Project Report

Research Topic:

Implement a fraud detection classifier using deep learning models by ensemble method and evaluation with available solutions.

Abstract:

Credit cards become the most common popular payment method for both regular and online transactions because of recent advancements in electronic commerce systems and communication technologies; thus, there is significantly increased fraud associated with such transactions. Every year, fraudulent credit card transactions cost businesses and customers a significant amount of money, and fraudsters are constantly looking for new technology and methods to commit criminal acts. The identification of fraudulent transactions has become a major factor affecting the greater utilization of electronic payment. As a result, efficient and effective methods for detecting fraud in credit card transactions are demanded. This paper proposes an ensemble method based on majority voting(Hard Voting Classifier) of deep learning models to detect fraud transaction. Artificial neural network(ANN), convolution neural network(CNN), recurrent neural network(RNN), long short term memory(LSTM) and gated recurrent units(GRU) has been used as deep learning models. A popular European dataset called "credit card fraud detection" has been used for research. The maximum performance comes with 99.89 percent accuracy after implementing multiple runs of the suggested classifier.

Table of Contents

List of Figures Page

Appendix A – Python Code

1.1 Introduction

Credit card fraud has always been a financial problem ever since they were first introduced in the 1950s. This is especially prevalent today as an increased amount of shopping is done online. This was only exacerbated by the COVID-19 pandemic where families were advised to stay at home to protect themselves from large gatherings such as that in retail stores. Also, using credit cards in person is incentivized by banks to build the individual's credit score. In general, credit card usage has overall increased over the years. This has resulted in millions of dollars worth of transactions occurring everyday in what is now likely a trillion dollar industry.

Intuitively, it makes sense that as more transactions occur and more credit cards are in circulation, the odds of credit card fraud will increase. Credit card fraud is presently one of the most frequently occurring problems globally as there has not been a permanent fix. This is despite numerous attempts, research papers and proposed solutions. As of 2019, credit card fraud accounts for $28.65 billion dollars worth of transactions. In addition, though credit card fraud had been steadily increasing for years, but it has exploded since the pandemic began in 2019, as seen by a 72.4% increase in reports from 2018(Identity theft and credit card fraud statistics, 2020).

There mainly two types of credit card fraud. Application fraud (Phua and Gayler, 2009) and behavior fraud (Bolton and Hand, 2002). Fraudulent credit card applications are referred to as application fraud. When a fraudster uses fake personal details to start a new credit card process, the issuer accepts the request. After a credit card is correctly issued, behavior fraud happens, and it denotes credit card transactions that contain fraudulent activity. It is essential for financial institutions to deploy an effective credit card fraud detection system, as even a tiny bit of fraud detection can save a lot of money in the long run.

The data distribution evolves over time as the fraudster employs new approaches and methods. Credit card fraud detection has become a difficult task for data scientists due to the limited number of fraudulent transactions that occur on a regular basis (Dal and Boracchi,2017).

This paper proposes an intelligent approach for detecting fraudulent credit card transactions that uses hard voting classifier of five deep learning models. Artificial neural network(ANN), convolution neural network(CNN), recurrent neural network(RNN), long short term memory(LSTM) and gated recurrent

units(GRU) has been used as deep learning models. Traditional five machine learning classifier logistic regression, support vector classifier, decision tree, random forest and k nearest neighbor has also been used for comparative analysis of the performance.

2.1 Literature Review:

Several research works have proposed different methods in tackling credit card fraud.From Regression, Random Forest and KNN to Neural networks. Ghosh et al. were the first to apply neural networks for fraud detection (1994). They used a large sample of labeled credit card transactions to train half a dozen neural networks, which was then validated on validation data consisting of account activities over a two-month period. Lost or stolen cards, application fraud, counterfeit fraud, mail-order fraud, and NRI (non-received issue) fraud were all utilized to train the neural network (Misra et al., 2020). Brause et al. (1999) used association rules mining and neural networks to reduce the false positive rate in their study. Several supervised and unsupervised machine learning and optimization algorithms have been used to detect credit card fraud in recent years.

Fraud detection is primarily a binary classification problem which can be solved by using supervised learning techniques. It's a frequent learning method that requires a dataset with "fraud" and "non-fraud" labels, as well as the training of a classifier. One of the benefits of employing supervised learning is that all of the class outputs altered by the algorithm are relevant to humans, and it can be utilized for discriminative patterns with ease (Abdallah et al., 2016). Unsupervised learning, on the other hand, looks at datasets that aren't labeled. It gathers information from fresh transactions and searches for unusual patterns among them. Many methods are used in both supervised and unsupervised learning, including classification techniques such as artificial neural networks, K-nearest neighbors, Decision trees, logistic regression, Naive–Bayes, and the support vector machine (SVM) methodology (Bolton et al, 2002)

Esenogho *et al* proposed an efficient approach to detect credit card fraud using a neural network ensemble classifier and a hybrid data resampling method. Where they used a LSTM network as a base learner in the adaptive boosting (AdaBoost) technique and the hybrid resampling is achieved using the synthetic minority oversampling technique and edited the nearest neighbor (SMOTE-ENN) method. This method is significant for two reasons: the LSTM is a robust algorithm for modeling sequential data. Secondly, the AdaBoost technique builds strong classifiers that are less likely to overfit, with lesser false-positive predictions. This proved to be a sufficient approach to detecting credit card fraud.

Ileberi *et al* investigated the use of the Synthetic Minority Over-sampling Technique (SMOTE) for oversampling and Machine learning algorithms Support Vector Machine (SVM), Random Forest (RF), Extra Tree (ET), Extreme Gradient Boosting (XGBoost), Logistic Regression (LR), and Decision Tree (DT) were evaluated individually. Additionally, the Adaptive Boosting (AdaBoost) algorithm was paired with each methods to increase their robustness. The outcome being that using the AdaBoost algorithm has a positive impact on the proposed ML methods. Moreover, the framework proposed in this research was validated using a highly skewed synthetic credit card fraud dataset and the results were optimal. Where the mimment across all classification metrics was 97%.

Makki, *et al* had their solution presented and discussed in two phases. In the first phase, eight classification the most suitable algorithms including the following: C5.0, SVM and ANN.
In the second phase, the selected algorithms are used in comparing selected imbalance classification approaches such as Random Oversampling, One-Class Classification and Cost Sensitive. All algorithms the accuracies are higher than 90% with different sensitivity and AUPRC values. Hoever, they found that the LR, C5.0 decision tree algorithm, SVM and ANN are the best methods according to the 3 considered performance measures (Accuracy, Sensitivity and AUPRC).

Tingfei,, *et al* examined three different oversampling models used to detect credit card fraud: SMOTE, GAN, and VAE. In addition, the baseline used is not a classical machine learning method, but a deep learning method. They saw that the use of oversampling methods to differentially increase the number of positive cases does have different degrees of impact on the classifier. The recall rate of the SMOTE and GAN methods has increased by 0.02 and 0.03 respectively, reaching 0.85 and 0.86, which is a significant improvement compared to the baseline of 0.83.

Carcillo et al. used a hybrid technique to broaden the set of features in the fraud detection classifier by using unsupervised outlier scores. Their key contribution was to implement and evaluate different levels of granularity for defining outlier scores. Their findings show that their recommended strategy is effective and improves detection accuracy.

Carcillo et al. have also presented the SCAlable Real-time Fraud Finder (SCARFF), a machine learning method that addresses nonstationarity, imbalance, and feedback latency by incorporating big-data tools (Cassandra, Kafka, and Spark) into a machine learning method. Experiments using a huge data set of real credit card transactions showed that the system is effective, accurate, and scalable.

3.1 Dataset:

Credit card fraud dataset has been collected from research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. The dataset has been downloaded from Credit Card Fraud Detection | Kaggle.

It contains 284807 credit card transactions of two days based on European card holder for September, 2013, whereas 492 frauds has been counted. So, the dataset is highly imbalanced, where as 0.172% of fraud class has supervised in all transaction. The dataset contains 31 attributes which represents.

No	Columns	Data Type	Description
1	Time	Numeric	'Time' represents the seconds elapsed between each transaction and the first transaction in the dataset.
2…29	V1,V2…..V28	Numeric	V1 to V28 are transformed with Principal Component Analysis. For the security purpose attribute name is not mentioned.
30	Amount	Numeric	The transaction amount.
31	Class	Numeric	Binary classification, '1' means fraud detected otherwise '0'.

4.1 Initial Analysis:

Attributes properties and characteristics are reviewed carefully in initial analysis stages. Distribution of variable, correlation dependencies data driven story were the main focus on this stage. We divided the data analysis into three focal criteria. They are describing in subsection.

4.1.1 Univariate Analysis:

After checking null values and data types, 1081 duplicate observations has been deleted.

From the target variable 'Class' count plot, a huge data imbalance was detected. So that, you can't even see the amount of credit card fraud. There are 284315 instances of no credit card fraud and 492 instances of credit card fraud. That is less than 1% of the entire dataset. Later this issue will be resolved by oversampling.

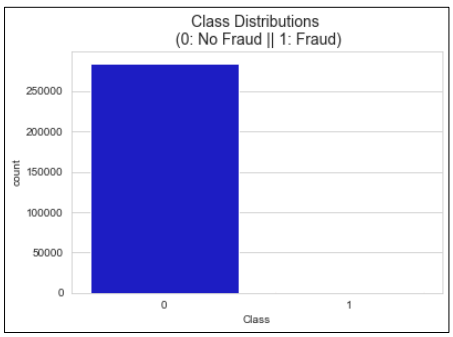

Figure 1: Distribution of Class variable

Next we emphasized on outlier detection. An outlier is a value in a random sampling from a population that deviates abnormally from other values. we used Box Plot to detect the outliers of 'Amount' features in our dataset, where any point above or below the whiskers represents an outlier. As Amount is the only numeric feature that has meaning.

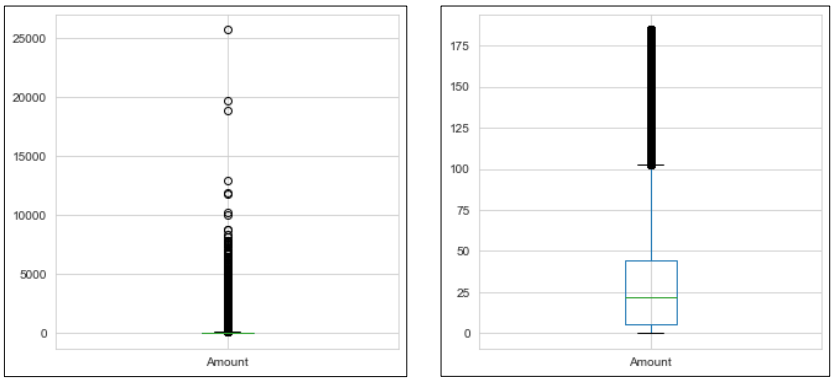

Figure 2: Anomaly detection and transform to interquartile range

Amount value has plotted vertically and it has been clear that the values mostly concentrate smaller amount. There are very few big amount. If we don't deal with this, then the detection model could be biased significantly during prediction.

We will apply Median Imputation to deal with outliers after we've detected them. The extreme numbers are replaced by median values in this strategy.

IQR = Q3-Q1 is the formula to calculate it.

For each of the variables in the dataset feature "Amount", the lines of code below calculate and report the interquartile range. Repeating the same graph above after dealing with the anomalies results in Amount staying in the range of approximately $100 to $200.

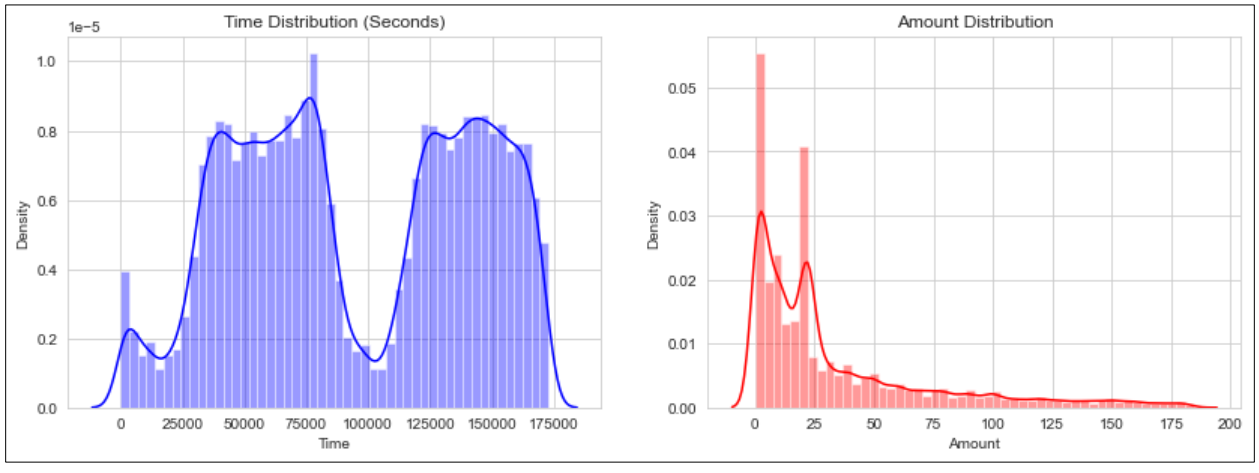

Figure 3: 'Time' and 'Amount distribution plot

From the distribution of ‘Time’ and ‘Amount’, there were no notable things found. On the contrary, it is clear that very small amount close to ‘0’ has the highest concentration; whereas day time has high density for transaction. Here time is plotted for two consecutive days in seconds. So, any amount divide by ‘3600’ will give hour time.

