A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

ASSIGNMENT 2

CT100-3-M-DL

DEEP LEARNING

HAND OUT DATE: 15 FEBRUARY 2021

HAND IN DATE: 05 MAY 2021

WEIGHTAGE: 60%

INSTRUCTIONS TO CANDIDATES:

1 Submit your Assignment Online via Webspace

2 Students are advised to underpin their answers with the use of references (cited using the Harvard Name System of Referencing)

3 Late submission will be awarded zero (0) unless Extenuating Circumstances (EC) are upheld

4 Cases of plagiarism will be penalized

5 You must obtain 50% overall to pass this module

6 SUBMISSION: The TWO (2) Assessments. Corresponding deadlines for are as enclosed.

Version Online-editable-MCO

DEEP LEARNING –
Cover Sheet

ASSIGNMENT

Instruction:

- Marks will be awarded for good report and thoroughness in your approach.
- Referencing Code: If you use some code, or ideas for code, which are taken or adapted from another source (book, magazine, internet, discussion forum, etc), then this **must** be cited and referenced using the Harvard Name convention within your source code. Failure to reference code properly is considered as plagiarism.
- Complete this cover sheet and attach it to your project.
- This project is to be attempted by an individual student.

Student declaration:

I declare that:
- *I understand what is meant by plagiarism*
- *The implication of plagiarism has been explained to me by our lecturer*
- *This project is all my work and I have acknowledged any use of the published or unpublished works of other people.*

Student Signature: ***Kyle Lai*** .... Date: 28th APRIL 2021 ....

Intake:	APUMF2006AI(PR)
Project Title:	**Deep Learning Models for Credit Card Fraud Detection**
Name	Lai Kai Lok
Signature	***Kyle Lai***
TP Number	TP061241
Lecturer	Prof. Dr. Mandava Rajeswari

ACKNOWLEDGEMENT

I wish to express my sincere appreciation to Professor Dr. Mandava Rajeswari for her guidance in completing this assignment. My appreciation extends to the contributors of the Python libraries and also those who shared their ideas in the Kaggle website or through publications.

ABSTRACT

Credit card fraud had caused billions of loss to both the card issuers and the merchants. Current fraud detection systems involve heavy feature engineering steps and manual inspections. The dynamic property of credit card transactions and the highly imbalanced distribution of fraud and normal samples are the two main challenges in creating an accurate fraud detection system. Deep learning is able to solve these issues without the need of heavy feature engineering. Besides, it is showing better performance compared to the standard machine learning models. Accurate fraud detection system can improve both the customers' satisfaction and companies' net income. After reviewing the related works done on the ULB credit card transaction dataset and also on other fraud dataset, the most suitable model for fraud detection is the 2 stage hybrid model of auto-encoder and classifier. Thus, this study will mainly focus on the effect of auto-encoder architecture on the model performance. The need of resampling technique, different architecture of auto-encoder, the type of input to the second stage classifier from first stage auto-encoder and the parameter tuning of auto-encoder will be investigated. As the dataset is highly imbalanced, the evaluation metric used to assess the model performance are recall, precision, F1 score and area under the precision-recall curve (PR-AUC). The results and analysis are discussed in details in Section 4 and 5.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Networks
AUROC	Area Under the Receiver Operating Characteristics
CNN	Convolutional Neural Networks
DBN	Deep Belief Networks
GBC	Gradient Boosting Classifier
GRU	Gated Recurrent Unit
HOBA	Homogeneity-Oriented Behaviour Analysis
KNN	K-nearest Neighbour
LR	Logistic Regression
LSTM	Long-Short-Term Memory
MLP	Multi Layered Perceptron
NB	Naïve Bayes
PR-AUC	Area Under the Precision-Recall Curve
RF	Random Forest
SVM	Support Vector Machine
TP	True Positives
TN	True Negatives
FP	False Positives
FN	False Negatives

SECTION 1

INTRODUCTION

The number of online transactions by using credit cards surged during the COVID-19 pandemic as the majority opt for online shopping to reduce the risk of getting infected. The amount of card fraud transactions rises alongside this. In the year 2019 alone, there are $28.65 billion losses worldwide due to payment card fraud (*Nilson Report | News and Statistics for Card and Mobile Payment Executives*, 2021). When a fraud transaction occurs, the losses are either absorbed by the card issuers or the merchants. These incurred unnecessary expenses to the company and reduced the profit significantly.

Credit card fraud is defined as the use of a credit card account for making payment or cash withdrawal without the consent of the cardholder. It is considered as a type of anomaly in the finance domain. Table 1.1 shows few types of credit card fraud (Makki *et al.*, 2019).

Table 1.1: Type of credit card fraud

Type	**Descriptions**
Stolen / offline	Fraudster use the stolen credit card for transactions. Easiest and fastest to be detected.
Application	Use fake personal information for credit card application
Bankruptcy	Insolvent individual spending without the ability to pay back
Internal	Bank employees use the customer's credit card illegally
Cardholder not present	Illegal remote transactions are made by using the details of a legitimate credit card.

As the amount of transactions occurring everyday are so huge that manual inspection is not a practical solution, an automatic fraud detection system is needed. Current credit card fraud detection systems are built by using rule-based system and tree-based machine learning classification models (Branco *et al.*, 2020). The overview of the current fraud detection system is shown in Figure 1.1. However, these methods involved heavy feature engineering which required some domain knowledge. To resolve this issue, deep learning can be deployed as feature extraction is done within the algorithm. This eliminates the needs of extensive feature engineering steps and simplifies the process of creating an accurate and robust automated fraud detection system (Yu *et al.*, 2020). Deep learning is able to perform better than standard machine learning models in credit card fraud detection (Abakarim, Lahby and Attioui, 2018). Various types of deep learning models like convolutional neural network (CNN), recurrent neural network (RNN) and auto-encoder are being used for fraud detection and show promising results. Details of the reviews are presented in Section 2 where auto-encoder is performing the best.

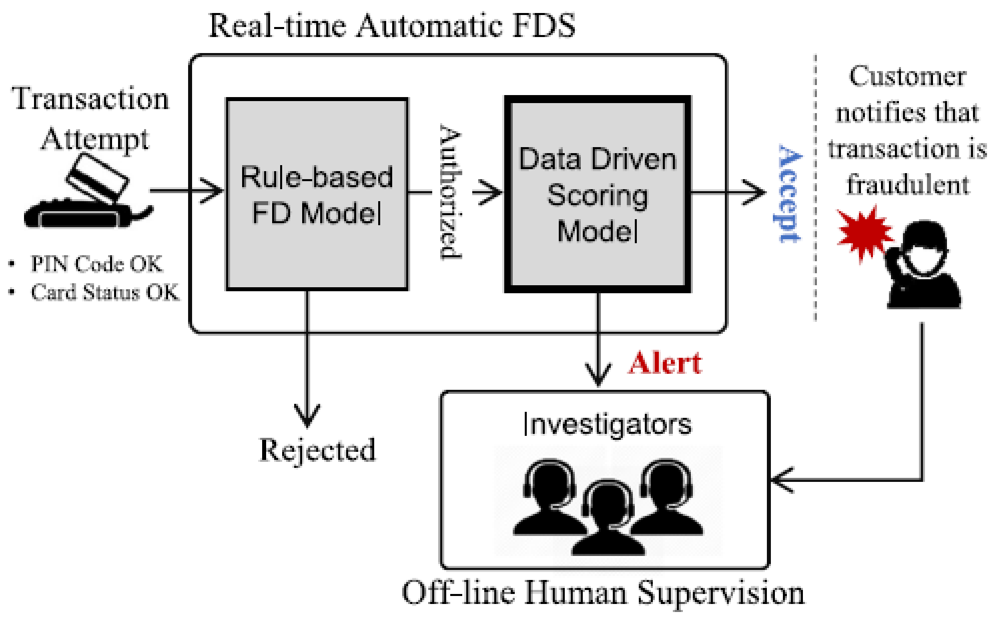


Figure 1.1: The overview of current fraud detection system (Kim *et al.*, 2019).

An accurate automated credit card fraud detection system is crucial as it not only reduces the huge amount of loss incurred to the companies, customers' satisfaction can also be improved significantly by reducing the amount of false alarm calls and avoiding wrong charges to their accounts. There are two main challenges when it comes to building an accurate fraud detection model. The first issue is the dynamic characteristic of the transactions. The strategies for committing a fraud are evolving while the normal transactions made by the cardholder are varying from time to time. The imbalance distribution of the normal and fraudulent transactions are the second issue. This makes it hard for the models to recognize the patterns of fraud (Jurgovsky *et al.*, 2018).

1.1 Problem Statement

Credit card fraud has caused huge amounts of loss to the companies and also resulted in customers' dissatisfaction. Modern fraud detection systems involved heavy feature engineering in order to improve its performance. The imbalance distribution of classes and the dynamic characteristic of the credit card transactions are the two main problems while creating an accurate fraud detection system. Deep learning models are good candidates in solving these issues.

1.2 Research Goal

The main purpose of this research is to improve the performance credit card fraud detection by using deep learning so that the loss faced by the companies can be reduced. Different deep learning models and its architecture will be reviewed and the most suitable model will be selected for building a robust fraud detection system.

1.3 Research Objectives

a) To detect fraudulent credit card transactions from a highly imbalanced dataset.
b) To review and identify suitable deep learning models for fraud detection.
c) To study and compare different types of Auto-encoder architectures.
d) To evaluate the effect of different optimizers and activation functions

1.4 Scope of Work

Auto-encoder is the main deep learning model that will be implemented for detecting fraud credit card transactions. 8 different architectures of auto-encoders will be compared and the effect of using different optimizers and activation function will be evaluated. There is only one dataset being used and discussed in this assignment, which is the ULB credit card transactions dataset.

1.5 Significance of the Research

This study will evaluate the performance of credit card fraud detection by using deep learning method particularly the auto-encoder model. How the architecture of the auto-encoder model affects the fraud detection performance will be studied and this can be a reference for future research and application.

1.6 Structure of the Report

The rest of the assignment is presented as follows. Basic concept of auto-encoder and reviews of related works done on fraud detection using deep learning are shown in Section 2. In Section 3, description of the chosen dataset and the research methodology are explained. Implementation is presented in Section 4 while Section 5 showed the results and discussion. Conclusion is discussed in the final section.

SECTION 2

LITERATURE REVIEW

In this section, the basic concepts of auto-encoder are presented followed by the reviews of related works done on fraud detection.

2.1 Auto-encoder

This type of deep learning model reconstructed the input data samples by learning the feature representations in different dimensional space. The weight in the hidden layer nodes are adjusted to minimize the reconstruction errors for normal instances. Those instances that have large reconstruction error are considered as fraud or anomaly as it is hard for the model to reconstruct an anomaly from the learned normal feature representations (Pang *et al.*, 2021). In auto-encoder networks, the input layer and the output layer have the same number of nodes. The hidden layers are made up from two main components, which is the encoder and decoder as shown in Figure 2.1.

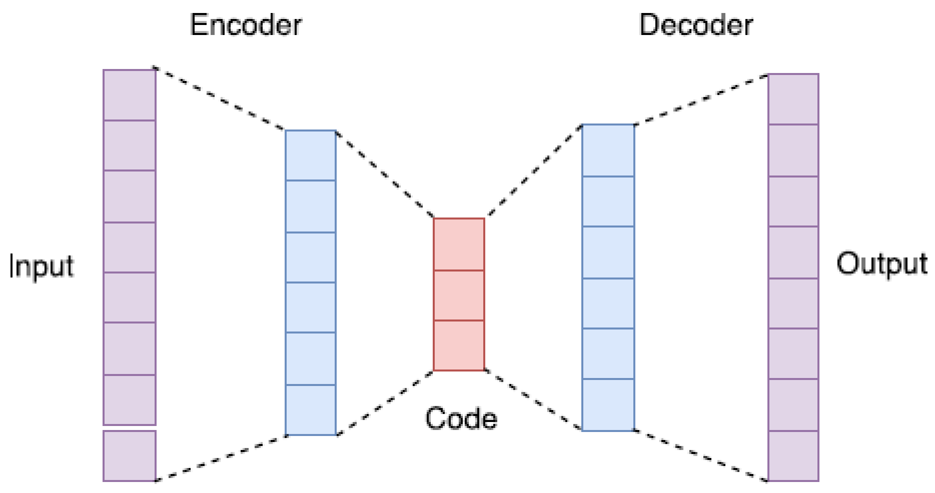


Figure 2.1: The structure of a basic auto-encoder (Ozbayoglu, Gudelek and Sezer, 2020).

The encoder hidden layers convert the input data into the code by using a set of weights and biases while the decoder hidden layers convert the code into the output by using different sets of weights and biases (Misra *et al.*, 2020). The reconstruction error or the differences between the output and input is then used as a guide for deciding whether the instance is a fraud or not.

The advantages of auto-encoder is that it can be applied to different types of data for feature learning and dimensionality reduction. Besides, only the normal instances are needed to train the model for creating satisfied results. On the other hand, the disadvantages of the auto-encoder is its performance can easily be affected by noise, rare normal instances or the presence of fraud in the training process. Moreover, as an auto-encoder is built to extract the salient features of normal instances, it is not optimized to detect irregularities (Pang *et al.*, 2021). As auto-encoder only needs to learn the features of normal instances, it can reduce the negative impact of highly imbalanced fraud detection dataset.

2.2 Related Works

In this section, the reviews on related works done on the same ULB dataset and on fraud detection are presented followed by the summary and gap analysis.

2.2.1 Related Works Done on the Same Dataset

Several works were done on the same ULB credit card transactions dataset which can be obtained from Kaggle website (*Credit Card Fraud Detection | Kaggle*, 2018) by using different deep learning models. A 5 layers simple CNN model was built to detect fraudulent transactions (Krutarth Darji, 2020). In this work, the hidden layers consist of 2 Conv1D layers and 1 flatten layer. Only accuracy is being used as the evaluation metric. However, as this is a highly imbalanced dataset, accuracy metric cannot represent the whole situation. This is because by predicting all the instances as normal could have yielded more than 95% accuracy. Thus, recall, precision, F1 score and area under the precision-recall curve (PR-AUC) are more suitable evaluation metrics.

An 8 layers auto-encoder is shown to perform better than logistic regression (LR), support vector machine (SVM) and a basic artificial neural network (ANN) in term of F1 score and precision (Abakarim, Lahby and Attioui, 2018). The threshold or the way this work used to decide whether an instance is fraud or not is not discussed in the paper. Furthermore, Random Forest (RF) should be used in comparison as it can perform well in classification problems (Nadim *et al.*, 2019). In another work which is also using 8 layers auto-encoder (Robin

Teuwens, 2021), the threshold for fraud classification is set by using modified version of z-score, which is derived from median absolute deviation. With the new version of threshold, it is able to achieve better recall and precision compared to the previous work done by *Abakarim et.al*. A 6 layers auto-encoder with the threshold of fraud classification set by using the 99th percentile (Chang, 2020), the model is able to achieve a false positive rate of 1 % and true positive rate of 83%. However, the evaluation metrics are not comprehensive enough to assess the model performance.

A 2 stage hybrid model consisting of a 6 layers auto-encoder followed by a classifier model was developed and showed significant improvement for fraud detection (Misra *et al.*, 2020). Multi Layered Perceptron (MLP) yielded the overall best result as the second stage classifier compared to K-nearest Neighbour (KNN) and LR. As the first stage auto-encoder only learned to reconstruct the normal instances, the latent representation of the normal and fraud instances after passing through the encoder layer can easily be distinguished. The classifier model can easily be trained for detecting the fraud. A similar 2 stage hybrid model is developed but using different auto-encoder architecture (Shivam Bansal, 2019). This model is able to produce a better result than the hybrid model created by *Misra et al*. This showed that the architecture of the auto-encoder plays a crucial role in determining the performance. The summary of the related works done on the same dataset is presented in Table 2.1.

2.2.2 Related Works Done on Fraud Detection

A 5 layers DNN was developed for credit card fraud detection and it performed the best compared to the Naïve Bayes (NB), SVM and LR in terms of accuracy and area under the receiver operating characteristics (AUROC) (Yu *et al.*, 2020). As the dataset is imbalanced, precision, recall, F1 score and PR-AUC should be used instead. Besides, RF should be included for comparison. The customer details record mobile communication dataset are converted into image form so that it can be used for training a CNN model (Chouiekh and El Haj, 2018). This CNN model performed better than SVM, RF and gradient boosting classifier (GBC) in terms of accuracy. More evaluation metrics like the F1 score and PR-AUC are needed to check for the model robustness. A homogeneity-oriented behaviour analysis (HOBA) based feature engineering framework is proposed to be included alongside with deep learning models (Zhang *et al.*, 2019). With the HOBA features, the deep belief network (DBN) performed the best compared to CNN and RNN models. It shows that the domain related feature engineering is able to improve the performance of deep learning models.

A Long-Short-Term Memory (LSTM) type of RNN is developed to detect both online and offline credit card fraud (Jurgovsky *et al.*, 2018). This LSTM was then compared with RF and only showed better performance in detecting offline fraud. This indicated that the characteristics of the data can impact the performance of the models. A Gated Recurrent Unit (GRU) type of RNN with stacked architecture was developed and compared with the existing model used by financial institutions for fraud detection (Branco *et al.*, 2020). It is shown that the proposed model is able to outperform the existing tree based model and save millions of Euro with only mini-seconds delayed. Thus, it is viable to deploy a deep learning model for practical usage in the financial institution. By using Focalloss rather than CrossEntropy as the loss function during model training, both the DNN and LSTM models showed improvement in detecting bank fraud transactions (Zhan, 2020). This showed that there is a need to adjust the parameter accordingly based on the type of dataset as it can impact the result.

Ensemble of DNN is proposed to be used for credit card fraud detection and it displayed better results compared to the standard ensemble models (Kim *et al.*, 2019). In another work, ensemble of RNN was proposed (Forough and Momtazi, 2021). Even though the LSTM-based ensemble model performed better, the GRU-based model is more efficient. Ensemble of deep learning models show promising results and have great potential for fraud detection. A 2 phase hybrid model consisting of auto-encoder and K-means unsupervised model is proposed for detecting fraud in network traffic dataset and displayed close to perfect results (Dawoud, Shahristani and Raun, 2019). Instead of using the latent representation right after the encoder layers for the second stage classifying process, as done by *Misra et al* and *Shivam Bansal,* this work is using the reconstruction error as the input for the second stage model. Both types of models are showing promising results. The summary of the related works done on the fraud detection is presented in Table 2.2.

2.2.3 Summary and Gap Analysis

Throughout the literature review on related works, the 2 stage hybrid model which consists of the auto-encoder followed by a classifier model is performing well in fraud detection consistently. Thus, the two stage hybrid model will be the main focus in this assignment. The

2 stage hybrid models are developed without involving any resembling technique. Therefore, a study to check for the need of resembling technique will be conducted. Besides, the work done on the same dataset is only focused on one architecture of auto-encoder. Thus, different architectures of auto-encoders will be built to check on how the architectures affect the performance. Encoded latent representation or the reconstruction errors from the auto-encoder are used in different studies for second stage classifier training. Hence, the more suitable type of input for the second stage model training will be identified in this study. Lastly, as the related works are using different optimizer and activation function, these two parameters will be tuned to identify the best candidate.

Table 2.1: Summary of related works done on the ULB credit card transactions dataset

Citation	**Dataset & Problem Studied**	**Model**	**Optimizer & Regularization**	**Activation**	**Number of Layers**	**Results**	**Remarks**
(Abakarim, Lahby and Attioui, 2018)	ULB Credit Card Transactions Dataset (Fraud Detection)	• Auto-encoder	NA	tanh	8 (3 encoders 3 decoders)	Accuracy = 0.9855 Recall = 0.582 Precision = 0.197 F1 Score = 0.294	• Auto-encoder showed best F1 Score and Precision compared to LR, SVM and ANN
(Misra *et al.*, 2020)	ULB Credit Card Transactions Dataset (Fraud Detection)	• 2 stage model (Auto-encoder + MLP)	• ADAM	NA	6 (2 encoders 2 decoders)	Accuracy = 0.9994 Recall = 0.8015 Precision = 0.8534 F1 Score = 0.8265	• The proposed two stage model showed significant improvement in F1 score compared to others' work
(Chang, 2020)	ULB Credit Card Transactions Dataset (Fraud Detection)	• Auto-encoder	NA	NA	6 (2 encoders 2 decoders)	false positive rate = 1% true positive rate = 83%	• The model is using 99th percentile threshold
(Krutarth Darji, 2020)	ULB Credit Card Transactions Dataset (Fraud Detection)	• CNN	• ADAM • Dropout • Batch normalization	RELU	5 (2 Conv1D 1 Flatten)	accuracy = 0.9512	• A simple CNN model focus only on accuracy

Table 2.1: Summary of related works done on the ULB credit card transactions dataset (cont'd)

Citation	Dataset & Problem Studied	Model	Optimizer & Regularization	Activation	Number of Layers	Results	Remarks
(Shivam Bansal, 2019)	ULB Credit Card Transactions Dataset (Fraud Detection)	• 2 stage model (Auto-encoder + LR)	• adadelta	RELU tanh	6 (2 encoders 2 decoders)	Accuracy = 0.9828 Recall = 0.87 Precision = 1.0 F1 Score = 0.93	• LR used the latent representation of AE for fraud classification
(Robin Teuwens, 2021)	ULB Credit Card Transactions Dataset (Fraud Detection)	• Auto-encoder	• ADAM	ELU	8 (3 encoders 3 decoders)	Accuracy = 0.9832 Recall = 0.7337 Precision = 0.2180	• Threshold is set by using modified z-score using median absolute deviation

Table 2.2: Summary of related works done on fraud detection

Citation	Dataset & Problem Studied	Model	Optimizer & Regularization	Activation	Number of Layers	Results	Remarks
(Yu *et al.*, 2020)	IEEE CIS Transactions Dataset	• DNN	• NADAM • Dropout • Batch normalization	GELU	5	Accuracy = 0.957 AUROC = 0.901	• DNN performed better than NB, SVM and LR
(Jurgovsky *et al.*, 2018)	e-commerce and face-to-face credit card transaction datasets	• LSTM	• ADAM • Dropout	tanh	4 (2 recurrent 1 hidden 1 LR classifier)	PR-AUC = 0.178 ~ 0.402	• LSTM showed better performance in offline transactions

Table 2.2: Summary of related works done on fraud detection (cont'd)

Citation	**Dataset & Problem Studied**	**Model**	**Optimizer & Regularization**	**Activation**	**Number of Layers**	**Results**	**Remarks**
(Zhang *et al.*, 2019)	real-life dataset from a commercial bank in China	DBN–HOBA features	NA	NA	NA	F1 Score = 0.568	• HOBA-feature engineering is able to improve the performance of model with DBN showing the best result
		CNN–HOBA features				F1 Score = 0.518	
		RNN–HOBA features				F1 Score = 0.540	
(Chouiekh and El Haj, 2018)	customer details records from mobile communication telecom company (Fraud Detection)	• CNN	• SGD • No regularization	RELU	7	Accuracy = 0.82	• CNN showed better performance compared to SVM, RF and GBC
(Branco *et al.*, 2020)	two different datasets from European financial institutions	• RNN-GRU	NA	tanh	varying number of stacked GRUs	save 1 M Euro (data A) and 8 M Euro (data B) compared to existing deployed model	• The proposed model is able to outperformed tree-based model with only millisecond-level latencies
(Kim *et al.*, 2019)	credit card transaction datasets from April 2015 to May 2016 in South Korea	• Ensemble of DNN	• ADAM • Dropout	Sigmoid	3~6	integrated circuit K-S stat = 0.704 AUROC = 0.907	• The proposed model is better than the standard ensemble model
						magnetic swipe K-S stat = 0.804 AUROC = 0.948	

Table 2.2: Summary of related works done on fraud detection (cont'd)

Citation	**Dataset & Problem Studied**	**Model**	**Optimizer & Regularization**	**Activatio n**	**Number of Layers**	**Results**	**Remarks**
(Dawoud, Shahristani and Raun, 2019)	KDD99 network traffic dataset	• 2 phase model (Auto-encoder + K-means)	• ADAM	NA	NA	Accuracy = 0.9988 Precision = 1 F1 Score = 0.9987	• AE with k-means showed better result than AE with means-shift
(Zhan, 2020)	TESTIMON transaction dataset	• LSTM • DNN	• ADAM • Dropout	RELU	4	Recall = 0.9926 Accuracy = 0.9864 FNR = 0.0074 FPR = 0.0136	• LSTM showed overall best performance using focalloss loss function with alpha = 0.999
(Forough and Momtazi, 2021)	one dataset from European card holders and one dataset from Brazilian bank	• Ensemble of RNN	NA	tanh	NA	LSTM-based model outperform GRU-based and standard ensemble model	• GRU-based model is more efficient

SECTION 3

RESEARCH METHODOLOGY

The dataset description and the Python libraries used for model implementation are presented in the Section 3.1 and 3.2 respectively. Section 3.3 explains the chosen model performance evaluation metrics. The main flow of the research methodology is shown in Figure 3.1.

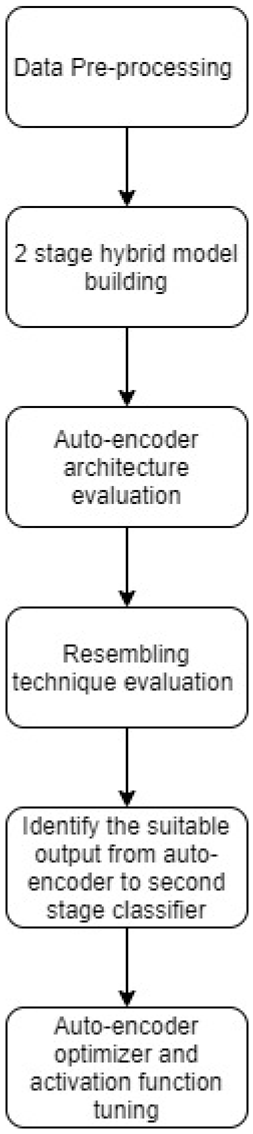

Data Pre-processing
2 stage hybrid model building
Auto-encoder architecture evaluation
Resembling technique evaluation
Identify the suitable output from auto-encoder to second stage classifier
Auto-encoder optimizer and activation function tuning

Figure 3.1: The main flow of the research methodology

3.1 Dataset Description

The ULB credit card transactions dataset was obtained from the Kaggle website (*Credit Card Fraud Detection | Kaggle*, 2018). This dataset contains the credit card transactions made by European cardholders in September 2013. The targeted outcome of the model is to identify the fraudulent transactions. This dataset consists of 284807 instances and 31 features. Only 492 or 0.172% of the instances are fraud, a highly imbalanced dataset. The summary of the features in the dataset is shown in Table 3.1 below.

