Assignment No. 4

Problem Statement

Use Autoencoder to implement anomaly detection. Build the model by using

a. Import required libraries

b. Upload/access the dataset

c.Encoder converts it into latent representation

d. Decoder networks convert it back to the original input

e. Compile the models with Optimizer, Loss, and Evaluation

Solution Expected

AutoEncoders are widely used in anomaly detection. The reconstruction errors are used as the anomaly scores. Let us look at how we can use AutoEncoder for anomaly detection using TensorFlow.

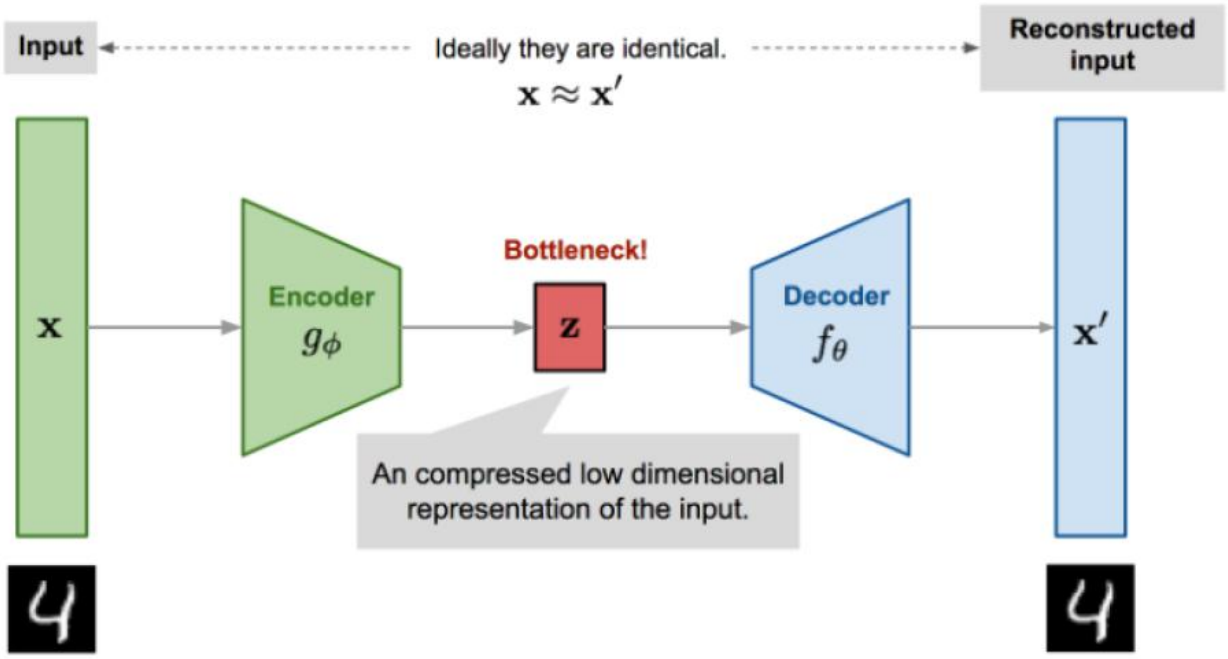
Import the required libraries and load the data. Here we are using the ECG data which consists of labels 0 and 1. Label 0 denotes the observation as an anomaly and label 1 denotes the observation as normal.

Objectives to be achieved

1) Use Autoencoder to implement anomaly detection.

Methodology to be used

AutoEncoder is a generative unsupervised deep learning algorithm used for reconstructing high-dimensional input data using a neural network with a narrow bottleneck layer in the middle which contains the latent representation of the input data.



Q/A

1)What is Anomaly Detectection ?

2) What are Autoencoders in Deep learning ?

3) Enlist different applications with Autoencoders in DL.

4) Enlist different types of anomaly detection Algorithms.

5)What is difference between Anomaly detection and Novelty Detection.

6) Explain different blocks and working of Autoencoders.

7) What is reconstruction and Reconstruction errors .

8) What is Minmaxscaler from sklearn.

8) Explain . train_test_split from sklearn.

9) What is anomaly scores.

10) Explain tensorfloe dataset.

11) Describe the ECG Dataset.

