ASSIGNMENT 1 - CNN, RNN and LSTM Architectures

Team Members

Charan Kumar Nara 50545001

Neema George 50545746

Part I: CNN Classification

CNN Model Using VGG-13

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
VGG13Network                             --
├─Sequential: 1-1                        --
│    └─Conv2d: 2-1                       640
│    └─ReLU: 2-2                         --
│    └─Conv2d: 2-3                       36,928
│    └─ReLU: 2-4                         --
│    └─MaxPool2d: 2-5                    --
│    └─Conv2d: 2-6                       73,856
│    └─ReLU: 2-7                         --
│    └─Conv2d: 2-8                       147,584
│    └─ReLU: 2-9                         --
│    └─MaxPool2d: 2-10                   --
│    └─Conv2d: 2-11                      295,168
│    └─ReLU: 2-12                        --
│    └─Conv2d: 2-13                      590,080
│    └─ReLU: 2-14                        --
│    └─MaxPool2d: 2-15                   --
│    └─Conv2d: 2-16                      1,180,160
│    └─ReLU: 2-17                        --
│    └─Conv2d: 2-18                      2,359,808
│    └─ReLU: 2-19                        --
│    └─MaxPool2d: 2-20                   --
│    └─Conv2d: 2-21                      2,359,808
│    └─ReLU: 2-22                        --
│    └─Conv2d: 2-23                      2,359,808
│    └─ReLU: 2-24                        --
│    └─MaxPool2d: 2-25                   --
├─Sequential: 1-2                        --
│    └─Linear: 2-26                      8,392,704
│    └─ReLU: 2-27                        --
│    └─Dropout: 2-28                     --
│    └─Linear: 2-29                      16,781,312
│    └─ReLU: 2-30                        --
│    └─Dropout: 2-31                     --
│    └─Linear: 2-32                      4,097,000
│    └─ReLU: 2-33                        --
│    └─Dropout: 2-34                     --
│    └─Linear: 2-35                      3,003
==========================================================================
Total params: 38,677,859
Trainable params: 38,677,859
Non-trainable params: 0
==========================================================================
```

Number of classes: 3

Training set size: 21000

Validation set size: 4500

Test set size: 4500

Input Layer

The model expects to input images with a single color as indicated by the first nn.Conv2d layer's input channel size of 1. This differs from the typical VGG model which expects 3 channels - RGB.

Convolutional Layers

First Block:

Two convolutional layers with 64 filters each, using a kernel size of 3, stride of 1, and padding of 1, followed by ReLU activation functions. This block is concluded with a max pooling layer with a kernel size of 2 and stride of 2.

Second Block:

Similar to the first block, but with 128 filters in each convolutional layer.

Third Block:

Two convolutional layers with 256 filters each, followed by ReLU activations and a max pooling layer.

Fourth Block:

Two convolutional layers with 512 filters each, followed by ReLU activations and a max pooling layer.

Fifth Block:

Similarly, two convolutional layers with 512 filters each, followed by ReLU activations and another max pooling layer.

