Project: Credit card Fraud Detection We are going to build project on The Deep Learning model to detect the Credit Card fraud done by European cardholders in september 2013. We will learn Deep Learning concepts such as Artification Neural Networks, Layers, Activation Functions, Neurons. and top of that we will build Deep Learning Algorithm which mimics humain br ain We analyse data to get meaning information from them and Visualize them to recognise different patterns among them and showing meaning full information It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. **Importing neccessary Libraries** In [1]: import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense from sklearn.model selection import train test split from sklearn.metrics import classification report from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA #Over Sampling and under sampling libraries from imblearn.over sampling import SMOTE from imblearn.under sampling import RandomUnderSampler from keras.models import Sequential from keras.layers import Dense, Dropout import warnings warnings.filterwarnings("ignore") /usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 since we've dropped support for Python 2.7. Please rely on the official version of six (https://pypi.org/project/six/). "(https://pypi.org/project/six/).", FutureWarning) /usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.neighbors.b ase module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / func tions should instead be imported from sklearn.neighbors. Anything that cannot be imported from sklearn.neighbor s is now part of the private API. warnings.warn(message, FutureWarning) Importing Data set The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, In [2]: path="/content/drive/MyDrive/Ml\AI/Deep Learning Itvedant/Project DL/creditcard.csv" df=pd.read csv(path) df.head() V10 V11 Out[2]: Time V1 V2 **V**3 **V4 V**5 V6 **V7 V8** V9 V12 0.239599 0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.098698 0.363787 0.090794 -0.551600 -0.617801 -0.991 0.060018 -0.082361 -0.078803 1.191857 0.266151 0.166480 0.448154 0.085102 -0.255425 -0.166974 1.612727 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461 0.247676 -1.514654 0.207643 In [3]: df.shape (284807, 31)Out[3]: In [4]: df.size 8829017 Out[4]: In [5]: df.info() <class 'pandas.core.frame.DataFrame'> RangeIndex: 284807 entries, 0 to 284806 Data columns (total 31 columns): # Column Non-Null Count Dtype Time 284807 non-null float64 V1 284807 non-null float64 284807 non-null float64 V2 284807 non-null float64 V3 284807 non-null float64 V4284807 non-null float64 5 V5 284807 non-null float64 V6 284807 non-null float64 7 V7 284807 non-null float64 8 V8 284807 non-null float64 9 V9 10 V10 284807 non-null float64 284807 non-null float64 11 V11 284807 non-null float64 12 V12 284807 non-null float64 13 V13 284807 non-null float64 14 V14 15 V15 284807 non-null float64 284807 non-null float64 16 V16 284807 non-null float64 17 V17 284807 non-null float64 18 V18 284807 non-null float64 19 V19 284807 non-null float64 20 