Exercise 1

In this exercise, you will implement a damage classifier using our own Pytorch version of Resnet called SimpleResNet, which we discussed in lecture 5. Your model will classify damages into 5 classes using synthetic data. The classes are called: circle_damage, line_damage, star_damage, no_damage, multiple_damages.

1.1 1. Write code to generate a synthetic dataset with the 5 classes mentioned above

```
import os
import random
from PIL import Image, ImageDraw
import math
import matplotlib.pyplot as plt

# Set random seed for reproducibility
random.seed(42)

# Configuration
IMAGE_SIZE = (64, 64)  # Frame size is 64x64
NUM_IMAGES_PER_CLASS = 1000  # Number of images per class
OUTPUT_DIR = 'dataset_bw'  # Output directory for images

# Define classes
CLASSES = ['circle_damage', 'line_damage', 'star_damage', 'no_damage',
'multiple_damages']

# Configuration for number of shapes per damage type
SHAPES_PER_DAMAGE = {
    'circle_damage': (1, 3),  # 1 to 3 circles per image
    'line_damage': (1, 3),    # 1 to 3 lines per image
    'star_damage': (1, 3),    # 1 to 3 stars per image
}

# Create output directories
def create_directories(base_dir, classes):
    if not os.path.exists(base_dir):
        os.makedirs(base_dir)
    for cls in classes:
        class_dir = os.path.join(base_dir, cls)
        if not os.path.exists(class_dir):
            os.makedirs(class_dir)

# Draw filled circle damage
def draw_circle(draw):
    radius = random.randint(5, 15)  # Adjusted for 64x64 frame size
    x = random.randint(radius, IMAGE_SIZE[0] - radius)
    y = random.randint(radius, IMAGE_SIZE[1] - radius)
```

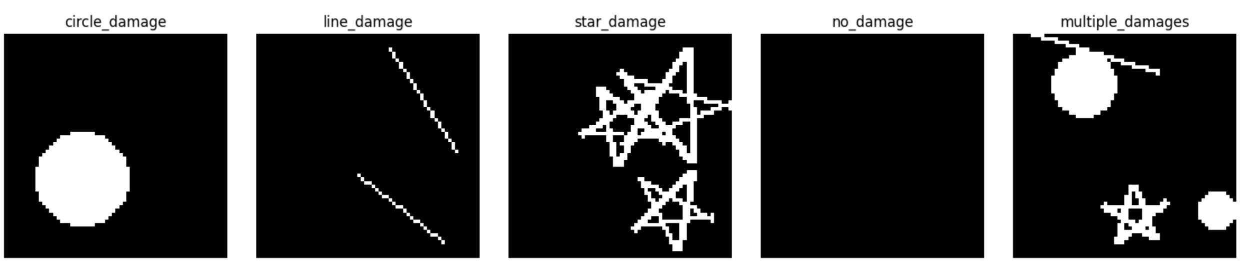
```
    bounding_box = [x - radius, y - radius, x + radius, y + radius]
    color = 'white'
    draw.ellipse(bounding_box, outline=color, fill=color, width=2)

# Draw line damage
def draw_line(draw):
    x1 = random.randint(0, IMAGE_SIZE[0])
    y1 = random.randint(0, IMAGE_SIZE[1])
    x2 = random.randint(0, IMAGE_SIZE[0])
    y2 = random.randint(0, IMAGE_SIZE[1])
    color = 'white'
    width = random.randint(1, 2)  # Adjusted line width
    draw.line([x1, y1, x2, y2], fill=color, width=width)

# Draw star damage
def draw_star(draw):
    num_points = 5
    outer_radius = random.randint(8, 20)  # Adjusted for 64x64 frame
size
    x_center = random.randint(outer_radius, IMAGE_SIZE[0] -
outer_radius)
    y_center = random.randint(outer_radius, IMAGE_SIZE[1] -
outer_radius)

    rotation_angle = random.uniform(0, 2 * math.pi)
    points = []
    angle_offset = rotation_angle - math.pi / 2
    for i in range(num_points):
        angle = angle_offset + (2 * math.pi * i) / num_points
        x = x_center + outer_radius * math.cos(angle)
        y = y_center + outer_radius * math.sin(angle)
        points.append((x, y))

    connect_order = [0, 2, 4, 1, 3, 0]
    color = 'white'
    width = 2
    for i in range(num_points):
        start_point = points[connect_order[i]]
        end_point = points[connect_order[i + 1]]
        draw.line([start_point, end_point], fill=color, width=width)

# Draw multiple damages
def draw_multiple(draw):
    damage_types = ['circle', 'line', 'star']
    num_damages = random.randint(2, 4)
    for _ in range(num_damages):
        damage = random.choice(damage_types)
        if damage == 'circle':
            draw_circle(draw)
        elif damage == 'line':
```

```python
            draw_line(draw)
        elif damage == 'star':
            draw_star(draw)

# Generate no_damage image
def generate_no_damage():
    return Image.new('RGB', IMAGE_SIZE, color='black')

# Generate damage image with random number of shapes
def generate_damage_image(damage_type):
    image = Image.new('RGB', IMAGE_SIZE, color='black')
    draw = ImageDraw.Draw(image)

    if damage_type in SHAPES_PER_DAMAGE:
        min_shapes, max_shapes = SHAPES_PER_DAMAGE[damage_type]
        num_shapes = random.randint(min_shapes, max_shapes)
        for _ in range(num_shapes):
            if damage_type == 'circle_damage':
                draw_circle(draw)
            elif damage_type == 'line_damage':
                draw_line(draw)
            elif damage_type == 'star_damage':
                draw_star(draw)
    elif damage_type == 'multiple_damages':
        draw_multiple(draw)

    return image

# Plot one sample image from each class
def plot_sample_images(base_dir, classes):
    num_classes = len(classes)
    plt.figure(figsize=(15, 3))
    for idx, cls in enumerate(classes):
        class_dir = os.path.join(base_dir, cls)
        # Get list of image files
        img_files = [f for f in os.listdir(class_dir) if
f.endswith('.png')]
        if not img_files:
            print(f"No images found in {class_dir}")
            continue
        # Select the first image
        img_path = os.path.join(class_dir, img_files[0])
        img = Image.open(img_path)
        plt.subplot(1, num_classes, idx + 1)
        plt.imshow(img, cmap='gray')
        plt.title(cls)
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

