Report

a. My code

```
from resnet20_cifar import resnet20
import os
import sys
import matplotlib.pyplot as plt
import numpy as np
import random
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader
from PIL import Image
from thop import profile
import time


# Set random seed for reproducibility
seed_value = 30
random.seed(seed_value)
torch.manual_seed(seed_value)
torch.cuda.manual_seed(seed_value)


# Check if CUDA is available and set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


CIFAR10_CLASSES = [
    'Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
    'Dog', 'Frog', 'Horse', 'Ship', 'Truck'
```

```
]


class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.bn1 = nn.BatchNorm2d(32)
        self.bn2 = nn.BatchNorm2d(64)
        self.bn3 = nn.BatchNorm2d(128)
        self.conv1 = nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=0)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(128 * 3 * 3, 256)
        self.fc2 = nn.Linear(256, 10)

    def forward(self, x):
        x = nn.functional.relu(self.bn1(self.conv1(x)))
        x = nn.MaxPool2d(2)(x)
        x = nn.functional.relu(self.bn2(self.conv2(x)))
        x = nn.MaxPool2d(2)(x)
        x = nn.functional.relu(self.bn3(self.conv3(x)))
        x = nn.MaxPool2d(2)(x)
        x = x.view(x.size(0), -1)
        x = nn.functional.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Define transformations
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),  # Updated
normalization
```

```
])

# Load CIFAR-10 dataset
def load_data(batch_size=64):
    train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
    test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
    return train_loader, test_loader


# Training function
def train_model(model, num_epochs=10, batch_size=64):
    train_loader, test_loader = load_data(batch_size)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # To store accuracy for all epochs
    train_accuracies = []
    test_accuracies = []

    # Training loop
    for epoch in range(num_epochs):
        model.train()
        running_train_loss = 0.0
        correct_train_predictions = 0
        total_train_samples = 0
```

```
# Training
for images, labels in train_loader:
    images, labels = images.to(device), labels.to(device)

    # Forward pass
    outputs = model(images)
    loss = criterion(outputs, labels)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    running_train_loss += loss.item()
    _, predicted = torch.max(outputs.data, 1)
    total_train_samples += labels.size(0)
    correct_train_predictions += (predicted == labels).sum().item()

train_accuracy = correct_train_predictions / total_train_samples
avg_train_loss = running_train_loss / len(train_loader)
train_accuracies.append(train_accuracy)

# Testing
model.eval()
running_test_loss = 0.0
correct_test_predictions = 0
total_test_samples = 0

with torch.no_grad():
    for images, labels in test_loader:
```

```
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)

            running_test_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total_test_samples += labels.size(0)
            correct_test_predictions += (predicted == labels).sum().item()

        test_accuracy = correct_test_predictions / total_test_samples
        avg_test_loss = running_test_loss / len(test_loader)
        test_accuracies.append(test_accuracy)

        print(f'Epoch [{epoch + 1}/{num_epochs}], Train Loss: {avg_train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, Test Loss: {avg_test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}')


    # Calculate and print mean and std of train and test accuracies
    train_mean = np.mean(train_accuracies)
    train_std = np.std(train_accuracies)
    test_mean = np.mean(test_accuracies)
    test_std = np.std(test_accuracies)

    print(f'\nTrain Accuracy: Mean = {train_mean:.4f}, Std = {train_std:.4f}')
    print(f'Test Accuracy: Mean = {test_mean:.4f}, Std = {test_std:.4f}')

    # Calculate MACs and parameters
    dummy_input = torch.randn(1, 3, 32, 32).to(device)  # Create a dummy input tensor
    macs, params = profile(model, inputs=(dummy_input,))  # Pass the dummy input as a single-element tuple
    print(f"SimpleCNN: MACs = {macs / 1e6:.2f} M, Parameters = {params / 1e6:.2f} M")
```

```
    # Save the trained model after training
    os.makedirs('./model', exist_ok=True)
    torch.save(model.state_dict(), './model/cnn_model.pth')
    print('Model saved to ./model/cnn_model.pth')

# Testing function for custom images
def test_image(model, image_path):
    model.eval()

    # Load the image and transform it
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0).to(device)

    # Make prediction
    outputs = model(image)
    _, predicted = torch.max(outputs.data, 1)
    class_idx = predicted.item()

    print(f"Predicted class index: {class_idx}, Class name: {CIFAR10_CLASSES[class_idx]}")

    # Visualize the first conv layer
    first_conv_output = model.conv1(image).detach().cpu()
    visualize_conv_layer(first_conv_output)

# Visualize convolutional layer
def visualize_conv_layer(conv_output, save_path='CONV_rslt.png'):
    conv_output = conv_output.squeeze(0)
    fig, axes = plt.subplots(4, 8, figsize=(10, 5))
    for i, ax in enumerate(axes.flat):
        if i < conv_output.size(0):
```

```
            ax.imshow(conv_output[i].numpy(), cmap='gray')
        ax.axis('off')
    plt.savefig(save_path)
    print(f"Convolutional layer visualization saved to {save_path}")

# Test accuracy of pre-trained ResNet-20
def test_resnet20():
    # Define device
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    # Load the pretrained ResNet-20 model
    model = resnet20()
    model.load_state_dict(torch.load("resnet20_cifar10_pretrained.pt", map_location=device))
    model = model.to(device)
    model.eval()

    # Prepare the CIFAR-10 test dataset
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010))  # Updated
normalization for ResNet-20
    ])

    testset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
    testloader = DataLoader(testset, batch_size=100, shuffle=False)

    # Evaluate the model
    correct = 0
    total = 0

    with torch.no_grad():
```

```
        for images, labels in testloader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f'Accuracy of the model on the CIFAR-10 test dataset: {accuracy:.2f}%')
    # Count parameters and MACs for ResNet-20
    dummy_input = torch.randn(1, 3, 32, 32).to(device)  # Create a dummy input tensor
    macs, params = profile(model, inputs=(dummy_input,))  # Pass the dummy input as a single-element tuple
    print(f"ResNet-20: MACs = {macs / 1e6:.2f} M, Parameters = {params / 1e6:.2f} M")
# Inference speed test function

def inference_speed_test(model, num_iterations=1000):
    model.eval()

    image_path = './dog_small.png'  # Replace with your image path
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0).to(device)  # Transform and add batch dimension
    input_tensor = image

    # Warm-up iterations
    for _ in range(10):
        with torch.no_grad():
            _ = model(input_tensor)

    # Measure inference time
    start_time = time.time()
```

