ECS659U/P/7026P Coursework

Task 1: Read dataset and create data loaders

Read dataset and create data loader (5%)

```python
#Loading the dataset
## Define the CIFAR 10 Classes
LABELS = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# Transaform the dataset by changing the values to Tensors and Normalising the Tensor Values
transform = torchvision.transforms.Compose([
    torchvision.transforms.RandomCrop(32),
    torchvision.transforms.ColorJitter(),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Batch Sizes: The amount of images used in each epoch
BATCH_SIZE = 256

# Download the CIFAR Dataset
train_data = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_data = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

# Create a datset loader
train_loader = torch.utils.data.DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True, num_workers=2)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=BATCH_SIZE, shuffle=False, num_workers=2)
```

✓ 1.6s

```
Files already downloaded and verified
Files already downloaded and verified
```

```python
X, y = next(iter(train_loader))
print(X.size())
#Check the batch size: returns ([batch_size,3 channels, 32 pixel, 32 pixel])
```

✓ 13.2s

```
torch.Size([256, 3, 32, 32])
```

Figure 1: Dataset being read in and data loader creation

Task 2: Create the model

```
CNN(
  (backbone): Backbone(
    (block1): ConvBlock(
      (conv): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (activation): ReLU(inplace=True)
      (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
    (blocks): ModuleList(
      (0-3): 4 x ConvBlock(
        (conv): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        (bn): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (activation): ReLU(inplace=True)
        (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
      )
    )
  )
  (classifier): Classifier(
    (pool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Sequential(
      (0): Linear(in_features=64, out_features=512, bias=True)
      (1): ReLU(inplace=True)
      (2): Linear(in_features=512, out_features=10, bias=True)
    )
  )
)
```

Figure 2: output of model

Task 3: Create the loss and optimiser

create loss and optimiser 5%

```python
import torch.optim as optim

# Define the loss function and optimizer
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

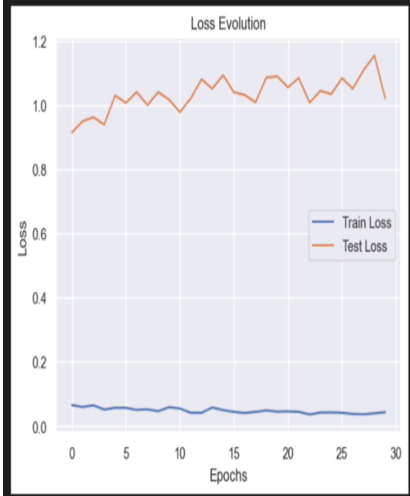
✓ 0.3s

Task 4:

- For task 4 I defined the number of epochs which sets the number of times the model will iterate over the entire training dataset.
- Then I check if a GPU is available and if so move the model to function from the GPU for faster computations.
- Then I create a training loop, for each epoch the model is set to train and the training data is iterated over in batches. The inputs and labels are moved to the GPU if available. The optimisers gradients are zeroed and then the forward pass is performed on the model using the input data. The loss is calculated using the loss function and the output of the model. The gradients are then calculated using the backward pass and the optimiser takes a step. The running loss and accuracy are updated for current batch.
- Once this is complete I plot the evolution of Loss and accuracy for training and testing as.

```python
plt.plot(range(num_epochs), train_losses, label='Train Loss')
plt.plot(range(num_epochs), test_losses, label='Test Loss')
plt.title('Loss Evolution')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

✓ 0.1s




```python
import matplotlib.pyplot as plt

# Plot the training and testing accuracy curves
plt.plot(train_accs, label='Training Accuracy')
plt.plot(test_accs, label='Testing Accuracy')
plt.title('Accuaracy evolution')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

✓ 0.1s

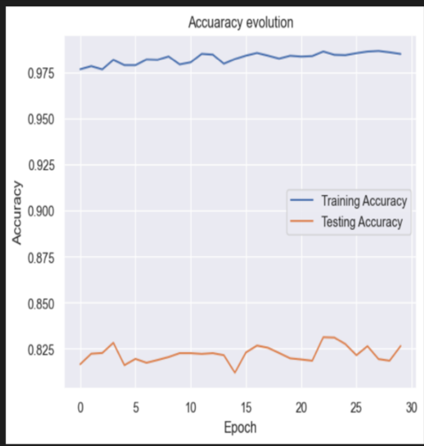


Figure 3: Curves for evolution of loss

Figure 4: Curves for evolution of Accuracy

Hyperparameters:

Learning rate: 0.001

Epochs: 30

Momentum: 0.9

Task 5: Display final model accuracy

```python
# Calculate the final test accuracy
model.eval()  # Set the model to evaluation mode
running_corrects = 0  # Initialize the number of correct predictions to 0

# Iterate over the test data
for data, target in test_loader:
    if torch.cuda.is_available():
        data = data.cuda()  # Move the inputs to the GPU if available
        target = target.cuda()  # Move the labels to the GPU if available
    output = model(data)  # Forward pass
    _, predictions = torch.max(output, 1)  # Get the predicted classes
    running_corrects += torch.sum(predictions == target.data)  # Count the number of correct predictions

final_test_acc = running_corrects.double() / len(test_set)  # Calculate the final test accuracy

# Print the final test accuracy
print('Final Test Accuracy: {:.4f}'.format(final_test_acc))
```

✓ 15.6s

```
Final Test Accuracy: 0.8246
```

Figure 5: Final model Accuracy on test set