Assignment 2
Autoencoder and Transformer Architectures

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

TASKS:

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

Total Parameters=(input_size * output_size) + output_size

Layer-wise Calculation:

Input Layer to Hidden Layer 1:

Weights: 1000×512=512,000

Biases: 512

Total: 512,000+512=512,512

Hidden Layer 1 to Bottleneck Layer:

Weights: 512×32=16,384

Biases: 32

Total: 16,384+32=16,416

Bottleneck Layer to Hidden Layer 2:

Weights: 32×512=16,384

Biases: 512

Total: 16,384+512=16,896

Hidden Layer 2 to Output Layer:

Weights: 512×1000=512,000

Biases: 1000

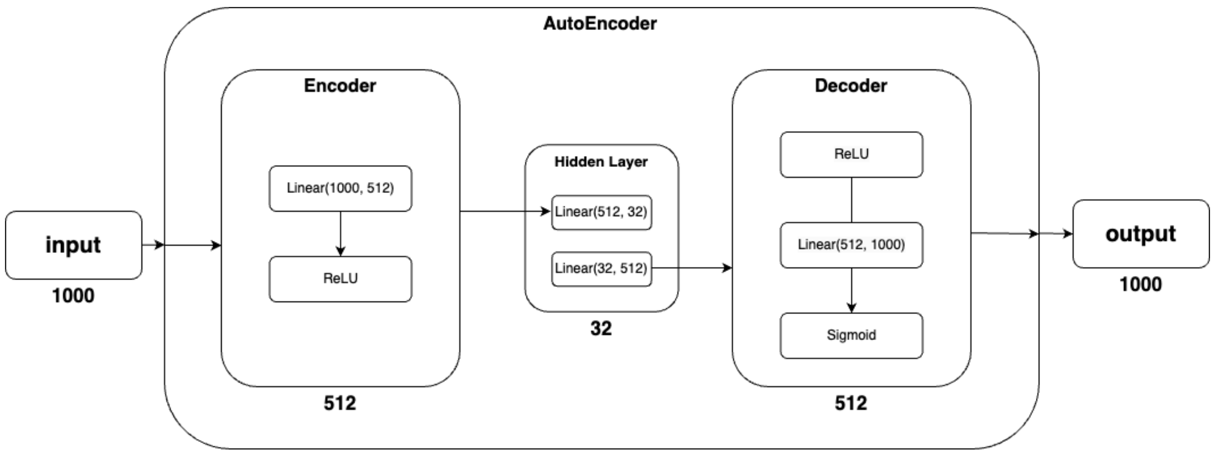
Total: 512,000+1000=513,000

Total Parameters in Autoencoder:

512,512+16,416+16,896+513,000=1,058,824

Total Parameters in Autoencoder: 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.

1. If the anomalies are not well represented in the training data, autoencoders will not be effective.
2. Lot of trial and error will be required for finding the optimal model for anomaly detection. This may be time consuming and computationally expensive.

3. The bottleneck layer may lead to the loss of features that are necessary for the identification of anomalies.
4. Maintaining a balance between generalization and overfitting becomes a challenging task with autoencoders. Overfitting may lead to poor performance of the model.

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

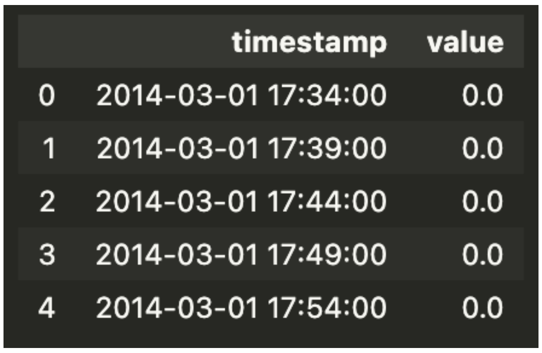
1. Using hybrid models can be effective to improve the models performance. Combining autoencoders with other machine learning models specifically trained for anomaly detection can boost the accuracy.
2. Training the model to be domain specific can increase models reliability and ability to recognize anomalies. This can be achieved by feature engineering.
3. Implementing hyperparameter tuning techniques such as Regularization, Dropout, Early Stopping can improve the models effectiveness for anomaly detection.
4. Introducing samples with anomalies in the training will increase the models anomaly detection capability.

Part II: Autoencoders for Anomaly Detection

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?

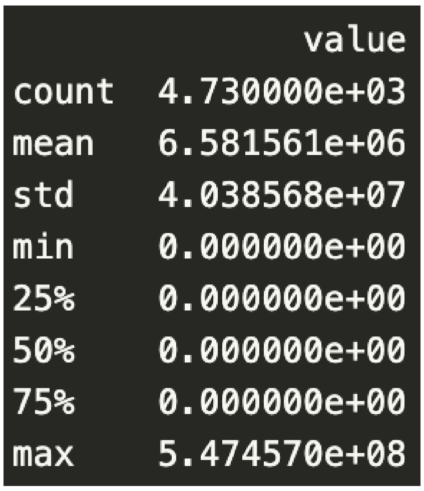
The "AWS_ec2_disk_write_bytes.csv" is a time-series dataset with numerical values representing disk write activity at different time intervals in an AWS EC2. It captures the fluctuation in disk write activity, which could be influenced by various factors such as application usage, system processes, and workload demands.