4.1.2 Bivariate Analysis:

We plotted correlation heatmap to review 2D correlation matrix between two discrete dimensions. In our case all the 31 attributes has considered in each dimension. The positive correlation ‘+1’ to negative correlation around ‘-0.5’ has been plotted by two color ‘red’ and ‘blue’. In this figure ‘red’ describe positive correlation whereas ‘blue’ depicts negative. The stronger the color the larger the magnitude. Though the values are also plotted here.

It has been found out that column V1 to V28 has no correlation between each other. On the other hand, these attributes has no meaning full name for security purpose, so the descriptive statistics among then won’t make valuable sense.

The above image looks messy due to how big it is (it is readable in the notebook file). The key takeaways are that Time/V3 = -0.42, Amount/V2 = -0.53, Amount/V5 = -0.39, Amount/V7 = 0.4, Amount/V20 = 0.34. Other minor corrleations ranging from -0.3 to 0.3 can be found but are not significant.

The image looks chaotic due to how big it is (it is readable in the notebook file). The key takeaways about the negative magnitude comes out are ‘Time/V3 = -0.42’, ‘Amount/V2 = -0.53’ and

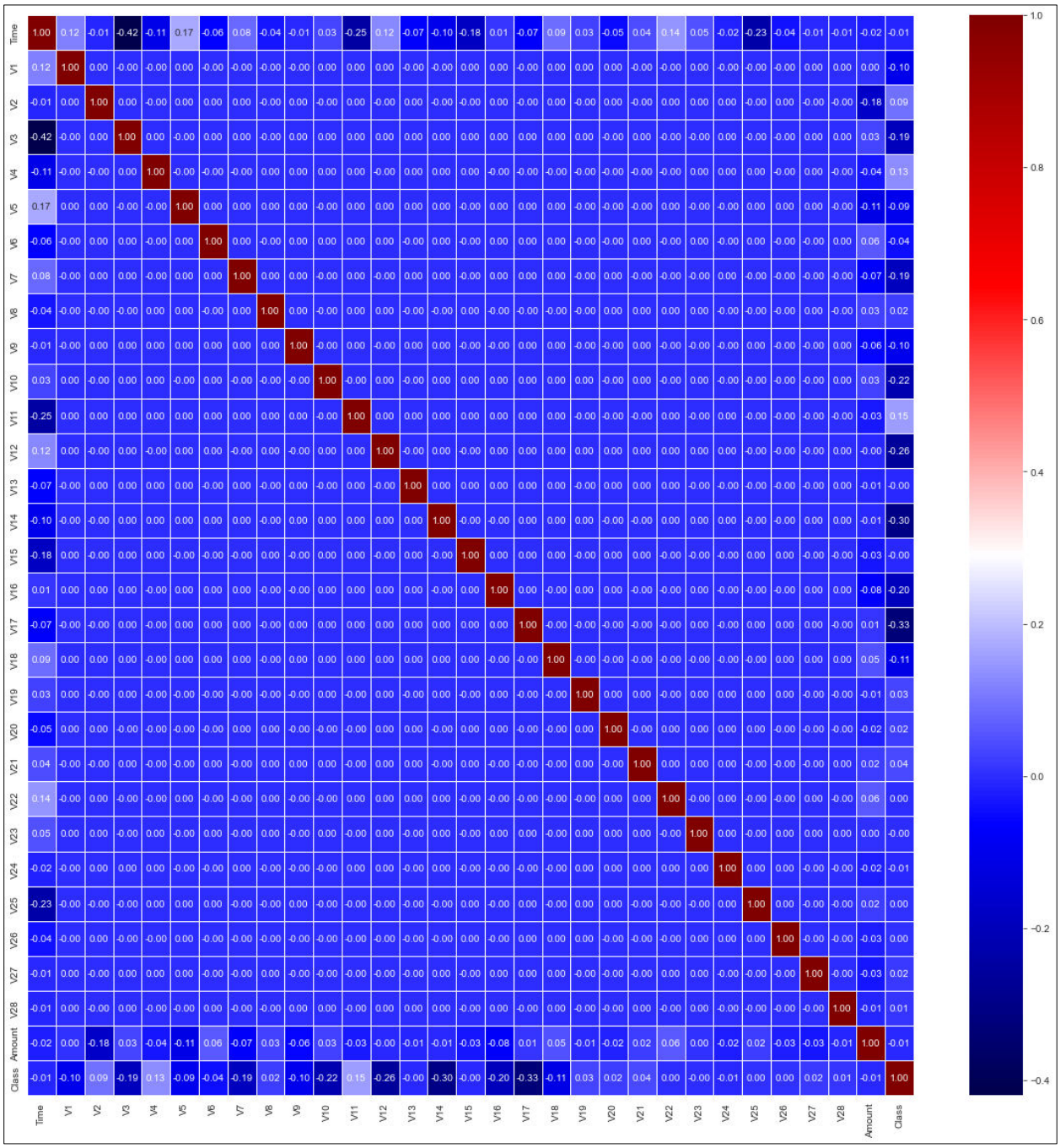

Figure 4: Correlation heatmap of creditcard dataset

'Amount/V5 = -0.39'; furthermore, 'Amount/V7 = 0.4' and 'Amount/V20 = 0.34' are notable in positive scale. Other minor correlations ranging from -0.3 to 0.3 can be found but are not so significant.

We can conclude that 'Time' and 'Amount' are the most important variables.

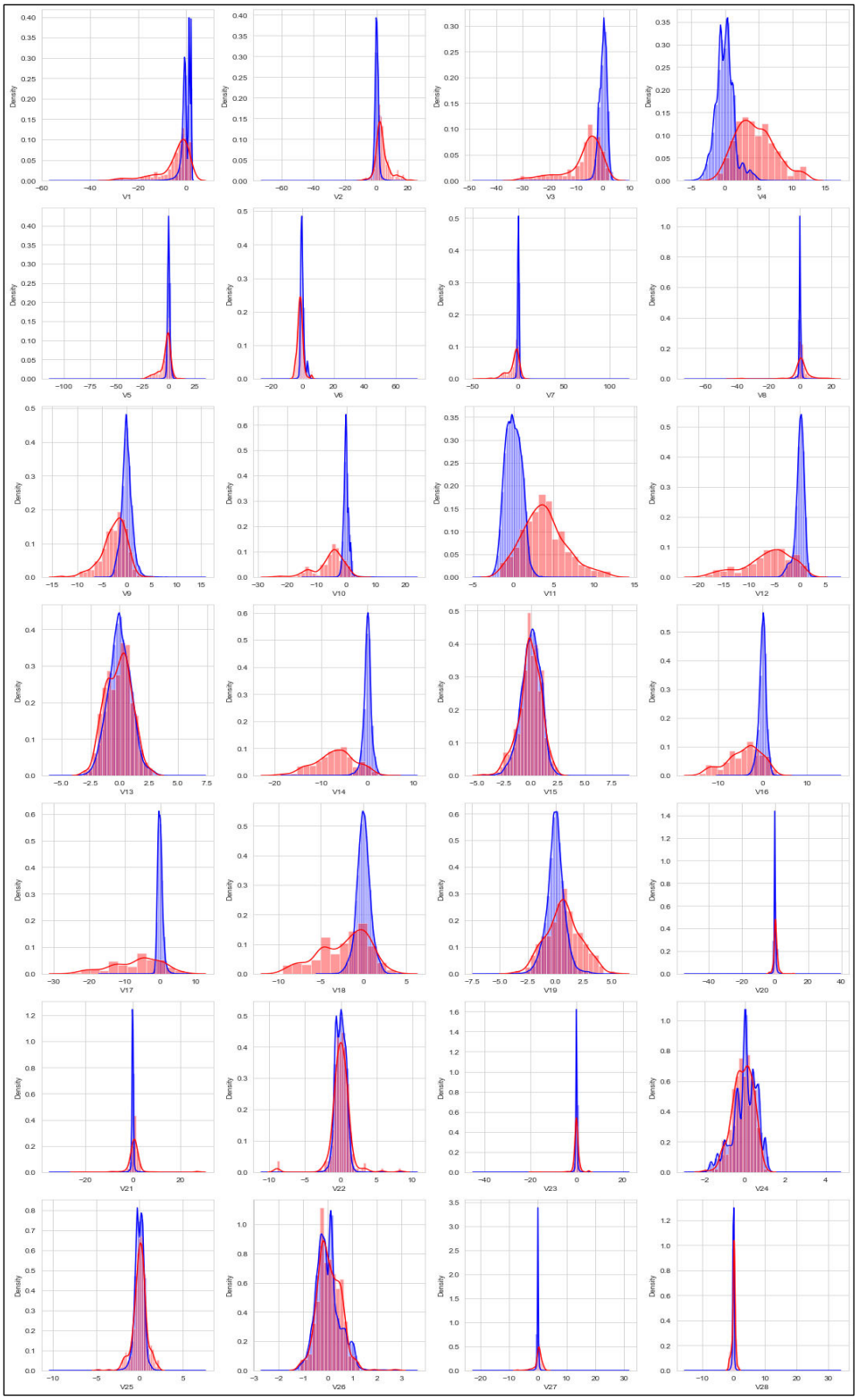

Figure 5: Class distribution in terms from V1 to V28 features

Next, we plot the distribution of the data from V1 to V28 pairing by two classes, that are 'Genuine' and 'Fraud'. The density has shown in vertical axis where as each attributes plotted in horizontal axis. Each row contains four features and seven rows contain total 28 features. The aim is to understand how the distribution varies for two classes by values of these features.
Distributions for the both classes are like Gaussian bell curve. Here, V3, V9, V10, V12, V14, V16, V17 and V18 values for fraud has higher probability of negative or lower values than other class. On the contrary, V4 and V11 has opposite story. The other features are most likely identical. If the feature has proper title then the findings would be very much valuable.

Next, we plotted 'Amount' distribution over two classes to derive the genuine and fraud amount value pattern. It has been clearly shown that fraud transaction occurs on very small amount. The reason might be fraudster want to be unnoticed to account holder or financial institution. In general, people negate very tiny amount transaction from their account. They thought it could be bank/account/interest charge. It is an art of forgery from fraudster.

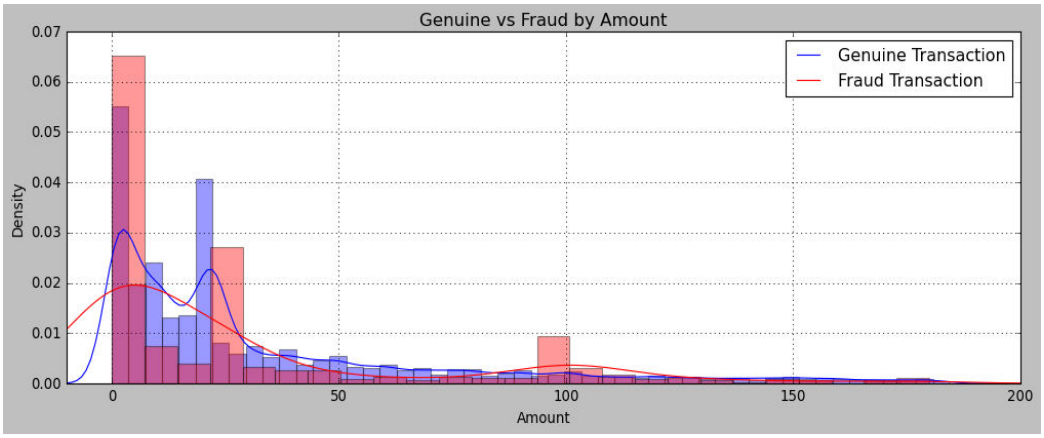

Figure 6: 'Amount' distribution based on 'Class' label

Furthermore, we noticed that transactions for genuine and fraud are alike. That means, we can't predict its class based on amount value. The amount values are previously modified by anomaly reduction.

Moreover, plotting the two class against 'Time' variable gives us this graph. The both class distributed look alike over vertical 'Time' axis. As 'Time' depicts the elapsed seconds from each transaction to first transaction and the dataset has been made based two consecutive day transaction so we can develop time slot in a way that each day contains 86400 seconds, each hours can be plotted with by (86400/(60 x 60)). Moreover, value of Time axis can be computed in hours as well. For instance for value '50000' the

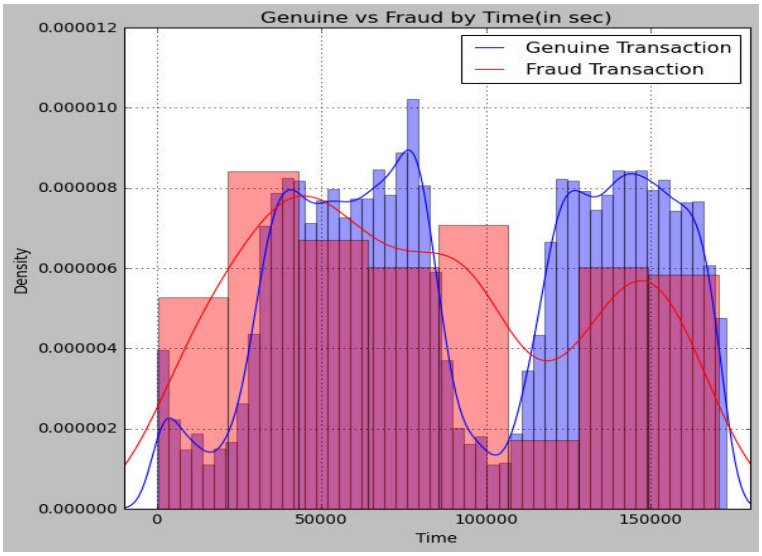

Figure 7: 'Class' distribution in terms of 'Time'

hours would be around 13.00, that means 1.00 pm. So, we can say that frequency of transaction are higher around 12.00 pm.

