# 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

```
from torchvision import datasets, transforms
from torch.utils.data import Subset

transform_cifar = transforms.Compose([
    transforms.ToTensor(),
])

selected_classes = [0, 2, 3, 5, 7]
cifar10_full = datasets.CIFAR10(root='./data', train=True,
                                download=True, transform=transform_cifar)

cifar_indices = [i for i, (_, label) in enumerate(cifar10_full)
                 if label in selected_classes]
subset_cifar10 = Subset(cifar10_full, cifar_indices[:50])

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

```
import os
from PIL import Image
import torch
from torch.utils.data import Dataset

class CustomAppleDataset(Dataset):
```

```
    def __init__(self, root, transform=None):
        self.root = root
        self.image_paths = [os.path.join(root, img)
                            for img in os.listdir(root)
                            if img.lower().endswith(('.png','.jpg','.jpeg'))]
        self.transform = transform

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

    def __getitem__(self, index):
        img_path = self.image_paths[index]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        # Return an image and a dummy label (0)
        return image, 0
```

# 3 Data Augmentation

## Transformations and Loading

```
import torchvision.transforms as T

transform_custom = T.Compose([
    T.Resize((32, 32)),
    T.RandomHorizontalFlip(),
    T.RandomRotation(10),
    T.ToTensor(),
    T.Normalize((0.5,), (0.5,))
])

custom_dataset = CustomAppleDataset(
    root='apple_photos',
    transform=transform_custom
)

# Augment up to 50 images
augmented_images = [custom_dataset[i % len(custom_dataset)][0]
                    for i in range(50)]

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

```
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader

# Visualize CIFAR10 (5 images)
cifar_loader = DataLoader(subset_cifar10, batch_size=5, shuffle=False)
batch = next(iter(cifar_loader))
cifar_images, cifar_labels = batch
```

```
plt.figure(figsize=(10,2))
for i in range(5):
    plt.subplot(1,5,i+1)
    img = cifar_images[i].permute(1, 2, 0)
    plt.imshow(img.numpy())
    plt.axis('off')
plt.suptitle("CIFAR10 Subset Samples")
plt.show()

# Visualize custom dataset (5 images)
plt.figure(figsize=(10,2))
for i in range(5):
    plt.subplot(1,5,i+1)
    img = augmented_images[i].permute(1, 2, 0)
    # Denormalize for display
    img = (img * 0.5) + 0.5
    plt.imshow(img.numpy().clip(0,1))
    plt.axis('off')
plt.suptitle("Custom Augmented Samples")
plt.show()
```

## Sample Figures


Figure 1: Five images from the filtered CIFAR10 subset.

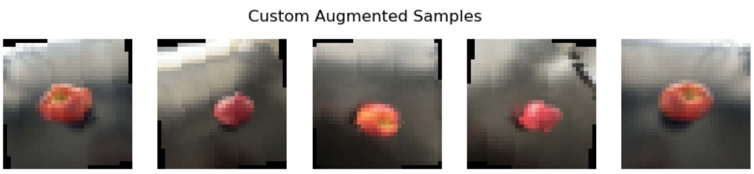


Figure 2: Five images from the custom dataset after augmentations.

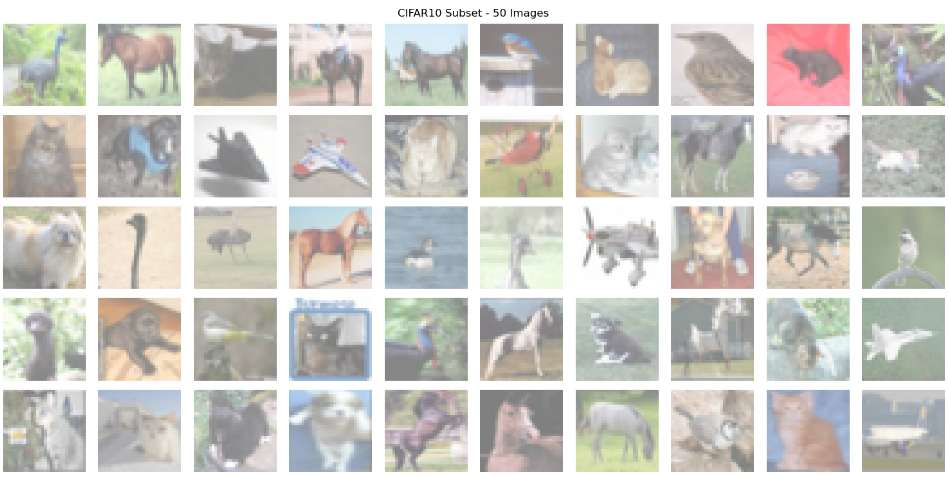


Figure 3: 50 images from the CIFAR10 dataset.

