CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING AND DEEP LEARNING ALGORITHMS

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

NAME (Paul Roy)
(Reg. No.: 124003214, B.Tech Computer Science & Engineering)
NAME (Deeban Sankar)
(Reg. No.: 124003077, B.Tech Computer Science & Engineering)
NAME (Barath Suresh)
(Reg. No.: 124003053, B.Tech Computer Science & Engineering)

May 2023

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled "**Credit Card Fraud Detection Using Machine Learning and Deep Learning Algorithms**" submitted as a requirement for the course, CSE300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Barath Suresh (Reg. No.124003053, B.Tech Computer Science & Engineering)** , **Mr. Deeban Sankar (Reg. No.124003077, B.Tech Computer Science & Engineering)** , **Mr. Paul Roy (Reg. No.124003214, B.Tech Computer Science & Engineering)** during the academic year 2022-23, in the School of Computing, under my supervision.

Signature of Project Supervisor:

Name with Affiliation : Dr. Pradeepa S,Assistant Professor-III,School of Computing

Date : 11-05-2023

Mini Project *Viva voce* held on _____

Examiner 1 **Examiner 2**

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providingus with an opportunity and the necessary infrastructure for carrying out this project as a part ofour curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan,** Dean**,** Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli,** Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. S. Gopalakrishnan, Associate Dean,** Department of Computer Application, **Dr. B.Santhi**, Associate Dean, Research, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology

Our guide **Dr. Pradeepa S**, Assistant Professor-III, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing mean opportunity to showcase my skills through project.

List of Figures

Abbreviations

ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
KNN	K-Nearest Neighbors
SVM	Support Vector Machine
AI	Artificial Intelligence

Abstract

Credit card fraud have become an increasing threat to financial institutions. Current credit card fraud detection algorithms have come a long way, but still there are plenty of issues that they face such have high false positive and false negative rates, new fraud techniques, high imbalanced data etc. Here, the main aim is to detect such fraud, which includes highly imbalanced data. This includes the usage of machine learning algorithms such as Decision Tree, Random Forest, Logistic Regression, K-Nearest Neighbors, XG Boost and Support Vector Machine. However due to low accuracy of ML algorithms, there is a need to use deep learning algorithms. Here, the usage of Convolutional Neural Network(CNN) can reduce loss and improve the performance. The current deep learning algorithms face the problem of less accuracy due to imbalanced dataset. Here using more layers in CNN can solve the issue and provide slightly more accuracy. A comparative analysis is done using both machine learning and deep learning algorithms to present the difference in accuracy and efficiency. After evaluating the models, CNN with 20 layers outperforms other model with a higher accuracy of 99.956%.

KEY WORDS: Credit card fraud, Machine Learning, Deep Learning, CNN, false positive, false negative

Table of Contents

CHAPTER 1

SUMMARY OF THE BASE PAPER

1.1 BASE PAPER DETAILS

TITLE: Credit Card Fraud Detection Using State-of-the-Art Machine Learning and Deep Learning Algorithms

AUTHOR: Fawaz Khaled Alarfaj , Iqra Malik , Hikmat Ullah Khan , Naif Almusallam , Muhammad Ramzan , and ,uzamil Ahmed

YEAR OF PUBLISH: 2022

INDEX: Science Citation Index Expanded

PUBLISHER: IEEE

DOI : https://doi.org/10.1109/ACCESS.2022.3166891

1.2 DATASET DETAILS

The dataset has transactions made by credit cards in 2018 by European cardholders. The dataset contains about 284,807 transactions, in which only 492 transactions are fraud. The dataset is highly imbalance as only 0.172% of the total transactions are fraud and the rest are not.

Principal Component Analysis(PCA) has been applied to the dataset due to which it contains only numerical variables. The dataset contains several columns that includes features V1, V2, V3…V28 that are obtained after applying PCA, Amount, Time and Class.

1.3 INTRODUCTION

Rapid improvement in the technological field has made us more advanced digitally, which also includes us making transactions through means of online banking. This results in the usage of credit card for online transactions to increase due to the development of e-banking and other online payment platforms. But that the usage of credit card in is rising, so is the misuse and crime against credit card, which costs the economy billions of dollars every year. CCF detection has emerged as one of the key objectives in digital payments. Therefore, a way to prevent these misuse much be implemented.

Machine Learning models have been used in numerous fields to solve many issues & challenges. ML is a field of AI that uses a system to make predictions based on prior data. The usage of machine learning algorithms in the credit card transaction dataset must distinguish between fraudulent and non-fraudulent transactions. Other factor has also been taken into consideration such as reaction time, false positives etc. In the recent years, Deep Learning models are also being applied due to high accuracy. Deep Learning is one of the machine learning models that has neural structure. To create high dimensional representations of data for use in addressing various issues, deep learning employs a nonlinear structure. It can also handle massive data volumes and a number of multidimensional information kinds, including audio and visual data. DL models are given more preference than ML models due to more efficiency. But the issue caused by unbalanced dataset cannot be solved by traditional deep learning algorithms.

1.4 PROPOSED ARCHITECTURE

In this study, initially machine learning models have been implemented to detect fraud. Machine learning models such as Decision Tree, Random Forest, Logistic Regression, K-Nearest Neighbors, XG Boost and SVM have been used in this study. Using these models, accuracy and f1 score have been computed. These scored are compared with each other to test for the model with better score.

But due to the dataset being imbalanced, chances of false positive and false negative are high. So the machine learning models are less preferable. In such case, Deep learning approaches can give a better accuracy. Here Convolutional Neural Network(CNN) model is used as it can automatically learn and extract features more effectively and efficiently. It is composed of multiple layers which include convolution layer, pooling layers and fully connected layers. The convolution layer helps to extract spatial and temporal features. But to get more accuracy and less loss than the traditional deep learning approach, more number of layers are added for feature extraction and classification of credit card transactions as fraud or not. Model with 5 layers, 11 layers, 13 layers, 14 layers, 17 layers and 20 layers have been used. As more number of layers are added, the accuracy is high compared to prior DL models.

Overall, this proposed model offers a more accurate reading to differentiate a fraudulent and non-fraudulent transaction by the means of CNN with more number of layers to get high training and validation accuracy than the existing model.

1.5 PROPOSED METHODOLOGY

Algorithms for Machine Learning models

1. **Logistic regression:** Method where the categorical dependent or other independent variable used to analyze the relationship and predict an output. The output of the model is between 0 and 1.

2. **K-Nearest Neighbors:** The dataset is classified into nodes and split into two classes. The input data point to the class is taken and distance between them is calculated. The most used distance metric is Euclidean distance. Then place the new case in the category that closely matches the other categories that are available.

3. **Support Vector Machine:** Algorithm where linear and non-linear data where the data is transformed into higher-dimensional space using kernel function. Support vectors are made to it maximizes the distance between the closest data points to different classes by finding a hyperplane.

4. **Decision Tree:** The algorithm is basically where the optimal tree is found to minimize the classification error. The data is basically splits the data based on information gain, gain ratio or Gini impurity and the portioning the data into subsets based on the outcome of the test.

5. **Random Forest:** It is based on decision tree. It builds multiple decision tree and combining their prediction to make a final prediction. The problem of overfitting is easily blown over as the randomness and variation in the construction of the tree.

