A

Project Report

ON

Plant Leaf Disease Detection Using Deep Learning

Submitted to

Rajiv Gandhi University of Knowledge Technologies,

RK Valley, KADAPA.

in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

A.SUBBA RAYUDU	**R170979**
T.SAI YOGESH	**R170268**
O.VENKATA KRISHNAIAH	**R170452**

Under the Guidance of

B.Madhan Mohan,M.Tech

Assistant Proffesor,H.O.D. of E.C.E.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES,

RK VALLEY, KADAPA (DIST.), ANDHRA PRADESH, PINCODE -516330.

December – 2022.

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES,

RK VALLEY, KADAPA (DIST.), ANDHRA PRADESH,

PINCODE -516330.December – 2022.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project report entitled **"Plant Leaf Disease Detection Using Deep Learning "** a bonafide record of the project work done and submitted by

T.SAI YOGESH	**R170268**
O.VENKATA KRISHNAIAH	**R170452**
A.SUBBA RAYUDU	**R170979**

for the partial fulfillment of the requirements for the award of B.Tech. Degree in **ELECTRONICS AND COMMUNICATION ENGINEERING,** RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES , RK VALLEY.

Project Internal GUIDE

Mr. B.MADHAN MOHAN ,
Head of Department,
Department of E.C.E.,
RGUKT, RK Valley,KADAPA
A.P.,PINCODE :516330.

Head of the Department

Mr. B.MADHAN MOHAN ,
Head of Department,
Department of E.C.E.,
RGUKT, RK Valley, KADAPA
A.P.,PINCODE :516330.

External Viva-Voce Exam Held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We hereby declare that the project report entitled "**Plant Leaf Disease Detection By Using Deep Learning**" submitted to the Department of **ELECTRONICS AND COMMUNICATION ENGINEERING** in partial fulfillment of requirements for the award of the degree of **BACHELOR OF TECHNOLOGY.** This project is the result of our own effort and that it has not been submitted to any other University or Institution for the award of any degree or diploma other than specified above.

By,

A.SUBBA RAYUDU	R170979
T.SAI YOGESH	R170268
O.VENKATA KRISHNAIAH	R170452

ACKNOWLEDGEMENTS

We are thankful to our guide **Mr.B.Madhan Mohan** for his valuable guidance and encouragement. His helping attitude and suggestions have helped us in the successful completion of the project.

We would like to express our gratefulness and sincere thanks to **Mr.B.Madhan Mohan**, Head of the Department of **ELECTRONICS AND COMMUNICATION ENGINEERING**, for his kind help and encouragement during the course of our study and in the successful completion of the project work.

We have great pleasure in expressing our hearty thanks to our beloved Director **Dr. SANDHYA RANI** for spending her valuable time with us to complete this project.

Successful completion of any project cannot be done without proper support and encouragement. We sincerely thanks to the Management for providing all the necessary facilities during the course of study.

We would like to thank our parents and friends, who have the greatest contributions in all our achievements, for the great care and blessings in making us successful in all our endeavors.

By,

A.SUBBA RAYUDU R170979

T.SAI YOGESH R170268

O.VENKATA KRISHNAIAH R170452

ABSTRACT

Since we are interested on searching for new areas and exploring new things and in Electronics World especially related to the area of Image processing using technologies Machine learning /Deep Learning .As We want to enhance and to Build our career, from the Machine learning Domain which is a part in Software industry Companies This research that made by us helps us to gain Confidence and research further in the field of Domain Deep Learning Neural Networks.The Main Objective of Our Project is "**Plant Leaf Disease Classification By using the Deep Learning Neural Network Layer Model** "With Good prediction Accuracy.

The major reason for minimizing crop productivity is various diseases in plants. To eliminate the disease-induced losses in plants during growth as well as to increase crop productivity, former disease detection and prevention on the crop are the most challenging factors. Thus, it is a suitable decision that can be taken by the farmers or villagers to avoid further losses. The project works on the technique of image processing which identifies the various diseases in plants. Here we use an efficient **convolutional neural network algorithm (CNN) algorithm** which can detect the type of diseases in various leaves.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABSERVATION

CNN	CONVOLUTIONAL NEURAL NETWORK
LRS	LEARNING RATE SCHEDULING
CPU	CONTROL PROCESSING UNIT
GPU	GRAPHICS PROCESSING UNIT
ResNet	Residual Networks
Relu	Rectified linear Activation Unit

1. INTRODUCTION

One of the major sources in India is the production of crops. It is of enhancing the technological advancement in the fields related to crop productivity for formers.Commonly humans are work sufficiently with minimum time with greater outcome.The detection of plant disease by human visualization is a more difficult task and at the same time, less efficient, and it's done with a limited set of leaf images and takes more time.Whereas the automatic identification technique will take less effort and time and a more accurate program. Here we use image processing to detect the diseases. We can put the image into a system and a computer can perform various phases for identification and detect the related classes to which that image belongs. This work aims to make a leaf recognition technique based on the specific characteristics derived from images.

The implementation model was developed with a system consisting of higher system requirements.Here users have to give leaf images and the specified thing in this model is users must install the various packages like TensorFlow, Opencv, Keras, etc. The existing system consisting of users has detected the diseases which present in the plant leaves.

