Title	**MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**
Authors	Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam.
Year of Publication	April 2017
Summary	In this paper a class of efficient models called MobileNets for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses Depth-wise separable convolutions to build light weight deep neural networks. We introduce two simple global hyperparameters that efficiently trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem. We present extensive experiments on resource and accuracy tradeoffs and show strong performance compared to other popular models on ImageNet classification. We then demonstrate the effectiveness of MobileNets across a wide range of applications and use cases including object detection, finegrain classification, face attributes and large scale geo-localization.

3.2 Survey-2

Title	**Using Deep Learning for Image-Based Plant Disease Detection**
Authors	Sharada P. Mohanty, David P. Hughes and Marcel Salathé
Year of Publication	September 2016
Summary	Crop diseases are a major threat to food security, but their rapid identification remains difficult in many parts of the world due to the lack of the necessary infrastructure. The combination of increasing global smartphone penetration and recent advances in computer vision made possible by deep learning has paved the way for smartphone-assisted disease diagnosis. Using a public dataset of 54,306 images of diseased and healthy plant leaves collected under controlled conditions, we train a deep convolutional neural network to identify 14 crop species and 26 diseases (or absence thereof). The trained model achieves an accuracy of 99.35% on a held-out test set, demonstrating the feasibility of this approach. Overall, the approach of training deep learning models on increasingly large and publicly available image datasets presents a clear path toward smartphone-assisted crop disease diagnosis on a massive global scale.

3.3 Survey-3

Title	**Plant Disease Detection Using Image Processing**
Authors	Sachin D. Khirade , A.B. Patil
Year of Publication	November 2017
Publishing details	2015 International Conference on Computing Communication Control and Automation
Summary	Identification of the plant diseases is the key to preventing the losses in the yield and quantity of the agricultural product. The studies of the plant diseases mean the studies of visually observable patterns seen on the plant. Health monitoring and disease detection on plant is very critical for sustainable agriculture. It is very difficult to monitor the plant diseases manually. It requires tremendous amount of work, expertize in the plant diseases, and also require the excessive processing time. Hence, image processing is used for the detection of plant diseases. Disease detection involves the steps like image acquisition, image pre-processing, image segmentation, feature extraction and classification. This paper discussed the methods used for the detection of plant diseases using their leaves images. This paper also discussed some segmentation and feature extraction algorithm used in the plant disease detection.

METHODOLOGY

4.1 System Design

4.1.1 Proposed Technology

MobileNets are a new family of convolutional neural networks

There are a few things that make MobileNets awesome:

- They’re insanely small
- They’re insanely fast
- They’re remarkably accurate
- They’re easy to tune for resources vs. accuracy

Fig 4.1: Mobilenet uses

Many mobile deep learning tasks are actually performed in the cloud. When you want to classify an image, that image is sent to a web service, it’s classified on a remote server, and the result is sent back to your phone.

The computational power on your phone is increasing rapidly, and the network complexity required for computer vision is shrinking

MobileNets are a class of convolutional neural network designed by researchers at Google. They are coined “mobile-first” in that they’re architected from the ground up to be resource-friendly and run quickly, right on your phone.

The main difference between the MobileNet architecture and a “traditional” CNN’s is instead of a single 3x3 convolution layer followed by batch norm and ReLU, MobileNets split the convolution into a 3x3 depthwise conv and a 1x1 pointwise conv. MobileNets are not usually as accurate as the bigger, more resource-intensive networks we’ve come to know. But finding that resource/accuracy trade-off is where MobileNets really shine.

MobileNets surface two parameters that we can tune to fit the resource/accuracy trade-off of our exact problem: width multiplier and resolution multiplier. The width multiplier allows us to thin the network, while the resolution multiplier changes the input dimensions of the image, reducing the internal representation at every layer.

4.1.2 Proposed Algorithm

Depth wise Separable Convolutional Neural Networks

Convolution is a very important mathematical operation in artificial neural networks(ANN's). Convolutional neural networks (CNN's) can be used to learn features as well as classify data with the help of image frames. There are many types of CNN's. One class of CNN's are depth wise separable convolutional neural networks.

These type of CNN's are widely used because of the following two reasons –

- They have lesser number of parameters to adjust as compared to the standard CNN's, which reduces overfitting
- They are computationally cheaper because of fewer computations which makes them suitable for mobile vision applications

Understanding Normal Convolution operation

Suppose there is an input data of size Df x Df x M, where Df x Df can be the image size and M is the number of channels (3 for an RGB image). Suppose there are N filters/kernels of size Dk x Dk x M. If a normal convolution operation is done, then, the output size will be Dp x Dp x N.

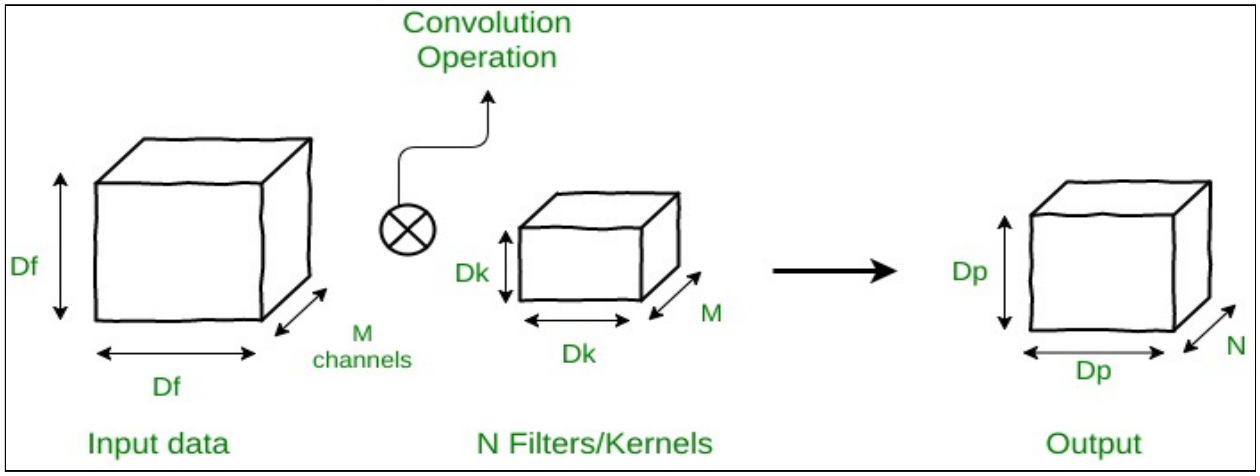


Fig 4.2 Standard convolution

The number of multiplications in 1 convolution operation = size of filter = Dk x Dk x M

Since there are N filters and each filter slides vertically and horizontally Dp times,

the total number of multiplications become N x Dp x Dp x (Multiplications per convolution)

So for normal convolution operation

Total no of multiplications = N x Dp2 x Dg2 x M

Depth-Wise Separable Convolutions

Now look at depth-wise separable convolutions. This process is broken down into 2 operations –

1. Depth-wise convolutions
2. Point-wise convolutions

Depthwise Convolutions

In depth-wise operation, convolution is applied to a single channel at a time unlike standard CNN's in which it is done for all the M channels. So here the filters/kernels will be of size Dk x Dk x 1. Given there are M channels in the input data, then M such filters are required.Output will be of size Dp x Dp x M

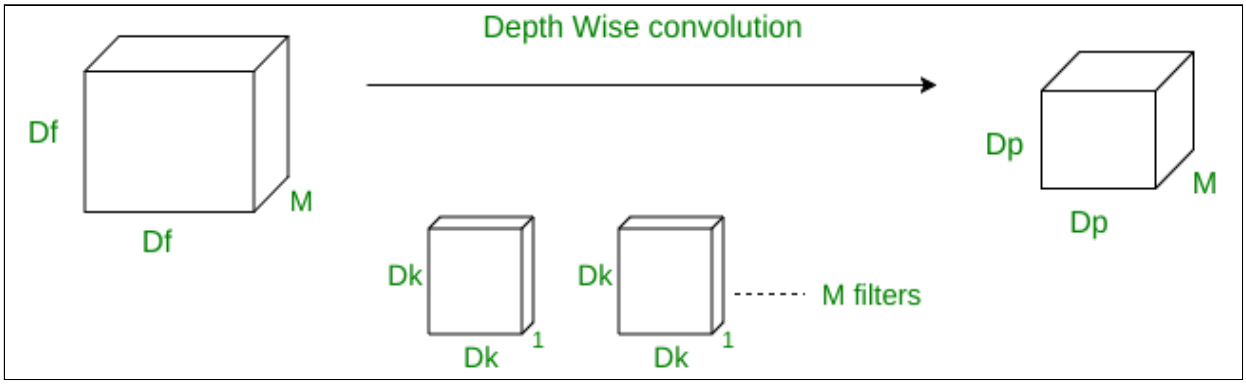


Fig 4.3 : Depthwise Convolutions

Cost of this operation:

A single convolution operation require Dk x Dk multiplications.

