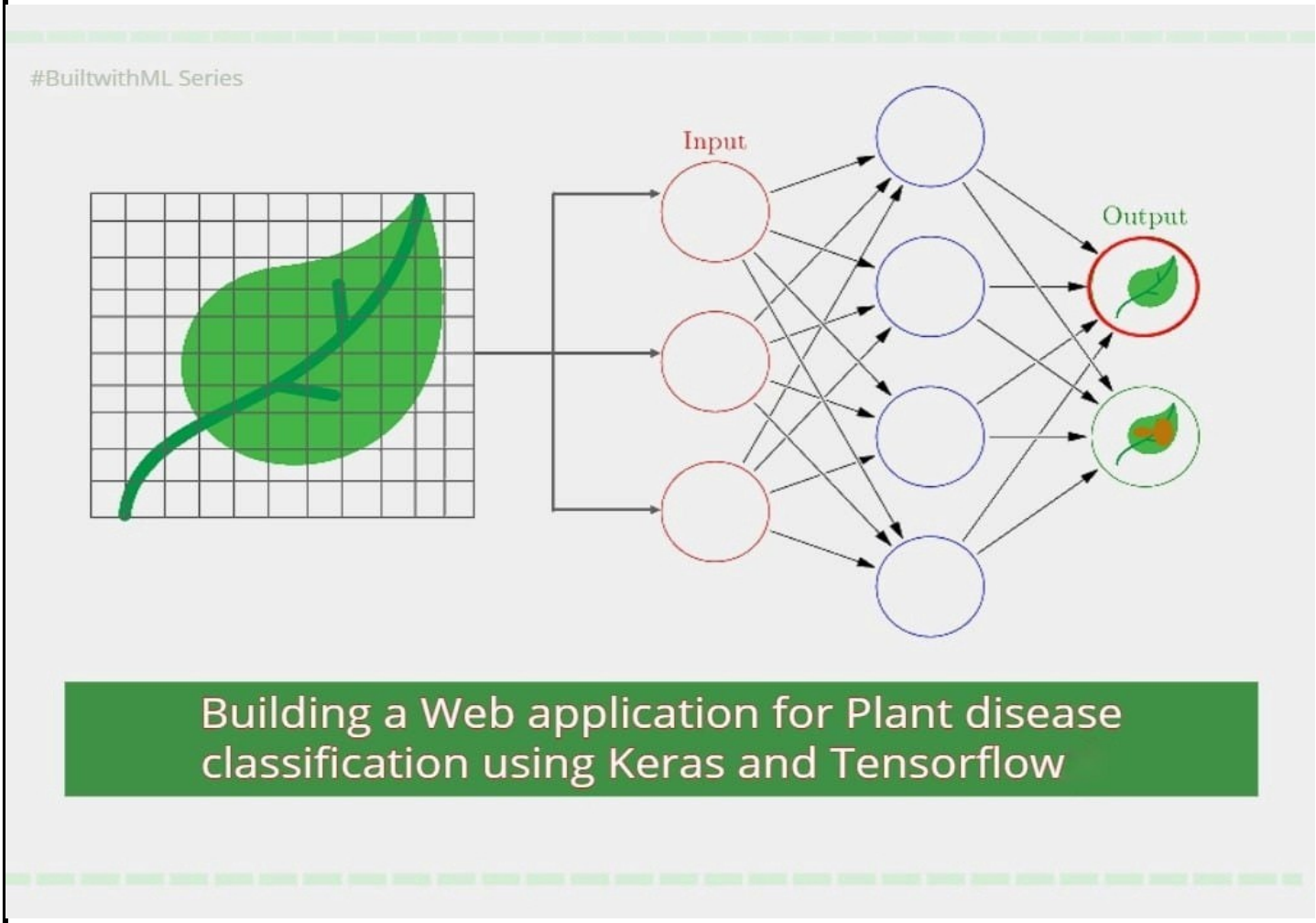
Plant Leaf Disease

Detection



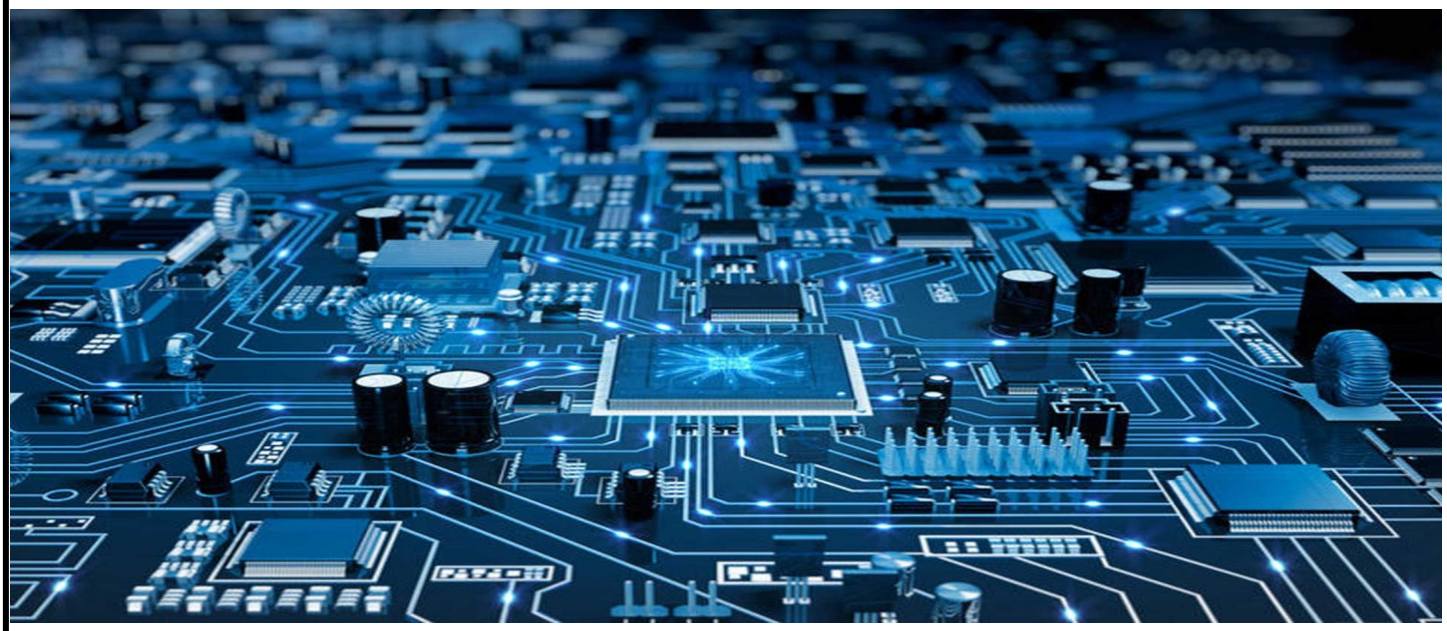
- **Introduction.**
- **Hardware and software requirements.**
- **Objectives & Procedure.**
- **Dataset description.**
- **Approach.**
- **Source Code.**
- **Screenshots.**
- **Advantages.**
- **Future scope.**
- **Conclusion.**