Table 3.1: Summary of the features in the dataset

Feature	**Definition**	**Type**
Time	The time period between the current transaction and the first transaction in seconds	Numeric
V1 – V28	28 columns of PCA transformation data, sue to confidentiality issue	Numeric
Amount	The amount of money involved in the particular transaction	Numeric
Class	The type of the transaction, normal (0) or fraud (1)	Categorical

3.2 Python Libraries

This 2 stage hybrid model will be developed by using Colaboratory or Colab in the browser. Python will be the main programming language in this assignment. The Python libraries that are going to be used are listed in Table 3.2.

Table 3.2: List of packages used in this assignment and its function

Packages	**Main Usage in this Study**
tensorflow & keras	For developing deep learning model, mainlt the auto-encoder
sklearn	For data pre-processing and developing the second stage classifier
pandas	For data manipulation
numpy	For applying mathematical functions
matplotlib	For plotting graph
seaborn	For data visualization

3.3 Model Performance Evaluation Metrics

As the class of the dataset is highly imbalanced, accuracy alone is not enough to assess the performance of the model. Thus, recall, precision, F1 score and PR-AUC are applied to evaluate the performance of the model. All of these metrics are derived from true positive (TP), true negative (TN), false positive (FP) and false negative (FN). The formula to calculate the accuracy, precision, recall and f1 core is shown in equation 1-4 below (Zhang *et al.*, 2019).

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FN} \tag{2}$$

$$Recall = \frac{TP}{TP + FP} \tag{3}$$

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FN + FP} \tag{4}$$

The AUROC is not used in this study as it is not sensitive to the change of false positive due to the highly imbalance distribution of fraud and normal classes. This issue can be solved by using the PR-AUC which can show a complete picture of the model's performance (Jurgovsky *et al.*, 2018).

SECTION 4

IMPLEMENTATION

The techniques used for data visualization and pre-processing, various auto-encoder architectures, the use of resampling technique, feeding the final reconstruction representation for the second stage classifier. The tuning of optimizer and activation function are presented in this section.

4.1 Data Pre-Processing and Visualization

The Credit Card Fraud dataset is highly imbalanced as only 492 or 0.172% of the instances belong to the fraud class. The distribution of each classes is shown in Figure 4.1.

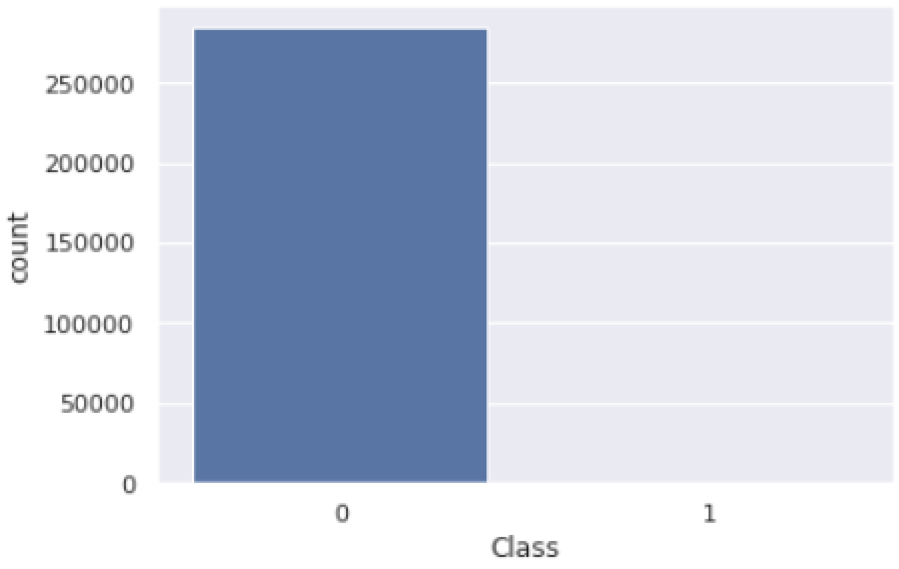
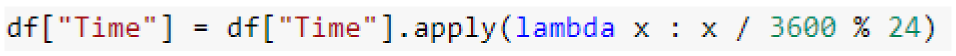


Figure 4.1: The distribution of normal (0) and fruad (1)

4.1.1 Data Pre-Processing

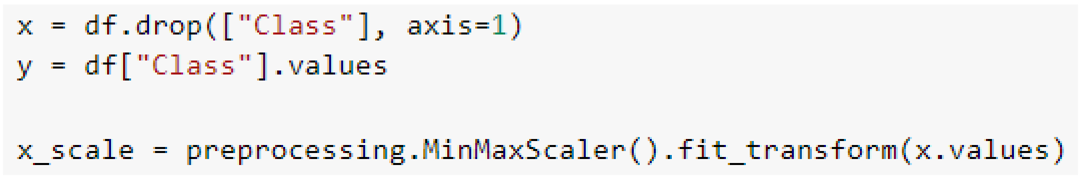
As the feature Time in the dataset is showing the period between the first transaction and the current transaction in seconds, it is converted to different time in a day to have a better meaning.



```
df["Time"] = df["Time"].apply(lambda x : x / 3600 % 24)
```

Figure 4.2: Code for converting the Time feature into different period in a day

Data scaling pre-processing is done on the independent variables to reduce the time of converging during model training.

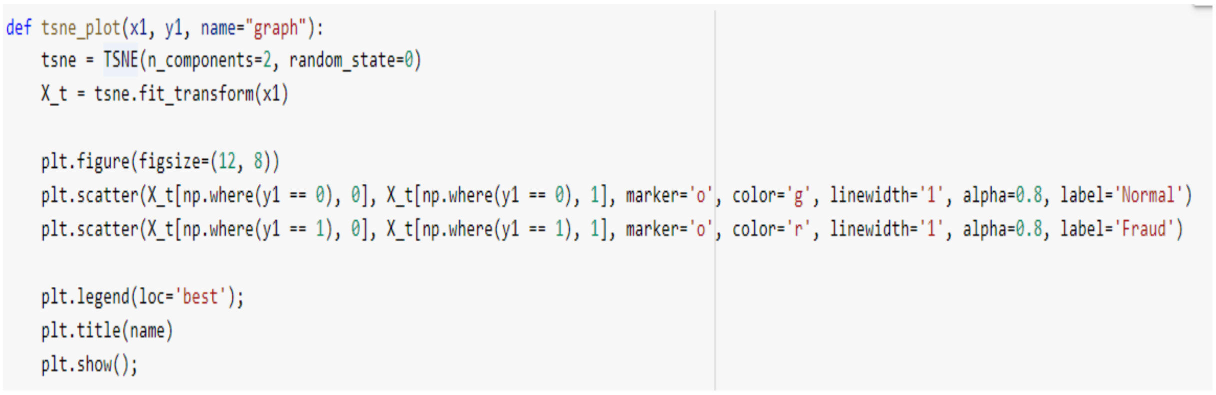


```
x = df.drop(["Class"], axis=1)
y = df["Class"].values

x_scale = preprocessing.MinMaxScaler().fit_transform(x.values)
```

Figure 4.3: Code for data scaling for the independent variables

4.1.2 Data Visualization

The distribution of the normal and fraud class based on the 30 features is reduced into a two dimensional space and visualized by using the TSNE function. Both the original data and those encoded data from the auto-encoder are visualized by using a defined function tsne_plot. This visualization indicates how well the normal instances separated from the fraud instances. One example of the plot is shown in Figure 4.5.



```
def tsne_plot(x1, y1, name="graph"):
    tsne = TSNE(n_components=2, random_state=0)
    X_t = tsne.fit_transform(x1)

    plt.figure(figsize=(12, 8))
    plt.scatter(X_t[np.where(y1 == 0), 0], X_t[np.where(y1 == 0), 1], marker='o', color='g', linewidth='1', alpha=0.8, label='Normal')
    plt.scatter(X_t[np.where(y1 == 1), 0], X_t[np.where(y1 == 1), 1], marker='o', color='r', linewidth='1', alpha=0.8, label='Fraud')

    plt.legend(loc='best');
    plt.title(name)
    plt.show();
```

Figure 4.4: Define function tsne_plot for visualizing the distribution of classes

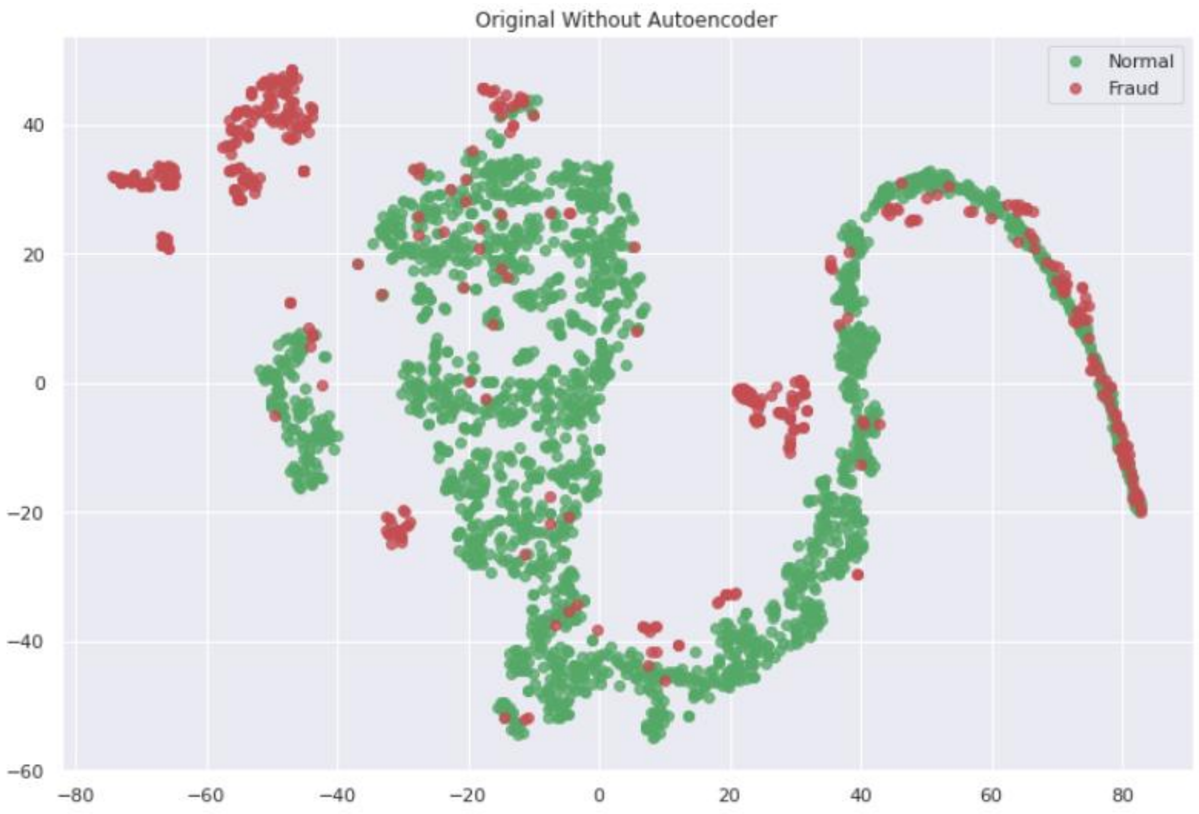
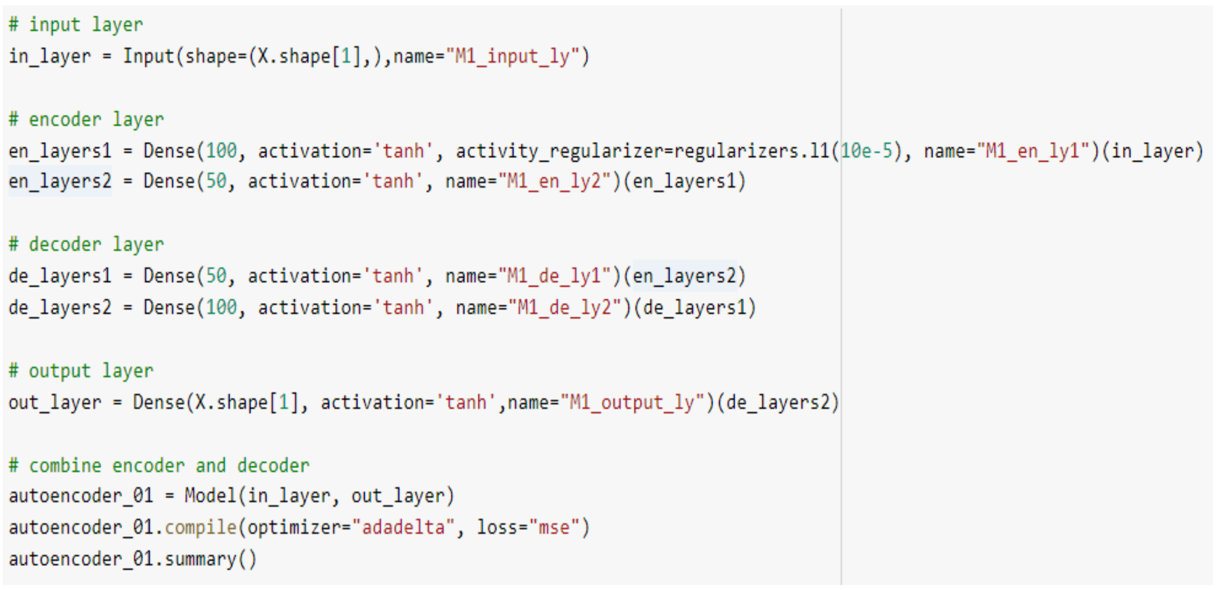


Figure 4.5: The distribution of classes in original data

4.2 Auto-encoder Architectures

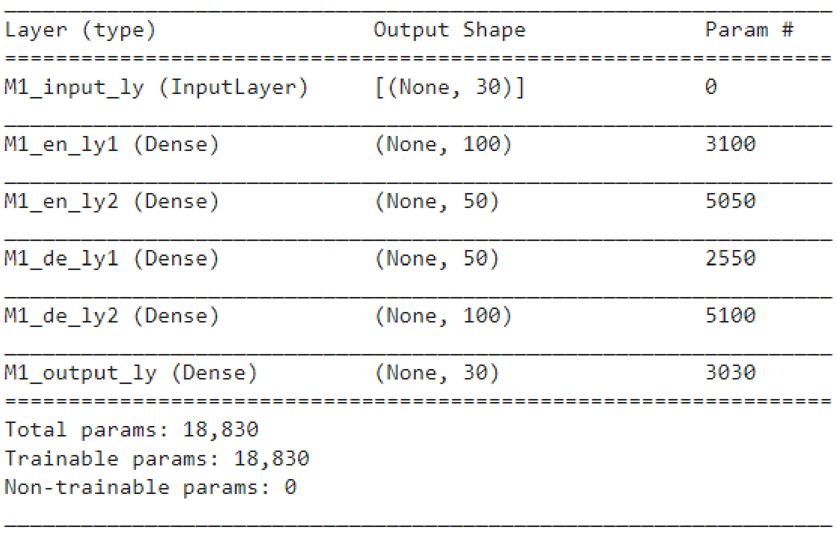
The two main architectures of auto-encoder used in this study are the over-complete and the under-complete type. In the over-complete type, the number of nodes in the hidden layers are more than the number of inputs while the under-complete type has lesser nodes in the hidden layers. 8 different models are built by using different architectures. All the models in this section are built using tanh activation function and adadelta as optimizer.

4.2.1 Model 1: Over-Complete, L1 Regularization, Without Middle Bottleneck Block



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M1_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='tanh', activity_regularizer=regularizers.l1(10e-5), name="M1_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='tanh', name="M1_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(50, activation='tanh', name="M1_de_ly1")(en_layers2)
de_layers2 = Dense(100, activation='tanh', name="M1_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M1_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_01 = Model(in_layer, out_layer)
autoencoder_01.compile(optimizer="adadelta", loss="mse")
autoencoder_01.summary()
```

Figure 4.6: The architecture building of Model 1



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
M1_input_ly (InputLayer)     [(None, 30)]              0

_________________________________________________________________
M1_en_ly1 (Dense)            (None, 100)               3100

_________________________________________________________________
M1_en_ly2 (Dense)            (None, 50)                5050

_________________________________________________________________
M1_de_ly1 (Dense)            (None, 50)                2550

_________________________________________________________________
M1_de_ly2 (Dense)            (None, 100)               5100

_________________________________________________________________
M1_output_ly (Dense)         (None, 30)                3030
=================================================================
Total params: 18,830
Trainable params: 18,830
Non-trainable params: 0

_________________________________________________________________
```

Figure 4.7: The Summary of Model 1

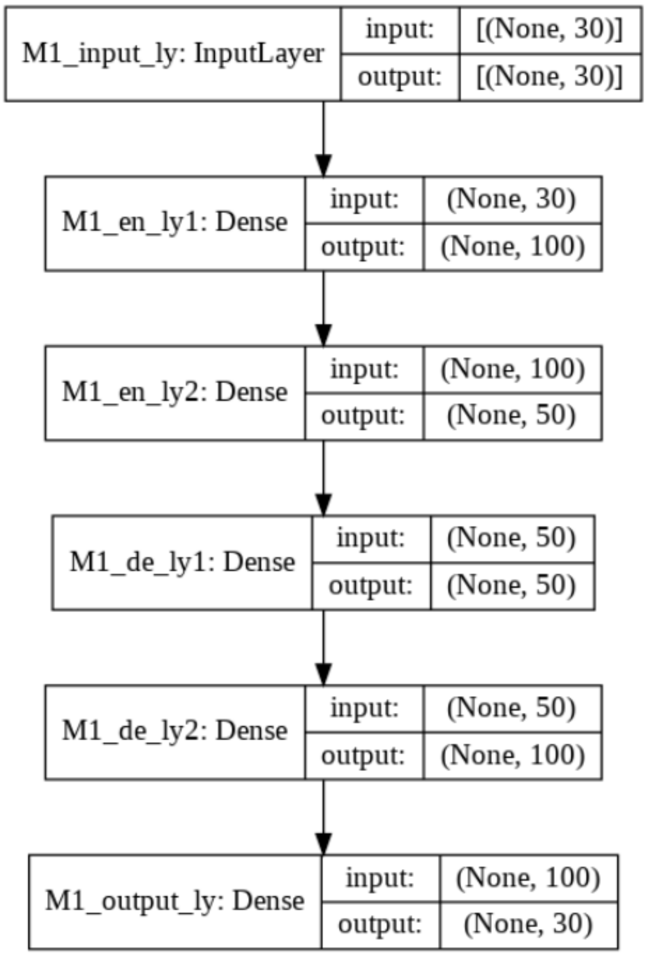


Figure 4.8: The Plot of Model 1

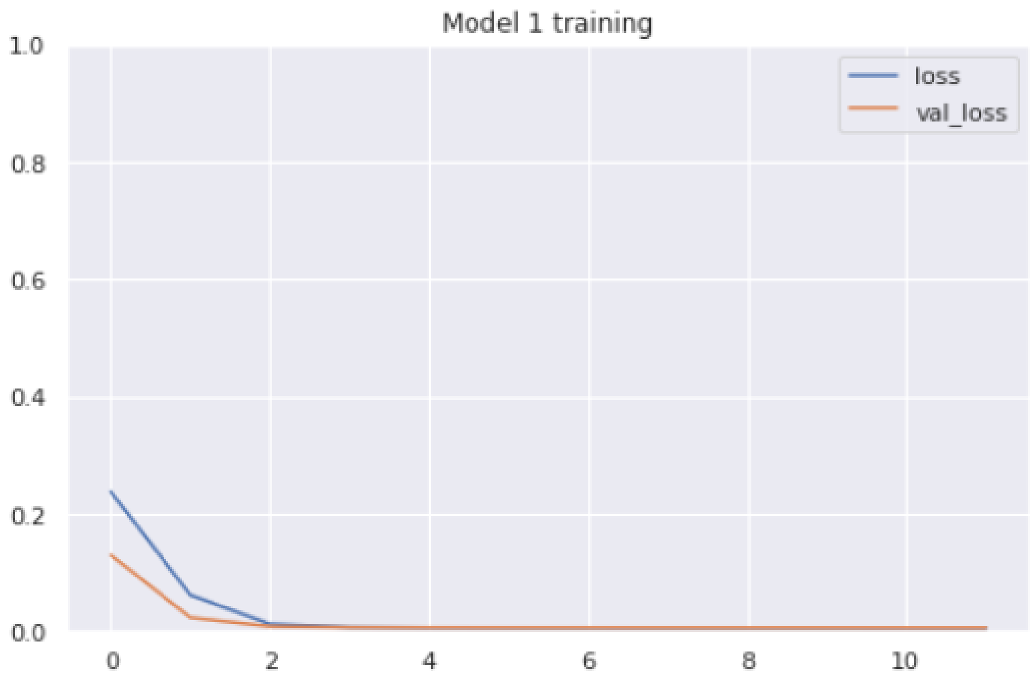
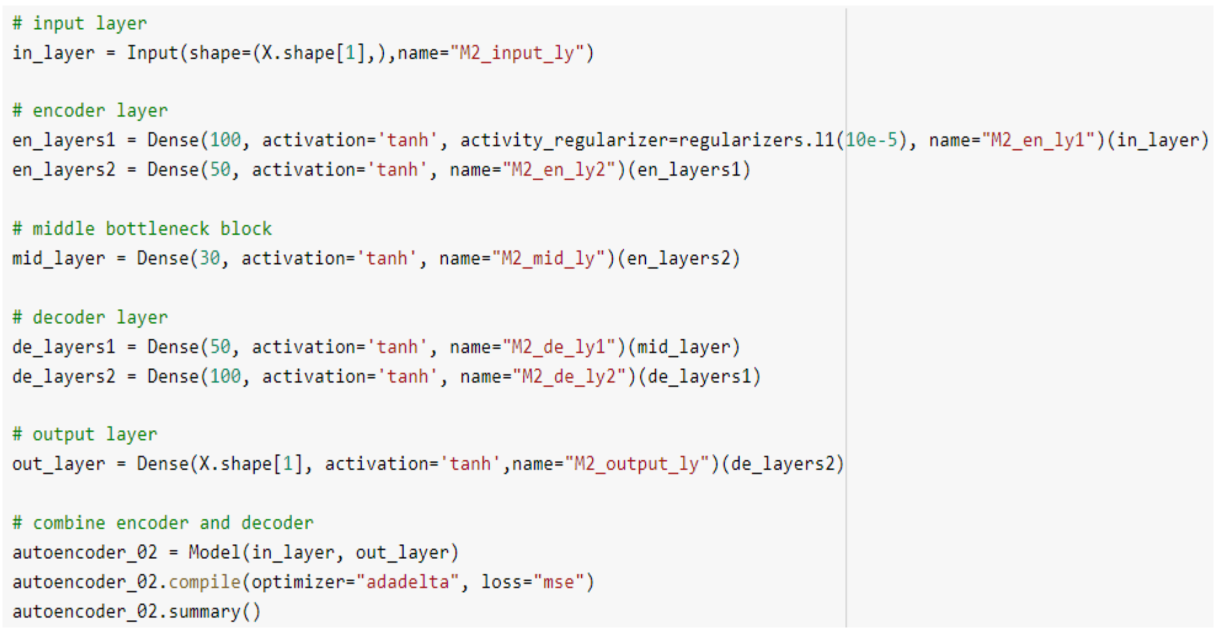


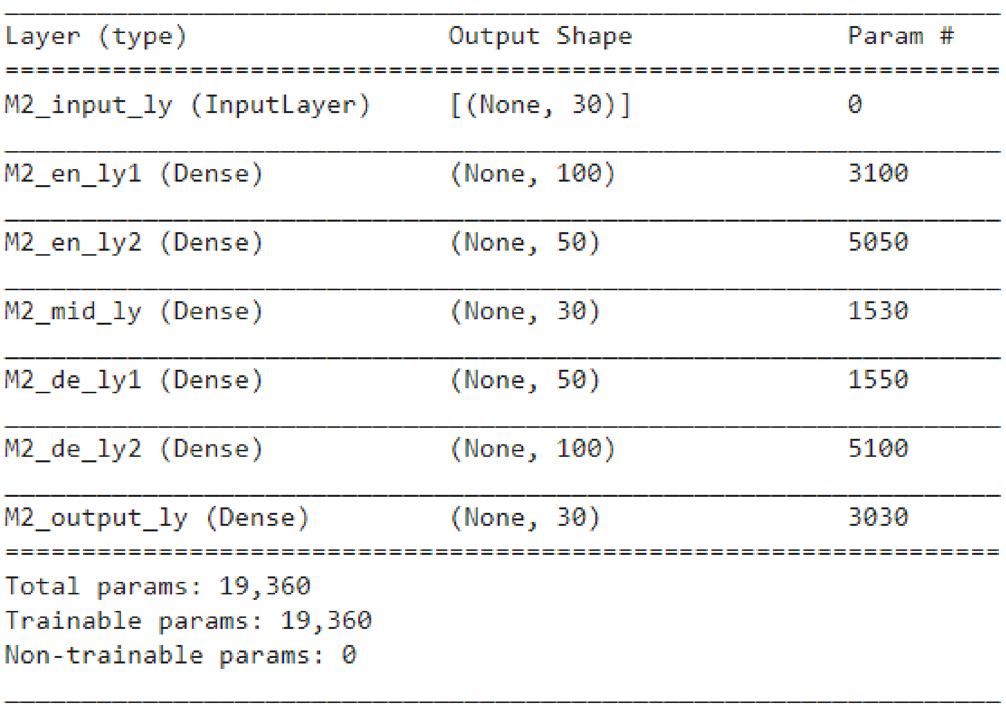
Figure 4.9: The Training progress of Model 1

