ECS659U/P/7026P Coursework: Classify Images in CIFAR-10 Dataset
Student ID: 220843155
Name: Farheen Dairkee

Task 1: Read dataset and create data loaders

In task 1, after importing the relevant libraries, the code defines two data transformations namely train_transform and test_transform. In the train_transform function, a number of data augmentation techniques were applied like random cropping, horizontal flipping, random rotation, and normalization. Other data augmentation techniques tried were gaussian blur and colour jitter but these techniques take up a lot of time to run and hence were discarded. The reason to apply data augmentation is to artificially increase the size of training dataset. It is done to improve the performance and generalisation of deep learning models.

The code then loads the CIFAR-10 dataset using 'datasets.CIFAR10' and applies the train_transform to the training set and test_transform to the test set. The dataset is stored in train_dataset and test_dataset variables respectively. Lastly, the code creates PyTorch data loaders for both the training and test sets using torch.utils.data.DataLoader and specifies the batch size and shuffling of the data. The training set data loader is stored in the train_loader variable and the test set data loader is stored in the test_loader variable.

Task 2: Create the model

The model consists of two classes namely Far_Block1 and Block Sequence.

Class Far_Block1 defines a block of the architecture. Each block of the architecture contains a linear layer and 3 convolution layers. A spatial average pooling is applied to the linear layer and then an activation function is used for linear layer which is the softmax function. The outputs of the linear layer is multiplied with the outputs of convolution layer as per the instructions given in the coursework.

Each convolution layer has differing kernel sizes of 3,5 and 7 with padding as 1,2 and 3. Different kernel sizes help to extract information of different scales or levels of abstraction from the image. Large kernel sizes detect global features of the image such as shape of an object whereas smaller kernel sizes capture local details like edges or textures. Therefore, it helps to recognize complex patterns from the image. Stride is kept as 1 for each convolution layer to avoid any loss of information from input.

In addition to convolution layers, more elements have been applied to the output such as activation function which introduces non-linearity in the model and batch normalisation which normalises the input of each layer and makes training faster and more stable. The activation function chosen is the ReLU function which is computationally efficient and avoids the vanishing gradient problem. Since it sets the negative values to 0, it also introduces sparsity in the network which increases generalisation.

The class Block_sequence defines the sequence of blocks to be used in the neural network. The number of blocks were decided by keeping in mind the increase in complexity and runtime of the training code. The input and output channels of the blocks are increased as we go deeper into the CNN architecture since we want the model to capture more complex patterns. The network is able to learn more abstract representations of the input because it takes output of previous layer as input and that output has learned higher level features.

Finally, the 'mish' activation function is found to work best when applied to the the output of the block sequence. The Mish activation has advantages such that it is differentiable everywhere which avoids the issue of vanishing gradient and it is a non-monotonic function which helps in

increasing expressivity of the network. According to the article [1], the Mish activation function performs the best among all the other activation functions.

Skip connections have been applied from blocks 1 to 3, 2 to 4 and 3 to 5. The skip connection is useful to alleviate any issues of vanishing gradient. The skip layer includes a convolution layer with in and out channels as per the block with a kernal size of 1.

The output of the final block is then fed into adaptive average pooling and finally into two fully connected layers with hidden sizes of 1000 and 200 and a final 10-dimensional output layer that corresponds to the number of possible output classes. The first two fully connected layer also go through batch normalization and dropout. A dropout of 0.2 is applied which is done to randomly dropout some neurons in training and prevent overfitting and learn more robust features.

Task 3: Create the loss and optimizer.

The loss function used is the Cross Entropy loss, it is a common choice for classification tasks and finds the difference between predicted class probabilities and the the true labels.

The optimizer used is the Adam optimizer which is an extension of the classical stochastic gradient descent. It provides the benefits of bias correction and adaptive learning rates of each weight based on first moment (mean) and second moment () of gradients. This adaptive learning rate makes it useful for different weights that need different learning rates.

Task 4: Loss and Accuracy Curves, Hyper-parameters used

Instead of using a fixed flearning rate, I have used a learning rate scheduler called the Cosine Annealing Learning Rate, it follows a cosine curve for learning rate and is used for longer training cycles. This scheduler reduces the learning rate smoothly and not drastically and helps in convergence and prevents overfitting. It also allows for learning rate to increase again which might help to explore different regions of loss landscape. The parameters for the scheduler are max. number of epochs which are set to the number of epochs that I have run and the minimum learning rate that it can decrease to is given as 0.00001 (be default it is 0).

