Multi-head CNN–RNN for multi-time series anomaly detection : An industrial case study

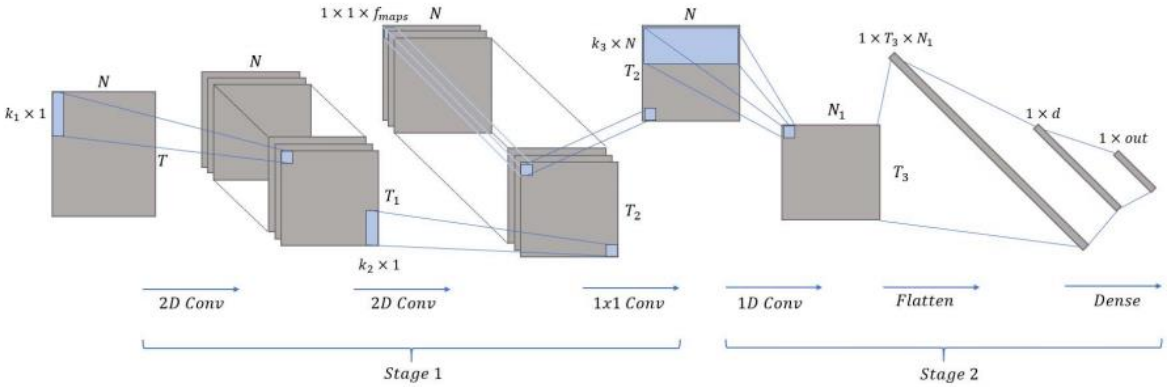
- Reading the paper in terms of Anomaly detection -

1. Introduction

시계열에서 anomaly detection은 이전부터 지속적으로 연구가 이루어진 분야입니다. 제조업에서 센서 시계열 데이터는 보통 기계의 상태나 공정의 상태를 수집하는 대표적인 데이터라고 할 수 있습니다. 특히, 최근 몇 년 동안 Industry 4.0과 사물 인터넷의 발전으로 하나의 기계에 여러 종류의 multi-sensor가 부착되면서 다양한 방면에서 모니터링이 가능해졌습니다.

하지만 다양한 형태의 센서가 부착이 되면서, 이 시계열 데이터들을 어떻게 해석하고 anomaly detection에 활용할 것인가는 어려운 문제입니다. 복잡한 기계의 경우, 다른 성격의 센서를 가지고 있기 때문에 서로 다른 형태로 데이터를 측정됩니다. 이런 상태에서 의미 있는 특징을 추출하거나 분석에 사용할 수 잇는 상태로 데이터를 전처리하는 것은 쉬운 일이 아닙니다.

위와 같은 문제에서 딥러닝은 데이터의 별도 전처리 없이 데이터의 특징을 추출한다는 점에서 굉장히 강력한 도구로 평가받습니다. 딥러닝 방법론 중에서도 합성곱 신경망(Convolution neural network, CNN)은 1D convolution을 활용하여 시계열 데이터의 특징을 추출해내는 강력한 방법론으로 알려져 있습니다. 이와 더불어 순환 신경망(Recurrent neural network, RNN)은 시간의 흐름을 가진 데이터에서 시계열 예측에 좋은 성능을 발휘하고 있습니다. 본 논문은 이러한 CNN과 RNN의 강점을 결합하여 다변량 센서 시계열 데이터를 이용해 anomaly detection을 하고자 합니다.