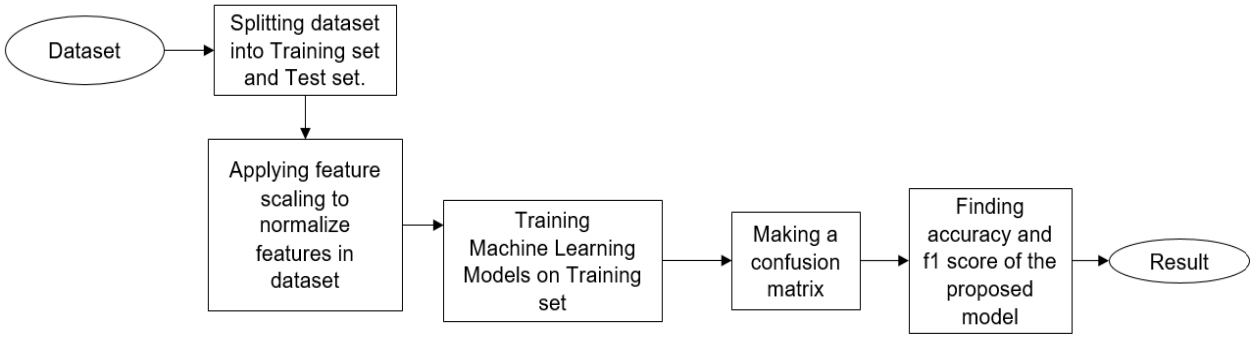
6. **XG Boost:** The basic idea behind XG Boost is to iteratively train a sequence of weak decision tree models on the residuals (the differences between the actual and predicted values) of the previous models, in order to improve the overall prediction accuracy

Convolutional Neural Network (CNN) layers

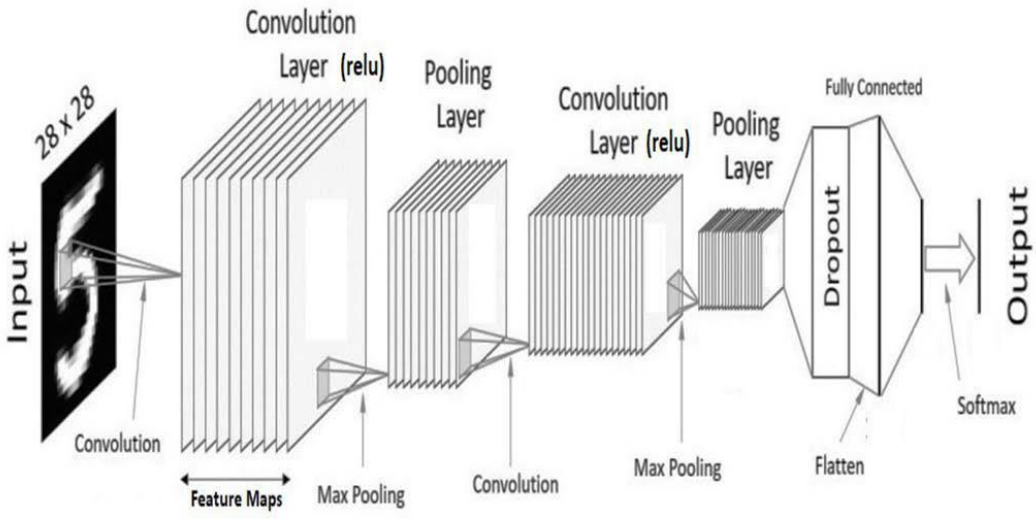
1. **Conv1D:** A neural network layer used for processing one-dimensional data.
2. **BatchNormalization:** A technique which enhances the performance and stability of the neural networks.
3. **Dropout:** A technique which is used to prevent overfitting. It is done by random deactivating some neurons during training.
4. **Flatten:** A technique which transforms multi-dimensional array into one-dimensional array. This is normally done before passing the convolution layer to the output layer.
5. **Dense:** A fully connected layer which is also known as the output layer where each neuron is connected to the previous and next layer.

Activation function used

1. **ReLu:** A simple non-linear function used in neural network which introduces non-linearity to the output. Defined as f(x) = max (0, x)
2. **Sigmoid:** A simple non-linear used in neural network which maps any input value between 0 and 1. Defined as f(x) = 1 / (1 + ex(-x))



1.1 Workfow for Machine learning models



1.2 CNN Architecture

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1 RELATED WORK

This paper addresses a number of related studies in credit card fraud detection, including machine learning (ML) and deep learning (DL) techniques. CCF detection frequently employs machine learning (ML) techniques like random forest (RF) and support vector machine (SVM). These strategies can be used with ensemble strategies to build reliable detection classifiers. It is recommended to handle large datasets with DL techniques like convolutional neural networks (CNN). The paper highlights the potential of DL algorithms for identifying credit card fraud and sets a standard for further investigation in this field. Furthermore, the study stresses how crucial data pre-processing is to the ML process.

2.2 MERITS OF THIS BASE PAPER

- The project gives a great statistical input to statistician for prediction of prediction of future frauds.
- The project runs on the latest algorithms with the combination of latest hardware technology
- The project is versatile to missing data, labels, etc.

2.3 DEMERITS OF THIS BASE PAPER

- The project runs on already preprocessed data.
- The project is time of running depend on amount of computation power available.
- The ratio of non-fraudulent data to fraudulent data is really high

