VIT
Vellore Institute of Technology

Apple Leaf Disease Identification and Classification using ResNet Models

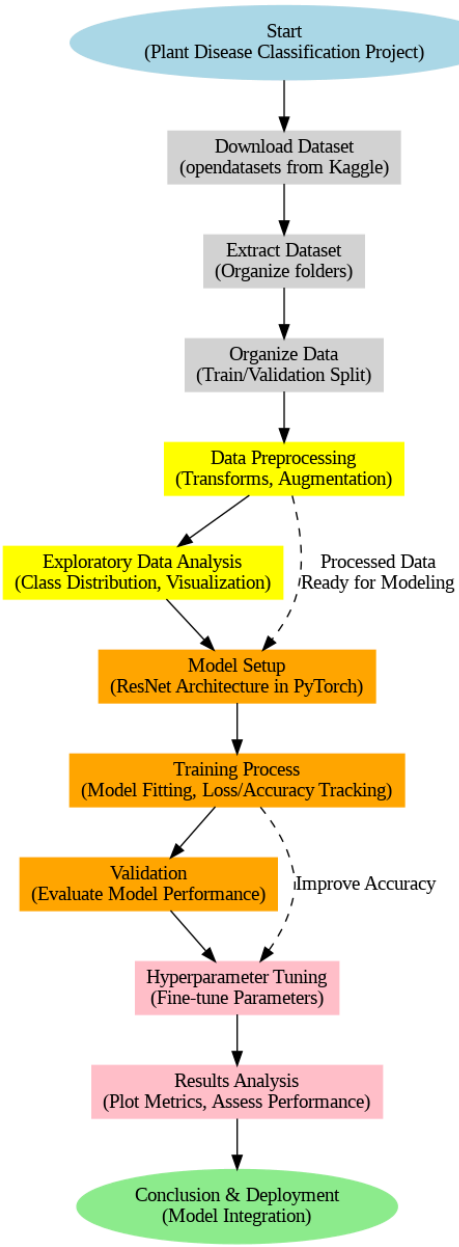
With the rapid growth of the fruit industry, apple leaf diseases have increasingly impacted apple yield and farmers' economic income. Traditional disease identification requires expertise and time, making automated detection crucial. Convolutional Neural Networks (CNN) are widely used for plant disease recognition due to their strong generalization, robustness, and automatic feature extraction. In this study, we analyzed apple leaf diseases using image segmentation and machine learning. We processed images by converting them to LAB color space, applying Otsu segmentation, and extracting disease spots. A Support Vector Machine (SVM) classifier and deep learning models like ResNet-18, ResNet-34, and VGG-16 were used for classification. ResNet-18 achieved the highest accuracy of 98.5%, outperforming other models. These findings provide an effective approach for large-scale, automated plant disease identification [3].

References

[3]:Li, Xin, and Laxmisha Rai. “Apple Leaf Disease Identification and Classification Using ResNet Models.” *2020 IEEE 3rd International Conference on Electronic Information and Communication Technology (ICEICT)*, IEEE, Nov. 2020, https://doi.org/10.1109/iceict51264.2020.9334214. Accessed 28 Feb. 2025.

Description for Apple Leaf Disease Prediction Using Resnet Architecture:

The notebook presents a deep learning approach to identifying plant diseases using a ResNet model. It utilizes a dataset of plant leaf images, applying data augmentation techniques to enhance model robustness. A custom ResNet architecture is implemented, and the model is trained using a one-cycle learning rate policy. The training process includes monitoring metrics such as training and validation loss to prevent overfitting. The final model achieves an impressive accuracy of 99.2% on the validation set, demonstrating its effectiveness in classifying plant diseases.



