DEEP LEARNING ASSIGNMENT - 1 2020-EE-64

```
import os
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from PIL import Image
```

Making the Class "my_DataLoader". In this we retrieve the paths of all image files in the directory and their associated class indices

```
class my_DataLoader(torch.utils.data.Dataset):
    def __init__(self, my_dir, transform=None):
        self.my_dir = my_dir
        self.transform = transform
        self.classes = os.listdir(my_dir)      # Getting our classes:
angular leaf spot,bean rust and healthy in the dataset directory
        self.class_to_idx = {cls: idx for idx, cls in
enumerate(self.classes)} # Getting an index as idx for the
corresponding classes
        self.my_img_paths = self.my_img_paths()
```

Making the "my_img_paths" method. It will give the paths of all image files within each class directory and associates each image path with its corresponding class label index. It will retirn a list of tuples where each tuple with image file path and its class label index.

```
    def my_img_paths(self):
        img_paths = []
        for cls in self.classes:
            class_dir = os.path.join(self.my_dir, cls) # Constructing
the directory path for the current class
            for img_name in os.listdir(class_dir):
                img_path = os.path.join(class_dir, img_name) #
constructs the full path to the current image file by joining the
class directory path (class_dir) with the image file name (img_name).
                img_paths.append((img_path, self.class_to_idx[cls]))
        return img_paths # It is the list of tuples containing the
paths of all image files and their associated class label indices.
```

The "**len**()" will give the the total number of images in the dataset.

```
     def __len__(self):
        return len(self.img_paths)
```

The "**getitem**()" method given an index idx it provides the corresponding image path and label stored in the self.img_paths list. It then loads the image using PIL's Image.open() method and converts it to the RGB color space. If data transformations are specified (self.transform is not None), such as resizing or normalization it applies these transformations to the image. Finally, it

returns a tuple containing the transformed image and its associated label. This method allows instances of the CustomDataset class to be indexed like lists or arrays, providing seamless access to individual data samples for training or evaluation.

```
    def __getitem__(self, idx):
        img_path, label = self.img_paths[idx] # Here we retrieve the
path of img and its corresponding label from the self.img_paths tuple
        img = Image.open(img_path).convert("RGB") # Opening image
using PIL and converting it to RGB for proper formating
        if self.transform:
            img = self.transform(img)
        return img, label

batch_size = 32
my_dir = "D:/deep learning/train"
```

Here input images are resized to a consistent size, converted to tensors and normalized to have a mean of approximately 0 and a standard deviation of approximately 1 for each channel.

```
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),    # Making image size 224 by 224
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225]), ])  # Normalizing the input image tensor

my_dir = "D:/deep learning/train"
custom_dataset = my_DataLoader(my_dir, transform=data_transform)

data_loader = DataLoader(custom_dataset, batch_size=batch_size,
shuffle=True) # Create data loader
```

Infinite data loader function

```
def infinite_dataloader(data_loader):
    while True:
        for batch in data_loader:
            yield batch
```

Now Using it

```
infinite_loader = infinite_dataloader(data_loader)
batch_images, batch_labels = next(infinite_loader)
```

For Output : We are going to get the Labels

```
batch_labels
```