Since the filter are slided by Dp x Dp times across all the M channels,

the total number of multiplications is equal to M x Dp x Dp x Dk x Dk

So for depth wise convolution operation

Total no of multiplications = M x Dk2 x Dp2

Point wise Convolution

In point-wise operation, a 1×1 convolution operation is applied on the M channels. So the filter size for this operation will be 1 x 1 x M. Say we use N such filters, the output size becomes Dp x Dp x N.

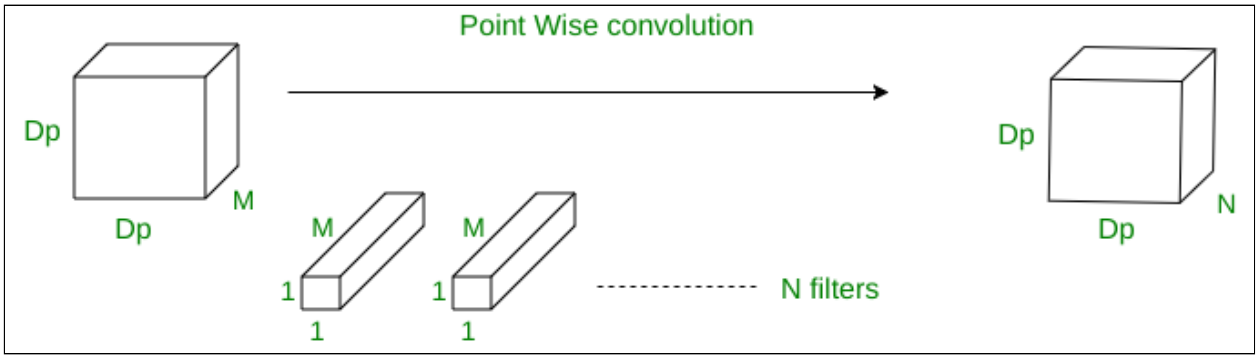


Fig 4.4 : Pointwise Convolution

Cost of this operation:

A single convolution operation require 1 x M multiplications.

Since the filter is being slided by Dp x Dp times,

the total number of multiplications is equal to M x Dp x Dp x (no. of filters)

So for point wise convolution operation

Total no of multiplications = M x Dp2 x N

Therefore, for overall operation:

Total multiplications = Depth wise conv. multiplications + Point wise conv. multiplications

Total multiplications = M * Dk2 * Dp2 + M * Dp2 * N = M * Dp2 * (Dk2 + n)

So for depth wise separable convolution operation

Total no of multiplications = M x Dp2 x (Dk2 + N)

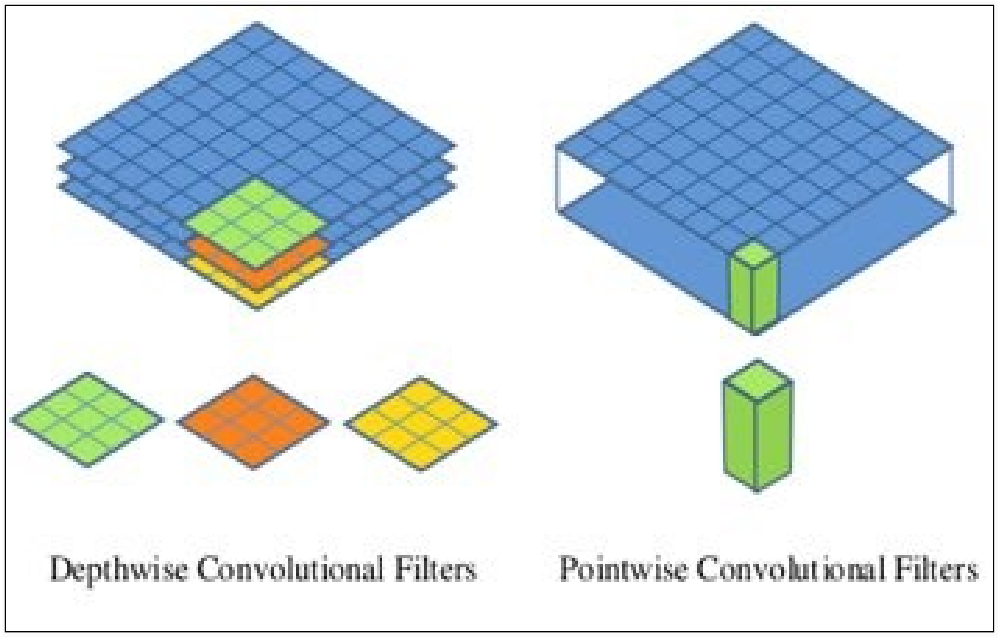


Fig 4.5 : Depth Separable Convolutions

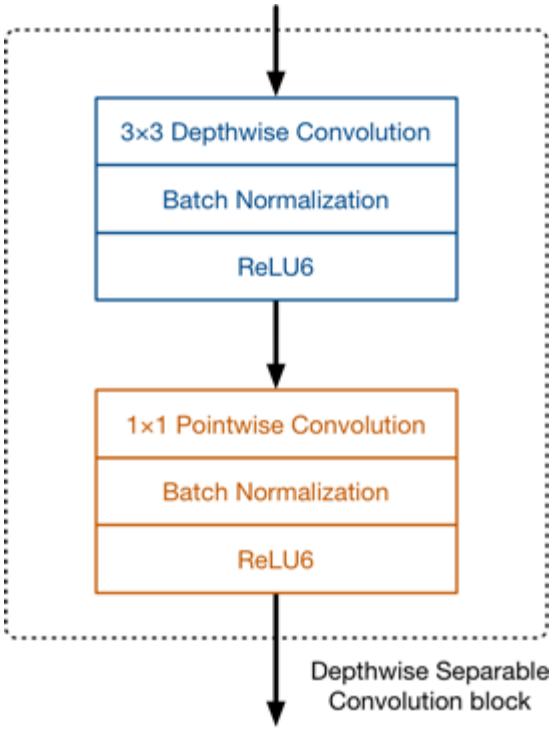


Fig 4.6 : Layers in Depth Separable Convolutions

Together, the depthwise and pointwise convolutions form a "depthwise separable" convolution block. It does approximately the same thing as traditional convolution but is much faster.

The full architecture of MobileNet V1 consists of a regular 3×3 convolution as the very first layer, followed by 13 times the above building block.

There are no pooling layers in between these depthwise separable blocks. Instead, some of the depthwise layers have a stride of 2 to reduce the spatial dimensions of the data. When that happens, the corresponding pointwise layer also doubles the number of output channels. If the input image is 224×224×3 then the output of the network is a 7×7×1024 feature map.

As is common in modern architectures, the convolution layers are followed by batch normalization. The activation function used by MobileNet is **ReLU6**. This is like the well-known ReLU but it prevents activations from becoming too big:

y = min(max(0, x), 6)

ReLU6 is more robust than regular ReLU when using low-precision computation. It also makes the shape of the function look more like a sigmoid:

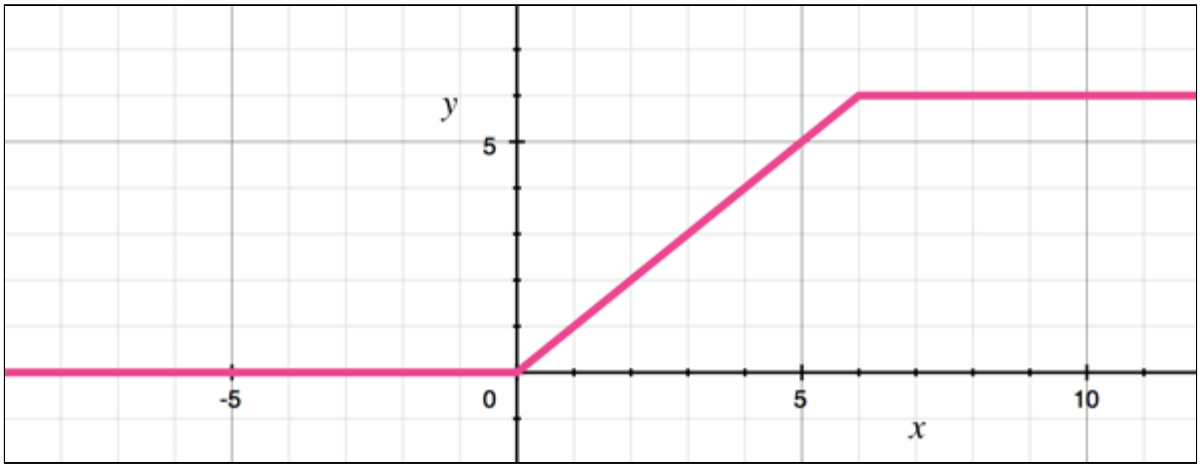


Fig 4.7 : ReLU Curve

In a classifier based on MobileNet, there is typically a global average pooling layer at the very end, followed by a fully-connected classification layer or an equivalent 1×1 convolution, and a softmax.

There is actually more than one MobileNet. It was designed to be a family of neural network architectures. There are several hyperparameters that let you play with different architecture trade-offs.

