

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY,  
BELGAUM, KARNATAKA**



**PROJECT REPORT**

**ON**

**“PLANT LEAF DISEASE DETECTION”**

*Submitted in partial fulfillment of the requirement for the award of the degree of*

**BACHELOR OF ENGINEERING  
IN  
COMPUTER SCIENCE AND ENGINEERING**

**Submitted by**

**USN**

**NAME**

**2SD17CS108**

**TEJAS M P**

**2SD17CS044**

**MANOJ C NAIK**

**2SD17CS066**

**PRASHANT KALLI**

**2SD17CS072**

**RAHUL A**

**Under the Guidance of**

**Dr. U P Kulkarni.**

**Dept. of CSE, SDMCET, Dharwad**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
S.D.M. COLLEGE OF ENGINEERING & TECHNOLOGY,  
DHARWAD-580002**

**2020-2021**

**S.D.M COLLEGE OF ENGINEERING & TECHNOLOGY,  
DHARWAD –580002**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

*Certified that the project work and presentation entitled “**PLANT LEAF DISEASE DETECTION**” is a bonafide work carried out by **TEJAS M P (2SD17CS108) PRASHANT KALLI (2SD17CS066), RAHUL A (2SD17CS072),** and **MANOJ C NAIK (2SD17CS044),** students of **S. D. M. College of Engineering & Technology, Dharwad,** in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum,** during the year 2020-2021. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The Project has been approved, as it satisfies the academic requirements in respect of project report prescribed for the said degree.*

**Dr. U P Kulkarni**

Project Guide and HoD-CSE

**External Viva**

Name of Examiners

Signature with date

1. \_\_\_\_\_

2. \_\_\_\_\_

## **ABSTRACT**

*Crop disease is a serious concern for safety of food, but its fast detection still remains difficult in different parts of the lack of proper infrastructure Automatic identification of plant disease is necessary for food security, yield loss estimation and management of disease. With the worldwide increase in digital cameras and continuous improvements in computer vision domain, the automated techniques for detection of disease are highly in demands in precision agriculture. Working on a dataset which includes images of crop leaves, a Residual Network was trained to perform this task of classification, The proposed ResNet model accomplished a 99.40% accuracy on a test set, illustrating the viability of the proposed model. Overall the process of training ResNet models on an open image dataset provides a sound way towards crop disease detection using automated networks on an enormous global scale. Providing the user-friendly website for leaf disease detection to farmers*

# PLANT LEAF DISEASE DETECTION

## Table of Contents

<a href="#">PROBLEM STATEMENT</a> .....	3
<a href="#">CHAPTER 1: INTRODUCTION</a> .....	4
<a href="#">CHAPTER 2: LITERATURE SURVEY</a> .....	5
<a href="#">CHAPTER 3: DETAILED DESIGN</a> .....	6
<a href="#">CHAPTER 4: PROJECT SPECIFIC REQUIREMENTS</a> .....	9
<a href="#">CHAPTER 5: IMPLEMENTATION</a> .....	10
<a href="#">CHAPTER 6: RESULTS</a> .....	14
<a href="#">REFERENCES</a> .....	17

## **PROBLEM STATEMENT**

Plant Leaf Disease Detection using PyTorch and Deep Learning

## CHAPTER 1: INTRODUCTION

In agricultural crops, leaves play a vital role to provide information about the amount and nature of horticultural yield. Several factors affect food production such as climate change, presence of weed, and soil infertility. Apart from that, plant or leaf disease is a global threat to the growth of several agricultural products and a source of economic losses. The failure to diagnose infections/bacteria/virus in plants leads subsequently to insufficient pesticide/fungicide use. Therefore, plant diseases have been largely considered in the scientific community, with a focus on the biological features of diseases. Precision farming uses the most advanced technology for the optimization of decision-making. The visual inspections by experts and biological review are usually carried out through plant diagnosis when required. This method, however, is typically time-consuming and cost ineffective. To address these issues, it is necessary to detect plant diseases by advanced and intelligent techniques

As a solution to this problem, we have devised a system that uses deep learning to analyze, detect and classify any disease that might have affected a plant by taking an image of the leaf. The processing pipeline goes as follows:

1. The leaf is detected in the given image and cropped out
2. The extracted leaf is then run through a classifier to identify which plant the leaf belongs to
3. The leaf is then checked for the disease class, if any, based on the result it displays

### CHAPTER 2: LITERATURE SURVEY

**Kaiming He et al., (2015)** Proposed on " **Deep Residual Learning for Image Recognition** " Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8x deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set.