12) Explain keras Optimizers

13) Explain keras layers dense and dropouts

14) Explain keras losses and meansquarelogarthmicerror

15) Explain Relu activation function

Steps/ Algorithm

1. Dataset link and libraries :

Dataset : http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv

Libraries required :

Pandas and Numpy for data manipulation

Tensorflow/Keras for Neural Networks

Scikit-learn library for splitting the data into train-test samples, and for some basic model evaluation

For Model building and evaluation following libraries:

sklearn.metrics import accuracy_score

tensorflow.keras.optimizers import Adam

sklearn.preprocessing import MinMaxScaler

tensorflow.keras import Model, Sequential

tensorflow.keras.layers import Dense, Dropout

tensorflow.keras.losses import MeanSquaredLogarithmicError

Ref:https://www.analyticsvidhya.com/blog/2021/05/anomaly-detection-using-autoencoders-awalk-through-in-python/

a) Import following libraries from SKlearn :

i) MinMaxscaler (sklearn.preprocessing)

ii) Accuracy(sklearn.metrics) .

iii) train_test_split (model_selection)

b) Import Following libraries from tensorflow.keras : models , layers,optimizers,datasets, and set to respective values.

c) Grab to ECG.csv required dataset

d) Find shape of dataset

e) Use train_test_split from sklearn to build model (e.g. train_test_split(features, target, test_size=0.2, stratify=target)

f) Take usecase Novelty detection hence select training data set as Target class is 1 i.e. Normal class

g) Scale the data using MinMaxScaler.

h) Create Autoencoder Subclass by extending model class from keras.

i) Select parameters as i)Encoder : 4 layers ii) Decoder : 4 layers iii) Activation Function : Relu iv) Model : sequential.

j) Configure model with following parametrs : epoch = 20 , batch size =512 and compile with Mean Squared Logarithmic loss and Adam optimizer.

e.g. model = AutoEncoder(output_units=x_train_scaled.shape[1])

configurations of model

model.compile(loss='msle', metrics=['mse'], optimizer='adam')

history = model.fit(x_train_scaled, x_train_scaled,epochs=20,batch_size=512, validation_data=(x_test_scaled, x_test_scaled)

k) Plot loss,Val_loss, Epochs and msle loss

l) Find threshold for anomaly and do predictions :

e.g. : find_threshold(model, x_train_scaled):

reconstructions = model.predict(x_train_scaled)

provides losses of individual instances

reconstruction_errors = tf.keras.losses.msle(reconstructions, x_train_scaled)

threshold for anomaly scores

threshold = np.mean(reconstruction_errors.numpy()) \

+ np.std(reconstruction_errors.numpy())

return threshold

m) Get accuracy score

Sample Code with comments : Attach Printout with Output .

Import required libraries

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score,
accuracy_score, precision_score

RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

Read the dataset

I had downloaded the data from Kaggle and stored it in the local directory.

```
dataset = pd.read_csv("creditcard.csv")
```

Exploratory Data Analysis

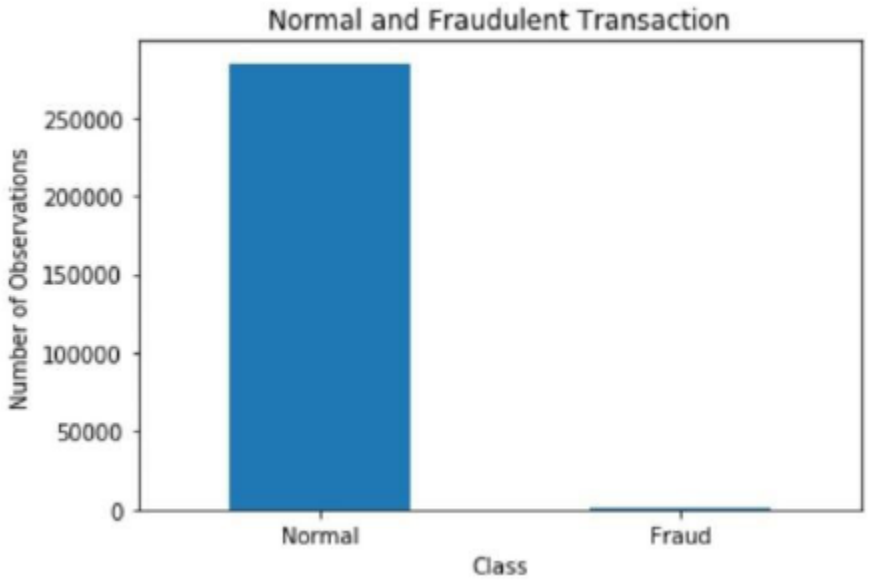
```
#check for any  nullvalues
print("Any nulls in the dataset
",dataset.isnull().values.any() )
print('-------')
print("No. of unique labels ",
len(dataset['Class'].unique()))
print("Label values
",dataset.Class.unique())

#0 is for normal credit card transaction
#1 is for fraudulent credit card
transaction
print('-------')
print("Break down of the Normal and Fraud
Transactions")
print(pd.value_counts(dataset['Class'],
sort = True) )
```

```
Any nulls in the dataset  False
-------
No. of unique labels  2
Label values  [0 1]
-------
Break down of the Normal and Fraud Transactions
0    284315
1       492
Name: Class, dtype: int64
```