Dataset Head



	timestamp	value
0	2014-03-01 17:34:00	0.0
1	2014-03-01 17:39:00	0.0
2	2014-03-01 17:44:00	0.0
3	2014-03-01 17:49:00	0.0
4	2014-03-01 17:54:00	0.0

Dataset Statistics Summary


```
              value
count  4.730000e+03
mean   6.581561e+06
std    4.038568e+07
min    0.000000e+00
25%    0.000000e+00
50%    0.000000e+00
75%    0.000000e+00
max    5.474570e+08
```

Reason for choice of the above dataset:

Relevance to Cloud Computing: The dataset reflects real-world scenarios in AWS EC2 instances, making it effective for studying system behaviors and anomalies in cloud environments.

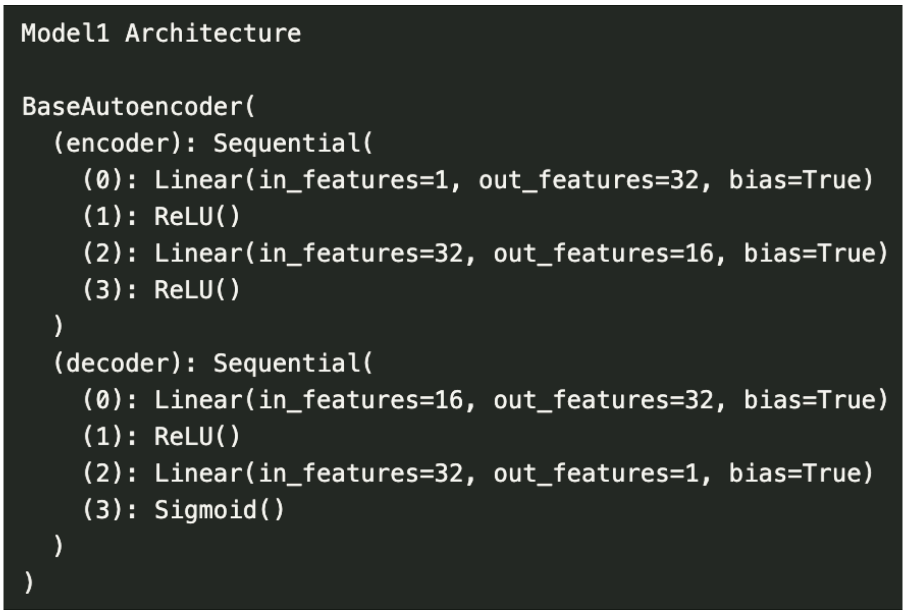
Practical Application: Analyzing disk write activity is crucial for optimizing resource usage and identifying potential performance issues in cloud systems.

Research Opportunities: The dataset enables exploring novel anomaly detection techniques using autoencoders, and many other Machine Learning models.

Number of entries (rows): 4730
Number of variables (columns): 2

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

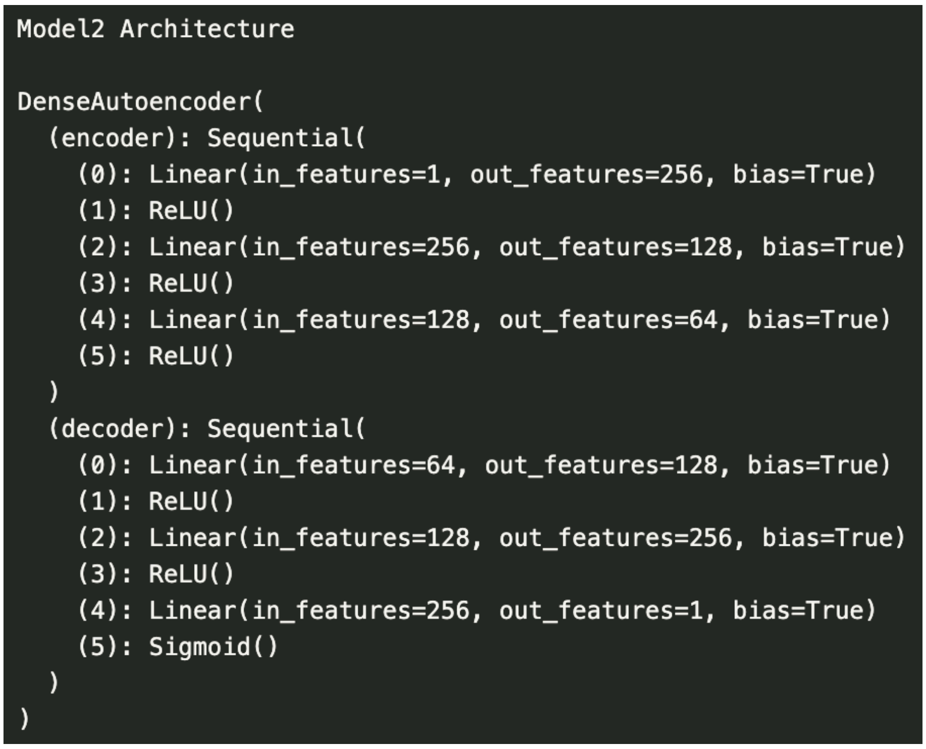
Model 1: Base Autoencoder



```
Model1 Architecture

BaseAutoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=1, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=16, bias=True)
    (3): ReLU()
  )
  (decoder): Sequential(
    (0): Linear(in_features=16, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=1, bias=True)
    (3): Sigmoid()
  )
)
```