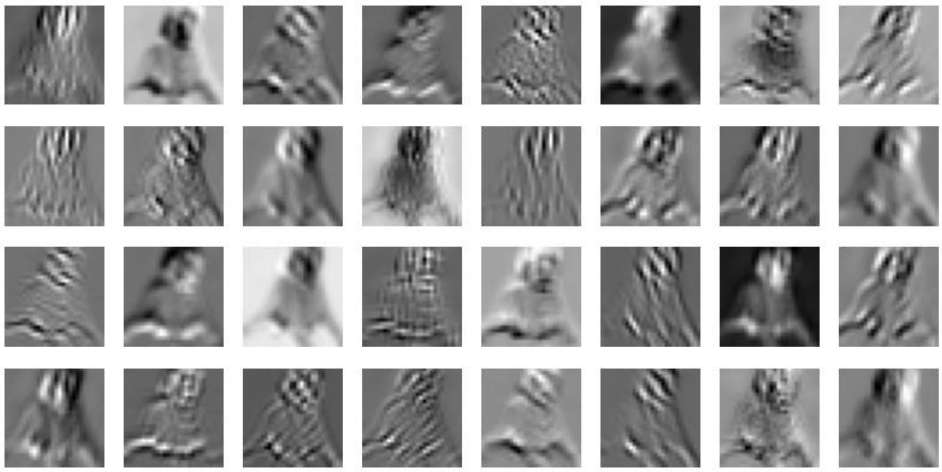
```
    with torch.no_grad():
        for _ in range(num_iterations):
            _ = model(input_tensor)

    end_time = time.time()

    # Calculate average inference time
    total_time = end_time - start_time
    average_time = total_time / num_iterations

    print(f"Average inference time for input: {average_time:.6f} seconds")


# Main function to handle different commands
if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage:")
        print("python CNNclassify.py train")
        print("python CNNclassify.py test <image_path>")
        print("python CNNclassify.py resnet20")
        sys.exit(1)

    command = sys.argv[1]

    if command == 'train':
        model = SimpleCNN().to(device)
        train_model(model)

    elif command == 'test':
        if len(sys.argv) < 3:
```

```
            print("Please provide the path to the image to test.")
            sys.exit(1)
        image_path = sys.argv[2]
        model = SimpleCNN().to(device)
        model.load_state_dict(torch.load('./model/cnn_model.pth'),strict=False)
        test_image(model, image_path)

    elif command == 'resnet20':

        test_resnet20()
    elif command == 'speed_test':
        if len(sys.argv) < 3:
            print("Please provide the model type (simple or resnet) and input type (dummy or real).")
            sys.exit(1)
        model_type = sys.argv[2]

        if model_type == 'simple':
            model = SimpleCNN().to(device)
            model.load_state_dict(torch.load('./model/cnn_model.pth'), strict=False)
            inference_speed_test(model)
        elif model_type == 'resnet':
            model = resnet20()
            model.load_state_dict(torch.load("resnet20_cifar10_pretrained.pt",
map_location=device))
            model = model.to(device)
            inference_speed_test(model)
    else:
        print("Invalid command.")
```

