Import Dependencies

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
import torch
from torchvision import datasets, transforms, models  # datsets  , transforms
from torch.utils.data.sampler import SubsetRandomSampler
import torch.nn as nn
import torch.nn.functional as F
from datetime import datetime
```

In [4]:
```
# %Load ext nb black
```

Import Dataset

Dataset Link (Plant Vliiage Dataset):
https://data.mendeley.com/datasets/tywbtsjrjv/1 (https://data.mendeley.com/datasets/tywbtsjrjv/1)

In [5]:
```
transform = transforms.Compose(
    [transforms.Resize(255), transforms.CenterCrop(224), transforms.ToTensor()]
```

In [6]:
```
dataset  =  datasets.ImageFolder("Dataset", transform=transform)
```

In [7]:
```
dataset
```

Out[7]:
```
Dataset ImageFolder
    Number of datapoints: 61486
    Root Location: Dataset
    Transforms (if any): Compose(
                             Resize(size=255, interpolation=PIL.Image.BILINEAR)
                             CenterCrop(size=(224, 224))
                             ToTensor()
                         )
    Target Transforms (if any): None
```

In [8]:
```
indices= list(range(len(dataset)))
```

In [9]:
```
split= int(np.floor(0.85 * len(dataset)))  # train size
```

In [10]:
```
validation= int(np.floor(0.8*split)) #  validation
```

In [11]:
```
print(0, validation, split, len(dataset))
```

```
0 41810 52263 61486
```

In [12]:
```
print(f'length of train size :{validation}')
print(f'length of validation size :{split - validation}')
print(f'length of test size :{len(dataset)-validation}')
```

```
length of train size :41810
length of validation size :10453
length of test size :19676
```

In [13]:
```
np.random.shuffle(indices)
```

Split into Train and Test

In [14]:
```
train_indices, validation_indices, test_indices = (
    indices[:validation],
    indices[validation:split],
    indices[split:],
```

In [15]:
```
train_sampler = SubsetRandomSampler(train_indices)
validation_sampler = SubsetRandomSampler(validation_indices)
test_sampler = SubsetRandomSampler(test_indices)
```

In [16]:
```
targets_size = len(dataset.class_to_idx)
```

Model

Convolution Aithmetic Equation : (W - F + 2P) / S + 1

W = Input Size

F = Filter Size

P = Padding Size

S = Stride

Transfer Learning

In [17]:
```
#model=  models.vgg16(pretrained=True)
```

In [18]:
```
# for params in model.parameters():
#     params.requires_grad = False
```

In [19]:
```
# model
```

In [20]:
```
# n_features = model.classifier[@J.in_features
# n_features
```

In [21]:
```
#model.classifier= nn.Sequential(
#     nn.Linear(n_features, 1624),
#     nn.ReLU(),
#     nn.Dropout(@.4),
#     nn.Linear(1624,  targets_size),
# )
```

In [22]:
```
# model
```

Original Modeling

In [23]:

```
class CNN(nn.Module):
    def _init_(self, K):
        super(CNN, self)._init_()
        self.conv_layers = nn.Sequential(
            # conv1
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),
            # conv2
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2),
            # conv3
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2),
            # conv4
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1
            nn.ReLU(),
            nn.BatchNorm2d(256),
            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1
            nn.ReLU(),
            nn.BatchNorm2d(256),
            nn.MaxPool2d(2),
        )

        self.dense_layers = nn.Sequential(
            nn.Dropout(0.4),
            nn.Linear(50176, 1024),
            nn.ReLU(),
            nn.Dropout(0.4),
            nn.Linear(1024, K),
        )

    def forward(self, X):
        out= self.conv_layers(X)

        # Flatten
        out= out.view(-1, 50176)

        # Fully connected
        out= self.dense_layers(out)

        return out
```

In [24]:
```
device= torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

In [25]:
```
device = "cpu"
```

In [26]:
```
model = CNN(targets_size)
```

In [27]:
```
model.to(device)
```

Out[27]:
```
CNN(
  (conv_layers): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    (3): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=Fa
lse)
    (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU()
    (9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    (10): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU()
    (12): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=F
alse)
    (14): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU()
    (16): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
    (17): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU()
    (19): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=F
alse)
    (21): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU()
    (23): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
    (24): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU()
    (26): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil mode=F
alse)
  )
  (dense_layers): Sequential(
    (0): Dropout(p=0.4, inplace=False)
    (1): Linear(in_features=50176, out_features=1024, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.4, inplace=False)
    (4): Linear(in_features=1024, out_features=39, bias=True)
```

