ASSIGNMENT 2

Full Name	UBIT Name	UBID
Kusha Kumar Dharavath	Kushakum	50532663
Charan Kumar Nara	cnara	50545001

Part I

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

Given,

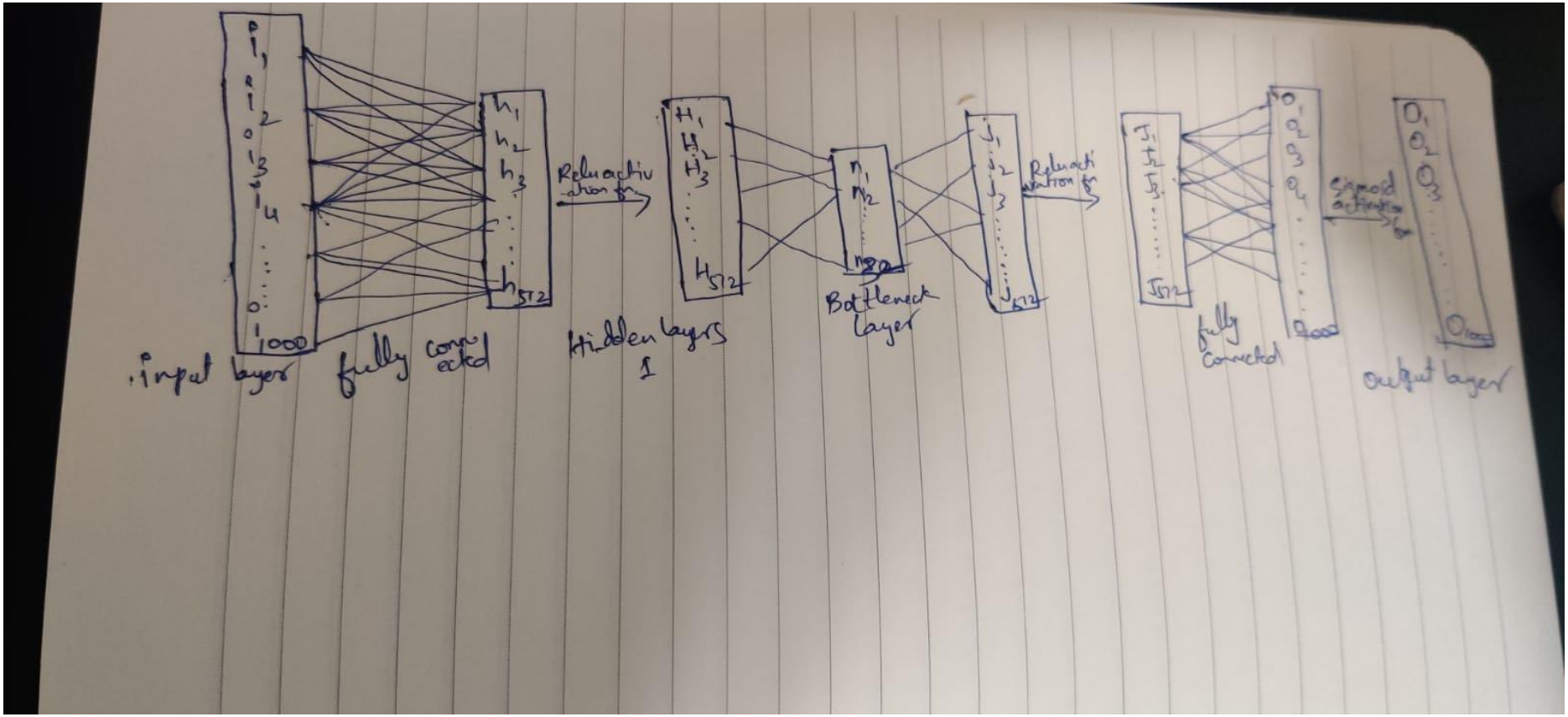
Input layer: 1000 units
Output from input layer to hidden layer1: 512
Input for hidden layer: 512
Output to Bottleneck layer: 32
Input for Bottleneck layer: 32
Output to hidden layer2: 512
Input for hidden layer2: 512
Output to output layer: 1000

Number of Parameters = (Number of Inputs×Number of Outputs)+Number of Outputs

- Input Layer to Hidden Layer 1:
  Weights: 1000×512=512,0001000×512=512,000
  Biases: 512512
  Total: 512,000+512=512,512512,000+512=512,512
- Hidden Layer 1 to Bottleneck Layer:
  Weights: 512×32=16,384512×32=16,384
  Biases: 3232
  Total: 16,384+32=16,41616,384+32=16,416
- Bottleneck Layer to Hidden Layer 2:
  Weights: 32×512=16,38432×512=16,384
  Biases: 512512
  Total: 16,384+512=16,89616,384+512=16,896
- Hidden Layer 2 to Output Layer:
  Weights: 512×1000=512,000512×1000=512,000
  Biases: 10001000
  Total: 512,000+1000=513,000512,000+1000=513,000

Adding all the results, 512,512+16,416+16,896+513,000=1,058,824 parameters

2. Generate a computational graph for the autoencoder


input layer
fully connected
Hidden layers
1
Bottleneck Layer
fully connected
output layer

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

Data Quality and Availability:
Autoencoders heavily rely on the availability of extensive, high-quality, and representative training data, which can be challenging to obtain in manufacturing settings, especially for complex components. Lack of diverse and well-labeled anomaly samples in the training data can severely limit the autoencoder's ability to accurately detect anomalies during the production process.

Limited Feature Learning:
An autoencoder's ability to detect anomalies depends on learning detailed and distinctive features from data. However, architectural or capacity limitations might prevent it from capturing essential patterns and subtleties that define anomalies in manufacturing. This could significantly hinder its performance, especially in identifying anomalies that are minor deviations from normal conditions.

Generalisation to unseen anamolies:
Autoencoders trained primarily on normal data may generalize excessively to nominal operating conditions, hindering their ability to reliably detect anomalies that significantly deviate from the training distribution, potentially resulting in a high rate of missed anomalies or false negatives.

Adapting to changing conditions:
Manufacturing operations often evolve, influenced by shifts in operating conditions, variations in raw materials, or equipment aging. Autoencoders based on past data might not keep pace with these changes, increasing the risk of incorrect anomaly detection or overlooked defects. Periodic updates or real-time learning methods are essential to maintain the model's relevance and accuracy in line with the changing manufacturing landscape.

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

Transfer learning adaption:
Employing transfer learning techniques enables the autoencoder to adapt more effectively to evolving conditions or data variations in the manufacturing process. By pre-training the model on a larger, diverse dataset and subsequently fine-tuning it on the specific manufacturing data, the autoencoder can better capture relevant features while reducing the training time required for adaptation to new tasks or scenarios.

Integrating Expert Knowledge:
Harnessing the knowledge and insights from manufacturing domain experts can significantly enhance the anomaly detection system's performance and relevance. This involves incorporating feature engineering techniques to extract domain-specific attributes that complement the learned representations from the autoencoder. Also, domain experts can advise on prioritizing certain anomaly types and assist in analyzing detected irregularities, enriching the model's relevance and interpretability in the manufacturing context.

Attention Mechanisms:
Integrate attention mechanisms into the autoencoder architecture to allow the model to focus on the most relevant features or regions of the input data during the encoding and decoding processes.

This can help capture important patterns and nuances that might be overlooked by standard autoencoders.

Incorporating Interpretability and Diagnostics:
Integrating interpretability tools and diagnostic methods enables understanding the root causes of detected anomalies, facilitating prompt interventions in the manufacturing process.

Part II

Dataset: art_daily_flatmiddle (artificialWithAnomaly folder) from Numenta Anomaly Benchmark (NAB) (kaggle.com)

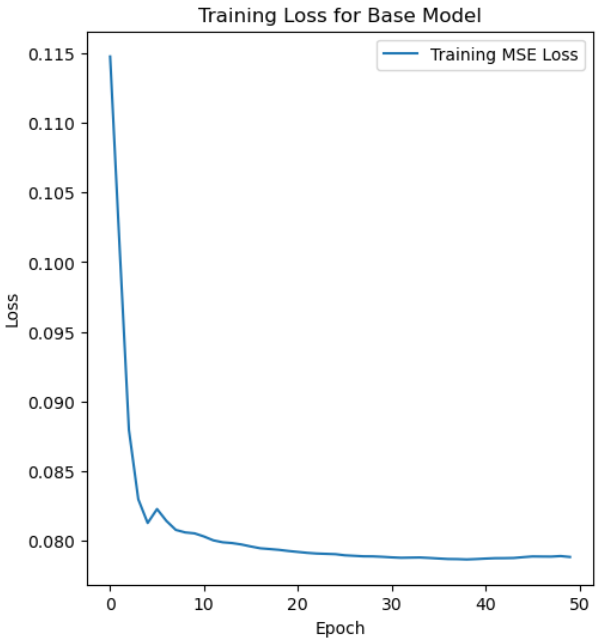
Reason for choosing this dataset: We chose this dataset because it is related to our background in data analysis and keen interest in time-series anomaly detection. Working with this dataset allows us to leverage our expertise in analysing complex time series data, where detecting anomalies is crucial for monitoring systems, identifying potential issues, and taking proactive measures.

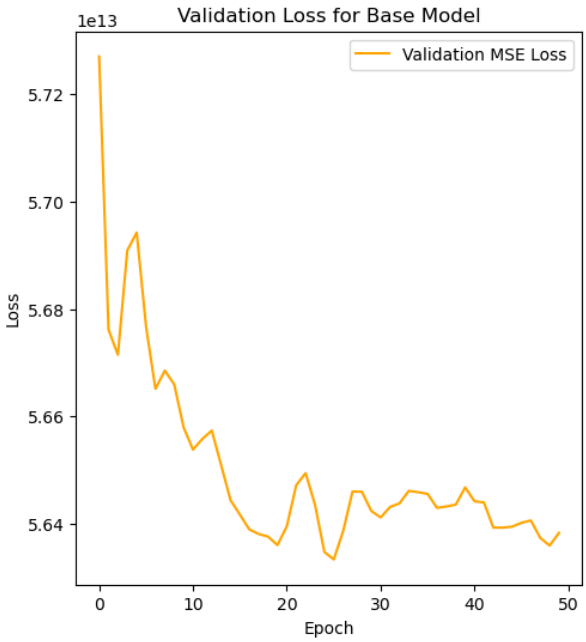
Details about the dataset: Is a time-series data collection that includes observations of events or measurements recorded sequentially over time. The dataset contains 4032 entries and two columns named timestamp (datetime64 type) and value(float64 type).