b. two screenshots of training and testing results of your neural network design.

```
G VISHAL@DESKTOP-20BA5T1 MINGW64 ~/OneDrive/Desktop/temp/CNN
$ python CNNclassify.py train
Files already downloaded and verified
Files already downloaded and verified
Epoch [1/15], Train Loss: 1.2160, Train Accuracy: 0.5621, Test Loss: 1.0082, Test Accuracy: 0.6439
Epoch [2/15], Train Loss: 0.8596, Train Accuracy: 0.6969, Test Loss: 0.8414, Test Accuracy: 0.7090
Epoch [3/15], Train Loss: 0.7009, Train Accuracy: 0.7561, Test Loss: 0.7531, Test Accuracy: 0.7386
Epoch [4/15], Train Loss: 0.6032, Train Accuracy: 0.7872, Test Loss: 0.7146, Test Accuracy: 0.7555
Epoch [5/15], Train Loss: 0.5106, Train Accuracy: 0.8207, Test Loss: 0.7451, Test Accuracy: 0.7470
Epoch [6/15], Train Loss: 0.4398, Train Accuracy: 0.8454, Test Loss: 0.7011, Test Accuracy: 0.7660
Epoch [7/15], Train Loss: 0.3767, Train Accuracy: 0.8685, Test Loss: 0.7127, Test Accuracy: 0.7727
Epoch [8/15], Train Loss: 0.3172, Train Accuracy: 0.8878, Test Loss: 0.7160, Test Accuracy: 0.7752
Epoch [9/15], Train Loss: 0.2610, Train Accuracy: 0.9081, Test Loss: 0.7974, Test Accuracy: 0.7651
Epoch [10/15], Train Loss: 0.2205, Train Accuracy: 0.9218, Test Loss: 0.8345, Test Accuracy: 0.7722
Epoch [11/15], Train Loss: 0.1801, Train Accuracy: 0.9376, Test Loss: 0.8621, Test Accuracy: 0.7743
Epoch [12/15], Train Loss: 0.1636, Train Accuracy: 0.9424, Test Loss: 0.9968, Test Accuracy: 0.7530
Epoch [13/15], Train Loss: 0.1342, Train Accuracy: 0.9530, Test Loss: 0.9794, Test Accuracy: 0.7643
Epoch [14/15], Train Loss: 0.1221, Train Accuracy: 0.9568, Test Loss: 1.0348, Test Accuracy: 0.7569
Epoch [15/15], Train Loss: 0.1011, Train Accuracy: 0.9645, Test Loss: 1.0851, Test Accuracy: 0.7657

Train Accuracy: Mean = 0.8539, Std = 0.1100
Test Accuracy: Mean = 0.7506, Std = 0.0330
Model saved to ./model/cnn_model.pth
```

