2020학년도 겨울학기 M2177.006800 001 인공지능/빅데이터의 실무 응용 연구
IAB 코딩 기술서

Conv-LSTM모델을 이용한 다중 센서 기반 이상 감지

조선해양공학과
에너지시스템연구실
이나영
2018-28756

목차

1. 서론

공정 설비를 진단하는 예지보전 시스템은 공정의 무인화와 자동화를 위하여 필수적인 부분이다. 화학 공정은 많은 기계 장치 및 장치들 사이를 잇는 배관과 그 부속품으로 구성되어 있으며, 이러한 장치들이 고장나거나 wear가 진행될 경우 성능이 저하되는 문제 뿐 아니라 참사가 일어날 수 있는 가능성이 있다.

공정 장비에는 다양한 센서들이 설치되어있는데 이 센서들은 공정 장비의 상태를 감지하기 위해서 사용된다. 주로 사용되는 센서는 공정 내 유체의 상태를 측정하기 위한 것으로 온도, 압력, 유량에 관한 것이 많고, 장비 자체의 이상 감지를 위한 것으로는 진동 센서가 추가되기도 한다. 진동 센서가 추가되는 경우는 주로 회전 장비(Rotary Equipment)를 사용하는 경우로, 베어링이나 축의 진동이 장비 자체의 이상에 직접적인 연관이 있는 경우이다.

공정 장비의 이상에는 다양한 종류가 있고 그 원인도 다양하며, 각각의 이상 유형(failure mode) 별로 감지 할 수 있는 센서의 종류 역시 다양할 수 있다. 지금까지는 이러한 다양한 이상에의 감지와 분석이 자동적으로 이루어지기가 쉽지 않았다. 대부분의 경우 이상을 감지했을 때 사람이 직접 이상 상황을 확인하고 그 원인을 예측하는 방식으로 원인의 진단과 분석이 이루어졌으며, 그나마 자동화가 된 시스템의 경우에도 기계가 완전하게 이상을 진단하는 것보다는 전문가 시스템을 이용하는 방법이 다수를 차지했다. 또한 기계학습, 특히 딥러닝을 이용한 이상 감지의 경우

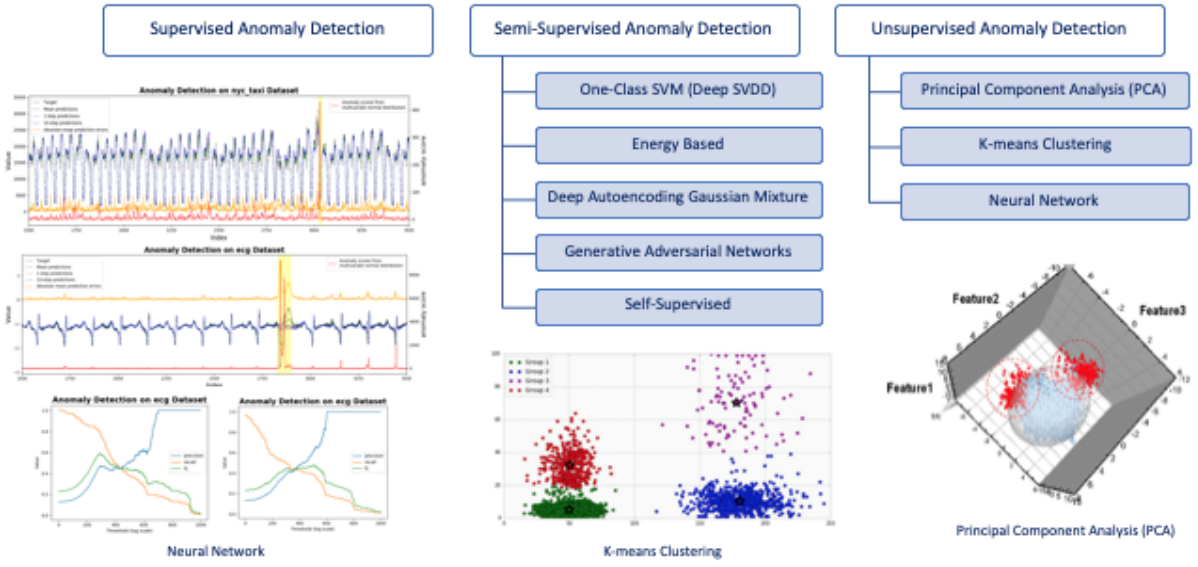
2. 기존 문헌 분석

2.1. 방법론

2.1.1. Anomaly Detection for Sequential Data

Anomaly Detection은 말 그대로 이상 감지를 하는 것으로, 이상 상태와 정상 상태를 구분하는 것이 가장 주된 Task가 된다. 이를 위해 데이터들 중 다수를 차지하는 것들을 찾아내어 정상 상태를 정의하는 과정을 필요로 하며, Clustering등의 통계적 기법을 이용하여 일정한 부분을 찾아내는 방법을 주로 사용한다. 이상 감지 알고리즘은 크게 Supervised, Semi-supervised, Unsupervised 이렇게 세 가지로 나뉘며, 뉴럴 네트워크를 이용한 Anomaly Detection은 Unsupervised방법에 속한다.

그러나 모든 Anomaly Detection이 시계열을 대상으로 하는 것은 아니다. 하지만 공정 장비와 같은 장비의 이상 예측은 시계열 데이터 형식의 센서 계측 데이터를 기반으로 하여 이상 감지를 수행하기 때문에, 데이터의 특성상 시계열 기반 이상 감지를 수행하게 된다. 시계열 기반 이상 감지 모델으로는 SVM, DAGMM, ARMA등의 비뉴럴넷 방법이 있고, 뉴럴넷을 이용하는 방법에는 RNN (Recurrent Neural Network)기반의 LSTM(Long-Short Term Memory), LSTM-ED(Long-Short Term Memory Encoder-Decoder), conv-LSTM(Convolution Long-Short Term Memory)모델이 있다. 비 뉴럴넷 방법은 주로 점들 사이의 거리를 기반으로 근접 데이터들과 비근접 데이터를 분류하는 방식을 이용한다. 뉴럴넷을 이용하는 방법은 주로 RNN을 기반으로 한 시계열 데이터 분석인 LSTM(Hochreiter & Schmidhuber, 1997)을 이용하되 이를 응용한 것들이 대부분이다. 단변수의 경우 LSTM모델을 이용하여 이상 감지 분석을 수행하였고, 많은 변수를 이용하여 이상 감지를 수행하는 경우에는 LSTM만을 사용하지 않고 LSTM에 Encoder 와 Decoder를 앞뒤에 추가한 LSTM-ED(Malhotra et al., 2016)나 Convolution Layer와 Encoder-Decoder를 추가한 conv-LSTM(Chuxu et al., 2019)등 새로운 방법론이 지속적으로 고안되는 과정에 있다.



2.2. Case Specific

본 연구는 공정 장비의 이상 예측이라는 특수한 케이스를 대상으로 하는 것으로, 본 연구에서 참고할 수 있는 특정 분야의 연구들에 관하여 간략하게 개괄하였다.

2.2.1. Wind Turbine Anomaly Detection

딥러닝을 이용한 이상 감지를 장비 분야에 응용한 경우로는 Wind Turbine에 관한 연구가 대부분이었다. Wind Turbine의 진동 데이터와 여러 다른 데이터를 기반으로, 딥러닝 및 기타

다른 통계 방법론을 이용한 이상 감지 알고리즘을 적용한 연구(Zhao et al., 2018)가 많음을 알 수 있었다.

2.2.2. Process Equipment Anomaly Detection

본 연구는 “A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data”라는 문헌의 프레임워크와 모델을 주로 참고하여 수행되었다.

