CSE 676 B Deep Learning Assignment 0

Data Analysis, ML models and pytorch

Charan Kumar Nara - 50545001

Part I

Dataset Name: Popular Baby Names dataset

Dataset Link: https://catalog.data.gov/dataset/popular-baby-names

Short Description:

Popular Baby Names by Sex and Ethnic Group Data were collected through civil birth registration. Each record represents the ranking of a baby name in the order of frequency. Data can be used to represent the popularity of a name.

Statistics:

	Year of Birth	Count	Rank
count	57582.000000	57582.000000	57582.000000
mean	2013.283352	33.929596	57.066114
std	2.056076	39.027451	25.519447
min	2011.000000	10.000000	1.000000
25%	2012.000000	13.000000	38.000000
50%	2013.000000	20.000000	59.000000
75%	2014.000000	36.000000	78.000000
max	2019.000000	426.000000	102.000000

Number of samples:

57582 Entries with 6 features

Data Preprocessing and Cleaning:

Maintaining string format:

Handling the categorical data by maintaining all the letters as uppercase. As we found some inconsistency in various entries.

Converting String data to categorical:

Converted the string valued features Rank and Count to categorical features.

Handling Categorical Data:

One Hot Encoding:

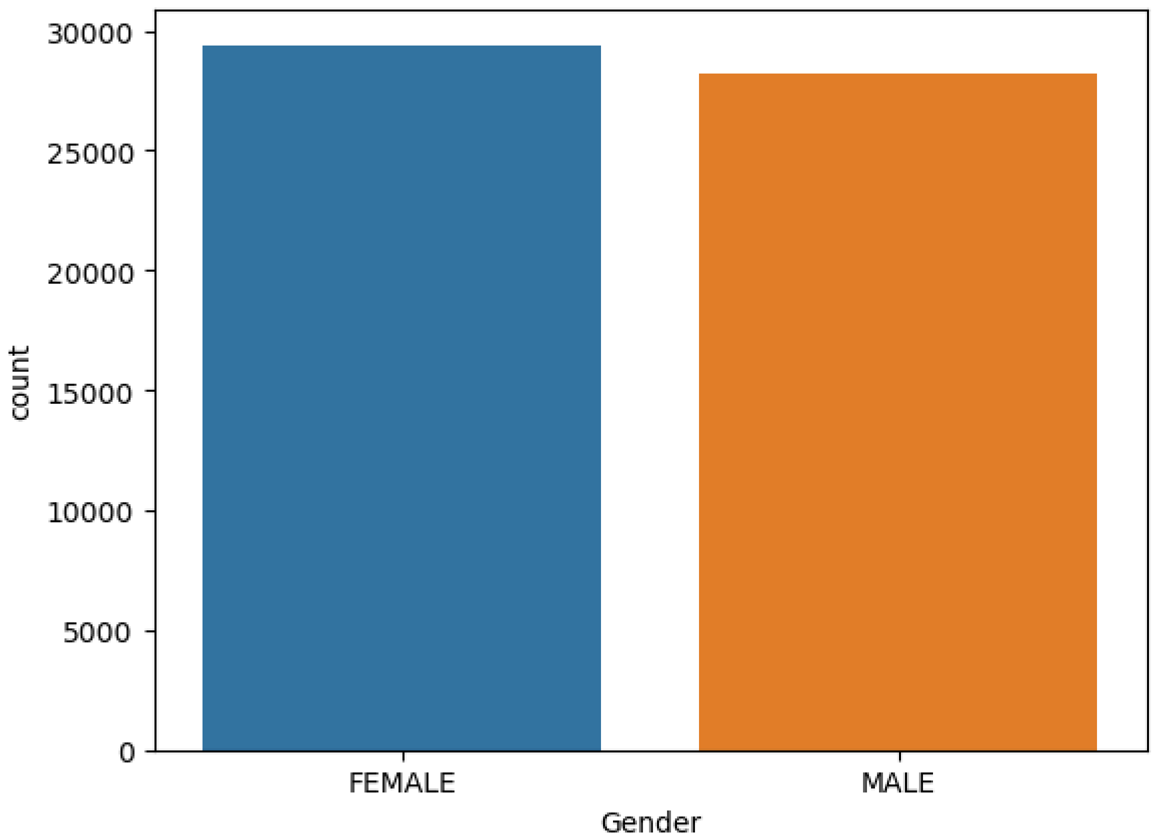
Created Dummy features for categorical attributes. Generally we create dummy variables or one hot encoding to categorical features in order to feed the data to the machine learning model. As some of the algorithms consume only the numerical data.

Scaling the features:

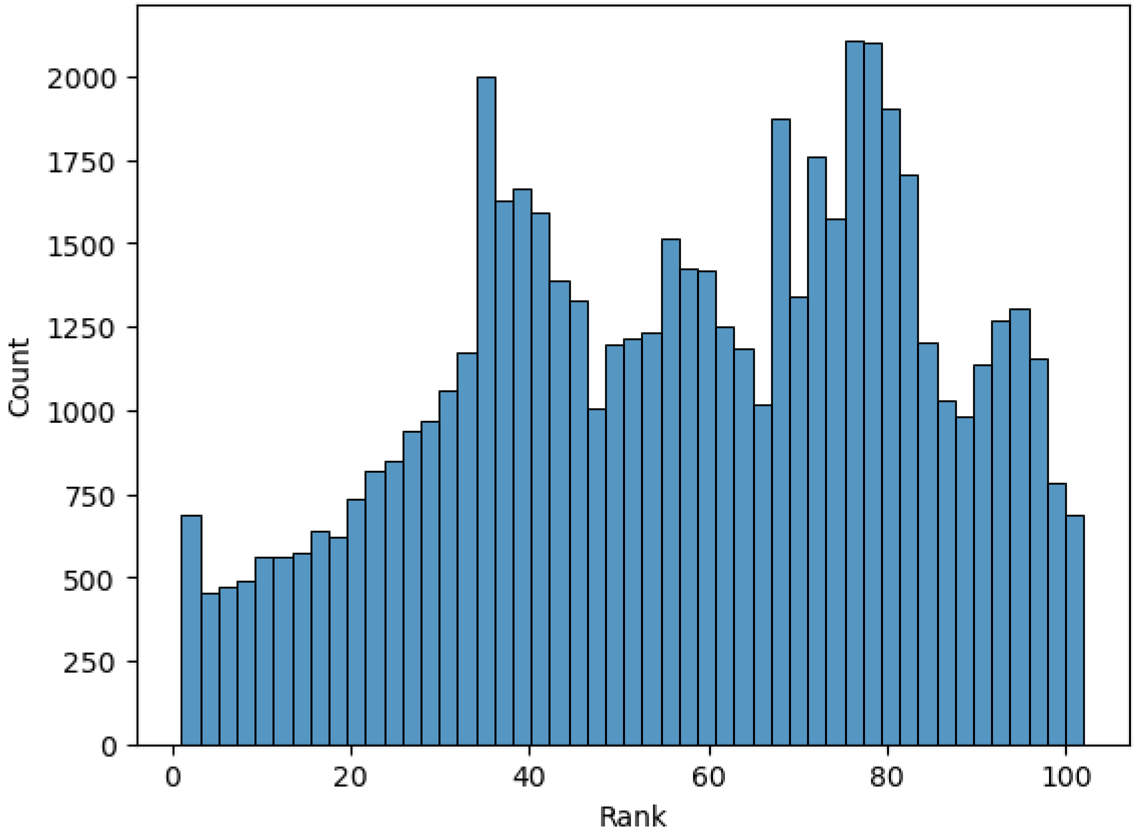
Scaling the features can be really useful for training a machine learning model because scaling maintains uniformity in the values present in the dataset. Which gives equal scale for each feature. Scaling used here is Standard scaler, which maintains overall mean close to 0 and standard deviation as 1.

Correlation Matrix:

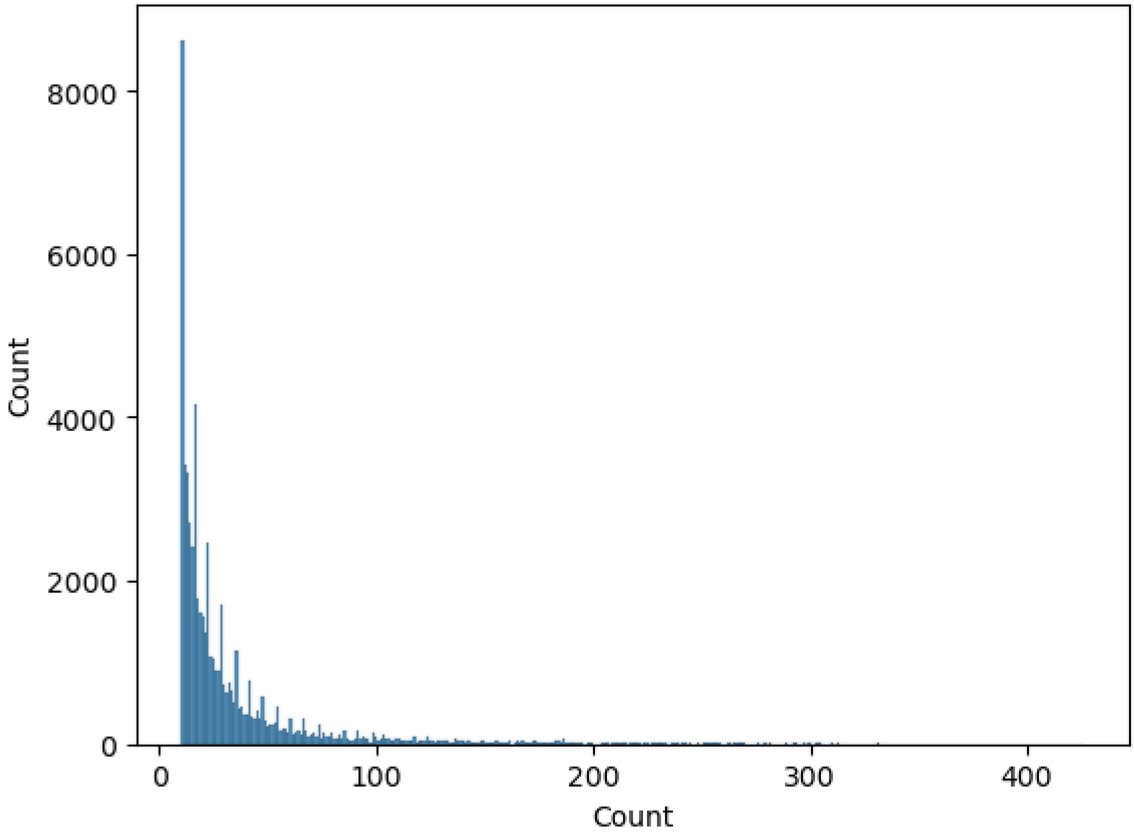
Data Visualization:



