Deep Learning for Prediction of Business Outcomes
Final Project

Credit Card Fraud Detection

Group 30: Ying Chen 502466, Lihan Ni 502585

Washington University in St.Louis Olin Business School

Faculty Advisor: Salih Tutun

Abstract

Credit card fraud has been an ongoing safety issue for financial institutions and merchants around the globe. Historical data showed that the number of suspected fraud digital transactions increased 46% year on year in 2021, and direct losses by merchants and financial institutions exceeded $28.58 billion globally in 2020. Some attempts to apply machine learning techniques to detect card fraud automatically may be deceptive due to a failure to account for fraud sequences and behavioral changes. Our study focused on developing a credit card detection model that employs both supervised learning (Keras Dense Layer) and unsupervised learning (Autoencoder networks) to extract the hidden data points from our given dataset and further detect the Fraud (1) or Non-fraud (0) credit card transactions. The results of our study showed that our model achieved 99.13% and 99.6% fraud detection performance using our dataset of transactions made by credit cards in September 2013 by European cardholders.

Introduction

Financial institutions had the highest monthly fraud attacks in 2021 compared to previous years. According to CNBC, the average number of monthly fraud assaults for banks with more than $10 million in yearly revenue has jumped from 1,977 to 2,320 since 2020. Nonetheless, there were about 2.8 million customers reported the fraud to the agency in 2021, the largest amount in a decade, and approximately 25% of the frauds resulted in a financial loss, with the average customer losing $500, and worst of all, the actual number could be larger due to the fact that some incidents were most likely not reported to the agency (Iacurci, 2022). Given this ongoing concern, our study focused on using deep learning methods to help to reduce lower false positive rates (FPR), chargebacks, and potential fraud.

Compared to machine learning, the use of deep learning and neural networks have two major competitors. One of which is that learning-based models have a larger learning capacity, which refers to their ability to automatically extract more high-level features, learn more sophisticated detection models, and enhance performance with more or larger training data as the size of our database increases (IBM, 2022). A further major benefit is their capacity to deal with a variety of and a large number of input features or characteristics without being overly constrained by the statistical qualities of attributes (Khodayari, 2020). In light of these characteristics, deep learning is an effective way to identify fraudulent or criminal patterns in transactional data. Various implementations of deep learning methods offer end-to-end solutions for highly complicated problems, and this would potentially bring less work on feature extraction or the curse of dimensionality. To further explore our analysis, our study utilized two models: Keras Dense Layer and Autoencoder as models to evaluate the accuracy of credit card fraud detection.

Review of Literature

Fraud detection methods can be broadly categorized as supervised or unsupervised. In the supervised approach, fraud detection is viewed as a binary classification problem, where we are trying to classify if a transaction is legitimate (negative class) or fraudulent (positive class). In an unsupervised approach, fraud detection can be viewed as a system identifying outliers in a dataset, assuming that outlier transactions represent potential fraudulent activities.

Past research has been focusing on using supervised learning to detect fraudulent activities. Singh et al. (2012) proposed the SVM (support vector machine) based method with multiple kernel involvement and the inclusion of several fields of the user profile compared to the exclusively spending profile, and their results showed an improvement in the true positive and true negative, false positive and false negative rates, whereas a decrease in the false positive rate. When using a self-organizing map neural network (SOMNN) technique to solve the problem of optimally classifying each transaction into its associated group given an unknown prior output, Ogwueleka (2011) found that a receiver-operating curve (ROC) for fraud detection watches detected over 95% of fraud cases without causing false signals.

However, researchers have been pointing out that a combination of different models predicts higher accuracy in fraudulent detection. Sohony et al. (2018) used an ensemble model that combines the best of Random Forest and Feed Forward Networks (3 feed-forward neural networks) to accurately detect fraud, and they concluded that unsupervised learning, specifically the ensemble model of Random Forest and Neural Networks have outperformed supervised learning methods. Additionally, Misra et al. (2019) have found that their proposed two-stage model outperforms systems that rely on single classifiers by transforming transaction attributes into lower-dimension feature vectors with an autoencoder as the first stage and then using the feature vector as input to a classifier as the second stage.