1.1.Block Diagram:-

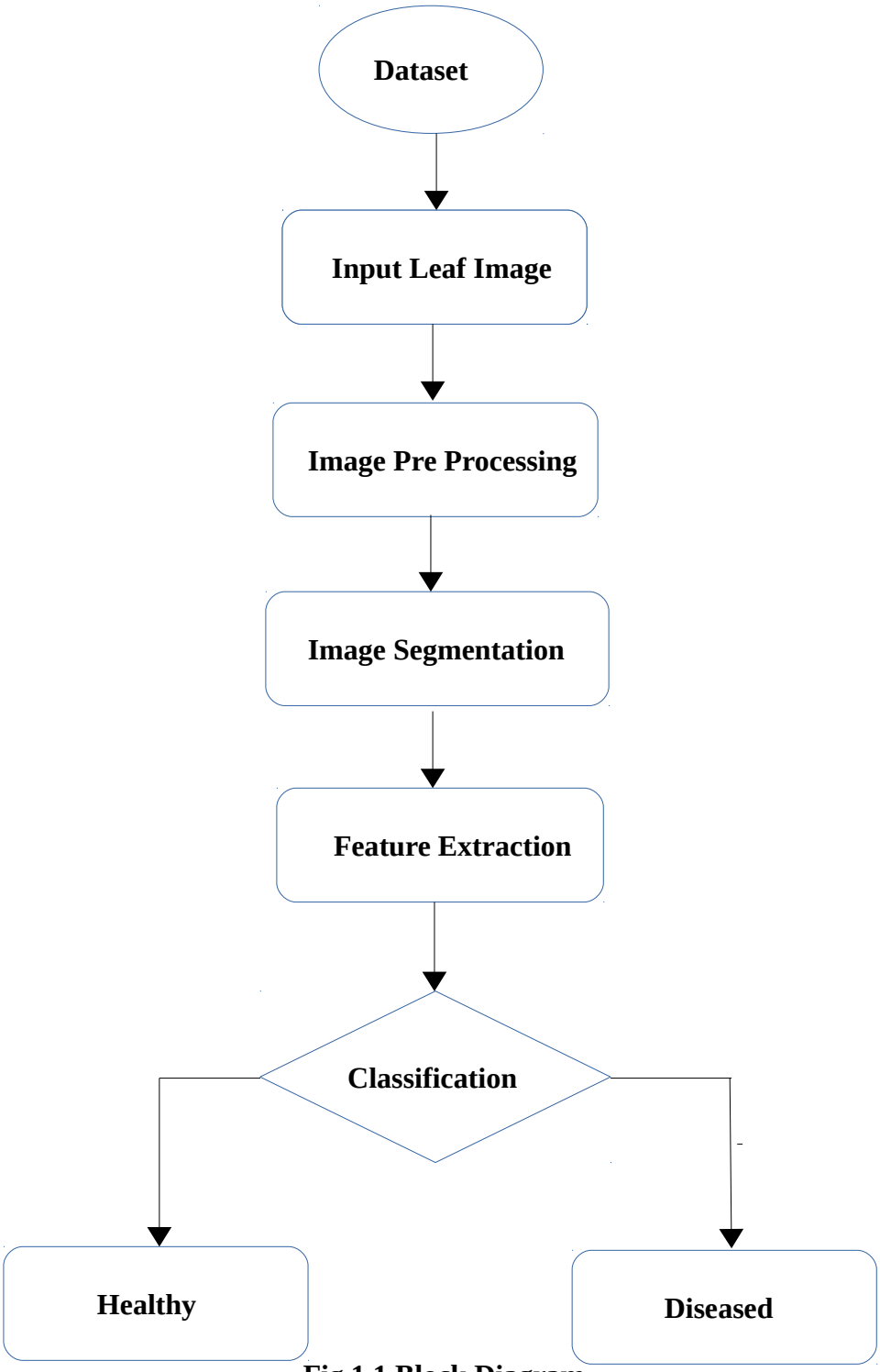


Fig.1.1.Block Diagram

2. DATASET

This is a set of images for specified purposes. We use a plant leaf dataset and each of them is divided for preprocessing and classification. The Leaf dataset consists of more than 1000 images of various plants. This contains both healthy and diseased leaves. The diseased class includes in name of the specified disease and provides remedies for overcoming the deficiency. Here we train the large dataset and detect the disease present on each leaf.Click Here

2.1.Data Set Understanding

The Data Set contains 38 classes of layers and The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose.

2.1.1 Importing DataSet

Importing the data set from Google drive to the Google Colab Online Notebook which supports to Graphics Processing Unit (GPU) to train our models faster if our execution platform is connected to a GPU manufactured by NVIDIA.

```
>>from google.colab import drive

>>drive.mount('/content/data')
```

OUTPUT:-Mounted at /content/data

Unziping the DataSet from the Google Drive file archeive.zip

```
>> !unzip -q "/content/data/MyDrive/archive.zip"
```

2.1.2.Importing Important Libraries

```
>>import os   # for working with files in System

>>import numpy as np # for numerical computationss
```

```
>>import pandas as pd  # for working with dataframes
>>import torch  # Pytorch module is an open source machine learning torch library used for
   developing and training neural network based deep leaning models.
>>import matplotlib.pyplot as plt # for plotting informations on graph and images using tensors
>>import torch.nn as nn # for creating neural networks
>>from torch.utils.data import DataLoader # for dataloaders
>>from PIL import Image # for checking images
>>import torch.nn.functional as F # for functions for calculating loss
>>import torchvision.transforms as transforms # for transforming images into tensors
>>from torchvision.utils import make_grid # for data checking
>>from torchvision.datasets import ImageFolder # for working with classes and images
>>from torchsummary import summary # for getting the summary of our model.
```

2.1.3.Spliting the Dataset and Checking Dataset

The dataset folder contains 2 dataset images with helathy and diseased of 38 class layers.Dataset divided into 80% of data for the training model purpose and 20% data Images for validatining the model.

```
>>data_dir = "/content/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset”
>>train_dir = data_dir + "/train"
>>valid_dir = data_dir + "/valid"
>>diseases = os.listdir(train_dir)
```

```
# printing the disease names and Number of classes

>>print(diseases)

>>print("Total disease classes are: {}".format(len(diseases)))
```

Output:-Total disease classes are:38

['Tomato___healthy','Pepper,_bell___healthy','Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Apple___ Cedar _apple_rust', 'Apple___Apple_scab', 'Cherry_(including_sour)___healthy', 'Straw berry___healthy', 'Potato___Late_blight', 'Apple___healthy', 'Tomato___Late_blight', 'Tomato___Septoria_leaf_spot', 'Tomato___Early_blight', 'Straw berry___Leaf_ scorch', 'Corn_(maize)___healthy', 'Blueberry___healthy', 'Tomato___Bacterial_spot', 'Peach___Bacterial_spot', 'Orange___Haunglongbing_(Citrus_greening)', 'Squash ___Powdery_mildew', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus', 'Rasp berry ___healthy', 'Tomato___Target_Spot', 'Corn_(maize)___Northern_Leaf_Blight', 'Potato___healthy', 'Soybean___healthy', 'Corn_(maize)___Common_rust_', 'Pepper,_bell___Bacterial_spot', 'Tomato___Spider_mites Two-spotted_spider_mite', 'Tomato___Tomato_mosaic_virus', 'Potato___Early_blight', 'Peach___healthy', 'Grape___Esca_(Black_Measles)', 'Grape___healthy', 'Tomato___Leaf_Mold', 'Cherry_(including_sour)___Powdery_mildew', 'Apple___Black_rot', 'Grape ___ Black _rot']

```
#printing the number of unique plants and Number of Diseases

>>plants = []

>>NumberOfDiseases = 0

>>for plant in diseases:

>>if plant.split('___')[0] not in plants:

>>plants.append(plant.split('___')[0])
```