V20 284807 non-null float64 21 V21 284807 non-null float64 22 V22 284807 non-null float64 23 V23 284807 non-null float64 24 V24 284807 non-null float64 25 V25 284807 non-null float64 26 V26 284807 non-null float64 27 V27 284807 non-null float64 28 V28 29 Amount 284807 non-null float64 30 Class 284807 non-null int64 dtypes: float64(30), int64(1) memory usage: 67.4 MB In [6]: df.describe() **V1** V2 **V**3 V4 **V5 V6 V7 V8** Out[6]: Time **count** 284807.000000 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.848070e+05 2.010663e-15 94813.859575 3.919560e-15 5.688174e-16 -8.769071e-15 2.782312e-15 -1.552563e-15 -1.694249e-15 -1.927028e-16 47488.145955 1.651309e+00 1.516255e+00 1.415869e+00 1.380247e+00 1.332271e+00 1.237094e+00 std 1.958696e+00 1.194353e+0C min 0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -8.486401e-01 -6.915971e-01 54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 25% -7.682956e-01 -5.540759e-01 -2.086297e-01 -1.984653e-02 -5.433583e-02 -2.741871e-01 50% 84692.000000 1.810880e-02 6.548556e-02 1.798463e-01 4.010308e-02 2.235804e-02 **75%** 139320.500000 1.315642e+00 8.037239e-01 1.027196e+00 7.433413e-01 6.119264e-01 3.985649e-01 5.704361e-01 3.273459e-01 max 172792.000000 2.454930e+00 2.205773e+01 9.382558e+00 1.687534e+01 3.480167e+01 7.330163e+01 1.205895e+02 2.000721e+01 In [7]: # compare the values for both transactions df.groupby('Class').mean() Out[7]: **Time** V1 V2 **V3 V4 V**5 **V6 V7 V8 V9** V10 V11 Class **0** 94838.202258 0.008258 -0.006271 0.012171 -0.007860 0.005453 0.002419 0.009637 -0.000987 0.009824 0.004467 -0.006576 **1** 80746.806911 -4.771948 3.623778 -7.033281 4.542029 -3.151225 -1.397737 -5.568731 0.570636 -2.581123 -5.676883 3.800173 -6.25 Checking missing values In [8]: df.isna().sum() Time Out[8]: V10 V2 0 V3 0 V40 V5 0 V6 0 V7 0 V8 0 V9 0 V10 0 V11 0 V12 0 V13 0 V14 0 V15 0 V16 0 V17 0 V18 0 V19 0 V20 0 V21 0 V22 0 V23 0 V24 0 V25 0 V26 0 V27 V28 0 Amount 0 Class 0 dtype: int64 **Data Visualization** In [33]: df['Class'].value counts().plot(kind='bar') <matplotlib.axes. subplots.AxesSubplot at 0x7f128ddebad0> Out[33]: 250000 200000 150000 100000 50000 sns.displot(df['Time']) <seaborn.axisgrid.FacetGrid at 0x7f6c26be7750> Out[]: 7000 6000 5000 4000 3000 2000 1000 25000 50000 75000 100000125000150000 175000 sns.scatterplot(x='Time', y='Amount', hue='Class', data=df) <matplotlib.axes. subplots.AxesSubplot at 0x7f6c263ea450> Class 25000 20000 15000 10000 5000 0 75000 100000 125000 150000 175000 50000 plt.figure(figsize=(28,18)) sns.heatmap(df.corr(),annot=True) <matplotlib.axes. subplots.AxesSubplot at 0x7f6c26349e90> Out[]: 012 0.011 0.42 0.11 0.17 0.063 0.085 0.037 0.0087 0.031 0.25 0.12 0.066 0.099 0.18 0.012 0.013 0.09 0.029 0.051 0.045 0.14 0.051 0.016 0.23 0.041 0.0051 0.0094 0.011 0.012 1 47e-17-1 4e-1518e-17 6.4e-17 24e-16 2e-15 9.5e-17 22e-16 7.4e-17 24e-16 24e-16 24e-16-2.1e-169.4e-16-3.3e-166.3e-16 5e-16 29e-1618e-16 1.8e-167.5e-17 98e-16 7.4e-17 9.8e-168.6e-17 3.2e-17 9.8e-16 023 0.1 0.11 18e-17-1.1e-163.4e-16 1 -1.9e-152.7e-161.6e-16.5.2e-16.3.9e-16.6.1e-16-2.1e-165.7e-161.5e-168.5e-17-1.5e-166.9e-164.4e-161.5e-16-2.7e-163.2e-16-1e-16.2.1e-16.6.e-17. 