Despite significant results mentioned in prior research studies, they failed to emphasize the application of deep learning techniques to the basic classification problem of credit card fraud detection. Our findings are supported by the aforementioned publications, which reveal that a deep learning model can be used to identify credit card fraud when appropriate data is available. When data is sufficient, appropriate preprocessing is performed, and appropriate models are developed, deep learning can be used for identifying fraudulent transactions in a direct and reliable manner. Thus, our study focused on using an autoencoder and dense layer as our research models.

Problem Description

We have chosen to solve the problem of identifying credit card fraud transactions with deep learning techniques, thus the research question of our study is to detect fraudulent transactions using credit card information. Based on historical transaction records, we employed the supervised deep learning model Kera dense layers, and then we employed an unsupervised learning autoencoder to extract the latent(hidden) data points from the dataset.

Model Description

Model 1 - Autoencoder

We use an autoencoder to test the predicting accuracy because it simulates the typical behavior of data and identifies outliers, since the regular trend observed in the vast majority of transactions will be more accurately represented than uncommon occurrences, and therefore, the reconstruction error of "normal" data will be less than that of outlier data.

Model 2 - Dense Layers

We also use Keras dense layers as our second model that stacks these layers together and create calculation outputs = activation(dot(input, kernel) + bias) as our prediction model, and we chose sigmoid and relu as dense layer activation functions to either keep each value above zero or replace it with zero.

Challenges

The challenge of our study is that our dataset is highly unbalanced. Since just a very small percentage of transactions are fraudulent (0.17%). As a result, it is challenging to gather the information necessary to effectively identify fraudulent activities while simultaneously reducing the number of false positives (FPR). In order to solve this issue, we used SMOTE functions to achieve a more equitable distribution of classes by the random multiplication of minority classes. The 28 numerical input variables in our dataset were transformed by principal component analysis (PCA) to reduce dimensionality. Finally, we modify the parameters to determine model sufficiency.

Data and Experimental Results

Data Description

We got our dataset from Kaggle(https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud). The dataset includes credit card transactions performed by European cardholders in September 2013. This dataset contains 492 frauds out of 284,807 transactions that occurred over the course of two days. It has 28 numerical features that are the outcome of a PCA transformation, and "time" and "amount" that are not transformed by PCA. It is worth mentioning that the original 28 variables are not disclosed due to confidentiality. The dataset is very imbalanced, with positive transactions accounting for 0.172% of all transactions.

Data Pre-processing(1)

1. Since "time" and "amount" have not been transformed by PCA, then we use StandardScaler to transform these two features just as other features.
2. In order to get a more stable dataset and rule out the effect of the magnitude of numbers, we normalize all features to 0 - 1 by using the formula x = x - min(x) / max(x) - min(x).

Data Pre-processing(2) - Data Augmentation

The imbalanced dataset we have, 492 fraud cases and 284,315 non-fraud cases, might overestimate the normal non-fraud cases and underestimate the fraud cases, which might make the model less powerful and less persuasive. To balance our dataset, we need to oversample the minority class, which is the fraud cases. We choose to use the **Synthetic Minority Oversampling Technique (SMOTE)** to conduct this process. SMOTE increases the sample size of the minority class in the training dataset by duplicating the samples from the minority class, without adding or removing extra data to/from the dataset. In our project, after using SMOTE, we generated 199,006 fraud cases and 199,006 non-fraud cases in the training dataset.

Model Optimizing and Parameters Tuning

Autoencoder:

Our initial setting of the model has 4 layers: 128, 64, 32, and 16 filters for each; we use 'relu' and 'sigmoid' as the activation function, 'adam' as the optimizer, 'MSE' as the loss function, 'Accuracy' as metrics; we use 30 epochs and 100 batches as the default value.

