ENPM808Y Neural Networks Assignment 2 Report

Suraj Bandela 115802636

I have designed a Convolutional Neural Network in python 3.6 using PyTorch 1.5 which can perform Image Recognition and Image Classification with slight modifications for later. We are assigned MNIST and CIFAR-10 for Handwritten digit recognition and Image Classification.

MNIST Image recognition Classifier:

I have used TorchVision's inbuilt *DataLoader* to import and load MNIST dataset to local drive to use for the program.

Preprocessing was performed by converting the images to tensors and normalization transformations using global mean of 0.1307 and standard deviation of 0.3081 was applied to these tensors.

A training batch of 100 and 1000 for testing were considered. A sample tensor has [1000, 1, 28, 28] where 1000 is test batch size and 28 x 28 is pixels in grayscale.

Figure 1: Samples from Training data

I have calculated the contribution of each class in the training data and this is what I got

```
total = 0
counter_dict = {0:0, 1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0}

for training_data in trainset:
    Num_images,Num_labels = training_data
    for y in Num_labels:
        counter_dict[int(y)] += 1
        total += 1
print(counter_dict)

for i in counter_dict:
    print(f"{i}: {np.round(counter_dict[i]/total*100, 4)} %")
```

```
{0: 5923, 1: 6742, 2: 5958, 3: 6131, 4: 5842, 5: 5421, 6: 5918, 7: 6265, 8: 5851, 9: 5949}
0: 9.8717 %
1: 11.2367 %
2: 9.93 %
3: 10.2183 %
4: 9.7367 %
5: 9.035 %
6: 9.8633 %
7: 10.4417 %
8: 9.7517 %
9: 9.915 %
```

Figure 2: Share of individuals classes in Dataset

I have used the Pytorch framework to build my Network model for the assignment. My model consists for MNIST consists of Two 2D convolutional layers with kernel size 5. These are followed by maxpooling and Two Fully connected networks. For activation function I have selected ReLU and for regularization two dropout layers have been used. I have used SGD optimizer with momentum 0.5 and learning rate of 0.001.

I have utilized GPU to transfer the tensors from CPU which has improved the speed of training. A training and testing functions have been built. I have initialized the training with 15 epochs where Loss appeared to reduce and Accuracy of Test Set improved.

