2022 Inha AI Challenge

12181912 김현수
12181901 고영호

목차

- Abstract
- Feature
- Prepare
- Model
- Test
- Q&A

```
./
├── README.md
├── prepare.ipynb
├── model.ipynb
├── test.ipynb
├── data
│   ├── turbine_data1.csv
│   ├── turbine_data2.csv
...
│   ├── turbine_data134.csv
│   ├── train_data42.csv
│   ├── dist_dict.pickle
```

목차

인하대학교
INHA UNIVERSITY

1. Abstract

python--3.7.13
pandas==1.3.5
numpy==1.21.5
torch==1.12.0
sklearn==1.0.2

목표 : 134개 터빈의 2일 동안의 Patv 값 예측

TurbID - 발전기 ID

Day - 날짜

Tmstamp - 시간

Wspd - 풍속

Wdir - [illegible]이 [illegible]보는 [illegible]도와 [illegible]제 [illegible] 방향 [illegible]도 차이

Etmp [illegible]외부

Itmp - [illegible]도

Ndir - 터빈이 바라보는 방향 각도

Pab - 터빈 당 3개의 날이 있으며 각각의 각도가 다름

Prtv - 무효전력 : 에너지원을 필요로 하지 않는 전력

Patv - 유효전력 : 실제로 터빈을 돌리는 일을 하는 전력

GRU + CNN

TurbID	x	y
1	3349.852	5939.232
2	3351.002	6416.647
3	3314.78	6892.184
4	3352.094	7366.142
5	3355.342	7841.202
6	3329.428	8340.798
7	3360.547	8816.238
8	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
10	[illegible]	[illegible]
11	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
14	3328.78	719.0482
15	3329.759	1194.288
16	3333.904	1658.257
17	3334.256	2140.554
18	3336.848	2615.588
19	3338.886	3090.449
20	3361.034	3566.919
21	3249.022	4043.586

인하대학교
INHA UNIVERSITY

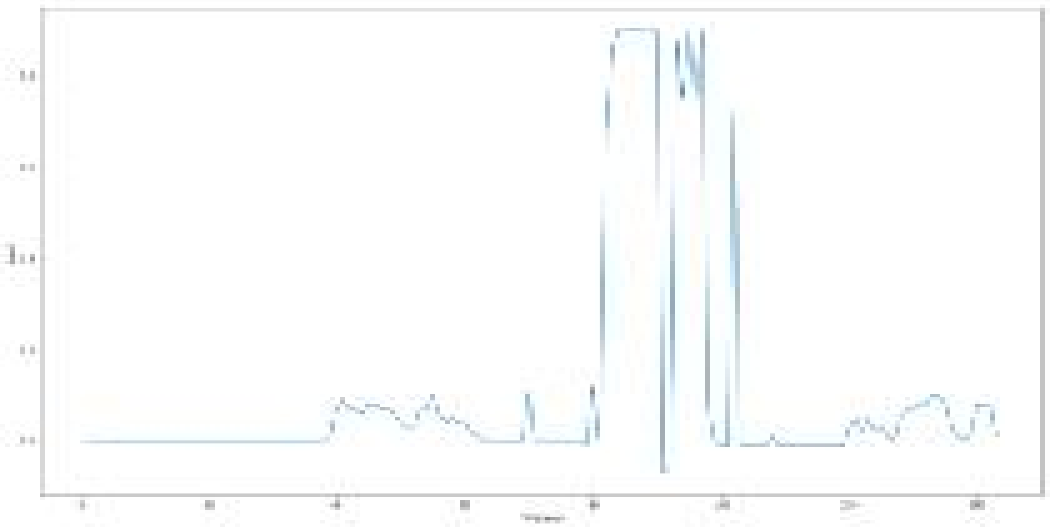
목차

- Abstract
- Feature
- Prepare
- Model
- Test
- Q&A

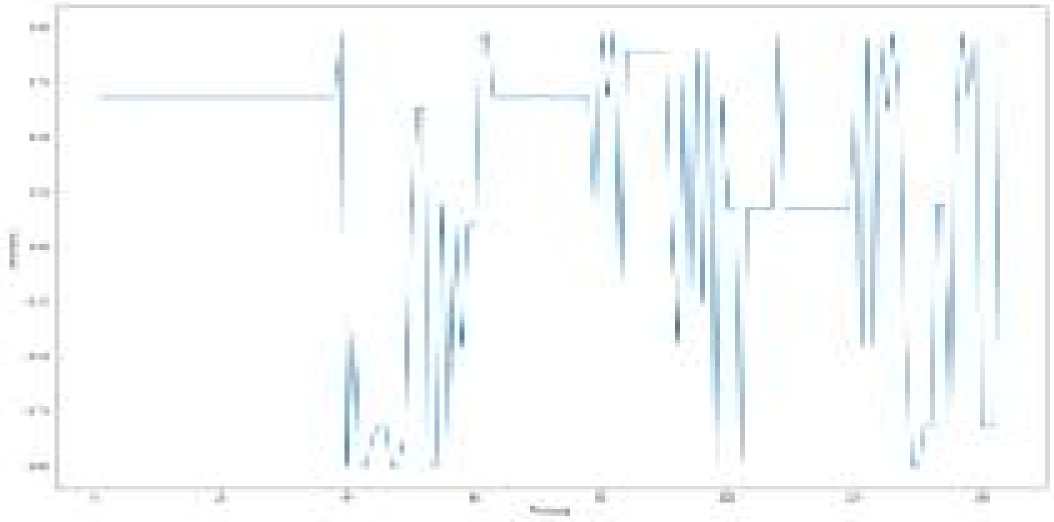
2. Feature

(TurbID == 1 & Day == 1)

pab1



$\sin(pab1)$

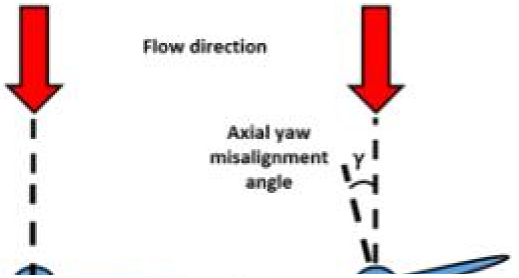


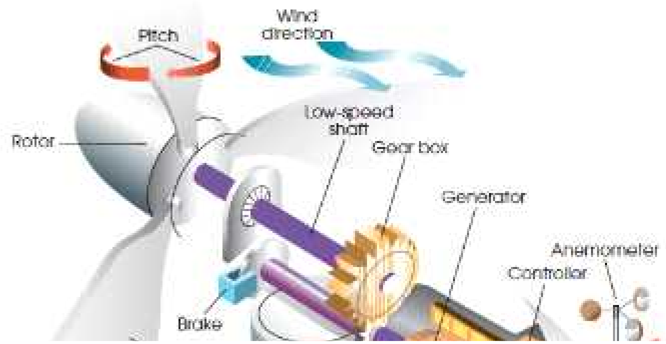
2. Feature


TurbID
Day
Wspd
Wdir
Etmp
Itmp
Ndir
Pab1
Pab2
Pab3
Prtv
Patv
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

2. Feature

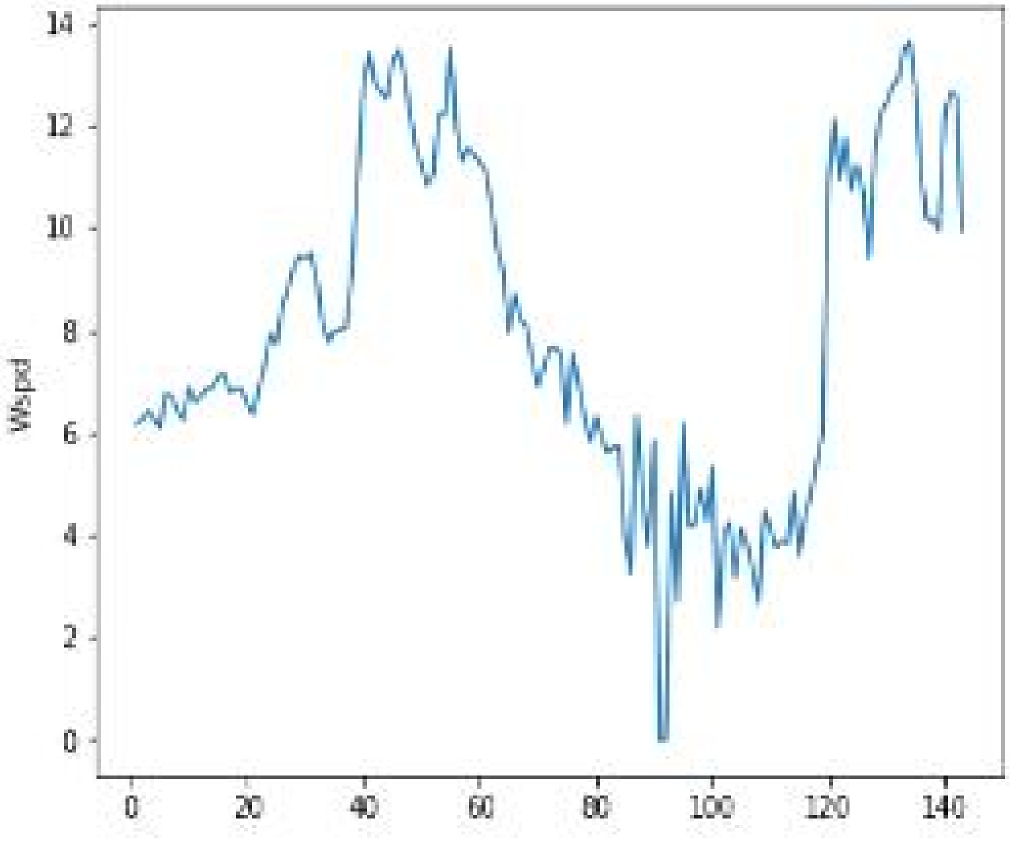
$$\overset{\text{Patv}}{\dot{P}} = \frac{1}{2} \times \rho \times A \times (\overset{\text{Wspd}}{V} \times \cos\gamma)^3$$

