Credit Card Fraud Detection

Abstract

This paper deals with detecting instances of credit card fraud using a publicly available credit card transaction dataset. The problem statement and the dataset is part of a recently conducted Kaggle competition. It is important for credit card companies to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase[https://www.kaggle.com/mlg-ulb/creditcardfraud/home]. The dataset consists of information about more than two hundred and eighty thousand credit card transactions made in September 2013 by European cardholders. Each transaction in the training dataset is labeled as genuine or fraudulent. This is an interesting problem as it involves a highly unbalanced dataset and a lot of anonymized features. These can be used to fit an appropriate learning classifier and classify the transaction type. We used neural networks and autoencoders to classify the transactions in this project. We have contributed a couple of models to identify credit card fraud. Additionally, we bring out the merits and demerits of each approach with a detailed analysis of the results.

Keywords: Credit card fraud, Neural Networks, Autoencoder, Variational Autoencoder, Kaggle

1 Introduction

Credit card fraud refers to theft and/or fraud committed using a credit or debit card as a fraudulent source of payment.["Credit card fraud - Wikipedia." https://en.wikipedia.org/wiki/Credit_card_fraud.] Credit card fraud is a major ethical issue faced in the credit card industry. Stolen credit cards, identity theft, phishing attacks and credit card skimming are a few examples of events leading to credit card fraud. It is estimated that around 0.1% of all card transactions are fraudulent. Although this is a small percentage, it still results in huge financial losses every year due to the massive number of transactions. Credit card fraud affects the customers as well as the credit card merchants. It causes harm and inconvenience to the victims who have to dispute these charges. It can also negatively affect a card owner's credit score. It affects card merchants as it leads to a loss of resources and revenue since they have to take the responsibility for frauds in most cases. It may also lead to a loss of business because customers who have had frauds committed against them may seek alternate payment methods.

In this project, we use TensorFlow and Keras to build three automated models to detect instances of credit card fraud. The first model is a Deep Neural Network (DNN) using 5 Densely connected layers. The second approach uses an Autoencoder to learn a representation of the data in order to reduce the number of dimensions. The third and final approach uses a Variational Autoencoder (VAE), which is a generative model containing an encoder, decoder and a custom loss function. We work with the data provided in the Kaggle competition titled "Credit Card Fraud Detection: Anonymized credit card transactions labeled as fraudulent or genuine".

Section II of this paper describes the dataset used in this project. Section III includes a detailed description of the proposed methodology. Section IV highlights some of the evaluation metrics commonly used in fraud detection. Section V brings out the results discovered over the course of this project while sections VI and VII talks about the future work and conclusion respectively.

2 Dataset description

This dataset from Kaggle[https://www.kaggle.com/mlg-ulb/creditcardfraud] contains transactions made by credit cards in September 2013 by European cardholders. The dataset has been collected and analysed during a 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. It includes transactions that occurred in two days, with 492 instances of fraud out of 284,807 transactions. The dataset is highly unbalanced as the positive class (frauds) account for only 0.172% of all transactions in the dataset. This dataset has one comma-separated value (CSV) file named creditcard.csv. This CSV file has 30 feature columns and one column containing the class of the transaction. The feature columns include time, amount and 28 other features named V1-V28. We do not have any additional information regarding these 28 features owing to confidentiality issues. These features have all been transformed using Principal component analysis (PCA). The feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount which can be used for example-dependant cost-sensitive learning. The output label 'Class' identifies the type of transaction taking the value 1 in case of fraud and 0 if the transaction is genuine.

3 Data analysis

In this section, we provide a more detailed description of data analysis techniques used in the project.

3.1 Preprocessing and analysis

Analyzing and cleaning data is an important first aspect of any data analytics project. In this section, we summarize the details and statistics of the dataset along with required preprocessing. The total dataset includes 284,807 transactions. Of these, 284,315 were genuine transactions while the remaining 492 were fraudulent transactions. We observe that none of the features in this dataset require interpolation as there are no missing or NaN values.

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67	0
4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99	0
7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80	0
7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20	0
9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68	0