The most important of these hyperparameters is the depth multiplier, confusingly also known as the “width multiplier”. This changes how many channels are in each layer. Using a depth multiplier of 0.5 will halve the number of channels used in each layer, which cuts down the number of computations by a factor of 4 and the number of learnable parameters by a factor 3. It is therefore much faster than the full model but also less accurate.

hyper parameters

MobileNet has two hyper parameters to adapt the architecture to needs, α and ρ. α defines the number of input and output channels, while ρ controls the image size.

α is defined explicitly between 0 and 1. 1 corresponds to the default number of channels in the convolutions. Other sensible choices are 0.75 and 0.5. Reducing the number of channels reduces the number of weights as well as the computational costs.

ρ is set implicitly by changing the size of the input images. Since the number of weights for the convolutional layers depend on the number of channels and filters, this doesn’t reduce the number of weights, but has a huge influence on the number of computations.

4.1.3 Diagrammatic Representation

Use case Diagram

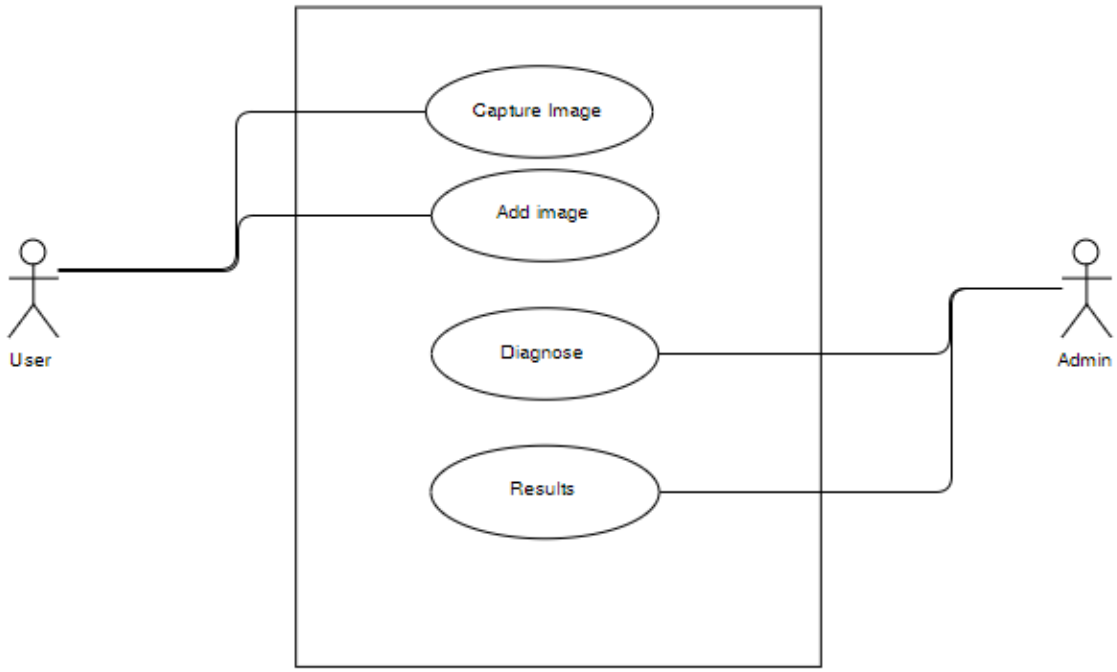


Fig 4.8: Use Case Diagram for artificial agricultural arborist application

1. Initially user captures the image of a leaf.
2. Adds the image to system through mobile application .
3. Arborist diagnoses the received image.
4. Accurate results will be predicted.

Sequence Diagram

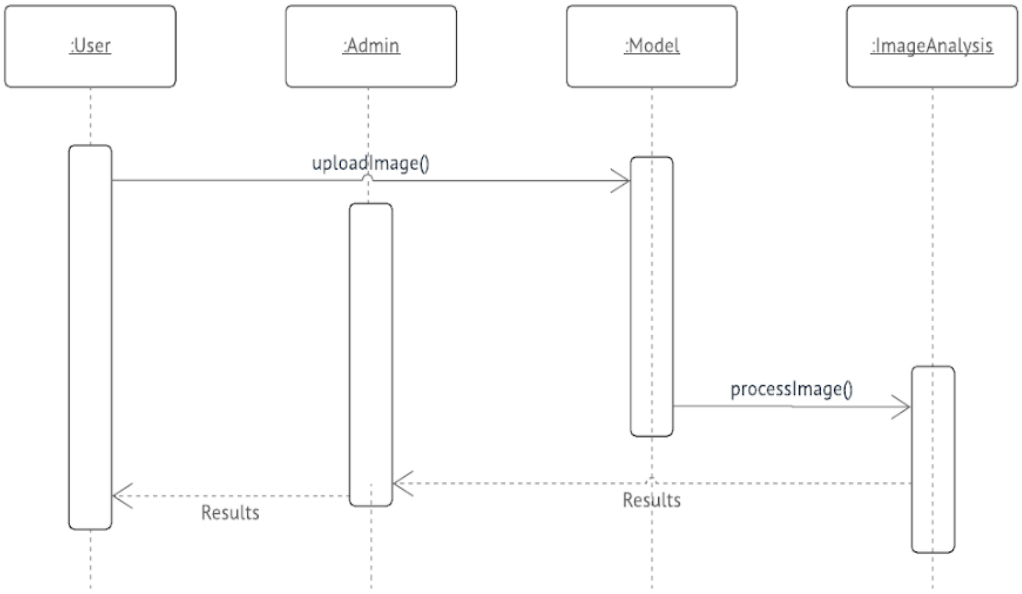


Fig 4.9 Sequence Diagram of artificial agricultural arborist application

1. User captures the image and Application sends it to model .
2. model sends image to image analysis for processing.
3. Image analysis results will be sent to application.
4. Application shows results to user.

Activity Diagram

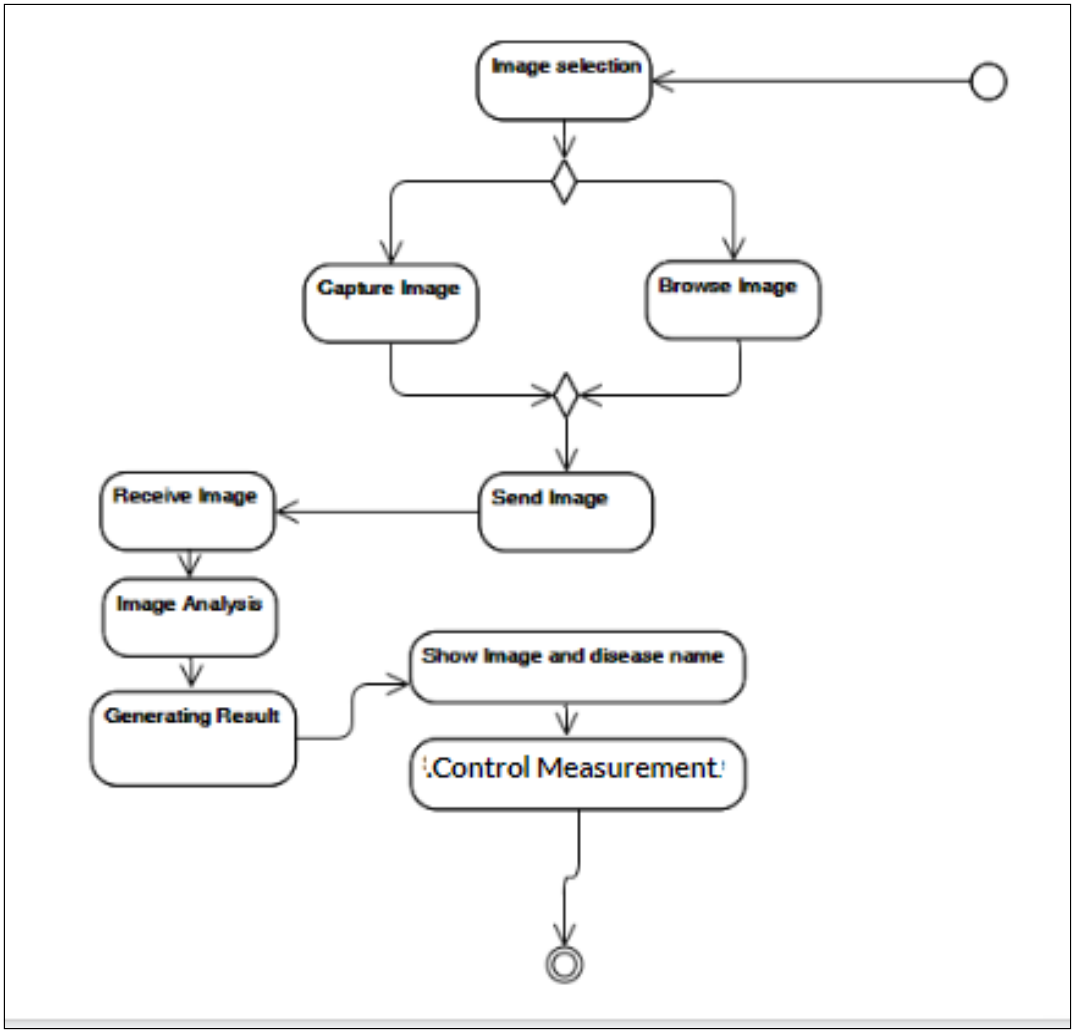


Fig 4.10 Activity Diagram of artificial agricultural arborist application

1. Image can be uploaded in two ways by browsing and by capturing
2. Send image for analysis. Receive image in other end
3. Image analysis will generate results
4. Generated result consists of image with disease name and control measurement of disease.