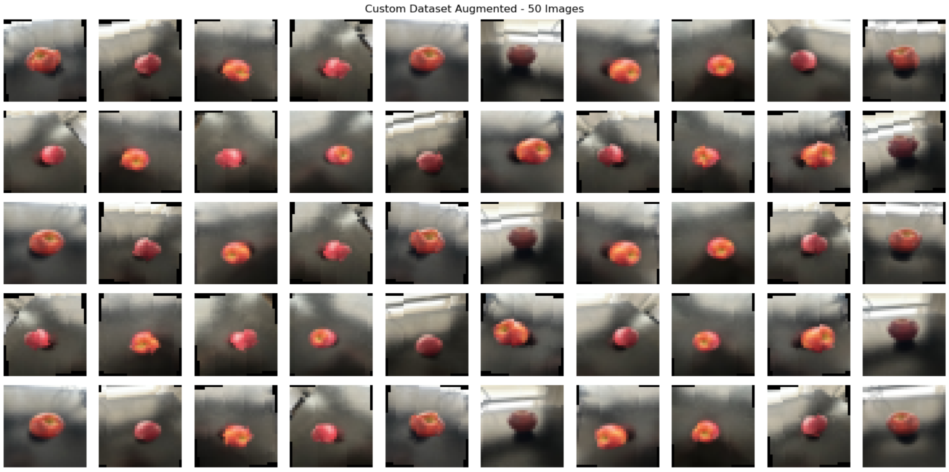


Figure 4: 50 images from the custom apple dataset.

## 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

```
combos = [(4, 1), (4, 2), (16, 1), (16, 2)]
results = []
for bs, nw in combos:
    t0 = time.time()
    count = 0
    for batch_imgs, _ in DataLoader(dataset_with_norm,
                                    batch_size=bs,
                                    shuffle=False,
                                    num_workers=nw):
        count += batch_imgs.size(0)
        if count >= 1000:
            break
    t1 = time.time()
    results.append((bs, nw, t1 - t0))

print("\n=== Performance Table ===")
print("BatchSize | NumWorkers | Time (s)")
for (bs, nw, t) in results:
    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.

```
loader_no_norm = DataLoader(dataset_no_norm, batch_size=4, shuffle=True,
    num_workers=2)
images_no_norm, _ = next(iter(loader_no_norm))

plt.figure(figsize=(8, 2))
for i in range(4):
    plt.subplot(1,4,i+1)
    img = images_no_norm[i].permute(1,2,0)
    plt.imshow(img.numpy())
    plt.axis('off')
plt.suptitle("One Batch - No Normalization")
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

```
# No seed set here
loader_no_seed = DataLoader(dataset_no_seed,
                            batch_size=2,
                            shuffle=True,
                            num_workers=0)

first_batch_no_seed, _ = next(iter(loader_no_seed))
# Plot the images from the first batch
...
# Then fetch again
second_batch_no_seed, _ = next(iter(loader_no_seed))
# 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

```
import random
import numpy as np

seed = 60146
torch.manual_seed(seed)
random.seed(seed)
np.random.seed(seed)

loader_with_seed = DataLoader(dataset_no_seed,
                              batch_size=2,
                              shuffle=True,
                              num_workers=0)

first_batch_with_seed, _ = next(iter(loader_with_seed))
# Plot
```

```
...

# Reset the seed again
torch.manual_seed(seed)
random.seed(seed)
np.random.seed(seed)

second_batch_with_seed, _ = next(iter(loader_with_seed))
# Plot
...
```

### 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.

```

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

###############################################################################
# 2.1 and 2.2: Pixel Value Scaling & Normalization
###############################################################################

# Transformation pipeline for CIFAR10
transform_cifar = transforms.Compose([
    transforms.ToTensor(),  # scales [0,255] to [0,1]
])

# Transformation pipeline for the custom dataset with random augmentations
transform_custom = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# New Transformation Pipeline for random seed demonstration (NO random flip/rotate)
transform_custom_no_flip_rotate = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

###############################################################################
# Custom Dataset Class
###############################################################################
class CustomAppleDataset(Dataset):
    """
    A simple Dataset that reads all images from a directory (apple_photos)
    and applies optional transformations.
    """
    def __init__(self, root, transform=None):
        self.root = root
        self.image_paths = [os.path.join(root, img) for img in os.listdir(root)
```

