ECE 60146 and BME 64600: Homework 2

Spring 2026
Due Date: Tuesday, Jan 27, 2026, 11:59 pm ET
TA: Somosmita Mitra (mitra26@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

The goal of this homework is to get you to start working with images in the deep learning context and, at the same time, to experiment with the image scaling and normalization concepts presented in your instructor's Week 2 lecture.

Being the first DL related homework, another goal of this homework is to get you started with setting up your own Anaconda environment. It is in this environment in your machine that you will install the Python packages such as PyTorch.

Through this homework, you will also acquire a deeper understanding of how to apply different kinds of transformation to images for the purpose of data augmentation that we talked in the Week 2 lecture.

For more information regarding the topics mentioned above, please consult Prof. Kak's Week 2 lecture [5].

1.1 About the Images You'll be Working With

You will be working with a baby version of the very large ImageNet dataset for this homework. That dataset is one of the most famous datasets whose size keeps on increasing with time. Back in 2012, it had around 1.2 million images and, more recently, it now contains 14 million images. Check out the Wikipedia page on the dataset.

2 Data Augmentation and Transformations

As you learned in the Week 2 lecture, data augmentation is a powerful technique to artificially expand the size and diversity of a training dataset by applying random transformations to images. This helps reduce overfitting and improves the model's ability to generalize to new, unseen data. In PyTorch, augmentations are performed using the `torchvision.transforms` module.

2.1 Common Data Augmentation Techniques

The following transformations are commonly used for image augmentation:

- **RandomHorizontalFlip**: Randomly flips the image horizontally with a given probability.
- **GaussianBlur**: Applies Gaussian blur with a specified kernel size and sigma range.
- **RandomAffine**: Applies random affine transformations including rotation, translation, and scaling.

Data augmentation is typically applied to the images after pixel-value scaling and pixel-value normalization. Prior to the augmentation transformations listed above, you will apply the following transformations for pixel value scaling and normalization:

- **ToTensor**: Converts PIL Image to PyTorch tensor and scales pixel values from [0, 255] to [0.0, 1.0].
- **Normalize**: Normalizes tensor values using channel-wise mean and standard deviation.

A typical transformation pipeline typically combines all of the transformations listed above through a callable instance of `torchvision.transforms.Compose` as shown below:

```
import torchvision.transforms as tvt
transform = tvt.Compose([
    tvt.RandomHorizontalFlip(p=0.2),
    tvt.RandomAffine(degrees=20, translate=(0.1, 0.1),scale=(0.
                                9, 1.1)),
    tvt.Resize((224, 224)),
    tvt.GaussianBlur(kernel_size=3, sigma=(0.3, 1.5)),
    tvt.ToTensor(),
    tvt.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224
                                , 0.225])])
```

Refer to pages 38-47 in Week 2 lecture slides for more on data augmentation.

2.2 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.

```
from torch.utils.data import DataLoader

# Create dataloader from dataset
dataloader = DataLoader(dataset, batch_size=32, shuffle=True,
                                num_workers=4)
```

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, set up an Anaconda [1] environment where PyTorch and other necessary packages can be installed:

1. A useful cheatsheet on 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=3.11`

4. Activate your new conda environment:

   `conda activate ece60146`

5. Install the necessary packages:

   `conda install pytorch torchvision -c pytorch`

   `pip install datasets huggingface_hub` # Only needed for Option B

   Note: The command above is an example. Depending on your hardware specifications and drivers, the command will vary. Find more at [4].

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

6. Export your environment:

   `conda env export > environment.yml`

7. Submit your `environment.yml` file as part of your zip.

3.2 Working with ImageNet

In this task, you will work with the ImageNet dataset and compare it with a custom dataset you create. You have two options for obtaining ImageNet images:

3.2.1 Option A: Use the Provided Subset

Download the pre-prepared ImageNet subset from Brightspace (Course Materials → Homework 2 → imagenet_subset.zip). This subset contains:

- 100 classes from ImageNet
- 20 images per class (2,000 total images)
- Folder names are integer labels (e.g., `1/`, `15/`, `151/`)
- A `class_mapping.txt` file with class names

