Neural Networks and Deep Learning

Coursework: CIFAR-10 Classification Problem

Task 1: Read Dataset and Create DataLoader

For the given task, CIFAR-10 dataset which is a collection of images commonly used for image recognition task. It consist of 60,000 32x32 color images in 10 classes, with 6,000 images per class. To load CIFAR-10 dataset using the PyTorch library, we can use the torchvision.datasets module. This module load the dataset and dataloaders namely trainloader and testloader are created using torch utils package. The batch size is specified to be equal to 100. These are accessed in batches using next, iter functions. Sample images with their labels have been presented in the notebook.

Task 2: Create a model

A Multi-Layer perceptron model has been created with the following parts: Backbone and classifier. Backbone consist of 11 blocks. Each block includes a linear/MLP layer predicting a vector with k elements from input tensor X. This is done by applying fully connected layer on the output of adaptive average pooling. Then k different convolution layers on input tensor is combined with the vector to produce a single output. If the input and output channels of a convolution layer are different then a 1x1 convolution layer followed by the batch normalization is added. To amplify the model's performance batch normalization, Leaky ReLU activation and max pooling layers have been included in the implementation of the backbone. Every next block take the input from it's previous block. The classifier takes as input the output of the last block. Then computes a mean feature using AdaptiveAvgPool2d which is further passed to a fully connected layer with a softmax activation. This classification layer produces the final class scores for the input image.

For initializing the weights of the linear layers in the model, kaming_normal_ algorithm has been used as it accounts for the non-linearity of the activation functions, like ReLU activations.

Task 3: Create the Loss and Optimizer

Loss is a measure of the dissimilarity between the predicted values produced by the model and the actual values. In this case, the CrossEntropyLoss (Log Loss) function was used to calculate the difference between the predicted probabilities and the true probabilities. The CrossEntropyLoss function achieves this by taking the negative sum of the product between the true probability and the logarithm of the predicted probability for each class. For multiclass classification, the cross-entropy loss uses softmax activation in the output layer to obtain output values between 0 and 1, which are used to evaluate the performance of the classification model.

The goal of training a model using cross-entropy loss is to minimize the loss function, which corresponds to maximizing the model's accuracy on the classification task. This is achieved through the use of optimization techniques which in this case is Adam (Adaptive Moment Estimation).

The Adam optimizer was utilized with a learning rate of 0.001 and a weight decay of 0.0005. This optimization algorithm is particularly suitable for large datasets due to its ability to combine two techniques: momentum and adaptive learning rates. The momentum technique incorporates both the gradient of the current step and the momentum of the previous step to determine the direction of the next step. Adam optimizer is an extension of stochastic gradient descent that updates the weights of the network during training. Compared to other optimization algorithms, it has faster run time, lower memory requirements, and requires less parameter tuning. Additionally, a weight decay of 0.0005 was set as a regularization technique, which adds a slight penalty to the loss function.

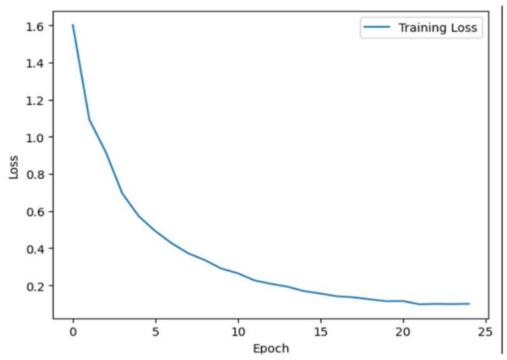
Task 4: Training script to train the model

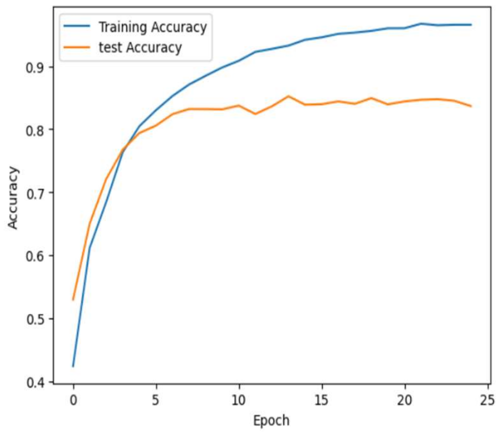
In order to train the neural network, the code has been modified to include the "cuda" module in the import section to utilize the GPU. The "to(device)" method has been used to transfer all tensors from the CPU to the GPU for improved training speed. The necessary functions such as accuracy, evaluate_accuracy, and train_epoch_ch3 have been utilized from the provided "my_utils.py" file for training the model. These functions were executed using the GPU for faster performance. Additionally, a custom function called "plot_train_test" was created, which incorporates the aforementioned functions and updates the lists of training loss, training accuracy, and test accuracy at each epoch. Finally, the matplotlib library was used to visualize the results.

Batch Size:
The batch size determines the number of images processed in the neural network during each iteration. In this case, the model was trained using a batch size of 256, meaning that the dataset of 60,000 images was divided into 256 separate batches. After experimenting with different batch sizes, it was found that the most optimal results were obtained when using a batch size of 256.

Optimizer
As mentioned in task 3, Adam optimizer with a learning rate of 0.001 and weight decay of 0.0005 has been used. Other parameters have been used with defaut values.

Epochs:
Increasing the number of epochs also increases the time taken by the model. Hence total 25 epoches have been used for training the model. Multiple runs with different epochs have been tested.





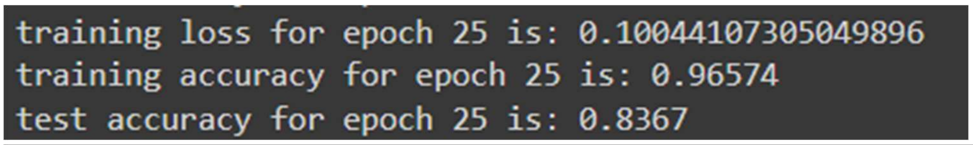
Task 5: Final model accuracy

On evaluation of the model on the testiter data, the best accuracy achieved is 85.22%. After the successful completion, the final metrics are as below:

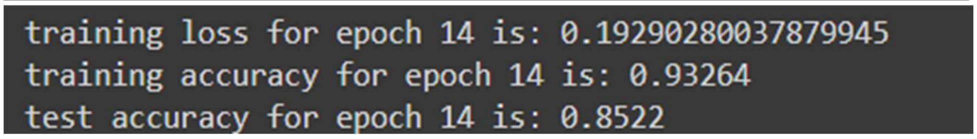
Final test accuracy:83.67%

Final Training Metrics:
Loss: 0.1
Training Accuracy: 96.57%

Through multiple attempts of manual hyperparameter tuning, a maximum testing accuracy of 83.47% was achieved. It can be concluded that utilizing the provided MLP structure facilitated the learning of diverse spatial features, and the selection of appropriate hyperparameters, including those related to the loss function, optimizer, and number of epochs, contributed to the improved performance of the model.



```
training loss for epoch 25 is: 0.10044107305049896
training accuracy for epoch 25 is: 0.96574
test accuracy for epoch 25 is: 0.8367
```



```
training loss for epoch 14 is: 0.19290280037879945
training accuracy for epoch 14 is: 0.93264
test accuracy for epoch 14 is: 0.8522
```