```
                            if img.lower().endswith(('.png', '.jpg', '.jpeg'))]
        self.transform = transform

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

    def __getitem__(self, index):
        img_path = self.image_paths[index]
        image = Image.open(img_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        return image, 0

###############################################################################
# 3.2 Comparing CIFAR10 with a Custom Dataset
###############################################################################
def compare_cifar10_with_custom():
    """
    1) Load CIFAR10 and filter 5 classes with 10 images each.
    2) Create a custom dataset from apple_photos with 20 images.
    3) Augment them to generate 30 additional images.
    4) Compare/visualize.
    """

    # Step 1: Load CIFAR10 and filter 5 classes with 10 images each => total 50
    selected_classes = [0, 2, 3, 5, 7]
    cifar10_full = datasets.CIFAR10(root='./data', train=True,
                                    download=True, transform=transform_cifar)

    # Indices of images belonging to the chosen 5 classes
    cifar_indices = [i for i, (_, label) in enumerate(cifar10_full)
                     if label in selected_classes]

    # Slice out the first 50 from those classes (10 images per class)
    subset_cifar10 = Subset(cifar10_full, cifar_indices[:50])

    # Step 2: Load the custom dataset of ~20 apple images
    custom_dataset = CustomAppleDataset(root='apple_photos',
                                        transform=transform_custom)

    # Step 3: Extrapolate (augment) to get 50 images.
    augmented_images = [custom_dataset[i % len(custom_dataset)][0]
                        for i in range(50)]

    # Step 4: Print sizes and visualize
    print(f"Size of CIFAR10 subset: {len(subset_cifar10)} images")
    print(f"Size of Custom Dataset (original): {len(custom_dataset)} images")

    # Visualization of a few samples from CIFAR10 Subset
    cifar_loader = DataLoader(subset_cifar10, batch_size=5, shuffle=False)
    batch = next(iter(cifar_loader))
    cifar_images, cifar_labels = batch

    plt.figure(figsize=(10, 2))
    for i in range(5):
        plt.subplot(1, 5, i+1)
        img = cifar_images[i].permute(1, 2, 0)
        plt.imshow(img.numpy())
        plt.axis('off')
    plt.suptitle("CIFAR10 Subset Samples")
    plt.savefig("cifar10_subset_samples.png")
    plt.close()

    # Visualization of some augmented custom images
    plt.figure(figsize=(10, 2))
    for i in range(5):
```

```
        plt.subplot(1, 5, i+1)
        img = augmented_images[i].permute(1, 2, 0)
        img = (img * 0.5) + 0.5
        img = torch.clamp(img, 0, 1)
        plt.imshow(img.numpy())
        plt.axis('off')
    plt.suptitle("Custom Augmented Samples")
    plt.savefig("custom_augmented_samples.png")
    plt.close()

    # Function to generate and save a grid of images
    def save_image_grid(images, title, filename, num_cols=10, num_rows=5):
        plt.figure(figsize=(num_cols * 1.5, num_rows * 1.5))
        for idx in range(num_cols * num_rows):
            if idx >= len(images):
                break
            plt.subplot(num_rows, num_cols, idx + 1)
            img = images[idx].permute(1, 2, 0)  # [C, H, W] -> [H, W, C]
            img = (img * 0.5) + 0.5
            img = torch.clamp(img, 0, 1)
            plt.imshow(img.numpy())
            plt.axis('off')
        plt.suptitle(title)
        plt.tight_layout()
        plt.subplots_adjust(top=0.95)
        plt.savefig(filename)
        plt.close()

    # Prepare images from CIFAR10 subset
    cifar_images_all = []
    cifar_loader_full = DataLoader(subset_cifar10, batch_size=50, shuffle=False)
    cifar_batch = next(iter(cifar_loader_full))
    cifar_images_all = cifar_batch[0]

    # Prepare images from Custom Dataset (augmented)
    custom_images_all = torch.stack(augmented_images)

    # Save CIFAR10 images grid
    save_image_grid(
        images=cifar_images_all,
        title="CIFAR10 Subset - 50 Images",
        filename="cifar10_50_images_grid.png",
        num_cols=10,
        num_rows=5
    )

    # Save Custom Dataset images grid
    save_image_grid(
        images=custom_images_all,
        title="Custom Dataset Augmented - 50 Images",
        filename="custom_dataset_50_images_grid.png",
        num_cols=10,
        num_rows=5
    )