In [28]:
```
# from torchsummary import summary

# summary(model, (3, 224, 224))
```

In [29]:
```
criterion= nn.CrossEntropyloss()  # this include softmax + cross entropy Loss
optimizer= torch.optim.Adam(model.parameters())
```

Batch Gradient Descent

In [34]:
```
def batch_gd(model, criterion, train_loader, test_laoder, epochs):
    train_losses = np.zeros(epochs)
    validation_losses  =  np.zeros(epochs)

    fore in range(epochs):
        t0 = datetime.now()
        train_loss = []
        for inputs, targets in train_loader:
            inputs, targets= inputs.to(device), targets.to(device)

            optimizer.zero_grad()

            output= model(inputs)

            loss= criterion(output, targets)

            train_loss.append(loss.item()) # torch to numpy world

            loss.backward()
            optimizer.step()

        train_loss = np.mean(train_loss)

        validation_loss = []

        for inputs, targets in validation_loader:

            inputs, targets= inputs.to(device), targets.to(device)

            output= model(inputs)

            loss= criterion(output, targets)

            validation_loss.append(loss.item()) # torch to numpy world

        validation_loss= np.mean(validation_loss)

        train_losses[e]  = train_loss
        validation_losses[e] = validation_loss

        dt = datetime.now() - t0

        print(
            f"Epoch   {e+1}/{epochs} Train_loss:{train_loss:.3f} Test_loss:{valic
        )

    return train_losses, validation_losses
```

In [35]:
```
device = "cpu"
```

In [36]:
```
batch size = 64
train  loader= torch.utils.data.Dataloader(
    dataset,  batch_size=batch_size, sampler=train_sampler
)
test_loader = torch.utils.data.Dataloader(
    dataset,  batch_size=batch_size, sampler=test_sampler
)
validation_loader = torch.utils.data.Dataloader(
    dataset,  batch_size=batch_size, sampler=validation_sampler
```

In [37]:
```
train_losses, validation_losses = batch_gd(model, criterion, train_loader, valid,
```

```
Epoch  1/5 Train_loss:1.158 Test_loss:0.974 Duration:2:07:23.036070
Epoch  2/5 Train_loss:0.896 Test_loss:0.780 Duration:2:10:21.588655
Epoch  3/5 Train_loss:0.734 Test_loss:0.667 Duration:12:04:14.439644
Epoch  4/5 Train_loss:0.582 Test_loss:0.703 Duration:2:17:11.741300
Epoch  5/5 Train_loss:0.493 Test_loss:0.557 Duration:1:52:20.861600
```

Save the Model

In [38]:
```
torch.save(model.state_dict(), 'plant_disease_model_l_latest.pt')
```

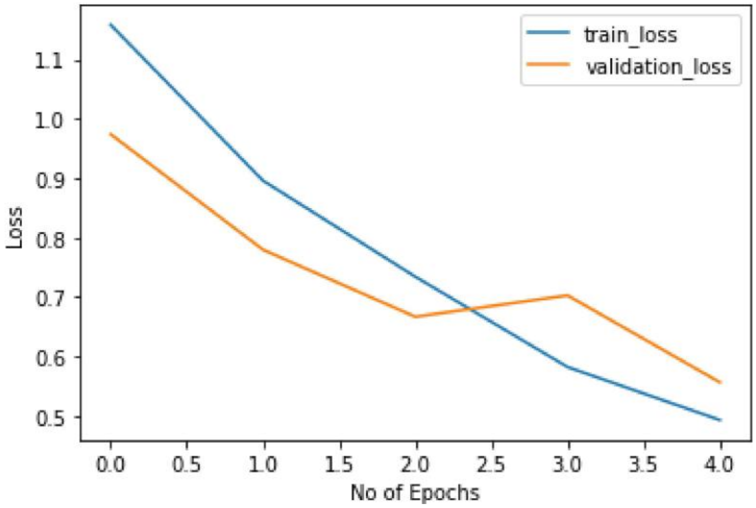
Load Model

In [39]:
```
#model=  CNN(targets_size)
# model. Load_state_ diet (torch. Load ( "plant_disease_model_l. pt"))
# model. eval ()
```