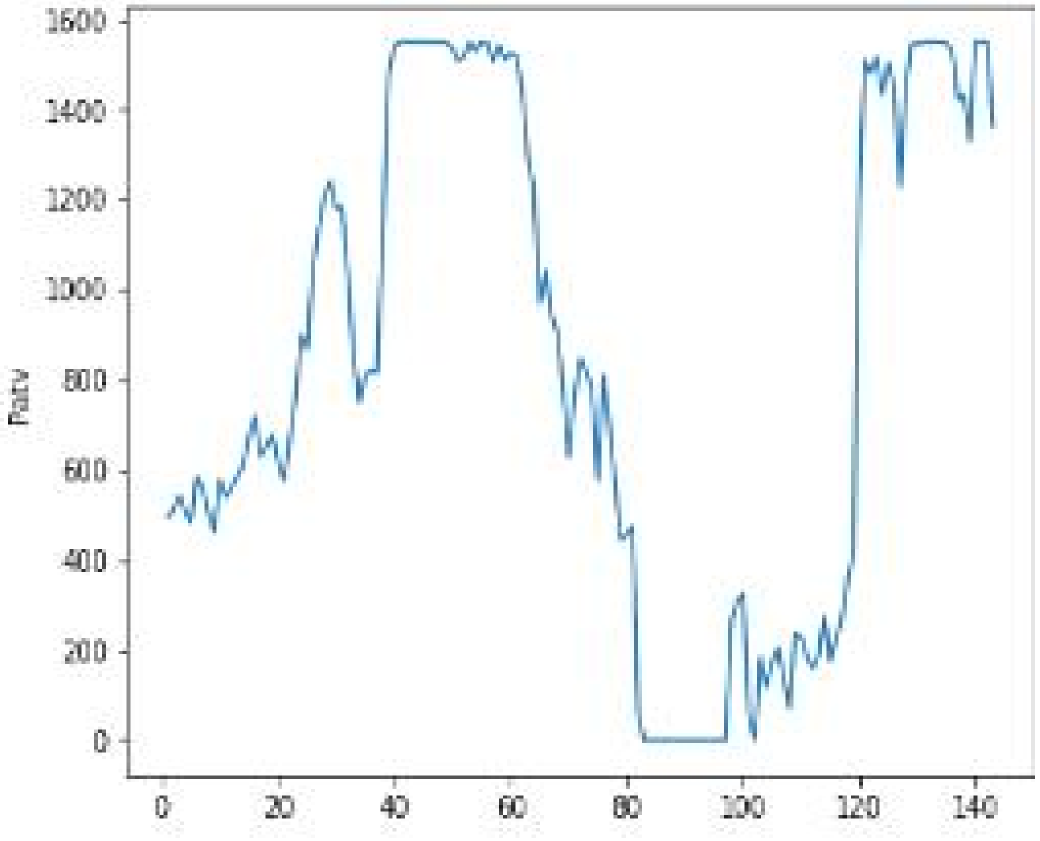




2. Feature

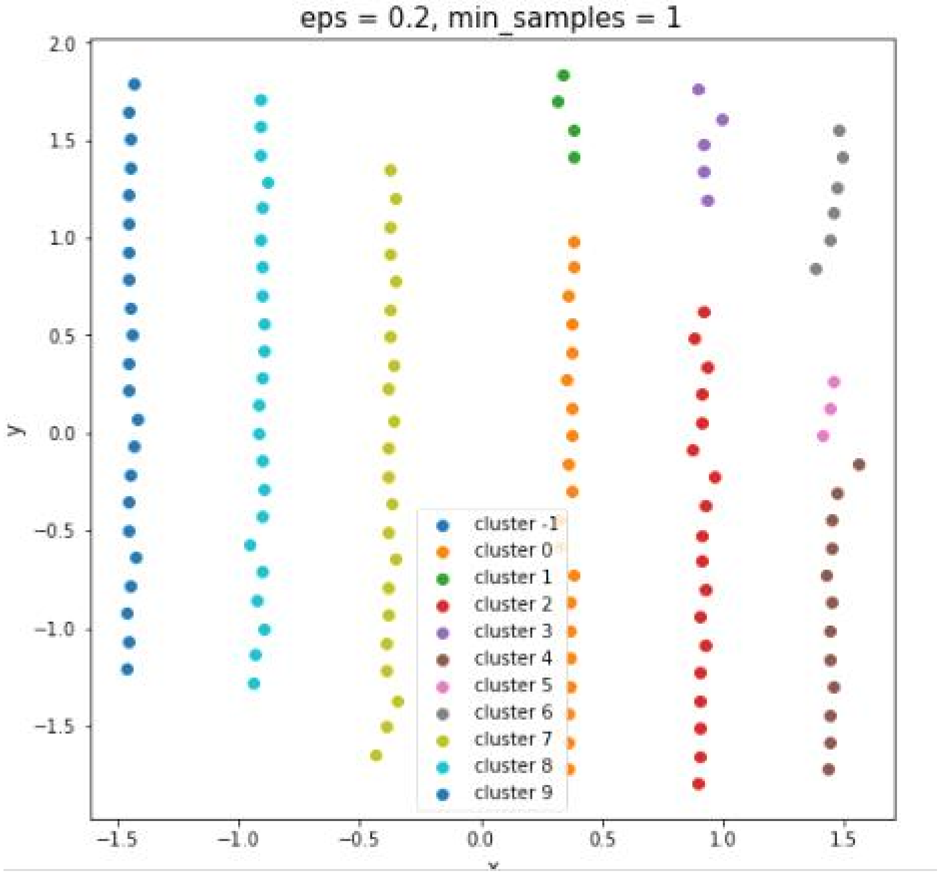
(TurbID == 1 & Day == 1)
Wspd 와 Patv 비교





2. Feature

공간정보 .. ?




```
1 cluster_df.iloc[[1,10,20,30,40,50,60,70],:]
```

	cl0	cl1	cl2	cl3	cl4	cl5	cl6	cl7	cl8	cl9	TurbID
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2
10	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	11
20	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	21
30	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	31
40	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	41
50	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	51
60	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	61
70	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	71

2. Feature

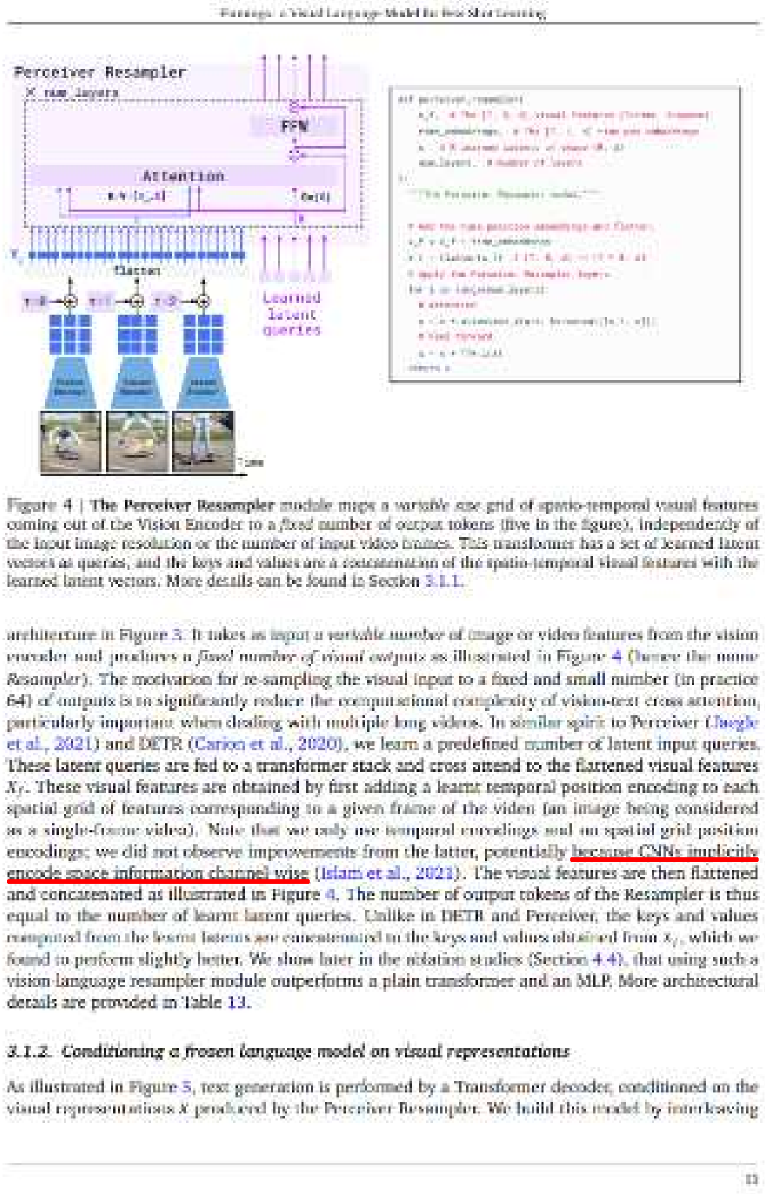

Flamingo: a Visual Language Model for Few-Shot Learning

Figure 4 | **The Perceiver Resampler** module maps a *variable* size grid of spatio-temporal visual features coming out of the Vision Encoder to a *fixed* number of output tokens (five in the figure), independently of the input image resolution or the number of input video frames. This transformer has a set of learned latent vectors as queries, and the keys and values are a concatenation of the spatio-temporal visual features with the learned latent vectors. More details can be found in Section 3.1.1.