INTRODUCTION

The primary occupation in India is agriculture. India ranks second in the agricultural output worldwide. Here in India, farmers cultivate a great diversity of crops. Various factors such as climatic conditions, soil conditions, various disease, etc., affect the production of the crops. The existing method for plants disease detection is simply naked eye observation which requires more man labour, properly equipped laboratories, expensive devices, etc. And improper disease detection may led to inexperienced pesticide usage that can cause development of long term resistance of the pathogens, reducing the ability of the crop to fight back. The plant disease detection can be done by observing the spot on the leaves of the affected plant. The method we are adopting to detect plant diseases is image processing using Convolution neural network (CNN). The first implementation of the plant disease detection using image processing was done by Shen WeizhegWuyachun Chen Zhanliang and Wi Hangda in their paper.

CNN combines well-read features with input data, and then it uses 2D convolutional layers, and hence makes this architecture more suitable for processing 2D data, like images. CNNs abolish the demand for manual feature removal and extraction for the classification of the images. The CNN model of its own extracts features straight from images. The features that are extracted aren't pre-trained; they are well-read while the network is trained on few groups of images. The Convolutional Neural Network (CNN) model has numerous of layers which execute the processing of image in convolutional layers include- Input layer, Output Layer, Convo Layer, , Fully, Soft-max layer, Connected layer, Pooling Layer.

HARDWARE AND SOFTWARE REQUIREMENTS



Software Requirements -_

- Operating system : Windows 10
- Programming Language : Python 3.8.12
- Software (IDE) : Jupyter IDE
- Platform : Anaconda

Hardware Requirements –_

- Speed : 2666 MHz
- Hard-disk : 8 GB
- RAM : 512 MB

OBJECTIVES & PROCEDURE

The objective of this project is timely identification and early prevention of crop diseases which are essential for improving the crop production. In this paper, deep convolutional-neural-network (CNN) models are implemented to identify and diagnose diseases in plants from their leaves, since CNNs have achieved impressive results in the field of machine vision. An automated system designed to help identify plant diseases by the plant's appearance and visual symptoms could be of great help to amateurs in the gardening process and also trained professionals as a verification system in disease diagnostics. In machine learning, a Convolutional Neural Network (CNN/ConvNet) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The accuracy results in the identification of diseases showed that the deep CNN model is promising and can greatly impact the efficient identification of the
diseases, and may have potential in the detection of diseases in real-time agricultural systems.

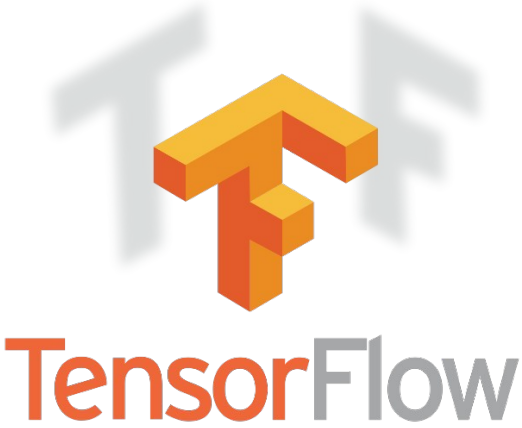
Using the deep convolutional neural network architecture, we have trained a model on images of plant leaves with the goal of classifying both crop species and the presence and identity of disease on images that the model had not seen before. Within the Plant data set of 87,867 images containing 38 classes of 14 crop species and 26 diseases.

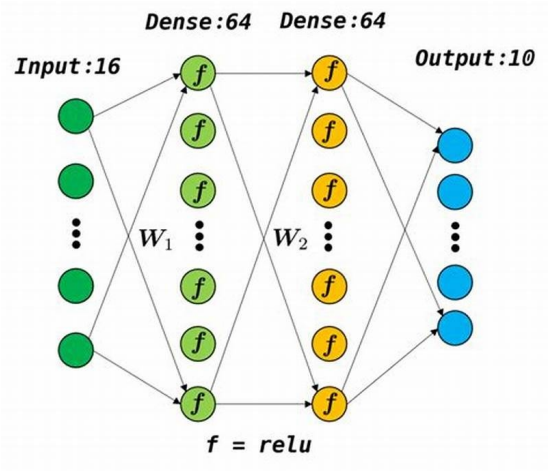
Here we have used some important libraries which includes **Tensorflow** and **Keras.**

TensorFlow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind. However, it proved to be very useful for deep learning development as well, and therefore Google open-sourced it.

TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors. Multi-dimensional arrays are very handy in handling large amounts of data.