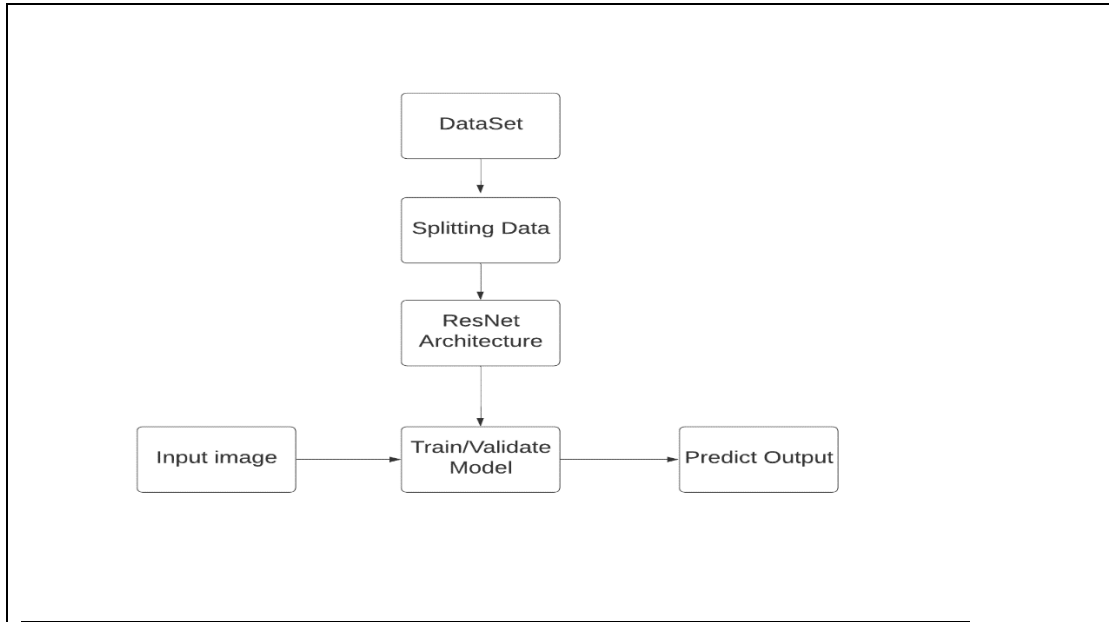
**BAOQI LI et al.,(2018)**, proposed work on "**An Improved ResNet Based on the Adjustable Shortcut Connections**" In this paper, ResNet can achieve deeper network and higher performance, but there is no good explanation for how identity shortcut connections solve the gradient fading problems. Moreover, it is not reasonable to adopt identity mapping for all layer parameters. In this paper, we first establish a simplified ResNet that is similar to the ResNet in principle, and deduce the back propagation of the networks.

**Heechul Jung et al.,(2017)**, proposed work on "**ResNet-Based Vehicle Classification and Localization in Traffic Surveillance Systems** " In this paper, we present ResNet-based vehicle classification and localization methods using real traffic surveillance recordings. We utilize a MIOvision traffic dataset, which comprises 11 categories including a variety of vehicles, such as bicycle, bus, car, motorcycle, and so on.

**Melike Sardogan et al.,(2018)**, proposed work on " **Plant Leaf Disease Detection and Classification Based on CNN** " early detection of diseases is important in agriculture for an efficient crop yield. bacterial spot, late blight, septoria leaf spot and yellow curved leaf diseases affect the crop quality of tomatoes. Automatic methods for classification of plant diseases also help taking action after detecting the symptoms of leaf diseases. paper presents a Convolutional Neural Network (CNN) model and Learning Vector Quantization (LVQ) algorithm base method for tomato leaf disease detection and classification. dataset contains 500 images of tomato leaves with four symptoms of diseases. We have modeled a CNN for automatic feature extraction and classification. Color information is actively used for plant leaf disease researches. In our model, the filters are applied to three channels based on RGB components.

