ECE695DL: Homework 2

Ashutosh Mishra

February 2021

hw02_ImageNet_Scrapper.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Feb  4 21:23:14 2021

@author: Ashutosh Mishra
"""
import torchvision,torch, torch.utils.data, glob,os, numpy, PIL, argparse, requests,logging,
from PIL import Image
from requests.exceptions import
ConnectionError,ReadTimeout,TooManyRedirects,MissingSchema,InvalidURL

parser = argparse.ArgumentParser(description ='HW02 Task1')
parser.add_argument('--subclass_list',nargs ='*',type =str,required = True)
parser.add_argument('--images_per_subclass', type =int,required = True)
parser.add_argument('--data_root', type =str , required =True )
parser.add_argument('--main_class',type =str , required =True )
parser.add_argument('--imagenet_info_json',type =str,required = True )
args, args_other = parser.parse_known_args()



#Reference for get_image method:https://github.com/johancc/ImageNetDownloader3
def get_image(img_url,class_folder):
    if len(img_url) <= 1:
        return 0
    try:
        img_resp = requests.get(img_url,timeout = 1)
    except ConnectionError:
        #print('Connection Error')
        return 0
    except ReadTimeout:
```

```
    #print("Read Time Out")
    return 0
except TooManyRedirects :
    #print("Too Many Redirects")
    return 0
except MissingSchema:
    #print("Missing Schema")
    return 0
except InvalidURL:
    #print("Invalid URL")
    return 0

if not 'content-type' in img_resp.headers:
    #print("Content type error")
    return 0
if not 'image' in img_resp.headers['content-type']:
    #print("Image type error")
    return 0
if (len(img_resp.content)< 1000):
    return 0

img_name = img_url.split('/')[-1]
img_name = img_name.split("?")[0]

if (len(img_name)<=1):
    #print('Name Error')
    return 0
if not 'flickr' in img_url:
    #print("Flicker Error")
    return 0

try:
    img_file_path = os.path.join(class_folder,img_name)
    with open(img_file_path,'wb') as img_f:
        img_f.write(img_resp.content)

        # Resize image to 64x64
    im = Image.open(img_file_path)
    if im.mode != 'RGB':
        im = im.convert(mode='RGB')
    im_resized = im.resize((64,64),Image.BOX)
    im_resized.save(img_file_path)
    return 1

except:
    #print("File Error")
```

```
        return 0

#Returns dictionary in the form ('class_name':'fileID')
def create_dict(path):          #path = path to the imagenet json file
    with open(path) as file:
        data = json.load(file)
    key = list(data.keys())
    val = list(data.values())
    arr = [];
    db  = dict()
    for i in range(len(key)):
        arr.append(val[i]['class_name'])
        db[arr[i]] = key[i]
    return db

#Returns the list of URLs based on subclass_list
def get_url(database,subclass_list):
    base_url = 'http://www.image-net.org/api/text/imagenet.synset.geturls?wnid='
    url_list = []
    for i in range(len(subclass_list)):
        url_list.append(base_url+database[subclass_list[i]])
    return url_list


############################################################
class_folder = os.path.join(args.data_root,args.main_class)

try:
    os.makedirs(class_folder,exist_ok=True)
except OSError:
    pass

database = create_dict(args.imagenet_info_json)
urls = get_url(database,args.subclass_list)
id = []
for i in urls:
    resp = requests.get(i)
    id.append([url.decode('utf -8') for url in resp.content.splitlines()])

for i in range(len(args.subclass_list)):
    count = 0
    for j in range(len(id[i])):
        get_image(id[i][j], class_folder)
        count = len([name for name in
        os.listdir(class_folder)])-i*args.images_per_subclass
        #print('Downloading...\t'+args.subclass_list[i]+'....'+str(count))
```