This structure closely follows the VGG design philosophy, featuring multiple convolutional layers with small kernel sizes, followed by max pooling layers to reduce spatial dimensions while deepening the feature maps.

Fully Connected Layers (Classifier)

The flattened output from the convolutional base feeds into a sequence of fully connected layers:

The first linear layer has an input feature size of 512 * 2 * 2, assuming the input image has been reduced through consecutive pooling to a spatial dimension of 2x2 by the final convolutional layer, and outputs 4096 features.

Followed by ReLU activation and dropout to prevent overfitting.

A second linear layer outputs another 4096 features, again followed by ReLU and dropout.

A third linear layer reduces the dimensionality to 1000 features, followed by ReLU and dropout.

The final linear layer outputs the class scores for num_classes categories.

Output

The model does not apply a softmax layer at the end since, in PyTorch, the softmax operation is typically integrated into the loss function for numerical stability and efficiency.

Training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss

Accuracy: 78.2

Confusion Matrix: [[1441 40 5]

[512 1011 10]

[270 144 1067]]

Precision: 82.67950034145564

Recall: 78.32225683356756

F1 Score: 78.36164068461406Training Loss: 0.3302

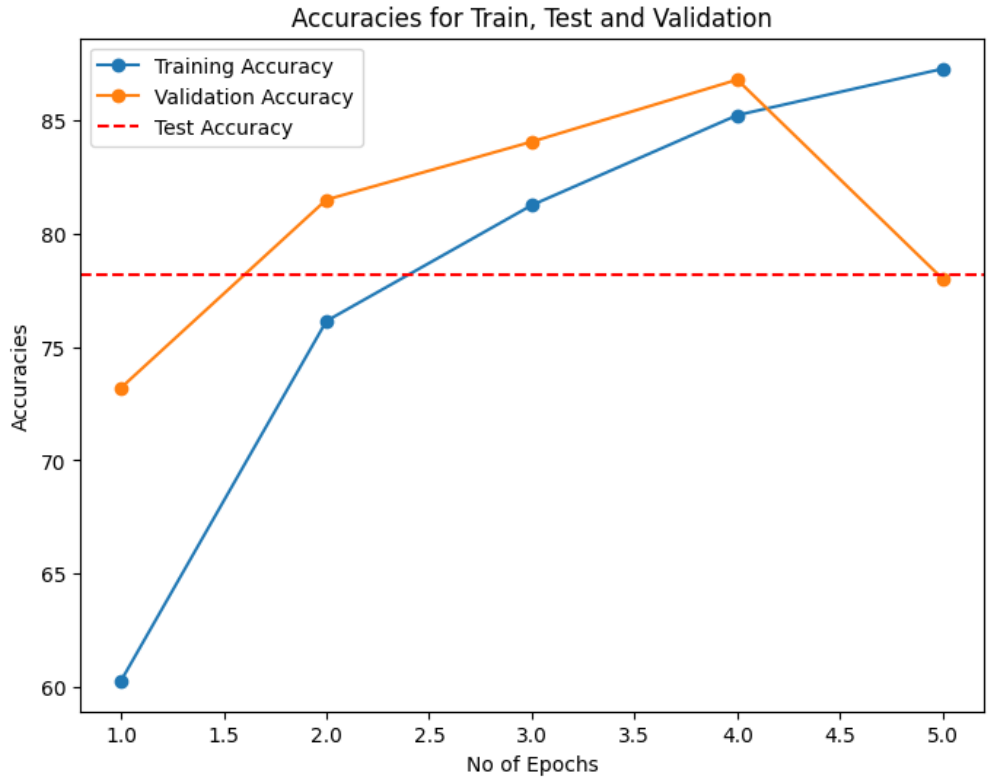
Validation Loss: 0.6629

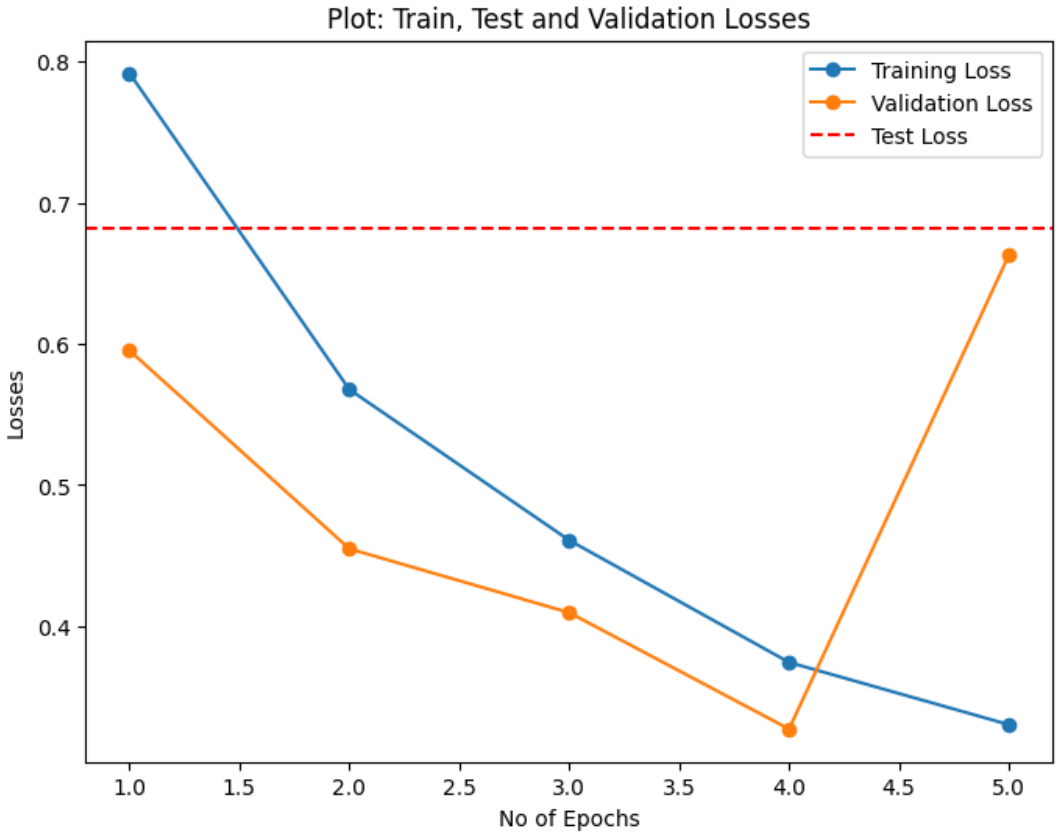
Train Accuracy: 87.25

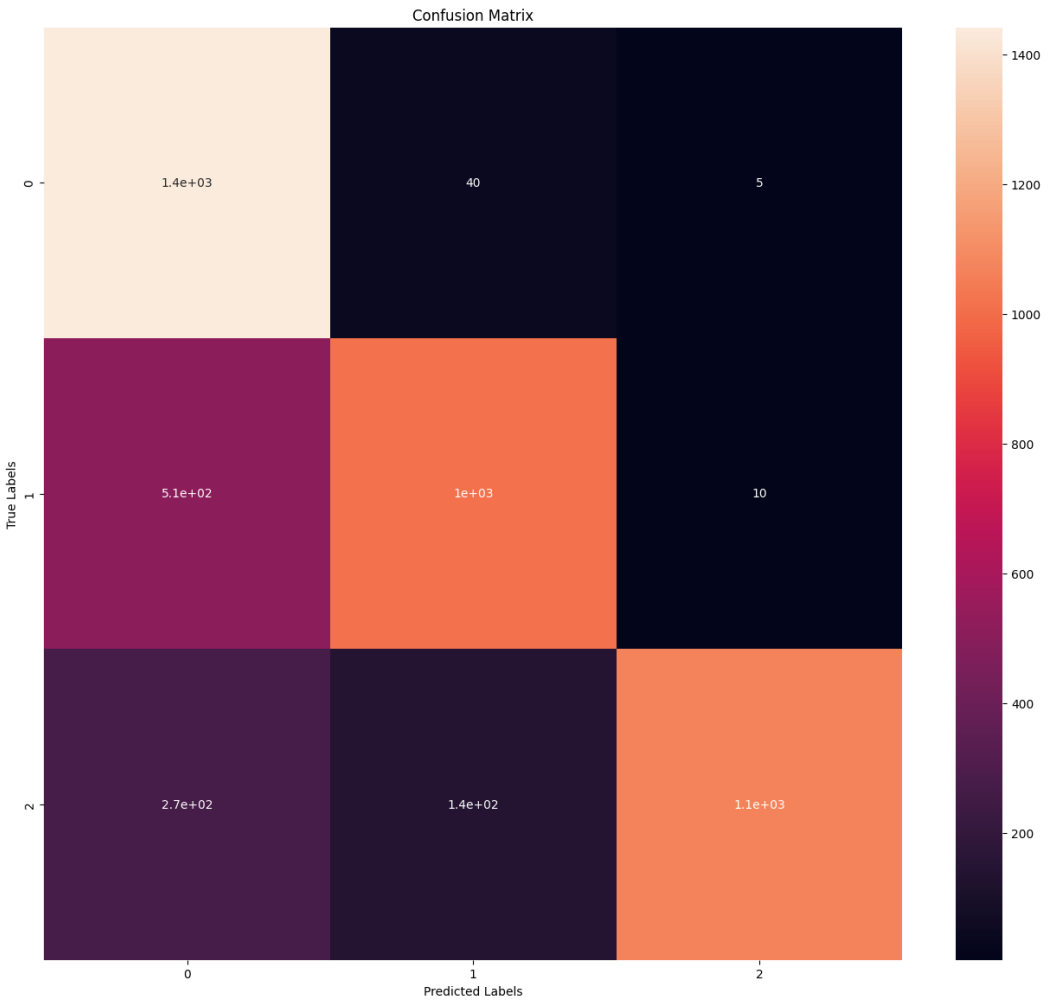
Validation Accuracy: 78.0

Recall: 88.37082581211581

F1 Score: 88.27981498860774


Accuracies for Train, Test and Validation
Training Accuracy
Validation Accuracy
Test Accuracy
Accuracies
No of Epochs
60
65
70
75
80
85
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Test Loss
Losses
No of Epochs
0.4
0.5
0.6
0.7
0.8
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0



Impact of Techniques on the Performance of the Model

- Drop Out

On using drop out the values are as shown below:

Training Loss: 0.3278

Validation Loss: 0.3182

Train Accuracy: 87.28571428571429

Validation Accuracy: 87.93333333333334

Accuracy: 87.84444444444445

Confusion Matrix: [[1278 177   34]

 [162 1270   71]

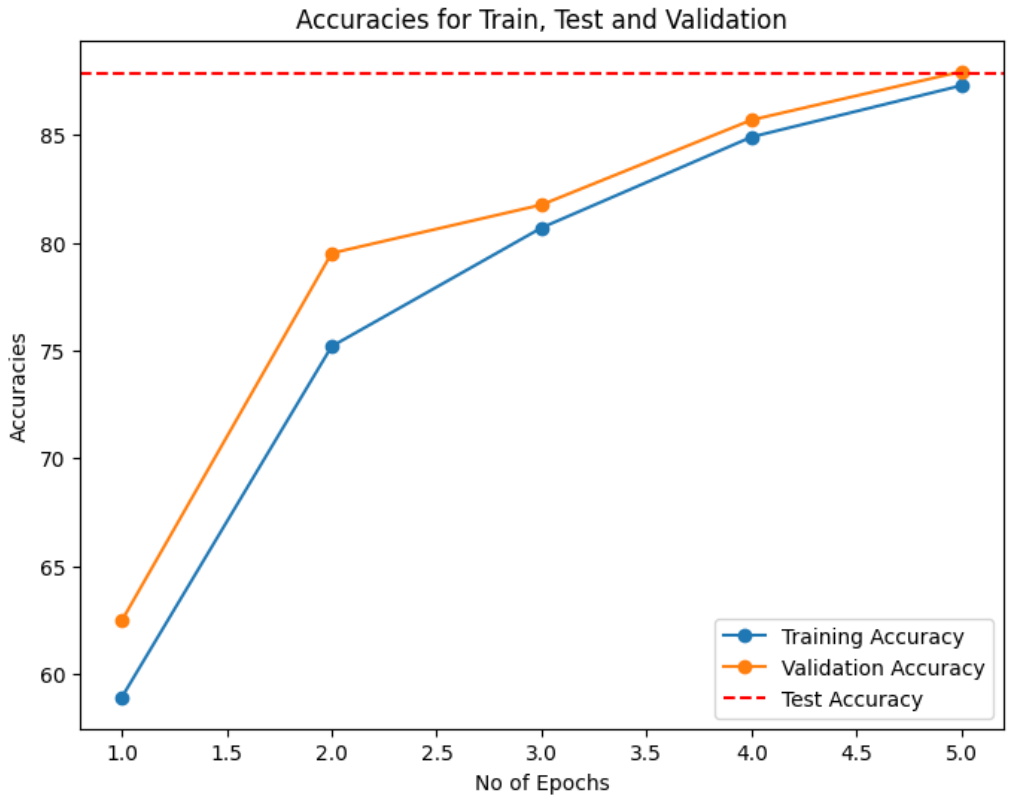
[63 40 1405]]

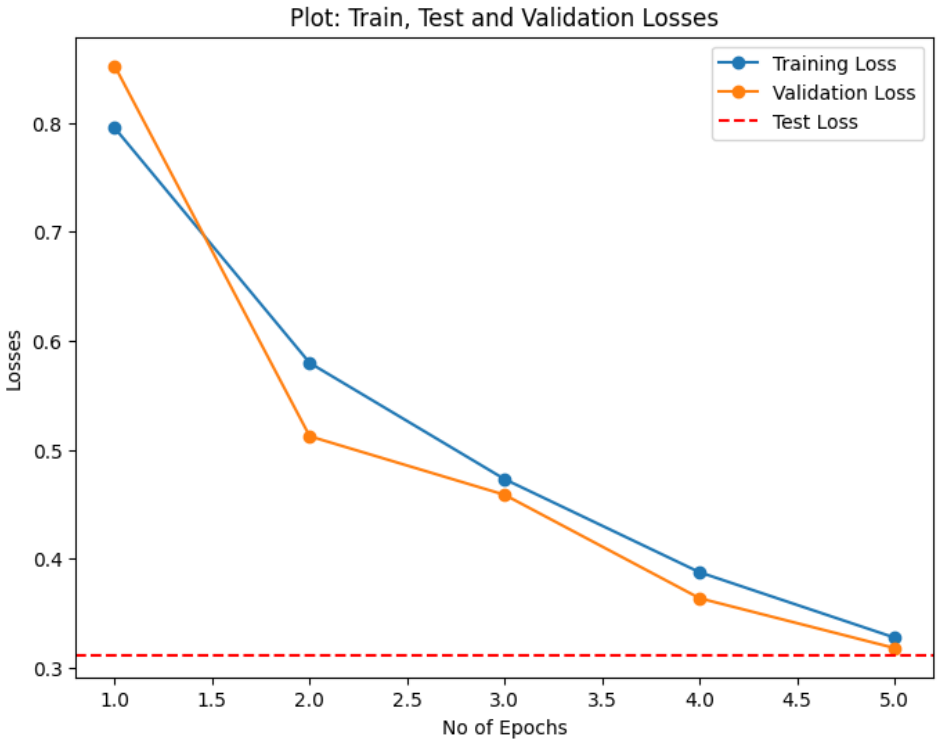
Precision: 87.82771906140289

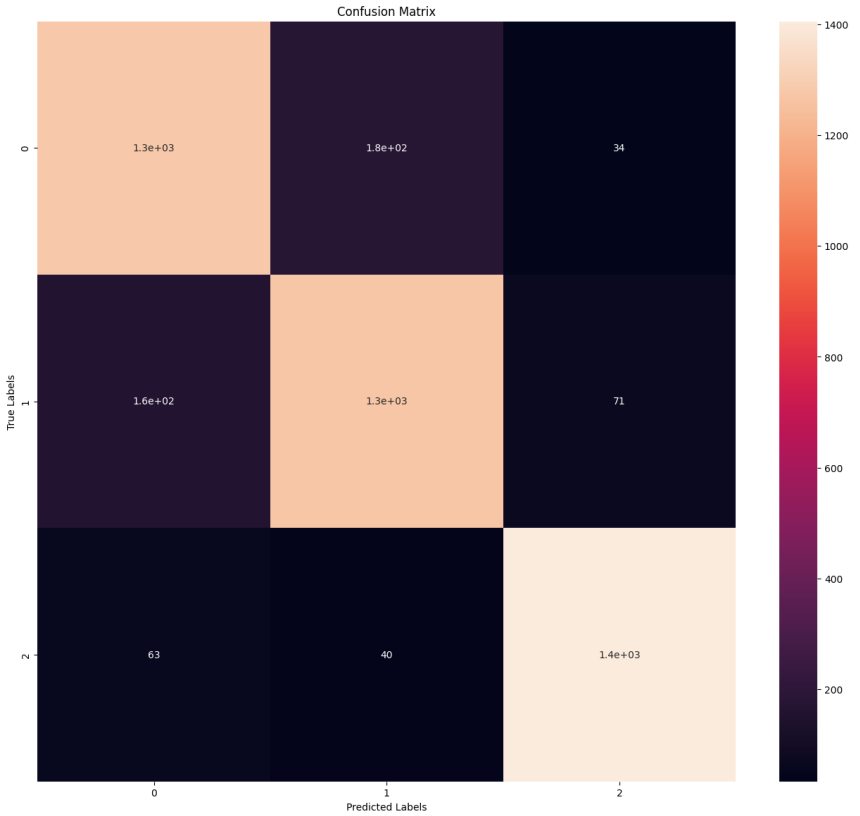
Recall: 87.83228277082443

F1 Score: 87.82855293929514

The plots are shown below:







- Early Stopping

Accuracy: 85.77777777777777

Confusion Matrix: [[1129 345   15]

[53 1435   15]

[36 176 1296]]

Precision: 87.93150132856948

Recall: 85.74668653235032

F1 Score: 85.94872089855046

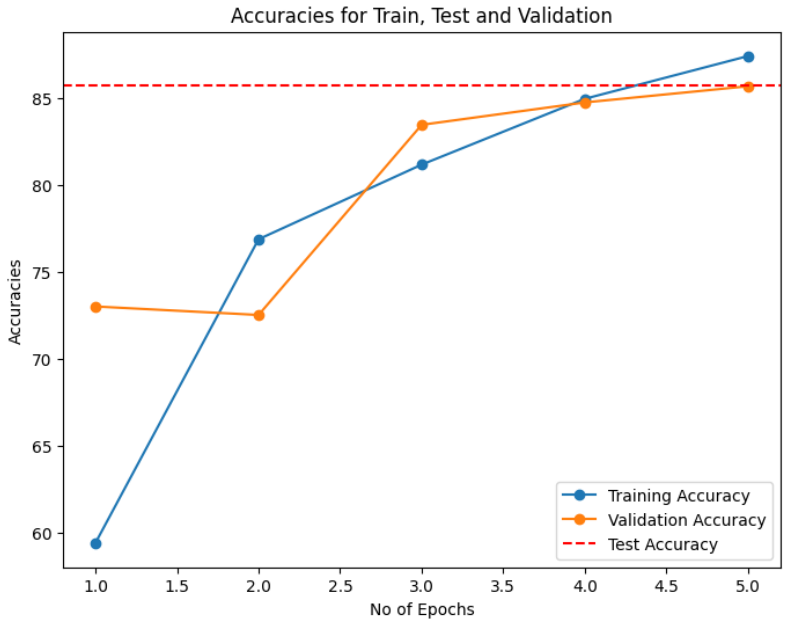
Training Loss: 0.3262

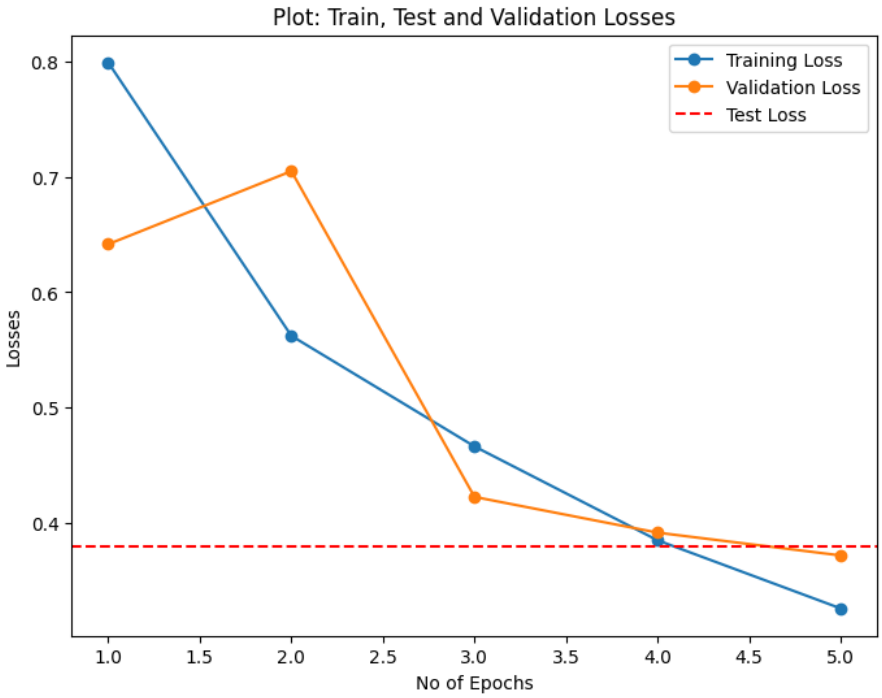
Validation Loss: 0.3721

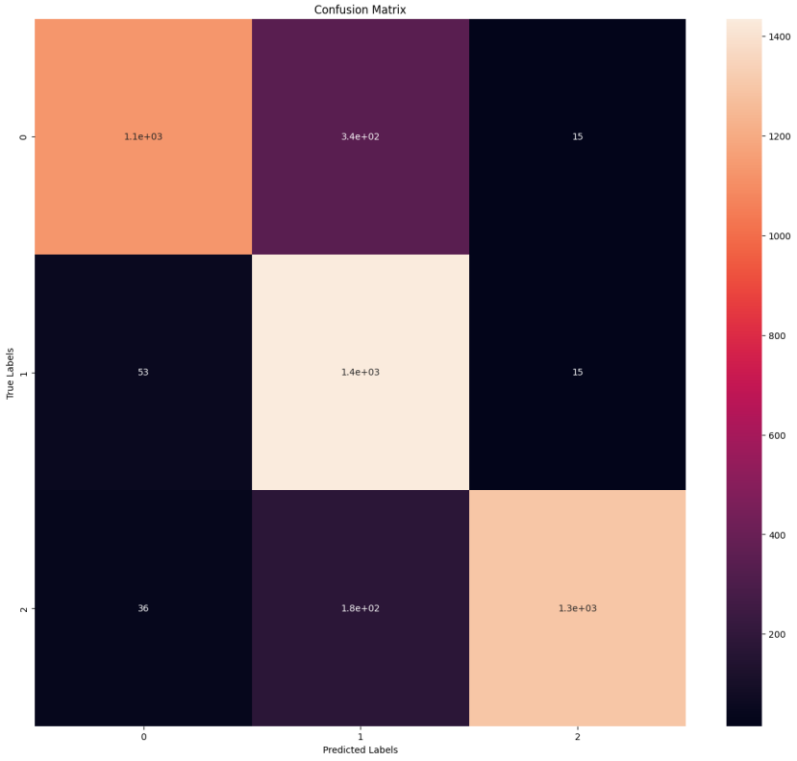
Validation Accuracy: 85.71111111111111

Train Accuracy: 87.44761904761904

The plots for it are as shown below:







- Image Augmentation

```
transforms_augmentation = v2.Compose([
    v2.RandomResizedCrop(size=(32, 32), antialias=True),
    v2.RandomHorizontalFlip(p=0.5),
    v2.ToTensor(),
    v2.Grayscale(),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.5, ], std=[0.5, ]),
])
```

We have used image augmentation by cropping the image, horizontally flipping the image and changing it to grayscale.