## CHAPTER 3: DETAILED DESIGN

### 3.1: ARCHITECTURE DIAGRAM:



**Architecture Diagram**

#### **Description:**

**1.Data set:** We have collected a Data set having different plants and its variety of diseases for each plant along with the healthy leaf images

**2.Splitting Data:** splitting of collected Data set into 80% and 20% as training Dataset and Validation data set respectively.

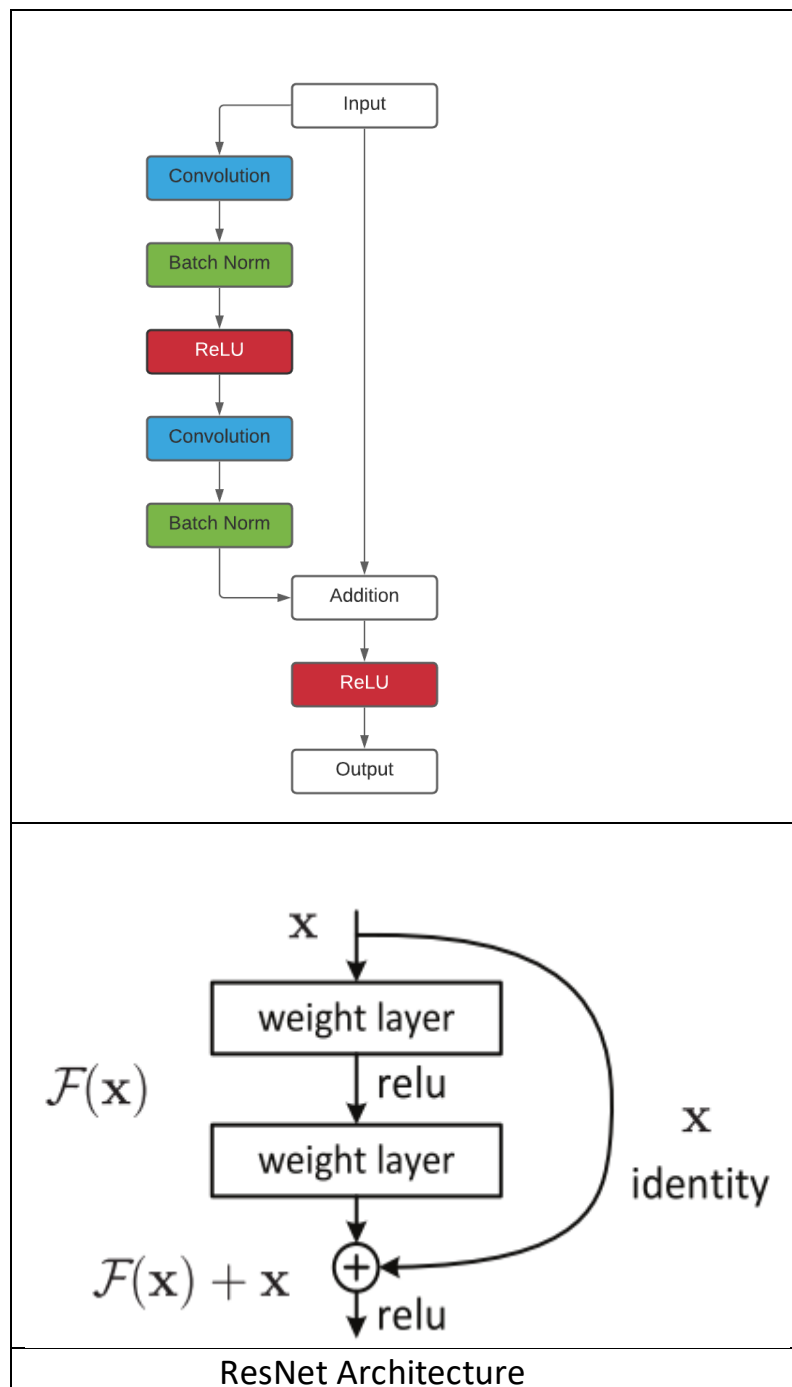
**3.ResNet Architecture:** In ResNets, unlike in traditional neural networks, each layer feeds into the next layer, we use a network with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away, to avoid over-fitting (a situation when validation loss stop 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.

