Lab Assignment 4

Aim:- Use autoencoder to implement anomaly detection. Build the model by using:-
a] Import required libraries
b] Upload dataset
c] Encode convert it into latent representation
d] Decoder networks convert back to original input.
e] Compile the model with Optimizer, Loss.

Objective:- To learn about autoencoders & developing autoencoder for anomaly detection.

Infrastructure:- Computer / Laptop

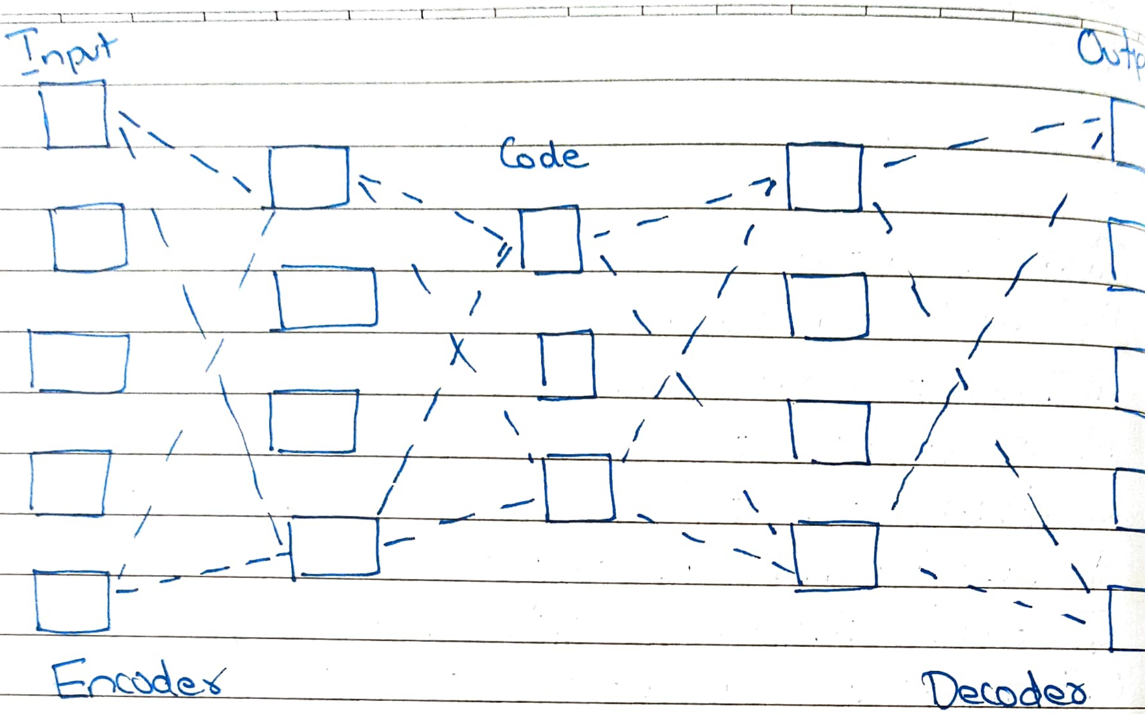
Software used:- Jupyter Notebook

Theory:-

Autoencoder

Autoencoder are ANN capable of learning efficient representation of the input data, called coding's, without any supervision. These coding's have typically a much lower dimensionality than the input data making autoencoder useful for dimensionality reduction & compression. These codings, the code is a compact "summary" or "compression" of the input.

· Autoencoder act as powerful feature detectors & can be used for unsupervised pre-training of deep neural networks. Similarly, they are capable of randomly generating new data that looks very similar to the training data. For example, you can train an autoencoder on picture of faces & it would then be able to generate new faces



An autoencoder has 3 components :-

1] Encoder :- It compresses the input into a latent s representation. It compresses the input in a reduced dimension

2] Code :- The compressed input which is fed to decode reconstructing the original input later.

3] Decoder :- It decodes the encoded ~~input~~ output in fo of code, back to the original input.