Using image augmentation, we get an impact on our training, testing and validation values as shown below:

Accuracy: 76.95555555555555

Confusion Matrix: [[1131 135 215]

[289 936 322]

[39 37 1396]]

Precision: 78.07157699529776

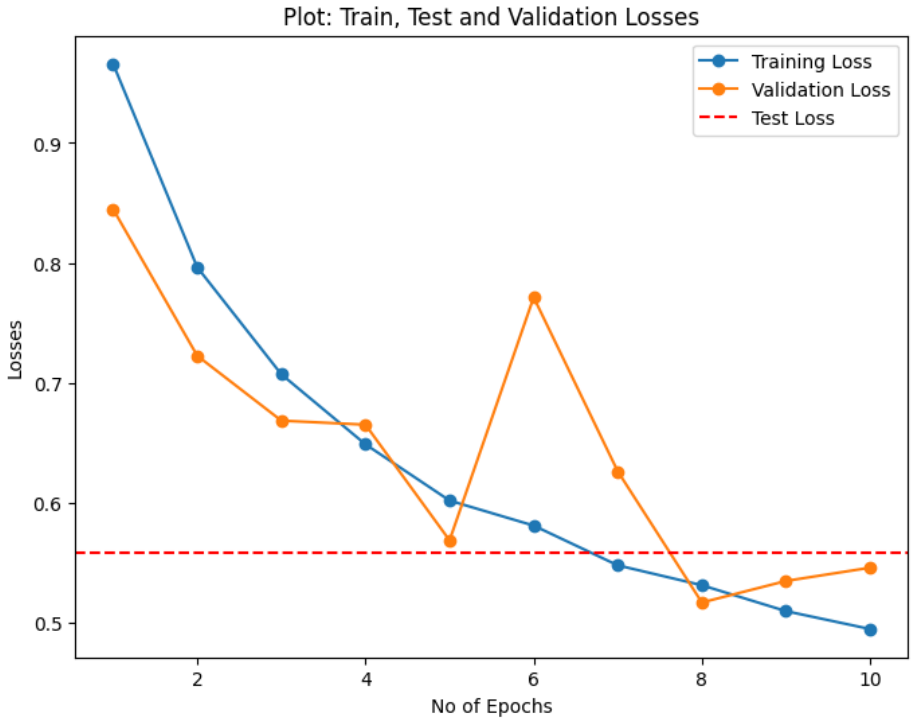
Recall: 77.2361591937365

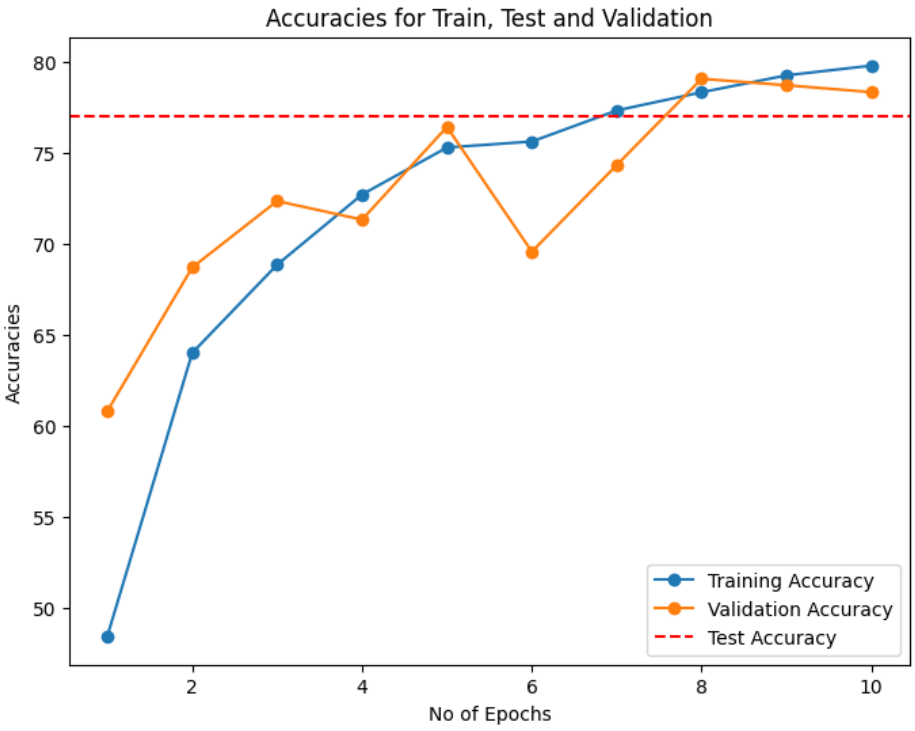
F1 Score: 76.48143774297095

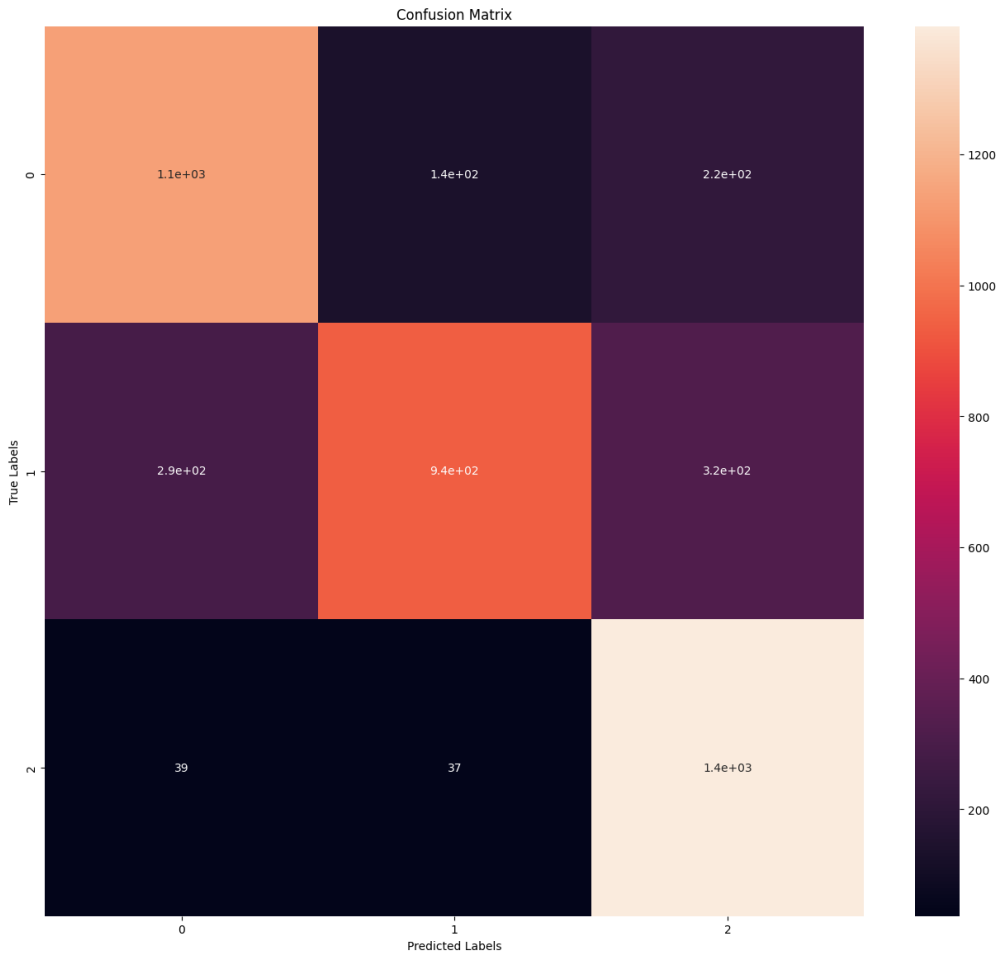
Training Loss: 0.4945

Validation Loss: 0.5457

The required plots are as shown below:


Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Test Loss
Losses
No of Epochs
0.9
0.8
0.7
0.6
0.5
2
4
6
8
10


Accuracies for Train, Test and Validation
Training Accuracy
Validation Accuracy
Test Accuracy
Accuracies
No of Epochs
80
75
70
65
60
55
50
2
4
6
8
10



- L2 Regularization

Using L2 regularization we obtain the following values:

Accuracy: 86.84444444444445

Confusion Matrix: [[1389 81 19]

[288 1173 42]

[90 72 1346]]

Precision: 87.57796092641391

Recall: 86.86176329424143

F1 Score: 86.86538460813657

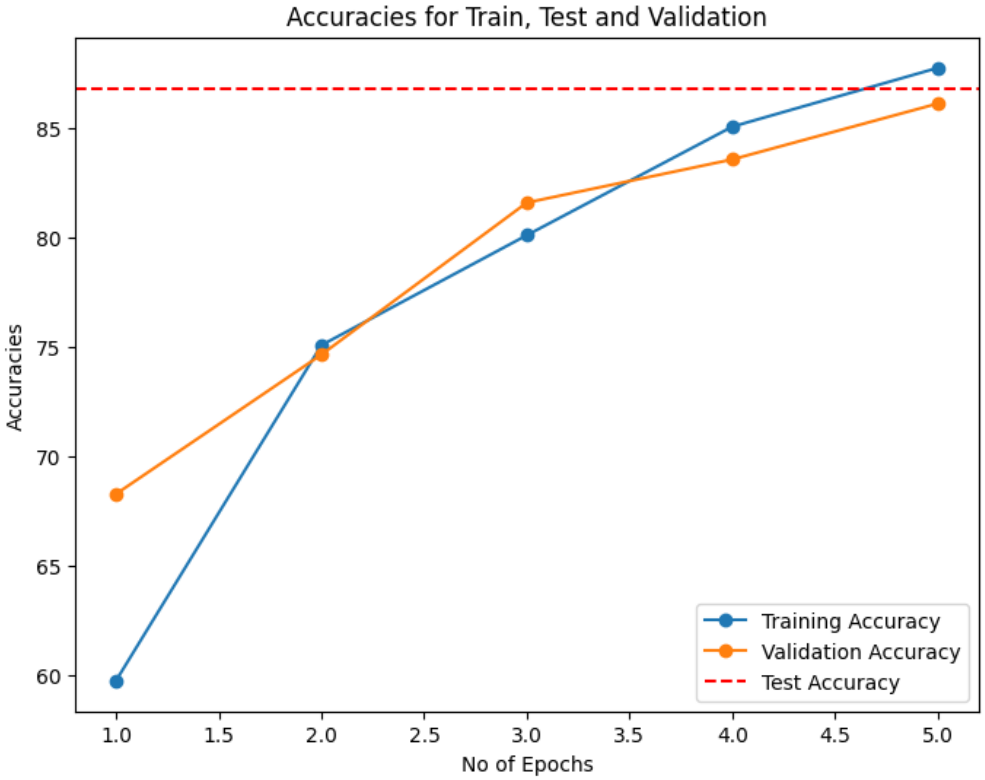
Training Loss: 0.3230

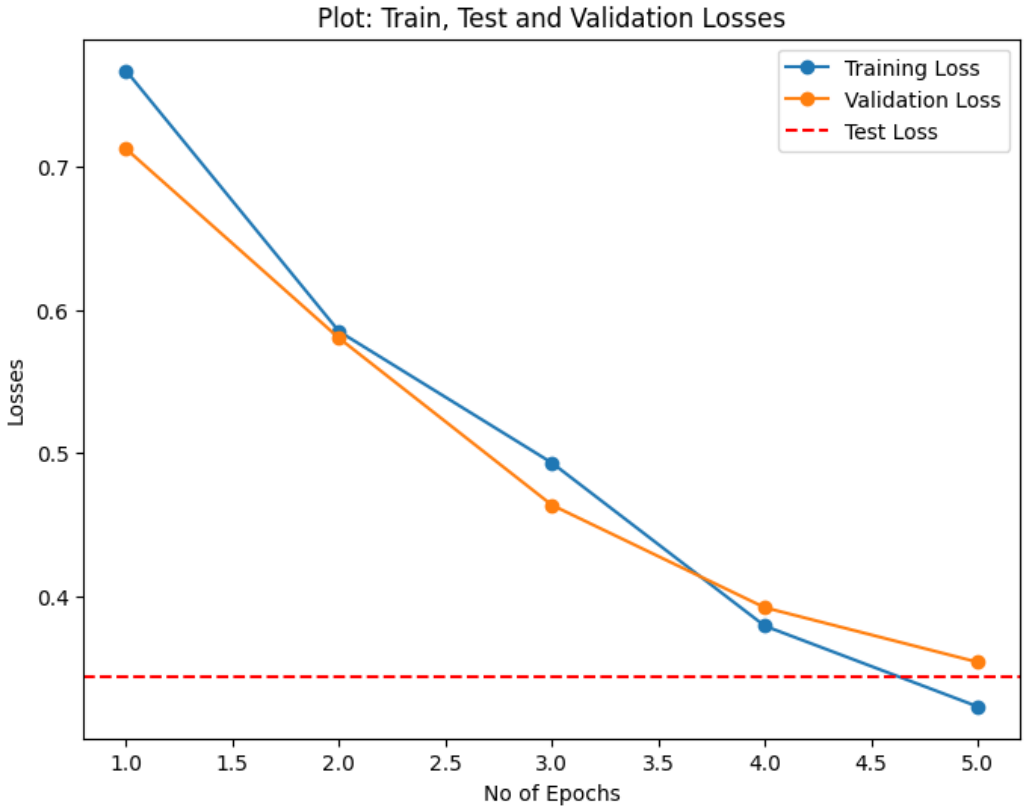
Validation Loss: 0.3541

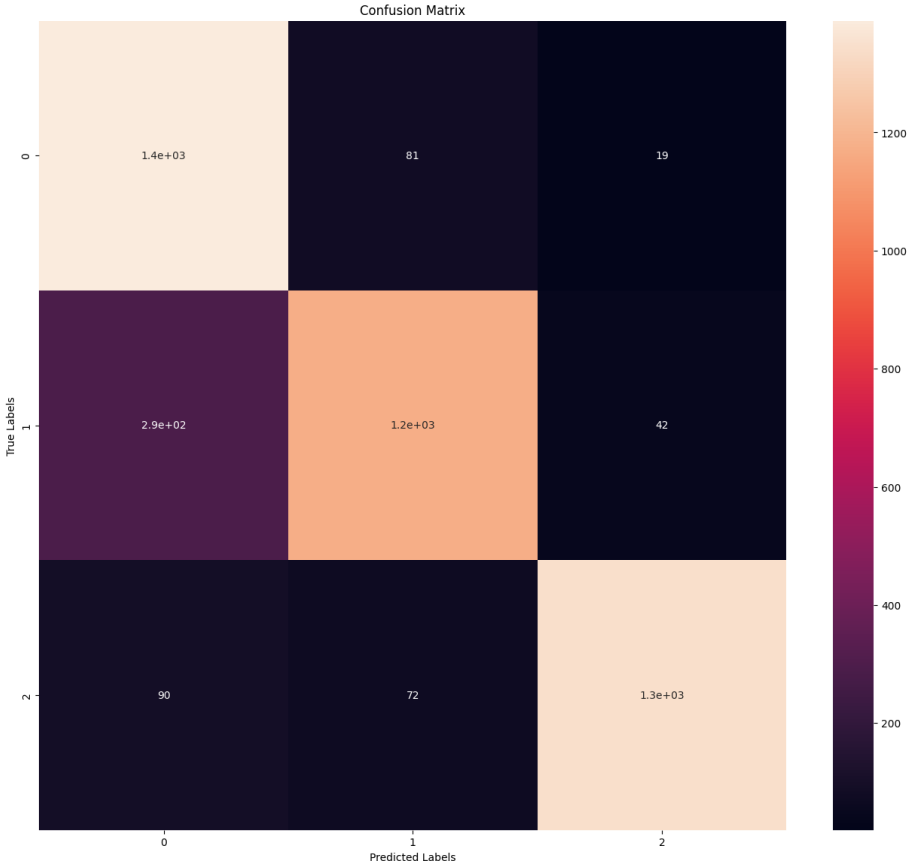
Validation Accuracy: 86.13

Train Accuracy: 87.76

The plots are as below:

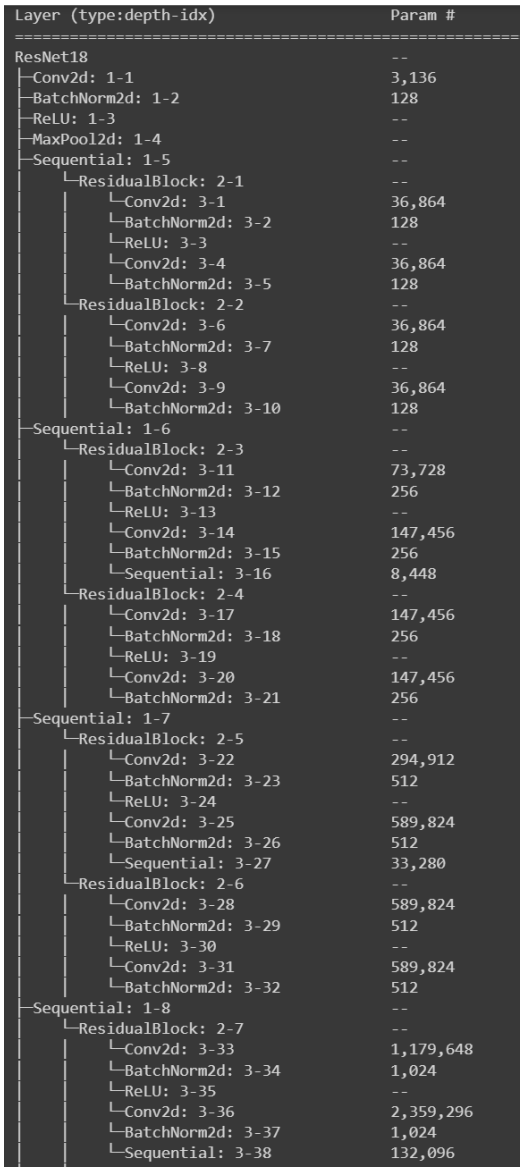



Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Test Loss
Losses
0.7
0.6
0.5
0.4
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
No of Epochs

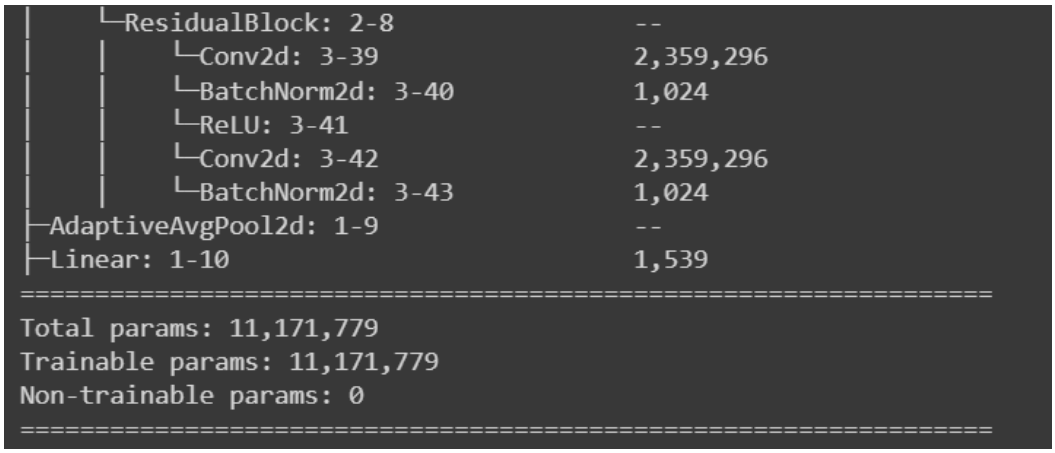

Confusion Matrix
1.4e+03
81
19
2.9e+02
1.2e+03
42
90
72
1.3e+03
True Labels
Predicted Labels
0
1
2
1200
1000
800
600
400
200

Part II: Implementing ResNet Architecture

ResNet Architecture



```
Layer (type:depth-idx)                   Param #
==========================================================
ResNet18                                 --
├─Conv2d: 1-1                            3,136
├─BatchNorm2d: 1-2                       128
├─ReLU: 1-3                              --
├─MaxPool2d: 1-4                         --
├─Sequential: 1-5                        --
│    └─ResidualBlock: 2-1                --
│    │    └─Conv2d: 3-1                  36,864
│    │    └─BatchNorm2d: 3-2             128
│    │    └─ReLU: 3-3                    --
│    │    └─Conv2d: 3-4                  36,864
│    │    └─BatchNorm2d: 3-5             128
│    └─ResidualBlock: 2-2                --
│    │    └─Conv2d: 3-6                  36,864
│    │    └─BatchNorm2d: 3-7             128
│    │    └─ReLU: 3-8                    --
│    │    └─Conv2d: 3-9                  36,864
│    │    └─BatchNorm2d: 3-10            128
├─Sequential: 1-6                        --
│    └─ResidualBlock: 2-3                --
│    │    └─Conv2d: 3-11                 73,728
│    │    └─BatchNorm2d: 3-12            256
│    │    └─ReLU: 3-13                   --
│    │    └─Conv2d: 3-14                 147,456
│    │    └─BatchNorm2d: 3-15            256