Nairit Das 24MAI0097

Code:

Nairit Das 24MAI0097

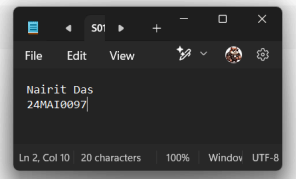
```
!pip install torchsummary
```

```
Requirement already satisfied: torchsummary in /usr/local/lib/python3.11/dist-packages (1.5.1)
```

```
!pip install opendatasets
```

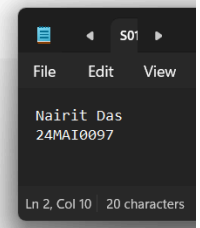
```
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from opendatasets) (4.67.1)
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (from opendatasets) (1.6.17)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from opendatasets) (8.1.8)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle->opendatasets) (1.17.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.11/dist-packages (from kaggle->opendatasets) (2025.1.31)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle->opendatasets) (2.32.3)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle->opendatasets) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.11/dist-packages (from kaggle->opendatasets) (2.3.0)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle->opendatasets) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from bleach->kaggle->opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.11/dist-packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle->opendatasets) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->kaggle->opendatasets) (3.10)
Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

```
import opendatasets as od
```




```
import opendatasets as od
od.download("https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset")
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: nairitdas24mai0097
Your Kaggle Key: ··········
Dataset URL: https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset
Downloading new-plant-diseases-dataset.zip to ./new-plant-diseases-dataset
100%|██████████| 2.70G/2.70G [02:05<00:00, 23.1MB/s]
```




```
import os                       # for working with files
import numpy as np              # for numerical computationss
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

# Define extract_path with the actual path where the dataset was extracted
extract_path = "new-plant-diseases-dataset"  # Replace with the correct path

print("Dataset extracted successfully to:", extract_path)
```

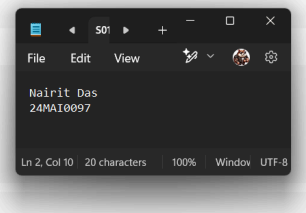
```
data_dir = "new-plant-diseases-dataset/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)"
train_dir = data_dir + "/train"
valid_dir = data_dir + "/valid"
diseases = os.listdir(train_dir)
```

```
# printing the disease names
print(diseases)
```

```
['Potato___Early_blight', 'Squash___Powdery_mildew', 'Tomato___Early_blight', 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot', 'Apple___Black_rot', 'Tomato___Spider_mites Two-spotted_spider_mite', 'P
```

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

```
Total disease classes are: 38
```




```
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 the dataset
print(f"Unique Plants are: \n{plants}")
```

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

```
# unique plants in the dataset
print(f"Unique Plants are: \n{plants}")
```

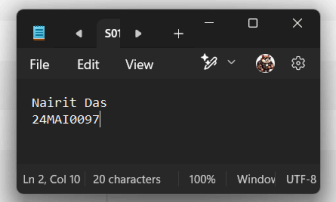
```
Unique Plants are:
['Potato', 'Squash', 'Tomato', 'Corn_(maize)', 'Apple', 'Raspberry', 'Grape', 'Peach', 'Orange', 'Cherry_(including_sour)', 'Strawberry', 'Pepper,_bell', 'Soybean', 'Blueberry']
```

```
# number of unique plants
print("Number of plants: {}".format(len(plants)))
```

```
Number of plants: 14
```

```
# number of unique diseases
print("Number of diseases: {}".format(NumberOfDiseases))
```

```
Number of diseases: 26
```

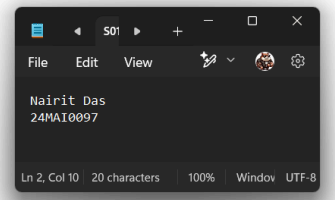



```
# Number of images for each disease
nums = {}
for disease in diseases:
    nums[disease] = len(os.listdir(train_dir + '/' + disease))