CHAPTER 3

CODE

```
# CREDIT CARD FRAUD DETECTION RESULTS OF DIFFERENT MODELS
## LIBRARIES
### IMPORTING NECESSARY LIBRARIES
import pandas as pd
import os
import numpy as np
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, BatchNormalization
from sklearn.utils import shuffle, resample
from tensorflow import keras
from keras import layers
import matplotlib.pyplot as plt
from tabulate import tabulate
import joblib
import pickle
class_label = ['Non-Default(0)','Default(1)'] # env var
## DATASET
### IMPORTING THE DATASET
data = pd.read_csv('creditcard.csv')
### SPLITING THE DATASET INTO FEATURES (X) AND TARGET (Y) AND SPLIT THEM
INTO TRAINING AND TEST SET
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
sns.countplot(x='Class',data=data)
## MACHINE LEARNING MODELS
### XGBOOST
#### CREATING XGBOOST AND FITTING
with os.fdopen(os.open("Saved Models\\xgb.joblib", os.O_WRONLY | os.O_CREAT)) as file:
    if os.path.getsize("Saved Models\\xgb.joblib") == 0:
        xgb_classifier = XGBClassifier()
        xgb_classifier.fit(X_train, Y_train)
        joblib.dump(xgb_classifier,filename="Saved Models\\xgb.joblib")
    else:
        xgb_classifier = joblib.load("Saved Models\\xgb.joblib")
#### MAKING PREDICTIONS ON TEST DATA
xgb_y_pred = xgb_classifier.predict(X_test)
### FEATURE SCALING
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
### DECISION TREE CLASSIFIER
#### CREATING DECISION TREE CLASSIFIER AND FITTING
with os.fdopen(os.open("Saved Models\\DecisionTreeClassifier.joblib", os.O_WRONLY |
os.O_CREAT)) as file:
    if os.path.getsize("Saved Models\\DecisionTreeClassifier.joblib") == 0:
        dtc = DecisionTreeClassifier(max_depth=4,random_state=42,criterion='entropy')
        dtc.fit(X_train, Y_train)
        joblib.dump(dtc,filename="Saved Models\\DecisionTreeClassifier.joblib")
    else:
        dtc = joblib.load("Saved Models\\DecisionTreeClassifier.joblib")
#### MAKING PREDICTIONS ON TEST DATA
dtc_y_pred = dtc.predict(X_test)
### K-NEAREST-NEIGHBOUR
#### CREATING KNNCLASSIFIER AND FITTING
with os.fdopen(os.open("Saved Models\\KNNClassifier.joblib", os.O_WRONLY | os.O_CREAT)) as
file:
    if os.path.getsize("Saved Models\\KNNClassifier.joblib") == 0:
        knn_classifier= KNeighborsClassifier(n_neighbors = 5)
        knn_classifier.fit(X_train, Y_train)
        joblib.dump(knn_classifier,filename="Saved Models\\KNNClassifier.joblib")
    else:
        knn_classifier = joblib.load("Saved Models\\KNNClassifier.joblib")
#### MAKING PREDICTIONS ON TEST DATA
knn_y_pred = knn_classifier.predict(X_test)
```

```
### LOGISTIC REGRESSION
#### CREATING LOGISTICREGRESSION AND FITTING
with os.fdopen(os.open("Saved Models\\LogisticRegressionClassifier.joblib", os.O_WRONLY | os.O_CREAT)) as file:
    if os.path.getsize("Saved Models\\LogisticRegressionClassifier.joblib") == 0:
        log_classifier= LogisticRegression(random_state=42)
        log_classifier.fit(X_train, Y_train)
        joblib.dump(log_classifier,filename="Saved Models\\LogisticRegressionClassifier.joblib")
    else:
        log_classifier = joblib.load("Saved Models\\LogisticRegressionClassifier.joblib")
#### MAKING PREDICTIONS ON TEST DATA
log_y_pred = log_classifier.predict(X_test)
### SUPPORT VECTOR MACHINE
#### CREATING SVC AND FITTING
with os.fdopen(os.open("Saved Models\\SVMClassifier.joblib", os.O_WRONLY | os.O_CREAT)) as file:
    if os.path.getsize("Saved Models\\SVMClassifier.joblib") == 0:
        svm_classifier = SVC(kernel = 'linear', random_state = 42)
        svm_classifier.fit(X_train, Y_train)
        joblib.dump(svm_classifier,filename="Saved Models\\SVMClassifier.joblib")
    else:
        svm_classifier = joblib.load("Saved Models\\SVMClassifier.joblib")
#### MAKING PREDICTION ON TEST DATA
svm_y_pred = svm_classifier.predict(X_test)
### RANDOM FOREST
#### CREATING RANDOMFOREST AND FITTING
with os.fdopen(os.open("Saved Models\\RandomForestClassifier.joblib", os.O_WRONLY | os.O_CREAT)) as file:
    if os.path.getsize("Saved Models\\RandomForestClassifier.joblib") == 0:
        rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
        rf_classifier.fit(X_train, Y_train)
        joblib.dump(rf_classifier,filename="Saved Models\\RandomForestClassifier.joblib")
    else:
        rf_classifier = joblib.load("Saved Models\\RandomForestClassifier.joblib")
#### MAKING PREDICTIONS ON TEST DATA
rf_y_pred = rf_classifier.predict(X_test)
### CONFUSION MATRICES ON DIFFERENT MODELS
#### DECISION TREE
cm_dt = confusion_matrix(Y_test, dtc_y_pred)
sns.heatmap(cm_dt,annot=True,fmt='d',xticklabels=class_label, yticklabels=class_label, cmap='Blues')
plt.xlabel("Predicted")
```

```
plt.ylabel("True")
plt.title("Confusion Matrix of Decision Tree")
plt.show()
#### K-NEAREST-NEIGHBOUR
cm_knn = confusion_matrix(Y_test, knn_y_pred)
sns.heatmap(cm_knn,annot=True,fmt='d',xticklabels=class_label, yticklabels=class_label,
cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix of KNN")
plt.show()
#### LOGISTIC REGRESSION
cm_log = confusion_matrix(Y_test, log_y_pred)
sns.heatmap(cm_log,annot=True,fmt='d',xticklabels=class_label, yticklabels=class_label,
cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix of Logistic Regression")
plt.show()
#### SUPPORT VECTOR MACHINE
cm_svm = confusion_matrix(Y_test, svm_y_pred)
sns.heatmap(cm_svm,annot=True,fmt='d',xticklabels=class_label, yticklabels=class_label,
cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix of Support Vector Machine")
plt.show()
#### RANDOM FOREST
cm_rf = confusion_matrix(Y_test, rf_y_pred)
sns.heatmap(cm_rf,annot=True,fmt='d',xticklabels=class_label, yticklabels=class_label,
cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix of Random Forest")
plt.show()
#### XGBOOST
cm_xgb = confusion_matrix(Y_test, xgb_y_pred)
sns.heatmap(cm_xgb,annot=True,fmt='d',xticklabels=class_label, yticklabels=class_label,
cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix of XGBoost")
```

```
plt.show()
### EVALUATING THE MODELS AND THE RESULTS
#### CALCULATING ACCURACY SCORE AND F-1 SCORE
# ACCURACY SCORE
dtc_acc_score = accuracy_score(Y_test, dtc_y_pred)*100
knn_acc_score = accuracy_score(Y_test, knn_y_pred)*100
log_acc_score = accuracy_score(Y_test, log_y_pred)*100
svm_acc_score = accuracy_score(Y_test, svm_y_pred)*100
rf_acc_score = accuracy_score(Y_test, rf_y_pred)*100
xgb_acc_score = accuracy_score(Y_test, xgb_y_pred)*100

# F-1 SCORE
dtc_f1_score = f1_score(Y_test, dtc_y_pred)*100
knn_f1_score = f1_score(Y_test, knn_y_pred)*100
log_f1_score = f1_score(Y_test, log_y_pred)*100
svm_f1_score = f1_score(Y_test, svm_y_pred)*100
rf_f1_score = f1_score(Y_test, rf_y_pred)*100
xgb_f1_score = f1_score(Y_test, xgb_y_pred)*100
#### RESULTS
header = ["Model","Accuracy","F-1"]
results = [
   ["Decicion Tree","{:.2f}%".format(dtc_acc_score),"{:.2f}%".format(dtc_f1_score)],
   ["KNN","{:.2f}%".format(knn_acc_score),"{:.2f}%".format(knn_f1_score)],
   ["Logistic Regression","{:.2f}%".format(log_acc_score),"{:.2f}%".format(log_f1_score)],
   ["SVM","{:.2f}%".format(svm_acc_score),"{:.2f}%".format(svm_f1_score)],
   ["Random Forest","{:.2f}%".format(rf_acc_score),"{:.2f}%".format(rf_f1_score)],
   ["XGBoost","{:.2f}%".format(xgb_acc_score),"{:.2f}%".format(xgb_f1_score)],
]
print("Results: ")
print(tabulate(results, headers=header, tablefmt="outline"))
## CONVOLUTIONAL NEURAL NETWORK MODEL
### SCALE THE FEATURES
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
data['Amount'] = scaler.fit_transform(data['Amount'].values.reshape(-1, 1))
data['Time'] = scaler.fit_transform(data['Time'].values.reshape(-1, 1))
### Reshape the data for CNN
X_train_cnn = (X_train - X_train.mean()) / X_train.std()
X_test_cnn = (X_test - X_train.mean()) / X_train.std()

X_train_cnn = X_train_cnn.values.reshape(-1, 30, 1)
```

```
X_test_cnn = X_test_cnn.values.reshape(-1, 30, 1)
### Balancing for the CNN 14
# Separate the fraud and non-fraud examples
fraud = data[data['Class'] == 1]
non_fraud = data[data['Class'] == 0]

# RANDOM UNDER-SAMPLING OF THE MAJORITY CLASS
non_fraud_downsampled = resample(non_fraud, replace=False, n_samples=len(fraud),
random_state=42)

# COMBINE MINORITY CLASS WITH DOWNSAMPLED MAJORITY CLASS
balanced_data = pd.concat([fraud, non_fraud_downsampled])

# SHUFFLE THE EXAMPLES
balanced_data = shuffle(balanced_data, random_state=42)
### SPLITTING THE BALANCED DATASET
X_bal = balanced_data.iloc[:, :-1]
y_bal = balanced_data.iloc[:, -1]
X_train_bal, X_test_bal, Y_train_bal, Y_test_bal = train_test_split(X_bal, y_bal, test_size=0.2,
random_state=42)
sns.countplot(x='Class',data=balanced_data)
### RESHAPING FOR THE BALANCED DATASET
X_train_cnn_bal = (X_train_bal - X_train_bal.mean()) / X_train_bal.std()
X_test_cnn_bal = (X_test_bal - X_train_bal.mean()) / X_train_bal.std()

X_train_cnn_bal = X_train_cnn_bal.values.reshape(-1, 30, 1)
X_test_cnn_bal = X_test_cnn_bal.values.reshape(-1, 30, 1)
### DEFINING THE CNN MODELS
#### MODEL WITH 5 LAYERS
print(X_train_cnn.shape[1])
model_5 = keras.Sequential([
    keras.layers.Conv1D(filters=32, kernel_size=3, activation='relu',
input_shape=(X_train_cnn.shape[1], 1)),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dense(units=1, activation='sigmoid')
])
model_5.summary()
#### MODEL WITH 11 LAYERS
model_11 = keras.models.Sequential([
    keras.layers.Reshape((30, 1), input_shape=(X_train_cnn.shape[1],1)),
```

```
    keras.layers.Conv1D(filters=32, kernel_size=2, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv1D(filters=64, kernel_size=2, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv1D(filters=64, kernel_size=2, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(100, activation='relu'),