```python
# Save images to respective directories and collect first images for plotting
def save_images():
    create_directories(OUTPUT_DIR, CLASSES)
    first_images = {}
    for cls in CLASSES:
        print(f"Generating images for class: {cls}")
        for i in range(NUM_IMAGES_PER_CLASS):
            if cls == 'no_damage':
                img = generate_no_damage()
            else:
                img = generate_damage_image(cls)
            img_filename = f"image_{i+1:04d}.png"
            img_path = os.path.join(OUTPUT_DIR, cls, img_filename)
            img.save(img_path)
            # Save the first image for plotting
            if cls not in first_images:
                first_images[cls] = img.copy()
            if (i+1) % 100 == 0:
                print(f"  Saved {i+1} images")
    print("Dataset generation complete.")
    # Plot the sample images
    plot_sample_images(OUTPUT_DIR, CLASSES)

if __name__ == "__main__":
    save_images()
```

```
Generating images for class: circle_damage
  Saved 100 images
  Saved 200 images
  Saved 300 images
  Saved 400 images
  Saved 500 images
  Saved 600 images
  Saved 700 images
  Saved 800 images
  Saved 900 images
  Saved 1000 images
Generating images for class: line_damage
  Saved 100 images
  Saved 200 images
  Saved 300 images
  Saved 400 images
  Saved 500 images
  Saved 600 images
  Saved 700 images
  Saved 800 images
  Saved 900 images
  Saved 1000 images
Generating images for class: star_damage
```

```
  Saved 100 images
  Saved 200 images
  Saved 300 images
  Saved 400 images
  Saved 500 images
  Saved 600 images
  Saved 700 images
  Saved 800 images
  Saved 900 images
  Saved 1000 images
Generating images for class: no_damage
  Saved 100 images
  Saved 200 images
  Saved 300 images
  Saved 400 images
  Saved 500 images
  Saved 600 images
  Saved 700 images
  Saved 800 images
  Saved 900 images
  Saved 1000 images
Generating images for class: multiple_damages
  Saved 100 images
  Saved 200 images
  Saved 300 images
  Saved 400 images
  Saved 500 images
  Saved 600 images
  Saved 700 images
  Saved 800 images
  Saved 900 images
  Saved 1000 images
Dataset generation complete.
```



1.2 Adapt the python code in resnet-synth-ex2sol.py (from the solutions folder to exercises in lecture 5) to use your new synthetic image generator to classify these images with SimpleResNet model. Train it and measure its classification accuracy.

```
import os
import csv
import torch
```

```
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
from torchvision.datasets import ImageFolder
import torchvision.transforms as transforms

# --------------------------------
# Utility Functions
# --------------------------------
def ensure_dir(dir_path):
    if not os.path.exists(dir_path):
        os.makedirs(dir_path)

# --------------------------------
# SimpleResNet Architecture
# --------------------------------
class SimpleResNet(nn.Module):
    def __init__(self, num_classes=5):
        super(SimpleResNet, self).__init__()
        # Initial convolutional layer
        self.conv1 = nn.Conv2d(1, 64, kernel_size=3, stride=1,
padding=1)  # Input channels set to 1 for grayscale
        self.bn1 = nn.BatchNorm2d(64)

        # First residual block (no downsampling)
        self.conv2 = nn.Conv2d(64, 64, kernel_size=3, stride=1,
padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1,
padding=1)
        self.bn3 = nn.BatchNorm2d(64)

        # Second residual block (downsampling)
        self.conv4 = nn.Conv2d(64, 128, kernel_size=3, stride=2,
padding=1)
        self.bn4 = nn.BatchNorm2d(128)
        self.conv5 = nn.Conv2d(128, 128, kernel_size=3, stride=1,
padding=1)
        self.bn5 = nn.BatchNorm2d(128)
        self.conv1x1_1 = nn.Conv2d(64, 128, kernel_size=1, stride=2,
padding=0)

        # Third residual block (downsampling)
        self.conv6 = nn.Conv2d(128, 256, kernel_size=3, stride=2,
padding=1)
        self.bn6 = nn.BatchNorm2d(256)
        self.conv7 = nn.Conv2d(256, 256, kernel_size=3, stride=1,
padding=1)
        self.bn7 = nn.BatchNorm2d(256)
        self.conv1x1_2 = nn.Conv2d(128, 256, kernel_size=1, stride=2,
```