# 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
```

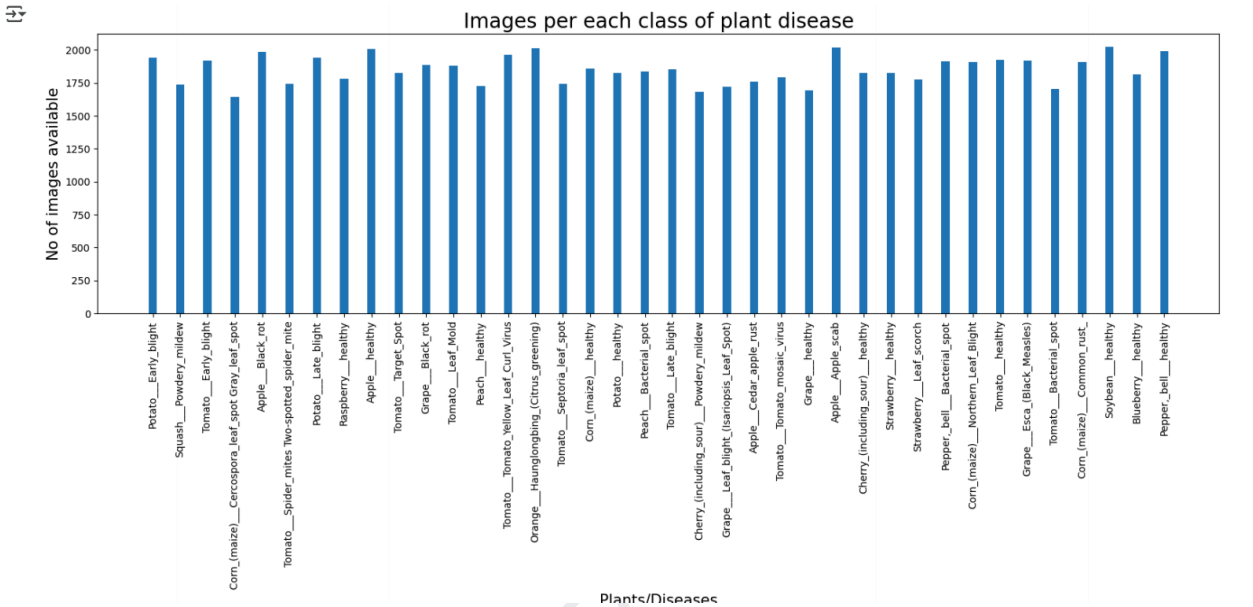
	no. of images
Potato___Early_blight	1939
Squash___Powdery_mildew	1736

```
# 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
```

Tomato___Early_blight	1920
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	1642
Apple___Black_rot	1987
Tomato___Spider_mites Two-spotted_spider_mite	1741
Potato___Late_blight	1939
Raspberry___healthy	1781
Apple___healthy	2008
Tomato___Target_Spot	1827
Grape___Black_rot	1888
Tomato___Leaf_Mold	1882
Peach___healthy	1728
Tomato___Tomato_Yellow_Leaf_Curl_Virus	1961
Orange___Haunglongbing_(Citrus_greening)	2010
Tomato___Septoria_leaf_spot	1745
Corn_(maize)___healthy	1859
Potato___healthy	1824
Peach___Bacterial_spot	1838
Tomato___Late_blight	1851



```
[ ] # plotting number of images available for each disease
    index = list(range(len(nums))) # Use the length of 'nums' for the index
    plt.figure(figsize=(20, 5))
    plt.bar(index, list(nums.values()), width=0.3) # Convert nums.values() to a list
    plt.xlabel('Plants/Diseases', fontsize=15) # Increased fontsize for x-axis label
    plt.ylabel('No of images available', fontsize=15) # Increased fontsize for y-axis label
    plt.xticks(index, list(nums.keys()), fontsize=10, rotation=90) # Increased fontsize for x-axis ticks
    plt.title('Images per each class of plant disease', fontsize=20) # Increased fontsize for title
    plt.show() # Added to display plot
```




```
    n_train = 0
    for value in nums.values():
        n_train += value
    print(f"There are {n_train} images for training")
```

