```
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
```

```
!pip install tensorflow --user
!pip install keras
!pip install daytime
!pip install torch
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/di
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: ml-dtypes==0.2.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: numpy>=1.23.5 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/di
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.2
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/loca
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/
Requirement already satisfied: tensorboard<2.15,>=2.14 in /usr/local/lib/python3
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in /usr/local/
Requirement already satisfied: keras<2.15,>=2.14.0 in /usr/local/lib/python3.10/
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.1
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /usr/local/lib/
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/loc
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.1
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/d
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages
Collecting daytime
  Downloading daytime-0.4.tar.gz (2.4 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: daytime
  Building wheel for daytime (setup.py) ... done
  Created wheel for daytime: filename=daytime-0.4-py3-none-any.whl size=2401 sha
  Stored in directory: /root/.cache/pip/wheels/cd/40/c7/fc109bc6716d31e4d5fdc0cd
Successfully built daytime
Installing collected packages: daytime
Successfully installed daytime-0.4
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/di
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

```
from google.colab import files
from IPython.display import Image
```

```
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
#dataset = pd.read_csv("E:\Teachning material\Deep learning BE IT 2019 course\creditca
dataset = pd.read_csv("creditcard.csv")
#dataset.head
print(list(dataset.columns))
dataset.describe()
```

```
#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) )


#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'].unique())), dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");


# 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
Normalizing the values between 0 and 1 did not work great for the dataset.'''


sc=StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-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 a
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
)


'''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 cre

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))


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


#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.regularizers.l2(learning_rate))
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()
```

```
"""Define the callbacks for checkpoints and early stopping"""

cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",
                               mode='min', monitor='val_loss', verbose=2, save_best_o
# 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)


#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



#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 th
```

```
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 thresl
#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
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 tl
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.valu
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']))


'''As our dataset is highly imbalanced, we see a high accuracy but a low recall and pı

Things to further improve precision and recall would add more relevant features,
different architecture for autoencoder, different hyperparameters, or a different alg


history
```