```
padding=0)

        # Fully connected layer
        # Assuming input images are 64x64, after two downsampling
steps: 16x16
        self.fc = nn.Linear(256 * 16 * 16, num_classes)

    def forward(self, x):
        # Initial convolution + batch norm + relu
        x = torch.relu(self.bn1(self.conv1(x)))

        # First residual block
        identity = x
        out = torch.relu(self.bn2(self.conv2(x)))
        out = self.bn3(self.conv3(out))
        x = torch.relu(out + identity)

        # Second residual block (downsampling)
        identity = self.conv1x1_1(x)
        out = torch.relu(self.bn4(self.conv4(x)))
        out = self.bn5(self.conv5(out))
        x = torch.relu(out + identity)

        # Third residual block (downsampling)
        identity = self.conv1x1_2(x)
        out = torch.relu(self.bn6(self.conv6(x)))
        out = self.bn7(self.conv7(out))
        x = torch.relu(out + identity)

        # Flatten and fully connected layer
        x = x.view(x.size(0), -1)
        # Debugging: Print the shape
        expected_features = 256 * 16 * 16
        if x.size(1) != expected_features:
            print(f"Warning: Expected input features to FC layer:
{expected_features}, but got {x.size(1)}")
        x = self.fc(x)
        return x

# ------------------------------
# Training Function
# ------------------------------
def train_model(model, train_loader, criterion, optimizer, device,
num_epochs=5, use_amp=False):
    model.train()
    if use_amp:
        from torch.amp import GradScaler, autocast
        scaler = GradScaler()
    else:
        scaler = None
```

```
    for epoch in range(num_epochs):
        running_loss = 0.0
        correct = 0
        total = 0

        for batch_idx, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            # Debugging: Check device assignment
            if batch_idx == 0:
                print(f"Epoch {epoch+1}, Batch {batch_idx}: Inputs on
{inputs.device}, Labels on {labels.device}")
                print(f"Model parameters on
{next(model.parameters()).device}")

            optimizer.zero_grad()

            if use_amp:
                with autocast(device_type='cuda'):
                    outputs = model(inputs)
                    loss = criterion(outputs, labels)
                scaler.scale(loss).backward()
                scaler.step(optimizer)
                scaler.update()
            else:
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

            running_loss += loss.item()

            # Calculate training accuracy for the batch
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        epoch_loss = running_loss / len(train_loader)
        epoch_accuracy = 100.0 * correct / total

        print(f'Epoch [{epoch+1}/{num_epochs}], Loss:
{epoch_loss:.4f}, Accuracy: {epoch_accuracy:.2f}%')

    return

# --------------------------------
# Testing Function
# --------------------------------
def test_model(model, test_loader, device):
```

```
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch_idx, (inputs, labels) in enumerate(test_loader):
            inputs, labels = inputs.to(device), labels.to(device)

            # Debugging: Check device assignment
            if batch_idx == 0:
                print(f"Test Batch {batch_idx}: Inputs on
{inputs.device}, Labels on {labels.device}")
                print(f"Model parameters on
{next(model.parameters()).device}")

            outputs = model(inputs)
            _, predicted = torch.max(outputs, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100.0 * correct / total
    print(f"Test Classification Accuracy: {accuracy:.2f}%")
    return accuracy

# -------------------------------
# Experiment Runner
# -------------------------------
def run_experiments(
    data_dir="dataset_bw",
    experiment_params=[
        {'batch_size':32, 'lr':0.01},
        {'batch_size':32, 'lr':0.005},
        {'batch_size':32, 'lr':0.001},
        {'batch_size':32, 'lr':0.0005},
        {'batch_size':16, 'lr':0.01},
        {'batch_size':16, 'lr':0.005},
        {'batch_size':16, 'lr':0.001},
        {'batch_size':16, 'lr':0.0005},
        {'batch_size':64, 'lr':0.01},
        {'batch_size':64, 'lr':0.005},
        {'batch_size':64, 'lr':0.001},
        {'batch_size':64, 'lr':0.0005},
    ],
    num_epochs=5,
    device=None,
    results_dir="results"
):
    """
    Runs multiple training experiments with different hyperparameters.

    Args:
        data_dir (str): Path to the dataset directory.
```

```
        experiment_params (list): List of dictionaries with
'batch_size' and 'lr'.
        num_epochs (int): Number of training epochs per experiment.
        device (torch.device): Device to run the experiments on.
        results_dir (str): Directory to save results.

    Returns:
        list: List of dictionaries containing experiment results.
    """
    if device is None:
        device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')

    ensure_dir(results_dir)

    # Define image transformations
    transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=1),  # Convert to
grayscale
        transforms.ToTensor(),                         # Convert PIL
images to tensors
        # Removed Resize and Normalize for speed
    ])

    # Load the entire dataset using ImageFolder
    full_dataset = ImageFolder(root=data_dir, transform=transform)

    # Get class names
    class_names = full_dataset.classes  # e.g., ['circle_damage',
'line_damage', 'star_damage', 'no_damage', 'multiple_damages']
    num_classes = len(class_names)
    print(f'Classes: {class_names}')

    # Split the dataset into training and testing subsets (80% train,
20% test)
    train_size = int(0.8 * len(full_dataset))
    test_size = len(full_dataset) - train_size
    train_dataset, test_dataset = random_split(full_dataset,
[train_size, test_size])

    results = []