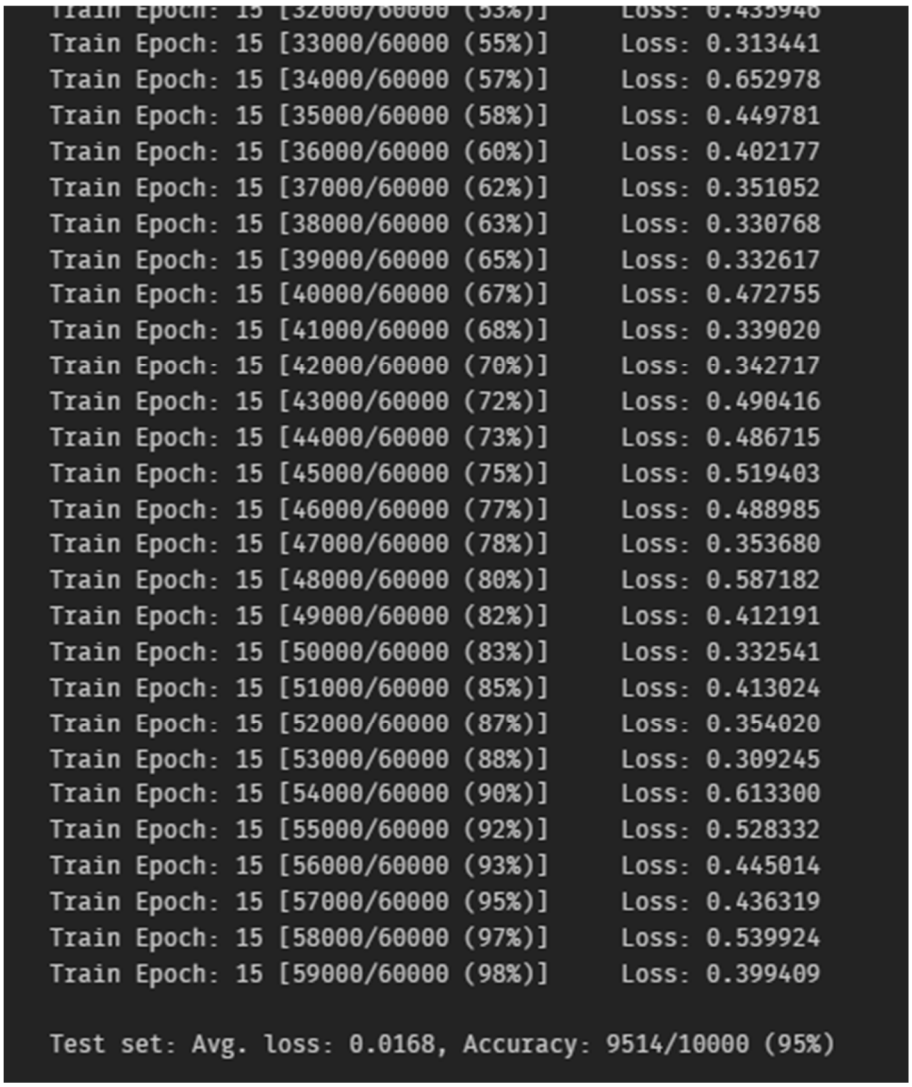

Figure 3: Training and Testing Logs

Finally, I plotted the Number of datasets vs Negative log likelihood to show the training loss.

I have achieved 93% accuracy for MNIST data.