    keras.layers.Dense(50, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
model_11.summary()
#### MODEL WITH 13 LAYERS
model_13 = keras.Sequential()

model_13.add(Conv1D(filters=32, kernel_size=2, activation='relu',
input_shape=(X_train_cnn.shape[1], 1)))
model_13.add(BatchNormalization())
model_13.add(Dropout(0.2))

model_13.add(Conv1D(filters=64, kernel_size=2, activation='relu'))
model_13.add(BatchNormalization())
model_13.add(Dropout(0.5))

model_13.add(Flatten())
model_13.add(Dense(256, activation='relu'))
model_13.add(Dropout(0.2))

model_13.add(Dense(100,activation="relu"))
model_13.add(Dense(50,activation="relu"))
model_13.add(Dense(25,activation="relu"))
model_13.add(Dense(1,activation="sigmoid"))
#### MODEL WITH 14 LAYERS
model_14 = keras.Sequential([

keras.layers.Conv1D(filters=32,kernel_size=2,activation="relu",input_shape=(X_train_cnn.shape[1],
1)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.2),

    keras.layers.Conv1D(filters=64,kernel_size=2,activation="relu"),
    keras.layers.BatchNormalization(),
```

```
    keras.layers.Dropout(0.5),

    keras.layers.Flatten(),
    keras.layers.Dense(64,activation="relu"),
    keras.layers.Dropout(0.5),

    keras.layers.Dense(100,activation="relu"),
    keras.layers.Dense(50,activation="relu"),
    keras.layers.Dense(25,activation="relu"),
    keras.layers.Dense(1,activation="sigmoid"),
])
#### MODEL WITH 14 LAYERS BALANCED DATASET
model_14_bal = keras.Sequential([

keras.layers.Conv1D(filters=32,kernel_size=2,activation="relu",input_shape=(X_train_cnn_bal.shape
[1],1)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.2),

    keras.layers.Conv1D(filters=64,kernel_size=2,activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.5),

    keras.layers.Flatten(),
    keras.layers.Dense(64,activation="relu"),
    keras.layers.Dropout(0.5),

    keras.layers.Dense(100,activation="relu"),
    keras.layers.Dense(50,activation="relu"),
    keras.layers.Dense(25,activation="relu"),
    keras.layers.Dense(1,activation="sigmoid"),
])
#### MODEL WITH 17 LAYERS
model_17 = keras.Sequential([

keras.layers.Conv1D(filters=32,kernel_size=2,activation="relu",input_shape=(X_train_cnn.shape[1],
1)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.2),

    keras.layers.Conv1D(filters=64,kernel_size=2,activation="relu"),
    keras.layers.BatchNormalization(),
```

```
    keras.layers.Dropout(0.5),

    keras.layers.Conv1D(filters=64,kernel_size=2,activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.25),

    keras.layers.Flatten(),
    keras.layers.Dense(64,activation="relu"),
    keras.layers.Dropout(0.5),

    keras.layers.Dense(100,activation="relu"),
    keras.layers.Dense(50,activation="relu"),
    keras.layers.Dense(25,activation="relu"),
    keras.layers.Dense(1,activation="sigmoid"),
])
model_17.summary()
#### MODEL WITH 20 LAYERS
model_20 = keras.Sequential([

keras.layers.Conv1D(filters=32,kernel_size=2,activation="relu",input_shape=(X_train_cnn.shape[1],
1)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.2),

    keras.layers.Conv1D(filters=64,kernel_size=2,activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.5),

    keras.layers.Conv1D(filters=64,kernel_size=2,activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.5),

    keras.layers.Conv1D(filters=64,kernel_size=2,activation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.25),

    keras.layers.Flatten(),
    keras.layers.Dense(64,activation="relu"),
    keras.layers.Dropout(0.5),

    keras.layers.Dense(100,activation="relu"),
```

```
    keras.layers.Dense(50,activation="relu"),
    keras.layers.Dense(25,activation="relu"),
    keras.layers.Dense(1,activation="sigmoid"),
])
model_20.summary()
### COMPILING AND FITTING
#### 5 LAYERED CNN
with os.fdopen(os.open("Saved Models\\CNNHistory_5.joblib", os.O_WRONLY | os.O_CREAT)) as
file:
    if os.path.getsize("Saved Models\\CNNHistory_5.joblib") == 0:
        model_5.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        history_5 = model_5.fit(X_train_cnn, Y_train, epochs=20, validation_split=0.2, batch_size=64)
        joblib.dump(history_5,filename="Saved Models\\CNNHistory_5.joblib")
    else:
        history_5 = joblib.load("Saved Models\\CNNHistory_5.joblib")
#### 11 LAYERED CNN
with os.fdopen(os.open("Saved Models\\CNNHistory_11.joblib", os.O_WRONLY | os.O_CREAT))
as file:
    if os.path.getsize("Saved Models\\CNNHistory_11.joblib") == 0:
        model_11.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        history_11 = model_11.fit(X_train, Y_train, epochs=20, batch_size=64, validation_split=0.2)
        joblib.dump(history_11,filename="Saved Models\\CNNHistory_11.joblib")
    else:
        history_11 = joblib.load("Saved Models\\CNNHistory_11.joblib")
#### 13 LAYERED CNN
with os.fdopen(os.open("Saved Models\\CNNHistory_13.joblib", os.O_WRONLY | os.O_CREAT))
as file:
    if os.path.getsize("Saved Models\\CNNHistory_13.joblib") == 0:
        model_13.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
        history_13 = model_13.fit(X_train, Y_train, epochs=50, batch_size=64, validation_split=0.2)
        joblib.dump(history_13,filename="Saved Models\\CNNHistory_13.joblib")
    else:
        history_13 = joblib.load("Saved Models\\CNNHistory_13.joblib")
#### 14 LAYERED CNN
with os.fdopen(os.open("Saved Models\\CNNHistory_14.joblib", os.O_WRONLY | os.O_CREAT))
as file:
    if os.path.getsize("Saved Models\\CNNHistory_14.joblib") == 0:
        model_14.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        history_14 = model_14.fit(X_train_cnn, Y_train, epochs=100,batch_size=64,
validation_split=0.2)
        joblib.dump(history_14,filename="Saved Models\\CNNHistory_14.joblib")
    else:
```

```
    history_14 = joblib.load("Saved Models\\CNNHistory_14.joblib")
#### 14 LAYERED BALANCED DATASET CNN
with os.fdopen(os.open("Saved Models\\CNNHistory_14_Balanced.joblib", os.O_WRONLY |
os.O_CREAT)) as file:
  if os.path.getsize("Saved Models\\CNNHistory_14_Balanced.joblib") == 0:
    model_14_bal.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    history_14_bal = model_14_bal.fit(X_train_cnn_bal, Y_train_bal, epochs=100,batch_size=64,
validation_split=0.2)
    joblib.dump(history_14,filename="Saved Models\\CNNHistory_14_Balanced.joblib")
  else:
    history_14_bal = joblib.load("Saved Models\\CNNHistory_14_Balanced.joblib")
#### 17 LAYERED CNN
with os.fdopen(os.open("Saved Models\\CNNHistory_17.joblib", os.O_WRONLY | os.O_CREAT))
as file:
  if os.path.getsize("Saved Models\\CNNHistory_17.joblib") == 0:
    model_17.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    history_17 = model_17.fit(X_train_cnn, Y_train, epochs=100,
validation_split=0.2,batch_size=64)
    joblib.dump(history_17,filename="Saved Models\\CNNHistory_17.joblib")
  else:
    history_17 = joblib.load("Saved Models\\CNNHistory_17.joblib")
#### 20 LAYERED CNN
with os.fdopen(os.open("Saved Models\\CNNHistory_20.joblib", os.O_WRONLY | os.O_CREAT))
as file:
  if os.path.getsize("Saved Models\\CNNHistory_20.joblib") == 0:
    model_20.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    history_20 = model_20.fit(X_train_cnn, Y_train, epochs=100, validation_split=0.2,
batch_size=64)
    joblib.dump(history_20,filename="Saved Models\\CNNHistory_20.joblib")
  else:
    history_20 = joblib.load("Saved Models\\CNNHistory_20.joblib")
### PLOTTING THE TRAINING AND VALIDATION ACCURACY
#### 5 LAYERED CNN
##### EPOCH - ACCURACY
plt.plot(history_5.history['accuracy'])
plt.plot(history_5.history['val_accuracy'])
plt.title('CNN_5 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
##### EPOCH - LOSS
```

```
plt.plot(history_5.history['loss'])
plt.plot(history_5.history['val_loss'])
plt.title('CNN_5 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
#### 11 LAYERED CNN
##### EPOCH - ACCURACY
plt.plot(history_11.history['accuracy'])
plt.plot(history_11.history['val_accuracy'])
plt.title('CNN_11 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
##### EPOCH - LOSS
plt.plot(history_11.history['loss'])
plt.plot(history_11.history['val_loss'])
plt.title('CNN_11 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
#### 13 LAYERED CNN
##### EPOCH - ACCURACY
plt.plot(history_13.history['accuracy'])
plt.plot(history_13.history['val_accuracy'])
plt.title('CNN_13 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
##### EPOCH - LOSS
plt.plot(history_13.history['loss'])
plt.plot(history_13.history['val_loss'])
plt.title('CNN_13 Model loss')
plt.ylabel('loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

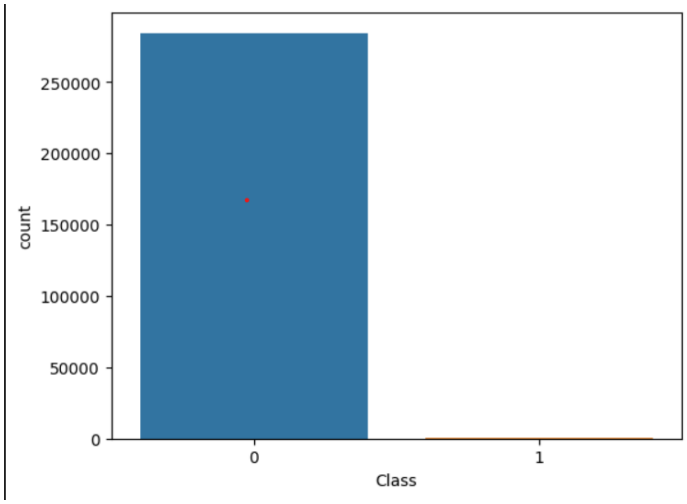
```
#### 14 LAYERED CNN
##### EPOCH - ACCURACY
plt.plot(history_14.history['accuracy'])
plt.plot(history_14.history['val_accuracy'])
plt.title('CNN_14 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
##### EPOCH - LOSS
plt.plot(history_14.history['loss'])
plt.plot(history_14.history['val_loss'])
plt.title('CNN_14 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
#### 14 LAYERED CNN (BALANCED DATASET)
##### EPOCH - ACCURACY
plt.plot(history_14_bal.history['accuracy'])
plt.plot(history_14_bal.history['val_accuracy'])
plt.title('CNN_14 (Balanced) Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
##### EPOCH - LOSS
plt.plot(history_14_bal.history['loss'])
plt.plot(history_14_bal.history['val_loss'])
plt.title('CNN_14 (Balanced) Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
#### 17 LAYERED CNN
##### EPOCH - ACCURACY
plt.plot(history_17.history['accuracy'])
plt.plot(history_17.history['val_accuracy'])
plt.title('CNN_17 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
```