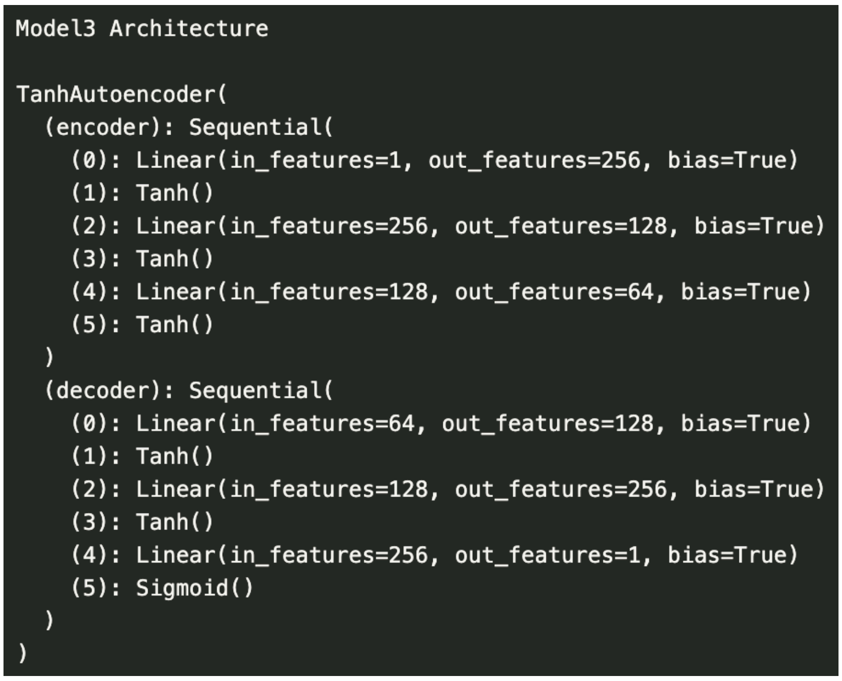
Model 2: Base Autoencoder and Dense Layers



```
Model2 Architecture

DenseAutoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=1, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=64, bias=True)
    (5): ReLU()
  )
  (decoder): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

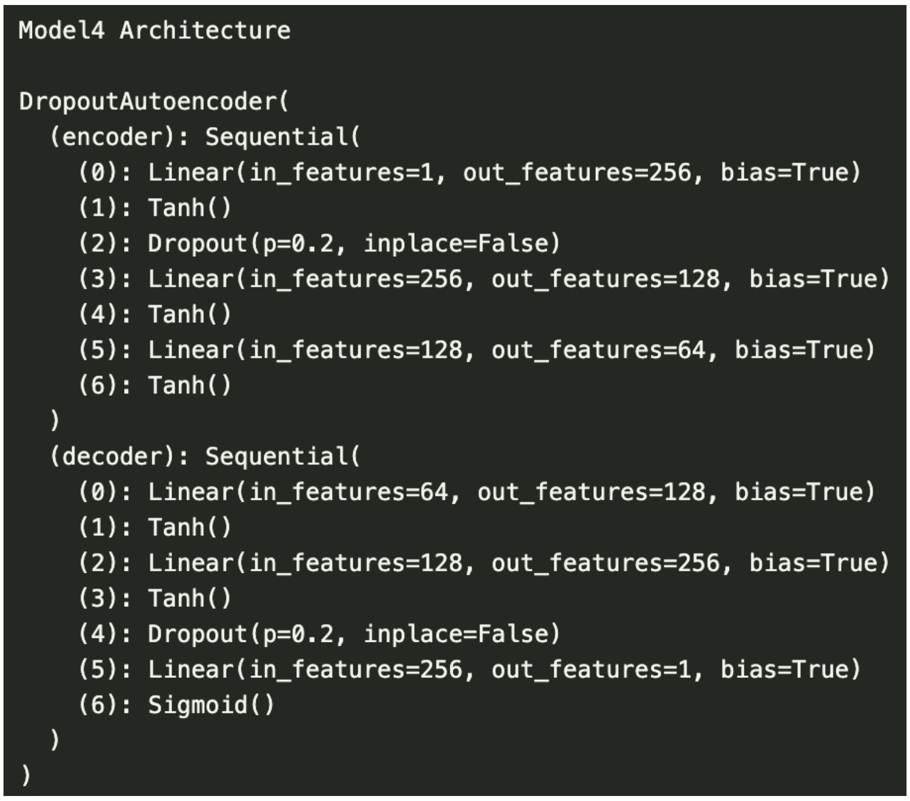
Model 3: Base Autoencoder, Dense Layers, and tanh Activation Function