4.2 Implementation of Proposed Solution

Classify The Problem

By understanding current situations in Agriculture we found that there is no proper and easy solution for former to help them in identifying plant disease.

In our project we classified our project in 4 ways

1. finding plant species
2. healthy or unhealthy condition of leaf , if there is any disease on plant finding disease name.
3. Providing details of treatment .
4. For easy usability Android application development.

Acquire Data

In this project we collected leaf images from plant-village dataset where 54306 images of 38 plant species.

4.11 Example of leaf images from the PlantVillage dataset, representing every crop-disease pair used.

(1) Apple Scab, Venturia inaequalis

(2) Apple Black Rot, Botryosphaeria obtusa

(3) Apple Cedar Rust, Gymnosporangium juniperi-virginianae

(4) Apple healthy

(5)Blueberry healthy

(6) Cherry healthy

(7) Cherry Powdery Mildew, Podoshaera clandestine

(8) Corn Gray Leaf Spot, Cercospora zeae-maydis

(9) Corn Common Rust, Puccinia sorghi

(10) Corn healthy

(11) Corn Northern Leaf Blight, Exserohilum turcicum

(12) Grape Black Rot, Guignardia bidwellii,

(13) Grape Black Measles (Esca), Phaeomoniella aleophilum, Phaeomoniella chlamydospora

(14)Grape Healthy

(15) Grape Leaf Blight, Pseudocercospora vitis

(16) Orange Huanglongbing (Citrus Greening), Candidatus Liberibacter spp.

(17) Peach Bacterial Spot, Xanthomonas campestris

(18) Peach healthy

(19) Bell Pepper Bacterial Spot, Xanthomonas campestris

(20) Bell Pepper healthy

(21) Potato Early Blight, Alternaria solani

(22) Potato healthy

(23) Potato Late Blight, Phytophthora infestans

(24) Raspberry healthy

(25) Soybean healthy

(26) Squash Powdery Mildew, Erysiphe cichoracearum

(27) Strawberry Healthy

(28) Strawberry Leaf Scorch, Diplocarpon earlianum

(29) Tomato Bacterial Spot, Xanthomonas campestris pv. vesicatoria

(30) Tomato Early Blight, Alternaria solani

(31) Tomato Late Blight, Phytophthora infestans

(32) Tomato Leaf Mold, Passalora fulva

(33) Tomato Septoria Leaf Spot, Septoria lycopersici

(34)Tomato Two Spotted Spider Mite, Tetranychus urticae

(35) Tomato Target Spot, Corynespora cassiicola

(36) Tomato Mosaic Virus

(37) Tomato Yellow Leaf Curl Virus

(38) Tomato healthy.

Classify Data

In this step we created 38 individual folders for 38 plant species.

We're detecting plant disease with mobile app that use computer vision. We want to be able to do this fast and at as little computational cost to the user as possible.

Since we're tackling a custom problem, we need to start with creating our dataset. Our target is to collect 54,306 images.We'll get data from plant village.We'll place each image into one of folders, each representing the class of that image.

Labeled with 38 different labels. every image of our dataset through the network and keep track of the images it classified incorrectly or with little confidence. Then we go through each of those images and move them to their proper classes, if applicable. This reduces the number of images we have to manually clean up significantly. And doing multiple passes of this technique helped us increase our accuracy by seven percentage points on Inception.

data/

healthy/[images...]

unhealthy/[images...]

we'll use TensorFlow and transfer learning to fine-tune MobileNets on our custom dataset.

We analyze 54,306 images of plant leaves, which have a spread of 38 class labels assigned to them. Each class label is a crop-disease pair, and we make an attempt to predict the crop-disease pair given just the image of the plant leaf. one example each from every

crop-disease pair from the PlantVillage dataset. we resize the images to 256 × 256 pixels, and we perform both the model optimization and predictions on these downscaled images.

Across all our experiments, we use the whole PlantVillage dataset. We start with the PlantVillage dataset as it is, in color.Segmentation was automated by the means of a script tuned to perform well on our particular dataset. We chose a technique based on a set of masks generated by analysis of the color, lightness and saturation components of different parts of the images in several color spaces. One of the steps of that processing also allowed us to easily fix color casts, which happened to be very strong in some of the subsets of the dataset, thus removing another potential bias.

This set of experiments was designed to understand if the neural network actually learns the "notion" of plant diseases, or if it is just learning the inherent biases in the dataset.

Model the problem

You should aim to choose the optimal model for your application based on performance, accuracy and model size. There are trade-offs between each of them.

Performance

We measure performance in terms of the amount of time it takes for a model to run inference on a given piece of hardware. The less time, the faster the model.

The performance you require depends on your application. Performance can be important for applications like real-time video, where it may be important to analyze each frame in the time before the next frame is drawn (e.g. inference must be faster than 33ms to perform real-time inference on a 30fps video stream).

Our quantized Mobilenet models' performance ranges from 3.7ms to 80.3 ms.

Accuracy

We measure accuracy in terms of how often the model correctly classifies an image. For example, a model with a stated accuracy of 60% can be expected to classify an image correctly an average of 60% of the time.

Top-1 refers to how often the correct label appears as the label with the highest probability in the model's output. Top-5 refers to how often the correct label appears in the top 5 highest probabilities in the model's output.

Our quantized Mobilenet models' Top-5 accuracy ranges from 67.4 to 89.9%.

Size

The size of a model on-disk varies with its performance and accuracy. Size may be important for mobile development (where it might impact app download sizes) or when working with hardware (where available storage might be limited).

Our quantized Mobilenet models' size ranges from 0.5 to 3.4 Mb.

Architecture

There are several different architectures of models available on List of hosted models, indicated by the model's name. For example, you can choose between Mobilenet, Inception, and others.

The architecture of a model impacts its performance, accuracy, and size. All of our hosted models are trained on the same data, meaning you can use the provided statistics to compare them and choose which is optimal for your application.

TensorFlow comes packaged with great tools that you can use to retrain MobileNets without having to actually write any code.

MobileNets is a family of mobile-first computer vision models for TensorFlow designed to effectively maximize accuracy, while taking into consideration the restricted resources for on-device or embedded applications. MobileNets are small, low-latency, low-power models

parameterized to meet the resource constraints for a variety of uses. They can be used for classification, detection, embeddings, and segmentation—similar to other popular large scale models, such as Inception. The MobileNet architecture uses only depthwise separable convolutions except for the first layer that uses a full convolution. Like SqueezeNet the output of the last convolutional layer is put into a global average pooling layer. But the output of the pooling layer isn't used directly for classification and is followed by a final fully connected layer. However, since global max pooling is applied first, the final fully connected layer is much smaller compared to classical architectures, where the output of the convolutional layer is used directly in a fully connected layer.

Now you can use the scripts to retrain MobileNet on your own data.

Let's retrain a small assortment and see how they perform. To kick off training, we'll run the following command from the root of the TensorFlow.

```
python tensorflow/examples/image_retraining/retrain.py \

--image_dir ~/ml/data/ \

--learning_rate=0.0001 \

--testing_percentage=20 \

--validation_percentage=20 \

--train_batch_size=32 \

--validation_batch_size=-1 \

--flip_left_right True \

--random_scale=30 \

--random_brightness=30 \

--eval_step_interval=100 \
```

--how_many_training_steps=600 \

--architecture mobilenet_1.0_224

The architecture flag is where we tell the retraining script which version of MobileNet we want to use. The 1.0 corresponds to the width multiplier, and can be 1.0, 0.75, 0.50 or 0.25. The 224 corresponds to image resolution, and can be 224, 192, 160 or 128. For example, to train the smallest version, you'd use --architecture mobilenet_0.25_128.

