BME646 and ECE60146: Homework 2

Spring 2024
Due Date: 11:59pm, Jan 22, 2024
TA: Akshita Kamsali (akamsali@purdue.edu)

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. Late submissions will be accepted with penalty: **-10 points per-late-day, up to 5 days**.

1 Introduction

The goal of this homework is to introduce you to the pieces needed to implement an image dataloader for training or testing your deep neural networks. To that end, this homework will help you familiarize yourself with the image representations as provided by PIL, Numpy, and PyTorch libraries. It will also make you more familiar with the idea of data augmentation. Upon completing this homework, you will be able to construct your own image dataloader for training deep neural networks using the `torchvision` library. For more information regarding the image representations and the usage of the `torchvision` library, you can refer to Prof. Kak's tutorial [5]. Note that this homework contains a "theory" task (Sec. 2) followed by the programming tasks (Sec. 3).

2 Understanding Pixel Value Scaling and Normalization

As you know from the Week 2 lecture by Prof. Kak, image data consists fundamentally of integers in the range 0 to 255 what a neural network likes to see at its input are floating point numbers between -1.0 and 1.0. Pixel value scaling refers to mapping the integer values to the floating point $(0, 1.0)$ range and pixel value normalization refers to a further transformation of the pixel values so that they span the floating-point $(-1.0, 1.0)$ range.

Your goal here is to compare manual pixel-value scaling using the call in Line (12) on Slide 26 with the more "automated" pixel-value scaling as provided by `tvt.ToTensor` as shown in Lines (15) and (16) on Slide 28.

For this comparative study, create two different versions of a simulated batch of images as shown at the bottom of Slide 19, one in which the pixel values are limited to the range 0 through 32 and other in which the pixel

values span the full one-byte range. Your first batch would be a simulation of a color photo recorded under conditions of poor illumination. And your second batch would be more or less the same as in Slide 19.

For each batch, compare the values you get with the manual approach with the values you get with the based on `tvt.ToTensor` and report your results.

If you wish, in each case, you can follow pixel-value scaling with pixel-value normalization using the statements shown on Slide 34.

2.1 Try it yourself

Load the `.npy` file provided to you. You may investigate the image by checking the minimum and maximum values. Next, print the maximum value in the image. Finally, divide the given image by max value and 255. What do you observe?

3 Programming Tasks

3.1 Setting Up Your Conda Environment

Before writing any code, you will first need to set up an Anaconda [1] environment, in which PyTorch and install other necessary packages. You should familiarize yourself with the basics of using conda for package management. Nonetheless, what is outlined below will help you get started:

1. A very useful cheatsheet on the conda commands can be found here [2].

2. If you are used to using pip, execute the following to download Anaconda:

   ```
   sudo pip install conda
   ```

   For alternatives to pip, follow the instructions here [3] for installation.

3. Create your ECE60146 conda environment:

   ```
   conda create --name ece60146 python=3.10
   ```

4. Activate your new conda environment:

   ```
   conda activate ece60146
   ```

5. Install the necessary packages (e.g. PyTorch, torchvision) for your solutions

   ```
   conda install pytorch==1.10.0 torchvision==0.11.0 cudatoolkit=10.2
   -c pytorch
   ```

   Note that the command above is specifically for a GPU-enabled installation of PyTorch version 1.10 and is only an example. Depending on your own hardware specifications and the drivers installed, the command will vary. You can find more about such commands for installing PyTorch here [4]. Most issues regarding installation can be resolved through stakoverflow solutions.

   While GPU capabilities are not required for this homework, you will need thm for later homeworks.

6. After you have created the conda environment and installed the all the dependencies, use the following command to export a snapshot of the package dependencies in your current conda environment:

   ```
   conda env export > environment.yml
   ```

7. Submit your `environment.yml` file to demonstrate that your conda enviroment has been properly set up.

3.2 Becoming Familiar with `torchvision.transforms`

This task is about the Data Augmentation material on Slide 37 through 47 of the Week 2 slides on `torchvision`. Review those slides carefully and execute the following steps:

1. Take a photo of a Laptop on a Desk with your cellphone camera while you are standing directly in front of the sign and the camera is pointing straight at it. Alternatively, you can also take a picture of any flat surfaced object such as book, tablet, portrait, etc. placed vertically on a desk or wall.

2. Take another photo of the same object, but this time from a very oblique angle – you may either move just the camera or your entire self to create this effect.

   Example: I have chosen a white board leaning against a wall. The images are in figures 1a and 1b correspond to front and oblique views.

(a) Front image (b) Oblique image

Figure 1: Example of images of a flat surfaced object to take.

3. Now experiment with applying the callable instance `tvt.RandomAffine` and the function `tvt.functional.perspective()` that are mentioned on Slides 46 and 47 of Week 2 to see if you can transform one image into the other.

4. Note that for measuring the similarity between two images of the object, you can measure the distance between the two corresponding histograms, as explained on Slides 65 through 73.

5. One possible way of solving this problem is to keep trying different affine (or projective) parameters in a loop until you find the parameters that will make one image look reasonably similar to the other.

6. In your report, first plot your front and oblique images side-by-side. Subsequently, display your best transformed image, that is the most similar to the target image, using either the affine or projective parameters. Also, plot the final histograms of the images and report the Wasserstein distance .

   Explain in one or two paragraphs on how you have solved this task.