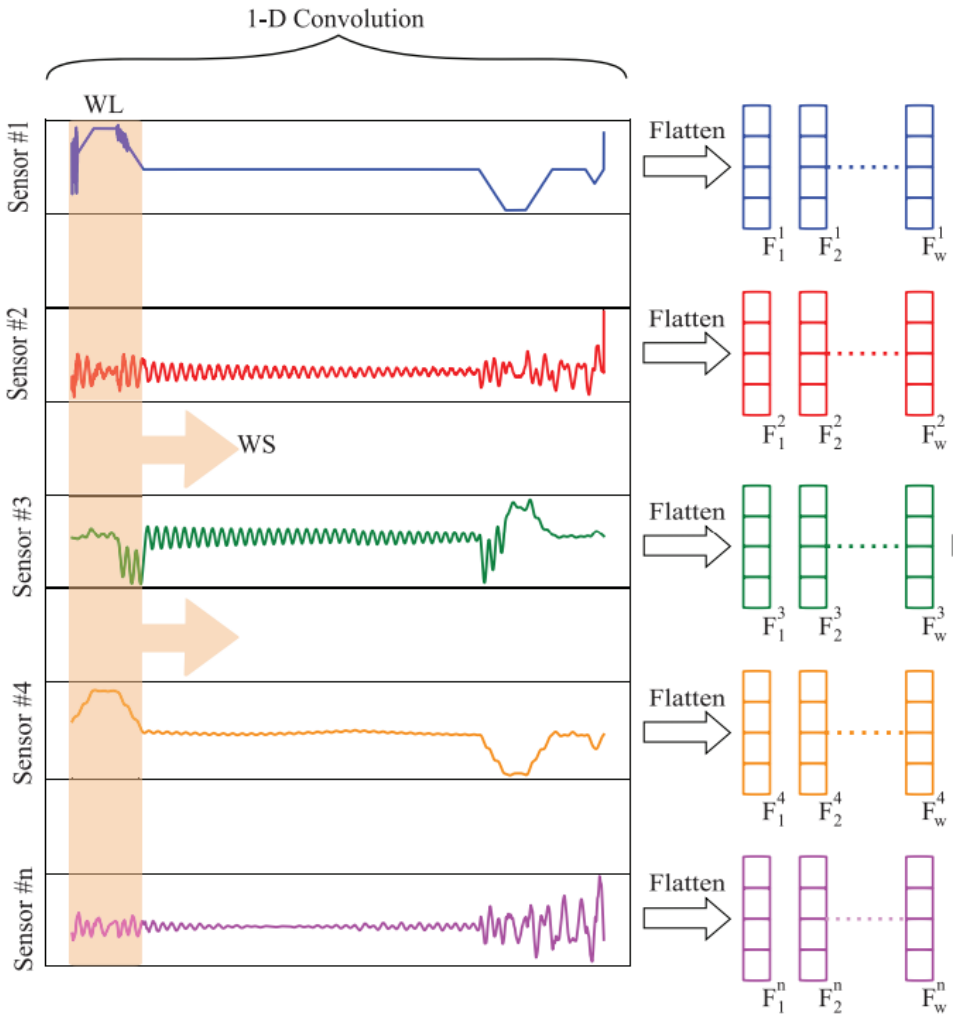


[그림1] 일반적인 다변량 시계열 데이터의 1D-convolution CNN 구조

특히 본 논문은 다변량 시계열 데이터를 처리하기 위한 새로운 방법론을 제시하고 있습니다. 기존의 방법론들은 시간의 흐름을 살리기 위해 1D convolution을 사용하지만, multi-sensor 데이터를 하나의 데이터로 처리하여 특징을 추출합니다. 위의 그림처럼 행을 시간의 흐름으로, 컬럼을 각 sensor로 구축하여 convolution을 진행하는 형태입니다. 이러한 방식은 시간의 흐름을 살릴 수 있지만, 각 센서

데이터의 특징을 추출한다는 면에서는 효율적이지 않습니다.

따라서 센서별로 별도의 CNN을 생성하여 센서 하나의 특징을 온전하게 추출하자는 것이 이 논문의 핵심 아이디어라고 할 수 있습니다.



[그림2] Convolution head

위의 그림을 보면 바로 이전 그림과의 차이를 알 수 있습니다. 기존에는 다변량 시계열 데이터를 하나의 데이터프레임으로 재구축하여 convolution을 실행한 반면, 센서 별로 1D-conolution을 수행하여 특징을 추출하는 것입니다. 이러한 방식은 각 센서의 특징을 온전하게 추출해낼 수 있다는 장점을 가지게 됩니다.

