

IMPROVEMENT OF FEATURES EXTRACTION
OF TIME SERIES DATASET BY USING
AUTOENCODER

by

Narongchai Limpiyapirom

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Computer Science).

Golden, Colorado

Date _____

Signed: _____
Narongchai Limpiyapirom

Signed: _____
Dr. Hua Wang
Thesis Advisor

Golden, Colorado

Date _____

Signed: _____
Dr. Tracy Camp
Professor and Head
Department of Computer Science

ABSTRACT

In this research, we studied how to extract features vector from the Multivariate Time Series dataset (MTS) by using various types of Autoencoder. Do the classification method from these encoded vectors to show that the compressed data from Autoencoder still keeping enough essential features, although the size was reduced. The dataset that we used is blood measurements collected from patients after surgery; the result after measurement will represent that patients have surgical site infections or not. This dataset contains many missing data, so not just only deal with the compressing method, we are also facing with sparse data problem so that the imputation method will be performed to recover the data back. To sum up, our research will show the capability to compress features vector from sparse multivariate time series datasets while keeping enough information.

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF FIGURES	vi
LIST OF TABLES	ix
LIST OF ABBREVIATIONS	x
ACKNOWLEDGMENTS	xi
CHAPTER 1 INTRODUCTION	1
1.1 Backgrounds and Motivations	1
1.2 Problem statement	8
1.3 Objectives of the research	9
1.4 Project process	10
CHAPTER 2 DATA PREPROCESSING	11
2.1 Dataset	12
2.2 Normalization (Z-normalization)	14
2.3 Imputation (Generative Adversarial Imputation Nets)	16
2.3.1 Generator	17
2.3.2 Discriminator	17
CHAPTER 3 FEATURES EXTRACTION METHODS	19
3.1 Autoencoder	20
3.1.1 Encoder	21
3.1.2 Decoder	22

3.1.3	Loss function	22
3.1.4	Experiment of reconstruction output	23
3.2	Split Autoencoder	25
3.2.1	Experiment of reconstruction output	26
3.3	Long Short Term Memory (LSTM) Autoencoder	28
3.3.1	Concept of Long Short Term Memory (LSTM)	28
3.3.2	Long Short Term Memory Autoencoder	30
3.3.3	Experiment of reconstruction output	33
3.4	Alternation of LSTM and Split Autoencoder	35
3.4.1	Experiment of reconstruction output	36
3.5	Deep Kernelized Autoencoder	38
3.5.1	Time Series Cluster Kernel	38
3.5.2	Deep Kernelized Autoencoder	38
3.5.3	Experiment of reconstruction output	40
CHAPTER 4 EXPERIMENT		41
4.1	Autoencoder	44
4.2	Split Autoencoder	45
4.3	LSTM Autoencoder	46
4.4	Split LSTM Autoencoder	47
4.5	Deep Kernelized Autoencoder	48
CHAPTER 5 CONCLUSION		49
CHAPTER 6 FUTURE WORK		50
REFERENCES CITED		51

LIST OF FIGURES

Figure 1.1	The sample image of offshore plant	1
Figure 1.2	The sample structure of onshore plant over view	2
Figure 1.3	The sample image of onshore main production process	3
Figure 1.4	The sample image of remote location	4
Figure 1.5	The sample image of measurement devices (Level, Flow, Temperature and Pressure)	5
Figure 1.6	The sample image of Programmable Logic Controller (PLC)	6
Figure 1.7	The sample graph of well performance	6
Figure 1.8	The sample graph of production rate vs time	7
Figure 1.9	Project process	10
Figure 2.1	The picture illustrated that the process of blood measurement	12
Figure 2.2	The picture illustrated the similarity between real dataset and blood sample dataset	13
Figure 2.3	The original input before normalization	14
Figure 2.4	The output after normalization	15
Figure 2.5	The architecture of GAIN that used in this project	16
Figure 2.6	The original input after normalization and imputation	18
Figure 3.1	The basic structure of Autoencoder	21
Figure 3.2	The original sample image	23
Figure 3.3	The reconstruction of sample image from Autoencoder	23
Figure 3.4	The original input data	24

Figure 3.5	The reconstruction of input data from Autoencoder	24
Figure 3.6	The structure of split Autoencoder	25
Figure 3.7	The original sample image	26
Figure 3.8	The reconstruction of sample image from Individual Autoencoder	26
Figure 3.9	The original input data	27
Figure 3.10	The reconstruction of input data from Individual Autoencoder	27
Figure 3.11	The structure of Long Short Term Memory cell (LSTM)	28
Figure 3.12	The basic structure of Long Short Term Memory Autoencoder	30
Figure 3.13	The setting of Long Short Term Memory layer which return_sequence = True	31
Figure 3.14	The setting of Long Short Term Memory layer which return_sequence = False, and using RepeatVector layer	32
Figure 3.15	The original sample image	33
Figure 3.16	The reconstruction of sample image from LSTM Autoencoder	33
Figure 3.17	The original input data	34
Figure 3.18	The reconstruction of input data from LSTM Autoencoder	34
Figure 3.19	The basic structure of Split Long Short Term Memory Autoencoder	35
Figure 3.20	The original sample image	36
Figure 3.21	The reconstruction of sample image from Split LSTM Autoencoder	36
Figure 3.22	The original sample image	37
Figure 3.23	The reconstruction of sample image from Split LSTM Autoencoder	37
Figure 3.24	The structure of TCK that used in this project	38
Figure 3.25	The structure of Deep Kernelized Autoencoder in this project	39
Figure 3.26	The original input dataset	40

Figure 3.27	The reconstruction of input dataset from Deep Kernelized Autoencoder	40
Figure 4.1	The input dataset after applied Normalization and Imputation	41
Figure 4.2	The comparison between reconstruction (left) and compressed (right) of Autoencoder	44
Figure 4.3	The comparison between reconstruction (left) and compressed (right) of Split Autoencoder	45
Figure 4.4	The comparison between reconstruction (left) and compressed (right) of LSTM Autoencoder	46
Figure 4.5	The comparison between reconstruction (left) and compressed (right) of Split LSTM Autoencoder	47
Figure 4.6	The comparison between reconstruction (left) and compressed (right) of Deep Kernelized Autoencoder	48

LIST OF TABLES

- Table 4.1 A table of size of features vector type for each type of Autoencoder 42
- Table 4.2 A table of accuracy of features vector type for each type of Autoencoder . . 43

LIST OF ABBREVIATIONS

Programmable Logic Controller	PLC
Multivariate Time Series	MTS
Generative Adversarial Imputation Nets	GAIN
Long Short Term Memory	LSTM
Time Series Cluster Kernel	TCK
Gaussian Mixture Models	GMM

ACKNOWLEDGMENTS

I would like to thank the academy for granting me this prestigious thesis. This project would never have succeeded without Dr.Hua Wang, Hoon Seo and Saad Elbeleidy.

CHAPTER 1

INTRODUCTION

1.1 Backgrounds and Motivations

Nowadays, data become more critical in many industries, especially in the oil and gas business. A massive number of data were collected every day to support the operation and maintenance team. Our team took responsibility for designing and modifying the control systems for process facilities in the oil and gas field. To maintain reliability and make sure that we can produce oil enough for daily usage in Thailand. In Thailand, we have both offshore and onshore plants.



Figure 1.1: The sample image of offshore plant

Regardless of whether it is offshore or onshore production, there is not much difference in the process. The production area will be divided into two parts, the main production process and the remote location. We split the production area into two parts because the total amount of oil produced will not come from a single large well. Nevertheless, it comes from the combination of many small production wells; these small-production wells will not be drilled in the same area but will spread out over a wide area. The unique characteristics

of production wells in Thailand are relatively small production wells that are different from others. Moreover, it spread far enough apart. For example, an onshore production area in Thailand covers three adjacent provinces.

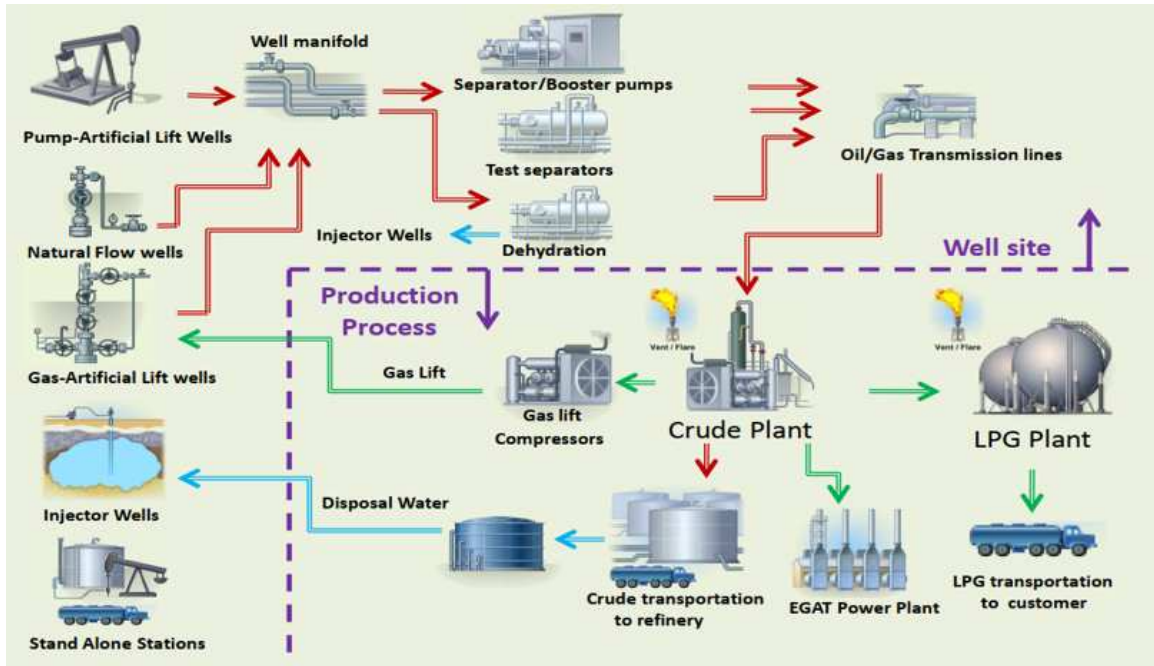


Figure 1.2: The sample structure of onshore plant over view

For the main production process, This part is to receive oil or gas from all remote locations and pass through the separating and conditioning process to obtain quality products that meet the customers' needs. In this part, there will be much major equipment, different from the remote location, which has only the necessary equipment for bringing oil from the sub-surface area. The primary equipment of the main production process consists of the main separator, which is responsible for separating oil, water and gas, and pass them on to various processes according to the form of the substance. The heat exchanger is responsible for reducing the products' temperature by exchanging the temperature between cold liquid and high-temperature products, helping reduce the temperature in the production process. The gas compressor is responsible for compressing and adding enough gas pressure before sending back for partial use in some remote locations. The tanks are used to store the oil,

which has been processed and ready to be delivered to customers. In addition to the main equipment, the primary control system is also part of the main production process. The primary control system is responsible for processing and sending commands to control the operation of various final control devices, not only in the main production process but also in the remote locations via our private network.



Figure 1.3: The sample image of onshore main production process

For the remote location, or sometimes we call it Well Site, it is responsible for bringing oil from the production wells. The complexity of the equipment in this section depends mainly on the type of tools, which is used for bringing oil to the surface area and for the emergency systems. If the production well has just been drilled and there is high pressure in the hole, we will not need to use many tools to bring the oil up. It can flow up using their natural pressure. However, if the well has been produced for a long time and there is not enough pressure inside. We need to install additional tools such as pumps to lift the oil out of the production well. The produced oil from every production wells in each remote location will be sent through the pipes and combined at the manifold and then sent through the main pipeline to the main production process. The information on the well's condition and flow rates will be collected at the sub-control system according to each remote location.



Figure 1.4: The sample image of remote location

We have four types of measurement values, which are typically used in oil and gas fields. The first one is the pressure. Because this industry has relatively high operating pressure, pressure measurement helps in both controls the working conditions and prevents or alerts in the emergency case. The second is the flow. This value needs to be measured to check how much oil is produced in each well. Furthermore, it will be used for calculating a summary of the daily production rate. We have also applied the flow measurement to help alert when the devices have a problem, such as when ordering the pump to work, but it appears that there is no flow, then we will send the team to inspect the pump's condition. The third one is temperature. The temperature measurement will help us to monitor the information or the condition of the wells. Moreover, it can also alert in case of rotating equipment working over the temperature limit. The last one is level. We will have a level measurement at the storage or vessel used for monitoring the total amount of oil. Moreover, it can alert when there is liquid coming more than tank capacity. Most of the data mainly comes from these four measurement devices, but there may be some additional information that we want to measure more as well.



Figure 1.5: The sample image of measurement devices (Level, Flow, Temperature and Pressure)

The primary controller used in remote locations is the programmable logic controller or PLC. We choose PLC because the PLC has high stability and can tolerate working in severe environments such as high temperatures or continuous working without system breaks. Although PLC can work in severe environments, it comes with a low specification limit such as low-speed CPU or low memories. The PLC's primary functions are receiving data directly from the measurement devices, sending the output signal to control the final control devices, receiving commands or sending data logs between the remote location and main production process. PLC programs are not as complicated as other computer programming languages. It uses simple commands in the form of a ladder diagram, which leads to applying some new techniques such as using machine learning models to run on PLC still limited. So if we would like to make calculations that are quite complex, we need to send data back to run in the primary control system instead.

The number of signals measured at each remote location will vary according to the number of production wells and additional equipment of that location. However, the number of signals is approximately 100 - 200 signals per location, after combined with the number of all remote locations, resulting in more than many thousand signals being sent back to the primary control system every day. Moreover, all signals are collected every second, so the total amount of data collected each day is enormous. Nevertheless, this large amount of information is significant and necessary to operate all production wells. Mainly, this information will be sent to use in many teams.