>>if plant.split('___')[1] != 'healthy':

>>NumberOfDiseases += 1

#unique plants in dataset

>>print(f"Unique Plants are: \n{plants}")

Output:-Unique Plants are: ['Tomato', 'Pepper,_bell', 'Grape', 'Corn_(maize)', 'Apple', 'Cherry_(including_sour)', 'Strawberry', 'Potato', 'Blueberry', 'Peach', 'Orange', 'Squash', 'Raspberry', 'Soybean']

number of unique plants

>>print("Number of plants: {}".format(len(plants)))

Output:-Number of plants: 14

number of unique diseases

>>print("Number of diseases: {}".format(NumberOfDiseases))

Output:- Number of diseases: 26

Number of images for each disease

>>nums = {}

>>for disease in diseases:

>>nums[disease] = len(os.listdir(train_dir + '/' + disease))

converting the nums dictionary to pandas dataframe passing index as plant name and number of images as column

>>img_per_class = pd.DataFrame(nums.values(), index=nums.keys(), columns=

["no. Of images"])

>>img_per_class

Table.*2.1.3.Spliting the Dataset and Checking Dataset*

	no. of images
Tomato___healthy	1926
Pepper,_bell___healthy	1988
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1722
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	1642
Apple___Cedar_apple_rust	1760
Apple___Apple_scab	2016
Cherry_(including_sour)___healthy	1826
Strawberry___healthy	1824
Potato___Late_blight	1939
Apple___healthy	2008
Tomato___Late_blight	1851
Tomato___Septoria_leaf_spot	1745
Tomato___Early_blight	1920
Strawberry___Leaf_scorch	1774
Corn_(maize)___healthy	1859
Blueberry___healthy	1816
Tomato___Bacterial_spot	1702
Peach___Bacterial_spot	1838
Orange___Haunglongbing_(Citrus_greening)	2010
Squash___Powdery_mildew	1736
Tomato___Tomato_Yellow_Leaf_Curl_Virus	1961
Raspberry___healthy	1781
Tomato___Target_Spot	1827
Corn_(maize)___Northern_Leaf_Blight	1908
Potato___healthy	1824
Soybean___healthy	2022
Corn_(maize)___Common_rust_	1907
Pepper,_bell___Bacterial_spot	1913
Tomato___Spider_mites Two-spotted_spider_mite	1741
Tomato___Tomato_mosaic_virus	1790
Potato___Early_blight	1939
Peach___healthy	1728
Grape___Esca_(Black_Measles)	1920
Grape___healthy	1692
Tomato___Leaf_Mold	1882
Cherry_(including_sour)___Powdery_mildew	1683
Apple___Black_rot	1987
Grape___Black_rot	1888

2.1.4.Visuvalizating number of images available for each disease.

```
>>index = [n for n in range(38)]
>>plt.figure(figsize=(20, 5))
>>plt.bar(index, [n for n in nums.values()], width=0.3)
>>plt.xlabel('Plants/Diseases', fontsize=10)
>>plt.ylabel('No of images available', fontsize=10)
>>plt.xticks(index, diseases, fontsize=5, rotation=90)
>>plt.title('Images per each class of plant disease')
```

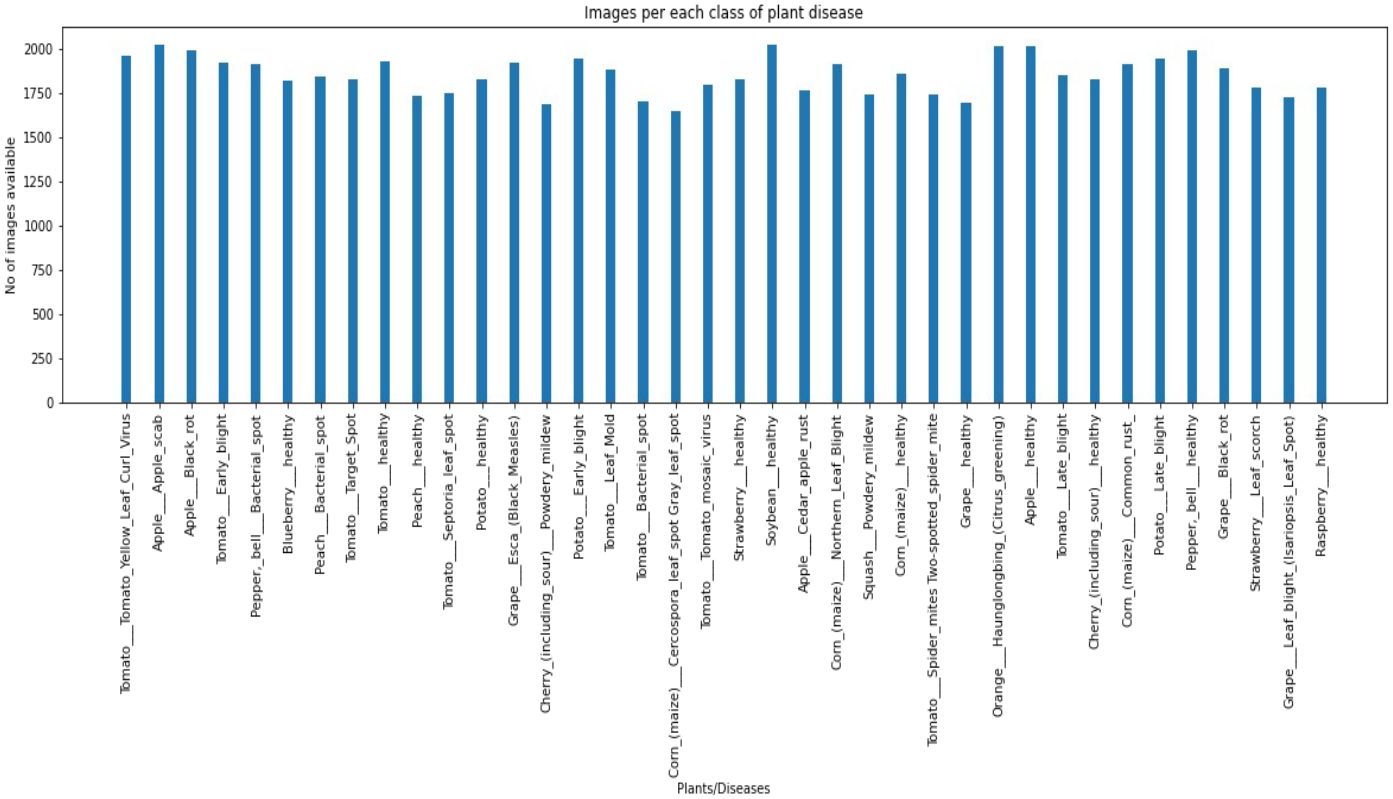


Fig.*2.1.4.Visuvalizating number of images available for each disease.*

2.1.5.Images for Training Model

```
>>n_train = 0

>>for value in nums.values():

>>n_train += value

>>print(f"There are {n_train} images for training")
```

