NEURAL NETWORKS AND DEEP LEARNING (ECS659U)

CIFAR-10 Dataset Problem

TASK 1: Read the Datasets and create dataloaders

Explanation:
The CIFAR-10 dataset is downloaded and loaded in order to use it for the model. The transformations are defined in order to perform preprocessing on the input dataset that will be used for the training set. It is applied to the tensors and I have used mean and standard deviation for normalising it. The hyperparameters are parameters used for the training model purpose to obtain better accuracy results and are subsequently set manually as per the predicted output requirement. In this context, hyperparameters used are Epoch, Learning rate, batch size and ReLU activation function for the optimal accuracy of the model. The trainloader and testloader object loads the dataset into the batch size and stores it in the variables defined with the same name.

TASK 2: Create the Model

Explanation:
The CustomModel is build by stacking multiple Block instances and fully connected layers in a sequential container. The total block instances are 5, hence n=5, while the image data is classified in 10 classes(as per the CIFAR-10 dataset) as inserted in linear layer of backbone. The CNN model is constructed which is named as CustomModel that is implemented into two parts. One is backbone and second one is classifier. Backbone part is responsible for extracting features from the samples. It consists of several blocks with convolutional layers, activation function batch normalization layers, and residual connections. The Residual connection acts as a bridge between convolutional layers to take care of gradient while I have used ReLU activation function as it is uncomplicated and the sparsity aspect of it prevents model overfitting.

Classifier network comprises of full fledged connect linear layers comparison of activation function, batch norm and blocks. The adaptive average pooling which reduces on the vector size and maps it to the ‘k’ vector size. The activation function I used is ReLU as it is uncomplicated and effective, also owing to its sparsity which keeps model from overfitting. Additionally, its allows backward navigation seamless as it seeks to maintain gradient which vanishes as it goes from layer to another layer. Since the function propogates non-linearity its reflects in the neural network during epochs as allows the model to learn data more effectively.

In the backbone, ReLu is attached to the convolutional layer and is connected as a part of classifier network layers. For the final layer, Logsoftmax function is used for effective gradient optimization and smooth training process during the epoch. The batch normalization is applied here for the normalisation process of each layers, it is essentially effective for CNN network especially for image processing tasks.

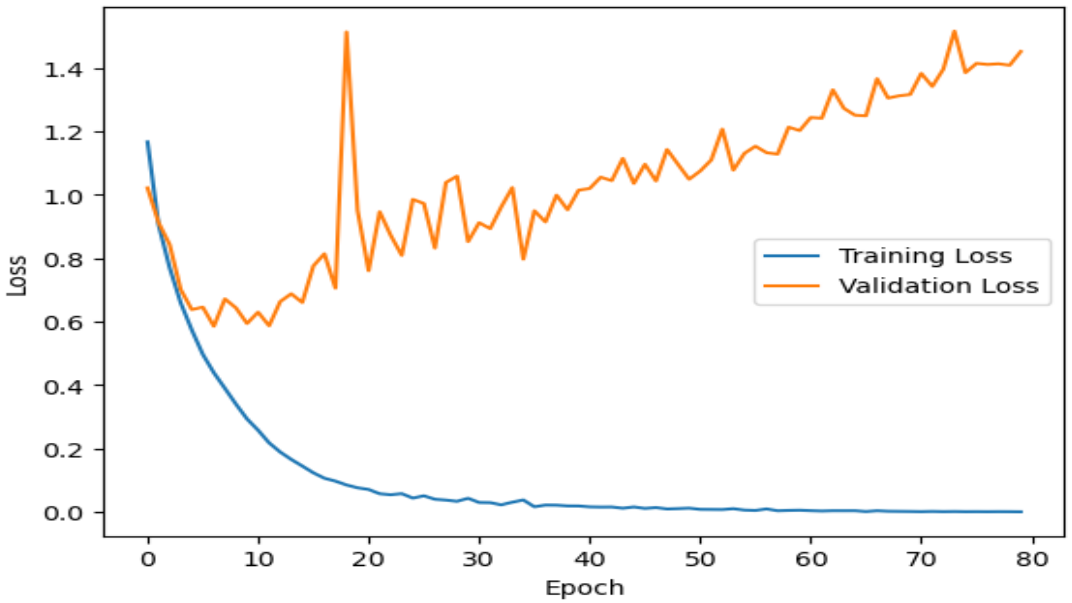
TASK 3: Create the Loss and optimizer

Explanation:
The Loss function determines the model's performance by estimating and comparing the predicted target output and the actual output. It overall works to ensure the performance and quality of model. Here, I have used Cross Entrophy Loss function as it is an effective multi classification function and in the context of dataset where 10 classes are inserted. Lower the value of loss function, closer the output is to the predicted one.