Figure 1.6: The sample image of Programmable Logic Controller (PLC)

To optimization the operation plan. These data are used in planning, which production wells we need to open or close to maintain the condition of the reservoir. We can not produce oil and gas from a particular zone for a long time because sometimes the reservoir may not be strong enough and cause the ground to collapse. Sometimes it may be necessary to inject water back into the wells that have already been produced. The information in this section is required to use so fast because opening and closing production wells will be a task that the operator team has to do every day.

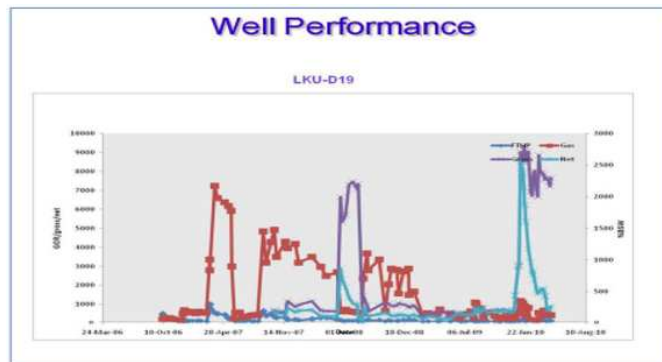


Figure 1.7: The sample graph of well performance

To predict the remaining oil. Using this information will improve the accuracy of future investment, making the decisions from the data collected. These data will help us know that each well will have a reserve or how long it can be produced. Furthermore, is it worth the investment or not? If not, we will plan to accelerate the production of this well and then reduce other parts instead. Then, ordered to shut down the operation.

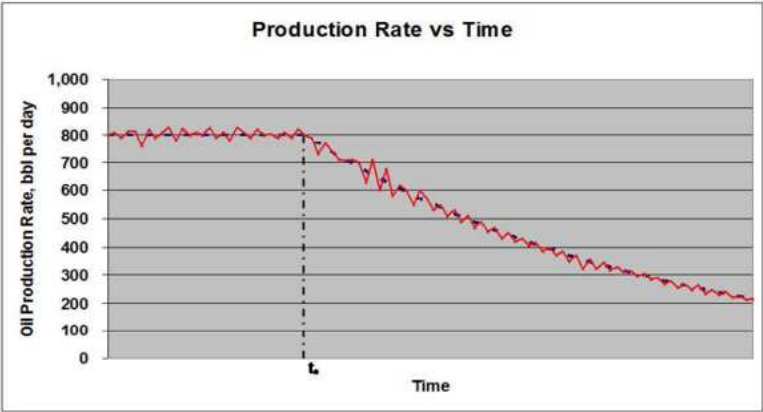


Figure 1.8: The sample graph of production rate vs time

To use in predictive maintenance, in addition to the operation team, another critical team is the maintenance team. The maintenance of the equipment to be fully operational and prolonging the use of time is another way to reduce the cost. The important part of using data in the maintenance team is to use it in the predictive maintenance process. Predictive maintenance is the use of data from the actual operating status of real-time collected devices for analysis, such as temperature and gas pressure. It can alert us before an abnormality occurs and assess when the machine or equipment needs repair, also allowing us to plan to swap other devices to work instead and then repair that equipment before it breaks down.

1.2 Problem statement

Based on the information mentioned in the previous section, the large amount of data is collected each day and is likely to increase as the number of production wells increases. These data are used in many essential processes such as operation, planning and maintenance. However, when the kept data is increasing every day, the amount of data that can be stored will be limited. We must delete the old data and overwrite it with a new one instead. On average, the data is kept not more than 5-10 years, depending on the type of data. If we use the data for the general proposes, the amount of data stored backward is still sufficient for users' needs because the technology that we used in current data analysis does not require much historical data. Nevertheless, in the future, if we want to change to use Machine Learning models, especially Deep Learning, the amount of data that will be used to train model is crucial[1]. Therefore, we have to find a new way to store the data, which can reduce the size of the data while still maintaining the important features. That will allow us to store data for longer and still be able to use it to train the machine learning model as we want. Aside from data size issues, we encounter the missing data problem. As all measurement and communication devices are still electrical devices, in the event of a power failure, the information in that period will disappear immediately, resulting in an incomplete dataset collected. Even if the backup power has been installed, the backup power in the remote location can sometimes not maintain the system until the problem will be solved. Therefore, the imputation process for recovering the lost data is another significant thing

1.3 Objectives of the research

From the two problems mentioned in the previous section, 1. find a new way to store the data that can reduce the size of the data while still maintaining the important features 2. find an imputation method for recovering the missing data. In this research, we aim to find solutions to both problems. Begin with focusing on the problems related to missing data. The missing data will affect the process used to compress the data. If the source data is incomplete, it will result in incorrect compression. After that, when the data is better, we will start to find ways to compress the data. Data compression is the main part of this research. We will focus on using Autoencoder in various ways to compare the results of which method is most suitable. The goal of this research is to

- apply imputation method (Generative Adversarial Imputation Nets [2]) to deal with missing data
- modify traditional Autoencoder [3] [4], LSTM [5] Autoencoder and Deep Kernelized Autoencoder [6] to compress features vector from pre-processed data
- Develop split Autoencoder to extract the unique features from each sub dataset
- Develop alternating methods between split Autoencoder and other methods

1.4 Project process

This project consists of 3 main part as shown in Figure 1.9

- Pre-processing: Since the original input dataset is incomplete, it contains attributes with different ranges and missing data issues. So we need to do the data preprocessing first to make sure that the information will be ready for the next step
- Features Extraction: After got the preprocessed data, We will train various types of Autoencoder and use the encoder part to compress a feature vector by keeping the concept to reduce the size of stored data but still keep the essential features necessary for the other machine learning model.
- Classification: Apply the traditional classification method to classify the result from each feature vector. This part is for proving that the compressed data is good enough for use in the future or not.

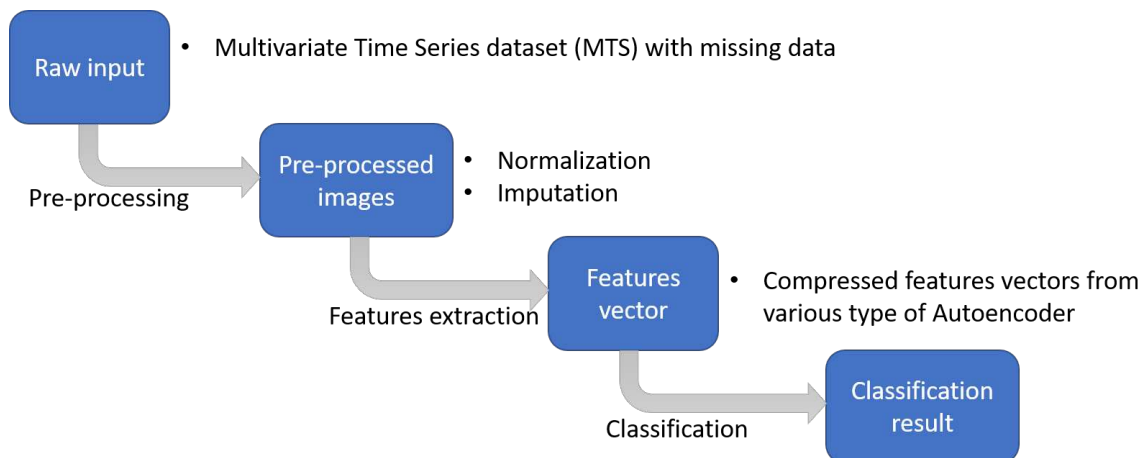


Figure 1.9: Project process

CHAPTER 2

DATA PREPROCESSING

Data preprocessing is an essential step for preparing data and make it ready for further steps. In general, the data which is collected from the real world is often incomplete and prone to many errors. Since the trend of using data has just happened in the past few years, it is leading to most of the historical data that is collected by the company is incomplete. Such as database is not good enough to store the data because it may not be planned in the future that requires large amounts of data. Or data is not collected complete as needed because, in those days, there may not be a data scientist to tell which data needs to be collected or not to be collected. Therefore, when the information that has been collected may not be very ready, we need to bring those data to use for training machine learning models. Data preprocessing is a way to help solve problems at this point. There are many techniques in data preprocessing, such as Data Cleansing, which are used to extract the data we do not need from our dataset so that the data in that section will not interfere with other calculations. Data Reduction will be used to reduce the complexity of the data before processing. The data preprocessing techniques that we used in this research are Normalization and Imputation. Because our data is stored in various variables, not on the same scale, we have to use Normalization to scaling it and, after that, followed by Imputation to manage the recovery of the missing data.

2.1 Dataset

Because the actual data from the wellhead is confidential and cannot present publicly, our team tries to find another dataset that has similar properties in terms of multivariate time series (MTS) and missing data. In this research, the dataset that we use is a blood measurement dataset [6] collected from patients undergoing surgery at the University Hospital of North Norway in the years 2004 - 2012. Each patient's data is blood measurements of 10 variables, which are alanine aminotransferase, albumin and alkaline phosphatase, creatinine, CRP, hemoglobin, leukocytes, potassium, sodium and thrombocytes. Moreover, the blood samples will be collected within 20 days after surgery. The measurement results will classify patients into two groups, which have surgical site infections and have not. Dataset labels are assigned according to International Classification of Diseases and NOMESCO Classification of Surgical Procedures, relative to patients with severe postoperative complications. The missing data is caused by the patient not receiving a blood test at the observation period of the day. Moreover, if some patients with the number of tests less than 2, they will not be counted in this dataset. Totally, the number of patients in this dataset is 882, and 232 are patients with infections.



Figure 2.1: The picture illustrated that the process of blood measurement

The reason that we chose the blood measurement dataset to replace data from the real data is the characteristics of both datasets are very close, both in terms of multivariate time

series (MTS) and missing data. For the dataset from the actual wellheads, We will assume that the commands sent to devices such as controlling valve or controlling pump represent the data in a Row. To send commands to start a pump or open a valve, the controller needs input data from many measurement devices and calculates them to check that we should turn that device on or not. We will think about each measurement device as an attribute. Because the calculation process will not use the data for just a one-time step, it will process data backward over multiple time intervals. Hence, the data from the actual wellheads has the features of being a multivariate time series (MTS). From the problem of missing data mentioned in Section 1.2, It also shows that there are features of missing data as well. Compared with the blood measurement data set, one patient is comparable to sending an output command to control one device. The measurement values from blood samples are similar to the values from the measurement devices. The 20-day blood test period is the same as the time steps that the controller used to calculate the output commands. Finally, the results of the tests, if they were infected, is the same as the output from the controller, whether the valve is opened or not. Looking at the similarities between the two sets of data, we decided that the blood measurement dataset could be used for the features extraction method instead of the actual dataset.

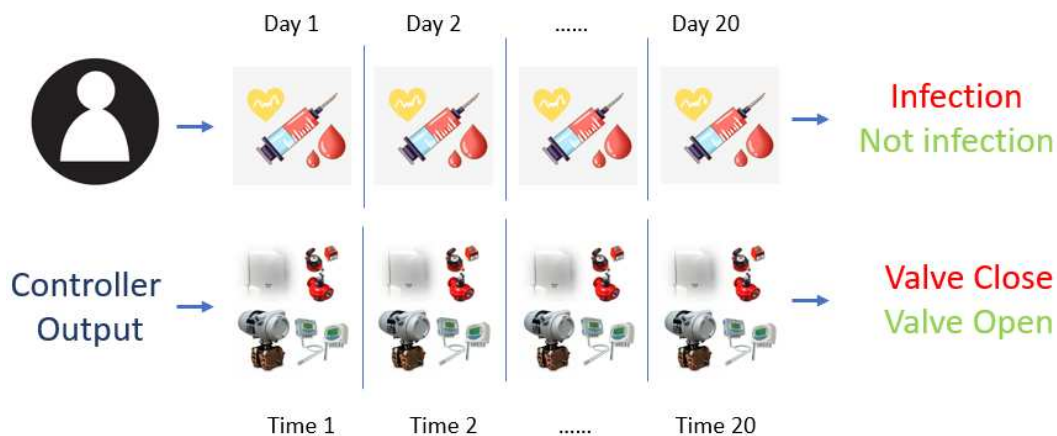


Figure 2.2: The picture illustrated the similarity between real dataset and blood sample dataset

2.2 Normalization (Z-normalization)

The normalization method that we used is Z-normalization or standardization. All elements of the input vector are transformed into the output vector whose mean is approximately 0, while the standard deviation is in a range close to 1. As shown in formula 2.1.

$$\hat{X}[:, i] = \frac{X[:, i] - \mu_i}{\sigma_i} \quad (2.1)$$

$$\mu_i = \frac{1}{N} * \sum_{k=1}^N X[k, i], \sigma_i = \sqrt{\frac{1}{N-1} * \sum_{k=1}^N (X[k, i] - \mu_i)^2} \quad (2.2)$$

Assume that we have a dataset X , which has N rows(entries) and D columns(features). $X[:, i]$ represent feature i and $X[j, :]$ represent entry j .

