🍀 PLANT DISEASE CLASSIFICATION USING RESNET-9 🍀

Corresponding Kaggle notebook can be accessed here

⚠⚠⚠DISCLAIMER: This notebook is beginner friendly, so don't worry if you don't know much about CNNs and Pytorch. Even if you have used TensorFlow in the past and are new to PyTorch, hang in there, everything is explained clearly and concisely. You will get a good overview of how to use PyTorch for image classification problems.

Description of the dataset 📝

This dataset is created using offline augmentation from the original dataset. The original PlantVillage Dataset can be found here.This dataset consists of about 87K rgb images of healthy and diseased crop leaves which is categorized into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation set preserving the directory structure. A new directory containing 33 test images is created later for prediction purpose.

Note: This description is given in the dataset itself

Our goal 🎯

Goal is clear and simple. We need to build a model, which can classify between healthy and diseased crop leaves and also if the crop have any disease, predict which disease is it.

Let's get started....

Importing necessary libraries

Let's import required modules

In [1]:
```
!pip install torchsummary
```

```
Collecting torchsummary
  Downloading torchsummary-1.5.1-py3-none-any.whl (2.8 kB)
Installing collected packages: torchsummary
Successfully installed torchsummary-1.5.1
```

We would require torchsummary library to print the model's summary in keras style (nicely formatted and pretty to look) as Pytorch natively doesn't support that

In [2]:
```
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
```

🧭 Exploring the data 🧭

Loading the data

In [3]:
```
data_dir = "../input/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)
```

In [4]:
```
# printing the disease names
print(diseases)
```

```
['Tomato___Late_blight', 'Tomato___healthy', 'Grape___healthy', 'Orange___Haunglongbing_(Citrus_greening)', 'Soybean___healthy', 'Squash___Powdery_mild
ew', '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___Bacteri
al_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_ru
st_', 'Grape___Esca_(Black_Measles)', 'Raspberry___healthy', 'Tomato___Leaf_Mold', 'Tomato___Spider_mites Two-spotted_spider_mite', 'Pepper,_bell___Bac
terial_spot', 'Corn_(maize)___healthy']
```

In [5]:
```
print("Total disease classes are: {}".format(len(diseases)))
```

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

In [6]:
```
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
```

The above cell extract the number of unique plants and number of unique diseases

In [7]:
```
# unique plants in the dataset
print(f"Unique Plants are: \n{plants}")
```

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

In [8]:
```
# number of unique plants
print("Number of plants: {}".format(len(plants)))
```

```
Number of plants: 14
```

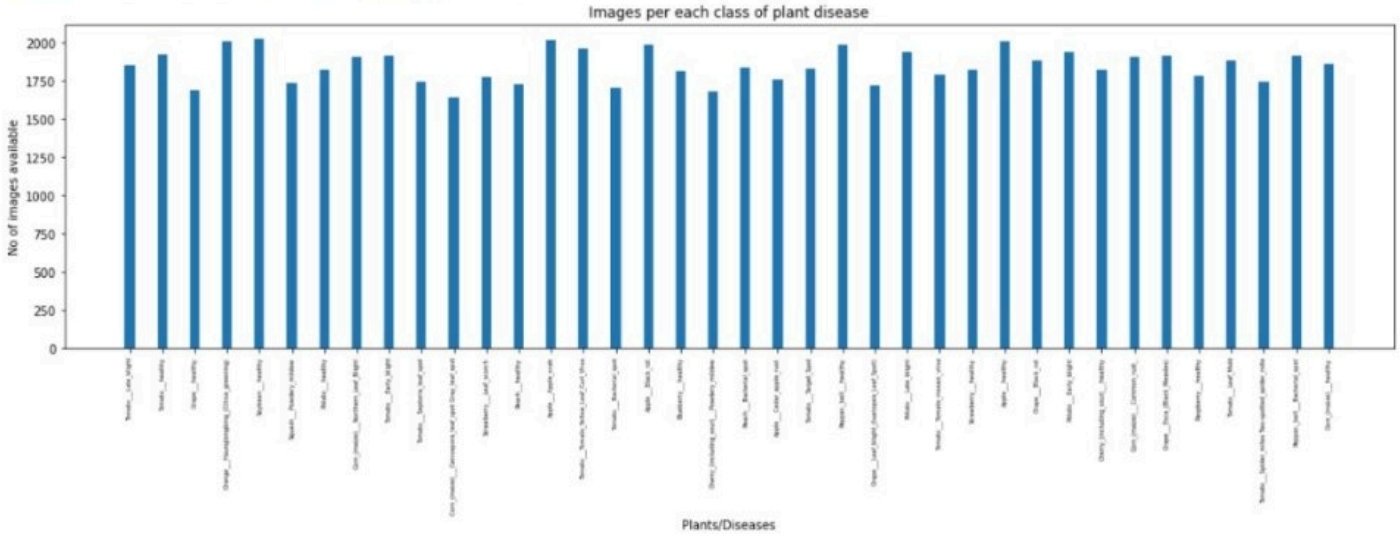
Out[10]:

	no. of images
Tomato__Late_blight	1851
Tomato__healthy	1926
Grape__healthy	1692
Orange__Haunglongbing_(Citrus_greening)	2010
Soybean__healthy	2022
Squash__Powdery_mildew	1736
Potato__healthy	1824
Corn_(maize)__Northern_Leaf_Blight	1908
Tomato__Early_blight	1920
Tomato__Septoria_leaf_spot	1745
Corn_(maize)__Cercospora_leaf_spot Gray_leaf_spot	1642
Strawberry__Leaf_scorch	1774
Peach__healthy	1728
Apple__Apple_scab	2016
Tomato__Tomato_Yellow_Leaf_Curl_Virus	1961
Tomato__Bacterial_spot	1702
Apple__Black_rot	1987
Blueberry__healthy	1816
Cherry_(including_sour)__Powdery_mildew	1683
Peach__Bacterial_spot	1838
Apple__Cedar_apple_rust	1760
Tomato__Target_Spot	1827
Pepper,_bell__healthy	1988

Visualizing the above information on a graph

In [11]:
```
# 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')
```

Out[11]:
```
Text(0.5, 1.0, 'Images per each class of plant disease')
```



We can see that the dataset is almost balanced for all classes, so we are good to go forward

🍳 Data Preparation for training 🍳

In [13]:
```
# datasets for validation and training
train = ImageFolder(train_dir, transform=transforms.ToTensor())
valid = ImageFolder(valid_dir, transform=transforms.ToTensor())
```

`torchvision.datasets` is a class which helps in loading all common and famous datasets. It also helps in loading custom datasets. I have used subclass `torchvision.datasets.ImageFolder` which helps in loading the image data when the data is arranged in this way:

---

root/dog/xxx.png

root/dog/xxy.png

root/dog/xxz.png

root/cat/123.png

root/cat/nsdf3.png

root/cat/asd932_.png

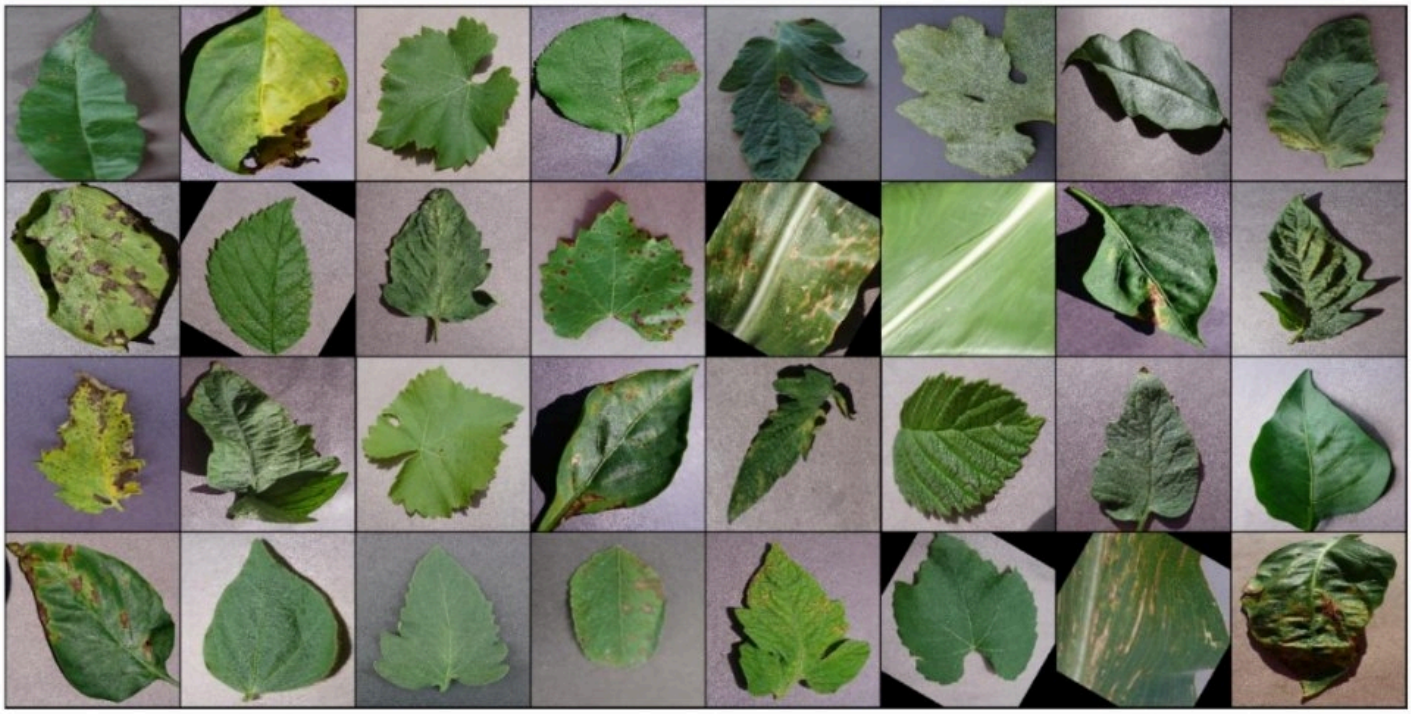
Next, after loading the data, we need to transform the pixel values of each image (0-255) to 0-1 as neural networks works quite good with normalized data. The entire array of pixel values is converted to torch tensor and then divided by 255. If you are not familiar why normalizing inputs help neural network, read this post.

Image shape

In [14]:
```
img, label = train[0]
print(img.shape, label)
```

In [24]:

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



Some helper functions

In [25]:

```
# 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)
```

Checking the device we are working with

In [26]:

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

1. **Save/Load `state_dict` (Recommended)**

When saving a model for inference, it is only necessary to save the trained model's learned parameters. Saving the model's `state_dict` with the `torch.save()` function will give you the most flexibility for restoring the model later, which is why it is the recommended method for saving models.

A common PyTorch convention is to save models using either a `.pt` or `.pth` file extension.

Remember that you must call `model.eval()` to set dropout and batch normalization layers to evaluation mode before running inference. Failing to do this will yield inconsistent inference results.

```
# saving to the kaggle working directory
PATH = './plant-disease-model.pth'
torch.save(model.state_dict(), PATH)
```

2. **Save/Load Entire Model**

This save/load process uses the most intuitive syntax and involves the least amount of code. Saving a model in this way will save the entire module using Python's pickle module. The disadvantage of this approach is that the serialized data is bound to the specific classes and the exact directory structure used when the model is saved. The reason for this is because pickle does not save the model class itself. Rather, it saves a path to the file containing the class, which is used during load time. Because of this, your code can break in various ways when used in other projects or after refactors.

```
# saving the entire model to working directory
PATH = './plant-disease-model-complete.pth'
torch.save(model, PATH)
```

Conclusion

ResNets perform significantly well for image classification when some of the parameters are tweaked and techniques like scheduling learning rate, gradient clipping and weight decay are applied. The model is able to predict every image in test set perfectly without any errors !!!!