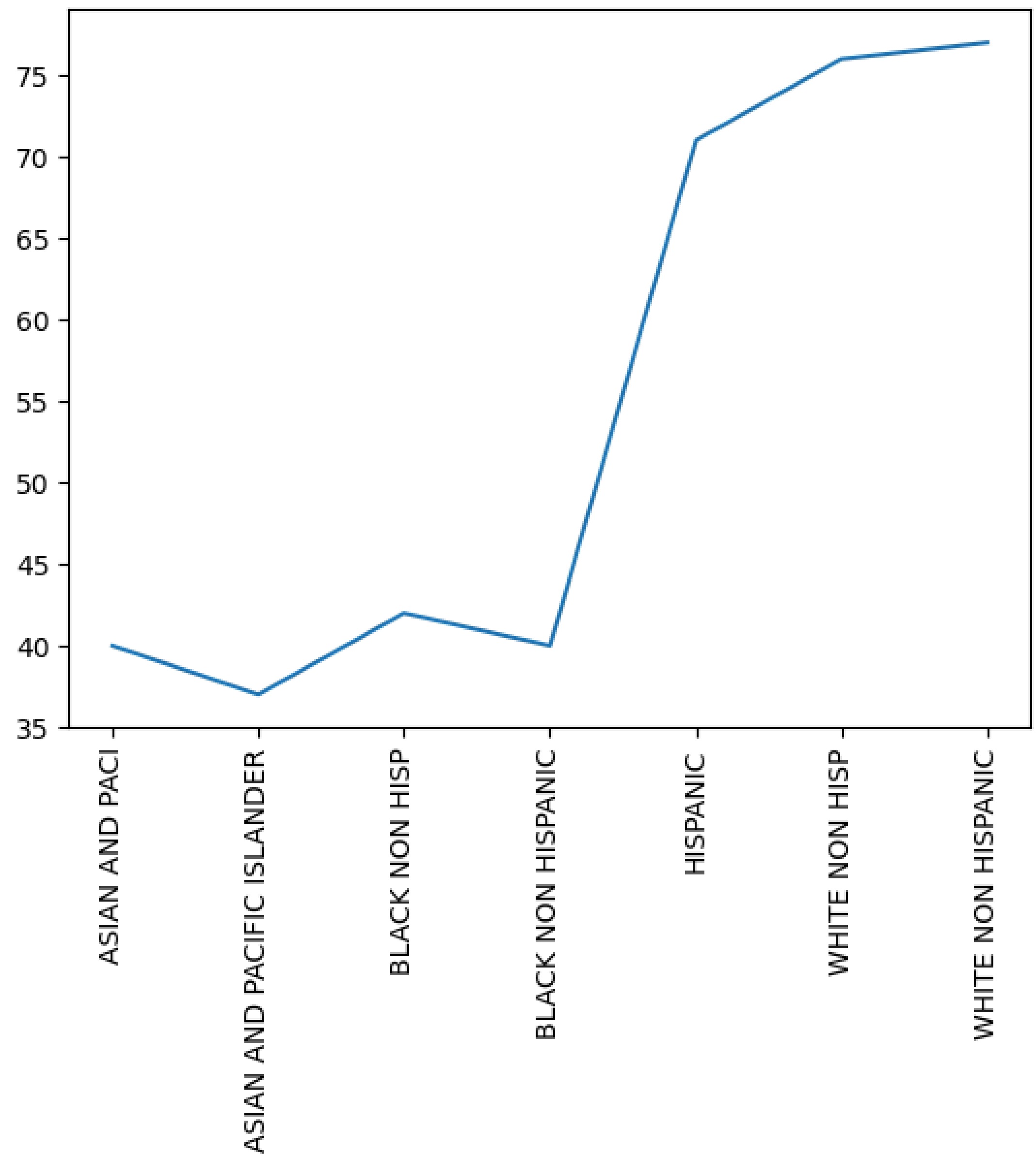
Observation: Here we can observe, the feature gender is properly balanced with male and female as the categories present. This is why we are most probably choosing gender as the target variable.



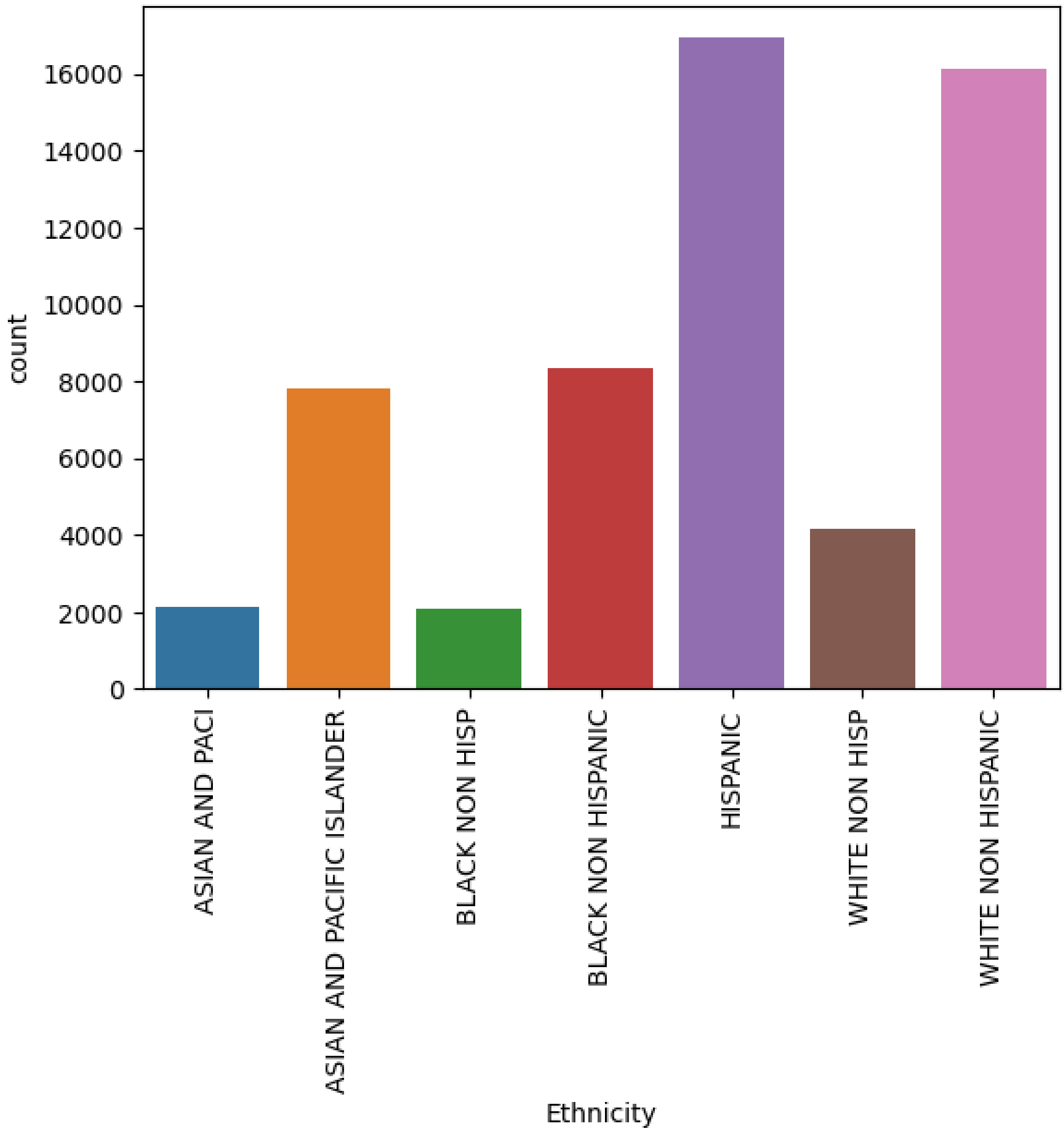
Observation: Here's the histogram of the target variable rank. We can observe that the graph is not a normal or uniform distribution. It is slightly right skewed.