Output:-There are 70295 images for training

3.IMAGE PREPROCESSING

Preprocessing of leaves is bringing all the image sizes to be reduced uniform size like 50*50 resolution. The main motive of this step is to remove the noise or other unwanted objects from the image.

3.1.Data Preparation for Training

```
# datasets for validation and training

>>train = ImageFolder(train_dir, transform=transforms.ToTensor())

>>valid = ImageFolder(valid_dir, transform=transforms.ToTensor())

#imagefolder used for when the dataset images in 1 row only
```

Now we transform the pixel values of each image (0-255) to 0-1 as neural networks works quite good with normalized data. The entire array of pixel values is converted to torch tensor and then divided by 255**.**

```
>>img, label = train[0]

>>print(img.shape, label)

Output:-torch.Size([3, 256, 256]) 0

# shape (3, 256 256) of the image. 3 is the number of channels (RGB) and 256 x 256 is the width and height of the image

# total number of classes in train set

>>len(train.classes)

Output:-38
```

4.IMAGE SEGMENTATION

Segmentation is a phase of image processing it segments the leaves into several parts and derives useful and meaningful information from the data. It derives the leaves based on leaf perimeter, shape, region edge, threshold, feature, and model. There are different types of segmentation techniques are available here we use neural network-based segmentation.

4.1.Checking some images from training dataset

```
>>def show_image(image, label):

>>print("Label :" + train.classes[label] + "(" + str(label) + ")")

>>plt.imshow(image.permute(1, 2, 0))

#Some Images from training dataset

>>show_image(*train[0])
```

Output:-Label : Apple___Apple_scab(0)

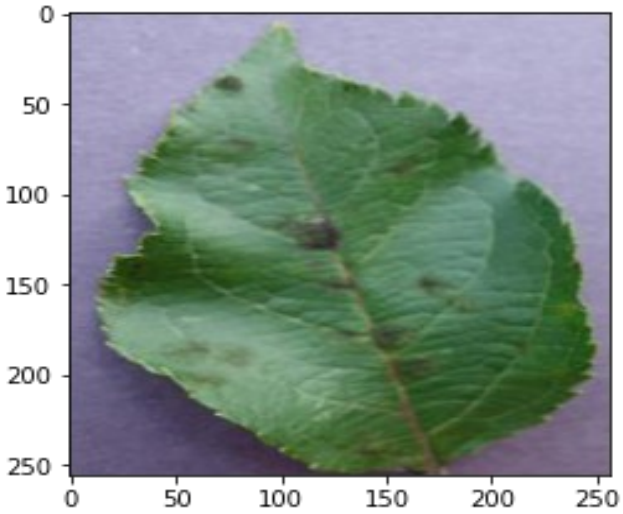


Fig.4.1.Checking some images from training dataset

>>show_image(*train[70000])

Output:-Label :Tomato___healthy(37)

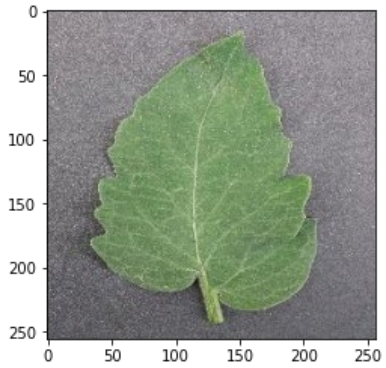


Fig.4.1.Checking some images from training dataset

4.2.Setting Seed value and Batch Size

>>random_seed = 7 # Setting the seed value

>>torch.manual_seed(random_seed)

>>batch_size = 32 # setting the batch size

batch_size is the total number of images given as input at once in forward propagation of the CNN. Basically, batch size defines the number of samples that will be propagated through the network.we have 1050 training samples and you want to set up a batch_size equal to 32. The algorithm takes the first 32 samples (from 1st to 32th) from the training dataset and trains the network. Next, it takes the second 32 samples (from 32st to 64th) and trains the network again. We can keep doing this procedure until we have propagated all samples through of the network.

DataLoaders for training and validation

>>train_dl = DataLoader(train, batch_size, shuffle=True, num_workers=2, pin_memory=True)

>>valid_dl=DataLoader(valid,batch_size, num_workers=2,pin_memory=True)

DataLoader is a subclass which comes from torch.utils.data. It helps in loading large and memory consuming datasets. It takes in batch_size which denotes the number of samples contained in each generated batch.

Setting **shuffle=True** shuffles the dataset. It is heplful so that batches between epochs do not look alike. Doing so will eventually make our model more robust.

num_workers, denotes the number of processes that generate batches in parallel. If you have more cores in your CPU, you can set it to number of cores in your CPU. Since, Kaggle provides a 2 core CPU, I have set it to 2.

4.3. Show a batch of training instances

```
>>def show_batch(data):

>>for images, labels in data:

>>fig, ax = plt.subplots(figsize=(30, 30))

>>ax.set_xticks([]); ax.set_yticks([])

>>ax.imshow(make_grid(images, nrow=8).permute(1, 2, 0))

>>break

>>show_batch(train_dl)
```

Fig.4.3. Show a batch of training instances

5.FEATURE EXTRACTION

CNN algorithm contains various layers which provide feature extraction and further classification of images. The key role of feature extraction in plant disease identification is to learn the features automatically. The basic geometrical features are derived in this step. Feature extracted based on the parameters like diameter, width, leaf area, leaf perimeter, morphological features, shape, texture,rectangular, etc.