architecture in Figure 3. It takes as input a variable number of image or video features from the vision encoder and produces a *fixed number of visual outputs* as illustrated in Figure 4 (hence the name *Resampler*). The motivation for re-sampling the visual input to a fixed and small number (in practice 64) of outputs is to significantly reduce the computational complexity of vision-text cross attention, particularly important when dealing with multiple long videos. In similar spirit to Perceiver (Jaegle et al., 2021) and DETR (Carion et al., 2020), we learn a predefined number of latent input queries. These latent queries are fed to a transformer stack and cross attend to the flattened visual features X_f. These visual features are obtained by first adding a learnt temporal position encoding to each spatial grid of features corresponding to a given frame of the video (an image being considered as a single-frame video). Note that we only use temporal encodings and no spatial grid position encodings; we did not observe improvements from the latter, potentially because CNNs implicitly encode space information channel-wise (Islam et al., 2021). The visual features are then flattened and concatenated as illustrated in Figure 4. The number of output tokens of the Resampler is thus equal to the number of learnt latent queries. Unlike in DETR and Perceiver, the keys and values computed from the learnt latents are concatenated to the keys and values obtained from X_f, which we found to perform slightly better. We show later in the ablation studies (Section 4.4), that using such a vision-language resampler module outperforms a plain transformer and an MLP. More architectural details are provided in Table 13.

3.1.2. Conditioning a frozen language model on visual representations

As illustrated in Figure 5, text generation is performed by a Transformer decoder, conditioned on the visual representations X produced by the Perceiver Resampler. We build this model by interleaving

because CNNs implicitly encode space information channel-wise.

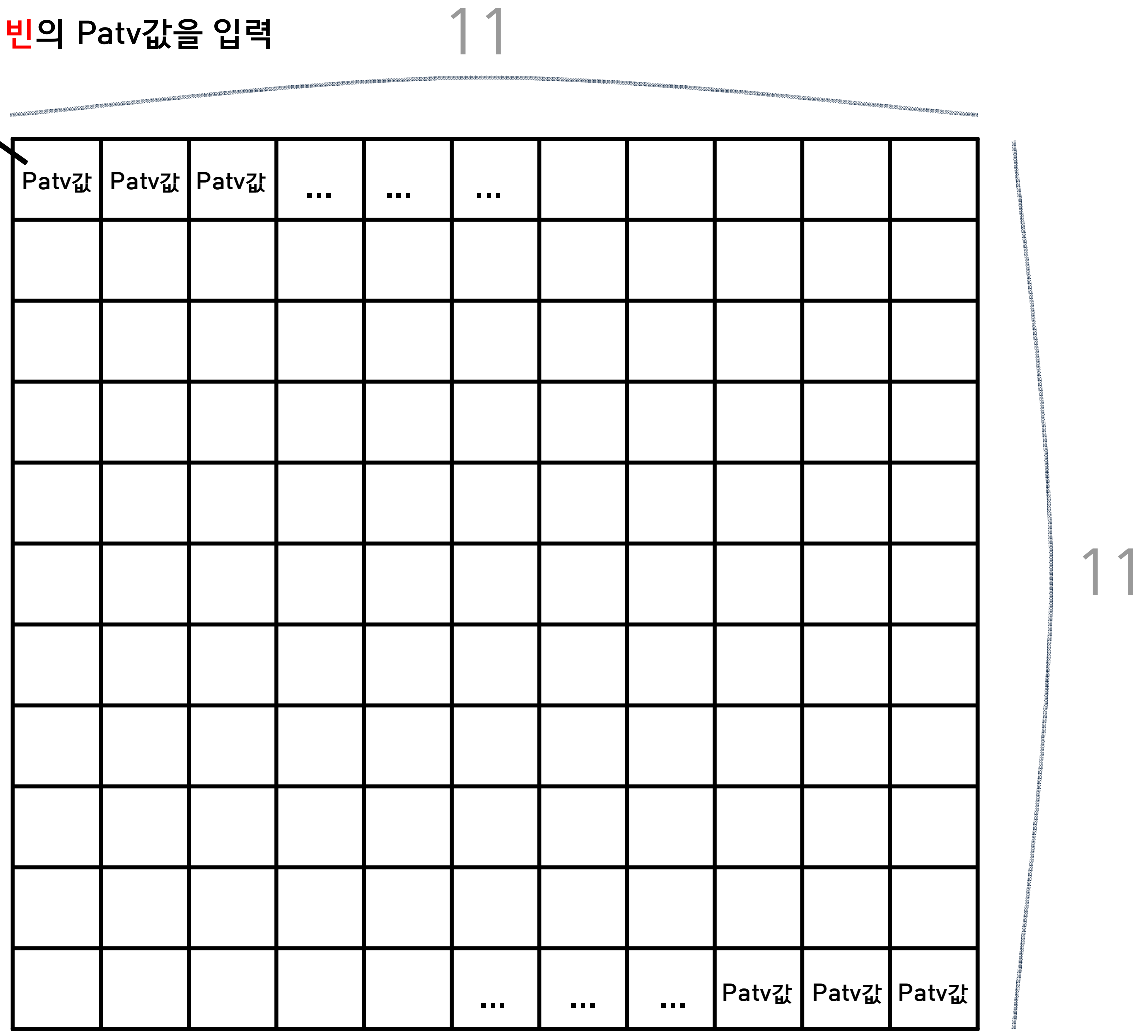
(CNN은 암묵적으로 공간 정보를 채널별로 인코딩하기 때문이다.)

CNN의 활용

2. Feature

하나의 터빈에 대하여 거리가 가까운 121개 터빈의 Patv값을 입력

256x256 이미지 예시



11x11 이미지 예시

목차

3. Prepare

```python
input_len = 144 # id 한 개당 하루에 생성되는 row의 갯수 (24hour / 10minute = 144)

for turbid in range(134): # 터빈 아이디의 갯수
    # print(turbid)
    full_dict = collections.defaultdict(list)
    turb_data = df[df['TurbID']==turbid+1].reset_index(drop=True)
    ls = list(turb_data['Patv'])
    for i in range(0, len(ls)-288-input_len): # range(0, ) # 288 (id 당 예측해야 하는 patv 갯수 2일 -> 144 * 2)
        if i % 10000 == 1: print(i)
        # i 가 0 이면 Wspd의 처음 시작 1일 0시 00분 부터 1일 23시 50분까지의 Wspd 리스트 추가
        full_dict['Wspd_seq'].append(list(turb_data['Wspd'])[i:i+input_len])
        full_dict['Patv_seq'].append(list(turb_data['Patv'])[i:i+input_len])
        full_dict['Etmp_seq'].append(list(turb_data['Etmp'])[i:i+input_len])
        full_dict['Itmp_seq'].append(list(turb_data['Itmp'])[i:i+input_len])
        full_dict['Pab1_seq'].append(list(turb_data['Pab1'])[i:i+input_len])

        full_dict['target'].append(list(turb_data['Patv'])[i+input_len:i+input_len+288])
        # i 가 0이면 [144: 144 + 288]  (288개의 patv 값 저장)
    df_his = pd.DataFrame(full_dict)
    df_his.to_csv('turbine_data'+str(turbid+1)+'.csv', index=False)
```

```
./
├── README.md
├── prepare.ipynb
├── model.ipynb
├── test.ipynb
├── data
│   ├── turbine_data1.csv
│   ├── turbine_data2.csv
...
│   ├── turbine_data134.csv
│   ├── train_data42.csv
│   ├── dist_dict.pickle
```