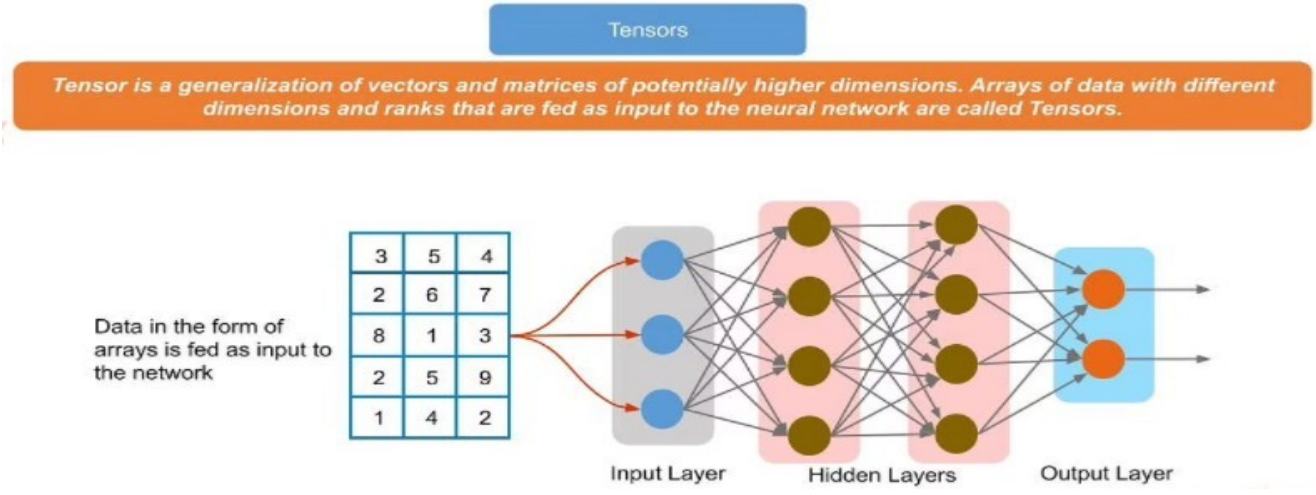
TensorFlow works on the basis of data flow graphs that have nodes and edges. As the execution mechanism is in the form of graphs, it is much easier to execute TensorFlow code in a distributed manner across a cluster of computers while using GPUs.





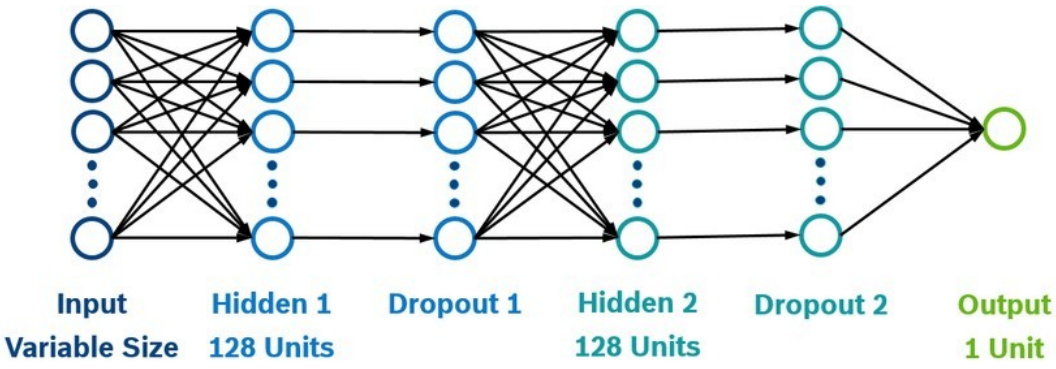
Tensor is a generalization of vectors and matrices of potentially higher dimensions. Arrays of data with varying dimensions and ranks that are fed as input to the neural network are called tensors.

For deep learning, especially in the training process, you will have large amounts of data that exist in a very complicated format. It helps when you are able to put, use, or store it in a compact way, which tensors provide, even if they appear in multi-dimensional arrays. When the data is stored in tensors and fed into the neural network, the output we get is as shown below:



Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.

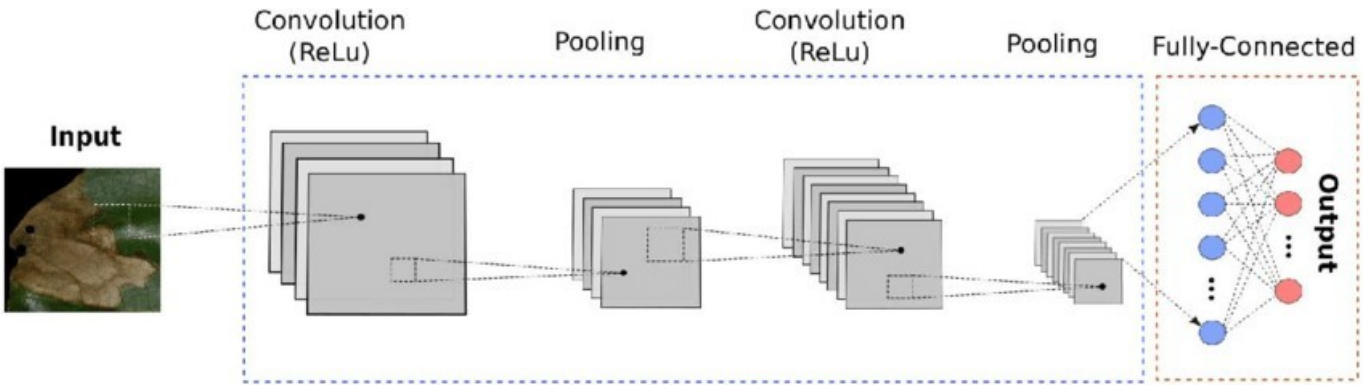
Keras empowers engineers and researchers to take full advantage of the scalability and cross-platform capabilities of TensorFlow 2: you can run Keras on TPU or on large clusters of GPUs, and you can export your Keras models to run in the browser or on a mobile device.