```
Model3 Architecture

TanhAutoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=1, out_features=256, bias=True)
    (1): Tanh()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): Tanh()
    (4): Linear(in_features=128, out_features=64, bias=True)
    (5): Tanh()
  )
  (decoder): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): Tanh()
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): Tanh()
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

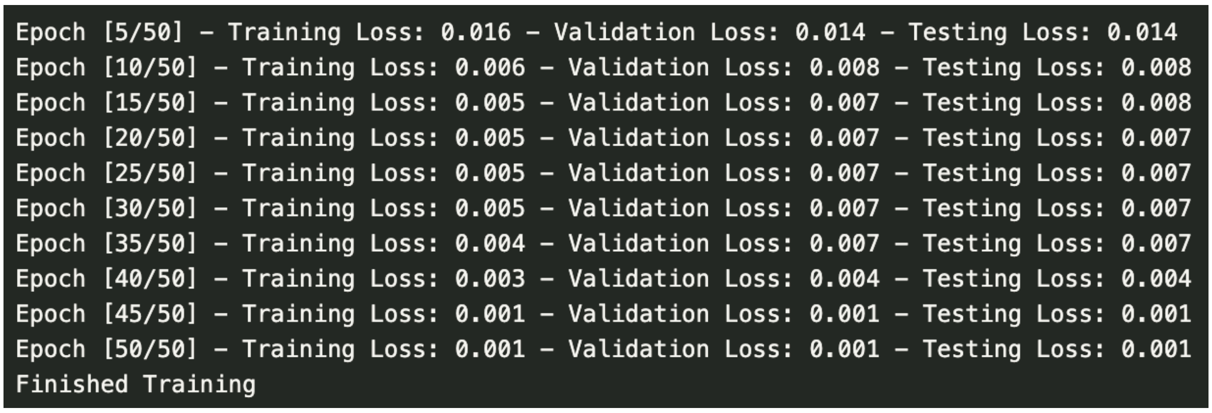
Model 4: Base Autoencoder, Dense Layers, tanh Activation Function, and Dropout



```
Model4 Architecture

DropoutAutoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=1, out_features=256, bias=True)
    (1): Tanh()
    (2): Dropout(p=0.2, inplace=False)
    (3): Linear(in_features=256, out_features=128, bias=True)
    (4): Tanh()
    (5): Linear(in_features=128, out_features=64, bias=True)
    (6): Tanh()
  )
  (decoder): Sequential(
    (0): Linear(in_features=64, out_features=128, bias=True)
    (1): Tanh()
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): Tanh()
    (4): Dropout(p=0.2, inplace=False)
    (5): Linear(in_features=256, out_features=1, bias=True)
    (6): Sigmoid()
  )
)
```

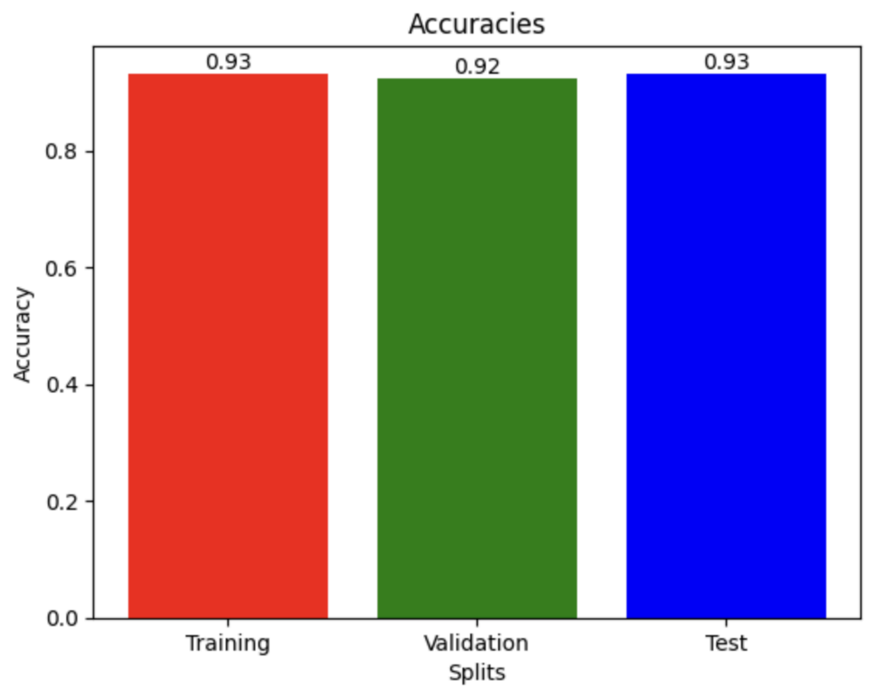
3. Discuss the results and provide relevant graphs:
a. Report training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss.