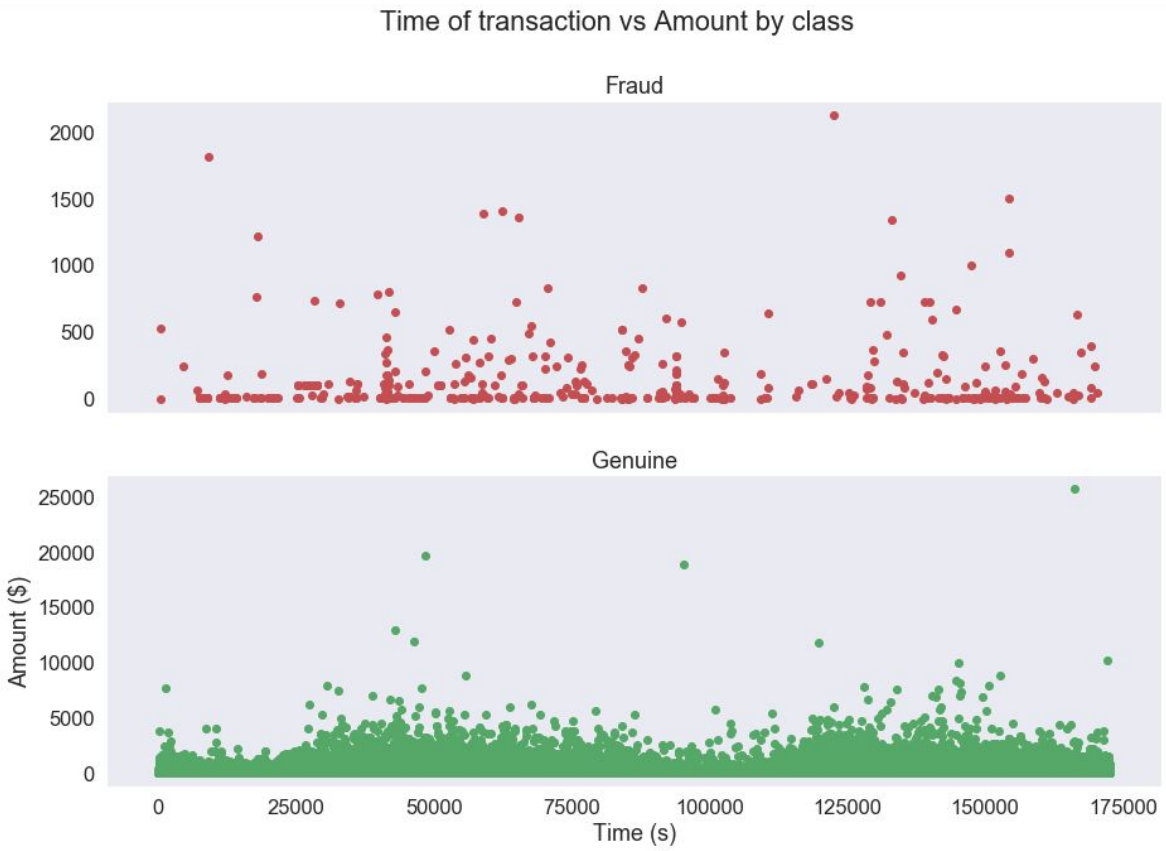
A summary of the data

3.2 Data Visualization

In this section, we conduct an exhaustive analysis of the dataset to draw some interesting visual observations.

In Figure 1, we observe how unbalanced the data is, with the positive class (frauds) accounting for only 0.172% of all transactions.

Figure 2 shows the time of transaction vs amount plot for fraudulent and genuine transactions. Based on this graph, we can infer that fraudulent transactions are usually smaller than genuine ones, presumably to avoid being flagged by fraud detection systems. There seems to be no real correlation between time of transaction and class of transaction since most of them are fairly evenly distributed across the entire 2 day window. We thus exclude the time of transaction before running it through our models.



	Genuine	Fraud
Count	284,315	492
Mean	88.29	122.21
Standard deviation	250.11	256.68
First quartile	5.65	1.00
Median	22.00	9.25
Third quartile	77.05	105.89
Maximum	25691.16	2125.87

Table: Measure of statistical dispersion for transaction amounts based on class

The above table provides a deeper insight into the data. Fraudulent transactions appear to be smaller based on the figure, but the average fraud transaction is for a larger amount than the average genuine transaction. This is because the number of transactions is much higher for the genuine class. In fact, the median fraud transaction amount is just $9.25. The few fraudulent transactions over $1,000 plays a huge role in increasing the overall fraudulent transaction

average to $122.21. Thus, the median is clearly a better indicator of central tendency for this dataset.

3.3 Feature engineering

As mentioned earlier, the 28 features labeled V1-V28 were transformed using PCA. We verify that all 28 features are scaled to a normal distribution having zero mean and unit variance. We apply a similar scaling to the transaction amount feature to ensure that all features are on the same scale before training our models.

4 Methodology

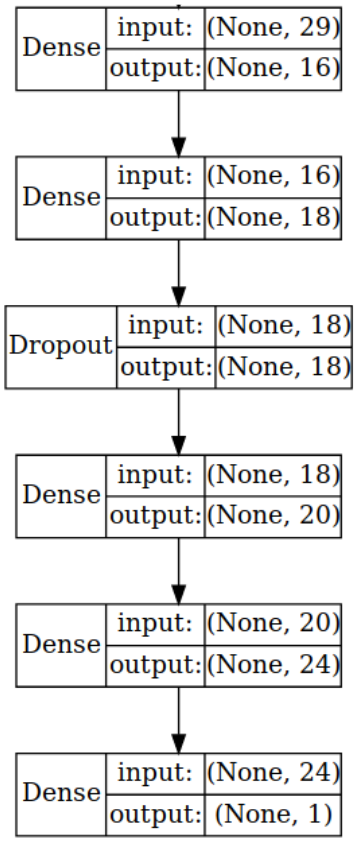
In this section, we describe and outline the various models used in this project. A train-test split of 80-20 was used in all models. The batch size was set to 16 and all models were run for 100 epochs. The loss function used was binary cross-entropy since there are only two output labels, genuine and fraud. Additionally, we made use of callback functions provided by Keras' API to dynamically improve our training. We used the ModelCheckPoint callback to save the model after every epoch. The ReduceLROnPlateau callback was used to reduce the learning rate by a factor of 5 if the validation loss plateaued. EarlyStopping callback was used to stop training after the validation loss plateaued for more than 10 epochs.

We ran several models with different hyperparameters for each of the three approaches but for brevity, only the best performing ones are described in the below subsections.