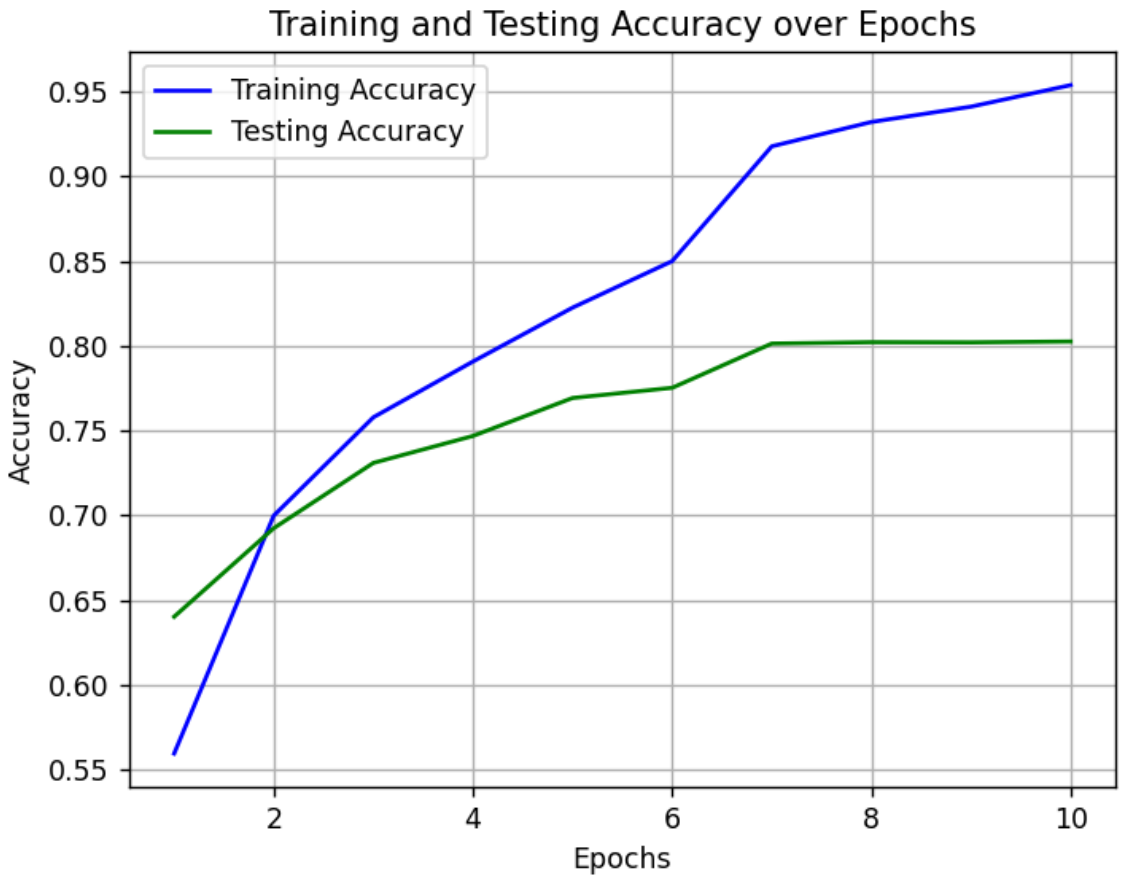
c. one screenshot of the visualization results from the first CONV layer



d. a screenshot to show the testing accuracy of ResNet-20 model on entire CIFAR10 testing dataset.

```
G VISHAL@DESKTOP-20BA5T1 MINGW64 ~/OneDrive/Desktop/temp/CNN
$ python CNNclassify.py resnet20
C:\Users\G VISHAL\OneDrive\Desktop\temp\CNN\CNNclassify.py:187: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defa
ult value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code duri
ng unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default valu
e for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer
be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you
 start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issu
es related to this experimental feature.
  model.load_state_dict(torch.load("resnet20_cifar10_pretrained.pt", map_location=device))
Files already downloaded and verified
Accuracy of the model on the CIFAR-10 test dataset: 92.06%
```

e. a screenshot of the plot shows the curves of the training accuracy and testing accuracy