```
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
##### EPOCH - LOSS
plt.plot(history_17.history['loss'])
plt.plot(history_17.history['val_loss'])
plt.title('CNN_17 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
#### 20 LAYERED CNN
##### EPOCH - ACCURACY
plt.plot(history_20.history['accuracy'])
plt.plot(history_20.history['val_accuracy'])
plt.title('CNN_20 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
##### EPOCH - LOSS
plt.plot(history_20.history['loss'])
plt.plot(history_20.history['val_loss'])
plt.title('CNN_20 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
### EVALUATING THE CNN LAYERS
#### CALCULATING THE ACCURACY SCORES
train_acc_5 = max(history_5.history['accuracy'])*100
val_acc_5 = max(history_5.history['val_accuracy'])*100

train_acc_11 = max(history_11.history['accuracy'])*100
val_acc_11 = max(history_11.history['val_accuracy'])*100

train_acc_13 = max(history_13.history['accuracy'])*100
val_acc_13 = max(history_13.history['val_accuracy'])*100

train_acc_14 = max(history_14.history['accuracy'])*100
val_acc_14 = max(history_14.history['val_accuracy'])*100
```

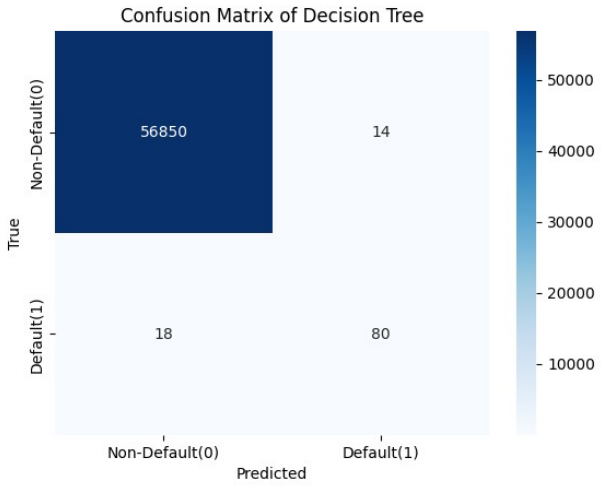
```
train_acc_14_bal = max(history_14_bal.history['accuracy'])*100
val_acc_14_bal = max(history_14_bal.history['val_accuracy'])*100

train_acc_17 = max(history_17.history['accuracy'])*100
val_acc_17 = max(history_17.history['val_accuracy'])*100

train_acc_20 = max(history_20.history['accuracy'])*100
val_acc_20 = max(history_20.history['val_accuracy'])*100
#### RESULTS
header = ["Model","Train Accuracy","Validation Accuracy"]
results = [
   ["5 Layered CNN","{:.6f}%".format(train_acc_5),"{:.6f}%".format(val_acc_5)],
   ["11 Layered CNN","{:.6f}%".format(train_acc_11),"{:.6f}%".format(val_acc_11)],
   ["13 Layered CNN","{:.6f}%".format(train_acc_13),"{:.6f}%".format(val_acc_13)],
   ["14 Layered CNN","{:.6f}%".format(train_acc_14),"{:.6f}%".format(val_acc_14)],
   ["14 Layered (Balanced)
CNN","{:.6f}%".format(train_acc_14_bal),"{:.6f}%".format(val_acc_14_bal)],
   ["17 Layered CNN","{:.6f}%".format(train_acc_17),"{:.6f}%".format(val_acc_17)],
   ["20 Layered CNN","{:.6f}%".format(train_acc_20),"{:.6f}%".format(val_acc_20)],
]
print("Results CNN: ")
print(tabulate(results, headers=header, tablefmt="outline"))
import gradio as gr
import numpy as np

model_results = {
   "SVM": [svm_acc_score, svm_f1_score," ", " "],
   "KNN": [knn_acc_score, knn_f1_score," ", " "],
   "Random Forest": [rf_acc_score, rf_f1_score," "," "],
   "Decision Tree": [dtc_acc_score, dtc_f1_score," " ," "],
   "Logistic Regression": [log_acc_score, log_f1_score," "," "],
   "XGBoost": [xgb_acc_score, xgb_f1_score," "," "],
   "CNN 5 Layers": [" ", " ", train_acc_5, val_acc_5],
   "CNN 11 Layers": [" ", " ", train_acc_11, val_acc_11],
   "CNN 13 Layers": [" ", " ", train_acc_13, val_acc_13],
   "CNN 14 Layers": [" ", " ", train_acc_14, val_acc_14],
   "CNN 14 Layers (Balanced Dataset)": [" ", " ", train_acc_14_bal, val_acc_14_bal],
   "CNN 17 Layers": [" ", " ", train_acc_17, val_acc_17],
   "CNN 20 Layers": [" ", " ", train_acc_20, val_acc_20]

}
```

```
def predict(model):
    return f"Accuracy (in %)  :        {model_results[model][0]} \nF1-Score  (in %):
{model_results[model][1]} \nTraining Accuracy   (in %):        {model_results[model][2]}
\nValidation Accuracy   (in %):        {model_results[model][3]} "



iface = gr.Interface(
    fn = predict,
    inputs = gr.Dropdown(
            [
                "SVM",
                "KNN",
                "Random Forest",
                "Decision Tree",
                "Logistic Regression",
                "XGBoost",
                "CNN 5 Layers",
                "CNN 11 Layers",
                "CNN 13 Layers",
                "CNN 14 Layers",
                "CNN 14 Layers (Balanced Dataset)",
                "CNN 17 Layers",
                "CNN 20 Layers"
            ], label="Model"
        ),
    outputs = "text",
)