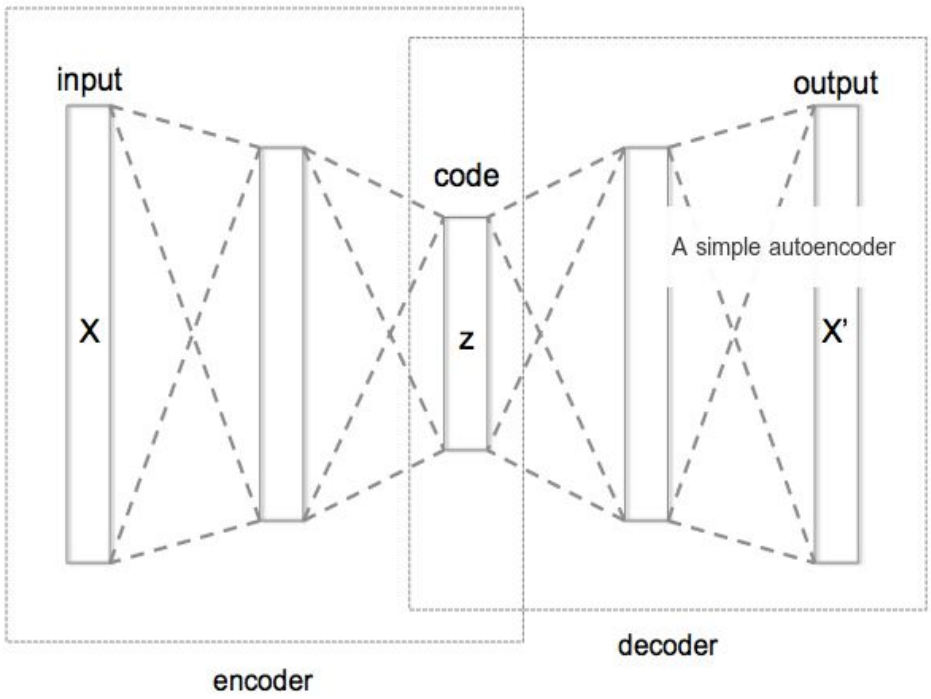
4.1 Deep Neural Networks

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output.[https://en.wikipedia.org/wiki/Deep_learning] A neural network can be thought of as an approximation of some function that is to be modeled.

To build a DNN model, we used Keras' sequential model API. Our model had 5 dense layers. The first four layers used a ReLU activation function while the final layer used a sigmoid activation function. We introduced a dropout layer between layers 2 and 3. The dropout probability was set to 25%. Dropout is a regularization technique used to prevent overfitting. The features V1-V28 and the scaled amount were the inputs to this DNN. We used the learning rate optimizer Adaptive Moment Estimation (Adam) with an initial learning rate of 0.001.



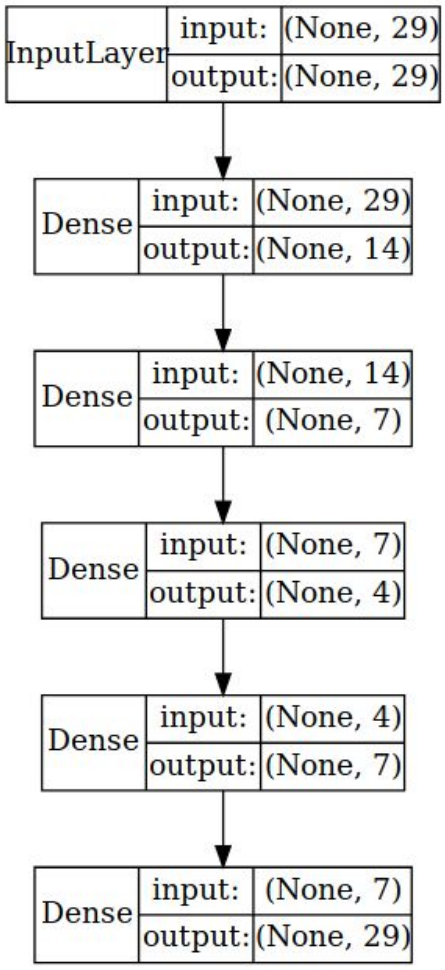
4.2 Autoencoder



Source: https://hsaghir.github.io/data_science/denoising-vs-variational-autoencoder/

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction.[https://en.wikipedia.org/wiki/Autoencoder] A typical autoencoder can usually encode and decode data very well with low reconstruction error. [https://hsaghir.github.io/data_science/denoising-vs-variational-autoencoder/]. Reconstruction loss measures how similar the encoded and then decoded data is compared to the original data.

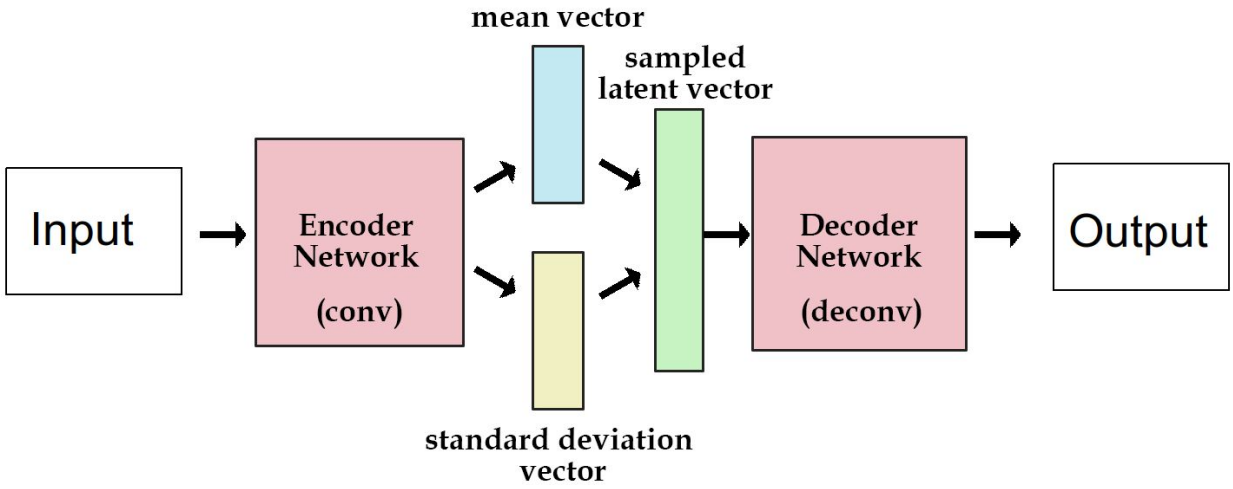
To build the autoencoder, we used Keras' functional model API. We used 3 hidden layers each for the encoder and the decoder. The features V1-V28 and the scaled amount were the inputs to the encoder. As a result, the first input layer had 29 units. In order to compress the data and reduce the dimensions, we reduced the number of units for each subsequent layer by half. The encoder layers had 14, 7 and 4 input units as a result of this dimensionality reduction. The input units to the decoder mirrored those of the encoder and the output of the decoder had the same number of units as the input to the encoder, 29. The output of this was fed into a neural network in order to classify the transactions. The learning rate optimizer used in this model was Stochastic Gradient Descent (SGD), with an initial learning rate of 0.015 and a learning rate decay of 0.001.