3. Prepare

```
1 df.head(10)
```

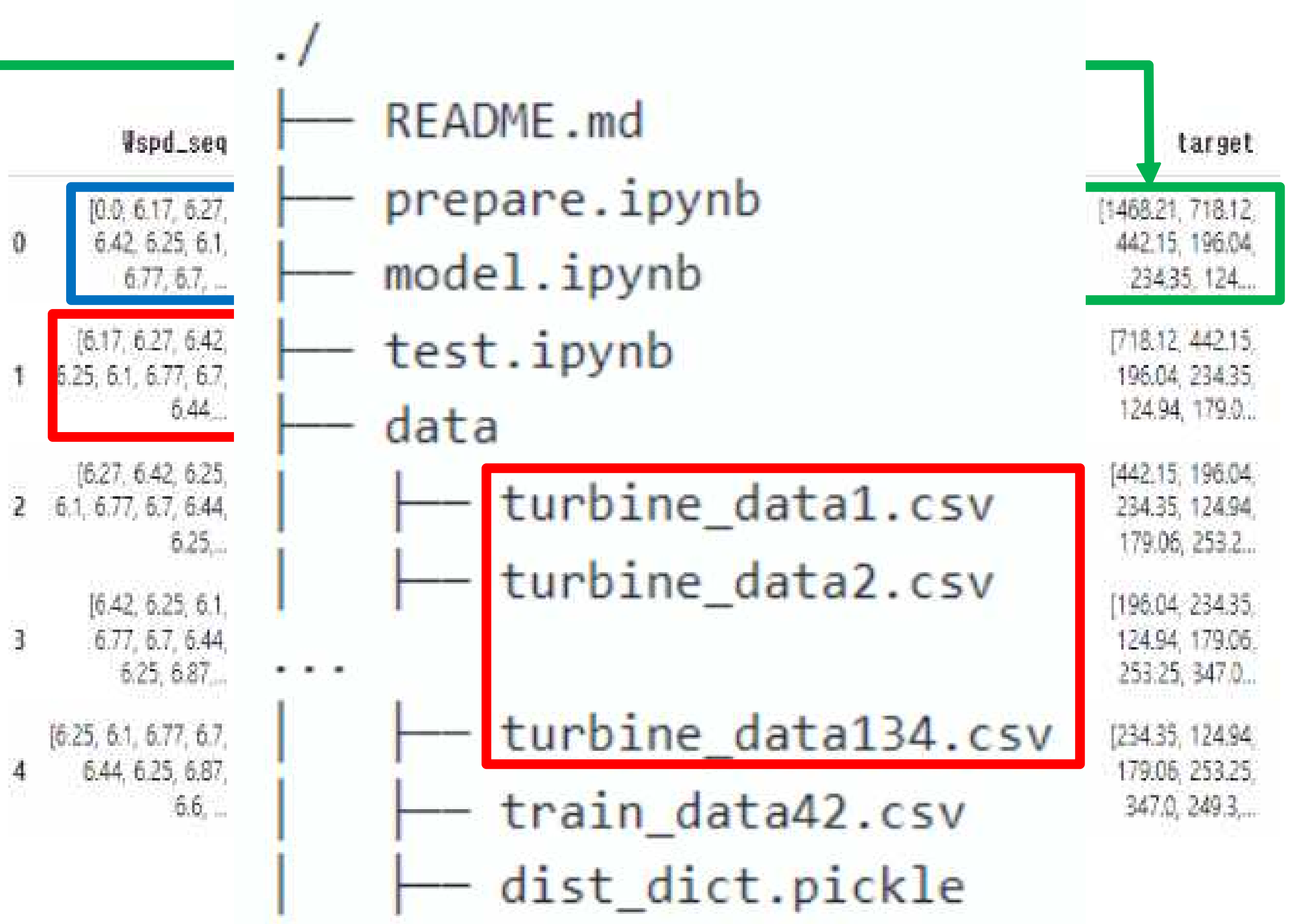
	TurbID	Day	Tmstamp	Wspd	Etmp	Itmp	Pab1	Patv
0	1	1	00:00	0.00	0.00	0.00	0.0	0.00
1	1	1	00:10	6.17	30.73	41.80	1.0	494.66
2	1	1	00:20	6.27	30.60	41.63	1.0	509.76
3	1	1	00:30	6.42	30.52	41.52	1.0	542.53
4	1	1	00:40	6.25	30.49	41.38	1.0	509.36
5	1	1	00:50	6.10	30.47	41.22	1.0	482.21
6	1	1	01:00	6.77	30.31	41.19	1.0	584.75
7	1	1	01:10	6.70	30.24	41.00	1.0	557.98
8	1	1	01:20	6.44	30.13	40.91	1.0	503.94
9	1	1	01:30	6.25	29.97	40.72	1.0	463.37

	Wspd_seq	Patv_seq	Etmp_seq	Itmp_seq	Pab1_seq	target
0	[0.0, 6.17, 6.27, 6.42, 6.25, 6.1, 6.77, 6.7, ...	[0.0, 494.66, 509.76, 542.53, 509.36, 482.21, ...	[0.0, 30.73, 30.6, 30.52, 30.49, 30.47, 30.31,...	[0.0, 41.8, 41.63, 41.52, 41.38, 41.22, 41.19,...	[0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...	[1468.21, 718.12, 442.15, 196.04, 234.35, 124...
1	[6.17, 6.27, 6.42, 6.25, 6.1, 6.77, 6.7, 6.44...	[494.66, 509.76, 542.53, 509.36, 482.21, 584.7...	[30.73, 30.6, 30.52, 30.49, 30.47, 30.31, 30.2...	[41.8, 41.63, 41.52, 41.38, 41.22, 41.19, 41.0...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...	[718.12, 442.15, 196.04, 234.35, 124.94, 179.0...
2	[6.27, 6.42, 6.25, 6.1, 6.77, 6.7, 6.44, 6.25,...	[509.76, 542.53, 509.36, 482.21, 584.75, 557.9...	[30.6, 30.52, 30.49, 30.47, 30.31, 30.24, 30.1...	[41.63, 41.52, 41.38, 41.22, 41.19, 41.0, 40.9...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...	[442.15, 196.04, 234.35, 124.94, 179.06, 253.2...
3	[6.42, 6.25, 6.1, 6.77, 6.7, 6.44, 6.25, 6.87,...	[542.53, 509.36, 482.21, 584.75, 557.98, 503.9...	[30.52, 30.49, 30.47, 30.31, 30.24, 30.13, 29....	[41.52, 41.38, 41.22, 41.19, 41.0, 40.91, 40.7...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...	[196.04, 234.35, 124.94, 179.06, 253.25, 347.0...
4	[6.25, 6.1, 6.77, 6.7, 6.44, 6.25, 6.87, 6.6, ...	[509.36, 482.21, 584.75, 557.98, 503.94, 463.3...	[30.49, 30.47, 30.31, 30.24, 30.13, 29.97, 29....	[41.38, 41.22, 41.19, 41.0, 40.91, 40.72, 40.7...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ...	[234.35, 124.94, 179.06, 253.25, 347.0, 249.3,...

3. Prepare

```
1 df.iloc[144:144+10,:]
```

	TurbID	Day	Tmstamp	Wspd	Etmp	Itmp	Pab1	Patv
144	1	2	00:00	10.64	28.10	42.17	1.01	1468.21
145	1	2	00:10	7.00	27.87	42.05	1.00	718.12
146	1	2	00:20	5.87	27.38	41.73	0.99	442.15
147	1	2	00:30	4.00	27.13	41.76	0.99	196.04
148	1	2	00:40	4.03	26.99	41.66	0.99	234.35
149	1	2	00:50	3.46	26.80	41.36	0.99	124.94
150	1	2	01:00	3.96	26.52	40.87	0.99	179.06
151	1	2	01:10	4.63	26.25	39.97	0.99	253.25
152	1	2	01:20	5.40	26.01	39.09	0.99	347.00
153	1	2	01:30	4.73	25.84	38.32	0.99	249.30



하루(144개) 의 Feature list * 5 → 이틀치의(288개) Target list 예측

3. Prepare

```
In [37]: for i in tqdm(range(134)):
             df_tmp = pd.read_csv('data/turbine_data'+str(i+1)+'.csv')
             df_tmp['index'] = list(range(len(df_tmp)))
             df_tmp = df_tmp.sample(frac=0.01, random_state=7)
             index_list = list(df_tmp['index'])
             df_tmp['TurbID'] = i+1

             turb_data = df[df['TurbID']==i+1].reset_index(drop=True)
             near_turbs = dist_dict[i+1][:121]
             index_new = [x+144-1 for x in index_list]
             selected_df = turb_data.loc[index_new]
             selected_df = pd.merge(selected_df, df_group, how='left', on=['Day','Tmstamp'])
             selected_df['Patv_space'] = selected_df['Patv_list'].apply(lambda x: [x[k-1] for k in near_turbs])
             df_tmp['Patv_space'] = list(selected_df['Patv_space'])


             if i==0:
                 train = df_tmp.copy()
             else:
                 train = pd.concat([train, df_tmp])
         train = train.reset_index(drop=True)
         train = train.sample(frac=1, random_state=2).reset_index(drop=True)

         100%|██████████████████████████████████████████████| 134/134 [03:45<00:0
         0,  1.63s/it]

In [38]: cols = [x for x in train.columns if 'seq' in x or x=='target']
         for col in cols:
             train[col] = train[col].apply(lambda x: json.loads(x))

         train = drop_abnormal(train)
         train.to_csv('data/train_data42.csv', index=False)

In [42]: dat42 = pd.read_csv('data/train_data42.csv')
         print(dat42)
```

```
./
├── README.md
├── prepare.ipynb
├── model.ipynb
├── test.ipynb
├── data
│    ├── turbine_data1.csv
│    ├── turbine_data2.csv
...
│    ├── turbine_data134.csv
│    ├── train_data42.csv
│    ├── dist_dict.pickle
```