Some other important parameters:

- learning_rate: we found 0.0001 to work well.
- testing and validation percentage: The script will split your data into train/val/test for you. It will use train to train, val to give performance updates every "eval_step_interval", and test will run after "how_many_training_steps" to give you your final score.
- validation_batch_size: Setting this to -1 tells the script to use all your data to validate on. When you don't have a lot of data, it's a good idea to use -1 here to reduce variance between evaluation steps.

```
INFO:tensorflow:global step 29790: loss = 0.3194 (0.236 sec/step)
INFO:tensorflow:global step 29800: loss = 0.1820 (0.175 sec/step)
INFO:tensorflow:global step 29810: loss = 0.1972 (0.230 sec/step)
INFO:tensorflow:global step 29820: loss = 0.2426 (0.232 sec/step)
INFO:tensorflow:global step 29830: loss = 0.2625 (0.241 sec/step)
INFO:tensorflow:global step 29840: loss = 0.1558 (0.188 sec/step)
INFO:tensorflow:global step 29850: loss = 0.1601 (0.230 sec/step)
INFO:tensorflow:global step 29860: loss = 0.2257 (0.245 sec/step)
INFO:tensorflow:global step 29870: loss = 0.3663 (0.269 sec/step)
INFO:tensorflow:global step 29880: loss = 0.1686 (0.198 sec/step)
INFO:tensorflow:global step 29890: loss = 0.3222 (0.216 sec/step)
INFO:tensorflow:global step 29900: loss = 0.2520 (0.217 sec/step)
INFO:tensorflow:global step 29910: loss = 0.3735 (0.243 sec/step)
INFO:tensorflow:global step 29920: loss = 0.2633 (0.204 sec/step)
INFO:tensorflow:global step 29930: loss = 0.2714 (0.185 sec/step)
INFO:tensorflow:global step 29940: loss = 0.3153 (0.194 sec/step)
INFO:tensorflow:global step 29950: loss = 0.1891 (0.215 sec/step)
INFO:tensorflow:global step 29960: loss = 0.2570 (0.197 sec/step)
INFO:tensorflow:global step 29970: loss = 0.1911 (0.203 sec/step)
INFO:tensorflow:global step 29980: loss = 0.1798 (0.222 sec/step)
INFO:tensorflow:global step 29990: loss = 0.1881 (0.218 sec/step)
INFO:tensorflow:global step 30000: loss = 0.1761 (0.226 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
```

Fig 4.12 : Training

Validation & Execution

Our model is trained with 80% train data, in this step we test our model with 20% Test data. We categorized healthy and unhealthy leaves from 20% Test data. Where true positive value is 6860 ,false negative is 1500, false positive is 1000 and true negative is 1500.

After replacing these values in precision,recall formulae we got 67% accuracy.Like this we worked on three models by making several changes in its parameters. initially we got accuracy of 50% and second time we got 59% accuracy and for third model we got 67% accuracy .each model training process took around 8-10 hours.

Positive (P) : Observation is positive (for example: is an apple).

Negative (N) : Observation is not positive (for example: is not an apple).

True Positive (TP) : Observation is positive, and is predicted to be positive.

False Negative (FN) : Observation is positive, but is predicted negative.

True Negative (TN) : Observation is negative, and is predicted to be negative.

False Positive (FP) : Observation is negative, but is predicted positive.

Classification Rate/Accuracy:

Classification Rate or Accuracy is given by the relation:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor or terrible depending upon the problem.

Recall:

Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized (small number of FN).

Recall is given by the relation:

$$Recall = \frac{TP}{TP + FN}$$

Precision:

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive (small number of FP).

Precision is given by the relation:

$$Precision = \frac{TP}{TP + FP}$$

High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.

Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP)

F-measure:

Since we have two measures (Precision and Recall) it helps to have a measurement that represents both of them. We calculate an F-measure which uses Harmonic Mean in place of Arithmetic Mean as it punishes the extreme values more.

The F-Measure will always be nearer to the smaller value of Precision or Recall.

$$F\text{ - measure} = \frac{2*\text{Recall}*\text{Precision}}{\text{Recall} + \text{Precision}}$$

```
INFO:tensorflow:Evaluation [5/19]
INFO:tensorflow:Evaluation [6/19]
INFO:tensorflow:Evaluation [7/19]
INFO:tensorflow:Evaluation [8/19]
INFO:tensorflow:Evaluation [9/19]
INFO:tensorflow:Evaluation [10/19]
INFO:tensorflow:Evaluation [11/19]
INFO:tensorflow:Evaluation [12/19]
INFO:tensorflow:Evaluation [13/19]
INFO:tensorflow:Evaluation [14/19]
INFO:tensorflow:Evaluation [15/19]
INFO:tensorflow:Evaluation [16/19]
INFO:tensorflow:Evaluation [17/19]
INFO:tensorflow:Evaluation [18/19]
INFO:tensorflow:Evaluation [19/19]
eval/Accuracy[0.6757894754]eval/Recall_5[0.78947389]
```

Fig 4.13: evaluation

Table 4.1 : evaluation metrics

Model	Recall	Precision	F1 Score	Accuracy
A	0.57	0.41	0.47	50%
B	0.59	0.53	0.55	59%
C	0.78	0.74	0.7	67%

Deployment

MobileNets are made for mobile devices. So, let's move model to an Android app Goals and Plan.Let's set some constraints so we have something specific to shoot for. We'll attempt to Retrain a MobileNet on a very small amount of purpose-built data.Achieve 67% classification accuracy on a hold out test set.Generate a new training dataset

Train several MobileNet configurations to find the smallest net that will hit our accuracy target. Update the TensorFlow Android example app to use our MobileNet model.Training MobileNet on our custom dataset

We start training with MoileNet 1.0 @ 128. And because we're going to put this on a mobile device, we'll use quantized weights, which will reduce the model memory footprint even further.

From the root TensorFlow folder, we'll run:

```
python tensorflow/examples/image_retraining/retrain.py \

  --image_dir ~/ml/data/ \

  --learning_rate=0.0005 \

  --testing_percentage=15 \

  --validation_percentage=15 \

  --train_batch_size=32 \

  --validation_batch_size=-1 \

  --flip_left_right True \

  --random_scale=30 \

  --random_brightness=30 \

  --eval_step_interval=100 \
```

```
  --how_many_training_steps=1000 \

  --architecture mobilenet_1.0_128_quantized
```

After 1,000 training steps, we achieve 67.2% accuracy on our hold out set. Apparently MobileNets are pretty good at classifying

Let’s give it a quick try to make sure it’s working as expected:

```
python tensorflow/examples/label_image/label_image.py \

  --graph=/tmp/output_graph.pb \

  --labels=/tmp/output_labels.txt \

  --image=/home/harvitronix/ml/test-image.jpg \

  --input_layer=input \

  --output_layer=final_result \

  --input_mean=128 \

  --input_std=128 \

  --input_width=128 \

  --input_height=128
```

Apple healthy leaf: 0.89023 confidence. Looks good!

Using our MobileNet model in an Android app

Now that we have a model that’s tiny, fast and accurate enough for our use case, let’s load it up in an Android app so we can test it in the real world. It contains both the .pb and label files. Sticking with our theme of using tools provided by TensorFlow .

Getting and building the project

If you haven’t already, go ahead and clone the TensorFlow repo:

Switching to MobileNet.Now let’s make a couple minor changes to the Android project to use our custom MobileNet model.

First, copy your model and labels into the project’s assets folder. Mine were at /tmp/output_graph.pb and /tmp/output_labels.txt.

Next, open up ClassifierActivity, which can be found in:

You’ll want to update the constants at the top of the file to define the settings for our new model. It looks like this when you first open it:

```
private static final int INPUT_SIZE = 224;

private static final int IMAGE_MEAN = 117;

private static final float IMAGE_STD = 1;

private static final String INPUT_NAME = "input";

private static final String OUTPUT_NAME = "output";

private static final String MODEL_FILE = "file:///android_asset/tensorflow_graph.pb";

private static final String LABEL_FILE =

    "file:///android_asset/imagenet_comp_graph_label_strings.txt";
```

Change it to:

```
private static final int INPUT_SIZE = 224;

private static final int IMAGE_MEAN = 117;

private static final float IMAGE_STD =1;
```

private static final String INPUT_NAME = "input";

private static final String OUTPUT_NAME = "final_result";

private static final String MODEL_FILE = "file:///android_asset/**`retrained_graph.pb`**";

private static final String LABEL_FILE =

"file:///android_asset/**`retrained_labels`**.txt";