4.3 Variational Autoencoder

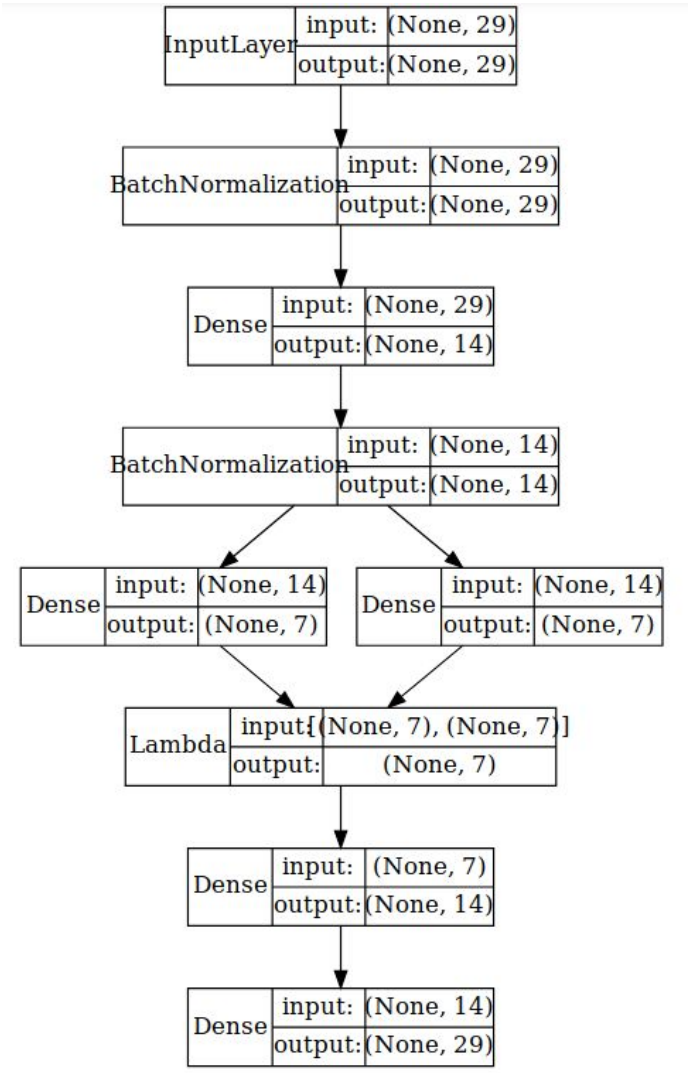
$$\log P(X) - K_{DL}(Q(z|X)||P(z|X)) = E_{z\sim Q(z|X)}[\log P(X|z)] - K_{DL}(Q(z|X)||P(z))$$

A Variational Autoencoder (VAE) improves upon a traditional autoencoder by learning the probability distribution of the data. Unlike an autoencoder, a VAE can be used to generate data since they explicitly define a probability distribution on the latent code. The architecture is very similar to a regular autoencoder but the difference is that the hidden code comes from a probability distribution that is learned during the training.[https://hsaghir.github.io/data_science/denoising-vs-variational-autoencoder/]. In case of a VAE, we sum up two separate losses: the generative loss, which is a mean squared error that measures how accurately the network reconstructed the images, and a latent loss, which is the Kullback–Leibler (KL) divergence that measures how closely the latent variables match a unit gaussian. [http://kvfrans.com/variational-autoencoders-explained/] The KL divergence can be optimized by generating a vector of means and standard deviations instead of generating actual values.



[http://anotherdatum.com/vae.html]

As was the case with the Autoencoder, we also used Keras' functional model API to construct the VAE. We used 3 hidden layers each for the encoder and the decoder. The first input layer had 29 units, the features V1-V28 and the scaled amount. This was passed through a BatchNormalization layer in order to maintain the mean activation close to 0 and the activation standard deviation close to 1. The BatchNormalization output was passed into the encoder with 14 units. This was followed by another BatchNormalization transformation. To represent the latent dimensions, we used Dense layers with 7 units. We used the optimizer Adam with an initial learning rate of 0.001. Additionally, we monitored the reconstruction loss and the KL loss to ensure that the learning does not plateau.



	DNN	Autoencoder	VAE
Trainable parameters	1,695	824	1,263
Non-trainable parameters	0	0	86
Total parameters	1,695	824	1,349

Parameter comparison - DNN vs Autoencoder vs VAE

5 Evaluation Metrics

Since we only classified the data points into two categories, fraud transaction and genuine transaction, it is not trivial to evaluate our models. Traditionally, a classifier can be evaluated by its accuracy, which is the number of data points that are correctly classified divided by the total number of data points. However, this approach is not suitable to evaluate an unbalanced binary dataset such as the one used in this project. For instance, if we had a classifier that classifies all transactions as genuine, we would get an accuracy of more than 99% as less than 0.2% of the transactions in our dataset are considered as fraudulent transactions. Despite the high accuracy, we can easily understand this model will not be what we actually wanted. Therefore, we made use of other evaluation metrics such as the Receiver Operating Characteristics (ROC), the Recall vs Precision curve, and the confusion matrix.

5.1 Receiver Operating Characteristics

The ROC curve is a plot commonly used to evaluate binary classifiers. It shows the false positive rate on the x-axis, and the true positive rate on the y-axis. *Figure x.x* (**TODO: label figures, and use the label of an ROC from results here)** shows an example ROC curve. Each point on the curve represents the false positive rate vs. true positive rate at a given discrimination threshold.

We assess each ROC curve by a metric called Area Under the Curve (AUC). It denotes the total area under the curve in the plot. A higher AUC highlights a more accurate binary classifier. AUC ranges from 0 to 1; an AUC of 1 signifies a perfect binary classifier.

