NEURAL NETWORKS AND DEEP LEARNING CIFAR-10 CLASSIFICATION- COURSEWORK REPORT

Goals to be achieved :

- The goal is to create a model that can effectively classify images in the CIFAR-10 dataset into 1 of the 10 classes.
- This involves implementing a training pipeline that can train the model to achieve the highest possible accuracy on the CIFAR-10 classification task.

Task 1 - Preparing the Dataset :-

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

- transforms.ToTensor() : to convert the images into PyTorch tensors.
- transforms.Normalize() : In this section, we perform tensor normalization by subtracting the mean, which is set at 0.5, and dividing by the standard deviation, also set at 0.5. These values are defined as hyperparameters in the code, and their assigned or set values are used for the normalization process.

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 – Creating the model :-

1) Block Class: This component is a building block of the CustomModel class and includes the following implementations: adaptive average pooling, a fully connected Linear layer, 'K' convolutional layers, and a residual connection.

2) CustomModel Class: The overall architecture of the model comprises two main components: the Backbone and the Classifier. The Backbone follows the specifications provided in the given coursework PDF and consists of a sequence of blocks and 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 the Loss Function and Optimizer :

1) Enabling the GPU for training :

The code allows for GPU utilization during model training if a GPU is available, otherwise, the execution will default to using the CPU.

2) Loss Function: CrossEntropyLoss()
3) Optimizer: Adam optimizer

   An adaptive learning rate optimization algorithm is implemented in the code, taking the learning rate as a parameter (e.g., lr = 0.0015). At the end of the code cell pertaining to this section, the torch.save() function is used to save the model in a .pt (.pytorch) file named 'model-1.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

   The learning rate scheduler in our code is utilized to dynamically adjust the learning rate (LR) during training based on the progress. It requires two parameters: the maximum number of training iterations (T_max) and the minimum learning rate (eta_min).

Task 4 – Script for Training the Model :

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

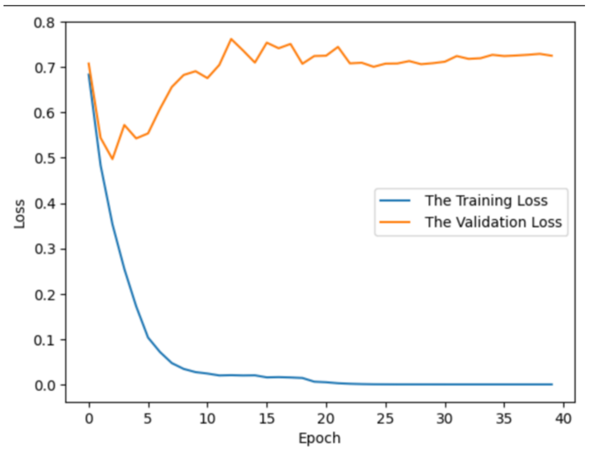
- *train_epoch* : The input parameters for this function include the model, dataloader, criterion, optimizer, device, and accumulation steps. The function begins by setting the model to 'training mode' using the appropriate method. Next, variables are initialized for total samples, correct predictions, and running loss. A for loop is then used to iterate through the training data. Within the loop, the images and labels are moved to the device, a forward pass is performed through the model, the given criterion is used to calculate the loss, gradient computation is performed using backpropagation, model parameters are updated using the optimizer, and counters for total samples and correct predictions are incremented. Finally, the average loss and accuracy for each epoch are returned.

- *valid_epoch* : This function takes the following inputs: model, dataloader, criterion, and device.

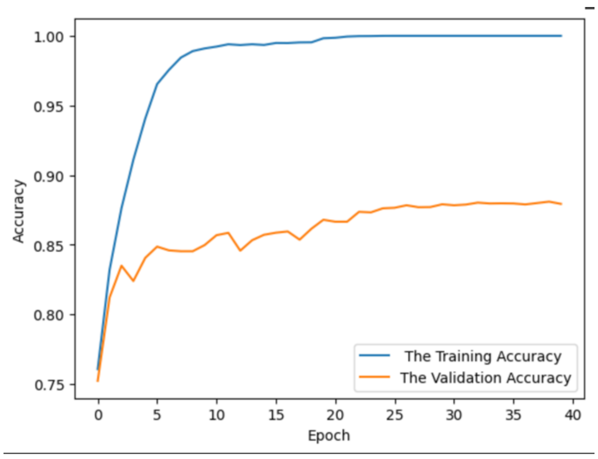
  The first step in the function is to set the model to 'evaluation mode' using the appropriate method. Then, variables are initialized for running loss, correct predictions, and total samples. The gradient calculation is disabled using torch.no_grad() to avoid unnecessary computations. Within the loop, the inputs and labels are moved back to the device to ensure consistency. The inputs are then passed through the model to obtain predictions. The loss is calculated based on the given criterion. The counters for total samples and correct predictions are updated accordingly. The gradient calculation is disabled using torch.no_grad() to avoid unnecessary computations. Within the loop, the inputs and labels are moved back to the device to ensure consistency. The inputs are then passed through the model to obtain predictions. The loss is calculated based on the given criterion. The counters for total samples and correct predictions are updated accordingly.

Task 5 – Final Model Accuracy :

This section focuses on the visualization of the loss and accuracy curves during the training and validation phases. 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 100.0% and a validation accuracy of 87.92%.
Training took 1997.92 seconds to complete.
The training loss can be seen decreasing drastically from 0.254 to 0.00, while the validation loss showed a marginal difference from 0.57 to 0.72.
The training accuracy can be seen changing from 0.91 to 1.00 whereas, the validation accuracy showed a deviation from 0.823 to 0.879
In conclusion, this project successfully implements a solution to the CIFAR-10 classification problem by following a custom defined architecture.