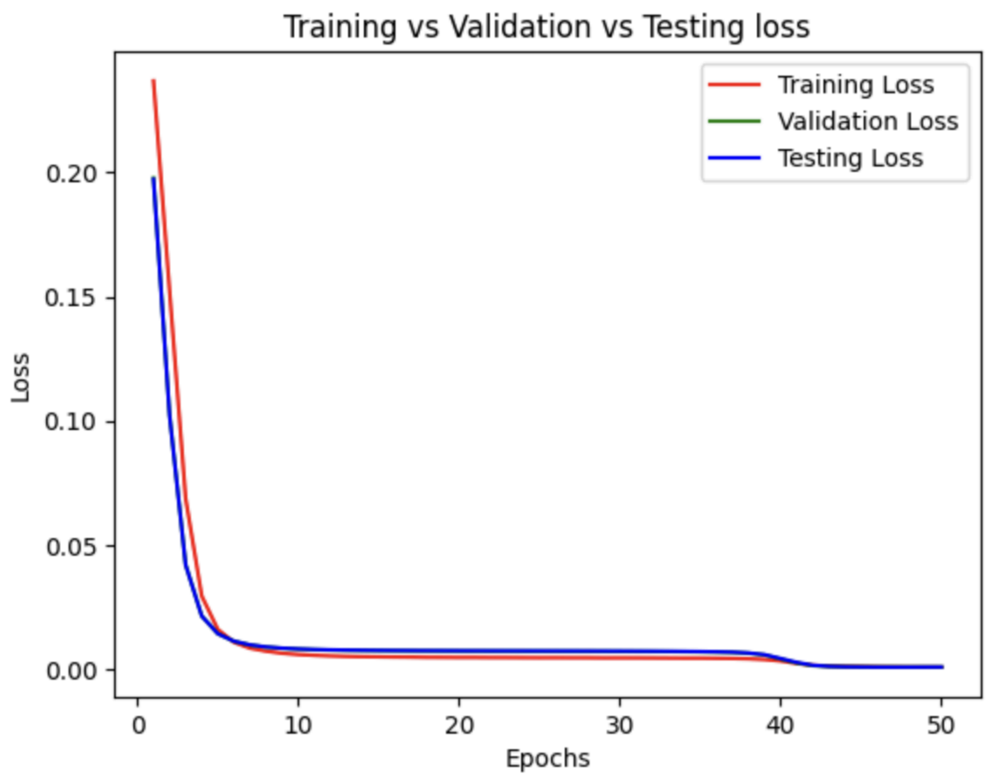
```
Epoch [5/50] - Training Loss: 0.016 - Validation Loss: 0.014 - Testing Loss: 0.014
Epoch [10/50] - Training Loss: 0.006 - Validation Loss: 0.008 - Testing Loss: 0.008
Epoch [15/50] - Training Loss: 0.005 - Validation Loss: 0.007 - Testing Loss: 0.008
Epoch [20/50] - Training Loss: 0.005 - Validation Loss: 0.007 - Testing Loss: 0.007
Epoch [25/50] - Training Loss: 0.005 - Validation Loss: 0.007 - Testing Loss: 0.007
Epoch [30/50] - Training Loss: 0.005 - Validation Loss: 0.007 - Testing Loss: 0.007
Epoch [35/50] - Training Loss: 0.004 - Validation Loss: 0.007 - Testing Loss: 0.007
Epoch [40/50] - Training Loss: 0.003 - Validation Loss: 0.004 - Testing Loss: 0.004
Epoch [45/50] - Training Loss: 0.001 - Validation Loss: 0.001 - Testing Loss: 0.001
Epoch [50/50] - Training Loss: 0.001 - Validation Loss: 0.001 - Testing Loss: 0.001
Finished Training
```