5.2 Recall vs. Precision

Similar to the ROC curve, the Recall vs. Precision curve is another common way to evaluate binary classifiers. Recall and Precision are defined using the formulae below.

$$\text{Precision} = \frac{true\ positive}{true\ positive + false\ positive}$$

$$\text{Recall} = \frac{true\ positive}{true\ positive + false\ negative}$$

Each point in the curve represents a certain discrimination threshold, and we also evaluate those curves by the Area Under the Curve.

5.3 Confusion Matrix

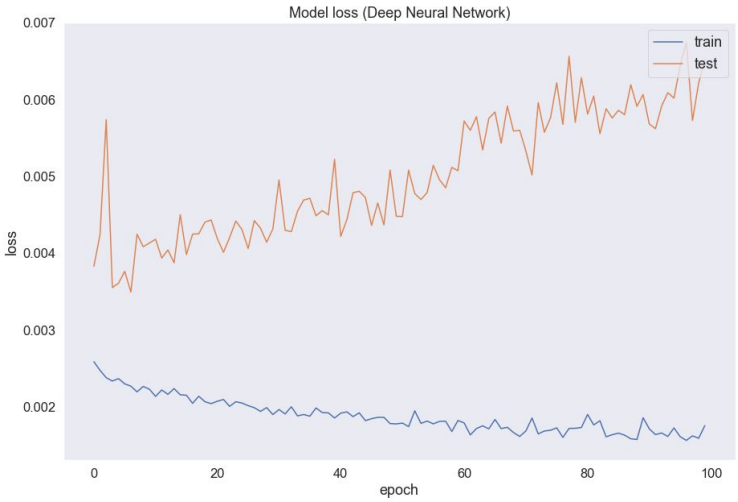
In the confusion matrices, we show the actual numbers of true positive, true negative, false positive, and false negative. The true positives are the number of frauds that are correctly classified as frauds while false negatives are number of frauds classified as genuine. True negatives are more important to us, since we want to identify as many fraudulent transactions as possible and we do not want to incorrectly classify any fraudulent transaction as a genuine one. False positives, the number of genuine transactions that are classified as fraud, have a small impact to the credit card users, as they will need to call their banks to authorize those transactions instead of losing money to a suspected fraud.

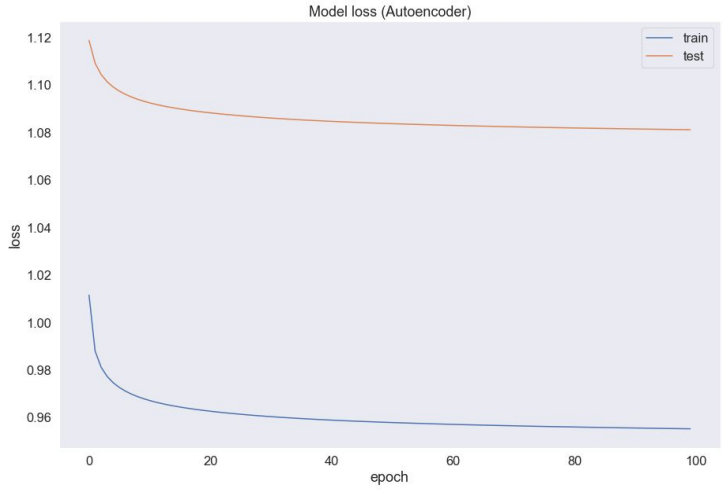
6 Results

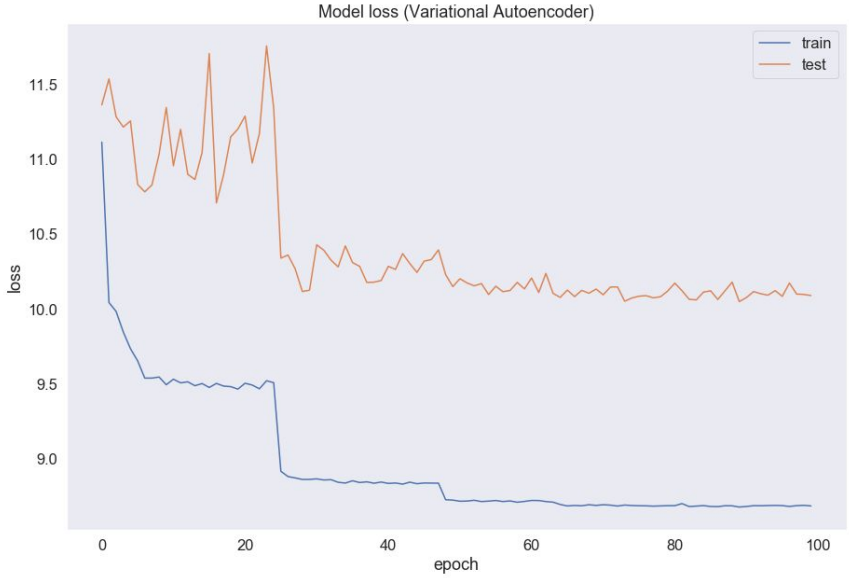
In *Figures xx - xx* and *Table xx - xx* [**TODO: label figures]**, we show the training process, the ROC curves, the recall vs.precision curves, and the confusion matrices for the three models, pure DNN, Autoencoder with DNN, and VAE with DNN, respectively.

6.1 Training Process

Figures xx - xx show the training process of the three models. It shows the training and testing loss on the y-axis and the number of epochs on the x-axis. For pure DNN, we can find out the training process is not taking any effect since there is no direct relationship between the loss and the number of epochs. As for traditional autoencoder, the training loss is continuously dropping smoothly with increase in the number of epochs. Therefore, the autoencoder does improve the learning process of the model. For the VAE, although less smooth, the loss is dropping. Based on these graphs, we can predict that the autoencoder and VAE is likely to outperform the pure DNN approach.