4.1.3 Multivariate Analysis:

To better understand the current class imbalance, the class instances have been plotted as a scatter graph with the genuine instances. Minority class has plotted with 5 times more weight for looking bigger. 'Amount' has plotted vertically and 'Time' horizontally. 'blue' dots resets genuine transaction and 'green' frauds. It has been exposed that most of the fraud transaction occurred in small amount which are close to '0$'. Moreover, other density of minor class shown around amount value of '20$'. The other characteristics we can derived that forgery density is much more higher in specific time slot. When normal transaction is higher then fraud transaction is also higher as well.

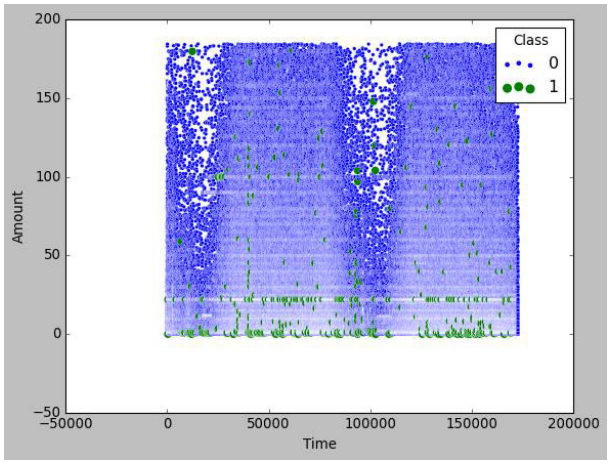

Figure 8: 'Class' scatter plot in terms of 'Amount' and 'Time'

5.1 Methodology:

The overall proposed model framework can be pictorially canvassed as below diagram

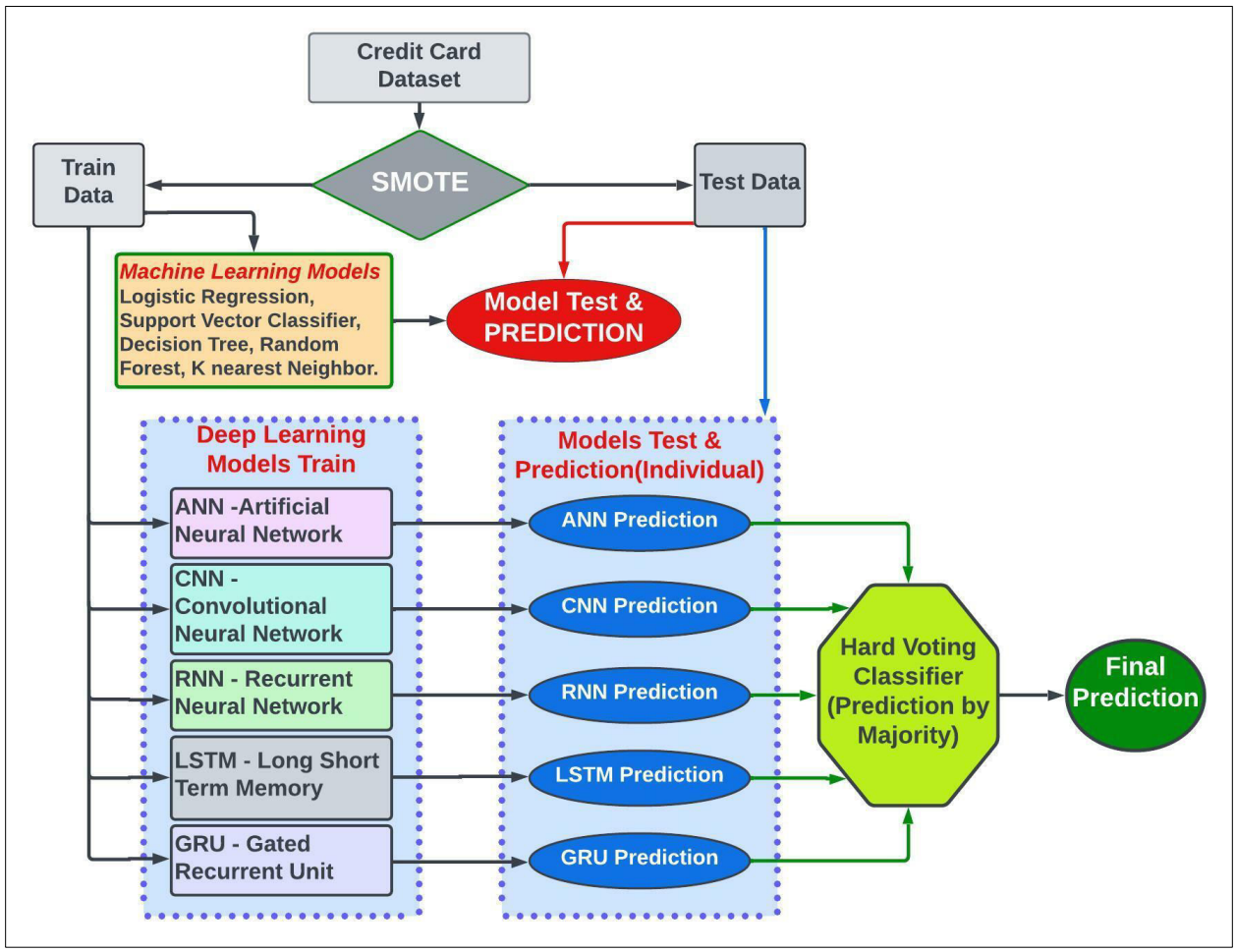

Figure 9: Proposed ensemble hard vote classifier architecture

As the dataset is highly imbalanced, So due to balancing the dataset we could do undersampling of majority class or oversampling the minority class. For our project, we will oversampling minority class due to balance the dataset. So, we will preserve all necessary values inside dataset.Ther are several oversampling technique available, we will use SMOTE for oversampling minority class. SMOTE stands for Synthetic Minority Oversampling Technique. It creates new synthetic samples to balance the dataset. SMOTE generates synthetic data using the k-nearest neighbour technique. Smote is used to construct step samples:

• Determine the feature vector's closest neighbor.

• Calculate the distance between the two sample points.

• Multiply the distance with a random number between 0 and 1.

• At the computed distance, find a new point on the line segment.

• Repeat the process for identified feature vectors.

After scaling ‘Time’ and ‘Amount’ features, we used SMOTE for oversample the imbalanced dataset. Then, used five machine learning models for classification on test purpose. The Following Machine Learning algorithms have been used for initial prediction.

-Logistic Regression

-Support Vector Classifier (SVC)

-Decision Tree Classifier

-Random Forest Classifier (maximum depth = 6)

-K neighbor Classifier(KNN) (k = 5)

We have created five neural networks which will be used for prediction individually.

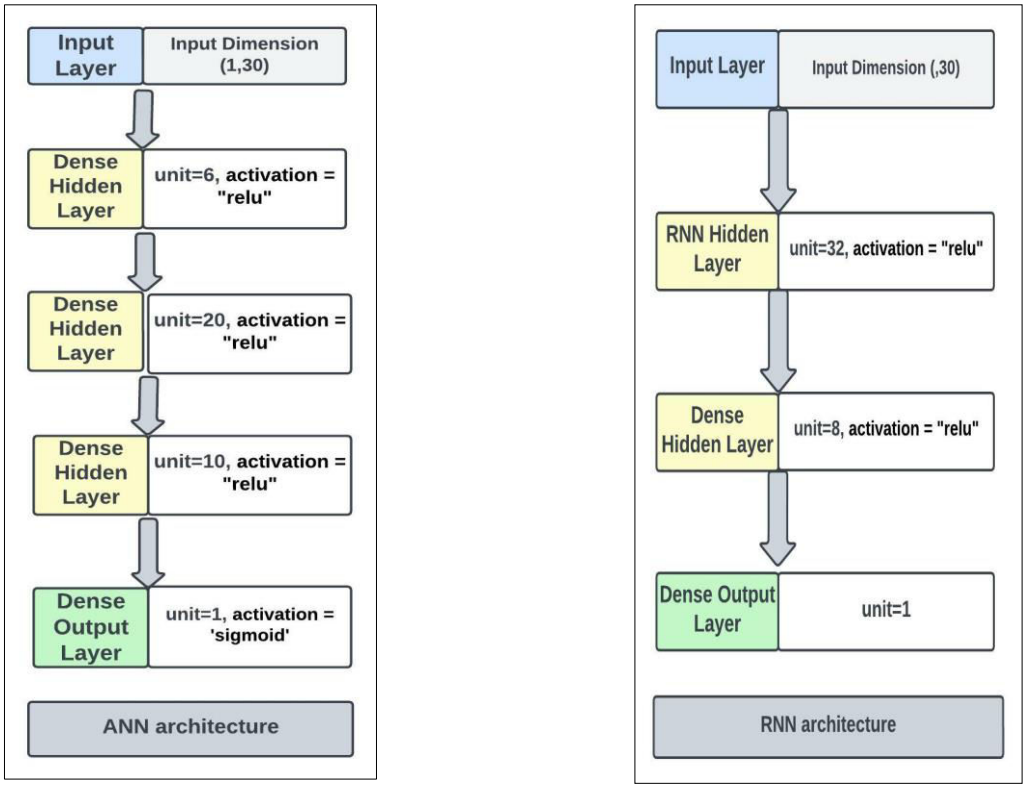

Figure 10: ANN and RNN models

We have created 4 layer artificial neural network(ANN) whereas, we used 6,20 and 10 units at hidden layer. There is only one node at output layer because as this is a binary classification. We used ‘relu’ activation function for all hidden layers and ‘sigmoid’ at output layer the input dimension is 1d array of 30 values, that means each observation. We used ‘binary cross entropy as loss function and adam as optimizer.

For recurrent neural network, we implemented three layer architecture with two hidden and one output layer. We used 32 and 8 units in hidden layer where 'relu' was the activation function. Though, the loss function was same as ANN, but here 'rmsprop' used as optimizer.

Binary cross entropy distinguishes each of the predicted probabilities to actual class output which can be either 0 or 1. The score is then calculated, penalizing the probabilities depending on their deviation from the predicted value. This refers to how close or far the value is to the actual value. The negative value of log of corrected predicted probabilities is binary cross entropy.

It can denote as

$$-\frac{1}{N}\sum_{i=1}^{N}(\log(p_i))$$

The Log loss formula is

$$\text{Log loss} = \frac{1}{N}\sum_{i=1}^{N} -(y_i * \log(p_i) + (1-y_i) * \log(1-p_i))$$

Root Mean Squared Propagation, or RMSProp, is a variation of gradient descent and the AdaGrad version of gradient descent that adapts the step size for each parameter using a declining average of partial gradients. In addition, adam is a substitute optimization algorithm for stochastic gradient descent in training period of models. Adam combines the finest features of the AdaGrad and RMSProp methods to create an optimization technique for noisy issues with sparse gradients.

Binary classification can be predicted in output layer by sigmoid function. As the output of the function relays from '0' to '1' so we can predict class by threshold. The equation is

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

The rectified linear activation function, or ReLU for short, is a piecewise linear function that, if the input is positive, outputs the input directly; else, it outputs zero.

During convolution neural network(CNN) training we make the training and testing data in three dimension. In the first layer of convolution layer we used 32 filter size, 2 kernel size, 'relu' as activation function and first rows of the input shape. We added batchnormalization , maxpool(1) and drop of 0.2 ratio. Batch normalisation is a transformation that keeps the mean output close to 0 and the standard deviation of the output close to 1. Maxpool has used for down sampling the input representation by taking the maximum value over a spatial window of size. The Dropout layer, which helps minimize overfitting, sets input units to 0 at random with a rate frequency at each step during training time. Inputs that aren't set to 0 are scaled up by 1/(1 - rate) so that the total sum remains the same. In our case, 20% neurons will be dropped randomly.

In the second layer of convolution, we used filter size 64 and making dropout parameter '0.5' before maxpool. Here we used batch normalization as well. We created a 'flatten' layer to change the input shape into a vector. The next layer is last hidden layer where as we used dense layer with 64 hidden units

and ‘relu’ activation function. Just before the output layer we made dropout of ‘0.5’ and used ‘sigmoid’ as activation function in output layer. During compile we used ‘adam’ optimizer with learning rate of ‘0.0001’.

CNN model summary is as follows

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 29, 32)	96
batch_normalization (BatchNo	(None, 29, 32)	128
max_pooling1d (MaxPooling1D)	(None, 14, 32)	0
dropout (Dropout)	(None, 14, 32)	0
conv1d_1 (Conv1D)	(None, 13, 64)	4160
batch_normalization_1 (Batch	(None, 13, 64)	256
max_pooling1d_1 (MaxPooling1	(None, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 64)	0
flatten (Flatten)	(None, 384)	0
dense_4 (Dense)	(None, 64)	24640
dropout_2 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 1)	65

During implementation of LSTM and GRU we set ‘**return_sequences=True’** when stacking LSTM layers so that the second LSTM layer has a three-dimensional sequence input.

The main distinction between GRU and LSTM is that GRU's bag has two gates: reset and update, but LSTM's bag has three gates: input, output, and forget. Because GRU has fewer gates than LSTM, it is less complicated. If the dataset is tiny, GRU should be used; otherwise, LSTM should be used for larger datasets. We have used both of the network.

We have created four layer LSTM deep learning models. First two layer is LSTM layer the other two are dense layer. For LSTM layers we used 100 and 50 units respectively. We used ‘relu’ activation function in both of the layer. GRU model have the same hyper parameter as LSTM. The last hidden dense layer contains 16 units and ‘relu’ activation function.

During compilation binary cross entropy has used and ‘rmsprop’ has used as optimizer.

