ECS659U Coursework

Arjun 210136027

Part 1:

The code starts by importing all the necessary libraries (PyTorch, torchvision, matplotlib, etc.) and checks for GPU availability, ensuring that training uses CUDA when available for faster execution. I have defined two sets of transformations: one for training data that includes random cropping, horizontal flipping, conversion to tensors, and normalization, and one for test data that applies only tensor conversion and normalization. These transformations augment the training images and standardize the data for better learning. After that, the dataset is loaded and split into training and test sets, and Data Loaders are created—configured with a batch size of 128 and shuffling enabled for the training set to maintain randomness across epochs.

Part 2:

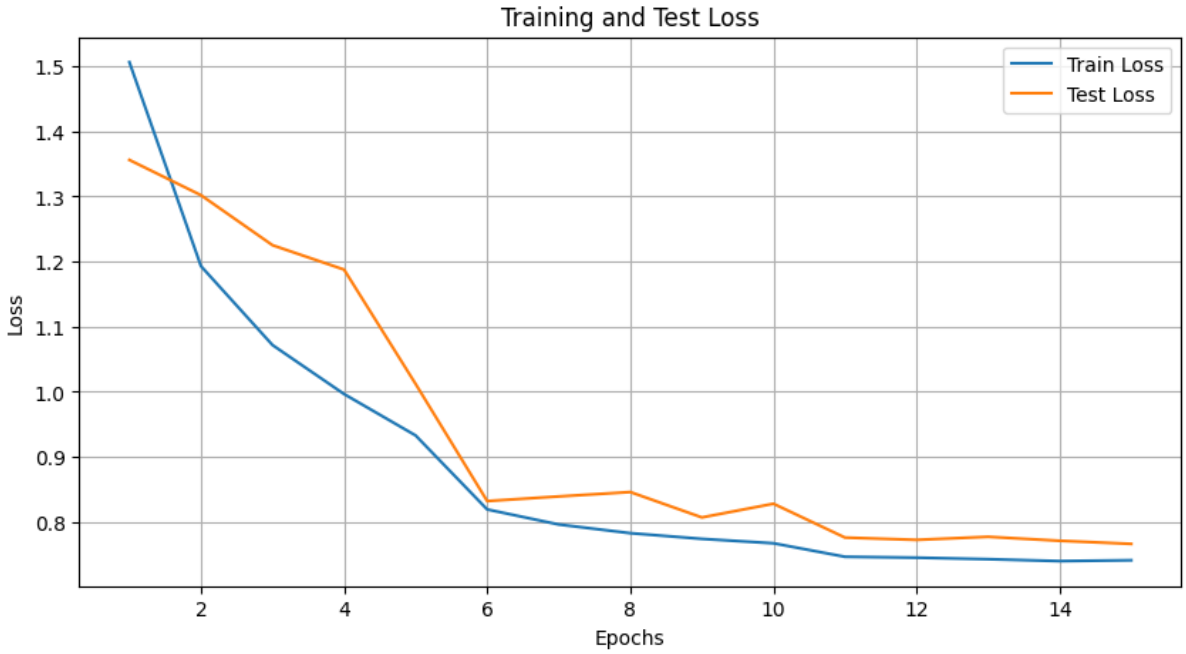
This part starts off with the given custom CNN architecture. I have defined the Expert Block class which has two branches: an expert branch that computes a weight vector using global average pooling and fully connected layers, and a convolution branch that processes the input with multiple parallel convolutional layers (with batch normalization and ReLU). The Backbone class stacks several Expert Blocks sequentially, and the Classifier class then applies adaptive average pooling followed by a fully connected layer to output the final class scores. Together, these modules form the Custom CNN model which begins with a stem (a basic convolution layer) and then passes data through the backbone and classifier in a single forward pass.