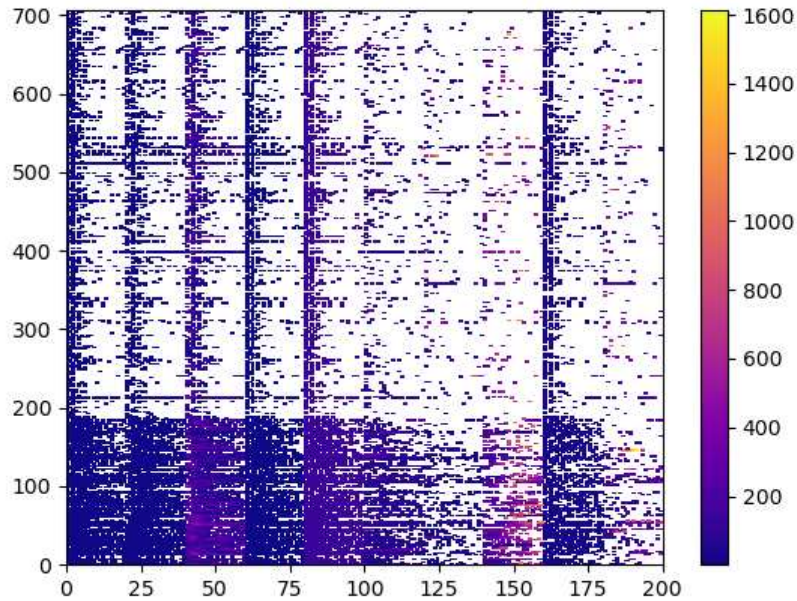


Figure 2.3: The original input before normalization

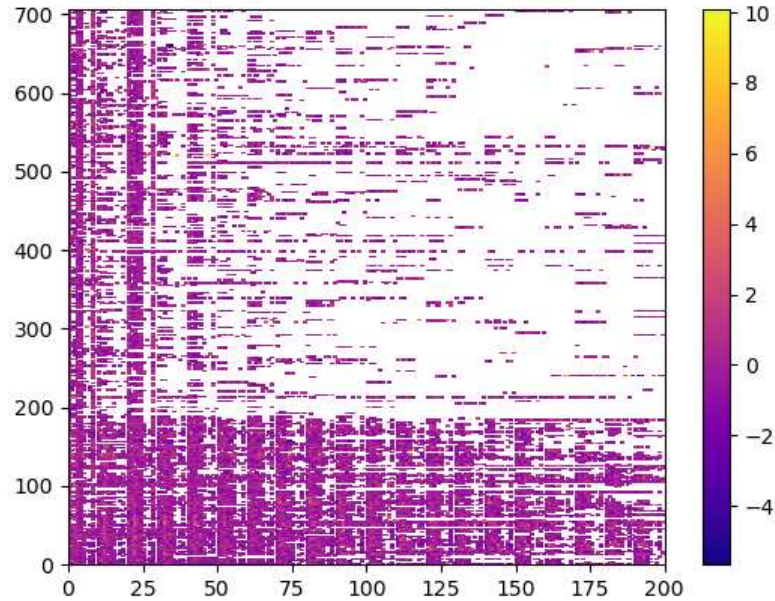


Figure 2.4: The output after normalization

The Z-normalization method is used only when we need to analyze any dataset that consists of several types of data with different ranges. If we use these data to train the model directly, it will lead to inaccuracies. Therefore, the Z-normalization method should be applied to adjust the data to the same scale first.

The Figure 2.3 and Figure 2.4 show the values in the blood measurement dataset. The y-axis represents 707 patients in the training set. The x-axis represents all ten measurement values over 20 days. The y-axis positions 0 - 10 will indicate the measurement values measured on the first day, 11-20 would indicate the second day, and so on until 191 - 200 would mean the last day. The color in the graph shows the value stored in that location. From the comparison between before and after doing z-normalization, we found that before doing z-normalization, the range of the data is very wide, with values from 0 - 1600 by the way most values will not exceed 400 approximately. After doing z-normalization, the range of data is reduced to around -6 - 10, and most of data are still grouped in the same range.

2.3 Imputation (Generative Adversarial Imputation Nets)

The blood samples dataset contained missing data around 85.72% of the total data points. Considering that we have less than 15% of the actual data, If we use only this amount of data to train machine learning models, the result may be inaccurate. Therefore, we need to use the imputation method to estimate some data back. From the latest research that our team has done, topic about the imputation of mineral distribution of sparse observation [7]. That dataset contains a much higher missing value than this work. We tried many methods and found that Generative Adversarial Imputation Nets [2] or GAIN gave us better results than other methods. Therefore, in this research, we chose to use GAIN again for this dataset.

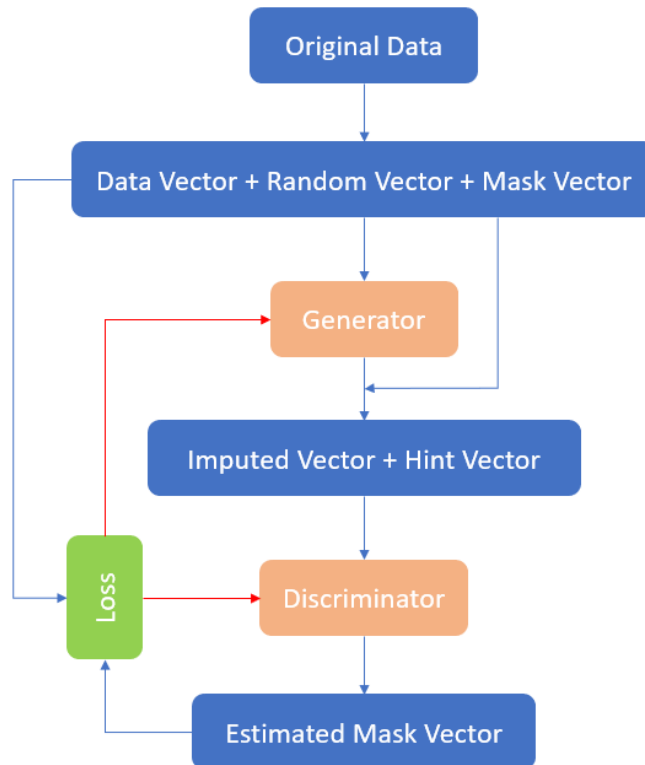


Figure 2.5: The architecture of GAIN that used in this project

GAIN is an imputation model based on two neural networks, Generator and Discriminator. The generator tries to generate or imputes the missing data based on what is observed from the original data vector and then produced the imputed vector as output. The discrim-

inator takes an imputed vector from the generator with an additional hint vector and then tries to determine which components were observed from the original data vector and which were imputed.

2.3.1 Generator

The generator will take Data vector X , Mask vector M and Noise vector Z , which are generated from the original input as an input of generator. Data vector X contained components from the original input that are observed and leave other unobserved components as zero. Mask vector M is the vector that contained value 1 in the same position as the observed position and value 0 for an unobserved position. Noise vector Z is generated as random noise. The generator will process by filling the unobserved point by using the values from the noise vector, As shown in formula 2.3. In this step, Mask vector M will tell the generator which location is observed or unobserved.

$$g : (M \odot X + (1 - M) \odot Z) \times M \rightarrow G \in R^n \quad (2.3)$$

2.3.2 Discriminator

The difference between normal GAN and GAIN is, discriminator in GAN will identify that the total output from the generator is real or fake. However, the discriminator in GAIN will identify which components are real (observed) or fake (imputed). The discriminator will take output vector from generator G and Hint vector H as inputs. Then G will be combined with original input X as formula 2.4. After that, the discriminator will concatenate the result with the hint vector H , and return the probability vector D as the result of the discriminator. The discriminator function is shown in formula 2.5.

$$M \odot X + (1 - M) \odot G \tag{2.4}$$

$$d : (M \odot X + (1 - M) \odot G) \times H \rightarrow D \in [0, 1]^n \tag{2.5}$$

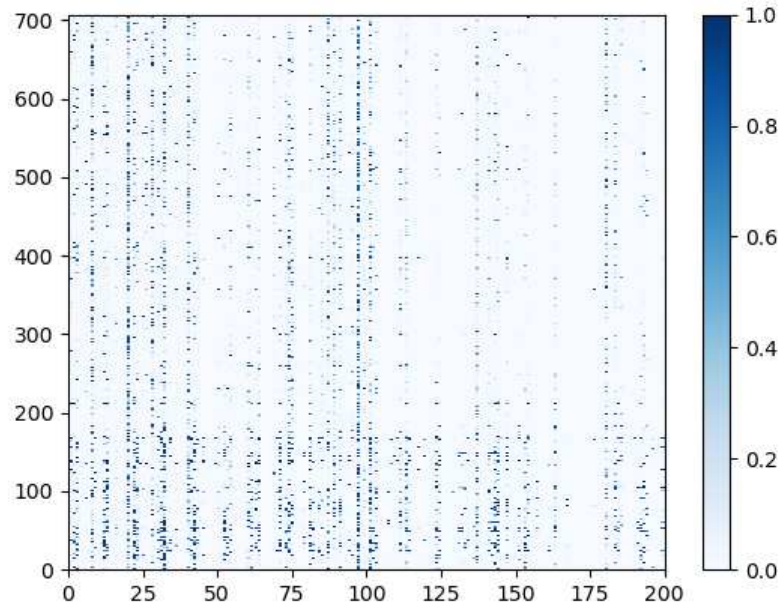


Figure 2.6: The original input after normalization and imputation

the Figure 2.6 represents the values in the blood measurement dataset after complete with z-normalization and GAIN. If we look at the color bar on the right, the values will be adjusted to the range 0 - 1. Although looking at the graph may not look obvious, the results after imputation helped to recover the data back about 15%, now we have about 30% of the data. Although it does not look like a lot, this information is quite ready for the next step. Another interesting point after doing imputation is if comparing Figure 2.3 with Figure 2.4 we can find that the dark dot line on the right of graph does not appear to stand out much, but after passing GAIN, the result seems that the line appears again.

CHAPTER 3

FEATURES EXTRACTION METHODS

In this research, we have applied five features extraction methods to obtain keeping good results of infection prediction from encoded features vectors. The main idea is that we tried to reduce the size of stored data but still keep the essential features necessary for the training process of other machine learning models for any prediction in the future. The reason that we choose an Autoencoder for the compression method instead of other models because Autoencoder has the ability to do the compression and also contains the decoder part for reconstruction method. The reconstruction part is important in terms of proving that the encoded vector contains important features. If the encoded vector can not reconstruct back to the original form, it means that some features are lost. And another reason is Autoencoder is widely made an improvement by many researchers so we can found many new techniques every year. The list below is the methods that we applied in this research.

- Traditional Autoencoder[3] [4]
- Split Autoencoder
- Long Short Term Memory (LSTM) Autoencoder [5]
- Alternation of LSTM and Split Autoencoder
- Deep Kernelized Autoencoder [6]

As mention in the topic name. Almost all methods that we applied were various types of Autoencoder and combined with another popular imputation method GAIN.

3.1 Autoencoder

Before talking about the concept of Autoencoder, we would like to talk about the origin of why we chose Autoencoder as the primary technique for data compression in this research. Back to the previous project that our team had worked. Our team worked on the project "Imputation of Mineral Distribution of Sparse Observation" which applied many imputation methods, including an Autoencoder. The type of Autoencoder that we tried at that time was the Autoencoder for the imputation method[8]. We added some noise to input with missing data. The encoder part compressed it into a smaller dimension, and then the decoder part reconstructed it back to the original size. We continue running this loop until the error is less than some points, then we considered that the imputation method was done. That project gave us the idea that if the data in the bottleneck layer can be reconstructed back to the original data, it's mean the encoded data must collect enough important features. That is why we decided to experiment with this research with various types of Autoencoders for data compression.

Autoencoder is a neural network model that learns to encode the original input to another smaller dimension and then reconstruct it back to be the same as an input. As shown in Figure 3.1, after input image passing through an encoder part of Autoencoder, the dimension will be reduced by each layer until the bottleneck layer. We will get the encoded representation or features vector. Then, passing through a decoder part, each layer's dimension will be increased to the same size as an input shape. Autoencoder will try to minimize the difference between input and output; that is why we will get a similar output as input. Autoencoder is widely use in many applications such as Anomaly Detection, Super-resolution or Image repair.

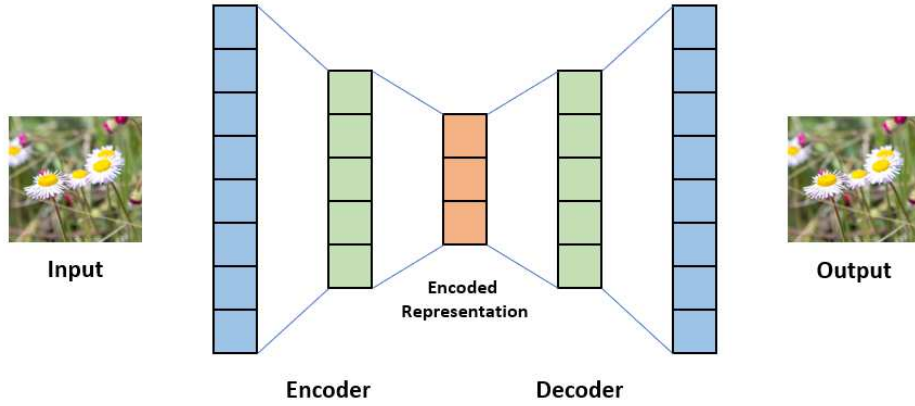


Figure 3.1: The basic structure of Autoencoder

Looking to the formula behind the Autoencoder, encoder function ϕ will maps the input data X to the features vector F in the bottleneck layer. Then decoder function ψ maps feature vector F to the output. The output of ψ also is X same as input data.

$$\begin{aligned}
 \phi &: \mathcal{X} \rightarrow \mathcal{F} \\
 \psi &: \mathcal{F} \rightarrow \mathcal{X} \\
 \phi, \psi &= \arg \min_{\phi, \psi} \|X - (\psi \circ \phi)X\|^2
 \end{aligned}
 \tag{3.1}$$

3.1.1 Encoder

The encoder part is very important for this research because after we finished training the Autoencoder model, we will use this encoder part to compress the features vector. The structure of the encoder part is fully-connected feedforward neural networks, which can contain many hidden layers as we want. Encoder part starts with an Input layer then connects to many hidden layers. For traditional Autoencoder, the hidden layer's size in the encoder part will keep smaller than the previous layers until the bottleneck layer. We will get the compressed features vector from a bottleneck layer, so the size that we choose for the bottleneck layer will be the features vector's size.

