BME695DL/ECE695DL: Homework 2

Spring 2021
Due Date: Monday, Feb. 15,2021 (11:59pm)

Turn in your solutions via BrightSpace.

1 Introduction

This homework consists of the following three goals:

- To introduce you to ImageNet, the world's largest database of images for developing image recognition algorithms with deep learning techniques.
- To familiarize you with the data loading facilities provided by Torchvision and for you to customize data loading to your needs.
- To have you hand-craft the backpropagation of loss by a direct calculation of the gradients of the loss with respect to the learnable parameters for a neural network with an input layer, two hidden layers, and one output layer.

A good place to start for this homework is to visit the website

`http://image-net.org/explore`

in order to become familiar with how the dataset of now over 14 million images is organized. As you will notice, the organization is hierarchical in the form of a tree structure. When you click on a node of the tree in the left window, you will see on the right (if you wait long enough for the thumbnails to download) the images corresponding to that category. It is best to click on the leaf nodes of the tree since the number of images for the non-terminal nodes will, in general, be larger and would take longer for the thumbnails to download. Fig. 1 shows a screenshot of Treemap visualization of the "domestic cat" category at the website listed above.

The URL's to the ImageNet images are stored in files with names like "n03173929" where "n" is a designator for such files and the number that follows is the

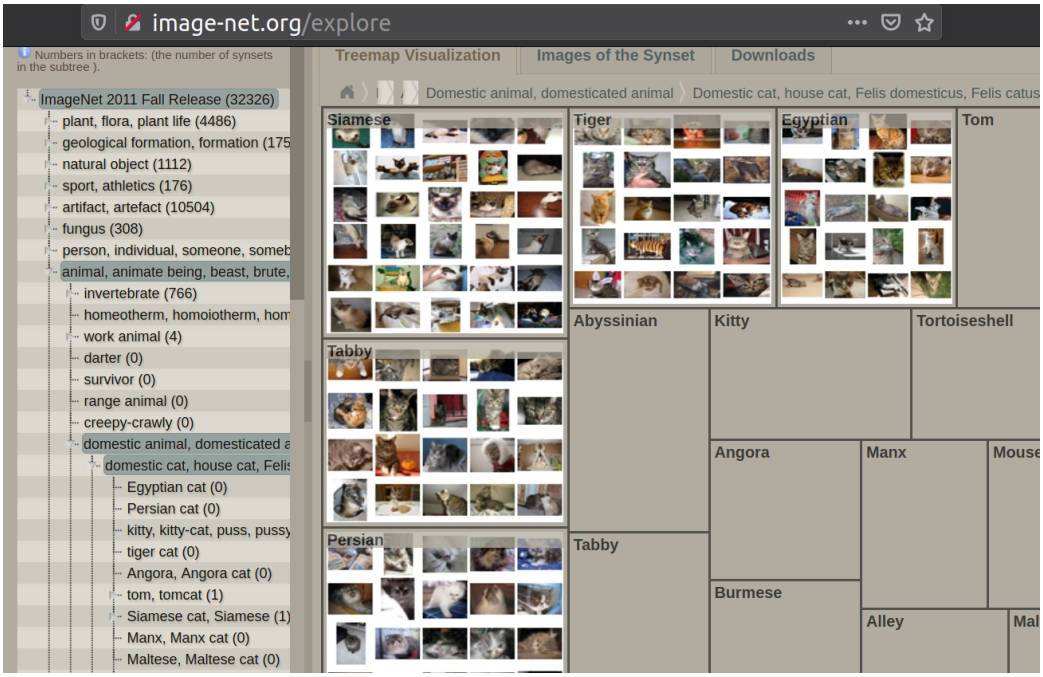

Figure 1: ImageNet Treemap visualization for the "domestic cat" category.

actual identifier for the file. For example, the URL's to the images for the "domestic cat" category reside in a file named "n02121808". That begs the question: Who or what is the keeper of the mappings from the symbolic names of the different image categories and the corresponding text files that store the URLs. That mapping resides in a file called

```
imagenet_class_info.json
```

If you have not encountered a JSON file before, JSON stands for "JavaScript Object Notation". It's purely a text file formatted as a sequence of "attribute-value" pairs that has become popular for several different kinds of data exchange between computers. Shown below is one of the entries in the very large file mentioned above:

```
"n02121808": {"img_url_count": 1831,
              "flickr_img_url_count": 1176,
              "class_name": "domestic cat" }
```

What this says is that the URLs for the "domestic cat" category are to be found in the ImageNet file named "n02121808" You will be provided with

the `imagenet_class_info.json` file or you can download it directly from GitHub.