    # Determine if AMP can be used
    use_amp = device.type == 'cuda'

    # Run each experiment
    for idx, params in enumerate(experiment_params):
        experiment_id = idx + 1
        batch_size = params['batch_size']
        lr = params['lr']
```

```
        print(f"\n=== Experiment {experiment_id} ===")
        print(f"Batch Size: {batch_size}, Learning Rate: {lr}")

        # Create DataLoaders
        train_loader = DataLoader(
            train_dataset,
            batch_size=batch_size,
            shuffle=True,
            num_workers=0,  # Set to 0 for Windows to avoid issues
            pin_memory=True if use_amp else False
        )
        test_loader = DataLoader(
            test_dataset,
            batch_size=batch_size,
            shuffle=False,
            num_workers=0,  # Set to 0 for Windows to avoid issues
            pin_memory=True if use_amp else False
        )

        # Initialize the model, criterion, and optimizer
        model = SimpleResNet(num_classes=num_classes).to(device)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=lr)

        # Train the model
        train_model(model, train_loader, criterion, optimizer, device,
num_epochs, use_amp)

        # Test the model
        test_accuracy = test_model(model, test_loader, device)

        # Save the results
        results.append({
            'experiment_id': experiment_id,
            'batch_size': batch_size,
            'lr': lr,
            'accuracy': test_accuracy
        })

    # Save results to CSV
    save_results_to_csv(results, filename=os.path.join(results_dir,
"experiment_results.csv"))

    # Print experiment summary
    print("\n===== Experiment Summary =====")
    for r in results:
        print(f"Exp {r['experiment_id']}: Batch
Size={r['batch_size']}, LR={r['lr']}, Accuracy={r['accuracy']:.2f}%")
```

```
    return results, class_names

# -------------------------------
# Save Results to CSV
# -------------------------------
def save_results_to_csv(results, filename="experiment_results.csv"):
    fieldnames = ['experiment_id', 'batch_size', 'lr', 'accuracy']
    with open(filename, mode='w', newline='') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for r in results:
            writer.writerow(r)
    print(f"\nResults saved to {filename}")

# -------------------------------
# Main Execution
# -------------------------------
def main():
    # Path to the dataset directory containing class subdirectories
    data_dir = 'dataset_bw'  # Update this path to your dataset
directory

    # Check if the dataset directory exists
    if not os.path.isdir(data_dir):
        print(f"Dataset directory '{data_dir}' not found. Please
ensure the path is correct.")
        return

    # Define experiment parameters (batch sizes and learning rates)
    experiment_params = [
        {'batch_size':32, 'lr':0.01},
        {'batch_size':32, 'lr':0.005},
        {'batch_size':32, 'lr':0.001},
        {'batch_size':32, 'lr':0.0005},
        {'batch_size':16, 'lr':0.01},
        {'batch_size':16, 'lr':0.005},
        {'batch_size':16, 'lr':0.001},
        {'batch_size':16, 'lr':0.0005},
        {'batch_size':64, 'lr':0.01},
        {'batch_size':64, 'lr':0.005},
        {'batch_size':64, 'lr':0.001},
        {'batch_size':64, 'lr':0.0005},
    ]

    # Run experiments
    results, class_names = run_experiments(
        data_dir=data_dir,
        experiment_params=experiment_params,
        num_epochs=5,
        device=torch.device('cuda') if torch.cuda.is_available() else
```

```
torch.device('cpu'),
        results_dir="results"
    )

if __name__ == "__main__":
    main()
```

```
Classes: ['circle_damage', 'line_damage', 'multiple_damages',
'no_damage', 'star_damage']

=== Experiment 1 ===
Batch Size: 32, Learning Rate: 0.01
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 6.8741, Accuracy: 66.15%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.5231, Accuracy: 84.88%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.3577, Accuracy: 89.72%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.2650, Accuracy: 92.12%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.2512, Accuracy: 91.85%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 83.40%

=== Experiment 2 ===
Batch Size: 32, Learning Rate: 0.005
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 3.2862, Accuracy: 69.65%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.4120, Accuracy: 86.10%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.2792, Accuracy: 90.60%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.2543, Accuracy: 90.72%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.1746, Accuracy: 93.88%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
```

```
Test Classification Accuracy: 89.10%

=== Experiment 3 ===
Batch Size: 32, Learning Rate: 0.001
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 1.0415, Accuracy: 75.92%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.3370, Accuracy: 88.92%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.2542, Accuracy: 90.85%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.1891, Accuracy: 93.60%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.2128, Accuracy: 92.38%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 89.90%

=== Experiment 4 ===
Batch Size: 32, Learning Rate: 0.0005
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 0.9092, Accuracy: 74.95%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.3173, Accuracy: 88.65%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.2843, Accuracy: 89.95%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.1984, Accuracy: 92.80%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.1744, Accuracy: 93.95%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 89.60%

=== Experiment 5 ===
Batch Size: 16, Learning Rate: 0.01
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 3.4858, Accuracy: 45.05%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
```

```
Model parameters on cuda:0
Epoch [2/5], Loss: 1.0237, Accuracy: 59.52%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.9051, Accuracy: 66.78%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.7833, Accuracy: 69.88%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.6720, Accuracy: 74.88%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 72.50%