5.1.CNN Algorithm Modelling:-

CNN Modeling advisable to use GPU instead of CPU when dealing with images dataset because CPUs are generalized for general purpose and GPUs are optimized for training deep learning models as they can process multiple computations simultaneously. They have a large number of cores,which allows for better computation of multiple parallel processes. Additionally, computations in deep learning need to handle huge amounts of data — this makes a GPU's memory bandwidth most suitable. To seamlessly use a GPU, if one is available, we define a couple of helper functions (get_default_device & to_device) and a helper class DeviceDataLoader to move our model & data to the GPU as required.

```
# for moving data into GPU (if available)
>>def get_default_device():
 #Pick GPU if available, else CPU
 >>if torch.cuda.is_available:
 >>return torch.device("cuda")
 >>else:
 >>return torch.device("cpu")
```

```
# for moving data to device (CPU or GPU)
>>def to_device(data, device):
#Move tensor(s) to chosen device
>>if isinstance(data, (list,tuple)):
>>return [to_device(x, device) for x in data]
>>return data.to(device, non_blocking=True)
# for loading in the device (GPU if available else CPU)
>>class DeviceDataLoader():
#Wrap a dataloader to move data to a device
>>def __init__(self, dl, device):
>>self.dl = dl
>>self.device = device
>>def __iter__(self):
#Yield a batch of data after moving it to device
>>for b in self.dl:
>>yield to_device(b, self.device)
>>def __len__(self):
>>#Number of batches
>>return len(self.dl)
```

5.1.1.Moving Data from CPU to GPU

>>device = get_default_device()

>>device

Output :- device(type='cuda')

Wrap up our training and validation data loaders using Device Data Loader for automatically transferring batches of data to the GPU (if available)

Moving data into GPU

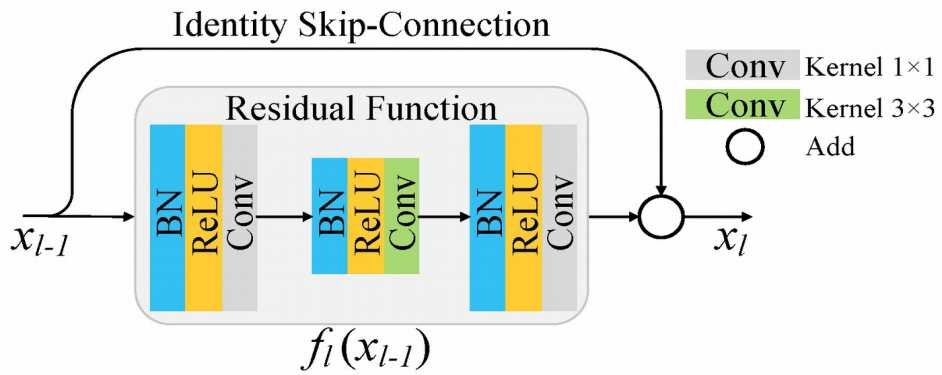
>>train_dl = DeviceDataLoader(train_dl, device)

>>valid_dl = DeviceDataLoader(valid_dl, device)

5.1.2.Bulding Model Architecture

We are using ResNets introduced Opencv Methodologies, unlike in traditional neural networks, each layer feeds into the next layer, we use a network with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away, to avoid over-fitting (a situation when validation loss stop decreasing at a point and then keeps increasing while training loss still decreases). This also helps in preventing vanishing gradient problem and allow us to train deep neural networks.

Fig.*5.1.2.Bulding Model Architecture*



5.1.3.Residual Block code implementation

```
>>class SimpleResidualBlock(nn.Module):

>>def __init__(self):

>>super().__init__()

>>self.conv1 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1,

padding=1)
```

6.CNN Neural Network Classification

Classification is a process of placing each of the images under specified classes. The classification is a step in which it compares various values received after the feature extraction, and it classifies the input leaf is diseased or healthy, to establish efficient relation analysts use data. Here, we classify four categories as leaf images. If the resulting status of the leaf is diseased it provides the remedies for removing the deficiency.

6.1.Calculating Accuracy

Training_step - To figure out how “wrong” the model is going after training or validation step.We are using this function other than just an accuracy metric that is likely not going to be differentiable (this would mean that the gradient can’t be determined, which is necessary for the model to improve during training).

Validation_step - Because an accuracy metric can’t be used while training the model, doesn’t mean it shouldn’t be implemented! Accuracy in this case would be measured by a threshold, and counted if the difference between the model’s prediction and the actual label is lower than that threshold.

Validation_epoch_end - We want to track the validation losses/accuracies and train losses after each epoch, and every time we do so we have to make sure the gradient is not being tracked.

Epoch_end - We also want to print validation losses/accuracies, train losses and learning rate too because we are using learning rate scheduler (which will change the learning rate after every batch of training) after each epoch.

An accuracy function is calculates the overall accuracy of the model on an entire batch of outputs, so that we can use it as a metric in fit_one_cycle.

```
>>def accuracy(outputs, labels):
>>_, preds = torch.max(outputs, dim=1)
>>return torch.tensor(torch.sum(preds == labels).item() / len(preds))# base class for
themodel
>>class ImageClassificationBase(nn.Module):
>>def training_step(self, batch):
>>images, labels = batch
>>out = self(images)# Generate predictions
>>loss = F.cross_entropy(out, labels) # Calculate loss
>>return loss
>>def validation_step(self, batch):
>>images, labels = batch
>>out = self(images) # Generate prediction
>>loss = F.cross_entropy(out, labels) # Calculate loss
>>acc = accuracy(out, labels)# Calculate accuracy
>>return {"val_loss": loss.detach(), "val_accuracy": acc}
>>def validation_epoch_end(self, outputs):
>> batch_losses = [x["val_loss"] for x in outputs]
>>batch_accuracy = [x["val_accuracy"] for x in outputs]
>>epoch_loss = torch.stack(batch_losses).mean() # Combine loss
```

```
>>epoch_accuracy = torch.stack(batch_accuracy).mean()
>>return {"val_loss": epoch_loss, "val_accuracy": epoch_accuracy}
# Combine accuracies
>>def epoch_end(self, epoch, result):
>>print("Epoch [{}], last_lr: {:.5f}, train_loss: {:.4f}, val_loss: {:.4f}, val_acc:
{:.4f}".format(epoch,result['lrs'][-1],result['train_loss'],result['val_loss'], result
['val_accuracy']))
```