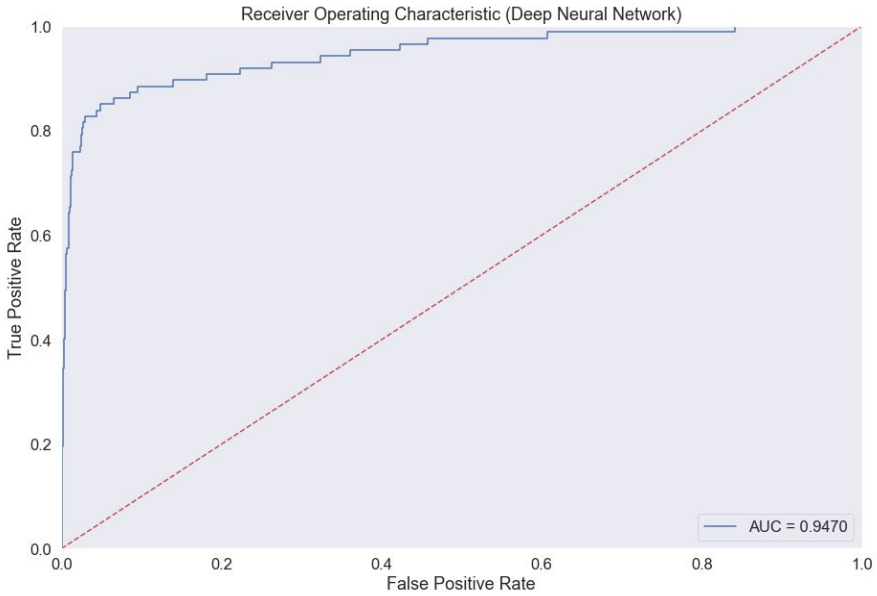

Model loss (Deep Neural Network)
train
test
loss
epoch
0.007
0.006
0.005
0.004
0.003
0.002
0
20
40
60
80
100

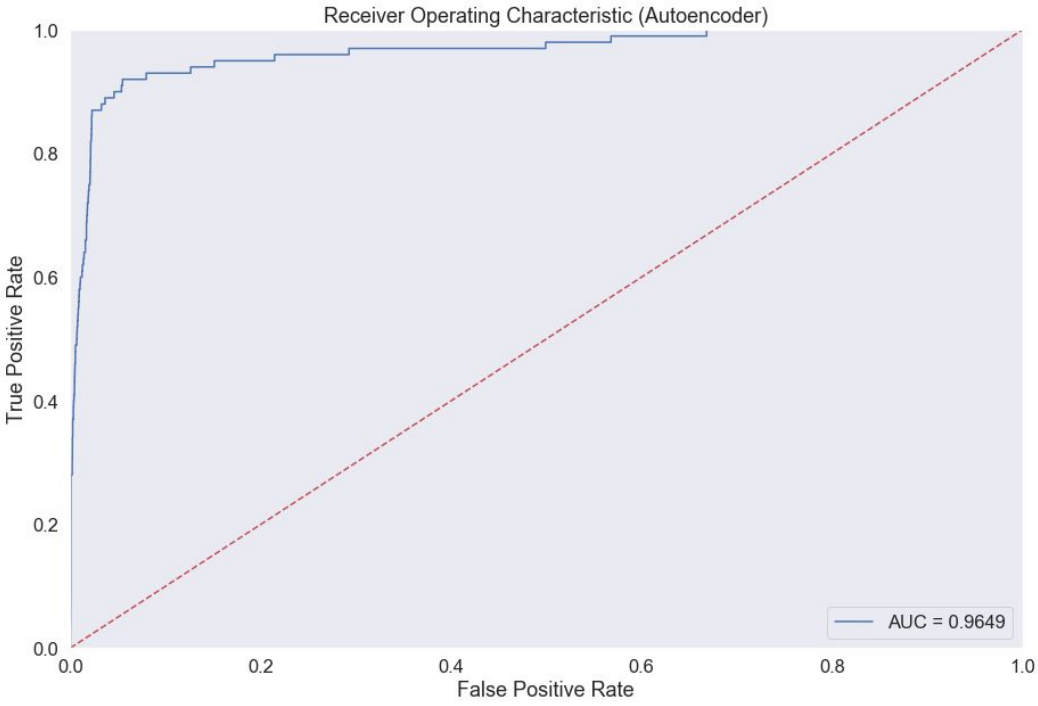

Model loss (Autoencoder)
train
test
loss
epoch
1.12
1.10
1.08
1.06
1.04
1.02
1.00
0.98
0.96
0
20
40
60
80
100

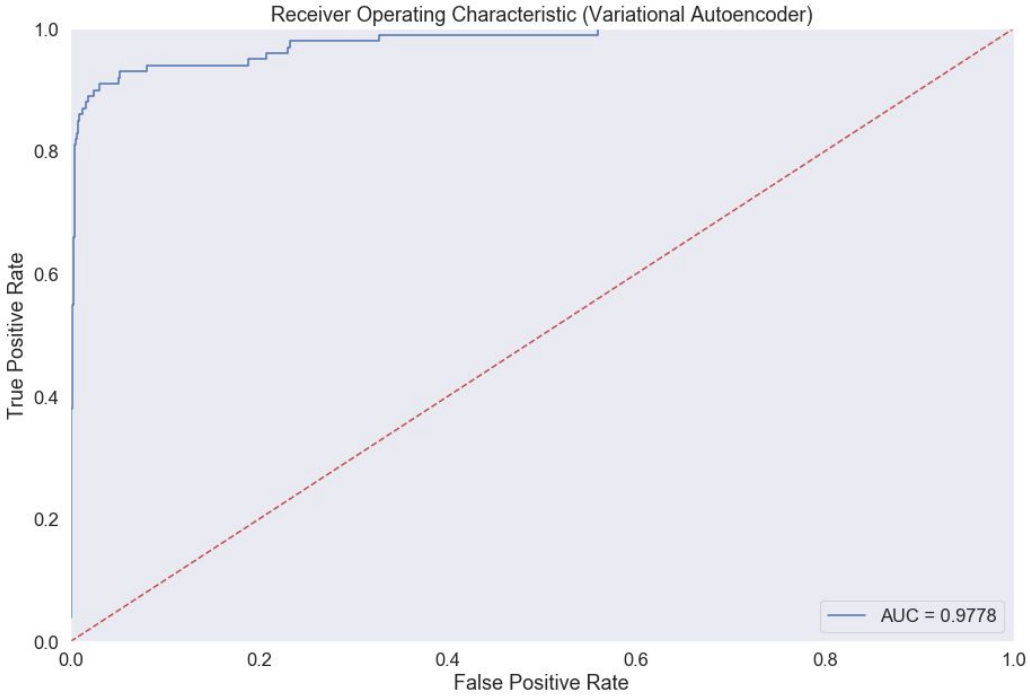

Model loss (Variational Autoencoder)
train
test
loss
epoch
11.5
11.0
10.5
10.0
9.5
9.0
0
20
40
60
80
100