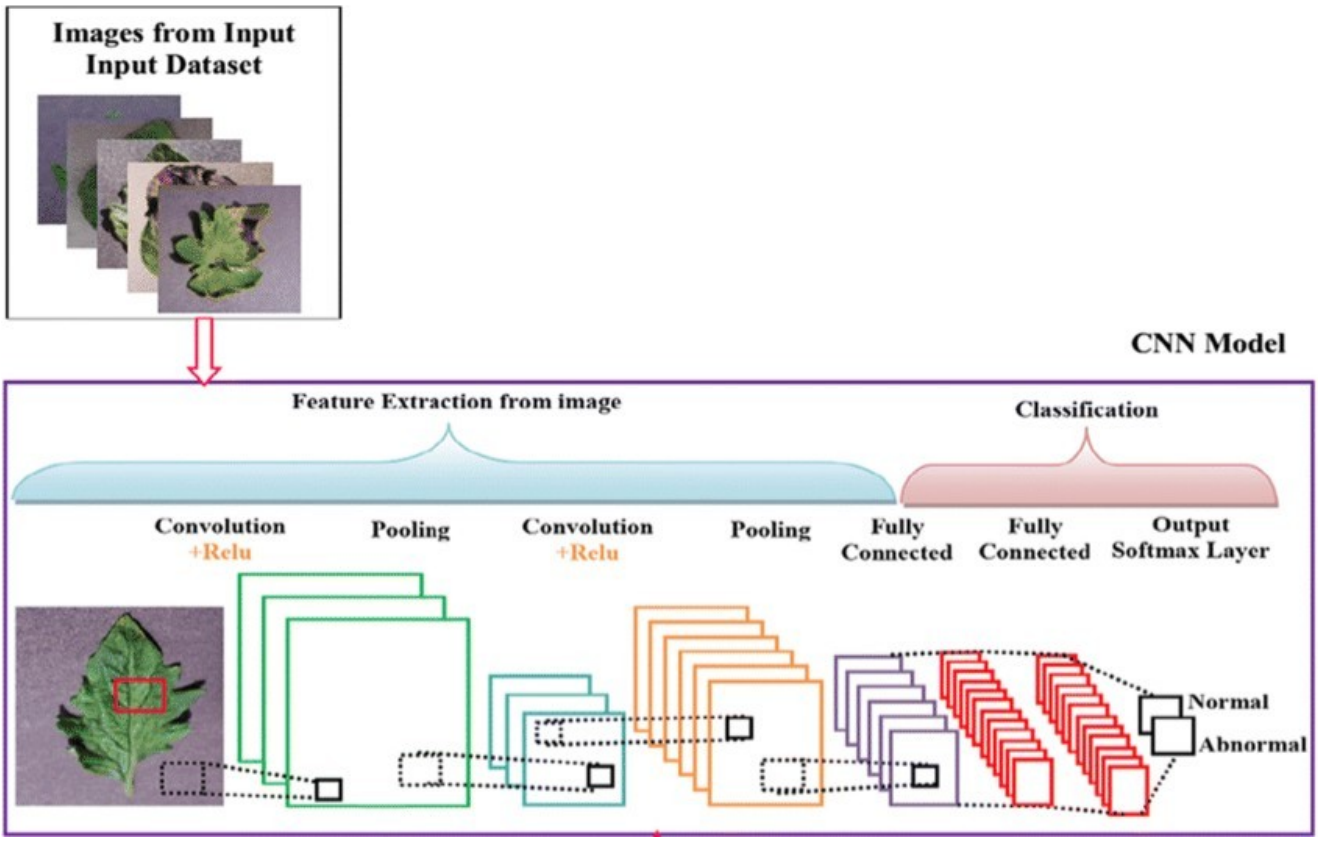
Keras Neural Network

My project uses **Convolutional Neural Network (CNN)** which is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics.

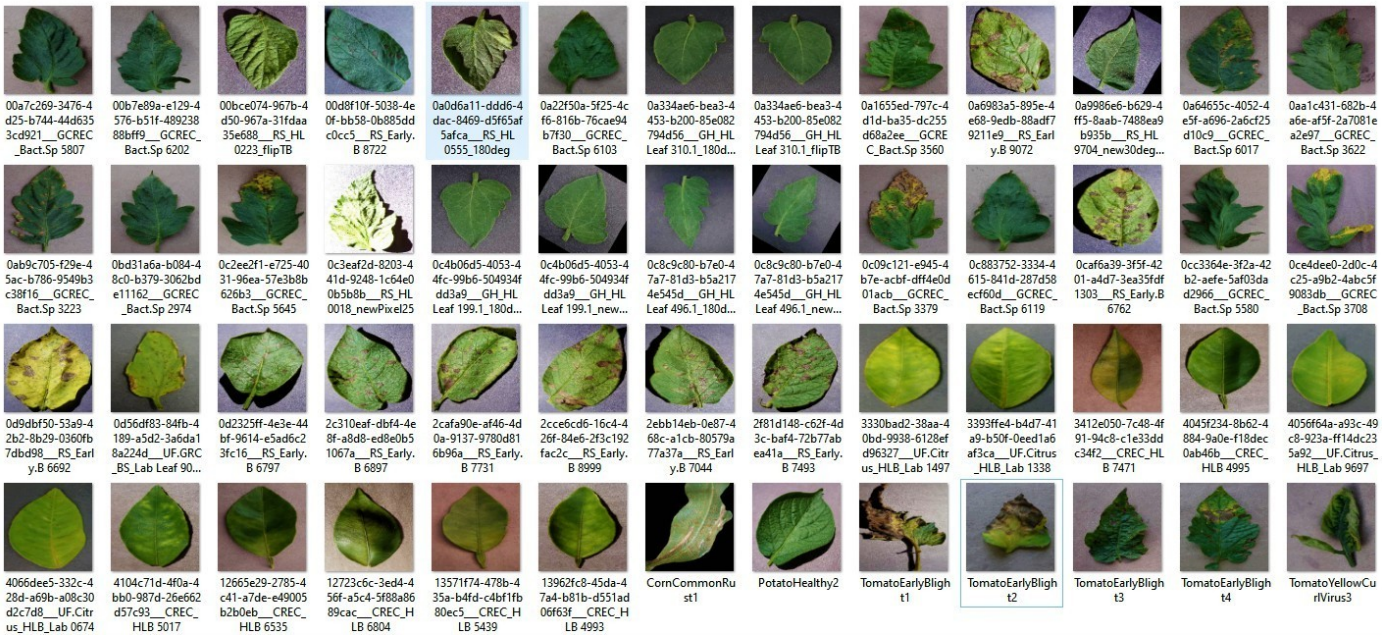
The role of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. I did Convolution operation on a 224x224x3 image matrix with a 3x3x3 Kernel.



More precise explanation is given below in the diagram:



Dataset Description



We have analysed 87,867 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. Figure 1 shows one example each from every crop-disease pair from the Plant dataset. In all the approaches described in this paper, we resize the images to 224 ×224 pixels and perform both the model optimization and predictions on these downscaled images.