4.2.2 Model 2: Over-Complete, L1 Regularization, With Middle Bottleneck Block



```python
# input layer
in_layer = Input(shape=(X.shape[1],),name="M2_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='tanh', activity_regularizer=regularizers.l1(10e-5), name="M2_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='tanh', name="M2_en_ly2")(en_layers1)

# middle bottleneck block
mid_layer = Dense(30, activation='tanh', name="M2_mid_ly")(en_layers2)

# decoder layer
de_layers1 = Dense(50, activation='tanh', name="M2_de_ly1")(mid_layer)
de_layers2 = Dense(100, activation='tanh', name="M2_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M2_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_02 = Model(in_layer, out_layer)
autoencoder_02.compile(optimizer="adadelta", loss="mse")
autoencoder_02.summary()
```

Figure 4.10: The architecture building of Model 2



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
M2_input_ly (InputLayer)     [(None, 30)]              0         
_________________________________________________________________
M2_en_ly1 (Dense)            (None, 100)               3100      
_________________________________________________________________
M2_en_ly2 (Dense)            (None, 50)                5050      
_________________________________________________________________
M2_mid_ly (Dense)            (None, 30)                1530      
_________________________________________________________________
M2_de_ly1 (Dense)            (None, 50)                1550      
_________________________________________________________________
M2_de_ly2 (Dense)            (None, 100)               5100      
_________________________________________________________________
M2_output_ly (Dense)         (None, 30)                3030      
=================================================================
Total params: 19,360
Trainable params: 19,360
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.11: The Summary of Model 2

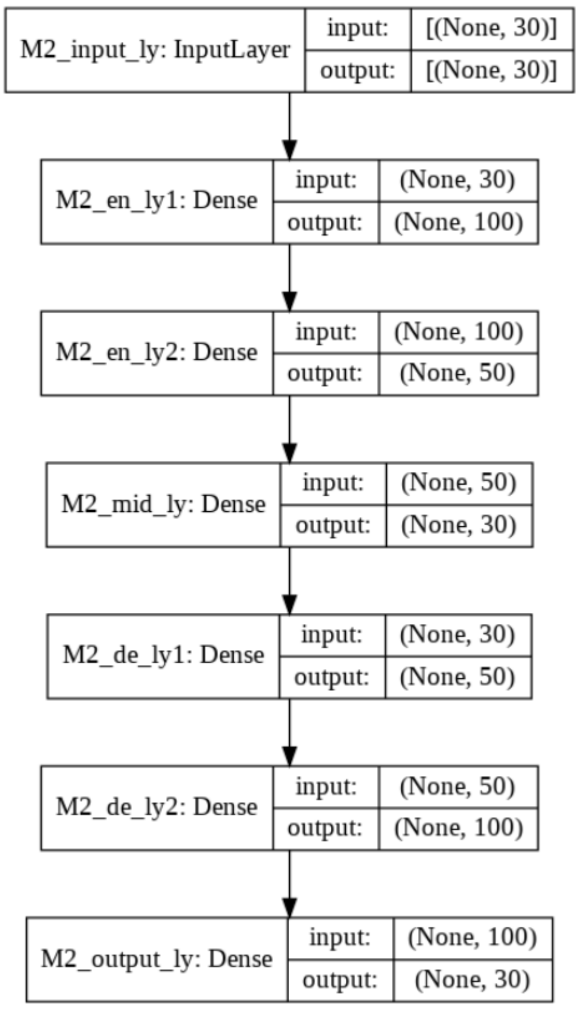


Figure 4.12: The Plot of Model 2

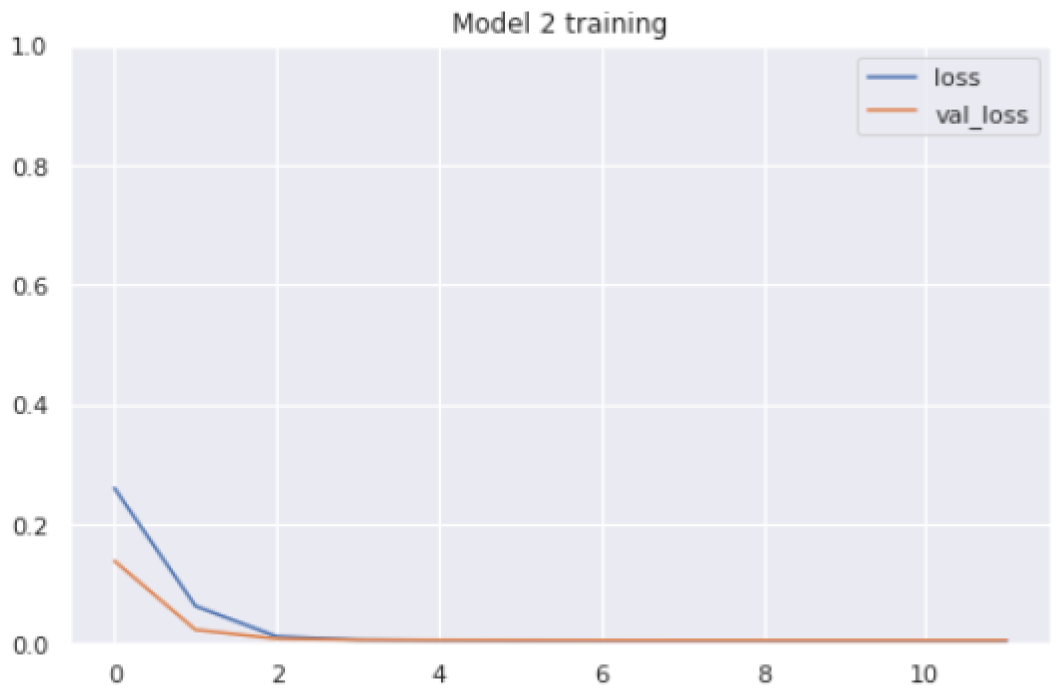
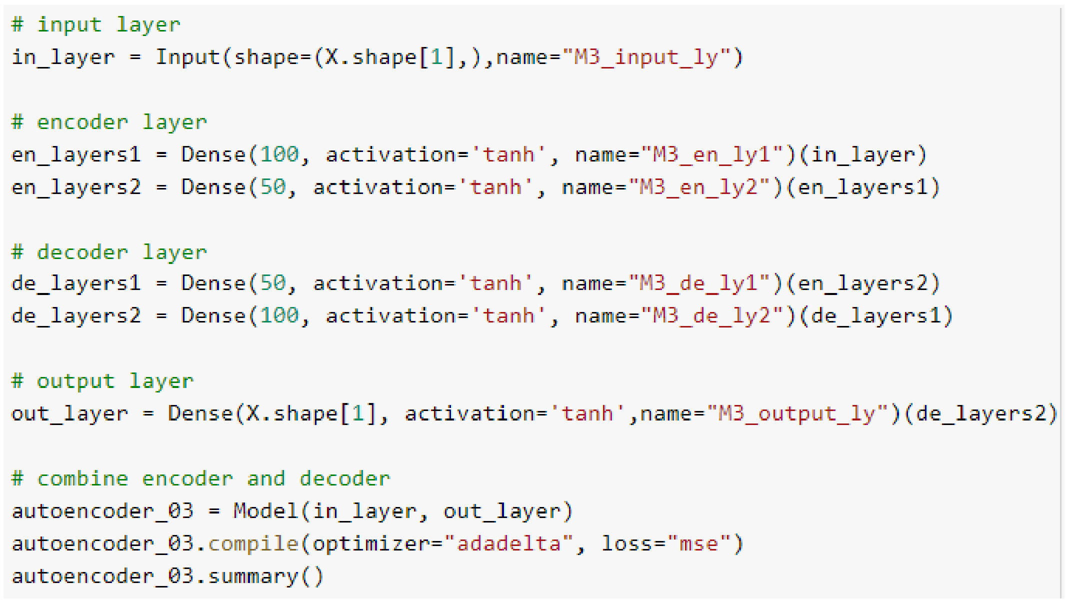


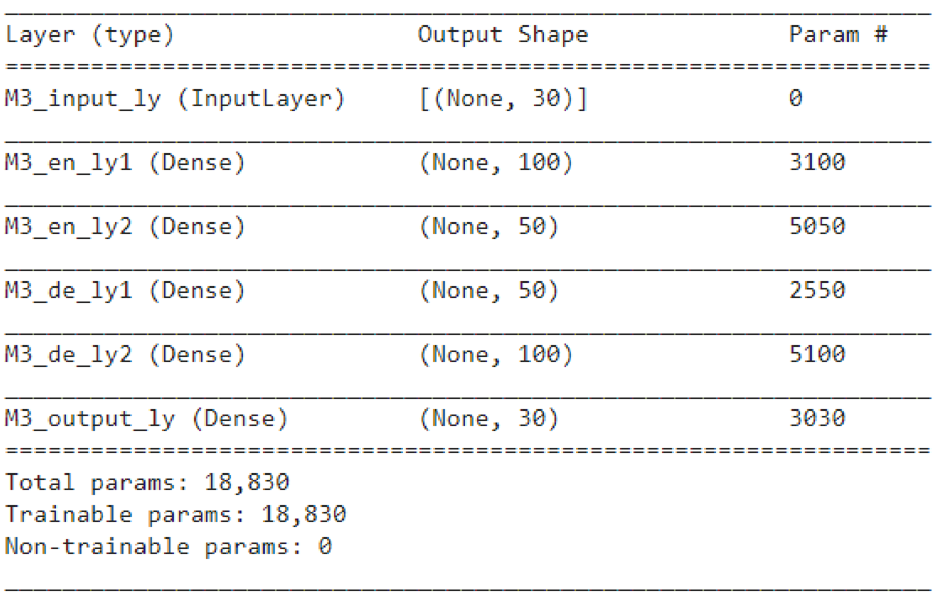
Figure 4.13: The Training progress of Model 2

4.2.3 Model 3: Over-Complete, Without L1 Regularization, Without Middle Bottleneck Block



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M3_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='tanh', name="M3_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='tanh', name="M3_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(50, activation='tanh', name="M3_de_ly1")(en_layers2)
de_layers2 = Dense(100, activation='tanh', name="M3_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M3_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_03 = Model(in_layer, out_layer)
autoencoder_03.compile(optimizer="adadelta", loss="mse")
autoencoder_03.summary()
```

Figure 4.14: The architecture building of Model 3



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
M3_input_ly (InputLayer)     [(None, 30)]              0         
                                                                 
_________________________________________________________________
M3_en_ly1 (Dense)            (None, 100)               3100      
                                                                 
_________________________________________________________________
M3_en_ly2 (Dense)            (None, 50)                5050      
                                                                 
_________________________________________________________________
M3_de_ly1 (Dense)            (None, 50)                2550      
                                                                 
_________________________________________________________________
M3_de_ly2 (Dense)            (None, 100)               5100      
                                                                 
_________________________________________________________________
M3_output_ly (Dense)         (None, 30)                3030      
=================================================================
Total params: 18,830
Trainable params: 18,830
Non-trainable params: 0

_________________________________________________________________
```

Figure 4.15: The Summary of Model 3

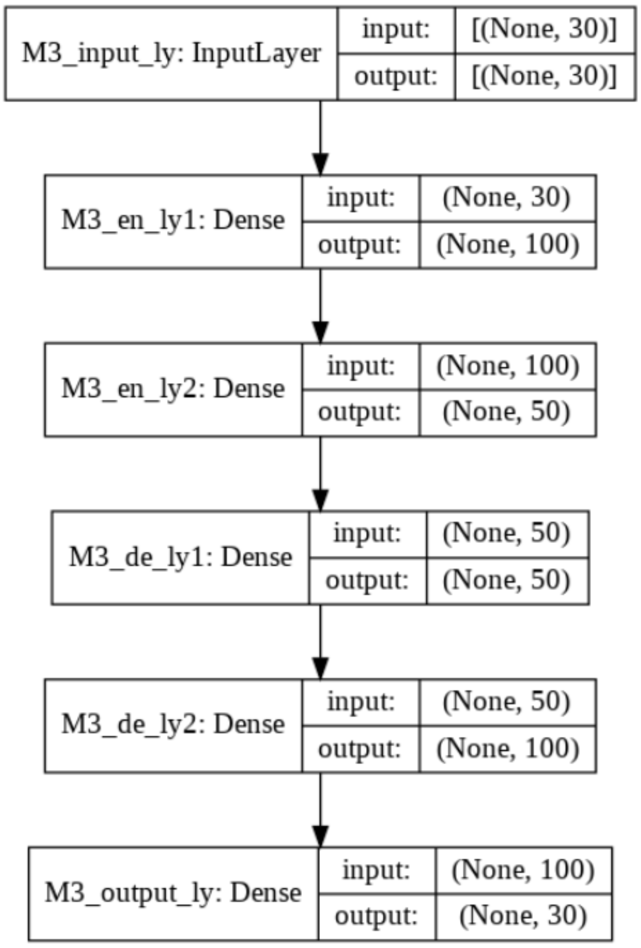


Figure 4.16: The Plot of Model 3

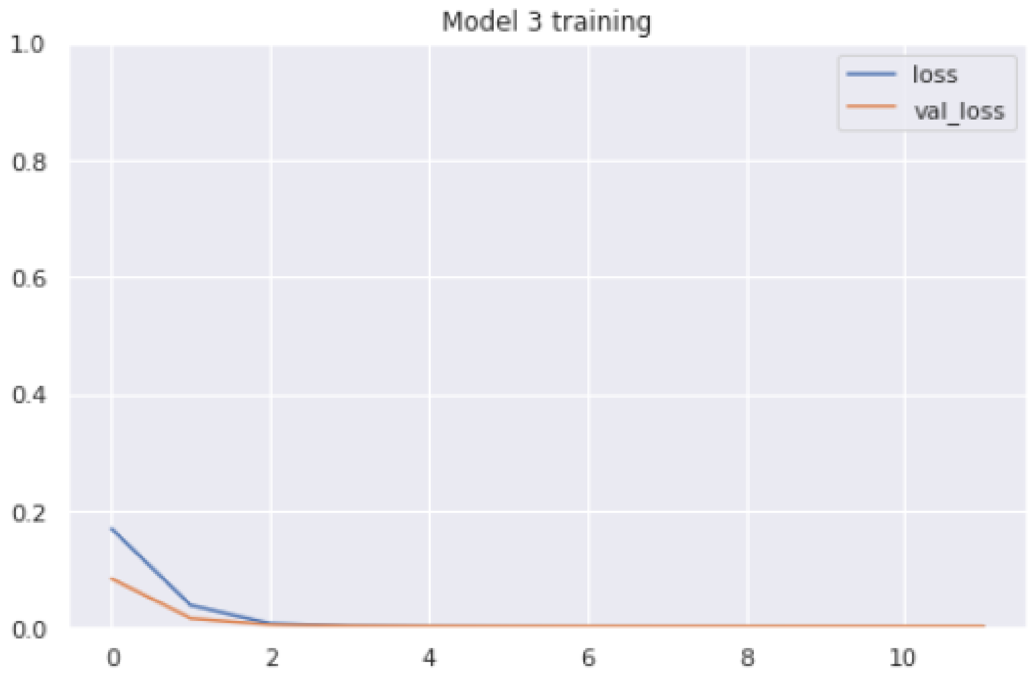
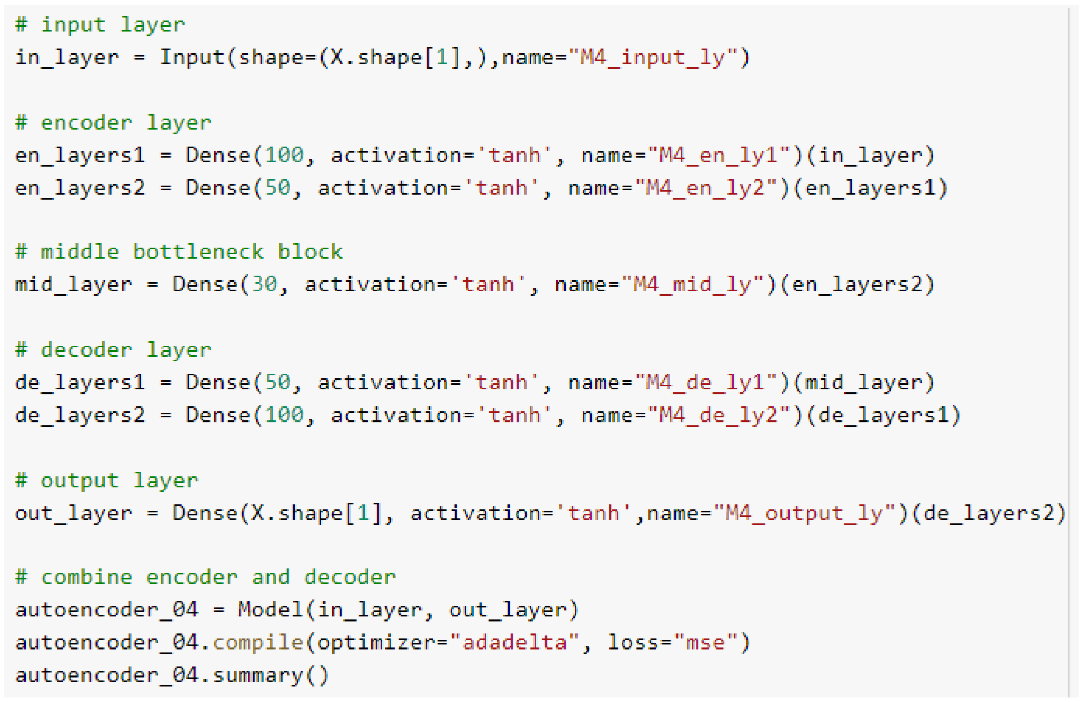


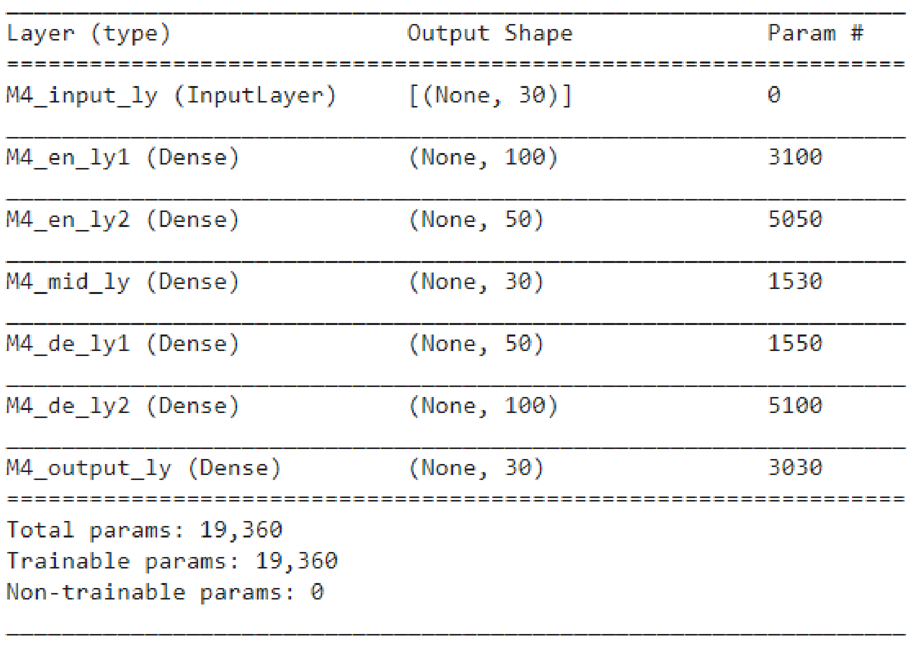
Figure 4.17: The Training progress of Model 3

4.2.4 Model 4: Over-Complete, Without L1 Regularization, With Middle Bottleneck Block



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M4_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='tanh', name="M4_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='tanh', name="M4_en_ly2")(en_layers1)

# middle bottleneck block
mid_layer = Dense(30, activation='tanh', name="M4_mid_ly")(en_layers2)

# decoder layer
de_layers1 = Dense(50, activation='tanh', name="M4_de_ly1")(mid_layer)
de_layers2 = Dense(100, activation='tanh', name="M4_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M4_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_04 = Model(in_layer, out_layer)
autoencoder_04.compile(optimizer="adadelta", loss="mse")
autoencoder_04.summary()
```

Figure 4.18: The architecture building of Model 4



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
M4_input_ly (InputLayer)     [(None, 30)]              0

_________________________________________________________________
M4_en_ly1 (Dense)            (None, 100)               3100
_________________________________________________________________
M4_en_ly2 (Dense)            (None, 50)                5050
_________________________________________________________________
M4_mid_ly (Dense)            (None, 30)                1530
_________________________________________________________________
M4_de_ly1 (Dense)            (None, 50)                1550
_________________________________________________________________
M4_de_ly2 (Dense)            (None, 100)               5100
_________________________________________________________________
M4_output_ly (Dense)         (None, 30)                3030
=================================================================
Total params: 19,360
Trainable params: 19,360
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.19: The Summary of Model 4

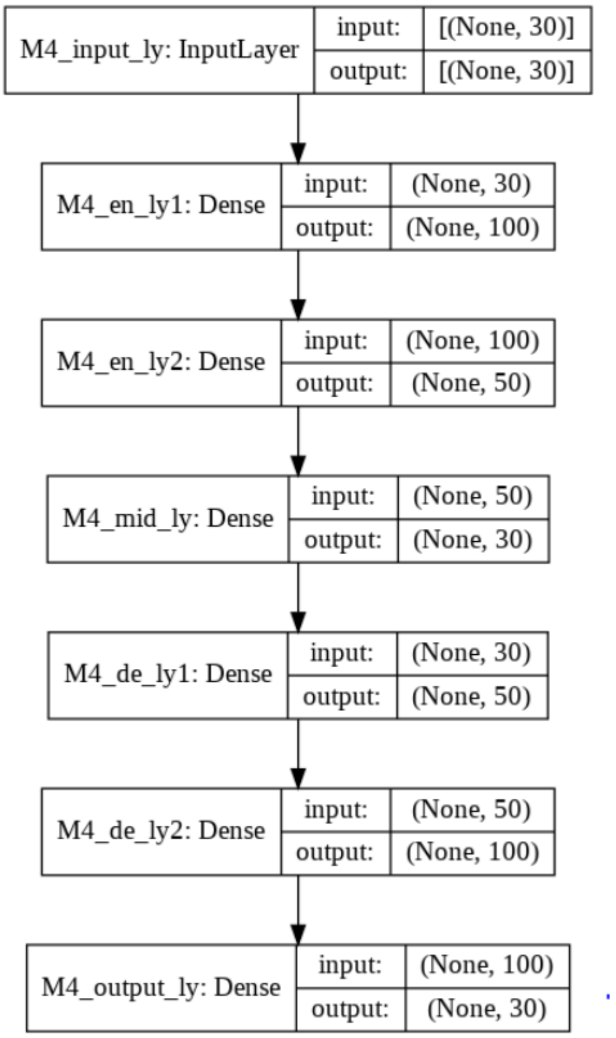


Figure 4.20: The Plot of Model 4

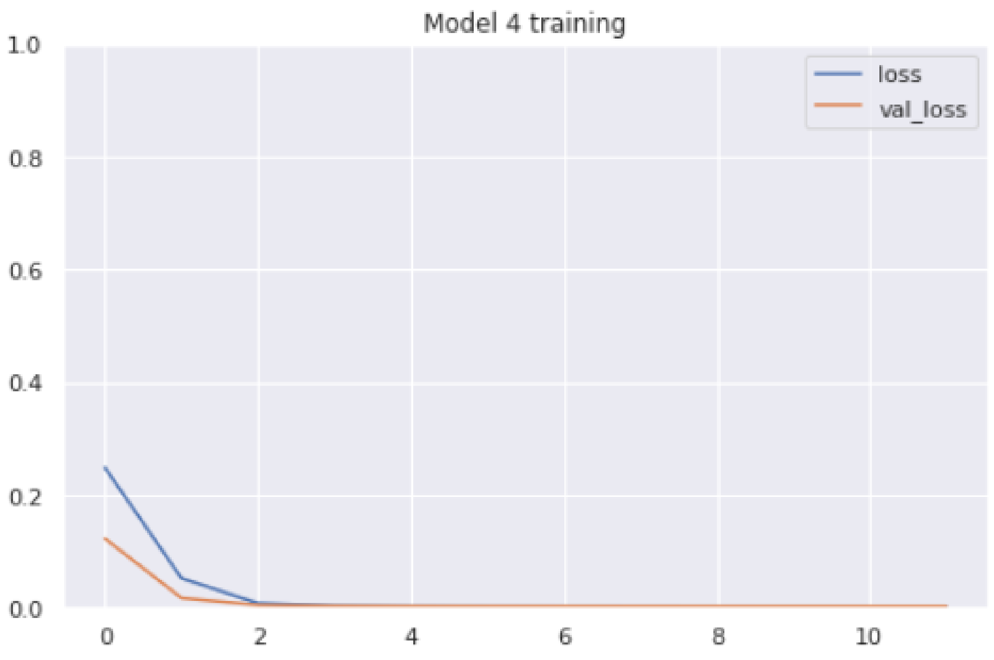
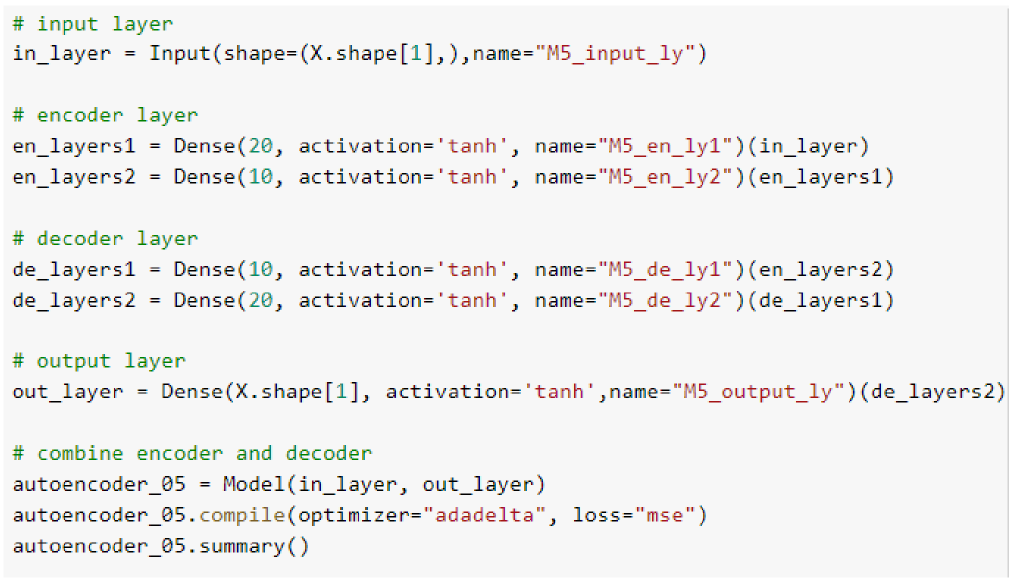


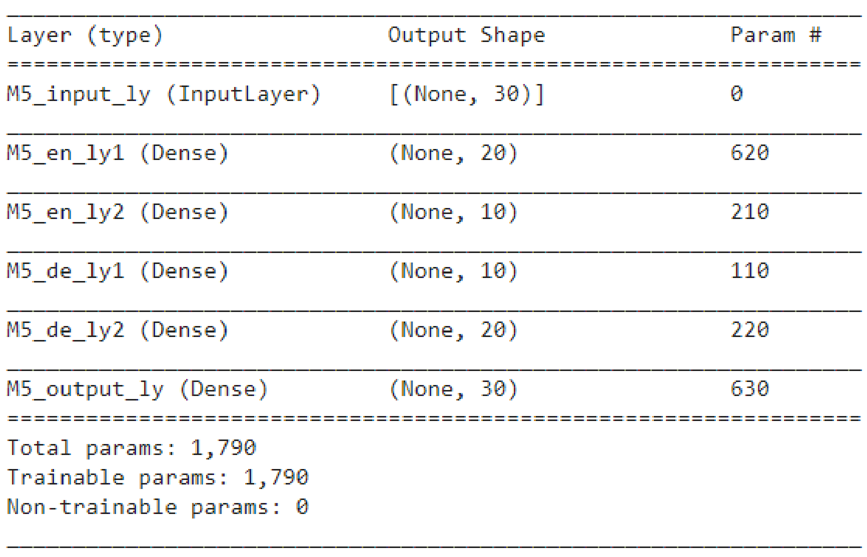
Figure 4.21: The Training progress of Model 4

4.2.5 Model 5: Under-Complete, Without Middle Bottleneck Block



