# PLANT LEAF DISEASE DETECTION MODEL

**A PROJECT REPORT**

*Submitted by*

**LAKSHMI VIBHA (RA2011042010039)**
**ELINA SINGH (RA2011042010041)**
**ESHITA MISHRA (RA2011042010062)**

***Under the Guidance of***

**Dr. M. Anand**

**(Assistant Professor, Department of Data Science and Business Systems)**


**DEPARTMENT OF DATA SCIENCE**

**AND**

**BUSINESS SYSTEMS**

**FACULTY OF DATA SCIENCE AND BUSINESS SYSTEMS**
**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**
**KATTANKULATHUR – 603203**


# SRM INSTITUTE OF SCIENCE AND

# TECHNOLOGY KATTANKULATHUR-603203

## BONAFIDE CERTIFICATE

Certified that this project report titled "***Plant Leaf Disease Detection Model***" is the bonafide work of "**Lakshmi Vibha [RA2011042010039], Elina Singh [RA2011042010041], Eshita Mishra [RA2011042010062].** Certified further, that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. M. Anand
**SUPERVISOR**
Assistant Professor
Department of Data Science
and Business Systems

Dr. M. Lakshmi
**HEAD OF DEPARTMENT**
Department of Data Science
and Business Systems

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR-603203

## OUR WORK DECLARATION

This sheet must be filled in (each box ticked to show the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

**Degree/ Course**: B.TECH, CSBS
**Student Name**: LAKSHMI VIBHA, ELINA SINGH, ESHITA MISHRA
**Registration Number:** RA2011042010039, RA2011042010041, RA2011042010062
**Title of Work:** PLANT LEAF DISEASE DETECTION MODEL

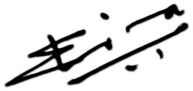
I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.
I / We confirm that all the work contained in this assessment is my / our own except where indicated and that I / We have met the following conditions:

- Clearly references/lists all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data, etc. that are not my own
- Not making any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow
- students, technicians, statisticians, and external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

| **DECLARATION:** |
| --- |
| I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above. |
| If you are working in a group, please write your registration numbers and sign with the date for every student in your group. |

**Lakshmi Vibha**
**RA2011042010039**



**Elina Singh**
**RA2011042010041**

**Eshita Mishra**
**RA2011042010062**

# ACKNOWLEDGEMENT

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice Chancellor (I/C), SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We sincerely thank **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her invaluable support.

We wish to thank **Dr. M. Lakshmi**, Professor & Head, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We are extremely grateful to our Academic Advisor **Dr. E. Sasikala**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for her great support at all the stages of project work.

We register our immeasurable thanks to our Faculty Advisor, **Dr. Paul T. Sheeba**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, **Dr. M. Anand**, Assistant Professor, Department of Data Science and Business Systems, SRM Institute of Science and Technology, for providing us an opportunity to pursue our project under his mentorship. He offered us the freedom and support to explore the research topics of our interest. His passion for solving real problems and making a difference in the world has always been inspiring.

We sincerely thank the staff and students of the Data Science and Business Systems, SRM Institute of Science and Technology, for their help during my research.

Finally, we would like to thank my parents, our family members, and our friends for their unconditional love, constant support, and encouragement.

**LAKSHMI VIBHA (RA2011042010039)**
**ELINA SINGH (RA2011042010041)**
**ESHITA MISHRA (RA2011042010062)**

# INDEX

# ABSTRACT

Agricultural productivity is something on which the economy highly depends. This is one of the reasons that disease detection in plants plays a vital role in the agricultural field. If proper care is not taken in this area then it might cause severe effects on plants and due to which respective product quality, quantity, or productivity is affected. Detection of plant diseases through some automatic technique is beneficial as it reduces the large work of monitoring big farms of crops, and at the very early stage itself, it detects the symptoms of diseases. Our agenda is to train all images of leaves concerning the classes, hence when we provide a new image that the model has not seen to view its accuracy and capability in generating or detecting whether that particular image belongs to that particular class of disease. Here, we will use concepts of Deep Learning.

# INTRODUCTION

Early plant leaf disease detection plays a significant role in efficient crop yield. Plant diseases like black measles, black rot, bacterial spot, etc. affect the growth, and crop quality of plants, and have economic impacts on the agriculture industry.

To avoid the impact of these diseases, expensive approaches and the use of pesticides are some solutions the farmers usually implement. The use of chemical means damages the plant and the surrounding environment. In addition, this kind of approach intensifies the cost of production and causes major monetary loss to farmers.

Early discovery of diseases as they occur is the most important period for efficient disease management. Manual disease detection through human experts to identify and recognize plant diseases is a usual practice in agriculture. With the improvements in technology, automatic detection of plant diseases from raw images is possible through computer vision and artificial intelligence.

Image processing techniques are now commonly employed in agriculture and it is applied for the detection and recognition of weeds, fruit-grading, identifying and calculating disease infestations of plants, and plant genomics. Currently, the introduction of deep learning methods turns out to be popular.

Deep learning is the advanced method of machine learning that uses neural networks that works like the human brain. Traditional methods involve the use of semantic features as the classification method. Deep learning is defined as a neural network learning process and it can automatically obtain features through image patterns.

A convolutional neural network (CNN) is a deep learning model that is widely used in image processing. We will focus on building a hybrid model to obtain the characteristics of leaves using CNN and classify the extracted features of leaves.

The methodology in the study involves three key stages:

- acquisition of data,
- pre-processing of data, and
- image classification.
- Predicting the class of disease

# PROBLEM EXPLANATION

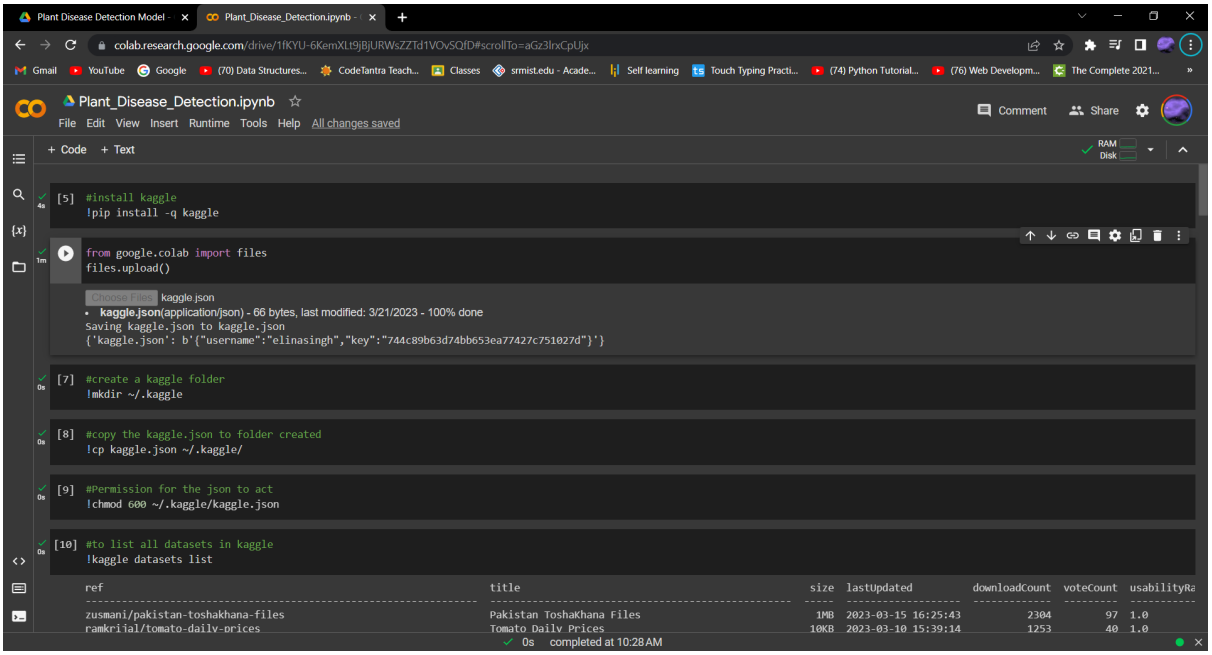
Plant diseases have turned into a dilemma as they can cause significant reductions in both the quality and quantity of agricultural products. Automatic detection of plant diseases is an essential research topic as it may prove beneficial in monitoring large fields of crops, and thus automatically detect the symptoms of diseases as soon as they appear on plant leaves. The proposed system is a software solution for the automatic detection and classification of plant leaf diseases. As a result, in the field of agriculture, disease identification in plants is important. Plants are highly susceptible to diseases that inhibit plant development, which affects the farmer's ecology. The use of an automated disease detection technique is advantageous in detecting a plant disease at an early stage. Plant diseases manifest themselves in various parts of the plant, such as the leaves. It takes a long time to manually diagnose plant disease using leaf photographs. As a result, computational methods must be developed to automate the process of disease detection and classification using leaf images.