The encoder function can be written as the neural network formula using activation function and training weight W and bias b . Features vector in the bottleneck layer will be represented as z .

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad (3.2)$$

3.1.2 Decoder

The structure of the decoder part is also a fully-connected feedforward neural networks as the Encoder part. The bottleneck layer will be an input for the decoder part. The hidden layer's size in the decoder part will keep larger than the previous layers until the output layer, which has the same size as the input layer of the encoder. Although in this research, we will use only the encoder part to generate features vector, we still need the decoder to make the Autoencoder work properly for the training process because we can not train encoder alone without a decoder. The decoder function also able to be written in the same way as encoder function but change weight \mathbf{W}' and bias \mathbf{b}' .

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}') \quad (3.3)$$

3.1.3 Loss function

The vital part of Autoencoder is the loss function. During the training process, Autoencoder will use the loss function to calculate and try to minimize the loss between input and reconstruction output. The Ideal goal is zero loss; it is mean the reconstruction output will be the same as an input. After replace z from formula 3.2 to z in formula 3.3 , then the loss function can then be written as shown in formula 3.4.

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2 \quad (3.4)$$

3.1.4 Experiment of reconstruction output

This section will show the reconstruction result from the traditional Autoencoder, not only reconstruction results from our input dataset but also the sample car images, which make us more clear about how each type of Autoencoder works. For reconstruction of input data, The y-axis represents 707 patients and the x-axis represents ten measurement values over 20 days, starting from 10 values of the first day and so on. The reconstruction output from traditional Autoencoder is not quite the same as the original input because it depends on the training dataset. For traditional Autoencoder, we train the model with the whole dataset. If data in the dataset is not similar to each other, then some incorrect elements can happen after the reconstruction method. For example, the car in the original image drives follow the curve road, but in reconstruction image, the car drivers follow the straight road. This error happens because some car pictures in the training set drives follow the straight road.

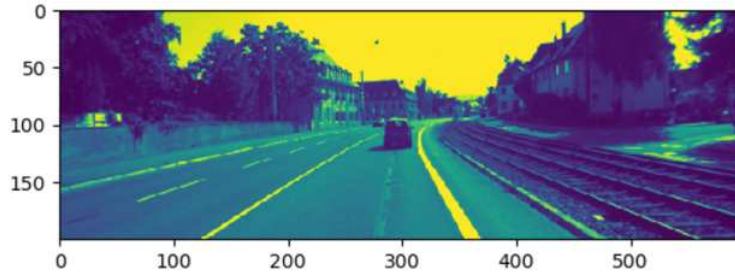


Figure 3.2: The original sample image

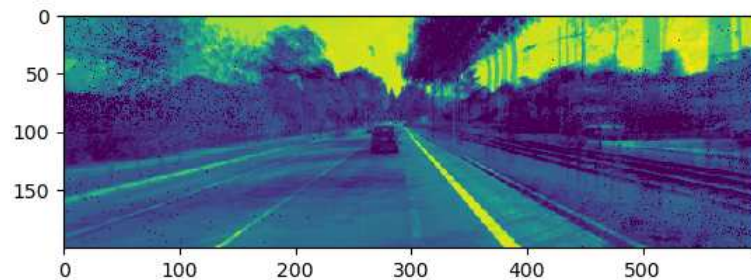


Figure 3.3: The reconstruction of sample image from Autoencoder

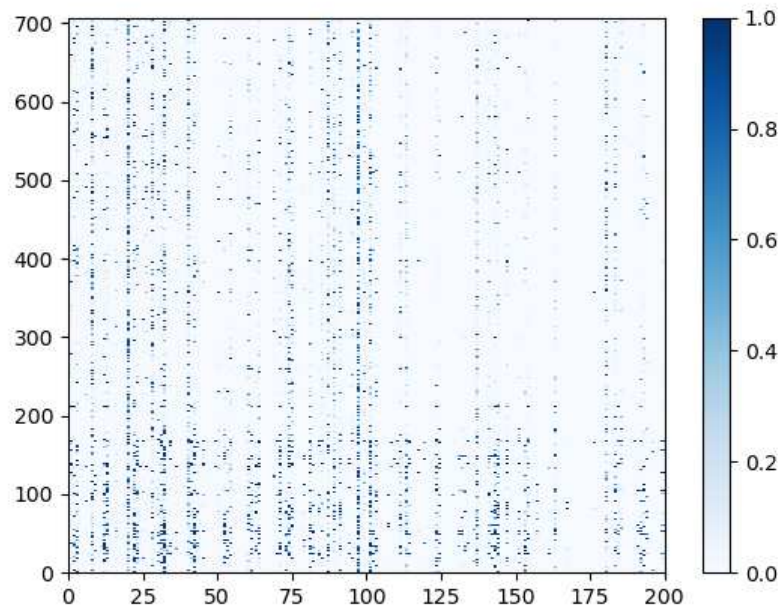


Figure 3.4: The original input data

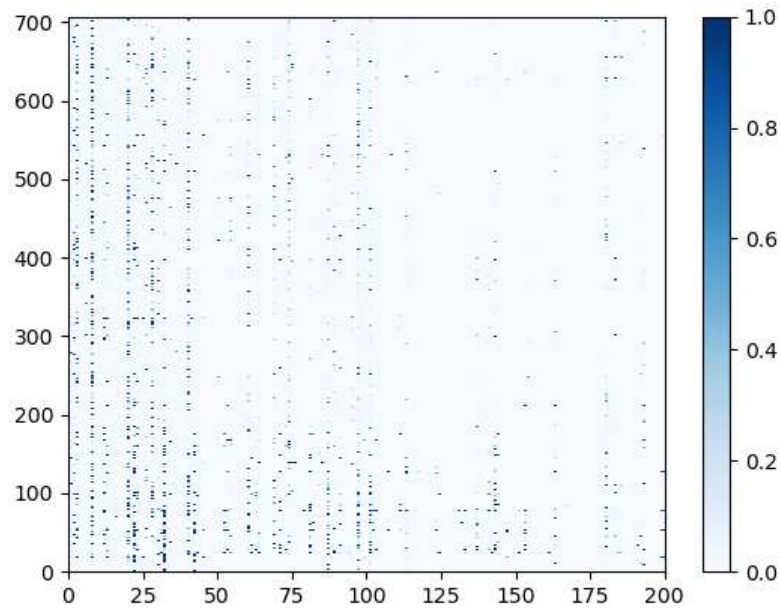


Figure 3.5: The reconstruction of input data from Autoencoder

3.2 Split Autoencoder

The second method is the Split Autoencoder. We trained different types of Autoencoders, individual Autoencoder and whole dataset Autoencoder. For the whole dataset Autoencoder, it was the same as traditional Autoencoder that we trained the model with the whole dataset. For the individual Autoencoder, We trained 1 Autoencoder by using one sub dataset of input. From 863 patients of the original dataset, we will have total 863 individual Autoencoder, and the outputs from both Autoencoders will be concatenated as new features vector. This vector will be the input for the next prediction process. This idea came when we were working on the previous project[7], the dataset of that project was very large tensor, so we had to split it out to many small tensors. After applying some methods on the sub-tensors, we found that we got unique information for each tensor. That is why we try to apply the splitting idea on this project. We think about using just only 1 Autoencoder for the whole dataset, it will work well if all sub dataset is not different too much. By the way, there are also have some unique features of each sub dataset that might be ignored when using trained Autoencoder from the whole dataset. That is why we tried to make an individual Autoencoder for features extraction method to keep both unique features and common features.

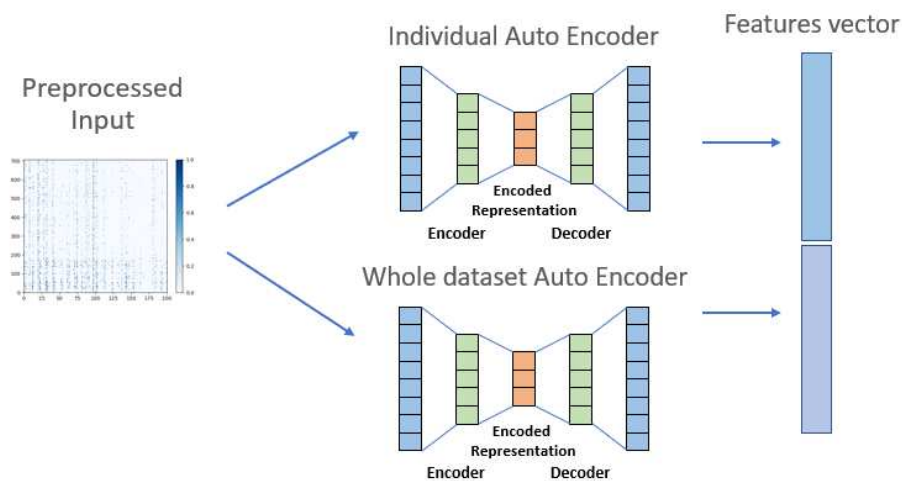


Figure 3.6: The structure of split Autoencoder

3.2.1 Experiment of reconstruction output

This section shows the comparison between reconstruction output from Individual Autoencoder and the original input. We choose to show the output from Individual Autoencoder because this is the main part that Split Autoencoder different from other models. Individual Autoencoder will be trained using only one data input, not the whole dataset, so the reconstruction errors will not be the same as traditional Autoencoder. As we can see in the reconstruction image, all of the incorrect elements are the elements from the original input, such as a pillar or signs. The same elements in the wrong location, but they still keep the correct features like a pillar is placed on the side of the road, not in the middle. This training concept using only one data for the compression method is so good, but we have to keep in mind that this trained model works for this data only. It seems like an overfitting problem but gives us a good compressed vector without noise from others.

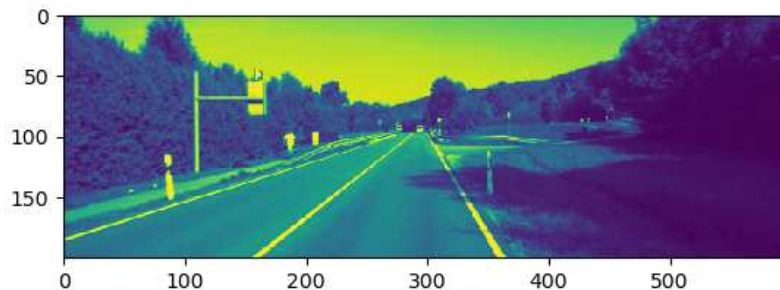


Figure 3.7: The original sample image

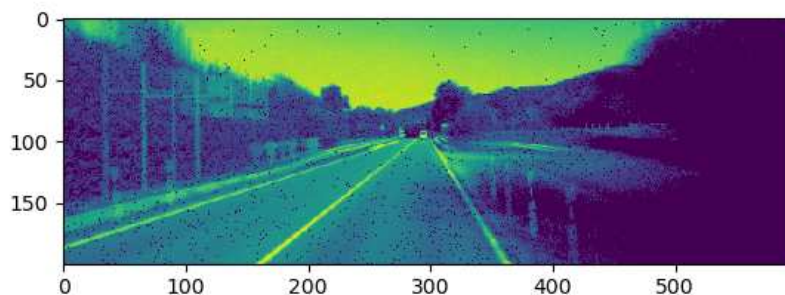


Figure 3.8: The reconstruction of sample image from Individual Autoencoder

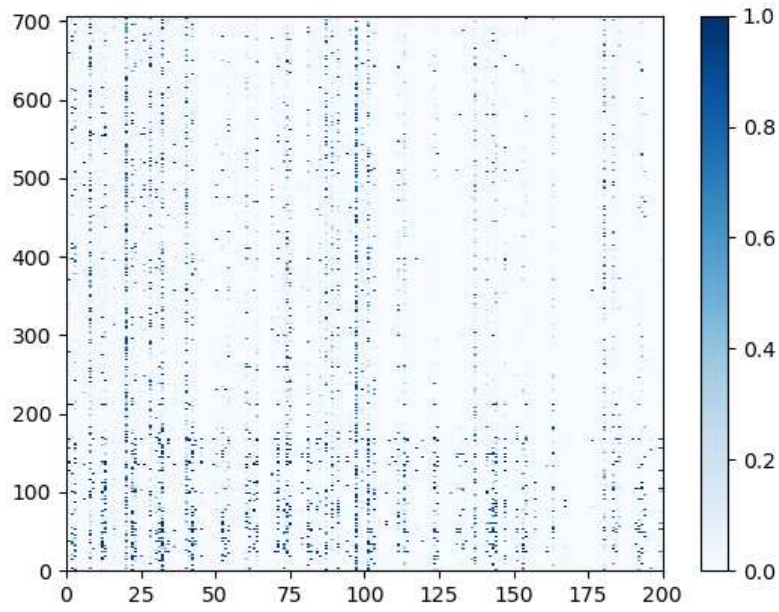


Figure 3.9: The original input data

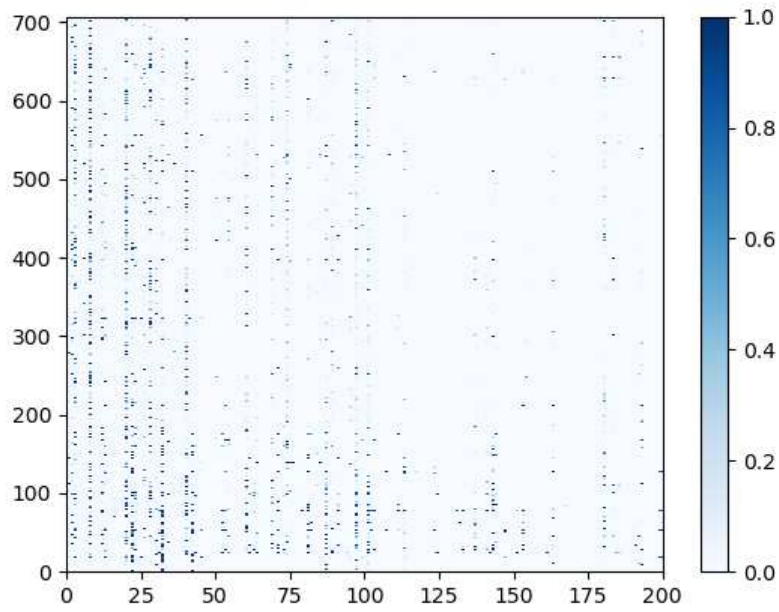


Figure 3.10: The reconstruction of input data from Individual Autoencoder

3.3 Long Short Term Memory (LSTM) Autoencoder

3.3.1 Concept of Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM) [5] is a type of Recurrent Neural Network[9] which works well for the sequence prediction problems. Our dataset is also the time series, so we try to combine LSTM and Autoencoder [10] to extract the essential features that came along with the sequential data.

