plant.ipynb

plant.ipynb > # for training

+ Code + Markdown | Interrupt | Clear All Outputs | Go To | Restart | Variables | Outline ... | Python 3.11.3

```python
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
```

[1] ✓ 14.8s Python

```python
data_dir = "D:\\Projects & coding\\Project\\Plant Disease Detectifier\\archive\\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)
```

[2] ✓ 0.0s Python

```python
print(diseases)
```

[3] ✓ 0.1s Python

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___healthy', 'Cherry_(including_sour)___healthy', 'Cherry_(including_sc
```

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

[4] ✓ 0.1s Python

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

```python
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
```

[5] ✓ 0.1s Python

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

[6] ✓ 0.0s Python

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

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

[7] ✓ 0.0s Python

```
Number of plants: 14
```

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

[8] ✓ 0.0s Python

```
Number of diseases: 26
```

```python
# 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
```

[9] ✓ 0.2s Python

	no. of images
Apple___Apple_scab	2016
Apple___Black_rot	1987
Apple___Cedar_apple_rust	1760
Apple___healthy	2008
Blueberry___healthy	1816
Cherry_(including_sour)___healthy	1826
Cherry_(including_sour)___Powdery_mildew	1683
Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot	1642
Corn_(maize)___Common_rust_	1907
Corn_(maize)___healthy	1859
Corn_(maize)___Northern_Leaf_Blight	1908
Grape___Black_rot	1888
Grape___Esca_(Black_Measles)	1920
Grape___healthy	1692
Grape___Leaf_blight_(Isariopsis_Leaf_Spot)	1722
Orange___Haunglongbing_(Citrus_greening)	2010
Peach___Bacterial_spot	1838
Peach___healthy	1728
Pepper,_bell___Bacterial_spot	1913
Pepper,_bell___healthy	1988
Potato___Early_blight	1939
Potato___healthy	1824
Potato___Late_blight	1939

plant.ipynb

plant.ipynb > %%time

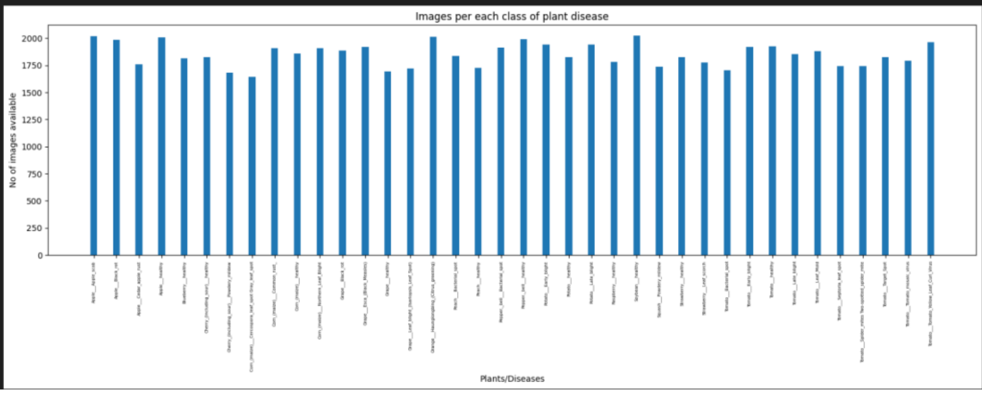
+ Code + Markdown | Interrupt Clear All Outputs Go To Restart | Variables Outline ... Python 3.11.3

Tomato___Target_Spot	1827
Tomato___Tomato_mosaic_virus	1790
Tomato___Tomato_Yellow_Leaf_Curl_Virus	1961

```python
# plotting number of images available for each disease
index = [n for n in range(38)]
plt.figure(figsize=(20, 5))
plt.bar(index, [n for n in nums.values()], width=0.3)
plt.xlabel('Plants/Diseases', fontsize=10)
plt.ylabel('No of images available', fontsize=10)
plt.xticks(index, diseases, fontsize=5, rotation=90)
plt.title('Images per each class of plant disease')
```

[10] ✓ 1.3s Python