Training Accuracy: 0.93
Validation Accuracy: 0.92
Testing Accuracy: 0.93

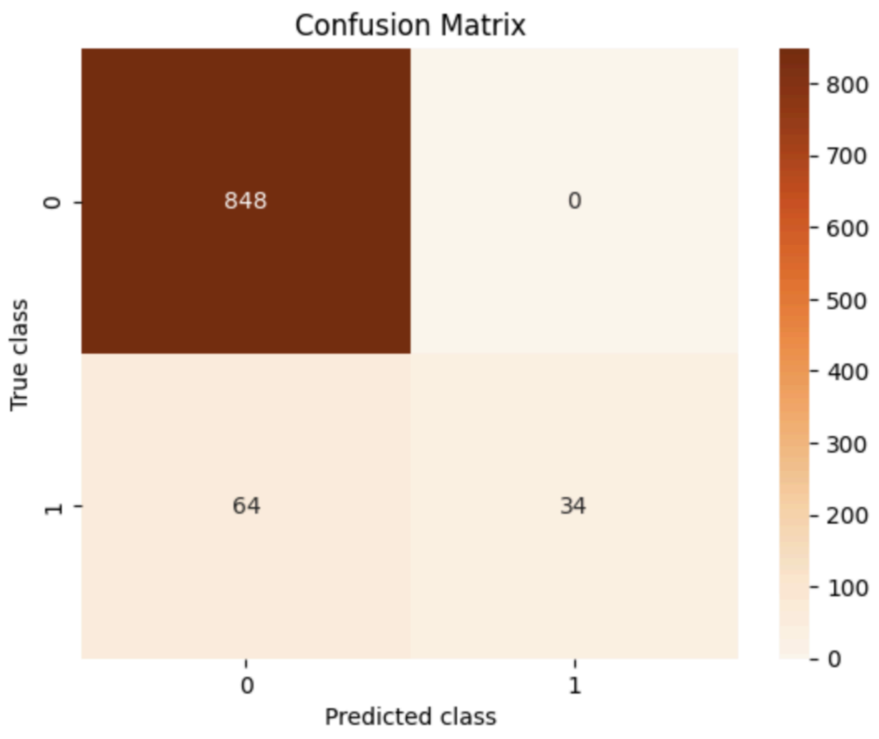
b. Plot the training and validation accuracy over time (epochs).



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

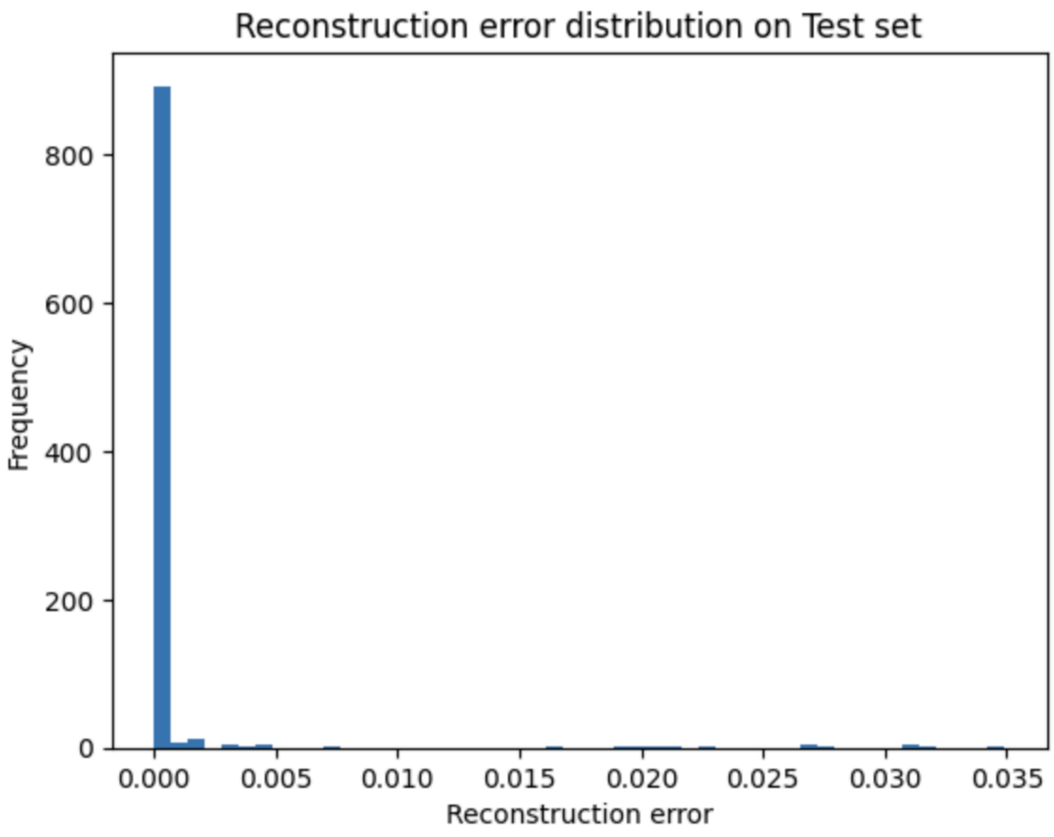


d. Generate a confusion matrix using the model's predictions on the test set.

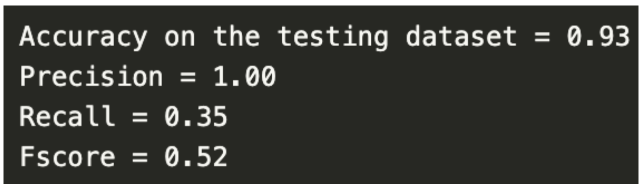


e. Report any other evaluation metrics used to analyze the model's performance on the test set.