**4.Train/Validate Model:** Training Data set is used to train the model, and Validation Data set is used to check how much the model is learned from training data set and to finding the accuracy of predicting the input image.

**5.input image** is converted into array and passed to the model having resnet architecture which has been trained by the data set and get predictions from the model, picks index with highest probability and prints the class label



## 3.2: ResNet ARCHITECTURE:



## PLANT LEAF DISEASE DETECTION

### Description:

Since neural networks are good function approximators, they should be able to easily solve the identify function, where the output of a function becomes the input itself.

$$F(x)=x$$

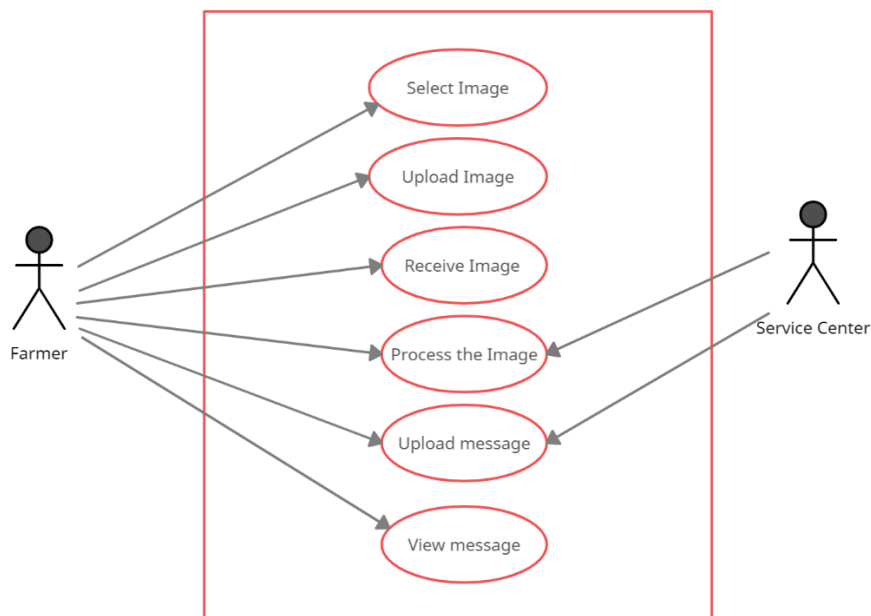
Following the same logic, if we bypass the input to the first layer of the model to be the output of the last layer of the model, the network should be able to predict whatever function it was learning before with the input added to it.

$$F(x) + x = h(x)$$

One of the problems ResNets solve is the famous known **vanishing gradient**. This is because when the network is too deep, the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. This result on the weights never updating its values and therefore, no learning is being performed.

With ResNets, the **gradients can flow directly through the skip connections backwards from later layers to initial filters**.

### 3.1: USECASE DIAGRAM:



Use Case -Diagram

## CHAPTER 4: PROJECT SPECIFIC REQUIREMENTS

### **4.1-Functional Requirements:**

- **User Level:**
  1. User shall be able to input test data (image).
  2. User shall be able to perform analysis after the results.
  3. User shall be able to train the data if required.
  4. User shall be able to view the result
- **System Level:**
  1. System should provide option to input the test image.
  2. System should provide option to display results.
  3. System should accept the input.
  4. System should predict the output.
  5. System should display result.
  6. System should provide the option for user's to retrain.

### **4.2-Nonfunctional Requirements:**

- 1.Portability:** The program should be platform Independent
- 2.Usability:** The system should be easy to deal and simple to understand
- 3.Speed and Response:** Execution of the operations must be in seconds
- 4.Flexibility:** The system should be easy to modify
- 5.Accuracy and Precision:** The system should perform its process with accuracy and precision to avoid problems

### **4.3- Software and Hardware Requirements:**

1. A computer with at least 2 virtual core processor, at least 4 GB RAM, ideally having a dedicated GPU.
2. Any of these Python IDLE-Jupyter Notebook, Pycharm with Pre-installed necessary libraries and modules.
3. Google Colab to design these models.

### CHAPTER 5: IMPLEMENTATION

#### **5.1 INCORPORATED PACKAGE:**

##### **5.1.1: Pytorch :**

-PyTorch is a python package that provides two high-level features: - Tensor computation (like numpy) with strong GPU acceleration - Deep Neural Networks built on a tape-based autograd system  
You can reuse your favorite python packages such as numpy, scipy and Cython to extend PyTorch when needed.

