plant-disease-identification

September 10, 2023

0.1 Calling all the required libraries

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

[2]:
```
import os
import torch
import torchvision
import tarfile
from torchvision.datasets.utils import download_url
from torch.utils.data import random_split
```

0.2 Loading Image Dataset

[3]:
```
data_dir = '/kaggle/input/new-plant-diseases-dataset/New Plant Diseases␣
 ↪Dataset(Augmented)/New Plant Diseases Dataset(Augmented)'

print(os.listdir(data_dir))
classes = os.listdir(data_dir + "/train")
print(classes)
```

```
['valid', 'train']
['Tomato___Late_blight', 'Tomato___healthy', 'Grape___healthy',
'Orange___Haunglongbing_(Citrus_greening)', 'Soybean___healthy',
'Squash___Powdery_mildew', 'Potato___healthy',
'Corn_(maize)___Northern_Leaf_Blight', 'Tomato___Early_blight',
'Tomato___Septoria_leaf_spot', 'Corn_(maize)___Cercospora_leaf_spot
Gray_leaf_spot', 'Strawberry___Leaf_scorch', 'Peach___healthy',
'Apple___Apple_scab', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Bacterial_spot', 'Apple___Black_rot', 'Blueberry___healthy',
'Cherry_(including_sour)___Powdery_mildew', 'Peach___Bacterial_spot',
'Apple___Cedar_apple_rust', 'Tomato___Target_Spot', 'Pepper,_bell___healthy',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Potato___Late_blight',
'Tomato___Tomato_mosaic_virus', 'Strawberry___healthy', 'Apple___healthy',
'Grape___Black_rot', 'Potato___Early_blight',
'Cherry_(including_sour)___healthy', 'Corn_(maize)___Common_rust_',
'Grape___Esca_(Black_Measles)', 'Raspberry___healthy', 'Tomato___Leaf_Mold',
```

```
'Tomato___Spider_mites Two-spotted_spider_mite',
'Pepper,_bell___Bacterial_spot', 'Corn_(maize)___healthy']
```

