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"]

dataset = pd.read\_csv("E:\Teachning material\Deep learning BE IT 2019 course\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()

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))

#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\_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()

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()

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})

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();

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']))