

ECE 60146 and BME 64600: Homework 2

Triparna Mahata

January 2026

1 Introduction

Goal of this homework:

- To set up a dedicated local development environment using Anaconda and install essential deep learning libraries, specifically PyTorch.
- To gain practical experience with fundamental image handling techniques in a deep learning context, with specific focus on image scaling and normalization concepts.
- To apply various image transformations to demonstrate a deeper understanding of data augmentation strategies and their utility in training models.

2 Data Augmentation and Transformations

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

Data augmentation is typically applied to the images after pixel-value scaling and pixel-value normalization. We 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.

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.

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

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.

3 Programming Tasks

```
1 """
2 ECE 60146 - Homework 2
3 Name: Triparna Mahata
4 Email: tmahata@purdue.edu
5
6 ImageNet Data Loading and Augmentation
7 """
8
9 import os
10 import torch
11 import numpy as np
12 import matplotlib.pyplot as plt
13 from torch.utils.data import Dataset, DataLoader
14 from torchvision import transforms
15 from PIL import Image
16 import time
17 import multiprocessing
18 multiprocessing.set_start_method('spawn', force=True)
```

3.1 Conda Environment

environment.yml file submitted as part of the zip file hw2_TriparnaMahata.zip.

3.2 Working with ImageNet

I obtained ImageNet images using the 'Option A': Downloaded the pre-prepared ImageNet subset from Brightspace.

```
1 REQUIRED_CLASSES = {
2     1: "goldfish",
3     151: "Chihuahua",
4     281: "tabby_cat",
5     291: "lion",
6     325: "sulphur_butterfly",
7     386: "African_elephant",
8     430: "basketball",
9     466: "bullet_train",
10    496: "Christmas_stocking",
11    950: "orange"
12 }
13
14 def get_class_name(label):
15     """Get class name from integer label."""
16     return REQUIRED_CLASSES.get(label, f"class_{label}")
17
18 # TRANSFORMS
19 transform_custom = transforms.Compose([
20     transforms.RandomHorizontalFlip(p=0.2),
21     transforms.RandomAffine(degrees=20, translate=(0.1,0.1), scale=(0.9,1.1)),
22     transforms.Resize((224,224)),
23     transforms.GaussianBlur(3,(0.3,1.5)),
24     transforms.ToTensor(),
25     transforms.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225])
26 ])
27
28 transform_basic = transforms.Compose([
29     transforms.Resize((224,224)),
30     transforms.ToTensor()
31 ])
32
33 # OPTION A : Dataset class for provided subset
34 class ImageNetSubset(Dataset):
35     """
36     Dataset for loading the provided ImageNet subset.
```

```

37 imagenet_subset/
38     1/      (goldfish)
39         00001.JPEG
40         ...
41     151/    (Chihuahua)
42         00001.JPEG
43         ...
44     ...
45 Args:
46     root(str): Path to imagenet_subset folder
47     class_labels(list): List of integer labels to load (e.g. [1, 151, 281])
48     images_per_class(int): Number of images to load per class
49     transform(callable): Transform to apply to images
50 """
51 def __init__(self, root, class_labels, images_per_class = 5, transform = None):
52     self.root = root
53     self.class_labels = class_labels
54     self.images_per_class = images_per_class
55     self.transform = transform
56
57     self.samples = [] # List of (image_path, label)
58     self._load_samples()
59
60     print (f"Loaded {len(self.samples)} images from {len(class_labels)} classes")
61
62 def _load_samples(self):
63     """Load image paths for each requested class."""
64     # For each label in self.class_labels:
65     #     1. Build path to class folder: os.path.join(self.root, str(label))
66     #     2. List all .JPEG /.jpg /.png files in that folder
67     #     3. Take first self.images_per_class images
68     #     4. Append (image_path, label) to self.samples
69
70     for label in self.class_labels:
71         class_dir = os.path.join(self.root, str(label))
72         all_files = sorted(os.listdir(class_dir))
73         images = [f for f in all_files if f.lower().endswith(('.jpeg', '.jpg', '.png'))]
74         selected_images = images[:self.images_per_class]
75         for img_name in selected_images:
76             img_path = os.path.join(class_dir, img_name)
77             self.samples.append((img_path, label))
78
79 def __len__(self):
80     return len(self.samples)
81
82 def __getitem__(self, index):
83     """
84     Returns:
85         image(Tensor): Transformed image
86         label(int): Class label (e.g., 1 for goldfish)
87     """
88     # 1. Get image_path, label from self.samples[index]
89     # 2. Load image using PIL: Image.open(path).convert('RGB')
90     # 3. Apply self.transform if not None
91     # 4. Return (image, label)
92
93     image_path, label = self.samples[index]
94     image = Image.open(image_path).convert('RGB')
95     if self.transform:
96         image = self.transform(image)
97     return image, label