=== Experiment 6 ===
Batch Size: 16, Learning Rate: 0.005
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 1.7488, Accuracy: 67.85%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.4411, Accuracy: 83.28%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.3463, Accuracy: 87.92%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.2863, Accuracy: 90.28%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.2996, Accuracy: 91.05%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 85.00%

=== Experiment 7 ===
Batch Size: 16, Learning Rate: 0.001
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 1.0652, Accuracy: 73.40%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.4052, Accuracy: 86.05%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.2722, Accuracy: 89.65%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
```

```
Epoch [4/5], Loss: 0.2732, Accuracy: 90.60%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.1732, Accuracy: 94.05%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 89.90%

=== Experiment 8 ===
Batch Size: 16, Learning Rate: 0.0005
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 0.7649, Accuracy: 76.88%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.3168, Accuracy: 89.10%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.2489, Accuracy: 92.08%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.1909, Accuracy: 92.90%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.1353, Accuracy: 95.03%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 90.20%

=== Experiment 9 ===
Batch Size: 64, Learning Rate: 0.01
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 9.2453, Accuracy: 46.50%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.8338, Accuracy: 74.10%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.6495, Accuracy: 81.42%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.5751, Accuracy: 81.90%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.4903, Accuracy: 84.20%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 53.10%
```

```
=== Experiment 10 ===
Batch Size: 64, Learning Rate: 0.005
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 5.3417, Accuracy: 57.58%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.4359, Accuracy: 84.12%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.3237, Accuracy: 88.25%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.2263, Accuracy: 92.62%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.1691, Accuracy: 94.03%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 90.40%

=== Experiment 11 ===
Batch Size: 64, Learning Rate: 0.001
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 1.3486, Accuracy: 68.42%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [2/5], Loss: 0.3837, Accuracy: 86.67%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.2760, Accuracy: 90.65%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.2177, Accuracy: 92.40%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.1825, Accuracy: 93.38%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 75.50%

=== Experiment 12 ===
Batch Size: 64, Learning Rate: 0.0005
Epoch 1, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [1/5], Loss: 1.0196, Accuracy: 72.28%
Epoch 2, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
```

```
Epoch [2/5], Loss: 0.3156, Accuracy: 89.38%
Epoch 3, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [3/5], Loss: 0.2596, Accuracy: 91.17%
Epoch 4, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [4/5], Loss: 0.1951, Accuracy: 93.67%
Epoch 5, Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Epoch [5/5], Loss: 0.1721, Accuracy: 94.17%
Test Batch 0: Inputs on cuda:0, Labels on cuda:0
Model parameters on cuda:0
Test Classification Accuracy: 90.80%

Results saved to results/experiment_results.csv

===== Experiment Summary =====
Exp 1: Batch Size=32, LR=0.01, Accuracy=83.40%
Exp 2: Batch Size=32, LR=0.005, Accuracy=89.10%
Exp 3: Batch Size=32, LR=0.001, Accuracy=89.90%
Exp 4: Batch Size=32, LR=0.0005, Accuracy=89.60%
Exp 5: Batch Size=16, LR=0.01, Accuracy=72.50%
Exp 6: Batch Size=16, LR=0.005, Accuracy=85.00%
Exp 7: Batch Size=16, LR=0.001, Accuracy=89.90%
Exp 8: Batch Size=16, LR=0.0005, Accuracy=90.20%
Exp 9: Batch Size=64, LR=0.01, Accuracy=53.10%
Exp 10: Batch Size=64, LR=0.005, Accuracy=90.40%
Exp 11: Batch Size=64, LR=0.001, Accuracy=75.50%
Exp 12: Batch Size=64, LR=0.0005, Accuracy=90.80%
```

1.3 Modify the architecture of SimpleResnet to improve its classification accuracy. Explain what you did, show the loss function after training the model, and compare your classification results with the original model.

Modifications: Learning Rate Scheduler, Weight Decay, Global Average Pooling, Dropout

```python
import os
import matplotlib.pyplot as plt
import csv
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader, random_split, Subset
from torchvision.datasets import ImageFolder
import torchvision.transforms as transforms
from collections import defaultdict

def create_directory(path):
```

```python
    if not os.path.exists(path):
        os.makedirs(path)

class OriginalResNet(nn.Module):
    """Original ResNet-like architecture without enhancements."""
    def __init__(self, num_classes=5):
        super(OriginalResNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, 3, 1, 1)
        self.bn1 = nn.BatchNorm2d(64)

        # Residual Block 1
        self.conv2 = nn.Conv2d(64, 64, 3, 1, 1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 64, 3, 1, 1)
        self.bn3 = nn.BatchNorm2d(64)

        # Residual Block 2 (Downsampling)
        self.conv4 = nn.Conv2d(64, 128, 3, 2, 1)
        self.bn4 = nn.BatchNorm2d(128)
        self.conv5 = nn.Conv2d(128, 128, 3, 1, 1)
        self.bn5 = nn.BatchNorm2d(128)
        self.downsample1 = nn.Conv2d(64, 128, 1, 2, 0)

        # Residual Block 3 (Downsampling)
        self.conv6 = nn.Conv2d(128, 256, 3, 2, 1)
        self.bn6 = nn.BatchNorm2d(256)
        self.conv7 = nn.Conv2d(256, 256, 3, 1, 1)
        self.bn7 = nn.BatchNorm2d(256)
        self.downsample2 = nn.Conv2d(128, 256, 1, 2, 0)

        self.fc = nn.Linear(256 * 16 * 16, num_classes)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))