# ABOUT DATASET

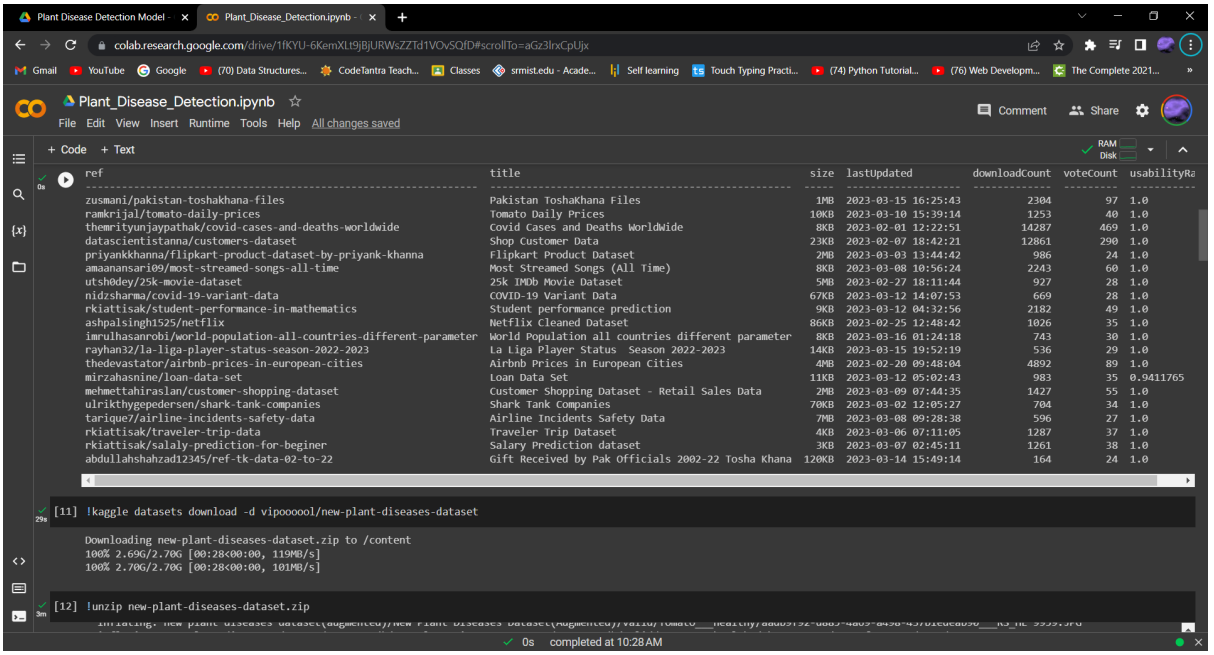
This dataset is recreated using offline augmentation from the original dataset. This dataset consists of about 87K RGB images of healthy and diseased crop leaves which are categorized into 38 different classes. The total dataset is divided into an 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purposes. It is a 2GB dataset. The new plant disease folder has two more folders train and valid. Inside each, there are multiple folders of different classes.

**Uploading data from Kaggle to google colab:**

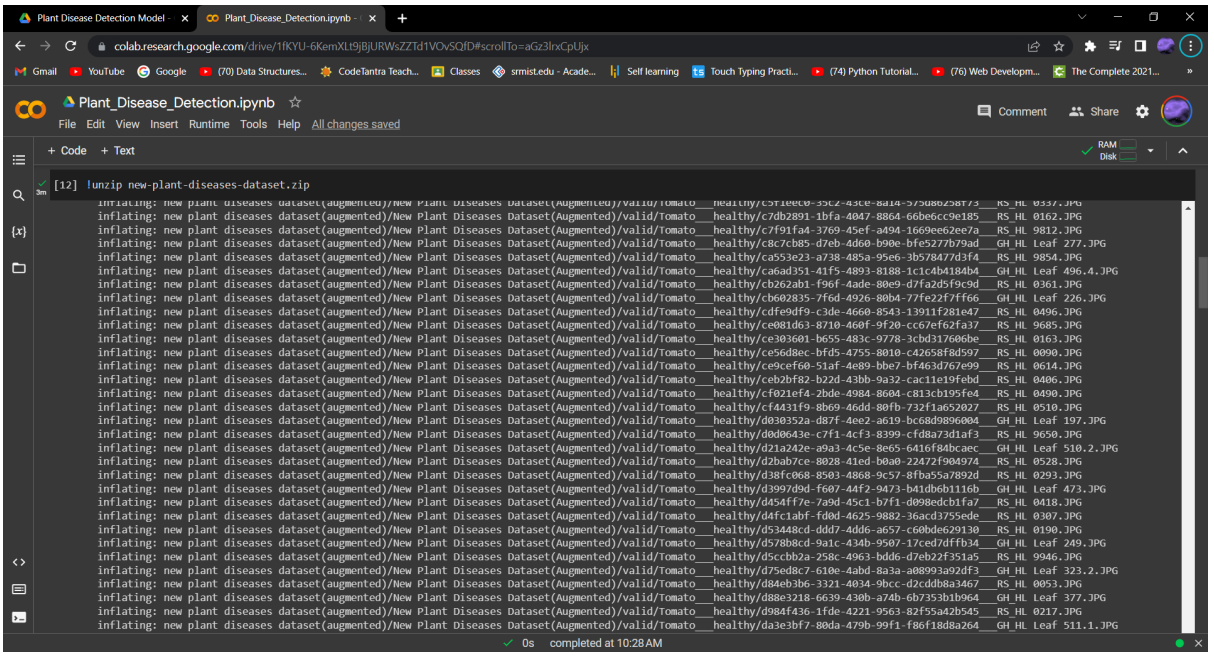
1) Go to your Kaggle website and click ***Account*** or you can register
2) Scroll a little down and click ***Create New API Token*** to generate your ***kaggle.json*** file from where you can fetch all datasets of kaggle from colab
3) Now go to the colab and run the following codes:




4) Now select Generate API for dataset you want to be downloaded and paste the link -> `!kaggle datasets download -d vipoooool/new-plant-diseases-dataset`

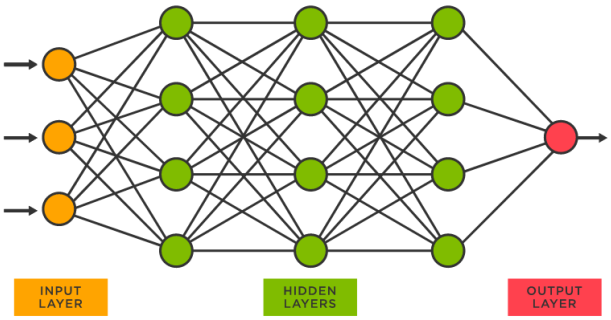



5) Unzip the folder-> `!unzip`

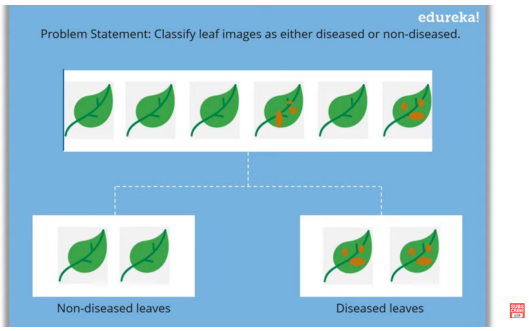


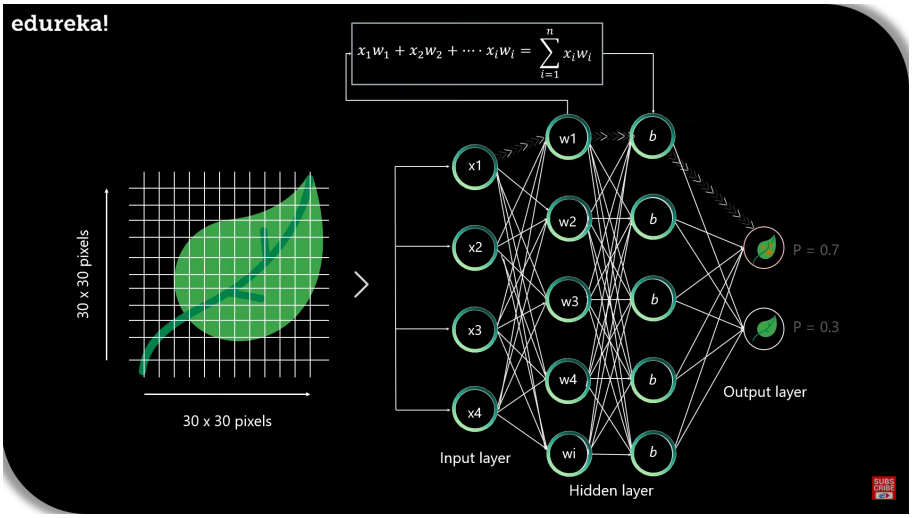
# CONCEPTS FOLLOWED

## Neural Networks:




The functional unit of deep learning, it mimics the behavior of the human brain to solve complex data-driven problems. A neural network functions when some input data is fed to it. This data is then processed via layers of Perceptrons to produce a desired output.



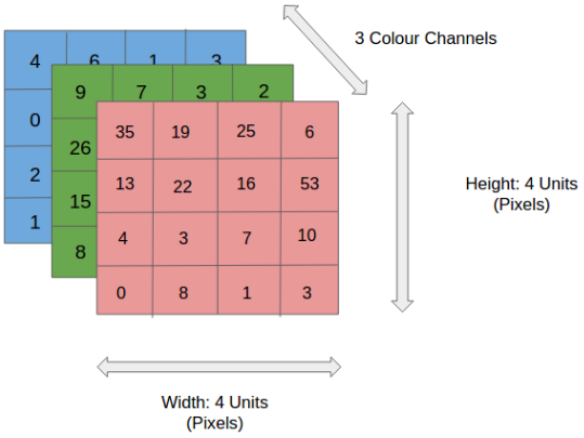
In this case, each leaf image will be broken down into pixels depending on the dimension of the image. For example, if the image is composed of 30x30 pixels then the total number of pixels will be 900. These pixels are represented as matrices which are then fed into the input layer of the neural network, just like how our brains have neurons that help build and connect thoughts. An Artificial Neural Network has perceptrons that accept inputs and process them by passing them from the input layer, then to the hidden layer, and finally to the output layer. Now as the inputs when passed from the input layer to the hidden layer, a random set of weights is assigned to each input. These inputs are then multiplied by their corresponding weights and their sum is further processed through the network. Then we assign a numerical value called the bias to each perceptron. Furthermore, each perceptron is passed through an activation or transformation function that determines whether a particular perceptron gets activated. An activated perceptron is used to transmit data to the next layer. The data is propagated forward through the neural network until the perceptrons reach the output layer. At the output layer, a probability is derived which decides whether the data belongs to Class A (diseased leaf) or Class B (non-diseased leaf).

**Backpropagation method:**

What if the predicted output is wrong? In such a situation we train the Neural Network by using Backpropagation. Within the hidden layer, we initialized the weights, which determine the importance of each input variable. Here, we propagate backward and compare the actual output to the predicted output we can readjust the weights in such a way that the error is minimized this results in a more accurate output.

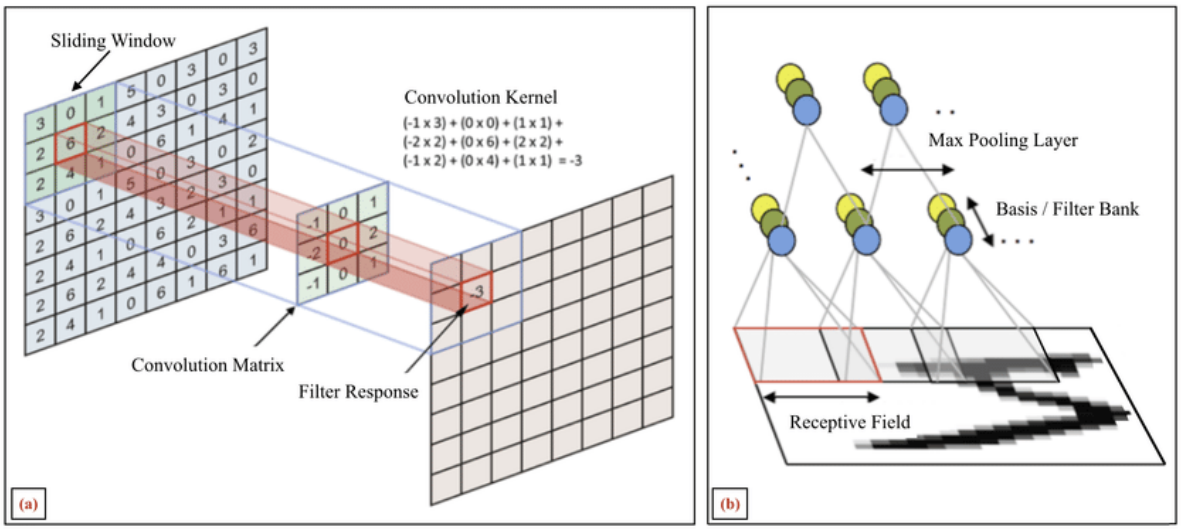
**Convolutional Neural Network (ConvNet/CNN):**

Before we go to the working of CNN's let's cover the basics such as what is an image and how is it represented. An RGB image is nothing but a matrix of pixel values having three planes whereas a grayscale image is the same but it has a single plane. Take a look at this image to understand more.