-Usually one uses PyTorch either as:

- A replacement for numpy to use the power of GPUs.
- a deep learning research platform that provides maximum flexibility and speed

-It provides a wide variety of tensor routines to accelerate and fit your scientific computation needs such as slicing, indexing, math operations, linear algebra, reductions. And they are fast

##### **5.1.2: torchsummary:**

- Torch-summary provides information complementary to what is provided by print(your model) in PyTorch, similar to Tensorflow's model.summary() API to view the visualization of the model, which is helpful while debugging your network. In this project, we implement a similar functionality in PyTorch and create a clean, simple interface to use in your projects.

##### **5.1.3: Matplotlib.pyplot:**

-Matplotlib is a plotting library for the Python programming language and its numericalmathematics extension NumPy. ere is also a procedural "pylab" interface based on astate machine (like OpenGL Matplotlib), designed to closely resemble that of MATLAB. Pyplot is a Matplotlib module which provides a MATLAB-like interface. Matplotlib is designed to beas usable as MATLAB

##### **5.1.4: torchvision:**

-The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision. Used for transforming image into tensors,for working with class and image.

# PLANT LEAF DISEASE DETECTION

```
import os # for working with files
import numpy as np # for numerical computations
import pandas as pd # for working with dataframes
import torch # Pytorch module
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

%matplotlib inline
```

## Importing modules

### 5.2 Exploring the data:

loading the data

```
[3] data_dir = "/content/drive/MyDrive/INPUT/DATASET"
    train_dir = data_dir + "/train"
    valid_dir = data_dir + "/valid"
    diseases = os.listdir(train_dir)

[4] #printing disease name
    print(diseases)

['Potato__Late_blight', 'Potato__healthy', 'Potato__Early_blight']
```

### 5.3 Data preparation for training:

```
# datasets for validation and training
train = ImageFolder(train_dir, transform=transforms.ToTensor())
valid = ImageFolder(valid_dir, transform=transforms.ToTensor())
```

torchvision.datasets is a class which helps in loading all common and famous datasets. It also helps in loading custom datasets. I have used subclass torchvision.datasets. ImageFolder which helps in loading the image data when the data is arranged in this way:

## PLANT LEAF DISEASE DETECTION

root/dog/xxx.png

root/dog/xyy.png

root/dog/xxz.png

root/cat/123.png

root/cat/nsdf3.png

root/cat/asd932\_.png

-Next, after loading the data, we need to transform the pixel values of each image (0-255) to 0-1 as neural networks work quite good with normalized data. The entire array of pixel values is converted to torch tensor and then divided by 255.

### 5.4: Modelling:

It is 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

### 5.5 Building Architecture:

We are going to use *ResNet*, one of the major breakthrough in computer vision since they were introduced in 2015.

In ResNets, 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. Here is a simple residual block:

# PLANT LEAF DISEASE DETECTION

```
[25] 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)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
        self.relu2 = nn.ReLU()

    def forward(self, x):
        out = self.conv1(x)
        out = self.relu1(out)
        out = self.conv2(out)
        return self.relu2(out) + x # ReLU can be applied before or after adding the input
```