3.2.2 Option B: Download from Hugging Face

You can download ImageNet images directly from Hugging Face. This requires:

1. Create a free account at https://huggingface.co
2. Visit https://huggingface.co/datasets/ILSVRC/imagenet-1k
3. Click "Access repository" and accept the terms of use
4. Login via terminal: `huggingface-cli login`
5. Enter your access token from https://huggingface.co/settings/tokens

Important: Both options use the **same integer labels** (0-999). The provided subset folder names match the labels returned by Hugging Face's `sample['label']`. Note: These integer labels correspond to ImageNet synset indices, not contiguous class IDs used by pretrained models. Treat them as categorical identifiers rather than model output indices.

3.2.3 Required Classes for Your Report

Regardless of which option you choose, **you must include the following 10 classes** in your report:

3.2.4 Required Transformations

You must apply the following transformations for the custom dataset:

Label	Class Name
1	goldfish
151	Chihuahua
281	tabby cat
291	lion
325	sulphur butterfly
386	African elephant
430	basketball
466	bullet train
496	Christmas stocking
950	orange

Table 1: Required 10 classes for your report. Include 5 images from each class (50 total).

```
import torchvision.transforms as tvt

transform_custom = tvt.Compose([
    tvt.RandomHorizontalFlip(p=0.2),
    tvt.RandomAffine(degrees=20, translate=(0.1, 0.1), scale=(0
                                .9, 1.1)),
    tvt.Resize((224, 224)),
    tvt.GaussianBlur(3, (0.3, 1.5)),
    tvt.ToTensor(),
    tvt.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224
                                , 0.225])
])

# Transform WITHOUT augmentation (for visualization/comparison)
transform_basic = tvt.Compose([
    tvt.Resize((224, 224)),
    tvt.ToTensor(),
])
```

3.2.5 Understanding Each Transform

1. **RandomHorizontalFlip(p=0.2)**: Randomly flips the image horizontally with 20% probability. This augmentation helps the model learn that objects can appear in different orientations. Useful for most objects but should be avoided for text or direction-sensitive data.

2. **RandomAffine(degrees=20, translate=(0.1, 0.1), scale=(0.9, 1.1))**: Applies random affine transformations:
   - `degrees=20`: Rotates image randomly between -20 and +20
   - `translate=(0.1, 0.1)`: Shifts image up to 10% horizontally and vertically
   - `scale=(0.9, 1.1)`: Scales image between 90% and 110% of original size

   This helps the model learn position and scale invariance.

3. **Resize((224, 224))**: Resizes all images to 224×224 pixels. This is the standard input size for many CNN architectures (e.g., ResNet, VGG). Ensures all images have consistent dimensions for batching.

4. **GaussianBlur(kernel_size=3, sigma=(0.3, 1.5))**: Applies Gaussian blur with a 3×3 kernel and random sigma between 0.3 and 1.5. This simulates out-of-focus images and helps the model become robust to slight blurriness in real-world images.

5. **ToTensor()**: Converts PIL Image (H×W×C, values 0-255) to PyTorch tensor (C×H×W, values 0.0-1.0). Also changes data type from uint8 to float32.

6. **Normalize(mean, std)**: Normalizes each channel using: $x_{norm} = \frac{x - mean}{std}$. This centers the data around zero and scales it to have unit variance.

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 ImageNet dataset is loaded. Set the batch size to 4 and enable multi-threading by setting num_workers to a value greater than 1.

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

3. Compare performance for parallel data loading:
    - Generate 1000 custom images (by increasing the augmentation factor). Note: You do not need to save these images or add them in your report. Generate them on the fly to measure time.
    - 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 combinations.
4. 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.
5. On Windows/Mac, add at the start of your script:

```
import multiprocessing
multiprocessing.set_start_method('spawn', force=True)
```

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 reproducibility.

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 using:

```python
import random
import numpy as np
import torch

def set_seed(seed=60146):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

set_seed(60146)
```

   - 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.