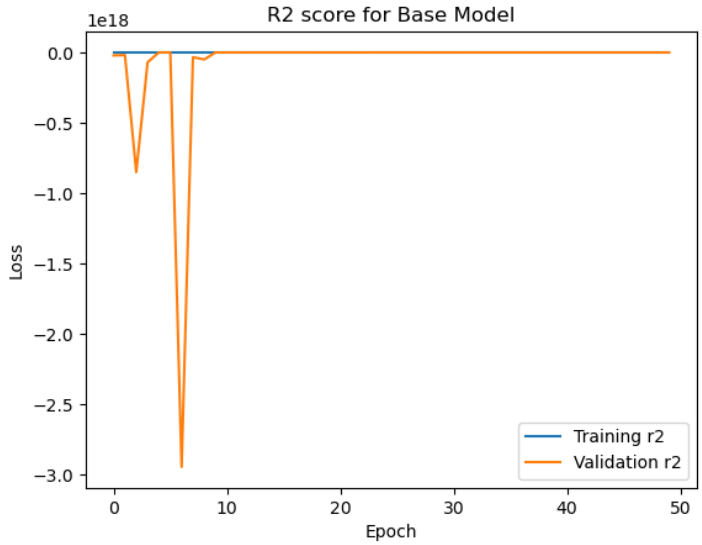
Standard Autoencoder:

Encoder: 1 Linear layer
Decoder: 1 Linear layer
Activation Function: ReLU (used after each linear layer)
Learning Rate: 0.01
Loss Function: Mean Squared Error
Optimizer: Adam
Threshold: Reconstruction errors exceeding the 95th percentile is considered anomalies

Train loss = 0.0788
Validation loss = 56383349194752.0
Test Loss: 1.523915410041809
R2 Score: -0.003650547707805263
Error (MSE): 0.680251





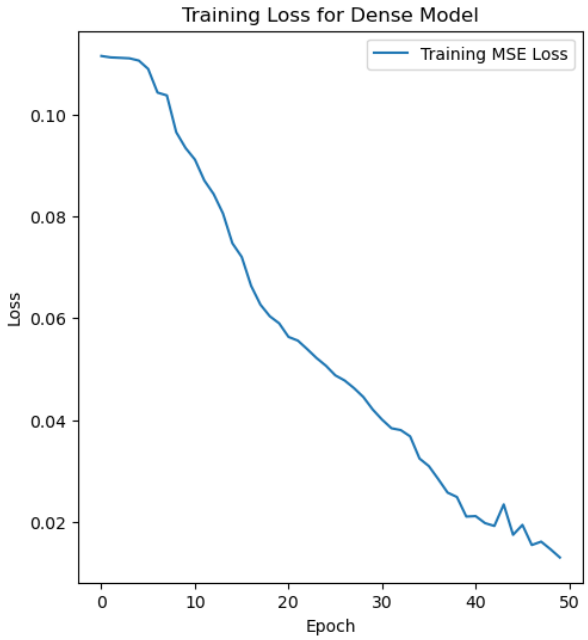


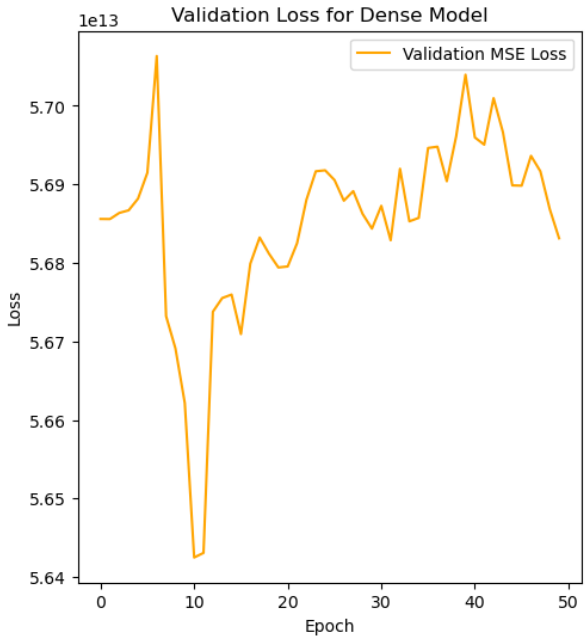
Autoencoder - Dense Layer:

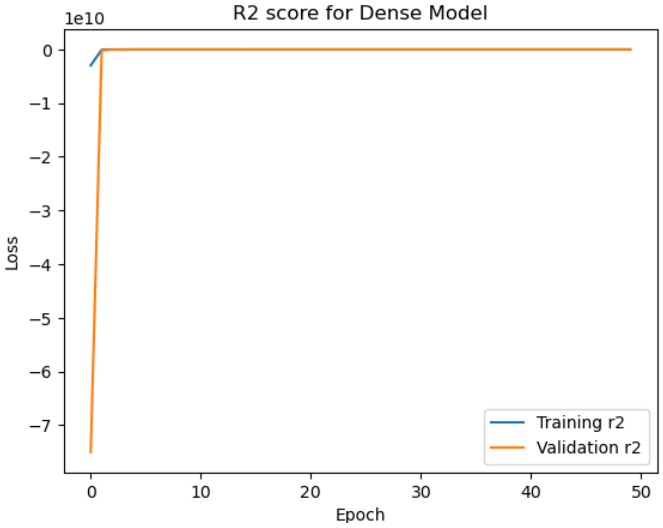
Encoder: 5 Linear layers
Decoder: 5 Linear layers
Activation Function: ReLU (used after each linear layer except the last layer of the decoder)
Learning Rate: 0.01
Loss Function: Mean Squared Error
Optimizer: Adam
Threshold: Reconstruction errors exceeding the 95th percentile is considered anomalies

Train loss = 0.0130
Validation loss = 56831313444864.0
Test Loss: 2.0110275745391846

R2 Score: -0.32446240165668505
Error (MSE): 0.07016533



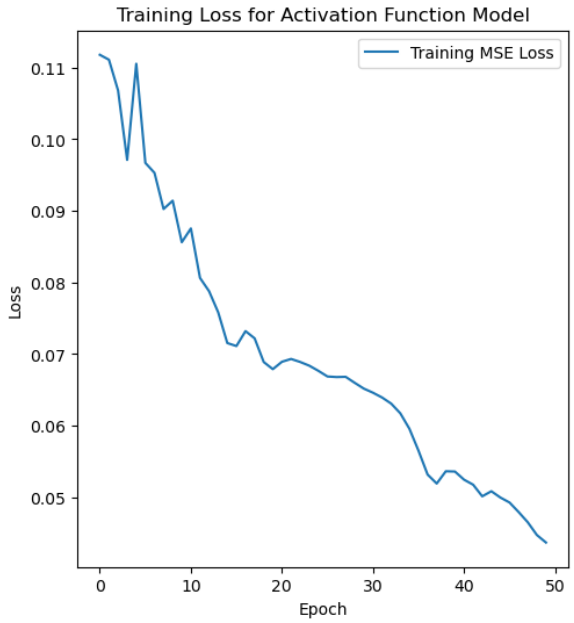


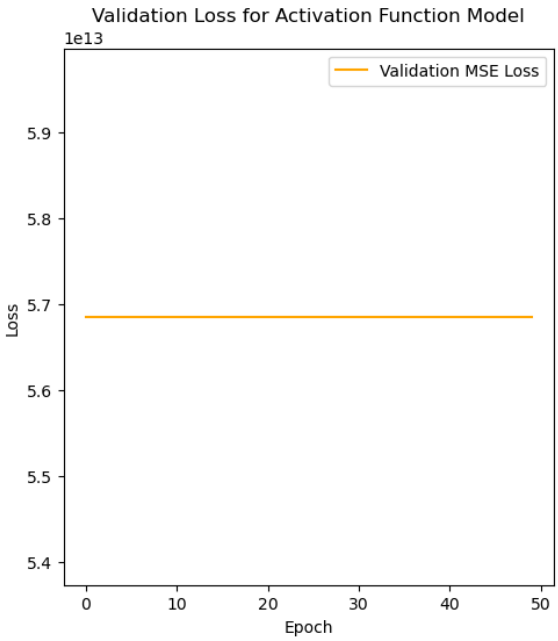


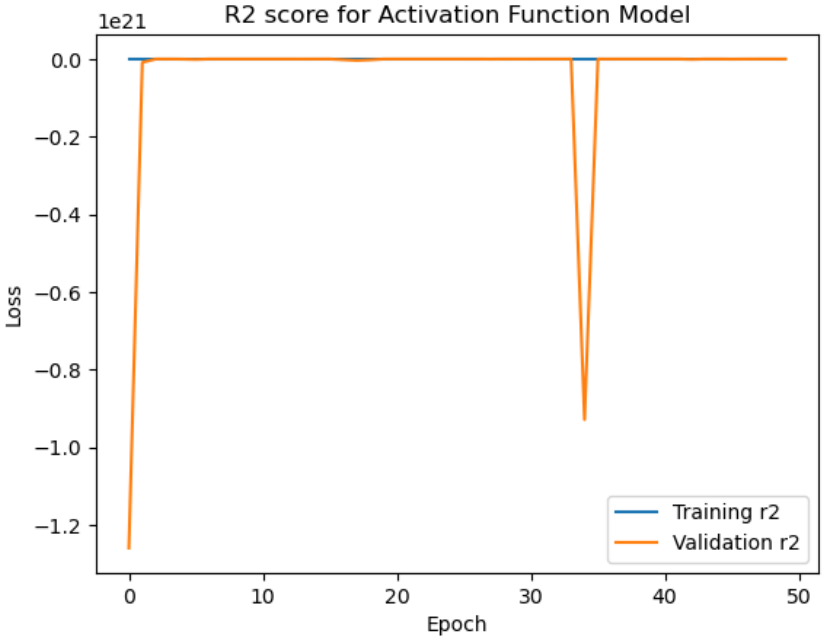
Autoencoder – Dense layer with activation function:

Encoder: 5 Linear layers
Decoder: 5 Linear layers
Activation Function: Tanh (used after each linear layer except the last layer of the decoder)
Learning Rate: 0.01
Loss Function: Mean Squared Error
Optimizer: Adam
Threshold: Reconstruction errors exceeding the 95th percentile is considered anomalies

Train loss = 0.0437
Validation loss = 56854294036480.0
Test Loss: 1.9293618202209473
R2 Score: -0.2706772370692123
Error (MSE): 0.33477625



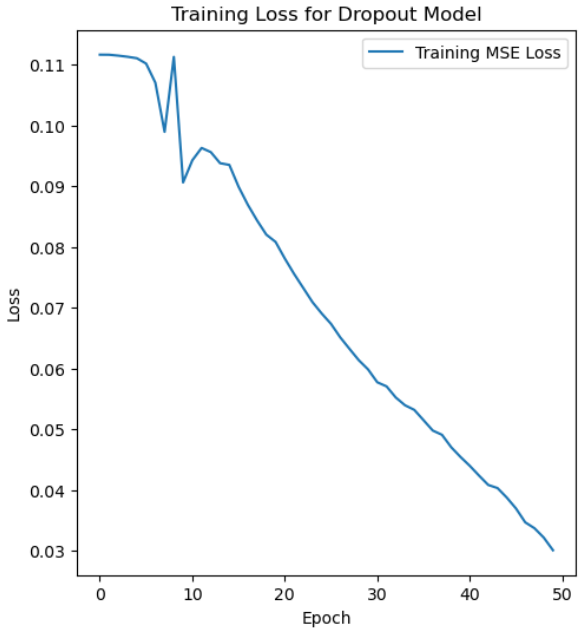


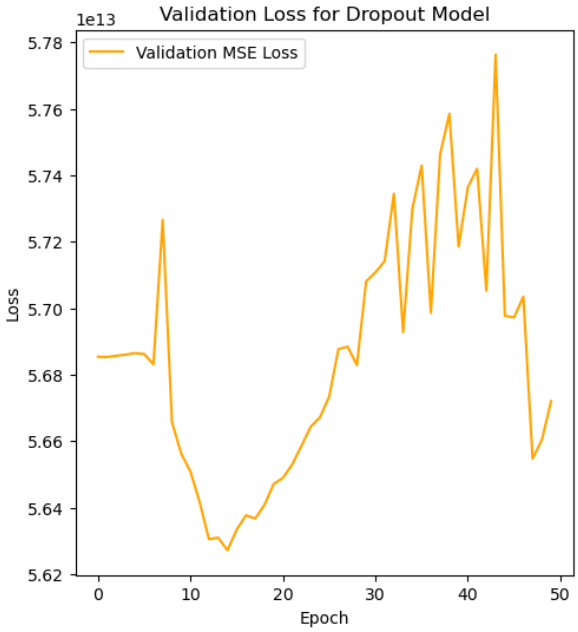


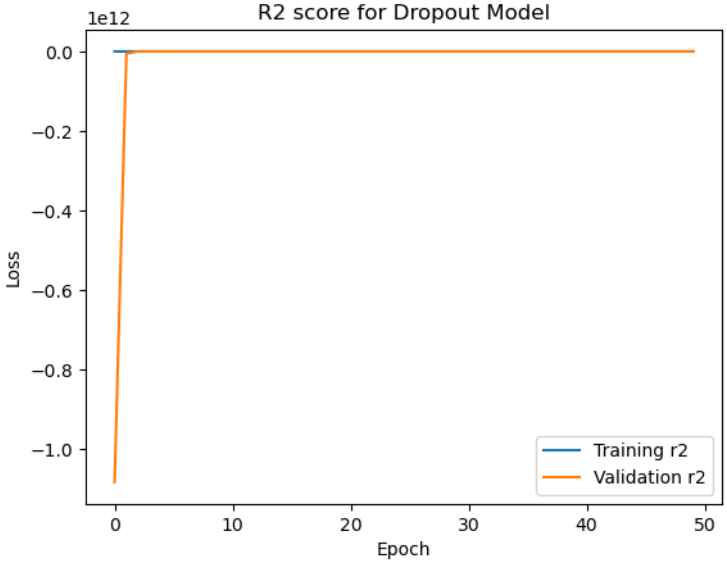
Autoencoder with Dropout:

Encoder: 5 Linear layers
Decoder: 5 Linear layers
Activation Function: Relu (used after each linear layer except the last layer of the decoder)
Learning Rate: 0.01
Loss Function: Mean Squared Error
Optimizer: Adam
Threshold: Reconstruction errors exceeding the 95th percentile is considered anomalies

Train loss = 0.0301
Validation loss = 56720743202816.0
Test Loss: 2.024054527282715
R2 Score: -0.3330420200650692
Error (MSE): 0.563371







Discuss the strengths and limitations of using autoencoders for anomaly detection

Strengths:

Dimensionality Reduction: They can reduce the dimensionality of the data, which helps in visualizing and understanding the structure of high-dimensional data that may be related to anomaly detection.