Reconstruction Error Plot:

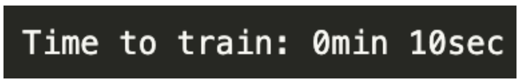


Accuracy, Precision, Recall, F-score:



```
Accuracy on the testing dataset = 0.93
Precision = 1.00
Recall = 0.35
Fscore = 0.52
```

Training Time:



```
Time to train: 0min 10sec
```

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

Strengths:

- Ability to capture complex patterns in the data without requiring feature engineering.
- They can reconstruct normal patterns effectively pointing out anomalies as significant differences in reconstruction errors.
- Autoencoders can handle high-dimensional data efficiently.

They can learn nonlinear relationships in the data.

Limitations:

- Anomalies that are poorly represented in the training set are challenging to find.
- A large amount of data is needed for autoencoder training.
- Because autoencoders are prone to noise in the data, anomaly detection may result in false positives or negatives.

Part III: Theoretical Part – Transformers

TASKS:
1. Break down the mathematical operations involved in self-attention. Explain how the input sequence x = (x1, x2, . . . , 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).

The first step involves calculating Embedding for the given Dataset.
The next step is linear transformation that can be calculated using the following steps:

Define weight matrices W_q, W_k, W_v for query key and value respectively.
Calculate Query(Q), Key(K), Value(V):
Query sequence: $Q_i = W_q x_i + b_q$
Key sequence: $K_i = W_k x_i + b_k$
Value sequence: $V_i = W_v x_i + b_v$
,for i belongs to (1, T)

This is another way of representing the above formula:
$Q = W_q x^T + b_q$
$K = W_k x^T + b_k$
$V = W_v x^T + b_v$

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

Scaled dot-product attention helps us to calculate the attention scores. Softmax activation function is applied to normalize the scores.

Calculate attention using the following formula:
$score(Q,\ K)\ =\ Softmax(\frac{QK^T}{\sqrt{d_k}})$
Where d_k is the dimensionality of the key vectors

Similarly, calculating attention scores using the query (q_i) and key (k_j) vectors

$score(q_i,\ k_j)\ =\ Softmax(\frac{q_i \cdot k_j}{\sqrt{d_k}})$
q_i is the query vector associated with ith element in sequence.
k_j is the key vector associated with the jth element in sequence
d_k is the dimensionality of the key vectors

The square root of the key vector dimension (d_k) is used in the normalization process to scale the dot product of query and key vectors. It helps to prevent the issue of large dot products when the dimensionality of the vectors increases.

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

  After the attention scores are calculated, we use them to calculate the weights of the value vectors.
  Calculate weighted sum context vector by the given formula:
  $$Weighted\,sum\,(c) \;=\; \sum_{i=1}^{n} score(Q,\,K) \;\times\; V_i$$
  Where score is the scaled dot-product attention which we calculated in the above steps.

- Optional output transformation: Explain the purpose of the optional final linear transformation ($W\,o$) and bias term (bo) in generating the latent representation z. How does this step potentially impact the final representation?

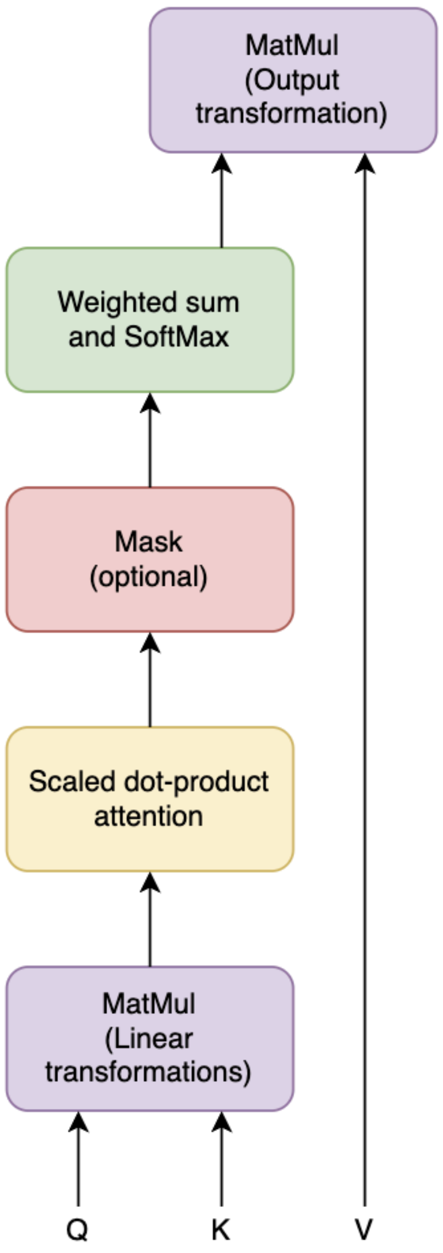
  The main reason for output transformation is to introduce non linearity, adjust dimensionality and learn bias.