│    │    └─Sequential: 3-16             8,448
│    └─ResidualBlock: 2-4                --
│    │    └─Conv2d: 3-17                 147,456
│    │    └─BatchNorm2d: 3-18            256
│    │    └─ReLU: 3-19                   --
│    │    └─Conv2d: 3-20                 147,456
│    │    └─BatchNorm2d: 3-21            256
├─Sequential: 1-7                        --
│    └─ResidualBlock: 2-5                --
│    │    └─Conv2d: 3-22                 294,912
│    │    └─BatchNorm2d: 3-23            512
│    │    └─ReLU: 3-24                   --
│    │    └─Conv2d: 3-25                 589,824
│    │    └─BatchNorm2d: 3-26            512
│    │    └─Sequential: 3-27             33,280
│    └─ResidualBlock: 2-6                --
│    │    └─Conv2d: 3-28                 589,824
│    │    └─BatchNorm2d: 3-29            512
│    │    └─ReLU: 3-30                   --
│    │    └─Conv2d: 3-31                 589,824
│    │    └─BatchNorm2d: 3-32            512
├─Sequential: 1-8                        --
│    └─ResidualBlock: 2-7                --
│    │    └─Conv2d: 3-33                 1,179,648
│    │    └─BatchNorm2d: 3-34            1,024
│    │    └─ReLU: 3-35                   --
│    │    └─Conv2d: 3-36                 2,359,296
│    │    └─BatchNorm2d: 3-37            1,024
│    │    └─Sequential: 3-38             132,096
```



The ResNet architecture are 18 layers deep. We have set the number of classes to 36.

Initial Convolution and Pooling:

The network starts with a single convolutional layer with a 7x7 kernel, stride of 2, and padding of 3, followed by batch normalization and ReLU activation. This is then followed by a max pooling layer. This setup processes the input image to reduce its spatial dimensions while increasing the depth (number of channels).

Residual Blocks:

The architecture utilizes a series of residual blocks grouped into 4 layers (layer1 to layer4). The make_layer method constructs these layers, where each consists of a number of residual blocks defined by the layers list passed during initialization ([2, 2, 2, 2] for ResNet18).

Each residual block within these layers may include a down sample component to match the dimensions between the input and the output of the block if necessary, using a 1x1 convolution.

The blocks double the number of filters (and thus the depth) at each subsequent layer (64, 128, 256, 512), while reducing the spatial dimension of the feature maps by half using a stride of 2 in the first block of each layer (except the first).

Adaptive Average Pooling and Fully Connected Layer:

Following the residual blocks, the network applies an adaptive average pooling layer to reduce each feature map to a single value, effectively producing a fixed-size vector of features regardless of the input image size.

Finally, a fully connected layer (fc) transforms this vector into the final output, where the number of output units corresponds to the number of classes.