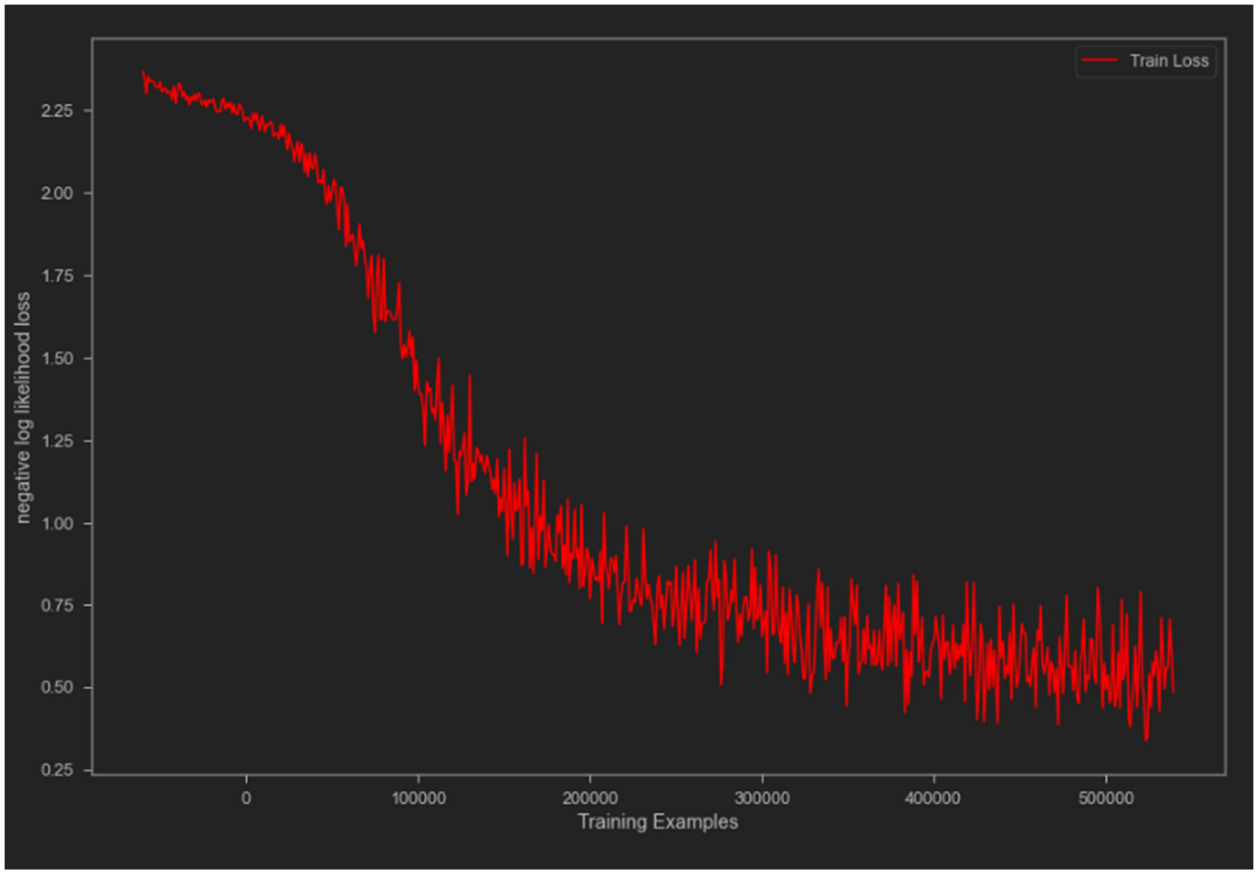

Figure 4: Training Examples vs Negative Log likelihood loss

Image Classification on CIFAR-10 using CNN

For this task I have used Pytorch 1.5 with CUDA 9.0 support. I have used Pytorch for the support it has for handling big image data files.

I have used inbuilt TorchVision to import and download CIFAR-10 dataset to local drive and work on it for further implementation.

The imported RAW files where in PIL format. I have converted them to PyTorch Tensors. Then I normalized the Tensors with 0.5 mean and 0.5 standard deviation for three channels of RGB. The data was converted from [0,1] to [-1,1].

A random sample from the training set was checked the integrity of image files which were in Tensor form.

```
sampledata = iter(trainset)
images, labels = sampledata.next()

fig = plt.figure(figsize=(15,20))
for i in np.arange(20):
    ax = fig.add_subplot(4,5, i+1, xticks = [], yticks = [])
    view_sample_image(images[i])
    ax.set_title(classes[labels[i]])
```

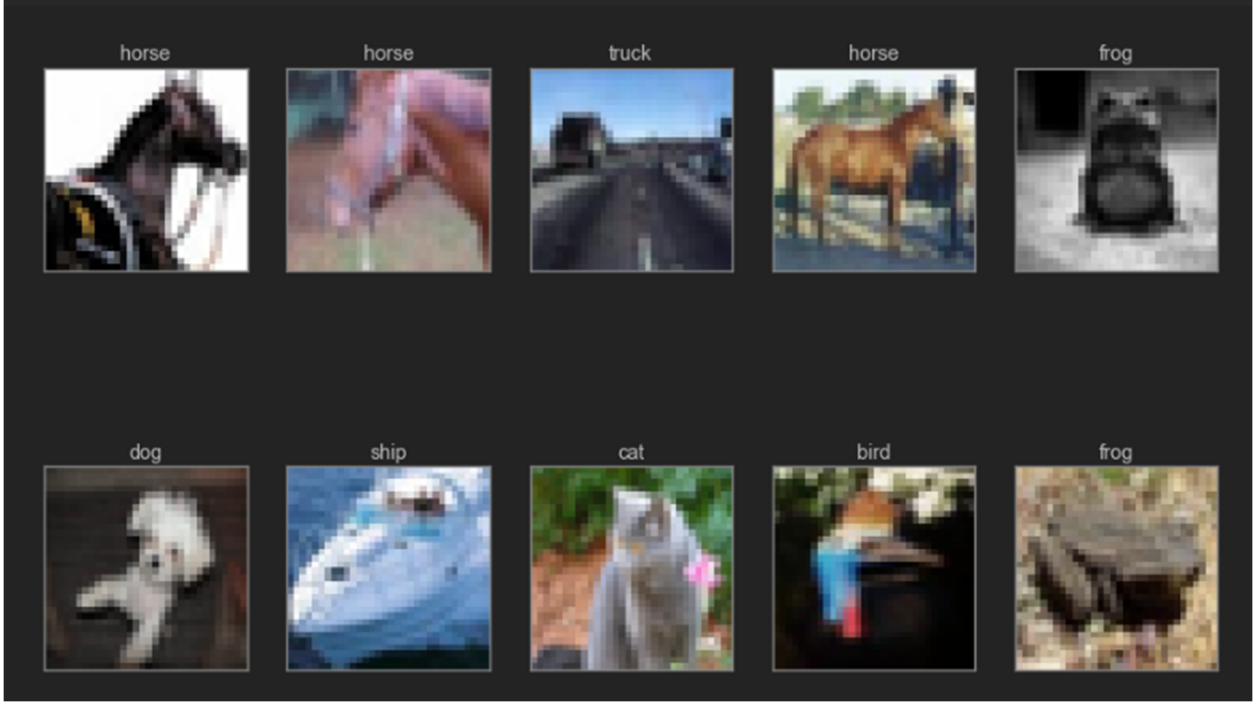

Figure 5: Training sample data

The image and label tensor are of the following dimensions

```
Shape of the images torch.Size([20, 3, 32, 32])
Shape of the labels torch.Size([20])
```

I have the built the CNN with the following elements layout

Input > Conv (ReLU) > MaxPool > Conv (ReLU) > MaxPool > FC (ReLU) > FC (ReLU) > FC (SoftMax) > 10 outputs

Where, Conv is a convolutional layer, ReLU is the activation function, MaxPool is a pooling layer, FC is a fully connected layer and SoftMax is the activation function of the output layer.