In [40]:
```
#  %matplotlib  notebook
```

Plot the loss

In [41]:
```
plt.plot(train_losses, label= 'train_loss')
plt.plot(validation_losses, label= 'validation_loss')
plt.xlabel('No of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Accuracy

In [42]:
```
def accuracy(loader):
    n_correct = 0
    n_total = 0

    for inputs, targets in loader:
        inputs, targets= inputs.to(device), targets.to(device)

        outputs= model(inputs)

        _, predictions= torch.max(outputs, 1)

        n_correct +=(predictions== targets).sum().item()
        n_total += targets.shape[0]

    ace= n_correct/ n_total
    return ace
```

In [43]:
```
train_acc = accuracy(train_loader)
test_acc = accuracy(test_loader)
validation_acc = accuracy(validation_loader)
```

In [44]:
```
print(
    f"Train Accuracy   {train_acc}\nTest Accuracy   {test_acc}\nValidation Accur;
```

```
Train Accuracy: 0.8749581439846926
Test Accuracy: 0.8376883877263364
Validation Accuracy: 0.8437769061513442
```

Single Image Prediction

In [45]:
```
transform_index_to_disease = dataset.class to idx
```

In [46]:
```
transform_index_to_disease = diet(
    [(value, key) for key, value in transform_index_to_disease.items()]
)  # reverse the index
```

In [47]:
```
transform index to disease
```

Out[47]:
```
{0: 'Apple_Apple_scab',
 1: 'Apple_Black_rot',
 2: 'Apple_Cedar_apple_rust',
 3: 'Apple_healthy',
 4: 'Background_without_leaves',
 5: 'Blueberry_healthy',
 6: 'Cherry_Powdery_mildew',
 7: 'Cherry_healthy',
 8: 'Corn_Cercospora_leaf_spot Gray_leaf_spot',
 9: 'Corn_Common_rust',
 10: 'Corn_Northern_Leaf_Blight',
 11: 'Corn_healthy',
 12: 'Grape_Black_rot',
 13: 'Grape_Esca_(Black_Measles)',
 14: 'Grape_Leaf_blight_(Isariopsis_Leaf_Spot)',
 15: 'Grape_healthy',
 16: 'Orange_Haunglongbing_(Citrus_greening)',
 17: 'Peach_Bacterial_spot',
 18: 'Peach_healthy',
 19: 'Pepper,_bell_Bacterial_spot',
 20: 'Pepper,_bell_healthy',
 21: 'Potato_Early_blight',
 22: 'Potato_Late_blight',
 23: 'Potato_healthy',
 24: 'Raspberry_healthy',
 25: 'Soybean_healthy',
 26: 'Squash_Powdery_mildew',
 27: 'Strawberry_Leaf_scorch',
 28: 'Strawberry_healthy',
 29: 'Tomato_Bacterial_spot',
 30: 'Tomato_Early_blight',
 31: 'Tomato_Late_blight',
 32: 'Tomato_Leaf_Mold',
 33: 'Tomato_Septoria_leaf_spot',
 34: 'Tomato_Spider_mites Two-spotted_spider_mite',
 35: 'Tomato_Target_Spot',
 36: 'Tomato_Tomato_Yellow_Leaf_Curl_Virus',
 37: 'Tomato_Tomato_mosaic_virus',
 38: 'Tomato_healthy'}
```

In [48]:
```
from PIL import Image
import torchvision.transforms.functional as TF
```

In [49]:
```
def single_prediction(image_path):
    image= Image.open(image_path)
    image= image.resize((224, 224))
    input_data = TF.to_tensor(image)
    input_data = input_data.view((-1, 3, 224, 224))
    output= model(input_data)
    output= output.detach().numpy()
    index= np.argmax(output)
    print("Original : ", image_path[12:-4])
    pred = transform_index_to_disease[index]
    plt.imshow(image)
    plt.title("Disease Prediction: "+ pred)
    plt.show()
```

In [50]:
```
single_prediction("test_images/ Apple_ceder _apple_rust. JPG")
```

```
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-50-feccb4fd75fa> in <module>
----> 1 single_prediction("test_images/Apple_ceder_apple_rust.JPG'')