Robust to Noise: Autoencoder models are robust to noise in the input data because they learn to reconstruct the original input from a compressed representation. This is beneficial for real-world scenarios where data often comes with noise that can mask true anomalies.

Adaptability: Autoencoders can be adapted to various types of data, including images, time-series, and structured data, by tailoring the network architecture and loss functions to the specific problem at hand.

Non-linearity: Unlike many traditional anomaly detection techniques, autoencoders can capture non-linear relationships in the data, which may lead to more effective identification of complex anomalies.

Limitations:

Interpretability: Autoencoders, like many deep learning models, can be challenging to interpret, making it difficult to understand the underlying patterns and features that the model has learned

Difficulty in Setting Thresholds: Determining the threshold that distinguishes between normal and anomalous data can be challenging and often requires domain knowledge or trial-and-error.

Computational Complexity: Training autoencoders, especially deep architectures are computationally intensive and so requires significant computational resources and time, especially for large datasets.

Overfitting: Autoencoders can overfit to the training data, especially if there is a lack of sufficient regularization or the network

Part III

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:

Query(Q) Transformation: Each input vector x_i is transformed into a query vector q_i using a weight matrix W^Q. The query vector is obtained by multiplying the input sequence with the weight matrix:

$$q_i = W^Q . x_i$$

The query vector $q \in \mathbb{R}$^($N \times d_k$) plays the role of "asking questions" about the sequence elements, where d_k is the dimensionality of the query/key vectors.

Key(K) Transformation: Each input vector x_i is transformed into a query vector k_i using a weight matrix W^K. The query vector is obtained by multiplying the input sequence with the weight matrix:

$$k_i = W^K . x_i$$

The query vector $q \in \mathbb{R}^{\wedge}(N \times d_k)$ plays the role of "asking questions" about the sequence elements, where d_k is the dimensionality of the query/key vectors.

Value(V) Transformation: Each input vector x_i is transformed into a query vector v_i using a weight matrix W^V. The query vector is obtained by multiplying the input sequence with the weight matrix:

$$v_i = W^V . x_i$$

Scaled dot product attention:
Scaled dot product attention calculates attention scores based on the interaction between the query (q_i) and key (k_j) vectors. Attention score between a query vector (q_i) and key vector (k_j) is calculated by taking the dot product of q_i and k_j and then then scaling the result by the inverse square root of the dimension of the key vector d_k.

So, the formula to calculate attention score using q_i and k_j is: $(q_i \, . \, k_j)/\sqrt{d_k}$

Role of $\sqrt{d_k}$:

The scaling factor $\sqrt{d_k}$plays a critical role in the normalization process by preventing the dot product values from growing too large in magnitude. The dimensionality d_k of the key vectors influences the magnitude of the dot products. As d_k increases, the expected value of the dot products increases linearly.

Weighted Sum: After the attention score is calculated and normalized, the next step in scaled dot product attention mechanism is to use these scores to compute a weighted sum of the value vectors (v_j).

The weighted sum is calculated by multiplying each value vector v_j with its corresponding weight a_{ij} and then add these weighted values for each position i in the output sequence.

Mathematical formula for the weighted sum is: $\sum_{j=1}^{N} a_{ij} v_j$

a_{ij} is the attention weight between the *i*th query and *j*th key. It indicates how much focus should be placed on the *j*th value vector when constructing the *i*th element of the output. The v_j are the value vectors obtained from the input sequence through a linear transformation. The result of this weighted sum is the ith output vector, which is a composite representation formed by aggregating information across the entire input sequence, weighted according to the calculated attention scores.

Optional output transformation: The optional final linear transformation W_o and bias term b_o are applied to the weighted sum output from the attention mechanism to generate the final latent representation z.

Purpose of W_o and b_o:

- The final linear transformation gives an additional point of customization in the model's architecture. The dimensions of W_o can be chosen to project the output into a higher or

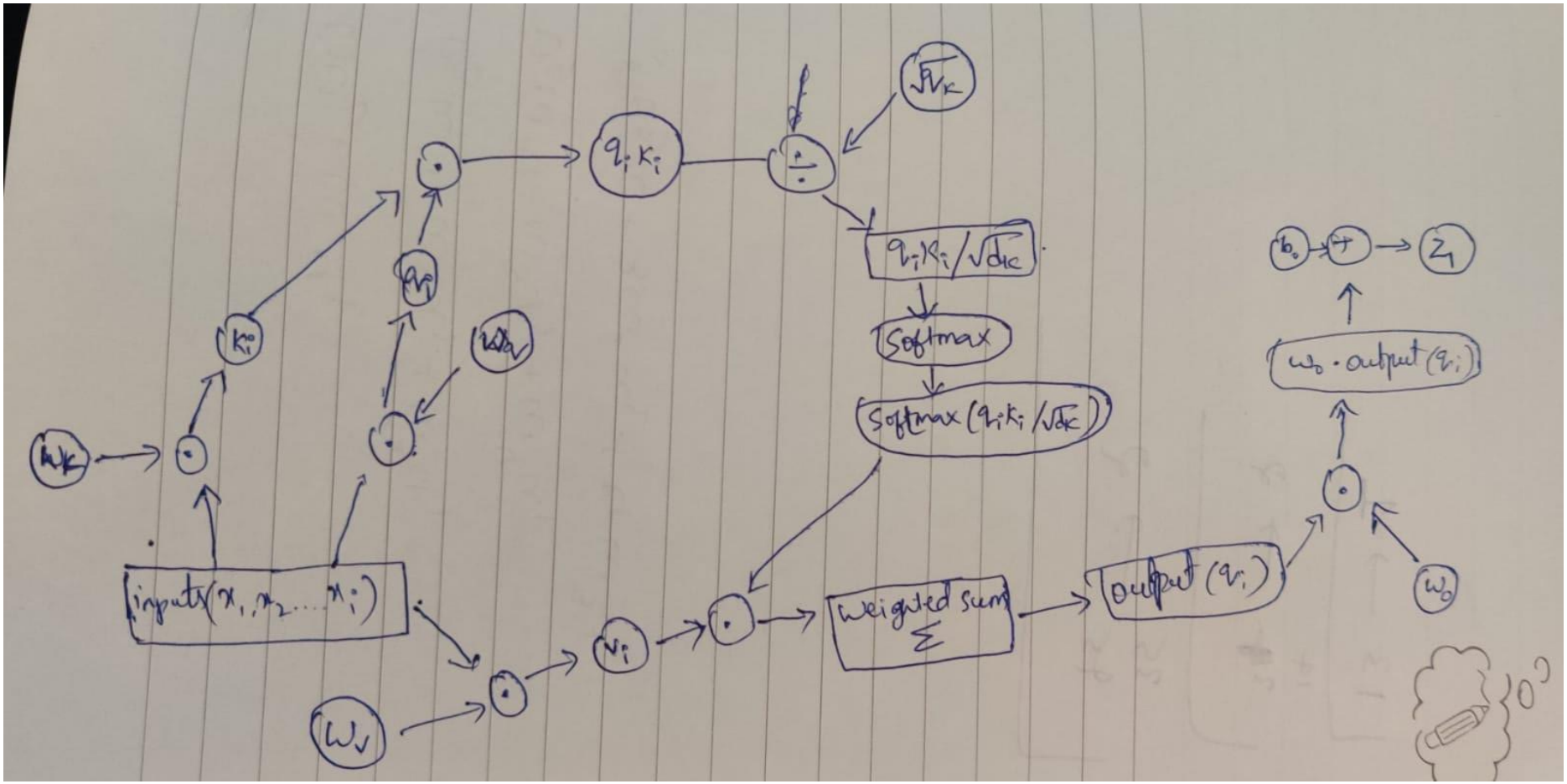
lower dimensional space depending on the task. This provides flexibility in designing models to meet specific requirements.

- The transformation via W_o and b_o allows the model to project the aggregated output from the attention mechanism into a space that is more suitable for the tasks the model is designed to perform. This step can refine the information content of the output, ensuring that the final latent representation (z) aligns better with the expected dimensions and characteristics for subsequent processing layers or for the final task output.
- Introducing parameters (W_o and b_o) allows model to learn more complex relationships and mappings from the input sequence to the output representation and also increases model's capacity. Model's ability to capture subtle nuances in the data also increases with the increased model capacity, leading to better performance on a wide range of tasks.

Potential Impact on the Final Representation:

- This transformation can also act as a filter which reduces noise or less relevant information present in the aggregated attention output, and so we get cleaner and more focused latent representation.
- It also helps in releasing the factors of variation in the data and makes the learning process more efficient, leading to a more generalized model.
- By carefully optimizing W_o and b_o, the model can better align the attention mechanism's output with the requirements of the specific task it is being used for

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


W_k
k_i
q_i
W_q
$q_i k_i$
$\sqrt{d_k}$
$q_i k_i / \sqrt{d_k}$
Softmax
Softmax $(q_i k_i / \sqrt{d_k})$
input $(x_1, x_2 \ldots x_i)$
W_v
v_i
weighted sum Σ
output (z_i)
w_0
$w_0 \cdot$ output (z_i)
b_0
z_1

Part IV

Dataset: ags_new_csv.tar.gz

Reason for choosing this dataset: This dataset is suitable for classification task, has balanced and diverse content and manageable size and easy to use. Due to these reasons, we chose ags_news_csv dataset for transformer model.