It is a Deep Learning Algorithm that is also called an artificial neural network. It possesses some specialization in being capable of picking patterns and making sense of them. Pattern detection makes it useful for image analysis.

CNN has hidden layers which are called Convolutional Layers. These layers receive input and then transform in some way which ultimately works as input for the next layer. This transformation is done by a Convolutional Operation. With each layer, we have to specify the number of filters. Filters are relatively small matrices for which we decide the number of rows and columns and values within them are initialized with random numbers. The Filters slide throughout the pixels of the image, the sliding action is referred to as Convolving.



The 3X3 Filter (Convolution Matrix) lands on the first 3X3 block of the image, then the dot product of the filter itself is computed and stored for the next layer. It occurs for each 3X3 block where the filter convolves.

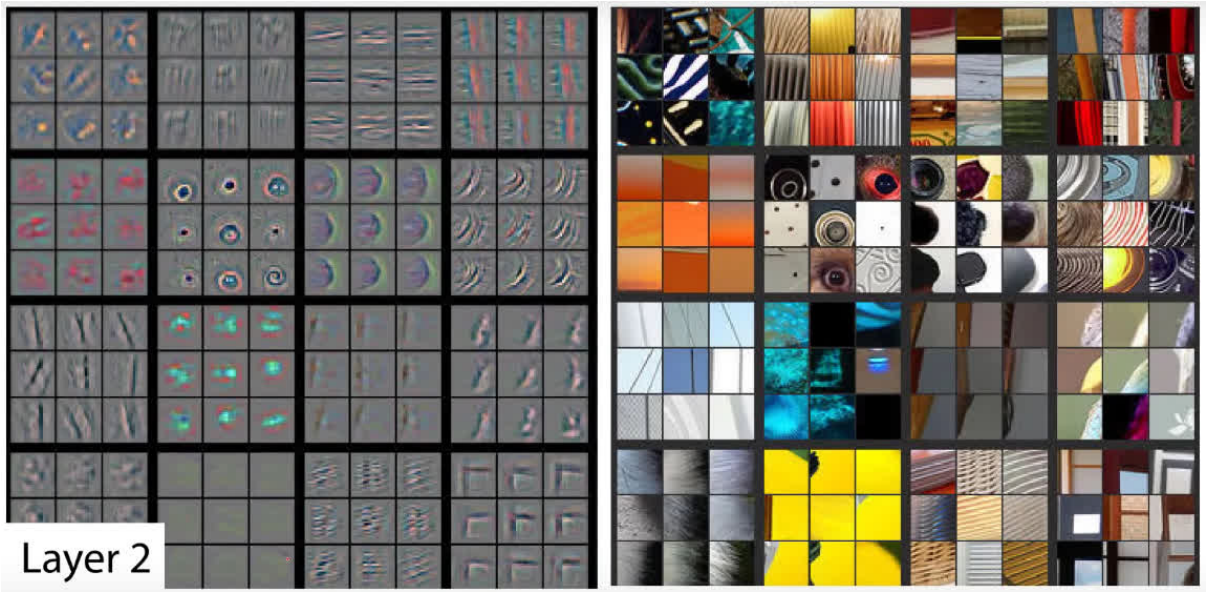

Here, the input image is a (7) with pixel values ranging from

- -1 = black
- 0 = white
- 1 = gray

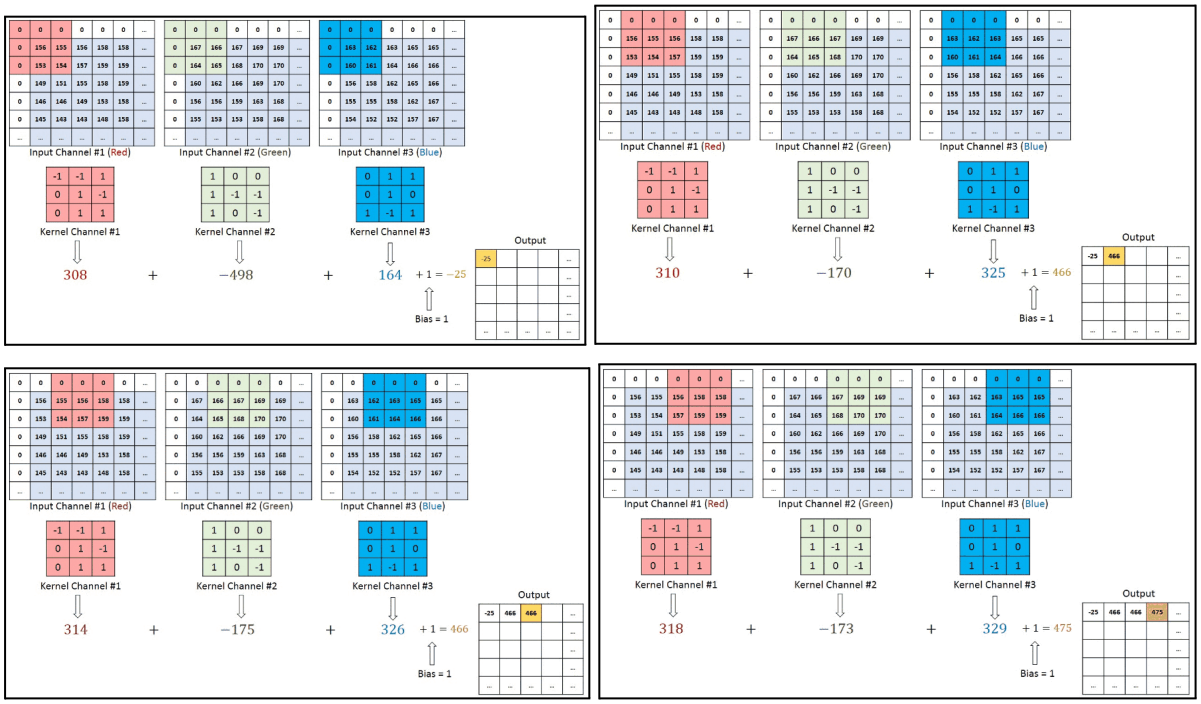
Each filter is slid across the image and the following outputs are received.