Approach

We evaluate the applicability of deep convolutional neural networks for the classification problem using AlexNet architecture which follows the design as the LeNet-5 (LeCun et al., 1989) architecture from the 1990s. The LeNet-5 architecture variants are usually a set of stacked convolution layers followed by one or more fully connected layers. The convolution layers optionally may have a normalization layer and a pooling layer right after them, and all the layers in the network usually have ReLu non-linear activation units associated with them. AlexNet consists of 5 convolution layers, followed by 3 fully connected layers, and finally ending with a softMax layer. The first two convolution layers (conv {1, 2}) are each followed by a normalization and a pooling layer, and the last convolution layer (conv5) is followed by a single pooling layer. The final fully connected layer (fc8) has 38 outputs in our adapted version of AlexNet (equaling the total number of classes in our dataset), which feeds the softMax layer. The softMax layer finally exponentially normalizes the input that it gets from (fc8), thereby producing a distribution of values across the 38 classes that add up to 1. These values can be interpreted as the confidences of the network that a given input image is represented by the corresponding classes. All of the first 7 layers of AlexNet have a ReLu non-linearity activation unit associated with them, and the first two fully connected layers (fc {6, 7}) have a dropout layer associated with them.

Source Code

Importing necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as mp
import os
import tensorflow as tf
import keras
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
```

Exploratory Data analysis

In [2]:
```
os.listdir("D:\\plant diseases detection\\New Plant Diseases Dataset(Augmented)\\
 New Plant Diseases Dataset(Augmented)\\train")
```

Out [2]:
```
['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Blueberry___healthy',
 'Cherry_(including_sour)___healthy',
 'Cherry_(including_sour)___Powdery_mildew',
 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
 'Corn_(maize)___Common_rust_',
 'Corn_(maize)___healthy',
 'Corn_(maize)___Northern_Leaf_Blight',
 'Grape___Black_rot',
 'Grape___Esca_(Black_Measles)',
 'Grape___healthy',
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 'Orange___Haunglongbing_(Citrus_greening)',
 'Peach___Bacterial_spot',
 'Peach___healthy',
 'Pepper,_bell___Bacterial_spot',
 'Pepper,_bell___healthy',
 'Potato___Early_blight',
 'Potato___healthy',
 'Potato___Late_blight',
```

```
 'Raspberry___healthy',
 'Soybean___healthy',
 'Squash___Powdery_mildew',
 'Strawberry___healthy',
 'Strawberry___Leaf_scorch',
 'Tomato___Bacterial_spot',
 'Tomato___Early_blight',
 'Tomato___healthy',
 'Tomato___Late_blight',
 'Tomato___Leaf_Mold',
 'Tomato___Septoria_leaf_spot',
 'Tomato___Spider_mites Two-spotted_spider_mite',
 'Tomato___Target_Spot',
 'Tomato___Tomato_mosaic_virus',
 'Tomato___Tomato_Yellow_Leaf_Curl_Virus']
```

In [3]:
```
#Number of leaf diseases present in our data len(os.listdir("D:\\
 plant diseases detection\\New Plant Diseases
Dataset(Augmented)\\New Plant Diseases Dataset(Augmented)\\train"))
```

Out [3]: 38

Image preprocessing

In [4]:
```
#Loading our images to Data Generator for Training data and Validation data in order
to preprocess the images
 new_train_data=ImageDataGenerator(

     rescale=1/255.0,

     width_shift_range=0.2,

     height_shift_range=0.2,

     zoom_range=0.2,

     shear_range=0.2,

     fill_mode='nearest'

 )

    new_valid_data=ImageDataGenerator(rescale=1/255.0)
```

In [5]:
```
#Loading our images
 train= new_train_data.flow_from_directory(directory="D:\\plant diseases detection\\
New Plant Diseases Dataset(Augmented)\\New Plant Diseases Dataset(Augmented)\\
train",target_size=(224,224),batch_size=128,class_mode = 'categorical',color_mode="rgb",
shuffle=True)

 valid=new_valid_data.flow_from_directory(directory="D:\\plant diseases detection\\
New Plant Diseases Dataset(Augmented)\\New Plant Diseases Dataset(Augmented)\\
```

```
valid",target_size=(224,224),batch_size=128,class_mode = 'categorical',color_mode="rgb",
shuffle=True)
```

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

In [6]:
```
class_dict =
    list(train.class_indices.keys())
    print(class_dict)
```

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust',
'Apple___healthy', 'Blueberry___healthy', 'Cherry_(including_sour)___Powdery_mildew',
'Cherry_(including_sour)___healthy', 'Corn_(maize)___Cercospora_leaf_spot
Gray_leaf_spot', 'Corn_(maize)___Common_rust_', 'Corn_(maize)___Northern_Leaf_Blight',
'Corn_(maize)___healthy', 'Grape_____Black_rot', 'Grape_____
_____Esca_(Black_Measles)',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy',
'Orange___Haunglongbing_(Citrus_greening)', 'Peach___Bacterial_spot',
'Peach___healthy', 'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy',
'Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy',
'Raspberry___healthy', 'Soybean___healthy', 'Squash___Powdery_mildew',
'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___
_____Bacterial_spot', 'Tomato__Early_blight', 'Tomato_____Late_blight',
'Tomato__Leaf_Mold',
'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']
```

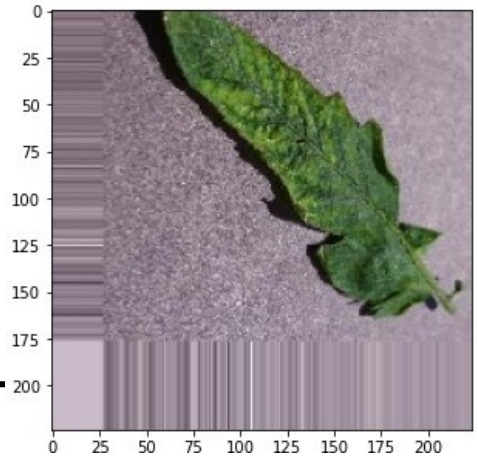
In [7]:
```
tst_img,label=train.next()
```

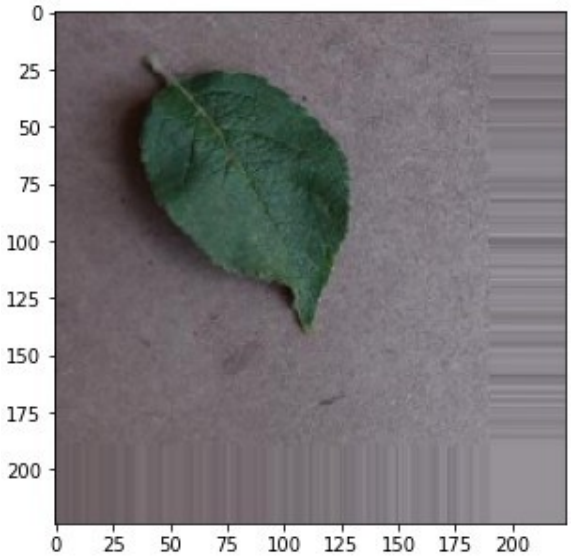
In [8]:
```
#the number of images, width, height and channels
    tst_img.shape
```

