ECE60146 Homework 2 Report

Lucas Nguyen <nguye800@purdue.edu>

****There seems to be a mismatch in instructions as the instructions in 3.3 seem to be similar to the Task Summary in 3.6 but with different numbers so I will just carry out the instructions laid out in 3.6 since the skeleton code is more oriented towards those instructions****

Introduction

The objective of this homework is to get familiar with datasets, dataloaders, and transformations. The assignments within this homework include creating datasets and using dataloaders to output the dataset in batches with various transformations.

Creating the Environment

The preliminary step for this homework is to create a conda environment which contains all packages required to complete this homework (except for matplotlib.pyplot which needs to be downloaded). The steps taken to create the environment is as follows:

1. Sudo pip install conda
    a. Installs conda so that a virtual environment can be created
2. conda create –name ece60146 python=3.11
    a. creates a new environment called ece60146 with python 3.11
3. conda activate ece60146
    a. Activates the newly created environment. Now any commands run in the terminal will be using this environment
4. Conda install pytorch torchvision -c pytorch
    a. Within the new environment pytorch and torchvision are installed which are dependencies for this homework
5. Conda env export > environment.yml
    a. Exports environment as a yml file for grading

1 Load ImageNet (Option A)

Implementing a dataset to load the ImageNet dataset using the zip file from Brightspace (Option A). The _load_samples function loops through each class label given in initialization and appends images_per_class images for each class label to self.samples. The assignment was to load 50 images total (5 from each class) so a ImageNetSubset instantiation was initialized with images_per_class=5 and batches were looped through 10 times in a for loop to give a total of 50 images. Code output proves that 5 images from each label class were correctly output.

_load_samples Implementation

```
def _load_samples(self):
    """Load image paths for each requested class."""
    for label in self.class_labels:
        base_path = os.path.join(self.root, str(label))
        count = 0
        for img in sorted(os.listdir(base_path)):
            if img.lower().endswith(('.png', '.jpg', '.jpeg')):
                self.samples.append((os.path.join(base_path, img), label))
                count += 1

            if count == self.images_per_class:
                break
```

__getitem__ Implementation

```
def __getitem__(self, index):
    """
    Returns:
        image (Tensor): Transformed image
        label (int): Class label
    """
    img_path = self.samples[index][0]
    image = Image.open(img_path).convert("RGB")
    if self.transform:
        image = self.transform(image)

    return image, self.samples[index][1]
```

Load 50 images (5 per class) Code Implementation

```
217     # Task 1: Load ImageNet (50 images: 10 classes x 5 images)
218     print("\n" + "=" * 60)
219     print("Task 1: Loading ImageNet")
220     print("=" * 60)
221
222     imagenet = ImageNetSubset(root='D:\Lucas College\Purdue\Y4\ECE60146-HW\HW2\imagenet_subset',
223                               class_labels=required_labels,
224                               images_per_class=5,
225                               transform=transform_basic)
226
227     dataloader = DataLoader(
228         imagenet,
229         batch_size=5,
230         shuffle=False,
231         num_workers=4,
232         pin_memory=True
233     )
234
235     it = iter(dataloader)
236     print("Total Images:", len(imagenet))
237     for i in range(10):
238         images, labels = next(it)
239         print("Class Label:", labels)
```

Code Output

```
============================================================
Task 1: Loading ImageNet
============================================================
Loaded 50 images from 10 classes
Total Images: 50
Class Label: tensor([1, 1, 1, 1, 1])
Class Label: tensor([151, 151, 151, 151, 151])
Class Label: tensor([281, 281, 281, 281, 281])
Class Label: tensor([291, 291, 291, 291, 291])
Class Label: tensor([325, 325, 325, 325, 325])
Class Label: tensor([386, 386, 386, 386, 386])
Class Label: tensor([430, 430, 430, 430, 430])
Class Label: tensor([466, 466, 466, 466, 466])
Class Label: tensor([496, 496, 496, 496, 496])
Class Label: tensor([950, 950, 950, 950, 950])
```