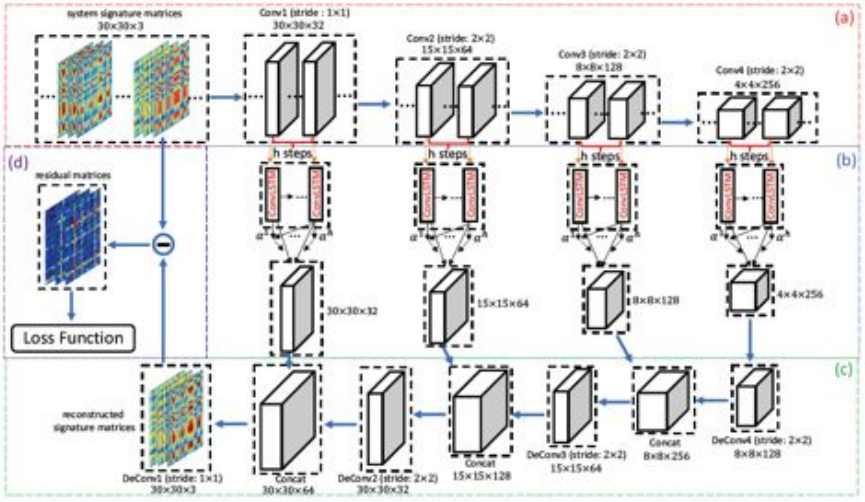
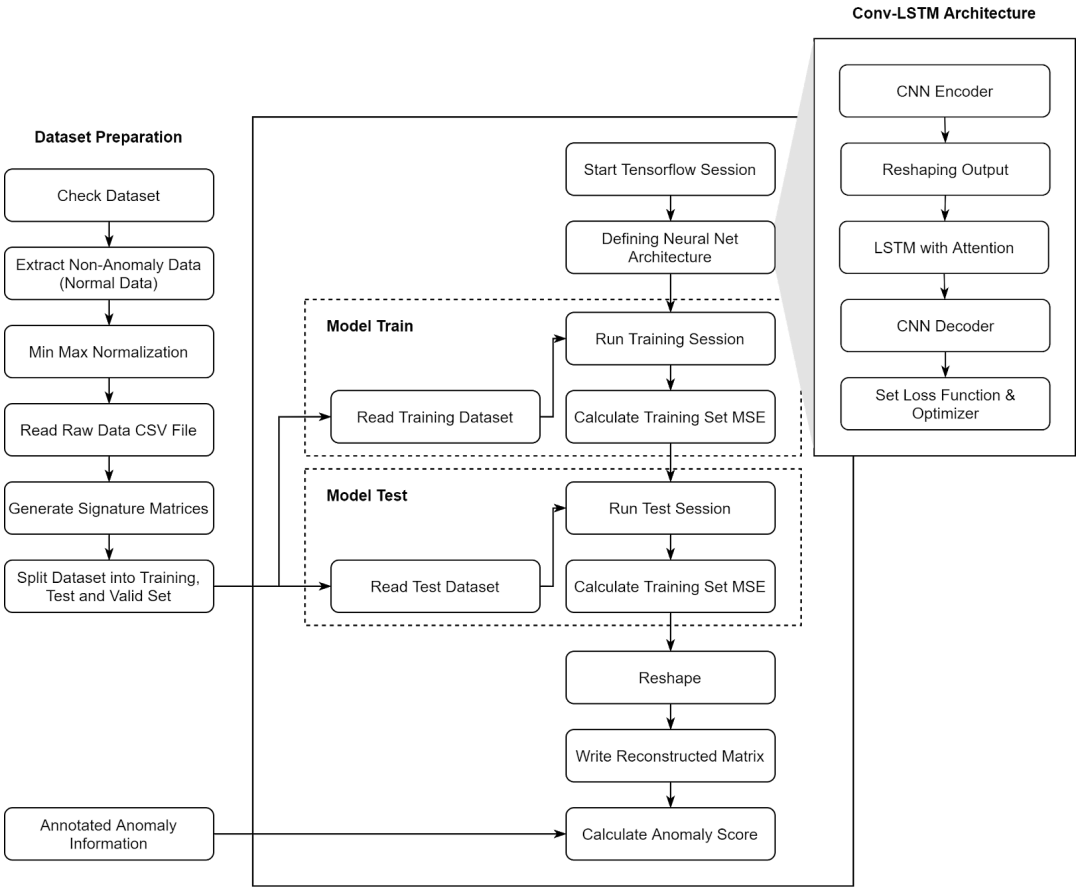


Figure 2: Framework of the proposed model: (a) Signature matrices encoding via fully convolutional neural networks. (b) Temporal patterns modeling by attention based convolutional LSTM networks. (c) Signature matrices decoding via deconvolutional neural networks. (d) Loss function.

3. 프레임워크 및 모델 Framework and Model



3.1. Signature Matrix를 활용한 상태 특징화

Signature Matrix는 이 Matrix는 1차원 시계열 데이터들의 모음을 Convolution을 사용할 수 있는 2차원 데이터로 변환시킨다는 데 의의가 있다고 할 수 있다. 이 2차원 행렬은 연관도행렬의 형식을 띄고 있으며, 그래서 행과 열의 항목은 동일하며 각 항목은 수집한 시계열의 항목들이라고 볼 수 있다. 즉 각 행과 각 열은 센서 하나에 대응된다고 볼 수 있다.

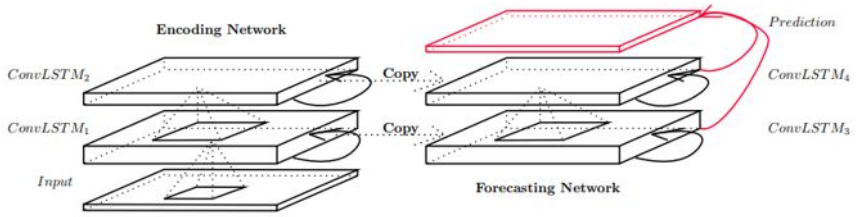
3.2. Attention

어텐션의 기본 아이디어는 디코더에서 출력을 예측하는 매 시점마다 인코더의 전체 입력 문장을 다시 한 번 참고하는 것이다. 다만 전체를 참고하는 것이 아니라 이 중에서 연관도가 높은 부분만을 참고한다.

3.3. Conv-LSTM

Conv-LSTM은 LSTM이전과 이후에 Convolution Encoder와 Convolution Decoder를 추가하는 형식을 가지고 있는 뉴럴넷 아키텍쳐이다. 해당 아키텍쳐는 원래 LSTM을 이용한 시계열 분야에서 고안된 것이 아니라 Convolution을 주로 이용하던 컴퓨터 비전 분야에서 동영상 등

사진이 시간에 따라 변화하는 경우에 대응하기 위하여 만들어진 것이다. 이러한 conv-LSTM 아키텍처는 기존의 LSTM과 다르게 한번에 여러 인풋을 받아서 병렬적으로 처리하며, Encoding하는 네트워크를 카피해와서 Forecasting 네트워크에 활용, 다음 상태를 예측하는 방법을 이용한다.



$$\tilde{\mathcal{X}}_{t+1},\dots,\tilde{\mathcal{X}}_{t+K} = \underset{\mathcal{X}_{t+1},\dots,\mathcal{X}_{t+K}}{\arg\max}\ p(\mathcal{X}_{t+1},\dots,\mathcal{X}_{t+K} \mid \hat{\mathcal{X}}_{t-J+1},\hat{\mathcal{X}}_{t-J+2},\dots,\hat{\mathcal{X}}_t)$$
$$\approx \underset{\mathcal{X}_{t+1},\dots,\mathcal{X}_{t+K}}{\arg\max}\ p(\mathcal{X}_{t+1},\dots,\mathcal{X}_{t+K} \mid f_{encoding}(\hat{\mathcal{X}}_{t-J+1},\hat{\mathcal{X}}_{t-J+2},\dots,\hat{\mathcal{X}}_t))$$
$$\approx g_{forecasting}(f_{encoding}(\hat{\mathcal{X}}_{t-J+1},\hat{\mathcal{X}}_{t-J+2},\dots,\hat{\mathcal{X}}_t))$$

LSTM

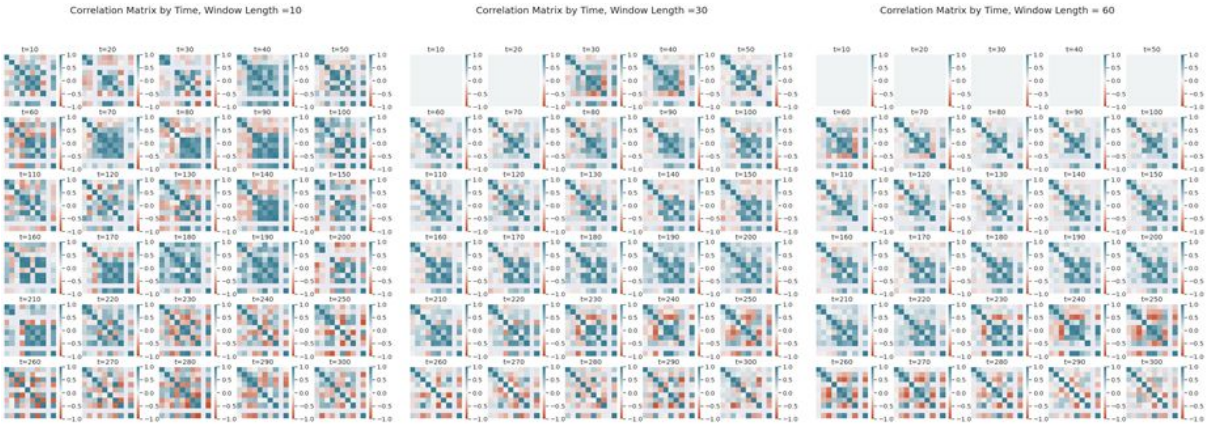
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \circ c_t + b_o)$$
$$h_t = o_t \circ \tanh(c_t)$$

Conv-LSTM

$$i_t = \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f)$$
$$\mathcal{C}_t = f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o)$$
$$\mathcal{H}_t = o_t \circ \tanh(\mathcal{C}_t)$$

3.4. 1D to 2D Transition for Convolution

2D 변환을 위해 1차원 시계열데이터들을 2차원으로 변환하는 방법에는 여러가지가 있을 수 있으며, 이에 따라서 결과값이 바뀐다. 본 연구에서는 2가지 방법을 사용하였다. 하나는 Signature Matrix를 이용하였고 다른 하나는 Correlation Matrix를 이용하였다.