###############################################################################
# 3.3 Using DataLoader for Parallel Processing
###############################################################################
def demo_dataloader_parallel():
    """
    1) Wrap custom dataset in a DataLoader, batch_size=4, num_workers>1.
    2) Plot all 4 images from a single batch.
    3) Compare performance when loading 1000 images manually vs. DataLoader.
    4) Compute max of each RGB channel before and after normalization.
    """

```

```
    no_norm_transform = transforms.Compose([
        transforms.Resize((32, 32)),
        transforms.ToTensor(),
    ])
    norm_transform = transforms.Compose([
        transforms.Resize((32, 32)),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])

    dataset_no_norm = CustomAppleDataset(root='apple_photos',
                                         transform=no_norm_transform)
    dataset_with_norm = CustomAppleDataset(root='apple_photos',
                                           transform=norm_transform)

    # 1) DataLoader with batch_size=4, num_workers=2
    loader_no_norm = DataLoader(dataset_no_norm, batch_size=4, shuffle=True,
                                num_workers=2)
    loader_with_norm = DataLoader(dataset_with_norm, batch_size=4, shuffle=True,
                                  num_workers=2)

    # 2) Plot all 4 images from a single batch
    images_no_norm, _ = next(iter(loader_no_norm))
    plt.figure(figsize=(8, 2))
    for i in range(4):
        plt.subplot(1, 4, i+1)
        img = images_no_norm[i].permute(1, 2, 0)
        plt.imshow(img.numpy())  # in [0,1]
        plt.axis('off')
    plt.suptitle("One Batch - No Normalization")
    plt.savefig("batch_no_normalization.png")
    plt.close()

    # 3) Compare performance loading 1000 images manually vs. DataLoader

    # (a) Manual getitem approach
    t0 = time.time()
    manual_images = []
    for i in range(1000):
        idx = i % len(dataset_with_norm)
        img, _ = dataset_with_norm[idx]
        manual_images.append(img)
    t1 = time.time()
    manual_time = t1 - t0

    # (b) DataLoader approach
    t2 = time.time()
    dataloader_images = []
    total_loaded = 0
    for batch_imgs, _ in DataLoader(dataset_with_norm, batch_size=10, shuffle=False,
                                    num_workers=2):
        for i in range(batch_imgs.size(0)):
            dataloader_images.append(batch_imgs[i])
            total_loaded += 1
            if total_loaded >= 1000:
                break
        if total_loaded >= 1000:
            break
    t3 = time.time()
    dataloader_time = t3 - t2

    print("Time taken (manual getitem, 1000 images): {:.4f}
        seconds".format(manual_time))
    print("Time taken (DataLoader w/ batch_size=10, num_workers=2, 1000 images):
        {:.4f} seconds".format(dataloader_time))
```

```
    # 4) Demonstrate different (batch_size, num_workers) combinations
    combos = [(4,1), (4,2), (16,1), (16,2)]  # for instance
    results = []
    for bs, nw in combos:
        t0 = time.time()
        count = 0
        for batch_imgs, _ in DataLoader(dataset_with_norm, batch_size=bs,
            shuffle=False,
                                        num_workers=nw):
            count += batch_imgs.size(0)
            if count >= 1000:
                break
        t1 = time.time()
        results.append((bs, nw, t1 - t0))

    print("\n=== Performance Table ===")
    print("BatchSize | NumWorkers | Time (s)")
    for (bs, nw, t) in results:
        print(f"{bs:9d} | {nw:10d} | {t:.4f}")

    # 5) Compute max of each channel for one batch "before and after" normalization
    images_before, _ = next(iter(loader_no_norm))
    images_after, _  = next(iter(loader_with_norm))

    # max over entire batch for each channel
    max_before_R = images_before[:, 0, :, :].max().item()
    max_before_G = images_before[:, 1, :, :].max().item()
    max_before_B = images_before[:, 2, :, :].max().item()

    max_after_R = images_after[:, 0, :, :].max().item()
    max_after_G = images_after[:, 1, :, :].max().item()
    max_after_B = images_after[:, 2, :, :].max().item()

    print("\nMax channel values BEFORE normalization: ",
          f"R={max_before_R:.3f}, G={max_before_G:.3f}, B={max_before_B:.3f}")
    print("Max channel values AFTER normalization:  ",
          f"R={max_after_R:.3f}, G={max_after_G:.3f}, B={max_after_B:.3f}")

##############################################################################
# 3.4 Exploring Random Seed and Reproducibility
##############################################################################
def demo_random_seed():
    """
    3.4 Exploring Random Seed and 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.
            Repeat the previous exercise and compare the results. Note if the
         same images appear in the first batch across iterations.
    """