  The latent representation(z) can be calculated as follows:
  $$z \;=\; ReLU(c\,W_o \;+\; b_o)$$
  c is the weighted sum calculated in the previous step.
  W_o is weight matrix
  b_o is the bias term

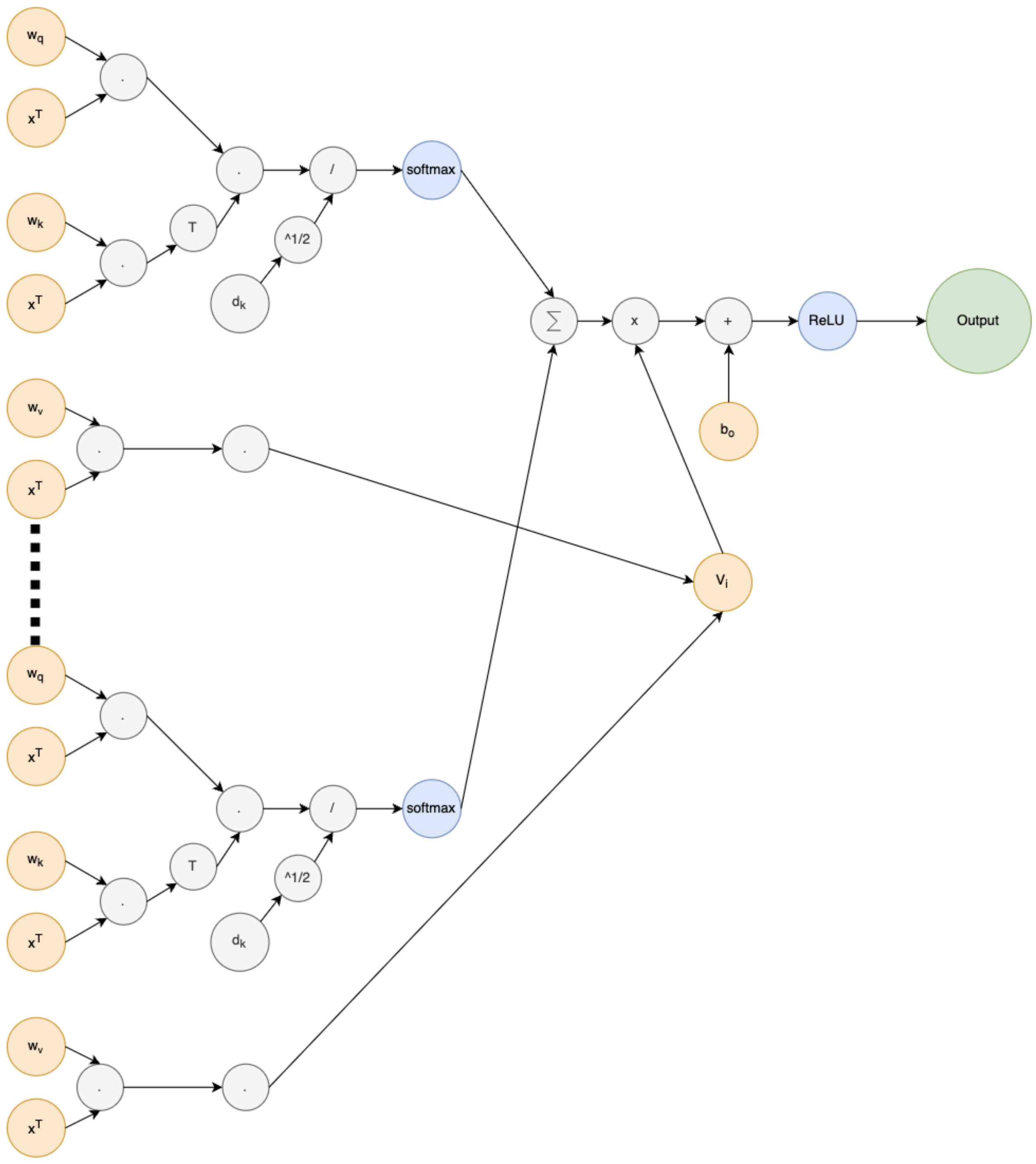
  ReLU activation function is used to add non linearity in the model. It allows the model to learn complex relationships.
  This transformation allows the model to adjust dimensionality of the output and makes it more flexible and adaptable.
  By providing the model weights and bias, it helps to learn and adapt features that are specific to the given dataset.
  We can capture complex patterns and relationships in the data, improving the models efficiency.

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

Flow Diagram:


MatMul (Output transformation)
Weighted sum and SoftMax
Mask (optional)
Scaled dot-product attention
MatMul (Linear transformations)
Q
K
V

Computational Diagram:



References:

1. https://pandas.pydata.org/docs/
2. https://numpy.org/doc/
3. https://matplotlib.org/stable/index.html
4. https://scikit-learn.org/stable/
5. https://seaborn.pydata.org/
6. https://pytorch.org/tutorials/
7. https://cybersecurity.springeropen.com/articles/10.1186/s42400-022-00134-9
8. https://www.kaggle.com/datasets/boltzmannbrain/nab
9. Part 3, Steps 4 and 6 is based on CSE 574 Machine Learning Assignment 2 Part 3 submission by Nikhil Gupta