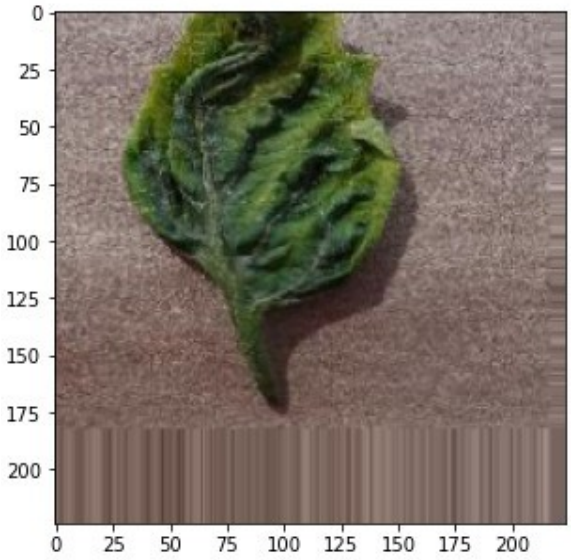
Out [8]:
```
(128, 224, 224, 3)
```

In [9]:
```
#plotting the some images
        def plotimag(img_arr,label):
        for im,l in zip(img_arr,label):
            mp.figure(figsize=(5,5))
            mp.imshow(im)
            mp.show()
```

In [10]:
```
plotimag(tst_img[:3],label[:3])
```







Building the model using AlexNet Architecture

In [11]: #importing necessary keras libraries and packages

```
from keras.models import Model,Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers.normalization import BatchNormalization
```

In [12]: # Initializing the CNN

```
classifier = Sequential()
# Convolution Step 1
classifier.add(Convolution2D(96, 11, strides = (4, 4), padding = 'valid', input_shape=
```

```
(224, 224, 3), activation = 'relu'))
# Max Pooling Step 1
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
classifier.add(BatchNormalization())
# Convolution Step 2
classifier.add(Convolution2D(256, 11, strides = (1, 1), padding='valid', activation = 'relu'))
# Max Pooling Step 2
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding='valid'))
classifier.add(BatchNormalization())

# Convolution Step 3
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid', activation = 'relu'))
classifier.add(BatchNormalization())
# Convolution Step 4
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid', activation = 'relu'))
classifier.add(BatchNormalization())
# Convolution Step 5
classifier.add(Convolution2D(256, 3, strides=(1,1), padding='valid', activation = 'relu'))
# Max Pooling Step 3
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
classifier.add(BatchNormalization())

# Flattening Step
classifier.add(Flatten())

# Full Connection Step
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 1000, activation = 'relu'))
classifier.add(Dropout(0.2))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 38, activation = 'softmax'))

# Model summary
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 27, 27, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       2973952
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
```

```
conv2d_2 (Conv2D)              (None, 6, 6, 384)         885120
batch_normalization_2 (Batch (None, 6, 6, 384)         1536
conv2d_3 (Conv2D)              (None, 4, 4, 384)         1327488
batch_normalization_3 (Batch (None, 4, 4, 384)         1536
conv2d_4 (Conv2D)              (None, 2, 2, 256)         884992
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0
batch_normalization_4 (Batch (None, 1, 1, 256)         1024
flatten (Flatten)              (None, 256)               0
dense (Dense)                  (None, 4096)              1052672
dropout (Dropout)              (None, 4096)              0
batch_normalization_5 (Batch (None, 4096)              16384
dense_1 (Dense)                (None, 4096)              16781312
dropout_1 (Dropout)            (None, 4096)              0
batch_normalization_6 (Batch (None, 4096)              16384
dense_2 (Dense)                (None, 1000)              4097000
dropout_2 (Dropout)            (None, 1000)              0
batch_normalization_7 (Batch (None, 1000)              4000
dense_3 (Dense)                (None, 38)                38038
=================================================================
Total params: 28,117,790
Trainable params: 28,096,654
Non-trainable params: 21,136
_________________________________________________________________
```

In [13]:
```
# let's visualize the layer names and layer indices to see how many layers
for i, layer in enumerate(classifier.layers[:20]):
    print(i, layer.name)
    layer.trainable = False
```

```
0 conv2d
1 max_pooling2d
2 batch_normalization
3 conv2d_1
4 max_pooling2d_1
5 batch_normalization_1
6 conv2d_2
7 batch_normalization_2
8 conv2d_3
9 batch_normalization_3
10 conv2d_4
11 max_pooling2d_2
12 batch_normalization_4
```

```
13 flatten
14 dense
15 dropout
16 batch_normalization_5
17 dense_1
18 dropout_1
19 batch_normalization_6
```

Loading the weights to the model

```
In [14]: classifier.load_weights('C:\\Users\\91910\\weights_dir\\best_weights_9.hdf5')
```

Compiling the model

```
In [15]: classifier.compile(optimizer=keras.optimizers.SGD(learning_rate=0.001,
  momentum=0.9,
  decay=0.005),loss=keras.losses.categorical_crossentropy,metrics=["accuracy"])
```

Early Stopping and Model Check point

```
In [16]: from keras.callbacks import ModelCheckpoint, EarlyStopping
    # early stopping
    early_stop=  EarlyStopping(monitor="val_accuracy",min_delta=0.01,patience=4,verbose=1  )

    # model check point
    weightpath = "best_weights_9.hdf5"
    model_check= ModelCheckpoint(weightpath
                        ,monitor="val_accuracy",
                        min_delta=0.01,
                        patience=4,
                        verbose=1,
                        save_best_only=True,
                        mode="max")

    #saving model
    filepath='best_model.h5'

    classifier.save(filepath)
    call_back=[early_stop,model_check]
```

