BME646 and ECE60146: Homework 2

Spring 2025
Due Date: Monday, Jan 27, 2025, 11:59pm
TA: Akshita Kamsali (akamsali@purdue.edu)

Turn in typed solutions via Gradescope. Post questions to Piazza. Additional instructions can be found at the end. Late submissions will be accepted with penalty: -10 points per-late-day, up to 5 days.

1 Introduction

Through this homework, you will gain familiarity with image representations provided by the PIL, Numpy, and PyTorch libraries. Additionally, you will explore the concept of data augmentation.

This homework will also introduce you to the essential components for building an image dataloader to train or test deep neural networks.

For more information regarding the image representations and the usage of the `torchvision` library, you can refer to Prof. Kak's tutorial [5].

2 Understanding Pixel Value Scaling, Normalization and Data Augmentation

Image data, as explained in the Week 2 lecture by Prof. Kak, consists of integers in the range of 0 to 255. However, neural networks prefer input data represented as floating-point numbers in the range of -1.0 to 1.0. To meet this requirement, the processes of pixel value scaling and normalization are employed.

2.1 Pixel Value Scaling

Pixel value scaling involves mapping the integer values of image data (0 to 255) to the floating-point range $(0, 1.0)$. This transformation ensures that the data is suitable for many machine learning algorithms that perform better with scaled input values. Refer pages 26-32 in Week 2 lecture slides for more on scaling.

2.2 Pixel Value Normalization

Pixel value normalization is an additional step where the scaled pixel values are transformed to span the floating-point range $(-1.0, 1.0)$. This normalization step is particularly important for neural networks as it centers the data around zero, which can improve the convergence behavior during training. Refer pages 33-36 in Week 2 lecture slides for more on Normalization.

2.3 Data Augmentation and Transformations

Data augmentation techniques, such as random rotations, flips, cropping, and color adjustments, introduce variability to the training dataset, reducing overfitting and improving generalization. In PyTorch, these augmentations can be performed using the `torchvision.transforms` module.

Transformations typically include:

- Scaling and Normalization
- Random Augmentations (e.g., rotation, horizontal flip)
- Conversion to Tensor

For example, a typical transformation pipeline can be constructed as:

```
from torchvision import transforms

transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
```

Refer pages 38-47 in Week 2 lecture slides for more on Normalization.

2.4 PyTorch Dataset and DataLoader

PyTorch provides the `Dataset` class for representing datasets. The `DataLoader` class works alongside `Dataset` to handle batching, shuffling, and parallel loading of data.

To create a dataset and load it into a dataloader:

```
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST

# Load dataset
```

```
dataset = MNIST(root='./data', train=True, transform=transform,
                                    download=True)

# Create dataloader
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

This setup simplifies data handling, ensuring seamless integration of data augmentation, transformations, and preprocessing into the training pipeline.

3 Programming Tasks (100 points)

3.1 Conda Environment

Before writing any code, it is essential to set up an Anaconda [1] environment, where PyTorch and other necessary packages can be installed. In case you are not familiar with environments, the steps outlined below will help you get started with `conda`:

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 torchvision -c pytorch`

   Note that the command above is specifically for a GPU-enabled installation of PyTorch stable version 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 them 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 as part of your zip to demonstrate that your conda enviroment has been properly set up.

3.2 Comparing CIFAR10 with a Custom Dataset

In this task, we will compare the widely used CIFAR10 dataset in PyTorch with a custom dataset created using 20 images taken by yourself. By applying data augmentation, we will extrapolate the custom dataset to include 30 new images, creating a total of 50 samples.

For the comparison:

- We will select 5 out of the 10 classes from CIFAR10 and load 10 images for each class (50 images in total).
- The custom dataset will contain 20 images captured by yourself, augmented to 50 using transformations.
- We will compare the datasets in terms of size, diversity, and distribution of classes.
- Skeleton code is provided to guide you through the implementation.

3.2.1 Steps to Implement

1. **Prepare CIFAR10 Dataset:** Load CIFAR10 using `torchvision.datasets.CIFAR10` and filter 5 classes with 10 images each.
2. **Create Custom Dataset:** Capture 20 images using your cellphone of a single object like a laptop, whiteboard, book or you can take these images of internet which have no copyright issues, store them in a folder, and augment them to create 30 new samples using transformations.

3. **Apply Data Augmentation:** Use `torchvision.transforms` to augment the custom dataset and generate additional samples.

4. **Compare Datasets:** Visualize the images.

3.2.2 Skeleton Code

Below is a Python code snippet to perform the tasks. You may choose to write your own code from scratch.

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

# Step 1: Load CIFAR10 dataset and filter 5 classes
transform_cifar = transforms.Compose([transforms.ToTensor()])
cifar10 = datasets.CIFAR10(root='./data', train=True, download=
                                True, transform=transform_cifar
                                )

# Filter 5 classes with 10 images each
selected_classes = [0, 2, 3, 5, 7]  # You may pick any 5
                                classes
cifar_indices = [i for i, (_, label) in enumerate(cifar10) if
                                label in selected_classes]
subset_cifar10 = Subset(cifar10, cifar_indices[:50])  # Take 10
                                 images per class

# Step 2: Create a custom dataset
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, root, transform=None):
        self.root = root
        self.image_paths = [os.path.join(root, img) for img in
                                os.listdir(root)]
        self.transform = transform

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

    def __getitem__(self, index):
        img_path = self.image_paths[index]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, 0  # Assuming no class labels for
                                simplicity
```

```
32
33 # Load 20 custom images and apply augmentation
34 transform_custom = transforms.Compose([
35     # fill code here
36 ])
37
38 custom_dataset = CustomDataset(root='./custom_images',
                                   transform=transform_custom)
39
40 # Data augmentation to extrapolate to 50 samples
41 augmented_images = [custom_dataset[i % len(custom_dataset)][0]
                                   for i in range(50)]
42
43 # Step 3: Compare the datasets
44 print(f"Size of CIFAR10 subset:  __fill code__ images")
45 print(f"Size of Custom Dataset:  __fill code__ images")
46
47 # Step 4: Visualize the datasets
48 import matplotlib.pyplot as plt
49
50 # Display a few samples from CIFAR10
51 cifar_loader = DataLoader(subset_cifar10, batch_size=5, shuffle
                                   =False)
52 batch = next(iter(cifar_loader))
53 cifar_images = batch[0]
54 # fill plot code
55
56
57 # fill plot code
58 plt.suptitle("CIFAR10 Subset Samples")
59 plt.show()
60
61 # Display a few augmented samples from the custom dataset
62 # fill plot code
63 # watch for dimensions
64 # fill plot code
```