Here is the summary of our models:

Model	batch size	epoch	Number of Layers	Regularization	dropout	optimizer	training accuracy	test accuracy
1	100	30	4	/	/	adam	0.9787	0.9776
2	100	30	4	/	/	RMSprop	0.9747	0.9778
3	100	30	3(delete layer with 16 filters)	/	/	RMSprop	0.981	0.9563
4	100	30	3(delete layer with 16 filters)	/	/	adam	0.9839	0.9913
5	100	30	5(add layer with 8 filters)	/	/	adam	0.9633	0.9588
6	100	30	5(add layer with 8 filters)	/	/	RMSprop	0.9669	0.9618
7	150	30	4	/	/	adam	0.9811	0.9785
8	150	30	4	/	/	RMSprop	0.9748	0.9715
9	80	30	4	/	/	adam	0.9771	0.9775
10	80	30	4	/	/	RMSprop	0.9788	0.9826

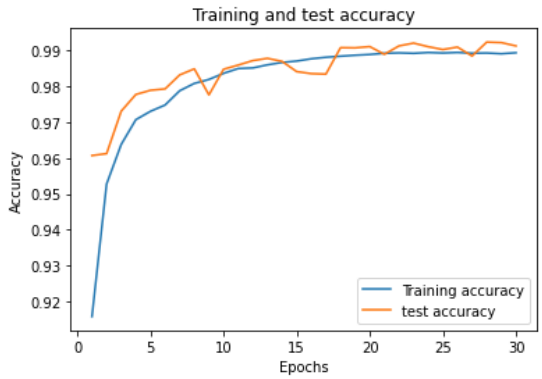
We first tried the initial model with different optimizers "adam" and "RMSprop" (which are the Model 1 and Model 2 in the chart above), then we found that the test accuracies of both models are similar. As we cannot tell which optimizer is better so far, we would try both optimizers in further analysis. The training accuracy and test accuracy are very close, so it means that we don't have an overfitting or underfitting problem here. Then adding dropout layers or regularization is not our primary task here.

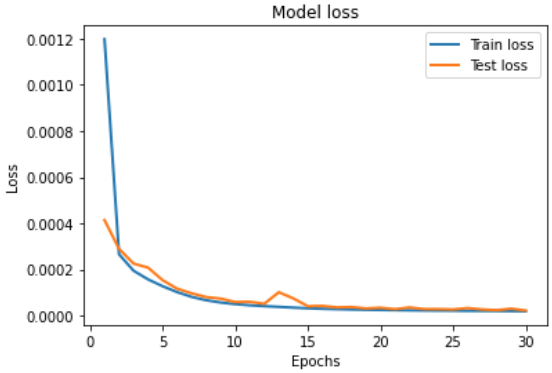
Then we tried to tune the number of layers.

We tried to use fewer layers first, so we removed the layer of 16 filters and kept other parameters unchanged. Then we tried different optimizers "adam" and "RMSprop" for this setting, which are the Model 3 and Model 4 in the chart above. The result shows that Model 3 using RMSprop has lower

accuracy than Model 4 using ‘adam’, and Model 3 has lower accuracy than Model 2 as they both use RMSprop but Model 3 has less layer. We may conclude that RMSprop doesn’t perform as well as ‘adam’ when we have fewer layers, and we should try more layers.

The good news is that we found our best result so far: Model 4 has the highest test accuracy rate at 0.9913: 100 batch, 30 epochs, 3 layers, and ‘adam’ as Optimizer.





Then we tried more layers: we added a layer of 8 filters to the architecture and kept other parameters unchanged. We still tried two optimizers. Then we got the Model 5 and Model 6 as the chart listed above. Both of them performed worse than the initial models (Model 1 and 2), so we may conclude that we don’t need to add more layers to the architecture and 3 or 4 layers is enough for this model.

So far, model 4 is still the best model.

Next step, we tried to tune the batch size.

We increased the batch size to 150 and kept other parameters unchanged, then we got Model 7 and Model 8. From the comparison between Model 1 and Model 7 (as they share the same parameters except Model 7 has larger batch size), we can find that increasing the batch size can improve the test accuracy rate from 0.9776 to 0.9785; From the comparison between Model 2 and Model 8 (as they share the same parameters except Model 8 has larger batch size), we can find that increasing the batch size doesn’t help. There is no impressive improvement as the result of increasing batch size.