The main concept of LSTM that makes it better than other types of traditional Recurrent Neural Network [11] [12] is LSTM be able to controls when the memory should be updated or forgot. The basic structure of the LSTM cell[13] [14] [15]is shown in Figure 3.11. In one LSTM cell, there is an input gate, forget gate, cell state and output gate.

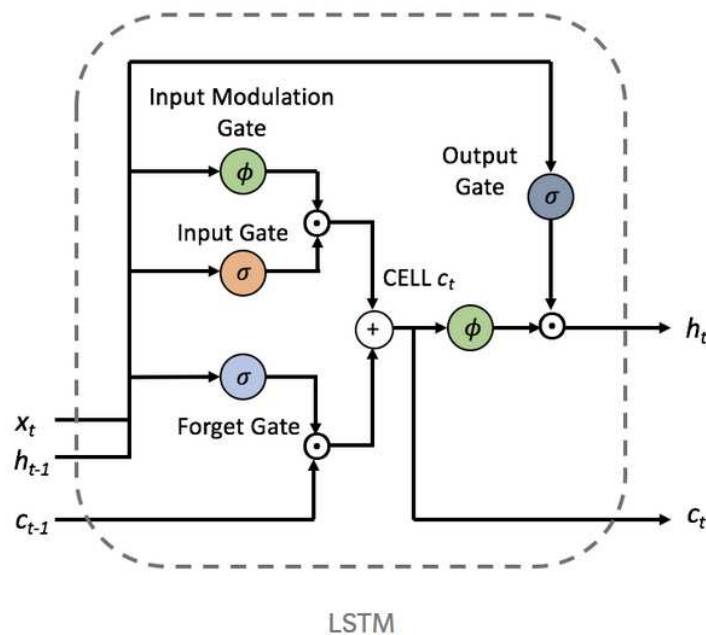


Figure 3.11: The structure of Long Short Term Memory cell (LSTM)

The first step in LSTM is to use forget gate to decide which information that we will forget it out by using x_t and $h_t - 1$ as input and then produce output f_t as shown in formula 3.5.

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f) \quad (3.5)$$

The next step is to decide which information we will keep on the cell state. This step contains one sigmoid layer to produce the information that will be updated i_t and one tanh layer to produce new candidate value \tilde{C}_t as shown in formula 3.6 and 3.7.

$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i) \quad (3.6)$$

$$\tilde{C}_t = \tanh (W_C \cdot [h_{t-1}, x_t] + b_C) \quad (3.7)$$

After we got f_t , i_t and \tilde{C}_t from previous step, now we can use these value to new cell state C_t . Firstly, we will for get some information from previous by multiply C_{t-1} and f_t , then add up with $i_t * \tilde{C}_t$ as shown in formula 3.8.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (3.8)$$

Finally, the last step is to decide what will be sending out as output from the LSTM cell. We will combine cell state, which is computed from the previous step with output from the sigmoid gate, then we will get the final output from the LSTM cell as shown in the formula 3.9 and 3.10.

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o) \quad (3.9)$$

$$h_t = o_t * \tanh (C_t) \quad (3.10)$$

3.3.2 Long Short Term Memory Autoencoder

The concept of Long Short Term Memory Autoencoder [5] [16] is similar to traditional Autoencoder, there is encoder and decoder part, and features vector at the bottleneck layer also be generated from the encoder part. The main difference is that traditional Autoencoder will not take a sequence as input, but Long Short Term Memory Autoencoder can take sequential data as input. The great features of Long Short Term Memory Autoencoder are that it can produce sequential data output, although only one dimensional vector is stored at the bottleneck layer.

The basic structure of Long Short Term Memory Autoencoder is shown in Figure 3.12. The input to the model is a sequence of vectors. The encoder part takes this sequence input. After that, the features vector will be generated at the bottleneck layer. Then the decoder part will take over and outputs a prediction for the target sequence.

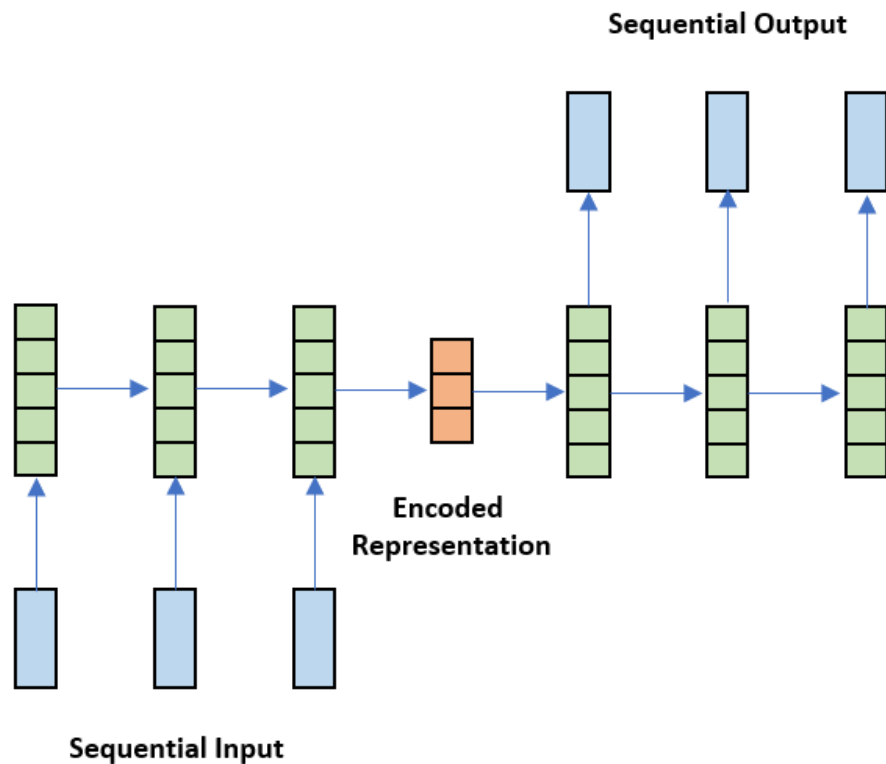


Figure 3.12: The basic structure of Long Short Term Memory Autoencoder

To set up a neural network for Long Short Term Memory Autoencoder, Normally an input shape of Long Short Term Memory layer will be three dimensional array, the first dimension is the batch size, the second dimension is the number of time steps that we are using and the last one is the number of features. The default output, which is produced from Long Short Term Memory also be three dimensional as shown in Figure 3.13. The parameter `return_sequence` will be set as `True` to makes each cell per time steps emit a signal.

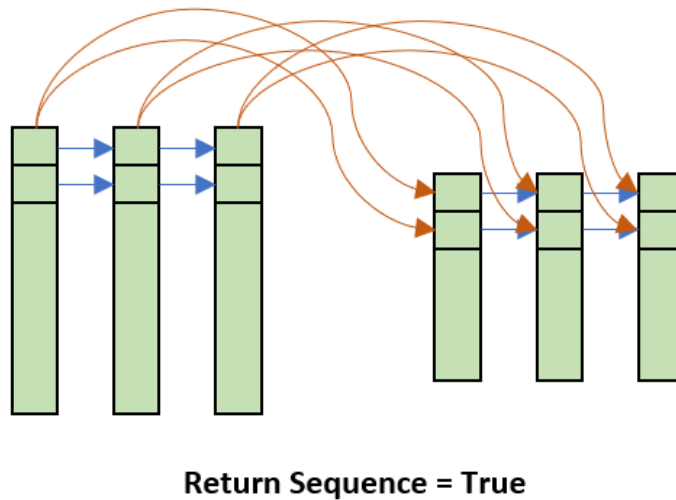


Figure 3.13: The setting of Long Short Term Memory layer which `return_sequence = True`

In order to build a Long Short Term Memory Autoencoder, we need to force the output of LSTM layer to be just only a vector at the bottleneck layer. So The parameter `return_sequence` will be set as `False`. By the way, the decoder part is still looking for three dimensional array as input, right now what we have from the bottleneck layer is three dimensional array (batch size x size of features vector). To convert the 2D output back to 3D input for the decoder part, we used the `RepeatVector` layer to duplicate the features vector as the number of time steps as shown in Figure 3.14.

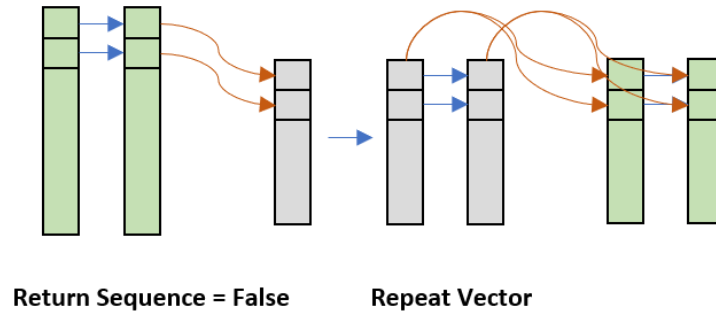


Figure 3.14: The setting of Long Short Term Memory layer which `return_sequence = False`, and using `RepeatVector` layer

After the last LSTM layer, we used `TimeDistributed` dense layer to create a vector of length equal to the number of features outputted from the previous layer for each time step.

The Differences between traditional LSTM network and LSTM Autoencoder:

- In traditional LSTM network, The parameter `return_sequence` will be set as `True`, to makes each cell per time steps emit a signal.
- There is no 1 dimensional encoded vector in the bottleneck layer of traditional LSTM network.
- We need an additional `RepeatVector` to duplicate vector for LSTM Autoencoder.
- We can use the encoded feature vectors as input for other Machine Learning model.

3.3.3 Experiment of reconstruction output

This section shows the comparison between reconstruction output from LSTM Autoencoder and the original input. The key strength of LSTM is that it works well with time series datasets, but we do not know if we combined it with Autoencoder because the encoded vector of LSTM Autoencoder at the bottleneck layer is just one vector and has to be recovered to many images over time. The sample images in Figure 3.15, we use 4 images/4 time step for training in each batch. From the reconstruction results in Figure 3.16, we found that the overall image looks good, every object can be recovered correctly, but we can see some noise from another time step. For instance, the car at the bottom right corner is recovered successfully, but we can see another blur car from the next frame stacked over that car. The reason might be the features vector has to store the information for every frame, so after the reconstruction method, data from another frame can appear on the focused frame.