    # Part 1: Without Setting a Random Seed
    print("\nPart 1: Without Setting a Random Seed")

    # Define the transformation without random augmentations
    dataset_no_seed = CustomAppleDataset(
        root='apple_photos',
        transform=transform_custom_no_flip_rotate
    )

    # Create DataLoader with batch_size=2 and shuffle=True
    loader_no_seed = DataLoader(dataset_no_seed, batch_size=2, shuffle=True,
        num_workers=0)
```

```
    # Get the first batch of images
    try:
        first_batch_no_seed, _ = next(iter(loader_no_seed))
    except StopIteration:
        print("Dataset is empty or not enough images.")
        return

    # Plot the first batch of images
    plt.figure(figsize=(4, 2))
    for i in range(first_batch_no_seed.size(0)):
        plt.subplot(1, 2, i+1)
        img = first_batch_no_seed[i].permute(1, 2, 0)
        img = (img * 0.5) + 0.5  # Unnormalize for display
        img = torch.clamp(img, 0, 1)
        plt.imshow(img.numpy())
        plt.axis('off')
    plt.suptitle("First Batch Without Seed")
    plt.savefig("first_batch_no_seed.png")
    plt.close()


    # Get the first batch of images
    try:
        first_batch_no_seed, _ = next(iter(loader_no_seed))
    except StopIteration:
        print("Dataset is empty or not enough images.")
        return

    # Plot the first batch of images
    plt.figure(figsize=(4, 2))
    for i in range(first_batch_no_seed.size(0)):
        plt.subplot(1, 2, i+1)
        img = first_batch_no_seed[i].permute(1, 2, 0)
        img = (img * 0.5) + 0.5  # Unnormalize for display
        img = torch.clamp(img, 0, 1)
        plt.imshow(img.numpy())
        plt.axis('off')
    plt.suptitle("Second Batch Without Seed")
    plt.savefig("second_batch_no_seed.png")
    plt.close()

    # Part 2: With Setting a Random Seed
    print("Part 2: With Setting a Random Seed")

    # Set the random seed to 60146
    seed = 60146
    torch.manual_seed(seed)
    random.seed(seed)
    np.random.seed(seed)

    # Create DataLoader with batch_size=2 and shuffle=True
    loader_with_seed = DataLoader(dataset_no_seed, batch_size=2, shuffle=True,
        num_workers=0)

    # Get the first batch of images
    try:
        first_batch_with_seed, _ = next(iter(loader_with_seed))
    except StopIteration:
        print("Dataset is empty or not enough images.")
        return

    # Plot the first batch of images
    plt.figure(figsize=(4, 2))
    for i in range(first_batch_with_seed.size(0)):
        plt.subplot(1, 2, i+1)
```

```python
            img = first_batch_with_seed[i].permute(1, 2, 0)
            img = (img * 0.5) + 0.5  # Unnormalize for display
            img = torch.clamp(img, 0, 1)
            plt.imshow(img.numpy())
            plt.axis('off')
        plt.suptitle("First Batch With Seed=60146")
        plt.savefig("first_batch_with_seed.png")
        plt.close()


        # Set the random seed to 60146
        seed = 60146
        torch.manual_seed(seed)
        random.seed(seed)
        np.random.seed(seed)

        # Create DataLoader with batch_size=2 and shuffle=True
        loader_with_seed = DataLoader(dataset_no_seed, batch_size=2, shuffle=True,
            num_workers=0)

        # Get the first batch of images
        try:
            first_batch_with_seed, _ = next(iter(loader_with_seed))
        except StopIteration:
            print("Dataset is empty or not enough images.")
            return

        # Plot the first batch of images
        plt.figure(figsize=(4, 2))
        for i in range(first_batch_with_seed.size(0)):
            plt.subplot(1, 2, i+1)
            img = first_batch_with_seed[i].permute(1, 2, 0)
            img = (img * 0.5) + 0.5  # Unnormalize for display
            img = torch.clamp(img, 0, 1)
            plt.imshow(img.numpy())
            plt.axis('off')
        plt.suptitle("Second Batch With Seed=60146")
        plt.savefig("second_batch_with_seed.png")
        plt.close()

###############################################################################
# Main Execution
###############################################################################
if __name__ == "__main__":
    # Compare CIFAR10 with Custom Apple Dataset
    compare_cifar10_with_custom()

    # Demo DataLoader parallel loading & performance
    demo_dataloader_parallel()

    # Random Seed & Reproducibility
    demo_random_seed()
```