Fitting images to CNN

```
In [17]: hist=classifier.fit_generator(train,
                        steps_per_epoch=30,
                        epochs=60,
                        verbose=1,
                        callbacks=call_back,
                        validation_data=valid,validation_steps=16
                         )
```

```
   WARNING:tensorflow:From
C:\Users\91910\AppData\Local\Temp/ipykernel_3368/1641481834.py:1:
Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated
and will be removed in a future version.
   Instructions for updating:
   Please use Model.fit, which supports generators.
   Epoch 1/60
   30/30 [==============================] - ETA: 0s - loss: 0.1111 - accuracy:
0.9599
   Epoch 00001: val_accuracy improved from -inf to 0.97119, saving model to
best_weights_9.hdf5
   30/30 [==============================] - 315s 10s/step - loss: 0.1111 -
accuracy: 0.9599 - val_loss: 0.0910 - val_accuracy: 0.9712
   Epoch 2/60
   30/30 [==============================] - ETA: 0s - loss: 0.0963 - accuracy:
0.9688
   Epoch 00002: val_accuracy did not improve from 0.97119
   30/30 [==============================] - 286s 10s/step - loss: 0.0963 -
accuracy: 0.9688 - val_loss: 0.0934 - val_accuracy: 0.9697
   Epoch 3/60
   30/30 [==============================] - ETA: 0s - loss: 0.1147 - accuracy:
0.9635
   Epoch 00003: val_accuracy did not improve from 0.97119
   30/30 [==============================] - 256s 9s/step - loss: 0.1147 - accuracy:
0.9635 - val_loss: 0.0904 - val_accuracy: 0.9707
   Epoch 4/60
   30/30 [==============================] - ETA: 0s - loss: 0.1065 - accuracy:
0.9628
   Epoch 00004: val_accuracy did not improve from 0.97119
   30/30 [==============================] - 237s 8s/step - loss: 0.1065 - accuracy:
0.9628 - val_loss: 0.0815 - val_accuracy: 0.9692
   Epoch 5/60
   30/30 [==============================] - ETA: 0s - loss: 0.0962 - accuracy:
0.9656
   Epoch 00005: val_accuracy did not improve from 0.97119
   30/30 [==============================] - 224s 7s/step - loss: 0.0962 - accuracy:
0.9656 - val_loss: 0.0930 - val_accuracy: 0.9702
   Epoch 00005: early stopping
```

Evaluating model

In [18]:
```
h=hist.history
h.keys()
#h_keys(['loss','accuracy','val_loss','val_accuracy'])
```

Out [8]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [19]:
```
from keras.models import load_model classifier=load_model("C:\\
Users\\91910\\best_model.h5")
```

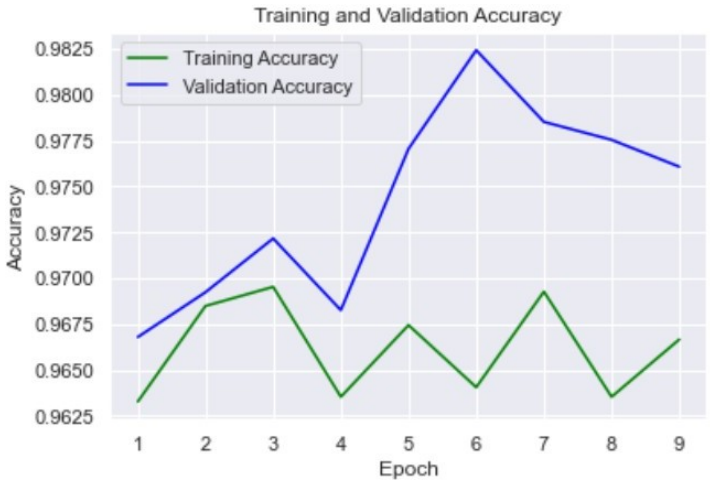
In [20]:
```
# Checking the accuracy of our model
acc=classifier.evaluate_generator(valid,verbose=1)[1]
print(f"The accuracy of your model is = {acc*100} %")
```

```
   WARNING:tensorflow:From
C:\Users\91910\AppData\Local\Temp/ipykernel_3368/2180682913.py:2:
```

```
Model.evaluate_generator (from tensorflow.python.keras.engine.training) is
deprecated and will be removed in a future version.
  Instructions for updating:
  Please use Model.evaluate, which supports generators.
  138/138 [==============================] - 307s 2s/step - loss: 0.1073 -
accuracy: 0.9633
  The accuracy of your model is = 96.32938504219055 %
```

Visualising Training progress

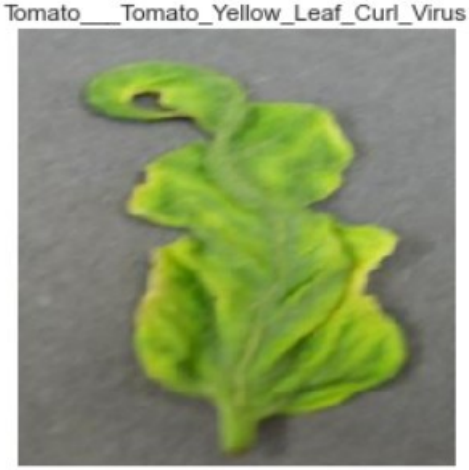
In [21]:
```
sns.set()
acc = h['accuracy']
val_acc = h['val_accuracy']
loss = h['loss']
val_loss = h['val_loss']
epochs = range(1, len(loss) + 1)
#accuracy plot
mp.plot(epochs, acc, color='green', label='Training Accuracy')
mp.plot(epochs, val_acc, color='blue', label='Validation Accuracy')
mp.title('Training and Validation Accuracy')
mp.ylabel('Accuracy')
mp.xlabel('Epoch')
mp.legend()
mp.figure()

# loss plot
mp.plot(epochs, loss, color='violet', label='Training Loss')
mp.plot(epochs, val_loss, color='red', label='Validation Loss')
mp.title('Training and Validation Loss')
mp.xlabel('Epoch')
mp.ylabel('Loss')
mp.legend()
mp.show()
```