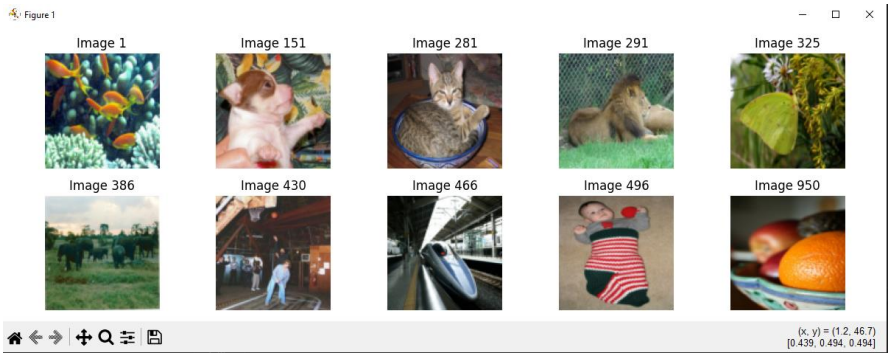
2 Visualize ImageNet

This assignment entailed loading 10 images into a 2x5 grid (1 per class) therefore ImageNetSubset was initialized with images_per_class=1 and the DataLoader was set to batch_size=10 to get all 10 images at once then plotted in a for loop from 0-9 inclusive.

Code Implementation

```
245  # Task 2: Visualize ImageNet (1 image per class = 10 images
246  print("\n" + "=" * 60)
247  print("Task 2: Visualizing ImageNet")
248  print("=" * 60)
249
250  imagenet = ImageNetSubset(root='D:\Lucas College\Purdue\Y4\ECE60146-HW\HW2\imagenet_subset',
251                            class_labels=required_labels,
252                            images_per_class=1,
253                            transform=transform_basic)
254
255  dataloader = DataLoader(
256      imagenet,
257      batch_size=10,
258      shuffle=False,
259      num_workers=4,
260      pin_memory=True
261  )
262
263  images, labels = next(iter(dataloader))
264  fig, axes = plt.subplots(2, 5, figsize=(12, 4))
265
266  for i in range(10):
267      img = images[i].permute(1, 2, 0)  # CHW → HWC
268      axes[i//5][i%5].imshow(img)
269      axes[i//5][i%5].axis("off")
270      axes[i//5][i%5].set_title(f"Image {labels[i]}")
271
272  plt.tight_layout()
273  plt.savefig("imagenet_samples.png", dpi=300, bbox_inches="tight")
274  plt.show()
```

Output (Image #: Class number listed in folder)



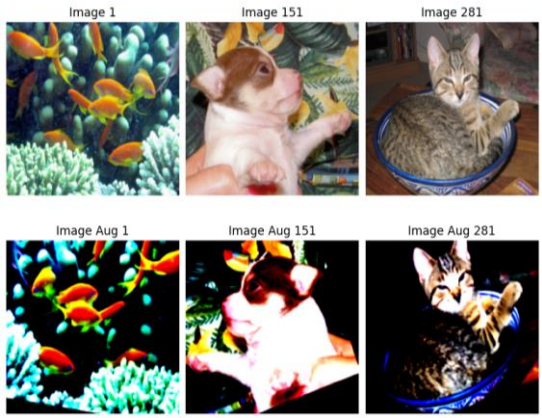
3 Augmentation Comparison

This section required plotting 3 images alongside their augmented versions. Two ImageNetSubsets were initialized, one with transform_basic to plot the original image, and one with transform_custom to show the augmented version. Then two dataloaders were initialized for the two datasets with shuffle=False so that they would both get the same image in the same order in order to properly plot them against each other.