Visualize the dataset

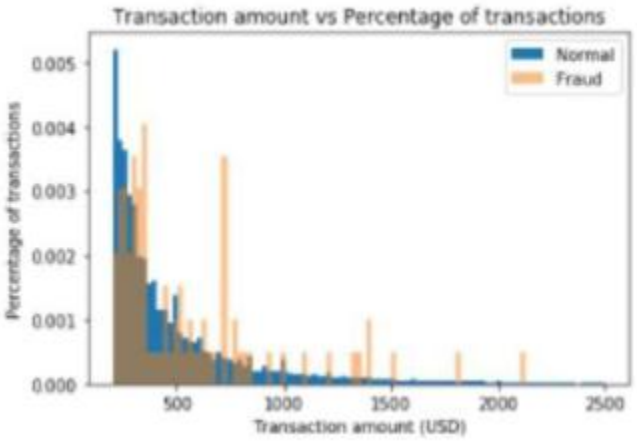
plotting the number of normal and fraud transactions in the dataset.

```
#Visualizing the imbalanced dataset
count_classes =
pd.value_counts(dataset['Class'], sort =
True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(dataset['Class'].uni
que())), dataset.Class.unique())
plt.title("Frequency by observation
number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
```



Visualizing the amount for normal and fraud transactions.

```
# Save the normal and fradulent
transactions in separate dataframe
normal_dataset = dataset[dataset.Class ==
0]
fraud_dataset = dataset[dataset.Class ==
1]

#Visualize transactionamounts for normal
and fraudulent transactions
bins = np.linspace(200, 2500, 100)
plt.hist(normal_dataset.Amount,
bins=bins, alpha=1, density=True,
label='Normal')
plt.hist(fraud_dataset.Amount, bins=bins,
alpha=0.5, density=True, label='Fraud')
plt.legend(loc='upper right')
plt.title("Transaction amount vs
Percentage of transactions")
plt.xlabel("Transaction amount (USD)")
plt.ylabel("Percentage of transactions");
plt.show()
```



Time and Amount are the columns that are not scaled, so applying StandardScaler to only Amount and Time columns. Normalizing the values between 0 and 1 did not work great for the dataset.

```
sc=StandardScaler()
dataset['Time'] =
sc.fit_transform(dataset['Time'].values.r
eshape(-1, 1))
dataset['Amount'] =
sc.fit_transform(dataset['Amount'].values
.reshape(-1, 1))
```

The last column in the dataset is our target variable.

```
raw_data = dataset.values
# The last element contains if the
transaction is normal which is
represented by a 0 and if fraud then 1
labels = raw_data[:, -1]

# The other data points are the
electrocadriogram data
data = raw_data[:, 0:-1]

train_data, test_data, train_labels,
test_labels = train_test_split(
    data, labels, test_size=0.2,
random_state=2021
)
```

Normalise the data to have a value between 0 and 1

Normalize the data to have a value between 0 and 1

```
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) /
(max_val - min_val)
test_data = (test_data - min_val) /
(max_val - min_val)

train_data = tf.cast(train_data,
tf.float32)
test_data = tf.cast(test_data,
tf.float32)
```

Use only normal transactions to train the Autoencoder.

Normal data has a value of 0 in the target variable. Using the target variable to create a normal and fraud dataset.