2.2e-165.4e-16-6.2e-16-6.4e-17-5.9e-17. 0.099 0.13 0.063 24e-16 5e-16 14e-15-2.7e-1679e-16 1 1.4e-16-1.7e-1611e-16 29e-1649e-16 21e-16-2.3e-1635e-16-3.5e-16-3.5e-16-3.6e-16 2.8e-16 2.7e-16 1.9e-16-1.6e-16-3.4e-16-7.2e-17-1.3e-1511e-15-2.4e-16-2.6e-1648e-16 022 0.044 0.085 2e-15 4e-16 22e-1516e-16-4.2e-1614e-16 1 8.7e-177.9e-16 3e-17 -1.1e-1515e-15-9.9e-17.1.7e-1619e-17 29e-1611e-15-1.1e-162.9e-161.7e-1619e-16-1.1e-152.3e-16-2.6e-1712e-15-7.3e-16-2.6e-177.9e-16-8e-17 0.4 0.19 0.037 9.5e-174.4e-173.4e-165.2e-167.6e-16-1.7e-168.7e-17 1 29e-169.1e-17 2e-16-6.3e-172.4e-161.1e-162e-16 5e-16-3.5e-164.1e-165.3e-161.1e-162.4e-165.5e-163.9e-16-1.8e-161.4e-161.2e-161.2e-161.2e-164.5e-16 0087 2 2e-16-5.7e-174.2e-163 9e-16 4 2e-16 1 1e-16 7.9e-16 2 9e-16 2 9e-16 1 2e-16 1 1e-16 7.9e-16 2 9e-16 1 2e-16 1 2e-16 1 2e-16 1 2e-16 1 2e-16 1 2e-16 4 3e-16 4 6e-17 2 9e-17 5 9e-16 2 3e-16 1 1e-15 4 3e-16 4 9e-16 2 9e-16 2 9e-16 2 3e-16 1 1e-15 4 3e-16 4 9e-16 2 9 0031 74e-17-4.8e-1663e-1661e-16-66e-1629e-16 3e-17 91e-17-2.8e-16 1 26e-1614e-15-8.9e-1626e-167.6e-161.7e-153.7e-15 4e-16 27e-17-1.1e-1581e-16-6.7e-1638e-16 4e-17 -2.9e-162.6e-16-3.1e-161.5e-16 0.1 0.22 0.25 24e-16 95e-16-5.5e-17-2.1e-167.3e-16 4.9e-16-1.1e-15 2e-16 4.7e-16 2.6e-16 3 2e-15 1.9e-15 3.6e-17 4.8e-16-6.2e-168 7e-16 6e-16 3.2e-16-3.9e-16-3.8e-17 2.2e-16 1.2e-15 4.6e-161.1e-16-2.6e-16 3.1e-16 0.0001 015 012 24e-16-6.6e-1622e-16-5.7e-16 38e-16 21e-1615e-15-6.3e-17-2 4e-151 4e-15 32e-15 1 -2.3e-1418e-16 89e-16 35e-16-9 9e-165-9e-169 3e-17 13e-16 32e-16-5.9e-161 4e-16 49e-16 51e-16-5.8e-16-2.3e-167.3e-16 0.0095 0.26 0.066 - 2.1e-16.39e-16.6.9e-161.5e-16.9.6e-162.3e-169.9e-17-2.4e-162.7e-168.9e-161.9e-15-2.3e-14 1 2.8e-15-4.7e-163.4e-16-3.2e-16-3.2e-16-2.3e-18.9.5e-17-2.7e-17-5.9e-16-5.5e-168.1e-17-2.1e-16-4.5e-16.1e-16.1 1 3e-15 3.2e-16 7.5e-16-8.6e-163.2e-16 1.9e-17-8.9e-161.1e-16-4.6e-163.9e-16 3.8e-16-1.3e-15-1.1e-15-0.003-0.004 V15 5e-16 -3 3e-161.7e-15-6 2e-1635e-1634e-167.9e-1713e-15 1 19e-15 -3e-15 1e-15 39e-16-3.9e-1639e-178.5e-164.3e-166.6e-165 2e-167.8e-168.6e-164.0039 -0.2 16 3.7e-15 8.7e-16-9.9e-163.2e-164.6e-15-3.2e-161.9e-15-1-5.6e-15-3.9e-169-9e-16-7.8e-168.4e-165.4e-16-5.5e-174.8e-16 4.9e-16 8.8e-16-2.2e-16.00073-0.33 7.5e-16 -3e-15 -5.6e-15 1 -2.4e-154.9e-16-1.1e-158.7e-17-3.6e-16-1.1e-16-2.3e-16-3.2e-16-2.4e-16-8.8e-16-9.9e-16-4.9e-162.9e-16 1 -1.1e-151.1e-15 5e-16 1.6e-16-1.5e-16 -3e-16 045 -1.8e-16.84e-17 -3e-17 -1e-16 -1.4e-161.6e-161.9e-16-2.4e-164.6e-17.81e-16-3.9e-163.2e-16.95e-17.16e-17.19e-17-3.9e-167.8e-16-1.1e-15 4e-16 -1.1e-15 4 39e-15.61e-161.3e-16-2.8e-164.9e-16 -1e-15 51e-16 0.11 V21 0.012 0.1 0.091 0.19 0.13 0.095 0.044 0.19 0.02 0.098 0.22 0.15 0.26 0.0046 0.3 0.0045 0.02 0.33 0.11 0.035 0.02 0.04 0.00081-0.0027-0.0072 0.0033 0.0045 0.018 0.0095 0.0056 **Checking class Distribution** In [9]: df["Class"].value counts() 284315 Out[9]: Name: Class, dtype: int64 In [10]: df['Class'].value counts().plot(kind='bar') <matplotlib.axes. subplots.AxesSubplot