        # Residual Block 1
        identity = x
        out = F.relu(self.bn2(self.conv2(x)))
        out = self.bn3(self.conv3(out))
        x = F.relu(out + identity)

        # Residual Block 2
        identity = self.downsample1(x)
        out = F.relu(self.bn4(self.conv4(x)))
        out = self.bn5(self.conv5(out))
        x = F.relu(out + identity)

        # Residual Block 3
        identity = self.downsample2(x)
        out = F.relu(self.bn6(self.conv6(x)))
```

```python
        out = self.bn7(self.conv7(out))
        x = F.relu(out + identity)

        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x

class EnhancedResNet(nn.Module):
    """Improved ResNet-like architecture with Dropout and Global
Average Pooling."""
    def __init__(self, num_classes=5):
        super(EnhancedResNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, 3, 1, 1)
        self.bn1 = nn.BatchNorm2d(64)

        # Residual Block 1
        self.conv2 = nn.Conv2d(64, 64, 3, 1, 1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 64, 3, 1, 1)
        self.bn3 = nn.BatchNorm2d(64)

        # Residual Block 2 (Downsampling)
        self.conv4 = nn.Conv2d(64, 128, 3, 2, 1)
        self.bn4 = nn.BatchNorm2d(128)
        self.conv5 = nn.Conv2d(128, 128, 3, 1, 1)
        self.bn5 = nn.BatchNorm2d(128)
        self.downsample1 = nn.Conv2d(64, 128, 1, 2, 0)

        # Residual Block 3 (Downsampling)
        self.conv6 = nn.Conv2d(128, 256, 3, 2, 1)
        self.bn6 = nn.BatchNorm2d(256)
        self.conv7 = nn.Conv2d(256, 256, 3, 1, 1)
        self.bn7 = nn.BatchNorm2d(256)
        self.downsample2 = nn.Conv2d(128, 256, 1, 2, 0)

        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(256, num_classes)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))

        # Residual Block 1
        identity = x
        out = F.relu(self.bn2(self.conv2(x)))
        out = self.bn3(self.conv3(out))
        x = F.relu(out + identity)

        # Residual Block 2
        identity = self.downsample1(x)
```

```python
        out = F.relu(self.bn4(self.conv4(x)))
        out = self.bn5(self.conv5(out))
        x = F.relu(out + identity)

        # Residual Block 3
        identity = self.downsample2(x)
        out = F.relu(self.bn6(self.conv6(x)))
        out = self.bn7(self.conv7(out))
        x = F.relu(out + identity)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.dropout(x)
        x = self.fc(x)
        return x

def train(model, loader, criterion, optimizer, device, epochs=5,
scheduler=None):
    model.train()
    losses = []
    for epoch in range(epochs):
        total_loss = 0.0
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        if scheduler:
            scheduler.step()
        avg_loss = total_loss / len(loader)
        losses.append(avg_loss)
        print(f"Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}")
    return losses

