Assignment 2

Autoencoder and Transformer Architectures

Part I: Theoretical Part – Autoencoders for Anomaly Detection in Manufacturing [10 points]

Autoencoder Architecture:

- Input layer: 1000-dimensional vectors
- Hidden layer 1: Fully connected layer with 512 units and ReLU activation
- Bottleneck layer: Fully connected layer with 32 units
- Hidden layer 2: Fully connected layer with 512 units and ReLU activation
- Output layer: Fully connected layer with 1000 units and sigmoid activation
- Autoencoder is trained using MSE loss

TASKS:

1. Calculate the total number of parameters in the autoencoder, including weights and biases.

Answer:

To calculate the total number of parameters in the autoencoder, we need to consider both the weights and the biases for each layer. The formula for calculating the parameters for a fully connected layer is:

Number of parameters = (input units x output units) + output units

Given the architecture:

Input layer to Hidden layer 1:

Weights: Each of the 1000 input units is connected to each of the 512 units in the first hidden layer. Therefore, the number of weights is 1000 x 512.

Biases: Each unit in the hidden layer has its bias, so there are 512 biases for this layer.

So the total number of the parameters for this layer are 512,512 parameters (511,488 weights + 512 biases).

Hidden layer 1 to Bottleneck layer:

Weights: Each of the 512 units in the first hidden layer is connected to each of the 32 units in the bottleneck layer, giving 512×32 weights.

Biases: Each unit in the bottleneck layer has its bias, totaling 32 biases.

So total number of the parameters for this layer are 16,416 parameters (16,384 weights + 32 biases)

Bottleneck layer to Hidden layer 2:

Weights : Each of the 32 units in the bottleneck layer is connected to each of the 512 units in the second hidden layer, leading to 32×512 weights.

Biases: Similar to the first hidden layer, there are 512 biases for the second hidden layer.

So total number of the parameters for this layer are 16,896 parameters (16,384 weights + 512 biases)

Hidden layer 2 to Output layer:

Weights: Each of the 512 units in the second hidden layer is connected to each of the 1000 output units, resulting in 512×1000 weights.

Biases: Each output unit has its bias, adding up to 1000 biases.

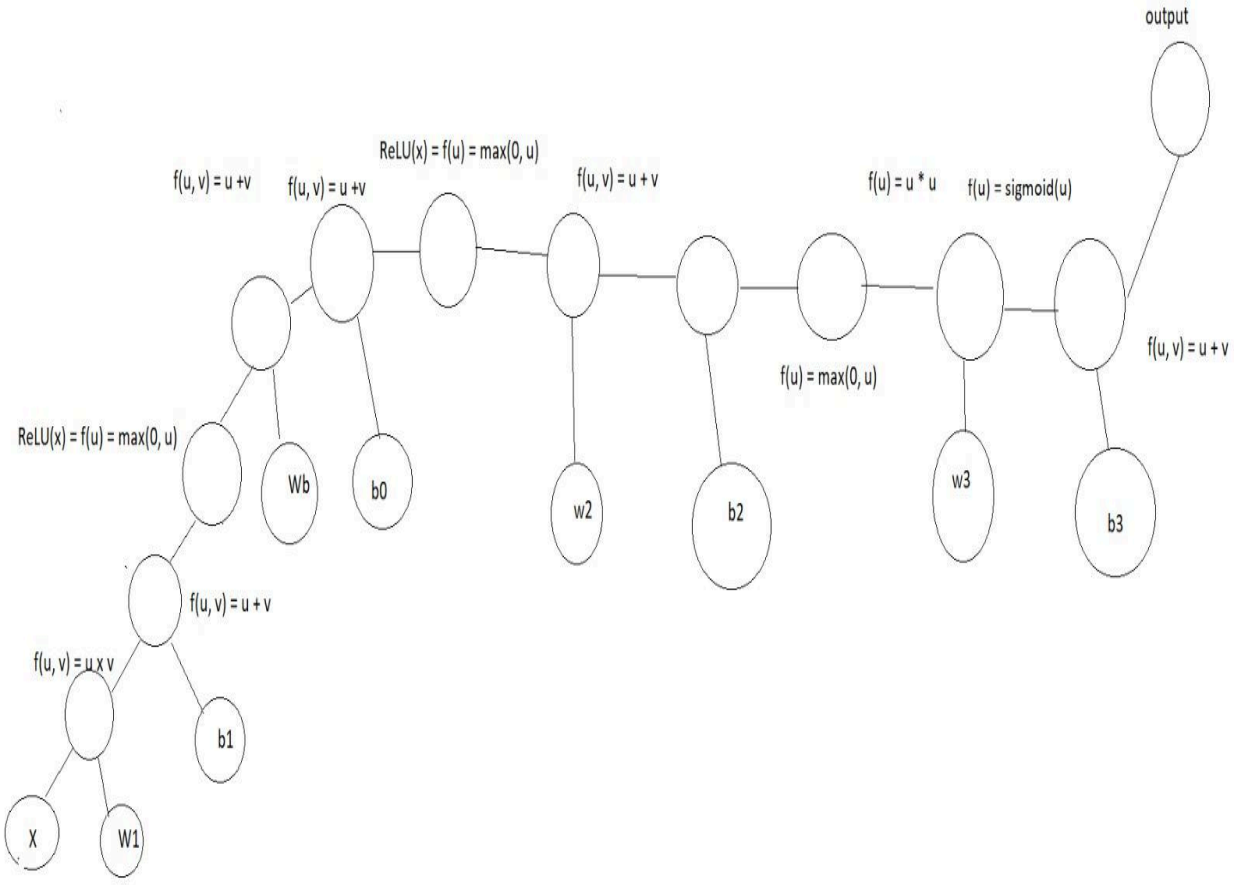
So total number of the parameters for this layer are 513,000 parameters (512,000 weights + 1000 biases)

So the total number of the parameters in the autoencoders (adding all the parameters in each layer) is 1058824.

---

2. Generate a computational graph for the autoencoder.

Answer:



3. Discuss potential challenges and limitations (at least 4) of using autoencoders for anomaly detection in manufacturing.
Answer:

Potential Challenges of using autoencoders for anomaly detection in manufacturing:

1. Data Quality and Diversity : Ensuring the training data encompasses a comprehensive range of normal operational scenarios is challenging. An autoencoder trained on limited or biased data might not accurately learn what constitutes "normal," impacting its ability to detect anomalies.

2. Complex Anomaly Patterns: Manufacturing processes can exhibit highly complex and subtle patterns that signify anomalies. Capturing and correctly interpreting these patterns, especially when they are rare or subtle, poses a significant challenge for autoencoders.

3. Real-Time Detection Requirements: In many manufacturing environments, detecting anomalies in real-time is crucial to prevent defects and maintain quality. However, training and deploying autoencoders that can process data and detect anomalies in real-time can be technically and computationally demanding.

4. Evolving Process Conditions: Manufacturing processes can evolve over time, introducing new normal conditions and types of anomalies. Continuously adapting the autoencoder to these changes without causing disruption or false detections is a persistent challenge.

Limitation of using auto encoders for anomaly detection in manufacturing:

1. Loss of Information: Autoencoders compress data into a lower-dimensional space, which can lead to the loss of critical information that might be crucial for identifying certain anomalies, especially those defined by subtle features.

2. Training Data Requirements: Autoencoders require large amounts of labeled training data to effectively learn the normal operational patterns. In many cases, obtaining such extensive, labeled datasets is difficult and costly in manufacturing environments.

3. Generalization to New Anomalies: Autoencoders trained on known types of anomalies may not generalize well to detecting new or unseen anomaly types. This limitation can reduce the model's effectiveness as new types of manufacturing defects emerge.

4: Interpretability and Actionable Insights: The "black box" nature of autoencoders makes it difficult to understand the basis of their decisions. This lack of interpretability can hinder the identification of the root causes of detected anomalies, making it challenging to take corrective actions based on the autoencoder's output alone.

4. Propose potential improvements or extensions (at least 4) to the system to enhance its effectiveness in detecting anomalies.

Answer:

Potential improvements or extensions (at least 4) to the system to enhance its effectiveness in detecting anomalies are stated below:

1. Integration of Domain Knowledge: **I**ncorporating domain knowledge into the autoencoder's design and training process can significantly improve its effectiveness. This could involve customizing the network architecture to better capture the characteristics of the manufacturing process or preprocessing data in a way that highlights relevant features for anomaly detection. Techniques such as feature engineering and the use of domain-specific loss functions can make the autoencoder more sensitive to anomalies that matter.

2. Semi-supervised Learning Approaches: Leveraging semi-supervised learning techniques can address the challenge of limited labeled data. By using a combination of a small amount of labeled data and a larger volume of unlabeled data, autoencoders can better learn the distribution of normal operations and identify outliers as potential anomalies. This approach can help in improving the model's generalization capabilities and its ability to detect new or unseen types of anomalies.

3: Ensemble Methods: Using an ensemble of autoencoders, each trained on different subsets of data or to detect different types of anomalies, can enhance detection accuracy. Ensemble methods can aggregate the outputs of multiple models to make a final decision, reducing the impact of any one model's biases or limitations. This approach can also be useful in handling evolving manufacturing processes, as new models can be added to the ensemble as needed without requiring extensive retraining.

4. Explainability and Diagnostic Features: Enhancing the model's explainability can address the interpretability challenges. Techniques like model distillation, where a simpler, more interpretable model is trained to approximate the autoencoder's output, or the use of explainability frameworks like SHAP (SHapley Additive exPlanations), can provide insights into why certain instances are flagged as anomalies. Incorporating diagnostic features into the autoencoder's architecture, such as attention mechanisms that highlight which aspects of the input contributed most to the anomaly detection, can further aid in diagnosing and addressing issues.

5: Continuous Learning and Adaptation: Implementing mechanisms for continuous learning can help the autoencoder adapt to new data, anomalies, and changes in the manufacturing process over time. Techniques such as online learning, where the model is updated in real-time with new data, or incremental learning, where the model periodically updates its parameters based on batches of new data, can ensure the model remains effective without the need for frequent, comprehensive retraining.

Part II: Autoencoders for Anomaly Detection [30 points]

1) Provide brief details about the nature of your dataset. What is it about? What type of data are we encountering? How many entries and variables does the dataset comprise?

We worked on 'nyc_taxi.csv' dataset (https://www.kaggle.com/datasets/boltzmannbrain/nab)

The dataset has 10320 rows with 2 columns, 1 column named 'timestamp' has datetime as object and another column named 'value' (integers) has the number of taxi passengers count that is collected for every 30 mins.

Some Statistics:

```
df.head(5)
```

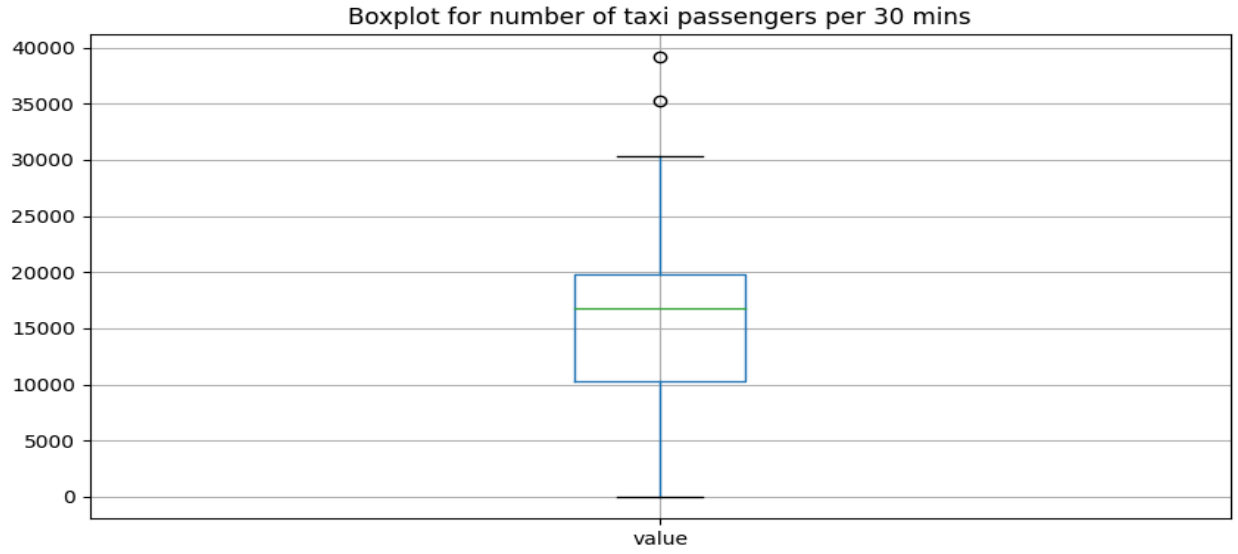
	timestamp	value
0	2014-07-01 00:00:00	10844
1	2014-07-01 00:30:00	8127
2	2014-07-01 01:00:00	6210
3	2014-07-01 01:30:00	4656
4	2014-07-01 02:00:00	3820

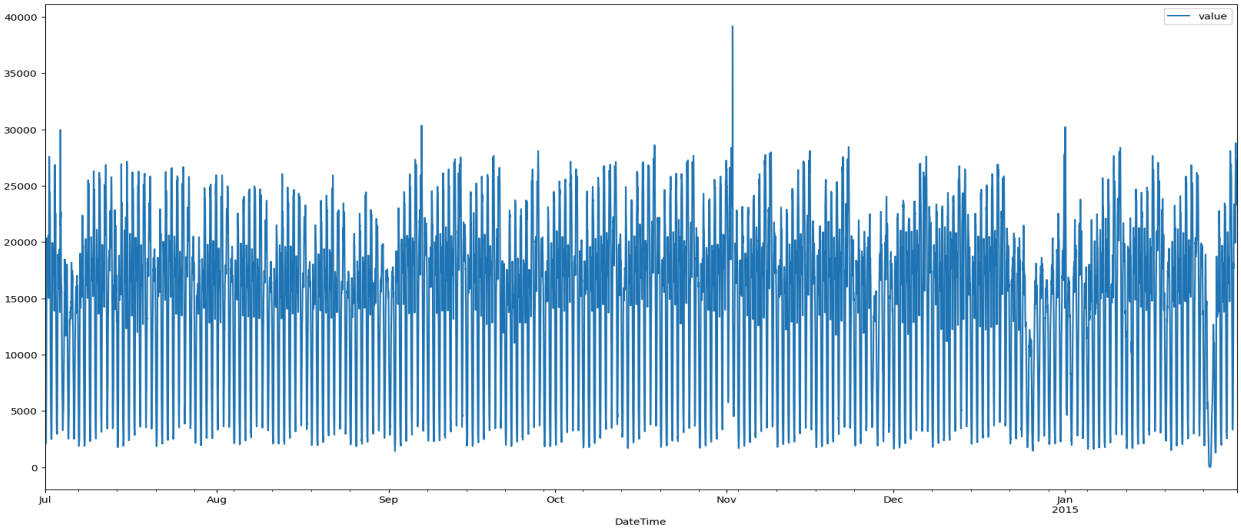
```
print(df['value'].median())
df.describe()
```

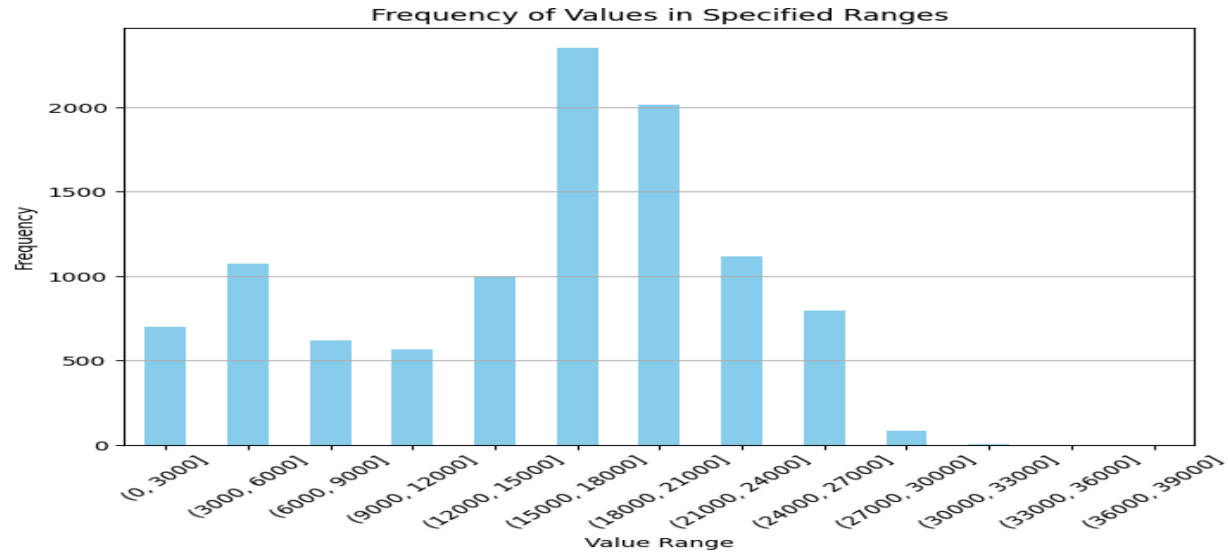
```
16778.0
```

	value
count	10320.000000
mean	15137.569380
std	6939.495808
min	8.000000
25%	10262.000000
50%	16778.000000
75%	19838.750000
max	39197.000000

Data Visualization Graphs







2) Describe the details of your autoencoder models, including the layers, activation functions, and any specific configurations employed.

Architecture 1:

The 1st model has 3 linear layers (fully connected) in the encoder block and has 3 linear layers in the decoder block. ReLU activation function is used between each Linear layer in encoder and decoder block and the output activation function in Decoder block is Sigmoid. In the encoder block, the first linear layer takes 5 input features and outputs 64 features, 2nd layer takes this and outputs 32 features, then finally the 3rd layer takes this and outputs 2 features as compressed or latent data features. The decoder block uses the exact opposite structure when compared with the layer and outputs 5 features at output, which is the same as input features.

```
AutoEncoder(
  (encoder_fc1): Linear(in_features=5, out_features=64, bias=True)
  (encoder_fc2): Linear(in_features=64, out_features=32, bias=True)
  (encoder_fc3): Linear(in_features=32, out_features=2, bias=True)
  (decoder_fc1): Linear(in_features=2, out_features=32, bias=True)
  (decoder_fc2): Linear(in_features=32, out_features=64, bias=True)
  (decoder_fc3): Linear(in_features=64, out_features=5, bias=True)
  (decoder_outputactivation): Sigmoid()
  (activation): ReLU()
)
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AutoEncoder                              [7193, 5]                 --
├─Linear: 1-1                            [7193, 64]                384
├─ReLU: 1-2                              [7193, 64]                --
├─Linear: 1-3                            [7193, 32]                2,080
├─ReLU: 1-4                              [7193, 32]                --
├─Linear: 1-5                            [7193, 2]                 66
├─Linear: 1-6                            [7193, 32]                96
├─ReLU: 1-7                              [7193, 32]                --
├─Linear: 1-8                            [7193, 64]                2,112
├─ReLU: 1-9                              [7193, 64]                --
├─Linear: 1-10                           [7193, 5]                 325
├─Sigmoid: 1-11                          [7193, 5]                 --
==========================================================================================
Total params: 5,063
Trainable params: 5,063
Non-trainable params: 0
Total mult-adds (M): 36.42
==========================================================================================
Input size (MB): 0.14
Forward/backward pass size (MB): 11.45
Params size (MB): 0.02
Estimated Total Size (MB): 11.62
==========================================================================================
```

Architecture 2:

The 2nd model has 4 linear layers (fully connected) in the encoder block and has 4 linear layers in the decoder block. ReLU activation function is used between each Linear layer in encoder and decoder block and the output activation function in Decoder block is Sigmoid. In the encoder block, first linear layer takes 5 input features and outputs 64 features, 2nd layer takes this and outputs 32 features, then finally 3rd layer takes this and outputs 16 features and finally, 4th linear layer takes this and outputs data as compressed or latent data features with 2 features. The decoder block uses the exact opposite structure when compared with the layer and outputs 5 features at output, which is the same as input features.

```
AutoEncoder2(
  (encoder_fc1): Linear(in_features=5, out_features=64, bias=True)
  (encoder_fc2): Linear(in_features=64, out_features=32, bias=True)
  (encoder_fc3): Linear(in_features=32, out_features=16, bias=True)
  (encoder_fc4): Linear(in_features=16, out_features=2, bias=True)
  (decoder_fc1): Linear(in_features=2, out_features=16, bias=True)
  (decoder_fc2): Linear(in_features=16, out_features=32, bias=True)
  (decoder_fc3): Linear(in_features=32, out_features=64, bias=True)
  (decoder_fc4): Linear(in_features=64, out_features=5, bias=True)
  (decoder_outputactivation): Sigmoid()
  (activation): ReLU()
)
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AutoEncoder2                             [7193, 5]                 --
├─Linear: 1-1                            [7193, 64]                384
├─ReLU: 1-2                              [7193, 64]                --
├─Linear: 1-3                            [7193, 32]                2,080
├─ReLU: 1-4                              [7193, 32]                --
├─Linear: 1-5                            [7193, 16]                528
├─ReLU: 1-6                              [7193, 16]                --
├─Linear: 1-7                            [7193, 2]                 34
├─Linear: 1-8                            [7193, 16]                48
├─ReLU: 1-9                              [7193, 16]                --
├─Linear: 1-10                           [7193, 32]                544
├─ReLU: 1-11                             [7193, 32]                --
├─Linear: 1-12                           [7193, 64]                2,112
├─ReLU: 1-13                             [7193, 64]                --
├─Linear: 1-14                           [7193, 5]                 325
├─Sigmoid: 1-15                          [7193, 5]                 --
==========================================================================================
Total params: 6,055
Trainable params: 6,055
Non-trainable params: 0
Total mult-adds (M): 43.55
==========================================================================================
Input size (MB): 0.14
Forward/backward pass size (MB): 13.29
Params size (MB): 0.02
Estimated Total Size (MB): 13.46
==========================================================================================
```

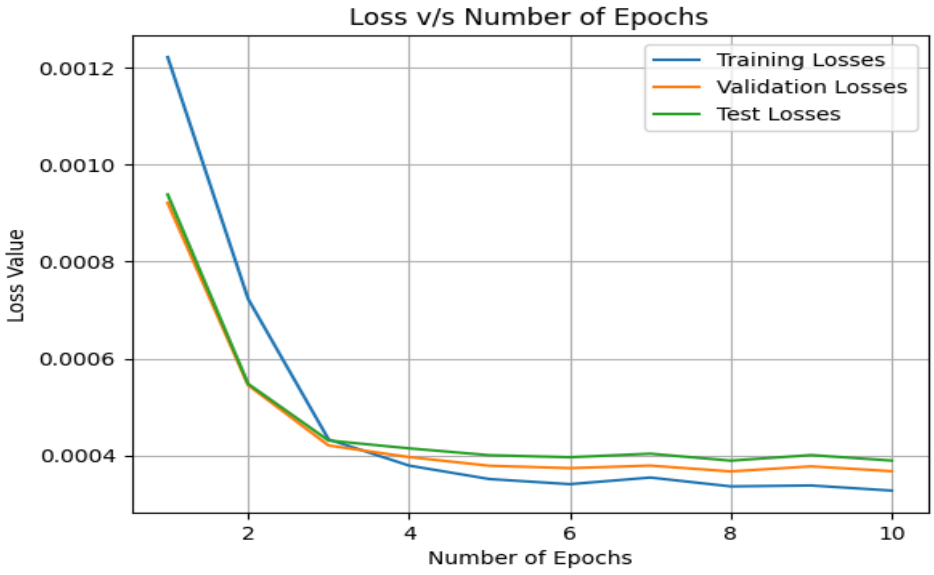
Architecture 3:

The 3rd model uses the LSTM layer in both the encoder and decoder block. In LSTM in the encoder block, it takes 5 features as input nodes and outputs 2 features as compressed data or latent space data. In LSTM in the decoder block, it takes 2 features as input nodes and outputs 5 features as decompressed data, signifying the number of features in the input.

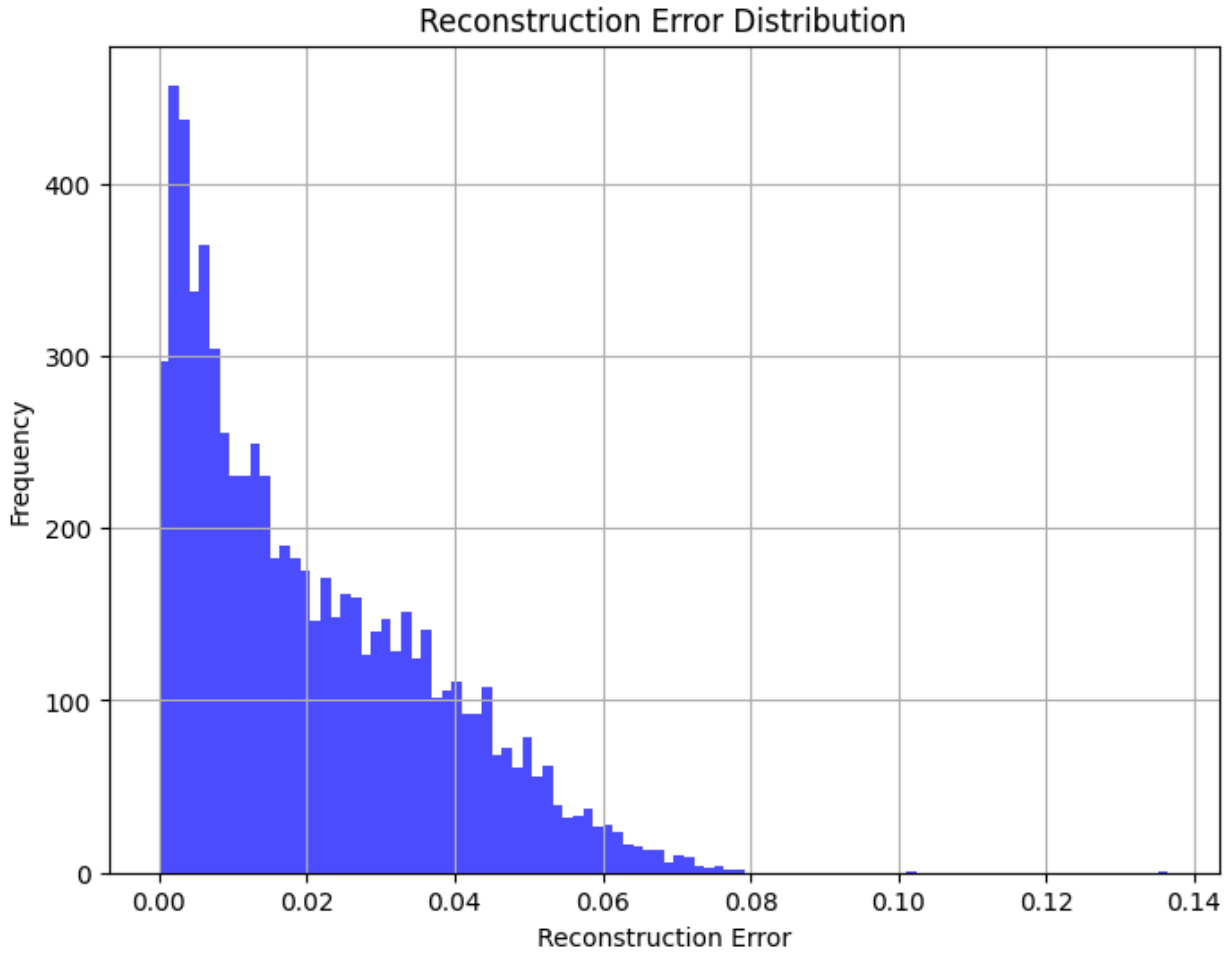
```
AutoEncoderLSTM(
  (encoder_lstm): LSTM(5, 2, batch_first=True)
  (decoder_lstm): LSTM(2, 5, batch_first=True)
  (activation): ReLU()
)
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
AutoEncoderLSTM                          [1, 5]                    --
├─LSTM: 1-1                              [7193, 2]                 72
├─LSTM: 1-2                              [1, 5]                    180
==========================================================================================
Total params: 252
Trainable params: 252
Non-trainable params: 0
Total mult-adds (M): 1.04
==========================================================================================
Input size (MB): 0.14
Forward/backward pass size (MB): 0.12
Params size (MB): 0.00
Estimated Total Size (MB): 0.26
==========================================================================================
```

3)Discuss the results and provide relevant graphs:
a. Report training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss.
b. Plot the training and validation accuracy over time (epochs).
c. Plot the training and validation loss over time (epochs).
d. Generate a confusion matrix using the model's predictions on the test set.
e. Report any other evaluation metrics used to analyze the model's performance on the test set.

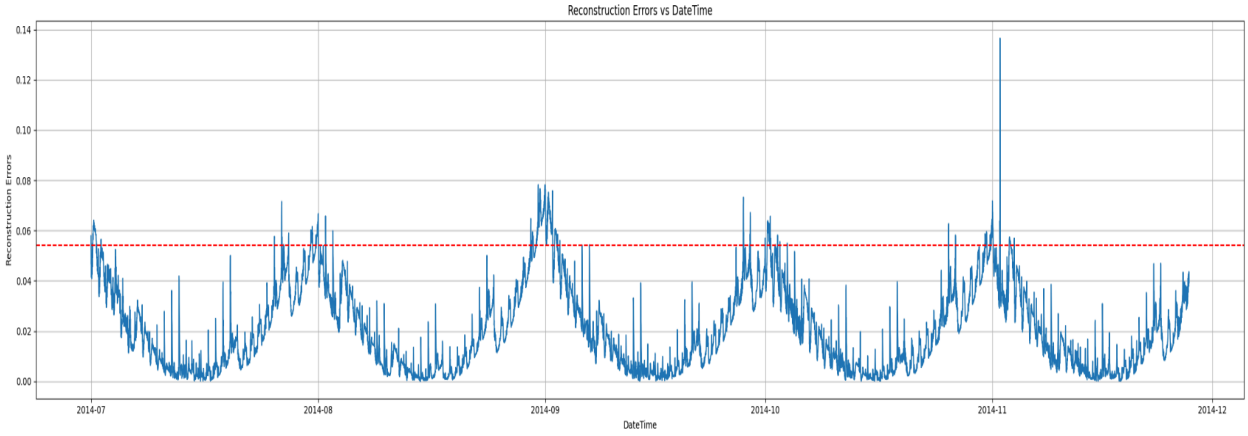
Architecture 1:
Loss vs Epoch Graph:



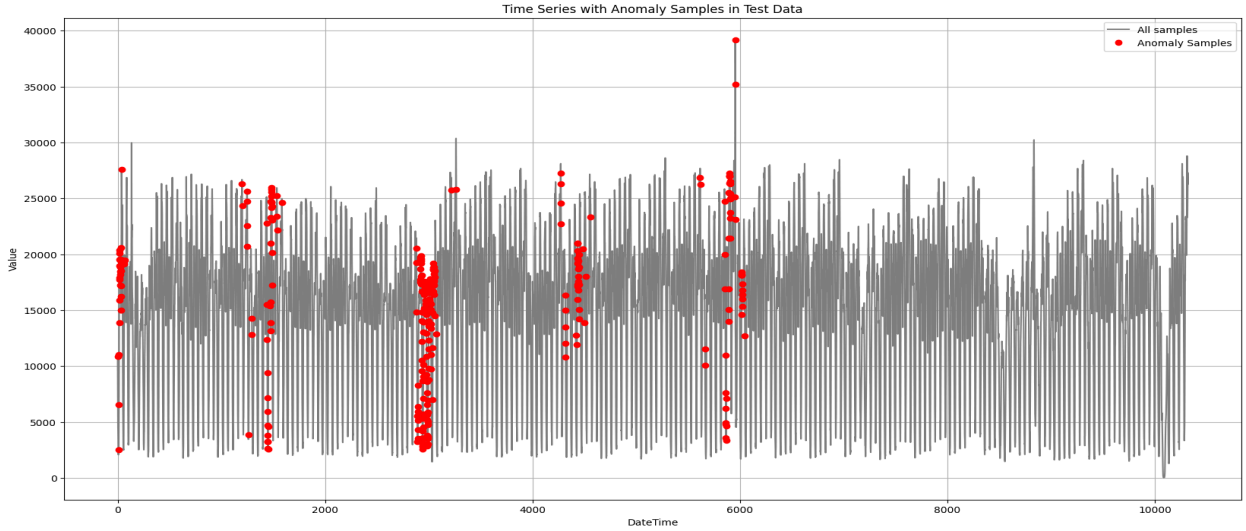
Reconstruction Error Distribution:



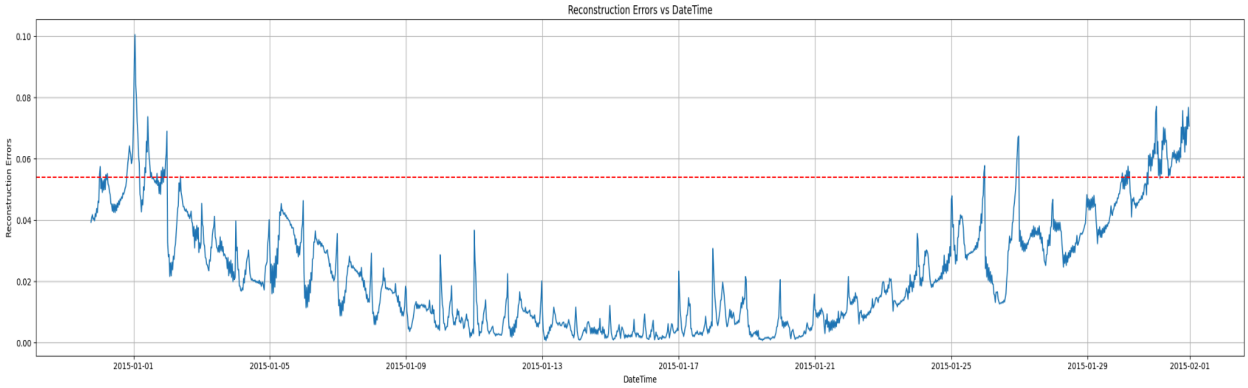
Time Series Analysis of Reconstruction Error on Train Data:



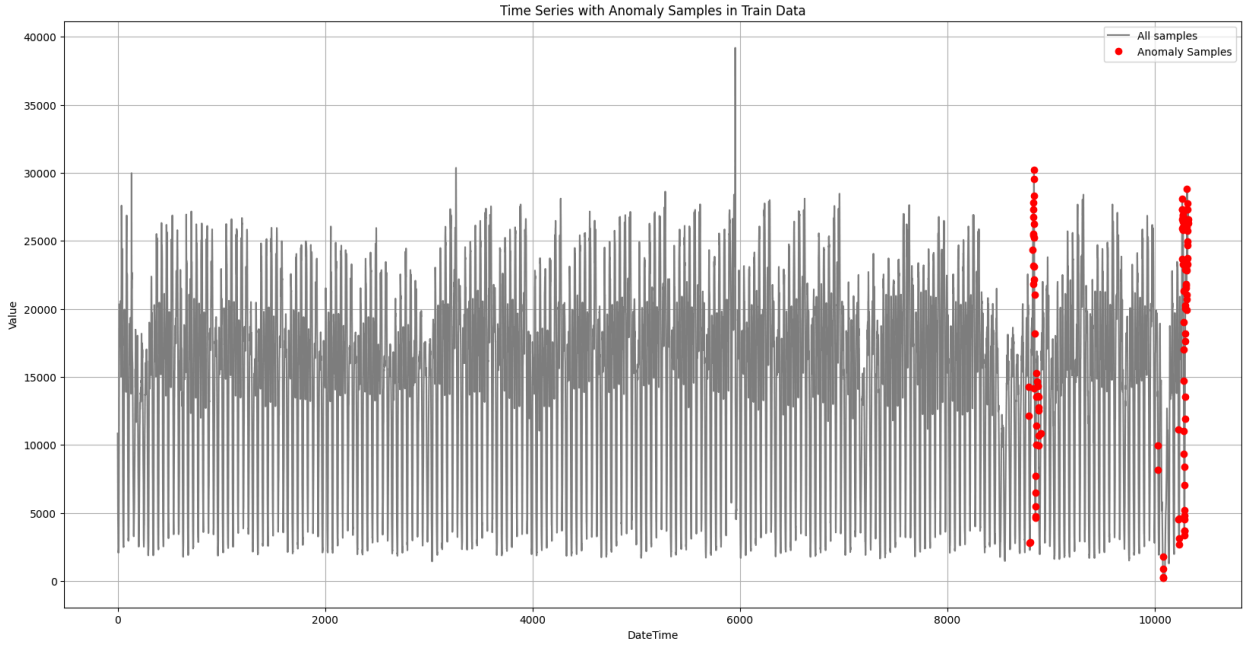
Anomaly Detection on Time Series Analysis on Train Data:



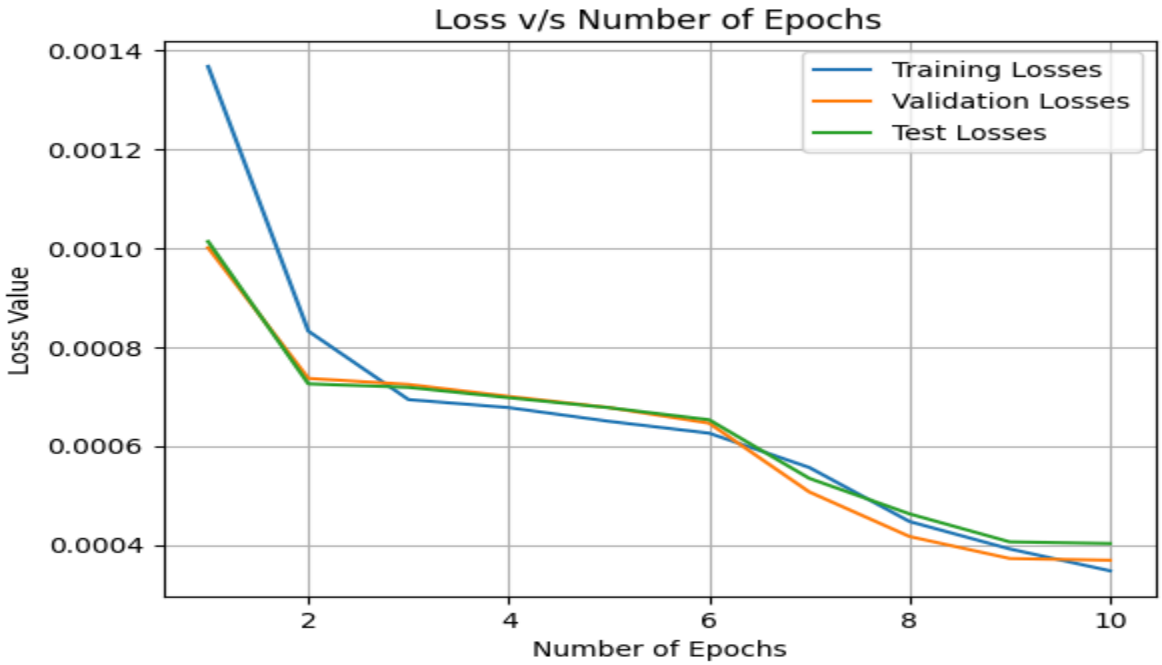
Time Series Analysis of Reconstruction Error on Test Data:



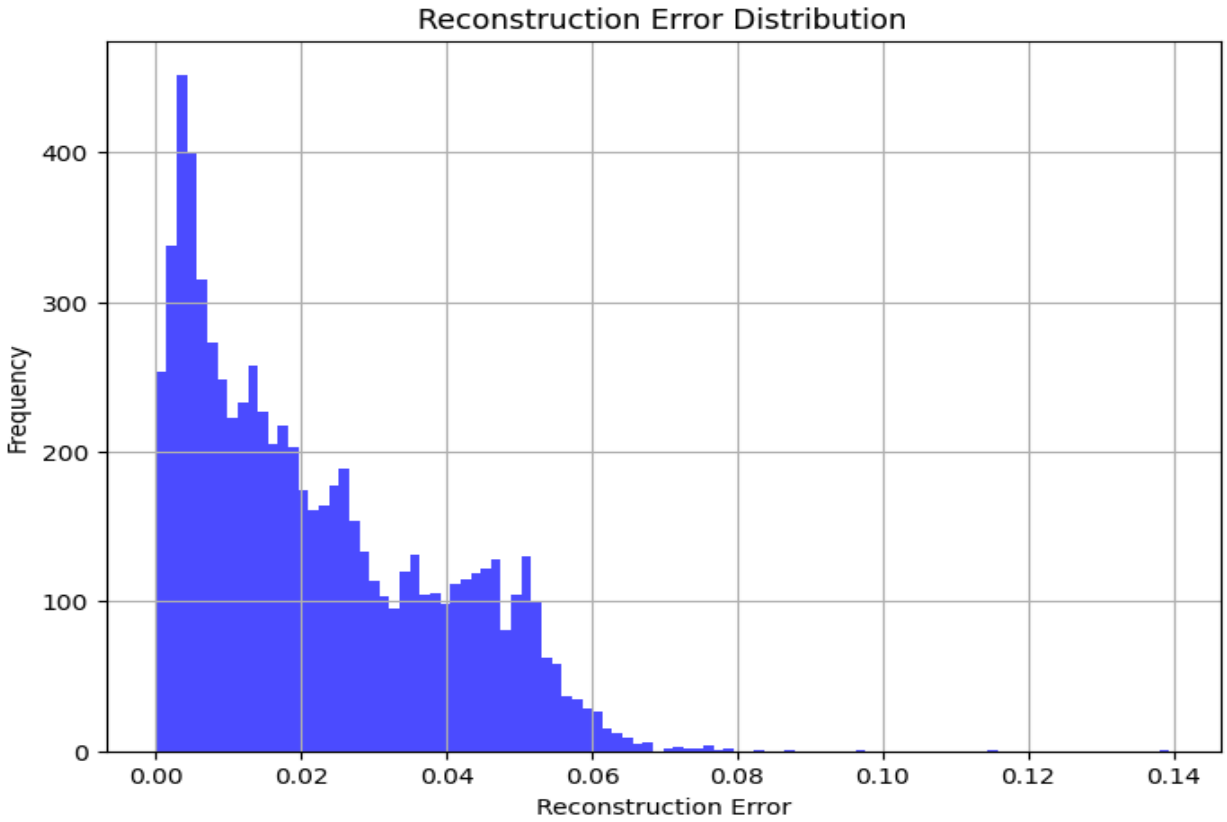
Anomaly Detection on Time Series Analysis on Test Data:



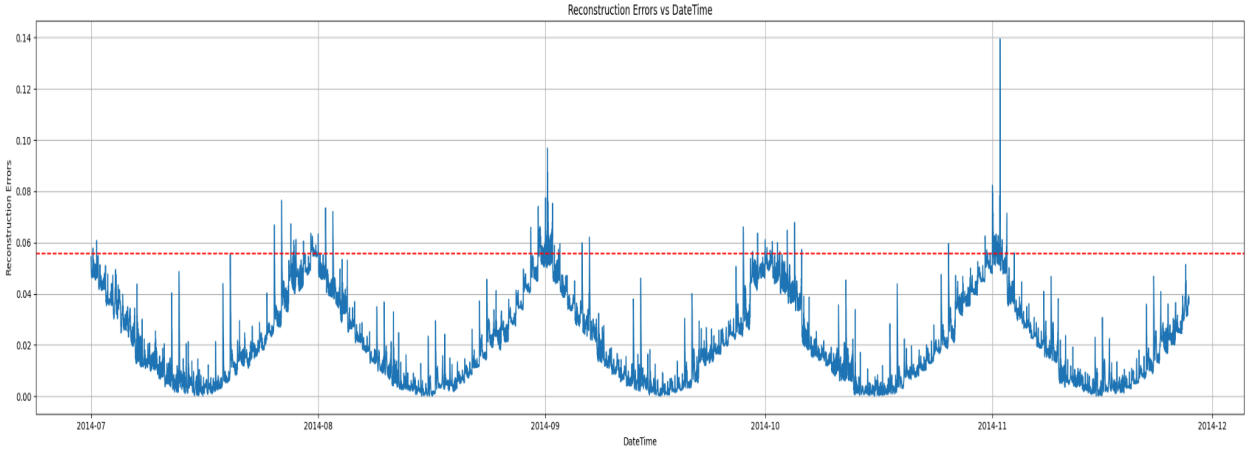
Architecture 2:
Loss vs Epoch Graph:



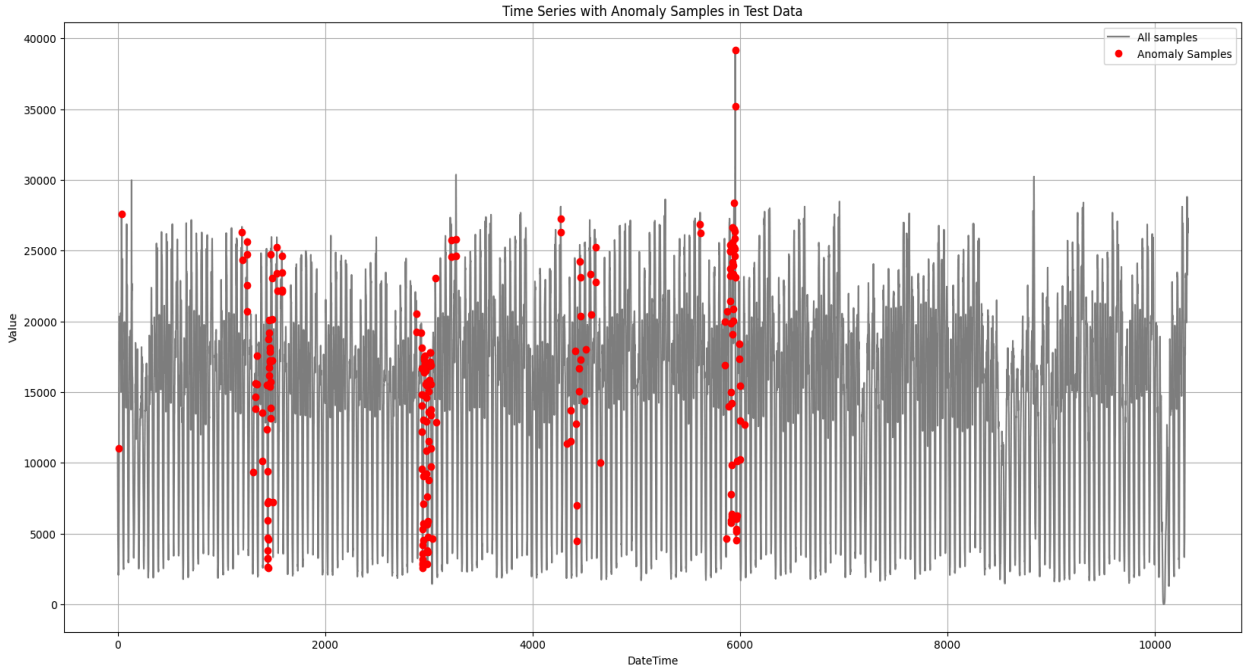
Reconstruction Error Distribution:



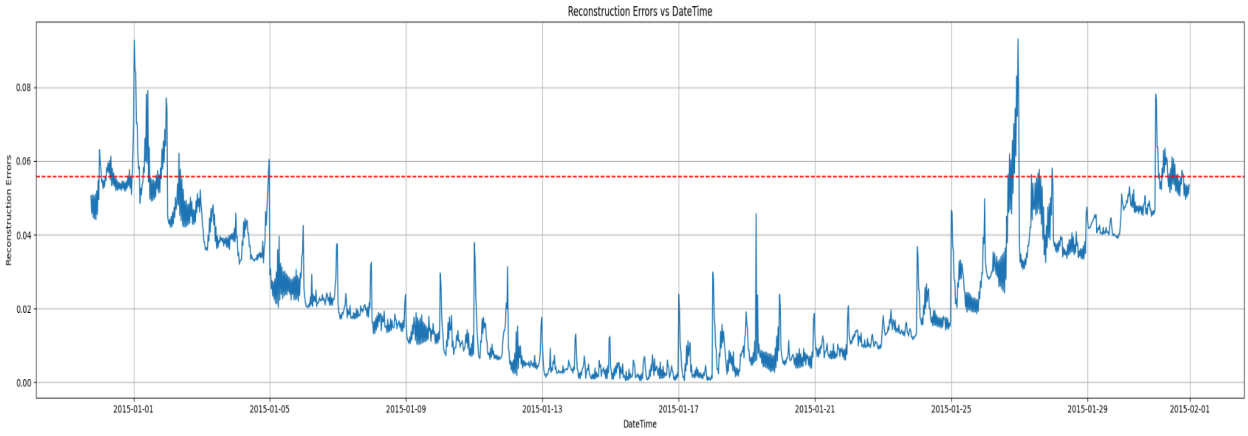
Time Series Analysis of Reconstruction Error on Train Data:



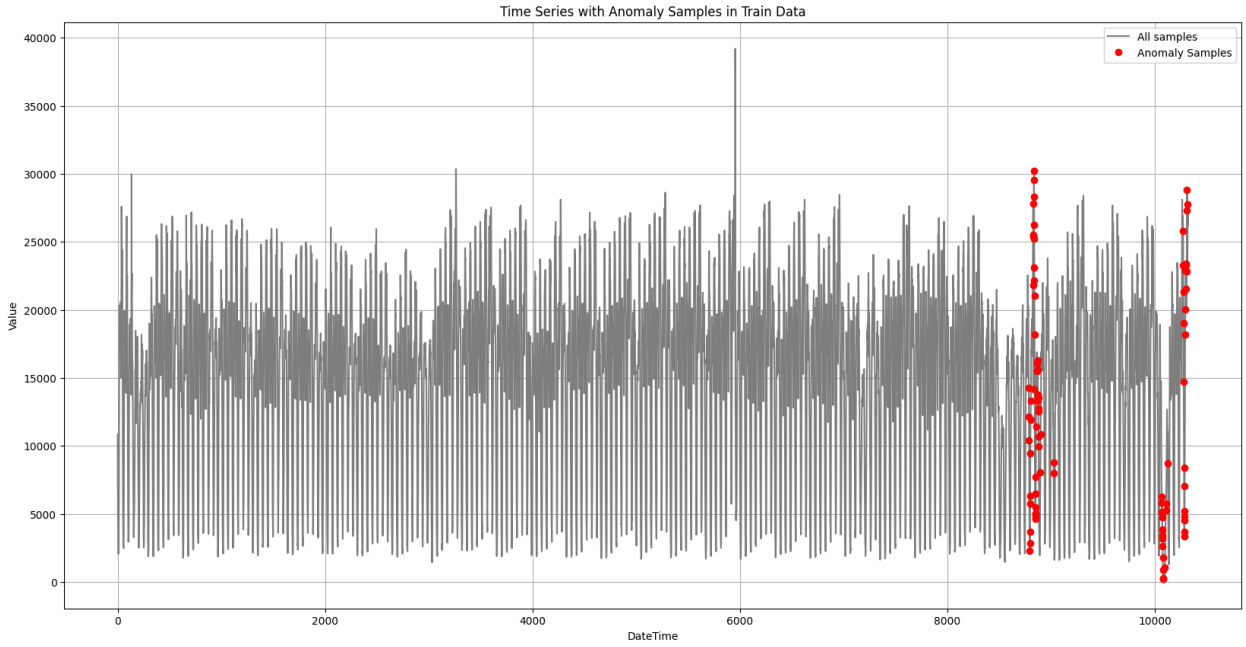
Anomaly Detection on Time Series Analysis on Train Data:



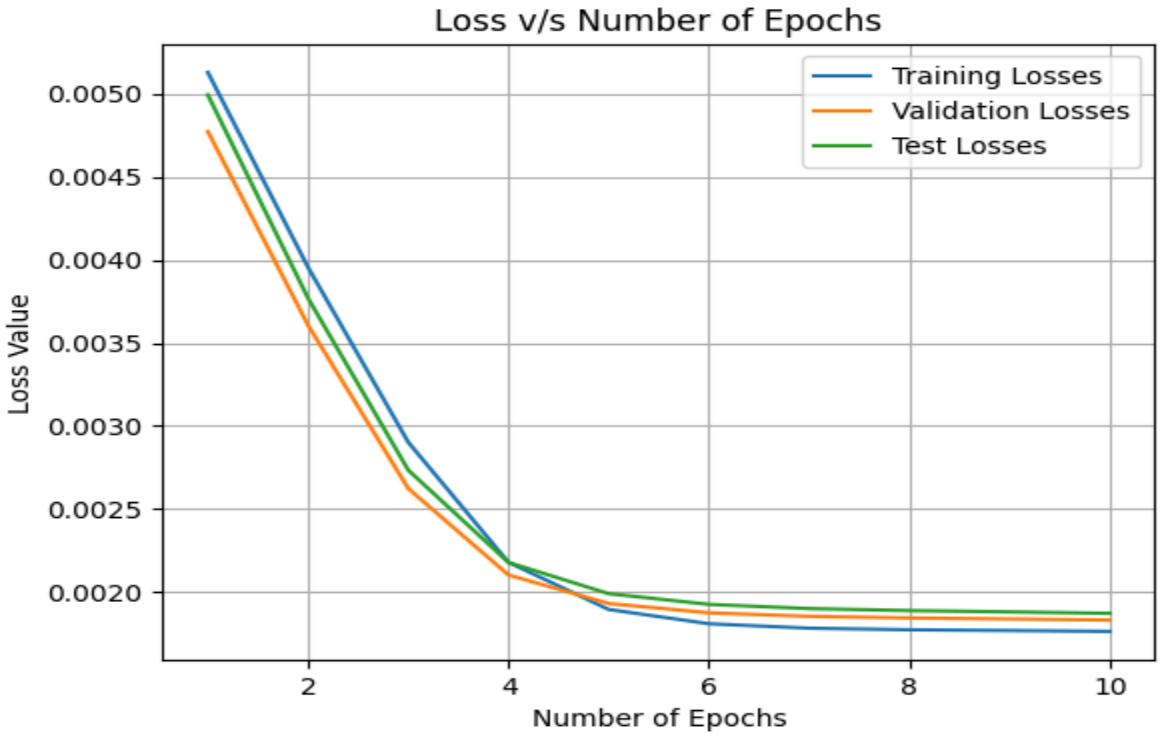
Time Series Analysis of Reconstruction Error on Test Data:



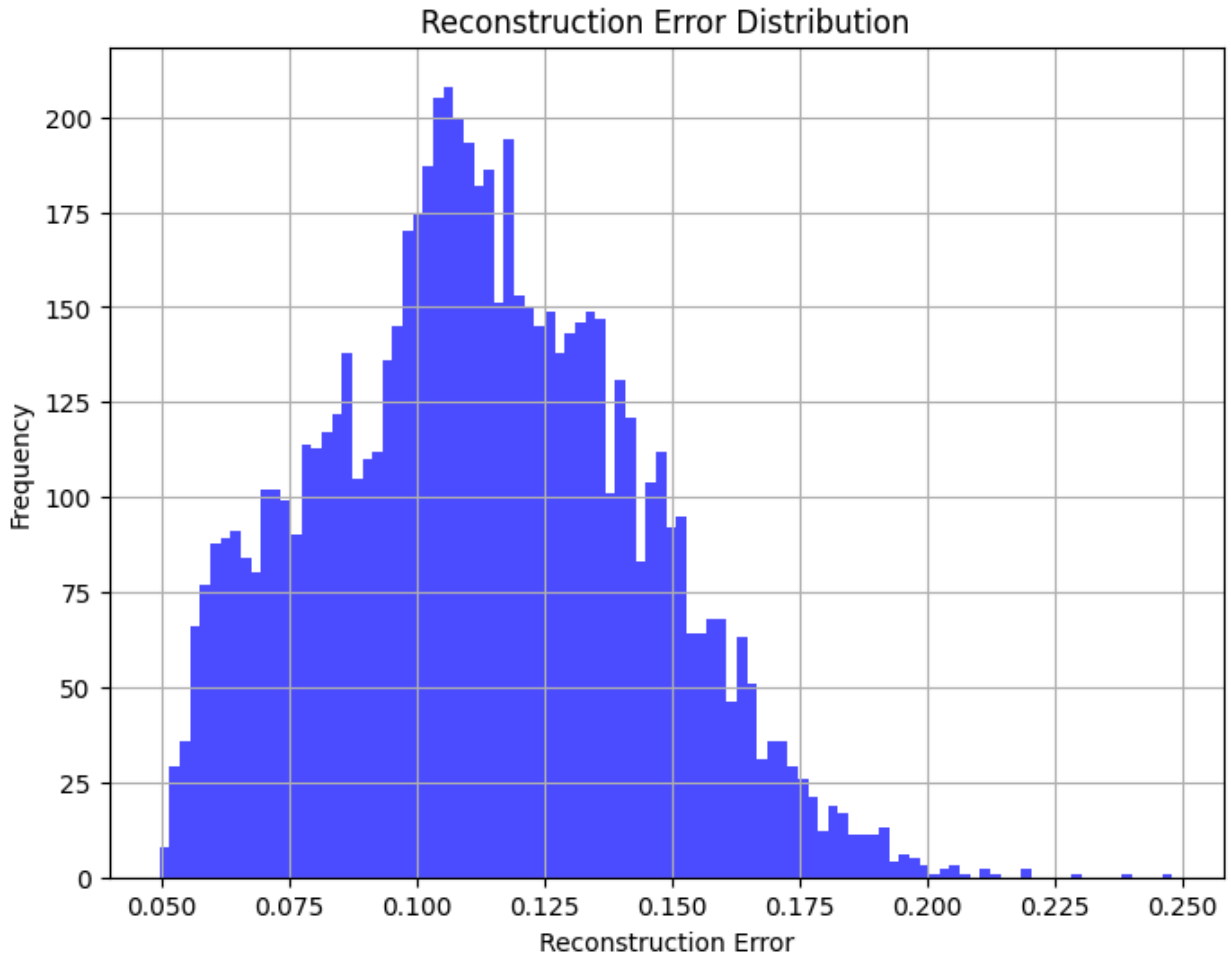
Anomaly Detection on Time Series Analysis on Test Data:



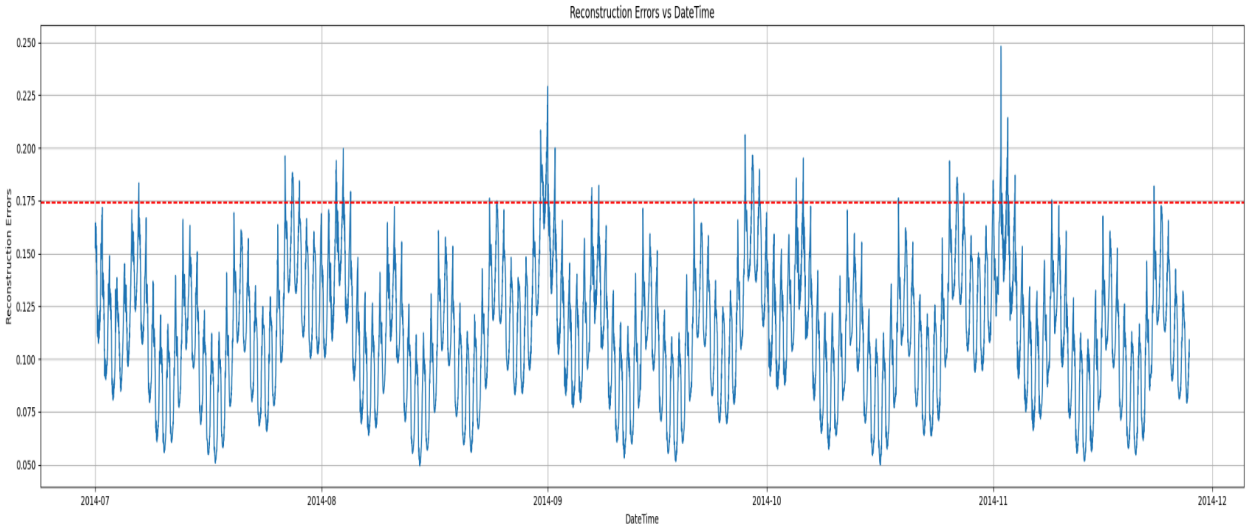
Architecture 3:
Loss vs Epoch Graph:



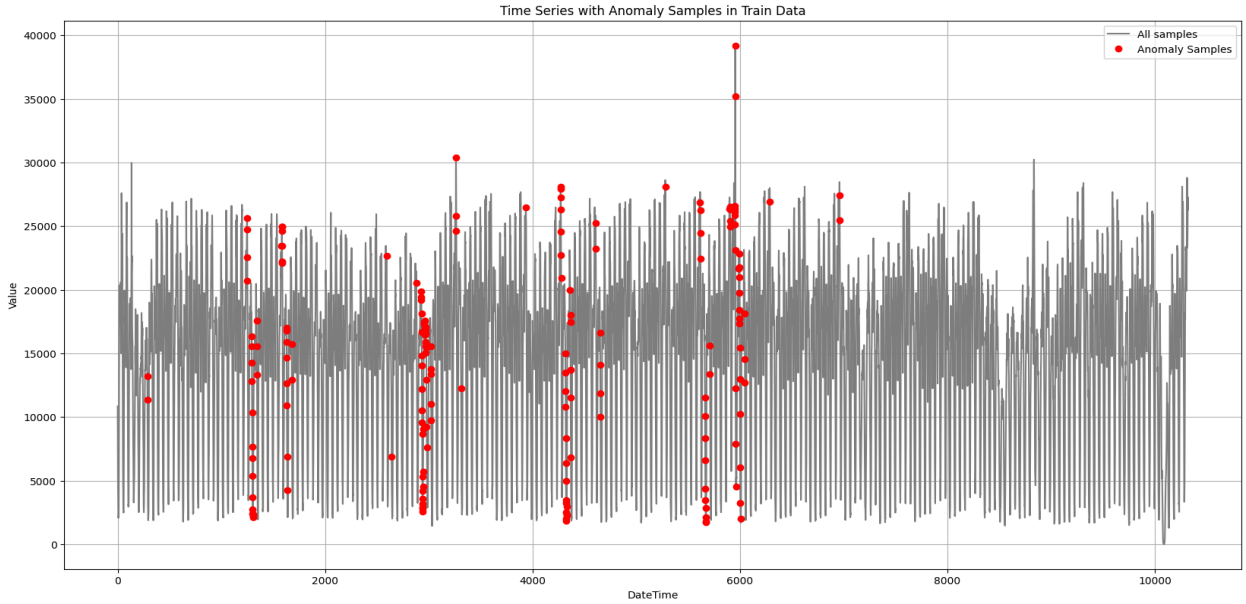
Reconstruction Error Distribution:



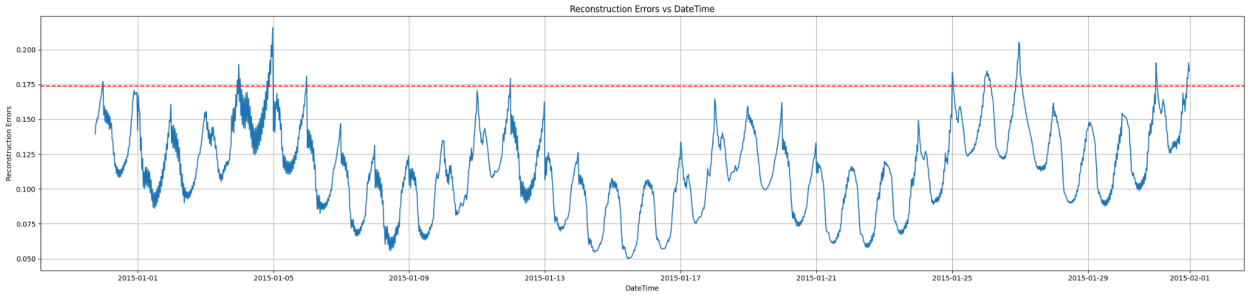
Time Series Analysis of Reconstruction Error on Train Data:



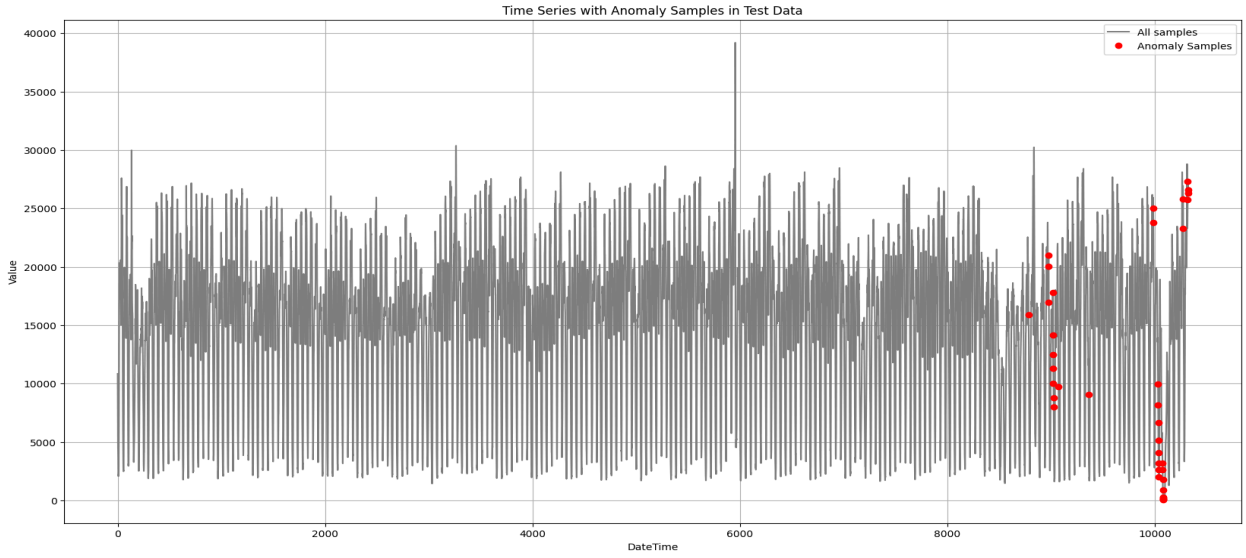
Anomaly Detection on Time Series Analysis on Train Data:



Time Series Analysis of Reconstruction Error on Test Data:



Anomaly Detection on Time Series Analysis on Test Data:



4. Discuss the strengths and limitations of using autoencoders for anomaly detection.

Answer:

Strengths:

1. Non-linear Mapping: Autoencoders can capture complex non-linear relationships in data, making them suitable for detecting anomalies in high-dimensional and non-linear datasets where simple statistical methods might fail.

2. Unsupervised Learning: Autoencoders can learn patterns and representations from unlabeled data, which is beneficial for anomaly detection tasks where labeled anomaly data might be scarce or unavailable.

3. Feature Learning: Autoencoders inherently learn a compressed representation of the input data, which can be interpreted as a set of features capturing the most important aspects of the data. These learned features can be effective for detecting anomalies by highlighting deviations from normal patterns.

4. Robust to Noise: Autoencoders are generally robust to noisy input data. They can reconstruct inputs even in the presence of noise, which can help in identifying anomalous patterns that may be obscured by noise.

5. Adaptability: Autoencoders can be tailored to specific types of data and anomalies by adjusting the architecture, loss function, and training process. This adaptability makes them suitable for various anomaly detection tasks across different domains.

Limitations:

1. Difficulty in Setting Thresholds: One of the challenges in using autoencoders for anomaly detection is determining an appropriate threshold for identifying anomalies. Setting this threshold often requires domain knowledge and experimentation, and it can be subjective.

2. Limited to Detected Patterns: Autoencoders are effective at detecting anomalies similar to those present in the training data. However, they may struggle with detecting novel or previously unseen anomalies that deviate significantly from learned patterns.

3. Overfitting: Autoencoders can potentially overfit to the normal patterns present in the training data, especially in cases where anomalies are rare. This might result in a high false positive rate or limited generalization to unseen data.

4. Complexity and Computational Cost: Training deep autoencoder models with large datasets can be computationally expensive and time-consuming. Additionally, tuning hyperparameters and selecting an appropriate architecture can add to the complexity of implementation.

5. Interpretability: While autoencoders learn meaningful representations of the data, interpreting these representations and understanding why a particular instance is flagged as an anomaly can be challenging, especially in deep or complex architectures.

Part III: Theoretical Part – Transformers [20 points]

1. Break down the mathematical operations involved in self-attention. Explain how the input sequence $x = (x1, x2, \ldots, xN)$ is processed through these stages:
• Linear transformations. Describe how three linear transformations project the input sequence into:

- **Query (q): this vector plays the role of "asking questions" about the sequence elements.**
- **Key (k): this vector helps determine which elements in the sequence are relevant for answering the queries.**
- **Value (v): this vector contains the actual information from each element in the sequence.**

Briefly explain how these transformations are typically performed using weight matrices (denoted by W_q ,W_k,and W_v) and bias vectors (optional).

1. Linear Transformations:
Answer:

-Query (q) Transformation: The Query vectors are representations of the input sequence that play the role of "asking questions" about other elements in the sequence. Each element of the sequence is transformed into a query vector, which is then used to score how well it matches with all key vectors.

This can be calculated using the following mathematical operation:

$$\mathbf{q} = \mathbf{W_q} \bullet \mathbf{x_i} + \mathbf{b_q}$$

-Key Transformation(k): The Key vectors help determine the relevance or "match" between different elements in the sequence. The compatibility between a query and all keys is calculated to understand which elements are relevant for answering the query.

This can be calculated using the following mathematical operation:

$$\mathbf{k} = \mathbf{W_k} \bullet \mathbf{x_i} + \mathbf{b_k}$$

-Value Transformations(v): The Value vectors contain the actual information from each element in the sequence that will be aggregated into the final output, based on the computed attention weights.

This can be calculated using the following mathematical operation:

$$\mathbf{v} = \mathbf{W_v} \bullet \mathbf{x_i} + \mathbf{b_v}$$

Explanation how these transformations are performed:

The linear transformations for queries, keys, and values are typically performed using separate weight matrices (W_k, W_v, W_q). that are learned during the training process. Each input vector x_i, in the sequence is multiplied by these weight matrices to produce the corresponding q_i, k_i, v_i vectors. The bias vectors (b_q, b_k and b_v) are optional and, when used, are added after the matrix multiplication to introduce an additional degree of freedom, helping the model to learn more complex patterns.

This setup allows the model to dynamically focus on different parts of the input sequence by adjusting the attention weights based on the compatibility of queries and keys, ultimately using the values to construct the output. The transformations are fundamental to the self-attention mechanism, enabling it to capture dependencies regardless of their distance in the input sequence.

2. Scaled dot-product attention. Explain how the model calculates attention scores using the query (q_i) and key (k_j) vectors. Include the formula for this step and discuss the role of the square root of the key vector dimension (d_k) in the normalization process.

Answer:

The scaled dot-product attention mechanism computes attention scores by measuring the similarity between each query and all key vectors. This similarity score determines how much focus, or "attention," should be allocated to the corresponding values when constructing the output of the attention mechanism.

Given query vectors q_i and the key vectors k_j for I, j = 1, ….N, where N is the sequence length, the attention score between i^{th} and the j^{th} key is computed as the dot product

$$q_i . k_j .$$

This dot product measures the compatibility or similarity between the query and the key, serving as a basis for determining how much attention should be paid to the corresponding value when computing the output.

To prevent the softmax function from entering regions where it has extremely small gradients (which can occur with large values of the dot product), the dot products are scaled down by a factor of $\sqrt{d_k}$ where d_k is the dimensionality of the key vectors. The formula for calculating the scaled dot-product attention score for a pair of query and key vectors is:

$$\text{Attention}(q_i, k_j) = (q_i . k_j) / (\sqrt{d_k})$$

Role of square root of the key vector dimension (d_k) in the normalization process:

The role of $\sqrt{d_k}$ where $\sqrt{d_k}$ is the dimension of the key vectors, in the scaled dot-product attention mechanism, is to normalize the magnitude of the dot products between query and key vectors. This normalization prevents the softmax function, which calculates attention weights, from having extremely large input values. Large input values to the softmax function can lead to gradients that are very small, making learning inefficient. By dividing the dot products by $\sqrt{d_k}$ we ensure the input to the softmax stays in a range where it can produce a more evenly distributed set of attention weights, facilitating smoother and more effective learning.

3. Weighted sum. Describe how the calculated attention scores are used to weight the value vectors (v_j). Essentially, this step focuses on the relevant elements in the sequence based on the attention scores. Explain the mathematical formula for this weighted sum.

Answer:

After calculating the attention scores, the model uses these scores to determine how much emphasis to place on each value vector in the sequence. This process creates a weighted sum of the value vectors, allowing the model to focus on the most relevant elements in the sequence based on the computed attention scores.

Weighted Sum of Value Vectors

Given the attention scores, which are obtained from the scaled dot-product attention mechanism, each score is applied to its corresponding value vector (v_j) to calculate a weighted sum. The attention scores effectively serve as weights that indicate the importance or relevance of each value vector in the context of a given query vector.

Weighted sum is calculated as follows:

$$\text{Output} = \sum_{j=1}^{N} \text{AttentionWeights}_{ij} \cdot V_j$$

where N is the number of elements in the sequence,

$\text{AttentionWeights}_{ij}$ are the attention weights derived from the attention scores between the i^{th} query and all key vectors, and v_j are the value vectors. The attention weights are obtained by applying a softmax function to the attention scores, ensuring that they sum up to 1:

$$\text{AttentionWeights}_{ij} = \text{Softmax}\left(\frac{q_i \cdot k_j}{\sqrt{d_k}}\right)$$

This softmax step transforms the raw attention scores into a probability distribution, where each weight reflects the relative importance of the corresponding value vector in the context of the query.

$$\text{Attention Weights}_{ij} = \frac{e^{\frac{q_i \cdot k_j}{\sqrt{d_k}}}}{\sum_{l=1}^{K} e^{\frac{q_i \cdot k_j}{\sqrt{d_k}}}}$$

Importance of the Weighted Sum

The weighted sum allows the model to aggregate information from across the entire sequence, emphasizing elements that are deemed more relevant based on the attention scores. This enables the model to dynamically focus on specific parts of the input sequence, making it particularly effective for tasks that require understanding the relationships and dependencies between different elements in the sequence.

The output of this step is a new vector that represents a combination of the input sequence's value vectors, weighted by their relevance as determined through the self-attention mechanism. This output can then be used in subsequent layers of the model or as the final output for prediction tasks.

4. Optional output transformation: Explain the purpose of the optional final linear transformation (W_o) and bias term (b_o) in generating the latent representation z. How does this step potentially impact the final representation?

Answer:

$$z = W_0 \cdot \sum_{j=1}^{N} \text{Attention Weights}_{ij} \cdot V_j + b_0$$

-z is the final latent representation that will be passed on to subsequent layers or used for predictions.

-W_0 is the weight matrix of the final linear transformation. This matrix is responsible for mapping the aggregated output vector to the desired dimensionality and shaping it according to the specific requirements of the task or the next layer in the model.

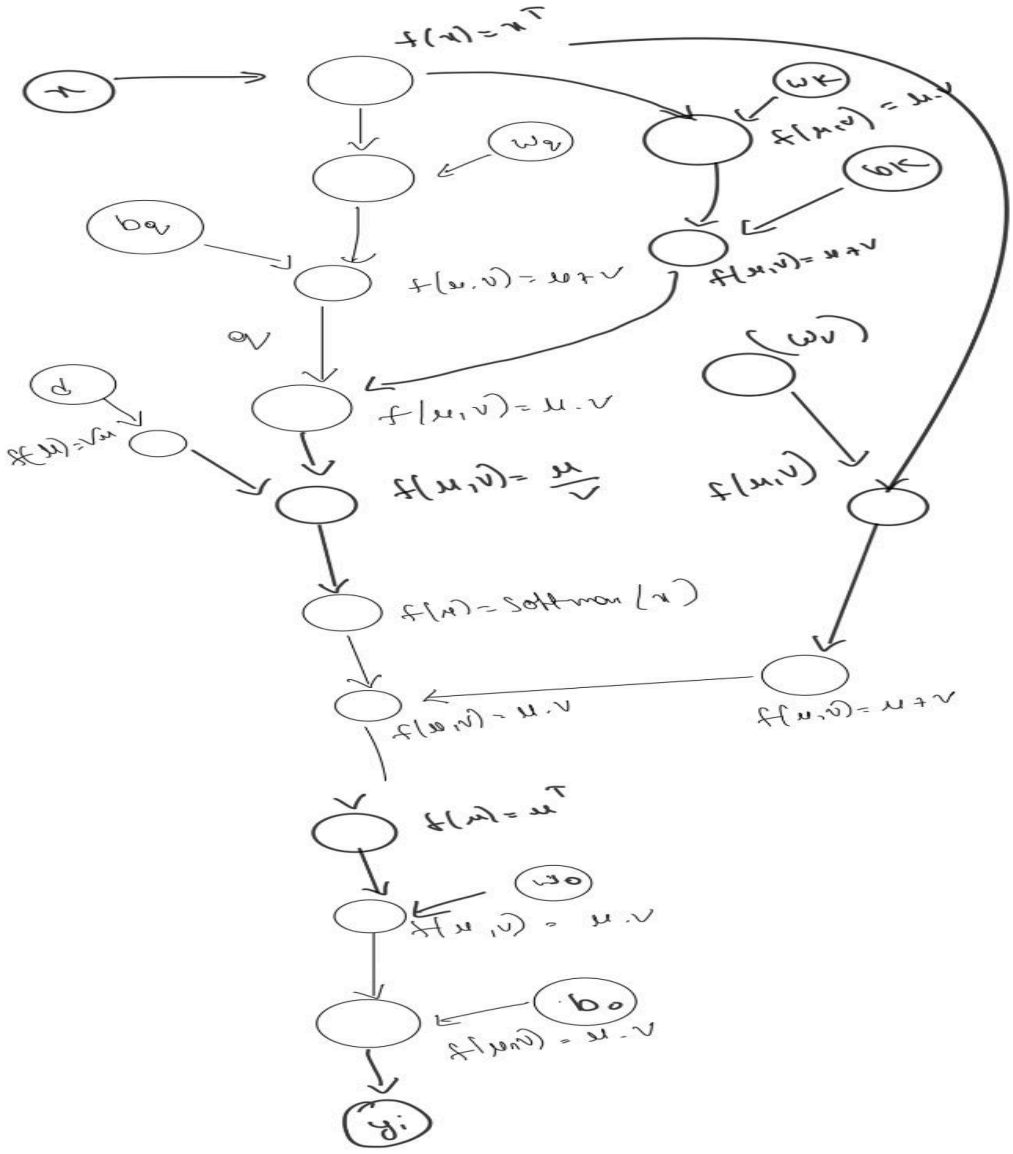
- $\sum_{j=1}^{N} \text{Attention Weights}_{ij} \cdot V_j$ is the aggregated output vector from the weighted sum of value vectors, which incorporates the attention mechanism's focus on relevant parts of the input sequence.

$-b_0$ is the optional bias term, which is added to introduce an additional degree of freedom and help the model learn offsets that may not be possible through a linear transformation alone.

This transformation step allows the model to further refine the output from the self-attention mechanism, adapting it to better suit the needs of subsequent processing stages or the final task objective. It provides an essential layer of flexibility and customization in the model architecture.

---

2. Draw the computational graph that depicts the flow of data through the self-attention mechanism. Include all the transformations mentioned in Question 1.
Answer:



Part IV: Build Transformer with PyTorch [40 points]

1. Describe the Transformer architecture you have defined.

Our transformer architecture starts with an embedding layer, which takes vectorized tokens of sequences of size 250 as input followed by a positional encoding layer, which adds positional information to the input embeddings. We then used a stack of 3 Transformer encoder layers to process the input sequence and added a dropout value of 0.3 at each layer. Each encoder layer also has a self-attention mechanism and uses 2 heads for multi-head attention. Then this encoder output is fed to the decoder as its input, that is the output of the encoder stack of layers is passed through Transformer decoder layer (3 layers) and also, used source and target mask whenever applicable. The output of the decoder layer stack is passed through the output linear layer with the desired output nodes. The model is trained with dropout regularization and generates predictions by averaging over the output sequence.

2. Describe how the techniques (regularization, dropout, early stopping) have impacted the performance of the model.

Dropout: We have used a dropout layer (with a dropout probability of 0.3) on the output of the Positional Encoder, which is then fed to the transformer encoder layer. During training, the dropout layer randomly sets the part of input units to zero with a probability defined by the dropout rate. Dropout prevents units from overfitting and forces the network to learn more robust features thus generalizing the model. In our model, though the model has not significantly performed well, almost had the same loss and accuracy (~69%) as compared to the base model, but the model has not over-fitted on the training data.

Regularization: Regularization techniques such as L1, L2 regularization along with weight decay prevents overfitting by adding a penalty term to the loss function and this penalty discourages large weights, thereby controlling the complexity of the model, regularization techniques can help reduce overfitting. We have used a weight decay of 1e-5 and the model slowly started performing better with the number of epochs. However, the base model has better performance with accuracy of about 86% percent in contrast to ~70% with regularization.

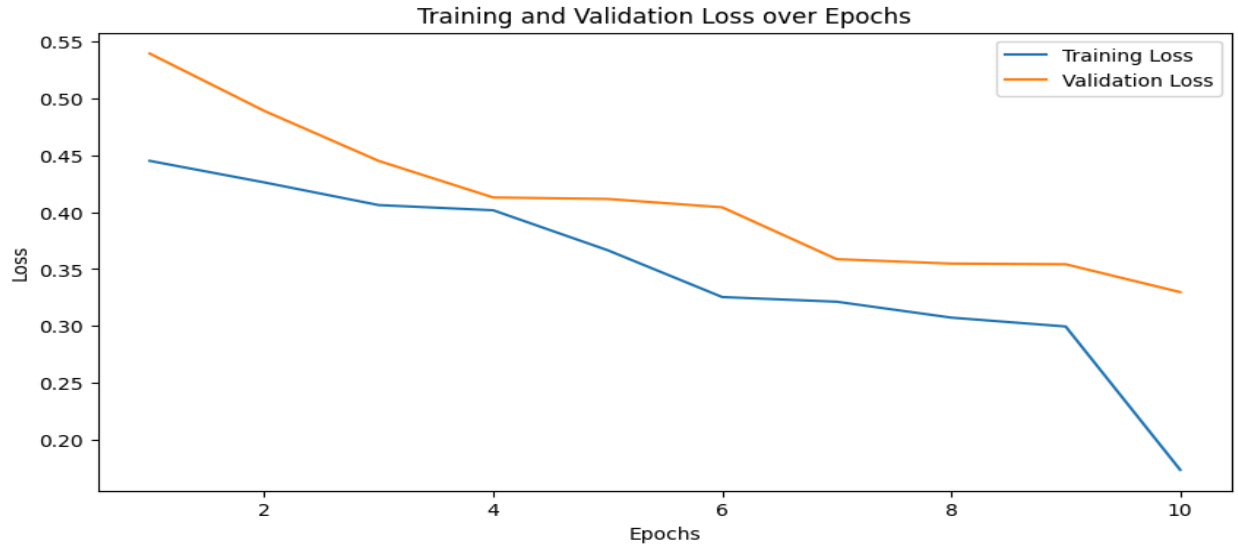
Learning Rate Tuning: The Learning rate is a hyperparameter that controls the size of the steps taken during the optimization process of the model to update the model's parameters during the backpropagation. Our base model has a learning rate of 0.01 and we have experimented with learning rates of 0.1 and 0.0001 which are larger and greater than the learning rate values used in the base model. However, there wasn't any significant improvement as the accuracies attained with these rates are around ~70% in contrast to ~86% with the base model.

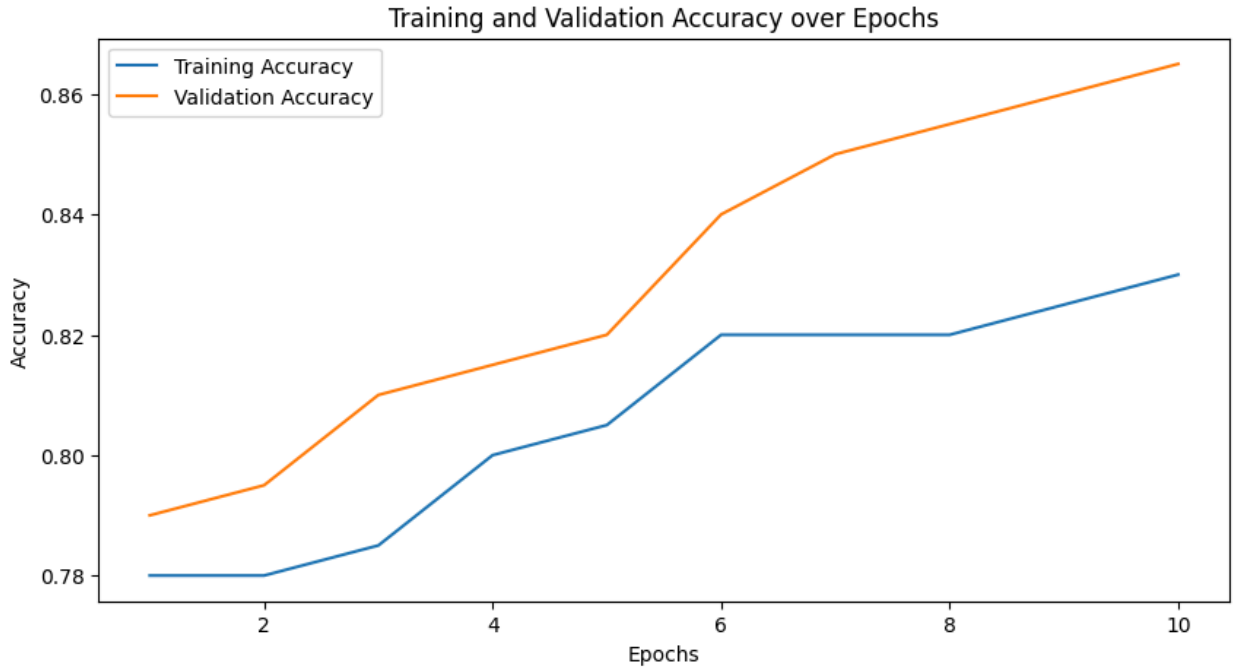
3. Discuss the results and provide the relevant graphs:

•Report training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss.

Accuracy, Loss on Base Model:

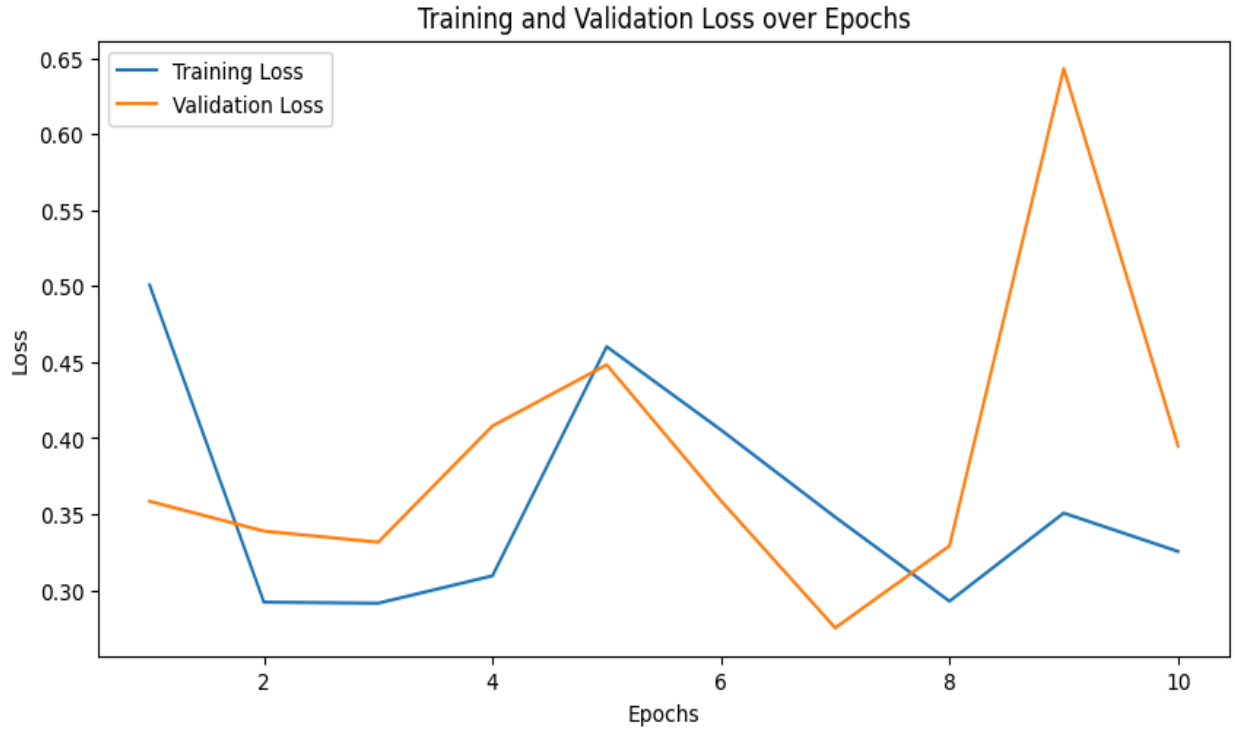
The Base Model has attained the best performance when compared with models using optimization techniques, though the models' has not attained better performance but has prevented the models from overfitting. The best model's accuracy is around ~**86%** with the loss being around **0.35** on the validation dataset.

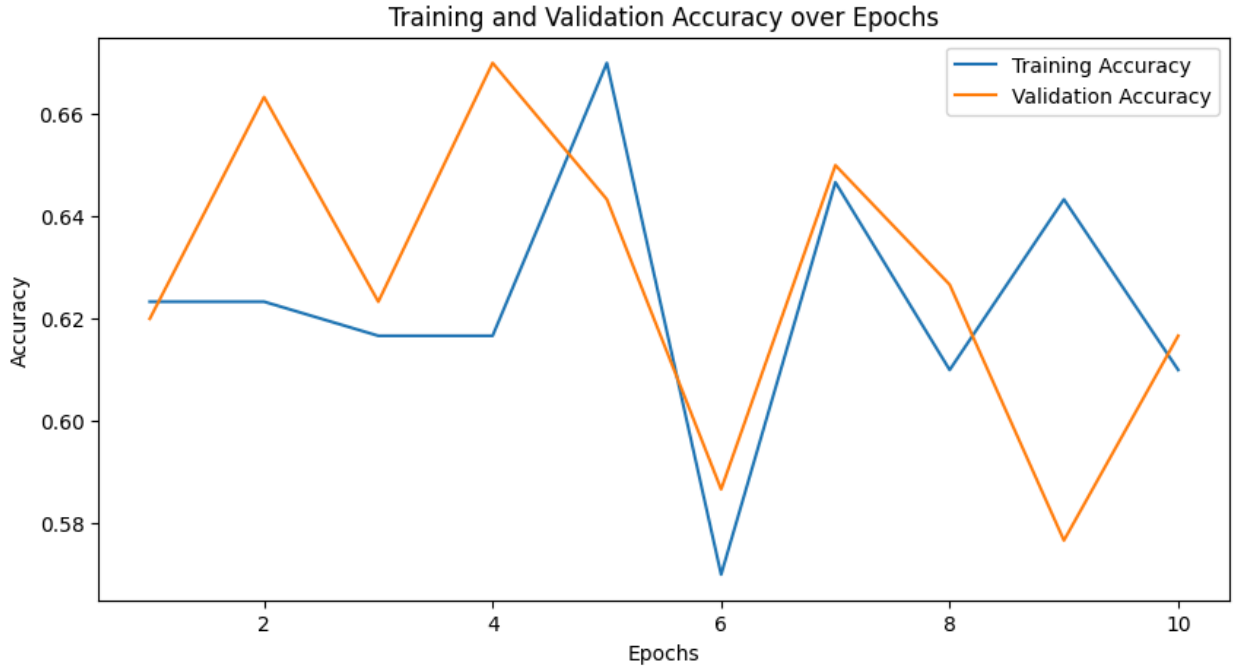




Accuracy, Loss on Dropout Model:

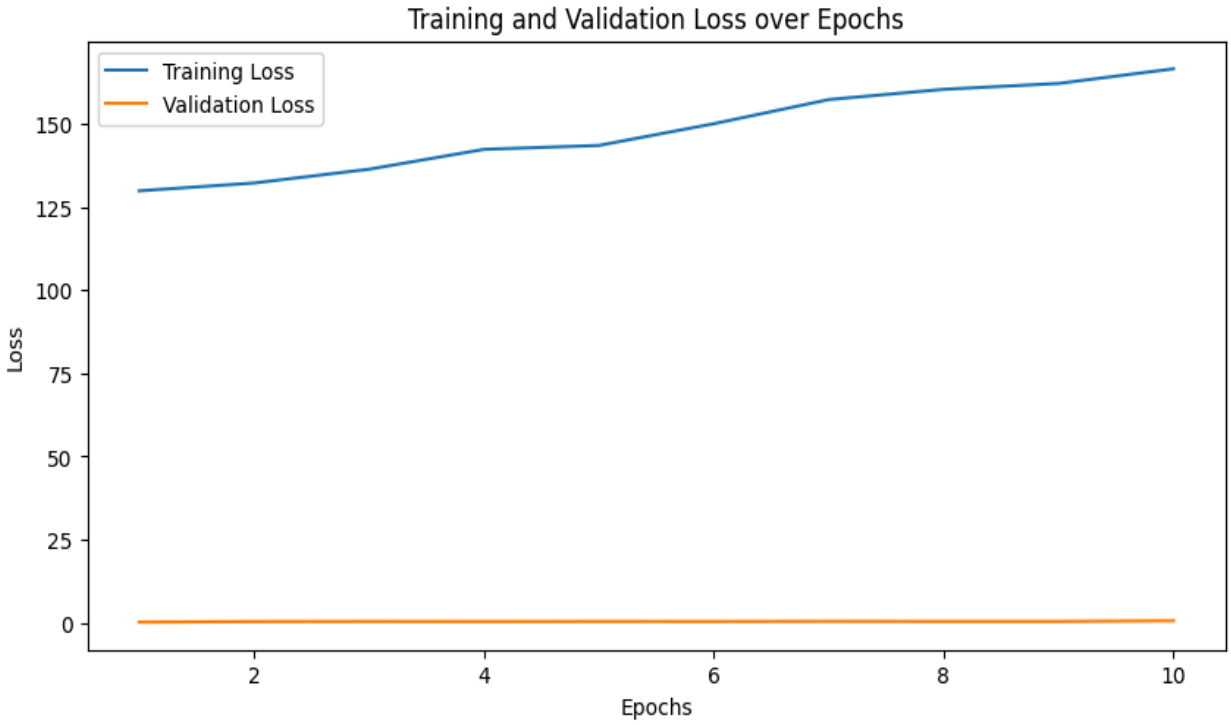
With dropout layer with a dropout probability of 0.3, the best accuracy is around ~68% and loss being 0.4

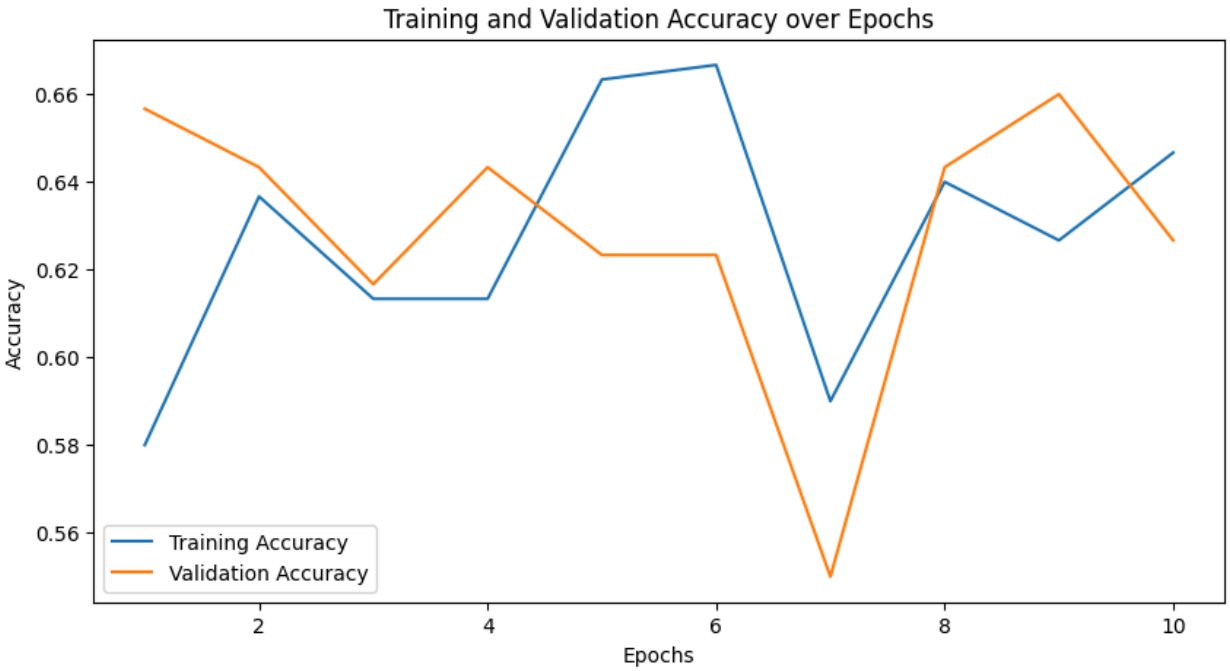




Accuracy, Loss on Regularization Model:

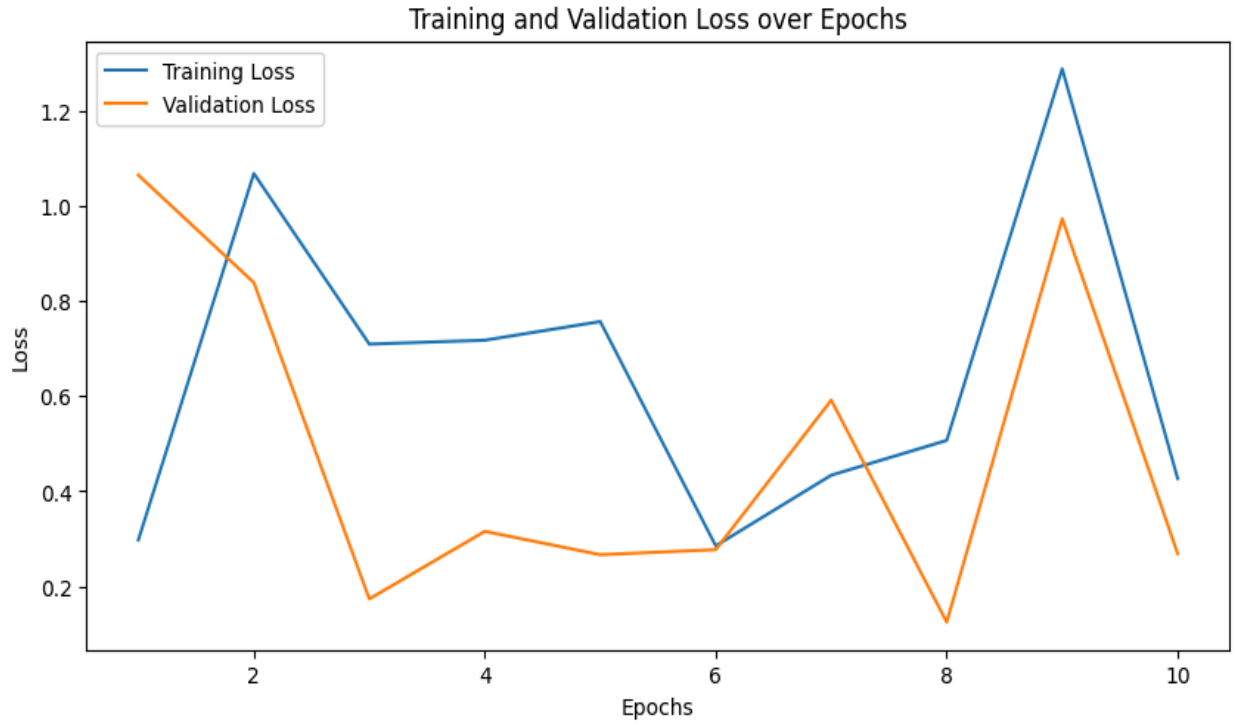
With Regularization, the model has achieved an accuracy of around ~70%.

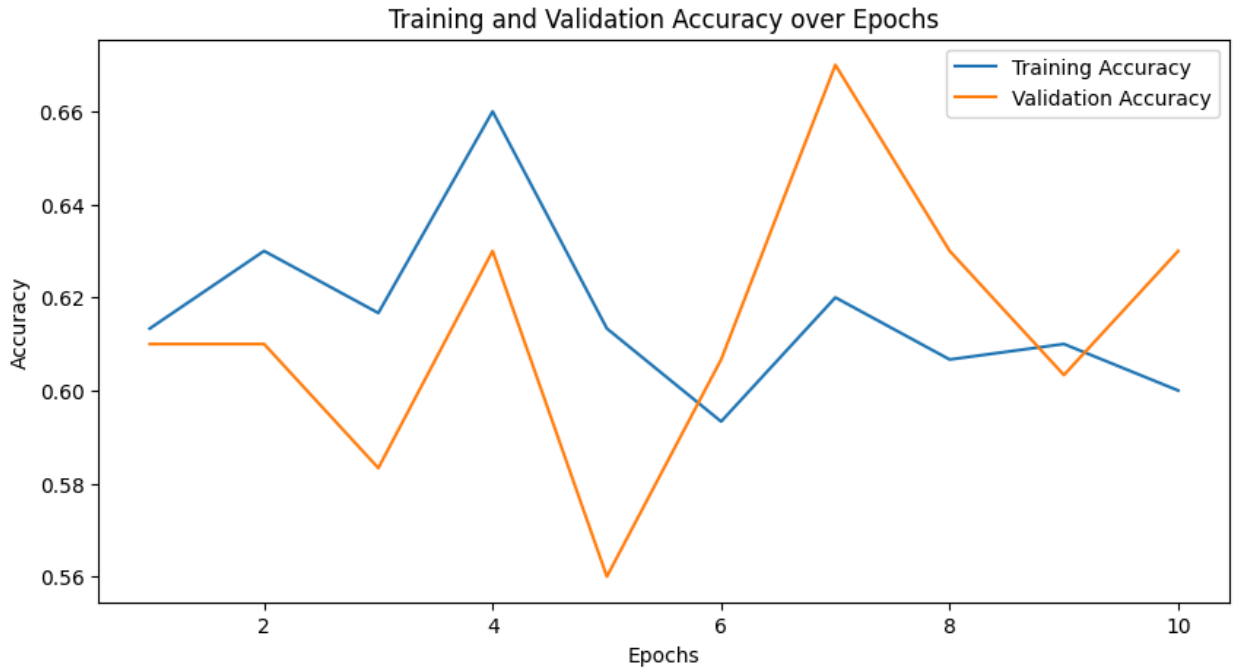




Accuracy, Loss on Learning Rate Tuned Model:

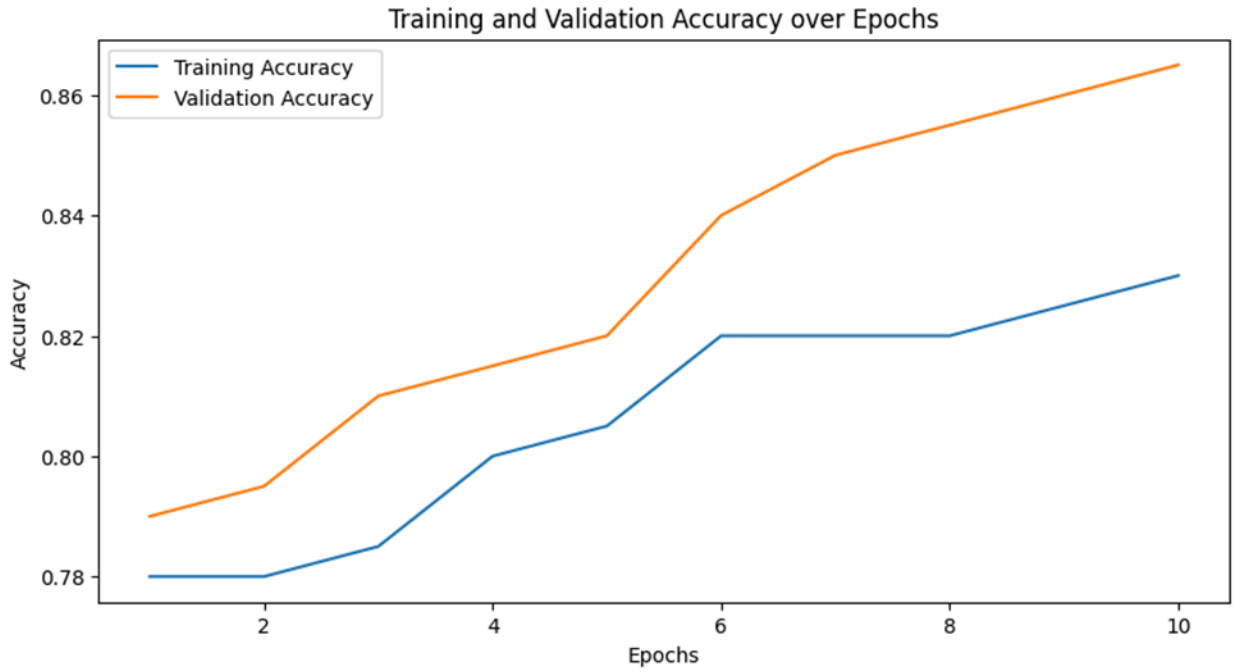
With LR=0.1,





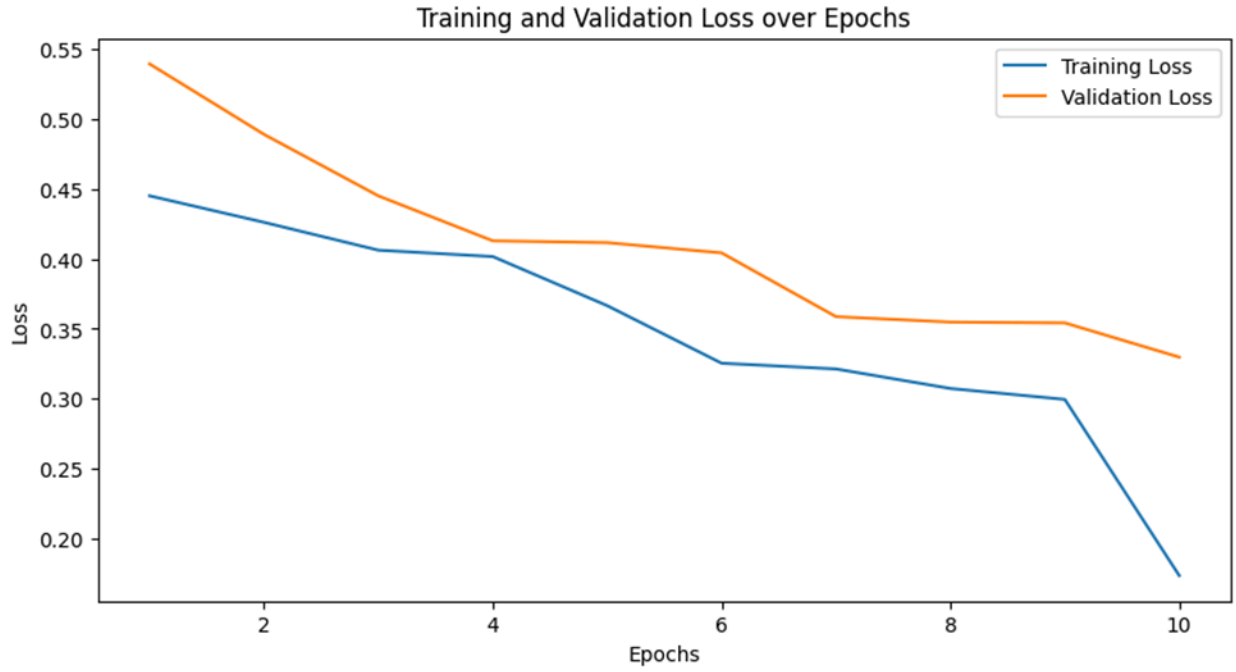
•Plot the training and validation accuracy over time (epochs).

Accuracy plot of training and validation sets for the best model.



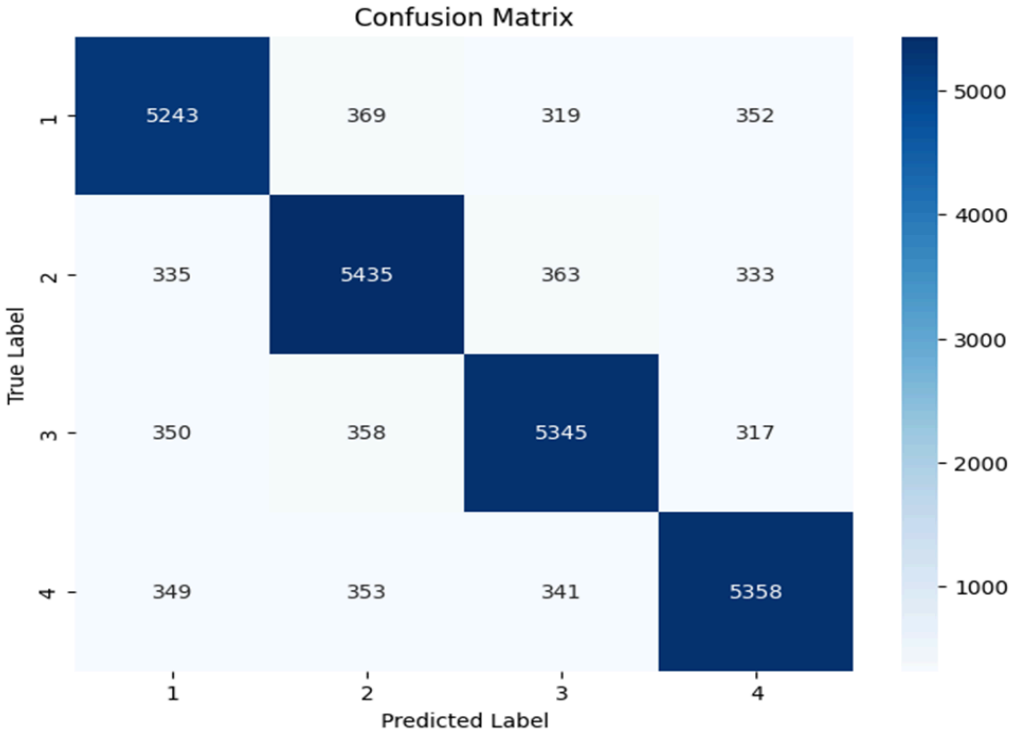
•Plot the training and validation loss over time (epochs).

Loss plot of training and validation sets for the best model.

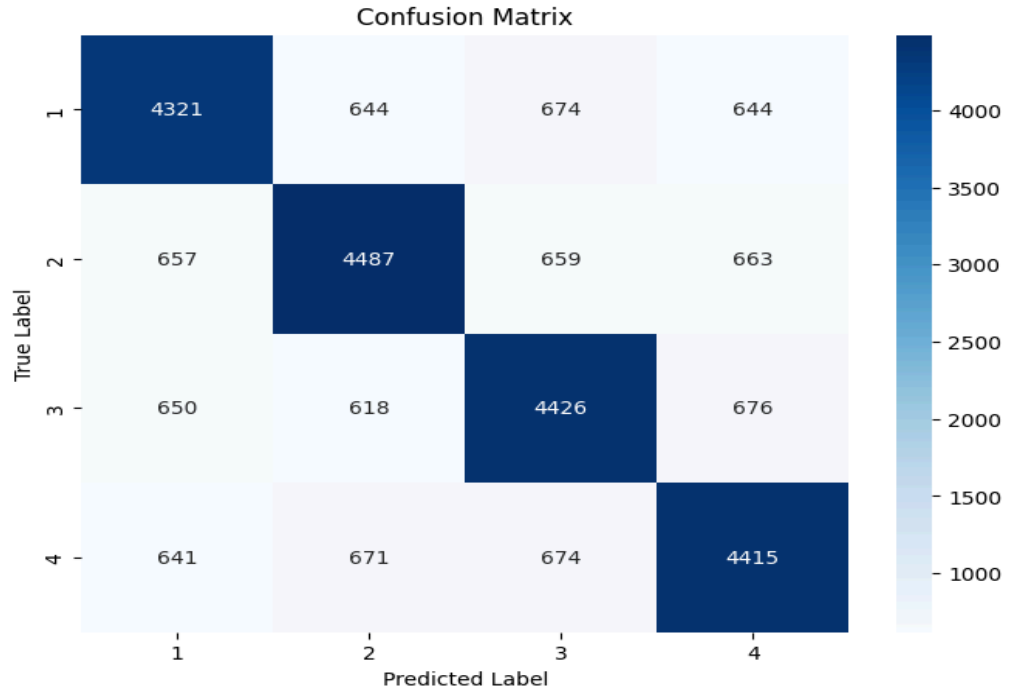


•Generate a confusion matrix using the model's predictions on the test set.

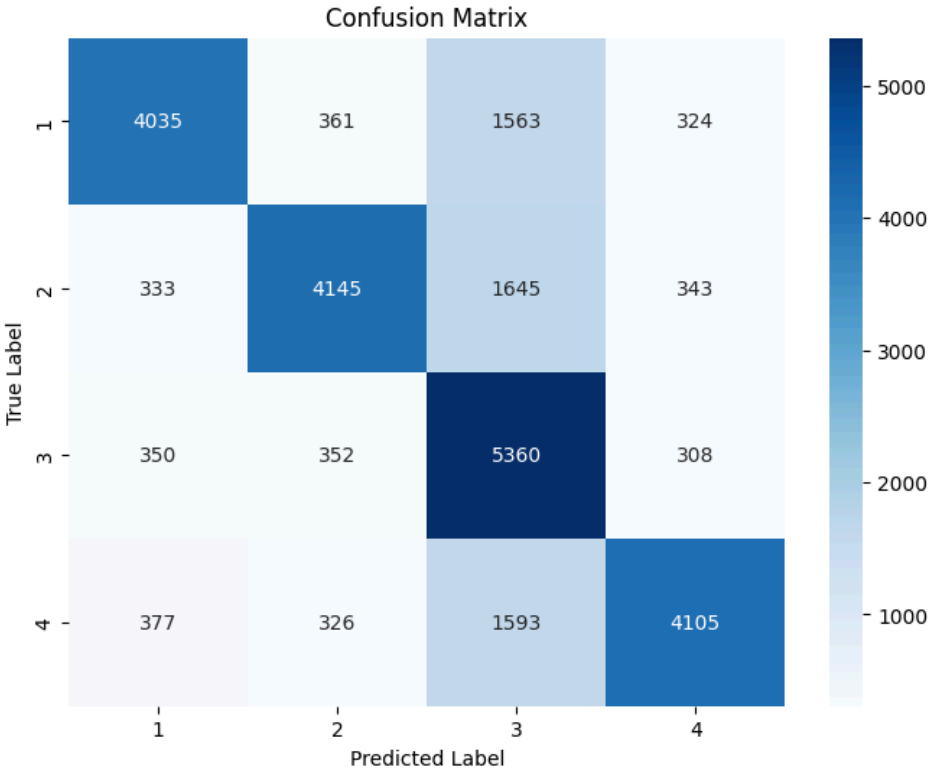
Confusion Matrix for the best Model are as follows:



Confusion Matrix for the Dropout Model are as follows:

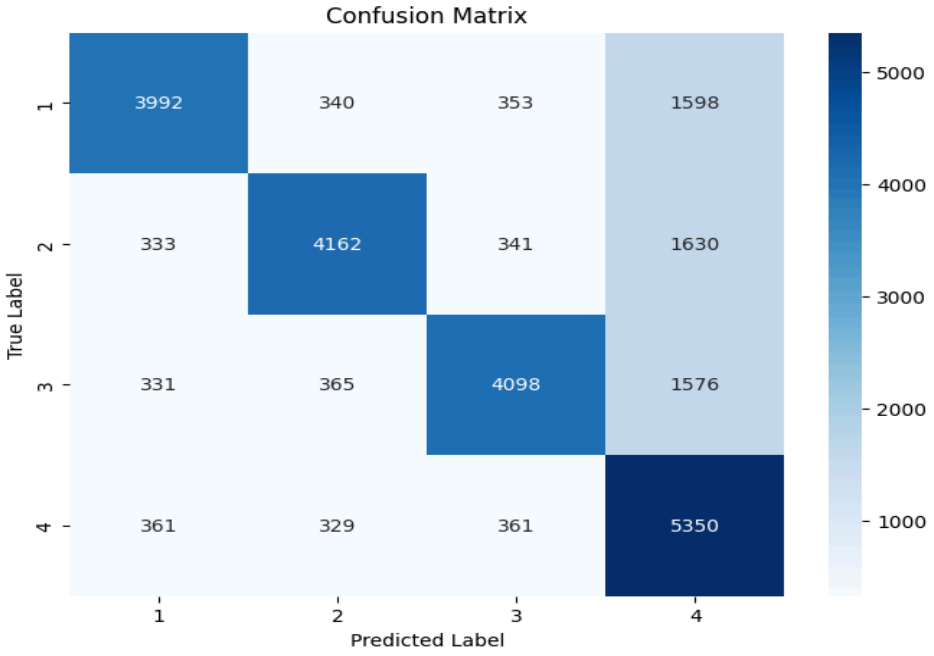


Confusion Matrix for the Regularization Model are as follows:

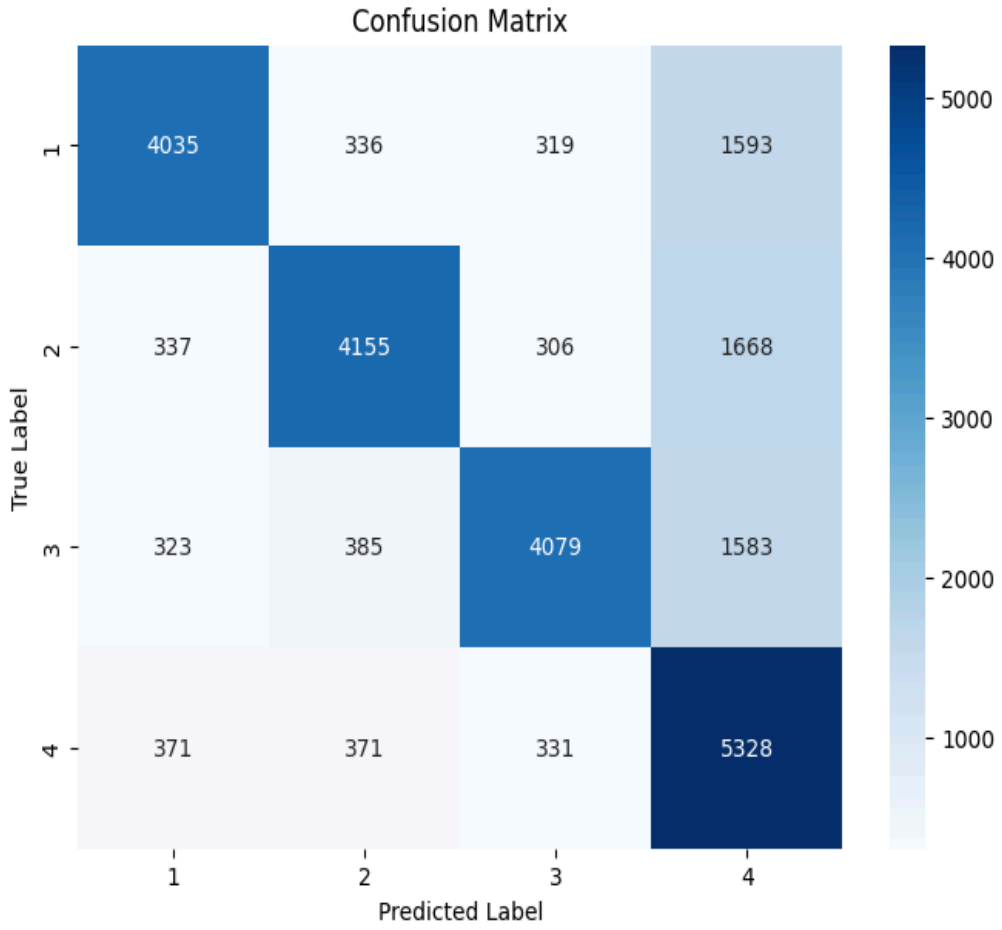


Confusion Matrix for the Learning Rate tuned Model are as follows:

With LR=0.1,



With LR=0.0001,



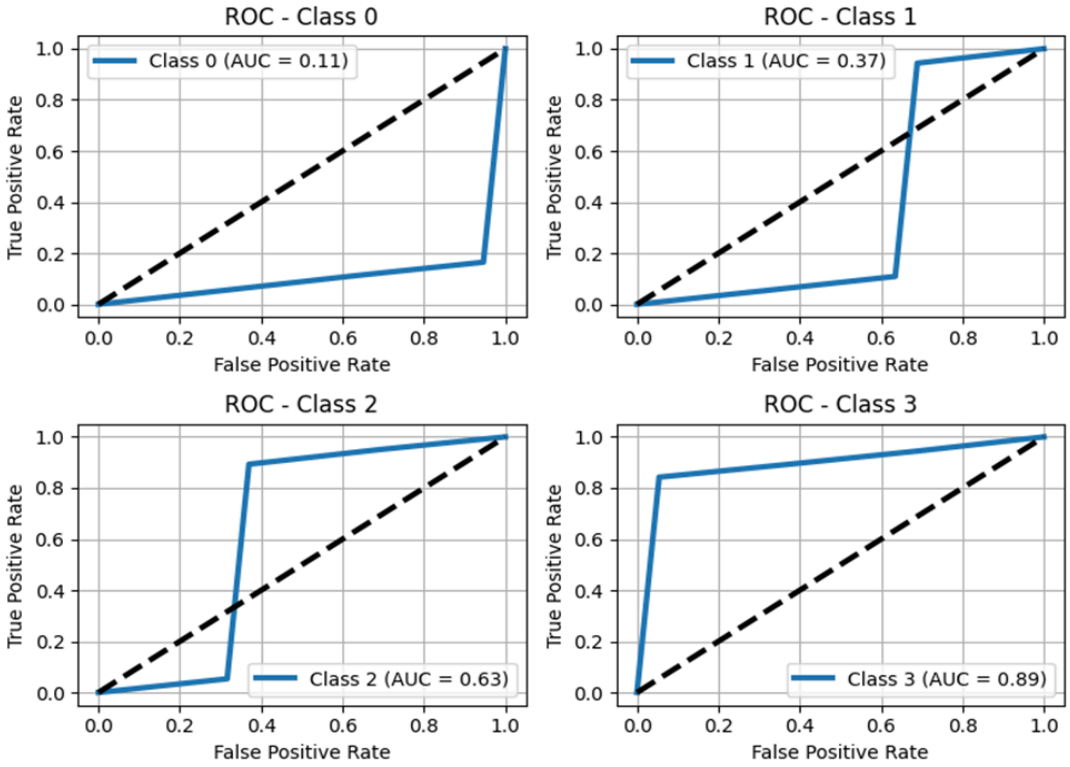
•Provide Precision, recall and F1 score.

The Precision, Recall and F1 Scores for the best model are 0.83782, 0.83779, 0.83780 respectively.

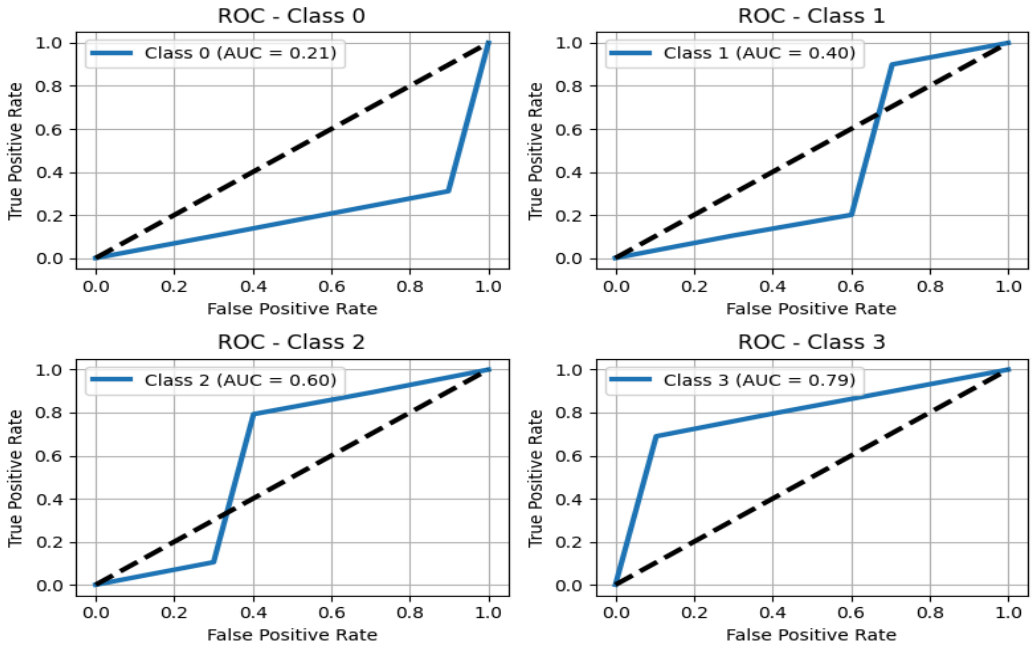
Precision: 0.8378265437075882
Recall: 0.8377926853647443
F1 Score: 0.8378044116419412

•Plot the ROC curve

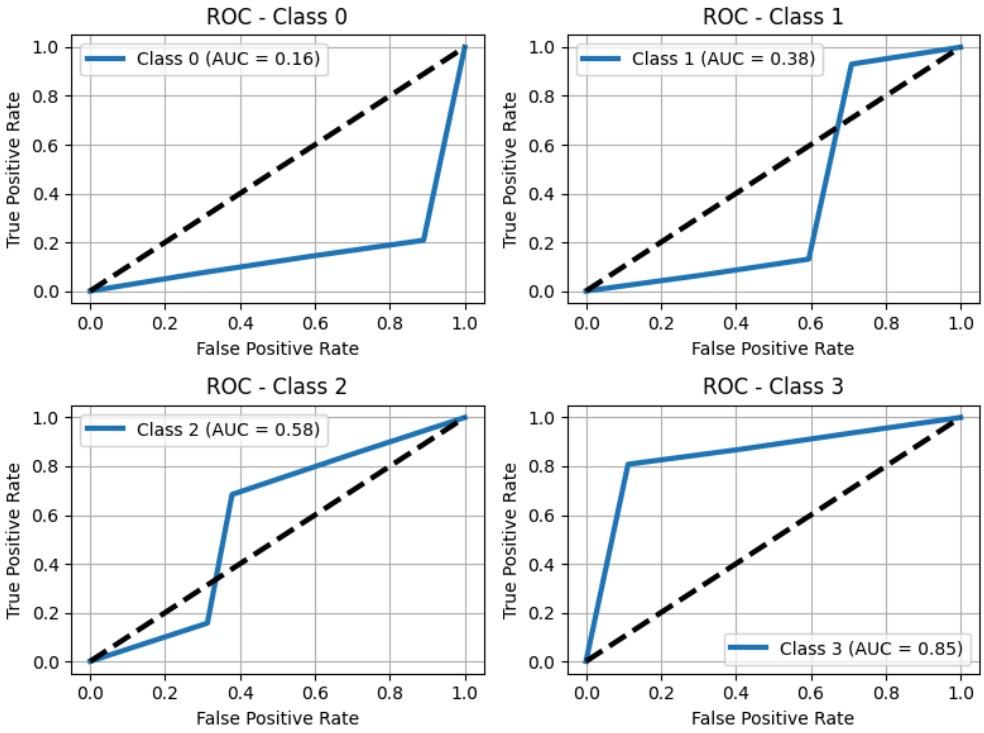
Receiver Operating Characteristics Curve for the best model,



Receiver Operating Characteristics Curve for the Dropout model,

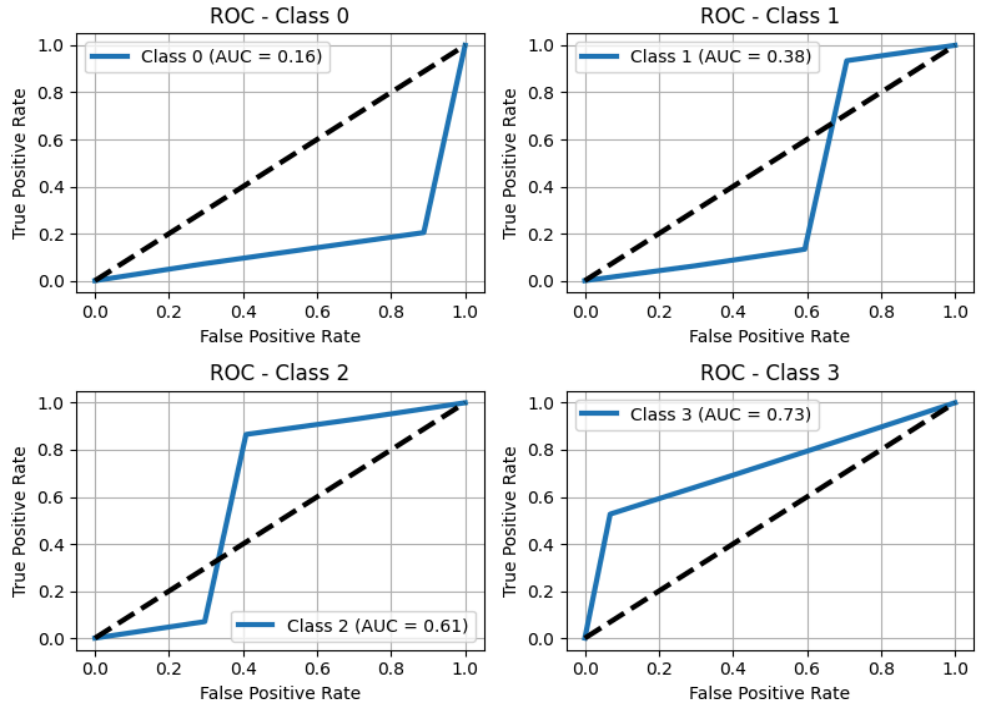


Receiver Operating Characteristics Curve for the Regularization model,



Receiver Operating Characteristics Curve for the Learning Rate tuned model,

With LR=0.1,



With LR=0.0001,