Transformer Model architecture

```
TransformerClassifier(
  (embedding): Linear(in_features=300, out_features=512, bias=True)
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      (0-2): 3 x TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=512,
out_features=512, bias=True)
        )
        (linear1): Linear(in_features=512, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=512, bias=True)
        (norm1): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (fc_out): Linear(in_features=512, out_features=4, bias=True)
)
```

Basic model results for Transformer Model

Test Loss: 0.4234, Test Accuracy: 0.8602

Accuracy: 86.02403343782655

Confusion Matrix: [[4139 140 249 220]

 [182 4592   51   79]

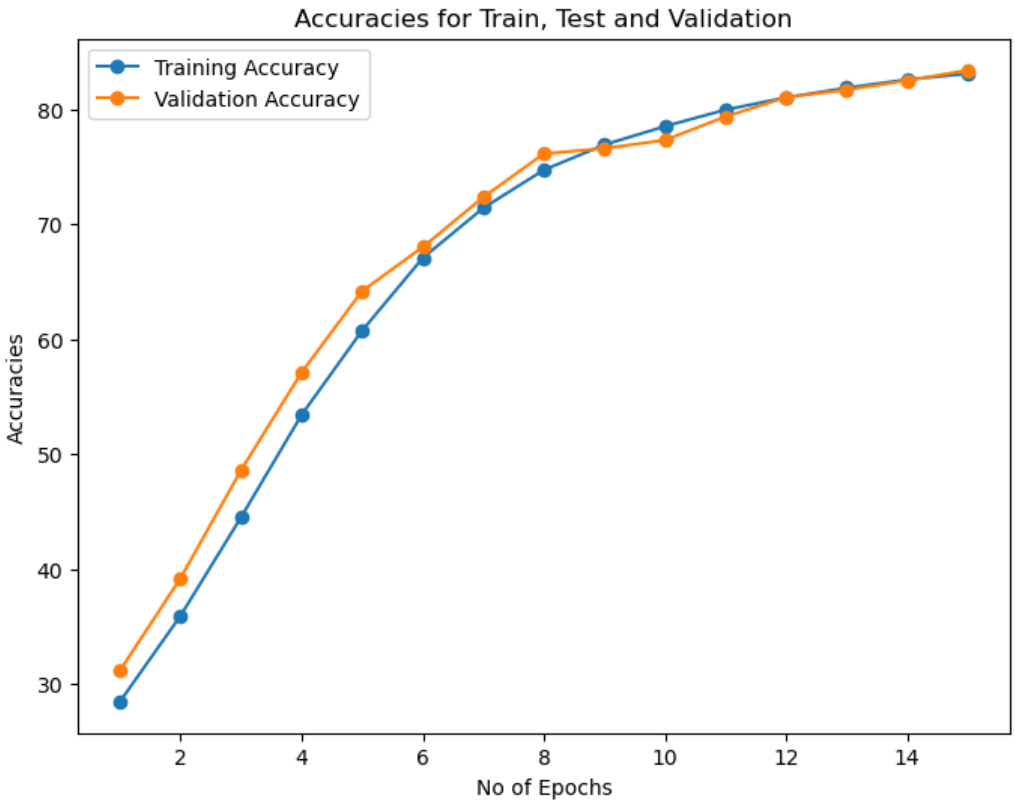
 [317   92 3989 630]

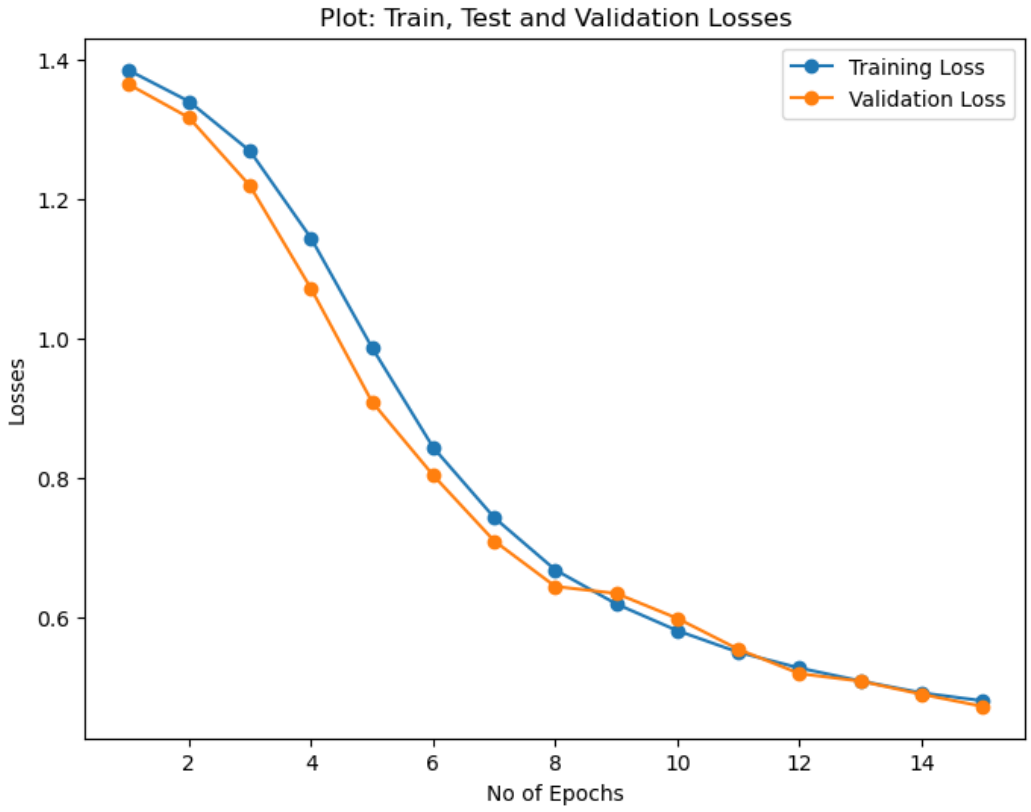
 [156   40 519 3745]]

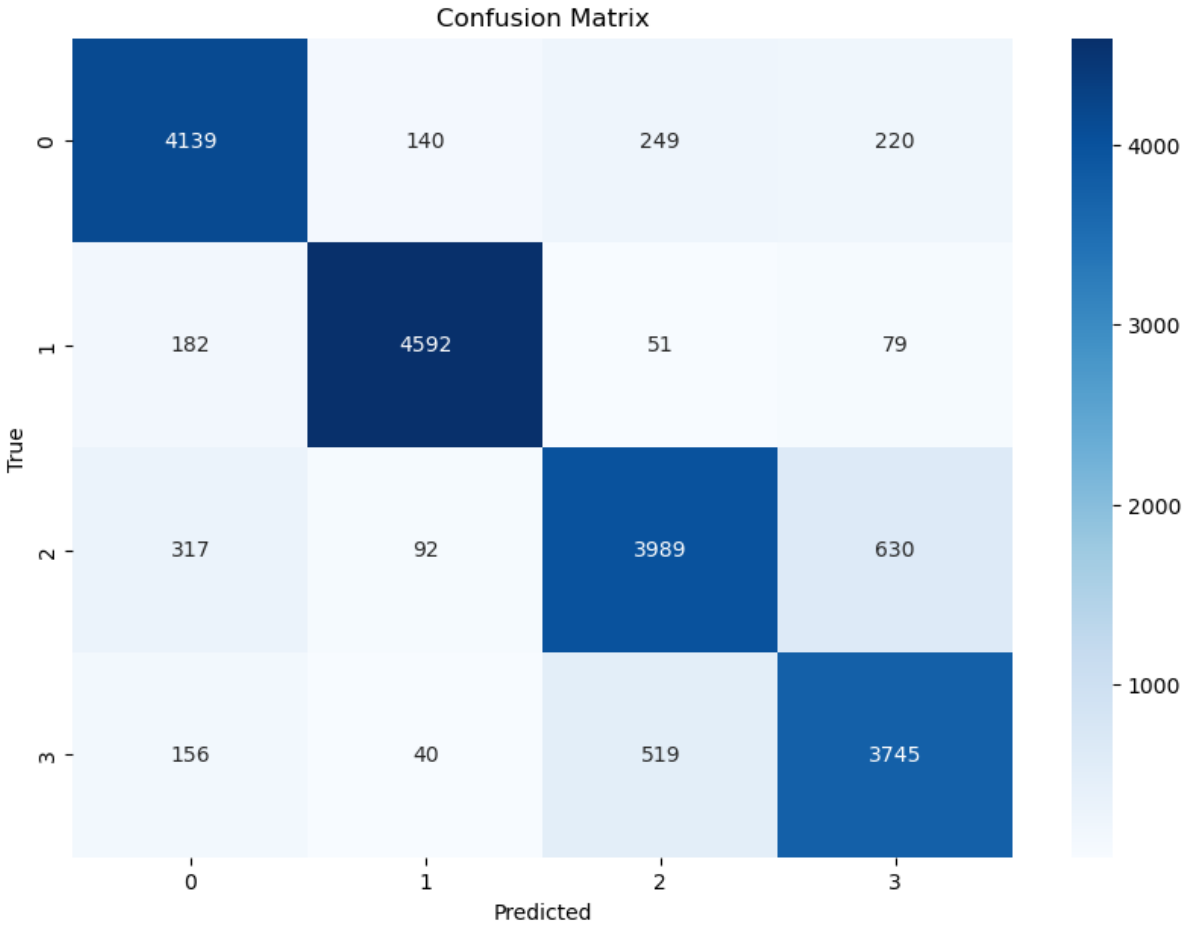
Precision: 85.95874091529045

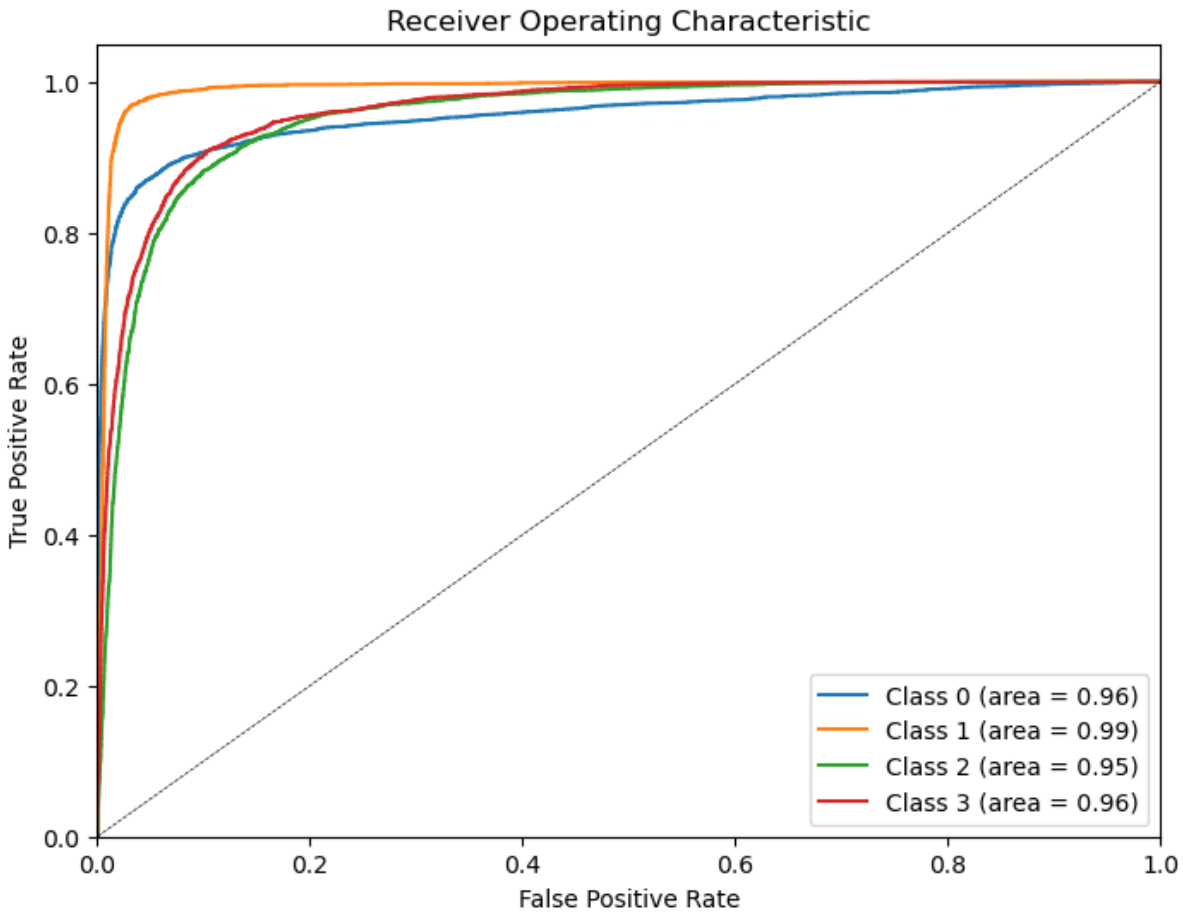
Recall: 86.02893081149232

F1 Score: 85.97152909752802




Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Losses
No of Epochs
1.4
1.2
1.0
0.8
0.6
2
4
6
8
10
12
14