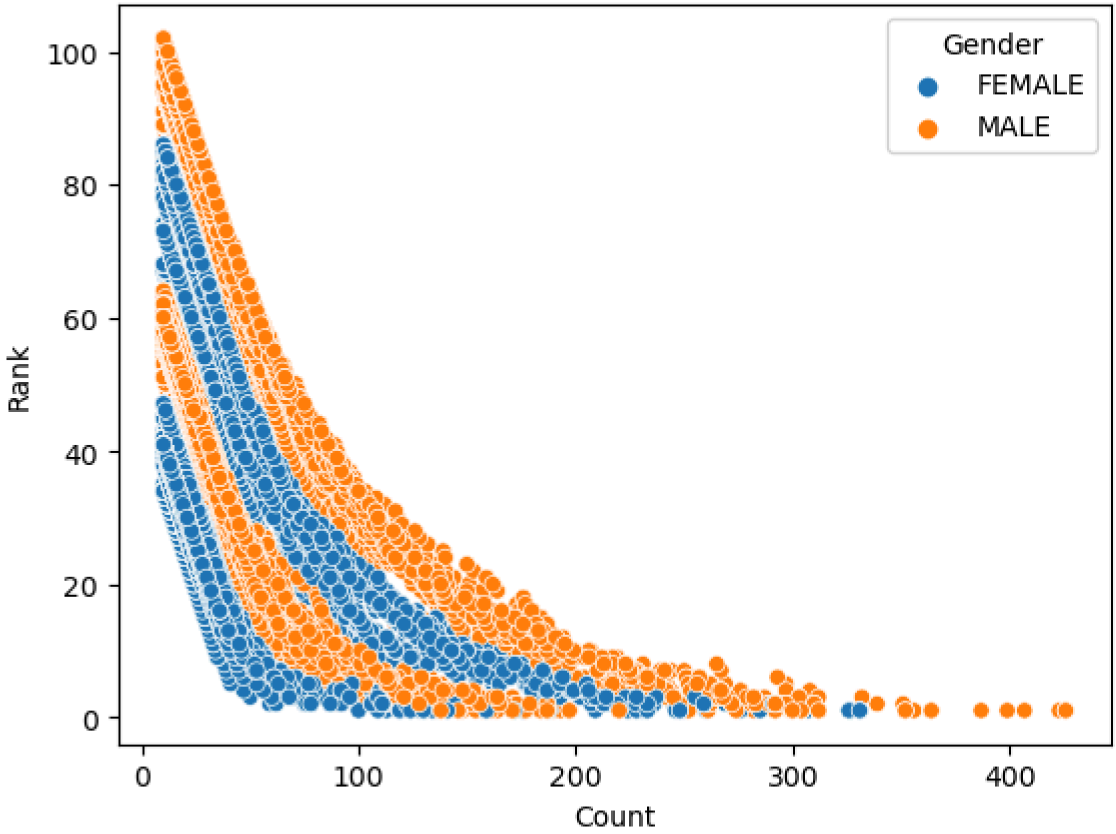
Observation: The count feature is left skewed clearly. Hence even if we have to impute any missing data, we need to impute or handle it with median.



Observation; The above graph depicts the line plot grouping the ethnic groups of people representing their median rank for each group.



Observation: The above graph depicts the count plot of various unique ethnic people present in the dataset representing the number of records present in each group.



Observation: The above scatter plot represents the graph between count and the rank with highlighting the target variable male and female. We can clearly observe a pattern, where the male and female gender groups lie with given rank and the count.

Machine Learning models used:

Logistic Regression:

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	5865
1	1.00	1.00	1.00	5652
accuracy			1.00	11517
macro avg	1.00	1.00	1.00	11517
weighted avg	1.00	1.00	1.00	11517

Random Forest:

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	5865
1	1.00	1.00	1.00	5652
accuracy			1.00	11517
macro avg	1.00	1.00	1.00	11517
weighted avg	1.00	1.00	1.00	11517

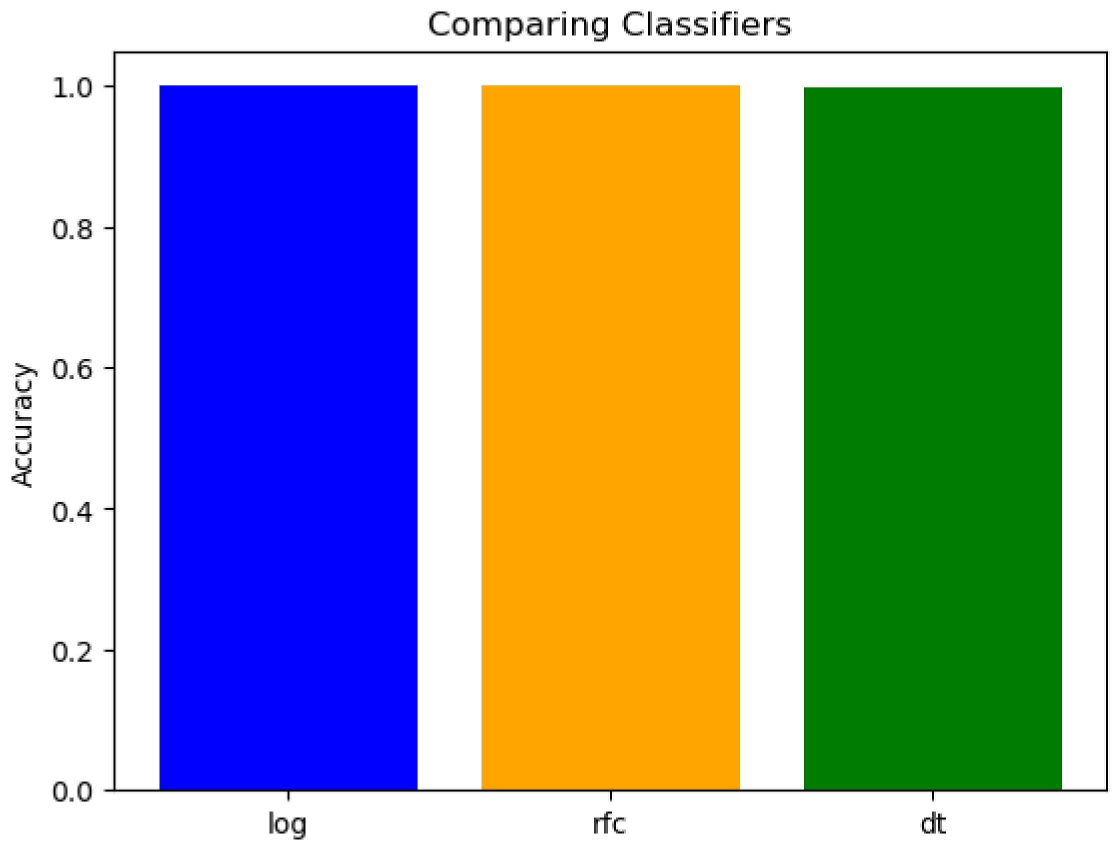
Decision Tree:

Classification Report:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	5865
1	1.00	0.99	1.00	5652
accuracy			1.00	11517
macro avg	1.00	1.00	1.00	11517
weighted avg	1.00	1.00	1.00	11517

Comparison Graph for Accuracy:

```
Logistic Regression Accuracy:  0.9993922028305983
Random Forest Accuracy:  1.0
Decision Tree Accuracy:  0.9967873578188764
Logistic Regression Loss: 0.0044573590738987805
```



Log - logistic regression
Rfc - Random forest classifier
Dt - Decision Tree classifier

From the three classifiers we have chosen Decision Tree classifier as a good decent classifier.

Shallow NN architecture:

```
ShallowNN(
  (fc1): Linear(in_features=1956, out_features=100, bias=True)
  (fc2): Linear(in_features=100, out_features=1, bias=True)
)
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
ShallowNN                                --
├─Linear: 1-1                            195,700
├─Linear: 1-2                            101
=================================================================
Total params: 195,801
Trainable params: 195,801
Non-trainable params: 0
=================================================================
```

Accuracy and Loss for ShallowNN:

Three Different setups with learning rate and Optimizer and number of Layers:

Tuning Optimizer:

Optimizer	Accuracy
Adam	99.89
SGD	99.89
Adamax	99.97

Tuning Learning Rate:

Learning Rate	Accuracy
0.01	99.89
0.001	99.99
0.1	51.06

Tuning Layers count:

# of Layers	Accuracy
ShallowNN (1 hidden layer)	99.89
2 hidden layers	100
3 hidden layers	100

The best model we got from hyperparameter tuning is with tuning 3 hidden layers.

Part III

OctMNIST classification

About dataset:

The OCTMNIST is based on a prior dataset of 109,309 valid optical coherence tomography (OCT) images for retinal diseases. Each example is a 28x28 image, associated with a label from 4 classes.