6.2 ROC Curves

Figure x.x to *Figure x.x* shows the ROC curves of those three models. The AUC value is labeled in the bottom right corners of those plots. As we can see, Variational Autoencoder has the highest AUC (0.9778) in those curves, while the pure DNN method has the lowest AUC (0.947). The AUC for Autoencoder (0.9649) falls in the middle, but it is closer to the Variational Autoencoder. These curves and values verifies our prediction in section 6.1. Both autoencoders perform better than the pure DNN, and their performance is fairly similar.

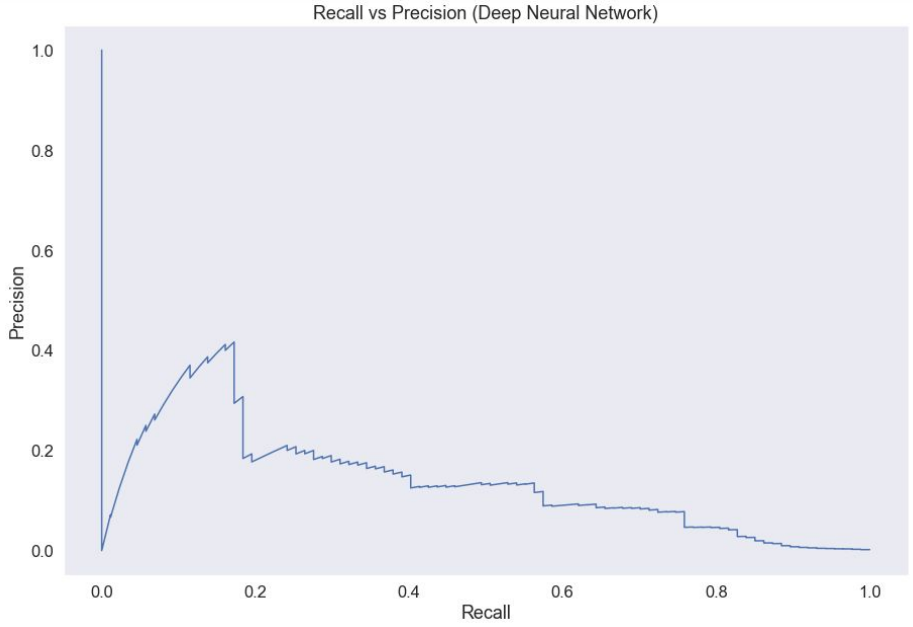


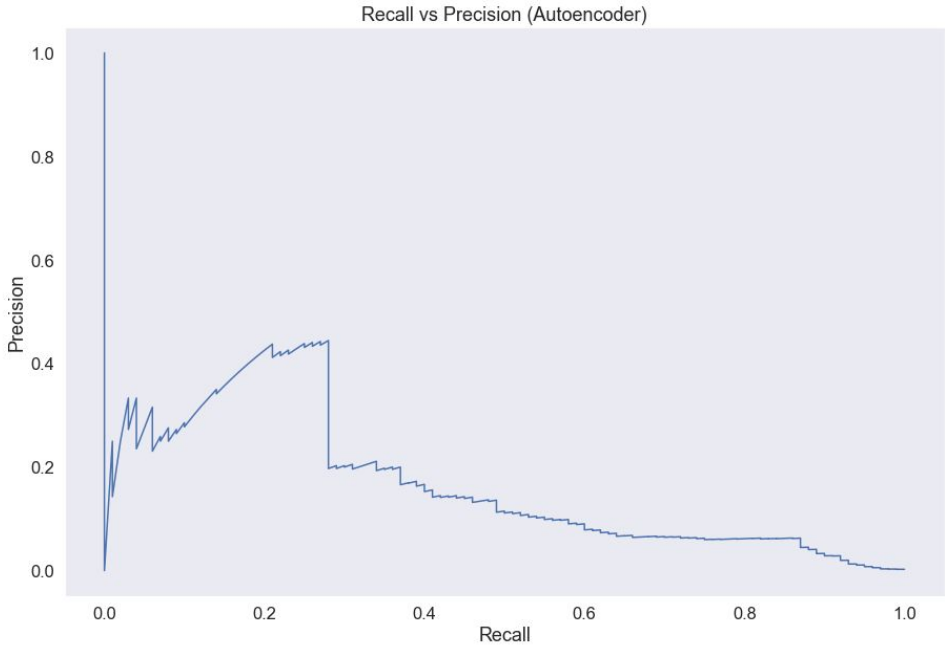

Receiver Operating Characteristic (Autoencoder)
True Positive Rate
False Positive Rate
AUC = 0.9649
0.0
0.2
0.4
0.6
0.8
1.0