Then we decreased the batch size to 80 and kept other parameters unchanged, then we got Model 9 and 10. The result of Model 9 doesn’t show any improvement, but the result of Model 10 shows us the second highest accuracy rate at 0.9826 among all models.

Overall, after all attempts, Model 4 is the best model we have, followed by Model 10.

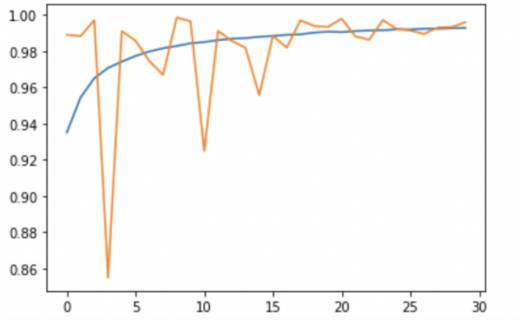
Dense Layers

We started from the model of 3 dense layers: 128, 32, 8 filters of each; we use ‘relu’ and ‘sigmoid’ as the activation function, ‘RMSprop’ as the optimizer, ‘MSE’ as the loss function, ‘Accuracy’ as metrics; we use 30 epochs and 80 batches as the default value.

The basic model has the test accuracy at 0.996, which is very high. We visualized the result below.

```
import matplotlib.pyplot as plt
# Plot loss, val_loss
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.imshow
```

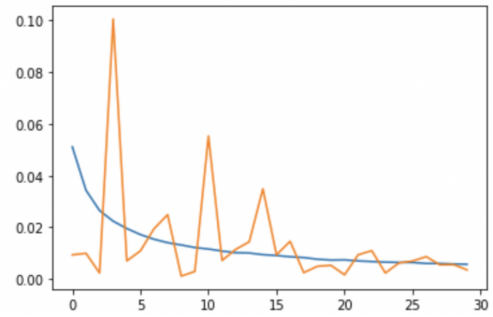
```
<function matplotlib.pyplot.imshow(X, cmap=None,
vmax=None, origin=None, extent=None, shape=<depr
parameter>, resample=None, url=None, *, data=Nor
```



Training accuracy vs. Validation accuracy

```
import matplotlib.pyplot as plt
# Plot loss, val_loss
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.imshow
```

```
<function matplotlib.pyplot.imshow(X, cmap=None,
vmax=None, origin=None, extent=None, shape=<depr
parameter>, resample=None, url=None, *, data=Nor
```

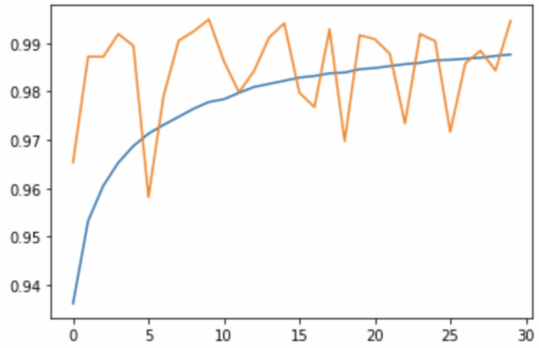


Training loss vs. Validation loss

After adding **dropout** (batch_size= 80, epochs=30) to our model, the model’s accuracy is 0.994.

```
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.imshow
```

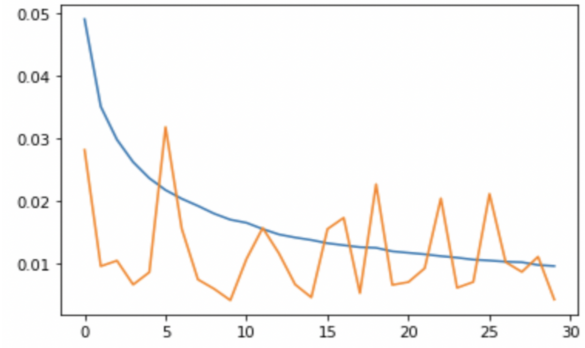
```
<function matplotlib.pyplot.imshow(X, cmap=None
vmax=None, origin=None, extent=None, shape=<dep
parameter>, resample=None, url=None, *, data=No
```



