Neural Networks and Deep Learning
CIFAR-10 Classification – Coursework Report

The Goals :

- To implement a **specific** model to solve the CIFAR-10 classification problem and classify every image in terms of 1 out of 10 classes.
- To implement the training pipeline to train the model and achieve the highest accuracy possible.

Task 1 - Preparing the Dataset :

The primary step was to prepare the dataset for our use case. This has been achieved by applying a few transformations on the training images:

- transforms.ToTensor() : to convert the images into PyTorch tensors
- transforms.Normalize() : we normalize the tensors by subtracting the mean, which is 0.5 and dividing by the standard deviation that is 0.5 too

A definition of hyperparameters have also been provided in the code relating to this section, and the following are assigned or set –
Number of epochs = 40 (num_epochs)
Learning Rate = 0.0015 (learning_rate)
Batch Size = 128 (batch_size)
The datasets are then loaded into DataLoader instances, enabling batching, shuffling and parallel processing during the training and testing cycles.

Task 2 - Model Creation :

1) Block Class: It depicts a building block of the CustomModel class.
   This component consists of the following implementations –
   Adaptive average pooling, a fully connected Linear layer, 'K' convolutional layers, a Residual Connection.

2) CustomModel Class: The overall architecture mainly consists of two components, which are – the Backbone and the Classifier.
   The Backbone follows the specification provided in the given PDF for the coursework, consists of a sequence of blocks and some layers.
   The components used here are – Batch Normalization, ReLU and MaxPool2D.
   The Classifier is used to process the output of the last block. It contains – AdaptiveAvgPool2D, Flatten function and the Linear layer.

Task 3 - Defining Loss Function and Optimizer :

1) Enable GPU for training :
   The code enables GPU usage for training the model if it is available, else the execution will take place through a CPU.
2) Loss Function : CrossEntropyLoss()
3) Optimizer : RMSProp
   Adaptive learning rate optimization algorithm, which takes the learning rate as a parameter, i.e., lr = 0.0015
   At the end of the code cell which relates to this section, I have used the torch.save() function to save the model in a .pt (.pytorch) file called 'model-01.pt'.
   The code can be used by calling this during execution rather than running the training section of the notebook again.
4) LR Scheduler : CosineAnnealing
   It is used to adjust the learning rate (LR) based on training progress of our code. It takes two parameters, which are maximum number of training iterations (T_max) and minimum learning rate (eta_min).

Task 4 – Script to Train the Model :

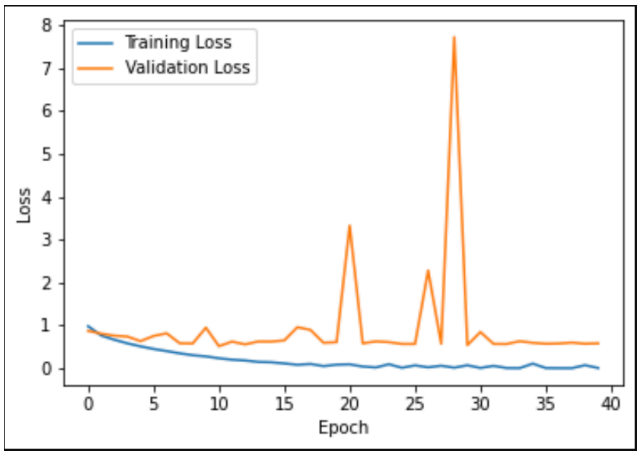
The two functions train_epoch and validate_epoch have been defined to perform the training and validation tasks respectively.

- *train_epoch* : It takes the following as input - model, dataloader, criterion, optimizer, device, accumulation_steps.
  The primary step is to set the model to 'training mode'. Then, variable assignment is done for total samples, correct predictions and running loss. Then we use a for loop to iterate through the training data. This loop consists of steps as listed here – moving images and labels back to device, performing a forward pass through, using the given criteria to calculate the loss, gradient computation using backpropagation, model parameter update, increment counter for total samples and correct predictions.
  Finally, the average loss and accuracy for every individual epoch is returned.
- *validate_epoch* : It takes the following as input - model, dataloader, criterion, device.
  The first step is to set the model to 'evaluation mode', then variables are assigned running loss, correct predictions, and total samples. We then disable the gradient calculation using torch.no_grad() and inside the loop, we move inputs, labels back to the device followed by a pass through the

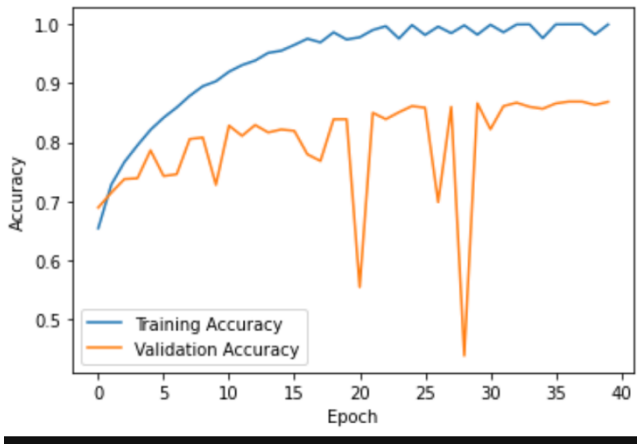
model. Then we calculate the loss keeping in mind the given criteria, and then update the counters for total samples and correct predictions.
Finally, the average loss and accuracy is returned for every individual epoch.

Task 5 – Final Model Accuracy :

This section deals with the visualisation part of the loss and accuracy curves, in terms of training and validation. The following graphs were obtained from our training pipeline :



Training vs Validation Loss



Training vs Validation Accuracy

After completing 40 epochs, the model managed to achieve a training accuracy of 99.96% and a validation accuracy of 86.85%.
Training took 687 seconds (or approximately 12 minutes) to complete.
The training loss can be seen decreasing drastically from 0.763 to 0.005, while the validation loss dropped from 0.805 to 0.580.
In conclusion, this project successfully implements a solution to the CIFAR-10 classification problem by following a custom defined architecture.