Training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss

Accuracy: 87.42222222222222

Confusion Matrix: [[1260 151 78]

[119 1257 127]

[52   39 1417]]

Precision: 87.42699392240111

Recall: 87.40626749249624

F1 Score: 87.35494661215996

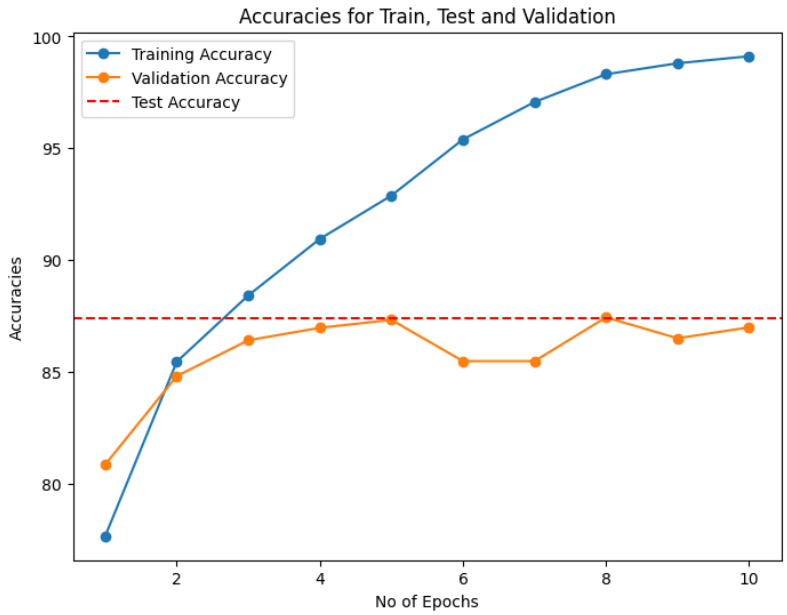
Training Loss: 0.0251

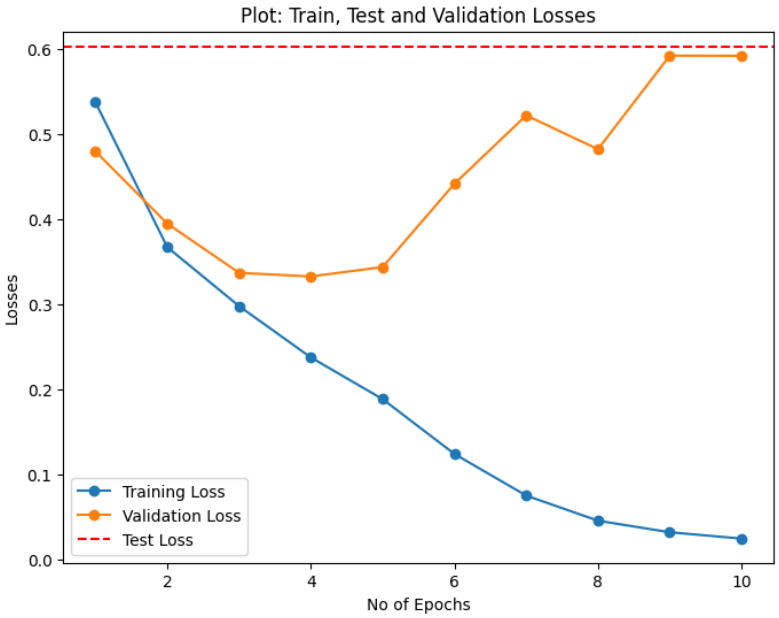
Validation Loss: 0.5915

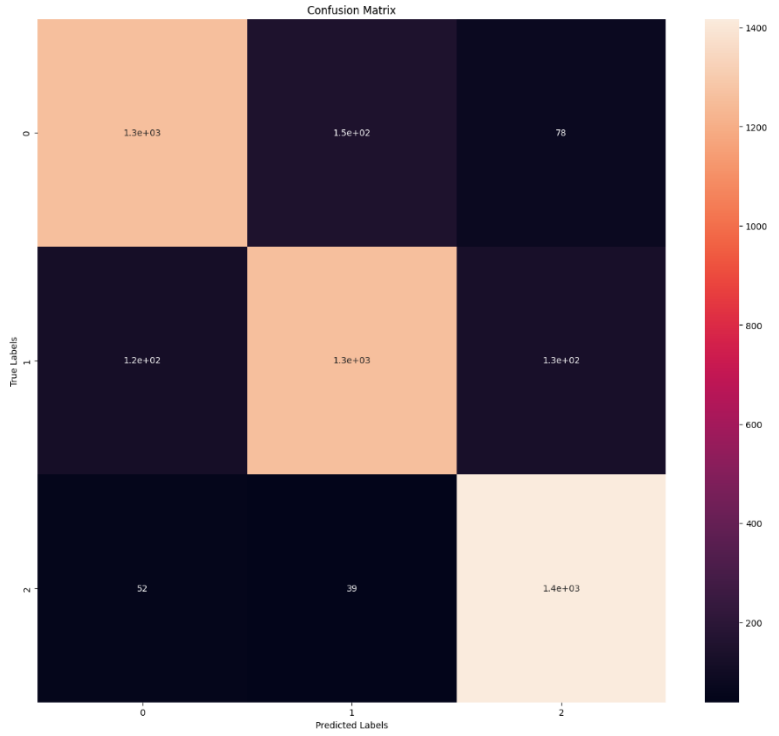
Train Accuracy: 99.11904761904762

Validation Accuracy: 87.0

The plots are shown below:







Impact of Techniques on the Performance of the Model

- L2 Regularization

The values obtained are :

Accuracy: 87.64444444444445

Confusion Matrix: [[1273 157 59]

[156 1267 80]

[44 60 1404]]

Precision: 87.59706694202562

Recall: 87.63171289353032

F1 Score: 87.60837187827116

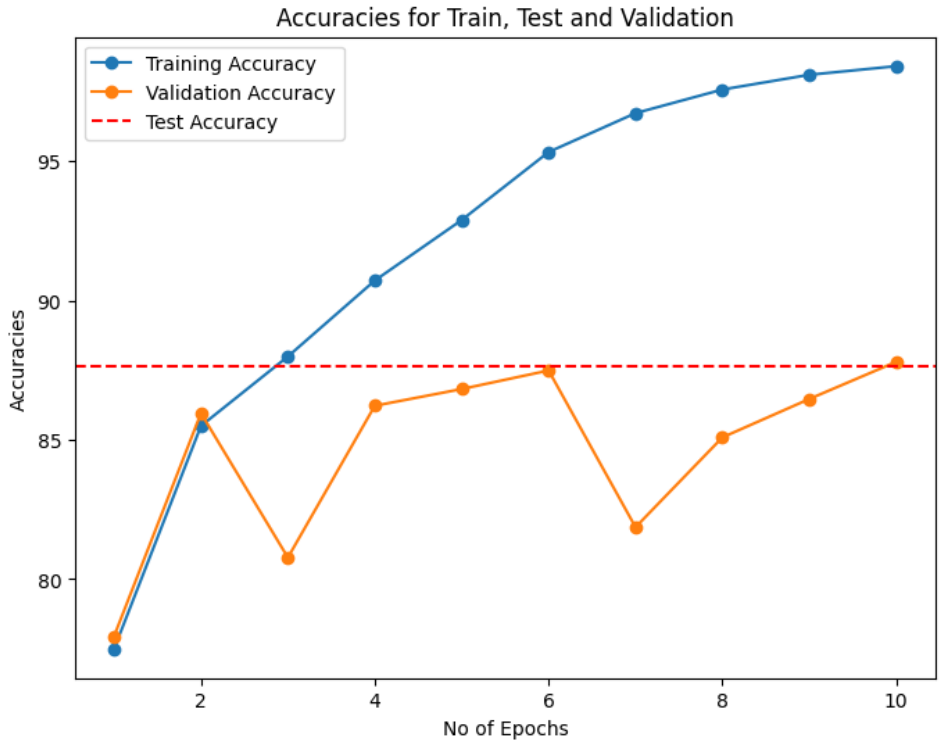
Training Loss: 0.0445

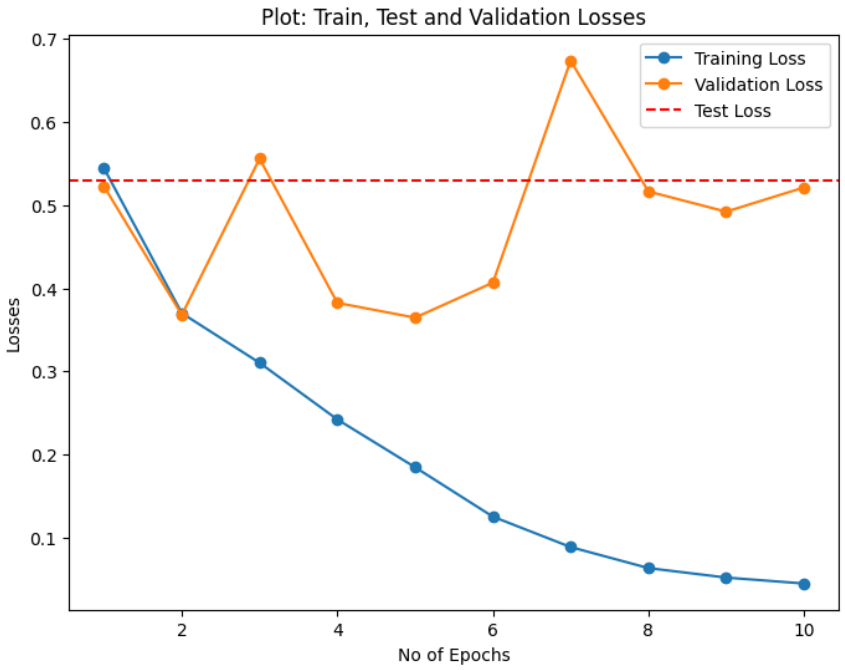
Validation Loss: 0.5213

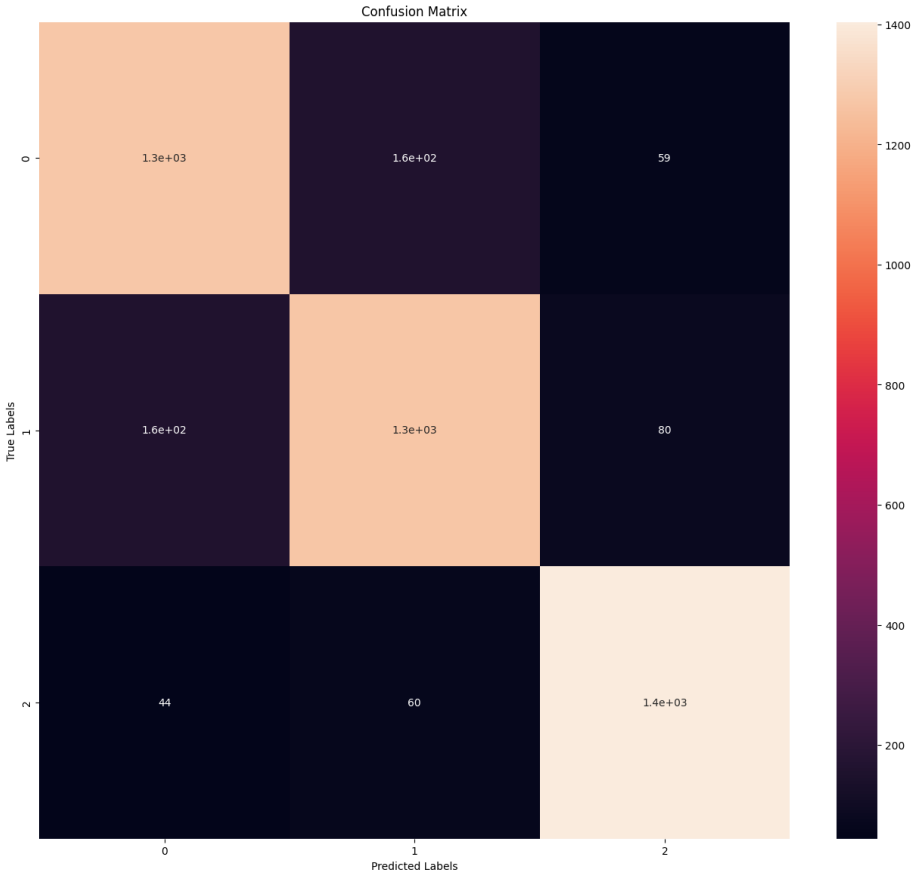
Train Accuracy: 98.4047619047619

Validation Accuracy: 87.8

The plots are as shown below:







- Early Stopping

The values obtained are:

Accuracy: 88.4

Confusion Matrix: [[1306 128   55]

 [157 1271   75]

 [54   53 1401]]

Precision: 88.37807401522406

Recall: 88.39286220162941

F1 Score: 88.3726505339010

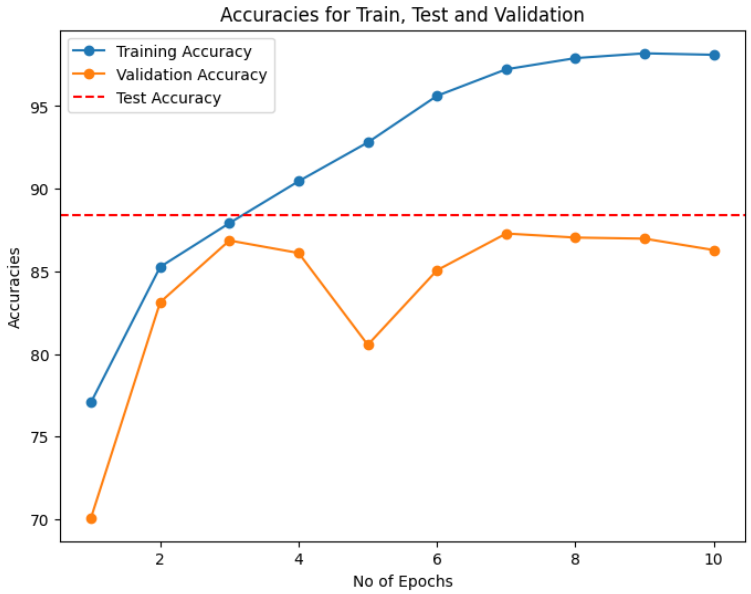
Training Loss: 0.0562

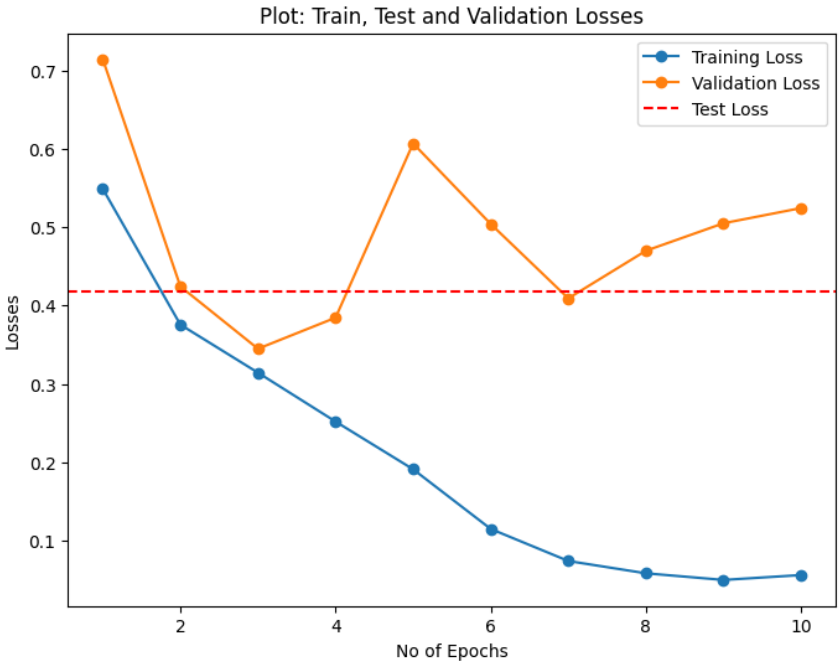
Validation Loss: 0.5243

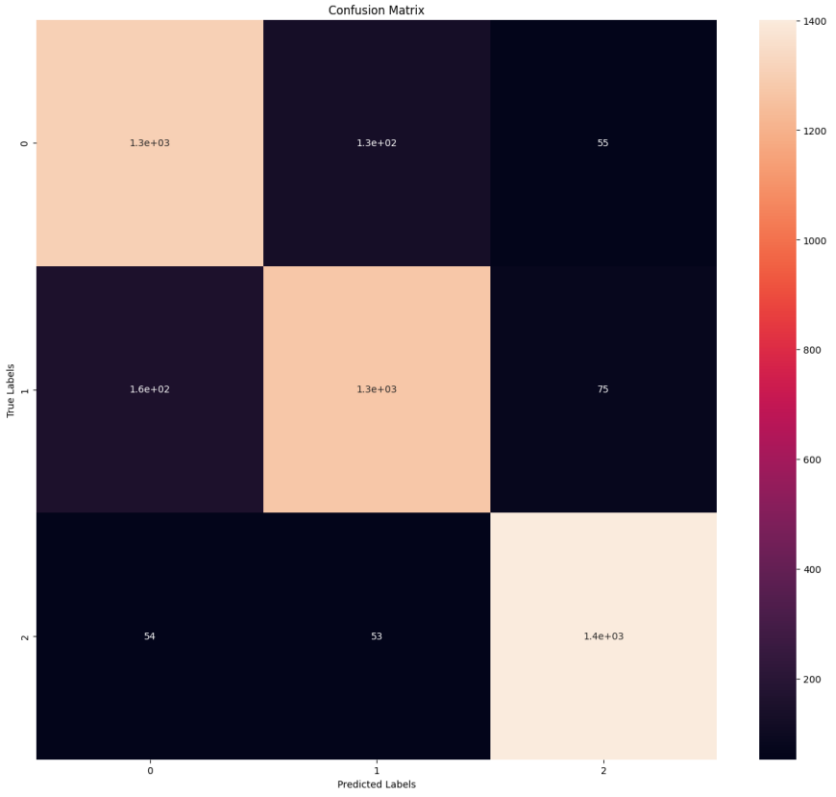
Train Accuracy: 98.1

Validation Accuracy: 86.28888888888889

The plots are shown below:




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
0.1
2
4
6
8
10


Confusion Matrix
True Labels
Predicted Labels
1.3e+03
1.3e+02
55
1.6e+02
1.3e+03
75
54
53
1.4e+03
0
1
2
1400
1200
1000
800
600
400
200

Part 3: Time-Series Forecasting using RNN

Nature of Dataset

We chose to work with the Retail Sales Dataset which shows sales figures of a retail store or chain over time, typically including features such as product categories, promotions, and seasonality. The features dataset is of the format as shown in the image attached below,

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	05/02/2010	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	12/02/2010	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	19/02/2010	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	26/02/2010	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	05/03/2010	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

We have the sales dataset and stores dataset which are as shown,

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	05/02/2010	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	12/02/2010	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	19/02/2010	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	26/02/2010	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	05/03/2010	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

We merge the sales dataset with the stores dataset to form a dataset having the dimensions 421570 rows × 7 columns. It is that ensured that the date columns are in the same format across all data frames before merging with the features dataset to have dimensions 421570 rows × 17 columns.