Training accuracy vs. Validation accuracy

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.imshow
```

```
<function matplotlib.pyplot.imshow(X, cmap=None
vmax=None, origin=None, extent=None, shape=<dep
parameter>, resample=None, url=None, *, data=No
```

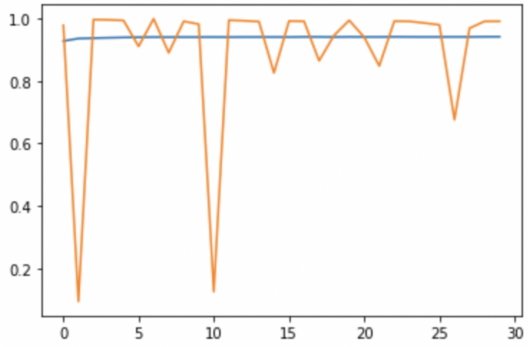


Training loss vs. Validation loss

After adding **L2 Weight Regularization**, the model's accuracy is 0.990.

```
# same as above, plot accuracy, val_accuracy
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.imshow
```

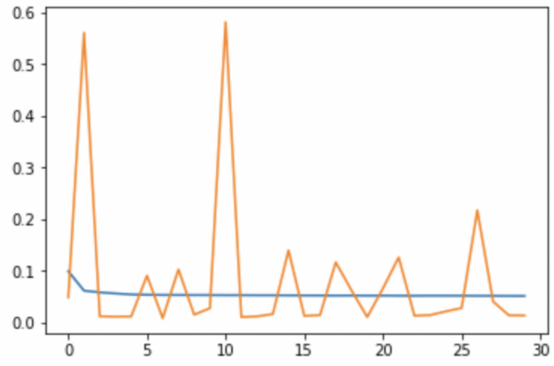
```
<function matplotlib.pyplot.imshow(X, cmap=Non
vmax=None, origin=None, extent=None, shape=<de
parameter>, resample=None, url=None, *, data=N
```



Training accuracy vs. Validation accuracy;

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.imshow
```

```
<function matplotlib.pyplot.imshow(X, cmap=None,
vmax=None, origin=None, extent=None, shape=<depre
parameter>, resample=None, url=None, *, data=None
```

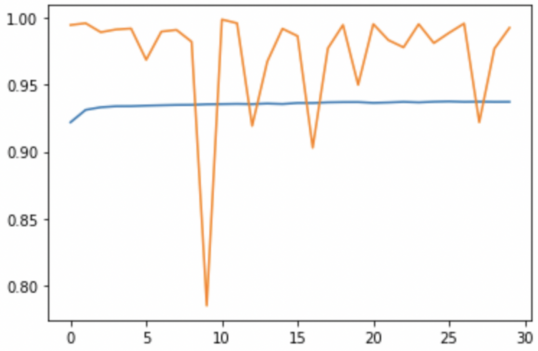


Training loss vs. Validation loss

After adding **L2 and dropout together,** we had the accuracy of 0.992.

```
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.imshow
```

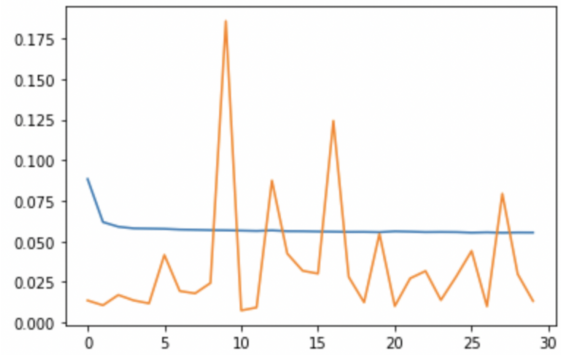
```
<function matplotlib.pyplot.imshow(X, cmap=None
vmax=None, origin=None, extent=None, shape=<dep
parameter>, resample=None, url=None, *, data=No
```



Training accuracy vs. Validation accuracy;

