**Import the libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

**Import the dataset**

data=pd.read\_csv('/kaggle/input/creditcardfraud/creditcard.csv')

data.head()

**Check the shape of the dataset**

data.shape

**Check null values in dataset**

data.isnull().sum()

**Check the data type of the value in all column**

data.info()

**Check that how many values are present in that class**

data['Class'].value\_counts()

**Divide data frame into fraud and nonfraud data**

non\_fraud=data[data['Class']==0]

fraud=data[data['Class']==1]

non\_fraud.shape, fraud.shape

**Select 492 non fraud data from dataset**

non\_fraud=non\_fraud.sample(fraud.shape[0])

non\_fraud.shape

data=fraud.append(non\_fraud, ignore\_index=True)

data

**Check the value counts**

data.Class.value\_counts()

**Divide the data frame into dependent and independent variable**

X=data.drop(['Class'], axis=1)

y=data.Class

**Check the shape**

X.shape, y.shape

**Divide the dataset into training and testing dataframe**

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=99)

**Check the shape again**

X\_train.shape,X\_test.shape,y\_train.shape,y\_test.shape

X\_train

**Scalar**

scaler=StandardScaler()

X\_train=scaler.fit\_transform(X\_train)

X\_test=scaler.fit\_transform(X\_test)

X\_train,y\_train

**Convert into array**

y\_train=y\_train.to\_numpy()

y\_test=y\_test.to\_numpy()

X\_train.shape

**Process the data**

X\_train=X\_train.reshape(X\_train.shape[0],X\_train.shape[1],1)

X\_test=X\_test.reshape(X\_test.shape[0],X\_test.shape[1],1)

**Check the shape again**

X\_train.shape, X\_test.shape

**Import the libraries**

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense,Flatten,Conv1D,BatchNormalization,Dropout

**Import the model**

model=Sequential()

**Import layers**

model.add(Conv1D(filters=32, kernel\_size=2, activation='relu', input\_shape=X\_train[0].shape))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(Conv1D(filters=64, kernel\_size=2, activation='relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2))

**Build ann**

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(1, activation='relu'))

**Summary**

model.summary()

**Compile the model**

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

**Fitting the model**

history=model.fit(X\_train,y\_train,epochs=20,validation\_data=(X\_test,y\_test))

**Plot**

def plot\_learningcurve(history,epochs):

epoch=range(1,epochs+1)

**Accuracy**

plt.plot(epoch, history.history['accuracy'])

plt.plot(epoch, history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.xlabel('epoch')

plt.ylabel('accuracy')

plt.legend(['train','val'], loc='upper left')

plt.show()

**Loss**

plt.plot(epoch, history.history['loss'])

plt.plot(epoch, history.history['val\_loss'])

plt.title('Model loss')

plt.xlabel('epoch')

plt.ylabel('loss')

plt.legend(['train','val'], loc='upper left')

plt.show()

plot\_learningcurve(history,20)