ASSIGNMENT 2

Full Name	UBIT Name	UB Number
Dharma Acha	Dharmaac	50511275
Adarsh Reddy Bandaru	adarshre	50527208

Part I: Theoretical Part – Autoencoders for Anomaly Detection in Manufacturing

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

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

Total number of parameters are

Input to hidden layer 1

Weights: 1000* 512 = 512000
Biases: 512
Total: 512000+512 = 512512

Hidden layer 1 to Bottleneck layer

Weights: 512*32 = 16384
Biases = 32
Total= 16384+32 = 16416

Bottleneck Layer to hidden Layer 2

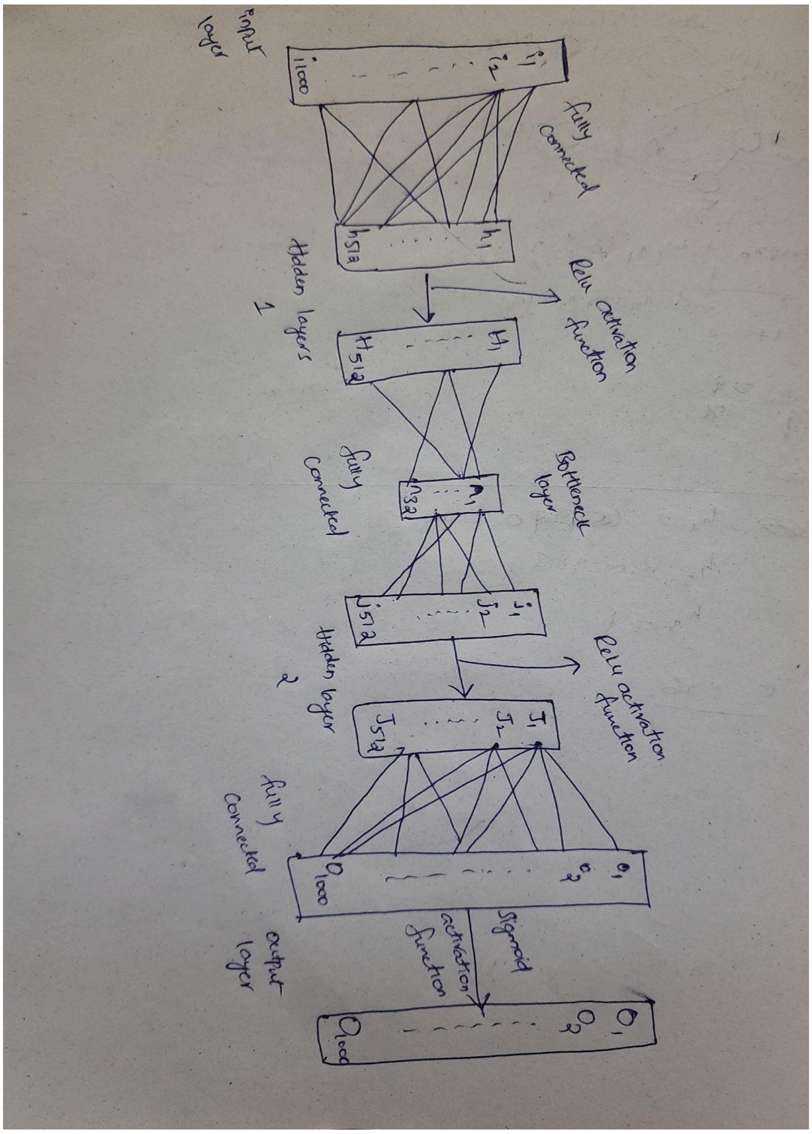
Weights: 32 *512 = 16, 384
Biases: 512
Total: 16384+512 = 16896

Hidden layer 2 to output layer

Weights: 512 *1000 =512000
Biases: 1000
Total: 512000+1000= 513000

So total number of parameters are (512,512 + 16,416 + 16,896 + 513,000 = 1,058,824)

2. Generate a computational graph for the autoencoder.



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

Imbalance Data and Labelling Challenges

The anomalies in manufacturing data are rare events which lead to imbalance datasets where normal instances considerably outnumber anomalies. The training models on imbalance data can result in biased models that prioritise normal instances over anomalies. And also getting labelled data for training autoencoders can be expensive and time-consuming, because anomalies may be rare and require domain expertise to identify accurately.

Adaptability to Dynamic Environments

The manufacturing process is dynamic, with operating conditions and system behaviour evolving with respect to time. The Autoencoders models trained on historical data may struggle to adapt to changes in the production environment, that leads to false positives and missed detections. The continuous training and adaptation strategies are necessary to ensure the effectiveness of autoencoder-based anomaly detection systems in dynamic manufacturing environments

Data Quality and variability

Manufacturing data consists of noise, missing values and outliers. Autoencoders are sensitive to such variations and may struggle to understand the difference between anomalies and normal variations in the data. So by making data quality and preprocessing techniques such as noise reduction and outlier removal are essential but cannot be always may not be feasible

Complexity and interpretability

Autoencoder models can become quite complex, especially when dealing with high-dimensional data such as sensor readings in manufacturing. This type of complexity might make it challenging to interpret the learned representations and understand why a particular is marked as anomaly. This interpretability is crucial in manufacturing settings that the manufacturers essential to understand the root cause of anomalies for effective troubleshooting and maintenance.

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

Ensemble and Multi-view Approaches:

Develop ensemble models which combine multiple autoencoders architectures or variations trained on different subsets of data or feature representation. These ensemble models can help in resolving the risk of overfitting and enhance robustness.

Attention Mechanisms and recurrent Architectures:

Use of attention mechanisms and recurrent neural network architectures to capture temporal dependencies and sequential patterns in manufacturing data. This attention mechanism focuses on relevant features and time steps, while RNN can model the temporal evolution of processes, which improves the ability to detect anomalies that manifest over time.

Hybrid models with Supervised Learning:

Implementing supervised learning techniques alongside autoencoders models to improve anomaly detection accuracy. Train separate classifiers for instance SVM, random forest) on labelled data to classify anomalies based on the encoded representation which is learn by autoencoder. The ensemble methods or cascaded architectures combine both unsupervised and supervised models' strengths to achieve better performance.

Domain knowledge integration and explainable AI

Following domain knowledge and expert insights into the model design and anomaly detection process. Feature engineering based on domain-specific insights can help enhance the discrimination power of the autoencoder and improve anomaly detection accuracy. And also focus on developing explainable AI techniques to provide interpretable explanations for detected anomalies, that makes operators and engineers to to understand the underlying causes and take appropriate corrective actions effectively.

Part-II

Dataset Selected: [Yahoo S5 Dataset]

Reason for choosing this dataset:

We elected this dataset as it aligns closely with my background as a programmatic analyst with prior experience working on web data, including Yahoo's Demand Side Platform data. This dataset focuses on anomalies such as traffic spikes, dips, etc which is interesting. My experience analyzing web traffic and user behaviour will be useful here.

Data exploration:

I chose A1Benchmark/real folder which contains 67 files with real-time-series data with labelled anomalies. The timestamps are integers with increments of 1, where each data point represents 1 hour's worth of data. The data includes timestamp (int64), value (float64), and is_anomaly (int64). The total number of rows in the dataset is 94866 combining all 67 files.

Data Cleaning and preprocessing

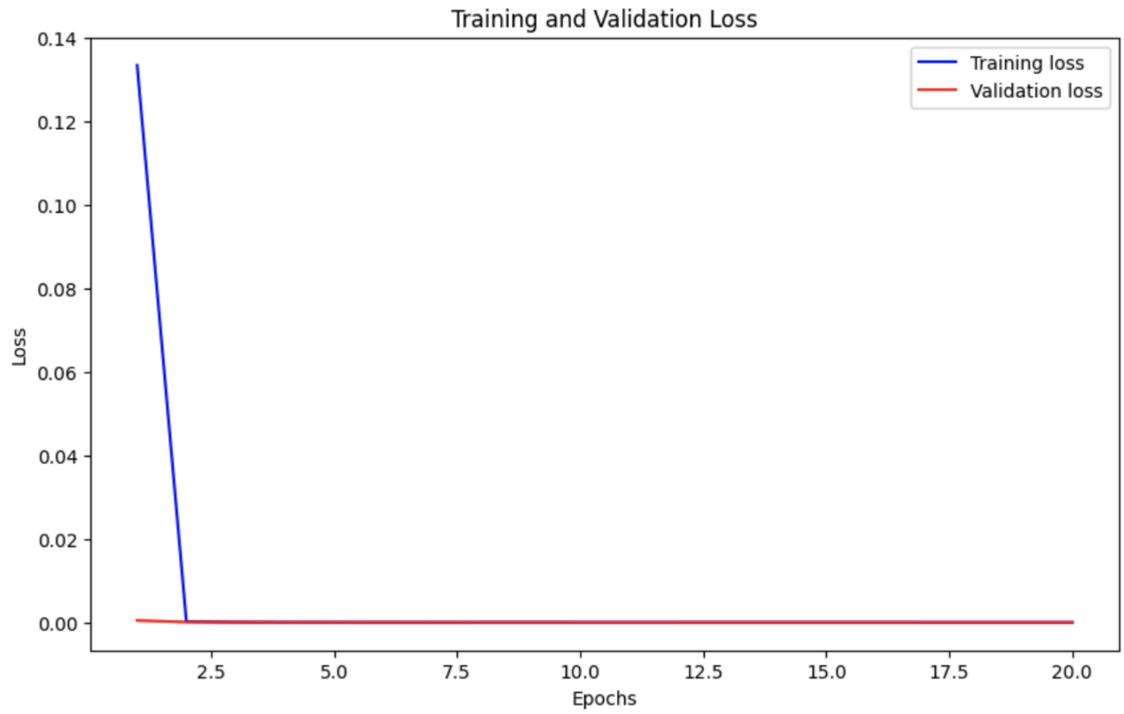
There are no missing values in the dataset. Before combining the individual files, we normalized the value column for better comparison. We also handled outliers in the Value column and replaced them with a median.

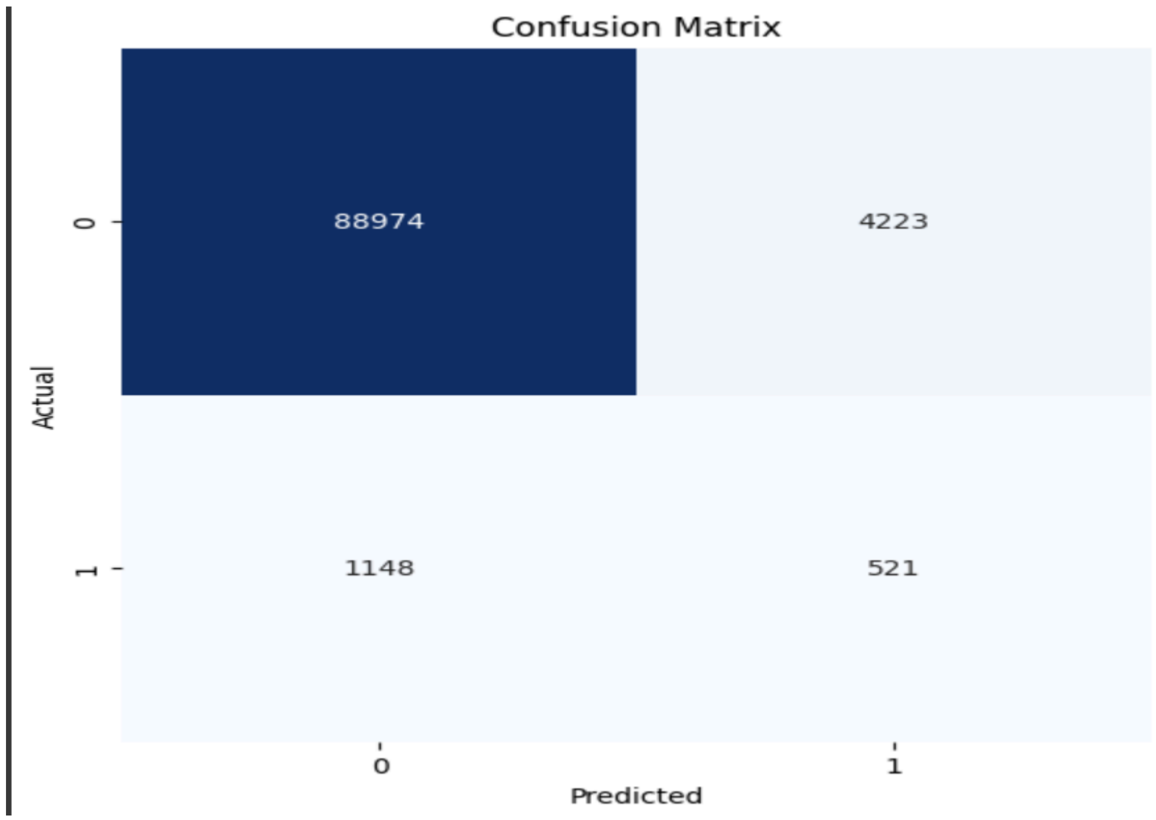
Standard Autoencoder:

Encoder: 2 Linear layers (2 to 16, then 16 to 4)
Decoder: 2 Linear layers (4 to 16, then 16 to 2)
Activation Function: ReLU (used after each linear layer except the last layer of the decoder)
Learning Rate: 0.001
Loss Function: Mean Squared Error
Optimizer: Adam
Threshold: Reconstruction errors exceeding the 95th percentile are considered anomalies.

Performance:

Accuracy: 94.34%
Training loss: 5.890449728790548e-06
Validation loss: 1.9139188377223087e-06
Test loss: 5.395394151599342e-06
Precision: 0.1098
Recall: 0.3122
F1 Score: 0.1625





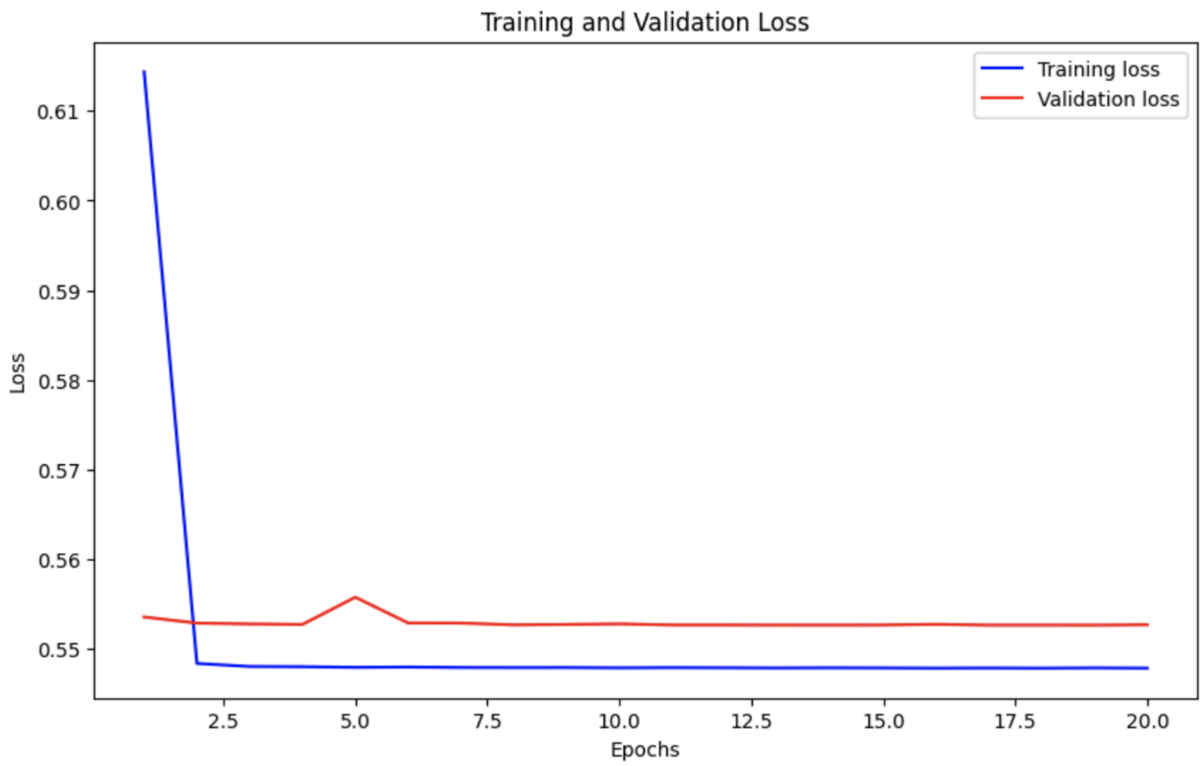
Dense Autoencoder Variation 1:

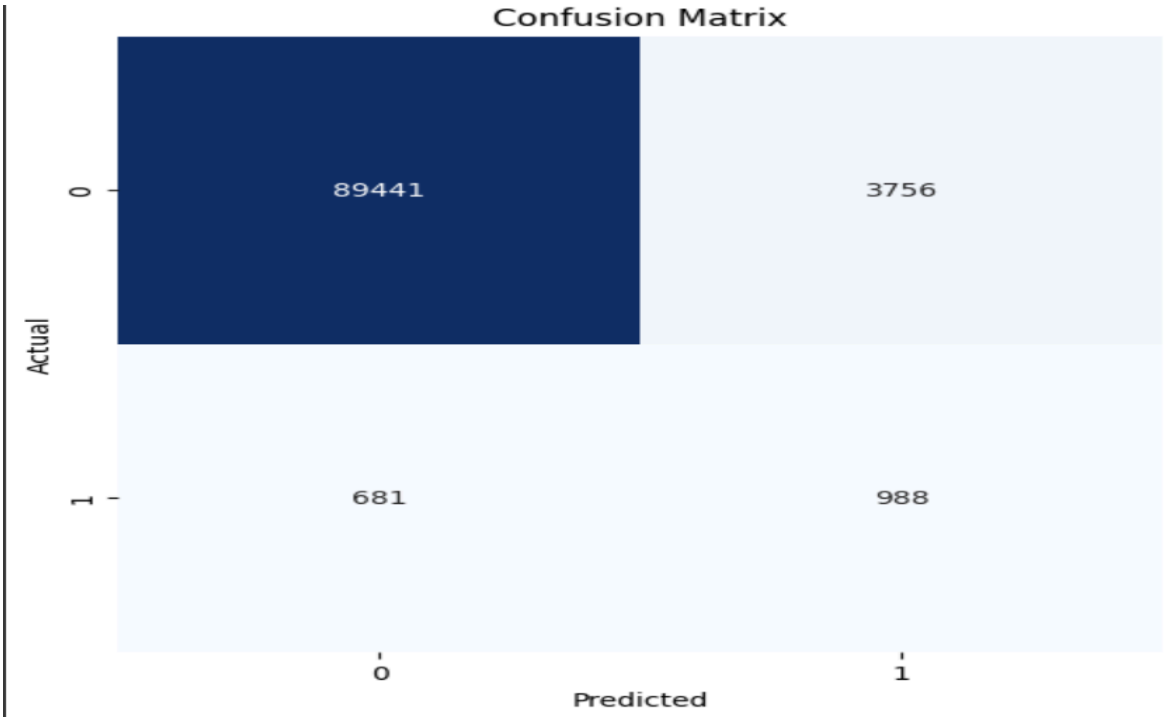
Encoder: 4 Linear layers (2 to 64, 64 to 32, 32 to 16, 16 to 4)
Decoder: 4 Linear layers (4 to 16, 16 to 32, 32 to 64, 64 to 2)

Activation Function: ReLU for encoding and decoding layers (Final activation function is Sigmoid)
Learning Rate: 0.001
Loss Function: Mean Squared Error
Optimizer: Adam

Performance:

Accuracy: 95.32%
Training loss: 0.54
Validation loss: 0.55
Test loss: 0.70
Precision: 0.20
Recall: 0.59
F1 Score: 0.30



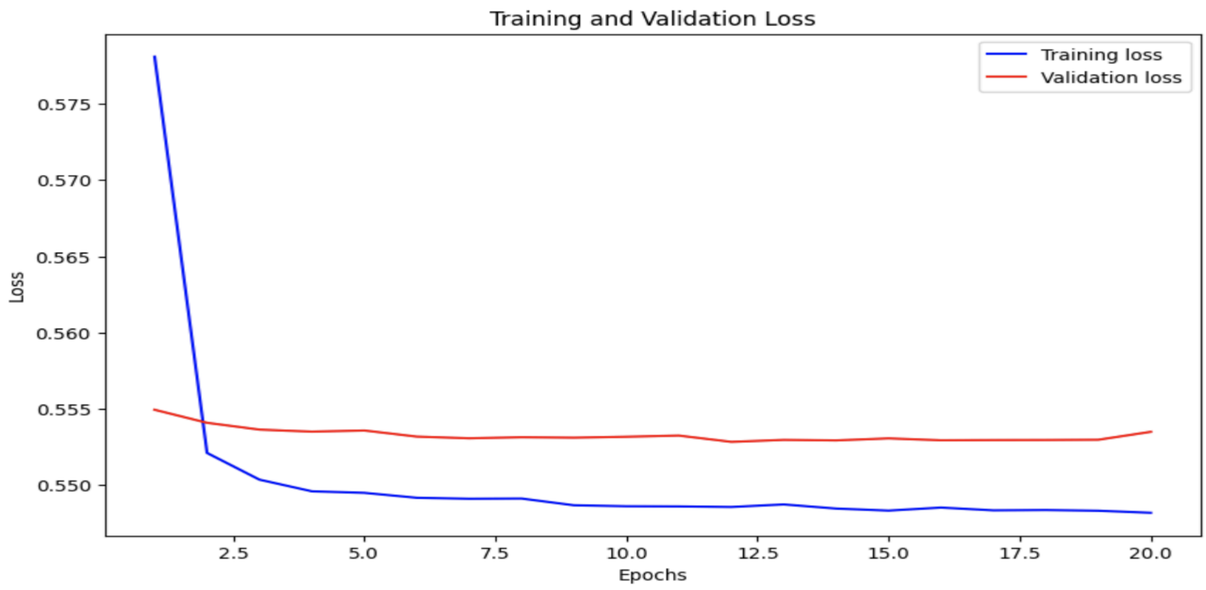


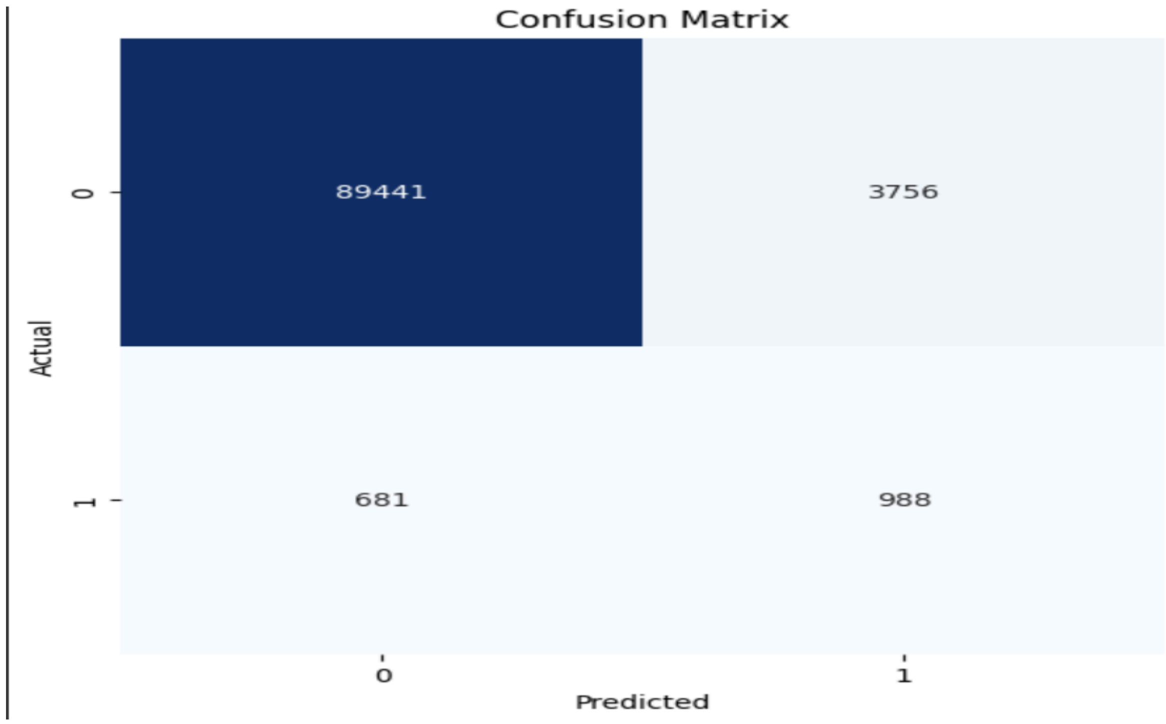
Dense Autoencoder Variation 2:

Encoder:6 Linear layers (2 to 32, 32 to 64, 64 to 128, 128 to 64, 64 to 32, 32 to 16)
Decoder:6 Linear layers (16 to 32, 32 to 64, 64 to 128, 128 to 64, 64 to 32, 32 to 2)
Activation Function: ReLU for encoding and decoding layers (Final activation function is Sigmoid)
Learning Rate: 0.001
Loss Function: Mean Squared Error
Optimizer: Adam
Extra: used batch normalization after each ReLU activation.

Performance:

Accuracy: 95.32%
Training loss: 0.54
Validation loss: 0.55
Test loss: 0.70
Precision: 0.20
Recall: 0.59
F1 Score: 0.30





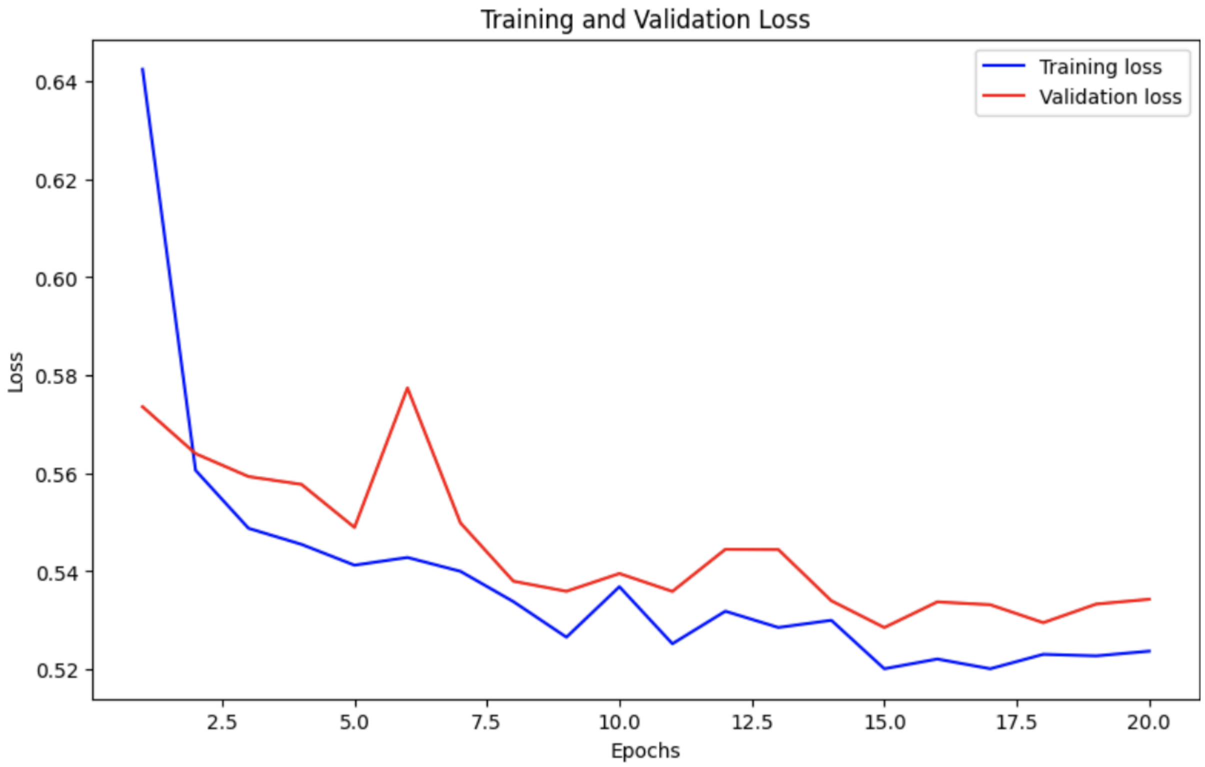
LSTM Autoencoder:

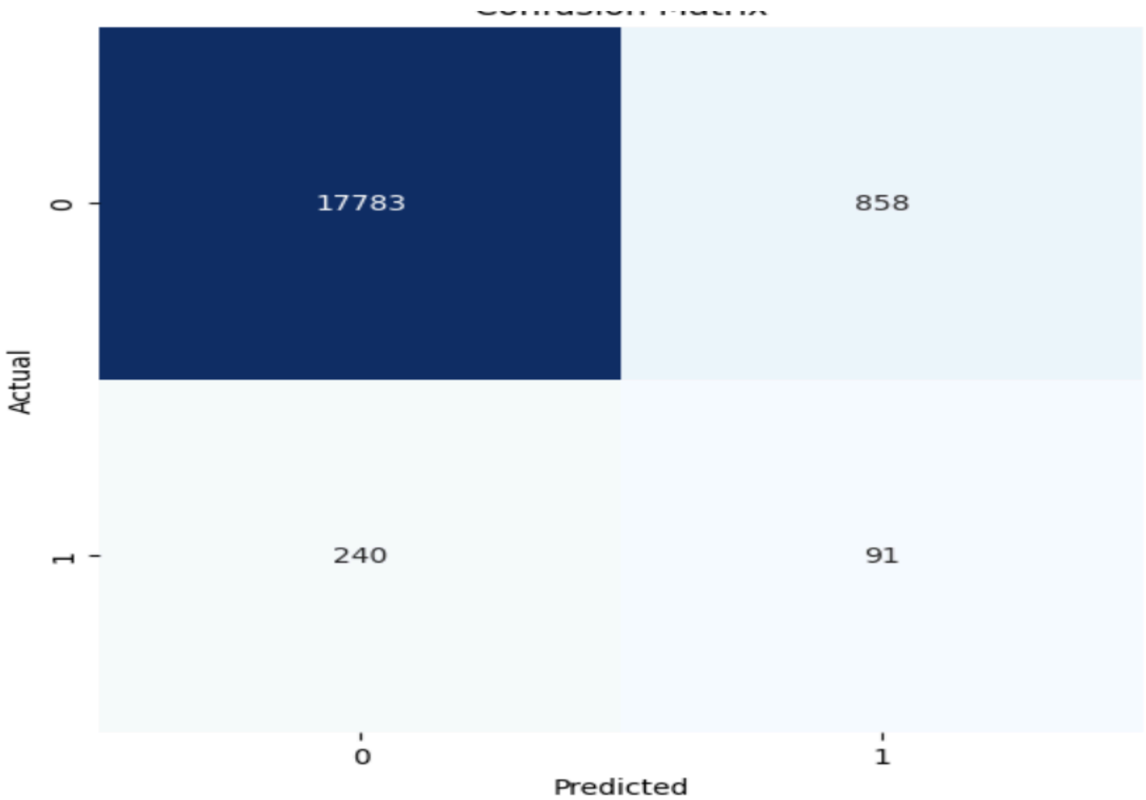
Encoder: Sequential layers starting with an LSTM layer that takes an input sequence of length seql with input_size features into a hidden state of hidden_size (64) neurons, followed by linear layers that compress the data down to a size of 4.
Decoder: Begins with a linear layer that expands the compressed size of 4 back into hidden_size * seql, followed by an LSTM layer that restores the sequence to the original input_size features.

Learning Rate: 0.01
Loss Function: Mean Squared Error
Optimizer: Adam

Performance:

Accuracy: 94.21%
Training loss: 0.52
Validation loss: 0.53
Test loss: 0.53
Precision: 0.09
Recall: 0.27
F1 Score: 0.14





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

Strengths

Adaptability: The autoencoders can adapt to different types of data and anomalies. We can adjust the architecture and hyperparameters of the autoencoder, it suits to different datasets and anomaly detection tasks.

Robust to Noise: Autoencoders can reconstruct clean data from noisy inputs, which makes them noisy data and capable of filtering out irrelevant information,that improves anomaly detection performance in real-time.

Unsupervised Learning: Autoencoders can learn representation of the data without requiring labelled examples of anomalies. This allows them to detect anomalies in situations where their labelled data is not available.

Limitations

Overfitting: Autoencoders are prone to overfitting, especially when the training data is limited. So, regularisation methods like dropout and weight decay can help solving overfitting.

Computational Intensive: Training deep autoencoders can be computationally intensive, especially for large datasets.

Data dependency: Autoencoders heavily depend on the quality and diversity of the training data. If anomalies are not well-represented in the training data, the autoencoders fails to detect them.

Limited interpretability: The learned representation in the hidden layers may not be easily interpretable, which makes it difficult to understand why a particular data point is marked as anomalous.

Part III: Theoretical Part – Transformers

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:

The each element X_i of the input sequence X is first subjected to three distinct linear transformations to generate respective Query(q), Key(k) and Value(v) vectors.

Query Transformations (Q):

The input vector X_i is transformed into query vector q_i with the help of weight matrix W^Q .The query vectors are used to determine the attention weights. Which is essential asking questions about other parts of the sequence. So mathematically each element X_i in the sequence,

The query vector q_i is calculated by

$$q_i = W^Q X_i$$

Where W^Q is the learned weight matrix for queries

Key Transformation (K)

Likewise each X_i is also transformed into key vector k_i using a different weight matrix W^k These key vectors help the model to decide how much attention to pay each part of the sequence while generating the output.

Each X_i the key vector k_i is calculated as

$k_i = w^k x_i$ where w^k is the learned weights matrix for keys

Value Transformations (V)

The each element x_i of the input sequence is transformed into value vector V_i with the help of another weight matrix W^v. The value vector represents the actual information from each part of the input sequence that will be combined to generate the output, and determined through the query-key matching process.

For each element x_i, the value vector V_i is calculated as

$$V_i = W^v x_i$$

Scaled dot product attention

This scaled dot product attention calculates the attention scores based on the interactions between the query q_i and the key vector k_j vectors

The attention score between these is calculated by taking dot product of q_i and k_j and then scaling the result by inverse square root of the dimension of the key vector d_k

So finally it is calculated by the attention score using q_i and k_j is $q_i \cdot k_j / \sqrt{d_k}$

The role of $\sqrt{d_k}$

The scaling factor $\sqrt{d_k}$ has a role in normalization process by preventing the dot product values which grow in bigger values. So without scaling the softmax function which is applied to the scores to obtain the attention weights, this could have very sharp distribution if the dot products are large.

The dimensionality d_k of the key vectors influences the magnitude of the dot products.

As d_k increases, the value of the dot product increases linearly. However, the variance of the softmax function's output is inversely proportional to $\sqrt{d_k}$

Weighted Sum

After calculating the scaled dot product attention scores for all pairs of query and key vectors, a softmax function is applied to each set of scores for each query q_i w.r.t all keys k_j to obtain a normalised weight for every value vector.

The weighted sum is calculated by multiplying each value vector V_j w.r.t weight a_{ij} and then add these weighted values for each position i in the output sequence.

The formula for weighted sum is $\sum_{j=1}^{N} a_{ij} V_j$

The attention weights a_{ij} are then used to compute a weighted sum of the value vectors, resulting in the final output for each position

Here a_{ij} is the attention weight between the ith query and the jth key. This represents how much focus should be placed on the jth value vector while constructing the ith element of the output. The V_j are the value vectors obtained from the input sequence through a linear transformation.

The result of the weighted sum is the ith output vector, which is a representation formed by combining across the entire input sequence, weighted according to the calculated attention scores.

Optional output transformation

The optional final linear transformation is represented by weight matrix W^o and the bias term b^o applied to the aggregated value vectors in the context of the self-attention or transformers, this serves as refine output into a sophisticated latent representation represented by Z

This transformation is particularly significant in complex model like Transformer, where multiple self-attention mechanisms are stacked or combined with other operations

Purpose and Impact:

Increasing Model capacity:

Introducing this additional set of learnable parameters W^o and b^o increases the model’s capacity allowing it to learn more complex relationships and mappings from the input sequence to the output representation.

The increased capacity can improve the model's ability, which leads to better performance on a wide range of tasks.

Refinement and Projection:

The transformation W^o and b^o allows the model to project the aggregated output from the attention mechanisms into a space that is potentially more suitable for the tasks the model is designed to perform.

This step can refine the info content of the output. Making the final latent representation Z aligns better with the expected dimensions and characteristics for subsequent processing layers.

Enhanced Flexibility and customization:

The final linear transformation offers an extra point of customization in the model's architecture. Thai depends on the task, the dimensions of W^o this can be chosen to project the output into a higher or lower dimensional space which provides flexibility in designing models to meet requirements

Potential impact on the Final representation:

Reduced information loss:

This transformation also acts as a filter, potentially reducing noise or less relevant information present in the aggregated attention output, which leads to clear latent representation.

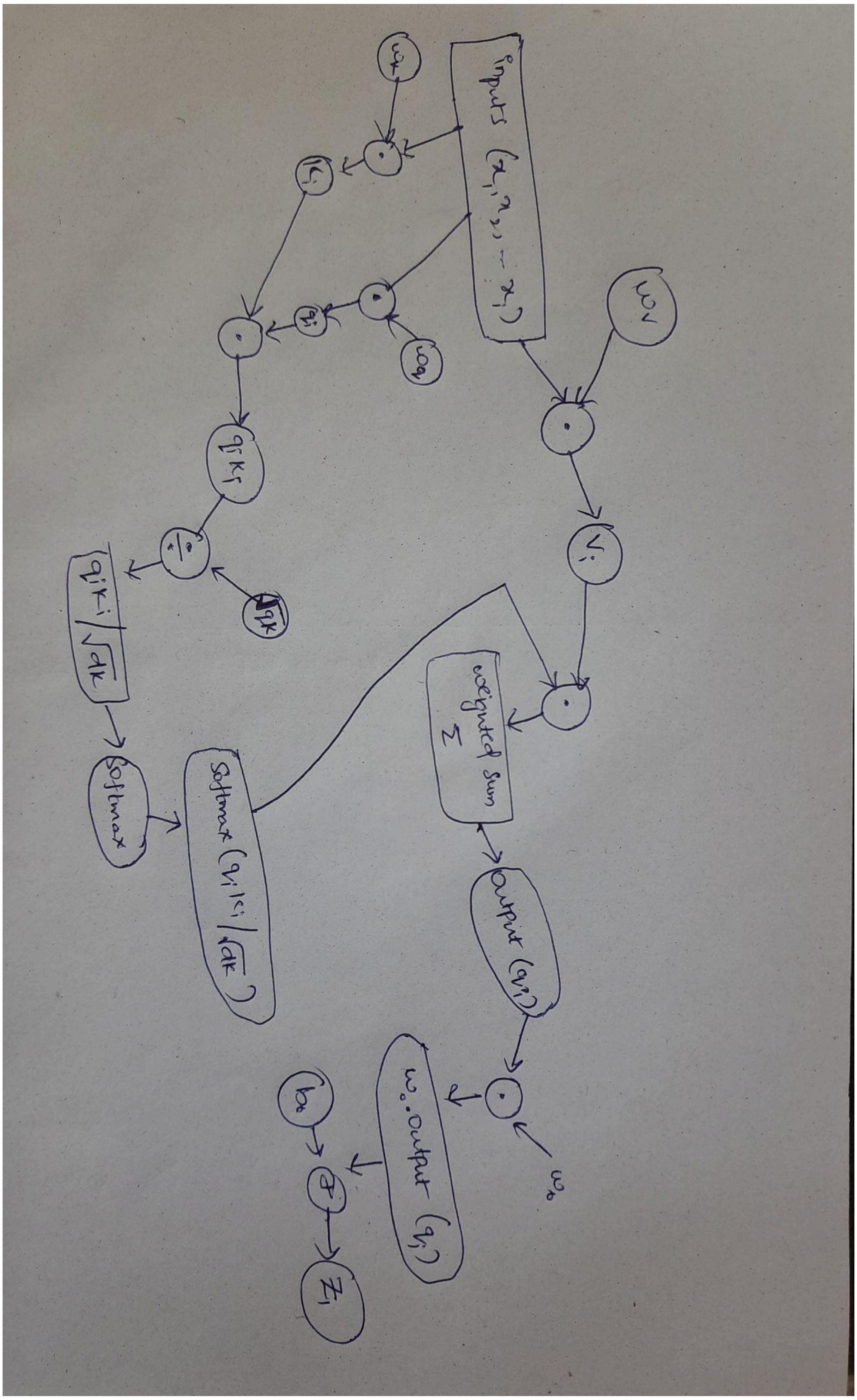
Enhanced Learning Dynamics:

This transformation affects the learning dynamics of the model. It introduces an additional layer of abstraction, which prevents the factors of variations in the data, which makes the learning process a more generalised model.

Improved Task Alignment:

Optimising the W^o and b^o , the model can better align the attention mechanism's output with the requirements of particular task which it is used for, for instance language translation, text generation,

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



Part- IV

Dataset: ag_news

Reason for choosing this dataset:

The AG news dataset is balanced across different classes, this helps in training a more stable and unbiased model. This balanced data makes the model perform uniformly across different text types. It has sufficient entries so that the model trains in time without consuming much time in training.

Base Transformer architecture:

```
transformer(
 (embedding): Embedding(56228, 256)
 (pos_encoder): positional_encoding()
 (encoder_layer): TransformerEncoderLayer(
  (self_attn): MultiheadAttention(
   (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256,
bias=True)
  )
  (linear1): Linear(in_features=256, out_features=512, bias=True)
  (dropout): Dropout(p=0.1, inplace=False)
  (linear2): Linear(in_features=512, out_features=256, bias=True)
  (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (dropout1): Dropout(p=0.1, inplace=False)
  (dropout2): Dropout(p=0.1, inplace=False)
 )
 (transformer_encoder): TransformerEncoder(
  (layers): ModuleList(
   (0-5): 6 x TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
     (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256,
bias=True)
    )
    (linear1): Linear(in_features=256, out_features=512, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=512, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
```

```
        (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (decoder_layer): TransformerDecoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256,
bias=True)
    )
    (multihead_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256,
bias=True)
    )
    (linear1): Linear(in_features=256, out_features=512, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=512, out_features=256, bias=True)
    (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
    (dropout3): Dropout(p=0.1, inplace=False)
  )
  (transformer_decoder): TransformerDecoder(
    (layers): ModuleList(
      (0-5): 6 x TransformerDecoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256,
bias=True)
        )
        (multihead_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=256, out_features=256,
bias=True)
        )
        (linear1): Linear(in_features=256, out_features=512, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=512, out_features=256, bias=True)
        (norm1): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (norm3): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
```

```
      (dropout3): Dropout(p=0.1, inplace=False)
    )
  )
 )
 (fc_out): Linear(in_features=256, out_features=4, bias=True)
)
```

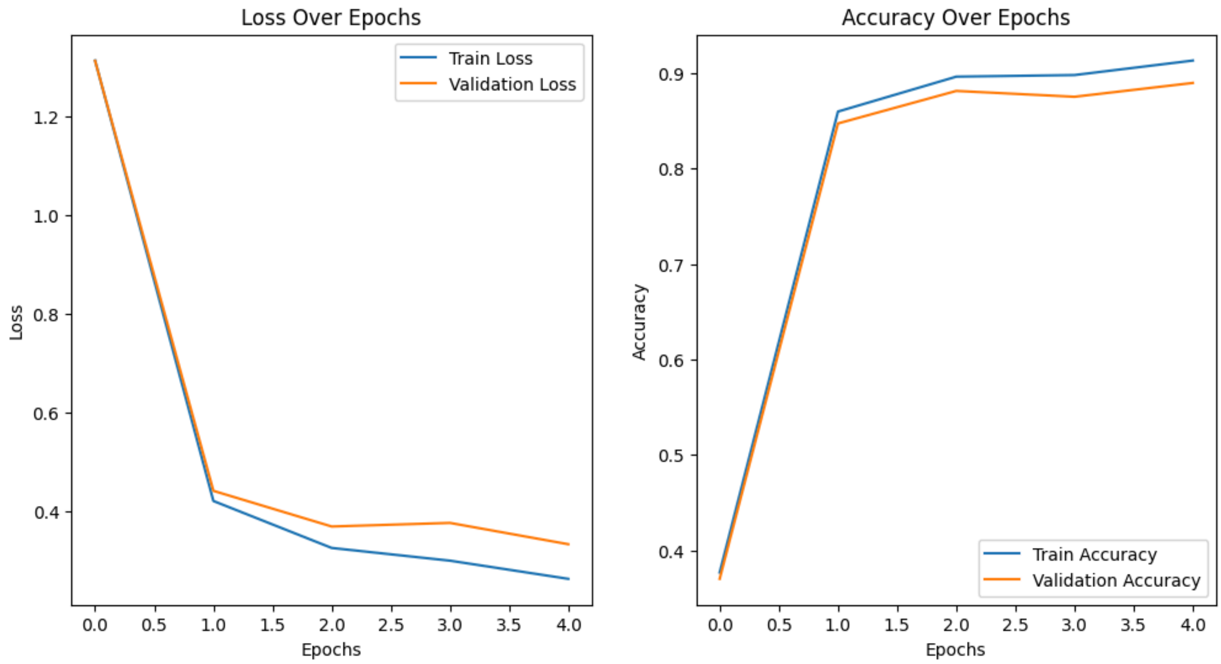
Base model performance curves

Training Accuracy: 91.31%
Training loss: 0.26
Validation Accuracy: 88.97
Validation loss: 0.33
Test Accuracy: 88.49
Test loss: 0.53

Performance metrics:

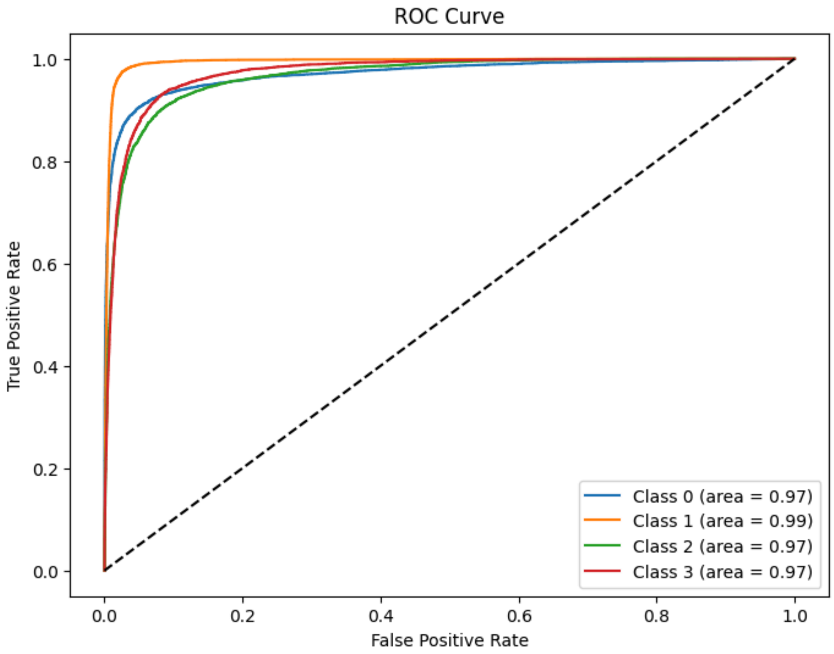
Precision: 0.884
Recall: 0.88
F1 Score: 0.88

Plot of Loss and Accuracy Curves:



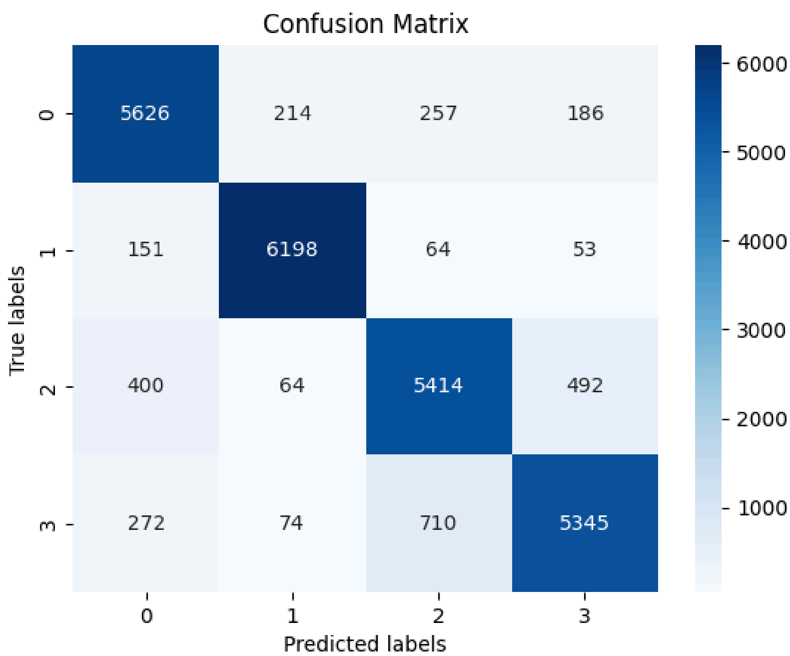
The ideal case is for the loss to steadily decrease and the accuracy to steadily increase over the epochs. This means that the model is learning and improving its performance on the training data.

ROC Curve:



The four curves in the plot have a high AUC greater than 0.95, which suggests that the model is performing well at classifying all four classes.

Confusion Matrix:



Techniques impacted the performance of the model:

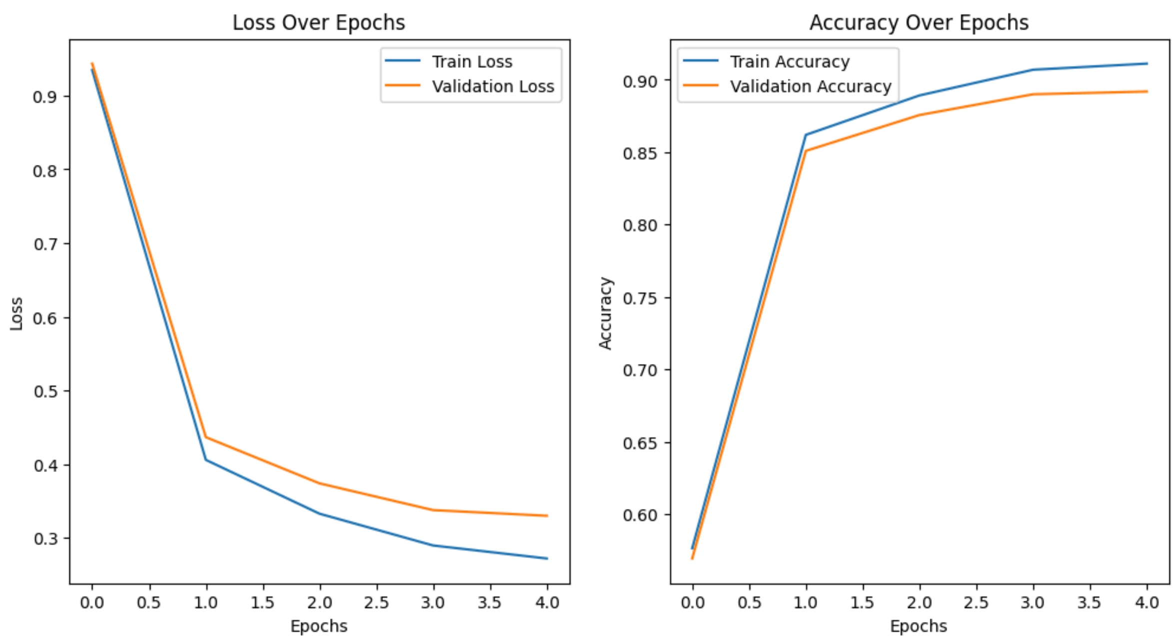
Dropout:

Training Accuracy: 91.11%
Training loss: 0.27
Validation Accuracy: 89.18
Validation loss: 0.33
Test Accuracy: 89.04
Test loss: 0.338

Performance metrics:

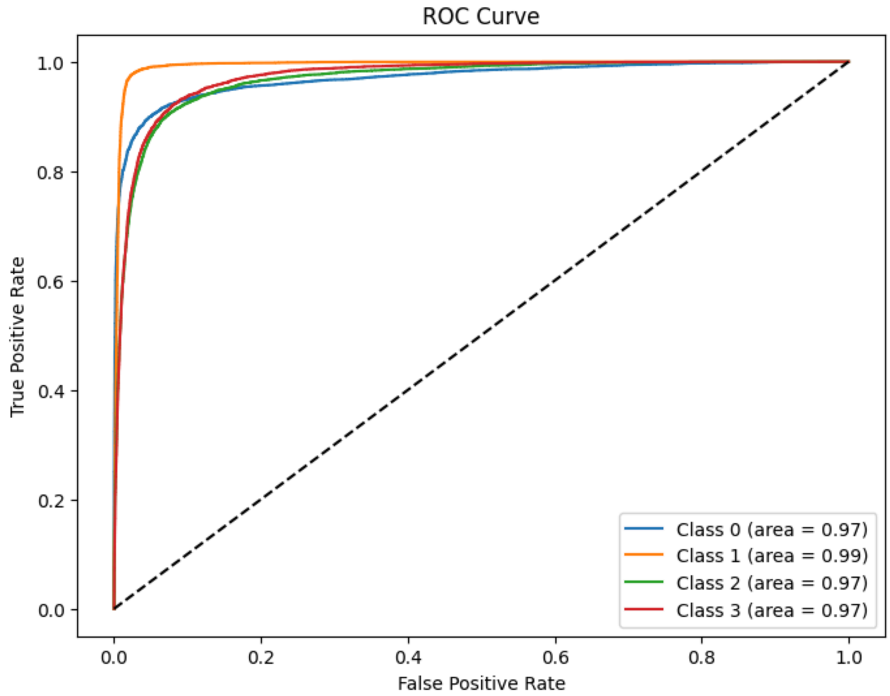
Precision: 0.8907
Recall: 0.8901
F1 Score: 0.8902

Plot of Loss and Accuracy Curves:



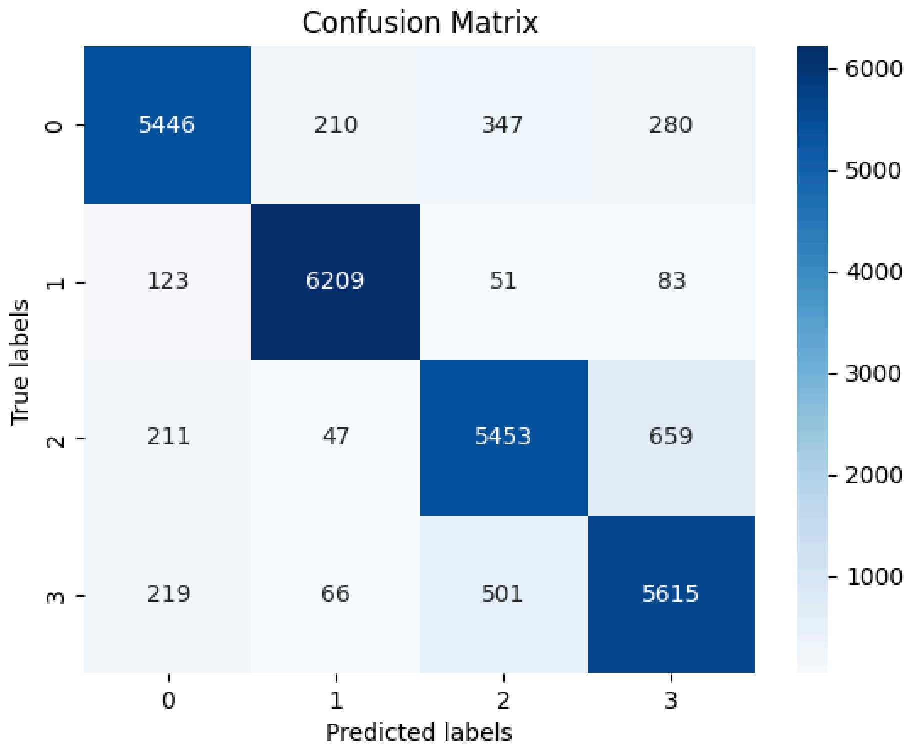
After applying dropout technique to the base model the performance metrics improved and also test accuracy increased and test loss decreased. That means model's performance is increased after applying this technique

ROC Curve



The area under the curve is approximately equal to 1 that means the model is performing well.

Confusion Matrix



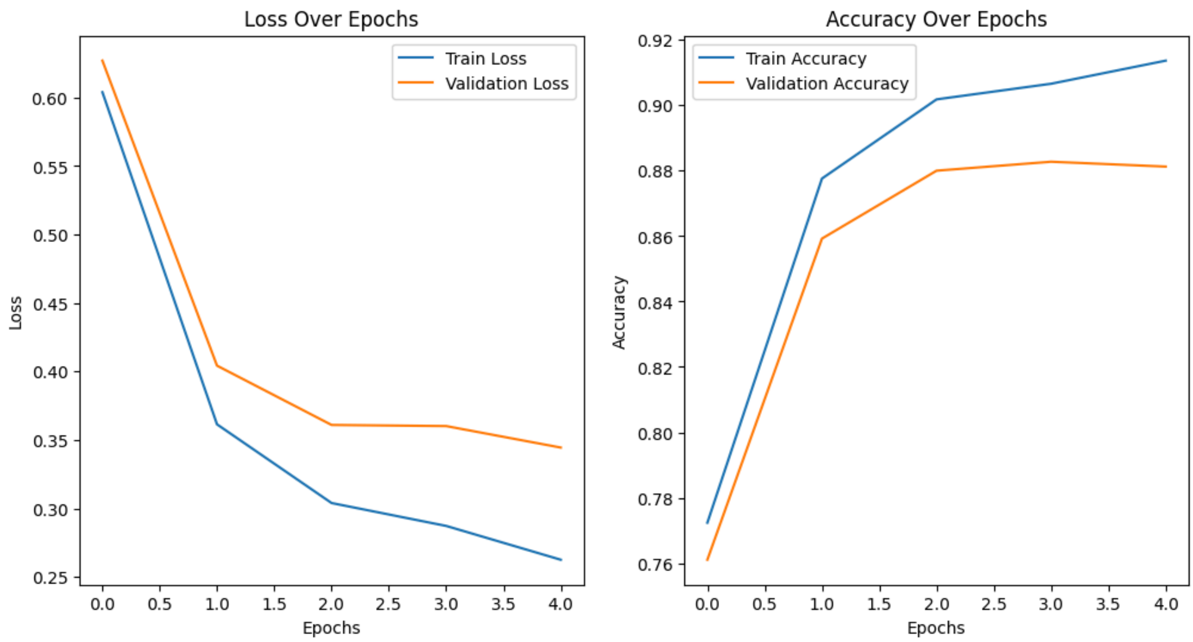
L2 Regularization:

Training Accuracy: 91.37%
Training loss: 0.26
Validation Accuracy: 88.13
Validation loss: 0.34
Test Accuracy: 88.59
Test loss: 0.34

Performance metrics:

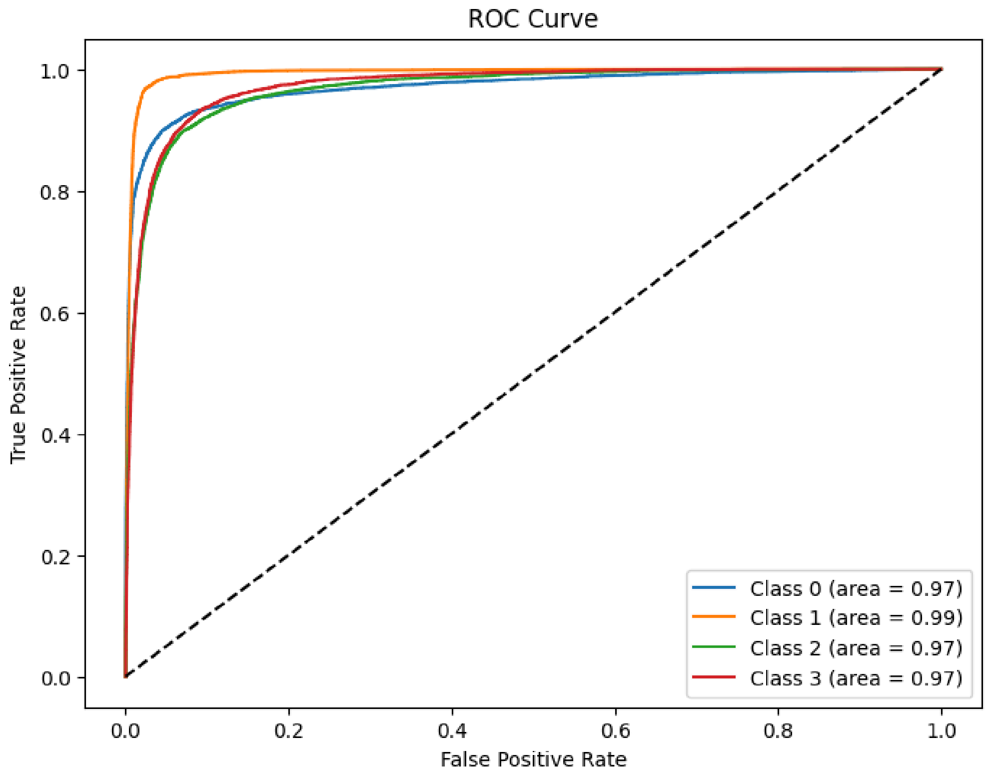
Precision: 0.88
Recall: 0.88
F1 Score: 0.88

Plot of Loss and Accuracy Curves:



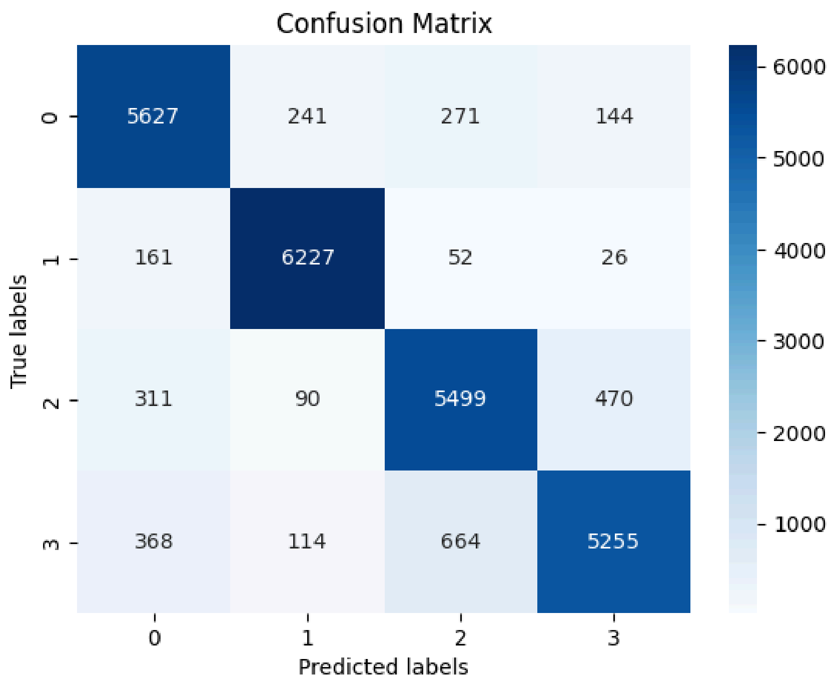
The performance of the model after applying L2 regularization improved slightly from 88.49 to 88.59, whereas performance metrics such as precision, recall, F1-score remained the same.

ROC Curve:



The area of the ROC curve is nearly equal to 1, basically it represents the model is predicting correct outputs.

Confusion Matrix



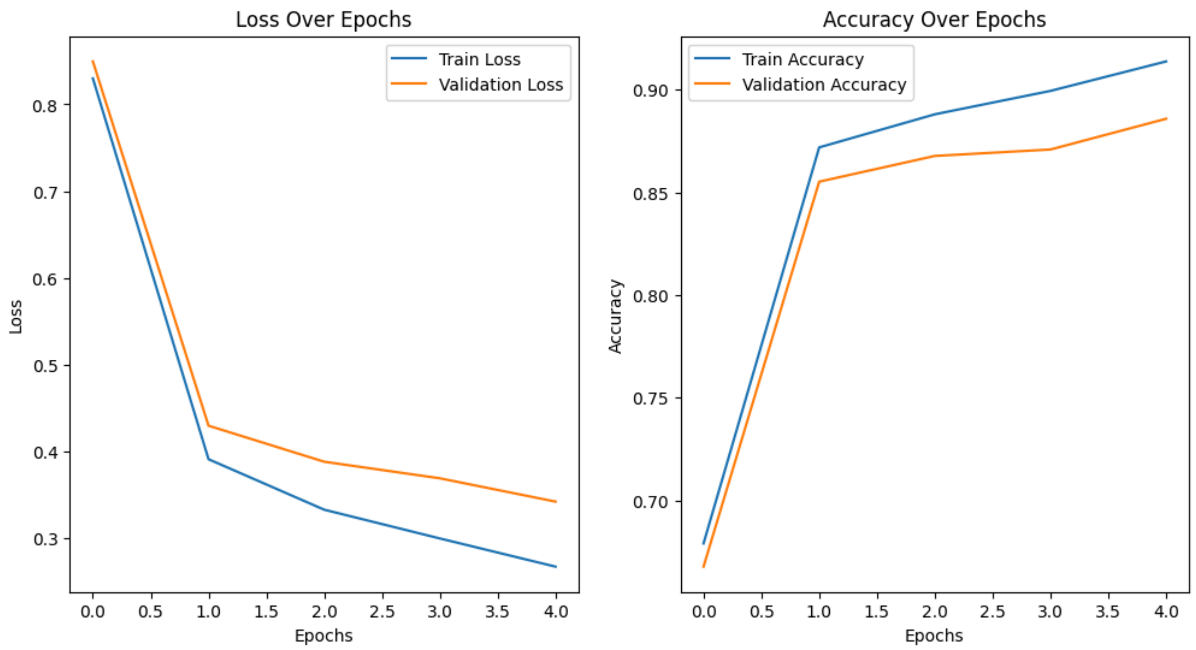
Early Stopping:

Training Accuracy: 91.37%
Training loss: 0.26
Validation Accuracy: 88.58
Validation loss: 0.34
Test Accuracy: 88.55
Test loss: 0.34

Performance metrics:

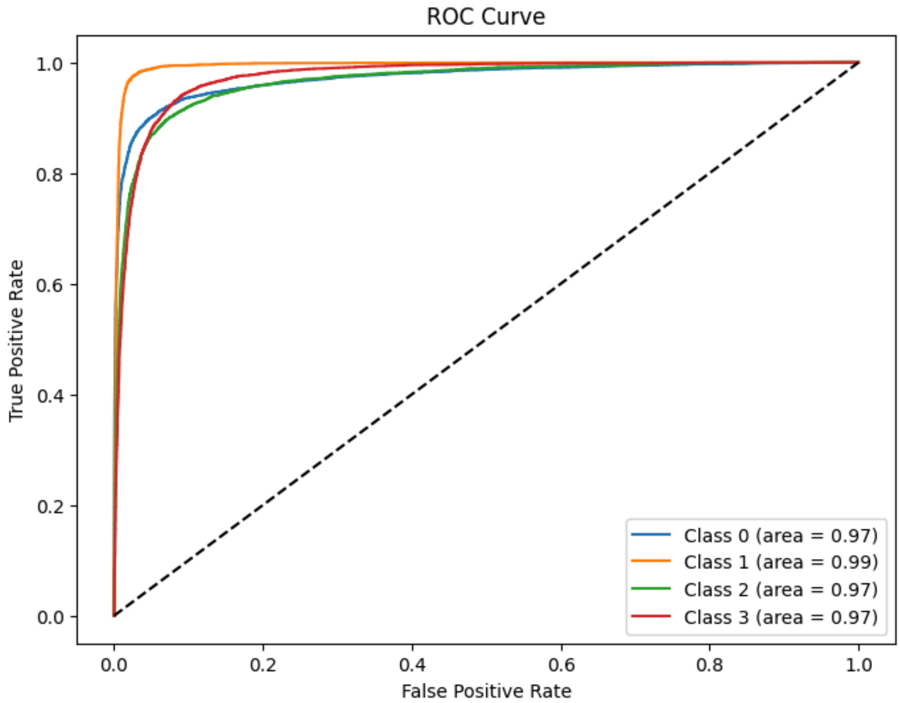
Precision: 0.88
Recall: 0.88
F1 Score: 0.88

Plot of Loss and Accuracy Curves:



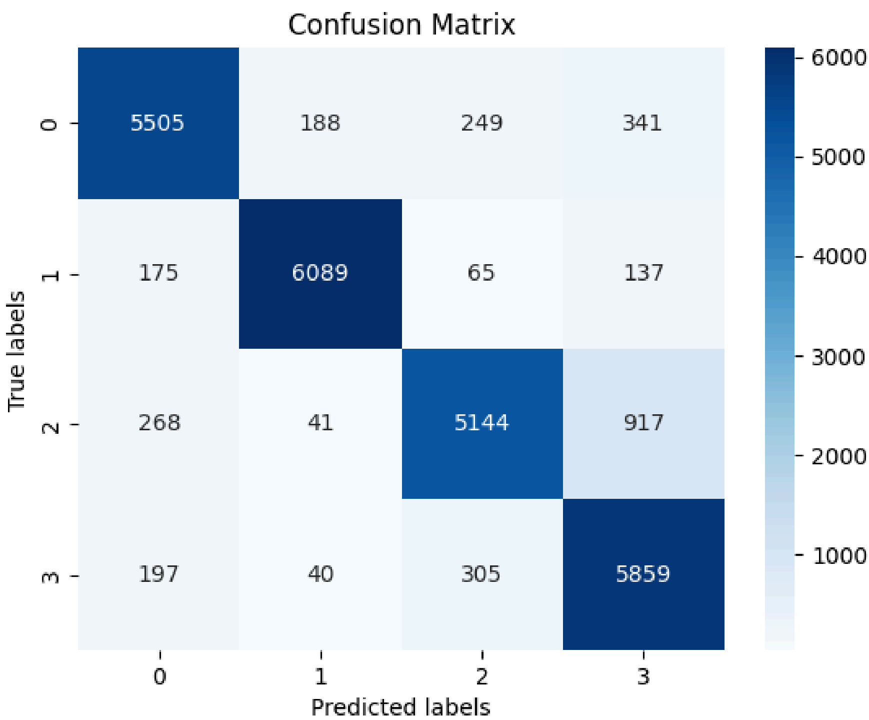
Applying Early stopping the model performance metrics improved very slightly and there is improvement in test and train accuracies.

ROC Curve



The area under the curve is ~ 1. The model is predicting all the classes perfectly.

Confusion Matrix:



Observation:

After applying the techniques they prevented overfitting of the model and improved performance. They obtain this by applying different constraints on the model's parameters or training process, which leads to the model being robust to unseen data.

References:

1. **https://arxiv.org/pdf/2003.05991.pdf**

2. **https://arxiv.org/abs/1606.05908**

3. **https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer**

4. **https://pytorch.org/text/stable/data_utils.html**

5. **https://arxiv.org/abs/2010.11929**

6. **https://arxiv.org/pdf/1905.11946.pdf**

7. **https://arxiv.org/abs/2010.11929**

8. **dharmaac_assignment1**

9. **adarshre_assignment1**