<ipython-input-49-alldd5e4e2b4> in single_prediction(image_path)
      1 def single_prediction(image_path):
----> 2     image= Image.open(image_path)
      3     image= image.resize((224, 224))
      4     input_data = TF.to_tensor(image)
      5     input_data = input_data.view((-1, 3, 224, 224))

N\anaconda3\envs\krishna\lib\site-packages\PIL\Image.py in open(fp, mode, forma
ts)
   2902
   2903     if filename:
-> 2904         fp = builtins.open(filename, "rb")
   2905         exclusive_fp = True
   2906

FileNotFoundError: [Errno 2] No such file or directory: 'test_images/Apple_cede
r_apple_rust.JPG'
```

Wrong Prediction

In [ ]:
```
single_prediction("test_images/ Apple_scab. JPG")
```

In [ ]:
```
single_prediction("test_images/Grape_esca.JPG")
```

In [ ]:
```
single_prediction("test_images/apple_black_rot.JPG")
```

In [ ]:
```
single_prediction("test_images/apple_healthy.JPG")
```

In [ ]:
```
single_prediction("test_images/background_without_leaves.jpg")
```

In [ ]:
```
single_prediction("test_images/blueberry_healthy.JPG")
```

In [ ]:
```
single_prediction("test_images/cherry_healthy.JPG")
```

In [ ]:
```
single_prediction("test_images/cherry_powdery_mildew.JPG")
```

In [ ]:
```
single_prediction( "test_images/  corn_cercospora_leaf.  JPG")
```

In [ ]:
```
single_prediction( "test_images/  corn_common_rust. JPG")
```

In [ ]:
```
single_prediction( "test_images/ corn_healthy. jpg")
```

In [ ]:
```
single_prediction( "test_images/ corn_northen_leaf_blight. JPG")
```

In [ ]:
```
single_prediction(  "test_images/grape_black_rot.   JPG")
```

In [ ]:
```
single_prediction(  "test_images/grape_healthy.   JPG")
```

In [ ]:
```
single_prediction(  "test_images/grape_leaf_blight.    JPG")
```

In [ ]:
```
single_prediction( "test_images/orange_haunglongbing. JPG")
```

In [ ]:
```
single_prediction ( "test_images/peach_bacterial_spot.JPG")
```

In [ ]:
```
single_prediction(  "test_images/peach_healthy.   JPG")
```

In [ ]:
```
single_prediction   (   "test_images/pepper_bacterial_spot.    JPG")
```

In [ ]:
```
single_prediction(  "test_images/pepper_bell_healthy.   JPG")
```

In [ ]:
```
single_prediction( "test_images/potato_early_blight.JPG")
```

In [ ]:
```
single_prediction(  "test_images/potato_healthy.   JPG")
```

In [ ]:
```
single_prediction(  "test_images/potato_late_blight.  JPG")
```

In [ ]:
```
single_prediction(  "test_images/raspberry_healthy.    JPG")
```

In [ ]:
```
single_prediction( "test_images/soyaben    healthy. JPG")
```

In [ ]:
```
single_prediction( "test_images/potato_late_blight. JPG")
```

In [ ]:
```
single_prediction( "test_images/squash_powdery_mildew. JPG")
```

In [ ]:
```
single_prediction( "test_images/starwberry_healthy. JPG")
```

In [ ]:
```
single_prediction("test_images/starwberry_leaf_scorch. JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_bacterial_spot.JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_early_blight. JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_healthy. JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_late_blight. JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_leaf_mold. JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_mosaic_virus. JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_septoria_leaf_spot. JPG")
```

In [ ]:
```
single_prediction("test_images/tomato_spider_mites_two_spotted_spider_mites. JPG":
```

In [ ]:
```
single_prediction( "test_images/tomato_target_spot.JPG")
```

In [ ]:
```
single_prediction( "test_images/tomato_yellow_leaf_curl_virus. JPG")
```

Image Outside of Dataset

In [ ]:
```
single_prediction("PHOTO. jpg")
```

In [ ]:
```
single_prediction( "test_images/tomato_yellow_leaf_curl_virus2. jpg")
```

In [ ]:
```
single_prediction( "test_images/tomato-leaf-curl-virus3. jpg")
```

In [ ]:
```
single_prediction( "test_images/tomato-bacterial-spot2. jpg")
```

In [ ]
```
single_prediction( "test_images/tomato-mold. jpg")
```