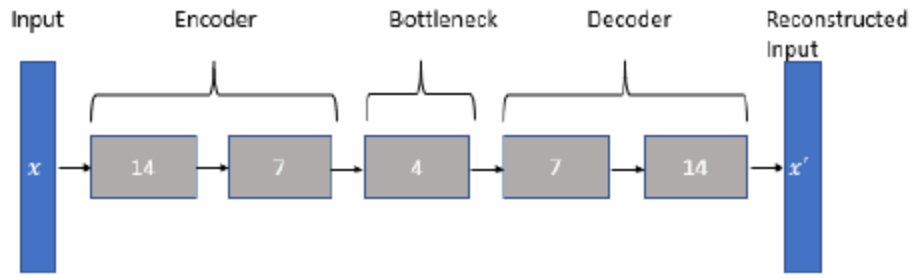
```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

#creating normal and fraud datasets
normal_train_data =
train_data[~train_labels]
normal_test_data =
test_data[~test_labels]

fraud_train_data =
train_data[train_labels]
fraud_test_data = test_data[test_labels]
print(" No. of records in Fraud Train
Data=",len(fraud_train_data))
print(" No. of records in Normal Train
data=",len(normal_train_data))
print(" No. of records in Fraud Test
Data=",len(fraud_test_data))
print(" No. of records in Normal Test
data=",len(normal_test_data))
```

Set the training parameter values

```
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]
#num of columns, 30
encoding_dim = 14
hidden_dim_1 = int(encoding_dim / 2) #
hidden_dim_2=4
learning_rate = 1e-7
```

Create the Autoencoder

The architecture of the autoencoder is shown below.



```
#input Layer
input_layer =
tf.keras.layers.Input(shape=(input_dim,
))

#Encoder

encoder =
tf.keras.layers.Dense(encoding_dim,
activation="tanh",
activity_regularizer=tf.keras.regularizer
s.l2(learning_rate))(input_layer)
encoder=tf.keras.layers.Dropout(0.2)
(encoder)
encoder =
tf.keras.layers.Dense(hidden_dim_1,
activation='relu')(encoder)
encoder =
tf.keras.layers.Dense(hidden_dim_2,
activation=tf.nn.leaky_relu)(encoder)

# Decoder
decoder =
tf.keras.layers.Dense(hidden_dim_1,
activation='relu')(encoder)
decoder=tf.keras.layers.Dropout(0.2)
(decoder)
decoder =
tf.keras.layers.Dense(encoding_dim,
activation='relu')(decoder)
decoder =
tf.keras.layers.Dense(input_dim,
activation='tanh')(decoder)

#Autoencoder
autoencoder =
tf.keras.Model(inputs=input_layer,
outputs=decoder)
autoencoder.summary()
```

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30)]              0
_________________________________________________________________
dense (Dense)                (None, 14)                434
_________________________________________________________________
dropout (Dropout)            (None, 14)                0
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 105
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 32
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 35
_________________________________________________________________
dropout_1 (Dropout)          (None, 7)                 0
_________________________________________________________________
dense_4 (Dense)              (None, 14)                112
_________________________________________________________________
dense_5 (Dense)              (None, 30)                450
=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
```

Define the callback for checkpoint and early stopping.

```
cp =
tf.keras.callbacks.ModelCheckpoint(filepa
th="autoencoder_fraud.h5",

mode='min', monitor='val_loss',
verbose=2, save_best_only=True)
# define our early stopping
early_stop =
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True
```

Plot training and test loss

Compile the Autoencoder

```
autoencoder.compile(metrics=['accuracy'],

loss='mean_squared_error',
                    optimizer='adam')
```

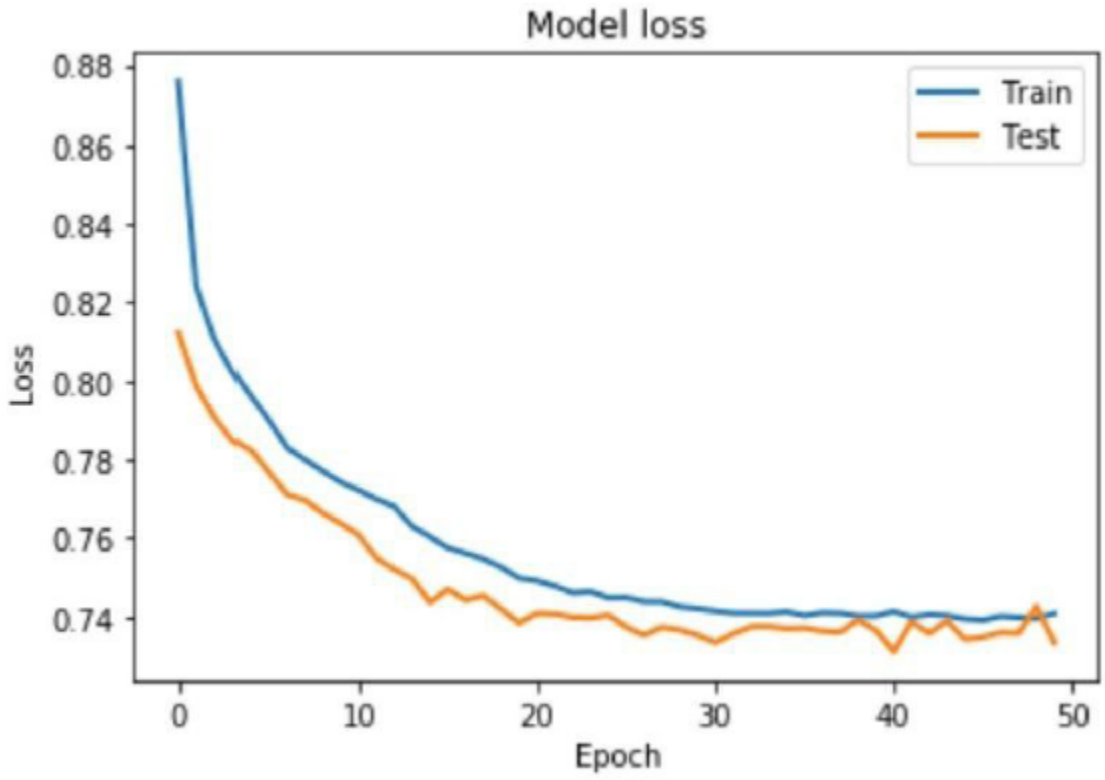
Train the Autoencoder