The total network has 62006 parameters.

I have used Cross entropy loss function and SGD optimizer.

Training and Testing functions are constructed, and the processed training data is sent to training. A training batch size of 4 is used over 15 epochs.

Testing is conducted on the test dataset and I have achieved 62.47% Accuracy

Finally, I plotted the Number of datasets vs Negative log likelihood to show the training loss.

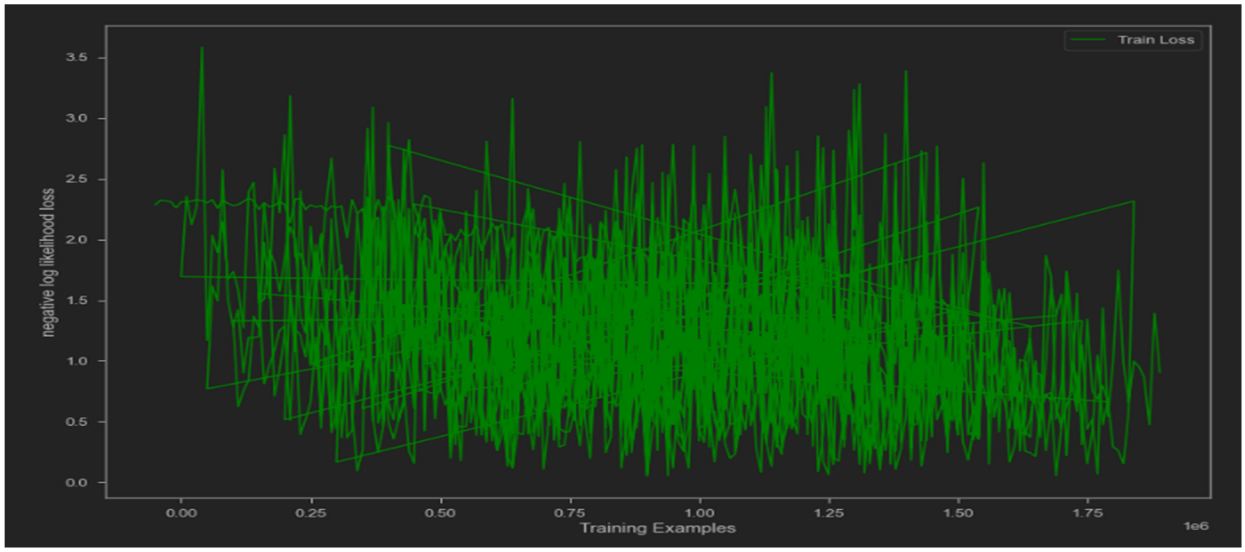

Figure 6: Training Examples vs Negative Log likelihood loss

I have calculated class accuracy for each category

```
Accuracy of plane : 73.200000
Accuracy of   car : 79.100000
Accuracy of  bird : 54.100000
Accuracy of   cat : 40.400000
Accuracy of  deer : 56.300000
Accuracy of   dog : 65.900000
Accuracy of  frog : 68.300000
Accuracy of horse : 61.700000
Accuracy of  ship : 71.600000
Accuracy of truck : 54.100000
```

Confusion Matrix was constructed and generated.

