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  , tra
from torch.utils.data.sampler import SubsetRandomSampler
import torch.nn as nn
import torch.nn.functional as F
from datetime import datetime
```

In [3]:
```
%load_ext nb_black
```

In [4]:
```
transform = transforms.Compose(
    [transforms.Resize(255), transforms.CenterCrop(224), transforms.To
)
```

In [5]:
```
dataset = datasets.ImageFolder("Dataset", transform=transform)
```

In [6]:
```
dataset

```

Out[6]:
```
Dataset ImageFolder
    Number of datapoints: 61486
    Root location: Dataset
    StandardTransform
Transform: Compose(
               Resize(size=255, interpolation=bilinear, max_size=None, an
tialias=warn)
               CenterCrop(size=(224, 224))
               ToTensor()
           )
```

In [78]:
```
indices = list(range(len(dataset)))

```

In [8]:
```
split = int(np.floor(0.85 * len(dataset)))  # train_size
split
```

Out[8]:
```
52263
```

In [9]:
```
validation = int(np.floor(0.70 * split))  # validation
validation
```

Out[9]:
```
36584
```

In [10]:
```
print(0, validation, split, len(dataset))
```

```
0 36584 52263 61486
```

In [11]:
```
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 :36584
length of validation size :15679
length of test size :24902
```

In [12]:
```
np.random.shuffle(indices)
```

In [13]:
```
train_indices, validation_indices, test_indices = (
    indices[:validation],
    indices[validation:split],
    indices[split:],
)
```

In [14]:
```
train_sampler = SubsetRandomSampler(train_indices)
validation_sampler = SubsetRandomSampler(validation_indices)
test_sampler = SubsetRandomSampler(test_indices)
```

In [15]:
```
targets_size = len(dataset.class_to_idx)
targets_size
```

Out[15]: 39

In [17]:

```
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, p
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3,
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),
            # conv2
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3,
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3,
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2),
            # conv3
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3,
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2),
            # conv4
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3
            nn.ReLU(),
            nn.BatchNorm2d(256),
            nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3
            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)

```

```
56          return out
```

In [18]:

```
1  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
2  print(device)
```

```
cpu
```

In [19]:

```
1  device = "cpu"
```

In [20]:

```
model = CNN(targets_size)
model
```

Out[20]:
```
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_runn
ing_stats=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_runn
ing_stats=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (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_runn
ing_stats=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_run
ning_stats=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
    (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_ru
nning_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_ru
nning_stats=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
    (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_ru
nning_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_ru
nning_stats=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
  )
  (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 [21]:

```
1 model.to(device)
```

Out[21]:
```
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_runn
ing_stats=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_runn
ing_stats=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (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_runn
ing_stats=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_run
ning_stats=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
    (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_ru
nning_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_ru
nning_stats=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
    (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_ru
nning_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_ru
nning_stats=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
  )
  (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 [22]:

```
from torchsummary import summary

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

```
|    └─Conv2d: 2-8                       [-1, 64, 112, 112]        18,
496
|    └─ReLU: 2-9                         [-1, 64, 112, 112]        --
|    └─BatchNorm2d: 2-10                 [-1, 64, 112, 112]        128
|    └─Conv2d: 2-11                      [-1, 64, 112, 112]        36,
928
|    └─ReLU: 2-12                        [-1, 64, 112, 112]        --
|    └─BatchNorm2d: 2-13                 [-1, 64, 112, 112]        128
|    └─MaxPool2d: 2-14                   [-1, 64, 56, 56]          --
|    └─Conv2d: 2-15                      [-1, 128, 56, 56]         73,
856
|    └─ReLU: 2-16                        [-1, 128, 56, 56]         --
|    └─BatchNorm2d: 2-17                 [-1, 128, 56, 56]         256
|    └─Conv2d: 2-18                      [-1, 128, 56, 56]         14
7,584
|    └─ReLU: 2-19                        [-1, 128, 56, 56]         --
|    └─BatchNorm2d: 2-20                 [-1, 128, 56, 56]         256
|    └─MaxPool2d: 2-21                   [-1, 128, 28, 28]         --
|    └─Conv2d: 2-22                      [-1, 256, 28, 28]         29
5,168
```