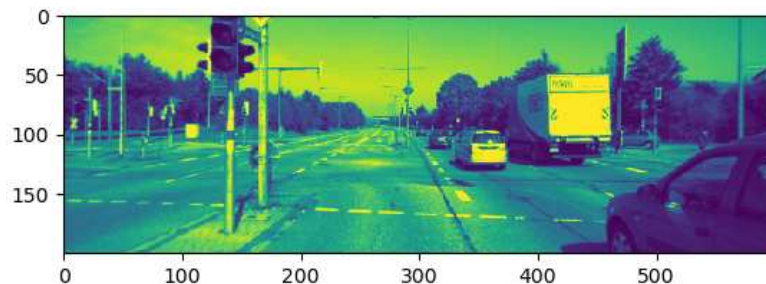


Figure 3.15: The original sample image

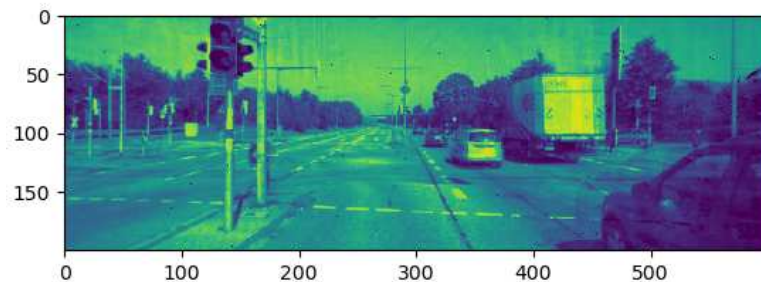


Figure 3.16: The reconstruction of sample image from LSTM Autoencoder

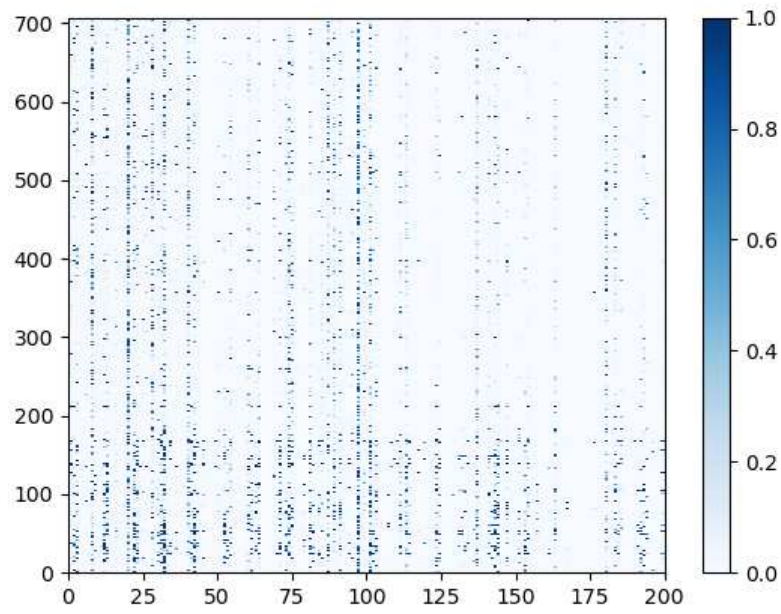


Figure 3.17: The original input data

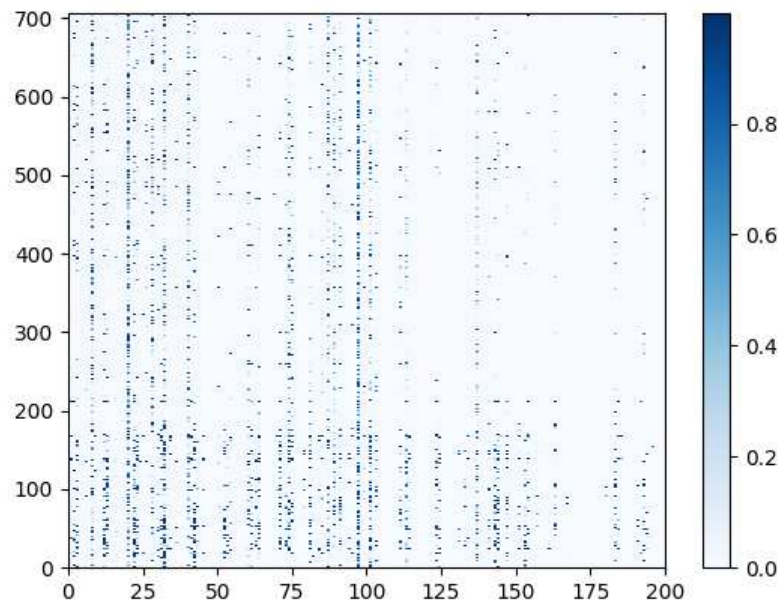


Figure 3.18: The reconstruction of input data from LSTM Autoencoder

3.4 Alternation of LSTM and Split Autoencoder

As the previous output from both LSTM Autoencoder and Split Autoencoder, we found strength for each type of Autoencoder. The strength of LSTM Autoencoder is that it works well when handling a sequential dataset and the strength of Split Autoencoder is that it can keep the unique information from the individual subset. So, this method we try to combine both strengths into one method. From 863 patients of the original dataset so totally we will have 863 individual Split LSTM Autoencoder, each output vector from both Autoencoder is size 30. After concatenation to the new features vector, size will be change to 60. Then this vector will be the input for the next prediction process.

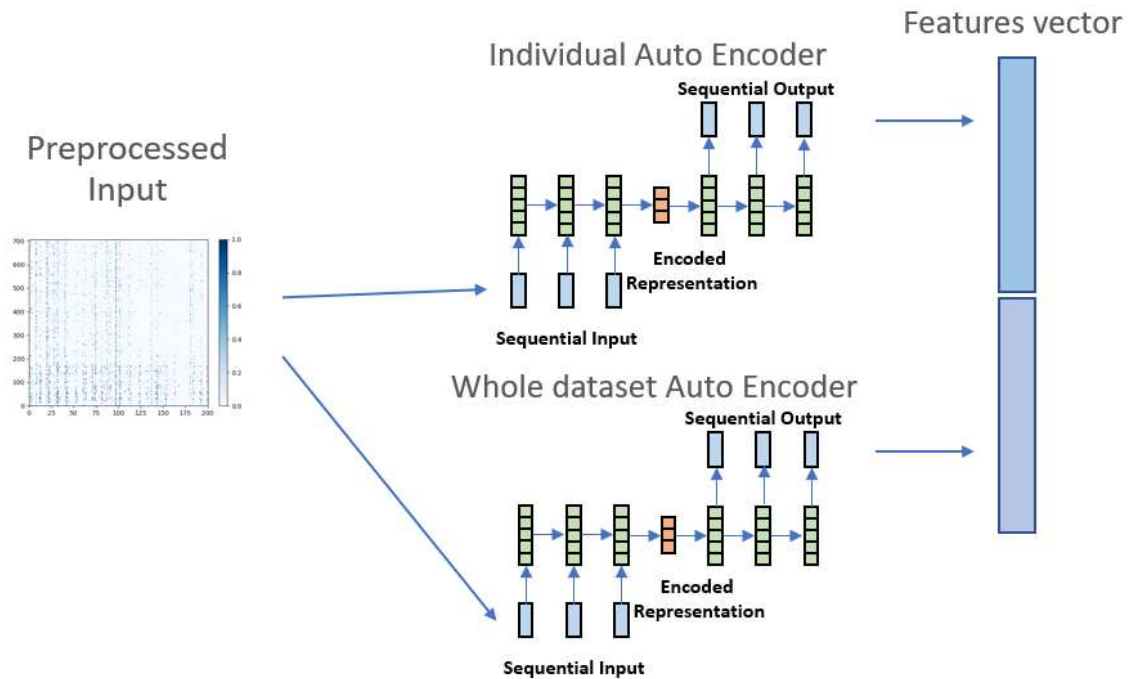


Figure 3.19: The basic structure of Split Long Short Term Memory Autoencoder

3.4.1 Experiment of reconstruction output

This section shows the comparison between reconstruction output from Individual LSTM Autoencoder and the original input. This method contains both strengths and weaknesses of two methods because of the combination of concepts between Split Autoencoder and LSTM Autoencoder. As a result in Figure 3.21, we can see all objects can be recovered correctly as the strength of Individual Autoencoder, but we still see the noises from another time step as the weakness of LSTM Autoencoder. Although the result from the reconstruction process will not come out as sharp as the original input, this does not mean that the features that we compressed lag of quality. Because the noise in the image may be a part of the data that is not important, another reason is that the vector compressed from LSTM Autoencoder needs to keep enough information to recover the set of images. Therefore, there are only necessary parts will be kept. As for testing whether the compression results are good or not, we have to continue testing the classification process.

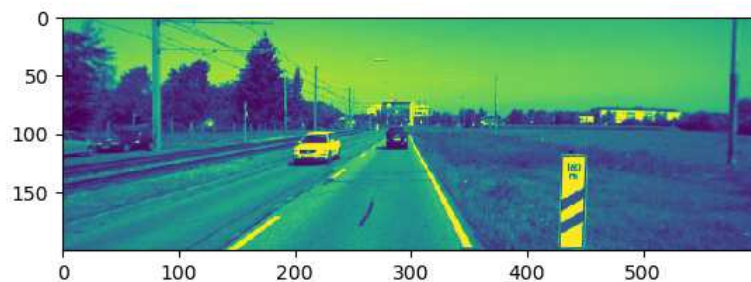


Figure 3.20: The original sample image

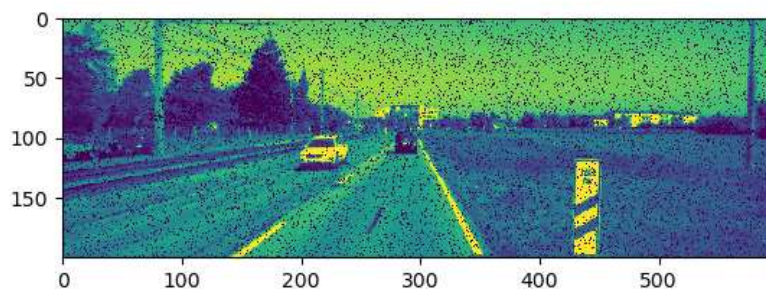


Figure 3.21: The reconstruction of sample image from Split LSTM Autoencoder

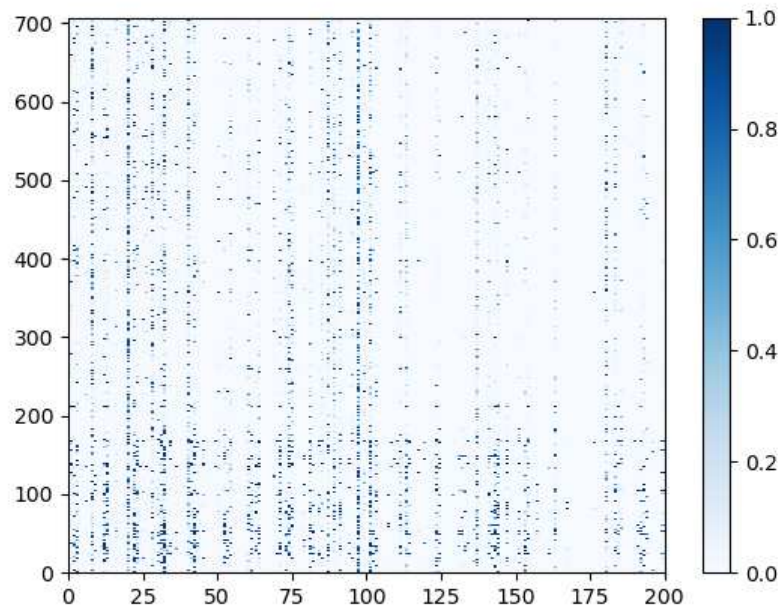


Figure 3.22: The original sample image

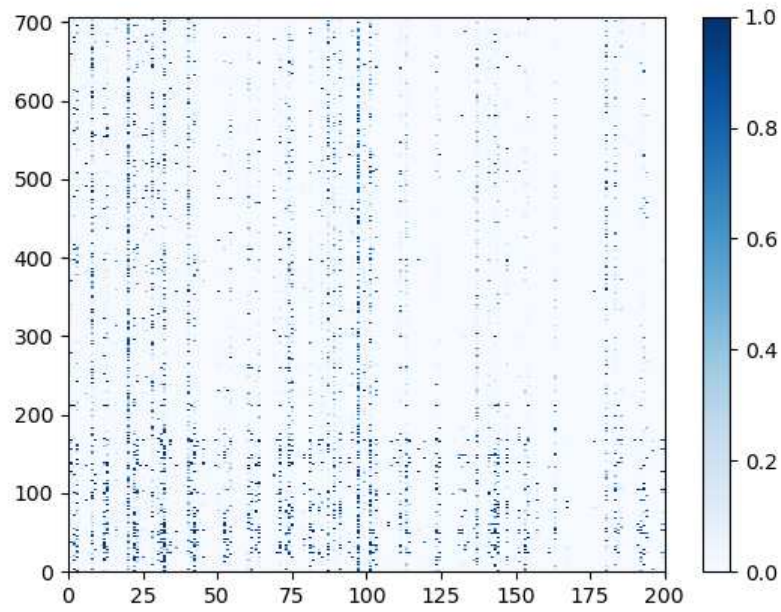


Figure 3.23: The reconstruction of sample image from Split LSTM Autoencoder

3.5 Deep Kernelized Autoencoder

Deep Kernelized Autoencoder is a method that combines the Time series Cluster Kernel (TCK)[17] with an Autoencoder. This method focuses on bringing the strengths of Time series Cluster Kernel (TCK), which can work well when faced with both multivariate time series (MTS) data and missing data problems, and the strengths of Autoencoder, which can compress the data to the lower dimension while keeping important information. By combining both techniques, we will get a method capable of generating features vector from the multivariate time series (MTS) with missing data.

3.5.1 Time Series Cluster Kernel

The Time series Cluster Kernel uses the missing patterns of multivariate time series dataset to calculate similarities in the dataset, instead of using other imputation methods, which might generate some biases. The kernel is created by joining the clustering results of many Gaussian mixture models (GMM) to ensured the robustness of hyperparameters.

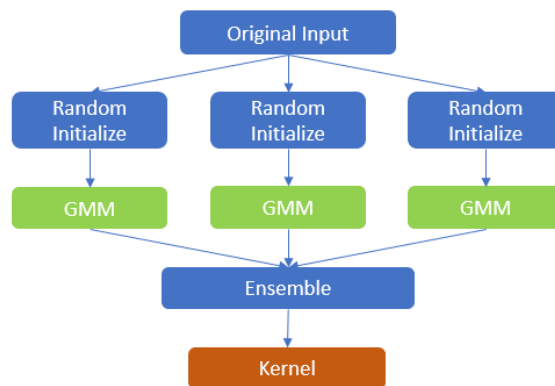


Figure 3.24: The structure of TCK that used in this project

3.5.2 Deep Kernelized Autoencoder

The Deep Kernelized Autoencoder [6] looks similar to the traditional Autoencoder but the different thing is how to calculate loss between input and output as shown in Figure 3.25.