iface.launch(share=True)
```

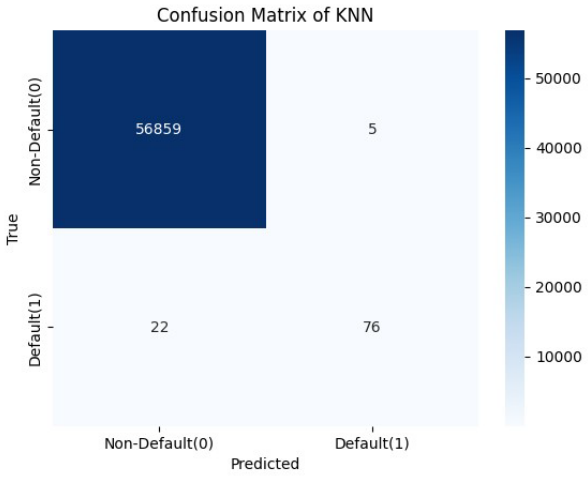
CHAPTER 4
SNAPSHOTS



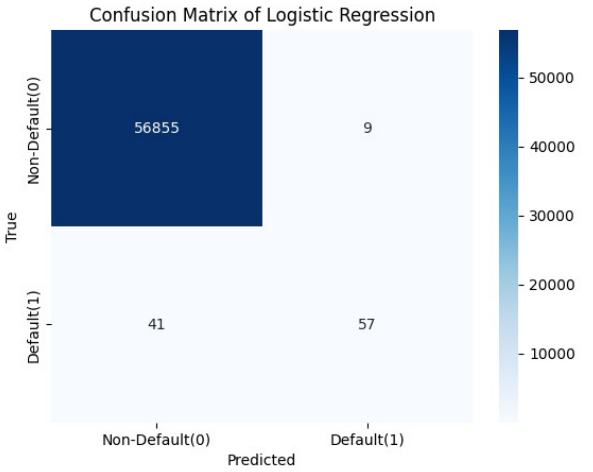
4.1 Count plot for the imbalanced dataset



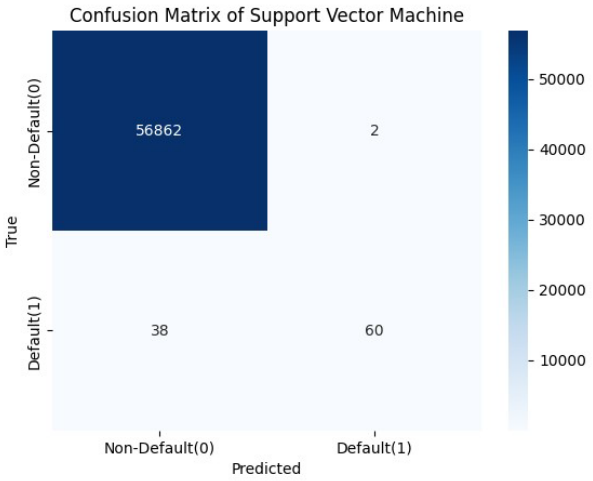
4.2. Confusion Matrix of Decision Tree



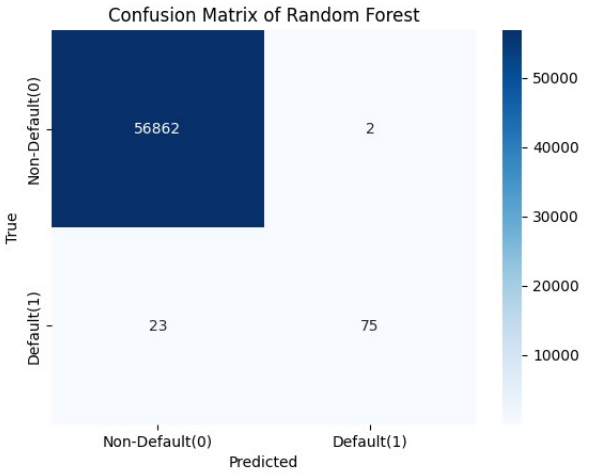
4.3. Confusion Matrix of KNN



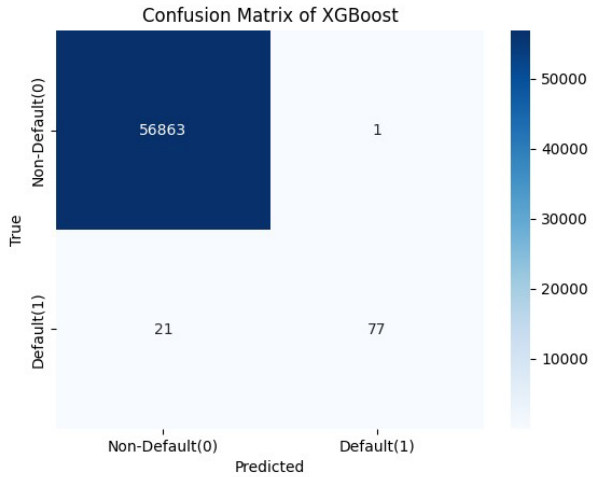
4.4. Confusion Matrix of Logistic Regression



4.5. Confusion Matrix of SVM



4.6. Confusion Matrix of Random Forest

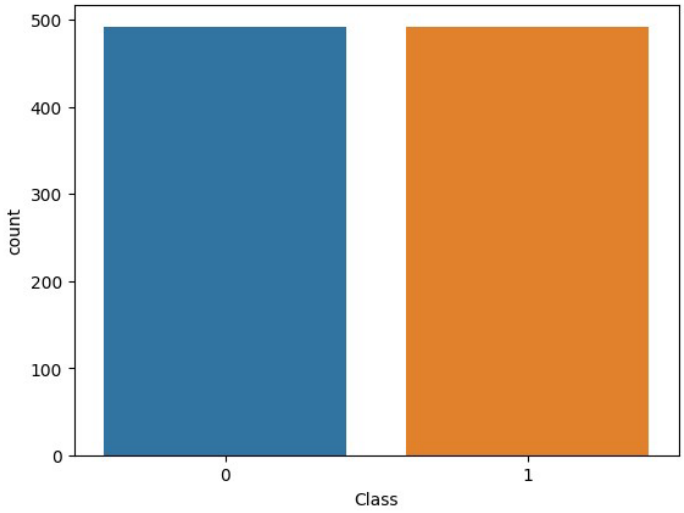


4.7. Confusion Matrix of XGBoost

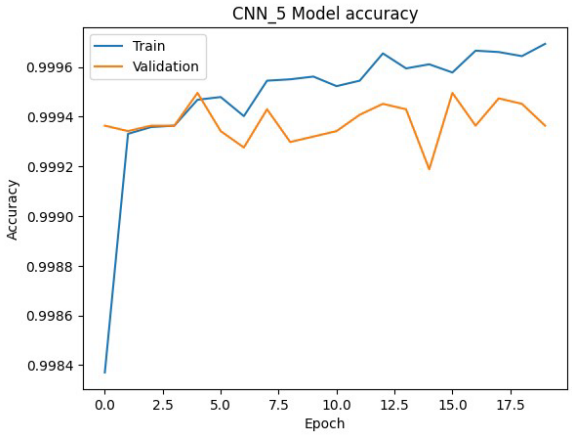
Results:

Model	Accuracy	F-1
Decicion Tree	99.94%	83.33%
KNN	99.95%	84.92%
Logistic Regression	99.91%	69.51%
SVM	99.93%	75.00%
Random Forest	99.96%	85.71%
XGBoost	99.96%	87.50%

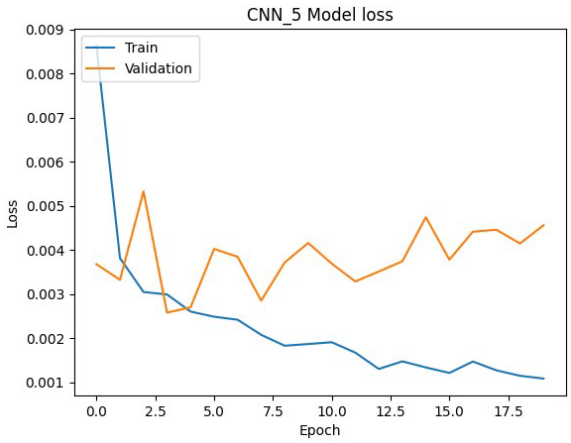
4.8. Results of Machine Learning Models



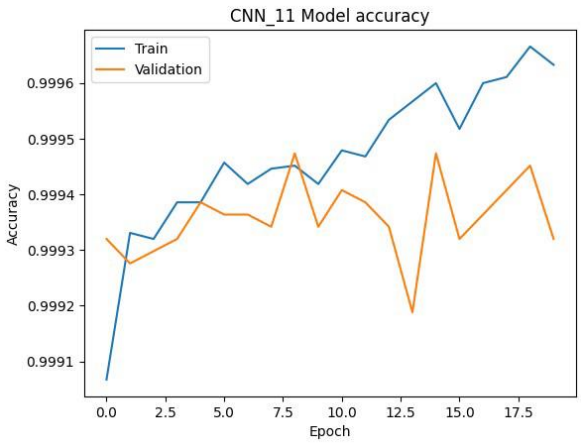
4.9 Count plot for the balanced dataset



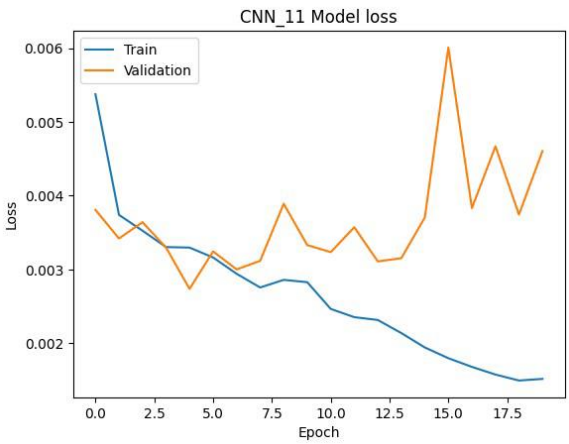
4.10 CNN 5 layered accuracy graph



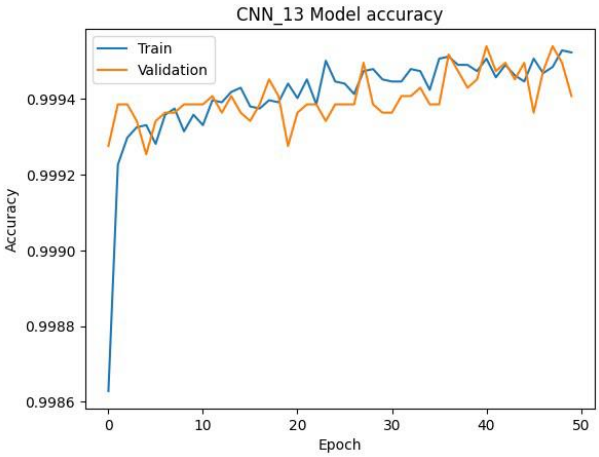
4.11 CNN 5 layered loss graph



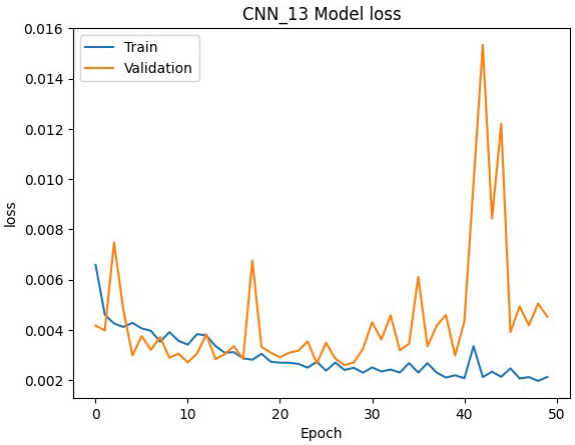