```
history =
autoencoder.fit(normal_train_data,
normal_train_data,
                    epochs=nb_epoch,

batch_size=batch_size,
                    shuffle=True,
                    validation_data=
(test_data, test_data),
                    verbose=1,
                    callbacks=[cp,
early_stop]
                    ).history
```

```
Epoch 11/25
7085/7108 [============================>.] - ETA: 0s - loss: 5.3796e-04
Epoch 00011: val_loss did not improve from 0.00053
Restoring model weights from the end of the best epoch.
7108/7108 [==============================] - 8s 1ms/step - loss: 5.3799e-04 - val_loss: 5.3531e-04
Epoch 00011: early stopping
```

Plot training and test loss

```
plt.plot(history['loss'], linewidth=2,
label='Train')
plt.plot(history['val_loss'],
linewidth=2, label='Test')
plt.legend(loc='upper right')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
#plt.ylim(ymin=0.70,ymax=1)
plt.show()
```



Detect Anomalies on test data

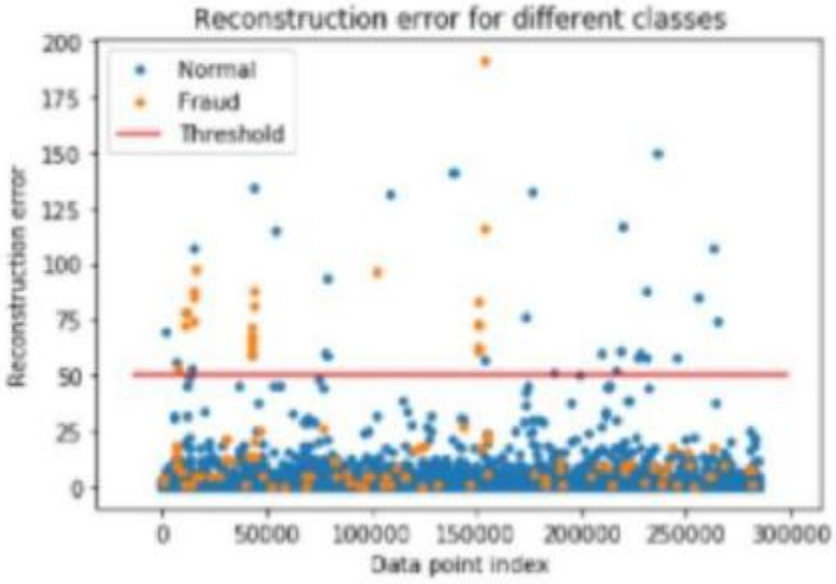
Anomalies are data points where the reconstruction loss is higher

To calculate the reconstruction loss on test data, predict the test data and calculate the mean square error between the test data and the reconstructed test data.

```
test_x_predictions =
autoencoder.predict(test_data)
mse = np.mean(np.power(test_data -
test_x_predictions, 2), axis=1)
error_df =
pd.DataFrame({'Reconstruction_error':
mse,
                          'True_class':
test_labels})
```

Plotting the test data points and their respective reconstruction error sets a threshold value to visualize if the threshold value needs to be adjusted.