def evaluate(model, loader, device):
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (preds == labels).sum().item()
    accuracy = 100 * correct / total
    print(f"Accuracy: {accuracy:.2f}%")
    return accuracy
```

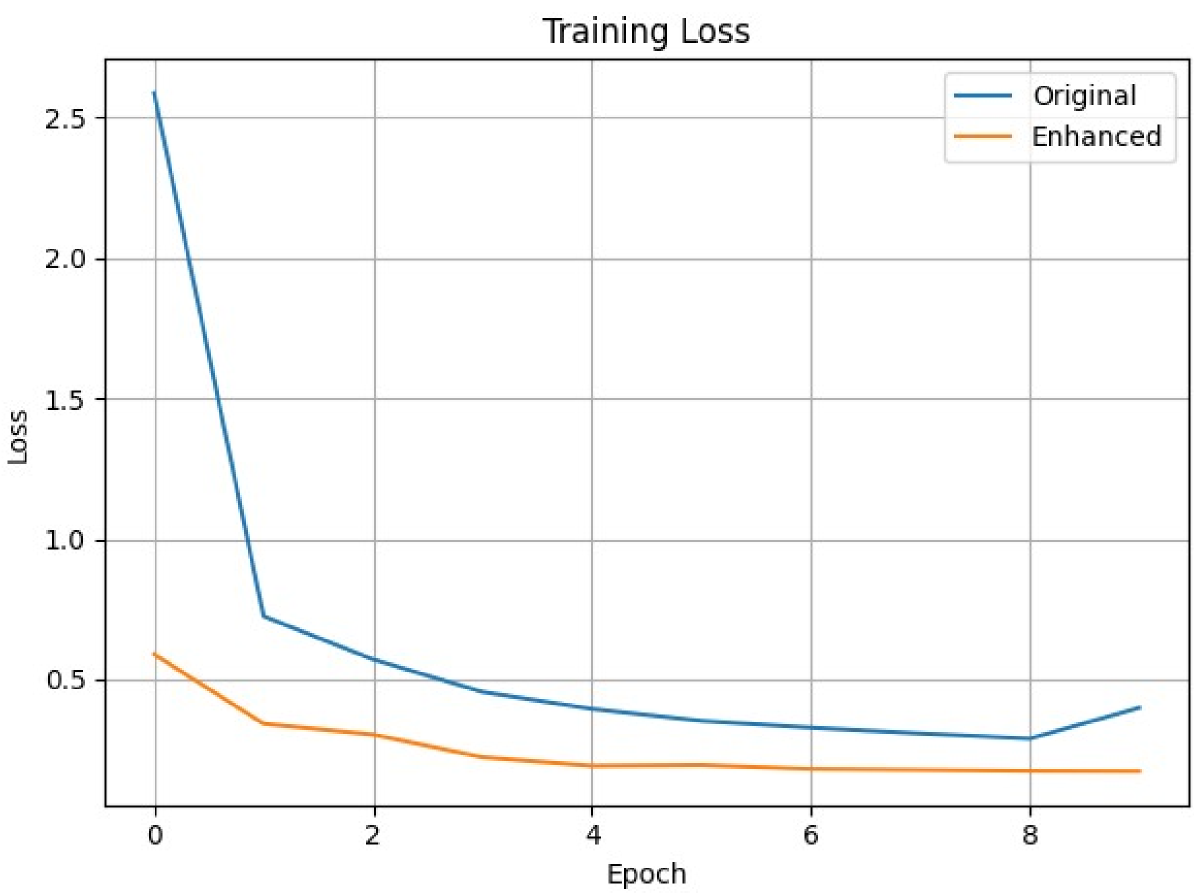
```
def execute_experiments(data_dir, params_list, epochs, device,
save_dir):
    create_directory(save_dir)
    transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=1),
        transforms.Resize((64, 64)),
        transforms.ToTensor()
    ])
    dataset = ImageFolder(root=data_dir, transform=transform)
    num_classes = len(dataset.classes)
    train_size = int(0.8 * len(dataset))
    test_size = len(dataset) - train_size
    train_set, test_set = random_split(dataset, [train_size,
test_size])
    results = []
    for idx, params in enumerate(params_list):
        print(f"\n--- Experiment {idx+1}: {params['model_name']} ---")
        batch = params['batch_size']
        lr = params['lr']
        wd = params['weight_decay']
        sched = params['use_scheduler']
        train_loader = DataLoader(train_set, batch_size=batch,
shuffle=True)
        test_loader = DataLoader(test_set, batch_size=batch,
shuffle=False)
        model = params['model_class'](num_classes).to(device)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=lr,
weight_decay=wd)
        scheduler_step = None
        if sched:
            scheduler_step = optim.lr_scheduler.StepLR(optimizer,
step_size=3, gamma=0.1)
        loss_curve = train(model, train_loader, criterion, optimizer,
device, epochs, scheduler_step)
        acc = evaluate(model, test_loader, device)
        results.append({
            'id': idx+1,
            'name': params['model_name'],
            'batch_size': batch,
            'lr': lr,
            'weight_decay': wd,
            'scheduler': sched,
            'final_loss': loss_curve[-1],
            'accuracy': acc,
            'loss_curve': loss_curve
        })
    save_results(results, os.path.join(save_dir, "results.csv"))
```

```
    plot_losses(results, save_dir)
    plot_accuracies(results, save_dir)
    summarize(results)

def save_results(results, filepath):
    fields = ['id', 'name', 'batch_size', 'lr', 'weight_decay',
'scheduler', 'final_loss', 'accuracy']
    with open(filepath, 'w', newline='') as file:
        writer = csv.DictWriter(file, fieldnames=fields)
        writer.writeheader()
        for res in results:
            writer.writerow({key: res[key] for key in fields})
    print(f"Results saved to {filepath}")

def plot_losses(results, save_dir):
    plt.figure()
    for res in results:
        plt.plot(res['loss_curve'], label=res['name'])
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    path = os.path.join(save_dir, "losses.png")
    plt.show()
    plt.savefig(path, dpi=300)
    plt.close()
    print(f"Loss plot saved to {path}")

def plot_accuracies(results, save_dir):
    names = [res['name'] for res in results]
    acc = [res['accuracy'] for res in results]
    plt.figure()
    plt.bar(names, acc, color=['blue', 'green'])
    plt.xlabel('Model')
    plt.ylabel('Accuracy (%)')
    plt.title('Model Accuracy Comparison')
    for i, v in enumerate(acc):
        plt.text(i, v + 1, f"{v:.2f}%", ha='center')
    plt.ylim(0, 100)
    plt.tight_layout()
    path = os.path.join(save_dir, "accuracies.png")
    plt.show()
    plt.savefig(path, dpi=300)
    plt.close()
    print(f"Accuracy plot saved to {path}")