```
There are 70295 images for training
```

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

```
[ ] img, label = train[0]
    print(img.shape, label)
```

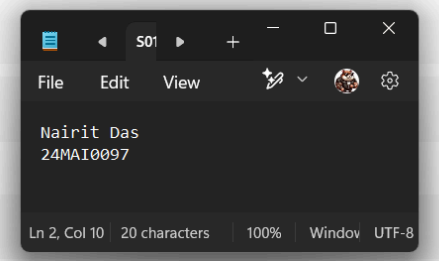
```
torch.Size([3, 256, 256]) 0
```

```
[ ] # total number of classes in train set
    len(train.classes)
```

```
38
```

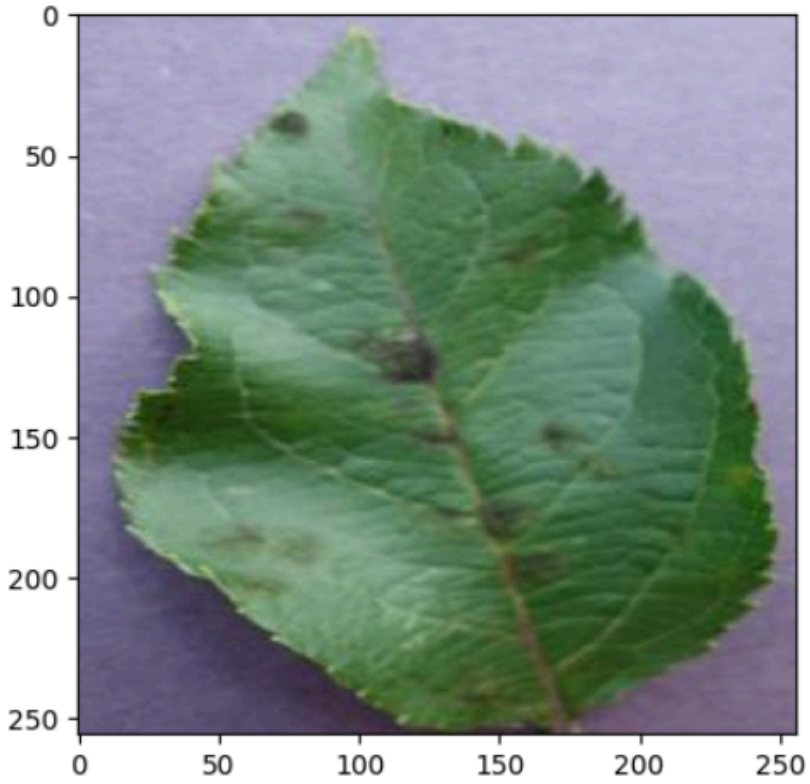
```
[ ] # for checking some images from training dataset
    def show_image(image, label):
        print("Label :" + train.classes[label] + "(" + str(label) + ")")
        plt.imshow(image.permute(1, 2, 0))
```

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



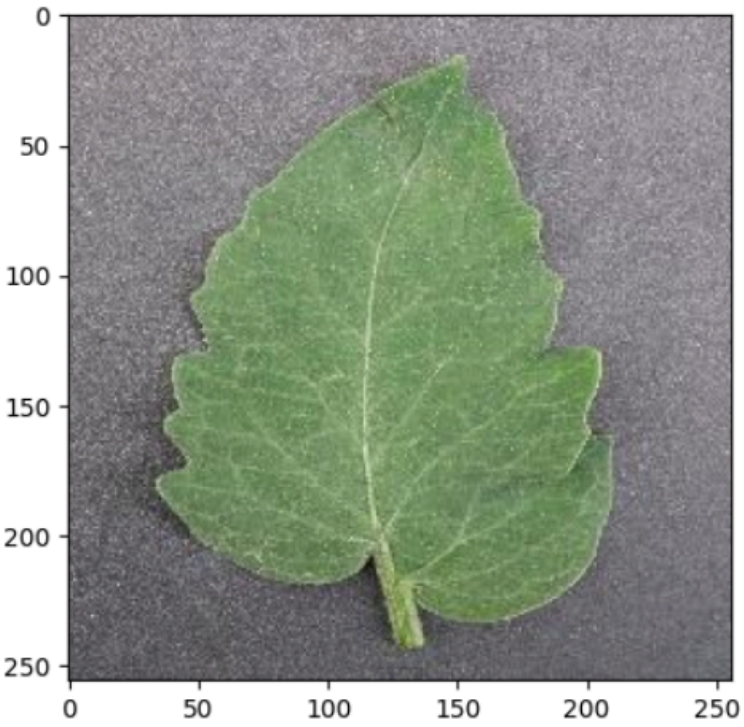
```
show_image(*train[0])
```