The layer between encoder & decoder i.e code is known as Bottleneck. This is a well-defined approach to decide which aspects of data should relevant information.

Parameters used for training in autoencoder are :-

1] Code Size :- It is number of nodes in coding layer. Smaller size results in more compression of results

2] Number of layers :- Autoencoder can be as deep as you like, you need to decide how much layers autoencoders could have

3] Number of nodes per layer :- No of nodes per layer decreases with each subsequent layer of encoder & increases with each layer in decoder

4] Loss Function :- We either use MSE (Mean Squared Error) & Binary cross entropy as loss function.

Implementation :-

1] Load necessary libraries

2] Import dataset from respective library

3] Shape data as per your needs

4] Encode input data to latent representation

5] Decode the output of encoder to convert it back to original input

6] Use models with Optimizers, Loss & Evaluation Metrics

Conclusion :-

We learnt how to detect anamoly using autoencoders

Name:- Ojas Dhananjay Bhat Roll No:- 012 PRN No:- 72176150L Class:- BE[IT] Subject:- Deep Learning

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
import seaborn as sns
from tensorflow.keras.models import Model
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from keras import Sequential
sns.set()
import numpy as np
from tensorflow.keras import layers, losses
```

In [2]:
```
df = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.
raw_data = df.values
df.head()
```

Out[2]:

	0	1	2	3	4	5	6	7	8
0	-0.112522	-2.827204	-3.773897	-4.349751	-4.376041	-3.474986	-2.181408	-1.818286	-1.250522
1	-1.100878	-3.996840	-4.285843	-4.506579	-4.022377	-3.234368	-1.566126	-0.992258	-0.754680
2	-0.567088	-2.593450	-3.874230	-4.584095	-4.187449	-3.151462	-1.742940	-1.490659	-1.183580
3	0.490473	-1.914407	-3.616364	-4.318823	-4.268016	-3.881110	-2.993280	-1.671131	-1.333884
4	0.800232	-0.874252	-2.384761	-3.973292	-4.338224	-3.802422	-2.534510	-1.783423	-1.594450

5 rows × 141 columns

In [3]:
```
labels = raw_data[:, -1]

data = raw_data[:, 0:-1]
```

In [4]:
```
pd.Series(labels).value_counts()
```

Out[4]:
```
1.0    2919
0.0    2079
dtype: int64
```

In [5]:
```
train_data,test_data, train_labels,test_labels = train_test_split(
    data, labels, test_size = 0.2, random_state=21
)
```

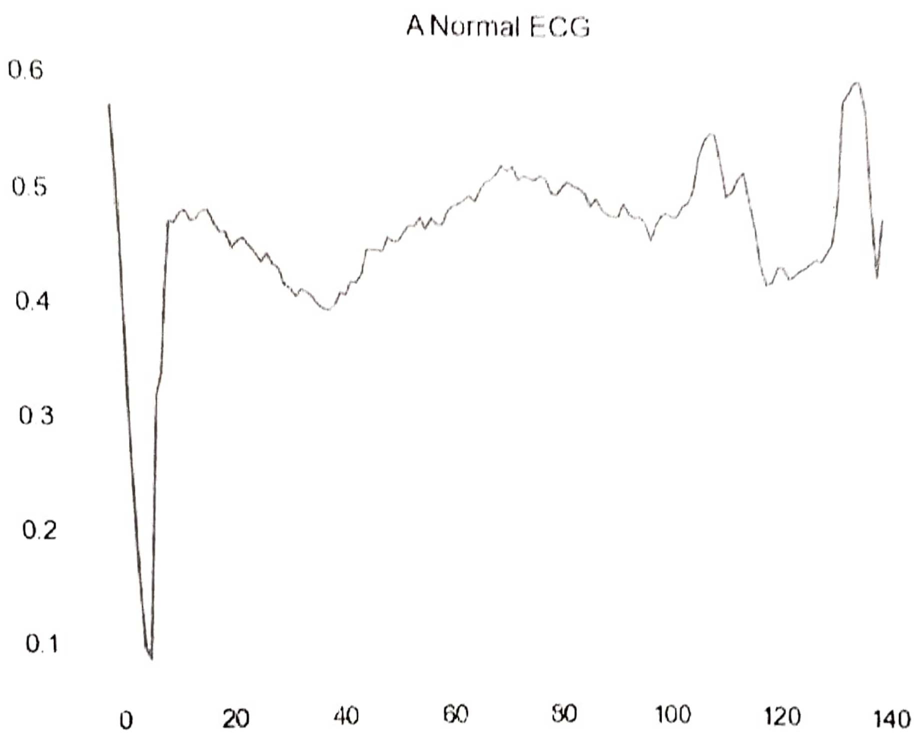
In [6]:
```
min = np.min(train_data)
max = np.max(train_data)