at 0x7f1291e2c850> Out[10]: 250000 200000 150000 100000 50000 Dividing independant Variable and Target Variable In [11]: x = df.drop(["Class", "Time"], axis=1) = df["Class"] **V7** Out[11]: **V1** V2 **V3 V4 V5 V6 V8 V9** V10 V12 -1.359807 -0.072781 -0.338321 0.462388 0.239599 0.098698 0.363787 0.090794 -0.551600 -0.617801 -0.99 2.536347 1.378155 0.266151 1.191857 0.166480 0.060018 -0.082361 -0.078803 0.085102 1.612727 0.448154 -0.255425 -0.1669741.065235 0.48 -1.358354 -1.340163 -0.503198 0.791461 0.247676 -1.514654 0.624501 0.066084 1.773209 0.379780 1.800499 0.207643 0.71 -0.966272 -0.010309 -0.1852261.792993 -0.863291 1.247203 0.237609 0.377436 -1.387024 -0.054952 -0.226487 0.178228 0.50-1.158233 1.548718 0.403034 -0.407193 0.592941 0.817739 0.753074 0.538196 0.877737 0.095921 -0.270533-0.8228431.34 -2.066656 -5.364473 4.356170 **284802** -11.881118 10.071785 -9.834783 -2.606837 -4.918215 7.305334 1.914428 -1.593105 284803 -0.055080 2.035030 -0.738589 0.294869 0.584800 -0.975926 0.915802 -0.732789 0.868229 1.058415 0.024330 -0.150189 3.031260 284804 -0.296827 0.432454 -0.484782 0.411614 1.919565 -0.301254 -3.249640 -0.557828 2.630515 0.708417 0.063119 -0.18 0.623708 0.392087 -0.399126 -0.962886 284805 -0.240440 0.530483 0.702510 0.689799 -0.377961 -0.686180 0.679145 -1.933849 284806 -0.533413 -0.189733 0.703337 -0.506271 -0.012546 -0.649617 1.577006 -0.414650 0.486180 -0.915427 -1.040458 -0.031513 -0.18284807 rows × 29 columns In [12]: 0 Out[12]: 0 2 0 3 0 284802 0 284803 0 284804 0 284805 0 284806 Name: Class, Length: 284807, dtype: int64 In [13]: # sc=StandardScaler() # scaled Amount=pd.DataFrame(df["Amount"]) # scaled_Amount=pd.DataFrame(sc.fit_transform(scaled_Amount)) # scaled_Amount In [14]: # x=pd.concat([x,scaled_Amount],axis=1) **Splitting Data Into Train and Test Data** In [15]: from sklearn.model_selection import train_test_split x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=1) In [16]: from imblearn.under_sampling import RandomUnderSampler rus = RandomUnderSampler(random_state=1) x_train_bal, y_train_bal = rus.fit_resample(x_train, y_train) pd.Series(y_train_bal).value_counts() 381 Out[16]: 381 dtype: int64 Scaling In [17]: sc=StandardScaler() x_train_ss=sc.fit_transform(x_train_bal) x_test_ss=sc.transform(x_test) x_train_ss array([[1.38543605e-01, -1.20612787e-01, 5.43003906e-01, ..., -3.48398357e-02, -1.05010852e-03, -4.41048280e-01], [1.58481250e-01, -1.53955277e-01, 6.37894043e-01, ..., 2.69919264e-01, -5.66765190e-01, -4.42669777e-01], [2.60320118e-01, -4.89798488e-01, 6.95967075e-01, ..., -1.26601442e-01, -4.54569552e-03, -3.17131748e-01], [6.48799392e-01, 2.63492459e-01, -7.02782645e-02, ..., -4.82815815e-02, 2.94618432e-01, -4.55471072e-01], [-3.79288956e-01, 4.78857964e-01, -2.65053220e-01, ...,-2.24940145e-01, -2.59406057e-01, -4.55471072e-01], [-1.21833684e-01, -4.360/9305e-01, 3.1396/005e-02, ...,7.27333757e-01, 4.48796292e-01, 1.35975251e+00]]) **Neural Network** In [18]: model=Sequential() model.add(Dense(32,activation='relu',input_dim=29)) model.add(Dense(64,activation='relu')) model.add(Dense(64,activation='relu')) model.add(Dropout(0.2)) model.add(Dense(1,activation='sigmoid')) In [19]: model.compile(optimizer="adam",loss="binary crossentropy") In [20]: trained_model = model.fit(x_train_ss,y_train_bal,epochs=25,batch_size=50) Epoch 1/25 16/16 [============] - 1s 2ms/step - loss: 0.6150 Epoch 2/25 16/16 [=============] - Os 2ms/step - loss: 0.4241 Epoch 3/25 16/16 [========] - Os 2ms/step - loss: 0.3149 Epoch 4/25 16/16 [========] - Os 2ms/step - loss: 0.2442 Epoch 5/25 16/16 [========] - Os 2ms/step - loss: 0.2073 Epoch 6/25 16/16 [========] - Os 2ms/step - loss: 0.1852 Epoch 7/25 16/16 [========] - Os 2ms/step - loss: 0.1732 Epoch 8/25 16/16 [========] - Os 2ms/step - loss: 0.1583 Epoch 9/25 16/16 [========] - Os 2ms/step - loss: 0.1535 Epoch 10/25 16/16 [========] - Os 2ms/step - loss: 0.1429 Epoch 11/25 16/16 [========] - Os 2ms/step - loss: 0.1348 Epoch 12/25 16/16 [========] - Os 2ms/step - loss: 0.1268 Epoch 13/25 16/16 [========] - Os 2ms/step - loss: 0.1291 Epoch 14/25 16/16 [========] - Os 3ms/step - loss: 0.1169 Epoch 15/25 16/16 [========] - Os 2ms/step - loss: 0.1197 Epoch 16/25 16/16 [========] - Os 2ms/step - loss: 0.1180 Epoch 17/25 16/16 [========] - Os 2ms/step - loss: 0.1129 Epoch 18/25 16/16 [========] - Os 2ms/step - loss: 0.1130 Epoch 19/25 16/16 [========] - Os 2ms/step - loss: 0.1090 Epoch 20/25 16/16 [========] - Os 3ms/step - loss: 0.0993 Epoch 21/25 16/16 [========] - Os 2ms/step - loss: 0.0972 Epoch 22/25 16/16 [========] - Os 2ms/step - loss: 0.0894 Epoch 23/25 16/16 [===========] - Os 2ms/step - loss: 0.0931 Epoch 24/25 16/16 [===========] - Os 2ms/step - loss: 0.0862 Epoch 25/25 16/16 [========] - Os 2ms/step - loss: 0.0896 In [21]: plt.plot(trained_model.history['loss']) [<matplotlib.lines.Line2D at 0x7f128c580b50>] Out[21]: 0.6 0.5 0.4 0.3 0.2 **Model Prediction** In [22]: y_pred = model.predict(x_test_ss) In [23]: $y_pred = np.where(y_pred >= 0.5,1,0)$ array([[0], Out[23]: [0], [0], [0], [0], [0]]) **Evaluation Of Model** In [24]: print(classification_report(y_test,y_pred)) recall f1-score precision support 0 1.00 0.97 0.98 71091 0.04 0.86 0.07 111 71202 0.96 accuracy macro avg 0.52 0.92 0.53 71202 weighted avg 1.00 0.96 0.98 71202 In [25]: # from imblearn.over_sampling import RandomOverSampler # ros = RandomOverSampler(random_state=1) # xtrain_bal1, ytrain_bal1 = ros.fit_resample(x_train, y_train) In [26]: # pd.Series(ytrain_ball).value_counts() In [27]: # sc=StandardScaler() # x_train_ss1=sc.fit_transform(xtrain_bal1) # x_test_ss1=sc.transform(x_test) # x_train_ss1 In [28]: model1=Sequential() # model1.add(Dense(32,activation='relu',input_dim=30)) # model1.add(Dense(64,activation='relu')) # model1.add(Dense(64,activation='relu')) # model1.add(Dense(1,activation='sigmoid')) In [29]: # model1.compile(optimizer="adam",loss="binary crossentropy") In [30]: # trained model1 = model1.fit(x train ss1,ytrain bal1,epochs=25,batch size=50) In [31]: # y_pred1 = model.predict(x_test_ss1) # y pred1 = np.where(y pred >= 0.5,1,0) # print(classification_report(y_test,y_pred1))