본 논문에서는 각 센서 데이터에 독립적인 CNN을 convolution-head로 정의하고 있습니다. 이러한 방식은 위에서 언급한 각 센서 별로 특징을 추출할 수 있다는 장점도 있지만, 모델의 구조를 유연하게 가져갈 수 있다는 장점도 존재합니다. 즉, convolution-head를 추가/수정/제거를 통해 센서 데이터의 변화에 쉽게 적응시킬 수 있다는 점입니다.

2. Proposed Method

2.1 Sliding window

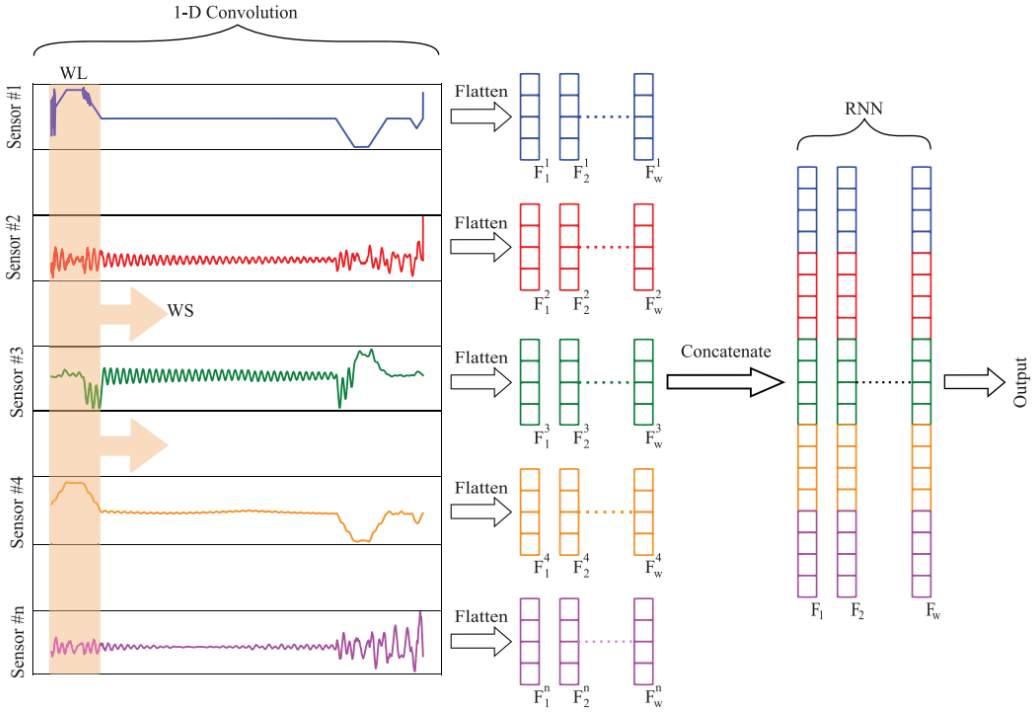
Multi-head convolution의 가장 큰 주요 특징은 sliding window에 따라 시계열을 처리한다는 점입니다. 시계열의 전체 시퀀스를 한 번에 처리하지 않고, sliding window를 사용하여 각 센서 데이터의 부분적인 특징을 더욱 효율적으로 처리할 수 있게 됩니다. 1D-convolution의 관점에서 window size는 filter size로 정의할 수 있게 되며, window step은 sliding을 수행할 때 몇 칸 식 전진하면서 sliding을 수행할 것인지를 정의하게 됩니다.

$$W_N = \frac{S_L - W_L}{W_S} + 1$$

각 센서 데이터를 sliding window로 진행할 때 S_L은 전체 데이터의 시간 시점, W_L은 window의 길이, W_S는 window step을 의미하고 W_N은 window의 개수를 의미합니다. 따라서 CNN의 입력데이터는 행의 개수가 W_N이고 열의 개수가 W_L인 행렬이 되게 됩니다.