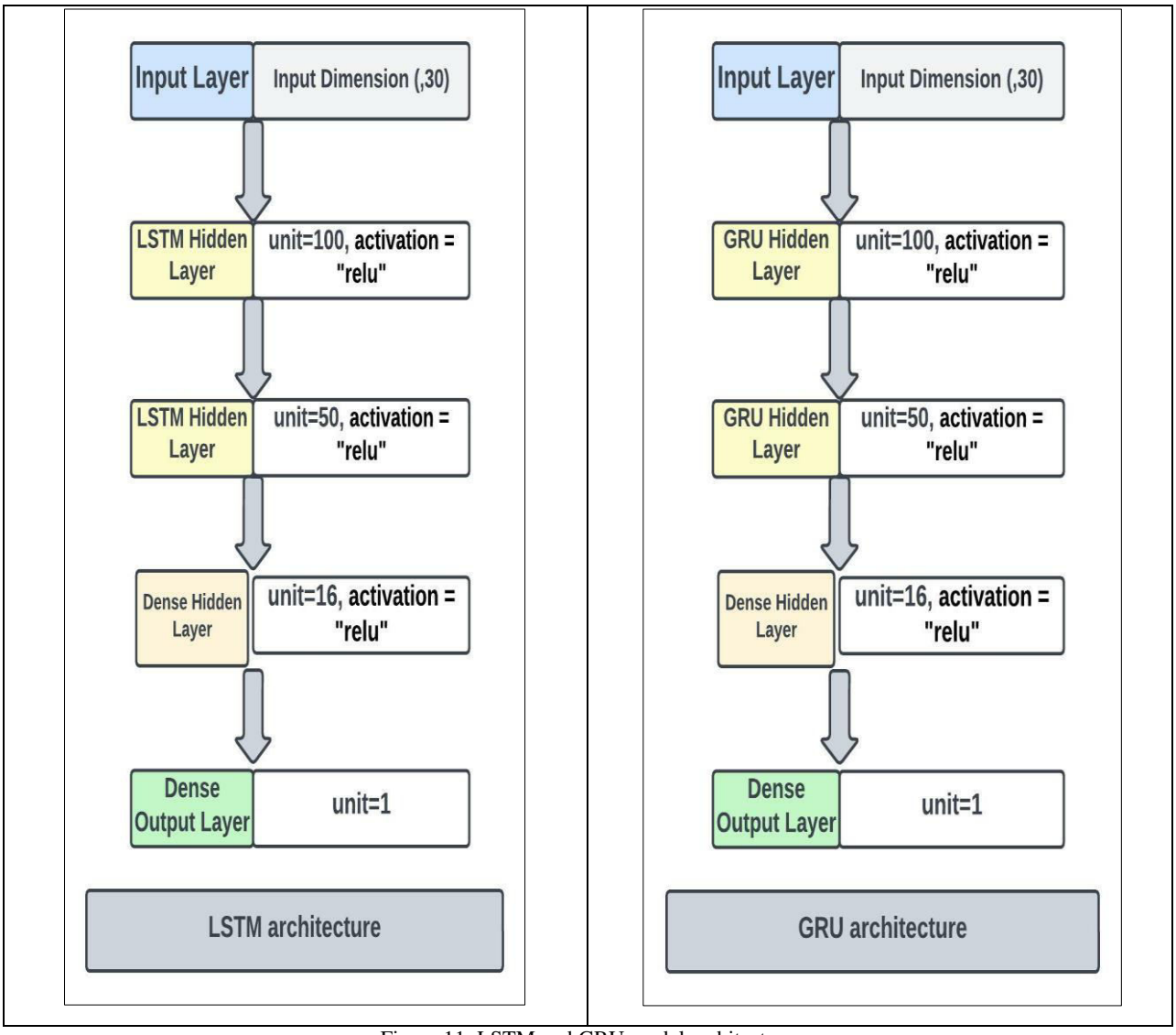

Figure 11: LSTM and GRU model architecture

Lastly, we made a hard voting classifier for final prediction. In hard voting (also known as majority voting), every individual classifier votes for a class, and the majority wins. In statistical terms, the predicted target label of the ensemble is the mode of the distribution of individually predicted labels.

6.1 Result and Evaluation:

Credit card fraud detection is a binary classification. To evaluate the model performance confusion matrix is the most popular option. It is a specific table layout to visualize the performance of the algorithms. Four cases can be happen during prediction. They are

TP(True Positive): Model predict the correct true label. In our cases, the number of instances where non fraud detection has predicted correctly.

TN(True Negative): Model predict the correct false label. That means in our problem the number of fraud detection predicted correctly.

FP(False Positive): Model predict wrong true label. In fraud detection problem when model predict genuine transaction but actually that is fraud transaction. This is the most important area where our concentration need to be centered.

FN(False Negative): Model predict false negative label. So, in our problem, when model predict fraud transaction but that is actually genuine. This is our next concern to deal with.

Mathmatically several terms computed to summarize overall model performance based on model prediction. They are

Sensitivity/Recall/True positive rate: It showed the probability of true positive prediction.

Sensitivity/Recall/TPR = TP/(TP + FN)

Specificity/True negative rate: It showed the probability of true negative prediction.

Specificity / TNR = TN/(TP + FP)

Precision/Positive predicted value: The probability to predict positive class among all positive class.

Precision/PPV = TP/(TP + FP)

Negative predicted value(NPV) which is just opposite of precision computed as TN/(TN + FN)

F1 score: To compute the F1 score, this assessment takes into account both precision and recall. The F1 score can be thought of as a weighted average of the precision and recall values

F1 score = 2TP/(2TP + FP + FN)

Accuracy: The accuracy of a model is determined by how well it perceives correlations and patterns between variables in a dataset using the input, or training, data.

Accuracy = (TP + TN)/(TP + FP + TN + FN)

AUC: AUC is a measurement of the complete two-dimensional area beneath the entire ROC curve.ROC curve plotted by TP vertically by TN horizontally.

As we use five machine learning algorithms(logistic regression,support vector machine, Decision tree, random forest and k nearest neighbor) initially, we got these confusion matrix for each.

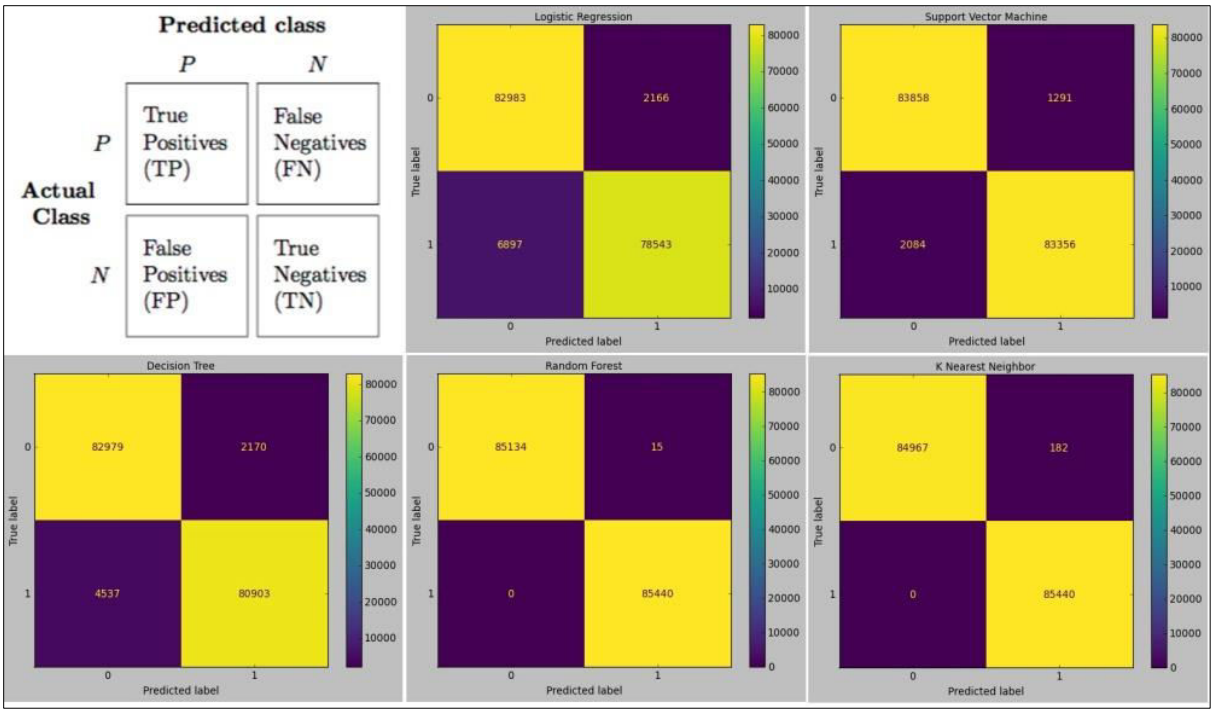

Figure 12: Confusion matrix heatmap of all machine learning models

And we got following summary values

Model	Precision	Recall/Sensitivity	F1-Score	Accuracy
LR	0.92	0.97	0.95	0.946
SVC	0.98	0.98	0.98	0.980
DT	0.95	0.97	0.96	0.960
RF	1.0	1.0	1.0	0.999
KNN	1.0	1.0	1.0	0.998

We can observe that random forest gave the greatest accuracy of 0.999 as a traditional machine learning approaches. Furthermore, KNN produced positive results. The false positive for random forest and knn is ‘0’ which is actually desired for classification. False negative for random forest is also less which is only ‘15’.

We deployed ANN, CNN, RNN, LSTM, and GRU for classification as deep learning algorithms, and then implemented a hard vote classifier, which will give the majority outcome for classification. We devised an approach in which all neural networks were predicted at the same time using epochs 50, 60, 70, and 80. We run the classifier for four times to evaluate and observe the differences. We created a heatmap of the confusion matrix based on the calculated value. So, let's look at the augmented confusion matrix for all deep learning techniques, as well as the classifier we've presented.

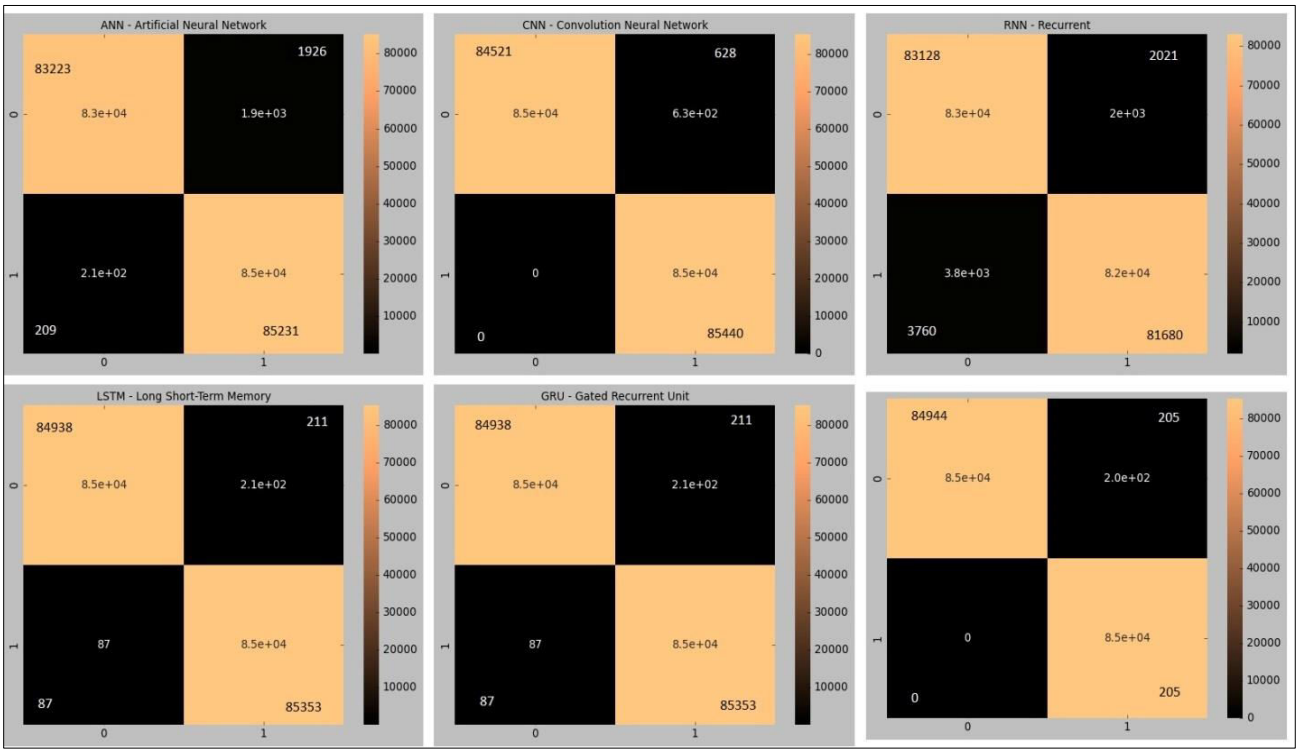

Figure 13: Confusion matrix heatmap of all deap learning models

We put the value here because it was machine scaled and colour coded. Individually, CNN had the most accuracy, with an FP score of '0' and a FN value of '628'. Our classifier has a surprise: just

'205' transactions have been misclassified as false negatives, meaning the model predicted the transaction as fraud when it was actually genuine.

We created a summary table for misclassification that included FN, FP, and total misclassification, TM = FN + FP, to better assess the performance of our hard voting classifier over time.