Include 5 images from each dataset in the report.

3.3 Using DataLoader for Parallel Processing

Efficiently handle batches of images using the `torch.utils.data.DataLoader` class. Perform the following tasks:

1. Wrap your custom dataset class within a DataLoader instance, similarly to how the CIFAR-10 dataset is loaded. Set the batch size to 4 and enable multi-threading by setting num_workers to a value greater than 1.

2. Set the batch size to 4 and enable multi-threading by setting `num_workers` greater than 1.

3. Plot all 4 images from a single batch as returned by the `DataLoader`.

4. Compare performance for parallel data loading:

   - Configure step 3 in Section 3.2.1 it generate 1000 custom images.
   - Measure the time taken to load and augment all 1000 images using `__getitem__` in a loop.
   - Measure the time taken by the `DataLoader` to process all 1000 images with varying `batch_size` and `num_workers`.
   - Report your findings in a table, experimenting with at least two different `batch_size` and `num_workers` values, total of four cominations.

5. For at least one batch, compute the maximum of each RGB channel value of the images before and after normalization.

Important Notes On Multi-threading

When running your code, you might encounter errors related to multi-threaded data loading or other OS-specific configurations. These issues can depend on your system setup and Python environment.

Should you encounter such errors, here are some suggestions to troubleshoot:

1. Review the error message carefully to identify the specific issue.

2. Ensure that your Python, PyTorch, and related libraries are up-to-date and compatible with your system.

3. Try setting `num_workers=0` in the `DataLoader` to bypass multi-threading if multi-processing causes issues.

4. Use resources like Stack Overflow or other developer forums for potential solutions tailored to your specific error.

Debugging is an essential skill for resolving compatibility issues, and exploring reliable resources online will often provide the fastest resolution. Understanding and addressing such challenges is a critical step in effective problem-solving for machine learning workflows.

3.4 Exploring Random Seed and Reproducibility

Reproducibility is essential for deep learning experiments. This task explores the impact of setting a random seed. Refer pages 72-73 in Week 2 lecture slides for more on Normalization.

1. Without setting a random seed:
   - Set the batch size to 2 and shuffle the dataset.
   - Plot the first batch of images. Exit the batch iterator and rerun it. Note if the same two images appear in the first batch.
2. With a random seed:
   - Set a random seed to `60146` at the beginning of your script.
   - Repeat the previous exercise and compare the results. Note if the same images appear in the first batch across iterations.

Explain in your report how setting the random seed impacts reproducibility and why this is important in deep learning experiments.

4 Submission Instructions

Include a typed report explaining how you solved the given programming tasks. You may refer to the homework solutions posted at the class website for the previous years for examples of how to structure your report

1. **Turn in a PDF file and mark all pages on gradescope**. Rename .pdf file as hw1_<First Name><Last Name>.pdf
2. Submit your code files(s) as zip file. Rename the .zip file as hw1_<First Name><Last Name>.zip and follow the same file naming convention for your pdf report too. **Not adhering to the above naming convention will lead to you receiving an automatic zero for the homework.**
3. For this homework, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert code to `.py` and submit that as source code. **Do NOT submit .ipynb notebooks.**

4. 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.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.

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

6. Your pdf must include a description of

   - Outputs from your implementation for the parameter values in the snippet.
   - Outputs for each of the provided snippets above whith input parameters of your choice.
   - Your source code. Make sure that your source code files are adequately commented and cleaned up. You may refer to the homework solutions posted at the class website for the previous years for examples for reference.

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 Different Versions of PyTorch. URL https://pytorch.org/get-started/locally/.

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