Date	Store	Dept	Weekly_Sales	IsHoliday	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	Year	Mont
2010-01-10	13	14	20922.18	False	A	219622	68.74	2.853	0.00	0.0	0.00	0.00	0.00	126.234600	7.795	2010	
2010-01-10	40	74	9872.41	False	A	155083	62.01	2.717	0.00	0.0	0.00	0.00	0.00	132.756800	5.287	2010	
2010-01-10	40	79	10840.77	False	A	155083	62.01	2.717	0.00	0.0	0.00	0.00	0.00	132.756800	5.287	2010	
2010-01-10	27	98	9837.86	False	A	204184	70.19	2.840	0.00	0.0	0.00	0.00	0.00	136.629757	8.021	2010	
2010-01-10	27	97	19095.76	False	A	204184	70.19	2.840	0.00	0.0	0.00	0.00	0.00	136.629757	8.021	2010	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2012-12-10	6	83	3690.95	False	A	202505	65.43	3.601	1905.34	0.0	13.83	559.50	8706.87	225.005473	5.329	2012	1
2012-12-10	6	85	1645.71	False	A	202505	65.43	3.601	1905.34	0.0	13.83	559.50	8706.87	225.005473	5.329	2012	1
2012-12-10	6	87	44915.71	False	A	202505	65.43	3.601	1905.34	0.0	13.83	559.50	8706.87	225.005473	5.329	2012	1

RNN Architecture

```
==========================================================================
Layer (type:depth-idx)                             Param #
==========================================================================
DropoutRNN                                         --
├─RNN: 1-1                                         20,600
├─Linear: 1-2                                      51
├─Dropout: 1-3                                     --
==========================================================================
Total params: 20,651
Trainable params: 20,651
Non-trainable params: 0
==========================================================================
```

input_size: The number of expected features in the input x. This is the dimensionality of the input data for each time step.

hidden_size: The number of features in the hidden state h. This represents the capacity of the network's memory or the internal features it can use to model the data.

output_size (default = 1): The size of the output

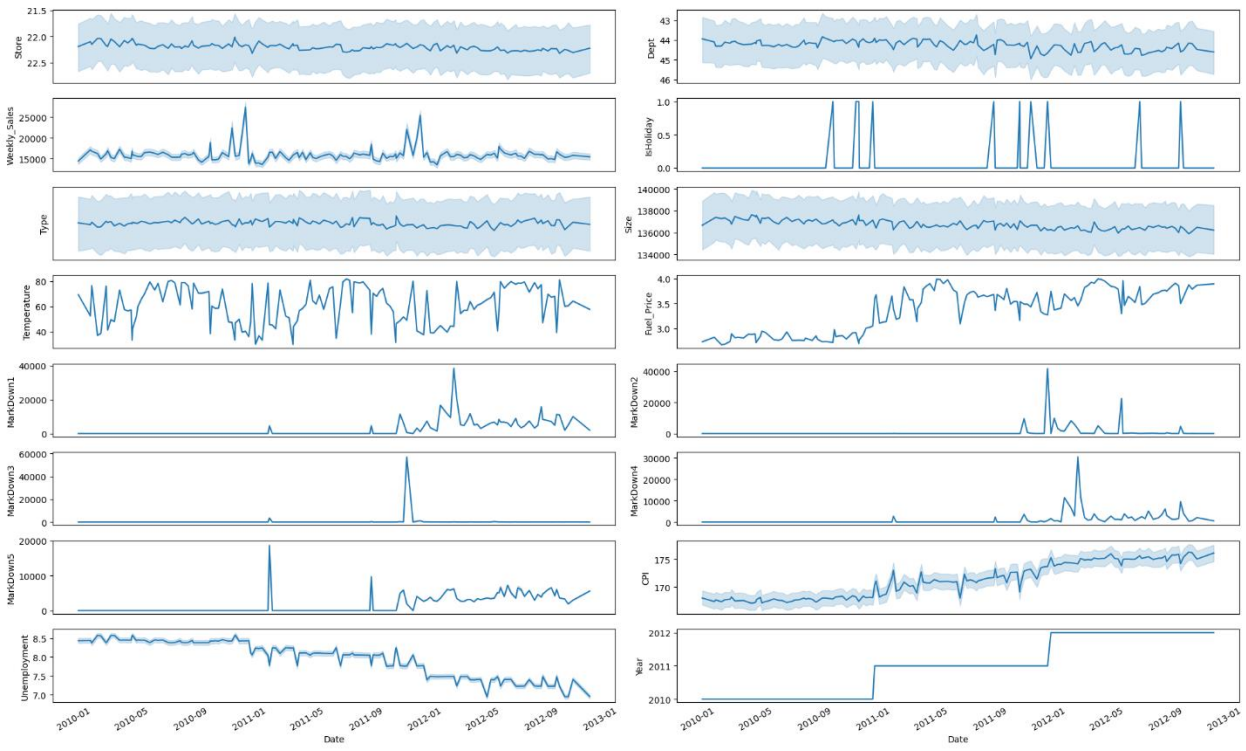
num_layers (default = 3): The number of recurrent layers.

An RNN layer is created using nn.RNN. It takes input_size, hidden_size, and num_layers as arguments, with batch_first=True to show that the input tensor's first dimension is expected to be the batch size.

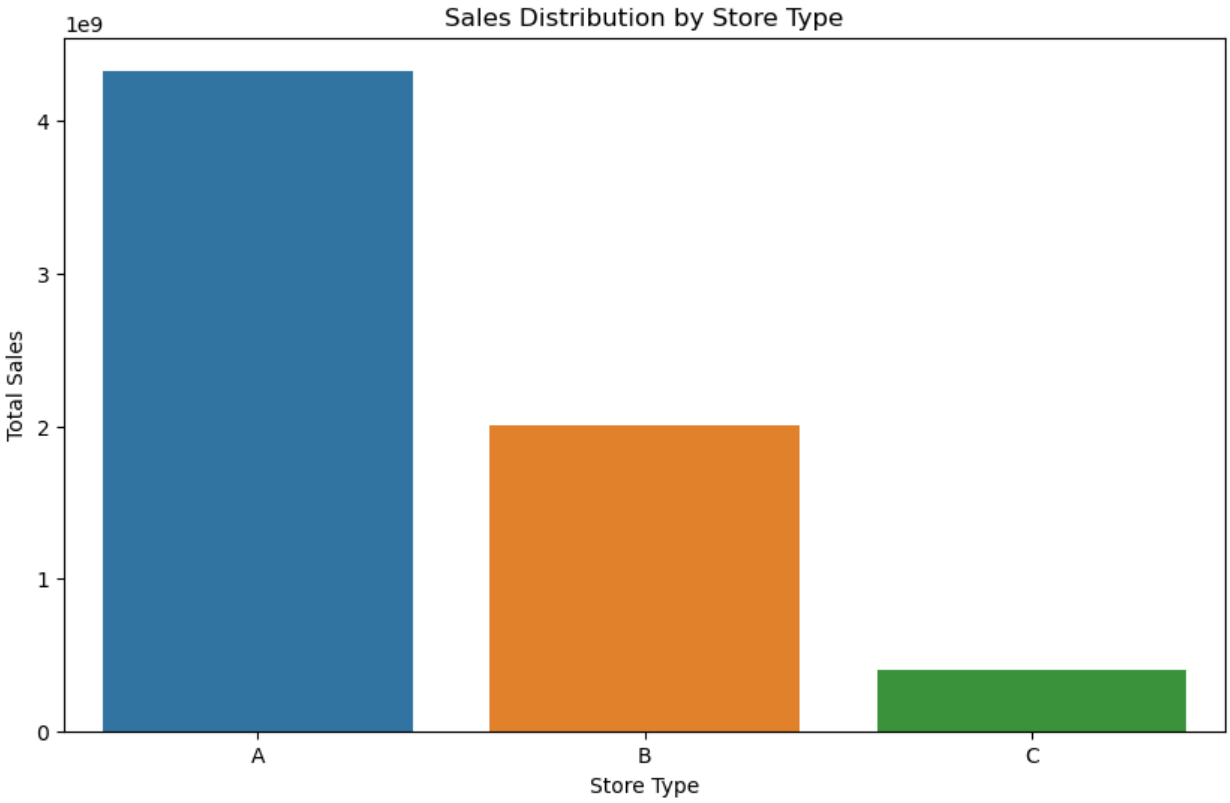
A fully connected layer is initialized with nn.Linear(hidden_size, output_size). This layer maps the hidden state of the RNN to the output.

The input x is processed to ensure it has three dimensions where seq_len is the sequence length for each sample. If x has only two dimensions (implying a seq_len of 1), it gets an additional dimension with unsqueeze(1). The output of the last time step is selected with x[:, -1, :] and passed through the fully connected layer to produce the final output.