Confusion Matrix
True
Predicted
4139
140
249
220
182
4592
51
79
317
92
3989
630
156
40
519
3745
0
1
2
3
4000
3000
2000
1000



L2 Regularization:

Test Loss: 0.4323, Test Accuracy: 0.8548

L2 Regularization model results for Transformer Model

Accuracy: 85.48066875653083

Confusion Matrix: [[4200 155 325 288]

[175 4595 53 92]

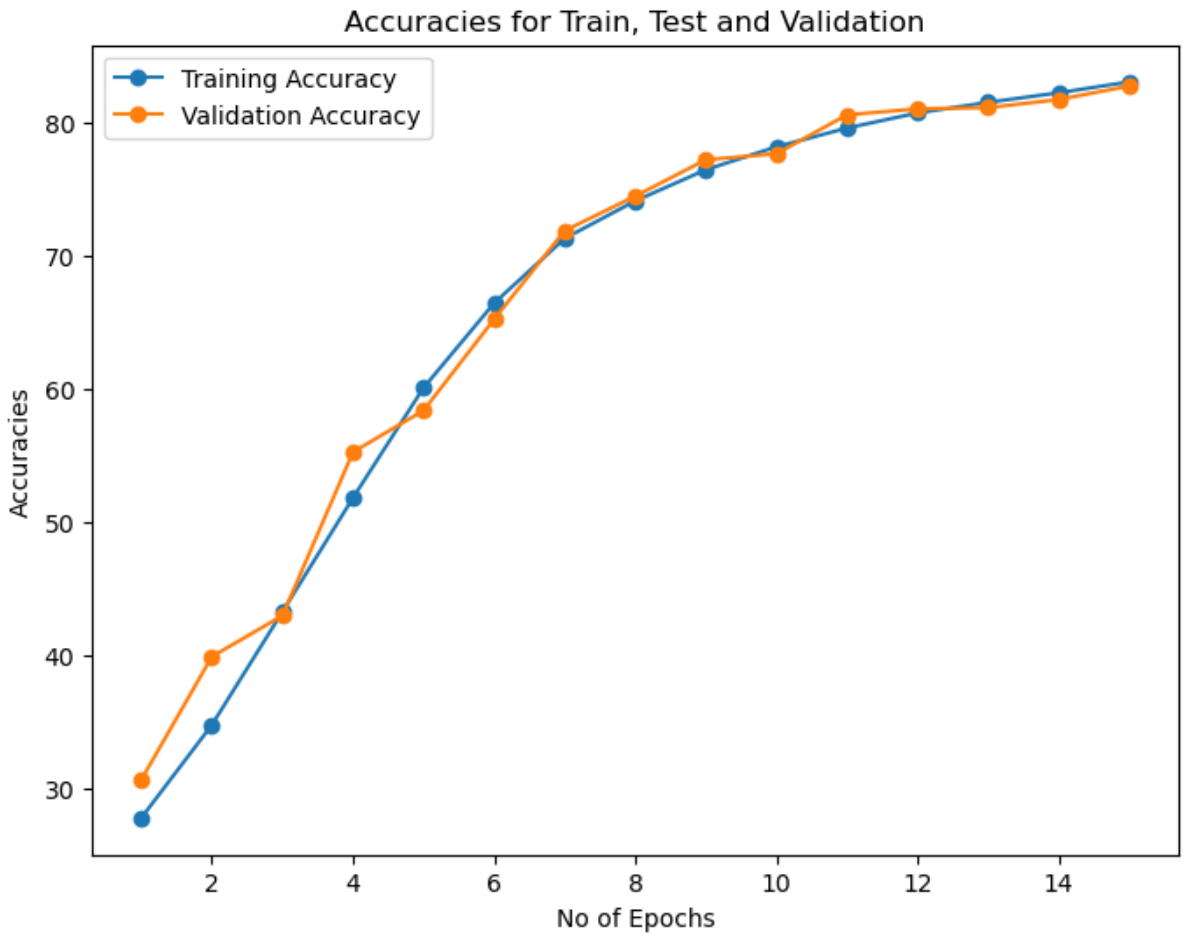
[301 87 3987 715]

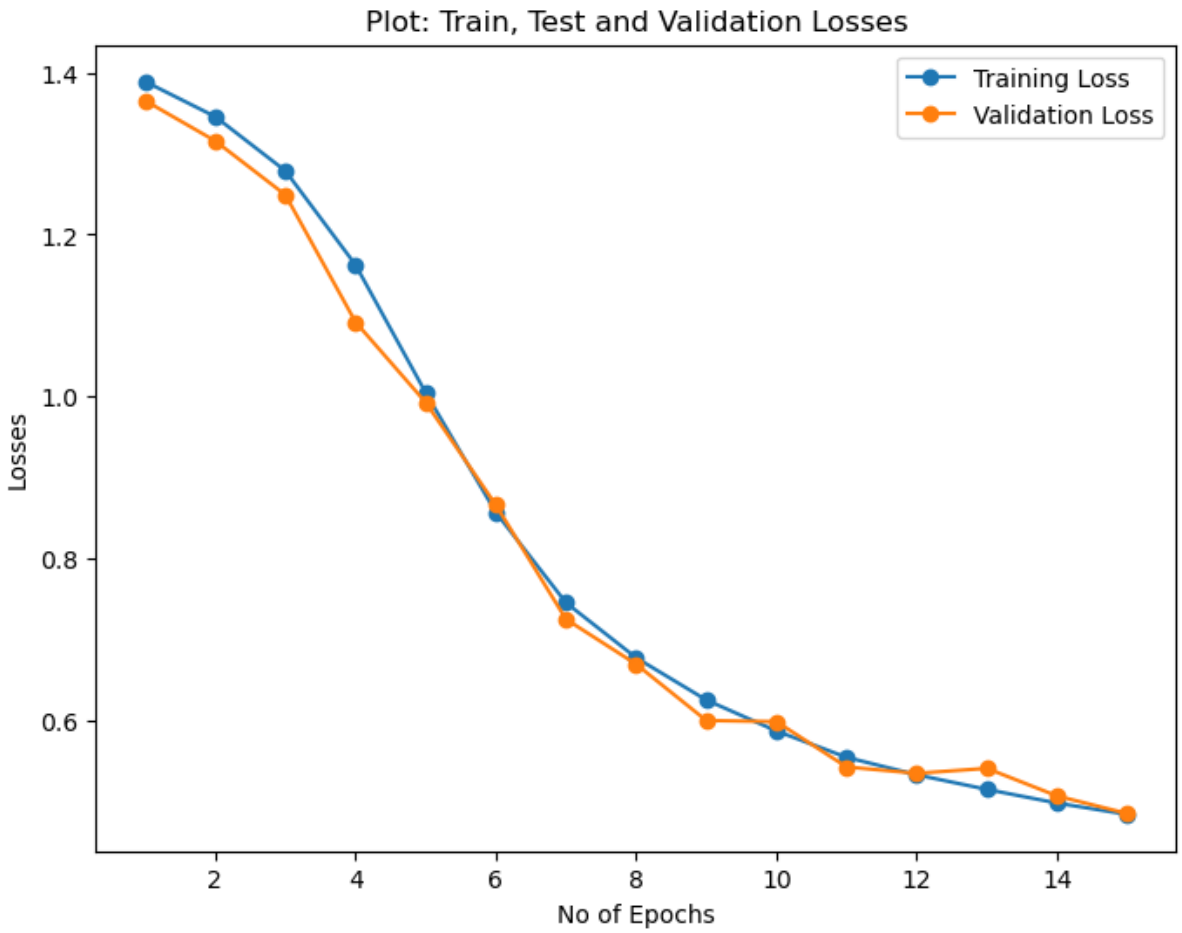
[118 27 443 3579]]