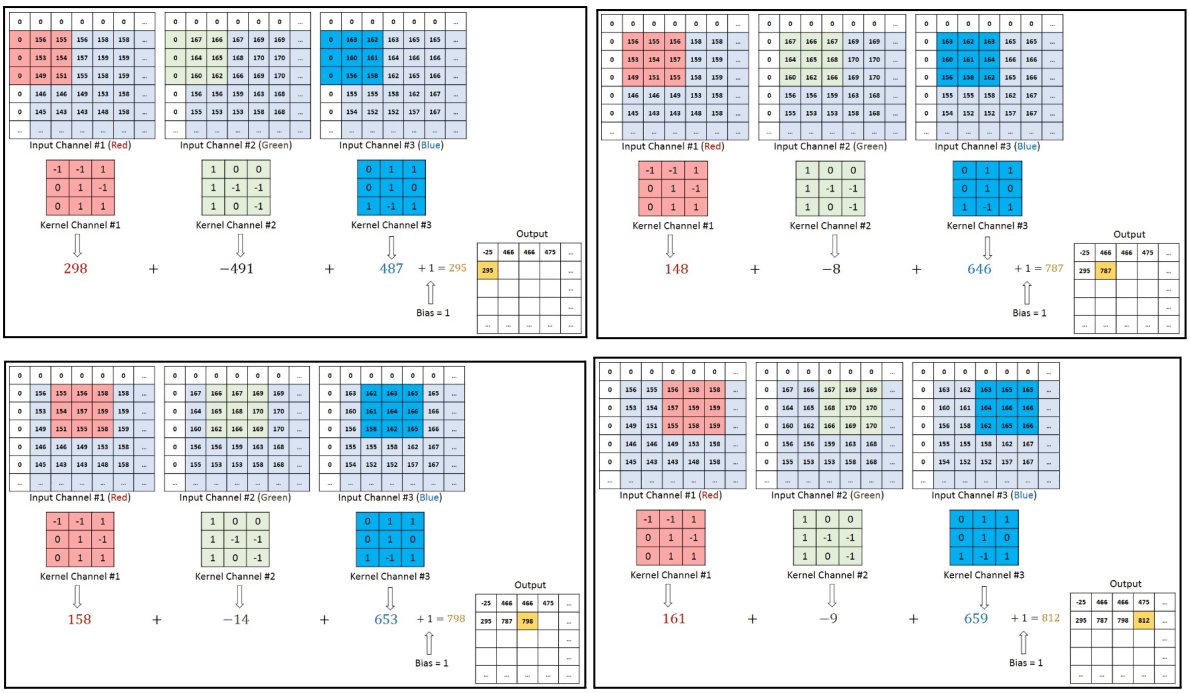
1. First image: detects top horizontal edges of 7
2. Second image: detects left vertical edges
3. Third image: bottom horizontal edges
4. Fourth image: right vertical edges

This is a simple representation of the patterns that are detected from the filters. Most CNNs detect complex features and patterns hence, best for classification.

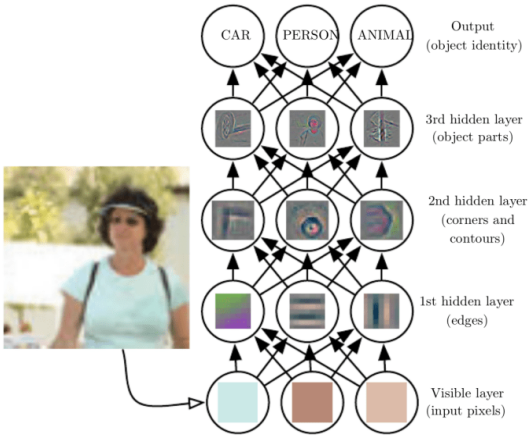



In the case of RGB color, the channel takes a look at this certain images to understand its working






Based on the activation map of the final convolution layer, the classification layer outputs a set of confidence scores (values between 0 and 1) that specify how likely the image is to belong to a "class." For instance, if you have a ConvNet that detects cats, dogs, and horses, the output of the final layer is the possibility that the input image contains any of those animals.

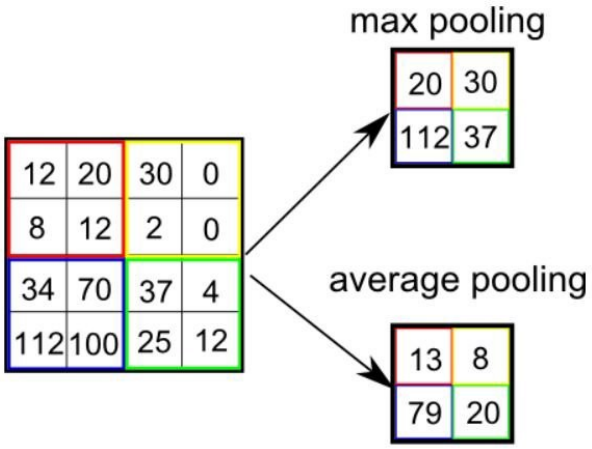



**Pooling Layer:**

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to decrease the computational power required to process the data by reducing the dimensions. There are two types of pooling average pooling and max pooling. I've only had experience with Max Pooling so far I haven't faced any difficulties.

So what we do in Max Pooling is find the maximum value of a pixel from a portion of the image covered by the kernel. Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.

On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel. Average Pooling simply performs dimensionality reduction as a noise-suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.




**VGG:**

VGG stands for Visual Geometry Group; it is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. The "deep" refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers. The VGG architecture is the basis of ground-breaking object recognition models. Developed as a deep neural network, the VGGNet also surpasses baselines on many tasks and datasets beyond ImageNet. Moreover, it is now still one of the most popular image recognition architectures.

**VGG16:**

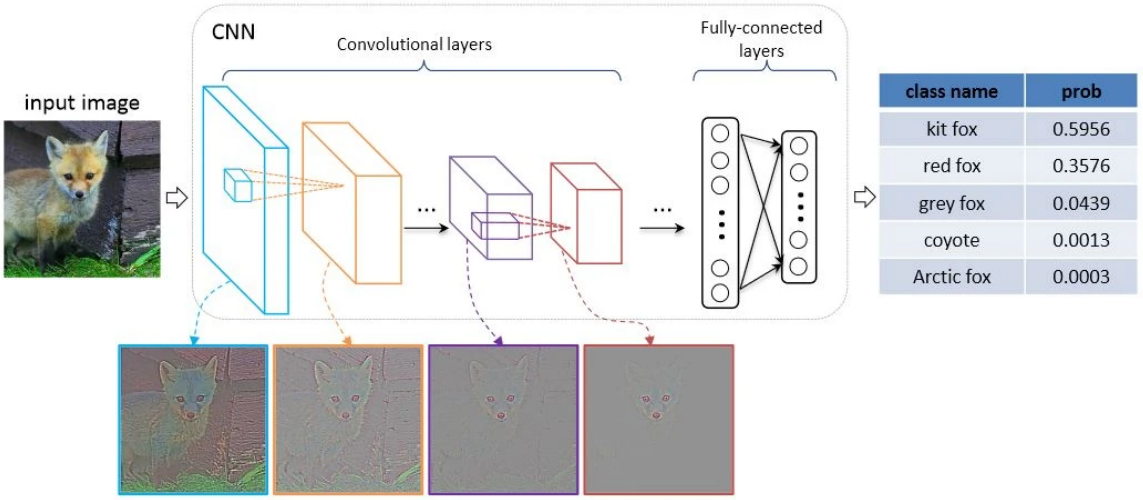
The VGG model, or VGGNet, that supports 16 layers is also referred to as VGG16, which is a convolutional neural network model proposed by A. Zisserman and K. Simonyan from the University of Oxford. These researchers published their model in the research paper titled, "Very Deep Convolutional Networks for Large-Scale Image Recognition." The VGG16 model achieves almost 92.7% top-5 test accuracy in ImageNet. ImageNet is a dataset consisting

of more than 14 million images belonging to nearly 1000 classes. Moreover, it was one of the most popular models submitted to ILSVRC-2014. It replaces the large kernel-sized filters with several 3×3 kernel-sized filters one after the other, thereby making significant improvements over AlexNet. The VGG16 model was trained using Nvidia Titan Black GPUs for multiple weeks. As mentioned above, the VGGNet-16 supports 16 layers and can classify images into 1000 object categories, including keyboard, animals, pencil, mouse, etc. Additionally, the model has an image input size of 224-by-224.