3.5. Anomaly Detection 방법론

이 프레임워크에서 차용하는 Anomaly Detection방법은 LSTM-ED(Malhotra et al., 2016)에서 사용하는 방법과 동일한 것으로, 기본적으로는 시계열 예측 모델을 이용하여 예측 오차(Prediction Error)를 구하는 방식을 채택한다. LSTM이전에 Encoder를, 이후에 Decoder를 삽입하여 LSTM을 학습시키되 이러한 LSTM 자체는 Anomaly와는 관련되지 않은 정상 데이터를 학습시켜 이후 단계에서의 값을 정상 범위 내에서 예측하도록 하고 실제 관측값이 예측 값과 그 범위에서 벗어날 경우를 Anomaly 상황으로 정의한다. 이 Anomaly의 정도를 나타낸 것이 Anomaly Score이다. encoder-decoder 모델과 reconstruction 모델을 사용하는 방법적인 부분은 LSTM-ED 모델에서의 사용과 동일하다.

Reconstruction Error $e^{(i)}$ at $t^{(i)}$: $e^{(i)} = \left| x^{(i)} - x'^{(i)} \right|$

Anomaly Score : $a^{(i)} = (e^{(i)} - \mu)^T \Sigma^{-1} (e^{(i)} - \mu)$

Anomaly Score에서의 μ, Σ 값은 Normal Distribution $\aleph(\mu, \Sigma)$ 상의 파라미터로, 이는 데이터셋의 일부를 대상으로 MLE(Maximum Liklihood Estimation)를 수행하여 얻은 분포이다. Supervised 세팅에서는 임의로 지정한 Threshold인 τ 에 대해 Anomaly Score인 $a^{(i)}$ 가 $a^{(i)} > \tau$ 인 경우 Anomaly로 분류하고 $a^{(i)} \leq \tau$ 인 경우 Normal값으로 분류한다. 또한 만약 데이터가 많을 경우 Threshold τ 도 역시 학습을 수행하도록 할 수 있으며, 이 때 학습은 validation set의 Recall R과 Precision P에 대해 $F_\beta = (1 + \beta^2) \times P \times R / (\beta^2 P + R)$ 값을 최대화하도록 하는 값을 찾는다.

4. 코드 Code

4.1. 학습 환경 Environment

본 코드는 Google Colab상에서 노트북 형태로 작성되었으며 Runtime 역시 Google Colab에서 제공하는 GPU를 이용하였다. 사용한 GPU는 NVIDIA SMI이다.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.27.04    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P0    26W /  70W |    111MiB / 15079MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+
```

4.2. 라이브러리 및 버전 Library and Version

사용한 딥러닝 라이브러리는 Tensorflow 1.5.1버전이다. 여기서 필요한 수치적 계산을 보완하기 위하여 Numpy 및 Scipy 라이브러리를 추가적으로 활용하였으며, 그래프 작성을 위하여는 Seaborn의 Heatmap모듈과 Matplotlib을 이용하였다. Cell별 Execution time을 모니터링하기 위한 모듈로는 ipython-autotime 라이브러리를 이용하였다.

4.3. 코드 해설 Line-by-Line Explanation

4.3.1. 라이브러리 임포팅 Importing

```
%tensorflow_version 1.x
```

```
import tensorflow as tf
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
from pylab import rcParams
rcParams['figure.figsize'] = 7,7
import seaborn as sns
sns.set(color_codes=True, font_scale=1.2)
%load_ext autotime

import os
import re
```

코드의 가장 첫 부분으로, 사용할 라이브러리등을 지정하는 부분이다. Version 1로 지정된 Tensorflow, Numpy, Pandas, Scipy, Matplotlib, Seaborn, Autotime 라이브러리를 미리 Import하였다. os는 파이썬에서 기본적으로 제공하는 기타 운영체제 인터페이스 라이브러리로 파일의 읽고 쓰기를 지원하며 re역시 파이썬에서 기본적으로 지원하는 내장 모듈로 정규표현식 사용을 지원한다.

```
tf.test.gpu_device_name()
!nvidia-smi
print(tf.__version__)
```

본 코드가 Google Colab에서 자동적으로 제공하는 GPU환경을 이용하는 만큼, 어떤 GPU가 할당되었는지 모르기 때문에 해당 하드웨어의 정보를 얻기 위한 부분을 작성해주었다.

4.3.2. 하이퍼 파라미터 지정 Hyperparameter Initialization

```
gap_time = 10  # gap time between each segment
win_size = [10, 30, 60]  # window size of each segment
step_max = 5 # maximum step of ConvLSTM
```

위의 세 Hyperparameter는 LSTM의 Window에 대한 Gap time, window size과 Convolution 에 대한 Maximum step을 지정한다. `gap_time`은 하나의 Window에서 다음 Window로 넘어갈 때 이동하는 거리이다. `win_size`는 윈도우 사이즈로, LSTM에서 기본 단위로 이용될 Window의 크기를 지정한다. 본 프레임워크에서는 3가지의 Window size를 지정했는데 이는 해당 시스템에서의 Anomaly 크기를 계산한 뒤 이에 맞게 설정한 것이다.

```
if not os.path.exists("../content/data/"):
    os.makedirs("../content/data/")
raw_data_path = '../content/data/FSRU_Trial_Pump_concat_on_normalied.csv'
valid_data_path =
'../content/data/FSRU_Trial_Pump_concat_on_normalized_validation.csv'

model_path = '../content/MSCRED/'
train_data_path = "../content/data/train/"
test_data_path = "../content/data/test/"
reconstructed_data_path = "../content/data/reconstructed/"
```

해당 파일 내에서의 모델 경로 및 파일명 지정을 데이터를 처리 전 미리 수행하여야 한다. 파일은 csv파일로 미리 만들어주었으며, 이는 이후에 Pandas 라이브러리 내의 csv reading 함수를 이용하여 파일 리딩을 수행하였다.

```
train_start_id = 60 # maximum window size matters
train_end_id = 200

test_start_id = 201
test_end_id = 400

valid_start_id = 400
valid_end_id = 450

training_iters = 20
save_model_step = 1
```

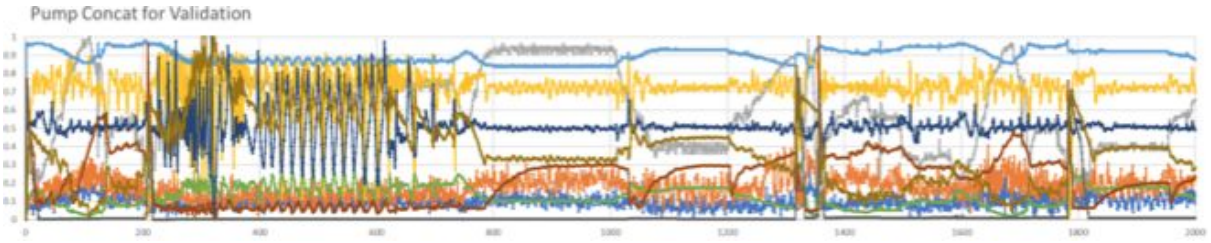
데이터의 숫자를 알고있으므로 이를 기반으로 train, test, validation set의 크기를 지정한다. 이는 `start_id`와 `end_id`처럼 인덱스를 지정하는 방식으로 수행하였다. `training_iters`은 Training Epoch를 의미하고, `save_model_step`은 저장 간격을 알려주는 부분이다.

```
learning_rate = 0.002
threshold = 0.005
alpha = 1.5
```

이외에도 학습에서 Optimizer의 속도를 결정하는 `learning_rate`, Anomaly Score지정 부분에서 사용되는 값인 `threshold`와 `alpha`를 지정해주었다.

4.3.3. 데이터 전처리 Preprocessing

사용한 데이터의 형태는 아래와 같은 1차원 Timeseries Data로, convolution을 위해 2차원 데이터로 변경하여 주는 과정이 필요했다.



2D Matrice로 변환하는 과정으로 Correlation 기반의 연산을 수행하였으며, 이를 통해 signature matrice를 생성하였다.