4.12 CNN 11 layered accuracy graph



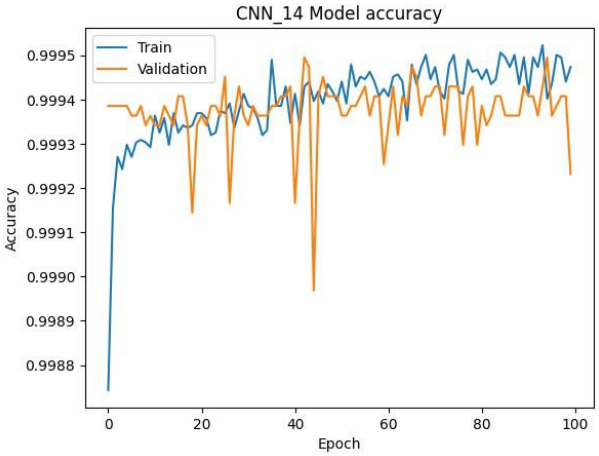
4.13 CNN 11 layered loss graph



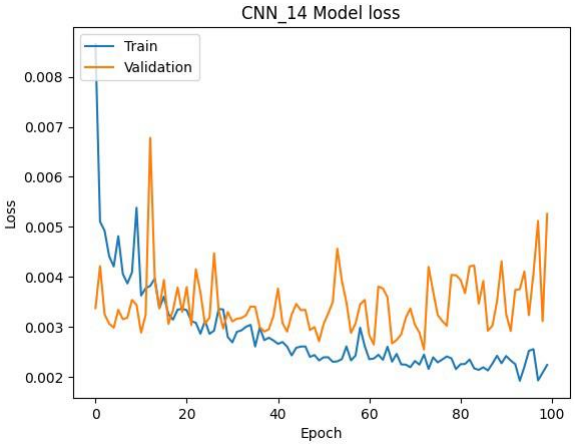
4.14 CNN 13 layered accuracy graph



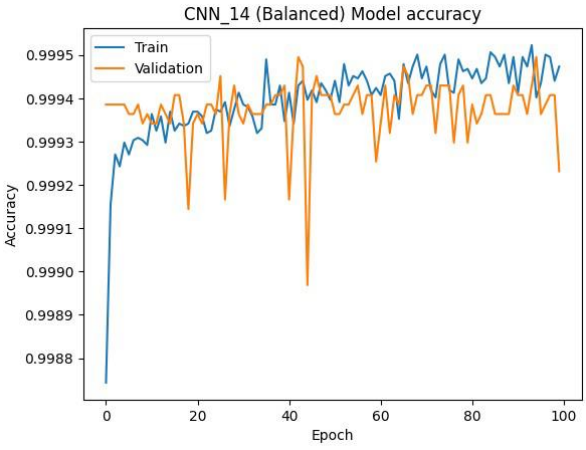
4.15 CNN 13 layered loss graph



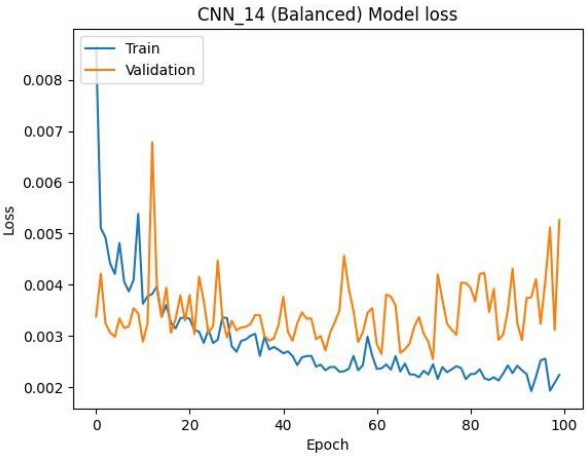
4.16 CNN 14 layered accuracy graph



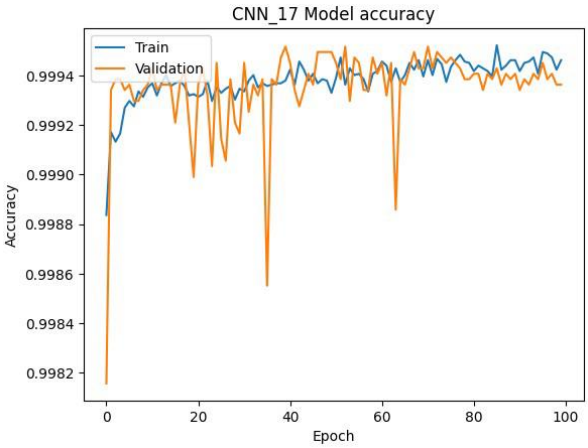
4.17 CNN 14 layered loss graph



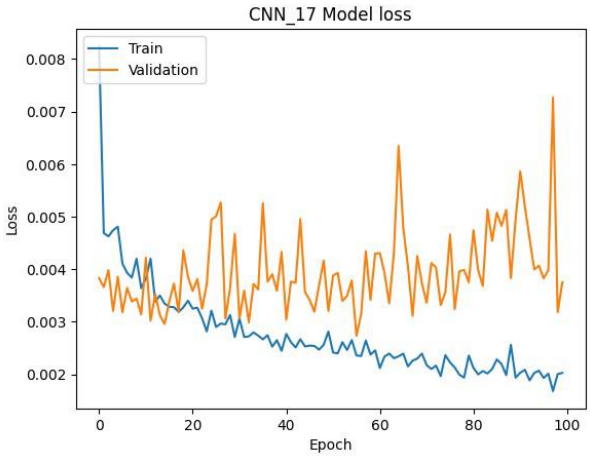
4.18 CNN 14(Balanced) layered accuracy graph



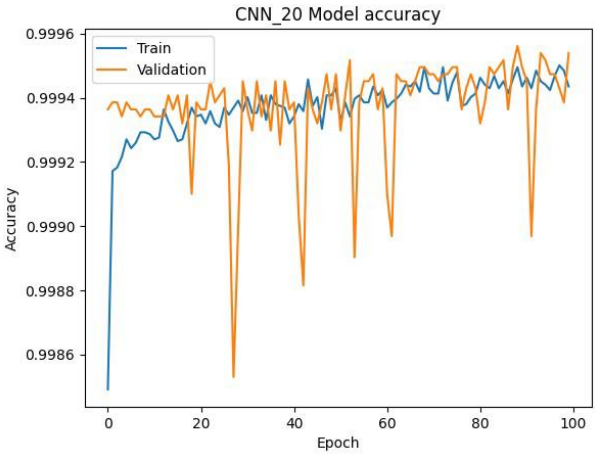
4.19 CNN 14(Balanced) layered loss graph



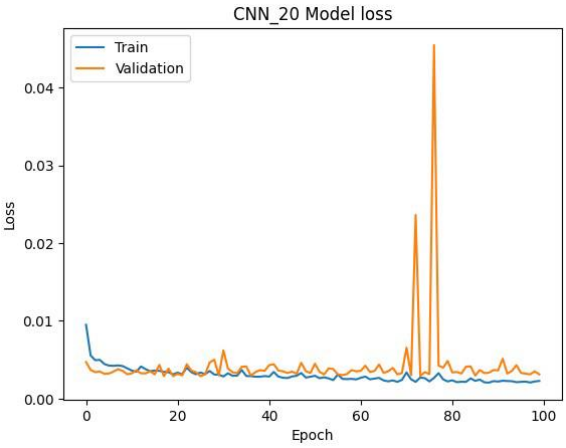
4.20 CNN 17 layered accuracy graph



4.21 CNN 17 layered loss graph



4.22 CNN 20 layered accuracy graph



4.23 CNN 20 layered loss graph

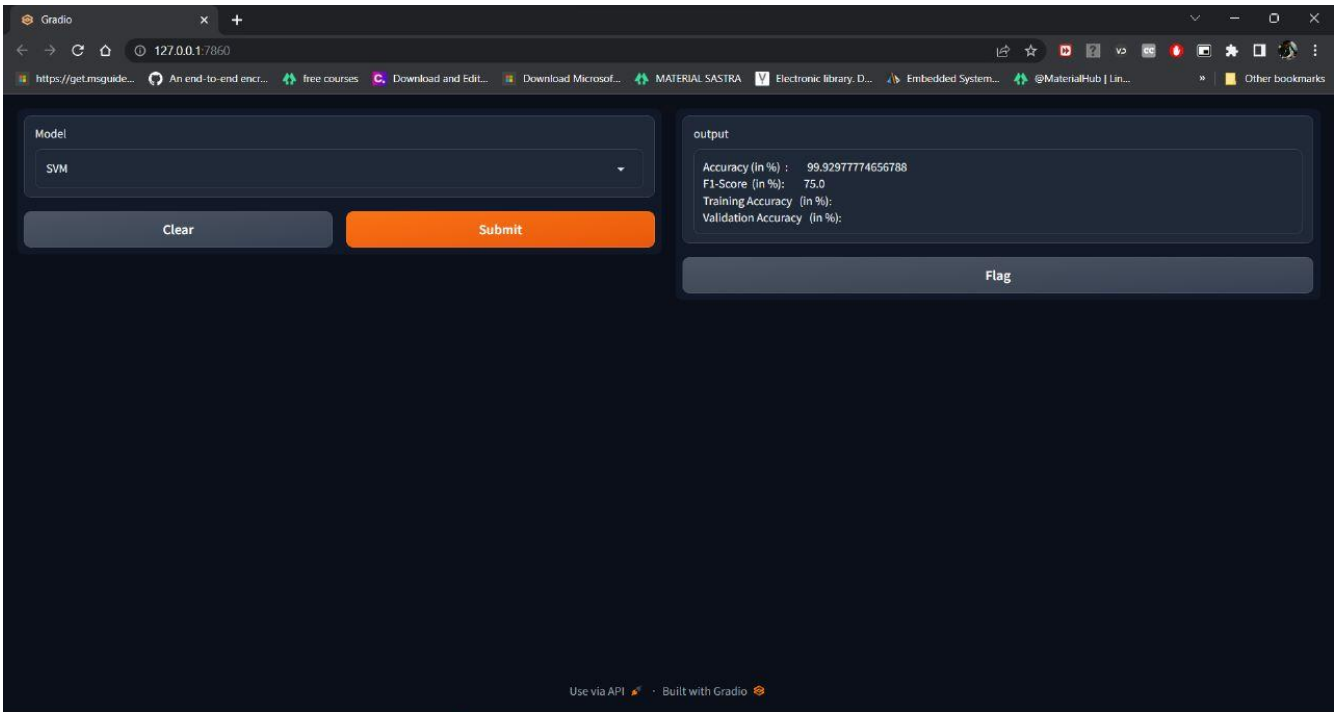
Results CNN:

Model	Train Accuracy	Validation Accuracy
5 Layered CNN	99.969280%	99.949527%
11 Layered CNN	99.966532%	99.947333%
13 Layered CNN	99.952817%	99.953914%
14 Layered CNN	99.952269%	99.949527%
14 Layered (Balanced) CNN	99.952269%	99.949527%
17 Layered CNN	99.952269%	99.951720%
20 Layered CNN	99.950075%	99.956113%

4.24 Results of CNN Models