```python
# input layer
in_layer = Input(shape=(X.shape[1],),name="M5_input_ly")

# encoder layer
en_layers1 = Dense(20, activation='tanh', name="M5_en_ly1")(in_layer)
en_layers2 = Dense(10, activation='tanh', name="M5_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(10, activation='tanh', name="M5_de_ly1")(en_layers2)
de_layers2 = Dense(20, activation='tanh', name="M5_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M5_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_05 = Model(in_layer, out_layer)
autoencoder_05.compile(optimizer="adadelta", loss="mse")
autoencoder_05.summary()
```

Figure 4.22: The architecture building of Model 5



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
M5_input_ly (InputLayer)     [(None, 30)]              0

_________________________________________________________________
M5_en_ly1 (Dense)            (None, 20)                620

_________________________________________________________________
M5_en_ly2 (Dense)            (None, 10)                210

_________________________________________________________________
M5_de_ly1 (Dense)            (None, 10)                110

_________________________________________________________________
M5_de_ly2 (Dense)            (None, 20)                220

_________________________________________________________________
M5_output_ly (Dense)         (None, 30)                630
=================================================================
Total params: 1,790
Trainable params: 1,790
Non-trainable params: 0

_________________________________________________________________
```

Figure 4.23: The Summary of Model 5

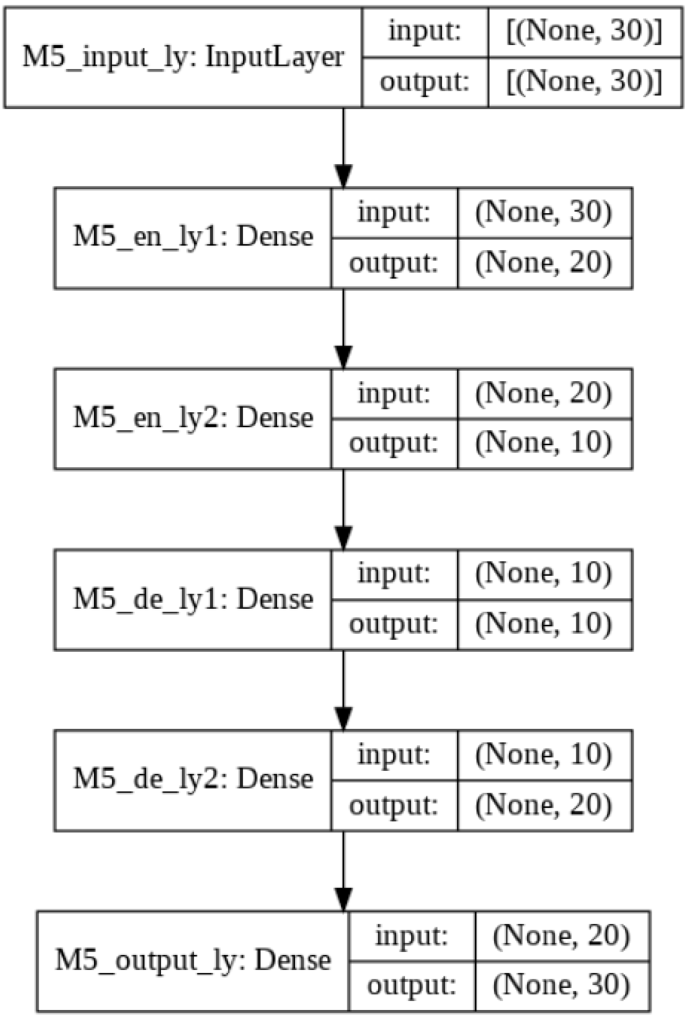


Figure 4.24: The Plot of Model 5

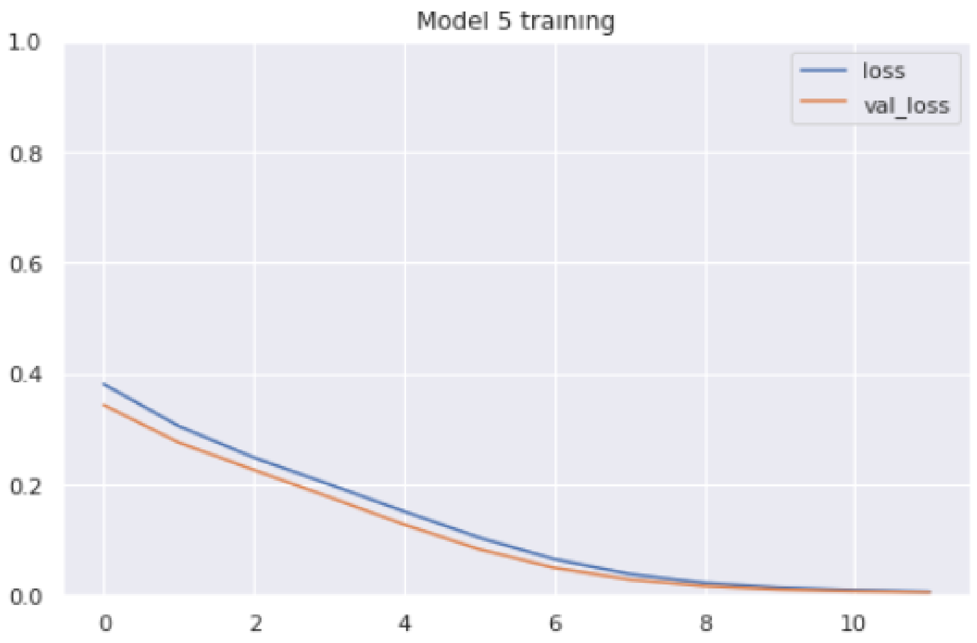
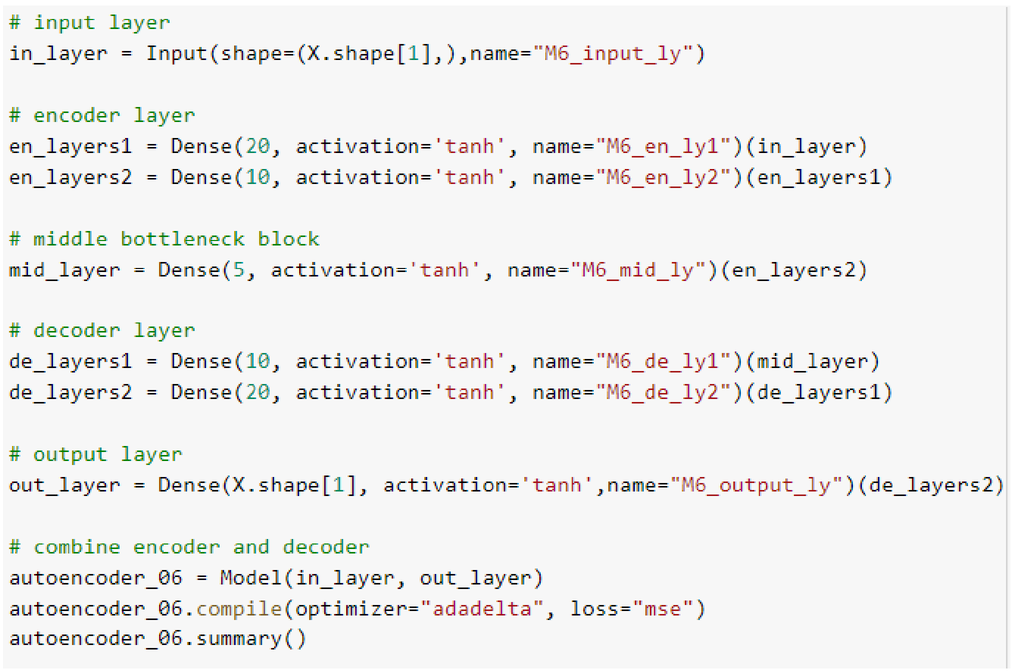


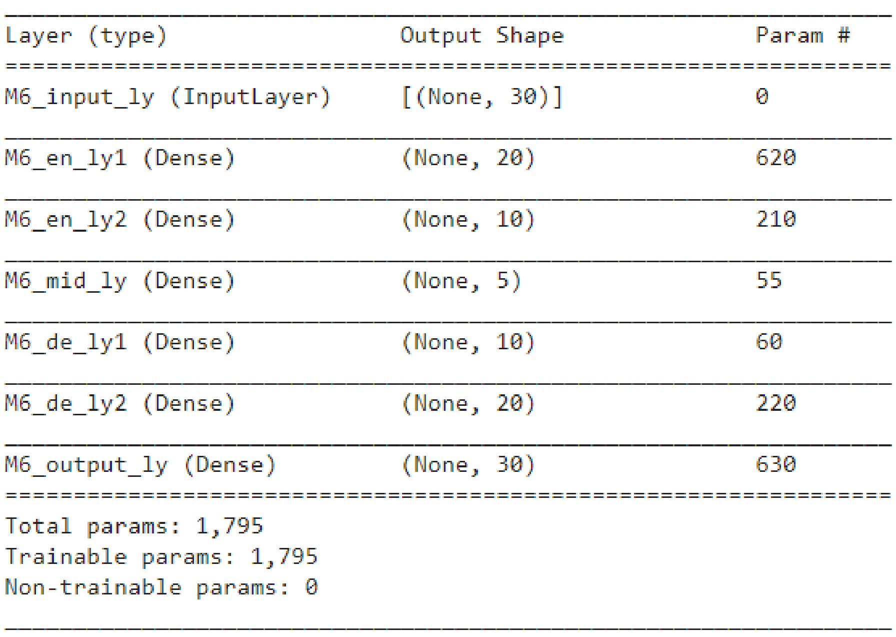
Figure 4.25: The Training progress of Model 5

4.2.6 Model 6: Under-Complete, With Middle Bottleneck Block



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M6_input_ly")

# encoder layer
en_layers1 = Dense(20, activation='tanh', name="M6_en_ly1")(in_layer)
en_layers2 = Dense(10, activation='tanh', name="M6_en_ly2")(en_layers1)

# middle bottleneck block
mid_layer = Dense(5, activation='tanh', name="M6_mid_ly")(en_layers2)

# decoder layer
de_layers1 = Dense(10, activation='tanh', name="M6_de_ly1")(mid_layer)
de_layers2 = Dense(20, activation='tanh', name="M6_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M6_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_06 = Model(in_layer, out_layer)
autoencoder_06.compile(optimizer="adadelta", loss="mse")
autoencoder_06.summary()
```

Figure 4.26: The architecture building of Model 6



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
M6_input_ly (InputLayer)     [(None, 30)]              0

_________________________________________________________________
M6_en_ly1 (Dense)            (None, 20)                620

_________________________________________________________________
M6_en_ly2 (Dense)            (None, 10)                210

_________________________________________________________________
M6_mid_ly (Dense)            (None, 5)                 55

_________________________________________________________________
M6_de_ly1 (Dense)            (None, 10)                60

_________________________________________________________________
M6_de_ly2 (Dense)            (None, 20)                220

_________________________________________________________________
M6_output_ly (Dense)         (None, 30)                630
=================================================================
Total params: 1,795
Trainable params: 1,795
Non-trainable params: 0

_________________________________________________________________
```

Figure 4.27: The Summary of Model 6

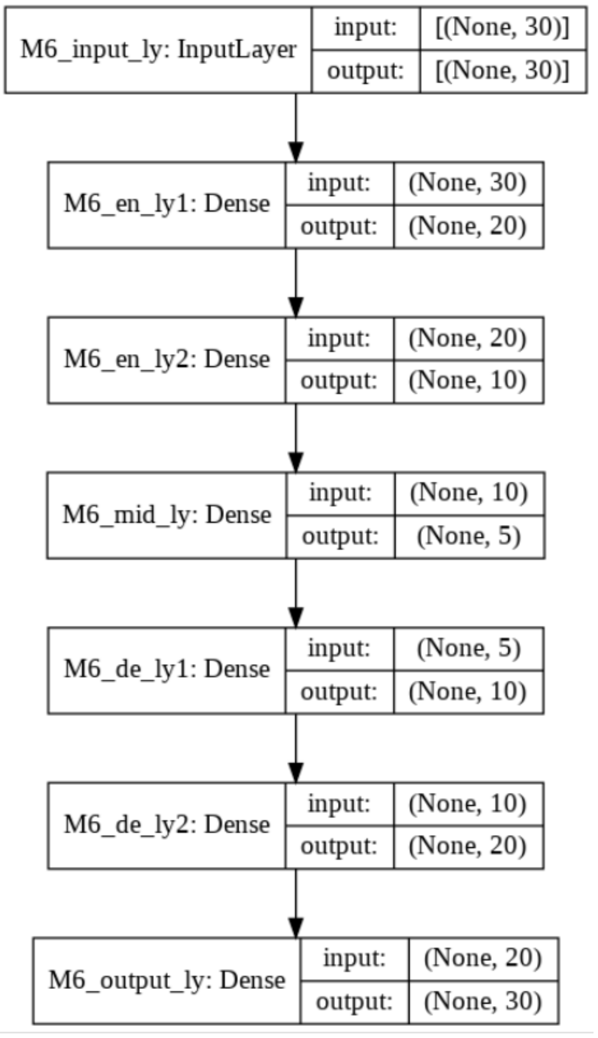


Figure 4.28: The Plot of Model 6

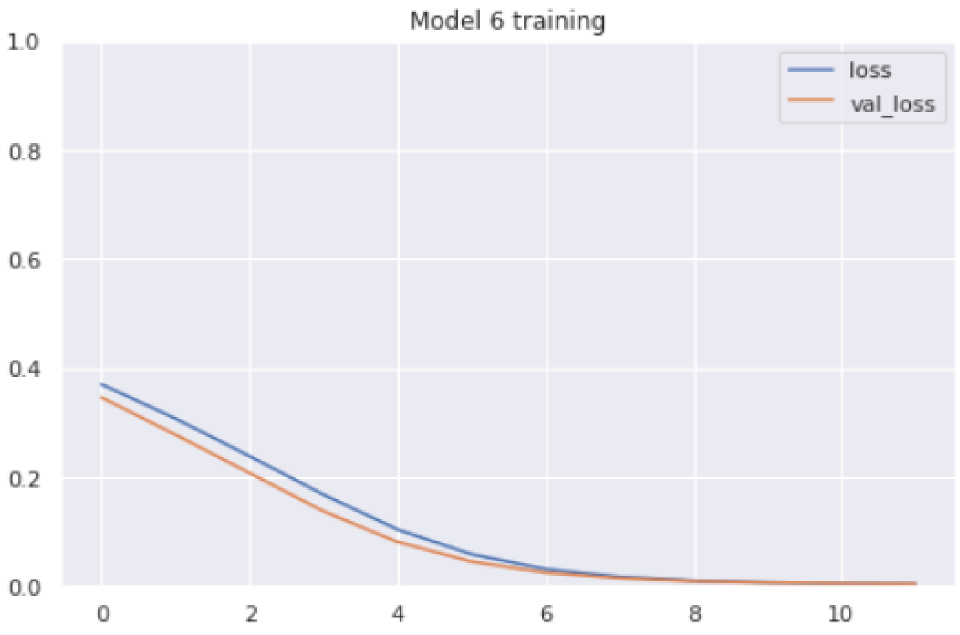
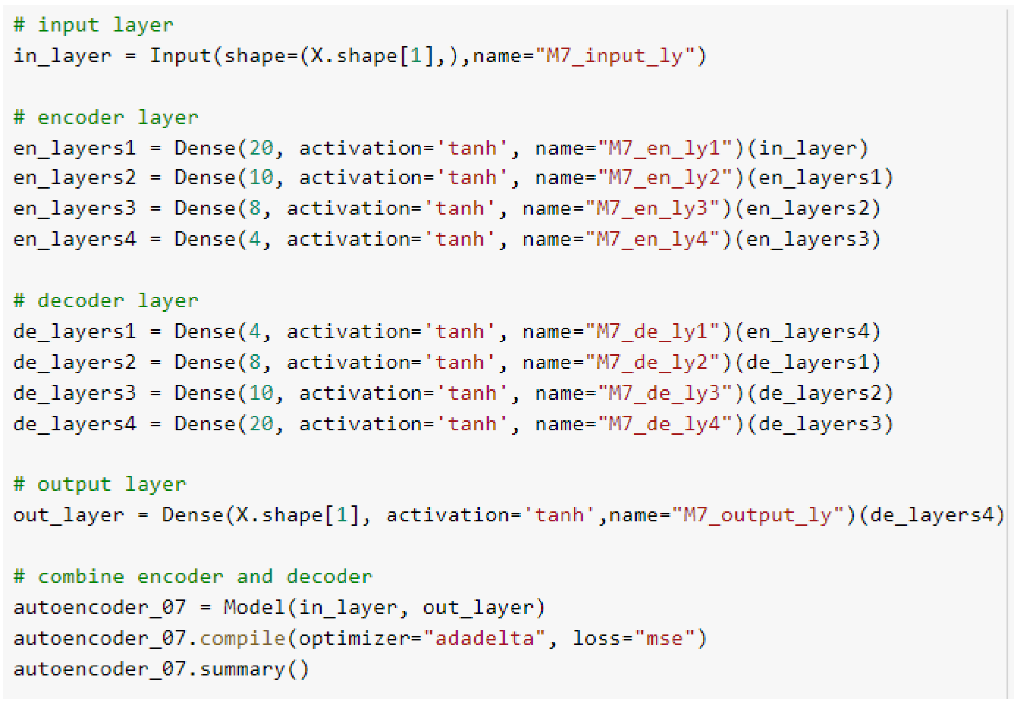


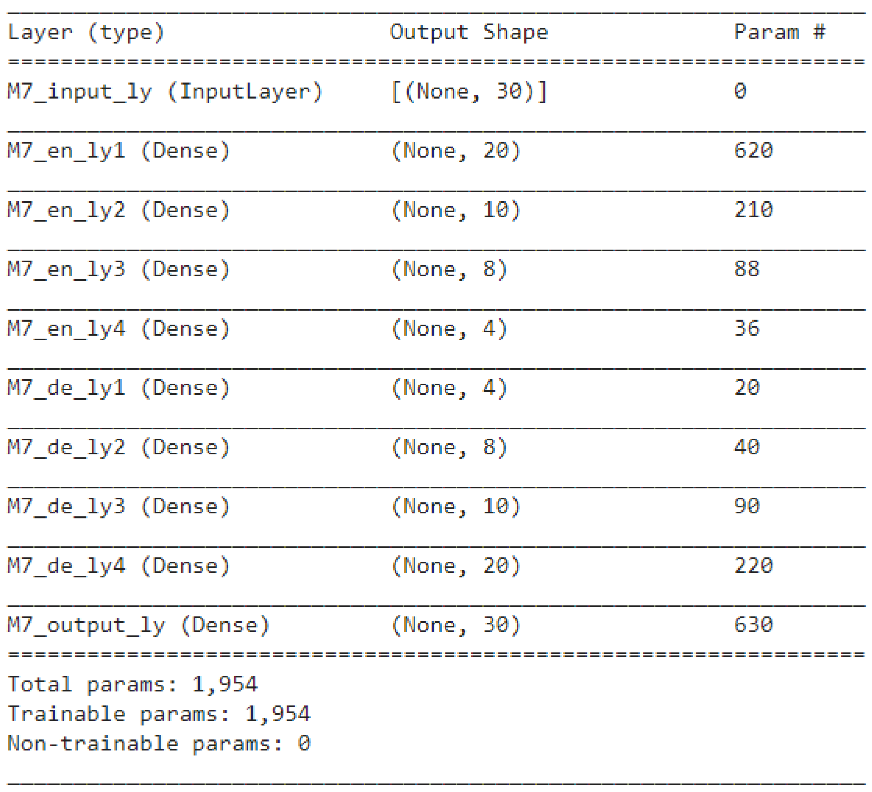
Figure 4.29: The Training progress of Model 6

4.2.7 Model 7: Under-Complete, Without Middle Bottleneck Block, Deeper/More Hidden Layers



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M7_input_ly")

# encoder layer
en_layers1 = Dense(20, activation='tanh', name="M7_en_ly1")(in_layer)
en_layers2 = Dense(10, activation='tanh', name="M7_en_ly2")(en_layers1)
en_layers3 = Dense(8, activation='tanh', name="M7_en_ly3")(en_layers2)
en_layers4 = Dense(4, activation='tanh', name="M7_en_ly4")(en_layers3)

# decoder layer
de_layers1 = Dense(4, activation='tanh', name="M7_de_ly1")(en_layers4)
de_layers2 = Dense(8, activation='tanh', name="M7_de_ly2")(de_layers1)
de_layers3 = Dense(10, activation='tanh', name="M7_de_ly3")(de_layers2)
de_layers4 = Dense(20, activation='tanh', name="M7_de_ly4")(de_layers3)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M7_output_ly")(de_layers4)

# combine encoder and decoder
autoencoder_07 = Model(in_layer, out_layer)
autoencoder_07.compile(optimizer="adadelta", loss="mse")
autoencoder_07.summary()
```

Figure 4.30: The architecture building of Model 7



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
M7_input_ly (InputLayer)     [(None, 30)]              0

_________________________________________________________________
M7_en_ly1 (Dense)            (None, 20)                620

_________________________________________________________________
M7_en_ly2 (Dense)            (None, 10)                210

_________________________________________________________________
M7_en_ly3 (Dense)            (None, 8)                 88

_________________________________________________________________
M7_en_ly4 (Dense)            (None, 4)                 36

_________________________________________________________________
M7_de_ly1 (Dense)            (None, 4)                 20

_________________________________________________________________
M7_de_ly2 (Dense)            (None, 8)                 40

_________________________________________________________________
M7_de_ly3 (Dense)            (None, 10)                90

_________________________________________________________________
M7_de_ly4 (Dense)            (None, 20)                220

_________________________________________________________________
M7_output_ly (Dense)         (None, 30)                630
=================================================================
Total params: 1,954
Trainable params: 1,954
Non-trainable params: 0

_________________________________________________________________
```