3. Prepare

TurbID	index	Wspd_seq	Patv_seq	Etmp_seq	Itmp_seq	Pab1_seq	Patv_space	target
1	62	[10.45, 9.61, 9.17, 7.90, 8.71, 8.22, 8.08, 7...	[1442.49, 1292.17, 1225.7, 960.02, 1013.58, 96...	[32.53, 32.8, 33.04, 33.27, 33.57, 33.67, 33...	[43.33, 45.42, 45.01, 40.84, 46.39, 46.09, 46...	[1.01, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.11...	[378.1, 498.75, 505.88, 505.58, 508.71, 390.74...	[598.99, 523.09, 591.37, 444.78, 400.76, 810.5...
1	151	[4.61, 4.7, 4.71, 4.43, 4.85, 3.14, 3.7, 4.57,...	[353.25, 347.0, 248.3, 205.12, 254.04, [illegible]...	[26.25, 26.0, 25.84, 25.49, 25.27, 24.95, 24...	[38.97, 39.09, 38.82, 37.88, 37.15, 36.7[illegible], 36...	[0.99, 0.99, 0.99, 0.99, 0.99, 1.0, 0.99, 0.99...	[980.43, 1457.23, 1176.17, 1100.2[illegible], 1474.57, 1...	[1548.82, 1546.65, 1549.3[illegible], 1498.51, 1504.11, ...
1	253	[5.04, 5.78, 4.05, 4.4, 3.65, 3.99, 3.11, 2.77...	[121.01, 164.86, 205.14, 240.6, 153.27, 145.6...	[31.84, 31.85, 31.6, 31.73, 31.53, 31.34, 31.1...	[39.06, 39.01, 39.86, 39.75, 39.51, 39.4, 39.3...	[0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.9...	[1527.3, 1466.76, 1493.2[illegible], 1482.34, 1447.56, 1...	[1490.34, 1481.3, 1489.68, 1530.56, 1521.49, 1...
1	381	[11.18, 11.57, 11.75, 11.6[illegible], 11.13, 11.[illegible], 10...	[151[illegible].29, 1522.87, 151[illegible], 1509.51, 1500.71...	[20.81, 20.26, 20.27, [illegible], 19.41, [illegible]...	[42.02, 43.02, 42.09, [illegible]...	[1.02, 1.04, 1.05, 1.04, 1.01, 1.04, 1.05, 1.0...	[343.36, 440.9, 378.82, 445.30, 537.92, 485.14...	[486.82, [illegible], 427.68, [illegible]...
1	467	[11.53, 11.55, 11.77, 11.[illegible], 12.02, 12.54, 12...	[145[illegible].01, 1544.07, 15[illegible], 1548.[illegible], 1531.59...	[22.28, 22.2, 22.14, 22.09, 22.03, 22.07, 22.1...	[37.19, 37.09, 37.09, 37.06, 37.06, 36.99, 36...	[1.07, [illegible].06, 1.07, 1.07, 1.07, 1.09, 1.08, 1.0...	[[illegible]919.56, 833.94, 900.01, 882.72, 1234.27, 833...	[1422.55, [illegible]209.05, 1197.9[illegible], 1064.28, 1090.52, ...
1	480	[11.89, 12.66, 12.66, 11.27, 11.0[illegible], 12.51, 12...	[1510.99, 1526.67, 1504.16, 1510.82, 737.8, 15...	[22.85, 22.94, 23.12, 23.3, 23.64, 23.97, 23.5...	[37.49, 37.41, 37.48, 37.52, 37.86, 38.01, 37...	[1.07, 1.09, 1.09, 1.05, [illegible].42, 1.06, 1.07, 1.0...	[-0.3, -0.3, 773.5, 583.16, -0.3, 693.54, 0.0...	[1543.78, 1508.18, 1483.59, 1517.56, 1513.0[illegible], ...
1	495	[12.23, 12.08, 12.12, 11.58, 11.79, 12.38, 12...	[1542.44, 1547.11, 1534.13, 1540.59, 1527.62, ...	[24.72, 25.05, 25.42, 25.7, 26.0, 25.56, 25.55...	[38.55, 38.36, 38.52, 39.17, 39.42, 39.[illegible], 40...	[1.07, 1.06, 1.07, 1.06, 1.06, 1.07, 1.06, 1.0...	[-0.3, -0.3, 241.65, 754.45, -0.3, 276.06, 0.0...	[1318.48, 1512.33, 1469.15, 1470.04, 1490.43, ...
1	505	[12.66, 13.02, 12.90, 12.65, 12.53, 12.55, 12...	[1528.21, 1545.58, 1552.75, 1515.57, 1548.12, ...	[[illegible].75, [illegible], [illegible], 29.1...	[41.08, [illegible], 41.58, [illegible], 42...	[1.06, 1.08, 1.06, 1.06, 1.08, 1.06, 1.07, 1.0...	[-0.3, -0.1, 225.95, 227.05, -0.1, 227.67, 0.0...	[1537.87, 1501.7, 1510.67, [illegible], 1409.51, 1...
1	562	[8.20, 7.79, 8.83, 9.07, 8.71, 8.99, 9.07, 9.3...	[938.45, 982.05, 1074.40, 1070.95, 1053.42, 10...	[29.97, 29.92, 29.87, 29.7, 29.6, 29.44, 29.3...	[39.3, 39.71, 39.79, 39.77, 39.74, 39.54, 40.0...	[0.99, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0...	[550.19, 871.86, 543.72, 775.68, 488.81, 865.0...	[400.06, 878.75, 1288.2, 1083.98, 1080.36, 873...
1	568	[8.71, 8.88, 8.07, 9.33, 9.25, 8.85, 9.31, 9.1...	[1093.42, 1095.11, 1154.13, 1181.06, 1124.35 ...	[29.61, 29.44, 29.52, 29.18, 28.90, 28.83, 29...	[39.74, 39.94, 40.08, 40.15, 40.22, 40.52, 40...	[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0...	[1081.51, 1060.06, 1183.40, 1[illegible]69.04, 700.06, 1...	[1080.56, 873.68, 754.98, 506.25, 381.97, 474...

거리가 가까운 121개의 터빈의
Patv 값을 feature로 사용

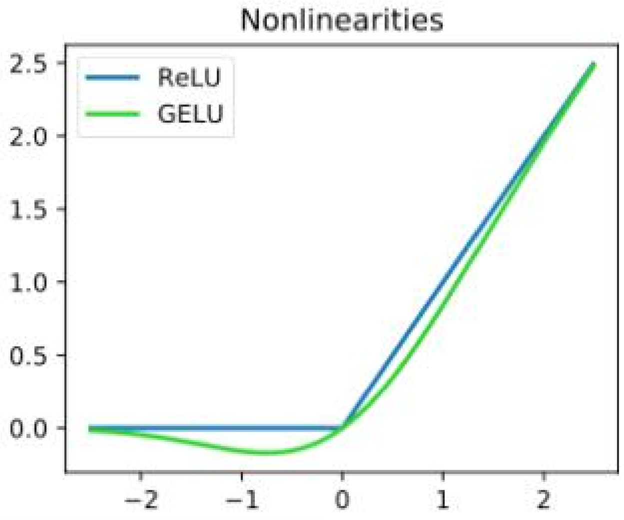
11 by 11 matrix로 만들어
CNN의 이미지로 활용

목차

4. Model(GRU + CNN)

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

        self.gru = nn.GRU(input_size=4, hidden_size=48, num_layers=2)
        self.dropout = nn.Dropout(0.05)

#       self.Linear = nn.Linear(48+2, 1, bias_attr=True)
        self.Linear = nn.Linear(48+2, 1)

        self.cnnLayer = nn.Sequential(
        nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1),
        nn.BatchNorm2d(64),
        nn.GELU(),
        nn.MaxPool2d((3,3)))

    def forward(self, X, space_data):
#       z = torch.zeros([X.shape[0], 144, X.shape[1]], dtype="float32")
        z = torch.zeros([X.shape[0], 144, X.shape[1]])
        z = z.to(device)
#       x = torch.concat((X.transpose([0,2,1]),z), axis=1)
#       print(X.shape)
#       print(z)
#       print(type(x))
#       print(type(X))
#       print(type(z))
        x = torch.concat((torch.transpose(X,2,1),z), axis=1)
#       print(1)
        out1, _ = self.gru(x)
        out1 = self.dropout(out1)

        cnn_out = self.cnnLayer(space_data)
#       print(cnn_out.shape)
        cnn_out = torch.reshape(cnn_out, (cnn_out.shape[0], 288, -1))
#       print(cnn_out.shape)
        out2 = self.Linear(torch.concat((out1, cnn_out), 2))

        return out2
```

[하이퍼파라미터]
batch = 16,32,64
lr = 0.0005, 0.0003, 0.00001
optimizer = GELU, RELU



인하대학교
INHA UNIVERSITY

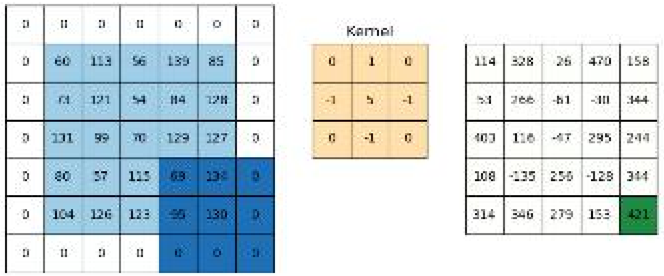
4. Model(GRU + CNN)

Image