In [23]:

```
criterion = nn.CrossEntropyLoss()  # this include softmax + cross entr
optimizer = torch.optim.Adam(model.parameters())
```

In [29]:

```
def batch_gd(model, criterion, train_loader, validation_loader, epochs
    train_losses = np.zeros(epochs)
    validation_losses = np.zeros(epochs)

    for e 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 worl

        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_
        )

    return train_losses, validation_losses
```

In [30]:
```
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 [31]:
```
train_losses, validation_losses = batch_gd(
    model, criterion, train_loader, validation_loader, 5
)

```

```
Epoch : 1/5 Train_loss:2.356 Test_loss:1.540 Duration:3:04:18.209353
Epoch : 2/5 Train_loss:1.350 Test_loss:1.187 Duration:2:49:28.855508
Epoch : 3/5 Train_loss:1.069 Test_loss:1.011 Duration:2:42:33.252418
Epoch : 4/5 Train_loss:0.910 Test_loss:0.864 Duration:2:58:01.601071
Epoch : 5/5 Train_loss:0.756 Test_loss:0.847 Duration:2:59:22.336592
```

In [32]:
```
torch.save(model.state_dict(), "plant_disease_model_final.pt")
```

In [35]:

```
targets_size = 39
model = CNN(targets_size)
model.load_state_dict(torch.load("plant_disease_model_final.pt"))
model.eval()
```

Out[35]:
```
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_runn
ing_stats=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_runn
ing_stats=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_m
ode=False)
    (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_runn
ing_stats=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_run
ning_stats=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
    (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_ru
nning_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_ru
nning_stats=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
    (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_ru
nning_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_ru
nning_stats=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
mode=False)
  )
  (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)
```

```
    )
)
```

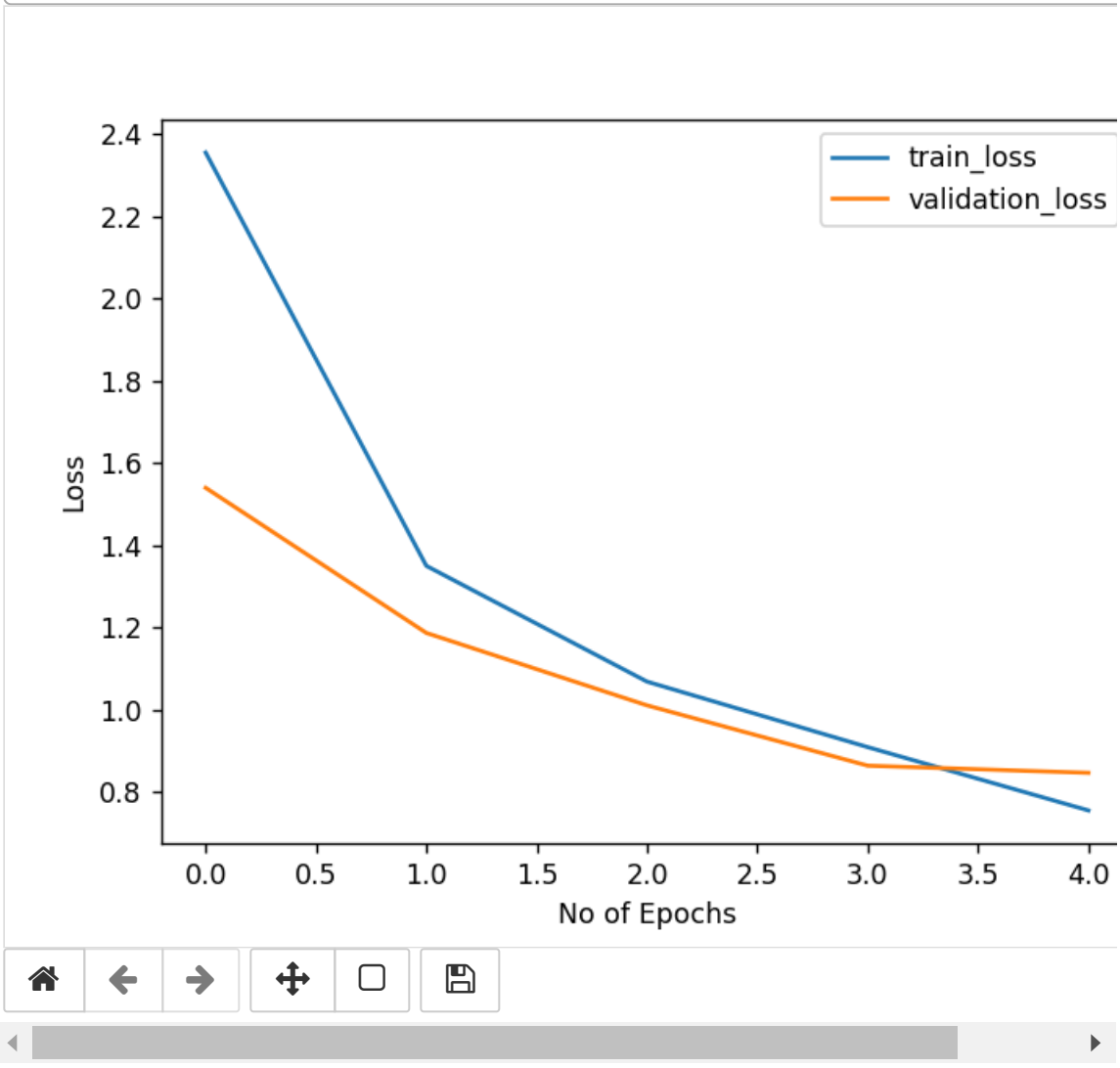
In [76]:

```
%matplotlib notebook
```

In [77]:

```
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()
```

**Figure 1**



In [39]:
```
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]

    acc = n_correct / n_total
    return acc
```

In [40]:
```
train_acc = accuracy(train_loader)
test_acc = accuracy(test_loader)
validation_acc = accuracy(validation_loader)
```

In [41]:
```
print(
    f"Train Accuracy : {train_acc}\nTest Accuracy : {test_acc}\nValida
)
```

```
Train Accuracy : 0.8853870544500329
Test Accuracy : 0.8574216632332213
Validation Accuracy : 0.8580904394412909
```

In [42]:
```
transform_index_to_disease = dataset.class_to_idx
```

In [43]:
```
transform_index_to_disease = dict(
    [(value, key) for key, value in transform_index_to_disease.items()
)
```

In [44]:
```
data = pd.read_csv(
    r"C:\Users\J POORNA CHANDER\Downloads\Plant-Disease-Detection-mair
    encoding="cp1252",
)
```

In [45]:
```
from PIL import Image
import torchvision.transforms.functional as TF
```

In [46]:
```
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_csv = data["disease_name"][index]
    print(pred_csv)
```

In [50]:
```
single_prediction(
    "C:\\Users\\J POORNA CHANDER\\Downloads\\Plant-Disease-Detection-m
)

```

```
Original :  OORNA CHANDER\Downloads\Plant-Disease-Detection-main\Plant-Di
sease-Detection-main\test_images\tomato_septoria_leaf_spot
Tomato : Septoria Leaf Spot
```

In [52]:
```
single_prediction(
    "C:\\Users\\J POORNA CHANDER\\Downloads\\Plant-Disease-Detection-m
)

```

```
Original :  OORNA CHANDER\Downloads\Plant-Disease-Detection-main\Plant-Di
sease-Detection-main\test_images\apple_healthy
Apple : Healthy
```

In [54]:
```
single_prediction(
    "C:\\Users\\J POORNA CHANDER\\Downloads\\Plant-Disease-Detection-m
)
```

```
Original :  OORNA CHANDER\Downloads\Plant-Disease-Detection-main\Plant-Di
sease-Detection-main\test_images\cherry_healthy
Cherry : Healthy
```

In [55]:
```
single_prediction(
    "C:\\Users\\J POORNA CHANDER\\Downloads\\Plant-Disease-Detection-m
)
```

```
Original :  OORNA CHANDER\Downloads\Plant-Disease-Detection-main\Plant-Di
sease-Detection-main\test_images\cherry_powdery_mildew
Cherry : Powdery Mildew
```

In [ ]:
```
1
2
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```