```
Label :Apple___Apple_scab(0)
```



```
show_image(*train[70000])
```

```
Label :Tomato___healthy(37)
```




```
# Setting the seed value
random_seed = 7
torch.manual_seed(random_seed)
```

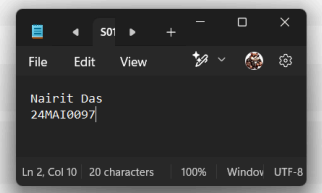
```
<torch._C.Generator at 0x7f8da26111f0>
```

```
# setting the batch size
batch_size = 32
```

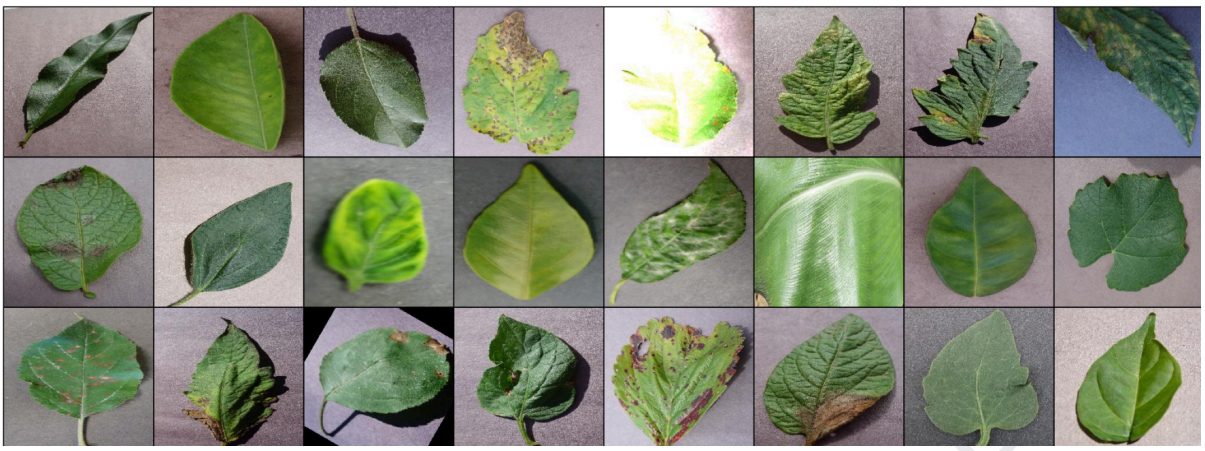
```
# DataLoaders for training and validation
train_dl = DataLoader(train, batch_size, shuffle=True, num_workers=2, pin_memory=True)
valid_dl = DataLoader(valid, batch_size, num_workers=2, pin_memory=True)
```