3.5 Skeleton Code

Below is skeleton code for this homework. You must complete the sections marked with `TODO`. Save this as `hw2_FirstNameLastName.py`.

```
"""
ECE 60146 - Homework 2
Name: [YOUR NAME]
Email: [YOUR EMAIL]

ImageNet Data Loading and Augmentation
"""

import os
import torch
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
from PIL import Image
import time

# REQUIRED CLASSES
# Labels match both the provided subset AND Hugging Face ImageNet
REQUIRED_CLASSES = {
    1: "goldfish",
    151: "Chihuahua",
    281: "tabby cat",
    291: "lion",
    325: "sulphur butterfly",
    386: "African elephant",
    430: "basketball",
    466: "bullet train",
    496: "Christmas stocking",
    950: "orange",
}

def get_class_name(label):
    """Get class name from integer label."""
    return REQUIRED_CLASSES.get(label, f"class_{label}")

# TRANSFORMS
transform_custom = transforms.Compose([
    transforms.RandomHorizontalFlip(p=0.2),
    transforms.RandomAffine(degrees=20, translate=(0.1, 0.1), scale=(0.9, 1.1)),
    transforms.Resize((224, 224)),
```

```
    transforms.GaussianBlur(3, (0.3, 1.5)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229
                                , 0.224, 0.225])
])

transform_basic = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# OPTION A: Dataset class for provided subset
class ImageNetSubset(Dataset):
    """
    Dataset for loading the provided ImageNet subset.

    Folder structure:
        imagenet_subset/
                  1/           (goldfish)
                        00001.JPEG
                        ...
                  15/          (robin)
                  ...

    Args:
        root (str): Path to imagenet_subset folder
        class_labels (list): List of integer labels to load (e.
                                g., [1, 15, 151])
        images_per_class (int): Number of images to load per
                                class
        transform (callable): Transform to apply to images
    """

    def __init__(self, root, class_labels, images_per_class=5,
                                transform=None):
        self.root = root
        self.class_labels = class_labels
        self.images_per_class = images_per_class
        self.transform = transform

        self.samples = []  # List of (image_path, label)
        self._load_samples()

        print(f"Loaded {len(self.samples)} images from {len(
                                class_labels)} classes")

    def _load_samples(self):
        """Load image paths for each requested class."""
        # TODO: Implement this method
```

```
        # For each label in self.class_labels:
        #   1. Build path to class folder: os.path.join(self.root, str(label))
        #   2. List all .JPEG/.jpg/.png files in that folder
        #   3. Take first self.images_per_class images
        #   4. Append (image_path, label) to self.samples

        pass  # Remove this line when implementing

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

    def __getitem__(self, index):
        """
        Returns:
            image (Tensor): Transformed image
            label (int): Class label (e.g., 1 for goldfish)
        """
        # TODO: Implement this method
        # 1. Get image_path, label from self.samples[index]
        # 2. Load image using PIL: Image.open(path).convert('RGB')
        # 3. Apply self.transform if not None
        # 4. Return (image, label)

        pass  # Remove this line when implementing


# OPTION B: Dataset class for Hugging Face ImageNet
class ImageNetHuggingFace(Dataset):
    """
    Dataset for loading ImageNet from Hugging Face.

    Requires: pip install datasets huggingface_hub
    And: huggingface-cli login

    Args:
        class_labels (list): List of integer labels to load (e.g., [1, 15, 151])
        images_per_class (int): Number of images per class
        transform (callable): Transform to apply
        split (str): 'train' or 'validation'
    """

    def __init__(self, class_labels, images_per_class=5, transform=None, split='train'):
        self.class_labels = set(class_labels)
        self.images_per_class = images_per_class
        self.transform = transform
```

```
131
132         self.samples = []  # List of (PIL.Image, label)
133         self._load_from_huggingface(split)
134
135         print(f"Loaded {len(self.samples)} images from {len(
                                class_labels)} classes")
136
137     def _load_from_huggingface(self, split):
138         """Load images from Hugging Face dataset."""
139         from datasets import load_dataset
140
141         print("Loading from Hugging Face (this may take a few
                                minutes)...")
142
143         # Load with streaming to avoid downloading entire
                                dataset
144         dataset = load_dataset(
145             "ILSVRC/imagenet-1k",