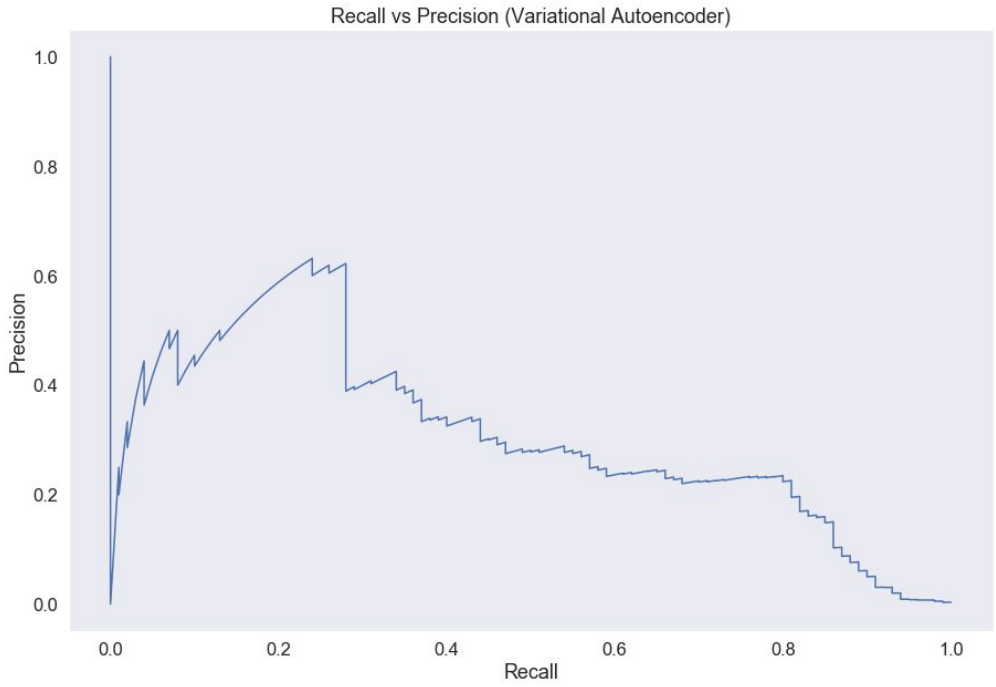

Receiver Operating Characteristic (Variational Autoencoder)
True Positive Rate
False Positive Rate
AUC = 0.9778
0.0
0.2
0.4
0.6
0.8
1.0

6.3 Recall vs. Precision Curves

The trends in Recall vs. Precision curves are similar to what we observed from the ROC curves. The curve of variational autoencoder has the highest AUC value, and the curve of pure DNN has the lowest AUC value. From Recall vs. Precision Curves, we can also draw the conclusion that using autoencoders will produce better results in the problem of credit card fraud detection.







6.4 Confusion matrices

As mentioned in Section 5, number of true positive data points, fraud predicted as fraud, and number of false negative data points, fraud predicted as genuine, are the most important information in the confusion matrices for this problem. As shown below, the traditional autoencoder has the highest number of true positive data points and lowest number of false negative data points. The VAE performs slightly worse than traditional autoencoder, and both autoencoders outperform the DNN.

Dense Neural Network			
True Class	Genuine	55,177	1,698
	Fraud	16	71
		Genuine	Fraud
	Predicted Class		

Autoencoder			
True Class	Genuine	54,856	2,006
	Fraud	12	88
		Genuine	Fraud
	Predicted Class		

Variational Autoencoder			
True Class	Genuine	56,450	412
	Fraud	17	83
		Genuine	Fraud
	Predicted Class		

6.5 Summary of Results

We have used the evaluation metrics mentioned in section 5 to compare all 3 models. From the results, we have observed that the Autoencoder and the VAE both outperform the DNN. This shows the effectiveness in using encoding and decoding layers to detect fraud. Moreover, the performance differences between autoencoder and variational autoencoder is really small, while traditional autoencoder performs slightly better than variational autoencoder.

7 Future Work

To further improve the performance of our model and the accuracy of our evaluation metrics, we can test our algorithm on larger and more recent datasets. Since the dataset we used only contains data from European credit card holders, we could find other datasets from other parts of the world such as the United States to analyze how our model performs.

Different classification models can also be explored. In this project, we used simple Dense layers in the Neural Network used for classification. We can attempt more complex Neural Network and layer structures such as Recurrent Neural Network (RNN) and Long Short-term Memory (LSTM).

8 Conclusion

In this project, we built three different machine learning models to perform credit card fraud prediction using the Kaggle dataset [x]. The models include a VAE with a deep neural network classifier, an Autoencoder with a deep neural network classifier, and a deep neural network classifier without encoding the data. We found that using an autoencoder or a VAE detects credit card fraud better than only using the DNN. We also conclude the traditional autoencoder performs slightly better than the variational autoencoder.

References

TODO: Add reference number to references in the paper.

L. Delamaire, H. Abdou, J. Pointon. (2009). Credit card fraud and detection techniques: A review. Banks and Bank Systems.

W. Yu & N. Wang, "Research on Credit Card Fraud Detection Model Based on Distance Sum," 2009 International Joint Conference on Artificial Intelligence, Hainan Island, 2009, pp. 353-356.doi: 10.1109/JCAI.2009.146

Z. Masoumeh, K.R. Seeja & A. Alam (2012). Analysis on Credit Card Fraud Detection Techniques: Based on Certain Design Criteria. International Journal of Computer Applications. 52. 35-42. 10.5120/8184-1538.

S. Yuan, X. Wu, J. Li & A. Lu. (2017). Spectrum-based deep neural networks for fraud detection.

X. Lu, W. Yu, T. Luwang, J. Zheng, X. Qiu, J. Zhao, L. Xia & Y. Li. (2018). Transaction Fraud Detection Using GRU-centered Sandwich-structured Model. 467-472. 10.1109/CSCWD.2018.8465147.

http://news.mit.edu/2018/machine-learning-financial-credit-card-fraud-0920

http://usir.salford.ac.uk/2595/1/BBS.pdf

Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra. “Stochastic backpropagation and approximate inference in deep generative models.” arXiv preprint arXiv:1401.4082 (2014).

Doersch, Carl. “Tutorial on variational autoencoders.” arXiv preprint arXiv:1606.05908 (2016).

https://blog.keras.io/building-autoencoders-in-keras.html

Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." https://arxiv.org/abs/1312.6114

https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea

9f6c