Predicting the image

```
In [21]: from keras.preprocessing import image
         def predict(image_path):
            new_img = image.load_img(image_path, target_size=(224,
            224)) img = image.img_to_array(new_img)
            img = np.expand_dims(img,
            axis=0) img = img/255
            print("Following is our prediction:")
            prediction = classifier.predict(img)
         # Decoding the results into a list of tuples (class, description, probability)
         # (one such list for each sample in the batch)
            d = prediction.flatten()
            j = d.max()
            for index,item in enumerate(d):
               if item = = j:
                  class_name = class_dict[index]
         # Ploting image with predicted class name
            mp.figure(figsize = (4,4))
            mp.imshow(new_img)
            mp.axis('off')
            mp.title(class_name)
            mp.show()
         predict("C:\\Users\\91910\\test\\TomatoYellowCurlVirus5.JPG")
```

```
Following is our prediction:
```



Screenshots

Predicting the image

In [30]:
```
from keras.preprocessing import image

def predict(image_path):
    new_img = image.load_img(image_path, target_size=(224, 224))
    img = image.img_to_array(new_img)
    img = np.expand_dims(img, axis=0)
    img = img/255

    print("Following is our prediction:")
    prediction = classifier.predict(img)
# Decoding the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
    d = prediction.flatten()
    j = d.max()
    for index,item in enumerate(d):
        if item == j:
            class_name = class_dict[index]



# Ploting image with predicted class name
    mp.figure(figsize = (4,4))
    mp.imshow(new_img)
    mp.axis('off')
    mp.title(class_name)
    mp.show()
```

In [31]:
```
predict("C:\\Users\\91910\\test\\PotatoEarlyBlight1.JPG")
```

```
Following is our prediction:
```

In [32]:
```
predict("C:\\Users\\91910\\test\\AppleCedarRust2.JPG")
```

```
Following is our prediction:
```

In [33]:
```
predict("C:\\Users\\91910\\test\\TomatoYellowCurlVirus4.JPG")
```

```
Following is our prediction:
```

In [ ]:

ADVANTAGES

- Plant leaf disease recognition using deep learning include high processing speed and high classification accuracy.
- After extensive training on diverse datasets our machine learning model will be capable of distinguishing a large number of different diseases.
- Use of estimators for automatic Initialization of cluster centres so there is no need of user input at the time of segmentation.
- Proposed method is fully automatic while existing methods require user input to select the best segmentation of input image.
- It also provides environment friendly recovery measures of the identified disease.

FUTURE SCOPE

A plant disease recognition system can work as a universal detector, recognizing general abnormalities on the leaves, such as scorching or mold. Automated computational systems for the detection and diagnosis of plant diseases assist farmers and agronomists with their high throughput and precision.
For example –

A. Potato Leaf.

Fig. 1.1

Fig. 1.2

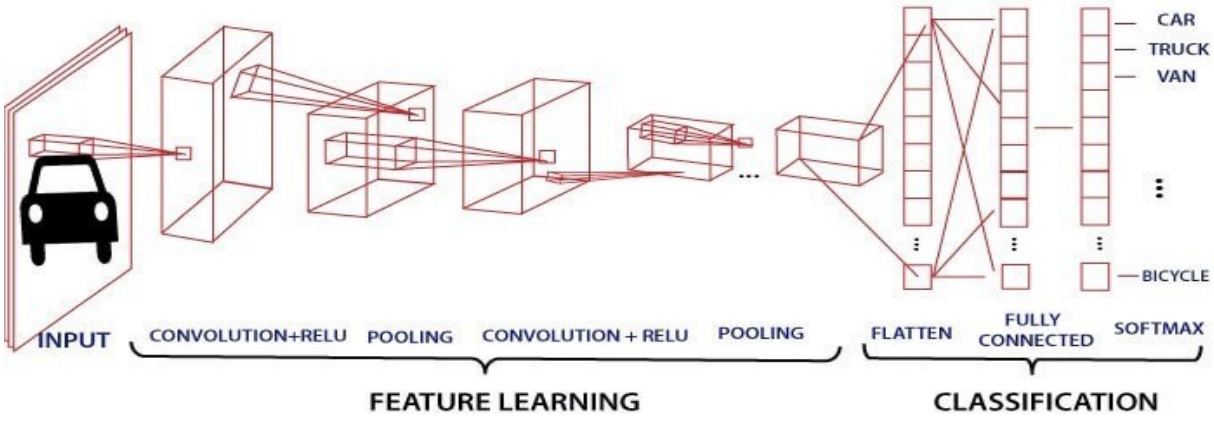
Fig. 1.1 – Shows a healthy potato leaf.
Fig. 1.2 – Shows early blight of potato leaf, which is generally caused by the fungus *Alternaria solani*. Symptoms first appear on the lower, older leaves as small brown spots with concentric rings that form a “bull’s eye” pattern.

B. Tomato Leaf.

Fig. 2.1

Fig. 2.2

Fig. 2.1 – Shows a healthy tomato leaf.
Fig. 2.2 – Shows a tomato leaf which is infected by Yellow Leaf Curl Virus.
Tomato yellow leaf curl virus (TYLCV) is a DNA virus from the genus *Begomovirus* and the family *Geminiviridae*. This virus is transmitted by an insect vector from the family *Aleyrodidae* and order *Hemiptera*, the whitefly *Bemisia tabaci*, commonly known as the silver leaf whitefly or the sweet potato whitefly.



Most deep learning-based solutions presented in literature have relied on well-known CNN architectures such as AlexNet, GoogLeNet, VGG, ResNet and Inceptionv3. This can be attributed mainly to the lack of sufficiently large datasets to train customized CNN architectures for plant disease recognition from scratch. Well known architectures provide a way around the problem of insufficient amounts of training data through the use of transfer learning technique. However, these well-known CNN models are very computationally demanding and overly complicated for the plant disease recognition task. Compact CNN models will be highly desirable especially in embedded, robotic and mobile applications where real-time performance and low computational cost are required.

CONCLUSION

A large part of the Indian population relies on agriculture, hence it becomes very essential to detect and recognize the leaf diseases that results in losses, since agriculture is critical to the growth of the economy. This project is based on deep learning approach called CNN which is utilized to build 38 different plant leaf disease identification, detection and recognition system. This approach utilized a minimum set of layers to identify the diseases of fourteen classes. The neural network is trained with Plant Village dataset. This model allows the user to choose the images from the dataset. User can select any image from the dataset and the image gets loaded, following which the prediction of the disease will be shown to the user. Convolutional neural network, is trained for identifying and recognizing the plant leaf disease, could classify and predict the diseases correctly for almost all the images with few anomalies thus and obtained 96.3% accuracy.