146             split=split,
147             streaming=True,
148             trust_remote_code=True
149         )
150
151         # TODO: Implement this method
152         # Track counts per class
153         # counts = {label: 0 for label in self.class_labels}
154         #
155         # for sample in dataset:
156         #     label = sample['label']  # Integer 0-999
157         #
158         #     if label in self.class_labels and counts[label] <
                                    self.images_per_class:
159         #         image = sample['image']
160         #         if image.mode != 'RGB':
161         #             image = image.convert('RGB')
162         #
163         #         self.samples.append((image, label))
164         #         counts[label] += 1
165         #
166         #         # Progress
167         #         total = sum(counts.values())
168         #         if total % 10 == 0:
169         #             print(f"  Collected {total} images...")
170         #
171         #     # Stop when done
172         #     if all(c >= self.images_per_class for c in counts
                                .values()):
173         #         break
174
```

```
        pass  # Remove when implementing

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

    def __getitem__(self, index):
        """Returns (image_tensor, label)"""
        # TODO: Implement
        pass


# 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

        # TODO: Load all image paths from root folder
        # Filter for: .jpg, .jpeg, .png, .bmp, .gif
        self.image_paths = []

        pass  # Remove and implement

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

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

    def __getitem__(self, index):
        # TODO: Implement
        pass


# Utility Functions
def denormalize(tensor, mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]):
    """Denormalize a tensor image for display."""
    mean = torch.tensor(mean).view(3, 1, 1)
    std = torch.tensor(std).view(3, 1, 1)
    return tensor * std + mean


def show_images(images, titles, rows, cols, figsize=(15, 10), save_path=None):
    """Display a grid of images."""
    fig, axes = plt.subplots(rows, cols, figsize=figsize)
```

```
    axes = axes.flatten() if rows * cols > 1 else [axes]

    for i, (img, title) in enumerate(zip(images, titles)):
        if i >= len(axes):
            break

        if isinstance(img, torch.Tensor):
            if img.min() < 0 or img.max() > 1:
                img = denormalize(img)
            img = img.permute(1, 2, 0).numpy()

        img = np.clip(img, 0, 1)
        axes[i].imshow(img)
        axes[i].set_title(title, fontsize=10)
        axes[i].axis('off')

    plt.tight_layout()
    if save_path:
        plt.savefig(save_path, dpi=150, bbox_inches='tight')
    plt.show()


def set_seed(seed=60146):
    """Set random seed for reproducibility."""
    import random
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)


# Main
if __name__ == "__main__":

    # Required class labels
    required_labels = list(REQUIRED_CLASSES.keys())
    print("Required classes:", required_labels)
    for label in required_labels:
        print(f"  {label}: {get_class_name(label)}")

    # Task 1: Load ImageNet (50 images: 10 classes x 5 images)
    print("\n" + "=" * 60)
    print("Task 1: Loading ImageNet")
    print("=" * 60)

    # TODO: Create dataset using Option A or B
    # imagenet_dataset = ImageNetSubset(
    #     root='./imagenet_subset',
```

```
    #     class_labels=required_labels,
    #     images_per_class=5,
    #     transform=transform_custom
    # )

    # Task 2: Visualize ImageNet (1 image per class = 10 images)
    print("\n" + "=" * 60)
    print("Task 2: Visualizing ImageNet")
    print("=" * 60)

    # TODO: Display 10 images (1 per class) in 2x5 grid
    # Save as 'imagenet_samples.png'

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

    # TODO: For 3 images, show original + 3 augmented versions

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

    # TODO: Create CustomDataset, augment 20 images to 50
    # Display 10 samples, save as 'custom_samples.png'

    # Task 5: DataLoader performance
    print("\n" + "=" * 60)
    print("Task 5: DataLoader performance")
    print("=" * 60)

    # TODO: Compare loading times with different batch_size/num_workers
    # Test: (16,0), (16,4), (64,0), (64,4)

    # Task 6: RGB statistics
    print("\n" + "=" * 60)
    print("Task 6: RGB statistics")
    print("=" * 60)

    # TODO: Compute min/max RGB before and after normalization

    # Task 7: Reproducibility
    print("\n" + "=" * 60)
    print("Task 7: Reproducibility")
    print("=" * 60)
```

```
317
318     # TODO: Test with and without set_seed(60146)
319
320     print("\n" + "=" * 60)
321     print("Done!")
322     print("=" * 60)
```