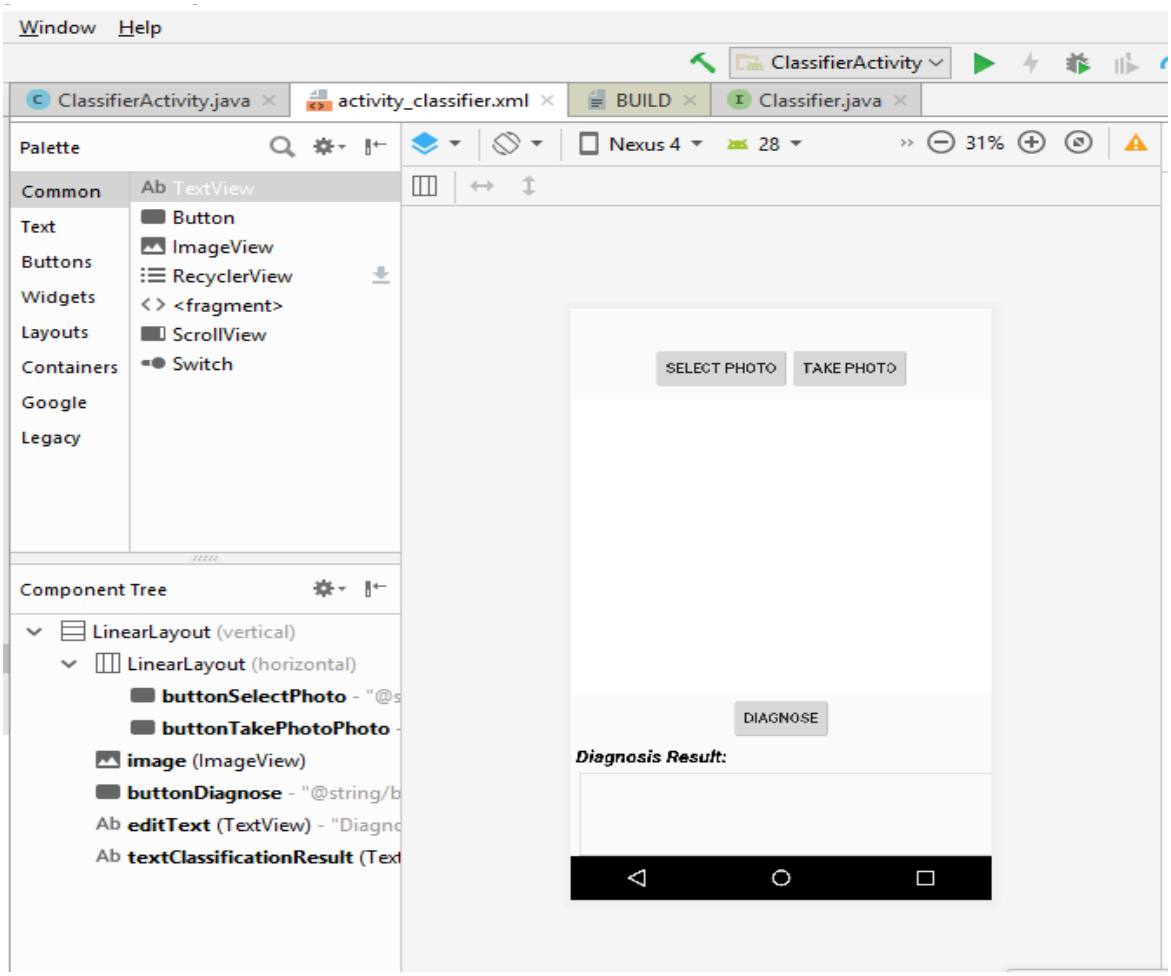


Fig 4.14 : Application Layout Design

Hit run to build the project and load the APK on your device,

4.3 Sample Code

ClassifierActivity.java:

```
package org.tensorflow.demo;
import android.Manifest;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.provider.MediaStore;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

import java.io.File;
import java.io.IOException;
import java.util.List;

public class ClassifierActivity extends Activity {



private static final int INPUT_SIZE = 224;
private static final int IMAGE_MEAN = 117;
private static final float IMAGE_STD = 1;
private static final String INPUT_NAME = "input";
//private static final String OUTPUT_NAME = "output";
private static final String OUTPUT_NAME = "final_result";
```

```
    //  private static final String MODEL_FILE =
"file:///android_asset/tensorflow_inception_graph.pb";
    //  private static final String LABEL_FILE =
    //      "file:///android_asset/imagenet_comp_graph_label_strings.txt";
  private static final String MODEL_FILE=
  "file://android_asset/retrained_graph.pb";
  private static final String LABEL_FILE =
          "file:///android_asset/retrained_labels.txt";


  private Classifier classifier;


  private Button mButtonSelectPhoto;
  private Button mButtonTakePhoto;
  private Button mButtonDiagnose;
  private ImageView mImageView;
  private TextView mTextView;
  private File temFile;
  private Bitmap mDiagnosisBitmap;

  private static final int PHOTO_REQUEST_CAMERA = 1;
  private static final int PHOTO_REQUEST_GALLERY = 2;
  private static final int PHOTO_REQUEST_CUT = 3;
  private static final int PHOTO_REQUEST_CODE = 4;
  private static final int PERMISSION_REQUEST_CODE = 5;
  private static final String PHOTO_FILE_NAME = "temp_photo.jpg";

    @Override
  protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_classifier);

  final AssetManager assetManager = getAssets();
  classifier =
  TensorFlowImageClassifier.create(assetManager,MODEL_FILE,LABEL_FILE,INPUT_SIZE,
    IMAGE_MEAN,IMAGE_STD,INPUT_NAME,OUTPUT_NAME);
```

```
mButtonSelectPhoto = (Button) findViewById(R.id.buttonSelectPhoto);
mButtonTakePhoto = (Button) findViewById(R.id.buttonTakePhotoPhoto);
mButtonDiagnose = (Button) findViewById(R.id.buttonDiagnose);
mImageView = (ImageView) findViewById(R.id.image);
mTextView = (TextView) findViewById(R.id.textClassificationResult);

mButtonTakePhoto.setOnClickListener(new View.OnClickListener() {
 @Override
public void onClick(View view) {
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
if (checkSelfPermission(Manifest.permission.CAMERA)!=
PackageManager.PERMISSION_GRANTED){
         requestPermissions(new String[]{Manifest.permission.CAMERA},

  PHOTO_REQUEST_CODE);
         }
         else
           camera(view);
       }
     }
   });
mButtonSelectPhoto.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View view) {
       gallery(view);
     }
   });
 mButtonDiagnose.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View view) {
final List<Classifier.Recognition> results =
classifier.recognizeImage(mDiagnosisBitmap);
       System.out.println(results.size());
       for (final Classifier.Recognition result: results){
         System.out.println("Result:"+result.getTitle()+"
"+result.getConfidence()+result.toString());
}
       if(results.get(0).getConfidence()>=0.7) {
```

```
  mTextView.setText(results.get(0).getTitle() + "  confidence:" +
  results.get(0).getConfidence());
       }
         else {
           mTextView.setText("* * * * * * * * *Oho..! unable to detect* * * * * *
* * *, Capture image correctly");
         }


         }

     });

  }

  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
     if (requestCode == PHOTO_REQUEST_GALLERY){
       if (data!=null){
         Uri uri = data.getData();
         Bitmap bitmap = null;
         try {
           bitmap =
  MediaStore.Images.Media.getBitmap(this.getContentResolver(),uri);
         } catch (IOException e) {
           e.printStackTrace();
         }
         System.out.println("Success!!!");
         mDiagnosisBitmap = scaleImage(bitmap);
         mImageView.setImageBitmap(mDiagnosisBitmap);


       }
     }else if (requestCode == PHOTO_REQUEST_CAMERA){
       if (hasSdcard()){
         Uri uri = Uri.fromFile(temFile);
         Bitmap bitmap = BitmapFactory.decodeFile(uri.getPath());
         mDiagnosisBitmap = scaleImage(bitmap);
         mImageView.setImageBitmap(mDiagnosisBitmap);
       }
```

```
}else if (requestCode == PHOTO_REQUEST_CUT){
  if(data!=null){
     Bitmap bitmap = data.getParcelableExtra("data");
    mImageView.setImageBitmap(bitmap);
   }try {
     temFile.delete();
   }catch (Exception e){
     e.printStackTrace();
   }
  }
 super.onActivityResult(requestCode, resultCode, data);
  }

 private void gallery(View view){
 Intent intent = new Intent(Intent.ACTION_PICK);
 intent.setType("image/*");
 startActivityForResult(intent, PHOTO_REQUEST_GALLERY);
 }

private void camera(View view){
Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
if (hasSdcard()){
  temFile = new
File(Environment.getExternalStorageDirectory(),PHOTO_FILE_NAME);
     Uri uri = Uri.fromFile(temFile);
     intent.putExtra(MediaStore.EXTRA_OUTPUT, uri);
   }
   startActivityForResult(intent, PHOTO_REQUEST_CAMERA);

}
private boolean hasSdcard(){
if (Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
     return true;
   else
    return false;
 }
```

```
public Bitmap scaleImage(Bitmap bitmap){
int orignalWidth = bitmap.getWidth();
int originalHeight = bitmap.getHeight();
float scaleWidth = ((float)INPUT_SIZE)/orignalWidth;
float scaleHeight = ((float)INPUT_SIZE)/originalHeight;
Matrix matrix = new Matrix();
matrix.postScale(scaleWidth,scaleHeight);
Bitmap scaledBitmap =
Bitmap.createBitmap(bitmap,0,0,orignalWidth,originalHeight,matrix,true);
return scaledBitmap;
 }
}
```

activity_classifier.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context="org.tensorflow.demo.ClassifierActivity">
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:gravity="center_horizontal"
    android:orientation="horizontal"
    android:padding="10dp">

    <Button
        android:id="@+id/buttonSelectPhoto"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="@string/buttonSelectPhoto" />
```

```
<Button
        android:id="@+id/buttonTakePhotoPhoto"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="@string/buttonTakePhoto" />
</LinearLayout>
<ImageView
    android:id="@+id/image"
    android:layout_width="match_parent"
    android:layout_height="321dp"
    android:layout_gravity="center"
    android:background="#FFFFFF"
    android:padding="1dp"
    android:visibility="visible" />


<Button
    android:id="@+id/buttonDiagnose"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="@string/buttonDiagnose" />

<TextView
    android:id="@+id/editText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:padding="5dp"
    android:text="Diagnosis Result:"
    android:textColor="@android:color/background_dark"
    android:textSize="18sp"
    android:textStyle="bold|italic" />
```