Augmentation Code

```
# Task 3: Show augmentation effects
print("\n" + "=" * 60)
print("Task 3: Augmentation comparison")
print("=" * 60)

imagenet_orig = ImageNetSubset(root='D:\Lucas College\Purdue\Y4\ECE60146-HW\HW2\imagenet_subset',
                               class_labels=required_labels,
                               images_per_class=1,
                               transform=transform_basic)
imagenet_aug = ImageNetSubset(root='D:\Lucas College\Purdue\Y4\ECE60146-HW\HW2\imagenet_subset',
                              class_labels=required_labels,
                              images_per_class=1,
                              transform=transform_custom)

dataloader_orig = DataLoader(
    imagenet_orig,
    batch_size=3,
    shuffle=False,
    num_workers=4,
    pin_memory=True
)

dataloader_aug = DataLoader(
    imagenet_aug,
    batch_size=3,
    shuffle=False,
    num_workers=4,
    pin_memory=True
)

images_orig, labels_orig = next(iter(dataloader_orig))
images_aug, labels_aug = next(iter(dataloader_aug))
fig, axes = plt.subplots(2, 3, figsize=(12, 14))

for i in range(3):
    img_orig = images_orig[i].permute(1, 2, 0)  # CHW → HWC
    axes[0][i].imshow(img_orig)
    axes[0][i].axis("off")
    axes[0][i].set_title(f"Image {labels_orig[i]}")

    img_aug = images_aug[i].permute(1, 2, 0)  # CHW → HWC
    axes[1][i].imshow(img_aug)
    axes[1][i].axis("off")
    axes[1][i].set_title(f"Image Aug {labels_aug[i]}")

plt.tight_layout()
plt.savefig("imagenet_aug.png", dpi=300, bbox_inches="tight")
plt.show()
```

Output



4 Custom Dataset

This section required implementing a custom dataset class where images from a given root path would be used to form the dataset. self.image_paths was implemented to loop through the root directory and build an array of every image type file. __getitem__ would convert the image to RGB then apply the desired transform before returning the image and a default label 0.

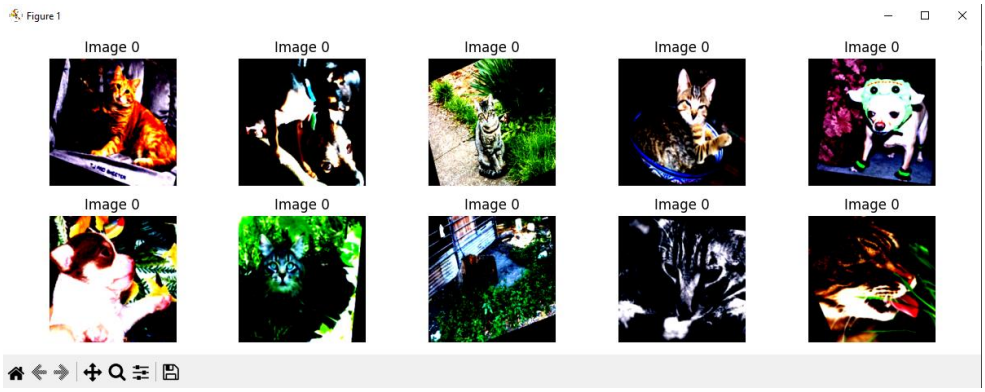
Class Implementation

```python
# ============================================================
# Custom Dataset for your own images
# ============================================================

class CustomDataset(Dataset):
    """Dataset for loading custom images from a folder."""

    def __init__(self, root, transform=None):
        self.root = root
        self.transform = transform

        self.image_paths = [os.path.join(root, img) for img in os.listdir(root) if img.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.gif'))]

        print(f"Found {len(self.image_paths)} images in {root}")

    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
```

Code Implementation

```python
    # Task 4: Custom dataset
    print("\n" + "=" * 60)
    print("Task 4: Custom dataset")
    print("=" * 60)

    fig, axes = plt.subplots(2, 5, figsize=(12, 4))

    customDataset_1 = CustomDataset(root='D:/Lucas College/Purdue/Y4/ECE60146-HW/HW2/imagenet_subset/151', transform=transform_custom)
    customDataset_2 = CustomDataset(root='D:/Lucas College/Purdue/Y4/ECE60146-HW/HW2/imagenet_subset/281', transform=transform_custom)

    dataset20 = ConcatDataset([customDataset_1, customDataset_2])

    dataloader_custom = DataLoader(
        dataset20,
        batch_size=10,
        shuffle=True,
        num_workers=4,
        pin_memory=True
    )
    num_generated = 0
    for epoch in range(5):
        images, labels = next(iter(dataloader_custom))
        for i in range(10):
            num_generated += 1
            if epoch == 4:
                img = images[i].permute(1, 2, 0)  # CHW → HWC
                axes[i//5][i%5].imshow(img)
                axes[i//5][i%5].axis("off")
                axes[i//5][i%5].set_title(f"Image {labels[i]}")

    print("Number of Augmented Images Generated:", num_generated)
    plt.tight_layout()
    plt.savefig("custom_samples.png", dpi=300, bbox_inches="tight")
    plt.show()
```