## 5.6: Building Final architecture of our model:

```
# defining the model and moving it to the GPU
model = to_device(ResNet9(3, len(train.classes)), device)
model

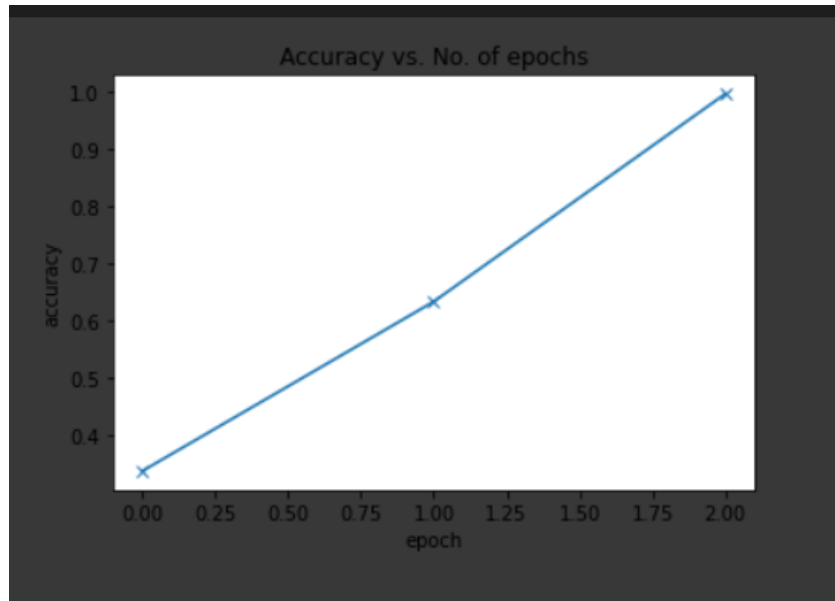
ResNet9(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 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)
  )
  (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=3, bias=True)
  )
)
```

## 5.7: Training the Model:

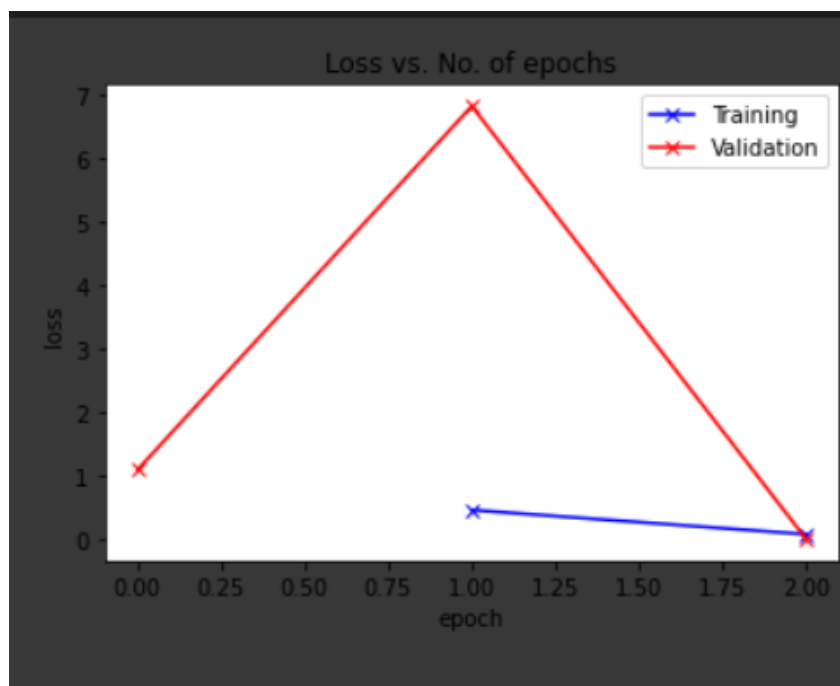
- Training
- Gradient Clipping
- Recording and updating learning rate
- Validation

