\* 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"]

\*

from google.colab import drive

\*

drive.mount('/content/drive')

\*

dataset = pd.read\_csv("/content/creditcard.csv")

\*

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 transcation

#1 is for fraudulent credit card transcation

print('-------')

print("Break down of Normal and Fraud Transcations")

print(pd.value\_counts(dataset['Class'],sort=True))

\*

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")

\*

normal\_dataset = dataset[dataset.Class == 0]

fraud\_dataset = dataset[dataset.Class == 1]

#Visualize transcation amounts for normal and fraudulent transcations

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("Transcation Amount vs Percentage of Transcations")

plt.xlabel("Transcation Amount (USD)")

plt.ylabel("Percentage of Transcations")

plt.show()

\*

sc = StandardScaler()

dataset['Time'] = sc.fit\_transform(dataset['Time'].values.reshape(-1,1))

dataset['Amount'] = sc.fit\_transform(dataset['Amount'].values.reshape(-1,1))

raw\_data = dataset.values

#The last element contains if the transcation is normal which is represented by 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)

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)

train\_labels = train\_labels.astype(bool)

test\_labels = test\_labels.astype(bool)

**IN** #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))

**IN** nb\_epoch = 50

batch\_size = 64

input\_dim = normal\_train\_data.shape[1]

#num of columns,30

encoding\_dim = 14

hidden\_dim1 = int(encoding\_dim / 2)

hidden\_dim2 = 4

learning\_rate = 1e-7

**IN** #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))(input\_layer)

encoder = tf.keras.layers.Dropout(0.2)(encoder)

encoder = tf.keras.layers.Dense(hidden\_dim1,activation='relu')(encoder)

encoder = tf.keras.layers.Dense(hidden\_dim2,activation=tf.nn.leaky\_relu)(encoder)

#Decoder

decoder = tf.keras.layers.Dense(hidden\_dim1,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()

**IN**

cp = tf.keras.callbacks.ModelCheckpoint(filepath="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=11,

                mode='min',

                restore\_best\_weights=True

)

autoencoder.compile(metrics=['accuracy'],loss= 'mean\_squared\_error',optimizer='adam')

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

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()

**IN** 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("Reconstructions error for normal and fraud data")

plt.ylabel("Reconstruction error")

plt.xlabel("Data point index")

plt.show()

**IN** 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']))