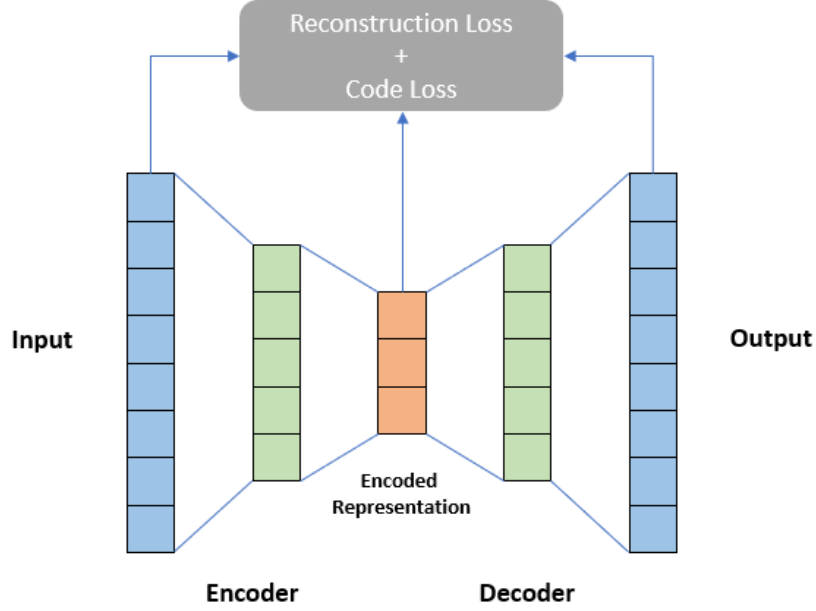


Figure 3.25: The structure of Deep Kernelized Autoencoder in this project

Loss function in Deep Kernelized Autoencoder (L) is computed base on 2 Loss, reconstruction loss $L_r(\cdot, \cdot)$ and code loss $L_c(\cdot, \cdot)$ as shown in formula 3.11 where λ is a hyperparameter for balancing 2 terms. Reconstruction loss is the same as traditional Autoencoder as shown in formula 3.12. Code loss is calculated from the normalized Frobenius distance between kernel (\mathbf{K}), result from TCK method, and code (\mathbf{C}), the bottleneck layer of Autoencoder as shown in formula 3.13.

$$L = (1 - \lambda)L_r(\mathbf{x}, \tilde{\mathbf{x}}) + \lambda L_c(\mathbf{C}, \mathbf{K}) \quad (3.11)$$

$$L_r(\mathbf{x}, \tilde{\mathbf{x}}) = E \{ \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 \} \quad (3.12)$$

$$L_c(\mathbf{C}, \mathbf{K}) = \left\| \frac{\mathbf{C}}{\|\mathbf{C}\|_F} - \frac{\mathbf{K}}{\|\mathbf{K}\|_F} \right\|_F \quad (3.13)$$

3.5.3 Experiment of reconstruction output

This section shows the comparison between reconstruction output from Deep Kernelized Autoencoder and the original input. We found that the result of this method looks different from others because it contains both imputation and compression techniques. The input data that used for training this model still contains the missing data. That is why after reconstruction, the model tries to recover data to the missing points.

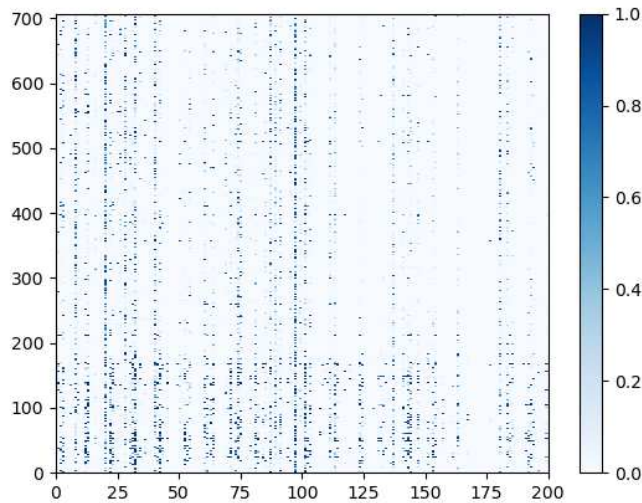


Figure 3.26: The original input dataset

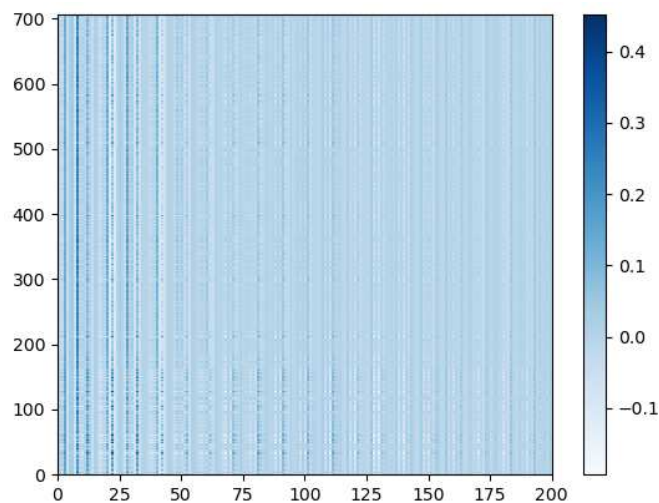


Figure 3.27: The reconstruction of input dataset from Deep Kernelized Autoencoder

CHAPTER 4

EXPERIMENT

Our experiments start with apply z-normalization method to the input dataset. For each patient in this dataset, there are 10 measurement parameters such as potassium and sodium over 20 time steps(day). The normalization method will bring all the attributes on the same scale. Not only scaling problem, but there is also a missing data problem, contains around 80% of missing data. After doing the normalization, we apply an imputation method to estimated more data. The imputation method we selected is Generative Adversarial Imputation Nets (GAIN). We worked with imputation methods in the previous project and found that GAIN is better than others when applying with a very sparse dataset. We gain around 15% of data back after applied Generative Adversarial Imputation Nets (GAIN). The input dataset after applied Normalization and Imputation is shown in Figure 4.1

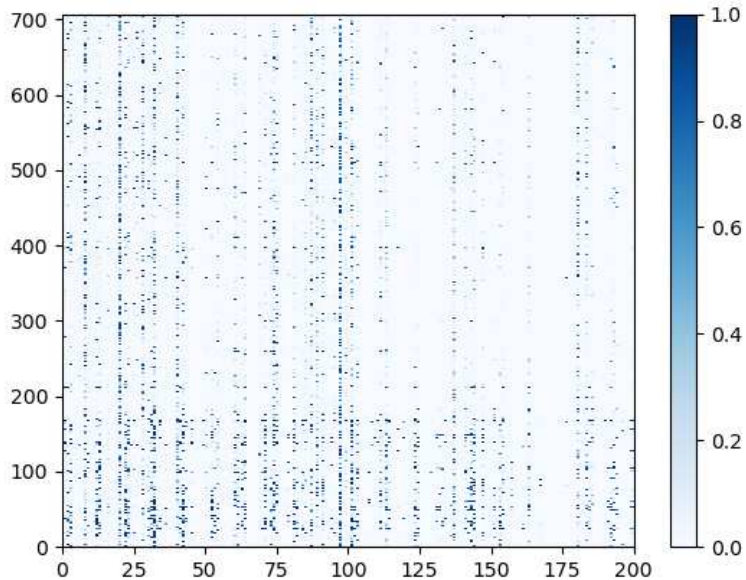


Figure 4.1: The input dataset after applied Normalization and Imputation

To generate the features vector we train various type of Autoencoder and use the encoder part of them as the compression method.

- Traditional Autoencoder
- Split Autoencoder
- Long Short Term Memory (LSTM) Autoencoder
- Alternation of LSTM and Split Autoencoder
- Deep Kernelized Autoencoder

The comparison of these methods in terms of data size that we can decrease will be shown in Table 4.2. For each patient, there are 10 attributes over 20 days, totally the original input contains 141,400 values (707 patients x 10 attributes x 20 time steps), after compressed the features vector of Traditional Autoencoder, LSTM Autoencoder and Deep Kernelized Autoencoder contain 30 values, and the features vector of Split Autoencoder and Split LSTM Autoencoder contains 60 values.

Table 4.1: A table of size of features vector type for each type of Autoencoder

No.	Method	Size (KB)	Percent of decrease (%)
1	Original Input	1,105	0
2	Pre-processed Input	553	49.95
3	Traditional Autoencoder	83	92.48
4	Split Autoencoder Individual	83	92.48
5	Split Autoencoder Combined	166	84.98
6	Long Short Term Memory (LSTM) Autoencoder	83	92.48
7	Split LSTM Autoencoder Individual	83	92.48
8	Split LSTM Autoencoder Combined	166	84.98
9	Deep Kernelized Autoencoder	83	92.48

For the classification step, we use traditional neural network 4 layers, the number of nodes in each layer is divided by 2 from the previous layer and set to one node in the last. Activation function for every layer except the last one is Rectified Linear Unit (ReLU). The last layer uses Sigmoid. A loss that we used to train Autoencoder is Binary Cross-Entropy Loss because in this sample dataset we have only 2 class (with and without surgical site infections)

Table 4.2: A table of accuracy of features vector type for each type of Autoencoder

No.	Method	Accuracy (%)	Percent of increase (%)
1	Original Input	73.86	0
2	Pre-processed Input	81.81	10.76
3	Traditional Autoencoder	69.88	-5.39
4	Split Autoencoder Individual	67.00	-9.29
5	Split Autoencoder Combined	71.60	-3.06
6	Long Short Term Memory (LSTM) Autoencoder	80.45	8.92
7	Split LSTM Autoencoder Individual	72.72	-1.54
8	Split LSTM Autoencoder Combined	80.86	9.23
9	Deep Kernelized Autoencoder	85.79	16.15

Next, we will show the pictures that represent to reconstruction and compressed features vector for each method. For reconstruction sample pictures, The y-axis represents 707 patients in the training set. The x-axis represents all ten measurement values over 20 days. The y-axis positions 0 - 10 will indicate the measurement values measured on the first day, 11-20 would indicate the second day, and so on until 191 - 200 would mean the last day. The color in the graph shows the value stored in that location. For compressed features vector, The y-axis also mean 707 patients in the training set. The x-axis will represent the data points in vector. The features vector's size is 30 for Autoencoder, LSTM Autoencoder and Deep Kernelized Autoencoder. The features vector's size is 60 for Split Autoencoder and Split LSTM Autoencoder.

4.1 Autoencoder

Starting with the Autoencoder first, the size of features vectors which are compressed by the Autoencoder has been reduced from the original input 1,105 KB to 83 KB, around 92.48% reduction. The accuracy is 69.88%, which results in 5.39% less than the original input, indicating that using the Autoencoder in the compression method may not be a good choice. The pattern of feature vectors is maintained similar to the original input. If we look at both figures, the data in rows 0 - 200 fluctuate and the pattern is not smooth. The row after that data will become smoother. From the experiment, we found that if we want to use the Autoencoder method to compress features vector, we should choose the dataset that the whole data is quite similar. Because of training an autoencoder, we use the whole input dataset to train only one Autoencoder, so there is a chance that reconstruction will be imperfect and can receive features from other data. To sum up, the Autoencoder implementation for the compression method is not good because the results are still lower than the original input, and other methods can do better.

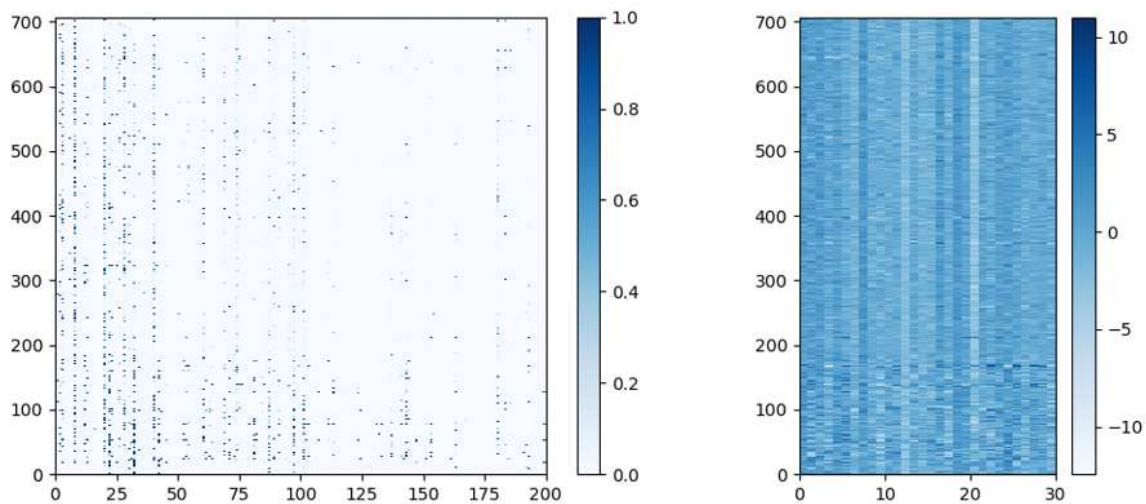


Figure 4.2: The comparison between reconstruction (left) and compressed (right) of Autoencoder

4.2 Split Autoencoder

Split Autoencoder is a technique that generated features vector by concatenating two features vectors from Individual Autoencoder and Whole dataset Autoencoder, resulting in a double size than compressing with only one Autoencoder. The size of the features vector from Split Autoencoder is 166 KB, decreasing 84.98% compared to the original input. Accuracy is 71.60%. We found that the accuracy of the combination of 2 Autoencoders gives better results than separating. Because an Individual Autoencoder will extract the unique features of the data while the common features of the data will become from the Whole dataset Autoencoder. From the figure of the features vector, the column 0 - 30, which is obtained from The Whole dataset Autoencoder is very smooth because the dataset is quite similar, so the common features extracted are not much different. The columns 31 - 60 obtained from Individual Autoencoder, are still keeping the unique information. In summary, using Split Autoencoder may not be the best way. But it may be necessary to experiment with other input datasets, which is not the time series dataset.