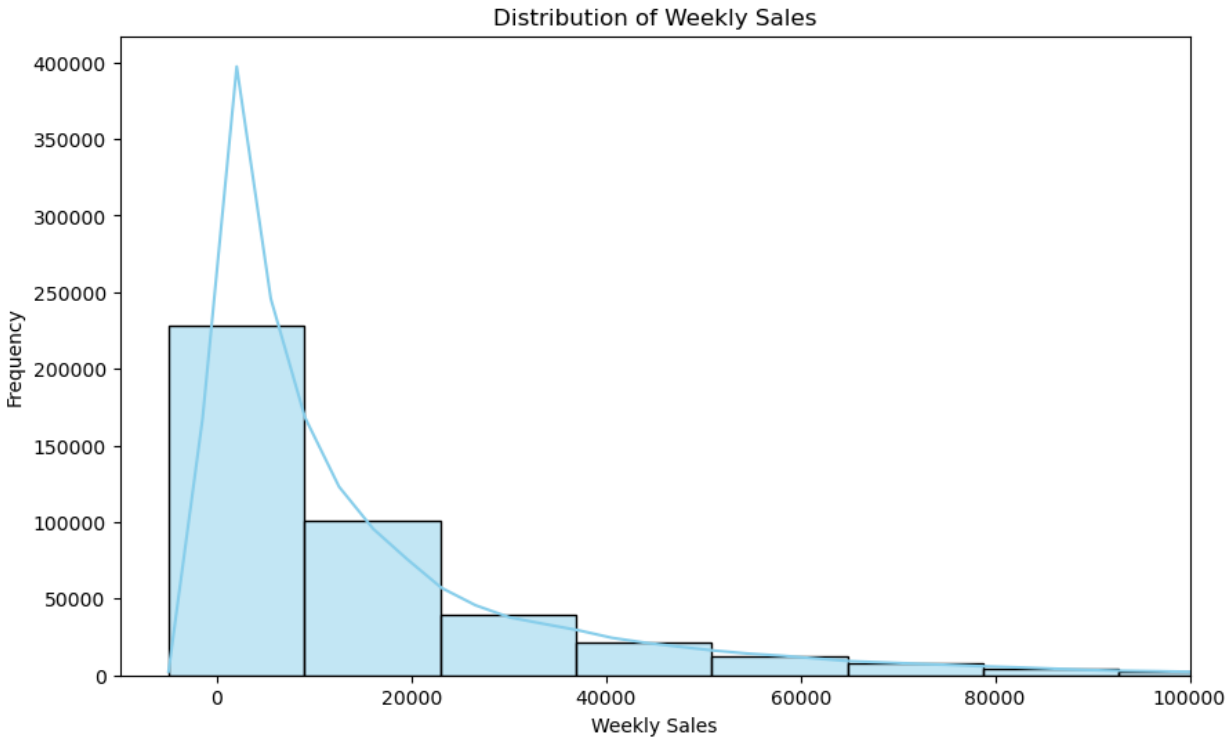
Features plot with time



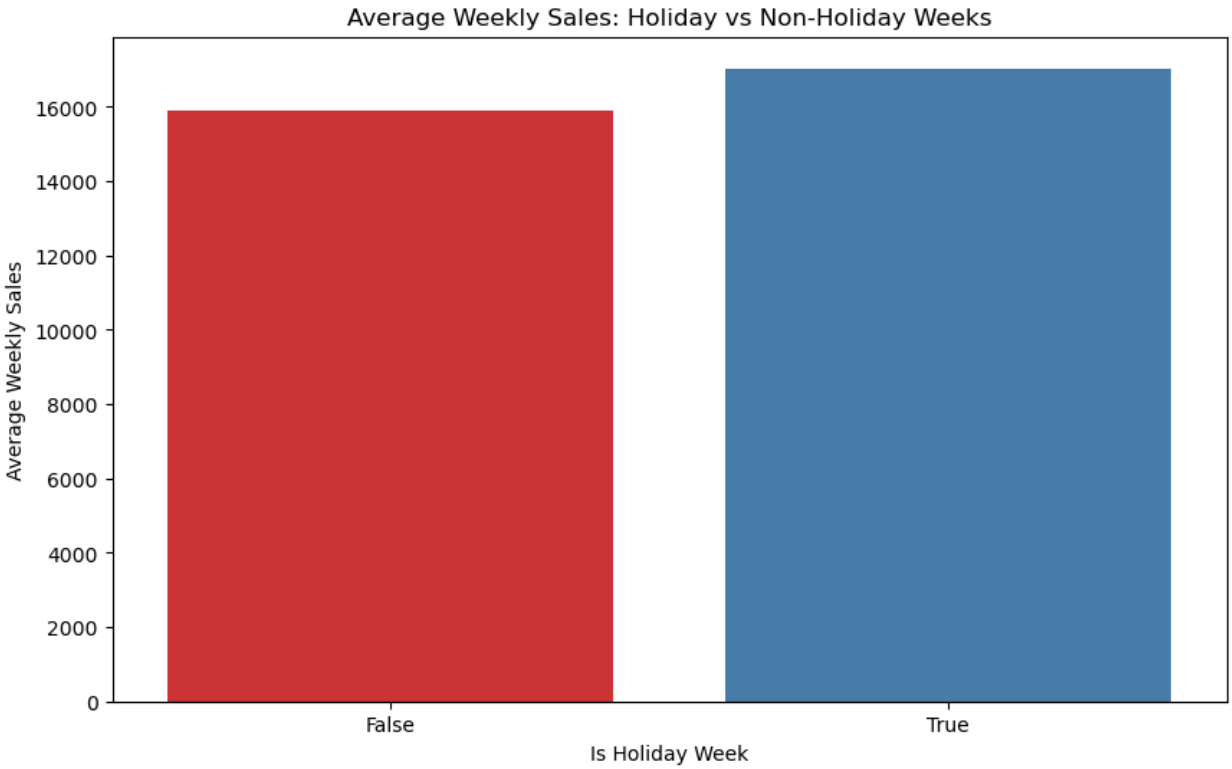
Sales Distribution by Store Type



Distribution of weekly sales



Average Weekly Sales: Holiday vs Non-Holiday Weeks



Plot of Average Weekly Sales vs. Unemployment Rate

Training loss, Validation loss, RMSE and R^2

Training Loss: 0.05331473056884611

Validation Loss: 0.09507193314796737

RMSE: 0.36817518

R^2: 0.8567684517095601

Using Dropout

Training Loss: 0.05331473056884611

Validation Loss: 0.09507193314796737

RMSE: 0.5084642

R^2: 0.7268189999610334

Using L2 Regularization

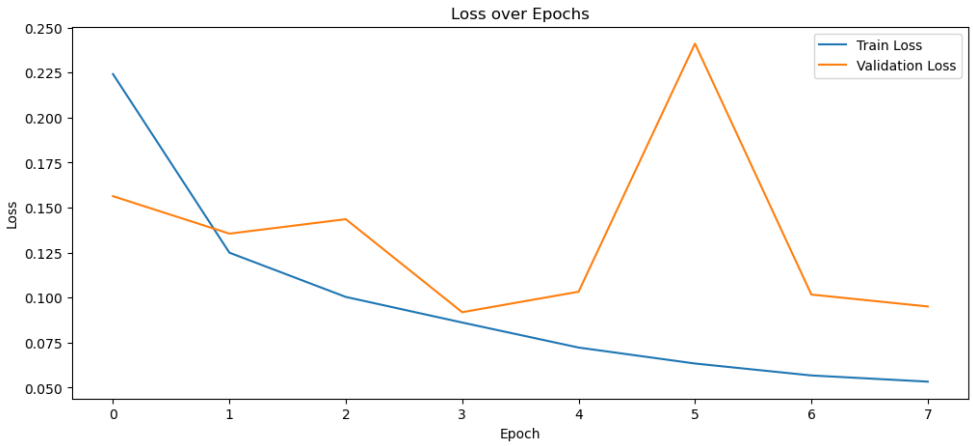
Training Loss: 0.05331473056884611

Validation Loss: 0.09507193314796737

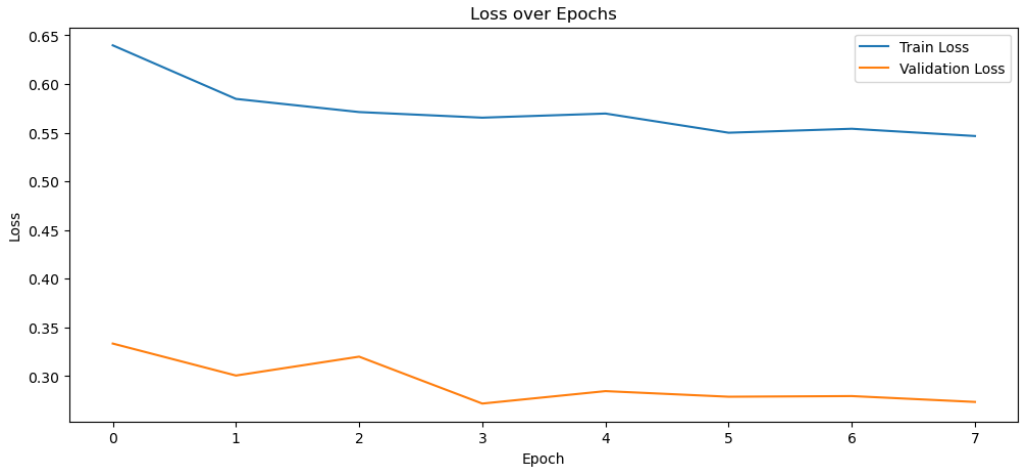
RMSE: 0.42845222

R^2: 0.806030029231339

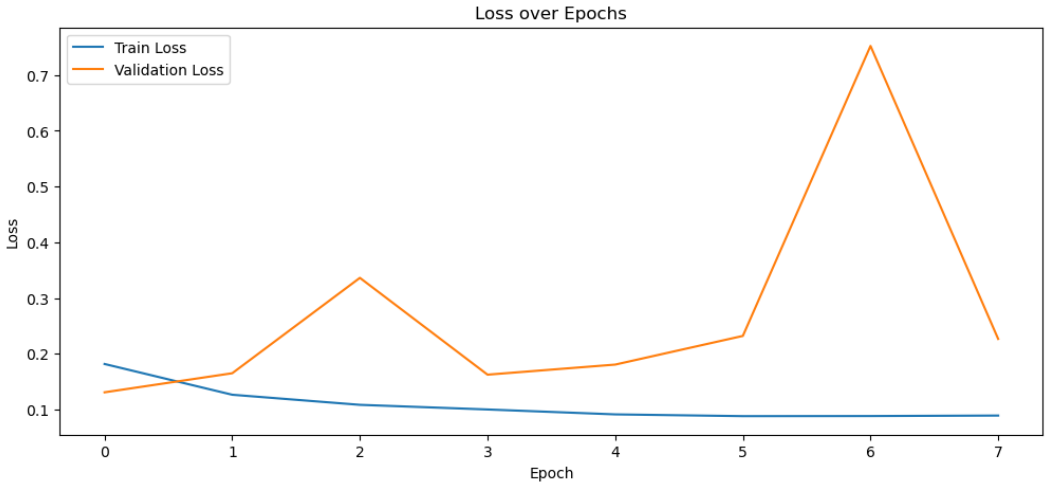
Plot of the training and validation loss over time



Using Dropout



Using L2 Regularization



Part 4: Sentiment analysis using LSTM

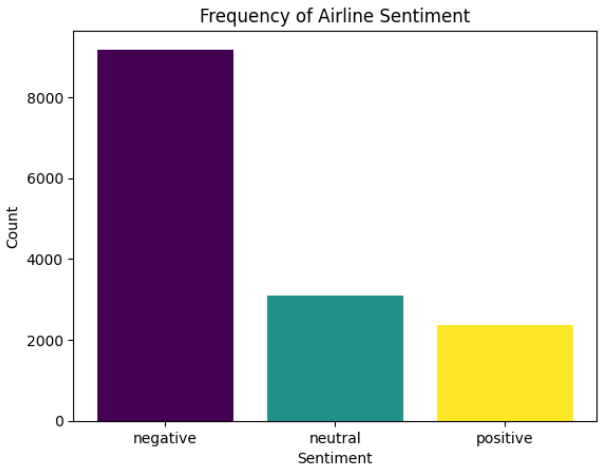
We have worked with the Twitter Airline Sentiment Dataset that tweets about airline experiences, labeled with sentiment. The features are shown in the image attached below with the dataset dimensions being 14640 rows x 15 columns,

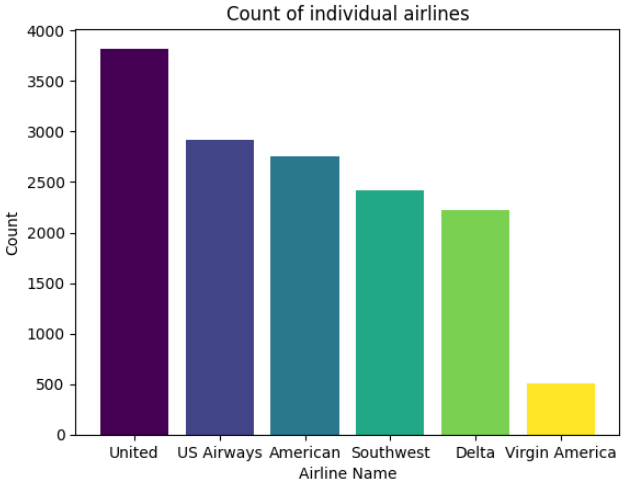
```
tweet_id                      14640
airline_sentiment             14640
airline_sentiment_confidence  14640
negativereason                 9178
negativereason_confidence     10522
airline                       14640
airline_sentiment_gold           40
name                          14640
negativereason_gold              32
retweet_count                 14640
text                          14640
tweet_coord                    1019
tweet_created                 14640
tweet_location                 9907
user_timezone                  9820
dtype: int64
```

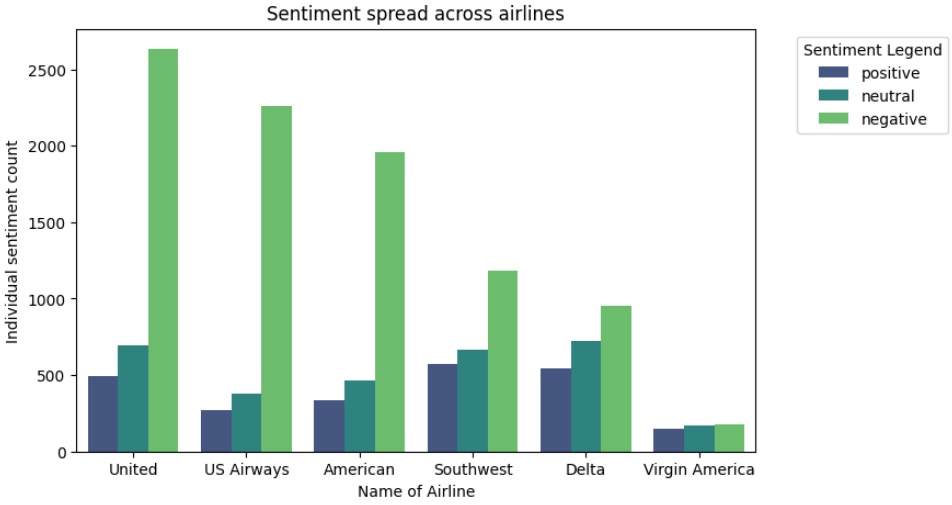
These are additional metrics of the dataset.