**VGG19:**

The concept of the VGG19 model (also VGGNet-19) is the same as the VGG16 except that it supports 19 layers. The "16" and "19" stand for the number of weight layers in the model (convolutional layers). This means that VGG19 has three more convolutional layers than VGG16. We'll discuss more on the characteristics of VGG16 and VGG19 networks in the latter part of this article.

**VGG Architecture:**




The VGG network is constructed with very small convolutional filters. The VGG-16 consists of 13 convolutional layers and three fully connected layers. Let's take a brief look at the architecture of VGG:

**Input:** The VGGNet takes in an image input size of 224×224. For the ImageNet competition, the creators of the model cropped out the center 224×224 patch in each image to keep the input size of the image consistent.

**Convolutional Layers:** VGG's convolutional layers leverage a minimal receptive field, i.e., 3×3, the smallest possible size that still captures up/down and left/right. Moreover, there are also 1×1 convolution filters acting as a linear

transformation of the input. This is followed by a ReLU unit, which is a huge innovation from AlexNet that reduces training time. ReLU stands for rectified linear unit activation function; it is a piecewise linear function that will output the input if positive; otherwise, the output is zero. The convolution stride is fixed at 1 pixel to keep the spatial resolution preserved after convolution (stride is the number of pixel shifts over the input matrix).
**Hidden Layers:** All the hidden layers in the VGG network use ReLU. VGG does not usually leverage Local Response Normalization (LRN) as it increases memory consumption and training time. Moreover, it makes no improvements to overall accuracy.
**Fully-Connected Layers:** The VGGNet has three fully connected layers. Out of the three layers, the first two have 4096 channels each, and the third has 1000 channels, 1 for each class.

**Early Stopping and Model Checkpoint:**
***Early stopping:*** stop the training when a condition is met

***Checkpoint:*** frequently save the model

The purpose of Early Stopping is to avoid overfitting by stopping the model before it happens using a defined condition. If you use it, and then you save the model when the training is stopped, you will get **a model** that is **assumed** to be good enough and not overfitted.

The purpose of the class ModelCheckpoint is to save models several times while training. This can be useful to find at which epoch the model gets the best performance. So, if you use it, you will get **several models** that are saved at different epochs (or, more generally, "checkpoints").

Even after using both methods, you will get some models, but they have different purposes. None of them is better than the other. I almost always use both methods at the same time. You can use early stopping to stop the training and save a lot of models while training using ModelCheckpoint. In most of my cases, the best model is around the epoch during early stopping.

*note: the model saving process is not done by EarlyStopping

# TOOLS AND FRAMEWORKS


**TensorFlow:**
TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

**Keras:**
Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, Theano, and PlaidML.

**Matplotlib:**
Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

**Google Colab:**
Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis, and education.

**Pandas:**
pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.

**Python:**
Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented, and functional programming.

**Numpy:**
NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

**Seaborn:**
Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. For a brief introduction to the ideas behind the library, you can read the introductory notes or the paper.

# TECHNIQUES TO BUILD THE MODEL

**Step 1:** import all datasets from the kaggle website using API keys

**Step 2:** Unzip the folder

**Step 3:** Import necessary Libraries

**Step 4:** Perform Exploratory Data Analysis on the dataset provided

**Step 5:** Preprocess the image dataset

**Step 6:** plot the images to visualize them

**Step 7:** Start building the model

**Step 8:** Use the built-in pre-trained model and tune it accordingly

**Step 9:** Select VGG19 from keras

**Step 10:** Add Flatten and  Dense layers to the VGG19 network

**Step 11:** Compile the model

**Step 12:** Apply early stopping and Model Checkpoint

**Step 13:** Train the model with 50 epochs using GPU hardware accelerator

**Step 14:** Plot the model's accuracy and loss curves

**Step 15:** Load the model as best_model.h5 file

**Step 16:** Evaluate the model

**Step 17:** store the class indices of each type of diseases in a dictionary

**Step 18:** make a function prediction where we preprocess the loaded image

**Step 19:** The function should return the predicted outcome of the image

**Step 20:** pass the path of the image to the function and run

# IMPLEMENTATION DETAILS

[11]:
```
#importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
```

[12]:
```
import keras
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.applications.vgg19 import VGG19, preprocess_input,decode_predictions
```

[13]:
```
#EDA
len(os.listdir("/content/New Plant Diseases Dataset(Augmented)/New Plant␣
↪Diseases Dataset(Augmented)/train"))
```

[13]: 38

[14]:
```
train_datagen = ImageDataGenerator(zoom_range=0.5, shear_range=0.3,␣
↪horizontal_flip=True,preprocessing_function=preprocess_input)
val_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
```

[15]:
```
#preprocessing of data using builtin function (preprocessing_input)
train = train_datagen.flow_from_directory(directory= "/content/New Plant␣
↪Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train",
                                          target_size=(256,256),
                                          batch_size=32)
val = val_datagen.flow_from_directory(directory= "/content/New Plant Diseases␣
↪Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/valid",
                                          target_size=(256,256),
                                          batch_size=32)
```

```
Found 70295 images belonging to 38 classes.
Found 17572 images belonging to 38 classes.
```

[16]:
```
t_img, label=train.next()
```

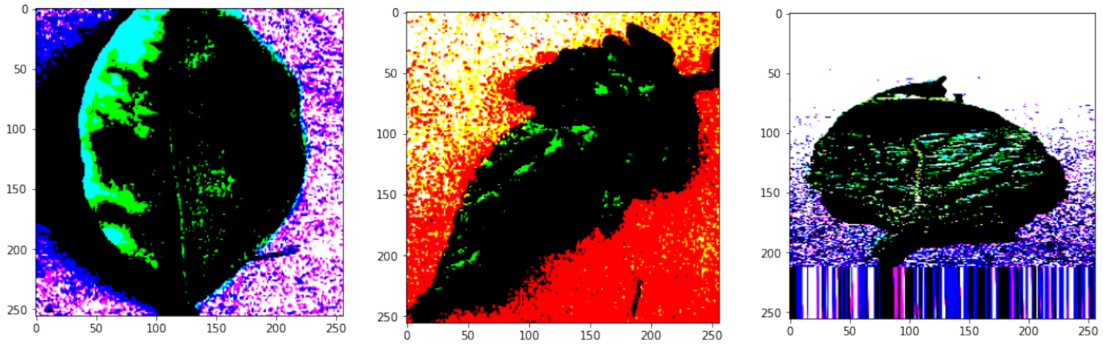
[17]:
```
t_img.shape
```

[17]: (32, 256, 256, 3)

[18]:
```
def plotImage(img_arr,label):
  for im, l in zip(img_arr,label):
    plt.figure(figsize=(5,5))
    plt.imshow(im)
    plt.show()
```

[19]:
```
#visualising our data
plotImage(t_img[:3],label[:3])
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```



# 1 Building The Model

[20]:
```
from keras.layers import Dense, Flatten
from keras.models import Model
from keras.applications.vgg19 import VGG19
import keras
```

[21]:
```
base_model = VGG19(input_shape=(256,256,3), include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 4s 0us/step
```

[22]:
```
for layer in base_model.layers:
  layer.trainable = False
```

[23]:
```
base_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
```

```
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv4 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv4 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808


 block5_conv3 (Conv2D)        (None, 16, 16, 512)       2359808

 block5_conv4 (Conv2D)        (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 0
```