```
        if(count == args.images_per_subclass):
            break
```

---

hw02_imagenet_task2.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb  9 20:59:34 2021

@author: Ashutosh Mishra
"""

import torchvision,torchvision.transforms as tvt,torch,torch.utils.data, glob,os,
numpy as np, PIL, argparse
from PIL import Image
from torch.utils.data import DataLoader, Dataset

seed = 0
torch.random.manual_seed(seed)

parser = argparse.ArgumentParser(description ='HW02 Task2')
parser.add_argument('--imagenet_root',type =str,required=True)
parser.add_argument('--class_list',nargs ='*',type =str,required=True)
args,args_other = parser.parse_known_args()

class ImageDataset(Dataset):

     def __init__(self,root,class_list,transform):
         self.label_array = []
         self.image_list = []
         self.transform = transform

         for i in range(len(class_list)):
             path = os.path.join(root,class_list[i])
             img_path = glob.glob(path+'/*')
             for j in img_path:
                 self.label_array.append(i)
                 im = Image.open(j)
```

```
                self.image_list.append(im)


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

    def __getitem__(self,index):
        image = self.transform(self.image_list[index])
        label =[]
        if (self.label_array[index] ==0):
            label=[1,0]
        if (self.label_array[index] ==1):
            label=[0,1]

        return image,label



###


transform = tvt.Compose([tvt.ToTensor(),tvt.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))]
)

train_dataset = ImageDataset(os.path.join(args.imagenet_root,'Train'),
args.class_list,transform)
train_data_loader = torch.utils.data.DataLoader
(dataset=train_dataset,batch_size=10,shuffle =True,num_workers=0)

val_dataset = ImageDataset(os.path.join(args.imagenet_root,'Val'),
args.class_list, transform)
val_data_loader = torch.utils.data.DataLoader(dataset=val_dataset,
batch_size=10,shuffle =True,num_workers =0)


dtype  = torch.float32
device = torch.device("cuda :0" if torch.cuda.is_available() else "cpu")
epochs = 40
D_in,H1,H2,D_out = 3*64*64,1000,256,2
w1 = torch.randn(D_in, H1, device=device,dtype=dtype)
w2 = torch.randn(H1,H2,device=device,dtype=dtype)
w3 = torch.randn(H2,D_out,device=device,dtype = dtype)
learning_rate = 1e-9
c=-1
epoch_loss=0
file = open('./output.txt','w')
```

```
for t in range(epochs):

    for i,data in enumerate(train_data_loader):
        inputs,labels = data
        y1 = labels[0]
        y2 = labels[1]
        y1 = [y1.tolist()]
        y2 = [y2.tolist()]
        y  = torch.transpose(torch.tensor(y1+y2),0,1)

        #inputs = inputs.to(device)
        #labels = labels.to(device)
        x = inputs.view(-1, D_in)
        h1 = x.mm(w1)
        h1_relu = h1.clamp(min=0)
        h2 = h1_relu.mm(w2)
        h2_relu = h2.clamp(min =0)
        y_pred = h2_relu.mm(w3)

        loss = (y_pred - y).pow(2).sum().item()
        epoch_loss += loss
        y_error = y_pred - y

        grad_w3 = h2_relu.t().mm(2*y_error)
        h2_error = 2.0*y_error.mm(w3.t())
        h2_error[h2<0] = 0
        grad_w2 = h1_relu.t().mm(2*h2_error)
        h1_error = 2.0*h2_error.mm(w2.t())
        h1_error[h1<0] = 0
        grad_w1 = x.t().mm(2*h1_error)

        w1 -= learning_rate * grad_w1
        w2 -= learning_rate * grad_w2
        w3 -= learning_rate * grad_w3

    print('Epoch %d:\t %0.4f'%(t, epoch_loss))
    file.write('Epoch %d:\t %0.4f\n'%(t, epoch_loss))
    epoch_loss = 0

torch.save({'w1':w1,'w2':w2,'w3':w3},'./wts.pkl')


pred_list = []
val_loss = 0

with open("./wts.pkl",'rb') as f:
```

```
    wts = torch.load(f)

wt1 = wts['w1']
wt2 = wts['w2']
wt3 = wts['w3']


for i,data in enumerate(val_data_loader):
    inputs,labels = data
    y1 = labels[0]
    y2 = labels[1]
    y1 = [y1.tolist()]
    y2 = [y2.tolist()]
    y  = torch.transpose(torch.tensor(y1+y2),0,1)

    #inputs = inputs.to(device)
    #labels = labels.to(device)
    x = inputs.view(-1, D_in)
    h1 = x.mm(wt1)
    h1_relu = h1.clamp(min=0)
    h2 = h1_relu.mm(wt2)
    h2_relu = h2.clamp(min =0)
    y_pred = h2_relu.mm(wt3)


    yp = torch.zeros_like(y_pred)

    idx = 0
    for i in y_pred:
        yp[idx] = (i.max()==i)
        idx+=1

    loss = (y_pred - y).pow(2).sum().item()
    val_loss += loss
    pred_list.extend((y[:,0]==yp[:,0]).tolist())

acc = sum(pred_list)*100/len(pred_list)
val_loss_str = 'Val Loss: '+str(val_loss)
acc_str = 'Val Accuracy: '+str(acc)+'%'
print(val_loss_str)
print(acc_str)
file.write('\n'+val_loss_str+'\n')
file.write(acc_str)
file.close()
```