Figure 4.31: The Summary of Model 7

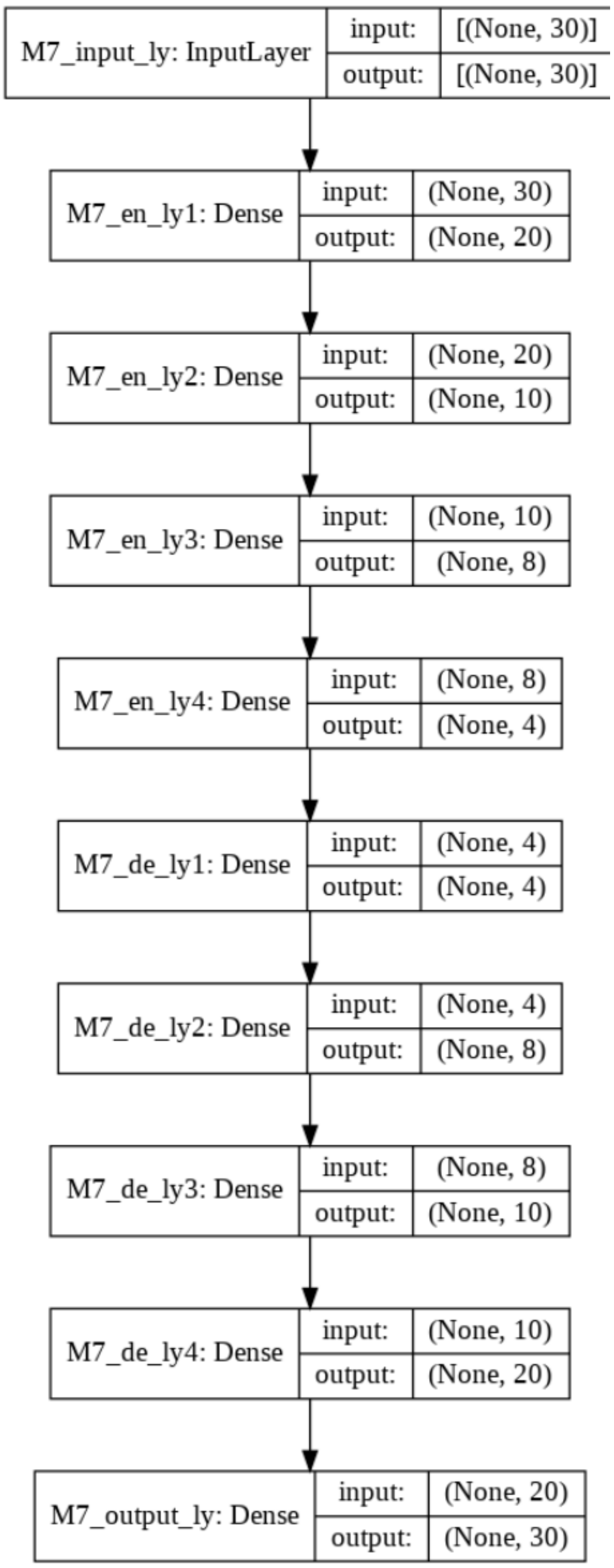


Figure 4.32: The Plot of Model 7

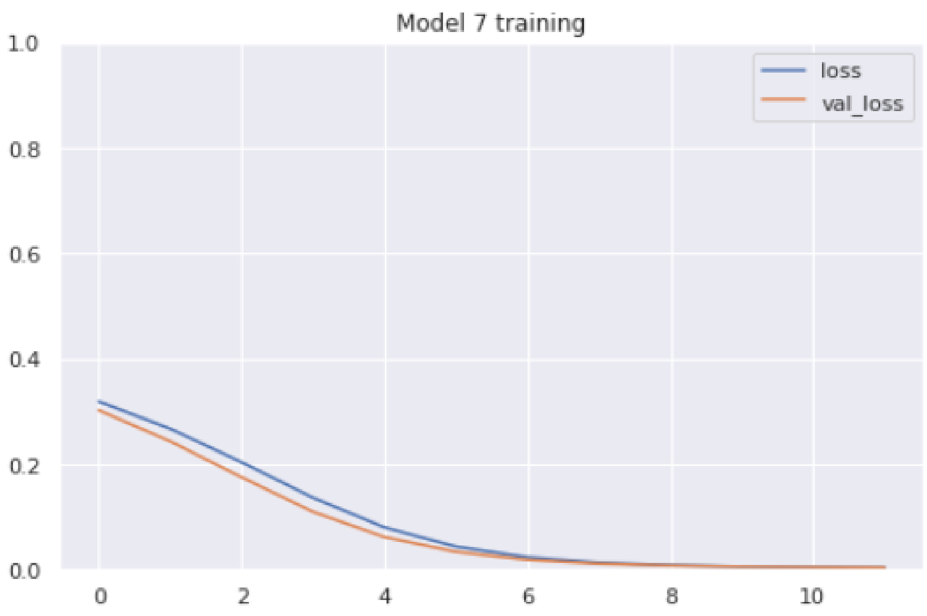
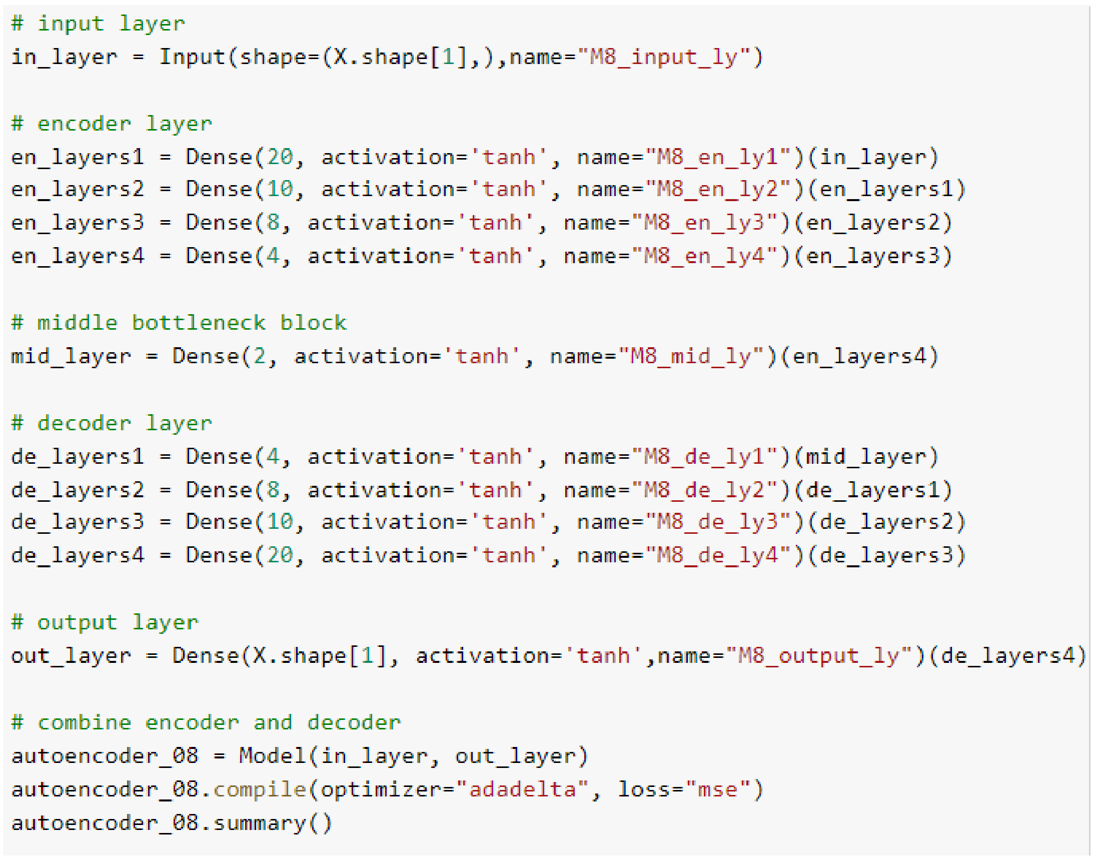


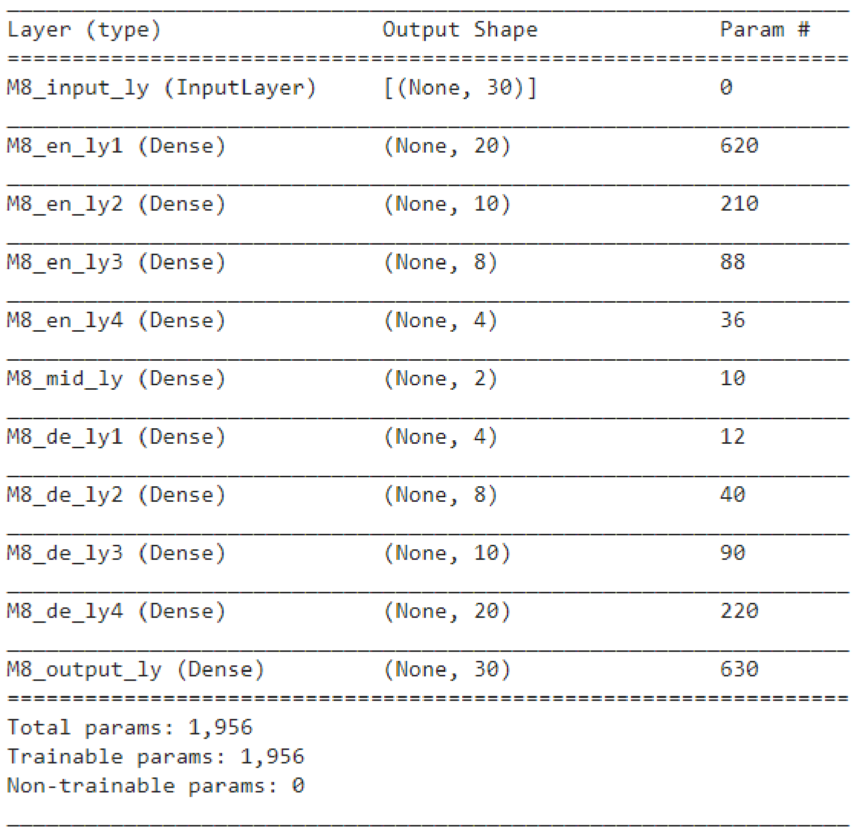
Figure 4.33: The Training progress of Model 7

4.2.8 Model 8: Under-Complete, With Middle Bottleneck Block, Deeper/More Hidden Layers



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M8_input_ly")

# encoder layer
en_layers1 = Dense(20, activation='tanh', name="M8_en_ly1")(in_layer)
en_layers2 = Dense(10, activation='tanh', name="M8_en_ly2")(en_layers1)
en_layers3 = Dense(8, activation='tanh', name="M8_en_ly3")(en_layers2)
en_layers4 = Dense(4, activation='tanh', name="M8_en_ly4")(en_layers3)

# middle bottleneck block
mid_layer = Dense(2, activation='tanh', name="M8_mid_ly")(en_layers4)

# decoder layer
de_layers1 = Dense(4, activation='tanh', name="M8_de_ly1")(mid_layer)
de_layers2 = Dense(8, activation='tanh', name="M8_de_ly2")(de_layers1)
de_layers3 = Dense(10, activation='tanh', name="M8_de_ly3")(de_layers2)
de_layers4 = Dense(20, activation='tanh', name="M8_de_ly4")(de_layers3)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M8_output_ly")(de_layers4)

# combine encoder and decoder
autoencoder_08 = Model(in_layer, out_layer)
autoencoder_08.compile(optimizer="adadelta", loss="mse")
autoencoder_08.summary()
```

Figure 4.34: The architecture building of Model 8



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
M8_input_ly (InputLayer)     [(None, 30)]              0

_________________________________________________________________
M8_en_ly1 (Dense)            (None, 20)                620

_________________________________________________________________
M8_en_ly2 (Dense)            (None, 10)                210

_________________________________________________________________
M8_en_ly3 (Dense)            (None, 8)                 88

_________________________________________________________________
M8_en_ly4 (Dense)            (None, 4)                 36

_________________________________________________________________
M8_mid_ly (Dense)            (None, 2)                 10

_________________________________________________________________
M8_de_ly1 (Dense)            (None, 4)                 12

_________________________________________________________________
M8_de_ly2 (Dense)            (None, 8)                 40

_________________________________________________________________
M8_de_ly3 (Dense)            (None, 10)                90

_________________________________________________________________
M8_de_ly4 (Dense)            (None, 20)                220

_________________________________________________________________
M8_output_ly (Dense)         (None, 30)                630
=================================================================
Total params: 1,956
Trainable params: 1,956
Non-trainable params: 0

_________________________________________________________________
```

Figure 4.35: The Summary of Model 8

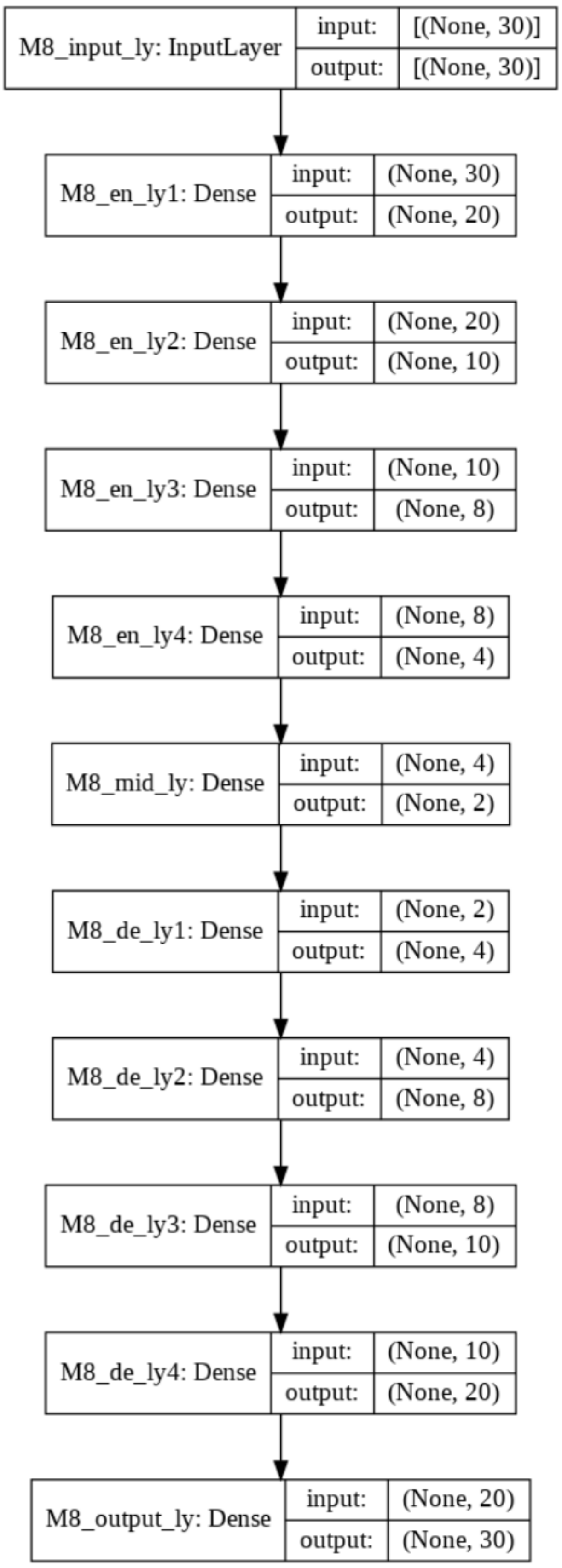


Figure 4.36: The Plot of Model 8

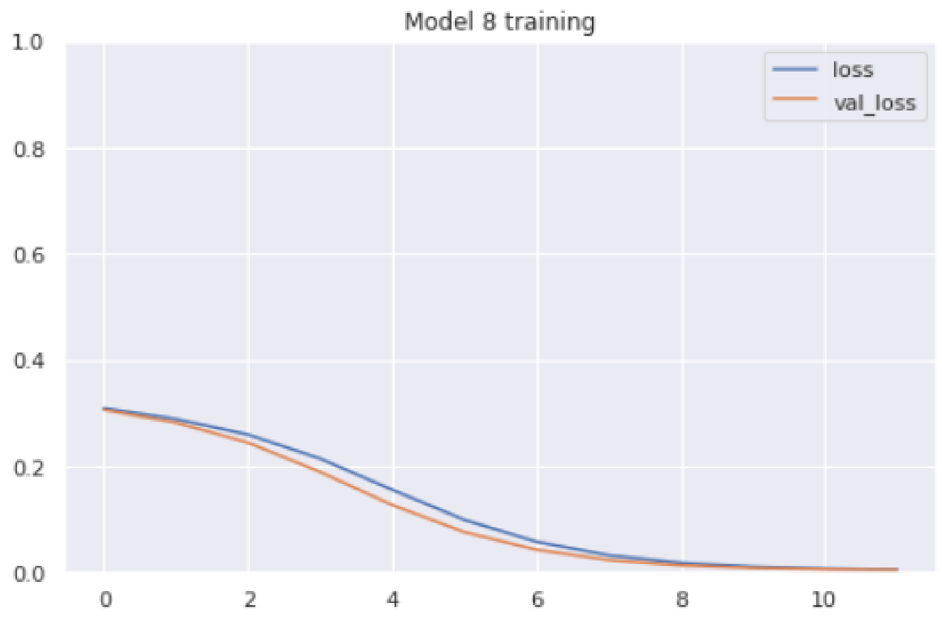


Figure 4.37: The Training progress of Model 8

4.2.9 Encoded Layer Representation and Random Forrest Classification

After finishing training all the models, the encoded layers are extracted from the auto-encoder to predict the encoded representation of each instance as shown in Figure 4.38.

```
encoder_layer_rep_m1 = Sequential([
                    autoencoder_01.layers[0],
                    autoencoder_01.layers[1],
                    autoencoder_01.layers[2]
])
```

Figure 4.38: The encoded layers extracted from Model 1

The scaled data is split into 30% train data and 70% test data to reduce the training time. This is not an issue here as the main objective is to evaluate how the architecture affects the performance. The train and test data are transformed by using the extracted encoded layers before using it for Random Forest training and classification as shown in Figure 4.39. The performance will then be evaluated in terms of accuracy, precision, recall, f1 score and PR-AUC.

```
train_x_transform_m1 = encoder_layer_rep_m1.predict(train_x)
val_x_transform_m1 = encoder_layer_rep_m1.predict(val_x)
clf_m1 = RandomForestClassifier().fit(train_x_transform_m1, train_y)
pred_y_m1 = clf_m1.predict(val_x_transform_m1)
```

Figure 4.39: Training of RF model by using transformed data (Model 1)

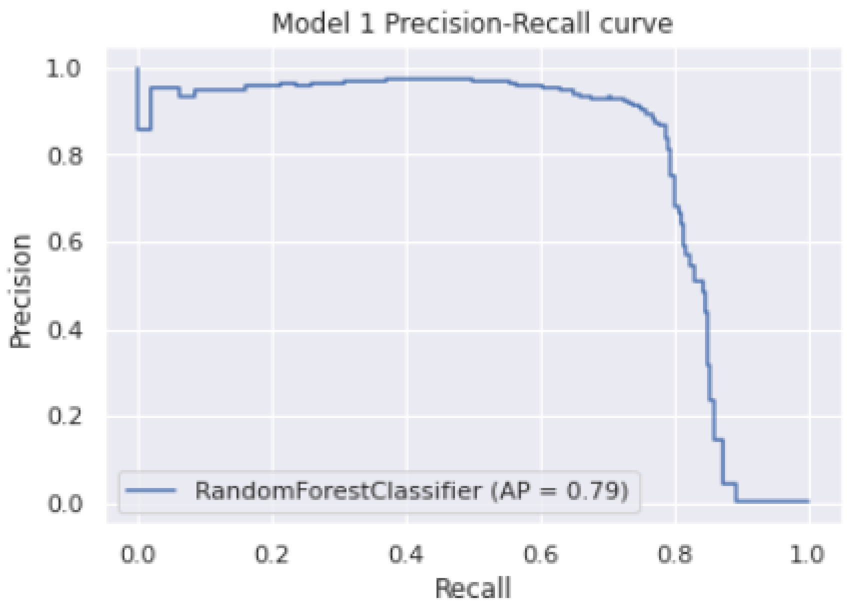
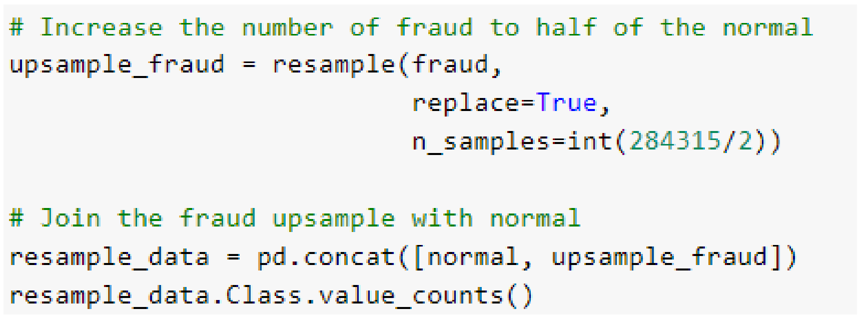


Figure 4.40: Model 1 Precision Recall Curve

4.3 Resampling Technique

To apply the resampling technique, the instances are first separated into normal and fraud groups. Then the number of the fraud instances are increased to half the number of normal instances by using the resample function as shown in Figure 4.41.



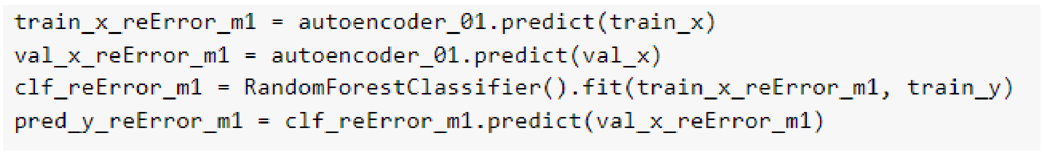
```
# Increase the number of fraud to half of the normal
upsample_fraud = resample(fraud,
                          replace=True,
                          n_samples=int(284315/2))

# Join the fraud upsample with normal
resample_data = pd.concat([normal, upsample_fraud])
resample_data.Class.value_counts()
```

Figure 4.41: Increase the number of Fraud instances using resample function

After the resampling process, the data are scaled and split. Model 1 and Model 5 are used to evaluate the effectiveness of resampling techniques.

4.4 Using Reconstruction Representation to Train RF Model

In this section, the reconstruction errors from the auto-encoder models are used for RF model training and classification instead of the encoded representation. Model 1 and Model 5 are used here for comparison.



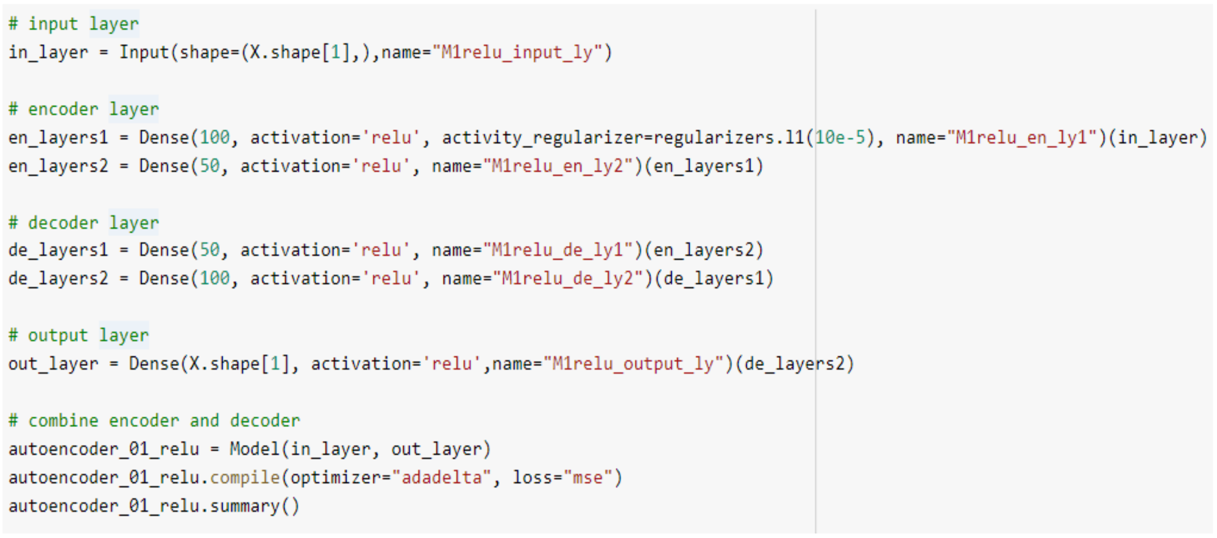
```
train_x_reError_m1 = autoencoder_01.predict(train_x)
val_x_reError_m1 = autoencoder_01.predict(val_x)
clf_reError_m1 = RandomForestClassifier().fit(train_x_reError_m1, train_y)
pred_y_reError_m1 = clf_reError_m1.predict(val_x_reError_m1)
```

Figure 4.42: Reconstruction error from Model 1 is used for RF Model training

4.5 Optimizer and Activation Function Tuning

All the tuning done in this section are based on Model 1.

