Neural Networks and Deep Learning Coursework

Mohammed Fahmidur Rahman

Goal:

- Implement a specific model to classify images from the CIFAR-10 dataset.
- Train the model to reduce the loss and achieve the highest accuracy possible.

Task 1 - Preparing the dataset:

```
# Data augmentation and normalisation
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

```
# Loading datasets and creating dataloaders
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)
testloader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False, num_workers=2)
```

I first prepared the dataset by applying transformations onto the images.

- transforms.ToTensor() is used to convert images to Pytorch tensors
- transforms.Normalize() is then used to normalise the tensors.

Then downloaded and prepared the CIFAR-10 training and test datasets and loaded them into data loaders with a batch size of 128.

Task 2 - Creating the Model:

We define a custom model called CIFAR10Model structured as required as STEM to Backbone to Classifier. The architecture was implemented using Pytorch and matches the block structure as required in the assignment specification.
The STEM is a single convolutional layer followed by batch normalisation and a ReLU activation function.
Backbone is composed of three sequential blocks, each block includes the expert branch and "K" convolutional layers.
A classifier then applies global average pooling to convert the feature map into a vector.

Task 3 - Defining the Loss Function and Optimiser:

```
# Initialisation
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = CIFAR10Model().to(device)

# Defining Loss function and Optimiser
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0008, weight_decay=5e-4)

# Learning rate scheduler - halves learning rate every 30 epochs
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100, eta_min=1e-6)
```

Definition the loss function using CrossEntropyLoss(). The optimiser with RMSprop() which has a learning rate of 0.0015. A learning rate scheduler is also utilised which is used to adjust the learning rate based on the training progress.

Task 4 - Training Script:

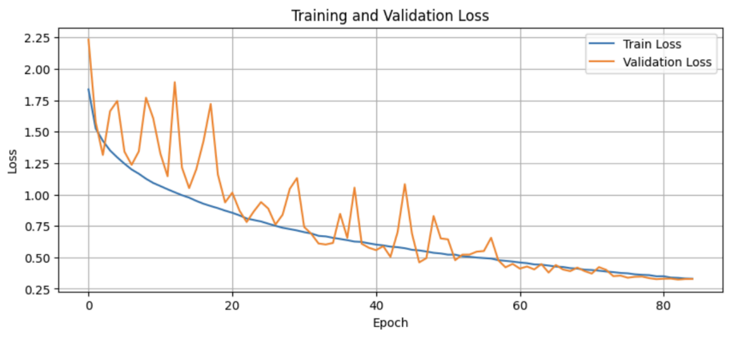
I used two key functions, train() and evaluate(), to perform the model training and validate processes effectively.

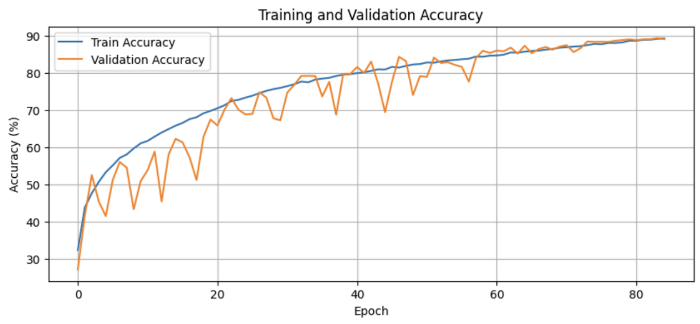
The train function handles the training logic including forward passes, loss calculation, and parameter updates. It begins by setting the model to training mode and initialising counters for the epoch's loss, correct predictions, and total samples. For each batch of images and labels, the data is moved to either the CPU or GPU, and the model makes a prediction (a forward pass). After that, it calculates how far off the predictions are using the loss function and then updates model parameters respectively via backpropagation and an optimiser step. The function keeps a running total of the training loss and how many predictions were correct. When all the batches are processed, the learning rate is adjusted using a scheduler and the model is evaluated on the test data using the evaluate function to assess how well it is generalising.

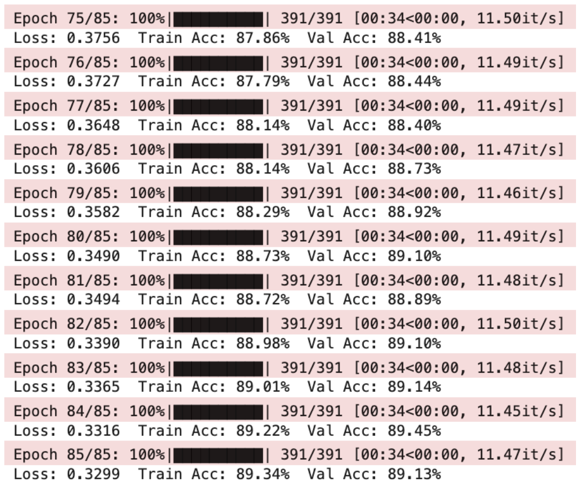
The evaluate function is used to assess the model's accuracy on the test dataset without modifying any weights. It puts the model into an evaluation mode to give a clear picture as to how well the model is performing, it disables dropout and gradient calculations to ensure results are consistent and quick. The function goes through the test data in batches, makes predictions and compares them to the actual labels to count how many it gets correct and once completed, it works out the overall accuracy and returns it.

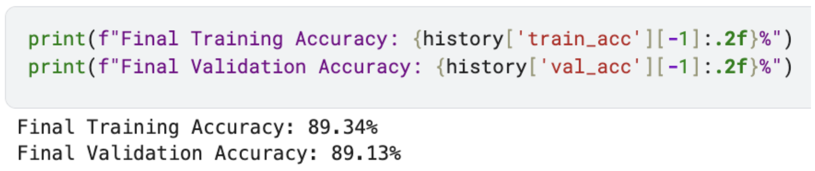
Hyperparameters used:

- epochs = 85 - utilised 85 epochs when running my training.
- lr = 0.0008 - used a learning rate of 0.0008 in my scheduler.
- batch_size = 128 and batch_size = 100 - batch sizes for trainloader and testloader respectively.
- num_blocks = 6 - number of backbone blocks I utilised.
- k = 4 - number of convolutional paths per block.