6.2 Convolution block with BatchNormalization

```
>>def ConvBlock(in_channels, out_channels, pool=False):
>> layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1), nn.Batch
Norm2d(out_channels),nn.ReLU(inplace=True)
>>if pool:
>>layers.append(nn.MaxPool2d(4))
>>return nn.Sequential(*layers)
```

6.3.Resnet architecture

```
>>class ResNet9(ImageClassificationBase):
>>def __init__(self, in_channels, num_diseases):
>>super().__init__()
>>self.conv1 = ConvBlock(in_channels, 64)
>>self.conv2 = ConvBlock(64, 128, pool=True) # out_dim : 128 x 64 x 64
```

>>self.conv3 = ConvBlock(128, 256, pool=True) # out_dim : 256 x 16 x 16

>>self.res1 = nn.Sequential(ConvBlock(128, 128), ConvBlock(128, 128))

>>self.conv4 = ConvBlock(256, 512, pool=True) # out_dim : 512 x 4 x 44

>>self.res2 = nn.Sequential(ConvBlock(512, 512), ConvBlock(512, 512))

>>self.classifier = nn.Sequential(nn.MaxPool2d(4),nn.Flatten(),nn.Linear(512,

num _diseases))

>>def forward(self, xb): # xb is the loaded batch

>>out = self.conv1(xb)

>>out = self.conv2(out)

>>out = self.res1(out) + out

>>out = self.conv3(out)

>>out = self.conv4(out)

>>out = self.res2(out) + out

>>out = self.classifier(out)

>>return out

6.4.Defining the model and moving Model From CPU to GPU

>>model = to_device(ResNet9(3, len(train.classes)), device)

>>model

Output:-ResNet9((conv1): Sequential((0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track _running _

stats=True)(2): ReLU(inplace=True))(conv2): Sequential((0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)(2): ReLU(inplace=True)(3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False))(res1): Sequential((0): Sequential((0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)(2): ReLU(inplace=True))(1): Sequential(

(0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)(2): ReLU(inplace=True)))

(conv3): Sequential((0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(2): ReLU(inplace=True)(3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False))(conv4): Sequential((0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)(2): ReLU(inplace=True)(3): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False))(res2): Sequential((0): Sequential((0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)(2): ReLU(inplace=True))

(1): Sequential((0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(2): ReLU(inplace=True)))(classifier): Sequential((0): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)(1): Flatten(start_dim=1, end_dim=-1)(2): Linear(in_features=512, out_features=38, bias=True)))

6.5.Summary of the model

>>INPUT_SHAPE = (3, 256, 256)

>>print(summary(model.cuda(), (INPUT_SHAPE)))

Output:-

---------- Layer (type) Output Shape Param # ==========

Conv2d-1 [-1, 64, 256, 256] 1,792 BatchNorm2d-2 [-1, 64, 256, 256] 128 ReLU-3 [-1, 64, 256, 256] 0 Conv2d-4 [-1, 128, 256, 256] 73,856 BatchNorm2d-5 [-1, 128, 256, 256] 256 ReLU-6 [-1, 128, 256, 256] 0 MaxPool2d-7 [-1, 128, 64, 64] 0 Conv2d-8 [-1, 128, 64, 64] 147,584 BatchNorm2d-9 [-1, 128, 64, 64] 256 ReLU-10 [-1, 128, 64, 64] 0 Conv2d-11 [-1, 128, 64, 64] 147,584 BatchNorm2d-12 [-1, 128, 64, 64] 256 ReLU-13 [-1, 128, 64, 64] 0 Conv2d-14 [-1, 256, 64, 64] 295,168 BatchNorm2d-15 [-1, 256, 64, 64] 512 ReLU-16 [-1, 256, 64, 64] 0 MaxPool2d-17 [-1, 256, 16, 16] 0 Conv2d-18 [-1, 512, 16, 16] 1,180,160 BatchNorm2d-19 [-1, 512, 16, 16] 1,024 ReLU-20 [-1, 512, 16, 16] 0 MaxPool2d-21 [-1, 512, 4, 4] 0 Conv2d-22 [-1, 512, 4, 4] 2,359,808 BatchNorm2d-23 [-1, 512, 4, 4] 1,024 ReLU-24 [-1, 512, 4, 4] 0 Conv2d-25 [-1, 512, 4, 4] 2,359,808 BatchNorm2d-26 [-1, 512, 4, 4] 1,024 ReLU-27 [-1, 512, 4, 4] 0 MaxPool2d-28 [-1, 512, 1, 1] 0 Flatten-29 [-1, 512] 0 Linear-30 [-1, 38] 19,494 ========== Total params: 6,589,734 Trainable params: 6,589,734 Non-trainable params: 0 ---------- Input size (MB): 0.75 Forward/backward pass size (MB): 343.95 Params size (MB): 25.14 Estimated Total Size (MB): 369.83 ---------- None

6.6.Training Model

evaluate function, which will perform the validation phase, and a fit_one_cycle function which will perform the entire training process.

Learning Rate Scheduling: Instead of using a fixed learning rate, we will use a learning rate scheduler, which will change the learning rate after every batch of training. There are many strategies for varying the learning rate during training, and the one we'll use is called the "One Cycle Learning Rate Policy", which involves starting with a low learning rate, gradually increasing it batch-by-batch to a high learning rate for about 30% of epochs, then gradually decreasing it to a very low value for the remaining epochs.

Weight Decay: We also use weight decay, which is a regularization technique which prevents the weights from becoming too large by adding an additional term to the loss function.

Gradient Clipping: Apart from the layer weights and outputs, it also helpful to limit the values of gradients to a small range to prevent undesirable changes in parameters due to large gradient values. This simple yet effective technique is called gradient clipping.

```
 # for training

@torch.no_grad()

>>def evaluate(model, val_loader):

>>model.eval()

>>outputs = [model.validation_step(batch) for batch in val_loader]

>>return model.validation_epoch_end(outputs)

>>def get_lr(optimizer):

>>for param_group in optimizer.param_groups:

>>return param_group['lr']

>>def fit_OneCycle(epochs, max_lr, model, train_loader, val_loader,
>>weight_decay=0,

>>grad_clip=None, opt_func=torch.optim.SGD):

>>torch.cuda.empty_cache()

>>history = []

>>optimizer = opt_func(model.parameters(), max_lr,weight_decay=weight_decay) #scheduler

for one cycle learniing rate
```

```
>>sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr,epochs=epochs,
steps_per_epoch=len(train_loader))
>>for epoch in range(epochs):# Training
>>model.train()
>>train_losses = []
>>lrs = []
>>for batch in train_loader:
>>loss = model.training_step(batch)
>>train_losses.append(loss)
>>loss.backward()# gradient clipping
>>if grad_clip:
>>nn.utils.clip_grad_value_(model.parameters(), grad_clip)
>>optimizer.step()
>>optimizer.zero_grad() # recording and updating learning rates
>>lrs.append(get_lr(optimizer))
>>sched.step() # validation
>>result = evaluate(model, val_loader)
>>result['train_loss'] = torch.stack(train_losses).mean().item()
>>result['lrs'] = lrs
>>model.epoch_end(epoch, result)
```