f. a screenshot of the computation costs (MACs) and the number of parameters of your Neural Network and ResNet-20 counted by using THOP package.

```
G VISHAL@DESKTOP-20BA5T1 MINGW64 ~/OneDrive/Desktop/temp/CNN
$ python CNNclassify.py train
Files already downloaded and verified
Files already downloaded and verified
Epoch [1/10], Train Loss: 1.2160, Train Accuracy: 0.5621, Test Loss: 1.0082, Test Accuracy: 0.6439
Epoch [2/10], Train Loss: 0.8596, Train Accuracy: 0.6969, Test Loss: 0.8414, Test Accuracy: 0.7090
Epoch [3/10], Train Loss: 0.7009, Train Accuracy: 0.7561, Test Loss: 0.7531, Test Accuracy: 0.7386
Epoch [4/10], Train Loss: 0.6032, Train Accuracy: 0.7872, Test Loss: 0.7146, Test Accuracy: 0.7555
Epoch [5/10], Train Loss: 0.5106, Train Accuracy: 0.8207, Test Loss: 0.7451, Test Accuracy: 0.7470
Epoch [6/10], Train Loss: 0.4398, Train Accuracy: 0.8454, Test Loss: 0.7011, Test Accuracy: 0.7660
Epoch [7/10], Train Loss: 0.3767, Train Accuracy: 0.8685, Test Loss: 0.7127, Test Accuracy: 0.7727
Epoch [8/10], Train Loss: 0.3172, Train Accuracy: 0.8878, Test Loss: 0.7160, Test Accuracy: 0.7752
Epoch [9/10], Train Loss: 0.2610, Train Accuracy: 0.9081, Test Loss: 0.7974, Test Accuracy: 0.7651
Epoch [10/10], Train Loss: 0.2205, Train Accuracy: 0.9218, Test Loss: 0.8345, Test Accuracy: 0.7722

Train Accuracy: Mean = 0.8055, Std = 0.1051
Test Accuracy: Mean = 0.7445, Std = 0.0386
[INFO] Register count_normalization() for <class 'torch.nn.modules.batchnorm.BatchNorm2d'>.
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
SimpleCNN: MACs = 9.58 M, Parameters = 0.39 M
Model saved to ./model/cnn_model.pth
```

```
G VISHAL@DESKTOP-20BA5T1 MINGW64 ~/OneDrive/Desktop/temp/CNN
$ python CNNclassify.py resnet20
C:\Users\G VISHAL\OneDrive\Desktop\temp\CNN\CNNclassify.py:195: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  model.load_state_dict(torch.load("resnet20_cifar10_pretrained.pt", map_location=device))
Files already downloaded and verified
Accuracy of the model on the CIFAR-10 test dataset: 92.06%
[INFO] Register count_convNd() for <class 'torch.nn.modules.conv.Conv2d'>.
[INFO] Register count_normalization() for <class 'torch.nn.modules.batchnorm.BatchNorm2d'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.activation.ReLU'>.
[INFO] Register zero_ops() for <class 'torch.nn.modules.container.Sequential'>.
[INFO] Register count_avgpool() for <class 'torch.nn.modules.pooling.AvgPool2d'>.
[INFO] Register count_linear() for <class 'torch.nn.modules.linear.Linear'>.
ResNet-20: MACs = 41.62 M, Parameters = 0.27 M
```

g. Show the 3 random seeds you used for training and the corresponding final accuracies obtained. Show the mean and std deviation of the 3 accuracies.

Seed=10

```
Files already downloaded and verified
Files already downloaded and verified
Epoch [1/10], Loss: 1.2000, Train Accuracy: 0.5663 Test Loss: 0.9792, Test Accuracy: 0.6596
Epoch [2/10], Loss: 0.8299, Train Accuracy: 0.7065 Test Loss: 0.7872, Test Accuracy: 0.7263
Epoch [3/10], Loss: 0.6835, Train Accuracy: 0.7620 Test Loss: 0.7181, Test Accuracy: 0.7523
Epoch [4/10], Loss: 0.5817, Train Accuracy: 0.7948 Test Loss: 0.7028, Test Accuracy: 0.7630
Epoch [5/10], Loss: 0.4949, Train Accuracy: 0.8276 Test Loss: 0.6882, Test Accuracy: 0.7645
Epoch [6/10], Loss: 0.4224, Train Accuracy: 0.8509 Test Loss: 0.7197, Test Accuracy: 0.7627
Epoch [7/10], Loss: 0.2464, Train Accuracy: 0.9191 Test Loss: 0.6198, Test Accuracy: 0.7972
Epoch [8/10], Loss: 0.2087, Train Accuracy: 0.9337 Test Loss: 0.6317, Test Accuracy: 0.7992
Epoch [9/10], Loss: 0.1876, Train Accuracy: 0.9413 Test Loss: 0.6475, Test Accuracy: 0.7983
Epoch [10/10], Loss: 0.1603, Train Accuracy: 0.9528 Test Loss: 0.6367, Test Accuracy: 0.8003

Train Accuracy: Mean = 0.8255, Std = 0.1171
Test Accuracy: Mean = 0.7623, Std = 0.0416
Predicted: Cat, Actual: Cat
Predicted: Automobile, Actual: Ship
Predicted: Ship, Actual: Ship
Predicted: Airplane, Actual: Airplane
Predicted: Deer, Actual: Frog
```