```
<TextView
```

```
    android:id="@+id/textClassificationResult"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:layout_marginLeft="5dp"
    android:background="@drawable/border"
    android:editable="false"
    android:paddingLeft="5dp"
    android:scrollbars="vertical"
    android:textColor="@android:color/black"
    android:textSize="18sp"
    android:textStyle="bold"
    android:visibility="visible" />
</LinearLayout>
```

5.RESULTS AND TESTING

5.1 System Testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects). Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test: meets the requirements that guided its design and development, responds correctly to all kinds of inputs, performs its functions within an acceptable time, is sufficiently usable, can be installed and run in its intended environments, and Achieves the general result its stakeholder‟s desire. As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). The job of testing is an iterative process as when one bug is fixed; it can illuminate other, deeper bugs, or can even create new ones. Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors. Software testing can be conducted as soon as executable software (even if partially complete) exists. The overall approach to software development often determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an Agile approach, requirements, programming, and testing are often done concurrently.

THE MAIN AIM OF TESTING

The main aim of testing is to analyze the performance and to evaluate the errors that occur when the program is executed with different input sources and running in different operating environments. In this project, we have developed a Android Application and a Image Processing code which helps in detection of disease. The main aim of testing this project is to check if the disease is getting detected accurately and check the working performance when different images are given as inputs.

The testing steps are:

- Unit Testing.
- Integration Testing.
- Validation Testing.
- User Acceptance Testing.
- Output Testing

UNIT TESTING:

Unit testing, also known as component testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors. Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. The following Unit Testing Table 7.1 shows the functions that were tested at the time of programming. The first column gives all the modules which were tested, and the second column gives the test results. Test results indicate if the functions, for given inputs are delivering valid outputs

INTEGRATION TESTING:

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed. Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

VALIDATION TESTING

At the culmination of integration testing, software is completed assembled as a package. Interfacing errors have been uncovered and corrected. Validation testing can be defined in many ways; here the testing validates the software function in a manner that is reasonably expected by the customer. In software project management, software testing, and software engineering, verification and validation (V&V) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. The following table indicates validation tests done for checking functionality of the project after its integration with the front-end

USER ACCEPTANCE TESTING

Performance of an acceptance test is actually the user‟s show. User motivation and knowledge are critical for the successful performance of the system. The above tests were conducted on the newly designed system performed to the expectations. All the above testing strategies were done using the following test case designs.

WHITE BOX TESTING

White Box Testing sometimes called glass-box testing is a test case design method that uses the control structure of the procedural design to derive test cases. Using White Box Testing, we can derive test cases that: Guarantee that all independent paths within a module have been exercised at least once. Exercise all logical decision on their true and false sides. Execute all loops at their boundaries and within their operational bounds