3.6 Task Summary

Complete the following tasks using the skeleton code:

1. **Load ImageNet** (10 pts): Implement the dataset class. Load 50 images (5 per class) from the 10 required classes.
2. **Visualize ImageNet** (10 pts): Display 10 images (one from each class) in a 2×5 grid.
3. **Augmentation Comparison** (15 pts): For 3 images, show original alongside 3 augmented versions.
4. **Custom Dataset** (15 pts): Collect 20 images, augment to 50, display 10 samples.
5. **DataLoader Performance** (20 pts): Compare loading times:

batch_size	num_workers	Time (sec)
16	0	_____
16	4	_____
64	0	_____
64	4	_____

6. **RGB Statistics** (15 pts): Compute min/max per channel before/after normalization.
7. **Reproducibility** (15 pts): Demonstrate effect of random seed on batch ordering.

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 hw2_<First Name><Last Name>.pdf
2. Submit your code file(s) as zip file. Rename the .zip file as hw2_<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 with 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.

5 Frequently Asked Questions (FAQ)

Q: I am getting "conda: command not found" when trying to create the environment.

A: You need to install Anaconda or Miniconda first. Download from https://docs.conda.io/en/latest/miniconda.html. For Linux/Mac: Run `bash Miniconda3-latest-*.sh`. For Windows: Download and run the installer. After installation, restart your terminal.

Q: Should I use Python 3.10 or 3.11?

A: Use Python 3.11 as specified in the homework. However, 3.10 should also work fine. Just be consistent and specify it in your environment.yml.

Q: The PyTorch installation command doesn't work on my machine.

A: Visit https://pytorch.org/get-started/locally/ and use the selector tool to get the correct command for your system. Select your OS, Package (Conda), and CUDA version (or CPU if no GPU). Example for CPU-only: `conda install pytorch torchvision torchaudio cpuonly -c pytorch`

Q: Do I need a GPU for this homework?

A: No, GPU is not required for HW2. All tasks can run on CPU. However, you will need GPU for later homeworks. You can use Google Colab.

Q: Where do I download the ImageNet subset?

A: Download `imagenet_subset.zip` from Brightspace (Course Materials → Homework 2 → imagenet_subset.zip). Extract it to your working directory.

Q: What's the difference between Option A (provided subset) and Option B (Hugging Face)?

A: Option A is pre-downloaded and ready to use; Option B downloads from Hugging Face (requires account and access approval). Choose Option A for simplicity.

Q: How do I get access to Hugging Face ImageNet (Option B)?

A: Create account at huggingface.co, visit the ImageNet dataset page, click "Access repository", accept terms, then run `huggingface-cli login` in terminal.

Q: What is the structure of the ImageNet subset folder?

A:

```
imagenet_subset/
 class_mapping.txt       # Label -> name mapping
 1/                      # goldfish
    00001.JPEG
    00002.JPEG
    ... (20 images)
 151/                    # Chihuahua
    ...
 ... (100 class folders total)
```

Q: How do I know which label corresponds to which class?

A: Check `class_mapping.txt` in the dataset folder, or use the `REQUIRED_CLASSES` dictionary in the skeleton code.

Q: Where do I get the 20 custom images?

A: You have several options: (1) Take photos yourself using your phone to photograph objects, animals, food, vehicles, etc. (2) Use copyright-free sources like Unsplash.com, Pexels.com, or Pixabay.com. (3) Use images from online sources for educational purposes only.

Q: What exactly counts as a valid custom image?

A: Any recognizable object: animals, vehicles, food items, household objects, electronics, sports equipment, plants, buildings, etc. Try to include diverse categories similar to ImageNet classes.

Q: Do my 20 custom images need to be from different categories?

A: The homework does not strictly require it, but diversity is recommended. Aim for at least 3-4 different categories (e.g., 5 animals, 5 vehicles, 5 food items, 5 household objects).

Q: What size should my custom images be?