train_data = ( train_data - min ) / ( max - min )
test_data = ( test_data - min ) / ( max - min )
```

```
In [7]: train_labels = train_labels.astype(bool)
        test_labels = test_labels.astype(bool)


        normal_train_data = train_data[train_labels]
        normal_test_data = test_data[test_labels]

        anamalous_train_data = train_data[~train_labels]
        anamalous_test_data = test_data[~test_labels]
```

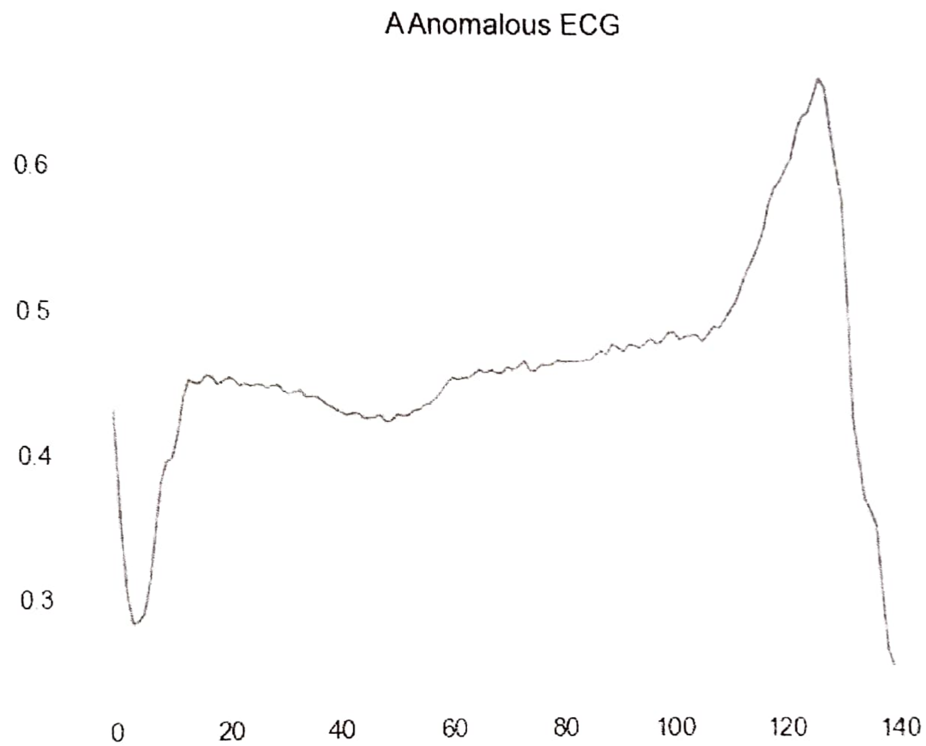
```
In [8]: ~train_labels
```

```
Out[8]: array([False, False, False, ..., False, False, False])
```

```
In [9]: plt.grid()
        plt.plot(np.arange(140),normal_train_data[0])
        plt.title('A Normal ECG')
        plt.show()
```



```
In [10]: plt.grid()
         plt.plot(np.arange(140),anamalous_train_data[0])
         plt.title('A Anomalous ECG')
         plt.show()
```