```
Text(0.5, 1.0, 'Images per each class of plant disease')
```




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

[11] ✓ 0.1s Python

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

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

[12] ✓ 1.4s Python

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

[13] ✓ 0.2s Python

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

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

[14] ✓ 0.1s Python

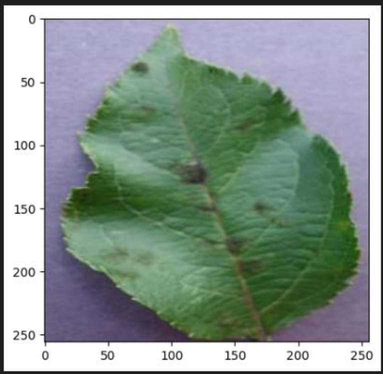
```
38
```

```python
# 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))
```

[15] ✓ 0.1s Python

```
show_image(*train[0])
```
[16] ✓ 0.6s

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



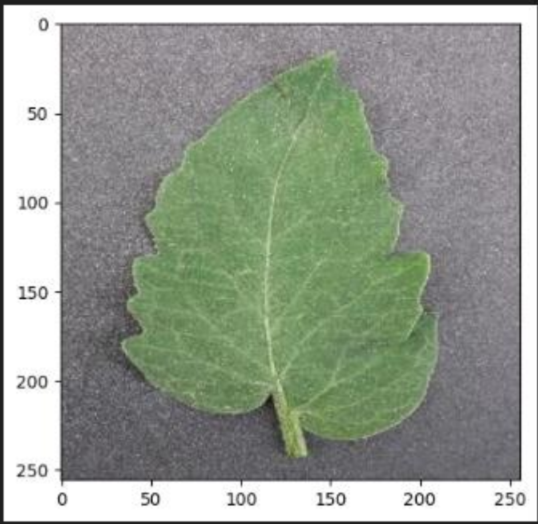

```
show_image(*train[100])
```
[17] ✓ 0.6s

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

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

[18] ✓ 1.1s

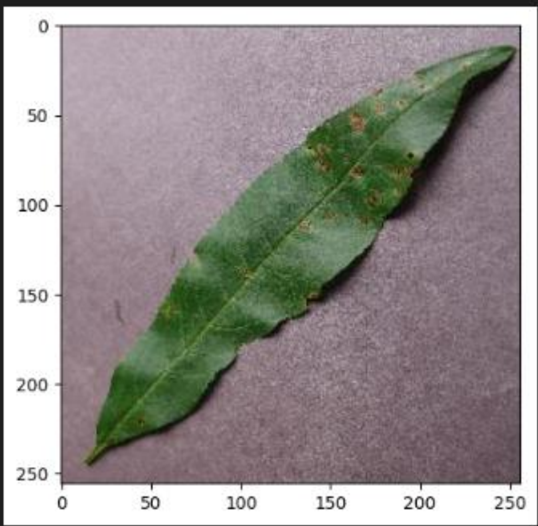
```
Label :Tomato___healthy(37)
```




```
show_image(*train[30000])
```

[19] ✓ 0.6s

```
Label :Peach___Bacterial_spot(16)
```



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

[20] ✓ 0.0s

```
<torch._C.Generator at 0x16e16691ef0>
```

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

[21] ✓ 0.0s

```python
# 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)
```

[22] ✓ 0.0s

```python
# 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
```

[23] ✓ 0.0s

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

✓ 16.7s

```python
# 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("cpu")
    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)
```

[25] ✓ 0.1s

```python
device = get_default_device()
device
```

[26] ✓ 0.1s

```
device(type='cpu')
```

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

[27] ✓ 0.0s

```python
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
```

[28] ✓ 0.0s

```python
# 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):

    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']))
```

[29] ✓ 0.1s

```python
# 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.conv3(out)
        out = self.conv4(out)
        out = self.res2(out) + out
        out = self.classifier(out)
        return out
```

[30] ✓ 0.0s

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

[31] ✓ 0.1s

```
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): 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)
  )
)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

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

[32] ✓ 0.9s

```
----------------------------------------------------------------
        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
...
Params size (MB): 25.14
Estimated Total Size (MB): 369.83
----------------------------------------------------------------
None
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

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

[34] ✓ 80m 22.0s

```
CPU times: total: 3h 12min 37s
Wall time: 1h 20min 21s

[{'val_loss': tensor(3.6378), 'val_accuracy': tensor(0.0282)}]
```

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

[35] ✓ 0.2s