```
Running on local URL:  http://127.0.0.1:7861
```

4.25 GUI URL



4.26 GUI

CHAPTER 5

CONCLUSION AND FUTURE WORK

Credit Card Fraud is a huge threat to Financial Institutions. A robust classifier is needed to handle this increasing threat. On analysing the results from both Machine learning models and CNNs models. The XGBoost outperforms giving an accuracy score of 99.96% and then comes the 20 Layered CNNs with the accuracy score 99.956% which is slightly closer to XGBoost Classifier. After 20 Layers even if we try to increase the number of CNNs Layers the results will be saturated and there will no difference. To improve the credit card fraud detection system, several aspects can be considered as future work which includes Transfer Learning**,** a machine learning technique where knowledge gained from training one model for a particular task and then transferring that knowledge to perform related task by another model. Usage of pre-defined models like VGG16 can be best suited for transfer learning. Transfer Learning mainly involves two steps Pre-training and Fine-tuning. Transfer learning is still in research phase so a few years is required to optimize this technique. Ensemble Learning can also be implemented with this dataset**.** Ensemble learning is extracting features from a trained model and using it in on another machine learning model is called as ensemble learning. They provide several benefits, including improved accuracy, robustness and model stability. The ability to predict a fraudulent transaction in real-time. This can be implemented by incorporating high accuracy classification models to banking systems and even in payment portal. New fraudulent patterns emerge over time. So, incorporating Continual learning techniques which helps to prevent future fraudulent transactions. In conclusion, future work must focus on enhancing the credit card fraud detection system and protect from future fraudulent transactions.

CHAPTER 6

REFERENCES

1. Y. Abakarim, M. Lahby, and A. Attioui, ``An efcient real time model for credit card fraud detection based on deep learning," in Proc. 12th Int. Conf. Intell. Systems: Theories Appl., Oct. 2018, pp. 17, doi:10.1145/3289402.3289530.

2. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

3. https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/

4. S. Makki, Z. Assaghir,Y. Taher, R. Haque, M.-S. Hacid, and H. Zeineddine, ``An experimental study with imbalanced classication approaches for credit card fraud detection," IEEE Access, vol. 7, pp. 9301093022, 2019, doi: 10.1109/ACCESS.2019.2927266.

5. https://www.javatpoint.com/logistic-regression-in-machine-learning

6. https://www.analyticsvidhya.com/blog/2021/08/decision-tree-algorithm/

7. https://www.geeksforgeeks.org/xgboost/

8. https://www.javatpoint.com/machine-learning-algorithms