```
In [11]: class AnomalyDetector(Model):
           def __init__(self):
             super(AnomalyDetector, self).__init__()
             self.encoder = Sequential([
                                                      layers.Dense(32, activation='relu'),
                                                      layers.Dense(16, activation='relu'),
                                                      layers.Dense(8, activation='relu')
             ])
             self.decoder = tf.keras.Sequential([
                                                      layers.Dense(16, activation='relu'),
                                                      layers.Dense(32, activation='relu'),
                                                      layers.Dense(140, activation='sigmoid')
             ])

           def call(self, x):
             encoded = self.encoder(x)
             decoded = self.decoder(encoded)
             return decoded

         autoencoder = AnomalyDetector()
```

```
In [12]: autoencoder.compile(optimizer='adam', loss='mae')
```

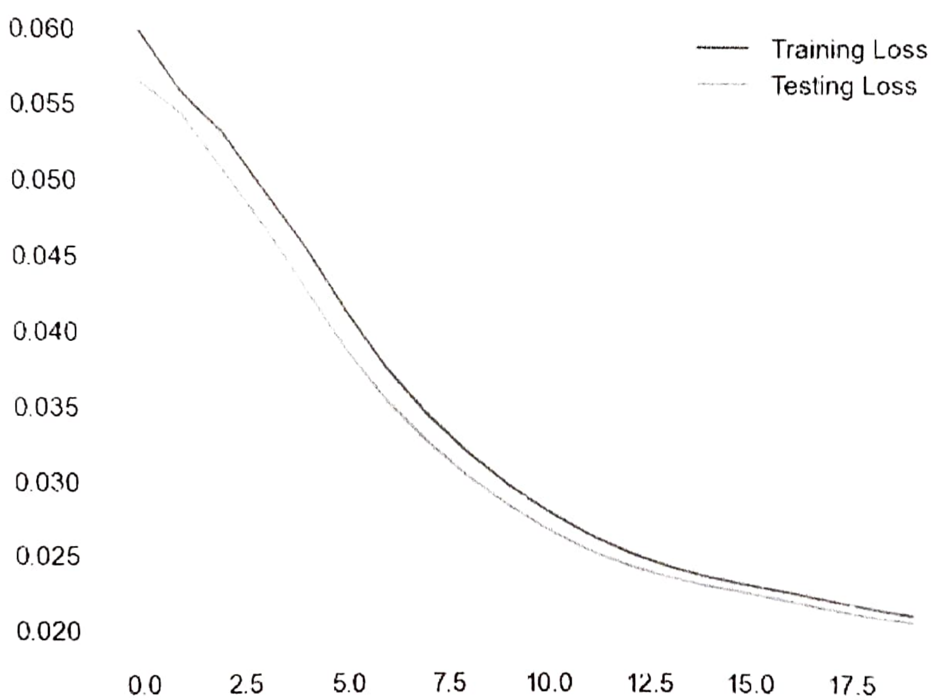
In [13]:
```
history = autoencoder.fit(normal_train_data, normal_train_data,
                          epochs = 20,
                          batch_size=512,
                          validation_data=(normal_test_data, normal_test_data),
                          shuffle=True)
```