```
class SignatureMatrices:
    def __init__(self):
        self.raw_data = pd.read_csv(raw_data_path, header=None).T
        self.series_number = self.raw_data.shape[0]
        self.series_length = self.raw_data.shape[1]
        self.signature_matrices_number = int(self.series_length / gap_time)

        print("series_number is", self.series_number)
        print("series_length is", self.series_length)
        print("signature_matrices_number is", self.signature_matrices_number)

    def signature_matrices_generation(self, win):
        if win == 0:
            print("The size of win cannot be 0")

        raw_data = np.asarray(self.raw_data)
        signature_matrices = np.zeros((self.signature_matrices_number,
self.series_number, self.series_number))

        for t in range(win, self.signature_matrices_number):
            raw_data_t = raw_data[:, t - win:t]
            signature_matrices[t] = np.dot(raw_data_t, raw_data_t.T) / win

        return signature_matrices
```

먼저 Signature Matrix를 만들기 위해, Min-Max Normalization을 사용하였다. Min-Max Normalization은 데이터의 Normalization을 위해 가장 자주 쓰는 방법으로, Minimum Value를 0으로, Maximum Value를 1로 하여 선형적으로 스케일링하는 방법이다.

$$x_{norm} = \frac{x-min}{max-min}$$

이 때 범위는 각 컬럼이며, min max 값은 각 컬럼에서의 min값과 max값을 찾은 뒤 전체 컬럼에 적용하였다. Normalization은 본 코드 상에서 작성된 것이 아니라 데이터를 넣기 전 엑셀 상에서 수행한 것이다.

```
 def generate_train_test(self, signature_matrices):
        train_dataset = []
        test_dataset = []

        for data_id in range(self.signature_matrices_number):
            index = data_id - step_max + 1
            if data_id < train_start_id:
                continue
            index_dataset = signature_matrices[:, index:index + step_max]
            if data_id < test_start_id:
                train_dataset.append(index_dataset)
            else:
                test_dataset.append(index_dataset)

        train_dataset = np.asarray(train_dataset)
        train_dataset = np.reshape(train_dataset, [-1, step_max,
self.series_number, self.series_number,
                                                   signature_matrices.shape[0]])
        test_dataset = np.asarray(test_dataset)
        test_dataset = np.reshape(test_dataset, [-1, step_max,self.series_number,
self.series_number,
                                                 signature_matrices.shape[0]])

        print("train dataset shape is", train_dataset.shape)
        print("test dataset shape is", test_dataset.shape)

        train_path = "../content/data/train/"
        if not os.path.exists(train_path):
            os.makedirs(train_path)
        train_path = train_path + "train.npy"

        test_path = "../content/data/test/"
        if not os.path.exists(test_path):
```

```
            os.makedirs(test_path)
        test_path = test_path + "test.npy"

        np.save(train_path, train_dataset)
        np.save(test_path, test_dataset)
```

위에서 만들어준 Signature Matrix를 Training Set과 Test Set으로 나누어 학습시켜주는 과정을 Class 내의 함수로 만들어주었다. 해당 파일들은 .npy형식으로 저장되어 사용자가 일반 뷰어로는 볼 수 없으나, 파이썬 코드가 읽기는 빠른 형식으로 저장되었다. 만약 Path가 없는 경우 해당 Path를 생성할 수 있도록 하는 코드가 포함되었다.

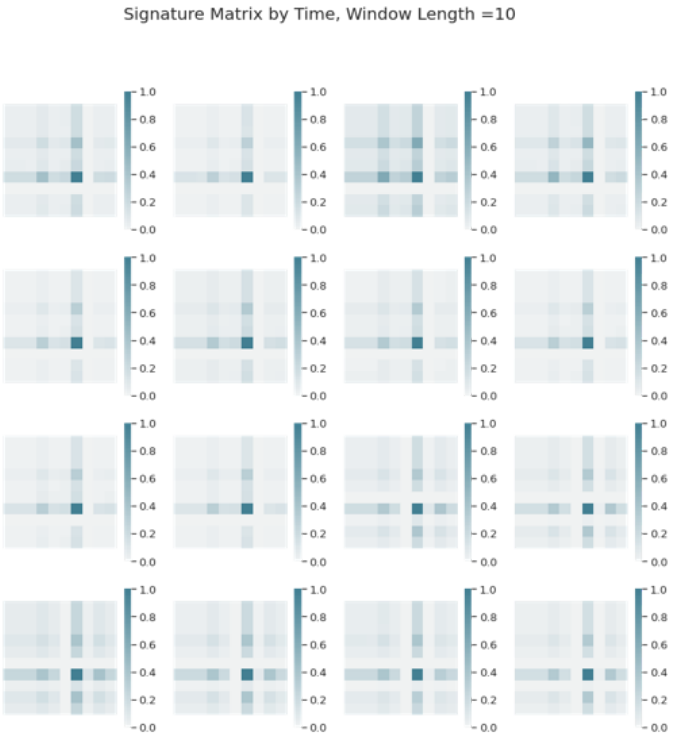
```
if __name__ == '__main__':
    Matrices = SignatureMatrices()
    signature_matrices = []

    # Generation signature matrices according the win size w
    for w in win_size:
        signature_matrices.append(Matrices.signature_matrices_generation(w))

    signature_matrices = np.asarray(signature_matrices)
    print("the shape of signature_matrices is", signature_matrices.shape)

    # Generate train and test dataset
    Matrices.generate_train_test(signature_matrices)
```

이 부분에서는 위의 Hyper Parameter 지정 부분에서 지정해주었던 Window 크기의 Array에 속하는 element의 갯수만큼 반복하여 Time Series 전체에 대한 Signature Matrix 생성을 수행해준다. 그리고 위에서 작성한 함수를 통해 만들어진 Signature Matrix들을 Train/Test Set으로 분할하여 저장해준다. 수행 결과는 아래와 같다.



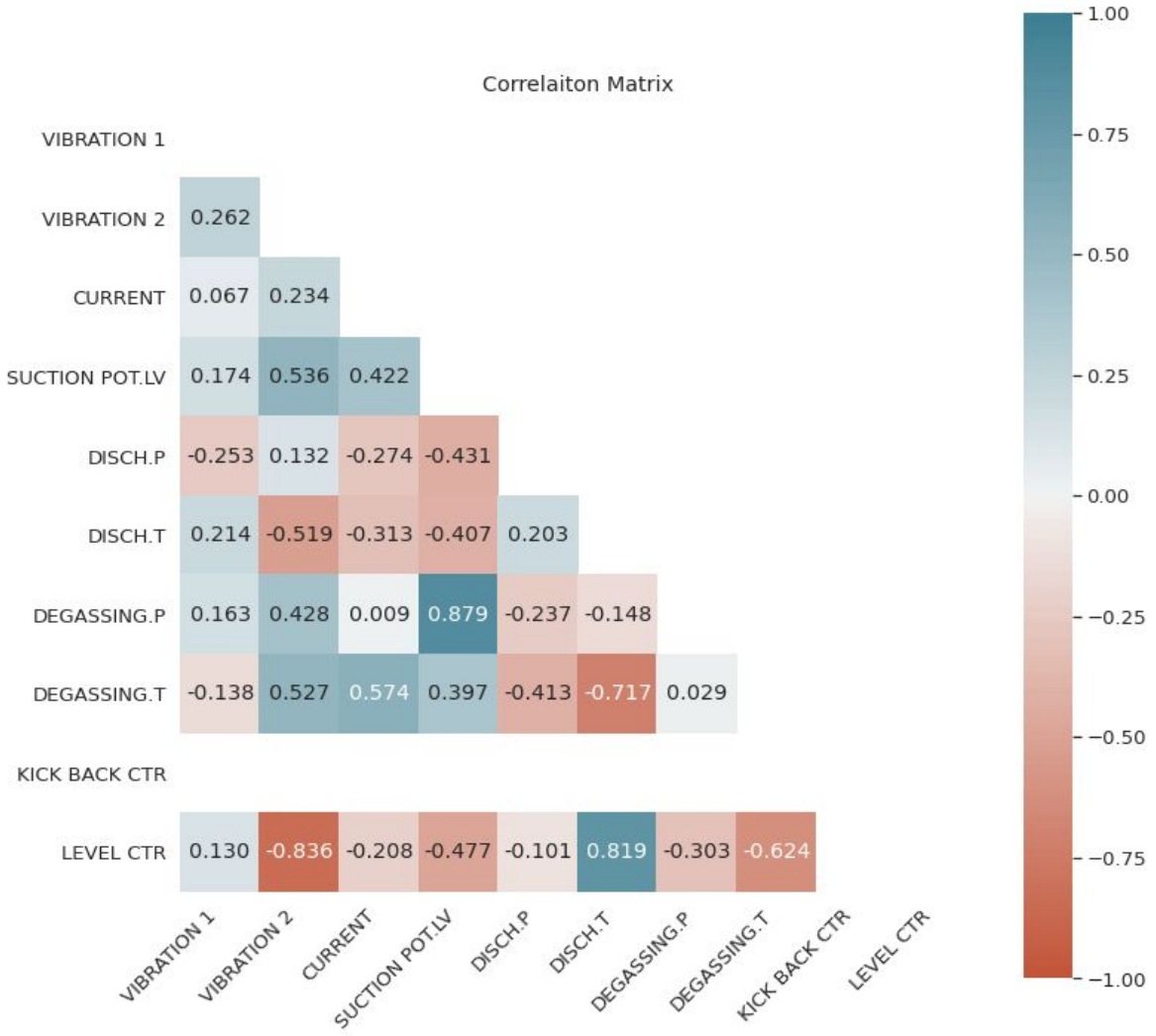