2.2 RNN

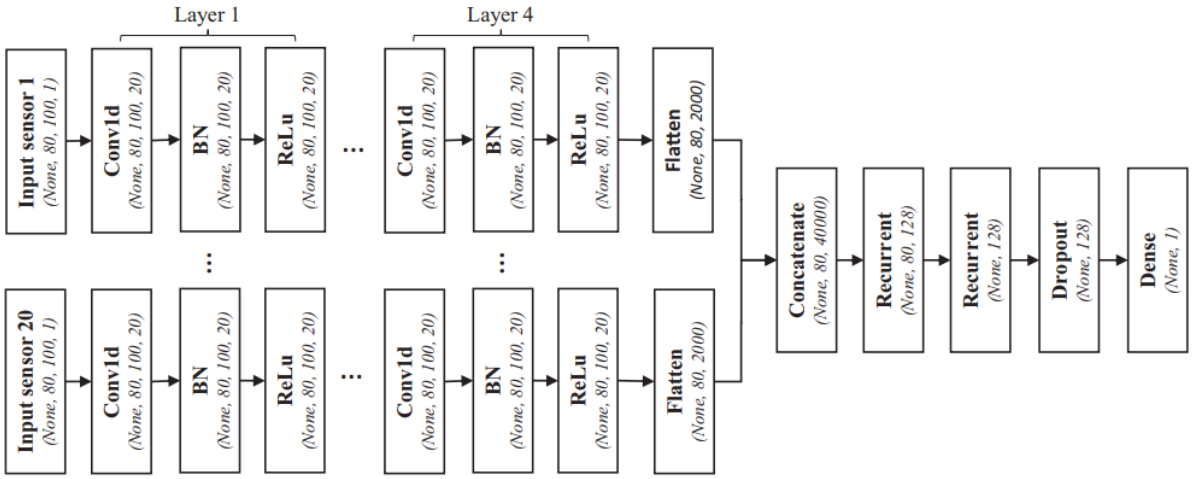
RNN은 과거 사건을 기억하는 기능을 갖춘 딥러닝 방법론 중 하나입니다. 본 모델 구조에서 RNN은 CNN으로 각 센서 데이터의 특징을 추출한 결과를 시간적 흐름에 따라 입력 받아 최종적으로 anomaly detection을 수행하는 역할입니다.