1 Data Visualization

[4]:
```
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
```

[5]:
```
dataset = ImageFolder(data_dir+'/train', transform=ToTensor())
```

[6]:
```
img, label = dataset[0]
print(img.shape, label)
```

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

[7]:
```
print(dataset.classes)
print(len(dataset.classes))
```

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust',
'Apple___healthy', 'Blueberry___healthy',
'Cherry_(including_sour)___Powdery_mildew', 'Cherry_(including_sour)___healthy',
'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
'Corn_(maize)___Common_rust_', 'Corn_(maize)___Northern_Leaf_Blight',
'Corn_(maize)___healthy', 'Grape___Black_rot', 'Grape___Esca_(Black_Measles)',
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)', 'Grape___healthy',
'Orange___Haunglongbing_(Citrus_greening)', 'Peach___Bacterial_spot',
'Peach___healthy', 'Pepper,_bell___Bacterial_spot', 'Pepper,_bell___healthy',
'Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy',
'Raspberry___healthy', 'Soybean___healthy', 'Squash___Powdery_mildew',
'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Tomato___Bacterial_spot',
'Tomato___Early_blight', 'Tomato___Late_blight', 'Tomato___Leaf_Mold',
'Tomato___Septoria_leaf_spot', 'Tomato___Spider_mites Two-spotted_spider_mite',
'Tomato___Target_Spot', 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
'Tomato___Tomato_mosaic_virus', 'Tomato___healthy']
38
```

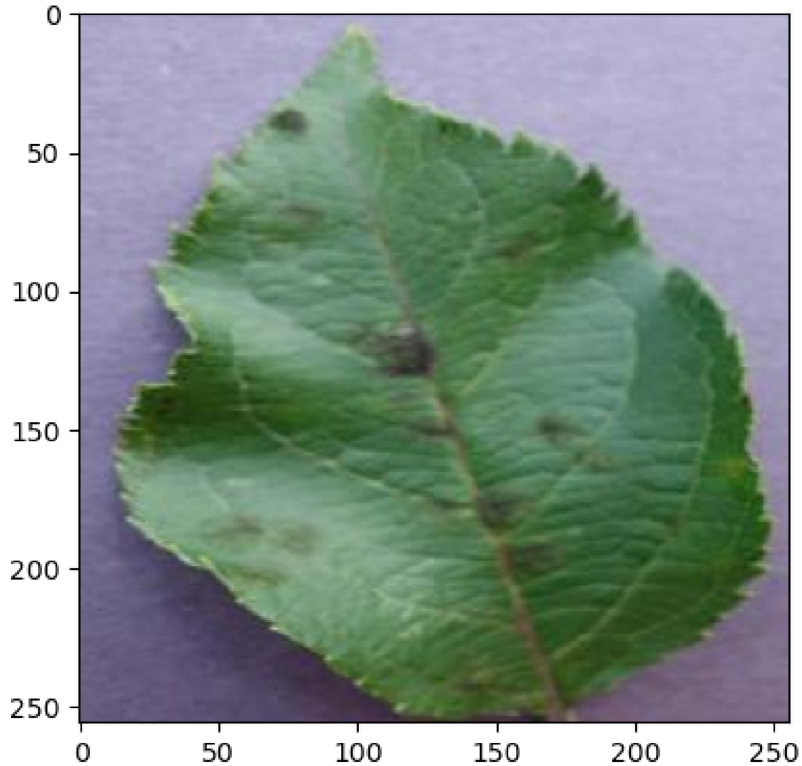
[8]:
```
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

matplotlib.rcParams['figure.facecolor'] = '#ffffff'
```

[9]:
```
def show_example(img, label):
    print('Label: ', dataset.classes[label], "("+str(label)+")")
    plt.imshow(img.permute(1, 2, 0))
```

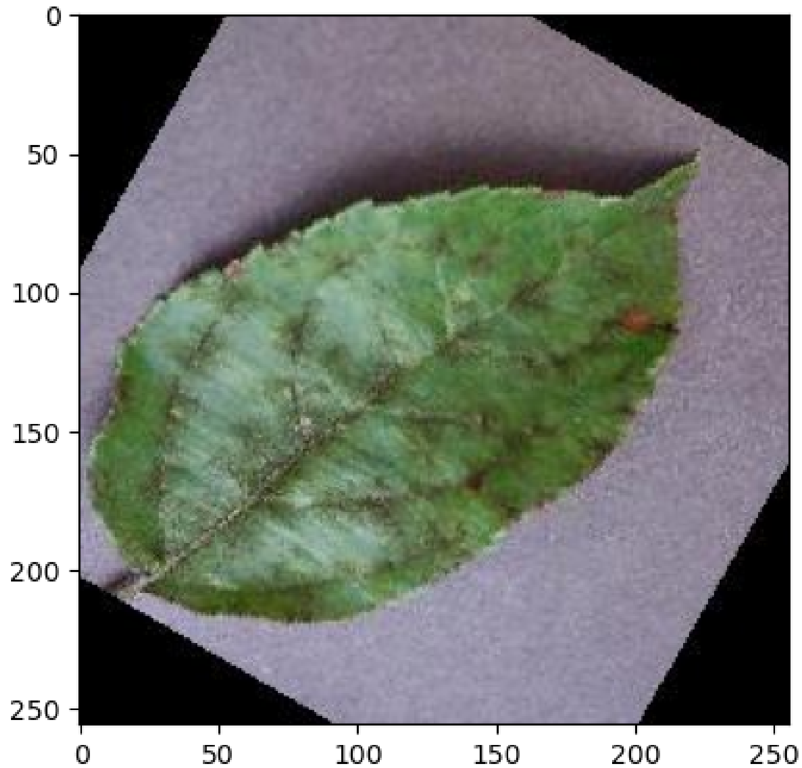
[10]:
```
show_example(*dataset[0])
```

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



[11]:
```
show_example(*dataset[1099])
```

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



2 Train Validation Set Split

[12]:
```
from torchvision import transforms
transform = transforms.Compose([
    transforms.Resize((256,256)),
    transforms.ToTensor(),
])
```

[13]:
```
train_ds = ImageFolder(data_dir+'/train', transform=transform)
valid_ds = ImageFolder(data_dir+'/valid', transform=transform)
```

[14]:
```
from torch.utils.data.dataloader import DataLoader
batch_size=32
```

[15]:
```
train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=2,␣
 ↪pin_memory=True)
val_dl = DataLoader(valid_ds, batch_size*2, num_workers=2, pin_memory=True)
```

[16]:
```
from torchvision.utils import make_grid
```