```
Epoch 1/20
5/5 [==============================] - 1s 45ms/step - loss: 0.0598 - val_loss:
0.0565
Epoch 2/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0559 - val_loss:
0.0545
Epoch 3/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0533 - val_loss:
0.0508
Epoch 4/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0494 - val_loss:
0.0470
Epoch 5/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0456 - val_loss:
0.0428
Epoch 6/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0413 - val_loss:
0.0387
Epoch 7/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0376 - val_loss:
0.0354
Epoch 8/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0344 - val_loss:
0.0326
Epoch 9/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0319 - val_loss:
0.0303
Epoch 10/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0297 - val_loss:
0.0283
Epoch 11/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0279 - val_loss:
0.0267
Epoch 12/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0264 - val_loss:
0.0253
Epoch 13/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0251 - val_loss:
0.0243
Epoch 14/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0242 - val_loss:
0.0235
Epoch 15/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0235 - val_loss:
0.0229
Epoch 16/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0230 - val_loss:
0.0224
Epoch 17/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0225 - val_loss:
0.0219
```

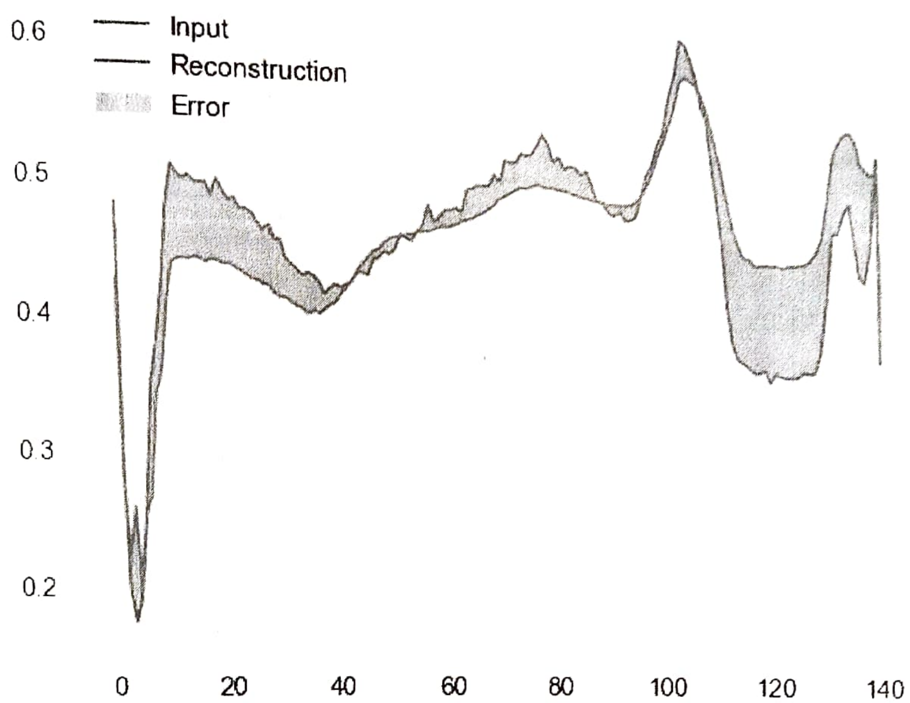
```
Epoch 18/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0220 - val_loss:
0.0214
Epoch 19/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0215 - val_loss:
0.0209
Epoch 20/20
5/5 [==============================] - 0s 9ms/step - loss: 0.0210 - val_loss:
0.0206
```

In [14]:
```python
plt.plot(history.history['loss'],label='Training Loss')
plt.plot(history.history['val_loss'],label='Testing Loss')
plt.legend()
```

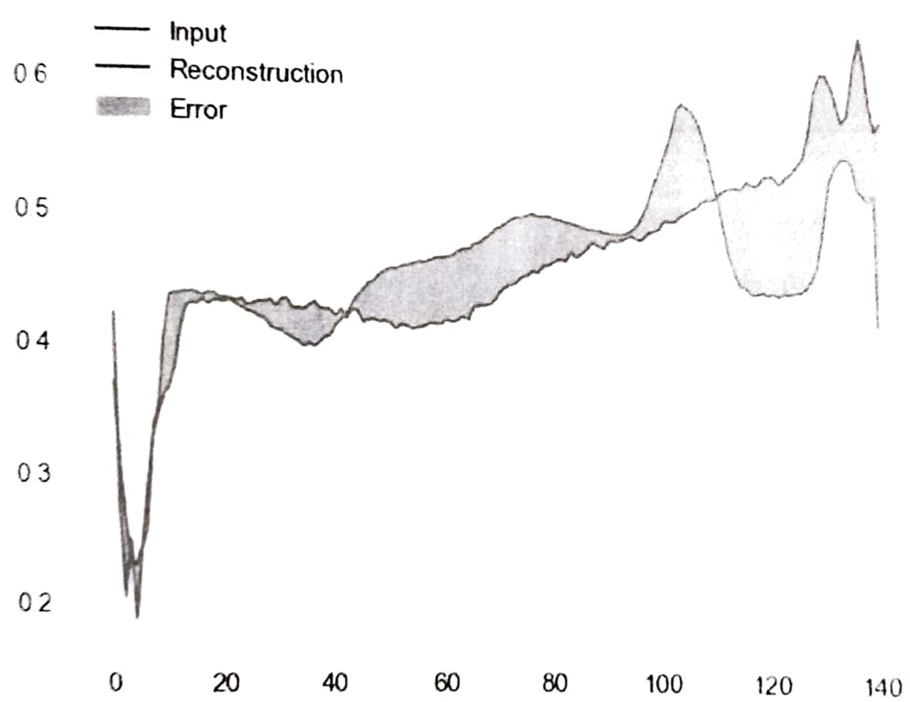
Out[14]: `<matplotlib.legend.Legend at 0x23e12fef790>`