```
# helper function to 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
```

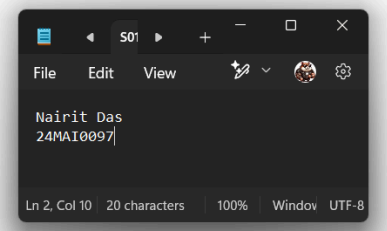
```
# Images for first batch of training
show_batch(train_dl)
```



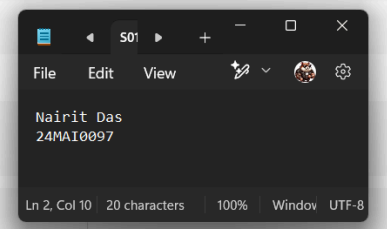
```
# Images for first batch of training
show_batch(train_dl)
```



```
# 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)
```



```
[ ] device = get_default_device()
    device

    device(type='cuda')
```

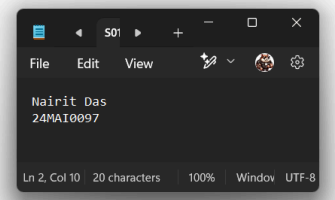



```
[ ] # Moving data into GPU
    train_dl = DeviceDataLoader(train_dl, device)
    valid_dl = DeviceDataLoader(valid_dl, device)
```

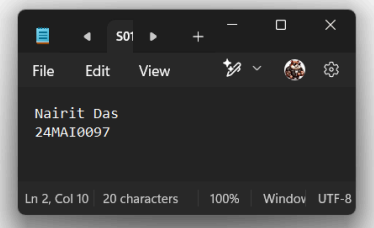
```
[ ] class SimpleResidualBlock(nn.Module):
        def __init__(self):
            super().__init__()
            self.conv1 = nn.Conv2d(in_channels=3, out_channels=3, kernel_size=3, stride=1, padding=1)
            self.relu1 = nn.ReLU()
            self.conv2 = nn.Conv2d(in_channels=3, out_channels=3, 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
```

```
[ ] # for calculating the accuracy
    def accuracy(outputs, labels):
        _, preds = torch.max(outputs, dim=1)
        return torch.tensor(torch.sum(preds == labels).item() / len(preds))


    # base class for the model
    class ImageClassificationBase(nn.Module):
```

```
[ ] # base class for the model
    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']))
```



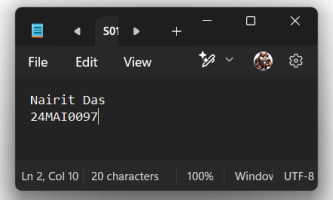

```
    # Architecture for training

    # 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.BatchNorm2d(out_channels),
                  nn.ReLU(inplace=True)]
        if pool:
            layers.append(nn.MaxPool2d(4))
        return nn.Sequential(*layers)


    # 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.res1 = nn.Sequential(ConvBlock(128, 128), ConvBlock(128, 128))

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



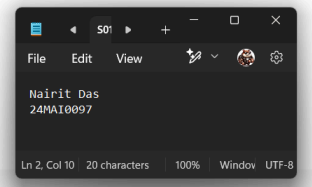
```
        out = self.res2(out) + out
        out = self.classifier(out)
        return out
```

```
[ ] # 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=38, bias=True)
  )
)
```

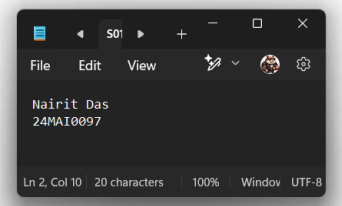



```
[ ] # getting summary of the model
    INPUT_SHAPE = (3, 256, 256)
    print(summary(model.cuda(), (INPUT_SHAPE)))
```

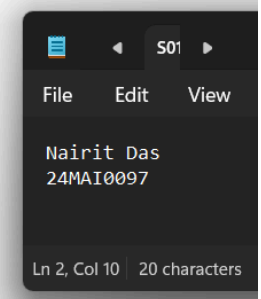
```
[ ] ----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 256, 256]           1,792
       BatchNorm2d-2         [-1, 64, 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
```

```
[ ] 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
```

```
[ ] # 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
```