A: Any size is fine. The transforms include `Resize((224, 224))`, so they will all be resized to 224×224 automatically.

Q: How does the 50 image augmentation work?

A: You have 20 original images and want 50 total. The code cycles through your 20 images 2.5 times, applying random transformations each time, creating different augmented versions.

Q: My code crashes with "BrokenPipeError" or multiprocessing errors.

A: This is a common Windows/Mac issue with `num_workers > 0`. Solutions: (1) Put all your code inside `if __name__ == "__main__":` (Recommended). (2) Set `num_workers=0`. (3) On Mac/Windows: `import multiprocessing; multiprocessing.set_start_method('spawn', force=True)`

Q: How do I plot images from a batch?

A: Images come as tensors in CHW format. Convert to HWC for matplotlib:

```
images, labels = next(iter(dataloader))
fig, axes = plt.subplots(1, 4, figsize=(12, 3))
for i in range(4):
    img = images[i].permute(1, 2, 0).numpy()  # CHW -> HWC
    img = np.clip(img, 0, 1)
    axes[i].imshow(img)
    axes[i].axis('off')
plt.show()
```

Q: How do I display normalized images correctly?

A: Use the `denormalize()` function provided in the skeleton code:

```
def denormalize(tensor, mean=[0.485, 0.456, 0.406], std=[0.229,
                                0.224, 0.225]):
    mean = torch.tensor(mean).view(3, 1, 1)
    std = torch.tensor(std).view(3, 1, 1)
    return tensor * std + mean
```

Q: How do I know if I set the random seed correctly?

A: Test it by running your script twice. You should get identical random numbers and identical batches.

Q: My images look weird/distorted after augmentation.

A: Check: (1) Dimension order (PyTorch uses CHW, matplotlib uses HWC), (2) Value range (should be [0, 1] for display), (3) If normalized, denormalize first: `img = denormalize(img)`

Q: "RuntimeError: DataLoader worker exited unexpectedly"

A: Common causes: Missing `if __name__ == "__main__"` guard, corrupted image, insufficient memory, incompatible num_workers. Solution: Add the guard or set `num_workers=0`.

Q: What should my directory structure look like?

A:

```
hw2_FirstNameLastName/
 hw2_FirstNameLastName.py
 environment.yml
 imagenet_subset/          # From Brightspace download
    class_mapping.txt
    1/
    151/
    ...
 custom_images/
     image_001.jpg
     image_002.jpg
     ... (20 total)
```

Q: Can I use/submit a Jupyter notebook (.ipynb)?

A: For development: Yes. For submission: No. Convert to .py using `jupyter nbconvert --to script notebook.ipynb` and submit the .py file. Include outputs in your PDF report.

Q: How should I structure my PDF report?

A: Include: (1) Introduction, (2) Environment Setup, (3) ImageNet Dataset

(10 samples, 1 per class), (4) Custom Dataset (10 samples), (5) Augmentation Comparison, (6) DataLoader & Batching, (7) Performance Analysis (table), (8) RGB Channel Analysis, (9) Reproducibility Experiment, (10) Code Appendix.

Q: How many pages should the report be?

A: Typically 8-15 pages including visualizations, tables, analysis, and code appendix. Quality > quantity.

Q: What happens if I do not follow the naming convention?

A: You will receive an automatic zero. Use exactly: `hw2_FirstNameLastName.pdf` and `hw2_FirstNameLastName.zip`

Q: Can I submit late?

A: Yes, with penalty: -10 points per day, maximum 5 days late. Only submit ONCE if late.

Q: I found code online that does something similar. Can I use it?

A: You can read documentation and tutorials, use standard PyTorch patterns, reference Prof. Kak's tutorials, and discuss concepts with classmates. You cannot copy code from other students, copy complete solutions from online, or submit someone else's work. When in doubt, cite your sources.

Q: How do I mark pages on Gradescope?

A: After uploading your PDF: (1) Gradescope shows question outline, (2) Click each question, (3) Select corresponding page(s), (4) Click "Save". Mark ALL pages for ALL questions.

For additional questions, post on Piazza or attend TA office hours.

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/kak/pdf-kak/Torchvision.pdf.