```
In [15]: encoded_image = autoencoder.encoder(normal_test_data).numpy()
         decoded_image = autoencoder.decoder(encoded_image).numpy()

         plt.plot(normal_test_data[0],'b')
         plt.plot(decoded_image[0],'r')
         plt.fill_between(np.arange(140),decoded_image[0],normal_test_data[0],color='light
         plt.legend(labels=['Input',"Reconstruction","Error"])
         plt.show()
```

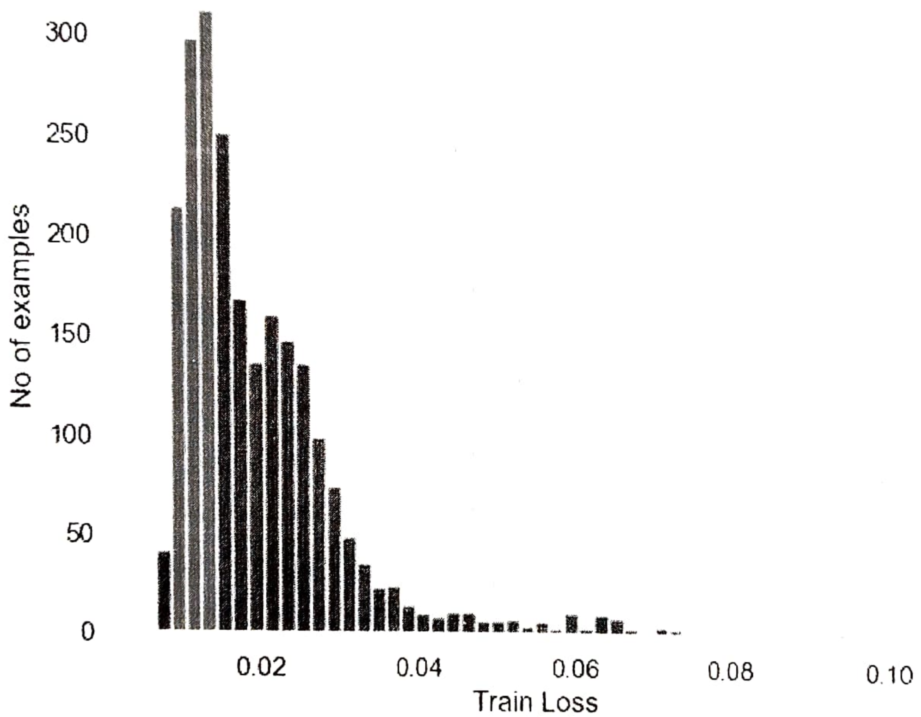


In [16]:
```
encoded_image = autoencoder.encoder(anamalous_test_data).numpy()
decoded_image = autoencoder.decoder(encoded_image).numpy()

plt.plot(anamalous_test_data[0],'b')
plt.plot(decoded_image[0],'r')
plt.fill_between(np.arange(140),decoded_image[0],anamalous_test_data[0],color='l:
plt.legend(labels=['Input',"Reconstruction","Error"])
plt.show()
```



In [17]:
```
reconstructions = autoencoder.predict(normal_train_data)
train_loss = tf.keras.losses.mae(reconstructions, normal_train_data)

plt.hist(train_loss[None,:],bins=50)
plt.xlabel("Train Loss")
plt.ylabel("No of examples")
plt.show()
```