Models	**ANN**			**CNN**			**RNN**			**LSTM**			**GRU**			**Proposed**		
Epochs	FN	FP	TM	FN	FP	TM	FN	FP	TM	FN	FP	TM	FN	FP	TM	FN	FP	TM
50	209	1926	2135	0	628	628	3760	2021	5781	87	211	298	87	211	298	0	205	205
60	87	1524	1611	0	653	653	1353	898	2251	129	242	371	129	242	371	15	203	218
70	589	1181	1770	7	619	626	20348	1608	21956	0	223	223	0	223	223	2	171	173
80	304	1323	1627	0	500	500	1012	3276	4288	89	174	263	89	174	263	5	195	200

Figure 14: Misclassification summary table of all neural networks

One intriguing finding was that our proposed classifier consistently outperformed other networks in terms of misclassification. The intended 'TM' value for '50' epochs has been highlighted. CNN has the best overall effectiveness in detecting false negatives. LSTM and GRU performance are very similar. In this model, the RNN performances are the most inconsistent. For example, at epoch 70, there were more than 20000 incorrect classifications on FN. As, we have limited time and resources, we tried to figure up with in short range of epochs. However, it was discovered that the performance of each model is not dependent on the difference in epochs over a short period of time.

As a result, when we combine the model accuracy for machine learning, deep learning (based on 50 epochs), and the proposed classifier, we get the following figure.

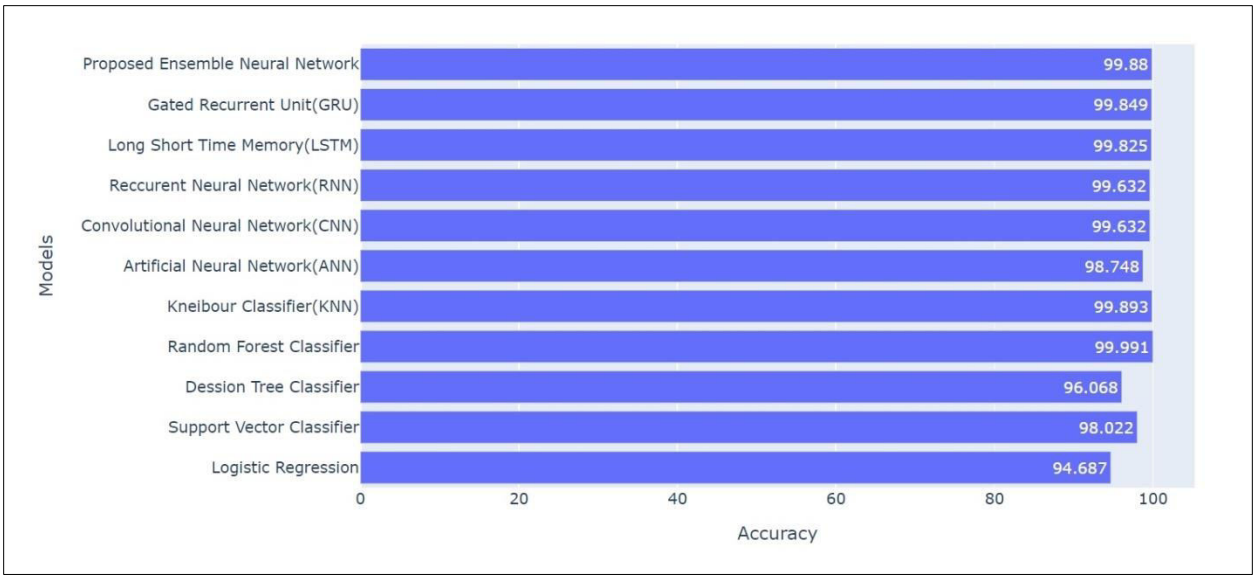

Figure 15: Accuracy plot of all machine and deap learning models

Although the random forest classifier with the maximum depth '6' provided the highest accuracy, performance is not often judged solely on accuracy or other computation figures. Here, too, time complexity is a major element. The model random forest and knn took more than 4 hours to train, whereas the 5 deep learning methods took around the same amount of time. When compared to neural networks, the proposed approach has the highest accuracy.

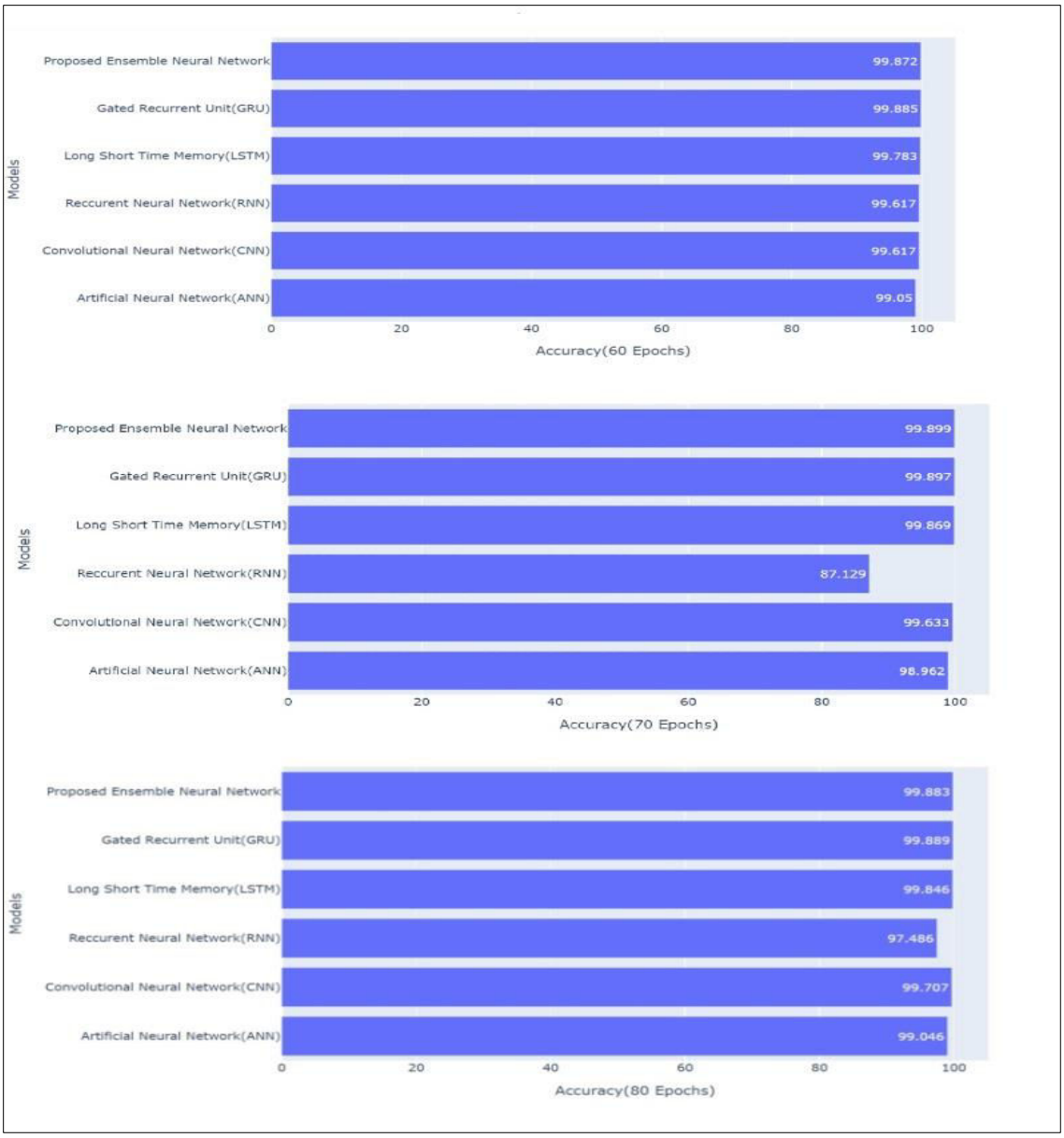

Figure 16:All neural network models and proposed ensemble model accuracy plot(Epochs -60,70,80)

When we formulate accuracy-based epochs, we find that our classifier has a somewhat lower reading than GRU. Next we wanted to deep down on each model performance measurement.

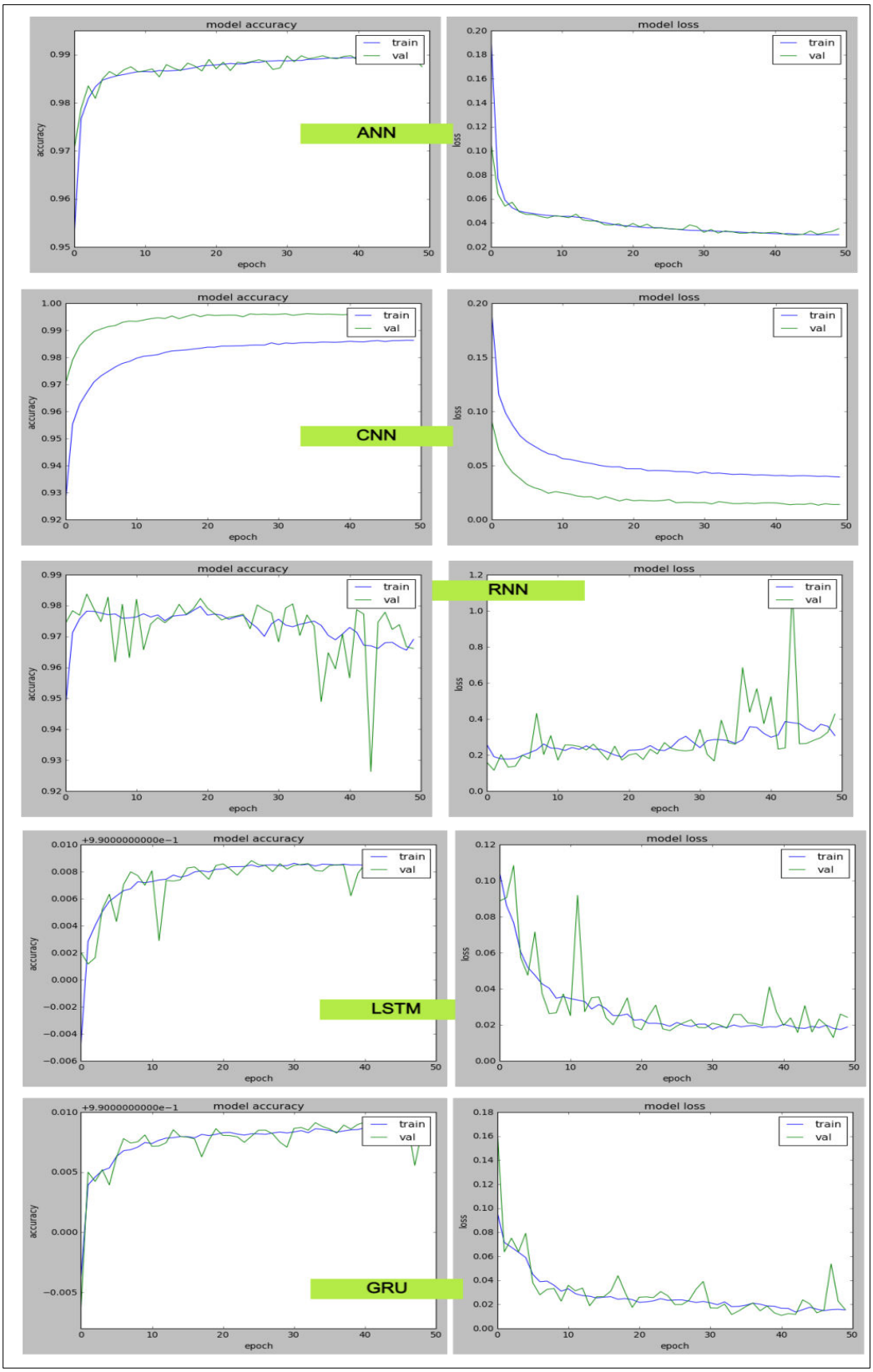

Figure 17: Accuracy and Loss curve of deap learning models individually

In the diagram above, the accuracy and loss curves for five neural networks with 50 epochs each are depicted. For all curves, the epoch range is scaled horizontally, whereas accuracy is presented vertically for all networks and is plotted in the first column. The loss values in the second images of each row are represented by the perpendicular scale. The corresponding neural network has written by highlighted box. For the accuracy curve, a value close to '1' indicates that the model is doing well, whereas, for the loss curve, values close to '0' indicate that the model is functioning well. ANN provides an organic growth of accuracy and consistent regression on loss curve with slight oscillation for both training and validation set. However, CNN has a similar curve, but the primary advantage is that it gave greater accuracy for the validation set than the training set, and it's just the opposite for loss. Moreover, the vacillation of validation accuracy and loss are very inconsistent than training set for RNN. We've already been notified of inconsistencies in RNN performance parameters, thus we can say that RNN isn't operating well in our implementation. The accuracy and loss curve for LSTM and GRU has shown similar swing in nature. We have been plotted accuracy curve and loss curve together two evaluate the performance of our networks in same epochs.