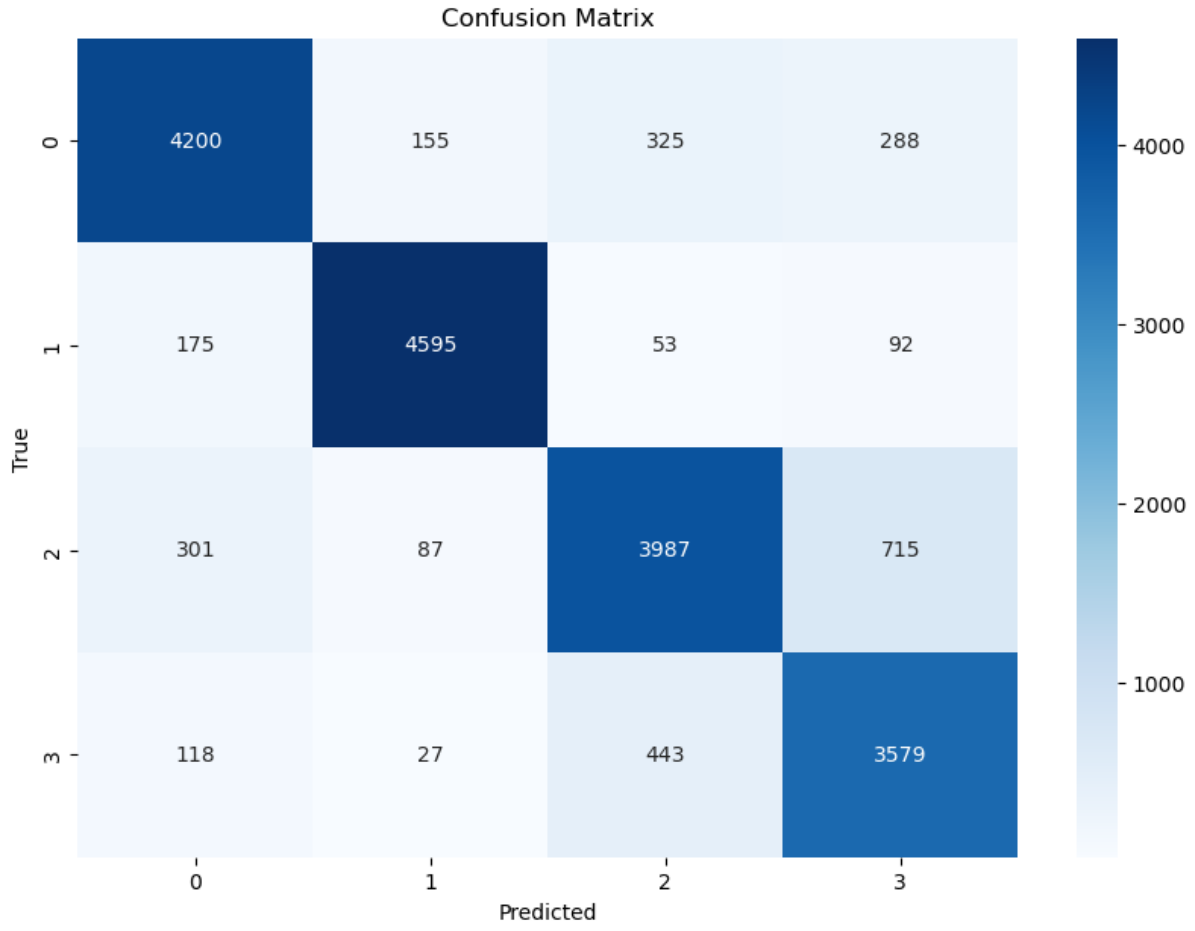
Precision: 85.3939764966812

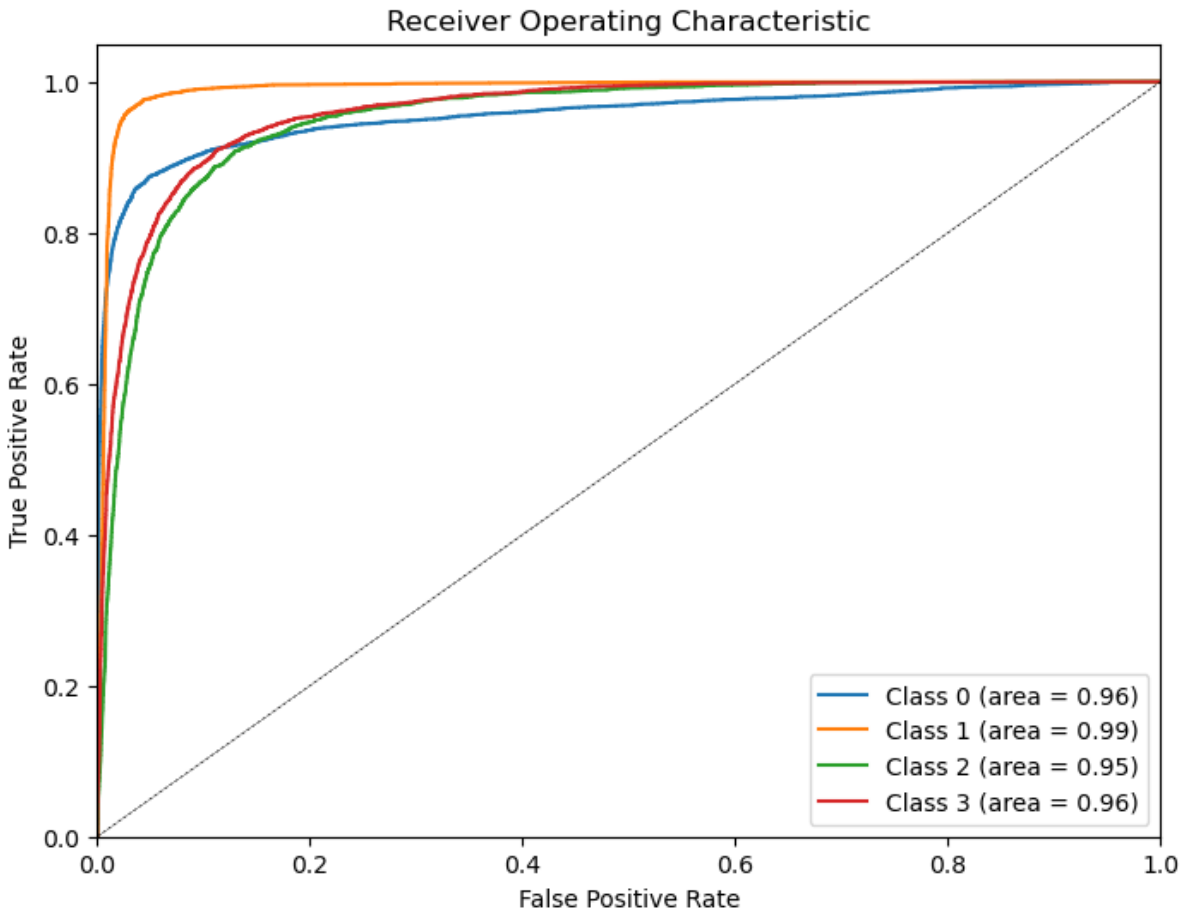
Recall: 85.56239225593512

F1 Score: 85.38756294514677


Accuracies for Train, Test and Validation
Training Accuracy
Validation Accuracy
Accuracies
No of Epochs
30
40
50
60
70
80
2
4
6
8
10
12
14


Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Losses
No of Epochs
1.4
1.2
1.0
0.8
0.6
2
4
6
8
10
12
14


Confusion Matrix
4200
155
325
288
175
4595
53
92
301
87
3987
715
118
27
443
3579
True
Predicted
0
1
2
3
4000
3000
2000
1000



Dropout:

Test Loss: 0.4323, Test Accuracy: 0.8548

dropout model results for Transformer Model

Accuracy: 85.48066875653083

Confusion Matrix: [[4200 155 325 288]

[175 4595 53 92]

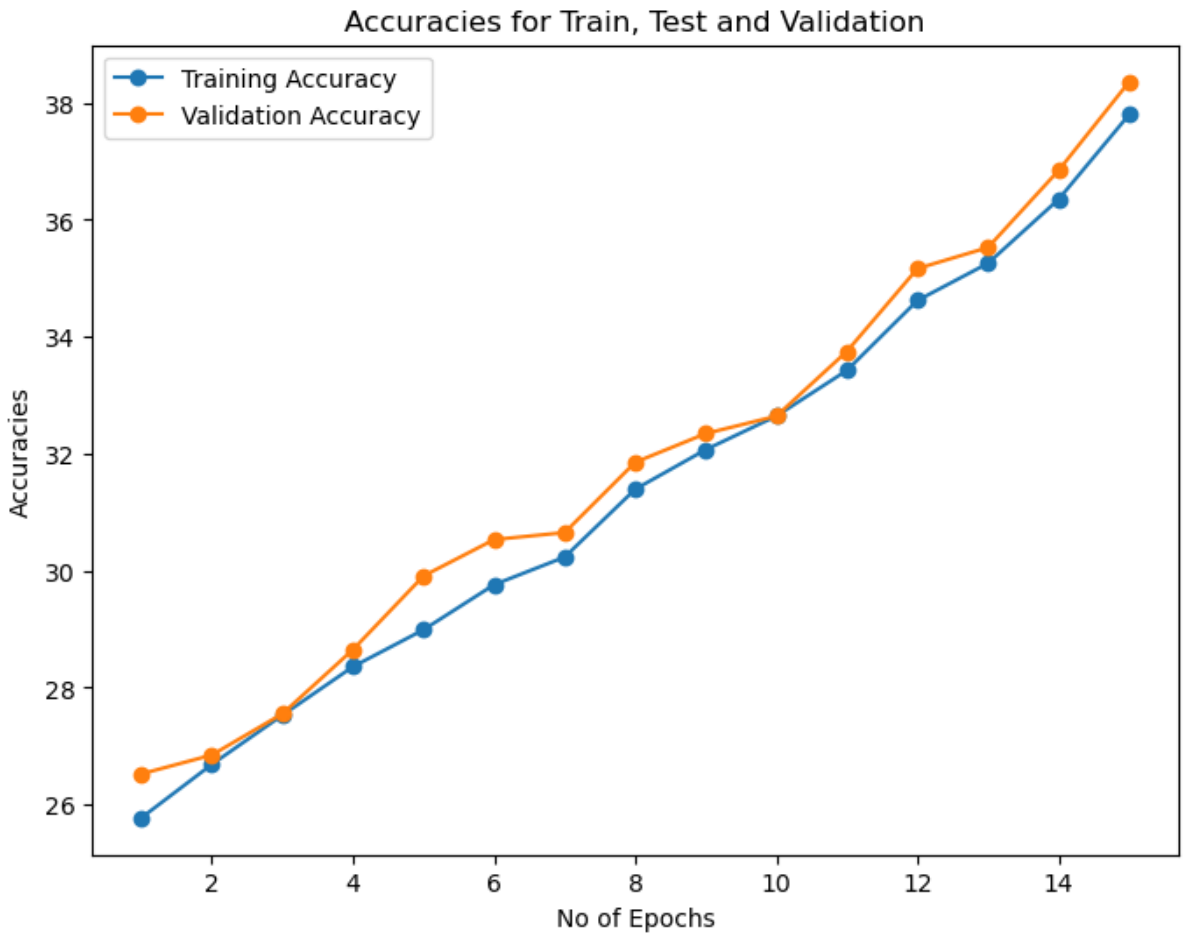
[301 87 3987 715]

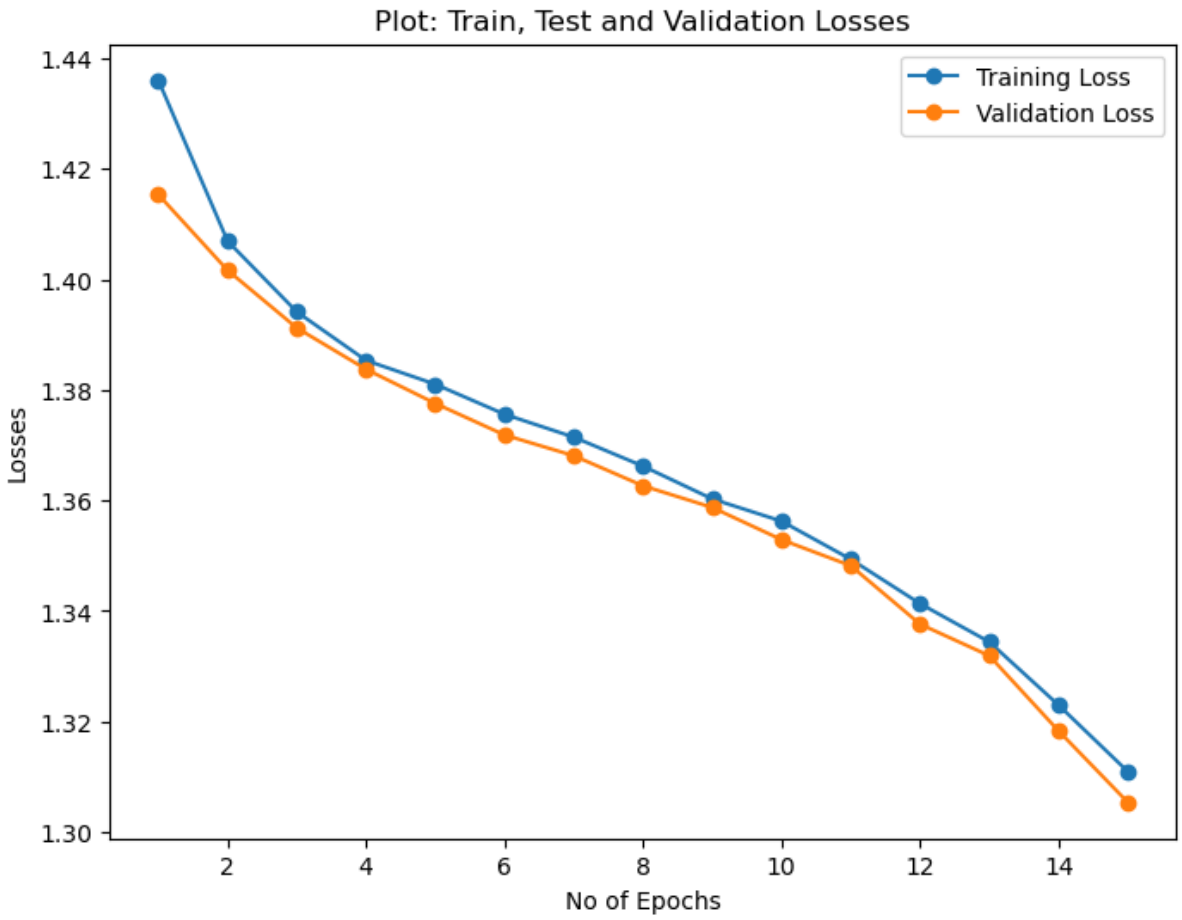
[118 27 443 3579]]