Neural Net architecture:

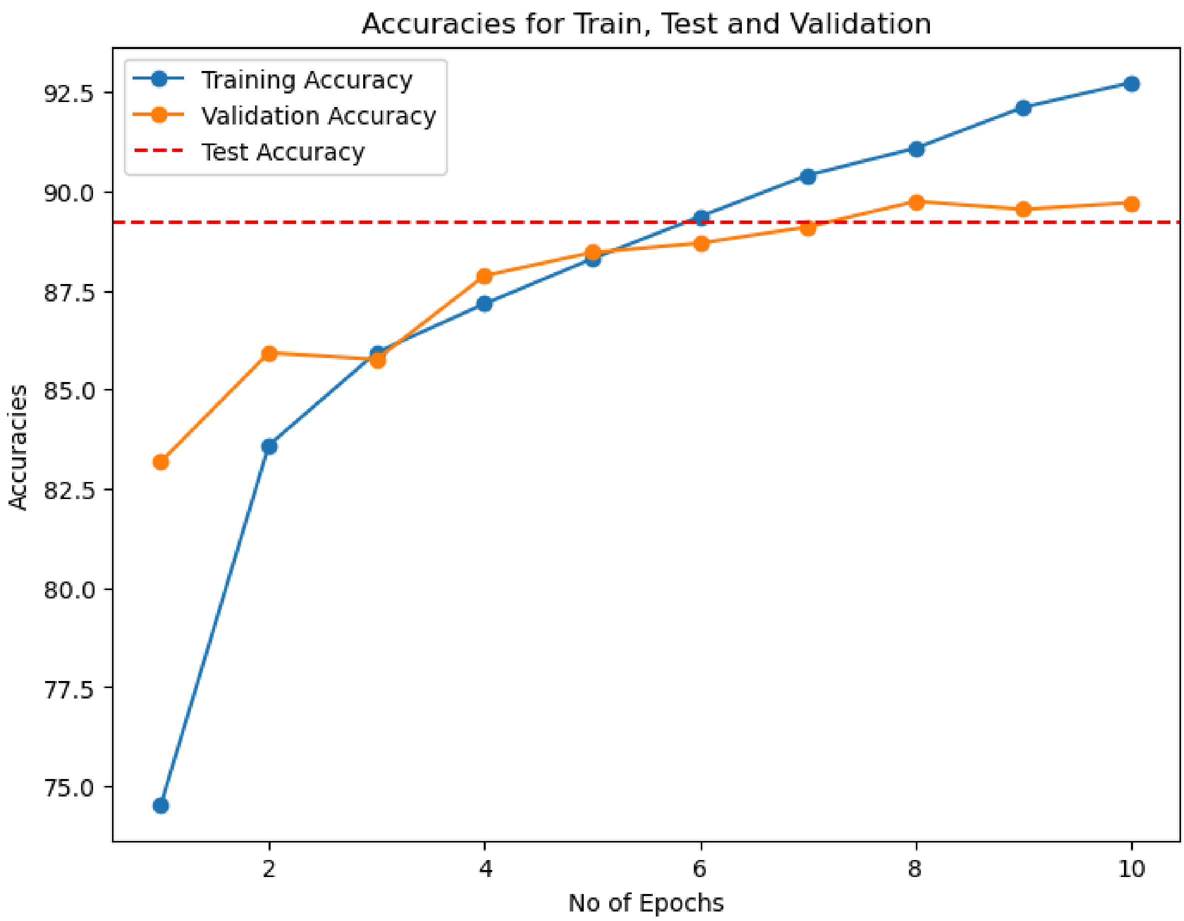
```
class cnn(nn.Module):
    def __init__(self):
        super(cnn, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(64 * 7 * 7, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 4)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 7 * 7)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

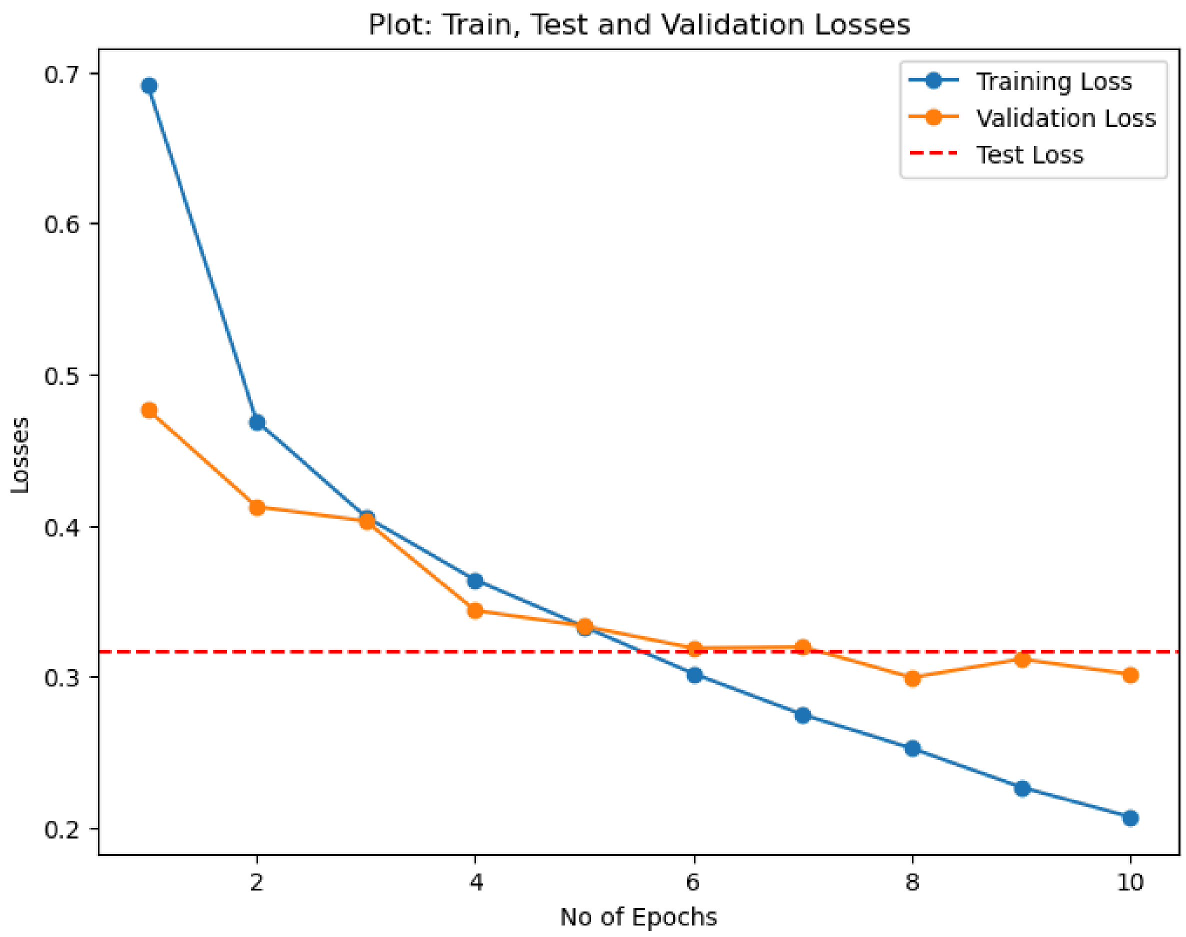
```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
cnn                                      --
├─Conv2d: 1-1                            320
├─MaxPool2d: 1-2                         --
├─Conv2d: 1-3                            18,496
├─Linear: 1-4                            803,072
├─Linear: 1-5                            32,896
├─Linear: 1-6                            516
=================================================================
Total params: 855,300
Trainable params: 855,300
Non-trainable params: 0
=================================================================
```

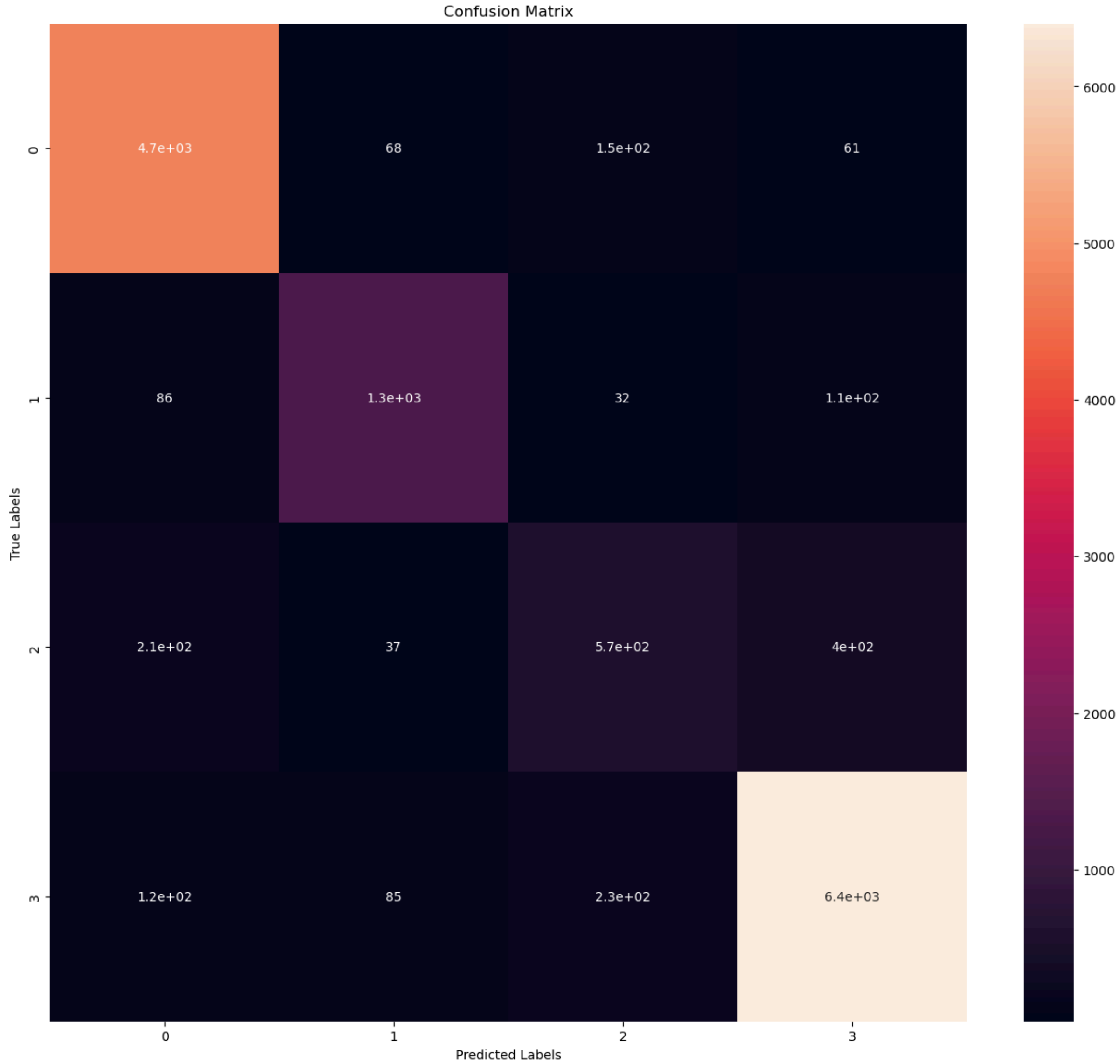
Basic model results without application of techniques:

```
Accuracy:  89.22929631402585
Confusion Matrix:  [[4734   68  146   61]
 [  86 1342   32  109]
 [ 211   37  572  398]
 [ 116   85  226 6400]]
Precision:  82.50720673942583
Recall:  80.18743101843393
F1 Score:  81.17696302486226
```

Basic model graphs for results:




Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Test Loss
Losses
No of Epochs
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10



Techniques to prevent overfitting:

a) L2 Regularization:

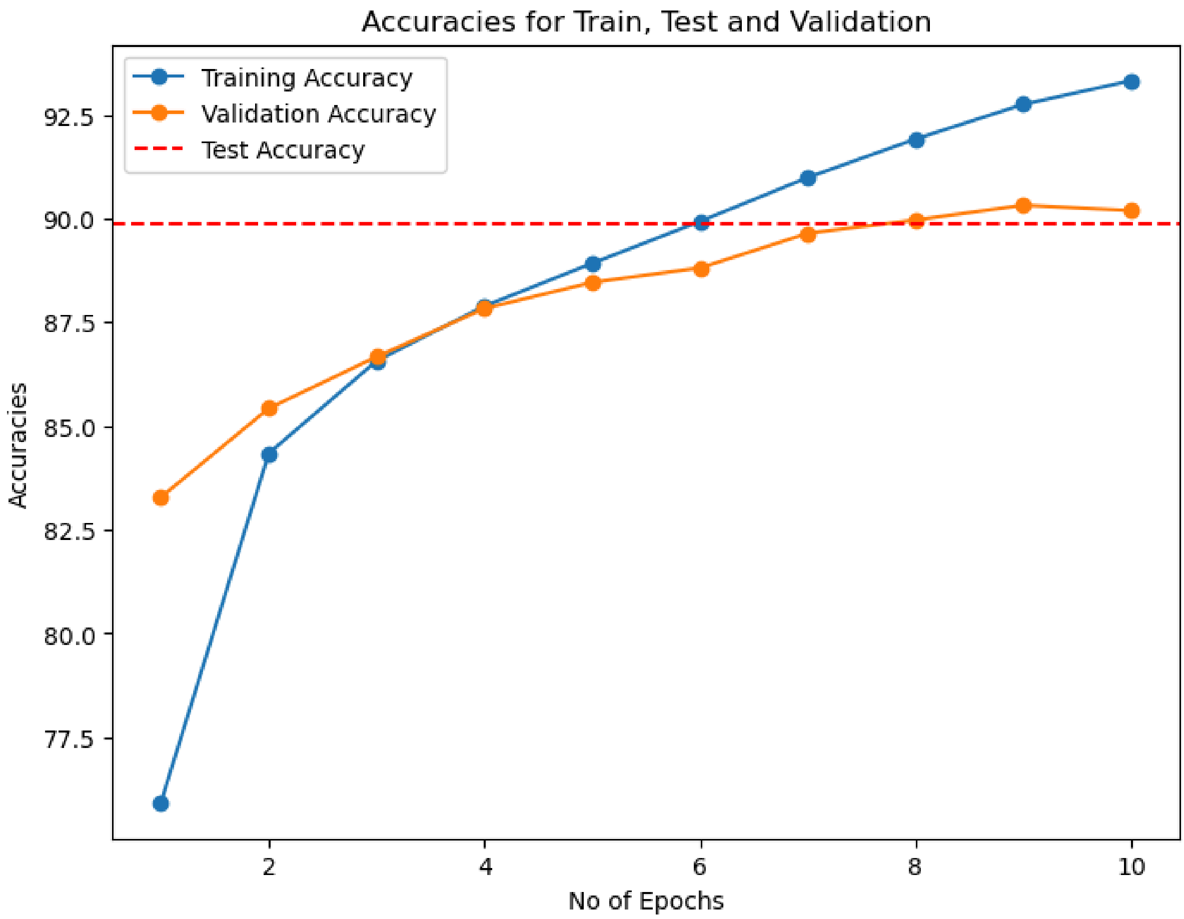
Weight_decay = 0.000001

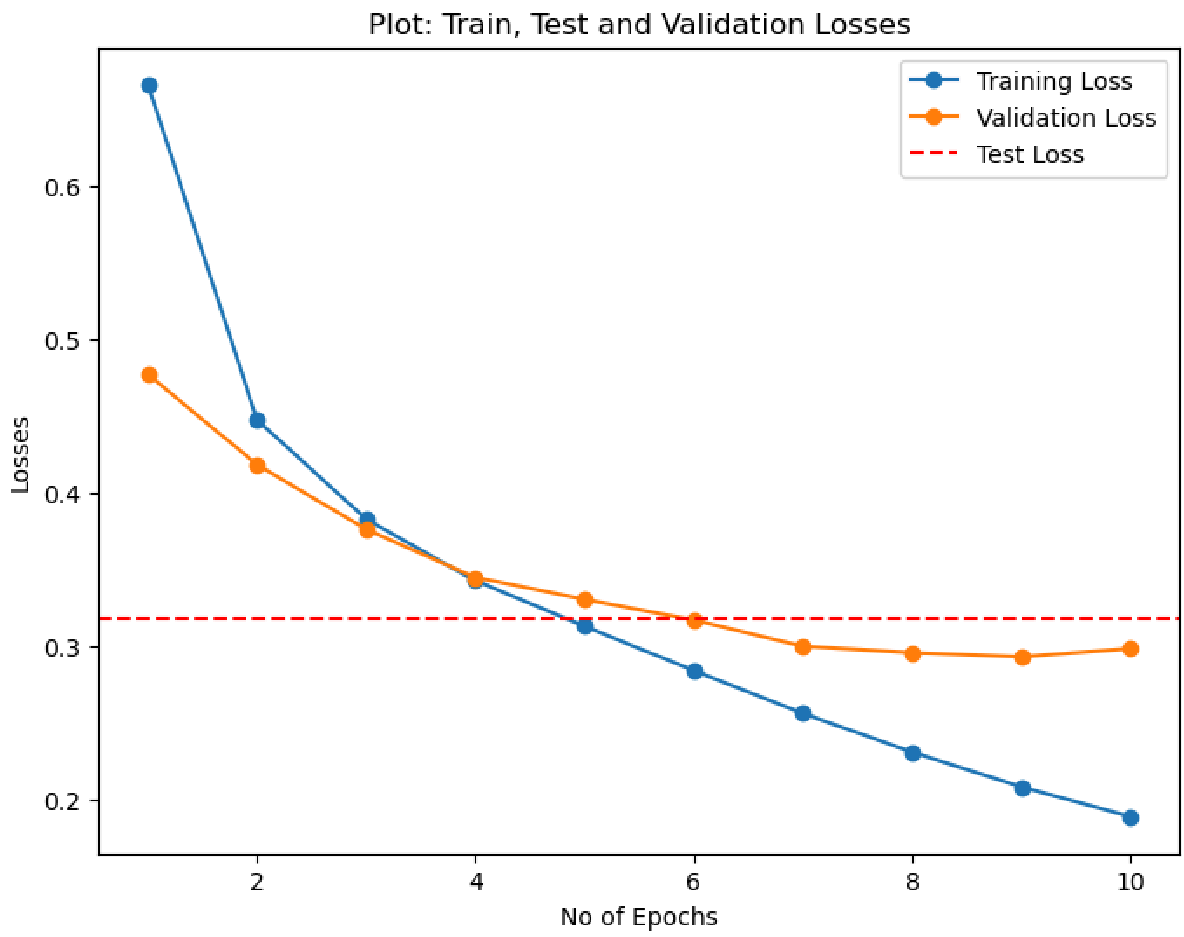
L2 Regularization results:

Impact on the results by using L2 regularization technique:

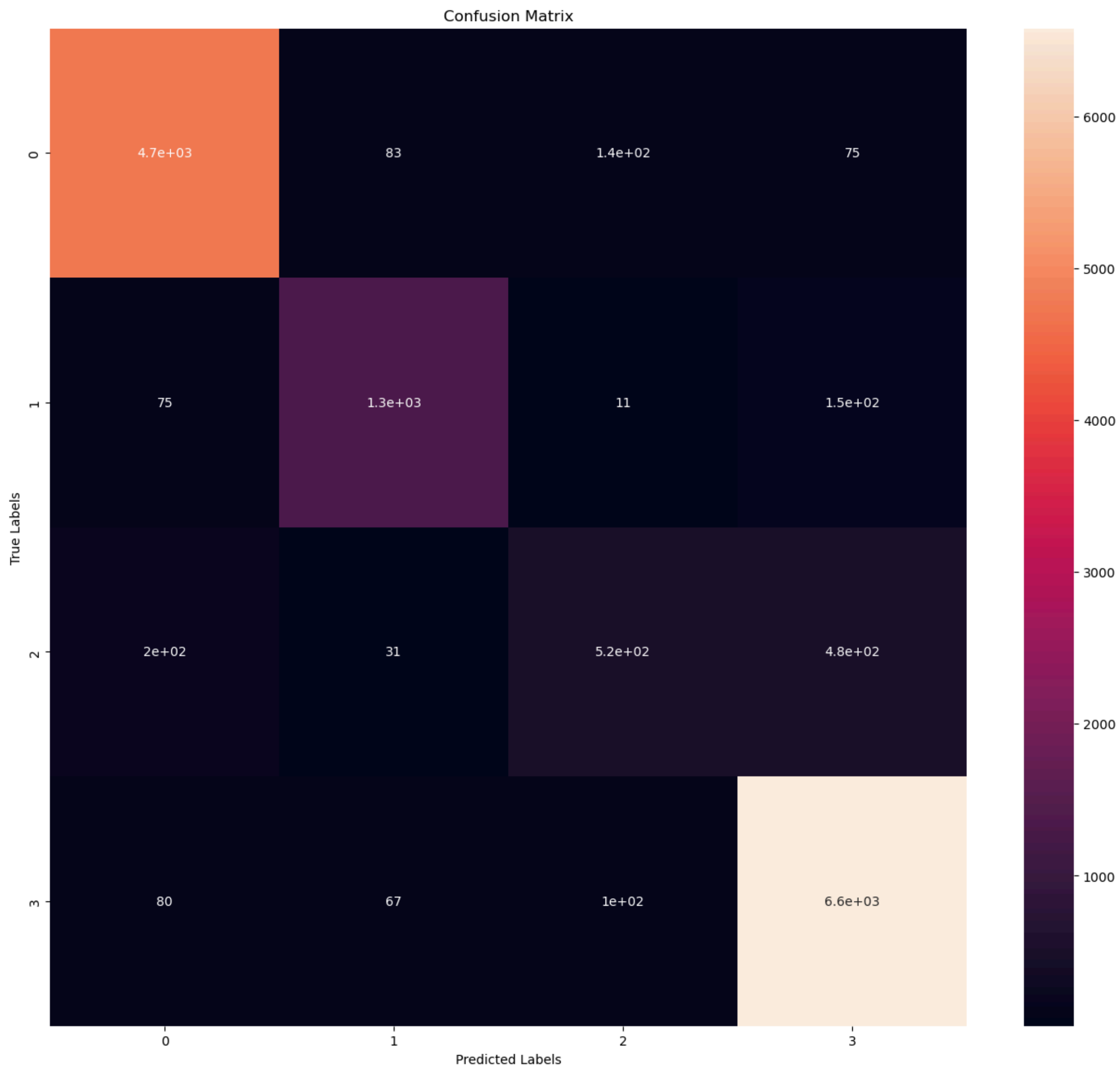
```
Accuracy:  89.86528072215005
Confusion Matrix:  [[4712   83  139   75]
 [  75 1335   11  148]
 [ 197   31  515  475]
 [  80   67  101 6579]]
Precision:  84.68754258981531
Recall:  79.45162748555401
F1 Score:  81.32821278882871
```

L2 Regularization graphs for the results:





Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Test Loss
Losses
0.2
0.3
0.4
0.5
0.6
No of Epochs
2
4
6
8
10



b) Dropout

Neural Net architecture with dropouts