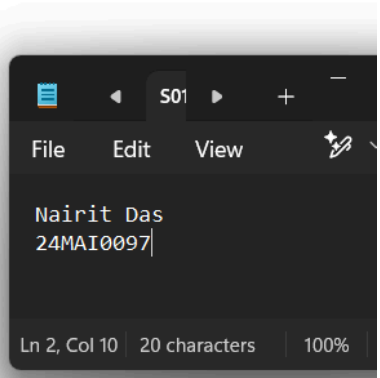
```
%%time
history = [evaluate(model, valid_dl)]
history
```

```
CPU times: user 54.6 s, sys: 5.97 s, total: 1min
Wall time: 1min 10s
[{'val_loss': tensor(3.6378, device='cuda:0'), 'val_accuracy': tensor(0.0282)}]
```

```
epochs = 2
max_lr = 0.01
grad_clip = 0.1
weight_decay = 1e-4
opt_func = torch.optim.Adam
```

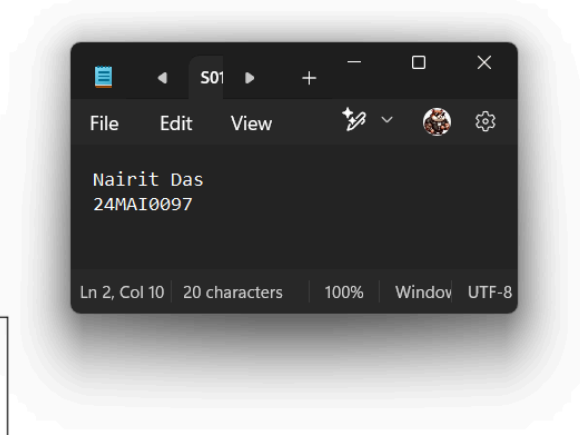
```
%%time
history += fit_OneCycle(epochs, max_lr, model, train_dl, valid_dl,
                        grad_clip=grad_clip,
                        weight_decay=1e-4,
                        opt_func=opt_func)
```

```
Epoch [0], last_lr: 0.00812, train_loss: 0.7494, val_loss: 0.5536, val_acc: 0.8310
Epoch [1], last_lr: 0.00000, train_loss: 0.1250, val_loss: 0.0271, val_acc: 0.9916
CPU times: user 15min 26s, sys: 15min 47s, total: 31min 14s
Wall time: 31min 31s
```



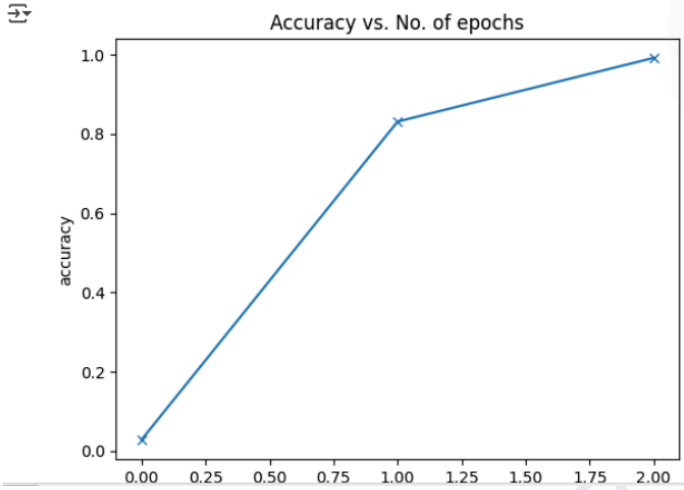

```
def plot_accuracies(history):
    accuracies = [x['val_accuracy'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
    plt.title('Accuracy vs. No. of epochs');

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.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.');
```