```
threshold_fixed = 50
groups = error_df.groupby('True_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index,
group.Reconstruction_error, marker='o',
ms=3.5, linestyle='',
            label= "Fraud" if name == 1
else "Normal")
ax.hlines(threshold_fixed, ax.get_xlim()
[0], ax.get_xlim()[1], colors="r",
zorder=100, label='Threshold')
ax.legend()
plt.title("Reconstruction error for
normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```



Detect anomalies as points where the reconstruction loss is greater

than a fixed threshold. Here we see that a value of 52 for the threshold will be good.

Evaluating the performance of the anomaly detection

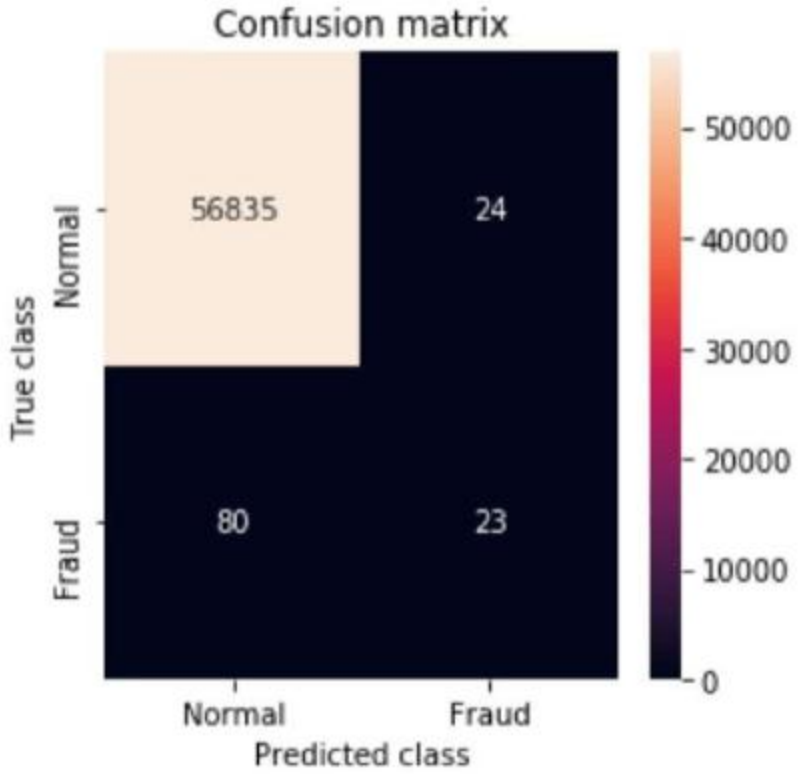
```
threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0
for e in
error_df.Reconstruction_error.values]
error_df['pred'] =pred_y
conf_matrix =
confusion_matrix(error_df.True_class,
pred_y)
```

```
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix,
xticklabels=LABELS, yticklabels=LABELS,
annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()

# print Accuracy, precision and recall
print(" Accuracy:
",accuracy_score(error_df['True_class'],
error_df['pred']))
print(" Recall:
",recall_score(error_df['True_class'],
error_df['pred']))
print(" Precision:
",precision_score(error_df['True_class'],
error_df['pred']))
```

Evaluating the performance of the anomaly detection

```
threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0
for e in
error_df.Reconstruction_error.values]
error_df['pred'] =pred_y
conf_matrix =
confusion_matrix(error_df.True_class,
pred_y)
```

```
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix,
xticklabels=LABELS, yticklabels=LABELS,
annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()

# print Accuracy, precision and recall
print(" Accuracy:
",accuracy_score(error_df['True_class'],
error_df['pred']))
print(" Recall:
",recall_score(error_df['True_class'],
error_df['pred']))
print(" Precision:
",precision_score(error_df['True_class'],
error_df['pred']))
```




```
Accuracy:  0.9981742214107651
Recall:  0.223300970873786 42
Precision:  0.48936170212765956
```

As our dataset is highly imbalanced, we see a high accuracy but a low recall and precision. Things to further improve precision and recall would add more relevant features, different architecture for autoencoder, different hyperparameters, or a different algorithm.

Conclusion:

Autoencoders can be used as an anomaly detection algorithm when we have an unbalanced dataset where we have a lot of good examples and only a few anomalies. Autoencoders are trained to minimise reconstruction error. When we train the autoencoders on normal data or good data, we can hypothesise that the anomalies will have higher reconstruction errors than the good or normal data.