```

3.3 Using DataLoader for Parallel Processing

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

```
1 # Custom dataset for my own images
2 class CustomDataset(Dataset):
3     """Dataset for loading custom images from a folder."""
4     def __init__(self, root, transform = None):
5         self.root = root
6         self.transform = transform
7
8         # Load all image paths from root folder
9         # Filter for: .jpg, .jpeg, .png, .bmp, .gif
10        self.image_paths = []
11
12        valid_extns = ('.jpg', '.jpeg', '.png', '.bmp', '.gif')
13
14        for filename in sorted(os.listdir(self.root)):
15            if filename.lower().endswith(valid_extns):
16                full_path = os.path.join(self.root, filename)
17                self.image_paths.append(full_path)
18
19        print(f"Found {len(self.image_paths)} images in {root}")
20
21    def __len__(self):
22        return len(self.image_paths)
23
24    def __getitem__(self, index):
25        img_path = self.image_paths[index]
26        image = Image.open(img_path).convert('RGB')
27        if self.transform:
28            image = self.transform(image)
29        return image, img_path
30#####
31# Utility Functions
32def denormalize(tensor, mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]):
33    """ Denormalize a tensor image for display."""
34    mean = torch.tensor(mean).view(3,1,1)
35    std = torch.tensor(std).view(3,1,1)
36    return tensor * std + mean
37
38def show_images(images, titles, rows, cols, figsize=(15,10), save_path=None):
39    """Display a grid of images."""
40    fig, axes = plt.subplots(rows, cols, figsize = figsize)
41    axes = axes.flatten() if rows*cols > 1 else [axes]
42
43    for i,(img,title) in enumerate(zip(images,titles)):
44        if i >= len(axes):
45            break
46        if isinstance(img, torch.Tensor):
47            if img.min() < 0 or img.max() > 1:
48                img = denormalize(img)
49            img = img.permute(1,2,0).numpy()
50
51        img = np.clip(img,0,1)
52        axes[i].imshow(img)
53        axes[i].set_title(title,fontsize = 10)
54        axes[i].axis('off')
55
56    plt.tight_layout()
57    if save_path:
58        plt.savefig(save_path, dpi=150, bbox_inches='tight')
59    plt.show()
60#####
61# Utility Class
```

```

62 class RepeatedDataset(Dataset):
63     def __init__(self, dataset, target_length=1000):
64         self.dataset = dataset
65         self.target_length = target_length
66         self.original_length = len(dataset)
67
68     def __len__(self):
69         return self.target_length
70
71     def __getitem__(self, index):
72         return self.dataset[index % self.original_length]
73 #####
74 def main():
75     custom_root = './my_images'
76     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)
77     loader = DataLoader(custom_dataset,
78                         batch_size = 4,
79                         shuffle = True,
80                         num_workers = 2)
81 #####
82 if __name__ == "__main__":
83     main()
84

```

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

```

1 def main():
2     custom_root = './my_images'
3     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)
4     loader = DataLoader(custom_dataset,
5                         batch_size = 4,
6                         shuffle = True,
7                         num_workers = 2)
8
9     # Plot all 4 images from a single batch as returned by the DataLoader.
10    dataiter = iter(loader)
11    images, paths = next(dataiter)
12    titles = [os.path.basename(p) for p in paths]
13    show_images(images, titles, rows=1, cols=4, save_path='batch_visualization.png')
14 #####
15 if __name__ == "__main__":
16     main()
17

```



3. Compare performance for parallel data loading:

- Generate 1000 custom images.

```

1 def main():
2     custom_root = './my_images'
3     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)

```

```

4
5     # Generate 1000 custom images
6     augmented_dataset = RepeatedDataset(custom_dataset, target_length=1000)
7 #####
8 if __name__ == "__main__":
9     main()
10

```

- Measure the time taken to load and augment all 1000 images using `__getitem__` in a loop.

```

1 def main():
2     custom_root = './my_images'
3     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)
4
5     # Generate 1000 custom images
6     augmented_dataset = RepeatedDataset(custom_dataset, target_length=1000)
7
8     # Measure time taken to load and augment all 1000 images using __getitem__ in a loop
9     print("Time taken to load and augment all 1000 images using __getitem__ in a loop")
10    start_time = time.time()
11    for i in range(1000):
12        _ = augmented_dataset[i]
13    end_time = time.time()
14    get_item_duration = end_time - start_time
15    print(f"__getitem__ Loop Time: {get_item_duration:.4f} seconds")
16 #####
17 if __name__ == "__main__":
18     main()
19

```

- Measure the time taken by the DataLoader to process all 1000 images with varying `batch_size` and `num_workers`.

```

1 def main():
2     custom_root = './my_images'
3     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)
4
5     # Generate 1000 custom images
6     augmented_dataset = RepeatedDataset(custom_dataset, target_length=1000)
7
8     # Measure time taken by the DataLoader to process all 1000 images with varying batch
9     size and num workers.
10    configs = [(16, 0),
11               (16, 2),
12               (16, 4),
13               (32, 4),
14               (64, 4)]
15
16    print(f"{'batch_size':<15} {'num_workers':<15} {'Time (sec)':<15}")
17    print("-" * 45)
18
19    for batch_size, num_workers in configs:
20        loader = DataLoader(augmented_dataset,
21                            batch_size=batch_size,
22                            shuffle=True,
23                            num_workers=num_workers)
24
25        start_t = time.time()
26        for batch_images, batch_paths in loader:
27            pass
28        end_t = time.time()
29        duration = end_t - start_t
30        print(f"{batch_size:<15} {num_workers:<15} {duration:.4f}")
31 #####
32 if __name__ == "__main__":
33     main()
34

```

- Report your findings in a table.

```

1 Time taken to load and augment all 1000 images using __getitem__ in a loop
2 __getitem__ Loop Time: 12.9741 seconds
3 batch_size      num_workers      Time (sec)
4 -----
5 16              0              12.6696
6 16              2              17.8498
7 16              4              25.1295
8 32              4              25.2466
9 64              4              24.8814
10

```

The multiprocessing start method ‘spawn’ creates fresh process instances. This introduces significant overhead because each worker process must re-import libraries and re-initialize the dataset environment every time a worker is spun up. For small datasets or simple transforms, this initialization cost outweighs the benefits of parallel loading, making `num_workers=0` significantly faster.

4. Compute the maximum of each RGB channel value of the images before and after normalization for one batch.

```

1 def main():
2     custom_root = './my_images'
3     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)
4
5     # Generate 1000 custom images
6     augmented_dataset = RepeatedDataset(custom_dataset, target_length=1000)
7
8     # Computing the maximum of each RGB channel value of the images before and after
9     # normalization for one batch.
10    check_loader = DataLoader(augmented_dataset, batch_size=32, shuffle=True)
11    images_norm, _ = next(iter(check_loader)) # These are AFTER normalization
12    images_unnorm = denormalize(images_norm)
13
14    def get_channel_max(tensor):
15        # Tensor shape: [Batch, 3, Height, Width]
16        # Permute to [3, Batch, Height, Width] then flatten to [3, -1]
17        # This groups all R pixels, all G pixels, all B pixels
18        flattened = tensor.permute(1, 0, 2, 3).reshape(3, -1)
19        return flattened.max(dim=1).values
20
21    max_vals_unnorm = get_channel_max(images_unnorm)
22    max_vals_norm = get_channel_max(images_norm)
23
24    print(f"{'Channel':<10} | {'Max (Before Norm)':<20} | {'Max (After Norm)':<20}")
25    print("-" * 55)
26
27    channel_names = ['R', 'G', 'B']
28    for i, name in enumerate(channel_names):
29        print(f"{name:<10} | {max_vals_unnorm[i]:.4f}{' ':>14} | {max_vals_norm[i]:.4f}")
30
31    #####
32    if __name__ == "__main__":
33        main()

```

Channel	Max (Before Norm)	Max (After Norm)
R	1.0000	2.2489
G	1.0000	2.4286
B	1.0000	2.6400

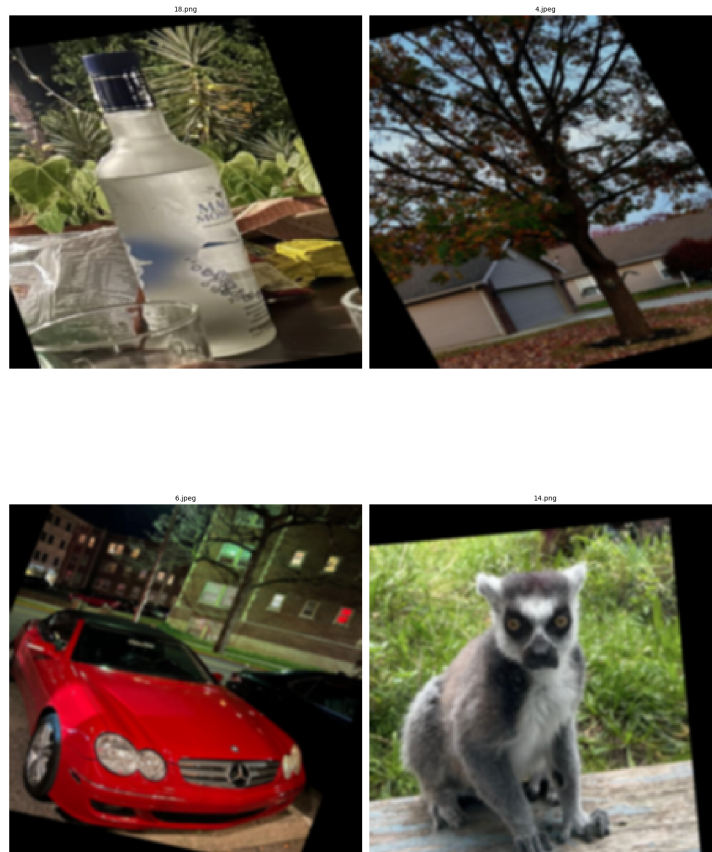
3.4 Exploring Random Seed and Reproducibility

1. Without setting a random seed:

```

1 def main():
2
3     custom_root = './my_images'
4     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)
5     loader = DataLoader(custom_dataset, batch_size=2, shuffle=True)
6
7     # --- RUN 1 ---
8     data_iter = iter(loader)
9     images_1, paths_1 = next(data_iter)
10    titles_1 = [os.path.basename(p) for p in paths_1]
11    print(f"Batch 1 contains: {titles_1}")
12    show_images(images_1, titles_1, rows=1, cols=2)
13
14    # --- RUN 2 ---
15    data_iter = iter(loader)
16    images_2, paths_2 = next(data_iter)
17    titles_2 = [os.path.basename(p) for p in paths_2]
18    print(f"Batch 2 contains: {titles_2}")
19    show_images(images_2, titles_2, rows=1, cols=2)
20
21    #####
22    if __name__ == "__main__":
23        main()
24

```



2. With a random seed:

```

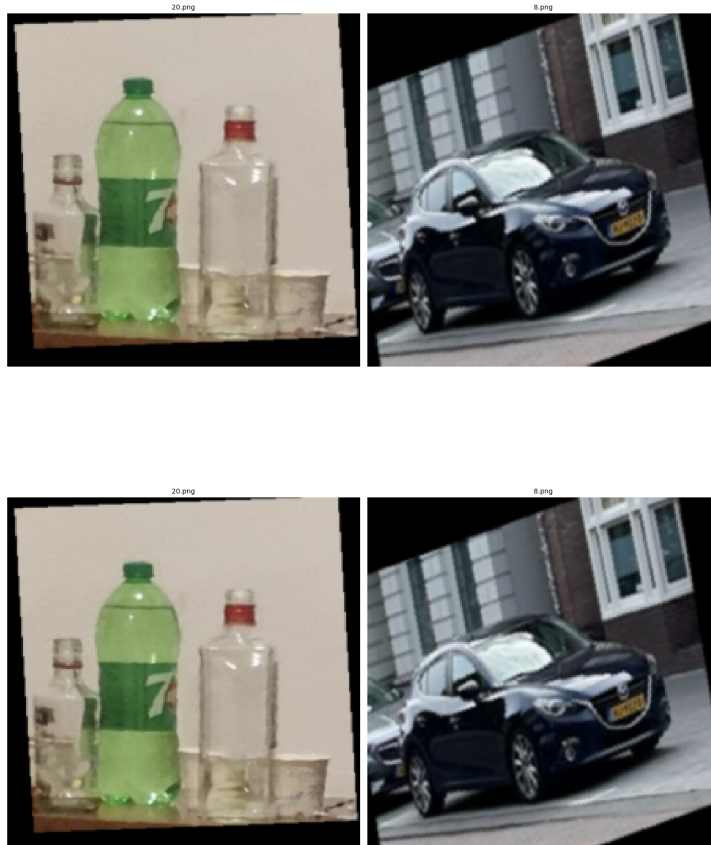
1 def set_seed(seed = 60146):
2     """Set random seed for reproducibility."""
3     import random
4     random.seed(seed)
5     np.random.seed(seed)
6     torch.manual_seed(seed)

```

```

7  if torch.cuda.is_available():
8      torch.cuda.manual_seed_all(seed)
9
1 def main():
2
3     custom_root = './my_images'
4     custom_dataset = CustomDataset(root=custom_root, transform=transform_custom)
5     loader = DataLoader(custom_dataset, batch_size=2, shuffle=True)
6
7     # --- RUN 1 ---
8     set_seed(60146)
9     data_iter = iter(loader)
10    images_1, paths_1 = next(data_iter)
11    titles_1 = [os.path.basename(p) for p in paths_1]
12    print(f"Batch 1 contains: {titles_1}")
13    show_images(images_1, titles_1, rows=1, cols=2)
14
15    # --- RUN 2 ---
16    set_seed(60146)
17    data_iter = iter(loader)
18    images_2, paths_2 = next(data_iter)
19    titles_2 = [os.path.basename(p) for p in paths_2]
20    print(f"Batch 2 contains: {titles_2}")
21    show_images(images_2, titles_2, rows=1, cols=2)
22
23    #####
24    if __name__ == "__main__":
25        main()
26

```



Setting a random seed initializes the random number generator to a fixed starting state, ensuring that stochastic processes like data shuffling produce the exact same sequence of results every time the script is run. This repro-

ducibility is critical in deep learning because it allows researchers to isolate variables, confirming that changes in model performance are due to actual code improvements rather than random chance.

3.5 Skeleton Code

```
1 """
2 ECE 60146 - Homework 2
3 Name: Triparna Mahata
4 Email: tmahata@purdue.edu
5
6 ImageNet Data Loading and Augmentation
7 """
8 #####
9 import os
10 import torch
11 import numpy as np
12 import matplotlib.pyplot as plt
13 from torch.utils.data import Dataset, DataLoader
14 from torchvision import transforms
15 from PIL import Image
16 import time
17 import multiprocessing
18 multiprocessing.set_start_method('spawn', force=True)
19 #####
20 REQUIRED_CLASSES = {
21     1: "goldfish",
22     151: "Chihuahua",
23     281: "tabby_cat",
24     291: "lion",
25     325: "sulphur_butterfly",
26     386: "African_elephant",
27     430: "basketball",
28     466: "bullet_train",
29     496: "Christmas_stocking",
30     950: "orange"
31 }
32
33 def get_class_name(label):
34     """Get class name from integer label."""
35     return REQUIRED_CLASSES.get(label, f"class_{label}")
36 #####
37 # TRANSFORMS
38 transform_custom = transforms.Compose([
39     transforms.RandomHorizontalFlip(p=0.2),
40     transforms.RandomAffine(degrees=20, translate=(0.1,0.1), scale=(0.9,1.1)),
41     transforms.Resize((224,224)),
42     transforms.GaussianBlur(3,(0.3,1.5)),
43     transforms.ToTensor(),
44     transforms.Normalize(mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225])
45 ])
46
47 transform_basic = transforms.Compose([
48     transforms.Resize((224,224)),
49     transforms.ToTensor()
50 ])
51 #####
52 # OPTION A : Dataset class for provided subset
53 class ImageNetSubset(Dataset):
54     """
55     Dataset for loading the provided ImageNet subset.
56     imagenet_subset/
57         1/      (goldfish)
58         00001.JPG
59         ...
60         151/    (Chihuahua)
```

```

61         00001.JPEG
62         ...
63         ...
64     Args:
65         root(str): Path to imagenet_subset folder
66         class_labels(list): List of integer labels to load (e.g. [1, 151, 281])
67         images_per_class(int): Number of images to load per class
68         transform (callable): Transform to apply to images
69     """
70     def __init__(self, root, class_labels, images_per_class = 5, transform = None):
71         self.root = root
72         self.class_labels = class_labels
73         self.images_per_class = images_per_class
74         self.transform = transform
75
76         self.samples = [] # List of (image_path, label)
77         self._load_samples()
78
79         print (f"Loaded {len(self.samples)} images from {len(class_labels)} classes")
80
81     def _load_samples(self):
82         """Load image paths for each requested class."""
83         # For each label in self.class_labels:
84         #     1. Build path to class folder: os.path.join(self.root, str(label))
85         #     2. List all .JPEG /.jpg /.png files in that folder
86         #     3. Take first self.images_per_class images
87         #     4. Append (image_path, label) to self.samples
88
89         for label in self.class_labels:
90             class_dir = os.path.join(self.root, str(label))
91             all_files = sorted(os.listdir(class_dir))
92             images = [f for f in all_files if f.lower().endswith(('.jpeg', '.jpg', '.png'))]
93             selected_images = images[:self.images_per_class]
94             for img_name in selected_images:
95                 img_path = os.path.join(class_dir, img_name)
96                 self.samples.append((img_path, label))
97
98     def __len__(self):
99         return len(self.samples)
100
101     def __getitem__(self, index):
102         """
103         Returns:
104             image(Tensor): Transformed image
105             label(int): Class label (e.g., 1 for goldfish)
106         """
107         # 1. Get image_path, label from self.samples[index]
108         # 2. Load image using PIL: Image.open(path).convert('RGB')
109         # 3. Apply self.transform if not None
110         # 4. Return (image, label)
111
112         image_path, label = self.samples[index]
113         image = Image.open(image_path).convert('RGB')
114         if self.transform:
115             image = self.transform(image)
116         return image, label
117 #####
118 # Custom dataset for my own images
119 class CustomDataset(Dataset):
120     """Dataset for loading custom images from a folder."""
121     def __init__(self, root, transform = None):
122         self.root = root
123         self.transform = transform
124
125         # Load all image paths from root folder
126         # Filter for: .jpg, .jpeg, .png, .bmp, .gif

```

```

127     self.image_paths = []
128
129     valid_extns = ('.jpg', '.jpeg', '.png', '.bmp', '.gif')
130
131     for filename in sorted(os.listdir(self.root)):
132         if filename.lower().endswith(valid_extns):
133             full_path = os.path.join(self.root, filename)
134             self.image_paths.append(full_path)
135
136     print(f"Found {len(self.image_paths)} images in {root}")
137
138     def __len__(self):
139         return len(self.image_paths)
140
141     def __getitem__(self, index):
142         img_path = self.image_paths[index]
143         image = Image.open(img_path).convert('RGB')
144         if self.transform:
145             image = self.transform(image)
146         return image, img_path
147 #####
148 # Utility Class
149 class RepeatedDataset(Dataset):
150     def __init__(self, dataset, target_length=1000):
151         self.dataset = dataset
152         self.target_length = target_length
153         self.original_length = len(dataset)
154
155     def __len__(self):
156         return self.target_length
157
158     def __getitem__(self, index):
159         return self.dataset[index % self.original_length]
160 #####
161 # Utility Functions
162 def denormalize(tensor, mean=[0.485,0.456,0.406], std=[0.229,0.224,0.225]):
163     """ Denormalize a tensor image for display."""
164     mean = torch.tensor(mean).view(3,1,1)
165     std = torch.tensor(std).view(3,1,1)
166     return tensor * std + mean
167
168 def show_images(images, titles, rows, cols, figsize=(15,10), save_path=None):
169     """Display a grid of images."""
170     fig, axes = plt.subplots(rows, cols, figsize = figsize)
171     axes = axes.flatten() if rows*cols > 1 else [axes]
172
173     for i,(img,title) in enumerate(zip(images,titles)):
174         if i >= len(axes):
175             break
176         if isinstance(img, torch.Tensor):
177             if img.min() < 0 or img.max() > 1:
178                 img = denormalize(img)
179             img = img.permute(1,2,0).numpy()
180
181         img = np.clip(img,0,1)
182         axes[i].imshow(img)
183         axes[i].set_title(title,fontsize = 10)
184         axes[i].axis('off')
185
186     plt.tight_layout()
187     if save_path:
188         plt.savefig(save_path, dpi=150, bbox_inches='tight')
189     plt.show()
190
191 def set_seed(seed = 60146):
192     """Set random seed for reproducibility."""

```

```

193 import random
194 random.seed(seed)
195 np.random.seed(seed)
196 torch.manual_seed(seed)
197 if torch.cuda.is_available():
198     torch.cuda.manual_seed_all(seed)
199 #####
200 def main():
201     # Required class labels
202     required_labels = list(REQUIRED_CLASSES.keys())
203     print("Required classes:", required_labels)
204     for label in required_labels:
205         print(f" {label}: {get_class_name(label)}")
206
207     print("\n" + "=" * 60)
208
209     ##### Task 1: Load ImageNet (50 images: 10 classes x 5 images) #####
210     print("Task 1: Loading ImageNet")
211     print("=" * 60)
212
213     # Creating dataset using Option A
214     imagenet_dataset = ImageNetSubset(root = './imagenet_subset',
215                                       class_labels = required_labels,
216                                       images_per_class = 5,
217                                       transform = transform_custom)
218
219     print("\n" + "=" * 60)
220
221     ##### Task 2: Visualize ImageNet (1 image per class => 10 images) #####
222     print("Task 2: Visualizing ImageNet")
223     print("=" * 60)
224
225     # Display 10 images (1 per class) in 2x5 grid
226     # Save as 'imagenet_samples.png'
227     images_display = []
228     titles_display = []
229     found_labels = set()
230
231     for i in range(len(imagenet_dataset)):
232         img, label = imagenet_dataset[i]
233
234         if label not in found_labels:
235             found_labels.add(label)
236             images_display.append(img)
237             titles_display.append(get_class_name(label))
238
239         if len(found_labels) == len(required_labels):
240             break
241
242     show_images(
243         images=images_display,
244         titles=titles_display,
245         rows=2,
246         cols=5,
247         save_path='imagenet_samples.png'
248     )
249
250     print("\n" + "=" * 60)
251
252     ##### Task 3: Show augmentation effects #####
253     print("Task 3: Augmentation comparison")
254     print("=" * 60)
255
256     # For 3 images, show original + 3 augmented versions
257     idx_aug = np.random.choice(50, 3, replace=False)
258     print(f"Showing augmentation for images at indices: {idx_aug}")

```

```

259 aug_images = []
260 aug_titles = []
261
262 for idx in idx_aug:
263     img_path, label = imagenet_dataset.samples[idx]
264     class_name = get_class_name(label)
265
266     pil_img = Image.open(img_path).convert('RGB')
267     orig_tensor = transform_basic(pil_img)
268     aug_images.append(orig_tensor)
269     aug_titles.append(f"Original\n{class_name}")
270
271     for k in range(3):
272         aug_tensor = transform_custom(pil_img)
273         aug_images.append(aug_tensor)
274         aug_titles.append(f"Augmented Image {k+1}")
275
276 show_images(
277     images=aug_images,
278     titles=aug_titles,
279     rows=3,
280     cols=4,
281     figsize=(12, 10),
282     save_path='augmentation_comparison.png',
283 )
284
285 print("\n" + "=" * 60)
286
287 ##### Task 4: Custom dataset #####
288 print("Task 4: Custom dataset")
289 print("=" * 60)
290
291 # Create CustomDataset, augment 20 images to 50
292 # Display 10 samples, save as 'custom_samples.png'
293
294 custom_root = './my_images'
295 my_orig_dataset = CustomDataset(root=custom_root, transform=None)
296
297 final_data = []
298
299 for i in range(len(my_orig_dataset)):
300     pil_img, _ = my_orig_dataset[i]
301     tensor_img = transform_basic(pil_img)
302     final_data.append((tensor_img, "Original"))
303
304 aug_count = 0
305 while len(final_data) < 50:
306     idx = aug_count % len(my_orig_dataset)
307     pil_img, _ = my_orig_dataset[idx]
308     tensor_img = transform_custom(pil_img)
309     final_data.append((tensor_img, "Augmented"))
310     aug_count += 1
311
312 sample_indices = np.random.choice(len(final_data), 10, replace=False)
313 disp_imgs = [final_data[i][0] for i in sample_indices]
314 disp_titles = [final_data[i][1] for i in sample_indices]
315
316 show_images(images=disp_imgs,
317             titles=disp_titles,
318             rows=2,
319             cols=5,
320             save_path='custom_samples.png')
321
322 print("\n" + "=" * 60)
323
324 ##### Task 5: DataLoader Performance #####

```

```

325 print("Task 5: DataLoader Performance")
326 print("=" * 60)
327
328 # Compare loading times with different batch_size / num_workers
329 # Test: (16,0), (16,4), (64,0), (64,4)
330
331 test_dataset = RepeatedDataset(imagenet_dataset, target_length=1000)
332 print(f"Base images: {len(imagenet_dataset)}")
333 print(f"Performance dataset size: {len(test_dataset)}")
334 print("-" * 45)
335 print("Measuring __getitem__ loop time (Single Process)")
336 start_time = time.time()
337 for i in range(len(test_dataset)):
338     _ = test_dataset[i]
339 end_time = time.time()
340 loop_duration = end_time - start_time
341 print(f"Time to load in simple loop: {loop_duration:.4f} sec")
342 print("-" * 45)
343
344 # Configurations: (batch_size, num_workers)
345 configs = [(16, 0),
346            (16, 4),
347            (64, 0),
348            (64, 4)]
349
350 print(f"{'batch_size':<15} {'num_workers':<15} {'Time (sec)':<15}")
351 print("-" * 45)
352
353 for batch_size, num_workers in configs:
354     loader = DataLoader(test_dataset,
355                        batch_size = batch_size,
356                        shuffle = True,
357                        num_workers = num_workers)
358     start_t = time.time()
359     for batch in loader:
360         pass
361     end_t = time.time()
362     duration = end_t - start_t
363     print(f"{batch_size:<15} {num_workers:<15} {duration:.4f}")
364
365 print("\n" + "=" * 60)
366
367 ##### Task 6: RGB Statistics #####
368 print("Task 6: RGB Statistics")
369 print("=" * 60)
370
371 # Compute min / max RGB before and after normalization
372 min_before = torch.tensor([float('inf'), float('inf'), float('inf')])
373 max_before = torch.tensor([float('-inf'), float('-inf'), float('-inf')])
374
375 min_after = torch.tensor([float('inf'), float('inf'), float('inf')])
376 max_after = torch.tensor([float('-inf'), float('-inf'), float('-inf')])
377
378 for i in range(len(imagenet_dataset)):
379     img, _ = imagenet_dataset[i]
380     flat_img = img.view(3, -1)
381
382     current_min = flat_img.min(dim=1).values
383     current_max = flat_img.max(dim=1).values
384
385     min_after = torch.min(min_after, current_min)
386     max_after = torch.max(max_after, current_max)
387
388     img_unnorm = denormalize(img)
389     flat_unnorm = img_unnorm.view(3, -1)
390

```

```

391     current_min_un = flat_unnorm.min(dim=1).values
392     current_max_un = flat_unnorm.max(dim=1).values
393
394     min_before = torch.min(min_before, current_min_un)
395     max_before = torch.max(max_before, current_max_un)
396
397     channels = ['R', 'G', 'B']
398     print("\nStatistics BEFORE Normalization:")
399     print(f"{'Channel':<10} {'Min':<10} {'Max':<10}")
400     print("-" * 35)
401     for i, c in enumerate(channels):
402         print(f"{c:<10} {min_before[i]:.4f}      {max_before[i]:.4f}")
403
404     print("\nStatistics AFTER Normalization:")
405     print(f"{'Channel':<10} {'Min':<10} {'Max':<10}")
406     print("-" * 35)
407     for i, c in enumerate(channels):
408         print(f"{c:<10} {min_after[i]:.4f}      {max_after[i]:.4f}")
409
410     print("\n" + "=" * 60)
411
412     ##### Task 7: Reproducibility #####
413     print("Task 7: Reproducibility")
414     print("=" * 60)
415
416     # Test with and without set_seed(60146)
417     def get_first_batch_sample():
418         loader = DataLoader(imagenet_dataset, batch_size=5, shuffle=True)
419         images, labels = next(iter(loader))
420         return images, labels.tolist()
421
422     print("--- Experiment A: With set_seed(60146) ---")
423
424     # Run 1
425     set_seed(60146)
426     imgs_1, labels_1 = get_first_batch_sample()
427
428     # Run 2 (Resetting seed)
429     set_seed(60146)
430     imgs_2, labels_2 = get_first_batch_sample()
431
432     # Compare Run 1 vs Run 2
433     labels_match_12 = (labels_1 == labels_2)
434     images_match_12 = torch.equal(imgs_1, imgs_2)
435
436     if labels_match_12 and images_match_12:
437         print("SUCCESS: Batch matches perfectly after resetting seed.")
438     else:
439         print("FAILURE: Mismatch detected despite fixed seed.")
440
441     print("\n--- Experiment B: Without resetting seed ---")
442
443     # Run 3 (No seed reset)
444     imgs_3, labels_3 = get_first_batch_sample()
445
446     # Compare Run 2 vs Run 3
447     labels_match_23 = (labels_2 == labels_3)
448     images_match_23 = torch.equal(imgs_2, imgs_3)
449
450     if not labels_match_23 and not images_match_23:
451         print("SUCCESS: Batch differs (as expected) when seed is not reset.")
452     else:
453         print(f"NOTE: Batch happened to match randomly.\n")
454
455     print("\n" + "=" * 60)
456     print("Done!")

```

```

457 print("\n" + "=" * 60)
458 #####
459 if __name__ == "__main__":
460     main()

```

3.6 Task Summary

1. Load ImageNet: Implement the dataset class. Load 50 images (5 per class) from the 10 required classes.

```

1 Required classes: [1, 151, 281, 291, 325, 386, 430, 466, 496, 950]
2 1: goldfish
3 151: Chihuahua
4 281: tabby_cat
5 291: lion
6 325: sulphur_butterfly
7 386: African_elephant
8 430: basketball
9 466: bullet_train
10 496: Christmas_stocking
11 950: orange
12
13 =====
14 Task 1: Loading ImageNet
15 =====
16 Loaded 50 images from 10 classes
17

```

2. Visualize ImageNet: Display 10 images (one from each class) in a 2×5 grid.

```

1 =====
2 Task 2: Visualizing ImageNet
3 =====
4

```



3. Augmentation Comparison: For 3 images, show original alongside 3 augmented versions.

```

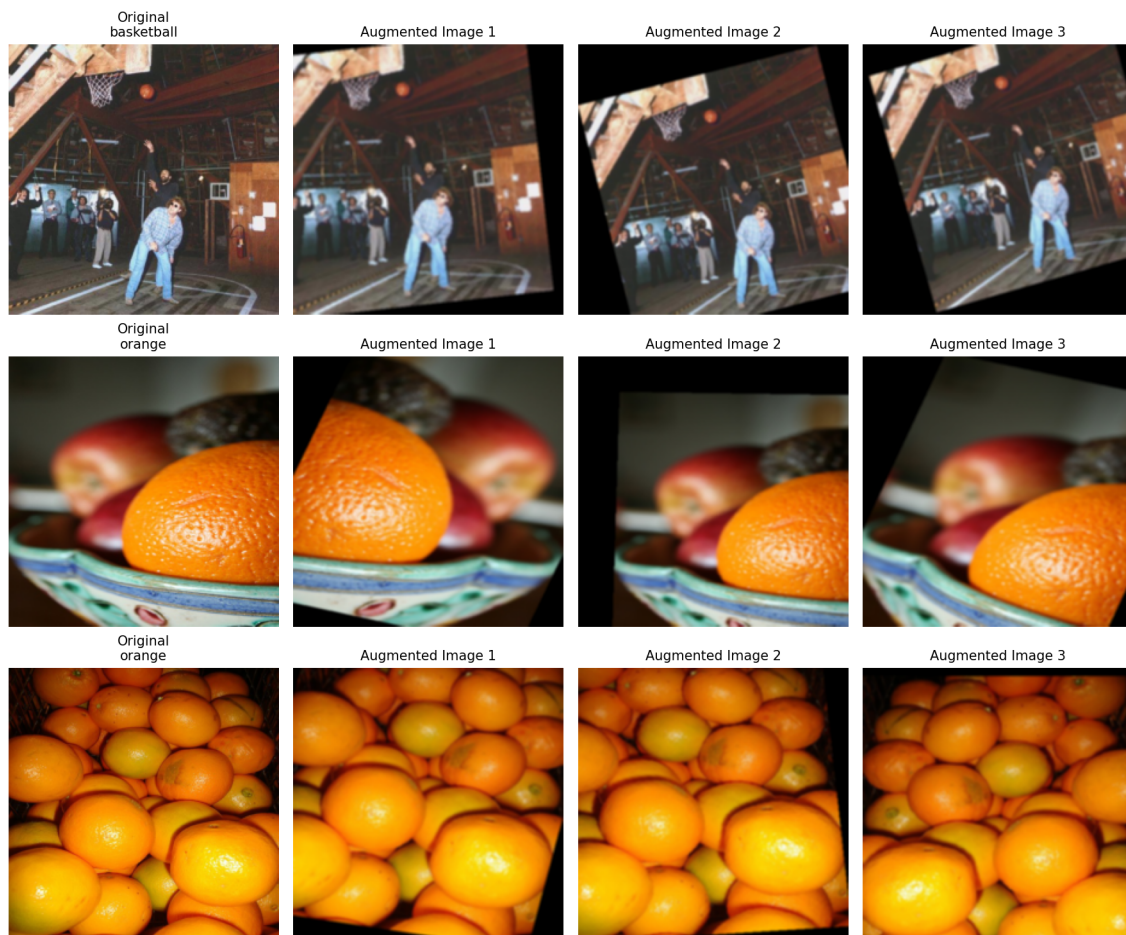
1 =====
2 Task 3: Augmentation comparison
3 =====

```

```

4 Showing augmentation for images at indices: [30 45 46]
5

```

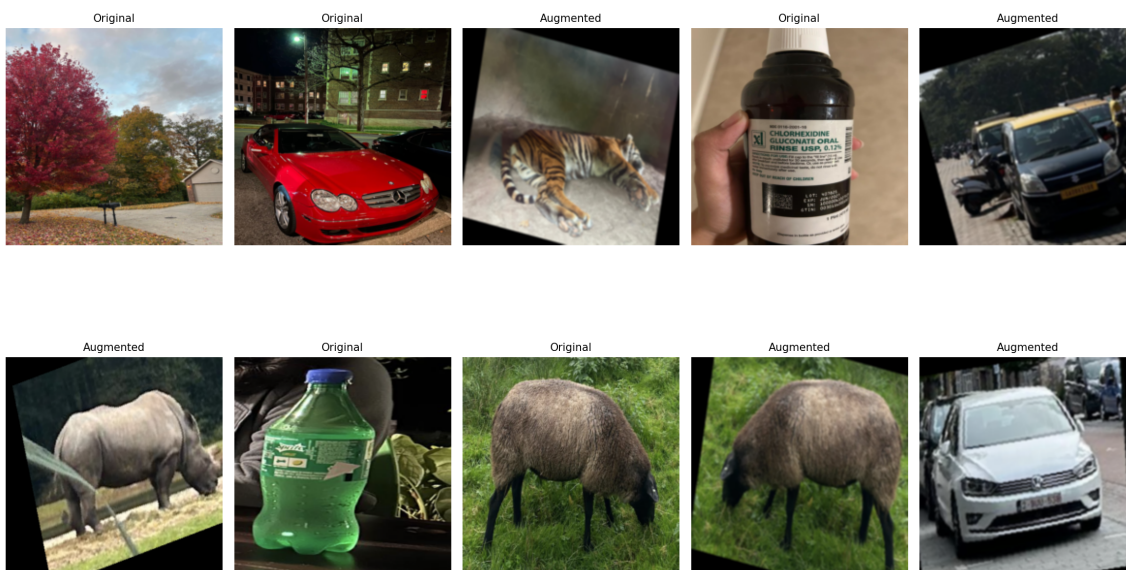


4. Custom Dataset: Collect 20 images, augment to 50, display 10 samples.

```

1 =====
2 Task 4: Custom dataset
3 =====
4 Found 20 images in ./my_images
5

```



5. DataLoader Performance: Compare loading times.

```
1 =====
2 Task 5: DataLoader Performance
3 =====
4 Base images: 50
5 Performance dataset size: 1000
6 -----
7 Measuring __getitem__ loop time (Single Process)
8 Time to load in simple loop: 3.1491 sec
9 -----
10 batch_size      num_workers      Time (sec)
11 -----
12 16               0               3.1638
13 16               4               22.5881
14 64               0               2.9990
15 64               4               22.4135
16
```

6. RGB Statistics: Compute min/max per channel before/after normalization.

```
1 =====
2 Task 6: RGB Statistics
3 =====
4
5 Statistics BEFORE Normalization:
6 Channel      Min      Max
7 -----
8 R            0.0000    1.0000
9 G            0.0000    1.0000
10 B           0.0000    1.0000
11
12 Statistics AFTER Normalization:
13 Channel      Min      Max
14 -----
15 R           -2.1179    2.2489
16 G           -2.0357    2.4286
17 B           -1.8044    2.6400
18
```

7. Reproducibility: Demonstrate effect of random seed on batch ordering.

```
1 =====
2 Task 7: Reproducibility
3 =====
4 --- Experiment A: With set_seed(60146) ---
5 SUCCESS: Batch matches perfectly after resetting seed.
6
7 --- Experiment B: Without resetting seed ---
8 SUCCESS: Batch differs (as expected) when seed is not reset.
9
```