```python
# Dropout

class cnndrop(nn.Module):
    def __init__(self):
        super(cnndrop, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(64 * 7 * 7, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 4)
        self.dropout1 = nn.Dropout(0.3)
        self.dropout2 = nn.Dropout(0.3)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 7 * 7)
        x = F.relu(self.fc1(x))
        x = self.dropout1(x)
        x = F.relu(self.fc2(x))
        x = self.dropout2(x)
        x = self.fc3(x)
        return x
```

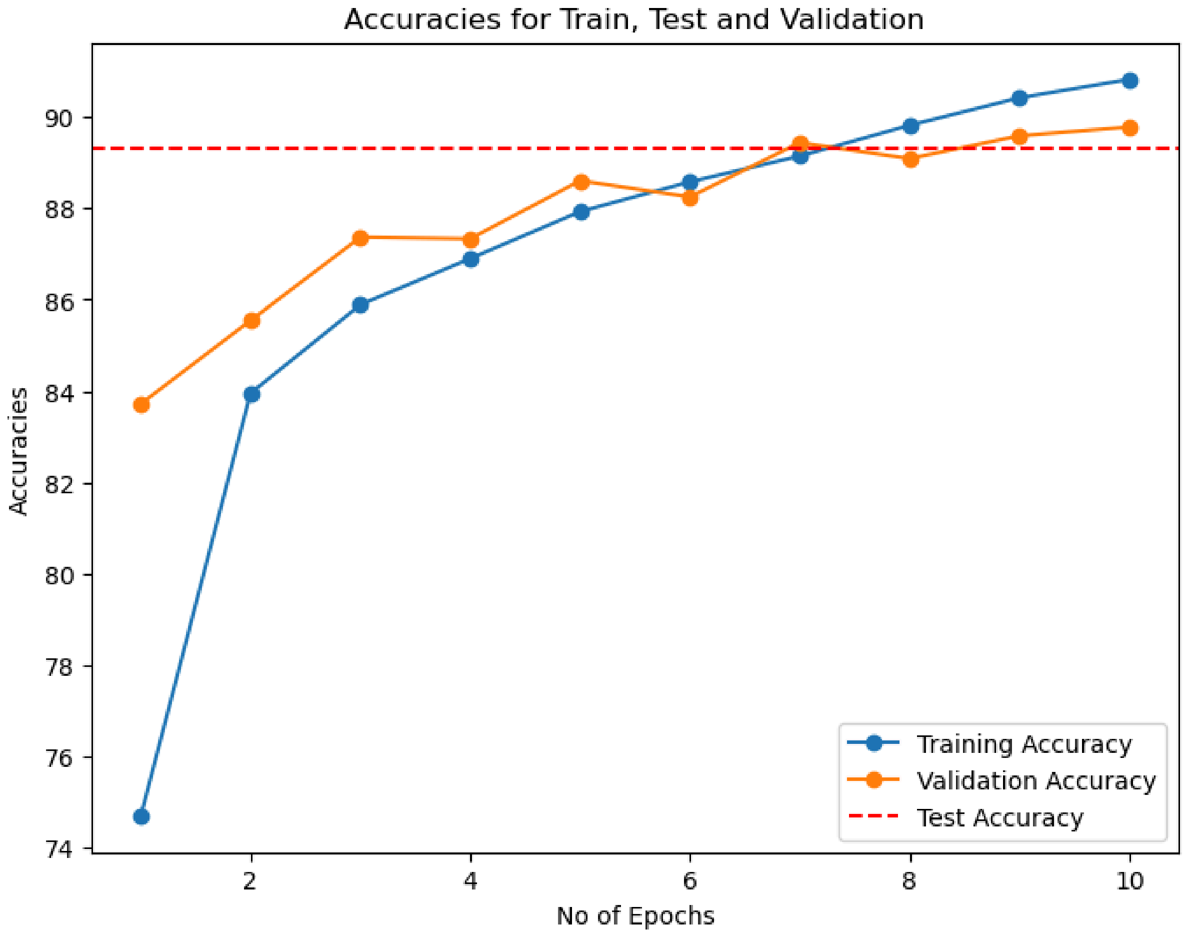
```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
cnn                                      --
├─Conv2d: 1-1                            320
├─MaxPool2d: 1-2                         --
├─Conv2d: 1-3                            18,496
├─Linear: 1-4                            803,072
├─Linear: 1-5                            32,896
├─Linear: 1-6                            516