```
axis_labels = ['VIBRATION 1', 'VIBRATION 2', 'CURRENT','SUCTION
POT.LV','DISCH.P', 'DISCH.T', 'DEGASSING.P', 'DEGASSING.T', 'KICK BACK
CTR','LEVEL CTR'] # labels for axis

mask = np.zeros_like(correlation_matrices[0,300,:,:])
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
  f, ax = plt.subplots(figsize=(12, 12))
  ax = sns.heatmap(correlation_matrices[0,300,:,:],vmin=-1, vmax=1,
center=0,annot=True, fmt=".3f", mask=mask,
cmap=sns.diverging_palette(20,220,n=200),square=True,xticklabels=axis_labels,
yticklabels=axis_labels)

ax.set_xticklabels(ax.get_xticklabels(),rotation=45,horizontalalignment='right');
  ax.set_title('Correlation Matrix');
```

위는 Correlation Matrix를 시각화하기 위한 코드이다. seaborn 라이브러리를 이용하였으며, 대칭행렬이 되는 Correlation Matrix의 특성상 Lower Triangle 부분만을 표시하도록 하였다.

Kickback Control의 경우 값이 0이어서 흰색의 빈 값으로 표시되었다.



위의 코드 중 그래프를 그리는 데 사용되는 주요 부분은 아래의 부분이다.

```
ax = sns.heatmap(correlation_matrices[0,300,:,:],vmin=-1, vmax=1,
center=0,annot=True, fmt=".3f", mask=mask,
cmap=sns.diverging_palette(20,220,n=200),square=True,xticklabels=axis_labels,
yticklabels=axis_labels)
```

첫 번째 인수인 `correlation_matrices[0,300,:,:]`는 이 Heatmap그래프의 대상이 되는 2D Matrix가 `correlation_matrices`의 3번째와 4번째 차원으로 이루어진 행렬이라는 것을 지정한다. 두번째와 세번째 인수인 `vmin=-1, vmax=1` 은 해당 Heatmap내부에 들어갈 값의 범위가 -1에서 1사이라는 것을 이야기하고, 네번째 인수인 `center=0` 은 Heat Level의 중앙이 0이라는 것을 알려준다. 다섯번째 인수인 `annot=True` 는 행렬의 값들을 그래프 상에 나타내어준다는 것을 뜻하며, 여섯번째 인수인 `fmt=".3f"` 는 소숫점 아래 3자리까지 표시한다는 것을, 일곱번째 인수인 `mask=mask` 는 이 부분이 아닌 윗 부분에서 정의한

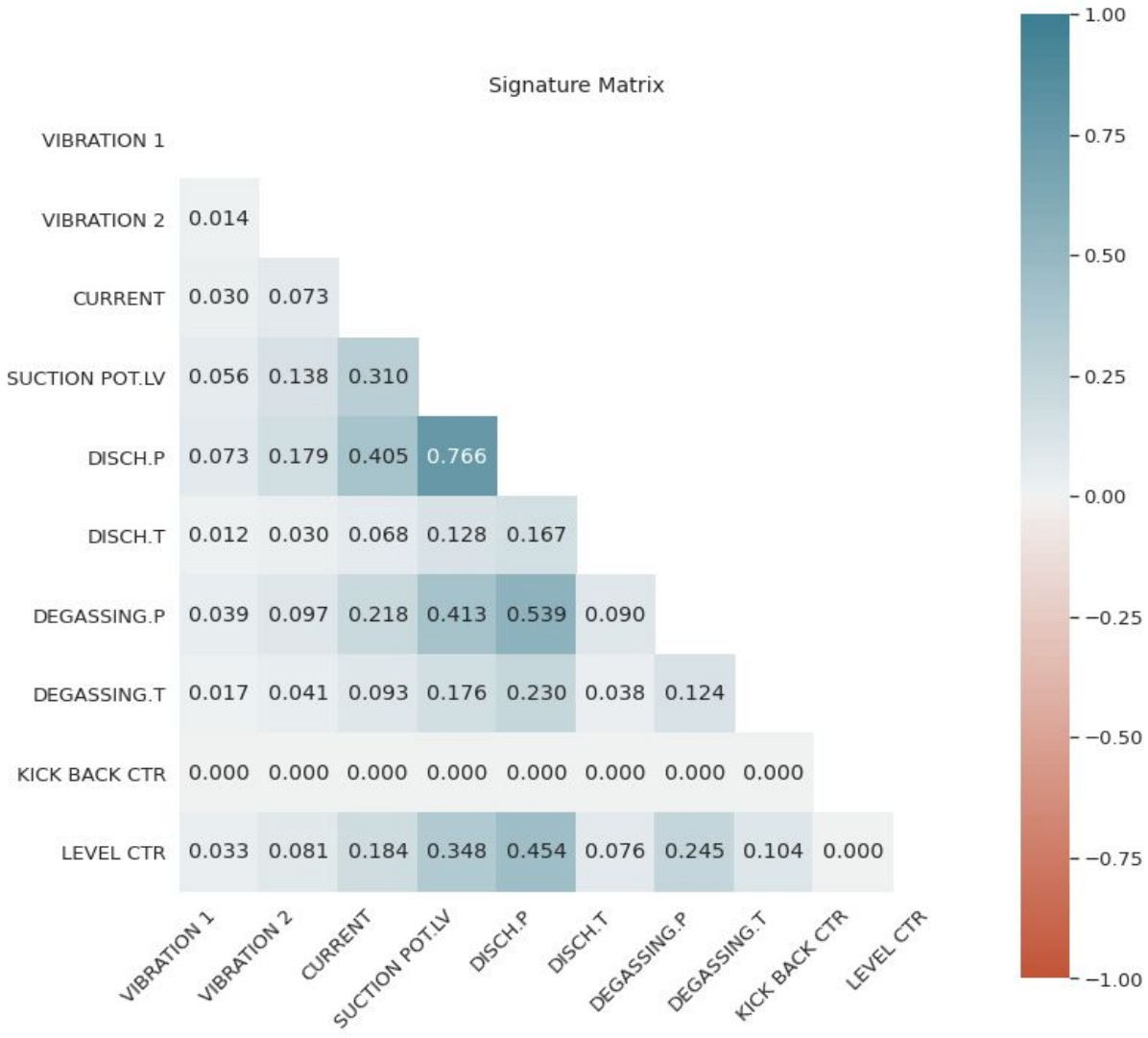
Mask값을 이용해서 Masking을 한다는 것을 의미한다. 여덟번째 인수인 `cmap=sns.diverging_palette(20,220,n=200)`은 Color Map을 의미하며 seaborn라이브러리에 내장된 palette를 사용한다는 것을 말한다. 아홉번째 인수인 `square=True`는 사각형의 Heatmap을 생성한다는 것을 의미하고 열번째와 열한번째 인수인 `xticklabels=axis_labels`, `yticklabels=axis_labels` 는 각각 x축과 y축의 레이블을 지정한다.

