ECS7026P

Neural Networks and Deep Learning

Coursework Assignment

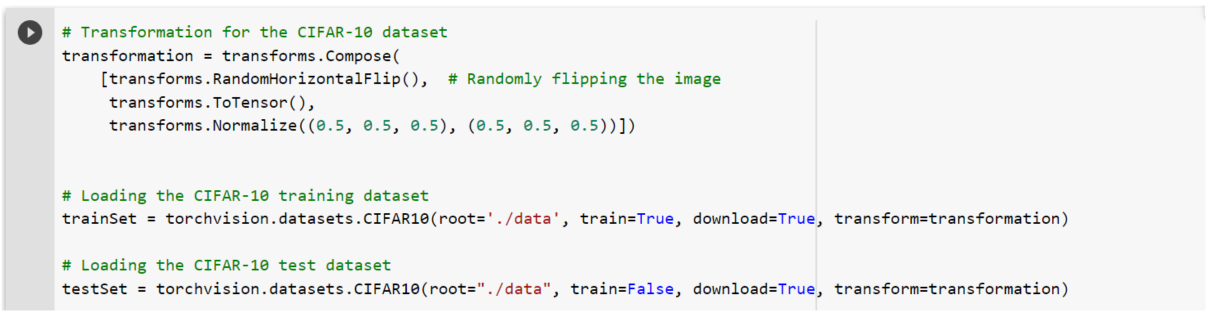
NITISH KRISHNA SADHU

220283937

EC22098

Task – 1:

The following code downloads the data and applies the defined transformation.



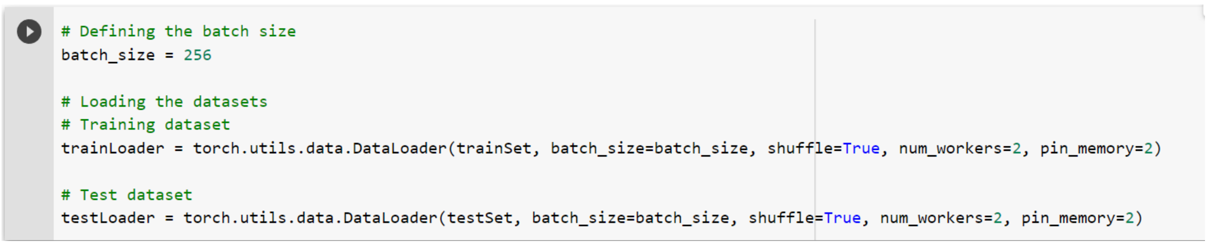
```
# Transformation for the CIFAR-10 dataset
transformation = transforms.Compose(
    [transforms.RandomHorizontalFlip(),  # Randomly flipping the image
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])


# Loading the CIFAR-10 training dataset
trainSet = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transformation)

# Loading the CIFAR-10 test dataset
testSet = torchvision.datasets.CIFAR10(root="./data", train=False, download=True, transform=transformation)
```

In the following code:

- The *batch_size* is defined as 256.
- The next line of the code converts the *trainSet* into a *trainLoader* which is in the dataloader format.
- The final line of the code converts the *testSet* into a *testLoader* which is in the dataloader format.



```
# Defining the batch size
batch_size = 256

# Loading the datasets
# Training dataset
trainLoader = torch.utils.data.DataLoader(trainSet, batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=2)

# Test dataset
testLoader = torch.utils.data.DataLoader(testSet, batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=2)
```

Task-2:

The model:

- Consists of four ***BackboneBlock***s.
- Each ***BackboneBlock*** consists of two parts:
  - The first part consists of the code that produces the vector **a**.
  - The second part consists of the convolution layers.
- In the forward function, the vector **a** is computed first.
  - The input first goes into an average pooling layer.
  - The output then goes into a Linear layer which outputs **k** features.
  - The ReLU activation is applied to the output.
- The next step is to apply the convolution layers to the input images and multiply the output of each convolution layer with the individual elements of the vector.
- The next step is to apply the batch normalization and max pooling which helps in improving the accuracy.
- The output from the four ***BackboneBlocks*** is comes out, it goes into the classifier part of the model which is an MLP layer, with two layers.

- The ***ReLU*** activation is given between the two layers.
- A dropout with a p=0.25 is also given between the two layers.

Task-3:

- The loss used is CrossEntropyLoss.
- The optimizer used is Adam with a learning rate of 0.001.

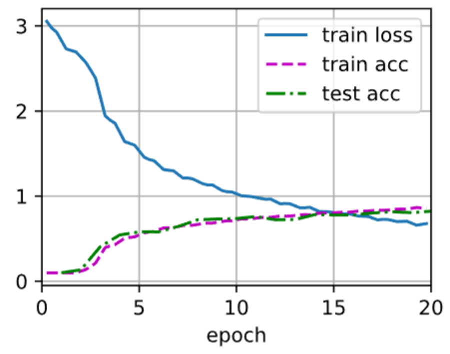
Task-4:

- Epochs used: 20
- The learning rate used: 0.001.
    - Tested with the learning rates 0.1, 0.01, 0.001, 0.0001.
    - The accuracy was lower for 0.1 and 0.01 than 0.001.
    - The time taken to train 0.0001 is high.
    - The Learning rate 0f 0.001 yielded the best accuracy.
- The batch size used: 256.
    - Tested with 4, 16, 128, 256.
    - The batch size of 256 yielded the best results.
- The k used is 16.
    - Tested with k = 8, 16 ,32.
    - K=16 yielded the better results among the three.

# Graph of loss, training accuracy and test accuracy.

```
[61] n_epochs = 20    # Defining the number of epochs
     lr = 0.001        # Defining the learning rate
     train_model(model, trainLoader, testLoader, n_epochs, lr, device)    # Training the model
```

```
loss 0.680, train acc 0.857, test acc 0.823
3616.0 examples/sec on cuda
```



<u>Task-5:</u>

- The accuracy initially obtained was around 30 percent.
- Has tested with 3, 4, 5 ***BackboneBlocks***, 4 blocks gave the best of both world, i.e., training reasonably fast and providing the best accuracy.
- The learning rate used: 0.001.
    - Tested with the learning rates 0.1, 0.01, 0.001, 0.0001.
    - The accuracy was lower for 0.1 and 0.01 than 0.001.
    - The time taken to train 0.0001 is high.
    - The Learning rate 0f of 0.001 yielded the best accuracy.
- The batch size used: 256.
    - Tested with 4, 16, 128, 256.
    - The batch size of 256 yielded the best results.
- The k used is 16.
    - Tested with k = 8, 16 ,32.
    - K=16 yielded the better results among the three.
- The addition of Batch normalization and Max pooling before emitting the output from ***BackboneBlock*** drastically improved the accuracy.
- The classifier has two linear layers, using the ReLU activation and dropout with p=0.3, improved the accuracy from around 70% to around 82.3%.

<u>References:</u>

- Used my_utils.py file with some modifications to write the training script and plot the graphs.