Code Output




```
============================================================
Task 4: Custom dataset
============================================================
Found 10 images in D:/Lucas College/Purdue/Y4/ECE60146-HW/HW2/imagenet_subset/151
Found 10 images in D:/Lucas College/Purdue/Y4/ECE60146-HW/HW2/imagenet_subset/281
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.343704].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.64].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.1134453].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.4831376].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.5877128].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.5702832].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.64].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.64].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..1.8382572].
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.117904..2.5877128].
Number of Augmented Images Generated: 50
```

5 DataLoader Performance

This step involved initializing a dataloader with different batch sizes and workers then timing how long it takes to load + transform + batch over one pass of the dataloader. From the numbers shown in the table the most likely reasoning is that the transform isn't computationally heavy enough to add significant time to augment images and thus the difference between num_workers=0 with different batch sizes is most likely either due to noise or to the smaller time required to internally batch the entire dataset in batches of 16 vs. 64. One major difference is that the two tests with num_workers=4 took significantly longer than the other tests. One reason for this is that the overhead to initialize several concurrent workers is much greater than the time the augment the images and therefore adds roughly a ~2.5 sec increase.

Code Implementation

```
356     # Task 5: DataLoader performance
357     print("\n" + "=" * 60)
358     print("Task 5: DataLoader performance")
359     print("=" * 60)
360
361     imagenet = ImageNetSubset(root='D:\Lucas College\Purdue\Y4\ECE60146-HW\HW2\imagenet_subset',
362                               class_labels=required_labels,
363                               images_per_class=10,
364                               transform=transform_custom)
365
366     for pair in [(16 ,0), (16 ,4), (64 ,0), (64 ,4)]:
367         t0 = time.perf_counter()
368         dataloader = DataLoader(
369             imagenet,
370             batch_size=pair[0],
371             shuffle=False,
372             num_workers=pair[1],
373             pin_memory=True
374         )
375         seen = 0
376         for images, labels in dataloader:
377             seen += images.size(0)
378         t1 = time.perf_counter()
379         print(f"For batch size: {pair[0]} and num workers: {pair[1]} the time was {t1-t0}")
```

Code Output

```
============================================================
Task 5: DataLoader performance
============================================================
Loaded 100 images from 10 classes
For batch size: 16 and num workers: 0 the time was 0.2791416998952627
For batch size: 16 and num workers: 4 the time was 2.930186300072819
For batch size: 64 and num workers: 0 the time was 0.26909289974719286
For batch size: 64 and num workers: 4 the time was 2.936049100011587
```

Performance Table

Batch_size	Num_workers	Time(sec)
16	0	0.40694050025194883
16	4	2.946450099814683
64	0	0.39657749980688095
64	4	2.9996744003146887

6 RGB Statistics

This section involved displaying the difference in values before and after the normalize transform. Before any transforms the min and max values for the pixels were 0 and 255 which makes sense because 8 bit rgb values can only range between 0-255. Secondly the values were taken after the transform_basic which just resizes and uses ToTensor. The value range makes sense because ToTensor transforms data to range from 0-1. Lastly the transform_custom was performed on the data which applies the normalize function to the data. The normalize function given had slightly different mean and std for each channel but overall normalized the RGB channels to roughly [-2.1, 2.6].