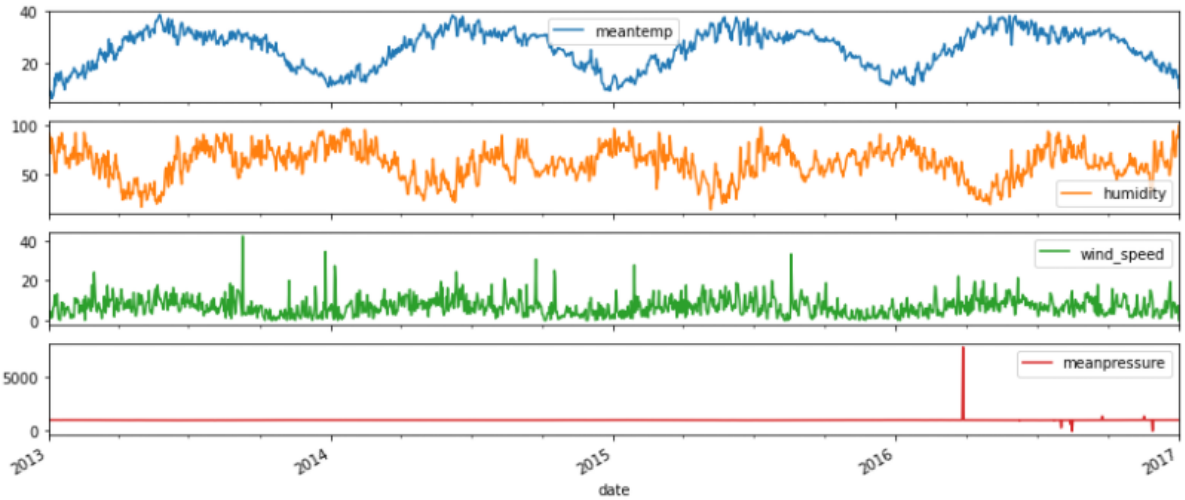
[그림3] RNN의 입력 데이터 구조

RNN의 입력 데이터는 각 센서 데이터의 같은 시점의 윈도우입니다. 즉 각 센서 데이터의 같은 시점의 윈도우 맵들은 concatenate가 되어 해당 시점을 대표하는 RNN의 입력 데이터가 됩니다. 이런 식으로 설계하게 되면 시간의 흐름을 잃지 않으면서 모든 센서 데이터의 특징을 RNN이 학습할 수 있게 되는 것입니다. RNN은 Multi-head convolution이 데이터의 각 센서 데이터 별로 처음부터 끝까지 특징을 추출하기 전까지 실행되지 않습니다. 각 센서 데이터의 CNN에서 모든 특징들을 추출하고 시간순에 따라 window 맵들을 재 구성하여 RNN을 위한 입력 데이터를 만들어야 하기 때문입니다.

4. Experiments(Code Review)



코드 구현을 위한 구조는 위의 그림과 같습니다. 먼저 독립적인 CNN을 센서 데이터 별로 생성한 후 단순이 concatenate를 수행하여 데이터를 재구축한 후, RNN으로 학습을 시키는 구조를 취하고 있습니다. 논문에서는 20개의 센서 데이터로 이루어진 엘리베이터 센서 데이터를 사용하였지만, 과제에서는 날씨 데이터를 사용하여 4개의 센서 데이터로 이루어진 데이터를 사용하였습니다.



(1) Multi-head CNN

Multi-head CNN의 구현을 위해 먼저 각 센서 데이터의 특징을 추출할 CNN의 모습입니다. 일반적인 CNN과 구조는 동일합니다.

```
class headCNN(nn.Module):
    def __init__(self):
        super(headCNN, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv1d(1, 20, kernel_size=3, stride=3, padding=0),
            nn.BatchNorm1d(20),
            nn.ReLU())

        self.conv2 = nn.Sequential(
            nn.Conv1d(20, 20, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(20),
            nn.ReLU(),
            nn.MaxPool1d(3, stride=3))

        self.conv3 = nn.Sequential(
            nn.Conv1d(20, 20, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(20),
            nn.ReLU(),
            nn.MaxPool1d(3,stride=3))

        self.fc = nn.Linear(20, 2)
        self.activation = nn.Sigmoid()

    def forward(self, x):

        x = x.view(x.shape[0], 1,-1)

        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)

        out = out.view(x.shape[0], out.size(1) * out.size(2))
        logit = self.fc(out)

        #logit = self.activation(logit)

        return logit
```

학습에는 각 시계열 데이터의 정상/비정상 학습을 위해 logit 노드까지 학습을 시키지만 RNN에 입력되는 feature는 conv3 계층을 통과한 feature이므로 forward_hook을 통해 feature를 추출합니다.

```
target_layer = "conv3"

feature = dict()
def forward_hook(module, input, output):

    feature['value'] = output.to(DEVICE)

    return None

hook = target_layer.register_forward_hook(forward_hook)
```

(2) RNN

(3) 결과

평가 지표로는 g_mean을 사용하였습니다. Geometric mean은 precision과 recall로 구성되어 있으며, threshold에 따라 계산된 precision과 recall의 연산값으로 평가됩니다. 높을수록 좋은 값을 의미합니다

$$g\_mean = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{TN+FP}}$$

비교 모델은 정상/비정상을 탐지하는 RNN모델을 다른 것으로 사용하였을 때 성능 변화와 CNN을 multi-head CNN과 일반적인 CNN(다변량 시계열을 하나의 데이터프레임)을 사용하였을 때 결과를 비교하였습니다.

	Multi-head Conv1d	**CNN**
RNN	**0.961**	0.922
LSTM	**0.980**	0.979

실험결과 LSTM이 RNN보다 좋은 결과를 보였고, 일반적인 CNN보다 Multi-head Conv1d를 사용하였을 때 더 결과가 좋다는 것을 관측할 수 있었습니다.