With that as an introduction to ImageNet, the sections that follow outline the required programming steps for each programming task. The class, variable, and method names, etc program-defined attributes are not strict. However, make sure to follow the file naming, input argument names and output file format specifications that are required for the evaluation. You won't need GPU for completing this homework.

For the training task, your homework will involve training a simple neural network that consists of an input layer, two hidden layers, and one output layer. We will use the matrix w_1 to represent the link weights between the input and the first hidden layer, the matrix w_2 the link weights between the first hidden layer and the second hidden layer, and, finally, the matrix w_3 the link weights between the second hidden layer and the output.

For each hidden layer, we will use the notation hi as the output before the application of the activation function and hi_{relu} for the output after the activation. So if x is the vector representation of the input data, we have the following relationships in the forward direction:

$$\begin{aligned} h1 &= x.mm(w1) \\ h1_{relu} &= h1.clamp(min = 0) \\ h2 &= h1_{relu}.mm(w2) \\ h2_{relu} &= h2.clamp(min = 0) \\ y_{pred} &= h2_{relu}.mm(w3) \end{aligned}$$

where $.mm()$ does for tensors what $.dot()$ does for Numpy's ndarrays. Basically, mm stands for matrix multiplication. Remember that with tensors, a vector is a one-row tensor. That is, when an n-element vector stored in a tensor, its shape is $(1, n)$. So what you see in the first line, "$h1 = x.mm(w1)$" amounts to multiplying a matrix $w1$ with a vector x.

Before listing the tasks, you need to also understand how the loss can be backpropagated and the gradients of loss computed for simple neural networks. The following 3-step logic involved is as follows for the case of MSE loss for the last layer of the neural network. You repeat it backwards for the rest of the network.

- The loss at the output layer:

$$L = (y - y_{pred})^t (y - y_{pred})$$

where y is the groundtruth vector and y_{pred} the predicted vector. The superscript 't' stands for transpose.

- Propagating the loss backwards and calculating the gradient of the loss with respect to the parameters in the link weights involves the following three steps:

1. Find the gradient of the loss with respect to the link matrix w_3 by:

$$grad_{w3} = h2^t_{relu}.mm(2 * y_{error})$$

where $y_{error} = y - y_{pred}$. Note that, logically speaking, the arithmetic operation shown here is an outer product between two vectors. The first operand in this outer product is the post-activation output from the second hidden later. And the second operand is twice the prediction error y_{error}.

2. Propagate the error to the post-activation point in the hidden layer h_2 by

$$h2_{error} = 2 * y_{error} \cdot mm(w_3^t)$$

3. Propagate the error past the activation in the layer $h2$ by

$$h2_{error}[h2 < 0] = 0$$

2 Recommended Python Packages

The following are some recommended python packages.

`torchvision`, `torch.utils.data`, `glob`,`os`, `numpy`, `PIL`, `argparse`, `requests` , `logging`, `json`

Note that the list is not exhaustive.

3 Programming Tasks

3.1 Task1: Scraping and Downsampling ImageNet Subset

1. Download the provided `imagenet_class_info.json` file. You can use the `json` python package to read this file.

2. Create `hw02_ImageNet_Scrapper.py`.

3. Specify the following input arguments

```
    ...
    #initial import calls
import argparse
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()

```

   Now call these user specified input arguments in your code using, *e.g.*, `args.images_per_subclass`. The python call itself would look like as follows

```
python hw02_ImageNet_Scrapper.py --subclass_list 'Siamese cat' 'Persian cat' 'Burmese cat' \
--main_class 'cat' --data_root <imagenet_root>/Train/ \
--imagenet_info_json <path_to_imagenet_class_info.json> --images_per_subclass 200
```

   Note that the arguments in the angular brackets are your system specific paths. The above call should download, downsample and save 200 flickr images for 'Siamese cat', 'Persian cat', and 'Burmese cat' each. The images should be stored in `<imagenet_root>/Train/cat` folder.

4. Understand the data-structure of `imagenet_class_info.json` and how to retrieve the necessary information from the ImageNet dataset. The following is an entry in the given .json file

```
    ...
    "n02123597": {"img_url_count": 1739,
```

```
      "flickr_img_url_count": 1434,
      "class_name": "Siamese cat"}
    ...

```

You can retrieve the url list corresponding to 'Siamese cat' subclass using the unique identifier 'n02123597'. If you open the following link in your browser, you will see the list of urls corresponding to the images of 'Siamese cat'.

`http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n02123597`.