3.3 Creating Your Own Dataset Class

Now that you have become familiar with implementing transforms using `torchvision`, the next step is to learn how to create a custom dataset class that is based on the `torch.utils.data.Dataset` class for your own images. Your custom dataset class will store the meta information about your dataset and implement the method that loads and augments your images.

The code snippet below provides a minimal example of a custom dataset within the PyTorch framework:

```
1 import torch
```

```python

class MyDataset(torch.utils.data.Dataset):

    def __init__(self, root):
        super().__init__()
        # Obtain meta information (e.g. list of file names)
        # Initialize data augmentation transforms, etc.
        pass

    def __len__(self):
        # Return the total number of images
        # the number is a place holder only
        return 100

    def __getitem__(self, index):
        # Read an image at index and perform augmentations
        # Return the tuple: (augmented tensor, integer label)
        # these dimension numbers are for illustrative purpose
                                    only.
        return torch.rand((3, 256, 256)), random.randint(0, 10)
```

Before proceeding, take ten images with your cellphone camera of any object, you may wish to continue with the same object. Now, store them together within a single folder. Now, based on the code snippet above, implement a custom dataset class that handles your own images. More specifically, your `__getitem__` method should:

1. Read from disk the image corresponding to the input index as a PIL image.

2. Subsequently, assuming that you are using your custom dataset to train a classifier, augment your image with any three different transforms of your choice that you think will make your classifier more robust. Note that a suitable transform could be either color-related or geometry-related. Note that you should use `tvt.Compose` to chain your augmentations into a single callable instance.

3. Finally, return a tuple, with the first item being the tensor representation of your augmented image and the second the class label. For now, you can just use a random integer as your class label.

The code below demonstrates the expected usage of your custom dataset class:

```python
# Based on the previous minimal example
my_dataset = MyDataset('./path/to/your/folder')
```

```
print(len(my_dataset))  # 100
index =
print(my_dataset[index][0].shape, my_dataset[index][1])
# torch.Size([3, 256, 256]) 6

index = 50
print(my_dataset[index][0].shape, my_dataset[index][1])
# torch.Size([3, 256, 256]) 8
```

In your report, for at least three of your own images, plot the original version side-by-side with its augmented version. Also briefly explain the rationale behind your chosen augmentation transforms.

3.4 Generating Data in Parallel

For reasons that will become clear later in this class, training a deep neural network in practice requires the training samples to be fed in batches. Since calling `__getitem__` will return you a single training sample, you now need to build a dataloader class that will yield you a *batch* of training samples per iteration. More importantly, by using a dataloader, the loading and augmentation of your training samples is done efficiently in a multi-threaded fashion.

For the programming part, wrap an instance of your custom dataset class within the `torch.utils.data.DataLoader` class so that your images for training can be processed in parallel and are returned in batches.

In your report, set your batch size to 4 and plot all 4 images together from the same batch as returned by your dataloader.

Additionally, compare and discuss the performance gain by using the multi-threaded `DataLoader` v.s. just using `Dataset`. First, record the time needed to load and augment 1000 random images in your dataset (with replacement) by calling `my_dataset.__getitem__` 1000 times. Then, record the time needed by `my_dataloader` to process 1000 random images. Note that for this comparison to work, you should set both your `batch_size` and `num_workers` to values greater than 1. You must report the times for atleast 2 different batchsizes and two different number of workers. In your report, tabulate your findings on the timings and experiment with different settings of the `batch_size` and `num_workers` parameters.

3.5 Random seed

Reproducibility is crucial in deep learning to ensure consistent results. In this section, we will explore the impact of setting random seeds on the

behavior of data loaders.

First, without setting the seed, set the batch size to 2 and plot all 2 images together from the same batch as returned by your data loader with shuffle set to true. Plot only one batch with two images and exit the batch iterator. Now, rerun the iterator. Do you see the same two images in the first iteration? Why or why not?

```
batch_size = 2
dataloader = DataLoader(my_dataset, batch_size=batch_size,
                                    shuffle=True)

# Plot the first batch of images
for batch in dataloader:
    images, labels = batch
    # Plot images (only two images for brevity)
    break

# Rerun the iterator
for batch in dataloader:
    images, labels = batch
    # Check if the same two images are in the first iteration
    break
```

Next, at the top just below your import library statements, set your random seed to '60146' (See Week 2 lecture's slides 72 and 73). Follow the previous exercise of printing the images in the first batch only in two different iterations. What do you see now?

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (ONLY .py files are accepted) (c) `.yaml` file of your conda environment Rename your .zip file as hw2_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too.

2. For all homeworks, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert it to `.py` and submit that as source code.

3. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous

submission. **If you are submitting late, do it only once on BrightSpace.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.

4. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

5. Your pdf must include a description of

   - Your explanation to the theory question as described in Sec. 2.
   - Your observations in Sec. 2.1
   - The various plots and descriptions as instructed by the subsections in Sec. 3 and 3.5.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up.
   - To help better provide feedback, make sure to **number your figures, tables** and refer them accordingly in your reports.

References

[1] Anaconda, . URL https://www.anaconda.com/.

[2] Conda Cheat Sheet, . URL https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf.

[3] Conda Installation, . URL https://conda.io/projects/conda/en/latest/user-guide/install/index.html.

[4] Installing Previous Versions of PyTorch. URL https://pytorch.org/get-started/previous-versions/.

[5] Torchvision and Random Tensors. URL https://engineering.purdue.edu/DeepLearn/pdf-kak/Torchvision.pdf.