```
74/74 [==============================] - 0s 983us/step
```

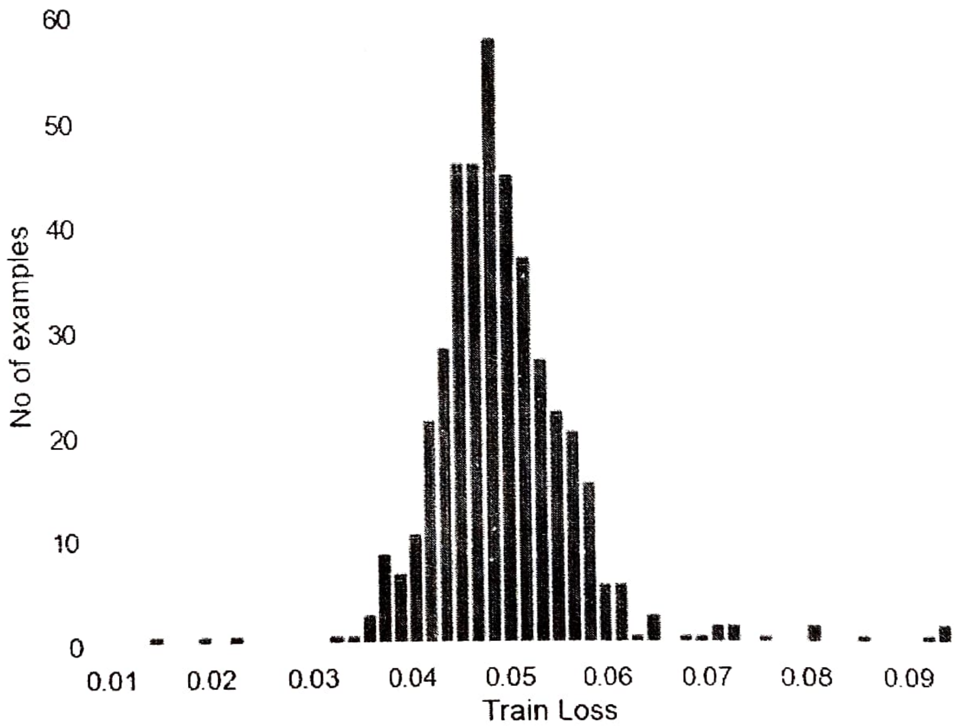


In [18]:
```
tnreshold = np.mean(train_loss) + np.std(train_loss)
print("Threshold: ",threshold)
```

```
Threshold:  0.03308283181023525
```

In [19]:
```
reconstructions = autoencoder.predict(anamalous_test_data)
test_loss = tf.keras.losses.mae(reconstructions, anamalous_test_data)

plt.hist(test_loss[None,:],bins=50)
plt.xlabel("Train Loss")
plt.ylabel("No of examples")
plt.show()
```

```
14/14 [==============================] - 0s 1ms/step
```



In [20]:
```
def predict(model,data,threshold):
  reconstructions = model(data)
  loss = tf.keras.losses.mae(reconstructions,data)
  return tf.math.less(loss, threshold)

def print_stats(predictions, labels):
  print("Accuracy = {}".format(accuracy_score(labels,preds)))
  print("Precision = {}".format(precision_score(labels,preds)))
  print("Recall = {}".format(recall_score(labels,preds)))
```

In [21]:
```
preds = predict(autoencoder, test_data, threshold)
print_stats(preds, test_labels)
```

```
Accuracy = 0.945
Precision = 0.9922027290448343
Recall = 0.9089285714285714
```