4.5.1 Optimizer: adadelta, Activation Function: RELU



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M1relu_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='relu', activity_regularizer=regularizers.l1(10e-5), name="M1relu_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='relu', name="M1relu_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(50, activation='relu', name="M1relu_de_ly1")(en_layers2)
de_layers2 = Dense(100, activation='relu', name="M1relu_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='relu',name="M1relu_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_01_relu = Model(in_layer, out_layer)
autoencoder_01_relu.compile(optimizer="adadelta", loss="mse")
autoencoder_01_relu.summary()
```

Figure 4.43: Model 1 with adadelta and RELU

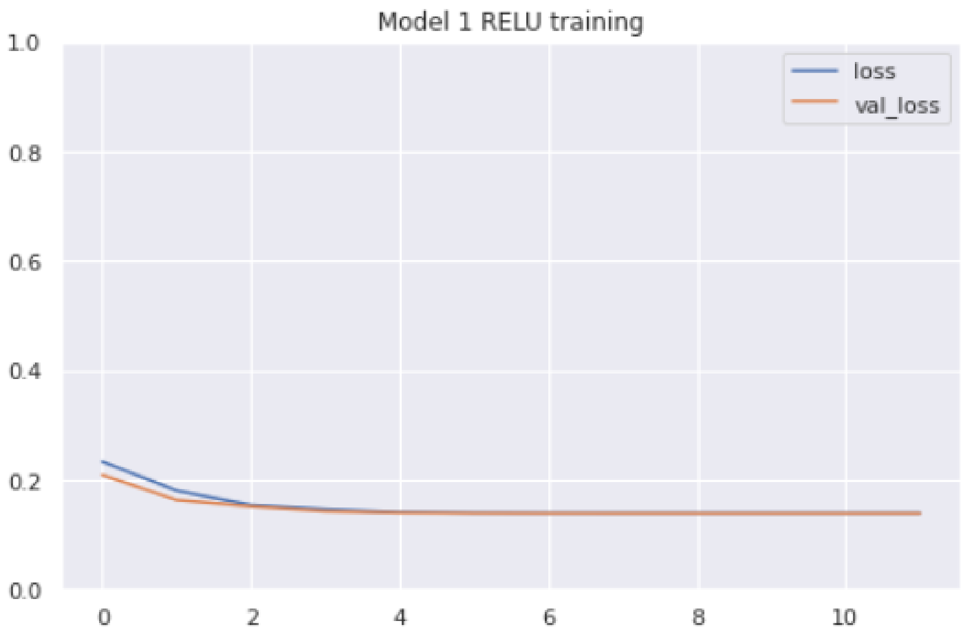
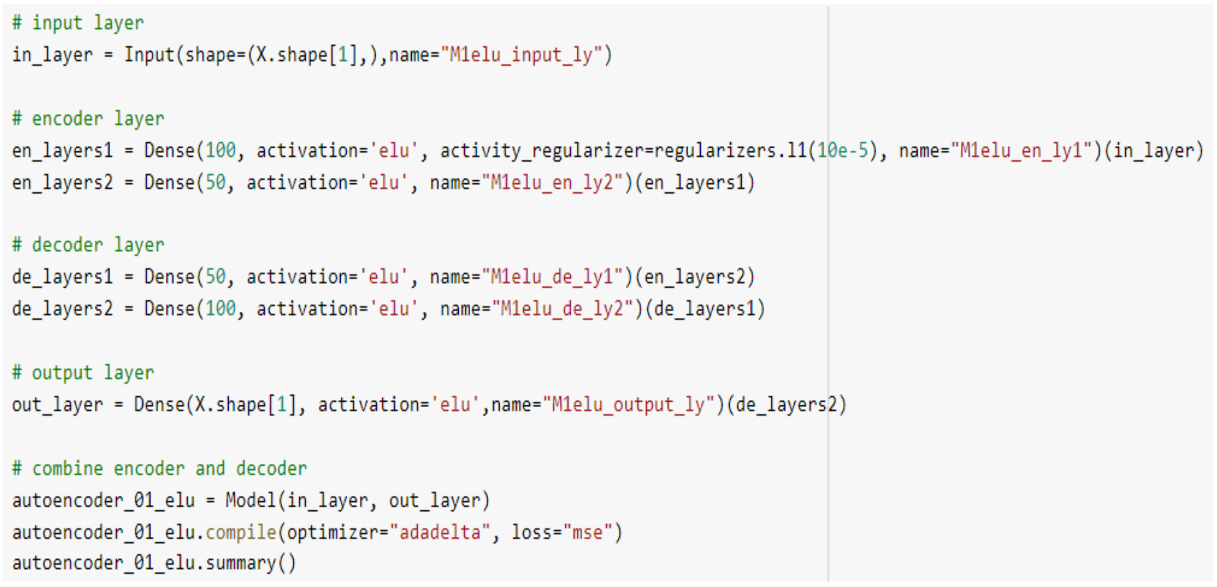


Figure 4.44: The Training progress of Model 1 RELU

4.5.2 Optimizer: adadelta, Activation Function: ELU


```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M1elu_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='elu', activity_regularizer=regularizers.l1(10e-5), name="M1elu_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='elu', name="M1elu_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(50, activation='elu', name="M1elu_de_ly1")(en_layers2)
de_layers2 = Dense(100, activation='elu', name="M1elu_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='elu',name="M1elu_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_01_elu = Model(in_layer, out_layer)
autoencoder_01_elu.compile(optimizer="adadelta", loss="mse")
autoencoder_01_elu.summary()
```
Figure 4.45: Model 1 with adadelta and ELU

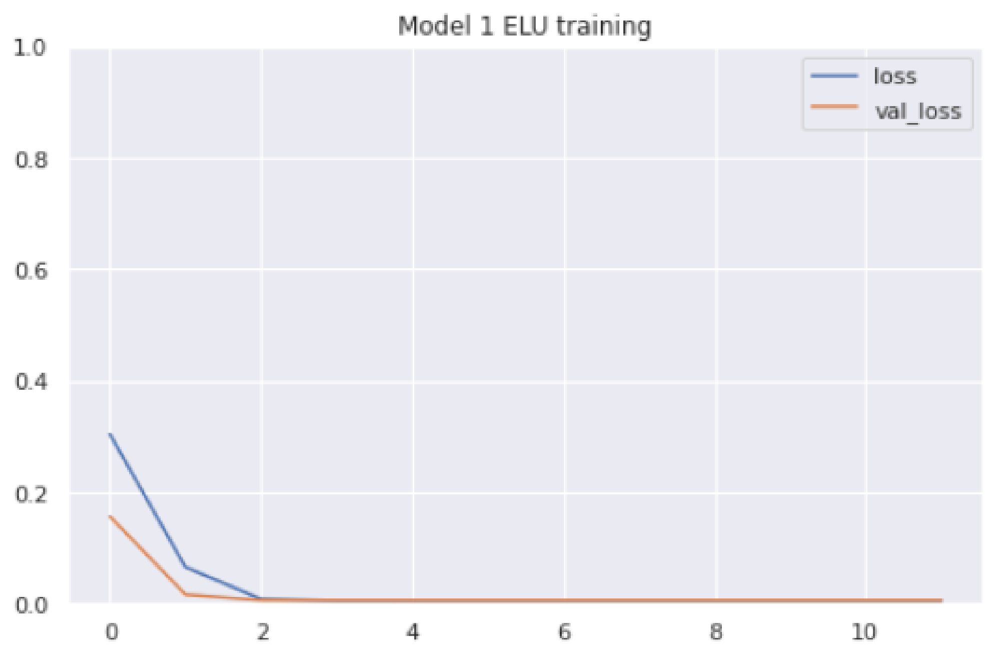
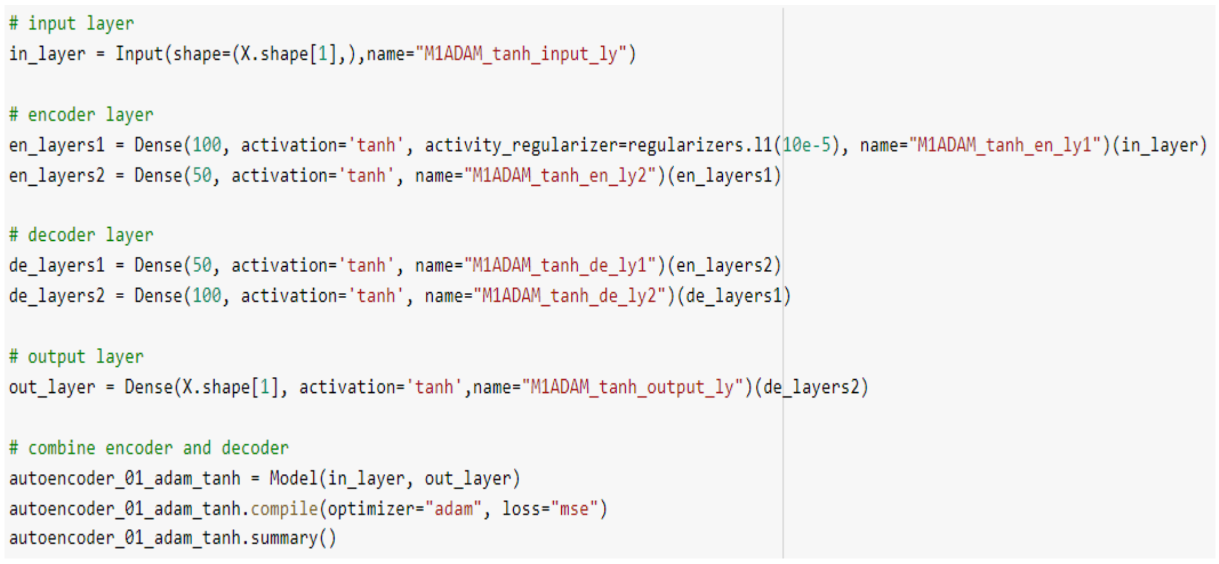


Figure 4.46: The Training progress of Model 1 ELU

4.5.3 Optimizer: adam, Activation Function: tanh



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M1ADAM_tanh_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='tanh', activity_regularizer=regularizers.l1(10e-5), name="M1ADAM_tanh_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='tanh', name="M1ADAM_tanh_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(50, activation='tanh', name="M1ADAM_tanh_de_ly1")(en_layers2)
de_layers2 = Dense(100, activation='tanh', name="M1ADAM_tanh_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='tanh',name="M1ADAM_tanh_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_01_adam_tanh = Model(in_layer, out_layer)
autoencoder_01_adam_tanh.compile(optimizer="adam", loss="mse")
autoencoder_01_adam_tanh.summary()
```

Figure 4.47: Model 1 with adam and tanh

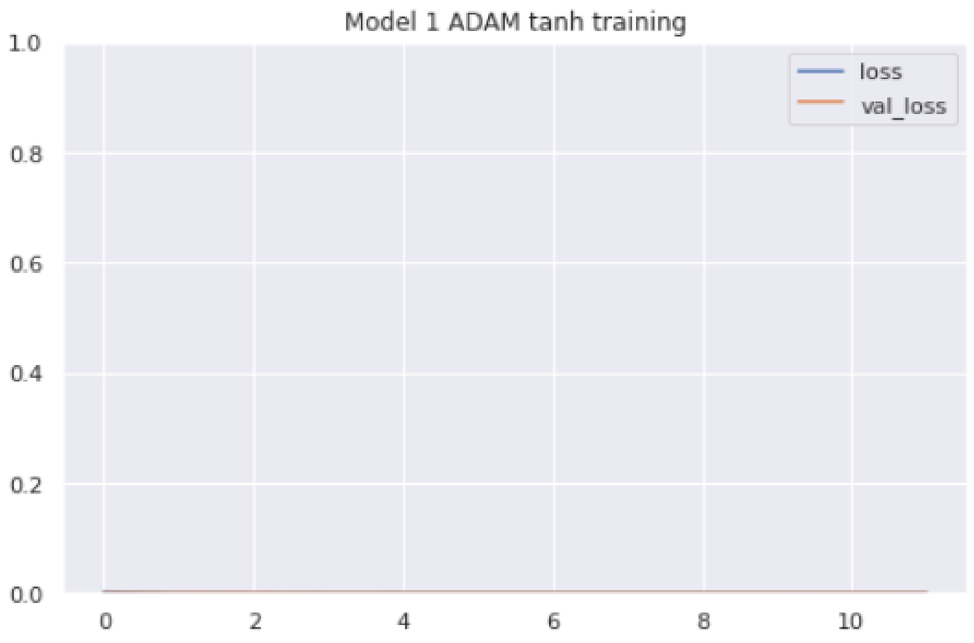
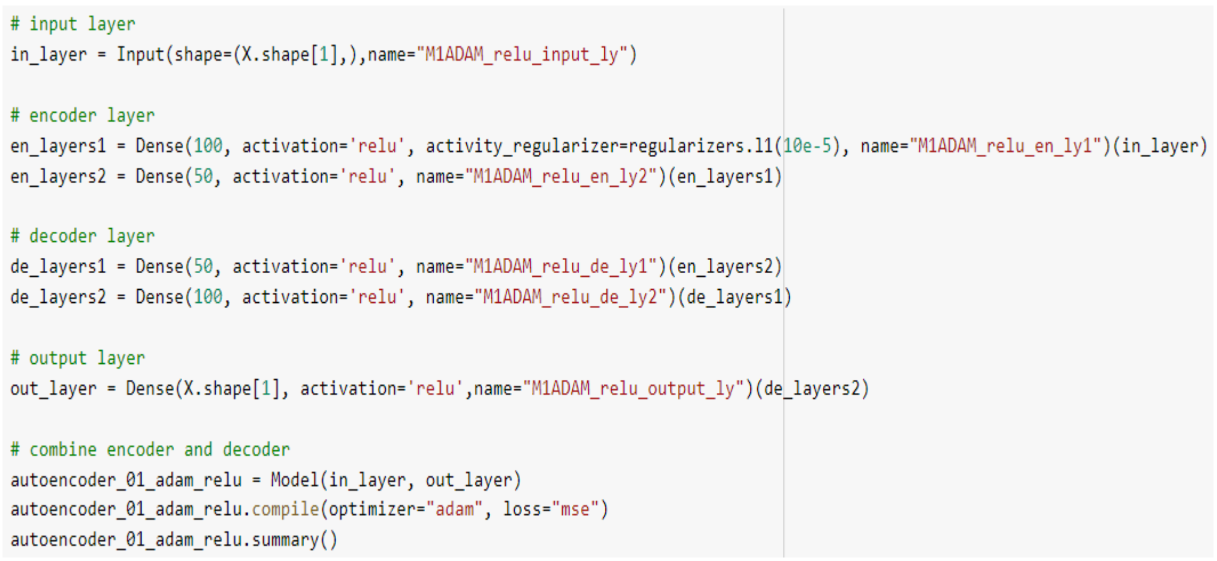


Figure 4.48: The Training progress of Model 1 adam and tanh

4.5.4 Optimizer: adam, Activation Function: RELU



```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M1ADAM_relu_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='relu', activity_regularizer=regularizers.l1(10e-5), name="M1ADAM_relu_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='relu', name="M1ADAM_relu_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(50, activation='relu', name="M1ADAM_relu_de_ly1")(en_layers2)
de_layers2 = Dense(100, activation='relu', name="M1ADAM_relu_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='relu',name="M1ADAM_relu_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_01_adam_relu = Model(in_layer, out_layer)
autoencoder_01_adam_relu.compile(optimizer="adam", loss="mse")
autoencoder_01_adam_relu.summary()
```

Figure 4.49: Model 1 with adam and RELU

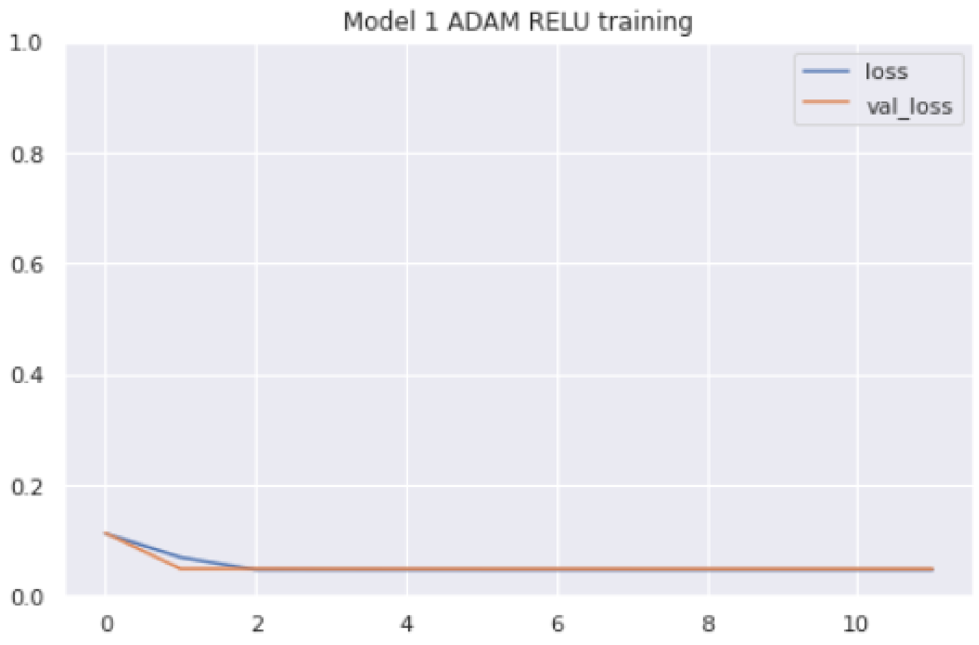
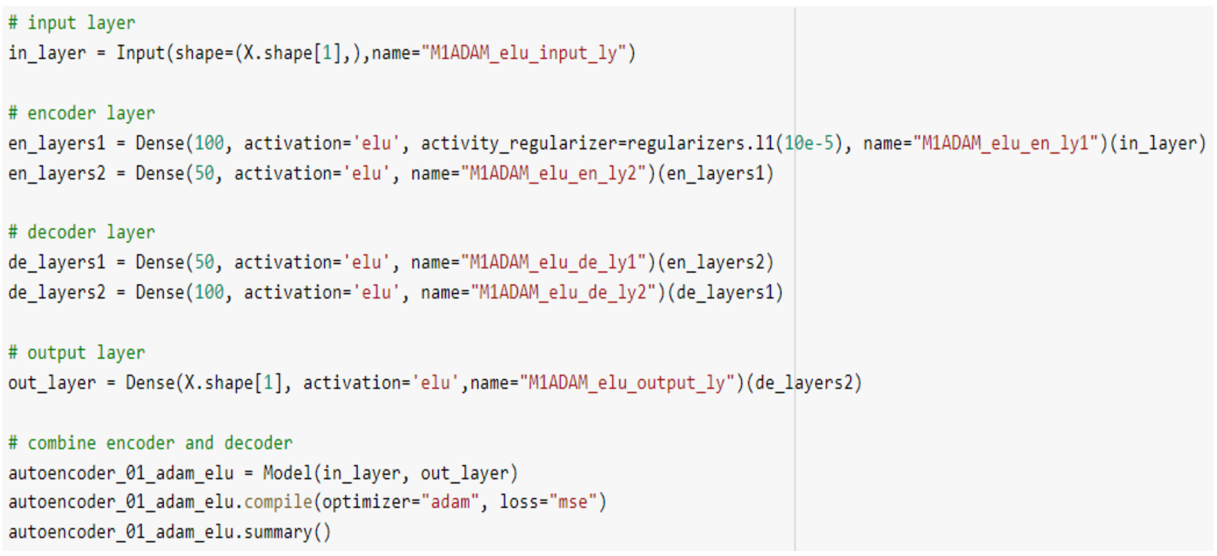


Figure 4.50: The Training progress of Model 1 adam and RELU

4.5.5 Optimizer: adam, Activation Function: ELU


```
# input layer
in_layer = Input(shape=(X.shape[1],),name="M1ADAM_elu_input_ly")

# encoder layer
en_layers1 = Dense(100, activation='elu', activity_regularizer=regularizers.l1(10e-5), name="M1ADAM_elu_en_ly1")(in_layer)
en_layers2 = Dense(50, activation='elu', name="M1ADAM_elu_en_ly2")(en_layers1)

# decoder layer
de_layers1 = Dense(50, activation='elu', name="M1ADAM_elu_de_ly1")(en_layers2)
de_layers2 = Dense(100, activation='elu', name="M1ADAM_elu_de_ly2")(de_layers1)

# output layer
out_layer = Dense(X.shape[1], activation='elu',name="M1ADAM_elu_output_ly")(de_layers2)

# combine encoder and decoder
autoencoder_01_adam_elu = Model(in_layer, out_layer)
autoencoder_01_adam_elu.compile(optimizer="adam", loss="mse")
autoencoder_01_adam_elu.summary()
```
Figure 4.51: Model 1 with adam and ELU

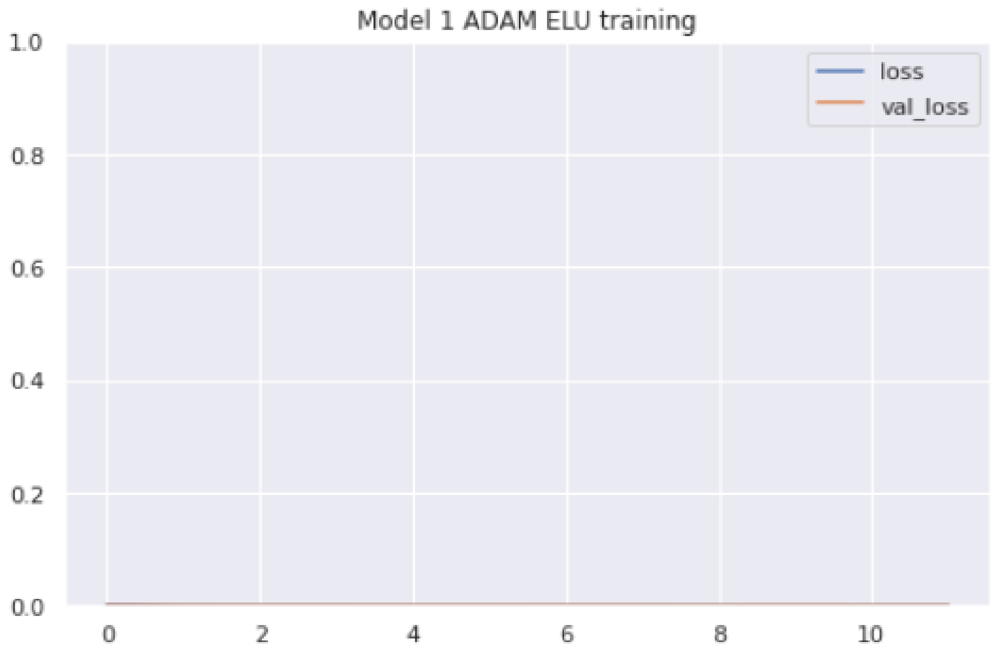


Figure 4.52: The Training progress of Model 1 adam and ELU

SECTION 5

RESULTS AND ANALYSIS

5.1 Visualization of Data Representation

The distribution of the data class is mixed around in the 2 dimensional TSNE transformed space as shown in Figure 4.5 in the previous section. For each model, the class distribution in the encoded layer is presented in this section to view the effectiveness of the model in differentiating the two classes.