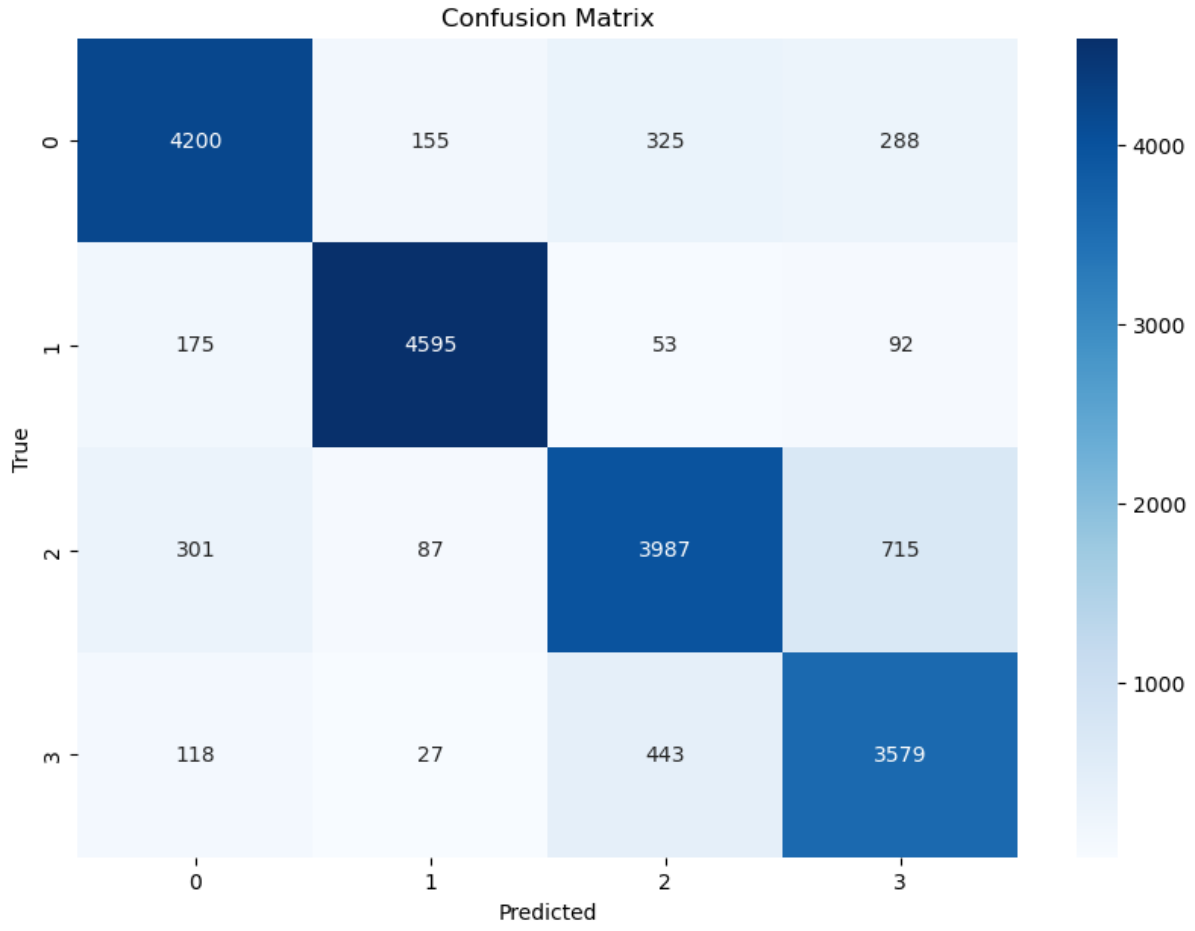
Precision: 85.3939764966812

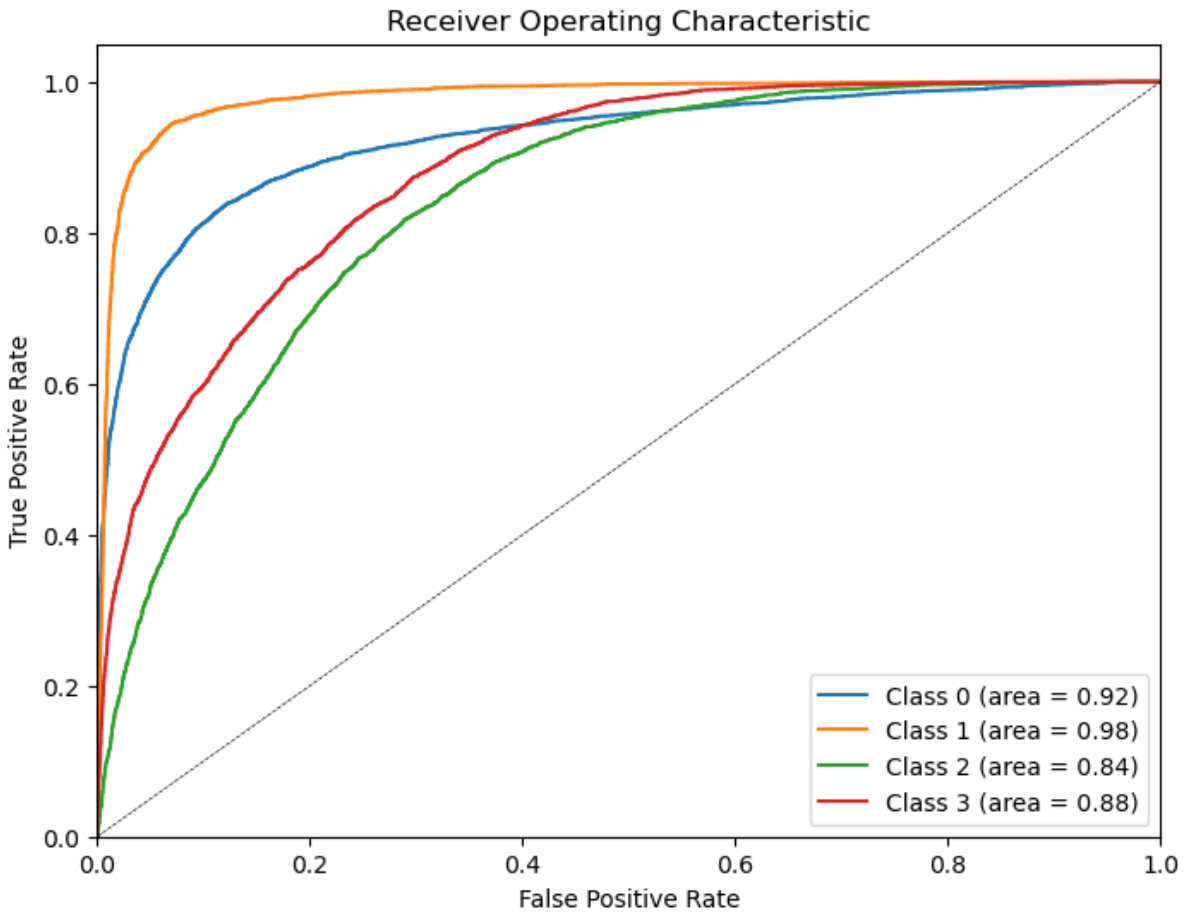
Recall: 85.56239225593512

F1 Score: 85.38756294514677


Accuracies for Train, Test and Validation
Training Accuracy
Validation Accuracy
Accuracies
No of Epochs
26
28
30
32
34
36
38
2
4
6
8
10
12
14


Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Losses
No of Epochs
1.44
1.42
1.40
1.38
1.36
1.34
1.32
1.30
2
4
6
8
10
12
14


Confusion Matrix
4200
155
325
288
175
4595
53
92
301
87
3987
715
118
27
443
3579
True
Predicted
0
1
2
3
4000
3000
2000
1000



Early Stopping:

Test Loss: 0.4323, Test Accuracy: 0.8548

Early stopping model results for Transformer Model

Accuracy: 85.48066875653083

Confusion Matrix: [[4200 155 325 288]

[175 4595 53 92]

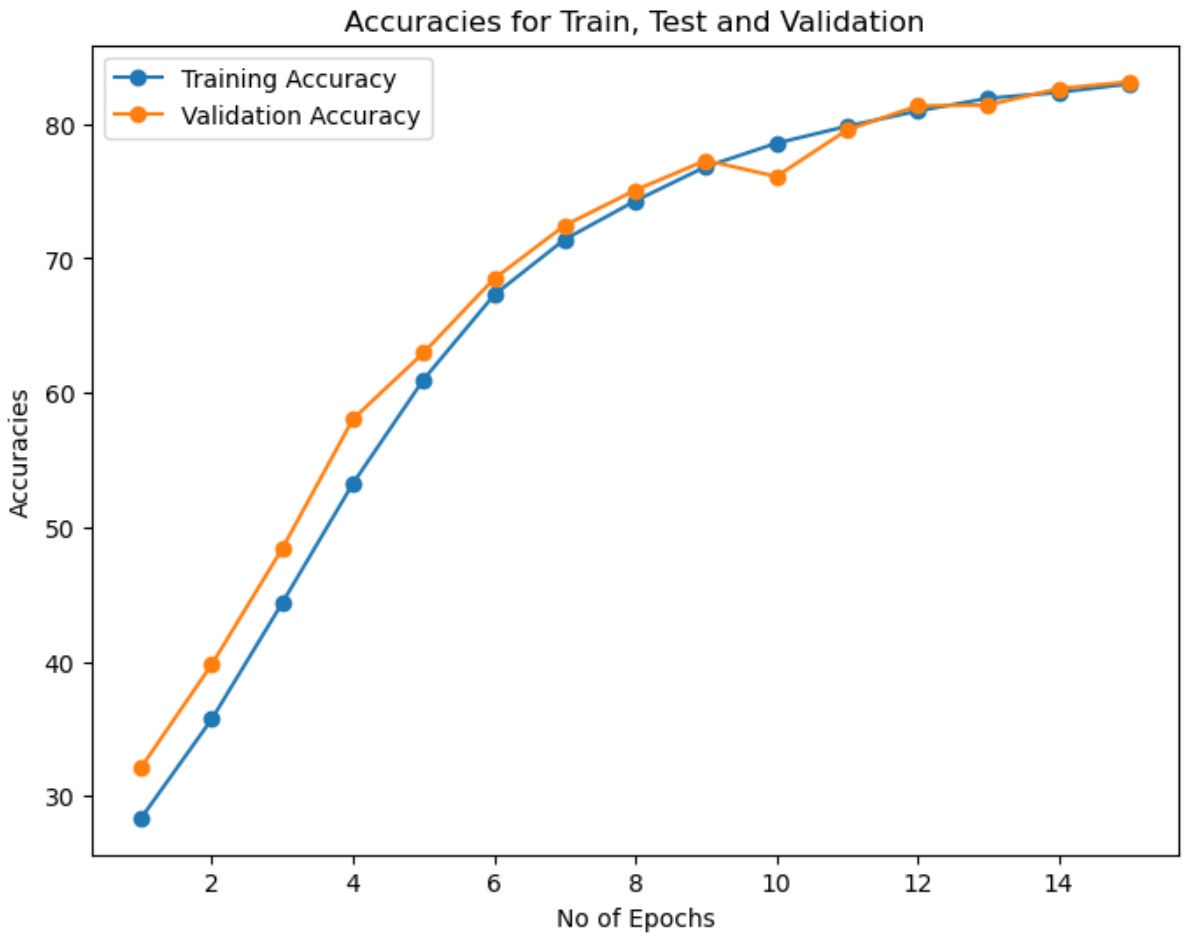
[301 87 3987 715]