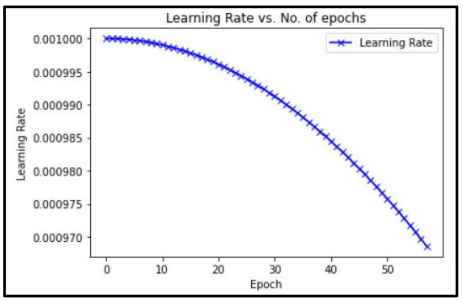

Figure1 : Change in Learning Rate with Epoch

As each input neuron is connected to each output neuron in linear layers, weights determine how strongly each input neuron affects the output neuron. The weights of convolution layers determine the significance of features extracted from input data. Weight initialization set weights to random small values for optimization algorithm. In the code, the He Weight Initialization method is used. As per the article [2], the standard approach for intialization of weights using the ReLU activation is '*he*'. The weight is a random number from a Gaussian distribution with mean 0.0 and standard deviation as (sqrt(2/n)) where n is number of inputs. It sets variance of output of layer same as that of input, which helps in vanishing or exploding gradients.

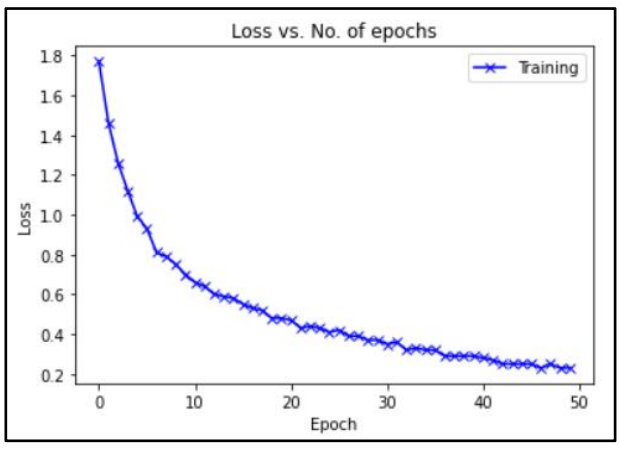

Figure2 : Plot of Training Loss

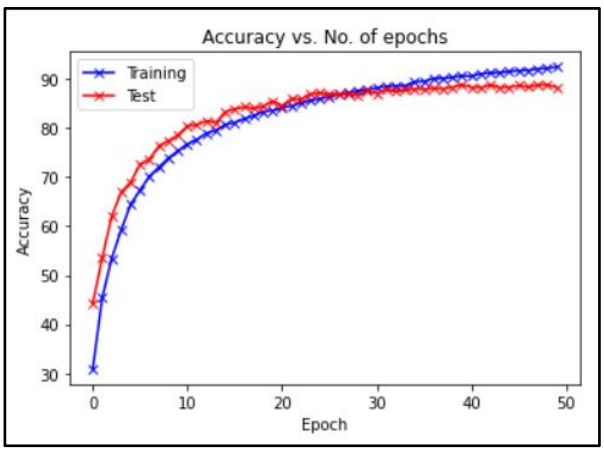

Figure3 : Accuracy of Training and Test

As can be seen from the accuracy plot, the test accuracy has plateaued at around 88% to 89% while the training accuracy continues to increase. Running till the 50th epoch is enough as the training accuracy increases the model could overfit to the the training dataset while there isn't much difference in the test accuracy.

Task 5: Final Model Accuracy

The final model test accuracy achieved is approximately 88%.

```
Epoch [50/500], Step [100/782], Train Loss: 0.2092
Epoch [50/500], Step [200/782], Train Loss: 0.2255
Epoch [50/500], Step [300/782], Train Loss: 0.2124
Epoch [50/500], Step [400/782], Train Loss: 0.2167
Epoch [50/500], Step [500/782], Train Loss: 0.2259
Epoch [50/500], Step [600/782], Train Loss: 0.2331
Epoch [50/500], Step [700/782], Train Loss: 0.2298

Accuracy of the model on the 157 test images: 88.22%

Accuracy of the model on the 782 train images: 92.42%
```

References:

[1] : https://wandb.ai/shweta/Activation%20Functions/reports/Activation-Functions-Compared-With-Experiments--VmlldzoxMDQwOTQ#the-mish-activation-function

[2] : https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/