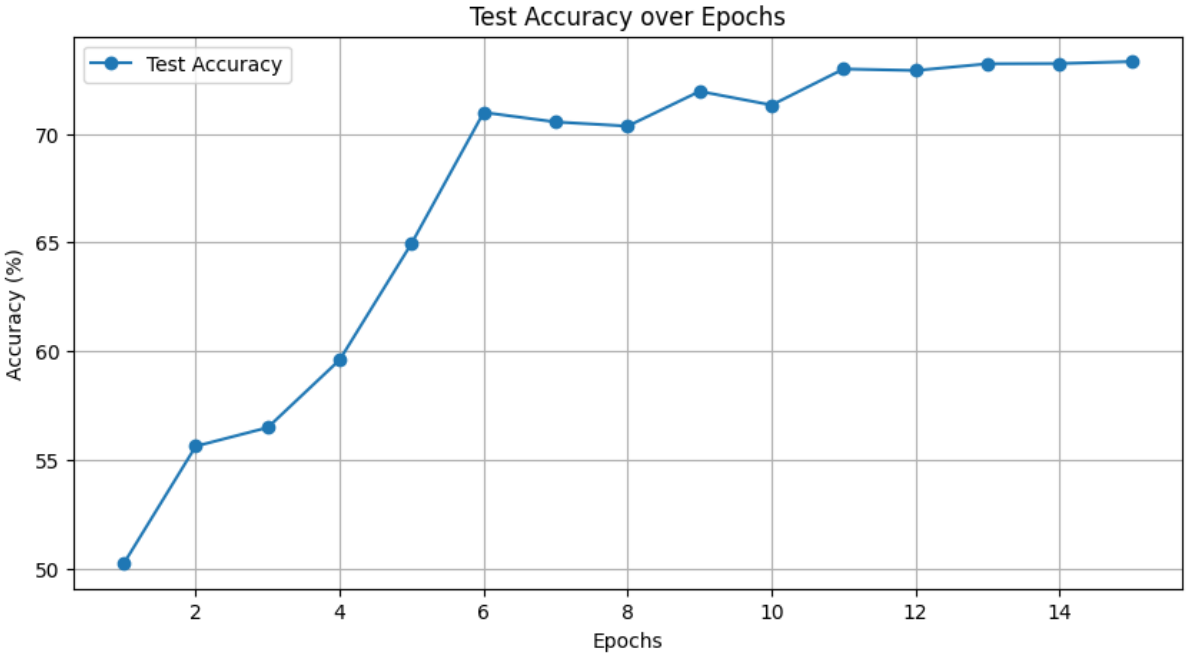
Part 3:

In this section I initialize an instance of the Custom CNN model with defined parameters (e.g., 3 blocks, K=2, dropout rate of 0.2) and moved to the selected device. The loss function chosen is CrossEntropyLoss, and the optimizer used is Adam with a learning rate of 0.001. Additionally, a learning rate scheduler (StepLR) is employed to adjust the learning rate every 5 epochs by a gamma factor of 0.1, helping to fine-tune the training process and improve performance.

Part 4:

The training loop runs for 15 epochs. For each epoch, the model is set to training mode using model.train(), and it iterates over batches provided by the training Data Loader. During each iteration, the inputs and labels are transferred to the device, gradients are zeroed out with optimizer.zero_grad(), and a forward pass is computed to obtain the outputs. The loss is then calculated using CrossEntropyLoss, and backpropagation is performed with loss.backward(), after which the optimizer updates the network parameters using optimizer.step(). I have also recorded the running loss, and after processing all batches, the average training loss and test accuracy. The learning rate scheduler is updated at the end of each epoch to adjust the learning rate as specified in its schedule.





Part 5:

Training Details:

Files already downloaded and verified
Files already downloaded and verified
Epoch 1/15

Train Loss: 1.5065 | Test Loss: 1.3562 | Test Accuracy: 50.22%
Epoch 2/15

Train Loss: 1.1930 | Test Loss: 1.3021 | Test Accuracy: 55.62%
Epoch 3/15

Train Loss: 1.0717 | Test Loss: 1.2252 | Test Accuracy: 56.48%
Epoch 4/15

Train Loss: 0.9965 | Test Loss: 1.1877 | Test Accuracy: 59.60%
Epoch 5/15

Train Loss: 0.9328 | Test Loss: 1.0119 | Test Accuracy: 64.97%
Epoch 6/15

Train Loss: 0.8192 | Test Loss: 0.8320 | Test Accuracy: 70.99%
Epoch 7/15

Train Loss: 0.7960 | Test Loss: 0.8391 | Test Accuracy: 70.55%
Epoch 8/15

Train Loss: 0.7827 | Test Loss: 0.8459 | Test Accuracy: 70.36%
Epoch 9/15

Train Loss: 0.7739 | Test Loss: 0.8069 | Test Accuracy: 71.96%
Epoch 10/15

Train Loss: 0.7672 | Test Loss: 0.8279 | Test Accuracy: 71.33%
Epoch 11/15

Train Loss: 0.7464 | Test Loss: 0.7759 | Test Accuracy: 72.99%
Epoch 12/15

Train Loss: 0.7451 | Test Loss: 0.7724 | Test Accuracy: 72.92%
Epoch 13/15

Train Loss: 0.7427 | Test Loss: 0.7771 | Test Accuracy: 73.23%
Epoch 14/15

Train Loss: 0.7398 | Test Loss: 0.7710 | Test Accuracy: 73.24%
Epoch 15/15

Train Loss: 0.7411 | Test Loss: 0.7662 | Test Accuracy: 73.33%
Model saved as cifar10_customcnn.pth