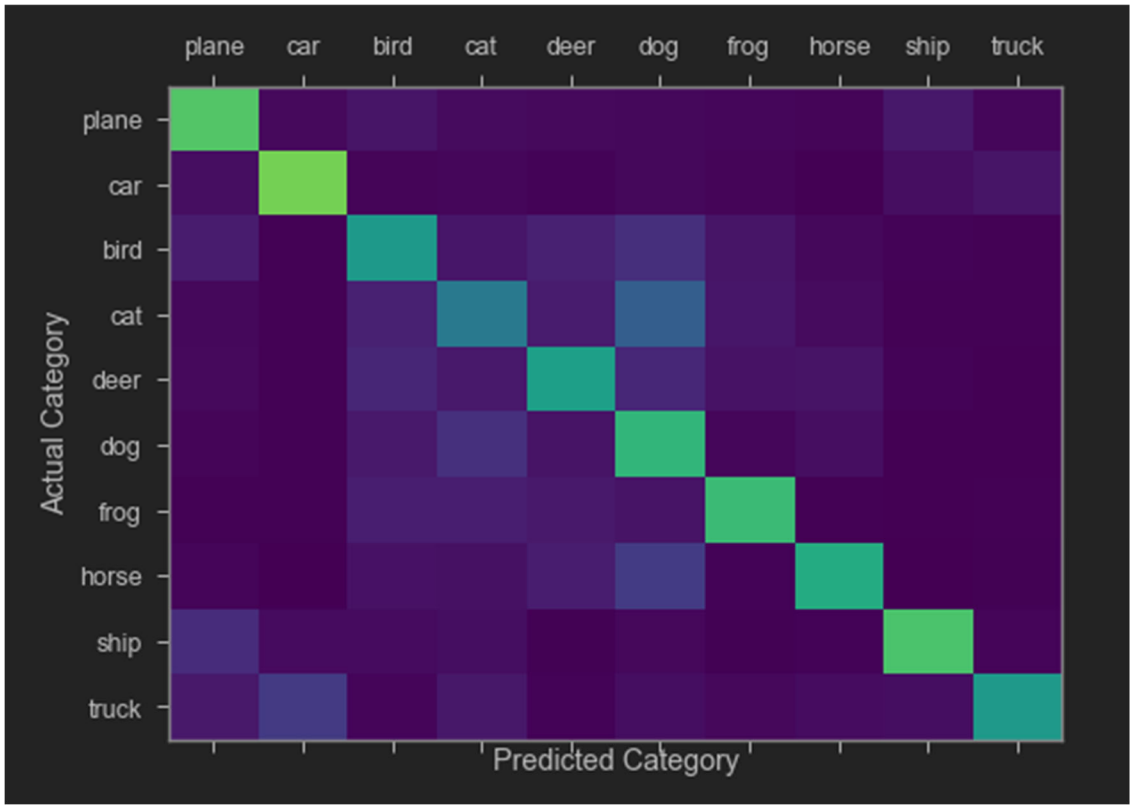

Figure 7: Actual category vs Predicted Category Confusion Matrix plot

actual/predicted	plane	car	bird	cat	deer	dog	frog	horse	ship	truck
plane	732	27	55	30	26	20	18	12	64	16
	0.732	0.027	0.055	0.03	0.026	0.02	0.018	0.012	0.064	0.016
car	39	791	14	18	10	20	12	3	38	55
	0.039	0.791	0.014	0.018	0.01	0.02	0.012	0.003	0.038	0.055
bird	75	4	541	62	92	135	55	23	9	4
	0.075	0.004	0.541	0.062	0.092	0.135	0.055	0.023	0.009	0.004
cat	22	6	92	404	76	298	62	29	6	5
	0.022	0.006	0.092	0.404	0.076	0.298	0.062	0.029	0.006	0.005
deer	27	5	108	69	563	111	50	54	11	2
	0.027	0.005	0.108	0.069	0.563	0.111	0.05	0.054	0.011	0.002
dog	14	5	70	140	52	659	18	37	3	2
	0.014	0.005	0.07	0.14	0.052	0.659	0.018	0.037	0.003	0.002
frog	5	7	84	85	69	52	683	7	3	5
	0.005	0.007	0.084	0.085	0.069	0.052	0.683	0.007	0.003	0.005
horse	14	2	50	45	81	175	10	617	0	6
	0.014	0.002	0.05	0.045	0.081	0.175	0.01	0.617	0.0	0.006
ship	128	29	30	38	7	21	7	9	716	15
	0.128	0.029	0.03	0.038	0.007	0.021	0.007	0.009	0.716	0.015
truck	67	171	19	64	10	37	20	33	38	541
	0.067	0.171	0.019	0.064	0.01	0.037	0.02	0.033	0.038	0.541

Figure 8: Confusion Matrix

Conclusion:

I have learned how Convolution Networks works. I should work and check the popular CIFAR-10 and CIFAR-100 architectures to improve my model's Accuracy and reduce Loss.