>>history.append(result)

>>return history

6.7.Validation loss and accuracy

>>history = [evaluate(model, valid_dl)]

>>history

Output:-CPU times: user 1min 8s, sys: 3.36 s, total: 1min 11s

Wall time: 1min 8s [{'val_loss': tensor(3.6378, device='cuda:0'), 'val_accuracy': tensor(0.0282)}]

Randomly initialized weights, that is why accuracy come to near 0.019 (that is 1.9% chance of getting the right answer or you can say model randomly chooses a class). Now, declare some hyper parameters for the training of the model. We can change it if result is not satisfactory.

>>epochs = 2

>>max_lr = 0.01

>>grad_clip = 0.1

>>weight_decay = 1e-4

>>opt_func = torch.optim.Adam

>>history+=fit_OneCycle(epochs,max_lr,model,train_dl,valid_dl,grad_clip=grad_clip,weight_

decay=1e-4,opt_func=opt_func)

Output:-Epoch [0], last_lr: 0.00812, train_loss: 0.7490, val_loss: 0.4811, val_acc: 0.8560 Epoch [1], last_lr: 0.00000, train_loss: 0.1239, val_loss: 0.0251, val_acc: 0.9934 CPU times: user 16min 20s, sys: 16min 39s, total: 32min 59s Wall time: 32min 45s

6.8.Plotting

```
>>def plot_accuracies(history):
>>accuracies = [x['val_accuracy'] for x in history]
>>        plt.plot(accuracies, '-x')
>>        plt.xlabel('epoch')
>>         plt.ylabel('accuracy')
>>        plt.title('Accuracy vs. No. of epochs');
>>def plot_losses(history):
>>     train_losses = [x.get('train_loss') for x in history]
>>     val_losses = [x['val_loss'] for x in history]
>>     plt.plot(train_losses, '-bx')
>>      plt.plot(val_losses, '-rx')
>>      plt.xlabel('epoch')
>>      plt.ylabel('loss')
>>      plt.legend(['Training', 'Validation'])
>>      plt.title('Loss vs. No. of epochs');
>>def plot_lrs(history):
>>     lrs = np.concatenate([x.get('lrs', []) for x in history])
 >>      plt.plot(lrs)
```

>>plt.xlabel('Batch no.')

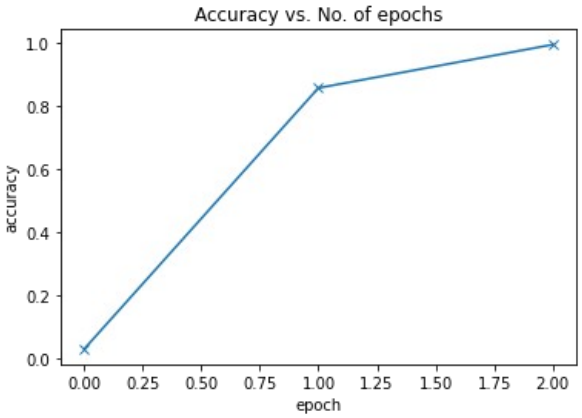
>>plt.ylabel('Learning rate')

>>plt.title('Learning Rate vs. Batch no.');

6.8.1.Validation Accuracy

>>plot_accuracies(history)

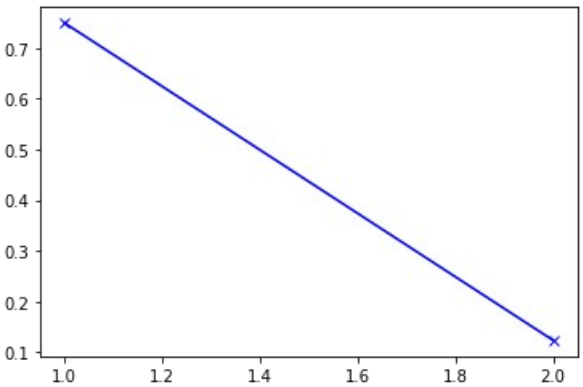
Fig.*6.8.1.Validation Accuracy*



6.8.2.Loss Accuracy

>>plot_losses(history)

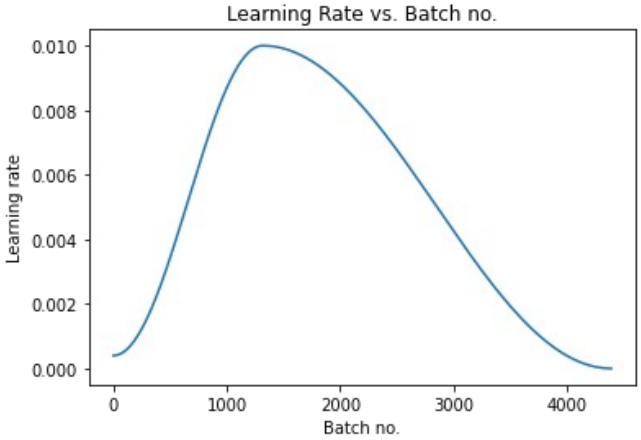
Fig.*6.8.2.Loss Accuracy*



6.8.3.IRS Accuracy

>>plot_lrs(history)

Fig.*6.8.3.IRS Accuracy*



7.TESTING MODEL

7.1.Testdata importing

#We have 33 images for testing data

>>test_dir = "/content/test"

>>test = ImageFolder(test_dir, transform=transforms.ToTensor())

>>test_images = sorted(os.listdir(test_dir + '/test')) # since images in test folder are in alphabetical order