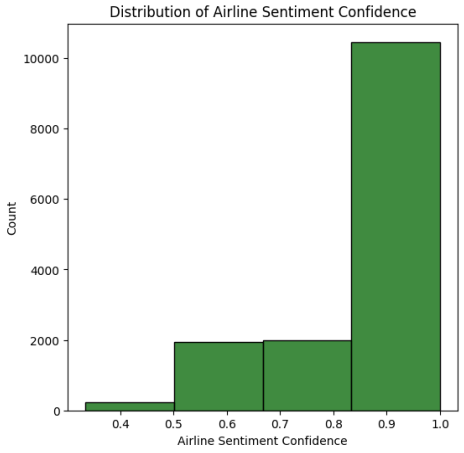
	tweet_id	airline_sentiment_confidence	negativereason_confidence	retweet_count
count	1.464000e+04	14640.000000	10522.000000	14640.000000
mean	5.692184e+17	0.900169	0.638298	0.082650
std	7.791112e+14	0.162830	0.330440	0.745778
min	5.675883e+17	0.335000	0.000000	0.000000
25%	5.685592e+17	0.692300	0.360600	0.000000
50%	5.694779e+17	1.000000	0.670600	0.000000
75%	5.698905e+17	1.000000	1.000000	0.000000
max	5.703106e+17	1.000000	1.000000	44.000000

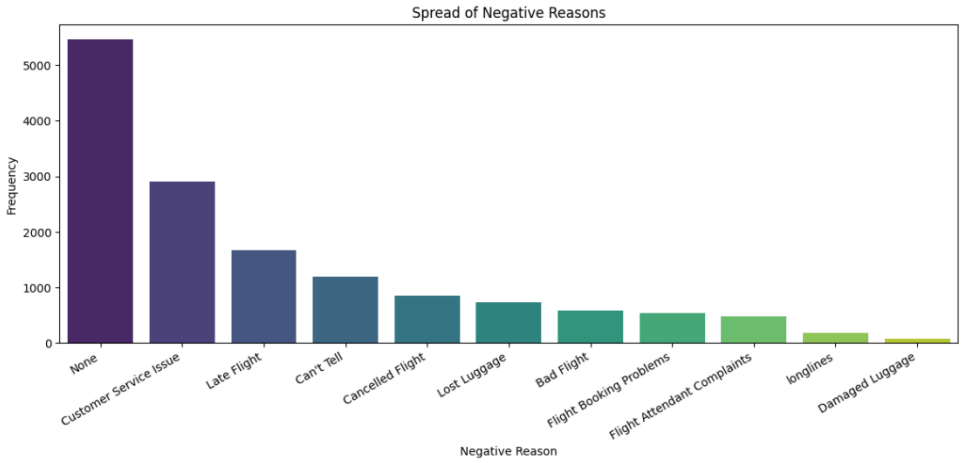
Visualization Graphs

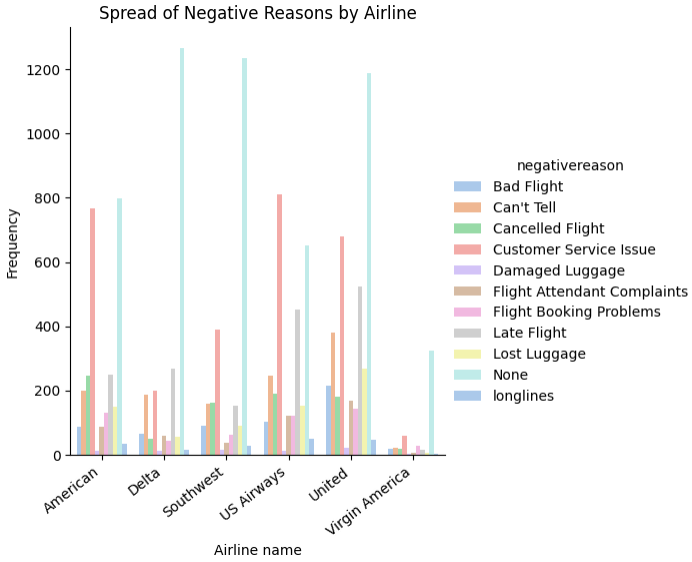



Count of individual airlines
Count
Airline Name
United
US Airways
American
Southwest
Delta
Virgin America


Sentiment spread across airlines
Individual sentiment count
Name of Airline
Sentiment Legend
positive
neutral
negative
United
US Airways
American
Southwest
Delta
Virgin America


Distribution of Airline Sentiment Confidence
Count
Airline Sentiment Confidence





LSTM Architecture

Three LSTM layers are stacked to increase the model's ability to learn from data. The output of each LSTM layer is fed into the next, with the final LSTM layer's output being used for classification. Applied after the final LSTM layer to reduce overfitting by randomly setting a fraction of the input units to 0 at each update during training. The dropout rate is set to 0.5. Maps the output of the last LSTM layer (after dropout) to the desired output size which corresponds to the number of classes for classification.

Test Accuracy: 0.7547

Test Accuracy (With dropout change) : 0.745

Test Accuracy (With lr value change) : 0.74

Improved LSTM Architecture

The LSTM layer is configured with an input size equal to the embedding size i.e., 128, a hidden size of 64. It has two layers. The dropout rate is set to 0.5. The optimizer used is Adam and the number of epochs used is 10. The embedding size is 128. There are three number of classes which are positive, neutral and negative. The model is trained using CrossEntropyLoss.

Test Accuracy: 0.7652185

Test Accuracy (With dropout change) : 0.74

Test Accuracy (With lr value change) : 0.73

Comparison of Results

The improved LSTM model, with a 76% accuracy, marginally outperforms the basic LSTM's 75% accuracy. This slight improvement can be attributed to enhancements like bidirectionality, capturing forward and backward context, and dropout regularization to prevent overfitting.

The strengths and limitations of using recurrent neural models for sentiment analysis

Recurrent Neural Networks are good for sentiment analysis due to their ability to process sequential data, allowing them to capture the context and dependencies within text, crucial for understanding sentiments influenced by word order and combinations. They handle variable-length inputs, making them suitable for analyzing texts of differing lengths without requiring manual feature engineering, as they learn features directly from raw text. However, RNNs face limitations, including the vanishing gradient problem, which affects their ability to remember information from long sequences, making them less effective for analyzing extensive texts. They are computationally intensive, limiting their speed and scalability. Furthermore, RNNs, especially more complex variants like LSTMs and GRUs, can be prone to overfitting and suffer from limited interpretability, challenging their application in scenarios requiring fast, explainable models. Despite these challenges, RNNs remain a powerful tool for sentiment analysis when applied with consideration to their limitations.

Part V: CNN & LSTM Theoretical Part

Part V.I: CNN

1. What is the output size after the first layer?

I/P images size = 32x28 RGB

W = 32

H = 28

S = 1

D = 10

P = 0

F = 5

O/P W = (W-F+2P)/S +1 = 28

O/P H = (H-F+2P)/S +1 = 24

The output size after the first layer is 28x24 for each of the 10 filters.

2. How many parameters are there in first layer?

Number of parameters per filter=(5×5×3)+1

Total number of parameters = Number of parameters per filter x10

Number of parameters in the first layer = (5x5x3+1)x10

= 760

3. What would be the output size if a padding of 1 was used instead of zero padding?

O/P W = (W-F+2P)/S +1 = (32-5+2x1)/1 +1 = 30

O/P H = (H-F+2P)/S +1 = (28-5+2x1)/1 +1 = 26

O/P size = 30x26x10

4. What would be the number of parameters if the input images were grayscale instead of RGB?

If the images are grayscale instead of RGB, the number of input channels is 1.

Hence,

Number of parameters per filter=(5×5×1)+1

Total number of parameters = Number of parameters per filter x10

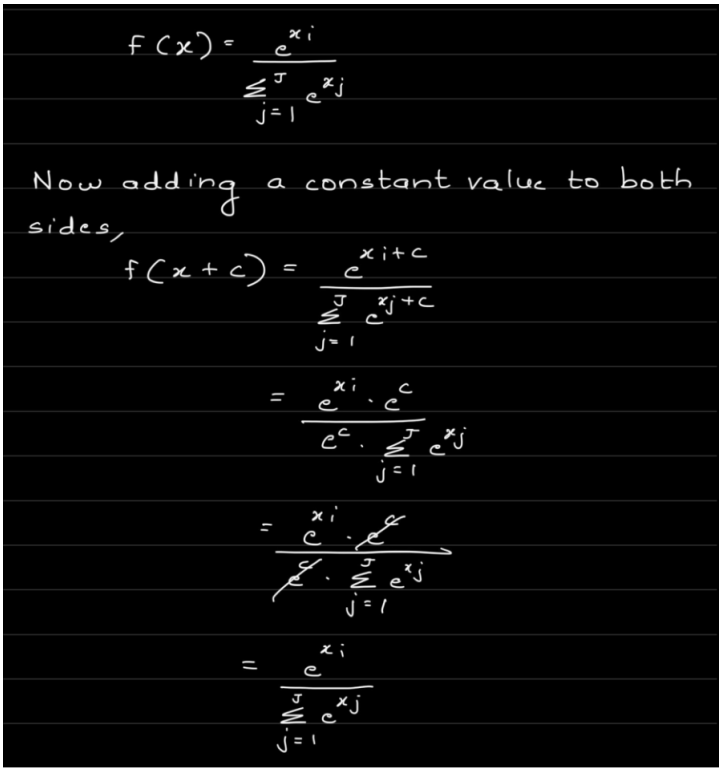
Number of parameters in the first layer = (5x5x1+1)x10

= 260

5. Considering the above specific task and the requirement of probabilistic outputs, which activation function would be most suitable for the output layer in this scenario?

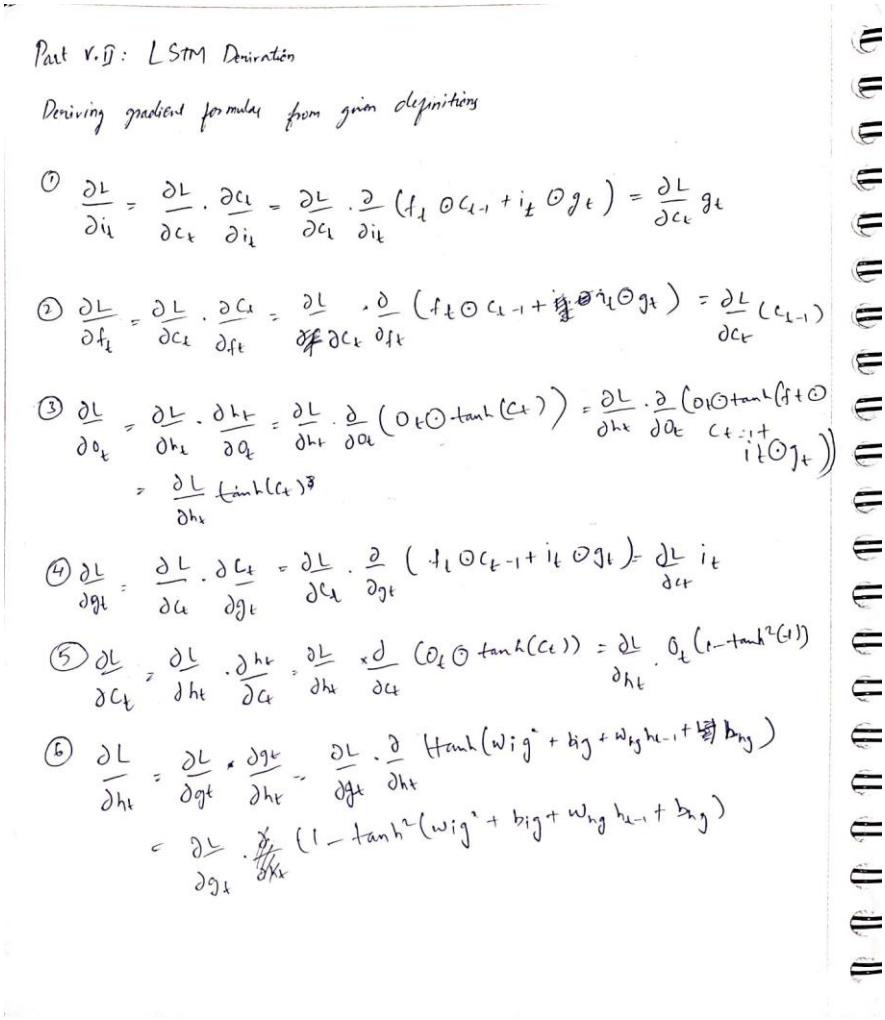
Softmax function because it can be used for multiclass classification.

6. Prove that the activation function used here is invariant to constant shifts in the input values, meaning that adding a constant value to all the input values will not change the resulting probabilities.



Hence adding a constant value to all the input values will not change the resulting probabilities.

Part V:2



References

- https://pytorch.org/hub/pytorch_vision_resnet/
- https://pytorch.org/vision/main/models/generated/torchvision.models.vgg13.html
- 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