5.1.1 Model 1

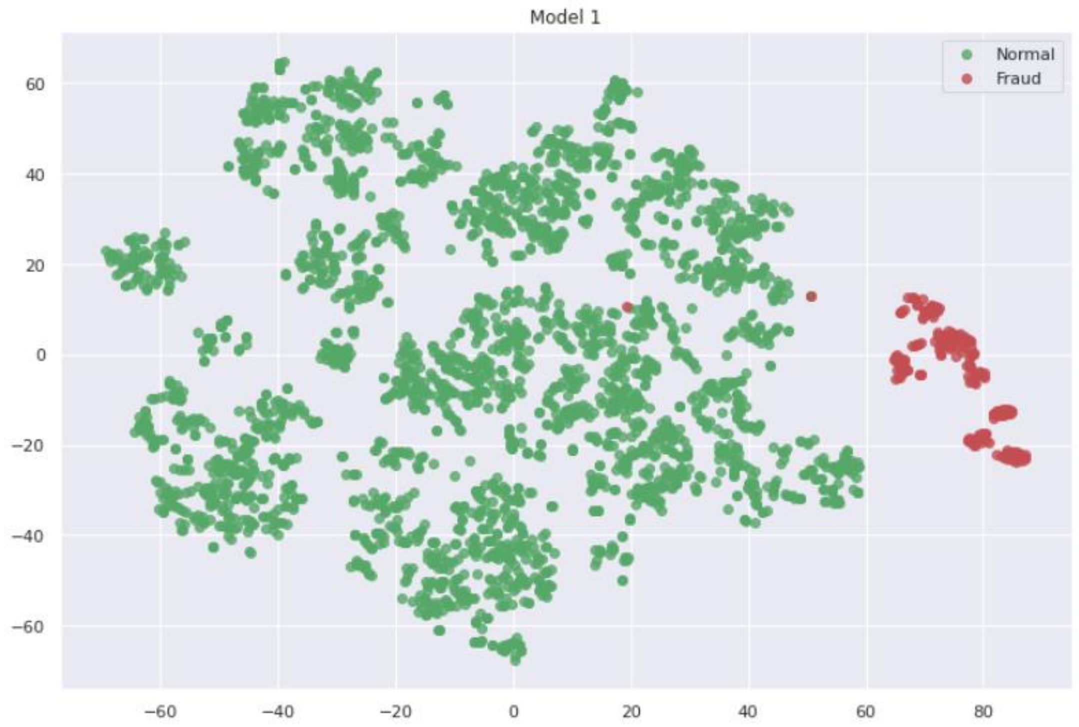


Figure 5.1: The encoded representaion of instances using Model 1

5.1.2 Model 2

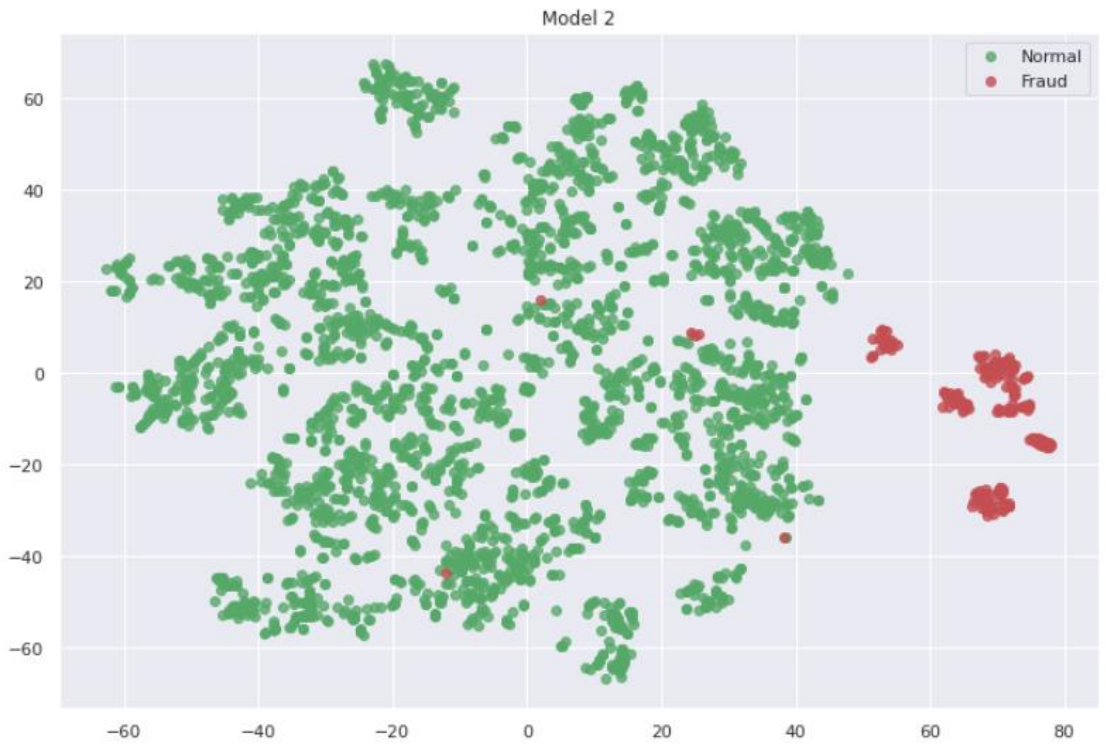


Figure 5.2: The encoded representaion of instances using Model 2

5.1.3 Model 3

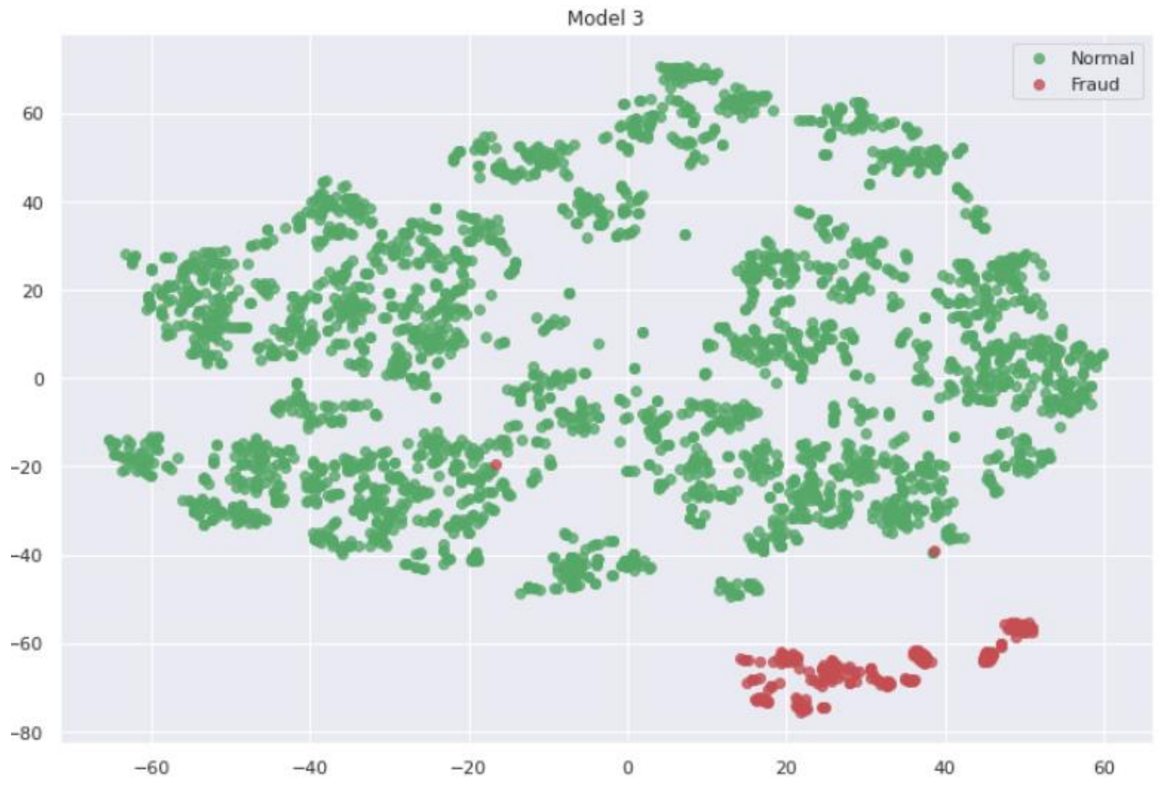


Figure 5.3: The encoded representaion of instances using Model 3

5.1.4 Model 4

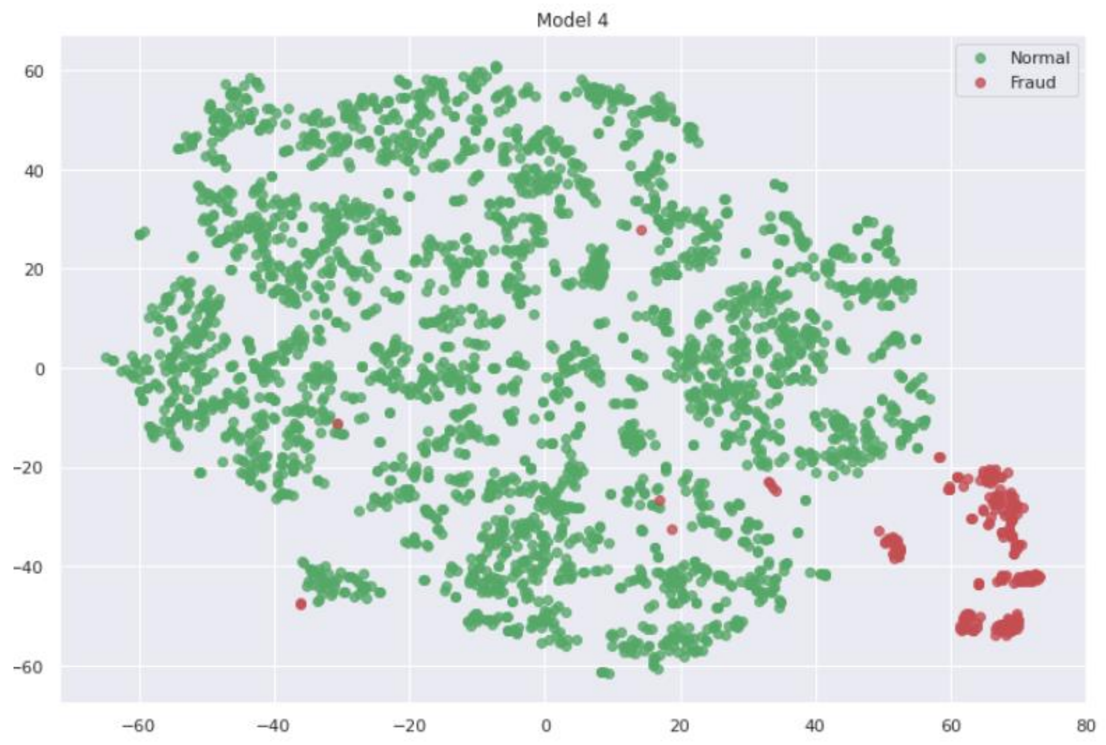


Figure 5.4: The encoded representaion of instances using Model 4

5.1.5 Model 5

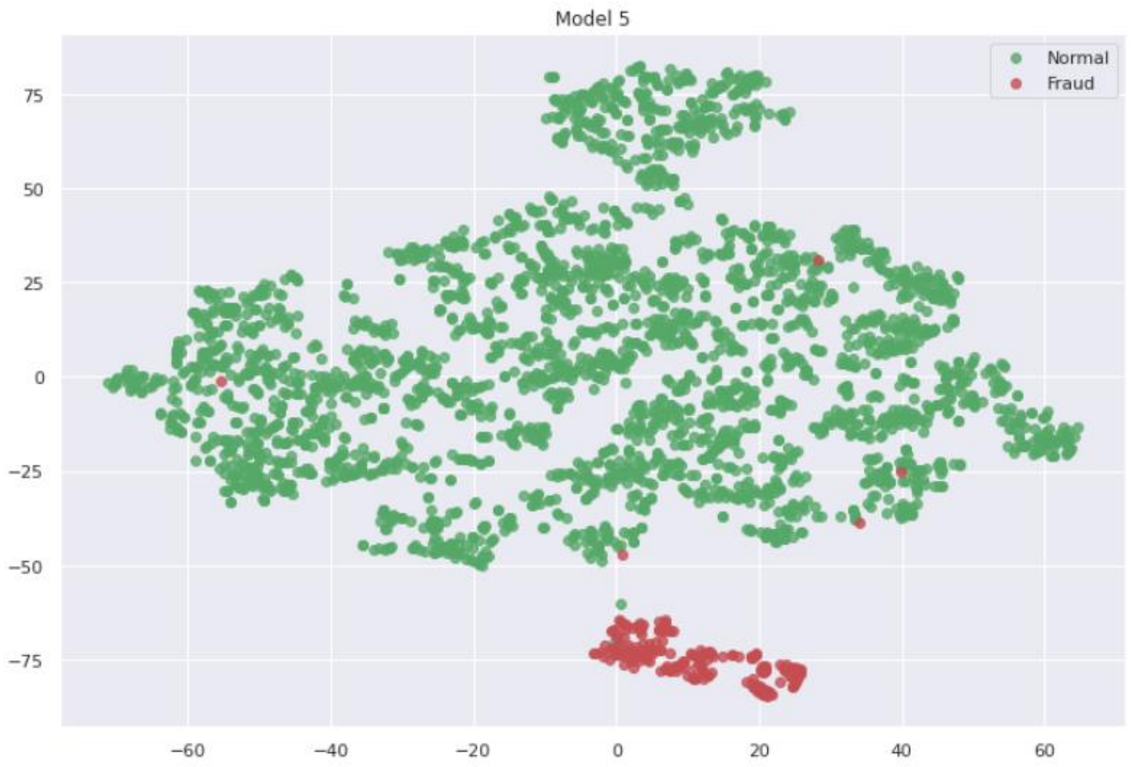


Figure 5.5: The encoded representaion of instances using Model 5

5.1.6 Model 6

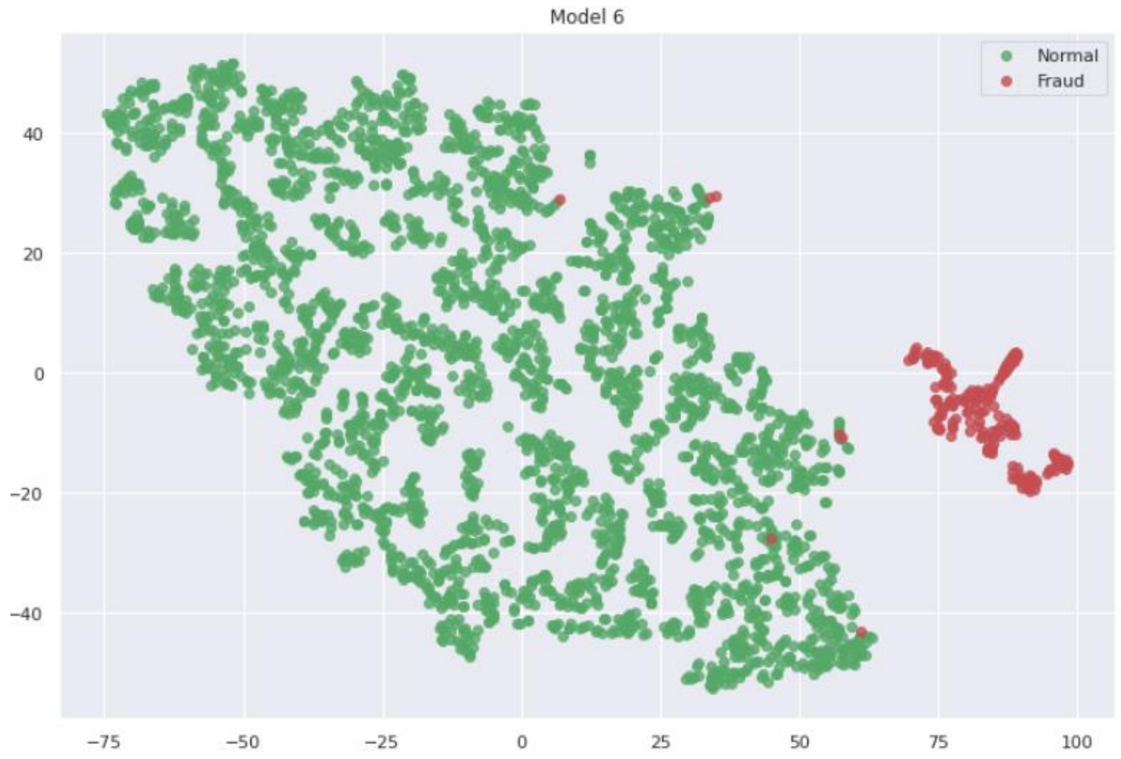


Figure 5.6: The encoded representaion of instances using Model 6

5.1.7 Model 7

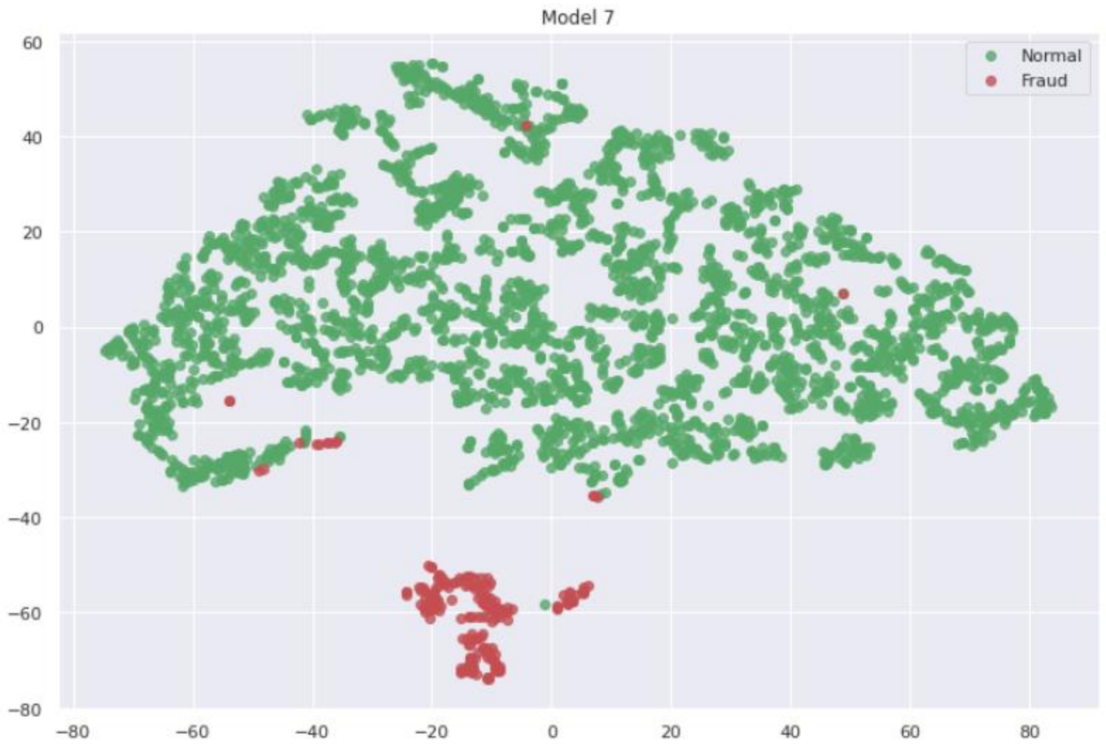


Figure 5.7: The encoded representaion of instances using Model 7

5.1.8 Model 8

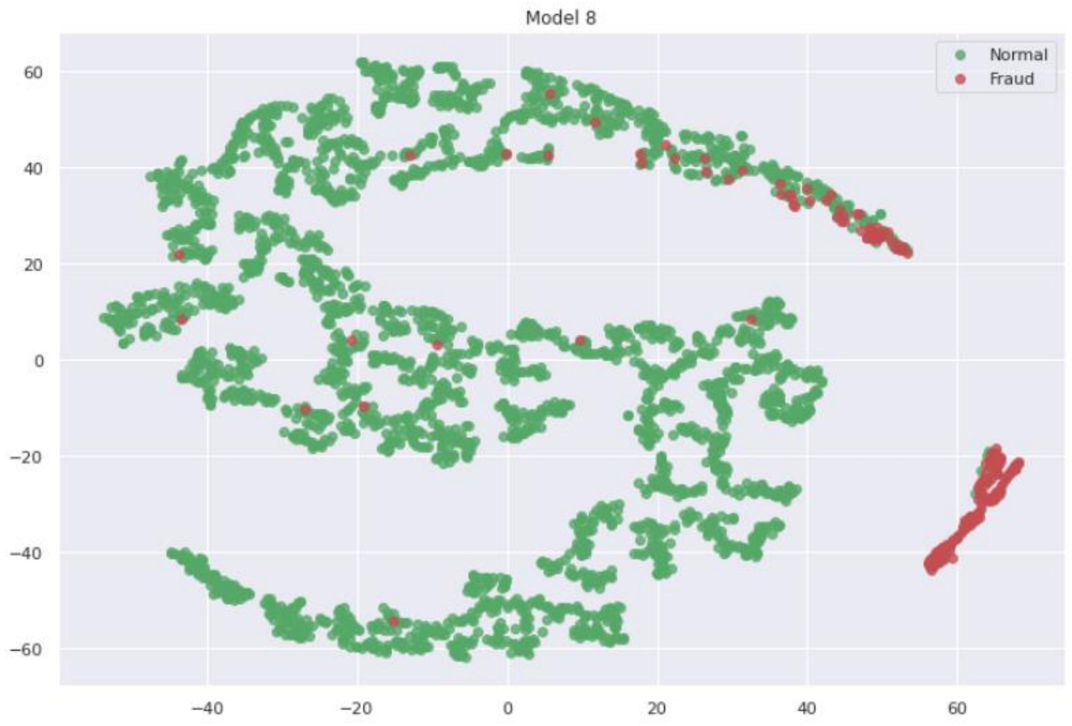


Figure 5.8: The encoded representaion of instances using Model 8

5.1.9 Model 1 adadelta, RELU

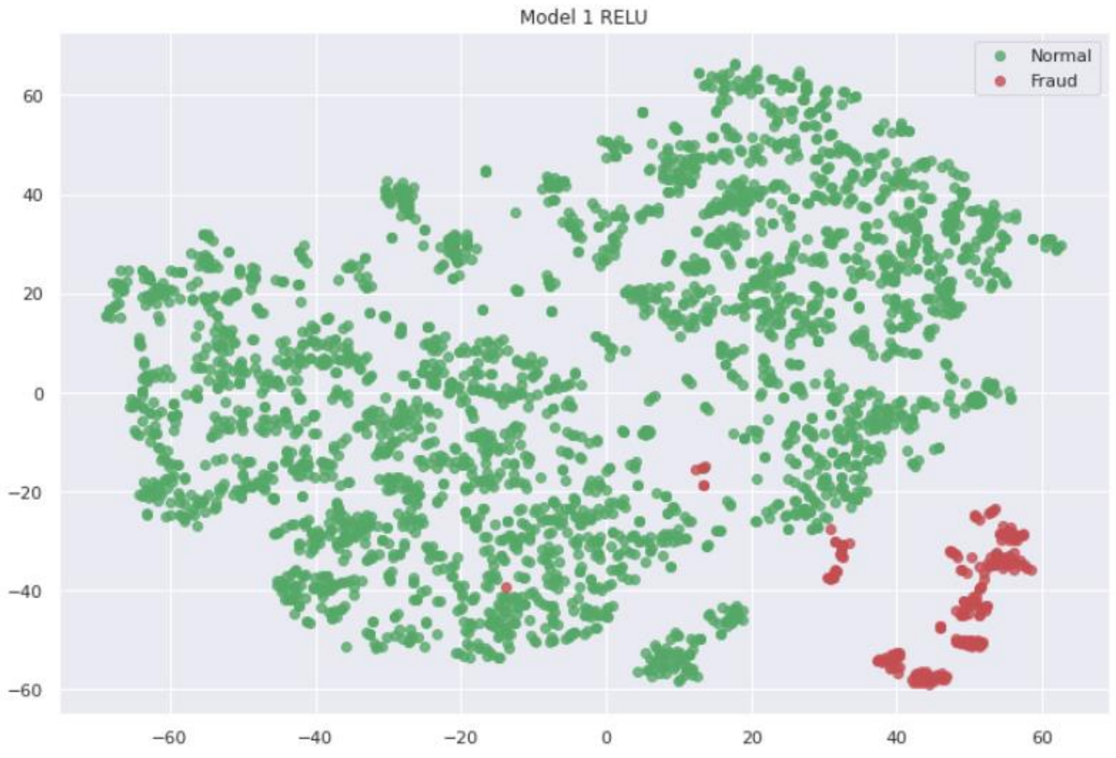


Figure 5.9: The encoded representaion of instances using Model 1 adadelta RELU

5.1.10 Model 1 adadelta, ELU

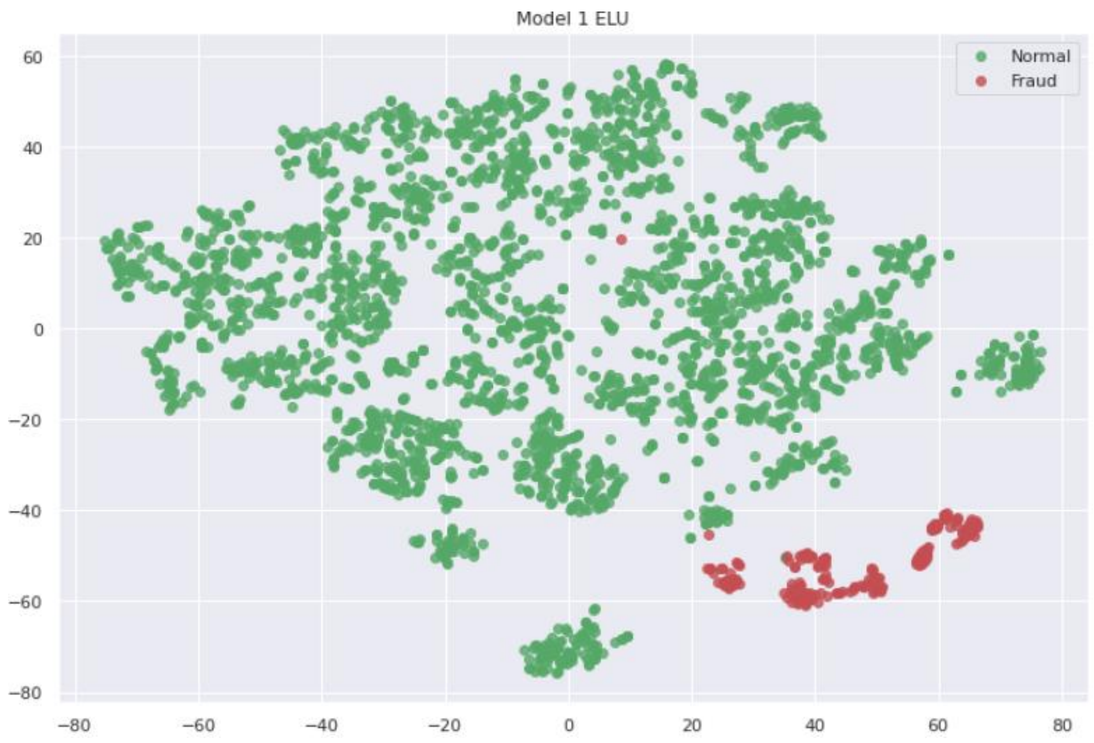


Figure 5.10: The encoded representaion of instances using Model 1 adadelta ELU

5.1.11 Model 1 adam, tanh

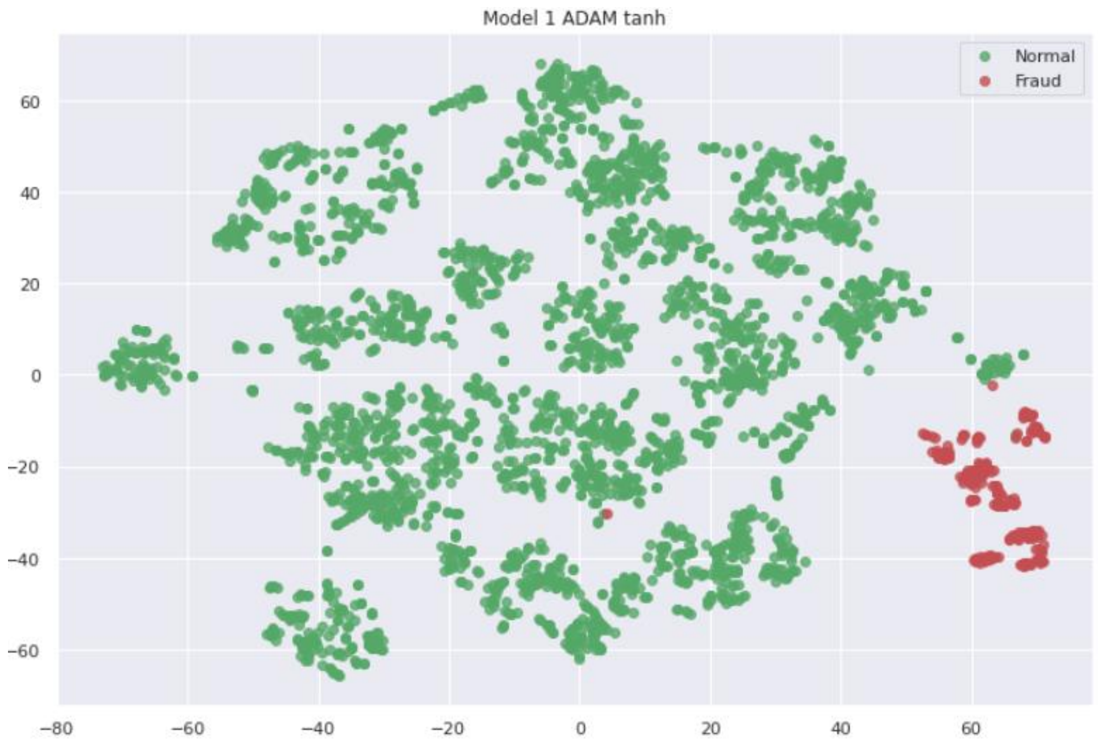


Figure 5.11: The encoded representaion of instances using Model 1 adam tanh

5.1.12 Model 1 adam, RELU

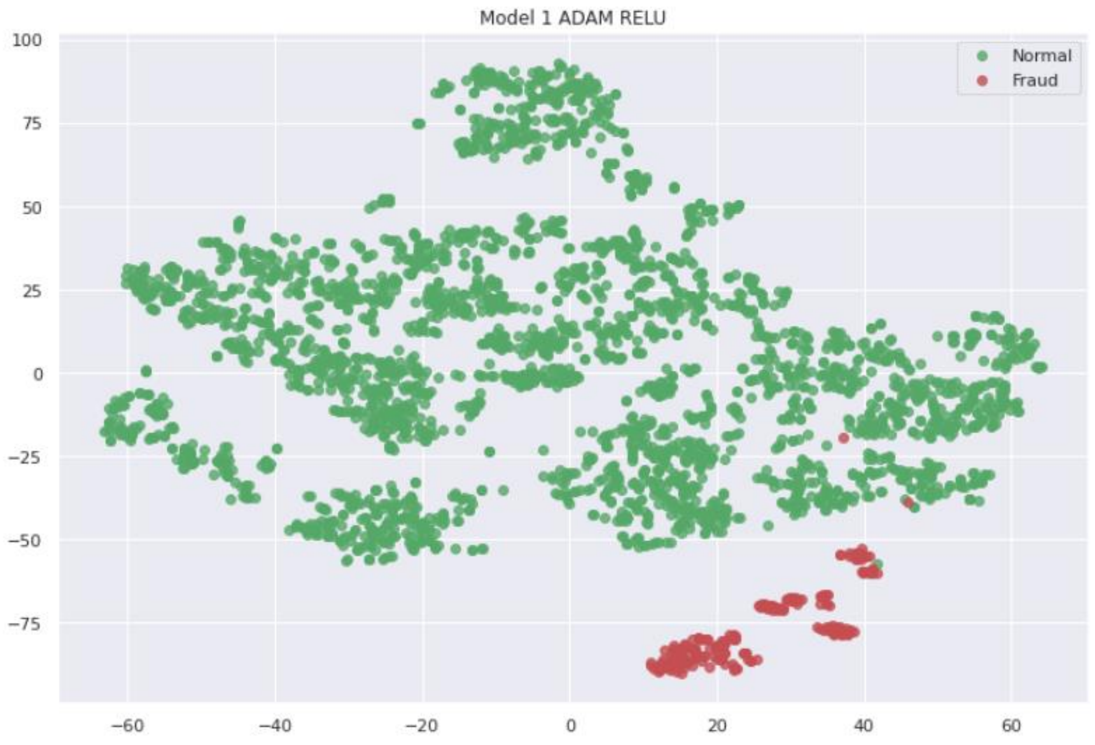


Figure 5.12: The encoded representaion of instances using Model 1 adam RELU

5.1.13 Model 1 adam, ELU

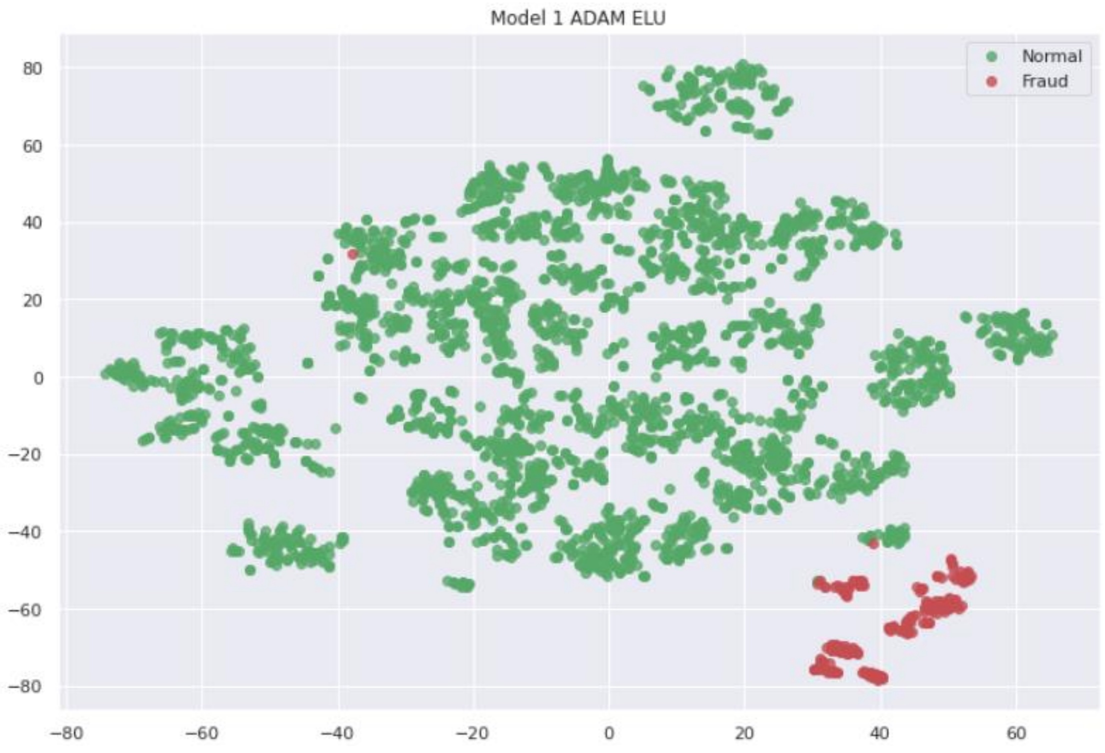


Figure 5.13: The encoded representaion of instances using Model 1 adam ELU

5.2 Summary of Results

5.2.1 Auto-Encoder with Different Architectures

A based RF model without auto-encoder is added for reference purpose.

Table 5.1: Performance of Various Auto-Encoder Models

Model	Accuracy	Precision	Recall	F1 Score	PR-AUC
RF	0.99949	0.91186	0.78198	0.84194	0.84
1	0.99942	0.92884	0.72093	0.81178	0.79
2	0.99929	0.93939	0.63081	0.75478	0.75
3	0.99934	0.90189	0.69477	0.78489	0.77
4	0.99937	0.91954	0.69767	0.79339	0.76
5	0.99902	0.95706	0.45349	0.61538	0.62
6	0.99879	0.80357	0.39244	0.52734	0.53
7	0.99907	0.89899	0.51744	0.65683	0.63
8	0.99850	0.79487	0.18023	0.29384	0.21

5.2.2 Resampling Technique and Reconstruction Representation

Table 5.2: Performance of Auto-Encoder Model with Resampling Technique or using the Reconstruction Representation

Model	Accuracy	Precision	Recall	F1 Score	PR-AUC
1	0.99942	0.92884	0.72093	0.81178	0.79
1 (Resampling)	0.99986	0.99958	1.00000	0.99979	1.00
1 (Reconstruction)	0.99923	0.93182	0.59593	0.72695	0.76
5	0.99902	0.95706	0.45349	0.61538	0.62
5 (Resampling)	0.99989	0.99968	1.00000	0.99984	1.00
5 (Reconstruction)	0.99907	0.91623	0.50872	0.65421	0.66

5.2.3 Optimizer and Activation Function Tuning

Table 5.3: Performance of Auto-Encoder Model 1 with different Optimizer and Activation Function

Optimizer	Activation Function	Accuracy	Precision	Recall	F1 Score	PR-AUC
adadelta	tanh	0.99942	0.92884	0.72093	0.81178	0.79
adadelta	RELU	0.99927	0.90947	0.64244	0.75298	0.75
adadelta	ELU	0.99943	0.91697	0.73837	0.81804	0.80
adam	tanh	0.99942	0.91941	0.72965	0.81361	0.79
adam	RELU	0.99912	0.93782	0.52616	0.67411	0.70
adam	ELU	0.99950	0.92683	0.77326	0.84311	0.81

5.3 Analysis and Discussion

5.3.1 Architecture of Auto-Encoder

Overall, the training progress of the model with or without the middle bottleneck block is able to achieve the same amount of loss within the same number of epochs. Both types of models are able to converge with the same amount of loss and number of epochs as adding one layer of middle bottleneck block does not complicate the model enough to have a big effect on the training progress. For the encoded representations shown in Section 5.1, the models without middle bottleneck block is better in separating the fraud and normal instances. This is because the extra middle layer further eliminates some features which are crucial in separating the class of the instances. The adding of the middle bottleneck block causes a slight regularization effect on the model. This can be seen in those models which are not overfitting, the performance of models reduced when the middle bottleneck is present. Only in the model 3 and 4, which are over-complete auto-encoders without any regularization and tend to over fit, model 4 with bottleneck layer performed slightly better than model 3. The introduction of a middle bottleneck block with lesser nodes is able to force the model to use a reduced number of features for generating back the original instance. Thus, only those salient and important features are being learnt. However, if the available features are already not sufficient to generate back the original instance, this will remove those significant features and affect the model's performance.

For over-complete auto-encoders, which are model 1 to 4 in this case, there are more nodes in the hidden layer compared to the number of inputs, thus, more features are extracted and tends to over fit the models. To avoid overfitting the model, L1 regularization is applied. The training and validation loss at the end of training progress for model 3 and 4 without L1 regularization are lower than that of model 1 and 2. This indicates that models without regularization are able to generate results that are closer to the original instance. The visualization of encoded representation does not show clear differences in separating fraud from normal between models with and without L1 regularization. For model 1 and model 3, where both are without the middle bottleneck block, even though model 3 is able to achieve smaller loss during the training, model 1 is the one that showed better performance across all the evaluation metrics. This clearly showed that model 3 without regularization is overfitting. For model 2 and 4 where the middle bottleneck block is added, model 4 where no L1 regularization is added, showed better results in Recall, F1 score and PR-AUC, indicating this model is better in detecting fraud. This is because the bottleneck layer is able to reduce the severity of the overfitting issue in model 4 while in model 2 where there is no overfitting issue, the performance is reduced as some significant features are being eliminated.

For under-complete auto-encoders, which are model 5 to 8 in this case, there are less nodes in the hidden layer compared to the number of inputs, thus, leading to a model which only extracts lesser and significant features needed to recreate back the original instance. However, as less features are being extracted, there is a possibility of under fitting the model. The training progress of under-complete auto-encoders need more than 6 epochs to stabilize compared to the 2 epochs needed in over-complete. This is because more adjustments are needed to be done in the case of under-complete auto-encoders as the model can only use a limited number of features to generate back the original instances. The visualization of the encoded representation showed that under-complete models are not able to separate the fraud from normal as good as the over-complete models. All the under-complete models are having lower values of recall, f1 score and PR-AUC compared to model 1 to 4. These indicate that the under-complete model is under fitting and the features extracted are not sufficient for detecting fraud.

More hidden layers are added to the under-complete models to create a deeper architecture. The training progress for the deeper models does not show much differences from the normal shallow models as the number of hidden layers added are not significant enough to have a big impact on the training process. For the visualization of encoded representation, model 8 with deeper architecture and middle bottleneck block clearly showed that it failed to differentiate between normal and fraud instances. This indicates that the middle bottleneck block with only two nodes had removed most of the significant features and resulted in under-fitting. Model 7, a deeper architecture without the middle bottleneck block performed the best among the 4 under-completes models in terms of recall, f1 score and PR-AUC. This indicated that a deeper architecture can improve the fraud detection performance as it can model a more complicated function and have a better parameter efficiency (Aurélien Géron, 2019).

All the 8 auto-encoder models are not able to outperform the base RF model. This indicates that the raw input itself is good enough for fraud detection without the need of feature extraction.

