dataloader

March 30, 2024

0.0.1 Submitted By: Arhum Ahmed

0.0.2 Roll No: 2020-EE-123

0.0.3 Section: B

[ ]:
```
# Importing Libraries
import os
import torch
from PIL import Image
from torchvision import transforms
```

[ ]:
```
''' A Custom Dataset class to load data.
    It returns a list of tuple elements, in format (img_array, label).
    The output data is in tensor format.
'''

class CustomDataSet():
    def __init__(self, data_dir):
        self.data_dir     = data_dir
        self.list_classes = os.listdir(data_dir) # Fetching classes using the␣
 ↪folders
        self.transform    = transforms.Compose([transforms.ToTensor()])  #␣
 ↪Transforms data to torch tensor
        self.images       = self.load_images()   # Function to load images

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image, label = self.images[idx]
        return image, label

    def load_images(self):
        data_set = []
        self.list_classes = os.listdir(self.data_dir)    # Defining the classes␣
 ↪of Images
        for i in self.list_classes:
            sub_dir   = os.path.join(self.data_dir, i)
```

```
            sub_class = os.listdir(sub_dir)
            for img in sub_class:
                img_path  = os.path.join(sub_dir, img)    # Fetching images
                img_array = self.transform(Image.open(img_path)) # Converting␣
 ↪images to tensor
                data_set.append((img_array, i))
        return data_set
```

[ ]:
```
# Extracting data using the dataset class

path_train      = 'train'
path_test       = 'test'
path_validation = 'validation'

data_set_train      = CustomDataSet(path_train)
data_set_test       = CustomDataSet(path_test)
data_set_validation = CustomDataSet(path_validation)

print('Length of dataset:',len(data_set_train))  # trying len function
print('Length of dataset:',len(data_set_test))
print('Length of dataset:',len(data_set_validation))
print('Data at index:',data_set_train[3])    # Trying getitem function
```

[ ]:
```
''' A custom DataLoader class to load data into batches
    Its features are batch sizes and shuffle mode
    It can iterate over the batch using loop
    The output is a list containing each batch-list as an element
'''

class CustomDataLoader():
    def __init__(self, dataset, batch_size, shuffle=True, infinite_iter=False):
        self.batch_size    = batch_size
        self.dataset       = dataset
        self.shuffle       = shuffle
        self.infinite_iter = infinite_iter # If true then an infinite loop of n␣
 ↪batches is covered
        self.step          = 0

    def __iter__(self):  # Function to iterate using loop
        return self

    def __next__(self): # Function to call the next batch
        if self.step >= len(self.dataset) and not(self.infinite_iter): # It is␣
 ↪used to stop the iteration of loop when limit reached
                raise StopIteration

        batch      = []
```

```
        label_out  = []
        sample_out = []
        batch_out  = []

        if self.shuffle:
            indices = torch.randperm(len(self.dataset)) # Generate random␣
↪indices for shuffling
            temp = []
            for i in indices:                # Iterate on those random indices
                temp.append(self.dataset[i]) # Add the corresponding elements␣
↪to a list
            self.dataset = temp              # Saving the shuffled list

        for j in range(0, len(self.dataset), self.batch_size): # Makes pointers␣
↪for batches
            if self.step >= len(self.dataset): # Break if limit reached
                if (self.infinite_iter): # if infinite_iter True then restart␣
↪the counter
                    self.step = 0
                else:
                    break
            for _ in range(0+j, j+self.batch_size): # Iterate over the batch␣
↪pointers
                if self.step >= len(self.dataset):  # Break if limit reached
                    if (self.infinite_iter):  # if infinite_iter True then␣
↪restart the counter
                        self.step = 0
                    else:
                        break
                sample, lbl = self.dataset[self.step]
                sample_out.append(sample)
                label_out.append(lbl)
                self.step += 1

            batch_out.append(tuple(sample_out))
            batch_out.append(tuple(label_out))
            batch.append(batch_out)
            batch_out  = []
            label_out  = []
            sample_out = []

        return batch
```

[ ]:
```
# Using the data extracted to create batches using custom dataloader

# Batches are retrived, when infinite_inter=False
```

```
train_data = CustomDataLoader(data_set_train, batch_size=32, shuffle=True,␣
 ↪infinite_iter=False)

for batch in train_data:
    print(batch)
```