```
tensor([[[379., 498., 506., 506., 509., 391., 328., 409., 490., 474., 575.],
         [455., -0., 462., 335., 293., -0., 301., 336., 278., -0., 424.],
         [427., 339., -0., 282., 215., 351., -0., -0., 180., -0., -0.],
         [ -0., 391., -0., -0., -0., -0., 423., 226., -0., -0., 217.],
         [ -0., -0., 350., -0., 135., 180., -0., -0., -0., -0., -0.],
         [ -0., -0., 324., 276., -0., -0., 131., 235., -0., -0., -0.],
         [312., -0., -0., -0., 120., 233., -0., -0., 110., -0., 94.],
         [ -0., 64., -0., 133., -0., 0., -0., -0., 135., -0., 120.],
         [110., -0., -0., -0., 104., 125., -0., 163., 131., -0., -0.],
         [ -0., -0., 244., -0., -0., -0., 129., -0., 79., -0., -0.],
         [ 42., -0., -0., 257., 111., -0., 2., -0., -0., -0., -0.]]])
```

Conv2d

Ex) Convolution




```
s2 =torch.reshape(np.round(s1,0),(-1,1,11,11))
```

```
conv1 = nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1)
out1 = conv1(s2)
```

```
out1.shape
```

```
torch.Size([1, 64, 11, 11])
```

```
torch.round(out1[0][0])[:,:]
```

```
tensor([[  14.,  -19.,  -16.,   70.,  133.,   16.,  -46.,   21.,  109.,  -47.,
          143.],
        [ -94.,  -65., -170., -158., -115.,  -74.,  -63., -155.,  -77., -161.,
         -119.],
        [-207.,   67.,  -52., -118.,  -95.,  -80.,  -62.,  -26.,  -24.,  -70.,
          -74.],
        [-164., -154.,   38.,  -37., -109., -111.,   22.,   53.,  -22.,  -44.,
           22.],
        [ -11., -221.,  -26.,  100.,   41.,  -15., -133.,  -31.,   30.,   -6.,
          -50.],
        [  32.,    3.,  -86.,   -2.,  -59.,  -22.,   12.,    7.,   31.,   -3.,
           10.],
        [  16.,   23., -103.,  -89.,  -15.,   30.,  -17., -112.,    9.,   -9.,
           22.],
        [ -69.,    6.,  -14.,  -16.,  -44.,  -28.,  -33.,  -39.,   34.,    9.,
           -9.],
        [   9.,  -67.,   17.,    1.,  -17.,  -11.,   -0.,   -1.,    4.,   13.,
          -28.],
        [ -21.,  -35.,  -39.,   16.,   34.,  -34.,    0.,  -41.,  -33.,   -3.,
           -0.],
        [   4.,   -4.,  -93.,   -6.,   33.,    5.,  -29.,  -11.,  -18.,   -6.,
           -0.]], grad_fn=<SliceBackward0>)
```


인하대학교 INHA UNIVERSITY

4. Model(GRU + CNN)

```
tensor([[  14.,  -19.,  -16.,   70.,  133.,   16.,  -46.,   21.,  109.,  -47.,
          143.],
        [ -94.,  -65., -170., -158., -115.,  -74.,  -63., -155.,  -77., -161.,
         -119.],
        [-207.,   67.,  -52., -118.,  -95.,  -80.,  -62.,  -26.,  -24.,  -70.,
          -74.],
        [-164., -154.,   38.,  -37., -109., -111.,   22.,   53.,  -22.,  -44.,
           22.],
        [ -11., -221.,  -26.,  100.,   41.,  -15., -133.,  -31.,   30.,   -6.,
          -50.],
        [  32.,    3.,  -86.,   -2.,  -59.,  -22.,   12.,    7.,   31.,   -3.,
           10.],
        [  16.,   23., -103.,  -89.,  -15.,   30.,  -17., -112.,    9.,   -9.,
           22.],
        [ -69.,    6.,  -14.,  -16.,  -44.,  -28.,  -33.,  -39.,   34.,    9.,
           -9.],
        [   9.,  -67.,   17.,    1.,  -17.,  -11.,   -0.,   -1.,    4.,   13.,
          -28.],
        [ -21.,  -35.,  -39.,   16.,   34.,  -34.,    0.,  -41.,  -33.,   -3.,
           -0.],
        [   4.,   -4.,  -93.,   -6.,   33.,    5.,  -29.,  -11.,  -18.,   -6.,
           -0.]], grad_fn=<SliceBackward0>)
```

BatchNormalization

```
batch1 = nn.BatchNorm2d(64)
out2 = batch1(out1)
print(out2.shape)
out2[0][0]
```

```
torch.Size([1, 64, 11, 11])
tensor([[ 6.6612e-01,  1.3315e-01,  1.8540e-01,  1.5603e+00,  2.5513e+00,
          6.8686e-01, -2.9704e-01,  7.7814e-01,  2.1786e+00, -3.0944e-01,
          2.7136e+00],
        [-1.0560e+00, -6.0338e-01, -2.2632e+00, -2.0789e+00, -1.3894e+00,
         -7.4707e-01, -5.6277e-01, -2.0304e+00, -7.9336e-01, -2.1309e+00,
         -1.4617e+00],
        [-2.8569e+00,  1.5107e+00, -3.9784e-01, -1.4395e+00, -1.0735e+00,
         -8.4347e-01, -5.4561e-01,  2.1713e-02,  5.2930e-02, -6.8428e-01,
         -7.4316e-01],
        [-2.1679e+00, -2.0094e+00,  1.0398e+00, -1.5237e-01, -1.3012e+00,
         -1.3358e+00,  7.8404e-01,  1.2858e+00,  7.9492e-02, -2.6088e-01,
          7.9342e-01],
        [ 2.6121e-01, -3.0881e+00,  2.5165e-02,  2.0338e+00,  1.0868e+00,
          1.9849e-01, -1.6854e+00, -5.0611e-02,  9.1003e-01,  3.3840e-01,
         -3.5249e-01],
        [ 9.5256e-01,  4.8590e-01, -9.3685e-01,  4.0024e-01, -5.0403e-01,
          8.7469e-02,  6.2727e-01,  5.5579e-01,  9.2688e-01,  3.9535e-01,
          5.9070e-01],
        [ 6.9449e-01,  8.0825e-01, -1.1992e+00, -9.7262e-01,  2.0402e-01,
          9.1987e-01,  1.6221e-01, -1.3533e+00,  5.8699e-01,  2.9566e-01,
          7.8926e-01],
        [-6.6234e-01,  5.4043e-01,  2.1175e-01,  1.8532e-01, -2.5611e-01,
         -1.4226e-02, -8.6193e-02, -1.7941e-01,  9.7519e-01,  5.7587e-01,
          2.9208e-01],
        [ 5.8813e-01, -6.2854e-01,  7.1162e-01,  4.5879e-01,  1.7431e-01,
          2.6236e-01,  4.3695e-01,  4.2049e-01,  5.0743e-01,  6.5013e-01,
         -6.2905e-04],
        [ 1.0536e-01, -1.1915e-01, -1.8992e-01,  6.9262e-01,  9.7283e-01,
         -9.7259e-02,  4.4404e-01, -2.2113e-01, -8.5840e-02,  3.9737e-01,
          4.3466e-01],
        [ 5.0410e-01,  3.8176e-01, -1.0441e+00,  3.3980e-01,  9.5678e-01,
          5.1907e-01, -2.9969e-02,  2.6303e-01,  1.4810e-01,  3.4940e-01,
          4.3466e-01]], grad_fn=<SelectBackward0>)
```

4. Model(GRU + CNN)

```
batch1 = nn.BatchNorm2d(64)
out2 = batch1(out1)
print(out2.shape)
out2[0][0]
```