```
axis_labels = ['VIBRATION 1', 'VIBRATION 2', 'CURRENT','SUCTION
POT.LV','DISCH.P', 'DISCH.T', 'DEGASSING.P', 'DEGASSING.T', 'KICK BACK
CTR','LEVEL CTR'] # labels for axis

mask = np.zeros_like(signature_matrices[0,300,:,:])
mask[np.triu_indices_from(mask)] = True
with sns.axes_style("white"):
  f, ax = plt.subplots(figsize=(12, 12))
  ax = sns.heatmap(signature_matrices[0,300,:,:],vmin=-1, vmax=1,
center=0,annot=True, fmt=".3f", mask=mask,
cmap=sns.diverging_palette(20,220,n=200),square=True,xticklabels=axis_labels,
yticklabels=axis_labels)

ax.set_xticklabels(ax.get_xticklabels(),rotation=45,horizontalalignment='right');
  ax.set_title('Signature Matrix');
```

위는 Signature Matrix를 시각화 하기 위한 코드이다. 위의 Correlation Matrix 시각화 코드와 동일하며, 대상만 Signature Matrix라고 생각하면 된다.



4.3.4. 모델링 Modelling

```
def cnn_encoder_layer(data, filter_layer, strides):
    result = tf.nn.conv2d(
        input=data,
        filter=filter_layer,
        strides=strides,
        padding="SAME")
    return tf.nn.selu(result)

def cnn_encoder(data):

    filter1 = tensor_variable([3, 3, 3, 32], "filter1")
    strides1 = (1, 1, 1, 1)
```

```
    cnn1_out = cnn_encoder_layer(data, filter1, strides1)

    filter2 = tensor_variable([3, 3, 32, 64], "filter2")
    strides2 = (1, 2, 2, 1)
    cnn2_out = cnn_encoder_layer(cnn1_out, filter2, strides2)

    filter3 = tensor_variable([2, 2, 64, 128], "filter3")
    strides3 = (1, 2, 2, 1)
    cnn3_out = cnn_encoder_layer(cnn2_out, filter3, strides3)

    return cnn1_out, cnn2_out, cnn3_out
```

`cnn_encoder_layer(data, filter_layer, strides)`는 Encoder로 사용되는 2D Convolution 레이어를 정의하는 함수이다. 이 함수는 Convolution을 이용하는만큼 Filter와 Stride, Padding을 지정해주어야 한다. `cnn_encoder_layer(data, filter_layer, strides)` 부분에서는 Convolution 레이어의 기본 아키텍쳐만을 지정해주었으며, 실제 사용되는 Filter, Stride, Padding의 경우 `cnn_encoder(data)`에서 데이터에 맞게 지정해준다. 이 함수에서 이용된 Activation Function인 selu 는 Scaled Exponential Linear Unit으로, 미리 정해져있는 constant 값인 $\lambda(\simeq 1.0507)$와 $\alpha(\simeq 1.6733)$에 대해 $x \geq 0$ 이면 $f(x) = \lambda \cdot x$, $x < 0$ 이면 $f(x) = \lambda \cdot \alpha \cdot (e^x - 1)$ 이다. 는 Convolution Layer의 갯수, Output Size에 맞는 Filter의 형태와 Stride를 지정해주었다. 본 연구에서 사용된 Convolution Layer는 3개로, 각각 사이즈가 10*10*32, 5*5*64, 3*3*128 인 Output을 생성하도록 되어있다.

```
def tensor_variable(shape, name):
    variable = tf.Variable(tf.zeros(shape), name=name)
    variable = tf.compat.v1.get_variable(name, shape=shape,
initializer=tf.contrib.layers.xavier_initializer())
    return variable
```

위 함수는 레이어들을 초기화 하는 함수이다.

```
def cnn_lstm_attention_layer(input_data, layer_number):
    convlstm_layer = tf.contrib.rnn.ConvLSTMCell(
        conv_ndims=2,
        input_shape=[input_data.shape[2], input_data.shape[3],
input_data.shape[4]],
        output_channels=input_data.shape[-1],
        kernel_shape=[2, 2],
        use_bias=True,
        skip_connection=False,
```

```
        forget_bias=1.0,
        initializers=None,
        name="conv_lstm_cell" + str(layer_number))


    outputs, state = tf.nn.dynamic_rnn(convlstm_layer, input_data,
dtype=input_data.dtype)

    # attention based on inner-product between feature representation of last
step and other steps
    attention_w = []
    for k in range(step_max):
        attention_w.append(tf.reduce_sum(tf.multiply(outputs[0][k],
outputs[0][-1])) / step_max)
    attention_w = tf.reshape(tf.nn.softmax(tf.stack(attention_w)), [1, step_max])

    outputs = tf.reshape(outputs[0], [step_max, -1])
    outputs = tf.matmul(attention_w, outputs)
    outputs = tf.reshape(outputs, [1, input_data.shape[2], input_data.shape[3],
input_data.shape[4]])

    return outputs, attention_w
```

`cnn_lstm_attention_layer(input_data, layer_number)`함수는 LSTM 레이어의 아키텍쳐를 정의하는 부분이다. 기본적인 LSTM 레이어가 아닌 Attention(`attention_w`)을 도입한 레이어라는 것을 알 수 있다.

```
def cnn_decoder_layer(conv_lstm_out_c, filter, output_shape, strides):

    deconv = tf.nn.conv2d_transpose(
        value=conv_lstm_out_c,
        filter=filter,
        strides=strides,
        output_shape=output_shape,
        padding="SAME")
    deconv = tf.nn.selu(deconv)
    return deconv


def cnn_decoder(lstm1_out, lstm2_out, lstm3_out):

    d_filter3 = tensor_variable([2, 2, 64, 128], "d_filter3")
    dec3 = cnn_decoder_layer(lstm3_out, d_filter3, [1, 5, 5, 64], (1, 2, 2, 1))
```

```
    dec3_concat = tf.concat([dec3, lstm2_out], axis=3)

    d_filter2 = tensor_variable([3, 3, 32, 128], "d_filter2")
    dec2 = cnn_decoder_layer(dec3_concat, d_filter2, [1, 10, 10, 32], (1, 2, 2,
1))
    dec2_concat = tf.concat([dec2, lstm1_out], axis=3)

    d_filter1 = tensor_variable([3, 3, 3, 64], "d_filter1")
    dec1 = cnn_decoder_layer(dec2_concat, d_filter1, [1, 10, 10, 3], (1, 1, 1,
1))

    return dec1
```

`cnn_decoder_layer(conv_lstm_out_c, filter, output_shape, strides)`는 이전에 정의했던 Couvolution Encoder부분을 역으로 되돌리는 과정을 수행하는 레이어를 정의하는 부분이고, 이에 맞게 사이즈를 수정해주는 부분은 `cnn_decoder(lstm1_out, lstm2_out, lstm3_out)`라고 할 수 있다.

4.3.5. 모델 컴파일링 및 훈련 Compiling and Fitting

```
def main():
    # Read dataset from file
    matrix_data_path = train_data_path + "train.npy"
    matrix_gt_1 = np.load(matrix_data_path)

    sess = tf.Session()
    # data 사이즈 수정 후 수정
    data_input = tf.compat.v1.placeholder(tf.float32, [step_max, 10, 10, 3])

    # cnn encoder - 3개짜리 레이어로 수정
    conv1_out, conv2_out, conv3_out = cnn_encoder(data_input)

    conv1_out = tf.reshape(conv1_out, [-1, 5, 10, 10, 32])
    conv2_out = tf.reshape(conv2_out, [-1, 5, 5, 5, 64])
    conv3_out = tf.reshape(conv3_out, [-1, 5, 3, 3, 128])

    # lstm with attention
    conv1_lstm_attention_out, atten_weight_1 =
cnn_lstm_attention_layer(conv1_out, 1)
    conv2_lstm_attention_out, atten_weight_2 =
cnn_lstm_attention_layer(conv2_out, 2)
    conv3_lstm_attention_out, atten_weight_3 =
cnn_lstm_attention_layer(conv3_out, 3)
```

```
    # cnn decoder
    deconv_out = cnn_decoder(conv1_lstm_attention_out, conv2_lstm_attention_out,
conv3_lstm_attention_out)
    # loss function: reconstruction error of last step matrix
    loss = tf.reduce_mean(tf.square(data_input[-1] - deconv_out))
    optimizer =
tf.compat.v1.train.AdamOptimizer(learning_rate=learning_rate).minimize(loss)

    # variable initialization
    init = tf.global_variables_initializer()
    sess.run(init)

    training_loss =[]
    test_loss =[]
    # training
    for idx in range(train_start_id, train_end_id):
        matrix_gt = matrix_gt_1[idx - train_start_id]
        feed_dict = {data_input: np.asarray(matrix_gt)}
        a, loss_value = sess.run([optimizer, loss], feed_dict)
        training_loss.append(loss_value)
        print("mse of last train data: " + str(loss_value))


    # test
    # Read the data from test file.
    matrix_data_path = test_data_path + "test.npy"
    matrix_gt_1 = np.load(matrix_data_path)
    result_all = []
    for idx in range(test_start_id, test_end_id):
        matrix_gt = matrix_gt_1[idx - test_start_id]
        feed_dict = {data_input: np.asarray(matrix_gt)}
        result, loss_value = sess.run([deconv_out, loss], feed_dict)
        result_all.append(result)
        test_loss.append(loss_value)
        print("mse of last test data: " + str(loss_value))

    plt.subplots(figsize=(20,5))
    plt.plot(training_loss, label="Training Loss")
    plt.plot(test_loss, label="Test Loss")
    plt.legend()