5.3.2 Effect of Resampling Technique and Reconstruction Representation

The use of resampling techniques in both over-complete Model 1 and under-complete Model 5 is able to significantly improve the performance of fraud detection. As more number of fraud instances are being used in RF model training, it is way better in recognizing the features of fraud. However, the study conducted considers all types of frauds are known types and presence in the dataset. The performance of the model with resampling technique will reduce if it is used to detect new or unseen types of fraud.

In over-complete model 1, the use of reconstruction instances at the end of the auto-encoder rather than the encoded representation results in lower performance of fraud detection as shown in lower value of recall, f1 score and PR-AUC. The plot of model 1 encoded representation in Figure 5.1 and reconstruction representation in Figure 5.14 showed that encoded representation is better in separating the fraud from normal. In the under-complete model 5, although the plot of model 5 encoded representation in Figure 5.5 and reconstruction representation in Figure 5.15 showed that encoded representation is better in separating the fraud from normal, RF model using the reconstruction instance for classification is better in fraud detection. As there is no clear trend in the performance of fraud detection when the reconstruction instances are used and most of the original or significant information can be lost, the encoded representation is clearly a better choice. Moreover, the values of recall, f1 score and PR-AUC of model 1 using encoded representation are much higher than that of model 5 using reconstruction representation.

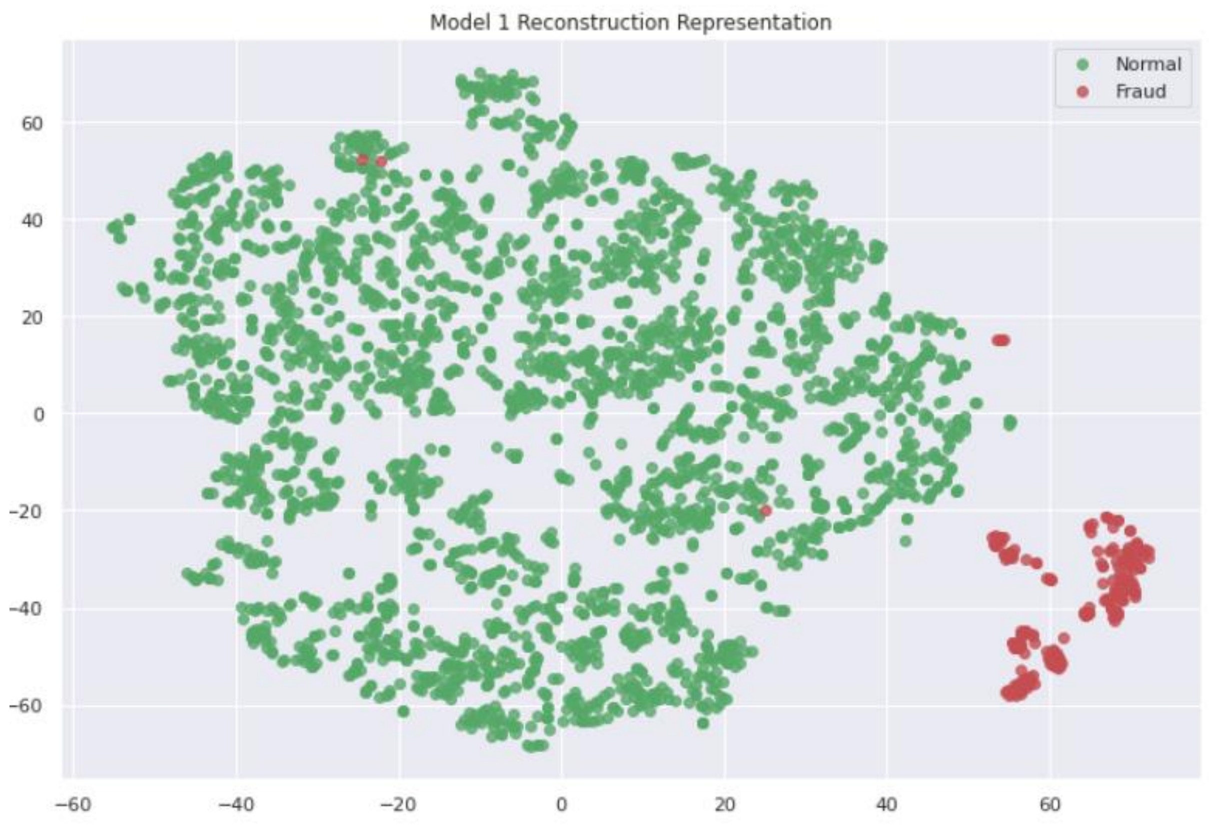


Figure 5.14: The reconstruction representaion of instances using Model 1

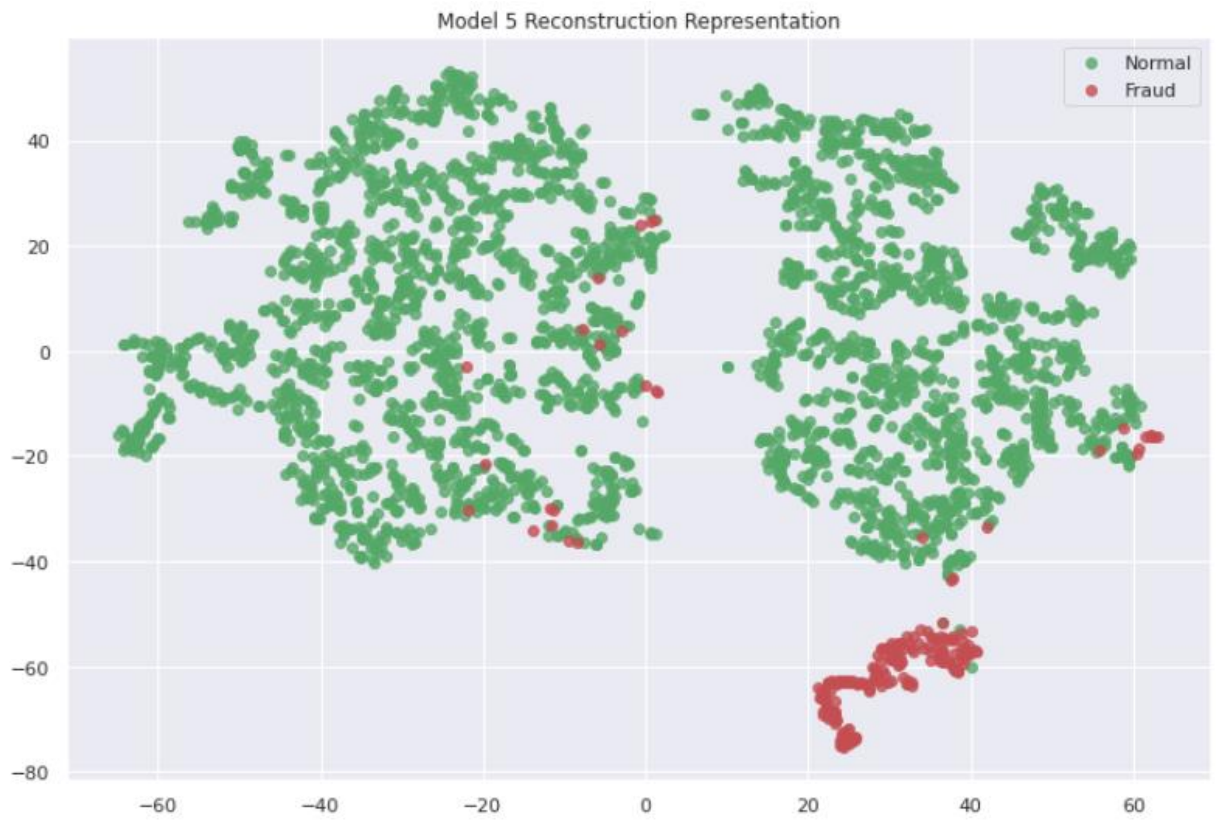


Figure 5.15: The reconstruction representaion of instances using Model 5

5.3.3 Effect of Different Optimizers and Activation Functions

The training progress by using adam as optimizer is converging within a lesser number of epochs and achieves smaller loss at the end of the training compared to adadelta. This is because adadelta is applying the adaptive learning rate without the momentum concept while adam added the momentum concept on top of the adaptive learning rate, which improves the converging process (Gunand Mayanglambam, 2020).

Among the three activation functions, ELU performed the best in fraud detection which can be seen in the values of recall, f1 score and PR-AUC. The best combination for the auto-encoder in this study is the adam as optimizer and ELU as the activation function.

SECTION 6

CONCLUSION AND RECOMMENDATION

The middle bottleneck block in the auto-encoder forced the model to learn a smaller number of important features. However, it can result in under fitting if too many features are being eliminated. Creating an overfitting model, in this case is the over-complete auto-encoder followed by regularization is an effective and simple way to control the model's performance. Under-complete auto-encoder is best to be applied when a smaller number of features compared to the original input is sufficient for classification purpose. Else, it can result in under fitting the model. A deeper architecture of hidden layers can improve the performance of auto-encoders by doing more features extraction and it is also more effective. Resampling technique is able to improve the model's performance while the usage of encoded representation rather than the reconstruction representation for second stage model training is a better choice. In this study, the combination of adam and ELU is the best for fraud detection.

In future, different dataset can be used to test on the robustness of the auto-encoder for anomaly detection. Furthermore, the application of GAN for anomaly detection can also be studied in the future.

REFERENCES

Abakarim, Y., Lahby, M. and Attioui, A. (2018) ‘An efficient real time model for credit card fraud detection based on deep learning’, in *ACM International Conference Proceeding Series*. New York, NY, USA: Association for Computing Machinery, pp. 1–7. doi: 10.1145/3289402.3289530.

Aurélien Géron (2019) *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems SECOND EDITION*. 2nd edn.

Branco, B. *et al.* (2020) ‘Interleaved Sequence RNNs for Fraud Detection’, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, pp. 3101–3109. doi: 10.1145/3394486.3403361.

Chang, C. H. (2020) ‘Managing Credit Card Fraud Risk by Autoencoders : (ICPAI2020)’, in *Proceedings - 2020 International Conference on Pervasive Artificial Intelligence, ICPAI 2020*. Institute of Electrical and Electronics Engineers Inc., pp. 118–122. doi: 10.1109/ICPAI51961.2020.00029.

Chouiekh, A. and El Haj, E. H. I. (2018) ‘ConvNets for fraud detection analysis’, in *Procedia Computer Science*. Elsevier B.V., pp. 133–138. doi: 10.1016/j.procs.2018.01.107.

Credit Card Fraud Detection | Kaggle (2018). Available at: https://www.kaggle.com/mlg-ulb/creditcardfraud (Accessed: 25 March 2021).

Dawoud, A., Shahristani, S. and Raun, C. (2019) ‘Deep learning for network anomalies detection’, in *Proceedings - International Conference on Machine Learning and Data Engineering, iCMLDE 2018*. Institute of Electrical and Electronics Engineers Inc., pp. 117–120. doi: 10.1109/iCMLDE.2018.00035.

Forough, J. and Momtazi, S. (2021) ‘Ensemble of deep sequential models for credit card fraud detection’, *Applied Soft Computing*. Elsevier Ltd, 99, p. 106883. doi: 10.1016/j.asoc.2020.106883.

Gunand Mayanglambam (2020) *Deep Learning Optimizers. SGD with momentum, Adagrad, Adadelta… | by Gunand Mayanglambam | Towards Data Science*. Available at: https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f (Accessed: 28 April 2021).

Jurgovsky, J. *et al.* (2018) 'Sequence classification for credit-card fraud detection', *Expert Systems with Applications*. Elsevier Ltd, 100, pp. 234–245. doi: 10.1016/j.eswa.2018.01.037.

Kim, E. *et al.* (2019) 'Champion-challenger analysis for credit card fraud detection: Hybrid ensemble and deep learning', *Expert Systems with Applications*. Elsevier Ltd, 128, pp. 214–224. doi: 10.1016/j.eswa.2019.03.042.

Krutarth Darji (2020) *Simple Credit card Fraud Detection 95% Accuracy | Kaggle*. Available at: https://www.kaggle.com/krutarthhd/simple-credit-card-fraud-detection-95-accuracy (Accessed: 24 March 2021).

Makki, S. *et al.* (2019) 'An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection', *IEEE Access*, 7, pp. 93010–93022. doi: 10.1109/ACCESS.2019.2927266.

Misra, S. *et al.* (2020) 'An Autoencoder Based Model for Detecting Fraudulent Credit Card Transaction', in *Procedia Computer Science*. Elsevier B.V., pp. 254–262. doi: 10.1016/j.procs.2020.03.219.

Nadim, A. H. *et al.* (2019) 'Analysis of machine learning techniques for credit card fraud detection', *Proceedings - International Conference on Machine Learning and Data Engineering, iCMLDE 2019*, pp. 42–47. doi: 10.1109/iCMLDE49015.2019.00019.

Nilson Report | News and Statistics for Card and Mobile Payment Executives (2021). Available at: https://nilsonreport.com/ (Accessed: 24 March 2021).

Ozbayoglu, A. M., Gudelek, M. U. and Sezer, O. B. (2020) 'Deep learning for financial applications: A survey', *Applied Soft Computing Journal*. Elsevier Ltd, p. 106384. doi: 10.1016/j.asoc.2020.106384.

Pang, G. *et al.* (2021) 'Deep Learning for Anomaly Detection', *ACM Computing Surveys*. ACM PUB27 New York, NY, USA , 54(2), pp. 1–38. doi: 10.1145/3439950.

Robin Teuwens (2021) *Anomaly Detection with Auto-Encoders | Kaggle*. Available at: https://www.kaggle.com/robinteuwens/anomaly-detection-with-auto-encoders (Accessed: 24 March 2021).

Shivam Bansal (2019) *Semi Supervised Classification using AutoEncoders | Kaggle*. Available at: https://www.kaggle.com/shivamb/semi-supervised-classification-using-autoencoders (Accessed: 24 March 2021).

Yu, X. *et al.* (2020) 'A Deep Neural Network Algorithm for Detecting Credit Card Fraud', in *Proceedings - 2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering, ICBAIE 2020*. Institute of Electrical and Electronics Engineers Inc., pp. 181–183. doi: 10.1109/ICBAIE49996.2020.00045.

Zhan, F. (2020) 'Research on bank fraud transaction detection based on LSTM-Focalloss', in *2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence*. New York, NY, USA: ACM, pp. 1–6. doi: 10.1145/3446132.3446176.

Zhang, X. *et al.* (2019) 'HOBA: A novel feature engineering methodology for credit card fraud detection with a deep learning architecture', *Information Sciences*. Elsevier Inc., 557, pp. 302–316. doi: 10.1016/j.ins.2019.05.023.