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.imshow
```

```
<function matplotlib.pyplot.imshow(X, cmap=None
vmax=None, origin=None, extent=None, shape=<dep
parameter>, resample=None, url=None, *, data=No
```



Training loss vs. Validation loss

Compared to the basic model's accuracy, the performance somehow went slightly worse, and this is potentially due to the fact that adding regularization to an already low model will only lower its capacity further. Further, in the line plots we created of the model accuracy on the train and test sets, we see a potential overfitting model pattern of inconsistent test accuracy that rises to a point and then declining. This could be potentially because the dense layer neural network is a simple model but we got a large and complicated dataset. For future study, we can try the following methods: 1. Model re-evaluation, including multiple fits and evaluations, and reporting mean and standard deviation of performance 2. Continue the training of a fit model with increasing levels of regularization.

Conclusion

We used and compared two models, Keras Dense Layers and Autoencoders model. The final test accuracy for the Keras Dense Layers was the result of the basic model at 0.996, for the Autoencoder it was 0.991. To be more specific, in the Keras Dense Layers, it is about 99.6% accurate to spot a fraud transaction, while in the Autoencoder, it is about 99.1% accurate to spot a fraud transaction. Although the Dense Layers model has a slightly higher accuracy rate, we still suggest our clients use the Autoencoder model to rule out potential fraudulent transactions because Autoencoder is a more reliable model that can better handle a complicated dataset.

Despite the significant level of predicting accuracy we received in our models, several recommendations could be made to further achieve experiment reliability and validity. The computer is able to learn the features of fraud detection and classify transactions, and thus the low error rate is due to the clarity of the data. However, this can also lead to a problem with low variance and high bias. The data set demonstrates a clear pattern of fraud, but to improve the model, we recommend attempting to reduce the number of features used by the neural network, increasing the variability of the data, or having a more balanced dataset (e.g: introducing more statistical noises). Besides, we might further conduct anomaly detection or add a confusion matrix to evaluate our model. It is also important to note that the data set used in this analysis was from 2013 in the EU region and financial technology has progressed significantly since then, and thus in our future study, we would like to select datasets covering a variety of regions and updated time to adjust our model.

Our model is effective for solving the business problem, because using the Autoencoder model, we have tested that model 4 has the highest test accuracy rate at 99.13%: 100 batch, 30 epochs, 3 layers, and ‘adam’ as optimizer, whereas using the Keras Dense Layers, we have test accuracy rate of 99.6% to spot a fraud transaction. The accuracy rates of our models will enable financial institutions to significantly screen out fraudulent transactions and reduce the financial costs that compensates for this loss. For future study, we can look at alternative datasets to further examine our test accuracy in identifying fraudulent activities.

References

Iacurci, G. (2022, February 22). *Consumers lost $5.8 billion to fraud last year - up 70% over 2020*. CNBC. Retrieved December 18, 2022, from https://www.cnbc.com/2022/02/22/consumers-lost-5point8-billion-to-fraud-last-year-up-70percent-over-2020.html

IBM Cloud Education. (n.d.). *What is deep learning?* IBM. Retrieved December 19, 2022, from https://www.ibm.com/cloud/learn/deep-learning

Khodayari, A. (2020, November 15). *Deep learning for fraud detection in retail transactions*. Medium. Retrieved December 19, 2022, from https://medium.com/walmartglobaltech/deep-learning-for-fraud-detection-in-retail-transactions-564d31e5d1a3

Singh, Gajendra & Gupta, Ravindra & Rastogi, Ashish & Chandel, Mahiraj & Ahemad, Riyaz. (2012). A Machine Learning Approach for Detection of Fraud based on SVM. International Journal of Scientific Engineering and Technology. 1. 194.

Ogwueleka, Francisca. (2011). *Data mining application in credit card fraud detection system*. Journal of Engineering Science and Technology. 6. 311-322.

Sohony, Ishan & Pratap, Rameshwar & Nambiar, Ullas. (2018). *Ensemble learning for credit card fraud detection*. 289-294. 10.1145/3152494.3156815.

Sumit Misra, Soumyadeep Thakur, Manosij Ghosh, Sanjoy Kumar Saha. (2020). *An Autoencoder Based Model for Detecting Fraudulent Credit Card Transaction*. Procedia Computer Science. 167. 254-262. ISSN 1877-0509. https://doi.org/10.1016/j.procs.2020.03.219.