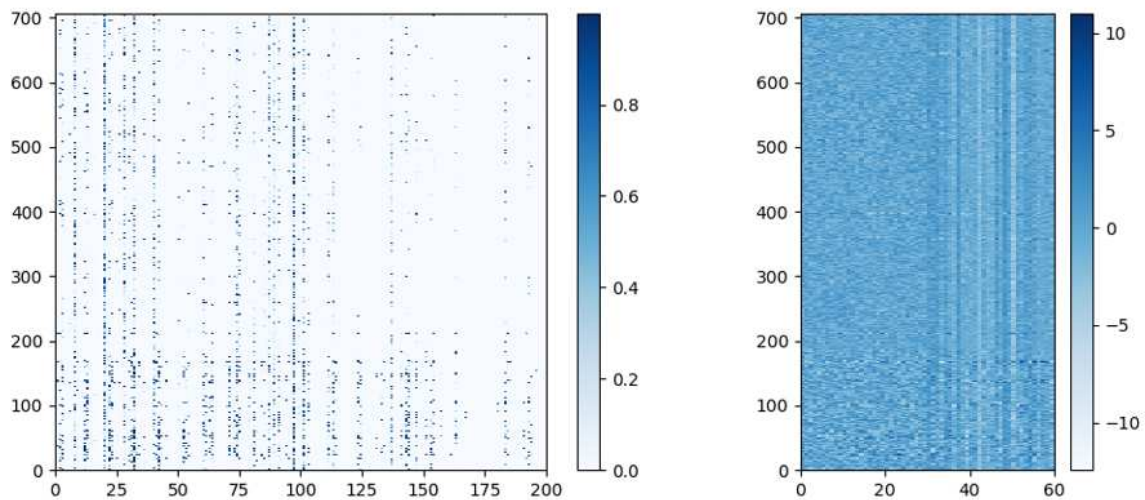


Figure 4.3: The comparison between reconstruction (left) and compressed (right) of Split Autoencoder

4.3 LSTM Autoencoder

LSTM Autoencoder is a method that combines the strength of working well with times series data from LSTM and compression capability from Autoencoder. So we have a method that works well for compressed times series data. The size of the features vector from LSTM Autoencoder is 83 KB, decreasing of 92.48% compared to the original input. The accuracy is 80.45% which is 8.92% higher than the original data as well. The characteristics of the features vector from LSTM Autoencoder has quite a distinct contrast, not as smooth as the previous Autoencoder. Another interesting point is that this feature vector shows darker lines in the middle rather clearly than other methods and also pulls the data in right zone of the Figure 4.4 to be more prominent. The reason may be that the information in the right section is information that comes later if viewed in terms of time. In the LSTM cell, information from the past will be pass through the future. That's why It makes us see that the data looks more prominent. Moreover, this feature vector still holds the common pattern that other methods can do, for instance, the fluctuating of data in rows 0 - 200. In summary, LSTM Autoencoder is an interesting method because the compressed vector has a small size, high accuracy and requires less train model when compared to Split concept.

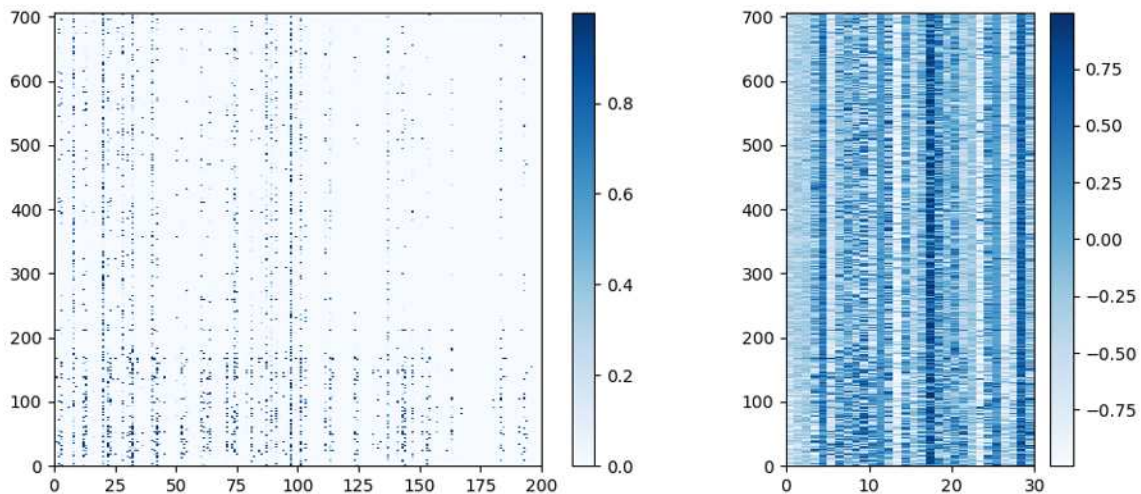


Figure 4.4: The comparison between reconstruction (left) and compressed (right) of LSTM Autoencoder

4.4 Split LSTM Autoencoder

Split LSTM Autoencoder method is similar to the Split Autoencoder method, and the difference is just changing from traditional Autoencoder for both individual and whole dataset Autoencoder to LSTM Autoencoder. The features vector's size of LSTM Autoencoder is 166 KB, decreasing 84.98% from the original. And the accuracy is 80.86%, which is the second-best compression method that we tried, around 9.23% better than the prediction result of the original input. As a result of the previous section, we found that the compression by using Split Autoencoder method is great for keeping both unique and common features, especially if we change from traditional Autoencoder to LSTM Autoencoder when we deal with the time series dataset. The pattern of this feature vector looks different from other methods. In Figure 4.5, we can see that the side of the feature vector from the whole dataset Autoencoder is high contrast compared to another side. The values are similar to a binary. The reason may be the same as the previous method; this part holds on the common features that do not represent the specific pattern. To sum up, Split LSTM Autoencoder is a good method to do the compression if we work with a time series dataset and keep both unique and common features.

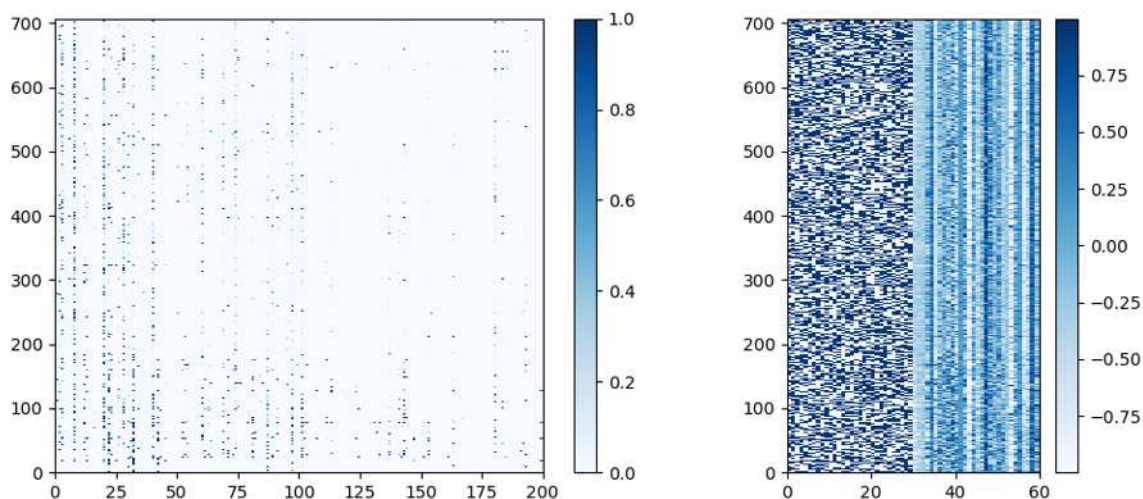


Figure 4.5: The comparison between reconstruction (left) and compressed (right) of Split LSTM Autoencoder

4.5 Deep Kernelized Autoencoder

The last method is Deep Kernelized Autoencoder, and this method gave us the best result compare to others. The size of the features vector from Deep Kernelized Autoencoder is 83 KB, decreasing 84.98% compared to the original input. The accuracy is 85.79%, which is the best accuracy that we found from this research, around 16.15% better than the original one. This method is different from other Autoencoder models in terms of loss calculation, loss of other models will calculate the reconstruction loss between output and input, but this method uses another Code loss which calculate between encoded vector and TCK. TCK is computed based on the similarities between data points. This result shows that if the input dataset has a similar input pattern, the similarities between data points will affect the compression performance. We think the similarities are not only related to the compression method but also important to other machine learning model when training with a similar input pattern. The pattern of this feature vector looks a bit different from others. It contains many bold dark vertical lines with high density in the middle a bit right. In summary, Deep Kernelized Autoencoder is a great tool for time series vector compression and gives us an idea to implement different kinds of loss for Autoencoder.

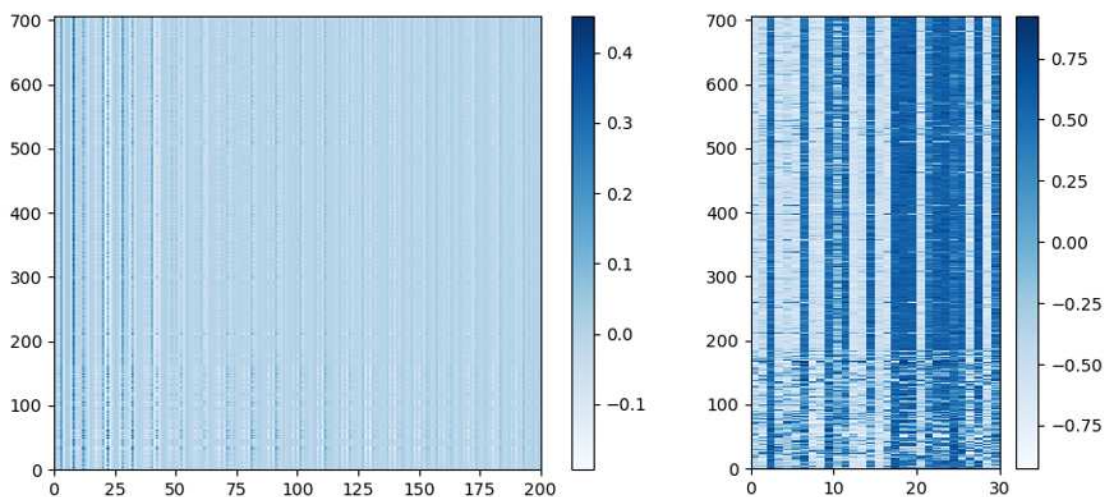


Figure 4.6: The comparison between reconstruction (left) and compressed (right) of Deep Kernelized Autoencoder

CHAPTER 5

CONCLUSION

As a result of the previous chapter, we found that after applied features extraction methods, the compressed size was reduced to around 10 percent of the original input. Not only the size of compressed features vector that was reduced but also the size of pre-processed input. After applying normalization and imputation method, the size was reduced to half of the original. In terms of accuracy, we found that most methods return the better accuracy than original input, because of the original input still contains missing data and not perform normalization. But the comparison with accuracy after pre-processed, just only LSTM Autoencoder, Split LSTM Autoencoder and Deep Kernelized Autoencoder can return higher or similar accuracy. Although the result from the Split concept in the individual part will not meet a good performance, after combined with the completed part, result has become better. Because only unique features are not enough to train the model, combining both unique and common features can help improve the model's performance. As our expectation, LSTM works well with the time series dataset and be the second-best rank in the table. The best result came from Deep Kernelized Autoencoder because this method focuses more on the similarities between each data point. During the training process, the loss function of the other method will compare only the same points but loss of function. Deep Kernelized Autoencoder combined reconstruction loss and code loss, focusing on the similarities with other points. To improve the result, one may combined strength of both LSTM with Deep Kernelized. We will get the model that can deal with time series and also focusing on the similarities.

CHAPTER 6

FUTURE WORK

Based on the current work, next step we plan to

- After graduation and back to work in the oil and gas industry, we will try to apply these methods to the real dataset. There might be other constraints and required some modification.
- The combined strength of LSTM Autoencoder which work well with time series dataset and Deep Kernelized which contains Time Series Cluster Kernel (TCK).
- Our team are working on another Reinforcement Learning so we will try to apply the concept of Reinforcement Learning to the compression method.

REFERENCES CITED

- [1] Michael Feindt Maximilian Christa, Andreas W. Kempa-Liehr. Distributed and parallel time series feature extraction for industrial big data applications. *arXiv:1610.07717v3*, 2017.
- [2] James Jordon Jinsung Yoon and Mihaela van der Schaar. Gain: Missing data imputation using generative adversarial nets. *arXiv:1806.02920*, 2018.
- [3] G.E. Hinton D.E. Rumelhart and R.J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing. Vol 1: Foundations.*, pages 3–10, 1986.
- [4] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems 6*, pages 3–10, 1994.
- [5] Gaurangi Anand Lovekesh Vig-Puneet Agarwal Pankaj Malhotra, Anusha Ramakrishnan and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv:1607.00148v2*, **1**, 2016.
- [6] Karl Øyvind Mikalsen Filippo Maria Bianchi and Robert Jenssen. Learning compressed representations of blood samples time series with missing data. *arXiv:1710.07547v1*, 2017.
- [7] Hoon Seo. *Estimating Mineral Distribution Using Machine Learning Models*. Colorado School of Mines, 2019.
- [8] Christian A. Hammerschmidt Ramiro D. Camino and Radu State. Improving missing data imputation with deep generative models. *arXiv:1902.10666v1*, 2019.
- [9] Timothy Wong and Zhiyuan Luo. Recurrent auto-encoder model for large-scale industrial sensor signal analysis. *arXiv:1807.03710v1*, 2018.
- [10] Elman Mansimov. Ruslan Salakhutdinov Nitish Srivastava. Unsupervised learning of video representations using lstms. *arXiv:1502.04681v3*, 2016.
- [11] Yang Yang Guillaume Sautière Adam Golinski, Reza Pourreza and Taco S Cohen. Feedback recurrent autoencoder for video compression. *arXiv:2004.04342v1*, 2020.
- [12] Idoia Ochoa Mohit Goyal, Kedar Tatwawadi. Shubham Chandak. Deepzip: Lossless data compression using recurrent neural networks. *arXiv:1811.08162v1*, 2018.

- [13] Michael Ph. Illustrated guide to lstm's and gru's. <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>, 2018.
- [14] Christopher Olah. Understanding lstm networks. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 2015.
- [15] Chitta Ranjan. Step-by-step understanding lstm. <https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352>, 2019.
- [16] Timothy Wong and Zhiyuan Luo. Recurrent auto-encoder model for large-scale industrial sensor signal analysis. *arXiv:1807.03710v1*, 2018.
- [17] Cristina Soguero-Ruiz Karl Øyvind Mikalsen, Filippo Maria Bianchi and Robert Jenssen. Time series cluster kernel for learning similarities between multivariate time series with missing data. *Pattern Recognition*, 76, 2017.