```
torch.Size([1, 64, 11, 11])
tensor([[ 6.6612e-01,  1.3315e-01,  1.8540e-01,  1.5603e+00,  2.5513e+00,
          6.8685e-01, -2.9704e-01,  7.7814e-01,  2.1785e+00, -3.0944e-01,
          2.7135e+00],
        [-1.0560e+00, -6.0335e-01, -2.2632e+00, -2.0789e+00, -1.3894e+00,
         -7.4707e-01, -5.5277e-01, -2.0304e+00, -7.9336e-01, -2.1309e+00,
         -1.4617e+00],
        [-2.6569e+00,  1.5107e+00, -3.9784e-01, -1.4395e+00, -1.0735e+00,
         -8.4347e-01, -5.4561e-01,  2.1713e-02,  5.2930e-02, -6.8428e-01,
         -7.4315e-01],
        [-2.1679e+00, -2.0094e+00,  1.0398e+00, -1.5237e-01, -1.3012e+00,
         -1.3358e+00,  7.8404e-01,  1.2858e+00,  7.9492e-02, -2.6088e-01,
          7.9342e-01],
        [ 2.6121e-01, -3.0881e+00,  2.5165e-02,  2.0338e+00,  1.0868e+00,
          1.9849e-01, -1.6854e+00, -5.0611e-02,  9.1003e-01,  3.3840e-01,
         -3.5249e-01],
        [ 9.5256e-01,  4.8590e-01, -9.3685e-01,  4.0024e-01, -5.0403e-01,
          8.7469e-02,  6.2727e-01,  5.5579e-01,  9.2688e-01,  3.9535e-01,
          5.9070e-01],
        [ 6.9449e-01,  8.0825e-01, -1.1992e+00, -9.7262e-01,  2.0402e-01,
          9.1987e-01,  1.5221e-01, -1.3533e+00,  5.8699e-01,  2.9566e-01,
          7.8926e-01],
        [-6.6234e-01,  5.4043e-01,  2.1175e-01,  1.8532e-01, -2.5611e-01,
         -1.4226e-02, -8.6193e-02, -1.7941e-01,  9.7519e-01,  5.7587e-01,
          2.9208e-01],
        [ 5.5813e-01, -6.2854e-01,  7.1162e-01,  4.5879e-01,  1.7431e-01,
          2.6236e-01,  4.3695e-01,  4.2049e-01,  5.0743e-01,  6.5013e-01,
         -6.2905e-04],
        [ 1.0536e-01, -1.1915e-01, -1.8992e-01,  6.9262e-01,  9.7283e-01,
         -9.7259e-02,  4.4404e-01, -2.2113e-01, -8.5840e-02,  3.9737e-01,
          4.3466e-01],
        [ 5.0410e-01,  3.8175e-01, -1.0441e+00,  3.3980e-01,  9.5678e-01,
          5.1907e-01, -2.9969e-02,  2.6303e-01,  1.4810e-01,  3.4940e-01,
          4.3466e-01]], grad_fn=<SelectBackward0>)
```

Activation function(GELU)

```
opti1 = nn.GELU()
out3 = opti1(out2)
out3.shape
(out3[0][0])
```

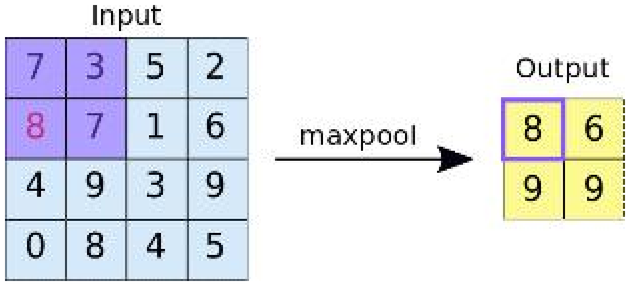
```
tensor([[ 4.9781e-01,  7.3624e-02,  1.0633e-01,  1.4677e+00,  2.5377e+00,
          5.1783e-01, -1.1383e-01,  6.0831e-01,  2.1466e+00, -1.1712e-01,
          2.7046e+00],
        [-1.5363e-01, -1.6480e-01, -2.6734e-02, -3.9113e-02, -1.1442e-01,
         -1.6997e-01, -1.6140e-01, -4.2958e-02, -1.6961e-01, -3.5265e-02,
         -1.0512e-01],
        [-6.1104e-03,  1.4119e+00, -1.3740e-01, -1.0796e-01, -1.5192e-01,
         -1.6826e-01, -1.5968e-01,  1.1045e-02,  2.7582e-02, -1.6895e-01,
         -1.6996e-01],
        [-3.2702e-02, -4.4703e-02,  8.8469e-01, -6.6958e-02, -1.2569e-01,
         -1.2131e-01,  6.1429e-01,  1.1581e+00,  4.2265e-02, -1.0359e-01,
          6.2381e-01],
        [ 1.5752e-01, -3.1101e-03,  1.2835e-02,  1.9911e+00,  9.3623e-01,
          1.1486e-01, -7.7459e-02, -2.4284e-02,  7.4495e-01,  2.1403e-01,
         -1.2768e-01],
        [ 7.9024e-01,  3.3356e-01, -1.6340e-01,  2.6236e-01, -1.5480e-01,
          4.6783e-02,  4.6089e-01,  3.9507e-01,  7.6283e-01,  2.5845e-01,
          4.2686e-01],
        [ 5.2525e-01,  6.3894e-01, -1.3818e-01, -1.6084e-01,  1.1850e-01,
          7.5537e-01,  9.1558e-02, -1.1907e-01,  4.2345e-01,  1.8220e-01,
          6.1959e-01],
        [-1.6815e-01,  3.8130e-01,  1.2363e-01,  1.0628e-01, -1.0217e-01,
         -7.0322e-03, -4.0136e-02, -7.6932e-02,  8.1455e-01,  4.1327e-01,
          1.7959e-01],
        [ 4.2450e-01, -1.6645e-01,  5.4200e-01,  3.1052e-01,  9.9213e-02,
          1.5833e-01,  2.9229e-01,  2.7876e-01,  3.5220e-01,  4.8252e-01,
         -3.1437e-04],
        [ 5.7102e-02, -5.3926e-02, -8.0656e-02,  5.2344e-01,  8.1200e-01,
         -4.4862e-02,  2.9817e-01, -9.1214e-02, -3.9984e-02,  2.6006e-01,
          2.9040e-01],
        [ 3.4929e-01,  2.4764e-01, -1.5476e-01,  2.1509e-01,  7.9476e-01,
          3.6238e-01, -1.4626e-02,  1.5880e-01,  8.2765e-02,  2.2243e-01,
          2.9040e-01]], grad_fn=<SelectBackward0>)
```

4. Model(GRU + CNN)

```
opti1 = nn.GELU()
out3 = opti1(out2)
out3.shape
(out3[0][0])
```

```
tensor([[ 4.9781e-01,  7.3624e-02,  1.0633e-01,  1.4677e+00,  2.5377e+00,
          5.1783e-01, -1.1383e-01,  5.0831e-01,  2.1466e+00, -1.1712e-01,
          2.7046e+00],
        [-1.5363e-01, -1.6480e-01, -2.6734e-02, -3.9113e-02, -1.1442e-01,
         -1.6997e-01, -1.6140e-01, -4.2958e-02, -1.6961e-01, -3.5265e-02,
         -1.0512e-01],
        [-6.1104e-03,  1.4119e+00, -1.3740e-01, -1.0796e-01, -1.5192e-01,
         -1.6826e-01, -1.5968e-01,  1.1045e-02,  2.7582e-02, -1.6895e-01,
         -1.6996e-01],
        [-3.2702e-02, -4.4703e-02,  8.8469e-01, -6.6958e-02, -1.2569e-01,
         -1.2131e-01,  6.1429e-01,  1.1581e+00,  4.2265e-02, -1.0359e-01,
          6.2381e-01],
        [ 1.5752e-01, -3.1101e-03,  1.2835e-02,  1.9911e+00,  9.3623e-01,
          1.1486e-01, -7.7459e-02, -2.4284e-02,  7.4495e-01,  2.1403e-01,
         -1.2768e-01],
        [ 7.9024e-01,  3.3356e-01, -1.6340e-01,  2.6236e-01, -1.5480e-01,
          4.6783e-02,  4.6089e-01,  3.9507e-01,  7.6283e-01,  2.5845e-01,
          4.2686e-01],
        [ 5.2525e-01,  6.3894e-01, -1.3818e-01, -1.6084e-01,  1.1850e-01,
          7.5537e-01,  9.1558e-02, -1.1907e-01,  4.2345e-01,  1.8220e-01,
          6.1959e-01],
```

Max-Pooling & Reshape




```
Pool1 = nn.MaxPool2d((3,3))
out4 = Pool1(out3)
out4.shape
cnn_out = torch.reshape(out4,(1,288,-1))
print(cnn_out.shape)
cnn_out
```