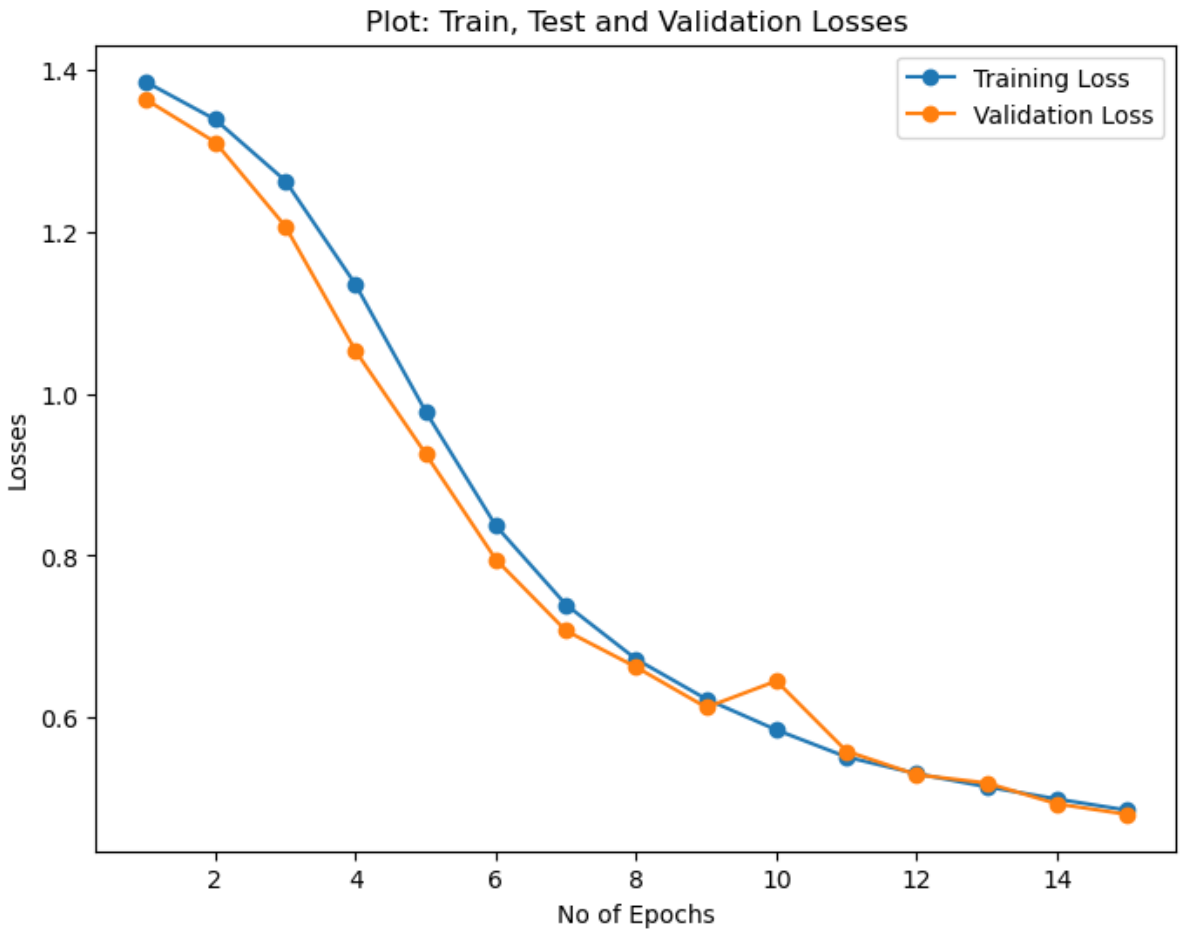
[118 27 443 3579]]

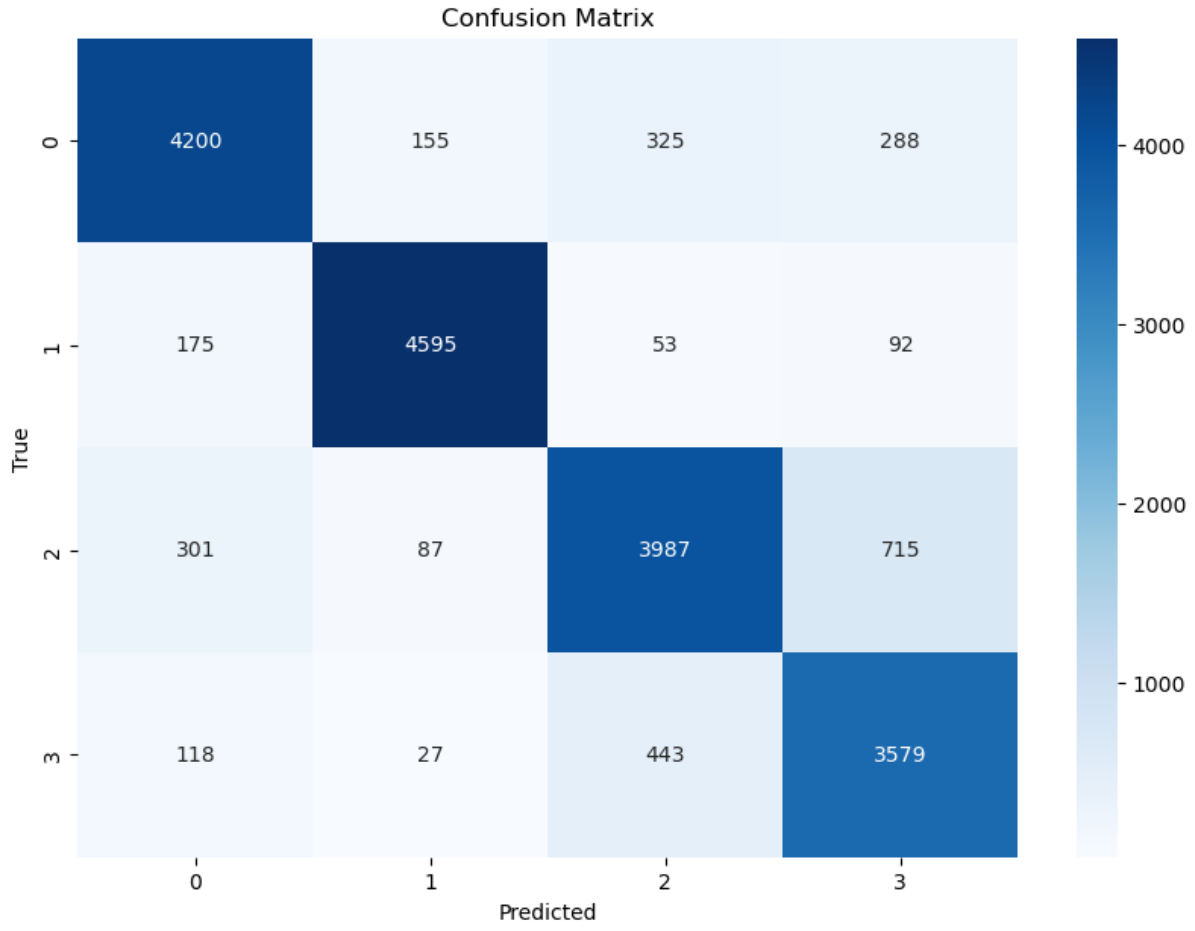
Precision: 85.3939764966812

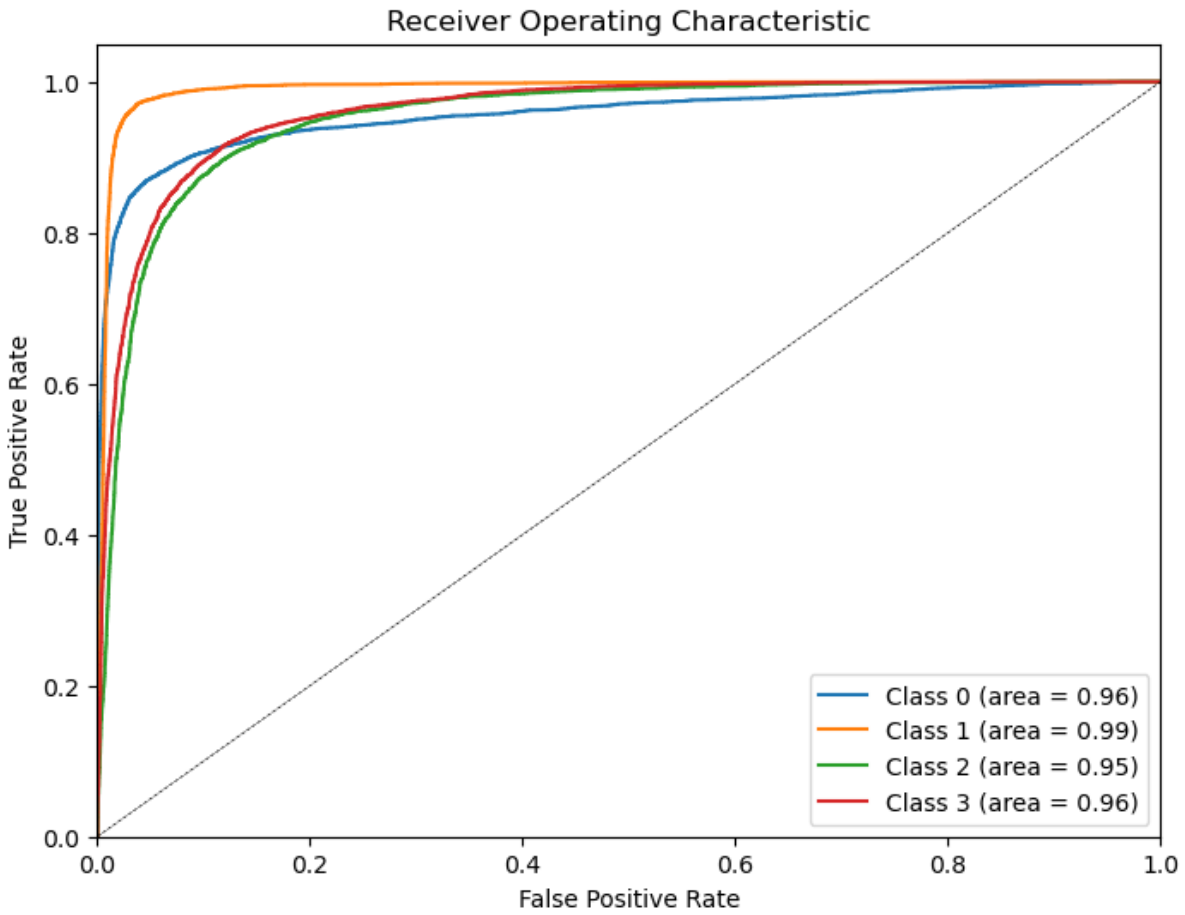
Recall: 85.56239225593512

F1 Score: 85.38756294514677


Accuracies for Train, Test and Validation
Training Accuracy
Validation Accuracy
Accuracies
No of Epochs
30
40
50
60
70
80
2
4
6
8
10
12
14


Plot: Train, Test and Validation Losses
Training Loss
Validation Loss
Losses
No of Epochs
1.4
1.2
1.0
0.8
0.6
2
4
6
8
10
12
14


Confusion Matrix
4200
155
325
288
175
4595
53
92
301
87
3987
715
118
27
443
3579
True
Predicted
0
1
2
3
4000
3000
2000
1000



Observation: Basemodel and other techniques have almost same results. Overall basemodel has accuracy 1% higher compared to the techniques applied.

Bonus:

VGG:

Train accuracy: 89.90%
Train loss: 0.26
Validation accuracy: 90.71%
Validation loss: 0.25
Test accuracy: 90%
Test Loss: 0.25

Precision: 0.91
Recall: 0.91
F1 Score: 0.91

ResNet:

Train accuracy: 97.14%
Train loss: 0. 07
Validation accuracy: 87.8%

Validation loss: 0.44
Test accuracy: 88%
Test Loss: 0.42

Precision: 0.89
Recall: 0.88
F1 Score: 0.88

Vision Transformer (ViT) for Image Classification

Train Accuracy: 33.31%
Train Loss: 1.099
Validation Accuracy: 33.29%
Validation Loss: 1.098
Test Accuracy: 33 %
Test Loss: 1.098

Precision: 0.11
Recall: 0.33
F1 Score: 0.17

Comparing this with VGG and ResNet, VGG and ResNet show robust performance whereas the performance of the Vision Transformer is very poor.

EfficientNet

Train Accuracy: 98.90%
Train Loss: 0.032
Validation Accuracy: 97.40%
Validation Loss: 0.093
Test Accuracy: 97 %
Test Loss: 0.085

Precision: 0.97
Recall: 0.97
F1 Score: 0.97

Comparing results of EfficientNet with ResNet and VGG, we can see that EfficientNet shows superior performance across all metrics compared to VGG and ResNet. It not only achieves higher accuracy (both in training and testing phases) but also maintains lower loss scores, which indicates more effective and efficient learning and generalization capabilities.

References:

- https://arxiv.org/pdf/2003.05991.pdf
- https://arxiv.org/abs/1606.05908
- https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/Tokenizer
- https://pytorch.org/text/stable/data_utils.html
- https://arxiv.org/abs/2010.11929
- https://arxiv.org/pdf/1905.11946.pd

- kushakum_assignment1
- cnara_assignment1