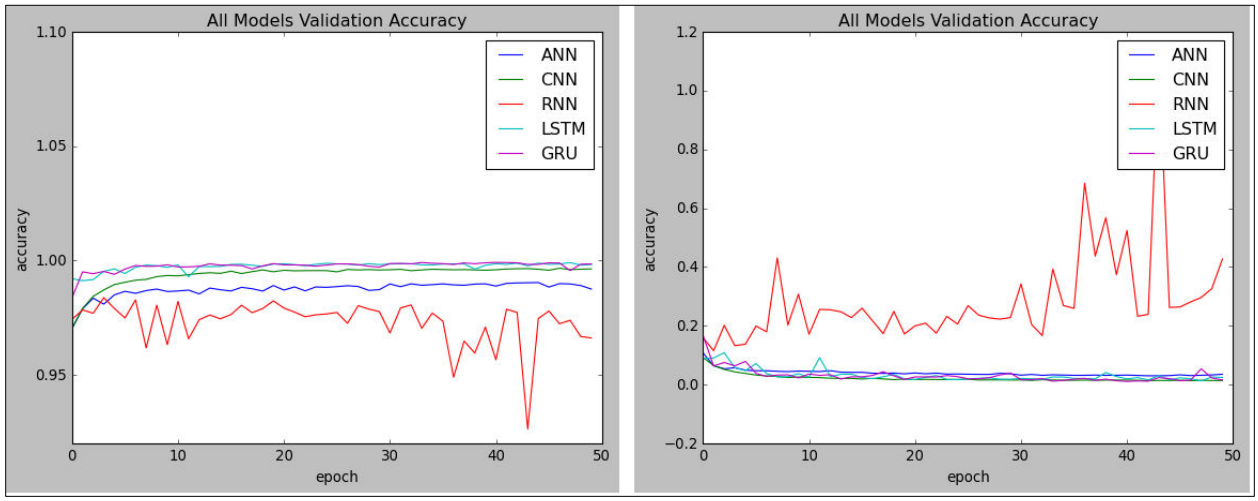

Figure 18: Accuracy and loss curve for all models together

It was evident from the start of the epochs that RNN performance was substandard, although the other four networks produced a similar curve pattern for accuracy and loss. Furthermore, because the LSTM and GRU architectures are similar by nature, and we selected similar hyper parameters in our model, the accuracy and loss for both networks are extremely similar. When we increase epochs the pattern is still very identical for these five networks.

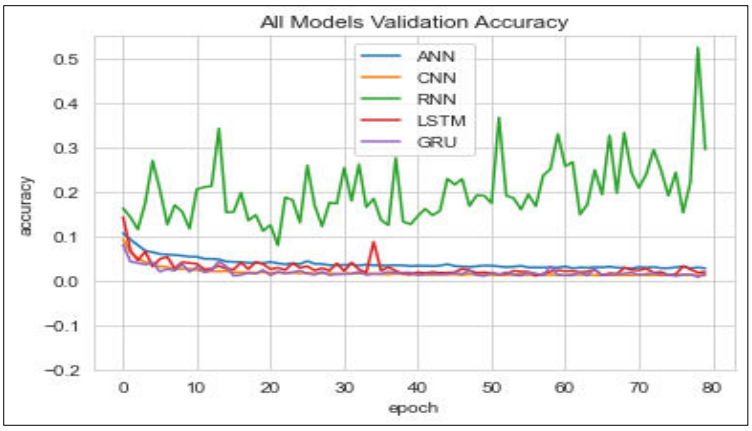

Figure 19: Accuracy for all neural network models(Epochs = 80)

For example RNN gave same type of performances for 80 epochs as well.

Lastly, we make an additional experiment based on 100 epochs for all neural network. This time we eliminated ANN and RNN from model. As a result, the voting classifier was tested on three different networks. CNN, LSTM, and GRU are the three algorithms. The highest accuracy has been reported for GRU individuals, whereas our classifier is slightly less accurate.

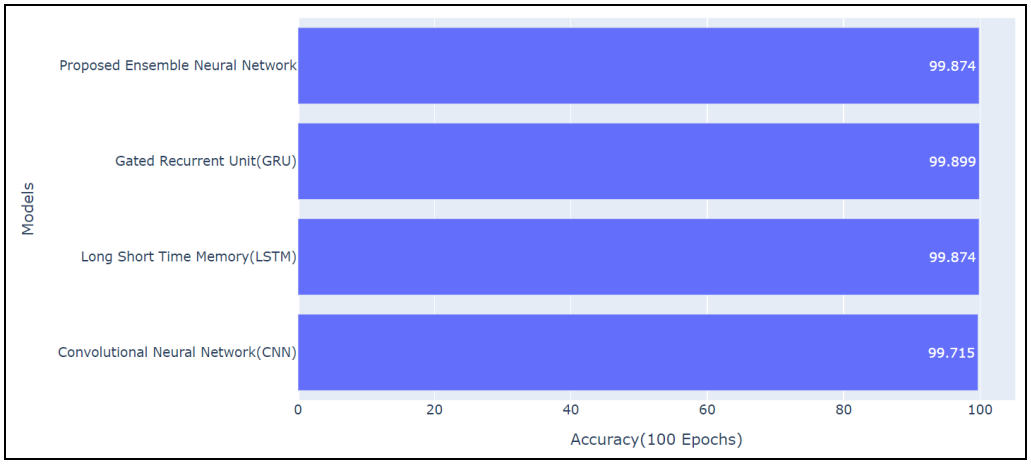

Figure 20: Accuracy plot of CNN,LSTM and GRU(Epochs=100)

We can deduce that there were some fraudulent transactions that GRU was able to detect whereas LSTM and CNN models were unable to.

7.1 Conclusion:

Credit card fraud detection is a sturdy problem in terms of detection. In this case, dataset was not properly described enough so we could not go through perfect feature selection. And, one additional unnecessary feature could make big difference in performance. The proposed ensemble hard voting neural network classifier had lower accuracy than individual neural networks on occasion. It happened because there might be some observation which was hard to classify from true label and most of the neural networks failed to predict them. So hard voting classifier failed to predict those false label transactions.

In terms of employing ensemble techniques on neural networks to solve this challenge, there is a lot of research could perform. We could not optimize hyper parameter because of limited time and resources. We might use different parameter and build neural networks to based on highest accuracy score. For instances, we could assemble more than 10 odd numbers of several GRU, LSTM or CNN and make prediction. In addition, we did not used k-fold validation process here during training. We didn't use it in our model because the training accuracy score was good in the range. Hyper parameter tuning could be very useful to build very robust model architecture. On the other hand, we applied SMOTE for oversampling on minority class label observations. There are six more oversampling technique available in research platform now. There are under sampling techniques available too. However, another ensemble method based on adaboost might be very vigorous too. In the optimised multiple neural network design, Adaboost could use GRU as the base learner. Furthermore machine learning algorithms and neural networks could use in ensemble methods as well.

Though our proposed technique did not defeat existing machine learning algorithms (random forest had the greatest accuracy rate of 99.9%, while our model had 99.89%), the gap could be bridged with future research.

Reference

- Abdallah, A., Maarof, A., & Zainal, A. (2016). Fraud detection system: A survey. https://doi.org/10.1016/j.jnca.2016.04.007
- Brause, R., Langsdorf, T., Hepp, M., (1999). Neural data mining for credit card fraud detection, in: Proceedings of the 11th International Conference on Tools with Artificial Intelligence, IEEE, Chicago, IL, USA. pp. 103–106.
- Bolton, R.J., Hand, D.J., (2002). Statistical fraud detection: A review. Statistical Science, 235–249.
- C. Phua, R. Gayler, V. Lee and K. Smith-Miles, "On the communal analysis suspicion scoring for identity crime in streaming credit applications", *Eur. J. Oper. Res.*, vol. 195, no. 2, pp. 595-612, Jun. 2009.
- E. Esenogho, I. D. Mienye, T. G. Swart, K. Aruleba and G. Obaido, "A Neural Network Ensemble With Feature Engineering for Improved Credit Card Fraud Detection," in IEEE Access, vol. 10, pp. 16400-16407, 2022, doi: 10.1109/ACCESS.2022.3148298.
- E. Ileberi, Y. Sun and Z. Wang, "Performance Evaluation of Machine Learning Methods for Credit Card Fraud Detection Using SMOTE and AdaBoost," in *IEEE Access*, vol. 9, pp. 165286-165294, 2021, doi: 10.1109/ACCESS.2021.3134330.
- F. Carcillo, A. D. Pozzolo, Y.-A. Le Borgne, O. Caelen, Y. Mazzer and G. Bontempi, "SCARFF: A scalable framework for streaming credit card fraud detection with spark", *Inf. Fusion*, vol. 41, pp. 182-194, May 2018.
- F. Carcillo, Y.-A. Le Borgne, O. Caelen, Y. Kessaci, F. Oblé and G. Bontempi, "Combining unsupervised and supervised learning in credit card fraud detection", *Inf. Sci.*
- H. Tingfei, C. Guangquan and H. Kuihua, "Using Variational Auto Encoding in Credit Card Fraud Detection," in IEEE Access, vol. 8, pp. 149841-149853, 2020, doi: 10.1109/ACCESS.2020.3015600.
- Identity theft and credit card fraud statistics (2020). The Ascent. https://www.fool.com/the-ascent/research/identity-theft-credit-card-fraud-statistics/
- Machine Learning Group-ULB. (2013, September). Credit Card Fraud Detection, Version 3, Retrieved March 1, 2022 from https://www.kaggle.com/mlg-ulb/creditcardfraud
- Misra, S., Thakur, S., Ghosh, M., & Saha, S. K. (2020). An autoencoder based model for detecting fraudulent credit card transactions. Procedia Computer Science, 167, 254-262.i/S1877050920306840
- P. A. Dal, G. Boracchi, O. Caelen, C. Alippi and G. Bontempi, "Credit card fraud detection: A realistic modeling and a novel learning strategy", *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3784-3797, Sep. 2017.
- R. Bolton and D. Hand, "Statistical fraud detection: A review", *Stat. Sci.*, vol. 17, no. 3, pp. 235-249, Aug. 2002.
- S. Makki, Z. Assaghir, Y. Taher, R. Haque, M. Hacid and H. Zeineddine, "An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection," in IEEE Access, vol. 7, pp. 93010-93022, 2019, doi: 10.1109/ACCESS.2019.2927266.