You can use the following call in your python code to retrieve the list.

```
    #the_url contains the required url to obtain the full
                                list using an identifier
    #the_list_url = http://www.image-net.org/api/text/
                                imagenet.synset.geturls?
                                wnid=n02123597
    resp = requests.get(the_list_url)
    urls = [url.decode('utf-8') for url in resp.content.
                                splitlines()]
        for url in  urls:
        # download and downsample the required number of
                                images

```

5. The following is a function skeleton to download an image from a given url. You're free to handle the `try ..except` blocks in your own way.

```
'''
    Reference:https://github.com/johancc/
                                ImageNetDownloader
'''
import requests
from PIL import Image

from requests.exceptions import ConnectionError,
                                ReadTimeout,
                                TooManyRedirects,
                                MissingSchema, InvalidURL
def get_image(img_url, class_folder):
    if len(img_url) <= 1:
        #url is useless Do something
    try:
        img_resp = requests.get(img_url, timeout = 1)
    except ConnectionError:
    #Handle this exception
    except ReadTimeout:
    #Handle this exception
    except TooManyRedirects:
```

```
        #handle exception
        except MissingSchema:
        #handle exception
        except InvalidURL:
        #handle exception

        if not 'content-type' in img_resp.headers:
        #Missing content. Do something
        if not 'image' in img_resp.headers['content-type']:
        # The url doesn't have any image. Do something
        if (len(img_resp.content) < 1000):
            #ignore images < 1kb

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

        if (len(img_name) <= 1):
            #missing image name
        if not 'flickr' in img_url:
            # Missing non-flickr images are difficult to
                                    handle. Do something.
        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)
        #Overwrite original image with downsampled image
        im_resized.save(img_file_path)
```

6. The desired output from the image scrapper is that you should be able to download 600 (200×3) training images for the `cat` class and 600 training images for the `dog` class.

7. Follow the following folder structure for saving your training and validation images. `<imagenet_root>/Train/cat/`, `<imagenet_root>/Train/dog/`, `<imagenet_root>/Val/cat/`, `<imagenet_root>/Val/dog/`. You can use `os.path.join(...)` and `os.mkdir(...)` for creating the required folder structure.

8. After the successful implementation of `hw02_ImageNet_Scrapper.py`, you can download the required training and validation sets for Task2 using the following four command-line calls (in any order).

```
python hw02_ImageNet_Scrapper.py --subclass_list 'Siamese cat' 'Persian cat' 'Burmese cat' \
--main_class 'cat' --data_root <imagenet_root>/Train/ \
--imagenet_info_json <path_to_imagenet_class_info.json> --images_per_subclass 200


python hw02_imagenetScraper.py --subclass_list 'hunting dog' 'sporting dog' 'shepherd dog' \
--main_class 'dog' --data_root <imagenet_root>/Train/ \
--imagenet_info_json <path_to_imagenet_class_info.json> --images_per_subclass 200


python hw02_ImageNet_Scrapper.py --subclass_list 'domestic cat' 'alley cat' \
--main_class 'cat' --data_root <imagenet_root>/Val/ \
--imagenet_info_json <path_to_imagenet_class_info.json> --images_per_subclass 100


python hw02_ImageNet_Scrapper.py --subclass_list 'working dog' 'police dog' \
--main_class 'dog' --data_root <imagenet_root>/Val/ \
--imagenet_info_json <path_to_imagenet_class_info.json> --images_per_subclass 100
```

4 Task2: Data Loading, Training, and Testing

1. Create `hw02_imagenet_task2.py`
2. Use the following `argparse` arguments

```python
import argparse
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()
```

The argument `imagenet_root` corresponds to the top folder containing both Train and Val subfolders as created in Task1. The following is an example call to this script

```
python hw02_imagenet_task2.py --imagenet_root <path_to_imagenet_root> --class_list 'cat' 'dog'
```

4.1 Sub Task1: Creating a Customized Dataloader

Note that you're free to choose your own program-defined class and variable names. You might find the `glob` python package useful for retrieving the list of images from a folder. Make sure to use the input arguments and also avoid using any hard-coded initialization in the class methods. All the required class or method variables for completing this task can be derived from the input arguments or should be initialized from the calling routines.

```
    ...
from torch.utils.data import DataLoader, Dataset
class your_dataset_class(Dataset):
    def __init__(...):
        '''
        Make use of the arguments from argparse
        initialize your program-defined variables
        e.g. image path lists for cat and dog classes
        you could also maintain label_array
        0 -- cat
        1 -- dog
        Initialize the required transform
        '''
    def __len__(...):
        '''
        return the total number of images
        refer pytorch documentation for more details

        '''
    def __getitem__(...):
        '''
        Load color image(s), apply necessary data conversion
                                and transformation
        e.g. if an image is loaded in HxWXC  (Height X Width
                                X Channels) format
        rearrange it in CxHxW format, normalize values from 0
                                -255 to 0-1
        and apply the necessary transformation.

        Convert the corresponding label in 1-hot encoding.
        Return the processed images
        and labels in 1-hot encoded format
        '''