```python
def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(12, 6))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(images, nrow=8).permute(1, 2, 0))
        break
```

[17]:
```python
show_batch(train_dl)
```

3 Defining Model (CNN Model)

[18]:
```python
import torch.nn as nn
import torch.nn.functional as F
```

[19]:
```python
class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        images = images.to(device)
        labels = labels.to(device)
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        images = images.to(device)
        labels = labels.to(device)
```

```
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.
↪4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))

def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))
```

[20]:
```
from torchvision import models
class CnnModel(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        resnet = models.resnet18(weights='ResNet18_Weights.DEFAULT')
        self.resnet_layers = nn.Sequential(*list(resnet.children())[:-1])
        self.fc1 = nn.Linear(512, 128)
        self.fc2 = nn.Linear(128, 38)

    def forward(self, xb):
        x = self.resnet_layers(xb)
        x = x.view(x.size(0), -1)
        out = self.fc1(x)
        out = self.fc2(out)
        return out
```

[21]:
```
model = CnnModel()
```

4 Training

[22]:
```
device = torch.device('cuda') if torch.cuda.is_available() else torch.
↪device('cpu')
```

[23]:
```
model.to(device)
```

[23]:
```
CnnModel(
  (resnet_layers): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (4): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (5): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (6): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (7): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
```

```
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (8): AdaptiveAvgPool2d(output_size=(1, 1))
  )
  (fc1): Linear(in_features=512, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=38, bias=True)
)
```

[24]:

```python
@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 fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
```

```
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

[25]:
```
evaluate(model, val_dl)
```

[25]: {'val_loss': 3.699171781539917, 'val_acc': 0.02704545482993126}

[28]:
```
num_epochs = 10
opt_func = torch.optim.Adam
lr = 0.0001
```

[29]:
```
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

```
Epoch [0], train_loss: 0.0204, val_loss: 0.0158, val_acc: 0.9948
Epoch [1], train_loss: 0.0108, val_loss: 0.0134, val_acc: 0.9956
Epoch [2], train_loss: 0.0074, val_loss: 0.0120, val_acc: 0.9963
Epoch [3], train_loss: 0.0054, val_loss: 0.0102, val_acc: 0.9969
Epoch [4], train_loss: 0.0043, val_loss: 0.0121, val_acc: 0.9964
Epoch [5], train_loss: 0.0038, val_loss: 0.0132, val_acc: 0.9961
Epoch [6], train_loss: 0.0034, val_loss: 0.0117, val_acc: 0.9966
Epoch [7], train_loss: 0.0035, val_loss: 0.0124, val_acc: 0.9961
Epoch [8], train_loss: 0.0032, val_loss: 0.0128, val_acc: 0.9964
Epoch [9], train_loss: 0.0032, val_loss: 0.0113, val_acc: 0.9965
```

[31]:
```
PATH = '/kaggle/working/model.pth'
torch.save(model.state_dict(), PATH)
```

4.1 Conclusion

Finally we can see that we are getting an accuracy of 99.65% in the validation dataset.

[ ]:
```

```