Appendix A:Python Code

```
# Project Code

# Md Mozahidur Rahman 501002626
# Muzammil Elahi 500754347
################################################################################
################
#ALL LIBRARIES

# Import sklearn Libraries

from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler

# Import Tensorflow & Keras Libraries

from tensorflow.keras.layers import Flatten, Dense, Dropout,
BatchNormalization
from tensorflow.keras.layers import Conv1D, MaxPool1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import SimpleRNN
from keras.layers import LSTM, Dense, Embedding, Dropout, Input, Attention,
Layer, Concatenate, Permute, Dot, Multiply, \
    Flatten, GRU
from keras.layers import RepeatVector, Dense, Activation, Lambda
from keras.models import Sequential

# Visulization Libraries
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import seaborn as sns
sns.set_style("whitegrid")
from matplotlib import pyplot

# Other Libraries

import collections
from imblearn.over_sampling import SMOTE
from collections import Counter
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd

################################################################################
#############
# Data load
```

```
"""
Test.csv - A very small subset of data(around 100 observation). It will take
4/5 minutes to run.
creditcard.csv - Original dataset. It will take around 8 hours to run whole
code.
"""
#df = pd.read_csv('Test.csv')
df = pd.read_csv('creditcard.csv')
df.head()
"""
Original features V1, V2,... V28 have been converted using PCA due to a
confidentiality issue; nonetheless, my guess is
that these features include credit card number, expiration date, CVV,
cardholder name, transaction location, transaction
date-time, and so on.
Only features which have not been transformed with PCA are 'Time',  'Amount'
and 'Class'.
  - 'Time' contains the seconds elapsed between each transaction and the
first transaction in the dataset.
  - The transaction Amount is represented by the feature 'Amount,' which can
be utilised for example-dependent cost-sensitive      learning.
  - The response or target variable is the Feature 'Class,' which takes the
value 1 in the case of fraud and 0 otherwise.
"""
# Data Prepocessing

df.shape


df.columns

"""
Because the majority of the columns V1, V2,... V28 are modified using PCA,
neither the features nor the descriptive
statistics will make much sense, so we'll ignore them and only consider Time
and Amount, which makes sense.
"""

df.info()

df[df.duplicated()].shape

# It looks like there are 1081 duplicate data, we should drop them.

df.drop_duplicates()

# After dropping the duplicate rows the dataset have 283726 rows and 31
columns.

################################################################################
################
################################################################################
################
# Initial Analysis

# Univariate Analysis
```

```
# The purpose of univariate analysis is to understand the distribution of
values for a single variable.

# Target variable/Class analysis

df['Class'].value_counts()
df[df.Class == 0].Amount.describe()
df[df.Class == 1].Amount.describe()
colors = ["#0101DF", "#DF0101"]
sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)

# Outlier Detection
"""
An outlier is a value in a random sampling from a population that deviates
abnormally from other
values. we are using Box Plot to detect the outliers of 'Amount' features in
our dataset, where
any point above or below the whiskers represent an outlier. As Amount is the
only numeric feature
which has meaning.
"""
plt.figure(figsize=[5, 5])
df.boxplot(column=['Amount'])
plt.show()
"""
-We will apply Median Imputation to deal with outliers after we've detected
them. The extreme numbers
are replaced by median values in this strategy. IQR = Q3-Q1 is the formula to
calculate it. For each
of the variables in the dataset feature "Amount", the lines of code below
calculate and report the
interquartile range.
"""
q1 = df['Amount'].quantile(0.25)
q3 = df['Amount'].quantile(0.75)
iqr = q3 - q1
Lower_tail = q1 - 1.5 * iqr
Upper_tail = q3 + 1.5 * iqr
med = np.median(df['Amount'])
for j in df['Amount']:
    if j > Upper_tail or j < Lower_tail:
        df['Amount'] = df['Amount'].replace(j, med)
plt.figure(figsize=[5, 5])
df.boxplot(column=['Amount'])
plt.show()

# Column 'Time' and 'Amount' Distribution
plt.figure(figsize=(14, 10))
plt.subplot(2, 2, 1)
plt.title('Time Distribution (Seconds)')
sns.distplot(df['Time'], color='blue');
plt.subplot(2, 2, 2)
plt.title('Amount Distribution')
sns.distplot(df['Amount'], color='red');

# Bivariate Analysis
```

```
# correlation investigation among features

# heatmap to find any high correlations
plt.figure(figsize=(20, 20))
sns.heatmap(data=df.corr(), cmap="seismic", linewidths=1, linecolor='white',
annot=True, fmt=".2f")
plt.show();
"""
Highest correlations come from:
 - Negative: Time & V3 (-0.42)
 - Positive: Time & V5 (0.17)

The correlation matrix shows also that none of the V1 to V28 PCA components
have any correlation to
 each other however if we observe Class has some form positive and negative
correlations with the V
 components but has no correlation with Time and Amount.
"""
# Distribution Plot with Class variable:

V = df[[col for col in df.columns if 'V' in col] + ['Class']]
f, ax = plt.subplots(ncols=4, nrows=7, figsize=(15, 1 * len(V.columns)))
for i, c in zip(ax.flatten(), V.columns):
    sns.distplot(V[c][V['Class'] == 0], color='blue', ax=i)  # Genuine
    sns.distplot(V[c][V['Class'] == 1], color='red', ax=i)  # Fraud
f.tight_layout()

# Amount distribution based on Class

plt.style.use("classic")
plt.figure(figsize=(15, 5))
sns.distplot(df[df['Class'] == 0]["Amount"], color='blue')  # Genuine - blue
sns.distplot(df[df['Class'] == 1]["Amount"], color='red')  # Fraud - Red
plt.title('Genuine vs Fraud by Amount', fontsize=15)
plt.xlim([-10, 200])
plt.grid(linewidth=0.7)
plt.legend(['Genuine Transaction', 'Fraud Transaction'])
plt.show()

"""
We can notice that transaction under 500$ have the most density. lower amount
have high density
in fraud transaction. It is for being unnoticable.
"""

# Two Class(Genuine and Fraud) with Time

plt.style.use("classic")
plt.figure(figsize=(8, 8))
sns.distplot(df[df['Class'] == 0]["Time"], color='blue')  # Genuine - blue
sns.distplot(df[df['Class'] == 1]["Time"], color='red')  # Fraud - Red
plt.title('Genuine vs Fraud by Time(in sec)', fontsize=15)
plt.xlim([-10000, 180000])
plt.grid(linewidth=0.7)
plt.legend(['Genuine Transaction', 'Fraud Transaction'])
plt.show()
```

```
# Multivariate Analysis

# Time and Amound plot based on Classes.

sns.scatterplot(data=df, x="Time", y="Amount", hue="Class", size="Class",
sizes=(50, 10))
# as the dataset is imbalance so minor classes points is plotted 5 times
bigger than majority class

# Feature Scaling
"""
As V1 to V28 features are already PCA transformed and 'Class' is classifier
so the feature "Time" and
"Amount" would need for scalling for better prediction in deep learning
model.

We will use 'StandardScaler' for scalling two features. StandardScaler
standardizes a feature by
subtracting the mean and then scaling to unit variance.Divide all of the
values by the standard
deviation to get unit variance. The StandardScaler function produces a
distribution with a standard
deviation of one.
"""
sc = StandardScaler()
df[['Time', 'Amount']] = sc.fit_transform(df[['Time', 'Amount']])

##############################################################################
####################
##############################################################################
####################
# Handling Imbalance Data
"""
As the dataset is highly imbalanced. So due to balancing the dataset we could
do undersampling
of majority class or oversampling the minority class. For our project, we
will oversampling minority
class due to balance the dataset. So, we will preserve all necessary values
inside dataset.Ther are
several oversampling technique available, we will use SMOTE for oversampling
minority class.

SMOTE stands for Synthetic Minority Oversampling Technique. It creates new
synthetic samples to
balance the dataset.SMOTE generates synthetic data using the k-nearest
neighbour technique. Smote
is used to construct step samples:

 - Determine the feature vector's closest neighbour.
 - Calculate the distance between the two sample points.
 - Multiply the distance with a random number between 0 and 1.
 - At the computed distance, find a new point on the line segment.
 - Repeat the process for identified feature vectors.
"""
Data = df.drop('Class', axis=1)
Target = df['Class']
```

```
print('shape of Data', Data.shape)
print('shape of Target', Target.shape)

oversample = SMOTE(random_state=2)
Data_new, Target_new = oversample.fit_resample(Data, Target)
print('shape of Data after oversampling', Data_new.shape)
print('shape of Target after oversampling', Target_new.shape)

# Class Distribution of Balanced Data

print('Before Oversampling: {}'.format(Counter(Target)))
print('After Oversampling: {}'.format(Counter(Target_new)))

d = collections.Counter(Target_new)
sns.barplot(list(d.keys()), list(d.values()))
##############################################################################
###############
# Split Data

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(Data_new, Target_new,
test_size=0.3, random_state=42)

##############################################################################
################
##############################################################################
################
# Machine Learning Models
"""
# The Following Machine Learning algorithms has been used for initial
prediction.
# -Logistic Regression
# -Support Vector Classifier(SVM)
# -Decision Tree Classifier
# -Random Forest Classifier
# -K nearest neibor Classifier(KNN)

# Function 'model_evaluate' - It will take model as parameter and print
results and return confusion matrix.
# At last part it will print confusion matrix heatmap for all ML models

# Record result- 'Accuracy_all_MODEL'is a list that will keep the record for
all accuracy score.
"""
Accuracy_all_MODEL = []

Accuracy_all_MODEL = []

def model_evaluate(model):
    model.fit(X_train, y_train)
    acc = model.score(X_test, y_test)
    train_score = model.score(X_train, y_train)
    y_prediction = model.predict(X_test)
    com_decision = confusion_matrix(y_test, y_prediction)

    Accuracy_model = ((com_decision[0][0] + com_decision[1][1]) /
```

```
com_decision.sum()) * 100
    Error_rate_model = ((com_decision[0][1] + com_decision[1][0]) /
com_decision.sum()) * 100
    Specificity_model = (com_decision[1][1] / (com_decision[1][1] +
com_decision[0][1])) * 100
    Sensitivity_model = (com_decision[0][0] / (com_decision[0][0] +
com_decision[1][0])) * 100

    class_report = classification_report(y_test, y_prediction)

    print("=========================================")
    print("Model Name", model)
    print("=========================================")
    print("Training Score:", train_score)
    print("")
    print("Confusion Matrix")
    print(com_decision)

    print('Accuracy Decision : ', Accuracy_model)
    print('Error Rate Decision : ', Error_rate_model)
    print('Specificity Decision : ', Specificity_model)
    print('Sensitivity Decision : ', Sensitivity_model)
    print("-----------------------------------------")
    print("CLASSIFICATION REPORT")
    print("-----------------------------------------")
    print(print(class_report))
    print("-----------------------------------------")
    print("MODEL ACCURACY SCORE : ", acc)
    print("=========================================")
    print("\n")
    Accuracy_all_MODEL.append(acc)
    return com_decision
lr = LogisticRegression()
svm = SVC()
dt = DecisionTreeClassifier(max_depth=6)
rf = RandomForestClassifier(max_samples=0.9)
knn = KNeighborsClassifier(n_neighbors=5)
models = [lr, svm, dt, rf, knn]
for model in models:
    model_evaluate(model)

plot_confusion_matrix(lr, X_test, y_test)
plt.title('Logistic Regression', fontsize=12)
plt.show()
plot_confusion_matrix(svm, X_test, y_test)
plt.title('Support Vector Machine', fontsize=12)
plt.show()
plot_confusion_matrix(dt, X_test, y_test)
plt.title('Decision Tree', fontsize=12)
plt.show()
plot_confusion_matrix(rf, X_test, y_test)
plt.title('Random Forest', fontsize=12)
plt.show()
plot_confusion_matrix(knn, X_test, y_test)
plt.title('K Nearest Neighbor', fontsize=12)
plt.show()
```

```
# SAVE MODELS and LOAD

"""
# save the model to disk
# lr,svm,dt,rf,knn
import joblib
Project = 'LR_model.sav'
joblib.dump(lr, Project)

Project = 'SVM_model.sav'
joblib.dump(svm, Project)

Project = 'DT_model.sav'
joblib.dump(dt, Project)

Project = 'RF_model.sav'
joblib.dump(rf, Project)

Project = 'KNN_model.sav'
joblib.dump(knn, Project)
"""

##########################################################################
######################
##########################################################################
######################
##########################################################################
######################
# Deep Learning Models
"""
METHODOLOGY:
-The code will run for Artificial Neural Network(ANN), Convolutional Neural
Network(CNN), Reccurent
Neural Network(RNN), Long Short Time Memory(LSTM) and Gated Recurrent Unit
individually.
-For ANN input shape is two dimensional, for the rest, input shape is three
dimensional.

Function confusion_metrix_calculation - It will taken neural network
prediction and y_test as parameter.
and Convert both parameter to list calculate confusion matrix and also will
make heat map from confusion matrix
"""

def confusion_metrix_calculation(ypred, ytest):
    y_target = list(ytest)
    y_prediction = list(ypred)
    from sklearn.metrics import confusion_matrix
    com_decision = confusion_matrix(y_target, y_prediction)
    Accuracy_model = ((com_decision[0][0] + com_decision[1][1]) /
com_decision.sum()) * 100
    Error_rate_model = ((com_decision[0][1] + com_decision[1][0]) /
com_decision.sum()) * 100
    Specificity_model = (com_decision[1][1] / (com_decision[1][1] +
com_decision[0][1])) * 100
    Sensitivity_model = (com_decision[0][0] / (com_decision[0][0] +
com_decision[1][0])) * 100
```

```
    print("Confusion Matrix")
    print(com_decision)
    print('Error Rate     : ', Error_rate_model)
    print('Specificity    : ', Specificity_model)
    print('Sensitivity    : ', Sensitivity_model)
    print('MODEL ACCURACY : ', Accuracy_model)
    sns.heatmap(com_decision, annot=True, cmap=plt.cm.copper)
##########################################################################
##################
##########################################################################
##################
# ANN

X_train_ann1 = X_train
X_test_ann1 = X_test
y_train_ann = y_train
y_test_ann = y_test

sc = StandardScaler()
X_train_ann = sc.fit_transform(X_train_ann1)
X_test_ann = sc.transform(X_test_ann1)

model_ANN = Sequential()
model_ANN.add(Dense(activation="relu", input_dim=30, units=6,
kernel_initializer='uniform'))
model_ANN.add(Dense(activation="relu", units=20,
kernel_initializer='uniform'))
model_ANN.add(Dense(activation="relu", units=10,
kernel_initializer='uniform'))
model_ANN.add(Dense(activation='sigmoid', units=1,
kernel_initializer='uniform', ))
model_ANN.summary()
model_ANN.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
history_ANN = model_ANN.fit(X_train_ann, y_train_ann,
