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

## Downloading the data

```
dir_path = tf.keras.utils.get_file(fname="/content/creditcard.csv",origin="https://datahub
```

```
Downloading data from https://datahub.io/machine-learning/creditcard/r/creditcard.csv
151114991/151114991 [==============================] - 2s 0us/step
```

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

```
df.head()
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.( |
| **1** | 0.0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.( |
| **2** | 1.0 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.: |
| **3** | 1.0 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.: |
| **4** | 2.0 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.: |

5 rows × 31 columns

```
df.describe()
```

| | Time | V1 | V2 | V3 | V4 | |
|---|---|---|---|---|---|---|
| **count** | 284807.000000 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.84 |
| **mean** | 94813.859575 | 1.168375e-15 | 3.416908e-16 | -1.379537e-15 | 2.074095e-15 | 9.60 |
| **std** | 47488.145955 | 1.958696e+00 | 1.651309e+00 | 1.516255e+00 | 1.415869e+00 | 1.38 |

```
df["Class"].unique()
```

```
array(["'0'", "'1'"], dtype=object)
```

### Type Casting the Label

```
classes = df["Class"]
```

```
df["Class"] = classes=="'0'"
```

```
df["Class"].unique()
```

```
array([ True, False])
```

```
print(pd.value_counts(df["Class"]))
```

```
True     284315
False       492
Name: Class, dtype: int64
```

### Normalization

```
sc = StandardScaler()
df["Time"] = sc.fit_transform(df["Time"].values.reshape(-1,1))
df["Amount"] = sc.fit_transform(df["Amount"].values.reshape(-1,1))
```

```
data = df.iloc[:,0:-1]
labels = df.iloc[:,-1]
```

### Train test spliting

```
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, test_siz
```

```
min = train_data.min()
max = train_data.max()
```

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

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

Making the distiction between normal and fraud data

```
normal_data_train = train_data[train_labels]
fraud_data_train = train_data[~train_labels]
```

Input Layer

```
input_shape = train_data.shape[1]
```

```
input_layer = tf.keras.layers.Input(shape=input_shape)
```

Encoder Layer

```
encoder = tf.keras.layers.Dense(14,activation="tanh")(input_layer)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(7, activation="relu")(encoder)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(4, activation=tf.nn.leaky_relu)(encoder)
```

Decoder Layer

```
decoder = tf.keras.layers.Dense(7, activation="relu")(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(14, activation="relu")(decoder)
decoder = tf.keras.layers.Dense(input_shape, activation="tanh")(decoder)
```

Combining the autoencoder parts

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

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 30)]              0

 dense_3 (Dense)             (None, 14)                434

 dropout_2 (Dropout)         (None, 14)                0

 dense_4 (Dense)             (None, 7)                 105

 dropout_3 (Dropout)         (None, 7)                 0
```

```
 dense_5 (Dense)             (None, 4)                 32

 dense_6 (Dense)             (None, 7)                 35

 dropout_4 (Dropout)         (None, 7)                 0

 dense_7 (Dense)             (None, 14)                112

 dense_8 (Dense)             (None, 30)                450

=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
_________________________________________________________________
```

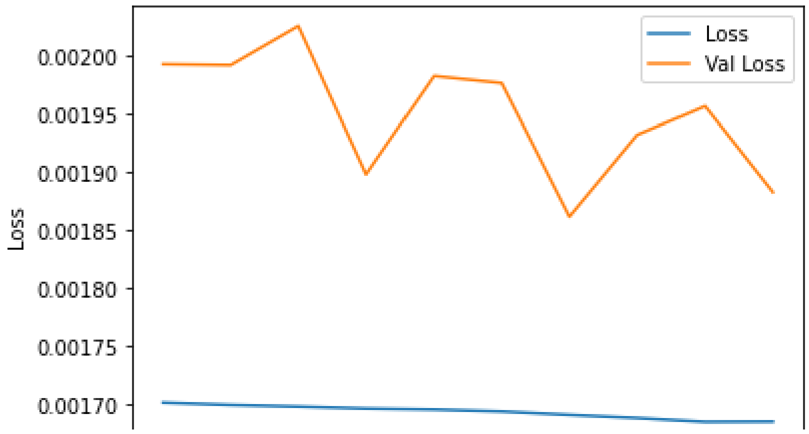
```
autoencoder.compile(metrics=["accuracy"], optimizer="adam", loss="mean_squared_error")
```

```
history = autoencoder.fit(normal_data_train, normal_data_train, epochs=10, batch_size=64,
```

```
Epoch 1/10
3555/3555 [==============================] - 9s 3ms/step - loss: 0.0017 - accuracy: 0
Epoch 2/10
3555/3555 [==============================] - 9s 3ms/step - loss: 0.0017 - accuracy: 0
Epoch 3/10
3555/3555 [==============================] - 9s 2ms/step - loss: 0.0017 - accuracy: 0
Epoch 4/10
3555/3555 [==============================] - 10s 3ms/step - loss: 0.0017 - accuracy:
Epoch 5/10
3555/3555 [==============================] - 9s 3ms/step - loss: 0.0017 - accuracy: 0
Epoch 6/10
3555/3555 [==============================] - 12s 3ms/step - loss: 0.0017 - accuracy:
Epoch 7/10
3555/3555 [==============================] - 10s 3ms/step - loss: 0.0017 - accuracy:
Epoch 8/10
3555/3555 [==============================] - 9s 2ms/step - loss: 0.0017 - accuracy: 0
Epoch 9/10
3555/3555 [==============================] - 9s 2ms/step - loss: 0.0017 - accuracy: 0
Epoch 10/10
3555/3555 [==============================] - 9s 2ms/step - loss: 0.0017 - accuracy: 0
```