```
Non-trainable params: 20,024,384
_________________________________________________________________
```

[24]:
```
X = Flatten()(base_model.output)
X = Dense(units = 38,activation = 'softmax')(X)

#Creating the model
model = Model(base_model.input, X)
```

[25]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

block4_conv1 (Conv2D)        (None, 32, 32, 512)       1180160

block4_conv2 (Conv2D)        (None, 32, 32, 512)       2359808

block4_conv3 (Conv2D)        (None, 32, 32, 512)       2359808

block4_conv4 (Conv2D)        (None, 32, 32, 512)       2359808

block4_pool (MaxPooling2D)   (None, 16, 16, 512)       0
```

```
block5_conv1 (Conv2D)        (None, 16, 16, 512)       2359808

block5_conv2 (Conv2D)        (None, 16, 16, 512)       2359808

block5_conv3 (Conv2D)        (None, 16, 16, 512)       2359808

block5_conv4 (Conv2D)        (None, 16, 16, 512)       2359808

block5_pool (MaxPooling2D)   (None, 8, 8, 512)         0

flatten (Flatten)            (None, 32768)             0

dense (Dense)                (None, 38)                1245222

=================================================================
Total params: 21,269,606
Trainable params: 1,245,222
Non-trainable params: 20,024,384
_________________________________________________________________
```

[26]:
```
model.compile(optimizer='adam',
              loss=keras.losses.categorical_crossentropy,
              metrics=['accuracy'])
```

## 2 Early Stopping and Model Check point

[27]:
```
from keras.callbacks import ModelCheckpoint, EarlyStopping

#early stopping
es = EarlyStopping(monitor='val_accuracy',
                   min_delta = 0.01,
                   patience = 3,
                   verbose=1)

#model check point
mc = ModelCheckpoint(filepath='best_model.h5',
                     monitor='val_accuracy',
                     min_delta = 0.01,
                     patience = 3,
                     verbose=1)

cb = [es,mc]
```

[28]:
```
his = model.fit_generator(train,
                          steps_per_epoch=16,
```

```
                        epochs=50,
                        verbose=1,
                        callbacks=cb,
                        validation_data=val,
                        validation_steps=16)
```

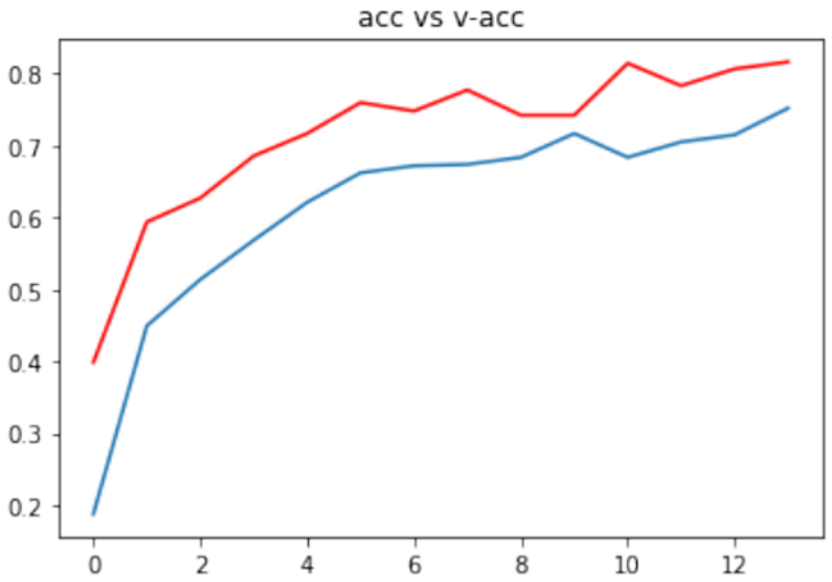
```
<ipython-input-28-cde0eb368c9c>:1: UserWarning: `Model.fit_generator` is
deprecated and will be removed in a future version. Please use `Model.fit`,
which supports generators.
  his = model.fit_generator(train,
Epoch 1/50
16/16 [==============================] - ETA: 0s - loss: 27.8529 - accuracy:
0.1875
Epoch 1: saving model to best_model.h5
16/16 [==============================] - 26s 861ms/step - loss: 27.8529 -
accuracy: 0.1875 - val_loss: 18.0845 - val_accuracy: 0.3984
Epoch 2/50
16/16 [==============================] - ETA: 0s - loss: 14.9172 - accuracy:
0.4492
Epoch 2: saving model to best_model.h5
16/16 [==============================] - 13s 787ms/step - loss: 14.9172 -
accuracy: 0.4492 - val_loss: 8.8087 - val_accuracy: 0.5938
Epoch 3/50
16/16 [==============================] - ETA: 0s - loss: 11.7913 - accuracy:
0.5137
Epoch 3: saving model to best_model.h5
16/16 [==============================] - 12s 729ms/step - loss: 11.7913 -
accuracy: 0.5137 - val_loss: 9.8874 - val_accuracy: 0.6270
Epoch 4/50
16/16 [==============================] - ETA: 0s - loss: 11.5055 - accuracy:
0.5684
Epoch 4: saving model to best_model.h5
16/16 [==============================] - 14s 862ms/step - loss: 11.5055 -
accuracy: 0.5684 - val_loss: 8.1355 - val_accuracy: 0.6855
Epoch 5/50
16/16 [==============================] - ETA: 0s - loss: 8.6887 - accuracy:
0.6211
Epoch 5: saving model to best_model.h5
```

```
Epoch 9/50
16/16 [==============================] - ETA: 0s - loss: 7.8228 - accuracy:
0.6836
Epoch 9: saving model to best_model.h5
16/16 [==============================] - 15s 915ms/step - loss: 7.8228 -
accuracy: 0.6836 - val_loss: 6.6605 - val_accuracy: 0.7422
Epoch 10/50
16/16 [==============================] - ETA: 0s - loss: 9.5581 - accuracy:
0.7168
Epoch 10: saving model to best_model.h5
16/16 [==============================] - 12s 757ms/step - loss: 9.5581 -
accuracy: 0.7168 - val_loss: 7.5905 - val_accuracy: 0.7422
Epoch 11/50
16/16 [==============================] - ETA: 0s - loss: 9.0940 - accuracy:
0.6836
Epoch 11: saving model to best_model.h5
16/16 [==============================] - 14s 873ms/step - loss: 9.0940 -
accuracy: 0.6836 - val_loss: 5.2088 - val_accuracy: 0.8145
Epoch 12/50
16/16 [==============================] - ETA: 0s - loss: 8.4204 - accuracy:
0.7051
Epoch 12: saving model to best_model.h5
16/16 [==============================] - 12s 769ms/step - loss: 8.4204 -
accuracy: 0.7051 - val_loss: 6.0539 - val_accuracy: 0.7832
Epoch 13/50
16/16 [==============================] - ETA: 0s - loss: 8.7285 - accuracy:
0.7148
Epoch 13: saving model to best_model.h5
16/16 [==============================] - 12s 743ms/step - loss: 8.7285 -
accuracy: 0.7148 - val_loss: 6.5708 - val_accuracy: 0.8066
Epoch 14/50
16/16 [==============================] - ETA: 0s - loss: 7.3961 - accuracy:
0.7520
Epoch 14: saving model to best_model.h5
16/16 [==============================] - 13s 791ms/step - loss: 7.3961 -
accuracy: 0.7520 - val_loss: 5.5839 - val_accuracy: 0.8164
Epoch 14: early stopping
```