```
Epoch 75/85: 100%|██████████| 391/391 [00:34<00:00, 11.50it/s]
Loss: 0.3756  Train Acc: 87.86%  Val Acc: 88.41%
Epoch 76/85: 100%|██████████| 391/391 [00:34<00:00, 11.49it/s]
Loss: 0.3727  Train Acc: 87.79%  Val Acc: 88.44%
Epoch 77/85: 100%|██████████| 391/391 [00:34<00:00, 11.49it/s]
Loss: 0.3648  Train Acc: 88.14%  Val Acc: 88.40%
Epoch 78/85: 100%|██████████| 391/391 [00:34<00:00, 11.47it/s]
Loss: 0.3606  Train Acc: 88.14%  Val Acc: 88.73%
Epoch 79/85: 100%|██████████| 391/391 [00:34<00:00, 11.46it/s]
Loss: 0.3582  Train Acc: 88.29%  Val Acc: 88.92%
Epoch 80/85: 100%|██████████| 391/391 [00:34<00:00, 11.49it/s]
Loss: 0.3490  Train Acc: 88.73%  Val Acc: 89.10%
Epoch 81/85: 100%|██████████| 391/391 [00:34<00:00, 11.48it/s]
Loss: 0.3494  Train Acc: 88.72%  Val Acc: 88.89%
Epoch 82/85: 100%|██████████| 391/391 [00:34<00:00, 11.50it/s]
Loss: 0.3390  Train Acc: 88.98%  Val Acc: 89.10%
Epoch 83/85: 100%|██████████| 391/391 [00:34<00:00, 11.48it/s]
Loss: 0.3365  Train Acc: 89.01%  Val Acc: 89.14%
Epoch 84/85: 100%|██████████| 391/391 [00:34<00:00, 11.45it/s]
Loss: 0.3316  Train Acc: 89.22%  Val Acc: 89.45%
Epoch 85/85: 100%|██████████| 391/391 [00:34<00:00, 11.47it/s]
Loss: 0.3299  Train Acc: 89.34%  Val Acc: 89.13%
```


```python
print(f"Final Training Accuracy: {history['train_acc'][-1]:.2f}%")
print(f"Final Validation Accuracy: {history['val_acc'][-1]:.2f}%")
```
```
Final Training Accuracy: 89.34%
Final Validation Accuracy: 89.13%
```

Task 5 - Final Model Accuracy:

After running 85 epochs, we can see my model achieved a final training accuracy of 89.34% and a final validation accuracy of 89.13%. The loss decreased from 1.8367 in the first epoch to a loss of 0.3299 in the 85th epoch. To conclude, I have successfully implemented a model to classify images from the CIFAR-10 dataset with an accuracy of approximately 90%.