Seed=20

```
Files already downloaded and verified
Files already downloaded and verified
Epoch [1/10], Loss: 1.2202, Train Accuracy: 0.5606 Test Loss: 1.0010, Test Accuracy: 0.6391
Epoch [2/10], Loss: 0.8470, Train Accuracy: 0.7024 Test Loss: 0.8530, Test Accuracy: 0.7033
Epoch [3/10], Loss: 0.6982, Train Accuracy: 0.7555 Test Loss: 0.7764, Test Accuracy: 0.7351
Epoch [4/10], Loss: 0.5948, Train Accuracy: 0.7917 Test Loss: 0.7309, Test Accuracy: 0.7513
Epoch [5/10], Loss: 0.5068, Train Accuracy: 0.8232 Test Loss: 0.6671, Test Accuracy: 0.7703
Epoch [6/10], Loss: 0.4341, Train Accuracy: 0.8476 Test Loss: 0.7176, Test Accuracy: 0.7666
Epoch [7/10], Loss: 0.2634, Train Accuracy: 0.9141 Test Loss: 0.6284, Test Accuracy: 0.7976
Epoch [8/10], Loss: 0.2240, Train Accuracy: 0.9288 Test Loss: 0.6320, Test Accuracy: 0.7981
Epoch [9/10], Loss: 0.2008, Train Accuracy: 0.9365 Test Loss: 0.6466, Test Accuracy: 0.7964
Epoch [10/10], Loss: 0.1739, Train Accuracy: 0.9492 Test Loss: 0.6400, Test Accuracy: 0.8010

Train Accuracy: Mean = 0.8210, Std = 0.1174
Test Accuracy: Mean = 0.7559, Std = 0.0494
Predicted: Cat, Actual: Cat
Predicted: Ship, Actual: Ship
Predicted: Ship, Actual: Ship
Predicted: Airplane, Actual: Airplane
Predicted: Frog, Actual: Frog
```

Seed=30

```
Files already downloaded and verified
Files already downloaded and verified
Epoch [1/10], Loss: 1.2140, Train Accuracy: 0.5596 Test Loss: 1.0311, Test Accuracy: 0.6403
Epoch [2/10], Loss: 0.8500, Train Accuracy: 0.6999 Test Loss: 0.8792, Test Accuracy: 0.6925
Epoch [3/10], Loss: 0.6917, Train Accuracy: 0.7579 Test Loss: 0.7746, Test Accuracy: 0.7310
Epoch [4/10], Loss: 0.5946, Train Accuracy: 0.7908 Test Loss: 0.7322, Test Accuracy: 0.7469
Epoch [5/10], Loss: 0.5020, Train Accuracy: 0.8226 Test Loss: 0.6845, Test Accuracy: 0.7693
Epoch [6/10], Loss: 0.4280, Train Accuracy: 0.8500 Test Loss: 0.6973, Test Accuracy: 0.7753
Epoch [7/10], Loss: 0.2510, Train Accuracy: 0.9177 Test Loss: 0.6302, Test Accuracy: 0.8014
Epoch [8/10], Loss: 0.2109, Train Accuracy: 0.9321 Test Loss: 0.6420, Test Accuracy: 0.8022
Epoch [9/10], Loss: 0.1876, Train Accuracy: 0.9411 Test Loss: 0.6464, Test Accuracy: 0.8021
Epoch [10/10], Loss: 0.1609, Train Accuracy: 0.9538 Test Loss: 0.6512, Test Accuracy: 0.8026

Train Accuracy: Mean = 0.8226, Std = 0.1194
Test Accuracy: Mean = 0.7564, Std = 0.0521
Predicted: Cat, Actual: Cat
Predicted: Ship, Actual: Ship
Predicted: Ship, Actual: Ship
Predicted: Airplane, Actual: Airplane
Predicted: Frog, Actual: Frog
```

h. Show the inference time of your own model and ResNet-20 model, respectively.

```
G VISHAL@DESKTOP-20BA5T1 MINGW64 ~/OneDrive/Desktop/temp/CNN
$ python CNNclassify.py speed_test simple
C:\Users\G VISHAL\OneDrive\Desktop\temp\CNN\CNNclassify.py:294: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defa
ult value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code duri
ng unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default valu
e for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer
be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you
 start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issu
es related to this experimental feature.
  model.load_state_dict(torch.load('./model/cnn_model.pth'), strict=False)
Average inference time for input: 0.000594 seconds

G VISHAL@DESKTOP-20BA5T1 MINGW64 ~/OneDrive/Desktop/temp/CNN
$ python CNNclassify.py speed_test resnet
C:\Users\G VISHAL\OneDrive\Desktop\temp\CNN\CNNclassify.py:298: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defa
ult value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code duri
ng unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default valu
e for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer
be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you
 start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issu
es related to this experimental feature.
  model.load_state_dict(torch.load("resnet20_cifar10_pretrained.pt", map_location=device))
Average inference time for input: 0.002046 seconds
```