validation_data=(X_test_ann, y_test_ann), batch_size=100,epochs=50)
#history_ANN=model_ANN.fit(X_train_ann,
y_train_ann,validation_data=(X_test_ann, y_test_ann), batch_size = 100,
epochs =60)
#history_ANN=model_ANN.fit(X_train_ann,
y_train_ann,validation_data=(X_test_ann, y_test_ann), batch_size = 100,
epochs =70)
#history_ANN=model_ANN.fit(X_train_ann,
y_train_ann,validation_data=(X_test_ann, y_test_ann), batch_size = 100,
epochs =80)
# model_ANN.save('model_ANN.hdf5')

# evaluate the model
_, train_acc = model_ANN.evaluate(X_train_ann, y_train_ann, verbose=0)
_, test_acc = model_ANN.evaluate(X_test_ann, y_test_ann, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
# Plot accuracy of training and validation
pyplot.plot(history_ANN.history['accuracy'])
pyplot.plot(history_ANN.history['val_accuracy'])
pyplot.title('model accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
```

```
pyplot.legend(['train', 'val'])
pyplot.show()
# Plot losses for training and validation set
pyplot.plot(history_ANN.history['loss'])
pyplot.plot(history_ANN.history['val_loss'])
pyplot.title('model loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

ypred_ann = model_ANN.predict_classes(X_test_ann)
confusion_metrix_calculation(ypred_ann, y_test)
plt.title('ANN - Artificial Neural Network', fontsize=12)
plt.show()

##############################################################################
######################
##############################################################################
######################
# CNN

X_train_cnn1 = X_train
X_test_cnn1 = X_test
y_train_cnn = y_train
y_test_cnn = y_test

sc = StandardScaler()
X_train_cnn = sc.fit_transform(X_train_cnn1)
X_test_cnn = sc.transform(X_test_cnn1)
X_train_cnn = X_train_cnn.reshape(X_train_cnn.shape[0], X_train_cnn.shape[1],
1)
X_test_cnn = X_test_cnn.reshape(X_test_cnn.shape[0], X_test_cnn.shape[1], 1)

epochs = 50
model_CNN = Sequential()
model_CNN.add(Conv1D(32, 2, activation='relu',
input_shape=X_train_cnn[0].shape))
model_CNN.add(BatchNormalization())
model_CNN.add(MaxPool1D(2))
model_CNN.add(Dropout(0.2))
model_CNN.add(Conv1D(64, 2, activation='relu'))
model_CNN.add(BatchNormalization())
model_CNN.add(MaxPool1D(2))
model_CNN.add(Dropout(0.5))
model_CNN.add(Flatten())
model_CNN.add(Dense(64, activation='relu'))
model_CNN.add(Dropout(0.5))
model_CNN.add(Dense(1, activation='sigmoid'))
model_CNN.summary()
model_CNN.compile(optimizer=Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])
history_CNN = model_CNN.fit(X_train_cnn, y_train_cnn, epochs=50,
validation_data=(X_test_cnn, y_test_cnn))
#history_CNN =
model_CNN.fit(X_train_cnn,y_train_cnn,epochs=60,validation_data=(X_test_cnn,y
_test_cnn))
```

```
#history_CNN =
model_CNN.fit(X_train_cnn,y_train_cnn,epochs=70,validation_data=(X_test_cnn,y
_test_cnn))
#history_CNN =
model_CNN.fit(X_train_cnn,y_train_cnn,epochs=80,validation_data=(X_test_cnn,y
_test_cnn))
# model_CNN.save('model_CNN.hdf5')

# evaluate the model
_, train_acc2 = model_CNN.evaluate(X_train_cnn, y_train_cnn, verbose=0)
_, test_acc2 = model_CNN.evaluate(X_test_cnn, y_test_cnn, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc2, test_acc2))

#Plot accuracy of training and validation set
pyplot.plot(history_CNN.history['accuracy'])
pyplot.plot(history_CNN.history['val_accuracy'])
pyplot.title('model accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot loss of training and validation set
pyplot.plot(history_CNN.history['loss'])
pyplot.plot(history_CNN.history['val_loss'])
pyplot.title('model loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot Confusion matrix
ypred_cnn = model_CNN.predict_classes(X_test_cnn)
confusion_metrix_calculation(ypred_cnn, y_test)
plt.title('CNN - Convolution Neural Network', fontsize=12)
plt.show()

##########################################################################
######################
##########################################################################
######################
# RNN

X_train_rnn1 = X_train
X_test_rnn1 = X_test
y_train_rnn = y_train
y_test_rnn = y_test

rc = StandardScaler()
X_train_rnn = rc.fit_transform(X_train_rnn1)
X_test_rnn = rc.transform(X_test_rnn1)
X_train_rnn = X_train_rnn.reshape(X_train_rnn.shape[0], X_train_rnn.shape[1],
1)
X_test_rnn = X_test_rnn.reshape(X_test_rnn.shape[0], X_test_rnn.shape[1], 1)

# Building RNN model for the dataset
model_RNN = Sequential()
```

```
model_RNN.add(SimpleRNN(32, input_shape=X_train_rnn[0].shape,
activation="relu"))
model_RNN.add(Dense(8, activation="relu"))
model_RNN.add(Dense(1))
model_RNN.compile(loss="binary_crossentropy", optimizer="rmsprop")
model_RNN.summary()
model_RNN.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])
history_RNN = model_RNN.fit(X_train_rnn, y_train_rnn, epochs=50,
validation_data=(X_test_rnn, y_test_rnn), verbose=1)
#history_RNN = model_RNN.fit(X_train_rnn,y_train_rnn, epochs=60,
validation_data=(X_test_rnn, y_test_rnn),verbose=1)
#history_RNN = model_RNN.fit(X_train_rnn,y_train_rnn, epochs=70,
validation_data=(X_test_rnn, y_test_rnn),verbose=1)
#history_RNN = model_RNN.fit(X_train_rnn,y_train_rnn, epochs=80,
validation_data=(X_test_rnn, y_test_rnn),verbose=1)
# model_RNN.save('model_RNN.hdf5')

# evaluate the model
_, train_acc3 = model_CNN.evaluate(X_train_rnn, y_train_rnn, verbose=0)
_, test_acc3 = model_CNN.evaluate(X_test_rnn, y_test_rnn, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc3, test_acc3))

# Plot accuracy of training and validation set
pyplot.plot(history_RNN.history['accuracy'])
pyplot.plot(history_RNN.history['val_accuracy'])
pyplot.title('model accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot loss of training and validation set
pyplot.plot(history_RNN.history['loss'])
pyplot.plot(history_RNN.history['val_loss'])
pyplot.title('model loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot Confusion matrix
ypred_rnn = model_RNN.predict_classes(X_test_rnn)
confusion_metrix_calculation(ypred_rnn, y_test)
plt.title('RNN - Recurrent', fontsize=12)
plt.show()

##############################################################################
####################
##############################################################################
####################
# LSTM

X_train_lstm1 = X_train
X_test_lstm1 = X_test
y_train_lstm = y_train
y_test_lstm = y_test
```

```
ls = StandardScaler()
X_train_lstm = ls.fit_transform(X_train_lstm1)
X_test_lstm = ls.transform(X_test_lstm1)
X_train_lstm = X_train_lstm.reshape(X_train_lstm.shape[0], 1,
X_train_lstm.shape[1])
X_test_lstm = X_test_lstm.reshape(X_test_lstm.shape[0], 1,
X_test_lstm.shape[1])

# Building LSTM model for the dataset
model_LSTM = Sequential()
model_LSTM.add(LSTM(100, input_shape=X_train_lstm[0].shape,
return_sequences=True, activation="relu"))
model_LSTM.add(LSTM(50, return_sequences=True, activation="relu"))
model_LSTM.add(Dense(16, activation="relu"))
model_LSTM.add(Dense(1))
model_LSTM.compile(loss="binary_crossentropy", optimizer="rmsprop")
model_LSTM.summary()
model_LSTM.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])
history_LSTM = model_LSTM.fit(X_train_lstm, y_train_lstm, epochs=50,
validation_data=(X_test_lstm, y_test_lstm),
                              verbose=1)
#history_LSTM = model_LSTM.fit(X_train_lstm,y_train_lstm, epochs=60,
validation_data=(X_test_lstm, y_test_lstm),verbose=1)
#history_LSTM = model_LSTM.fit(X_train_lstm,y_train_lstm, epochs=70,
validation_data=(X_test_lstm, y_test_lstm),verbose=1)
#history_LSTM = model_LSTM.fit(X_train_lstm,y_train_lstm, epochs=80,
validation_data=(X_test_lstm, y_test_lstm),verbose=1)
# model_LSTM.save('model_LSTM.hdf5')

# evaluate the model
_, train_acc4 = model_LSTM.evaluate(X_train_lstm, y_train_lstm, verbose=0)
_, test_acc4 = model_LSTM.evaluate(X_test_lstm, y_test_lstm, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc4, test_acc4))

#Plot accuracy of training and validation set
pyplot.plot(history_LSTM.history['accuracy'])
pyplot.plot(history_LSTM.history['val_accuracy'])
pyplot.title('model accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot loss of training and validation set
pyplot.plot(history_LSTM.history['loss'])
pyplot.plot(history_LSTM.history['val_loss'])
pyplot.title('model loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot Confusion matrix
ypred_lstm = model_LSTM.predict_classes(X_test_lstm)
ypred_lstm = ypred_lstm.reshape(ypred_lstm.shape[0], ypred_lstm.shape[1])
confusion_metrix_calculation(ypred_lstm, y_test)
```

```
plt.title('LSTM - Long Short-Term Memory', fontsize=12)
plt.show()

#########################################################################################
####################
#########################################################################################
####################
# GRU

X_train_gru1 = X_train
X_test_gru1 = X_test
y_train_gru = y_train
y_test_gru = y_test
ls = StandardScaler()
X_train_gru = ls.fit_transform(X_train_gru1)
X_test_gru = ls.transform(X_test_gru1)
X_train_gru = X_train_gru.reshape(X_train_gru.shape[0], 1,
X_train_gru.shape[1])
X_test_gru = X_test_gru.reshape(X_test_gru.shape[0], 1, X_test_gru.shape[1])

# Building GRU model for the dataset
model_GRU = Sequential()
model_GRU.add(GRU(100, input_shape=X_train_gru[0].shape,
return_sequences=True, activation="relu"))
model_GRU.add(GRU(50, return_sequences=True, activation="relu"))
model_GRU.add(Dense(16, activation="relu"))
model_GRU.add(Dense(1))
model_GRU.compile(loss="binary_crossentropy", optimizer="rmsprop")
model_GRU.summary()
model_GRU.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])
history_GRU = model_GRU.fit(X_train_gru, y_train_gru, epochs=50,
validation_data=(X_test_gru, y_test_gru), verbose=1)
#history_GRU = model_GRU.fit(X_train_gru,y_train_gru, epochs=60,
validation_data=(X_test_gru, y_test_gru),verbose=1)
#history_GRU = model_GRU.fit(X_train_gru,y_train_gru, epochs=70,
validation_data=(X_test_gru, y_test_gru),verbose=1)
#history_GRU = model_GRU.fit(X_train_gru,y_train_gru, epochs=80,
validation_data=(X_test_gru, y_test_gru),verbose=1)
# model_GRU.save('model_GRU.hdf5')

# evaluate the model
_, train_acc5 = model_GRU.evaluate(X_train_gru, y_train_gru, verbose=0)
_, test_acc5 = model_GRU.evaluate(X_test_gru, y_test_gru, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc5, test_acc5))

#Plot accuracy of training and validation set
pyplot.plot(history_GRU.history['accuracy'])
pyplot.plot(history_GRU.history['val_accuracy'])
pyplot.title('model accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot loss of training and validation set
pyplot.plot(history_GRU.history['loss'])
```

```
pyplot.plot(history_GRU.history['val_loss'])
pyplot.title('model loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'val'])
pyplot.show()

#Plot Confusion matrix

ypred_gru = model_GRU.predict_classes(X_test_gru)
ypred_gru = ypred_lstm.reshape(ypred_gru.shape[0], ypred_gru.shape[1])
confusion_metrix_calculation(ypred_gru, y_test)
plt.title('GRU - Gated Recurrent Unit', fontsize=12)
plt.show()

################################################################################
###############
# Performance Evaluation of All Deep Learning Models
# Plot Accuracy of validation set
pyplot.plot(history_ANN.history['val_accuracy'])
pyplot.plot(history_CNN.history['val_accuracy'])
pyplot.plot(history_RNN.history['val_accuracy'])
pyplot.plot(history_LSTM.history['val_accuracy'])
pyplot.plot(history_GRU.history['val_accuracy'])
pyplot.title('All Models Validation Accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['ANN', 'CNN', 'RNN', 'LSTM', 'GRU'])
pyplot.ylim(top=1.1)
pyplot.show()

# Plot Loss of all Deep Learning Models
pyplot.plot(history_ANN.history['val_loss'])
pyplot.plot(history_CNN.history['val_loss'])
pyplot.plot(history_RNN.history['val_loss'])
pyplot.plot(history_LSTM.history['val_loss'])
pyplot.plot(history_GRU.history['val_loss'])
pyplot.title('All Models Validation Accuracy')
pyplot.ylabel('accuracy')
pyplot.xlabel('epoch')
pyplot.legend(['ANN', 'CNN', 'RNN', 'LSTM', 'GRU'])
pyplot.ylim(bottom=-0.2)
pyplot.show()

################################################################################
###################
################################################################################
###################
################################################################################
###################
################################################################################
###################
# # Ensemble Voting Classifier(Hard):
"""
This is proposed classifier which will take multiple(3) prediction and make
neu prediction on majority vote.
-'voting_hard_classifier' is a function that make new prediction. Classic
```

```
voting package has been avoided as
  the input shape of 'X_test' and prediction shape 'y_test' varried by
networks.
-'ypred_ann_cnn_rnn' is a numpy array which contains 3  values of prediction
in a row. For example ([1,1,0],
  [0,0,1]...)
- function will count by rows. give one value for each row. For example
[1,0...]
"""
ypred_ann_cnn_rnn_lstm_gru = np.column_stack((ypred_ann, ypred_cnn,
ypred_rnn, ypred_lstm, ypred_gru))
def voting_hard_classifier(ypred, ytest):
    y_predict = ypred.tolist()
    y_target = ytest.tolist()
    vote_predict = []
    for i in range(len(y_predict)):
        count_0 = 0
        count_1 = 0
        for j in range(len(y_predict[i])):
            if y_predict[i][j] == 0:
                count_0 = count_0 + 1
            elif y_predict[i][j] == 1:
                count_1 = count_1 + 1
        if count_0 > count_1:
            x = 0
            vote_predict.append(x)
        elif count_0 < count_1:
            x = 1
            vote_predict.append(x)
    return vote_predict, y_target
predict_by_vote, target = voting_hard_classifier(ypred_ann_cnn_rnn_lstm_gru,
y_test)
c_for_all_accuracy = confusion_matrix(target, predict_by_vote)  # confusion
matrix
AccV = ((c_for_all_accuracy[0][0] + c_for_all_accuracy[1][1]) /
c_for_all_accuracy.sum())
confusion_metrix_calculation(predict_by_vote, target)

# Accuracy_all_MODEL will have all accuracy value
Accuracy_all_MODEL.append(test_acc)
Accuracy_all_MODEL.append(test_acc2)
Accuracy_all_MODEL.append(test_acc3)
Accuracy_all_MODEL.append(test_acc4)
Accuracy_all_MODEL.append(test_acc5)
Accuracy_all_MODEL.append(AccV)

#Accuracy_all_MODEL

# Accuracy of All Machine Learning and Deep Learning Models

ACCURACY_ALL_MODELS = []

#Plot Accuracy of all machine learning and deep learning models
for number in Accuracy_all_MODEL:
    ACCURACY_ALL_MODELS.append(number * 100)
MODEL = ('Logistic Regression',
         'Support Vector Classifier',
```

```
         'Dession Tree Classifier',
         'Random Forest Classifier',
         'Kneibour Classifier(KNN)',
         'Artificial Neural Network(ANN)',
         'Convolutional Neural Network(CNN)',
         'Reccurent Neural Network(RNN)',
         'Long Short Time Memory(LSTM)',
         'Gated Recurrent Unit(GRU)',
         'Proposed Ensemble Neural Network')
"""
MODEL = ('Artificial Neural Network(ANN)',
         'Convolutional Neural Network(CNN)',
         'Reccurent Neural Network(RNN)',
         'Long Short Time Memory(LSTM)',
         'Gated Recurrent Unit(GRU)',
         'Proposed Ensemble Neural Network')
"""
data_accuracy = list(zip(MODEL, ACCURACY_ALL_MODELS))
df1 = pd.DataFrame(data_accuracy, columns=['Models', 'Accuracy'])
fig = go.Figure()
fig.add_trace(go.Bar(x=df1["Accuracy"], y=df1["Models"], orientation="h"))
fig.update_layout(xaxis_title="Accuracy", yaxis_title="Models")
fig.show()
```