=================================================================
Total params: 855,300
Trainable params: 855,300
Non-trainable params: 0
=================================================================
```

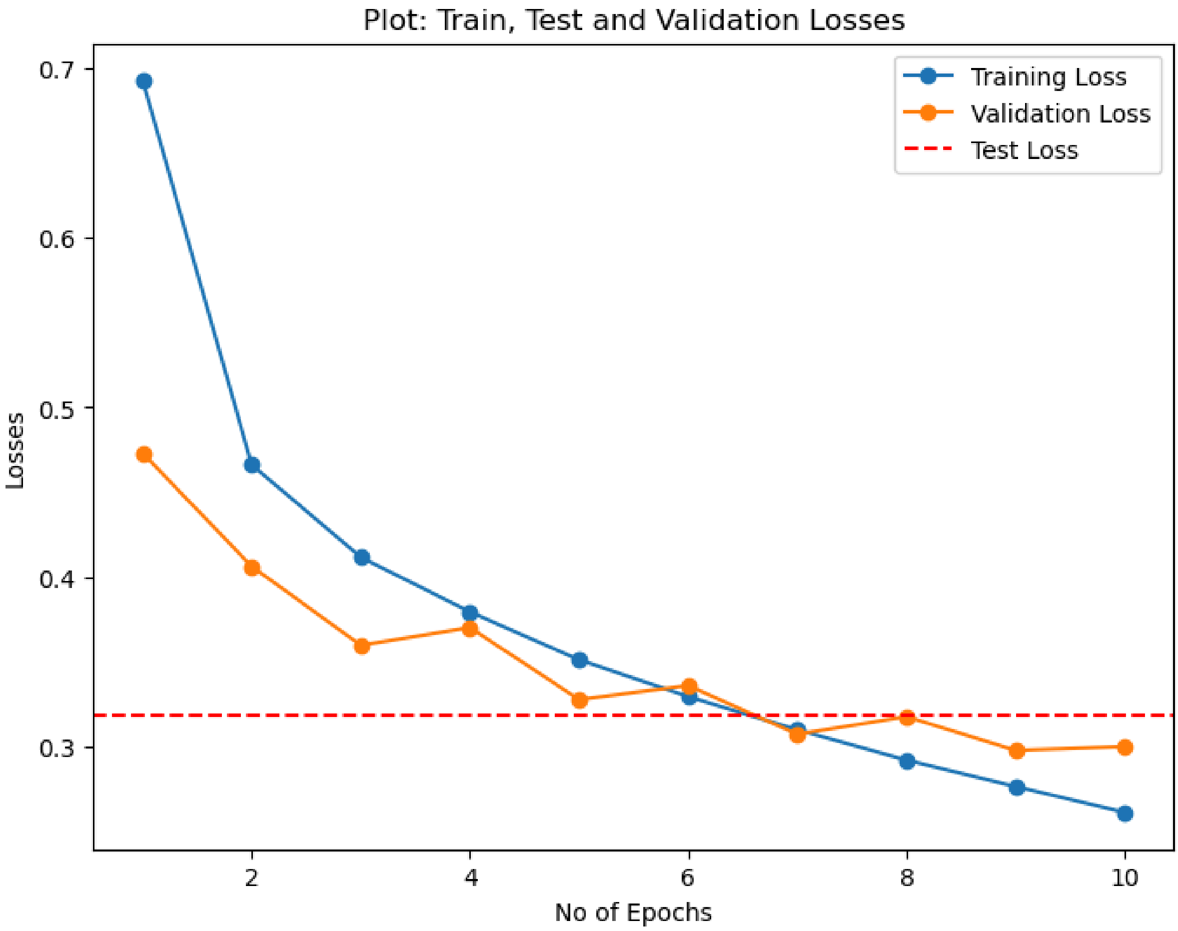
Results for Dropouts technique:

Impact on the results by using Dropouts technique with dropout value 0.3.

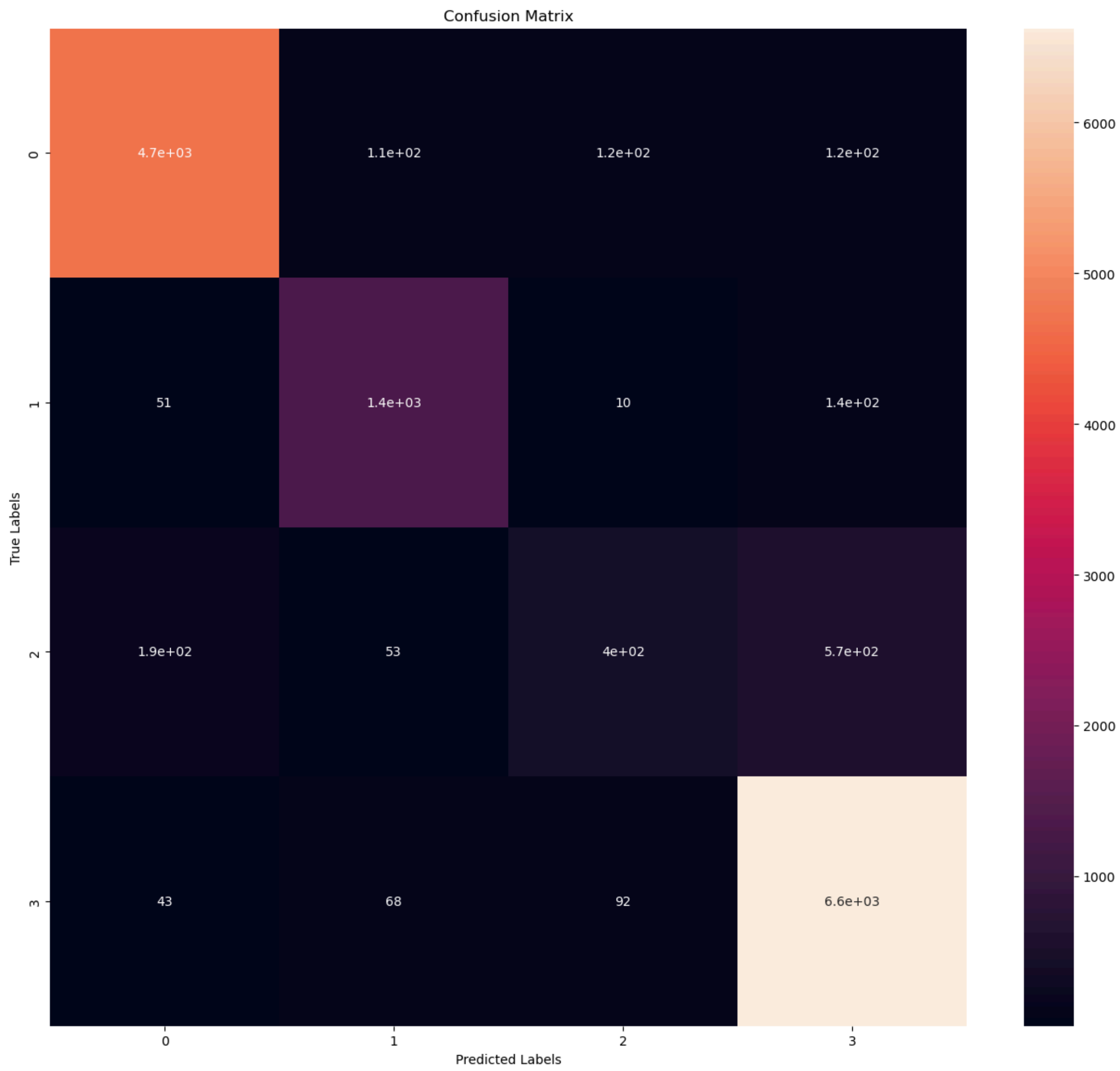
```
Accuracy:  89.3045202762771
Confusion Matrix:  [[4665  106  123  115]
 [  51 1369   10  139]
 [ 194   53  401  570]
 [  43   68   92 6624]]
Precision:  83.2391049606101
Recall:  77.58368145857578
F1 Score:  79.11564829672986
```

Dropout technique graphs for the results:





Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Test Loss
0.7
0.6
0.5
0.4
0.3
Losses
2
4
6
8
10
No of Epochs



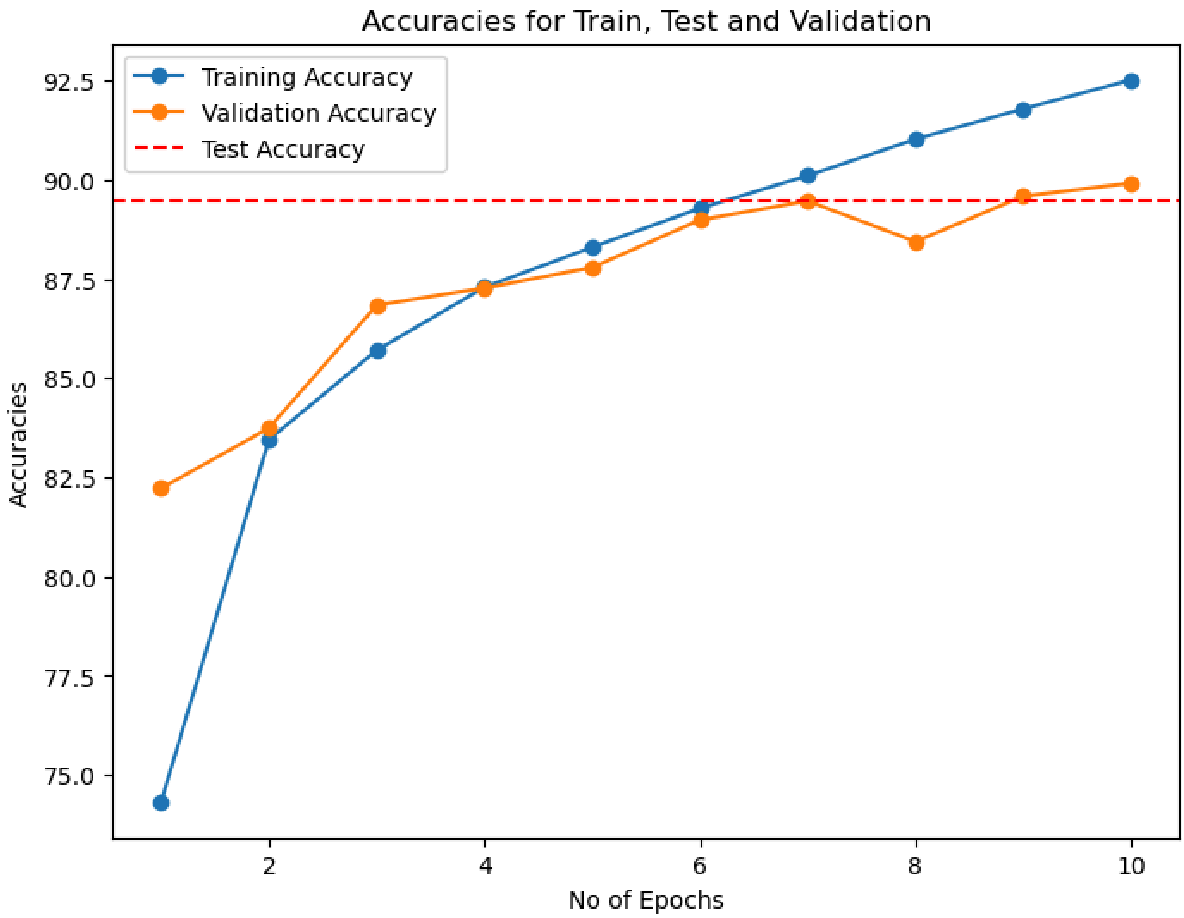
c) Early stopping

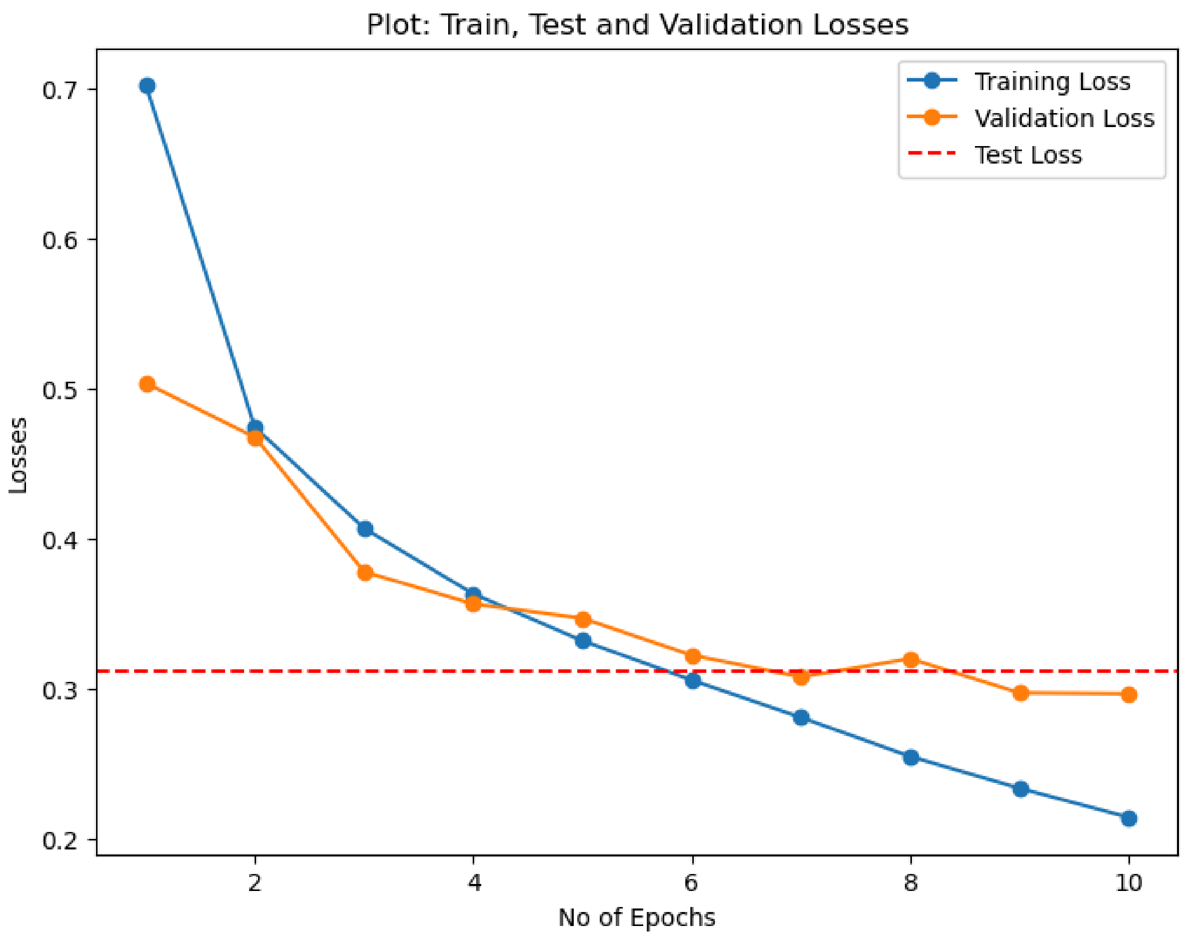
Results:

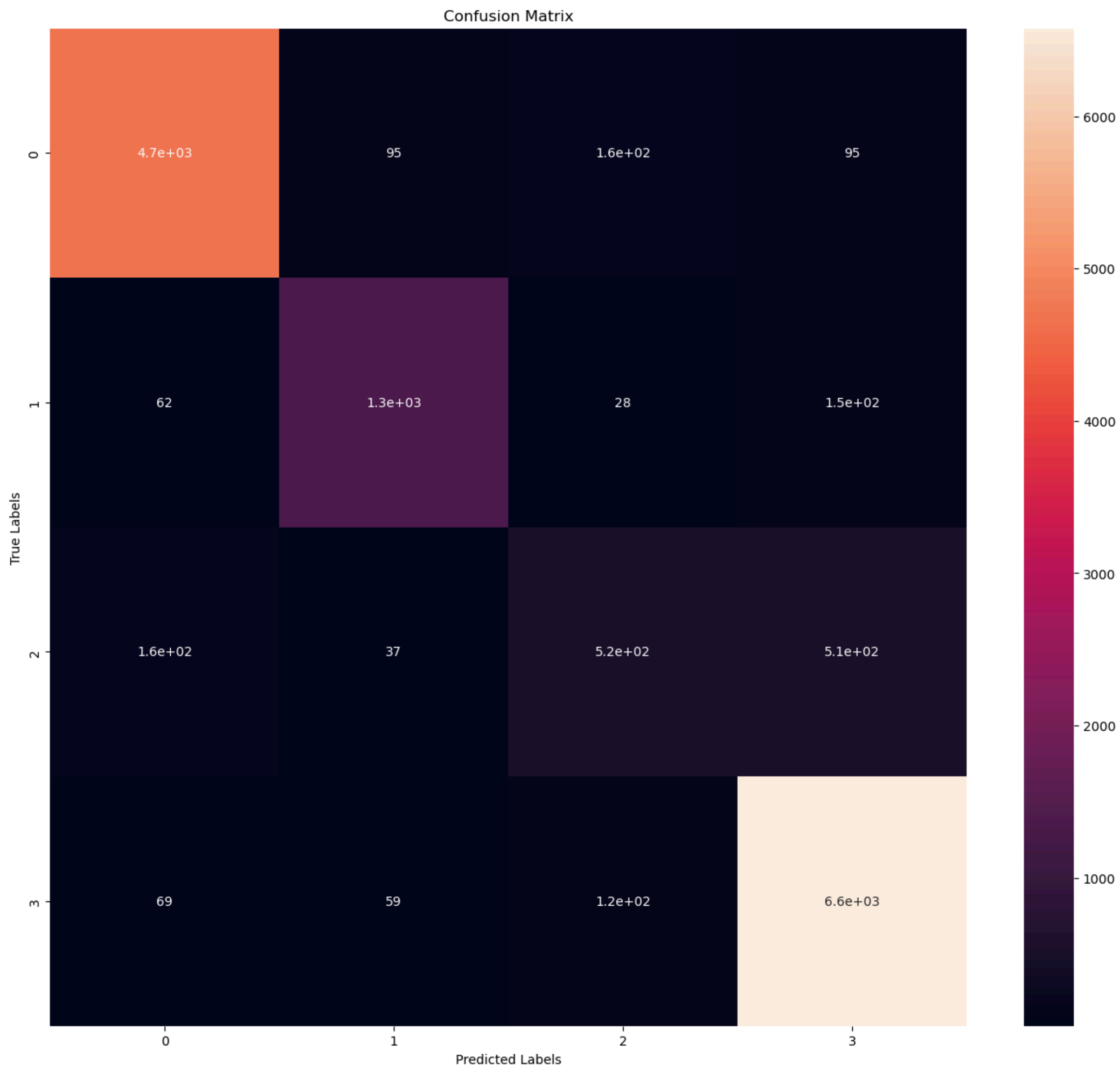
Impact on the results by using Early stopping technique

```
Accuracy:  89.4891609108938
Confusion Matrix:  [[4662   95  157   95]
 [  62 1333   28  146]
 [ 156   37  515  510]
 [  69   59  123 6576]]
Precision:  83.49855519419329
Recall:  79.15922345381499
F1 Score:  80.80351999108359
```

Early stopping graphs for the results:




Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Test Loss
Losses
No of Epochs
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10



Based on the various techniques, **L2 regularization** techniques gave better results when compared to others. So I am saving the model as the best model.

References:

- https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html
- https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html
- https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html
- https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
- https://pytorch.org/tutorials/beginner/saving_loading_models.html
- https://www.data.gov/