```

After the successful implementation of this class, you can use the following template to create the dataloaders for the training and validation sets.

```
transform = tvt.Compose([tvt.ToTensor(), tvt.Normalize((0.5,
                                0.5, 0.5), (0.5, 0.5, 0.5))])
train_dataset = your_dataset_class(...,transform,...)
train_data_loader = torch.utils.data.DataLoader(dataset=
                                train_dataset,
                                          batch_size=10,
                                          shuffle=True,
                                          num_workers=4)

val_dataset  = your_dataset_class(...,transform,...)
val_data_loader = torch.utils.data.DataLoader(dataset=
                                val_dataset,
```

```
                                                batch_size=10,
                                                shuffle=True,
                                                num_workers=4)

```

4.2 Sub Task2: Training

For this task train the three layer neural network using the code shown below. The code is shown only to give you an idea of how you can structure your program. But it should get you started.

```
import torch

#TODO Follow the recommendations from the lecture notes to
                                ensure reproducible results

dtype = torch.float64

device = torch.device("cuda:0" if torch.cuda.is_available()
                                else "cpu")

epochs = 40 #feel free to adjust this parameter
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

for t in range(epochs):

    for i, data in enumerate(train_data_loader):
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)
        x = inputs.view(x.size(0), -1)
        h1 = x.mm(w1)                                   ##  In
                                numpy, you would say    h1 = x
                                .dot(w1)
        h1_relu = h1.clamp(min=0)
        h2 = h1_relu.mm(w2)
        h2_relu = h2.clamp(min=0)
        y_pred = h2_relu.mm(w3)
        # Compute and print loss
        loss = (y_pred - y).pow(2).sum().item()
        y_error = y_pred - y

        #TODO : Accumulate loss for printing per epoch
        grad_w3 = h2_relu.t().mm(2 * y_error)   #<<<<<<
                                Gradient of Loss w.r.t w3
```

```
        h2_error  =  2.0 * y_error.mm(w3.t())   #
                                backpropagated error to the h2
                                 hidden layer
        h2_error[h < 0] = 0                      # We set
                                those elements of the
                                backpropagated error
        grad_w2 = h1_relu.t().mm(2 * h2_error)  #<<<<<<
                                Gradient of Loss w.r.t w2
        h1_error  =  2.0 * h2_error.mm(w2.t())  #
                                backpropagated error to the h1
                                 hidden layer
        h1_error[h < 0] = 0                      # We set
                                those elements of the
                                backpropagated error
        grad_w1 = x.t().mm(2 * h1_error)        #<<<<<<
                                Gradient of Loss w.r.t w2
        # Update weights using gradient descent
        w1 -= learning_rate * grad_w1
        w2 -= learning_rate * grad_w2
        w3 -= learning_rate * grad_w3

    #print loss per epoch
    print('Epoch  %d:\t %0.4f'%(t, epoch_loss))

#Store layer weights in pickle file format
torch.save({'w1':w1,'w2':w2,'w3':w3},'./wts.pkl')

```

4.3 Sub Task3: Testing on the Validation Set

Adapt the incomplete code template from the previous section to load the saved weights and evaluate on the validation set. Print the validation loss and the classification accuracy.

5 Output Format

Store your training and validation results in `output.txt` file, in the following format.

```
Epoch 0: epoch0_loss
Epoch 1: epoch1_loss
Epoch 2: epoch2_loss
.
.
.
```

```
Epoch n: epochn_loss
<blank line>
Val Loss: val_loss
Val Accuracy: val_accuracy_value%
```

6 Submission Instructions

- Make sure to submit your code in Python 3.x and not Python 2.x.
- Create a .zip archive with the following files: `hw02_ImageNet_Scrapper.py`, `hw02_imagenet_task2.py`, `wts.pkl`, `output.txt`. Name the .zip archive as `hw02_<Firstname><Lastname>.zip` (without any white spaces). Note that .rar file format is Windows specific so please do NOT submit your solutions in .rar format.
- Follow the required input argument list for the python source files. **Your code must be your own work.**
- You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission.