

# BME646 / ECE60146 Homework 2 Report

Jonathan Stoschek

January 27, 2025

Spring 2025, Purdue University

## 1 Prepare CIFAR10

In this section, we load the CIFAR10 dataset using PyTorch and select only 5 classes (10 images from each class, leading to 50 images total). We apply a simple transformation to convert them to tensors, optionally scaling them into  $[0, 1]$  range.

### Code Snippet

```
1 from torchvision import datasets, transforms
2 from torch.utils.data import Subset
3
4 transform_cifar = transforms.Compose([
5     transforms.ToTensor(),
6 ])
7
8 selected_classes = [0, 2, 3, 5, 7]
9 cifar10_full = datasets.CIFAR10(root='./data', train=True,
10                                download=True, transform=transform_cifar)
11
12 cifar_indices = [i for i, (_, label) in enumerate(cifar10_full)
13                 if label in selected_classes]
14 subset_cifar10 = Subset(cifar10_full, cifar_indices[:50])
15
16 print(f"Size of CIFAR10 subset: {len(subset_cifar10)} images")
```

### Output

Size of CIFAR10 subset: 50 images

## 2 Custom Dataset

We create a custom dataset of 20 images (e.g., pictures of apples). We then apply data augmentations (random horizontal flips, random rotations, etc.) and generate more images (up to 50 total) to match the size of the CIFAR10 subset.

### Code Snippet

```
1 import os
2 from PIL import Image
3 import torch
4 from torch.utils.data import Dataset
5
6 class CustomAppleDataset(Dataset):
```

```

7     def __init__(self, root, transform=None):
8         self.root = root
9         self.image_paths = [os.path.join(root, img)
10                             for img in os.listdir(root)
11                             if img.lower().endswith(('.png', '.jpg', '.jpeg'))]
12         self.transform = transform
13
14     def __len__(self):
15         return len(self.image_paths)
16
17     def __getitem__(self, index):
18         img_path = self.image_paths[index]
19         image = Image.open(img_path).convert("RGB")
20         if self.transform:
21             image = self.transform(image)
22         # Return an image and a dummy label (0)
23         return image, 0

```

### 3 Data Augmentation

#### Transformations and Loading

```

1 import torchvision.transforms as T
2
3 transform_custom = T.Compose([
4     T.Resize((32, 32)),
5     T.RandomHorizontalFlip(),
6     T.RandomRotation(10),
7     T.ToTensor(),
8     T.Normalize((0.5,), (0.5,))
9 ])
10
11 custom_dataset = CustomAppleDataset(
12     root='apple_photos',
13     transform=transform_custom
14 )
15
16 # Augment up to 50 images
17 augmented_images = [custom_dataset[i % len(custom_dataset)][0]
18                     for i in range(50)]
19
20 print(f"Size of Custom Dataset (original): {len(custom_dataset)} images")

```

#### Output

Size of Custom Dataset (original): 20 images

### 4 Comparison via Visualization

Below we show five example images from the CIFAR10 subset and five example images from the augmented custom dataset. In practice, we can generate up to 50 or more images for thorough comparison.

#### Code Snippet

```

1 import matplotlib.pyplot as plt
2 from torch.utils.data import DataLoader
3
4 # Visualize CIFAR10 (5 images)
5 cifar_loader = DataLoader(subset_cifar10, batch_size=5, shuffle=False)
6 batch = next(iter(cifar_loader))
7 cifar_images, cifar_labels = batch

```

```

8
9 plt.figure(figsize=(10,2))
10 for i in range(5):
11     plt.subplot(1,5,i+1)
12     img = cifar_images[i].permute(1, 2, 0)
13     plt.imshow(img.numpy())
14     plt.axis('off')
15 plt.suptitle("CIFAR10 Subset Samples")
16 plt.show()
17
18 # Visualize custom dataset (5 images)
19 plt.figure(figsize=(10,2))
20 for i in range(5):
21     plt.subplot(1,5,i+1)
22     img = augmented_images[i].permute(1, 2, 0)
23     # Denormalize for display
24     img = (img * 0.5) + 0.5
25     plt.imshow(img.numpy().clip(0,1))
26     plt.axis('off')
27 plt.suptitle("Custom Augmented Samples")
28 plt.show()

```

## Sample Figures



Figure 1: Five images from the filtered CIFAR10 subset.



Figure 2: Five images from the custom dataset after augmentations.

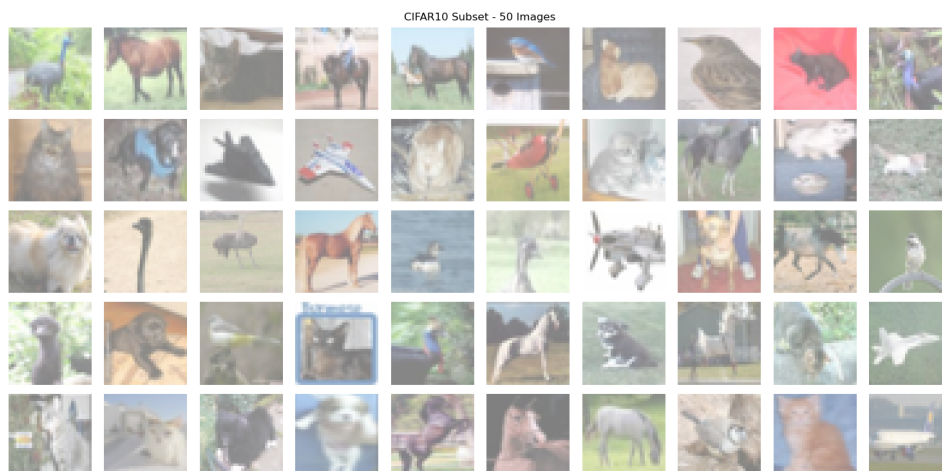


Figure 3: 50 images from the CIFAR10 dataset.

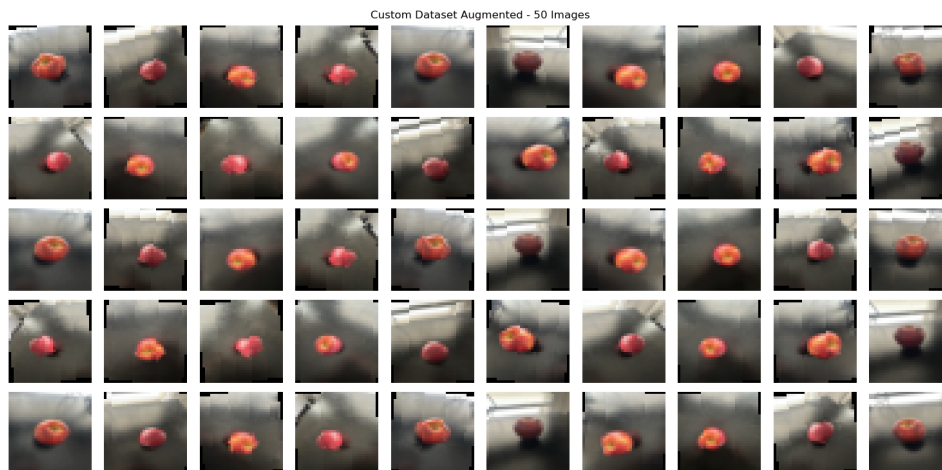


Figure 4: 50 images from the custom apple dataset.

## 5.1 Two Different Batch Sizes

We measure data loading performance for two different batch sizes (e.g., 4 and 16). We use a `DataLoader` around the same custom dataset and measure total time to load a fixed number of images (in this case, 1000).

### Code Snippet

```

1 combos = [(4, 1), (4, 2), (16, 1), (16, 2)]
2 results = []
3 for bs, nw in combos:
4     t0 = time.time()
5     count = 0
6     for batch_imgs, _ in DataLoader(dataset_with_norm,
7                                     batch_size=bs,
8                                     shuffle=False,
9                                     num_workers=nw):
10         count += batch_imgs.size(0)
11         if count >= 1000:
12             break
13     t1 = time.time()
14     results.append((bs, nw, t1 - t0))
15
16 print("\n=== Performance Table ===")
17 print("BatchSize | NumWorkers | Time (s)")
18 for (bs, nw, t) in results:
19     print(f"{bs:9d} | {nw:10d} | {t:..4f}")

```

### Output Example

```

=== Performance Table ===
BatchSize | NumWorkers | Time (s)
      4 |           1 | 6.9895
      4 |           2 | 12.3149
     16 |           1 | 7.0326
     16 |           2 | 12.0707

```

## 5.2 Two Different Workers

As shown in the table above, we also vary the `num_workers` parameter to see the performance difference. Some systems benefit from multi-threading, but others do not, depending on CPU and I/O overhead.

## 5.3 Plotting 4 Images

In addition, we plotted all 4 images from a single batch. Below is an example code snippet. Each image is displayed in a  $2 \times 2$  or  $1 \times 4$  grid.

```
1 loader_no_norm = DataLoader(dataset_no_norm, batch_size=4, shuffle=True,
    num_workers=2)
2 images_no_norm, _ = next(iter(loader_no_norm))
3
4 plt.figure(figsize=(8, 2))
5 for i in range(4):
6     plt.subplot(1,4,i+1)
7     img = images_no_norm[i].permute(1,2,0)
8     plt.imshow(img.numpy())
9     plt.axis('off')
10 plt.suptitle("One Batch - No Normalization")
11 plt.show()
```

## 6.1 Without Seed

When a random seed is *not* set, multiple runs with `shuffle=True` in the `DataLoader` can produce different orders of images in the first batch. We verify this by running the data loading code for a batch size of 2, observing that consecutive fetches from the `DataLoader` produce different images.

### Code Snippet

```
1 # No seed set here
2 loader_no_seed = DataLoader(dataset_no_seed,
3                             batch_size=2,
4                             shuffle=True,
5                             num_workers=0)
6
7 first_batch_no_seed, _ = next(iter(loader_no_seed))
8 # Plot the images from the first batch
9 ...
10 # Then fetch again
11 second_batch_no_seed, _ = next(iter(loader_no_seed))
12 # Plot the images from the second batch
```

## 6.2 With Seed

Once we fix the random seed (e.g., 60146) across `torch`, `Python`, and `NumPy`, the first batch remains consistent between runs.

### Code Snippet

```
1 import random
2 import numpy as np
3
4 seed = 60146
5 torch.manual_seed(seed)
6 random.seed(seed)
7 np.random.seed(seed)
8
9 loader_with_seed = DataLoader(dataset_no_seed,
10                              batch_size=2,
11                              shuffle=True,
12                              num_workers=0)
13
14 first_batch_with_seed, _ = next(iter(loader_with_seed))
15 # Plot
```

```

16 ...
17
18 # Reset the seed again
19 torch.manual_seed(seed)
20 random.seed(seed)
21 np.random.seed(seed)
22
23 second_batch_with_seed, _ = next(iter(loader_with_seed))
24 # Plot
25 ...

```

## Discussion

Setting the random seed ensures reproducible results in experiments, especially in deep learning pipelines where factors like shuffling data, initializing network parameters, or random data augmentations can cause variation between runs. This consistency is crucial for debugging and comparing different runs fairly.

```

1
2 import os
3 import random
4 import time
5 import numpy as np
6 import torch
7 from torch.utils.data import Dataset, DataLoader, Subset
8 import torchvision
9 from torchvision import datasets, transforms
10 from PIL import Image
11 import matplotlib.pyplot as plt
12
13 #####
14 # 2.1 and 2.2: Pixel Value Scaling & Normalization
15 #####
16
17 # Transformation pipeline for CIFAR10
18 transform_cifar = transforms.Compose([
19     transforms.ToTensor(), # scales [0,255] to [0,1]
20 ])
21
22 # Transformation pipeline for the custom dataset with random augmentations
23 transform_custom = transforms.Compose([
24     transforms.Resize((32, 32)),
25     transforms.RandomHorizontalFlip(),
26     transforms.RandomRotation(10),
27     transforms.ToTensor(),
28     transforms.Normalize((0.5,), (0.5,))
29 ])
30
31 # New Transformation Pipeline for random seed demonstration (NO random flip/rotate)
32 transform_custom_no_flip_rotate = transforms.Compose([
33     transforms.Resize((32, 32)),
34     transforms.ToTensor(),
35     transforms.Normalize((0.5,), (0.5,))
36 ])
37
38 #####
39 # Custom Dataset Class
40 #####
41 class CustomAppleDataset(Dataset):
42     """
43     A simple Dataset that reads all images from a directory (apple_photos)
44     and applies optional transformations.
45     """
46     def __init__(self, root, transform=None):
47         self.root = root
48         self.image_paths = [os.path.join(root, img) for img in os.listdir(root)]

```

```

49         if img.lower().endswith(('.png', '.jpg', '.jpeg'))]
50         self.transform = transform
51
52     def __len__(self):
53         return len(self.image_paths)
54
55     def __getitem__(self, index):
56         img_path = self.image_paths[index]
57         image = Image.open(img_path).convert("RGB")
58         if self.transform:
59             image = self.transform(image)
60         return image, 0
61
62 #####
63 # 3.2 Comparing CIFAR10 with a Custom Dataset
64 #####
65 def compare_cifar10_with_custom():
66     """
67     1) Load CIFAR10 and filter 5 classes with 10 images each.
68     2) Create a custom dataset from apple_photos with 20 images.
69     3) Augment them to generate 30 additional images.
70     4) Compare/visualize.
71     """
72
73     # Step 1: Load CIFAR10 and filter 5 classes with 10 images each => total 50
74     selected_classes = [0, 2, 3, 5, 7]
75     cifar10_full = datasets.CIFAR10(root='./data', train=True,
76                                     download=True, transform=transform_cifar)
77
78     # Indices of images belonging to the chosen 5 classes
79     cifar_indices = [i for i, (_, label) in enumerate(cifar10_full)
80                     if label in selected_classes]
81
82     # Slice out the first 50 from those classes (10 images per class)
83     subset_cifar10 = Subset(cifar10_full, cifar_indices[:50])
84
85     # Step 2: Load the custom dataset of ~20 apple images
86     custom_dataset = CustomAppleDataset(root='apple_photos',
87                                         transform=transform_custom)
88
89     # Step 3: Extrapolate (augment) to get 50 images.
90     augmented_images = [custom_dataset[i % len(custom_dataset)][0]
91                         for i in range(50)]
92
93     # Step 4: Print sizes and visualize
94     print(f"Size of CIFAR10 subset: {len(subset_cifar10)} images")
95     print(f"Size of Custom Dataset (original): {len(custom_dataset)} images")
96
97     # Visualization of a few samples from CIFAR10 Subset
98     cifar_loader = DataLoader(subset_cifar10, batch_size=5, shuffle=False)
99     batch = next(iter(cifar_loader))
100    cifar_images, cifar_labels = batch
101
102    plt.figure(figsize=(10, 2))
103    for i in range(5):
104        plt.subplot(1, 5, i+1)
105        img = cifar_images[i].permute(1, 2, 0)
106        plt.imshow(img.numpy())
107        plt.axis('off')
108    plt.suptitle("CIFAR10 Subset Samples")
109    plt.savefig("cifar10_subset_samples.png")
110    plt.close()
111
112    # Visualization of some augmented custom images
113    plt.figure(figsize=(10, 2))
114    for i in range(5):

```

```

115     plt.subplot(1, 5, i+1)
116     img = augmented_images[i].permute(1, 2, 0)
117     img = (img * 0.5) + 0.5
118     img = torch.clamp(img, 0, 1)
119     plt.imshow(img.numpy())
120     plt.axis('off')
121 plt.suptitle("Custom Augmented Samples")
122 plt.savefig("custom_augmented_samples.png")
123 plt.close()
124
125 # Function to generate and save a grid of images
126 def save_image_grid(images, title, filename, num_cols=10, num_rows=5):
127     plt.figure(figsize=(num_cols * 1.5, num_rows * 1.5))
128     for idx in range(num_cols * num_rows):
129         if idx >= len(images):
130             break
131         plt.subplot(num_rows, num_cols, idx + 1)
132         img = images[idx].permute(1, 2, 0) # [C, H, W] -> [H, W, C]
133         img = (img * 0.5) + 0.5
134         img = torch.clamp(img, 0, 1)
135         plt.imshow(img.numpy())
136         plt.axis('off')
137     plt.suptitle(title)
138     plt.tight_layout()
139     plt.subplots_adjust(top=0.95)
140     plt.savefig(filename)
141     plt.close()
142
143 # Prepare images from CIFAR10 subset
144 cifar_images_all = []
145 cifar_loader_full = DataLoader(subset_cifar10, batch_size=50, shuffle=False)
146 cifar_batch = next(iter(cifar_loader_full))
147 cifar_images_all = cifar_batch[0]
148
149 # Prepare images from Custom Dataset (augmented)
150 custom_images_all = torch.stack(augmented_images)
151
152 # Save CIFAR10 images grid
153 save_image_grid(
154     images=cifar_images_all,
155     title="CIFAR10 Subset - 50 Images",
156     filename="cifar10_50_images_grid.png",
157     num_cols=10,
158     num_rows=5
159 )
160
161 # Save Custom Dataset images grid
162 save_image_grid(
163     images=custom_images_all,
164     title="Custom Dataset Augmented - 50 Images",
165     filename="custom_dataset_50_images_grid.png",
166     num_cols=10,
167     num_rows=5
168 )
169
170 #####
171 # 3.3 Using DataLoader for Parallel Processing
172 #####
173 def demo_dataloader_parallel():
174     """
175     1) Wrap custom dataset in a DataLoader, batch_size=4, num_workers>1.
176     2) Plot all 4 images from a single batch.
177     3) Compare performance when loading 1000 images manually vs. DataLoader.
178     4) Compute max of each RGB channel before and after normalization.
179     """
180

```



```

181 no_norm_transform = transforms.Compose([
182     transforms.Resize((32, 32)),
183     transforms.ToTensor(),
184 ])
185 norm_transform = transforms.Compose([
186     transforms.Resize((32, 32)),
187     transforms.ToTensor(),
188     transforms.Normalize((0.5, ), (0.5, ))
189 ])
190
191 dataset_no_norm = CustomAppleDataset(root='apple_photos',
192                                     transform=no_norm_transform)
193 dataset_with_norm = CustomAppleDataset(root='apple_photos',
194                                       transform=norm_transform)
195
196 # 1) DataLoader with batch_size=4, num_workers=2
197 loader_no_norm = DataLoader(dataset_no_norm, batch_size=4, shuffle=True,
198                             num_workers=2)
199 loader_with_norm = DataLoader(dataset_with_norm, batch_size=4, shuffle=True,
200                               num_workers=2)
201
202 # 2) Plot all 4 images from a single batch
203 images_no_norm, _ = next(iter(loader_no_norm))
204 plt.figure(figsize=(8, 2))
205 for i in range(4):
206     plt.subplot(1, 4, i+1)
207     img = images_no_norm[i].permute(1, 2, 0)
208     plt.imshow(img.numpy()) # in [0,1]
209     plt.axis('off')
210 plt.suptitle("One Batch - No Normalization")
211 plt.savefig("batch_no_normalization.png")
212 plt.close()
213
214 # 3) Compare performance loading 1000 images manually vs. DataLoader
215
216 # (a) Manual getitem approach
217 t0 = time.time()
218 manual_images = []
219 for i in range(1000):
220     idx = i % len(dataset_with_norm)
221     img, _ = dataset_with_norm[idx]
222     manual_images.append(img)
223 t1 = time.time()
224 manual_time = t1 - t0
225
226 # (b) DataLoader approach
227 t2 = time.time()
228 dataloader_images = []
229 total_loaded = 0
230 for batch_imgs, _ in DataLoader(dataset_with_norm, batch_size=10, shuffle=False,
231                                num_workers=2):
232     for i in range(batch_imgs.size(0)):
233         dataloader_images.append(batch_imgs[i])
234         total_loaded += 1
235         if total_loaded >= 1000:
236             break
237     if total_loaded >= 1000:
238         break
239 t3 = time.time()
240 dataloader_time = t3 - t2
241
242 print("Time taken (manual getitem, 1000 images): {:.4f}
243       seconds".format(manual_time))
244 print("Time taken (DataLoader w/ batch_size=10, num_workers=2, 1000 images):
245       {:.4f} seconds".format(dataloader_time))

```

```

245 # 4) Demonstrate different (batch_size, num_workers) combinations
246 combos = [(4,1), (4,2), (16,1), (16,2)] # for instance
247 results = []
248 for bs, nw in combos:
249     t0 = time.time()
250     count = 0
251     for batch_imgs, _ in DataLoader(dataset_with_norm, batch_size=bs,
252                                     shuffle=False,
253                                     num_workers=nw):
254         count += batch_imgs.size(0)
255         if count >= 1000:
256             break
257     t1 = time.time()
258     results.append((bs, nw, t1 - t0))
259
260 print("\n=== Performance Table ===")
261 print("BatchSize | NumWorkers | Time (s)")
262 for (bs, nw, t) in results:
263     print(f"{bs:9d} | {nw:10d} | {t:.4f}")
264
265 # 5) Compute max of each channel for one batch "before and after" normalization
266 images_before, _ = next(iter(loader_no_norm))
267 images_after, _ = next(iter(loader_with_norm))
268
269 # max over entire batch for each channel
270 max_before_R = images_before[:, 0, :, :].max().item()
271 max_before_G = images_before[:, 1, :, :].max().item()
272 max_before_B = images_before[:, 2, :, :].max().item()
273
274 max_after_R = images_after[:, 0, :, :].max().item()
275 max_after_G = images_after[:, 1, :, :].max().item()
276 max_after_B = images_after[:, 2, :, :].max().item()
277
278 print("\nMax channel values BEFORE normalization: ",
279       f"R={max_before_R:.3f}, G={max_before_G:.3f}, B={max_before_B:.3f}")
280 print("Max channel values AFTER normalization: ",
281       f"R={max_after_R:.3f}, G={max_after_G:.3f}, B={max_after_B:.3f}")
282
283 #####
284 # 3.4 Exploring Random Seed and Reproducibility
285 #####
286 def demo_random_seed():
287     """
288     3.4 Exploring Random Seed and Reproducibility
289     1. Without setting a random seed:
290         Set the batch size to 2 and shuffle the dataset.
291         Plot the first batch of images. Exit the batch iterator and rerun
292         it. Note if the same two images appear in the first batch.
293     2. With a random seed:
294         Set a random seed to 60146 at the beginning of your script.
295         Repeat the previous exercise and compare the results. Note if the
296         same images appear in the first batch across iterations.
297     """
298
299     # Part 1: Without Setting a Random Seed
300     print("\nPart 1: Without Setting a Random Seed")
301
302     # Define the transformation without random augmentations
303     dataset_no_seed = CustomAppleDataset(
304         root='apple_photos',
305         transform=transform_custom_no_flip_rotate
306     )
307
308     # Create DataLoader with batch_size=2 and shuffle=True
309     loader_no_seed = DataLoader(dataset_no_seed, batch_size=2, shuffle=True,
310                                num_workers=0)

```

```

309
310 # Get the first batch of images
311 try:
312     first_batch_no_seed, _ = next(iter(loader_no_seed))
313 except StopIteration:
314     print("Dataset is empty or not enough images.")
315     return
316
317 # Plot the first batch of images
318 plt.figure(figsize=(4, 2))
319 for i in range(first_batch_no_seed.size(0)):
320     plt.subplot(1, 2, i+1)
321     img = first_batch_no_seed[i].permute(1, 2, 0)
322     img = (img * 0.5) + 0.5 # Unnormalize for display
323     img = torch.clamp(img, 0, 1)
324     plt.imshow(img.numpy())
325     plt.axis('off')
326 plt.suptitle("First Batch Without Seed")
327 plt.savefig("first_batch_no_seed.png")
328 plt.close()
329
330
331 # Get the first batch of images
332 try:
333     first_batch_no_seed, _ = next(iter(loader_no_seed))
334 except StopIteration:
335     print("Dataset is empty or not enough images.")
336     return
337
338 # Plot the first batch of images
339 plt.figure(figsize=(4, 2))
340 for i in range(first_batch_no_seed.size(0)):
341     plt.subplot(1, 2, i+1)
342     img = first_batch_no_seed[i].permute(1, 2, 0)
343     img = (img * 0.5) + 0.5 # Unnormalize for display
344     img = torch.clamp(img, 0, 1)
345     plt.imshow(img.numpy())
346     plt.axis('off')
347 plt.suptitle("Second Batch Without Seed")
348 plt.savefig("second_batch_no_seed.png")
349 plt.close()
350
351 # Part 2: With Setting a Random Seed
352 print("Part 2: With Setting a Random Seed")
353
354 # Set the random seed to 60146
355 seed = 60146
356 torch.manual_seed(seed)
357 random.seed(seed)
358 np.random.seed(seed)
359
360 # Create DataLoader with batch_size=2 and shuffle=True
361 loader_with_seed = DataLoader(dataset_no_seed, batch_size=2, shuffle=True,
362                               num_workers=0)
363
364 # Get the first batch of images
365 try:
366     first_batch_with_seed, _ = next(iter(loader_with_seed))
367 except StopIteration:
368     print("Dataset is empty or not enough images.")
369     return
370
371 # Plot the first batch of images
372 plt.figure(figsize=(4, 2))
373 for i in range(first_batch_with_seed.size(0)):
374     plt.subplot(1, 2, i+1)

```

```

374         img = first_batch_with_seed[i].permute(1, 2, 0)
375         img = (img * 0.5) + 0.5 # Unnormalize for display
376         img = torch.clamp(img, 0, 1)
377         plt.imshow(img.numpy())
378         plt.axis('off')
379     plt.suptitle("First Batch With Seed=60146")
380     plt.savefig("first_batch_with_seed.png")
381     plt.close()
382
383
384     # Set the random seed to 60146
385     seed = 60146
386     torch.manual_seed(seed)
387     random.seed(seed)
388     np.random.seed(seed)
389
390     # Create DataLoader with batch_size=2 and shuffle=True
391     loader_with_seed = DataLoader(dataset_no_seed, batch_size=2, shuffle=True,
392                                   num_workers=0)
393
394     # Get the first batch of images
395     try:
396         first_batch_with_seed, _ = next(iter(loader_with_seed))
397     except StopIteration:
398         print("Dataset is empty or not enough images.")
399         return
400
401     # Plot the first batch of images
402     plt.figure(figsize=(4, 2))
403     for i in range(first_batch_with_seed.size(0)):
404         plt.subplot(1, 2, i+1)
405         img = first_batch_with_seed[i].permute(1, 2, 0)
406         img = (img * 0.5) + 0.5 # Unnormalize for display
407         img = torch.clamp(img, 0, 1)
408         plt.imshow(img.numpy())
409         plt.axis('off')
410     plt.suptitle("Second Batch With Seed=60146")
411     plt.savefig("second_batch_with_seed.png")
412     plt.close()
413
414     #####
415     # Main Execution
416     #####
417     if __name__ == "__main__":
418         # Compare CIFAR10 with Custom Apple Dataset
419         compare_cifar10_with_custom()
420
421         # Demo DataLoader parallel loading & performance
422         demo_dataloader_parallel()
423
424         # Random Seed & Reproducibility
425         demo_random_seed()

```