>>test_images

Output:-['AppleCedarRust1.JPG', 'AppleCedarRust2.JPG', 'AppleCedarRust3.JPG', 'Apple Cedar Rust4.JPG', 'AppleScab1.JPG', 'AppleScab2.JPG', 'AppleScab3.JPG', 'CornCommonRust1.JPG', 'CornCommonRust2.JPG', 'CornCommonRust3.JPG', 'PotatoEarlyBlight1.JPG', 'Potato Early Blight 2.JPG', 'PotatoEarlyBlight3.JPG', 'PotatoEarlyBlight4.JPG', 'PotatoEarlyBlight5.JPG', 'Potato Healthy1.JPG', 'PotatoHealthy2.JPG', 'TomatoEarlyBlight1.JPG', 'TomatoEarlyBlight2.JPG', 'TomatoEarlyBlight3.JPG', 'TomatoEarlyBlight4.JPG', 'TomatoEarlyBlight5.JPG', 'Tomato Early Blight6.JPG', 'TomatoHealthy1.JPG', 'Tomato Healthy 2.JPG', 'TomatoHealthy3.JPG', 'Tomato Healthy4.JPG', 'TomatoYellowCurlVirus1.JPG', 'TomatoYellowCurlVirus2.JPG', 'Tomato Yellow CurlVirus3.JPG', 'TomatoYellowCurlVirus4.JPG', 'TomatoYellowCurlVirus5.JPG', 'Tomato Yellow CurlVirus6.JPG']

7.2.Predict test image

```
>>def predict_image(img, model):
"""Converts image to array and return the predicted class with highest probability"""
# Convert to a batch of 1
>>xb = to_device(img.unsqueeze(0), device)
>># Get predictions from model
>>yb = model(xb)
# Pick index with highest probability
>>_, preds = torch.max(yb, dim=1)
# Retrieve the class label
>>return train.classes[preds[0].item()]
# predicting the image
>>img, label = test[1]
>>plt.imshow(img.permute(1, 2, 0))
>>print('Label:', test_images[1], ', Predicted:', predict_image(img, model))
>> Output:- Label: AppleCedarRust2.JPG , Predicted: Apple___Cedar_apple_rust
```

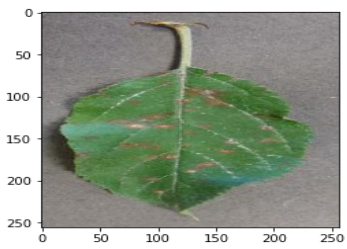


Fig.7.2.Predict test image

getting all predictions (actual label vs predicted)

>>for i, (img, label) in enumerate(test):

>>print('Label:', test_images[i], ', Predicted:', predict_image(img, model))

Output:-Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust

Label: AppleCedarRust2.JPG , Predicted: Apple___Cedar_apple_rust

Label: AppleCedarRust3.JPG , Predicted: Apple___Cedar_apple_rust

Label: AppleCedarRust4.JPG , Predicted: Apple___Cedar_apple_rust

Label: AppleScab1.JPG , Predicted: Apple___Apple_scab

Label: AppleScab2.JPG , Predicted: Apple___Apple_scab

Label: AppleScab3.JPG , Predicted: Apple___Apple_scab

Label: CornCommonRust1.JPG , Predicted: Corn_(maize)___Common_rust_

Label: CornCommonRust2.JPG , Predicted: Corn_(maize)___Common_rust_

Label: CornCommonRust3.JPG , Predicted: Corn_(maize)___Common_rust_

Label: PotatoEarlyBlight1.JPG , Predicted: Potato___Early_blight

Label: PotatoEarlyBlight2.JPG , Predicted: Potato___Early_blight

Label: PotatoEarlyBlight3.JPG , Predicted: Potato___Early_blight

Label: PotatoEarlyBlight4.JPG , Predicted: Potato___Early_blight

Label: PotatoEarlyBlight5.JPG , Predicted: Potato___Early_blight

Label: PotatoHealthy1.JPG , Predicted: Potato___healthy

Label: PotatoHealthy2.JPG , Predicted: Potato___healthy

Label: TomatoEarlyBlight1.JPG , Predicted: Tomato___Early_blight

Label: TomatoEarlyBlight2.JPG , Predicted: Tomato___Early_blight

Label: TomatoEarlyBlight3.JPG , Predicted: Tomato___Early_blight

Label: TomatoEarlyBlight4.JPG , Predicted: Tomato___Early_blight

Label: TomatoEarlyBlight5.JPG , Predicted: Tomato___Early_blight

Label: TomatoEarlyBlight6.JPG , Predicted: Tomato___Early_blight

Label: TomatoHealthy1.JPG , Predicted: Tomato___healthy

Label: TomatoHealthy2.JPG , Predicted: Tomato___healthy

Label: TomatoHealthy3.JPG , Predicted: Tomato___healthy

Label: TomatoHealthy4.JPG , Predicted: Tomato___healthy

Label: TomatoYellowCurlVirus1.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus

Label: TomatoYellowCurlVirus2.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus

Label: TomatoYellowCurlVirus3.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus

Label: TomatoYellowCurlVirus4.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus

Label: TomatoYellowCurlVirus5.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus

Label: TomatoYellowCurlVirus6.JPG , Predicted: Tomato___Tomato_Yellow_Leaf_Curl_Virus

7.3.Save The Model

7.3.1. Save/Load state_dict (Recommended)

When saving a model for inference, it is only necessary to save the trained model's learned parameters. Saving the model's state_dict with the torch.save() function will give you the most flexibility for restoring the model later, which is why it is the recommended method for saving models.A common PyTorch convention is to save models using either a .pt or .pth file extension.Remember that you must call model.eval() to set dropout and batch normalization layers to evaluation mode before running inference.Failing to do this will yield inconsistent inference results.

```
# saving to the kaggle working directory

>>PATH = './plant-disease-model.pth'

>>torch.save(model.state_dict(), PATH)
```

7.3.2.Save/Load Entire Model

This save/load process uses the most intuitive syntax and involves the least amount of code. Saving a model in this way will save the entire module using Python's pickle module. The disadvantage of this approach is that the serialized data is bound to the specific classes and the exact directory structure used when the model is saved. The reason for this is because pickle does not save the model class itself. Rather, it saves a path to the file containing the class, which is used during load time. Because of this, your code can break in various ways when used in other projects or after refactors.# saving the entire model to working directory

```
>>PATH = './plant-disease-model-complete.pth'

>>torch.save(model, PATH)
```

8.CONCLUSION

ResNets perform significantly well for image classification when some of the parameters are tweaked and techniques like scheduling learning rate, gradient clipping and weight decay are applied. The model is able to predict every image in test set perfectly without any errors !!!!We got Accuracy of prediction with 99.3%.