    # Write the reconstructed matrix to the file
    reconstructed_path = reconstructed_data_path
    if not os.path.exists(reconstructed_path):
```

```
        os.makedirs(reconstructed_path)
    reconstructed_path = reconstructed_path + "test_reconstructed.npy"

  #reshape
    result_all = np.asarray(result_all).reshape((-1, 10, 10, 3))
    print(result_all.shape)
    np.save(reconstructed_path, result_all)

if __name__ == '__main__':
    main()
```

모델링 부분에서 코딩한 conv-LSTM 모델을 실제로 구동시키는 부분으로, 데이터 전처리 부분에서 생성한 Training Set과 Test Set을 이용하여 학습과 검증을 진행한다. 이 때 Test Set을 이용했을 때의 Output인 tset_reconstruction 행렬은 따로 Anomaly Score 계산을 위해 .npy 파일로 저장해둔다.

4.3.6. 이상 정도 계산 Anomaly Score Calculation

```
valid_raw_data = pd.read_csv(valid_data_path, header=None).T
valid_len=valid_raw_data.shape[1]

# score initialization
valid_anomaly_score = np.zeros((valid_len, 1))
test_anomaly_score = np.zeros((valid_len, 1))

# load the data from file
test_data_path = test_data_path
reconstructed_data_path = reconstructed_data_path
test_data_path = os.path.join(test_data_path, "test.npy")
reconstructed_data_path = os.path.join(reconstructed_data_path,
"test_reconstructed.npy")
test_data = np.load(test_data_path)
test_data = test_data[:, -1, ...]  # only compare the last matrix with the
reconstructed data
reconstructed_data = np.load(reconstructed_data_path)
print("The shape of test data is {}".format(test_data.shape))
print("The shape of reconstructed data is {}".format(reconstructed_data.shape))
```

이상 상태를 계산하기 위하여 Test 데이터 및 conv-LSTM모델을 거쳐서 나온 Reconstruction 데이터를 불러온다. 각각의 Shape을 비교하여 Reconstruction 데이터가 잘 생성되었는지 확인하는 과정을 거친다.

```
for i in range(valid_len):
  error = np.square(np.subtract(test_data[i, ..., 0], reconstructed_data[i, ...,
0]))
  num_anom = len(np.where(error >threshold))
  valid_anomaly_score[i] = num_anom

max_valid_anom = np.max(valid_anomaly_score)
threshold = max_valid_anom * alpha

print("Max valid anom is %.2f" % max_valid_anom)
print("Threshold is %.2f" % threshold)
```

위의 구간에서는 validation부분의 데이터를 이용하여 Threshold값을 계산한다. 이 값은 `max_valid_anom`과 `alpha`값을 곱하여 만들어진다.

```
# compute the anomaly score in the test data.
for i in range(test_end_id - test_start_id-valid_len):
  error = np.square(np.subtract(test_data[i, ..., 0], reconstructed_data[i, ...,
0]))
  num_anom = len(np.where(error > threshold))
  test_anomaly_score[i - valid_len] = num_anom

# plot anomaly score curve and identification result
anomaly_pos = np.zeros(5)
root_cause_gt = np.zeros((5, 3))
anomaly_span = [10, 30, 90]

# Read the test_anomaly.csv, each line behalf of an anomaly, the first is the
position, the next three number is the
# root cause.
root_cause_f = open("../content/data/test_anomaly.csv", "r")