VALIDATION ACCURACY

```
plot_accuracies(history)
```

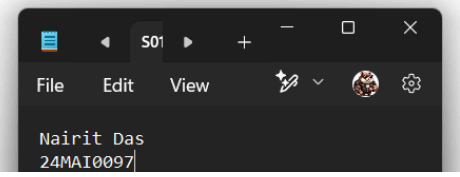


VALIDATION LOSS

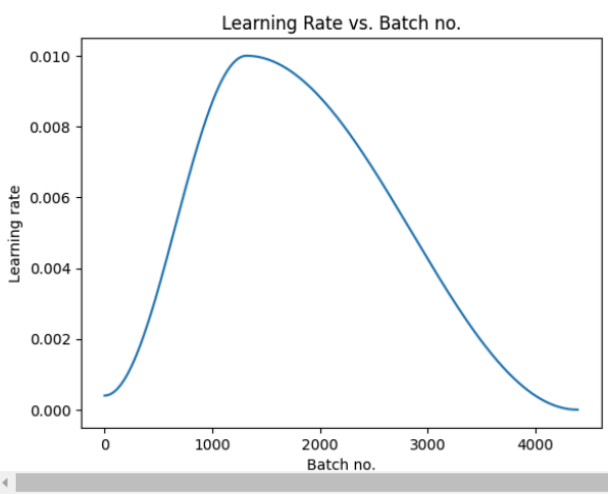
https://colab.research.google.com/drive/1gX0cDbx6rk6t8flgaccUQ8zAfqYmAsP8#scrollTo=LmXoUt-P7031&printMode=true

2/28/25, 1:18 AM RESNET.ipynb - Colab

```
 # This is formatted as code


def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'].cpu().numpy() for x in history] # Move tensors to CPU and convert to NumPy
    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');


plot_lrs(history)
```



```
plot_lrs(history)
```




```
test_dir = "new-plant-diseases-dataset/test"
test = ImageFolder(test_dir, transform=transforms.ToTensor())
```

```
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
```

File Edit View

Nairit Das
24MAI0097

Ln 2, Col 10 | 20 characters | 100% | Windov

s://colab.research.google.com/drive/1gX0cDbx6rk6t8flgaccUQ8zAfqYmAsP8#scrollTo=LmXoUt-P7031&printMode=true

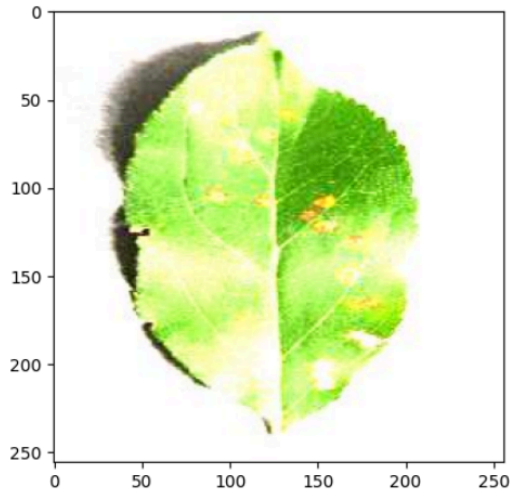
3/25, 1:18 AM RESNET.ipynb - Colab

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

```
Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust
```

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

```
Label: AppleCedarRust1.JPG , Predicted: Apple___Cedar_apple_rust
```



Accuracy:

```
Epoch [0], last_lr: 0.00812, train_loss: 0.7494, val_loss: 0.5536, val_acc: 0.8310
Epoch [1], last_lr: 0.00000, train_loss: 0.1250, val_loss: 0.0271, val_acc: 0.9916
CPU times: user 15min 26s, sys: 15min 47s, total: 31min 14s
Wall time: 31min 31s
```

Link to ALL the codes

LSTM Code:
https://colab.research.google.com/drive/1zOu4Ig2QMcPtDRedzJwLPUevBGjjSnGS?usp=sharing

BiLSTM Code:
https://colab.research.google.com/drive/1434fmGw6Nl0B-ubPEUUx6DPsw--ZiW_A?usp=sharing

RESNET Code:
https://colab.research.google.com/drive/1gX0cDbx6rk6t8flgaccUQ8zAfqYmAsP8?usp=sharing