def summarize(results):
    print("\n=== Summary ===")
```

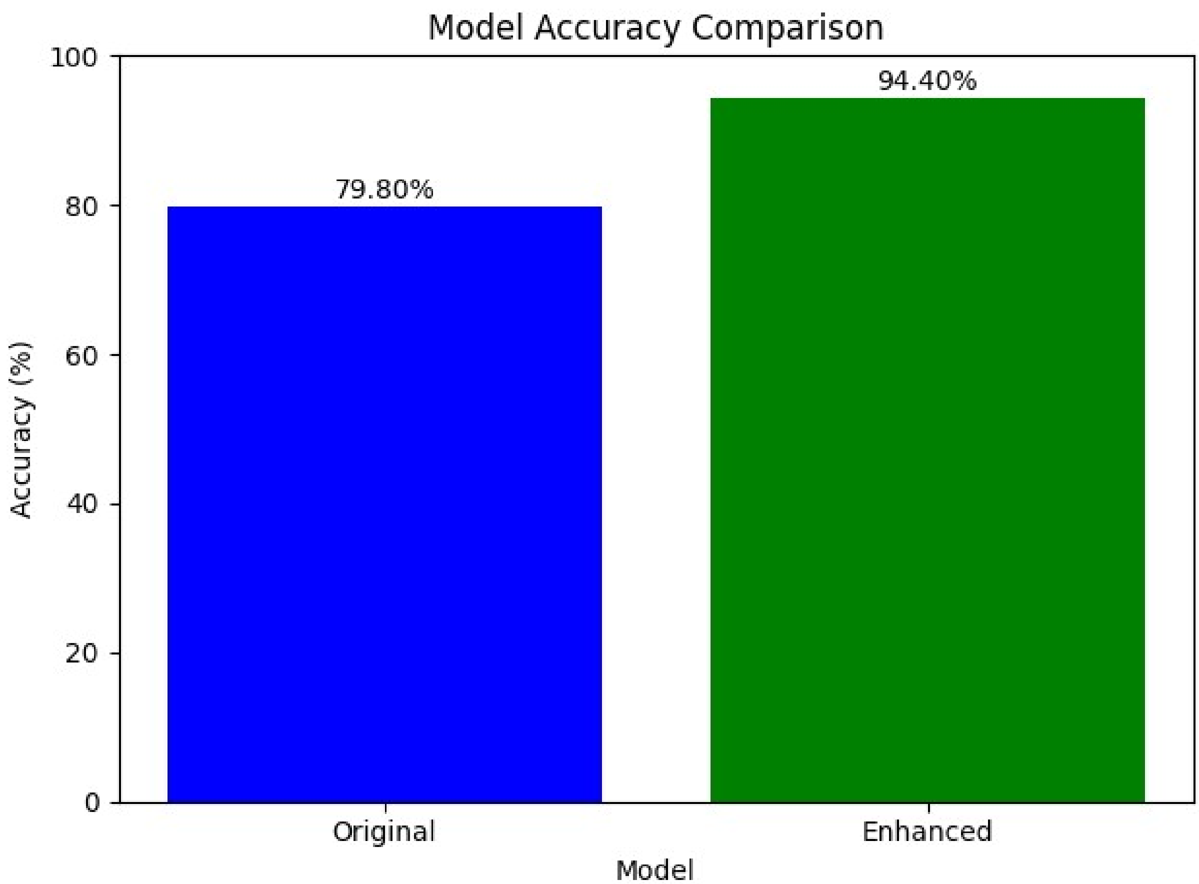
```
    for res in results:
        print(f"Model: {res['name']}, Batch: {res['batch_size']}, LR:
{res['lr']}, "
              f"WD: {res['weight_decay']}, Scheduler:
{res['scheduler']}, "
              f"Final Loss: {res['final_loss']:.4f}, Accuracy:
{res['accuracy']:.2f}%")

def main():
    data_directory = 'dataset_bw'  # Update this path as needed
    if not os.path.isdir(data_directory):
        print(f"Data directory '{data_directory}' not found.")
        return
    experiments = [
        {
            'model_name': 'Original',
            'model_class': OriginalResNet,
            'batch_size': 32,
            'lr': 0.01,
            'weight_decay': 0.0,
            'use_scheduler': False
        },
        {
            'model_name': 'Enhanced',
            'model_class': EnhancedResNet,
            'batch_size': 32,
            'lr': 0.01,
            'weight_decay': 1e-4,
            'use_scheduler': True
        }
    ]
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    results_directory = "results_plots"
    execute_experiments(data_directory, experiments, epochs=10,
device=device, save_dir=results_directory)

if __name__ == "__main__":
    main()
```

```
--- Experiment 1: Original ---
Epoch 1/10, Loss: 2.5873
Epoch 2/10, Loss: 0.7246
Epoch 3/10, Loss: 0.5708
Epoch 4/10, Loss: 0.4550
Epoch 5/10, Loss: 0.3945
```

```
Epoch 6/10, Loss: 0.3521
Epoch 7/10, Loss: 0.3279
Epoch 8/10, Loss: 0.3072
Epoch 9/10, Loss: 0.2888
Epoch 10/10, Loss: 0.3986
Accuracy: 79.80%

--- Experiment 2: Enhanced ---
Epoch 1/10, Loss: 0.5890
Epoch 2/10, Loss: 0.3419
Epoch 3/10, Loss: 0.3032
Epoch 4/10, Loss: 0.2224
Epoch 5/10, Loss: 0.1922
Epoch 6/10, Loss: 0.1950
Epoch 7/10, Loss: 0.1807
Epoch 8/10, Loss: 0.1775
Epoch 9/10, Loss: 0.1739
Epoch 10/10, Loss: 0.1728
Accuracy: 94.40%
Results saved to results_plots/results.csv
```



```
Loss plot saved to results_plots/losses.png
```




```
Accuracy plot saved to results_plots/accuracies.png

=== Summary ===
Model: Original, Batch: 32, LR: 0.01, WD: 0.0, Scheduler: False, Final
Loss: 0.3986, Accuracy: 79.80%
Model: Enhanced, Batch: 32, LR: 0.01, WD: 0.0001, Scheduler: True,
Final Loss: 0.1728, Accuracy: 94.40%
```