root_cause_gt = np.loadtxt(root_cause_f, delimiter=",", dtype=np.int32)
anomaly_pos = root_cause_gt[:, 0]
anomaly_pos = [(anomaly_pos[i]/gap_time-test_start_id-anomaly_span[i %
3]/gap_time) for i in range(5)]
for i in range(5):
  root_cause_gt[i][0] = anomaly_pos[i]
fig, axes = plt.subplots(figsize=(20,5))
test_num = test_end_id - test_start_id
plt.xticks(fontsize = 25)
plt.ylim((0, 100))
plt.yticks(np.arange(0, 101, 20), fontsize = 25)
```

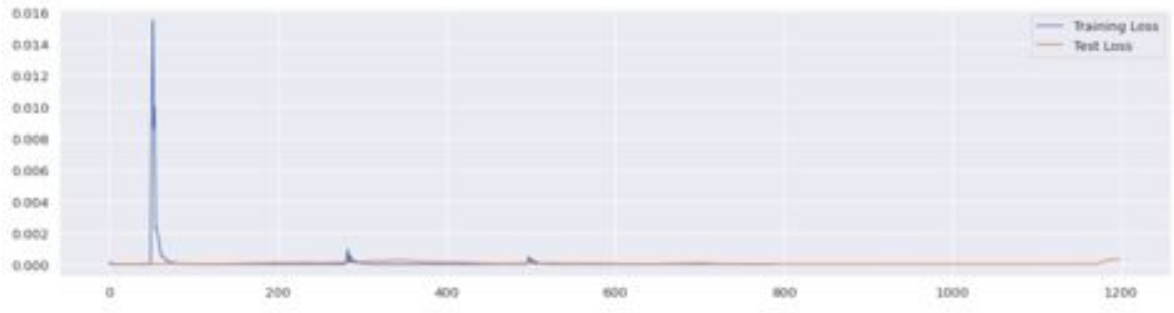
```
plt.plot(test_anomaly_score, 'b', linewidth = 2)
threshold = np.full((test_num), max_valid_anom * alpha)
axes.plot(threshold, color = 'black', linestyle = '--',linewidth = 2)
for k in range(len(anomaly_pos)):
  axes.axvspan(anomaly_pos[k], anomaly_pos[k] + anomaly_span[k%3]/gap_time,
color='red', linewidth=2)
labels = [' ', '0e3', '2e3', '4e3', '6e3', '8e3', '2000']
axes.set_xticklabels(labels, rotation = 25, fontsize = 20)
plt.xlabel('Test Time', fontsize = 25)
plt.ylabel('Anomaly Score', fontsize = 25)
axes.spines['right'].set_visible(False)
axes.spines['top'].set_visible(False)
axes.yaxis.set_ticks_position('left')
axes.xaxis.set_ticks_position('bottom')
fig.subplots_adjust(bottom=0.25)
fig.subplots_adjust(left=0.25)
plt.show()
```

위의 부분에서는 연산했던 Anomaly Score를 Timeseries 별로 출력해주고, 출력된 Anomaly Score값과 사람이 Labeling한 Anomaly 구간을 비교해서 보여준다.

5. 결과 Result

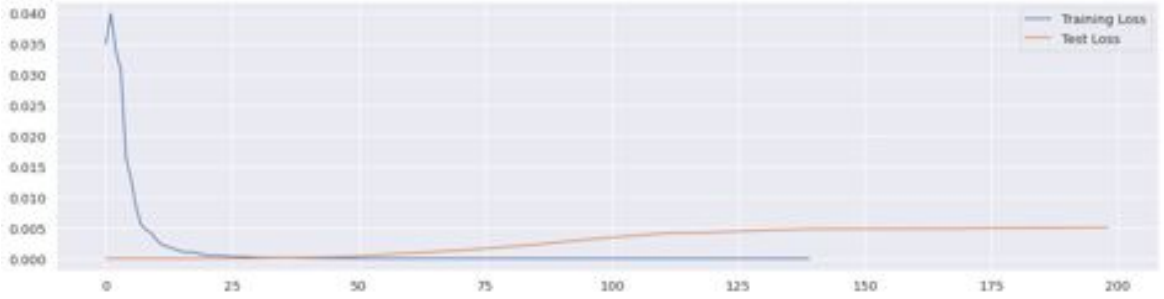
본 연구에서는 Pump와 그에 부착된 여러 센서를 이용하여 Anomaly Detection을 수행하는 과정을 보였다. Pump에 부착된 센서는 총 10개로, 각각 진동과 압력, 온도, 유량, 수위, 밸브 오프닝 등을 측정하였다. 이 센서들에 대한 이상 예측을 따로 진행하지 않고 해당 센서들의 값을 종합하여 이상 예측을 수행하였다. 아래는 해당 코드를 이용하여 학습을 수행한 결과이다.

(1) 데이터셋 Scaling & 정상 상태 데이터 추출 전



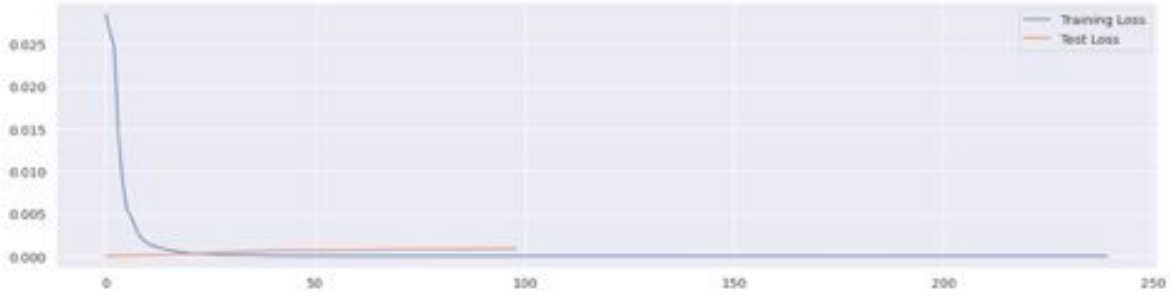
위는 가지고 있던 데이터셋 전체를 Raw상태로 입력 데이터셋으로 넣은 경우의 학습 결과로, 초기의 Training Loss가 거의 0에 가까운 비정상적인 모습을 보였다. 그 뿐만 아니라 Optimizer의 존재에도 불구하고 중간중간 국소적인 Peak가 발생하는 것을 볼 수 있었다.

(2) 데이터셋 전체 Scaling & 정상 상태 데이터 추출 후



위에서 제기된 문제점을 해결하기 위하여 원인을 분석한 결과, LSTM 모델이 기본적으로 Prediction Model로 사용되기 때문에 이 모델을 이용하기 위해서는 이상 상태 데이터를 제외한 정상 상태 데이터만을 입력값으로 넣어주어야 한다는 것을 알았다. 이에, 데이터셋 내용을 분석하고 이들 중 정상 데이터라고 여겨지는 부분만을 추출하여 재학습시켰다. 그 결과, 위에서 제기된 문제처럼 초반에 Training Loss 값이 0이 되는 부분이나 중간중간 국소적인 Peak가 생기는 부분이 사라졌다. 그러나 문제가 완전히 해결되지는 않았는데, 초반의 Training Loss 값이 약간 올라가는 부분이 아직 남아있다는 점과 Test Loss 값이 점점 높아지는 것이 관찰되었다는 점 때문이었다.

(3) 데이터셋 Column별 Scaling & 정상 상태 데이터 추출 후



위에서 제기된 문제점을 해결하기 위하여 다시 한 번 원인을 분석한 결과, 데이터셋의 Scaling 및 Training Data의 부족이 문제가 될 것이라고 생각하였다. 이에, 데이터셋 전체에 대해서 Min Max Scaling을 진행했던 이전 시도들과는 달리 데이터셋 컬럼별로 각각 Min Max Scaling을 진행하여 Convolution 시 각 센서 데이터의 크기 자체보다는 변동 폭이 해당 Convolution에 영향을 미치도록 하였다. 해당 과정을 수행한 것이 Test Loss 값이 올라가는 것 자체에는 크게 영향을 주지 않았으나 이전에 비해 낮은 값을 유지(<0.001)했다는 점에서 긍정적이었고, 초반부에서 Training Loss값이 올라가는 구간이 사라졌다.

6. 결론 및 제언 Conclusion and Discussion

6.1. 결론

6.1.1. Training Result

맨 처음에는 학습 결과가 원하는 대로 이루어지지 않아서 원인을 분석한 결과 모델에 필요한 데이터셋의 조건을 만족시키지 못했다는 것을 알게되었다. 이에 데이터셋을 조정하고 모델의 파라미터를 바꾸는 등 여러 시도를 반복하면서 문제를 해결하려고 노력하였다. 이 결과, 모델 자체의 변경보다는 데이터셋을 어떻게 만드느냐가 조금 더 주된 문제라는 것을 알게 되었고 좋은 데이터셋을 만드는 것이 시급한 문제라는 것을 실감하였다.

6.1.2. Correlation Matrix와 Signature Matrix

Correlation Matrix와 Signature Matrix를 비교하였으나 Correlation Matrix의 경우 제대로 학습이 이루어지지 않았다는 단점이 있었다. 처음에는 Signature Matrix보다 Correlation Matrix가 훨씬 더 적합하지 않을까 하는 생각이 있었으나 실제로 학습에 사용해보니 Correlation Matrix는 센서 데이터의 변화에 매우 민감하게 반응하여, Normal set과 Anomaly set을 구분하기가 쉽지 않았다. 결국 Normal Dataset 내에서 비슷한 변화를 보이는 Signature Matrix가 좀 더 적합하다는 결론을 얻었다.

6.2. 제언

본 연구에서 문제점으로 지적되었던 부분은 모델의 정확도 및 데이터의 부족이 가장 컸다. 사실상 두 가지는 서로 분리할 수 없는 문제로 데이터 부족을 개선해야만 모델의 정확도도 높일 수 있다. 데이터의 부족 문제는 많은 공정에 센서 데이터 수득 및 저장이 보편화되면 해결될 문제이나 짧은 시간 안에 해결할 수 있는 문제는 아니다. 그런 의미에서 해당 문제를 해결할 수 있는 대안적인 방법이 필요하다는 것을 실감했고, 이를 해결하기 위해 기존 데이터를 Regression하여 모델링하여 데이터를 합성하는 방법(Zhang. et al., 2019)이나 GAN을 이용하여 데이터의 형태를 모방하는 방법을 이용하는 방법(Smith and Smith, 2020)을 활용하는 방법을 검토중에 있다.

코드를 작성하면서 다른 대안을 검토해보고 싶었던 부분들도 있었다. 먼저 Normalization과정에서 minmax normalization을 이용하는 방법이 아닌 Anomaly를 잡아내는 데 더 효과적이라고 알려진 z-index normalization을 이용하는 방법을 이용하여 결과를 비교해보는 방식을 이용해보고자 한다. 다음으로 2D Signature Matrix를 만드는 과정에서

사용할 수 있는 다른 형태의 Correlation 수식들을 찾아서 적용해보고 그 결과를 관찰해보고자 한다.

또한, 본 방법론을 수행해 본 결과 여러 센서를 전부 인풋값으로 받는 Anomaly Detection만 하는 것보다는 각 센서들에 대한 Anomaly Detection을 수행한 뒤 Multivariate Detection을 보조로 사용할 수 있는 프레임워크를 개발하는 방법을 고안해보는 것이 좋을 것으로 생각되었다.

이에 더해, 본 코드는 참고 문헌을 따라 작성하기 시작한 것이라 Tensorflow 1버전으로 구현되어있으나 1버전은 현재 더이상 지원하지 않기 때문에 새로 2버전 코드로 업데이트하는 과정이 필요할 것으로 보인다.

6.3. 의의

본 연구는 여러 종류의 센서 데이터 인풋을 기반으로 공정 장비의 이상을 예측하는데 의의가 있다. 공정의 자동화가 큰 흐름으로 자리잡은 지금, 공정의 센서 데이터들을 기반으로 자동화 시스템을 구현할 수 있는 여러 방법론들의 필요성이 대두되고 있다. 본 연구는 그 중에서도 여러 센서의 연관도를 기반으로 한 다변수 기반 예지보전 방법론을 제안하고 해당 방법론을 딥러닝 모델과 함께 구현하는 과정을 보였다. 모델의 정확도 부족 및 데이터셋 불충분 등의 문제가 있지만 후속 연구를 통하여 해당 문제점들을 차차 보완해나갈 예정이다.

끝으로, 본 연구를 지속적으로 해나갈 수 있게 해 주신 IoT·인공지능·빅데이터의 실무응용 연구 1 수업, 수업을 담당해주신 강현구 교수님, TA활동을 해주신 최홍석 조교님, 리뷰해주신 위원분들, 같이 함께한 다른 학생 분들께도 감사드린다.

7. 참고문헌 Reference

1) Zhang, C., Song, D., Chen, Y., Feng, X., Lumezanu, C., Cheng, W., Ni, J., Zong, B., Chen, H., & Chawla, N. V. (2019). A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01), 1409-1416.
2) Shi, X., Chen, Z., Wang, H., Yeung, D. Y., Wong, W. K., & Woo, W. C. (2015). Convolutional LSTM network: A machine learning approach for precipitation nowcasting. Advances in neural information processing systems, 28, 802-810.

3) Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G. (2016). LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection.
4) Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (1997), 1735–1780.
5) Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015, April). Long short term memory networks for anomaly detection in time series. In *Proceedings* (Vol. 89, pp. 89-94). Presses universitaires de Louvain.
6) Smith, K. E., & Smith, A. O. (2020). Conditional GAN for Timeseries Generation. *arXiv preprint arXiv:2006.16477*.
7) Zhao, H., Liu, H., Hu, W., & Yan, X. (2018). Anomaly detection and fault analysis of wind turbine components based on deep learning network. *Renewable energy*, *127*, 825-834.