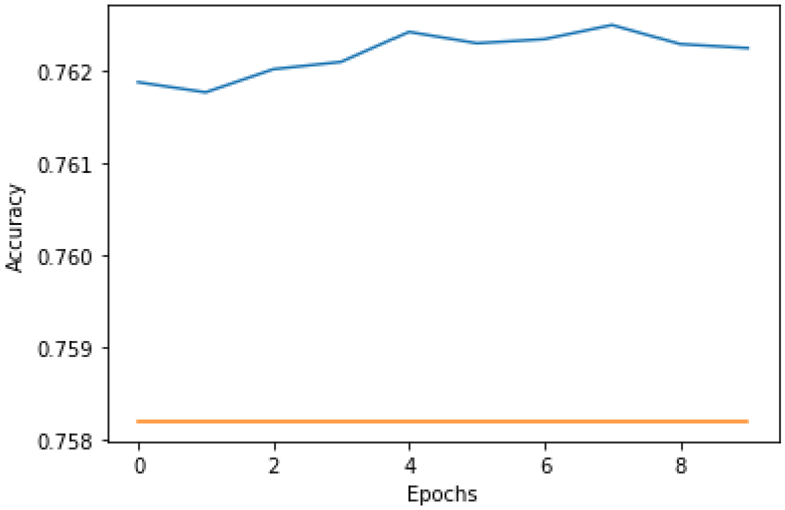
## Loss while training

```
plt.figure()
plt.plot(history.history["loss"], label="Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




Accuracy

```
plt.figure()
plt.plot(history.history["accuracy"], label="Accuracy")
plt.plot(history.history["val_accuracy"], label="Val Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.show()
```




Reconstruction to show distinction between fraud and normal transaction

```
reconstructions = autoencoder.predict(normal_data_train)
train_loss = tf.keras.losses.mae(reconstructions, normal_data_train)

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

```
7109/7109 [==============================] - 10s 1ms/step
```

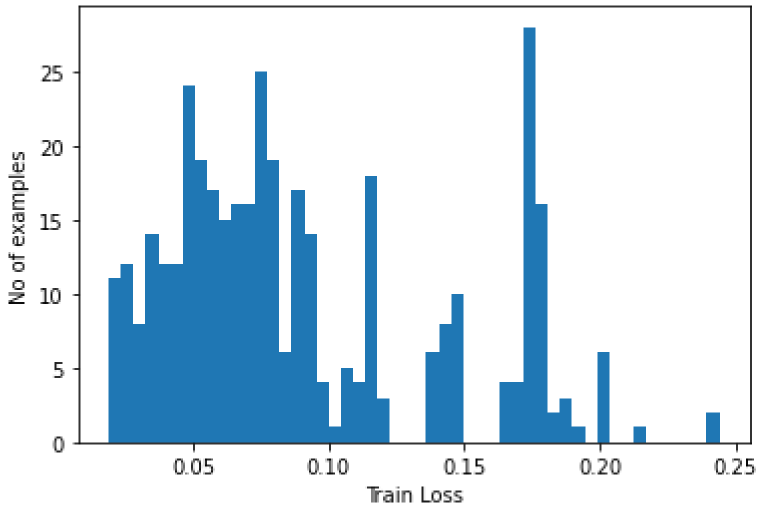

```
threshold = np.mean(train_loss) + np.std(train_loss)
print("Threshold: ",threshold)
```

```
Threshold:  0.034391352921359704
```

```
reconstructions = autoencoder.predict(fraud_data_train)
test_loss = tf.keras.losses.mae(reconstructions, fraud_data_train)

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

```
12/12 [==============================] - 0s 3ms/step
```




Conclusion: The model developed have a threshold of 0.0343.. below which the transaction is normal, otherwise fraud

✓ 0s completed at 10:27 PM