```
torch.Size([1, 288, 2])
tensor([[[ 1.4119e+00,  2.5377e+00],
         [ 2.1466e+00,  8.8469e-01],
         [ 1.9911e+00,  1.1581e+00],
         [ 6.3894e-01,  7.5537e-01],
         [ 8.1455e-01,  3.2357e+00],
         [ 2.0508e+00,  2.1387e+00],
         [ 1.2468e+00,  9.6917e-01],
         [ 5.1291e-01,  4.3646e-01],
         [ 1.8598e-02,  3.1642e-02],
         [ 2.6429e+00,  2.8486e-01],
         [ 6.5287e-01,  1.9474e+00],
         [ 8.2418e-01,  2.4485e-02],
         [ 1.1294e+00,  5.8659e-01],
         [ 9.8939e-01,  7.4055e-01],
         [ 6.3070e-01,  1.1087e+00],
         [ 1.6139e+00,  2.0318e+00],
         [ 2.4533e+00,  2.5359e+00],
         [ 1.0758e+00,  1.8824e+00],
         [ 2.2384e+00,  3.0895e-01],
         [ 2.6339e-01,  2.5383e+00],
         [ 5.5352e-01,  9.2229e-01],
         [ 1.0497e+00,  6.4886e-01],
         [ 1.1535e+00,  3.7798e+00],
         [ 7.8024e-01,  2.4156e+00],
         [ 2.7883e+00,  1.5911e+00],
         [ 3.0061e-01,  1.4013e+00],
         [ 4.1781e-01,  9.9199e-01],
         [ 3.4100e+00,  3.1764e+00],
         [ 2.9760e+00,  1.1447e+00],
         [ 5.9553e-01,  8.6133e-01],
         [ 1.5682e-01,  1.1398e-01],
         [ 1.4693e-01,  2.4255e+00],
```

4. Model(GRU + CNN)

In [19]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
def my_scale(lst):
    n = len(lst)
    x = np.array(lst).reshape(-1,1)
    scaler_robust = RobustScaler()
    scaler_robust.fit(x)
    k = scaler_robust.transform(x)
    kk = k.reshape(n)
    return list(np.round(kk, 4))
```

Standard Scaler	$\frac{x_i - \text{mean}(\boldsymbol{x})}{\text{stdev}(\boldsymbol{x})}$
MinMax Scaler	$\frac{x_i - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})}$
Robust Scaler	$\frac{x_i - Q_1(\boldsymbol{x})}{Q_3(\boldsymbol{x}) - Q_1(\boldsymbol{x})}$

In [13]:
```
df_train['Wspd_seq'] = df_train.apply(
    lambda x: [round(r**3,4) for r in x['Wspd_seq']],
    axis=1,
)
```

In [14]:
```
df_train['Wspd_seq']
```

Out[14]:
```
0        [0.0244, 0.2054, 0.9127, 2.6281, 7.3014, 11.08...
1        [4.2515, 1.5209, 6.2295, 14.1725, 5.0884, 5.35...
2        [19.0342, 20.7969, 18.6096, 18.8211, 14.1725, ...
3        [1423.8281, 1211.3555, 1201.157, 876.4675, 104...
4        [92.3454, 51.8951, 46.656, 42.5085, 27.8181, 2...
                               ...
37044    [353.3932, 382.6572, 430.3689, 433.7981, 460.0...
37045    [34.3281, 10.3602, 9.1293, 7.6454, 10.2183, 13...
37046    [6.2295, 58.4111, 84.6045, 51.4788, 59.7765, 5...
37047    [252.436, 211.7087, 228.0991, 174.6769, 125.0,...
37048    [296.741, 189.1192, 329.9394, 135.0057, 91.733...
Name: Wspd_seq, Length: 37049, dtype: object
```

4. Model(GRU + CNN)

```
class EarlyStopping:
    def __init__(self, patience=7, verbose=False, delta=0, path='checkpoint.pt', trace_func=print):

        self.patience = patience
        self.verbose = verbose
        self.counter = 0
        self.best_score = None
        self.early_stop = False
        self.val_loss_min = np.Inf
        self.delta = delta
        self.path = path
        self.trace_func = trace_func
    def __call__(self, val_loss, model):

        score = -val_loss

        if self.best_score is None:
            self.best_score = score
            self.save_checkpoint(val_loss, model)
        elif score < self.best_score + self.delta:
            self.counter += 1
            self.trace_func(f'EarlyStopping counter: {self.counter} out of {self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_score = score
            self.save_checkpoint(val_loss, model)
            self.counter = 0

    def save_checkpoint(self, val_loss, model):
        if self.verbose:
            self.trace_func(f'Validation loss decreased ({self.val_loss_min:.6f} --> {val_loss:.6f}).  Saving model ...')
        torch.save(model.state_dict(), self.path)
        self.val_loss_min = val_loss
```

4. Model(GRU + CNN)

```
In [30]:  def train(model, optimizer, train_loader, device):
              model.to(device)
              criterion = nn.MSELoss().to(device)
              metric = nn.L1Loss().to(device)

              ls = []
              pred_list = []
              label_list = []
              val_loss_lst = []
              val_mae_lst = []
              loss_list = []
              early_stopping = EarlyStopping(patience = 5, verbose = True)
              for epoch in range(100):
                  model.train()

                  for seq, space_data, label in tqdm(train_loader):
                      seq = seq.type(torch.float32).to(device)

                      space_data = torch.reshape(space_data, (-1,1,11,11 )).to(device)
                      space_data = space_data.type(torch.float32)

                      label = label.type(torch.float32).to(device)

                      pred = model(seq, space_data)

                      loss = criterion(pred.squeeze()/1000, label/1000)
                      loss_list.append(loss.item())
                      loss.backward()
                      optimizer.step()
                      optimizer.zero_grad()

                      pred_list.extend(pred.squeeze().cpu().detach().numpy())
                      label_list.extend(label.squeeze().cpu().detach().numpy())

                      del pred
                      del seq
                      del space_data
                      del label

                  for val_seq, val_space_data, val_label in tqdm(val_loader):
                      val_seq = val_seq.type(torch.float32).to(device)
                      val_space_data = torch.reshape(val_space_data, (-1,1,11,11 )).to(device)
                      val_space_data = val_space_data.type(torch.float32)

                      val_label = val_label.type(torch.float32)
                      val_pred = model(val_seq, val_space_data).cpu()
```

목차

5. Test

```python
class myDataset_mrd(Dataset):
    def __init__(self, df):

        self.seq_list1 = list(df['Rsod_seq'])
        self.seq_list2 = list(df['Patv_seq'])

        self.seq_list3 = list(df['Etmp_seq'])
        self.seq_list4 = list(df['Itmp_seq'])

        self.seq_list5 = list(df['Patv_space'])


        self.label_list = df.target.values

    def __getitem__(self, index):

        seq = np.vstack((self.seq_list1[index], self.seq_list2[index], self.seq_list3[index], self.seq_list
        seq = np.array(seq).astype('float')

        image = np.array(self.seq_list5[index]).astype('float')
        image.resize(11, 11, 1)

        label = np.array( self.label_list[index] ).astype( 'float' )

        seq = torch.tensor(seq, device = 'cuda')
        space_data = torch.tensor(image, device = 'cuda')

        return seq, space_data, label


    def __len__(self):
        return len(self.seq_list1)
```

```python
def pred_mrd(model, test_loader,device):
    model.to(device)
    model.eval()

    pred_list = []

    for seq, space_data, label in tqdm(test_loader):
        seq = seq.type(torch.float32).to(device)
        space_data = torch.reshape(space_data, (-1,1,11,11)).to(device)
        space_data = space_data.type(torch.float32)
        label = label.type(torch.float32).to(device)

        pred = model(seq, space_data)

        pred_list.extend(pred.squeeze().cpu().detach().numpy())

    return pred_list
```

인하대학교
INHA UNIVERSITY

5. Test

```
In [ ]: def forecast():
            device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

            os.environ['CUDA_VISIBLE_DEVICES'] = "0"


            test_x = pd.read_csv("train_data.csv")
            test_x = test_x.fillna(0)
            test_x = test_x.sort_values(['TurbID', 'Day', 'Tmstamp'], ascending=True).reset_index(drop=True)

            model = GRU()

            model.load_state_dict(torch.load('ourbest2batch32shuffleFtrain_data_normal_lr35scaling22yeswpdscale.ptparams'))



            path_to_distdict = 'dist_dict.pickle'
            with open(path_to_distdict, 'rb') as f:
                dist_dict = pickle.load(f)

            df_group = test_x[['Day','Tmstamp','Patv']].groupby(['Day','Tmstamp'], as_index=False).agg(list)
            print(df_group)
            df_group.columns = ['Day', 'Tmstamp','Patv_list']

            for turbid in range(134):

                turb_data = test_x[test_x['TurbID']==turbid+1].reset_index(drop=True)
                turb_data = pd.merge(turb_data, df_group, how='left', on=['Day', 'Tmstamp'])
                turb_data['Patv_space'] = turb_data['Patv_list'].apply(lambda x: [x[i-1] for i in dist_dict[turbid+1][:121]])

                df_test = pd.DataFrame({'Wspd_seq':[list(turb_data['Wspd'])[-144:]],
                                        'Patv_seq':[list(turb_data['Patv'])[-144:]],
                                        'Etmp_seq':[list(turb_data['Etmp'])[-144:]],
                                        'Itmp_seq':[list(turb_data['Itmp'])[-144:]],
                                        'Wdir_seq':[list(turb_data['Wdir'])[-144:]],
                                        'Patv_space':[list(turb_data['Patv_space'])[-1]],
                                        'target':[[0]*288],})

                df_test['tmp_seq'] = df_test.apply(get_diff, axis=1)
                if turbid==0:
                    test_new = df_test.copy()
                else:
                    test_new = pd.concat([test_new, df_test])


            test_dataset = WPFDataset_gru(test_new)
            test_loader = DataLoader(test_dataset,
                                     batch_size=32,
                                     shuffle=False,
                                     num_workers=0)

            pred_list = pred_gru(model,test_loader,device)
```

목차

감사합니다.