NEURAL NETWORK&DEEP LEARNING ASSIGNMENT REPORT

Name: Jude Damian Sequeira

Student ID: 220431413

1. TASK 1: Read Dataset and Create Dataloaders

The first task was loading the CIFAR 10 dataset using the *torchvision* library of pytorch and store the train and test datasets in its variables respectively. Also the images were converted to tensors in the same instruction while downloading the dataset by using the *ToTensor()* function. This converts the image pixels to a tensor mostly ranging between [0,1]. The training and test data have 50000 and 10000 images respectively with 10 class labels namely*: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'.* Data Loaders for train and test image tensors were created with a batch size of 128, with shuffling included only for the training set and *num_workers=2* as per recommendations from the local system.

2. TASK 2: Create the Model

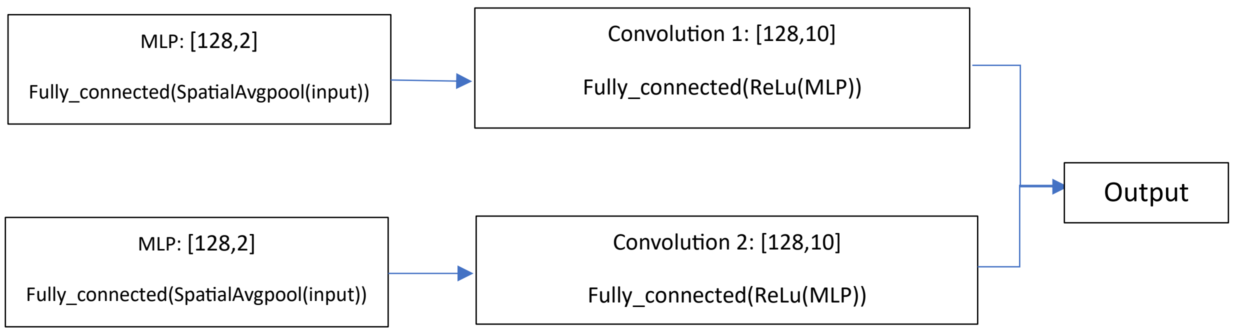
For the model, first the Spatial Average Pooling of size 1 is applied to the 3*32*32 input which is then flattened to a single dimension to apply the linear layer and stored in variable mlp. The tensor size of mlp is [128,2]. Two Convolutions are applied separately to the mlp with 3 input channels of the rgb input image, 6 output channels and kernel of size 5x5.

Further the *ReLu* activation function is applied to the two convolutions and flattened before inputting the Linear function with stride=1, padding=0 and input dimensions of the previous mlp layer which is 28*28*6, the size of 28 obtained using the formula:

((input_size-kernel_size+2*padding)/stride) +1 => ((32-5+2*0)/1) + 1 => 28.0

The two separate blocks are added together and the result is stored in the variable *'out'.*

The class *ImageClassifier()* performs the model creation in the notebook python file.


MLP: [128,2]
Fully_connected(SpatialAvgpool(input))
Convolution 1: [128,10]
Fully_connected(ReLu(MLP))
MLP: [128,2]
Fully_connected(SpatialAvgpool(input))
Convolution 2: [128,10]
Fully_connected(ReLu(MLP))
Output

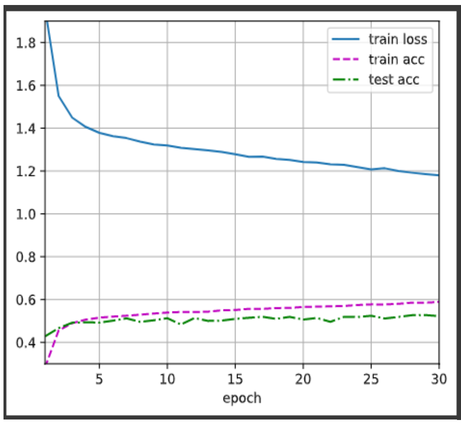
3. Task 3: Define the Loss and Optimizer

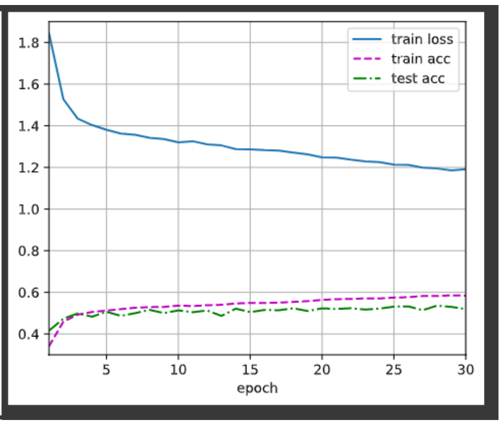
The Cross Entropy Loss and the SGD() gradient descent functions are used as the loss and optimizer functions for this task.

4. Task 4: Graphs and Hyperparameters

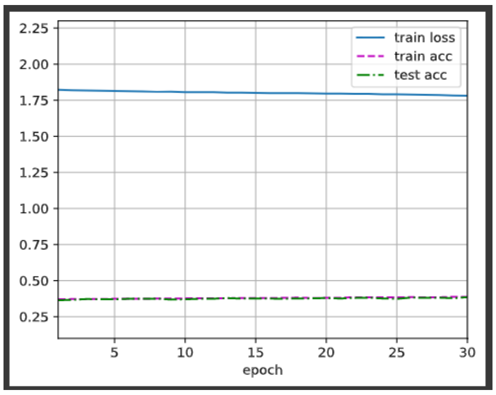
Hyperparameters such as learning rate(lr), momentum(mom) and weight_decay(wd) are used for further classification. In this coursework, the momentum value is set to 0.9 and weight decay of 5e-4 is kept fixed. Three different models are implemented with different learning rate values (i:e 0.01, 1e-3, 0.005) and their accuracies and graphs compared.

The figures below are the curves for the loss evolution and the training and testing accuracies for each different model with respective hyperparameter settings:





Fig(a) lr= 0.005 **Fig(b) lr= 0.01**



Fig(c) lr= 0.0001

Based on the above graphs, Fig(a) and Fig(b) obtain train and test accuracies around 50% on 30 epochs which is higher than the accuracy obtained in Fig(c) which is around 30%. Similar is the case for the loss function computed which is linearly decreasing with value around 1.2 after 30 epochs for both fig(a) and fig(b) whereas the loss in fig(c) is almost linearly decreasing but with a higher value around 1.75 after the iterations. Overall, Fig(c) is not a very good model compared to the other two.

5. Task 5: Find Model Accuracy

The following are the outputs obtained on the model with different Learning Rate Hyperparameters

No.	Learning Rate (lr)	Momentum (mom)	Weight_Decay (wd)	Accuracy
1	0.005	0.9	0.0005	52.24%
2	0.01	0.9	0.0005	51.97%
3	0.0001	0.9	0.0005	38.57%

Using the network architecture of the model and the hyperparameters used from the previous tasks, the best model accuracy is found to be the model with learning rate 0.005 given the constant values of momentum and weight decay as mentioned previously. The readings mentioned in the above table can also be viewed from the graphs in the previous tasks, which shows that fig(c) is the worst performing model whereas the other two models are almost similar in terms of performance. The model can be improved with different model architecture structure, loss functions and other variations of hyperparameters.

References:

- Deep Learning in PyTorch with CIFAR-10 dataset | by Sergio Alves | Medium
- https://qmplus.qmul.ac.uk/pluginfile.php/2182817/mod_resource/intro/Week05_Lab_solutions.ipynb?time=1674650455079