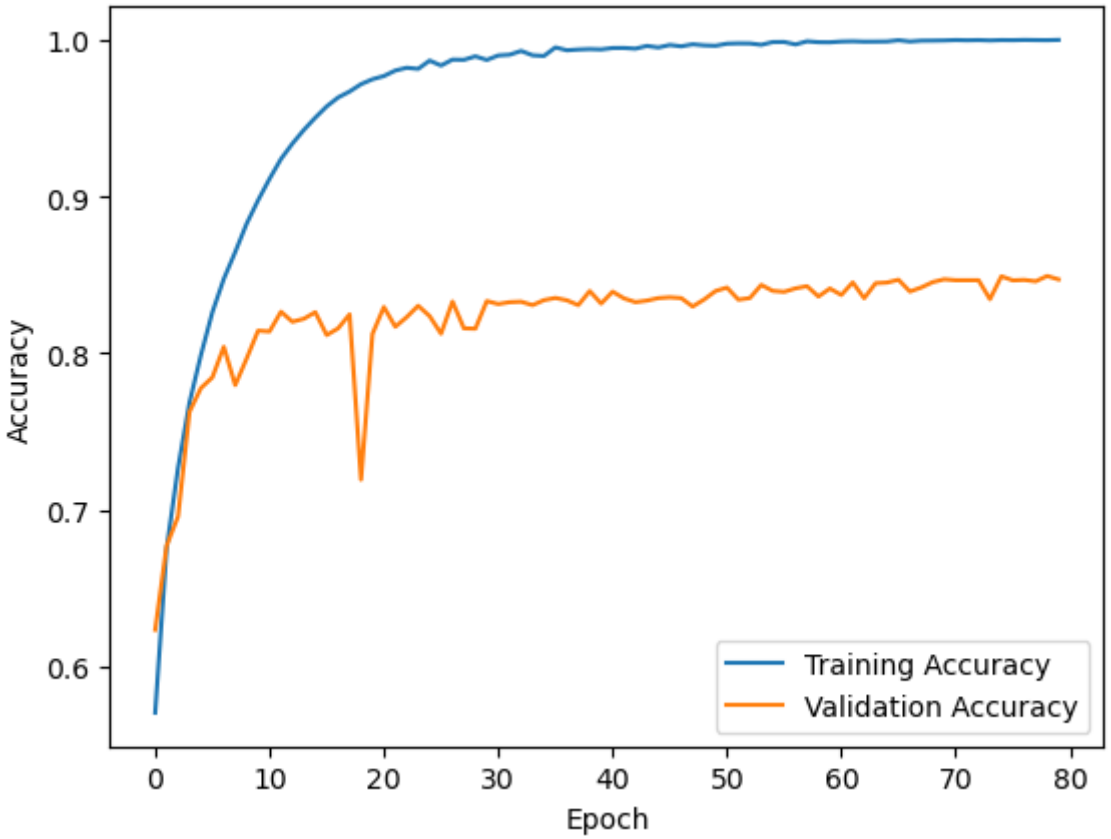
To minimize the value of loss function, optimiser comes into picture for the neural network. The optimizer is responsible for optimising and thus reducing loss function value. I choose RMSprop optimiser for the model as it is proven to be giving me better results for the optimization task of the neural architecture of the model. I previously experimented with SGD (Stochastic Gradient Descent)) as a optimizer for the model, but it seemed to be giving me validation accuracy of 62%, while RMSprop constantly gave better accuracy results, mostly over 80%.

TASK 4: Training Script to train the model

(i) Curves for evolution of loss :



(ii) Curves for evolution of training and validation accuracy :



Explanation:

I have defined two functions for training and validation functions. The train_epoch() uses the RMSprop optimiser to update the model hyperparameters as it runs the batches in dataloader. The Batch Size I took is 128. During each epoch, the model is trained iteratively. The loss is saved in LossFunc variable and is gathered for each batch, and the model's parameters are updated post every accumulation_steps batches. The Accumulation_steps makes track of the number of batches gradients will be accumulated for the model. I have set it to 4, so the hyperparamters of the constructed model will be updated at every 4 batches.The backward function in for loop is used to compute the gradients while zero.grad() function will zero the gradient before running the new batch so as to ensure proper propagation of the epochs.

The train_validate() is trained for the estimation of all the features and working mentioned in the training set. It intialises the input parameters like epoch, learning rate and iterates on the dataloader.

In validate_epoch() function the torch.no_grad function is implemented which stops the gradient accumulation in the train_validate() function while running the for loop for the batches and estimates the model performance for every step allowing to check the performance after every batch is executed. Thereby, I constantly altered hyperparamters to extract better results, in the sense better validation accuracy.

TASK 5: Final Model accuracy on CIFAR-10 validation set

Upon trying several different combination of altering hyperparameter values, I observed incredible variation in the final training and validation accuracy. The number of epochs count kept at 80, while learning rate as 0.0025, and batch size as 128 which gave the best optimal result with this model architecture using CIFAR-10 dataset.
The accuracies I got:
The training accuracy: 99.99
The validation Accuracy: 84.71

▾ Task 5 - Final Model Accuracy

```
[ ]  print('The training accuracy is: ', train_acc*100)
     print('The validation accuracy is: ', val_acc*100)

     The training accuracy is:  99.99
     The validation accuracy is:  84.71
```