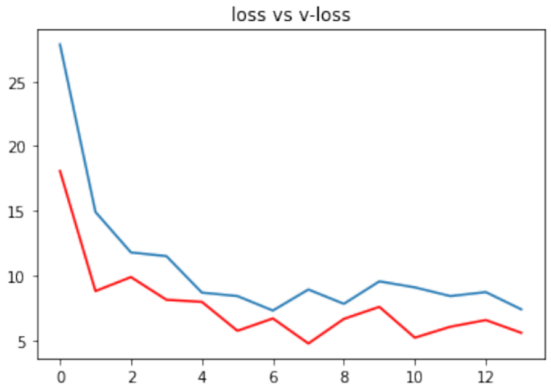
[29]:
```
#plotting the model
h=his.history
h.keys()
```

[29]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[30]:
```
plt.plot(h['accuracy'])
plt.plot(h['val_accuracy'],c="red")
plt.title("acc vs v-acc")
plt.show()
```




[31]:
```
plt.plot(h['loss'])
plt.plot(h['val_loss'],c="red")
plt.title("loss vs v-loss")
plt.show()
```



[32]:
```
#load best model
from keras.models import load_model
model = load_model("/content/best_model.h5")
```

[33]:
```
#evaluation of our model
acc = model.evaluate_generator(val)[1]
print(f"The accuracy of your model is= {acc*100}%")
```

```
<ipython-input-33-e8abd9b8e124>:2: UserWarning: `Model.evaluate_generator` is
deprecated and will be removed in a future version. Please use `Model.evaluate`,
which supports generators.
  acc = model.evaluate_generator(val)[1]
```

```
The accuracy of your model is= 82.1420431137085%
```

[46]:
```
ref = dict(zip(list(train.class_indices.values()) , list(train.class_indices.
 ↪keys())))
ref
```

[46]:
```
{0: 'Apple___Apple_scab',
 1: 'Apple___Black_rot',
 2: 'Apple___Cedar_apple_rust',
 3: 'Apple___healthy',
 4: 'Blueberry___healthy',
 5: 'Cherry_(including_sour)___Powdery_mildew',
```

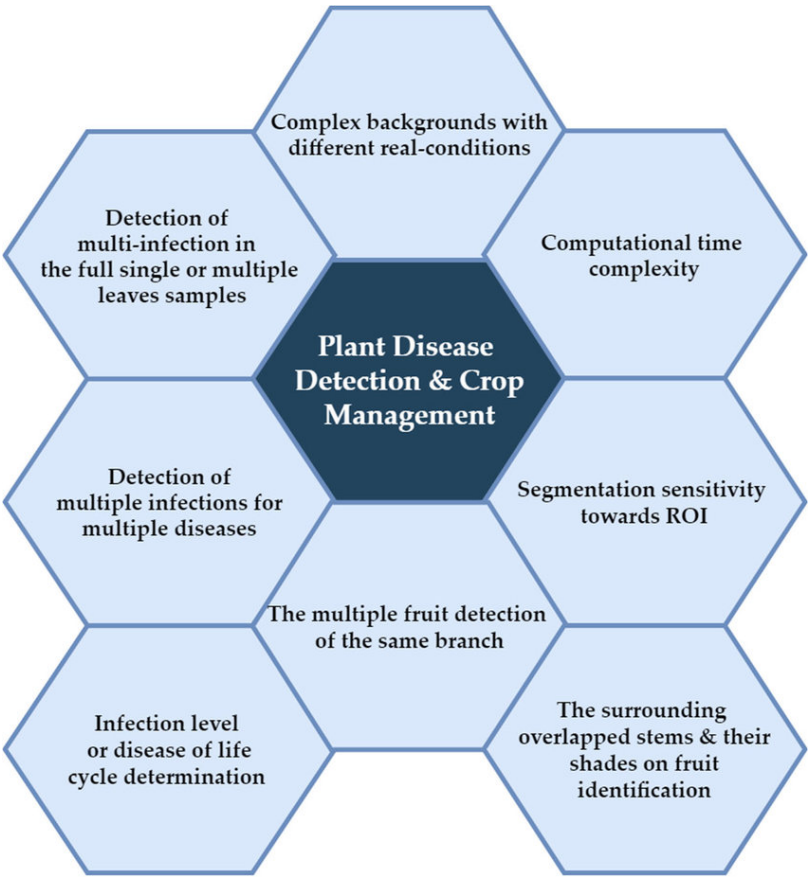
```
6: 'Cherry_(including_sour)___healthy',
7: 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
8: 'Corn_(maize)___Common_rust_',
9: 'Corn_(maize)___Northern_Leaf_Blight',
10: 'Corn_(maize)___healthy',
11: 'Grape___Black_rot',
12: 'Grape___Esca_(Black_Measles)',
13: 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
14: 'Grape___healthy',
15: 'Orange___Haunglongbing_(Citrus_greening)',
16: 'Peach___Bacterial_spot',
17: 'Peach___healthy',
18: 'Pepper,_bell___Bacterial_spot',
19: 'Pepper,_bell___healthy',
20: 'Potato___Early_blight',
21: 'Potato___Late_blight',
22: 'Potato___healthy',
23: 'Raspberry___healthy',
24: 'Soybean___healthy',
25: 'Squash___Powdery_mildew',
26: 'Strawberry___Leaf_scorch',
27: 'Strawberry___healthy',
28: 'Tomato___Bacterial_spot',
29: 'Tomato___Early_blight',
30: 'Tomato___Late_blight',
31: 'Tomato___Leaf_Mold',
32: 'Tomato___Septoria_leaf_spot',
33: 'Tomato___Spider_mites Two-spotted_spider_mite',
34: 'Tomato___Target_Spot',
35: 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
36: 'Tomato___Tomato_mosaic_virus',
37: 'Tomato___healthy'}
```

[50]:
```python
def prediction(path):
  img = load_img(path,target_size = (256,256))
  i = img_to_array(img)
  im = preprocess_input(i)
  img = np.expand_dims (im,axis=0)
  pred = np.argmax(model.predict(img))
  print(f"The image belongs to: {ref[pred]}")
```

[51]:
```python
path = "/content/test/test/AppleCedarRust4.JPG"
prediction(path)
```

```
1/1 [==============================] - 0s 33ms/step
The image belongs to: Apple___Cedar_apple_rust
```

# CHALLENGES AND LIMITATIONS




The problem associated with automatic plant disease identification using visible range images has received considerable attention in the last two decades, however, the techniques proposed so far are usually limited in their scope and dependent on ideal capture conditions in order to work properly. This apparent lack of significant advancements may be partially explained by some difficult challenges posed by the subject: the presence of complex backgrounds that cannot be easily separated from the region of interest (usually leaf and stem), boundaries of the symptoms often are not well defined, uncontrolled capture conditions may present characteristics that make the image analysis more difficult, certain diseases produce symptoms with a wide range of characteristics, the symptoms produced by different diseases may be very similar, and they may be present simultaneously. This paper provides an analysis of each one of those challenges, emphasizing both the problems that they may cause and how they may have potentially affected the techniques proposed in the past.

# RESOURCES


1. https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/
2. https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset
3. https://youtu.be/amt9ZmGofJk
4. https://youtu.be/57N1g8k2Hwc
5. https://youtu.be/YRhxdVk_sIs
6. https://youtu.be/vpOLiDyhNUA
7. https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
8. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53