Code Implementation

```
381     # Task 6: RGB statistics
382     print("\n" + "=" * 60)
383     print("Task 6: RGB statistics")
384     print("=" * 60)
385
386     def channel_min_max(x):
387         return [(x[c].min().item(), x[c].max().item()) for c in range(3)]
388
389     img = Image.open('D:/Lucas College/Purdue/Y4/ECE60146-HW/HW2/imagenet_subset/1/00001.JPEG').convert("RGB")
390     img_pre = transform_basic(img)
391     img_post = transform_custom(img)
392
393     img_stats = channel_min_max(np.array(img))
394     pre_stats = channel_min_max(img_pre)
395     post_stats = channel_min_max(img_post)
396
397     print("Before all transformation")
398     print("R:", img_stats[0], "G:", img_stats[1], "B:", img_stats[2])
399
400     print("After ToTensor before Normalize")
401     print("R:", pre_stats[0], "G:", pre_stats[1], "B:", pre_stats[2])
402
403     print("After Normalize")
404     print("R:", post_stats[0], "G:", post_stats[1], "B:", post_stats[2])
```

Code Output

```
============================================================
Task 6: RGB statistics
============================================================
Before all transformation
R: (0, 255) G: (0, 255) B: (0, 255)
After ToTensor before Normalize
R: (0.0, 0.9960784316062927) G: (0.0, 0.9960784316062927) B: (0.003921568859368563, 0.9960784316062927)
After Normalize
R: (-2.1179039478302, 2.231783390045166) G: (-2.0357141494750977, 2.393557548522949) B: (-1.804444432258606, 2.6051416397094727)
```

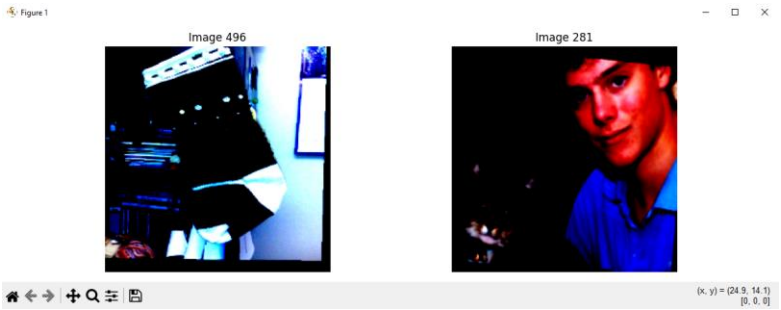
7 Reproducibility

This step was to demonstrate the effect of using random seeds on batch ordering. If no seed is specified, pytorch initializes its RNG from system entropy which explains the difference in images between the first 2 runs. The second 2 runs were initialized with the same seed meaning that even though shuffle=True, the "randomness" of the batch order is actually the same because the seed is the same, hence why the last 2 runs have the same images.

Code Implementation

```
406     # Task 7: Reproducibility
407     print("\n" + "=" * 60)
408     print("Task 7: Reproducibility")
409     print("=" * 60)
410
411     def set_seed(seed = 60146):
412         random.seed(seed)
413         np.random.seed(seed)
414         torch.manual_seed(seed)
415         if torch.cuda.is_available():
416             torch.cuda.manual_seed_all(seed)
417
418     set_seed()
419
420     imagenet = ImageNetSubset(root='D:\Lucas College\Purdue\Y4\ECE60146-HW\HW2\imagenet_subset',
421                               class_labels=required_labels,
422                               images_per_class=10,
423                               transform=transform_custom)
424     dataloader = DataLoader(
425             imagenet,
426             batch_size=2,
427             shuffle=True,
428             num_workers=0,
429             pin_memory=True
430         )
431
432     images, labels = next(iter(dataloader))
433     fig, axes = plt.subplots(1, 2, figsize=(12, 4))
434
435     for i in range(2):
436         img = images[i].permute(1, 2, 0)  # CHW → HWC
437         axes[i].imshow(img)
438         axes[i].axis("off")
439         axes[i].set_title(f"Image {labels[i]}")
440
441     plt.tight_layout()
442     plt.savefig("imagenet_samples.png", dpi=300, bbox_inches="tight")
443     plt.show()
```

No Seed First Run



No Seed Second Run

Seed 60146 First Run

Seed 60146 Second Run