## CHAPTER 6: RESULTS

### 6.1: Accuracy vs No of epochs:



### 6.2: Loss vs No. of epochs





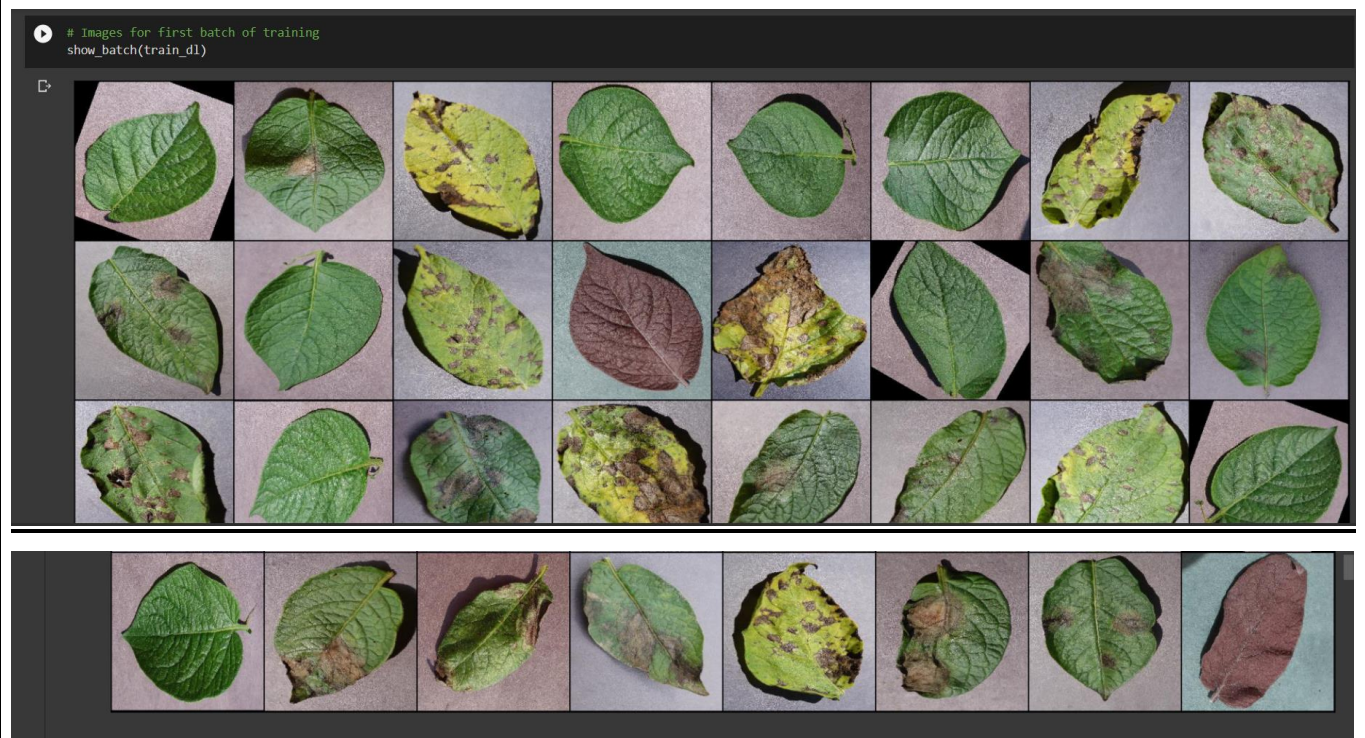
# PLANT LEAF DISEASE DETECTION

## 6.3:Accuracy:

```
[ ] %%time
history += fit_OneCycle(epochs, max_lr, model, train_dl, valid_dl,
                        grad_clip=grad_clip,
                        weight_decay=1e-4,
                        opt_func=opt_func)

Epoch [0], last_lr: 0.00812, train_loss: 0.4652, val_loss: 6.8177, val_acc: 0.6326
Epoch [1], last_lr: 0.00000, train_loss: 0.0843, val_loss: 0.0116, val_acc: 0.9951
CPU times: user 51.6 s, sys: 46.6 s, total: 1min 38s
Wall time: 24min 26s
```

## 6.3: Images of first batch of Training:



# PLANT LEAF DISEASE DETECTION


## 6.4: Output:

```
[39] 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 first image
img, label = test[0]
plt.imshow(img.permute(1, 2, 0))
print( 'Predicted:', predict_image(img, model))

Predicted: Potato Early blight
```



### REFERENCES

- [1]. <https://towardsdatascience.com/understanding-and-visualizing-resnets->
- [2]. <https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>
- [3]. <https://jovian.ai/aakashns/05b-cifar10-resnet>
- [4]. <https://pytorch.org/>
- [5]. Singh, D., Jain, N., Jain, P., Kayal, P., Kumawat, S., and Batra, N. (2020, January 5–7). PlantDoc: A dataset for visual plant disease detection. Proceedings of the 7th ACM IKDD CoDS and 25th COMAD, Hyderabad, India.  
<https://doi.org/10.1145/3371158.3371196>
- [6]. Bhangemanisha *et al.*  
**Smart farming: Pomegranate disease detection using image processing**  
Procedia Comput. Sci.  
(2015)
- [7]. LiuWeibo *et al.*  
**A survey of deep neural network architectures and their applications**  
Neurocomputing  
(2017)
- [8]. HarakannavarSunil S. *et al.*  
**Plant leaf disease detection using computer vision and machine learning algorithms**  
Glob. Transitions Proc.  
(2022)