BLACK BOX TESTING

Black Box Testing focuses on the functional requirements of the software. This enables us to derive sets of input conditions that will fully exercise all functional requirements for a program. Black Box testing attempts to find errors in the following categories: Incorrect or Missing Functions. Interface errors. Performance errors. Initialization and Termination errors.

5.2 TEST CASES

A Test case is a set of input data and expected results that exercises a component with the purpose of causing failure and detecting faults. Test case is an explicit set of instructions designed to detect a particular class of defect in a software system, by bringing about a failure. A Test case can give rise to many tests.

Test cases can be divided into two types. First one is positive test cases and second one is negative test cases. In positive test cases are conducted by the developer intention is to get the output. In negative test cases are conducted by the developer intention is to don’t get the output.

Table 5.1: Test Cases

Test case ID	**Test case Name**	**Input**	**Expected Result**	**Actual Result**	**Remarks**
1	camera	Take photo	Image Captured	Image captured	Success
2	Select Photo	Select image	Image added Successfully	Image added Successfully	Success
3	Diagnose	Click on Diagnose	Diagnosis Results	Diagnosis Results	Success
4	Apple leaf photo	Apple leaf image	Whether healthy or diseased leaf	Healthy leafApple healthy leaf	Success
5	Tomato unhealthy leaf	Tomato leaf image	Tomato mosaic virus	Tomato mosaic virus	Success

Results

1.Leaf disease identification using camera

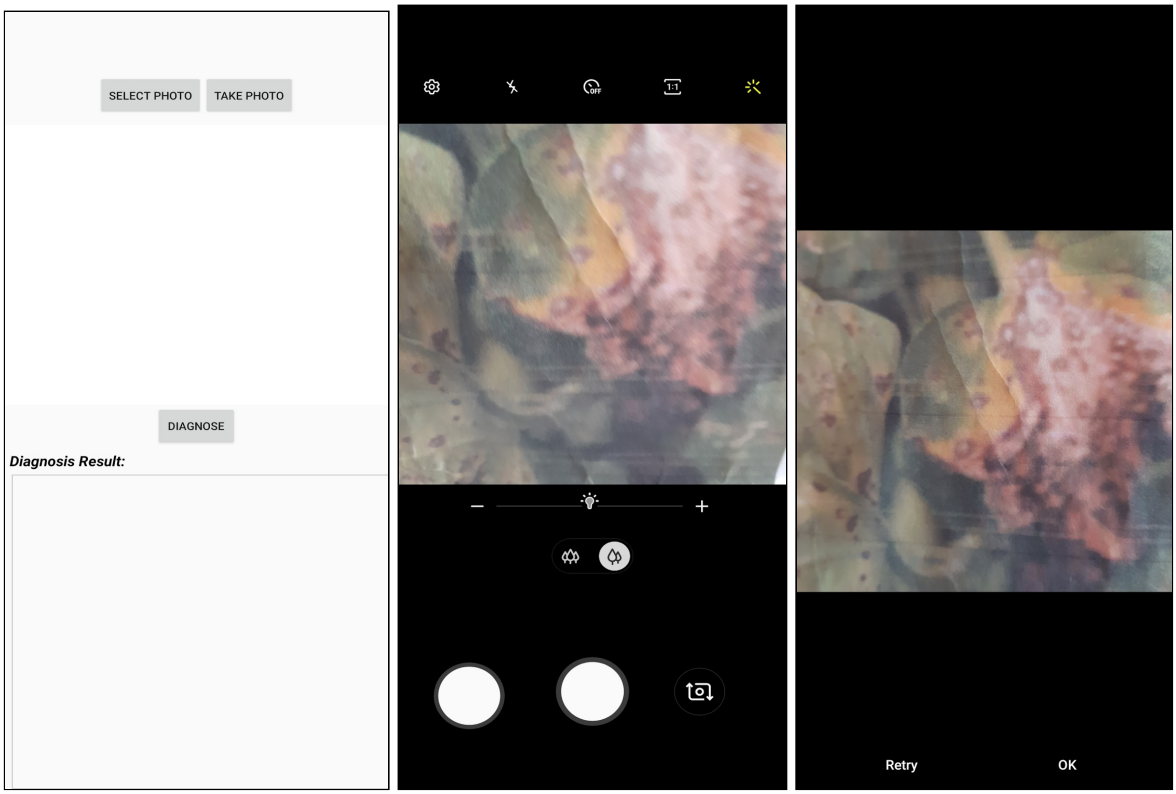
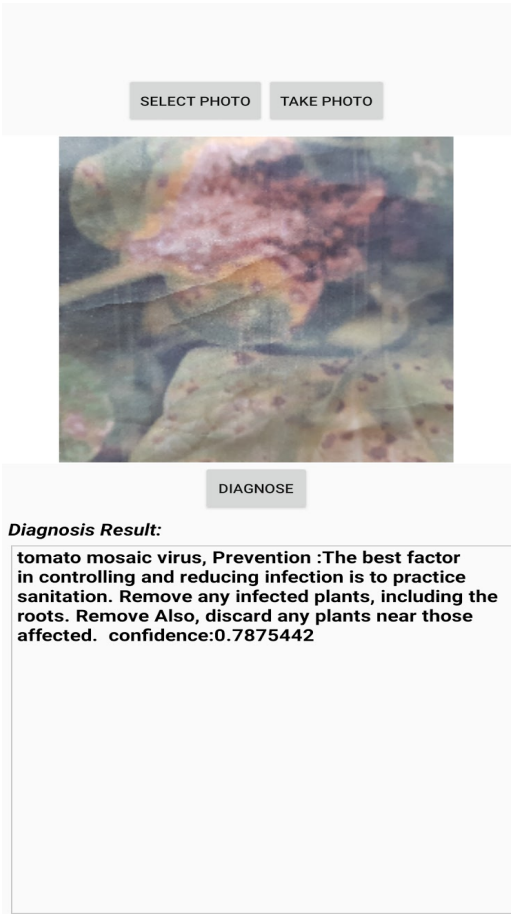


Fig 5.1 :(a) **(b)** **(c)**

(a) Application screen layout

(b) Capturing leaf image using camera

(c) Image captured



(d)

(d) Diagnosis Results of captured leaf image

2.Uploading image from gallery

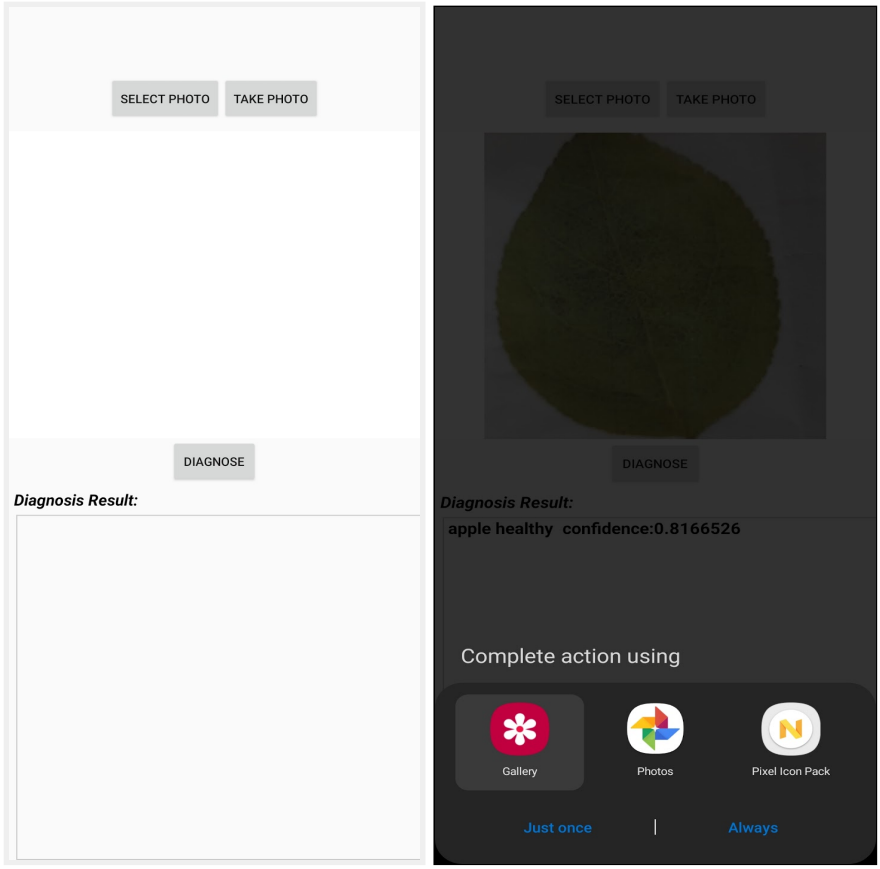
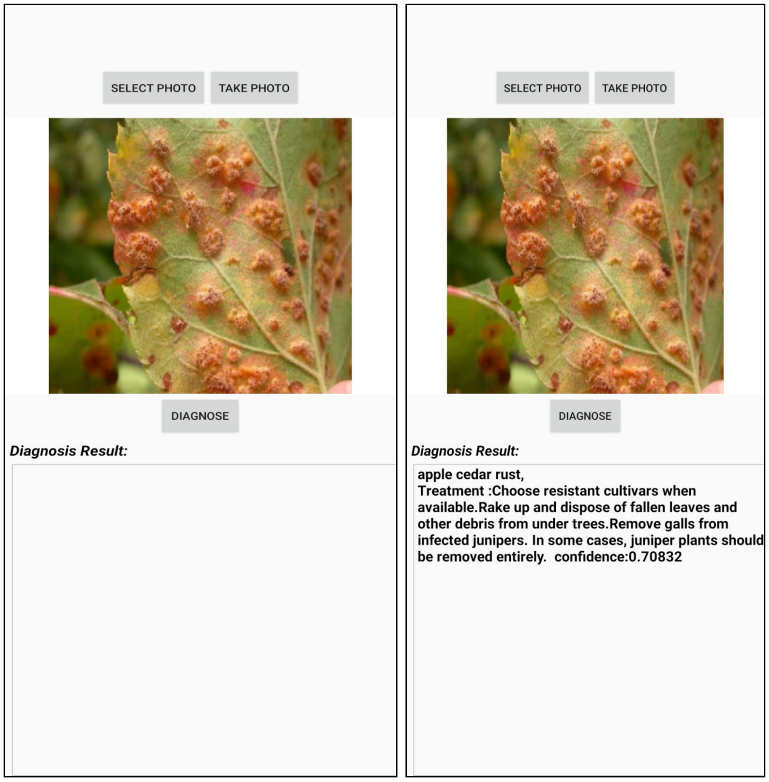


Fig 5.2: (a) **(b)**

(a) Application screen layout

(b) Uploading image from Device



(c) **(d)**

(c) Uploaded image

(d) Diagnosis results of uploaded image

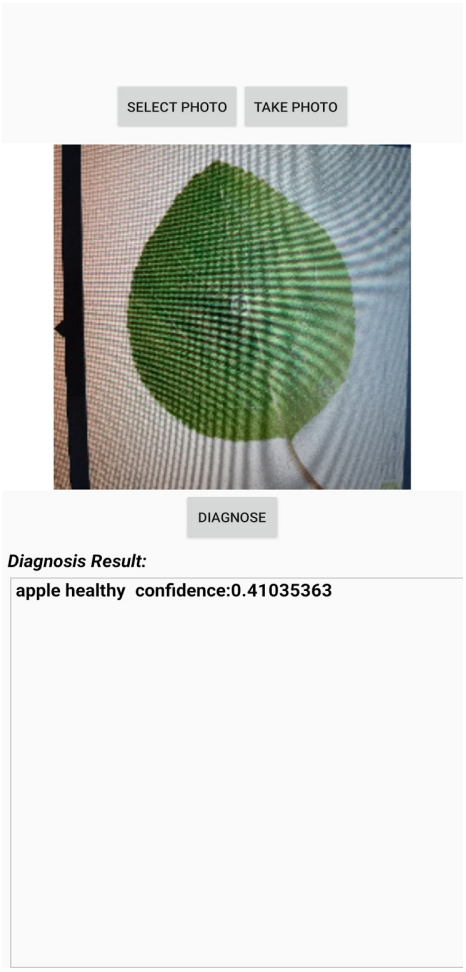


Fig 5.3 : Healthy leaf results

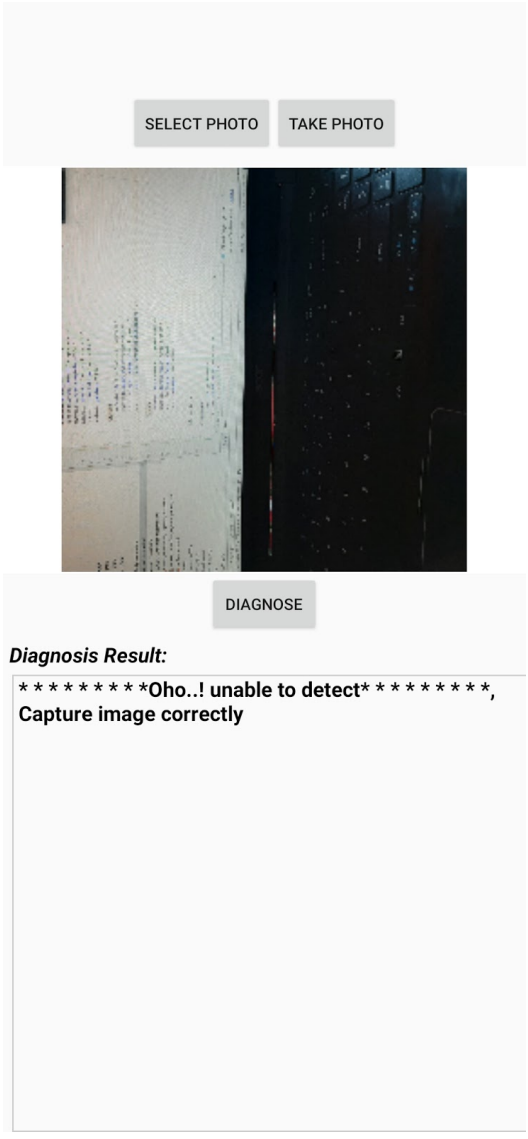


Fig 5.4: Invalid Image results

6.CONCLUSION AND FUTURE WORK

Conclusion

An application of detecting the plant diseases and providing the necessary suggestions for the disease has been implemented.The diseases specific to plants were considered for testing of the algorithm. The experimental results indicate the proposed approach can recognize the diseases with a little computational effort. By this method, the plant diseases can be identified at the initial stage itself .

In order to improve disease identification rate at various stages, the training samples can be increased with the optimal features given as input condition for disease identification.

Future work

Our project is focused on 38 species of plant leaves adding more number of plant species will help user to detect any kind of plant and know disease of it. In this project we only focused on detecting species of plant whether it is healthy or unhealthy. If unhealthy plant leaf detected then how to prevent it from spreading. we can also provide benefit of particular plant. it will help people in plant cultivation immensely. Due to less computing resources accuracy results are not upto mark but using few more additional computing resources will increase accuracy.

7. References

[1] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

[2] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M.Swetter, Helen M. Blau & Sebastian Thrun, Dermatologist-level classification of skin cancer with deep neural networks

[3] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

[4] Lin, M., Chen, Q., and Yan, S. (2013). Network in network. arXiv:1312.4400.