ECS659U/P/7026P Coursework

Name: meriam saad kharchef

ID: 221055119

Downloading Data

firstly we load and preprocess the CIFAR10 dataset using torchvision, defining data transforms and creating data loaders for both training and testing sets

```
# Define the data transforms
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))
])

# Load the data
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True, num_workers=2)
testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False, num_workers=2)
```

Defining the model:

The Block class defines a residual block with multiple convolutional layers, batch normalization, dropout, and a multi-layer perceptron (MLP). The input x passes through the MLP and the output is multiplied with the output of each convolutional layer in the block. The final output is the sum of these multiplied outputs and the input passed through a 1x1 convolutional layer.

The Classifier class consists of multiple Block objects connected in series, followed by a global average pooling layer and a fully connected layer that outputs the class probabilities.

More details of the code can be found in the **ipynb file**

Defining the loss and optimiser:

This code defines the model, optimizer, loss function, and learning rate scheduler for the image classification task.

The Classifier model is instantiated with the specified hyperparameters, and the model is moved to the device (GPU if available, else CPU) using .to(device) method.

The nn.CrossEntropyLoss() is used as the loss function for the classification task.

The optimizer is defined using optim.SGD with a learning rate of 0.01, momentum of 0.9, and weight decay of 5e-4.

A learning rate scheduler is defined using torch.optim.lr_scheduler.StepLR that reduces the learning rate by a factor of 0.1 after every 20 epochs.

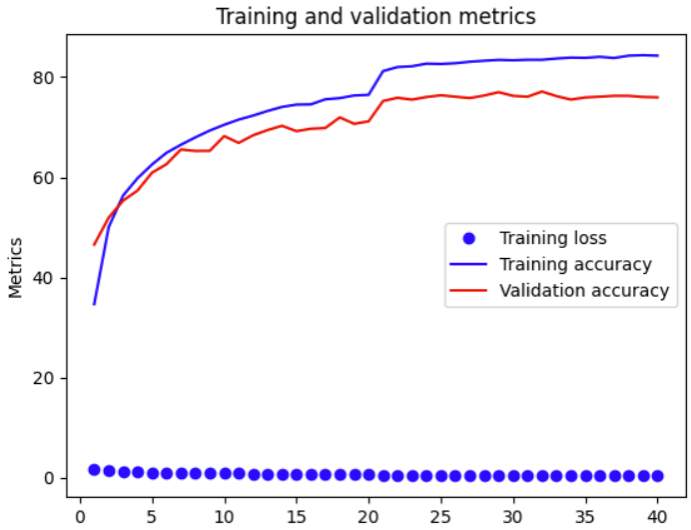
```
[ ]  # Define the model, optimizer, and loss function
     device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
     net = Classifier(num_blocks=3, num_filters=64, num_classes=10, num_convs=3).to(device)
     criterion = nn.CrossEntropyLoss()


     # Define the optimizer and learning rate scheduler
     optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=5e-4)
     lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.1)
```

Training and results:

After training by using 40 epochs, we get an accuracy result of 75%.

The loss is 0.451 and appears to have plateaued by this point.

```
Epoch [40] trainng loss: 0.451, training accuracy: 84 %
Epoch [40] validation accuracy: 75 %
Finished Training
```



Hyper parameters used:

Hyperparameters used:

Epochs: 40

Batch size: 128

Learning rate: 0.01

Weight decay: 5e-4.

Optimizer: SGD

Loss: Cross entropy loss

Number of blocks: 2