

머신러닝 프로그래밍

프로젝트 보고서

- IMU센서 데이터를 이용한 간단한 동작 분류

산업인공지능학과 대학원

2023254015

장욱진

목차

1. 서론

- a. 연구배경 및 목표

2. 데이터 전처리

- a. 데이터셋
 - b-1. 데이터 특성 파악
 - b-2. 데이터 라벨링
 - b-3. XYZ 데이터 만들기
 - b-4. 결측값 처리
 - b-5. 데이터 전처리
 - b-6. 데이터 전처리_스케일링

3. CNN 모델 기본 구조 파악하기

4. 학습에 필요한 변인

- a. 필터 수 변경해보기
- b. dropout 값 변경해보기
- c. epoch 수 조절해보기
- d. batch size 조절해보기
- e. 최종코드

5. 학습결과

6. 실패원인 예측

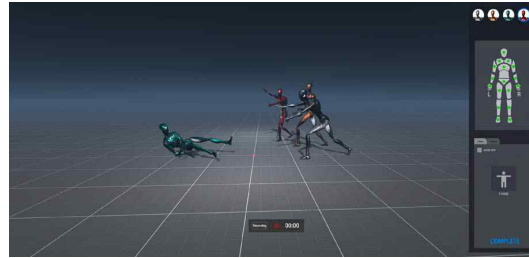
7. 실패원인 개선해보기

8. 결과비교

9. Reference

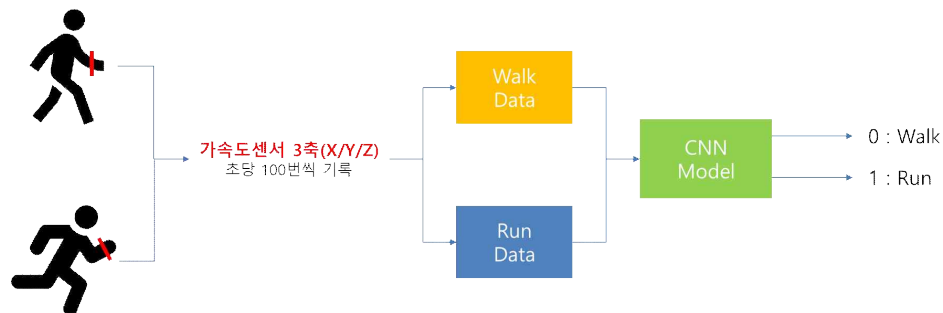
1. 서론

a. 연구배경 및 목표



IMU센서를 이용한 모션캡처

- IMU 센서를 이용한 모션캡처 연구를 진행하고 있으며, 2023년 1학기 머신러닝 프로그래밍 수업에서 배운 CNN모델을 이용하여 모션 데이터를 활용한 동작 분류 프로젝트를 진행하였다.
- IMU(Inertial Measurement Unit)란 관성을 측정하여 최종적으로 구하고자 하는 값은 물체가 기울어진 각도를 정확하게 측정하는 관성 측정 장치이다.
- 자이로스코프(각속도계)/가속도계/지자기센서로 구성된 센서로 구성되며, 사람의 감각 기관 중 귀속의 세반고리관 같은 역할을 한다. 이 장치에서 얻어진 데이터로 물체의 움직임을 읽어낸다.
- 각 센서는 관성을 이용해 물리량을 측정하는데, 이 장치로부터 이용할 수 있는 물리량은 다음과 같다.
- 자이로스코프는 각속도(rad/s)를 측정하고 시간당 몇 도(degree)를 회전했는지가 필요할 때 사용한다.
- 가속도계는 가속도(m/s^2)를 측정하고 초기값을 계산할 때 중력 가속도를 분해하여 얼마나 기울어졌는지를 측정하는데 사용하며, 속도와 이동거리를 가속도를 적분해서 사용할 수 있다.
- 지자기 센서는 지자기(magnet)를 측정한다. 자북을 기준으로 자기선속의 세기를 측정하여 자북을 기준으로 얼마나 틀어졌는지를 측정한다.



프로젝트 목표

- 이번 프로젝트에서는 IMU센서를 손목에 착용하고, 걷기, 달리기, 앉기, 눕기 등 다양한 동작을 하여 취득한 모션데이터를 학습시킨 후 특정 데이터를 입력하였을 때 어떤 동작인지 분류하는 모델을 만드는 것을 목표로 한다.

2. 데이터 전처리

a. 데이터셋

PAMAP2 Physical Activity Monitoring Data Set

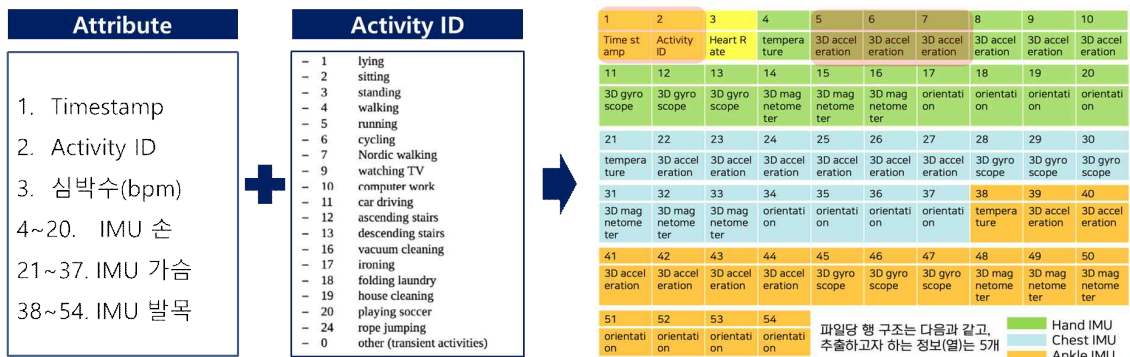
Download: [Data Folder](#), [Data Set Description](#)

Abstract: The PAMAP2 Physical Activity Monitoring dataset contains data of 18 different physical activities, performed by 9 subjects wear

Data Set Characteristics:	Multivariate, Time-Series	Number of Instances:	3850505	Area:	Computer
Attribute Characteristics:	Real	Number of Attributes:	52	Date Donated	2012-08-06
Associated Tasks:	Classification	Missing Values?	Yes	Number of Web Hits:	95176

사용한 데이터셋

- 이번 프로젝트에서 사용한 데이터셋은 9명의 피실험자가 3개의 IMU센서를 착용하고 18가지의 다른 신체활동을 한 결과를 측정한 데이터이다.
- 3개의 센서는 각각 손목, 가슴, 발목에 착용하여 측정하였으며, 1초에 100개의 데이터를 수집한다.
- 이번 프로젝트에서는 손목에 착용한 IMU센서 중 가속도센서 값만을 사용하여 학습에 이용하고자 한다.
- 데이터셋은 아래 그림과 같은 프로토콜로 정의되어 있다.

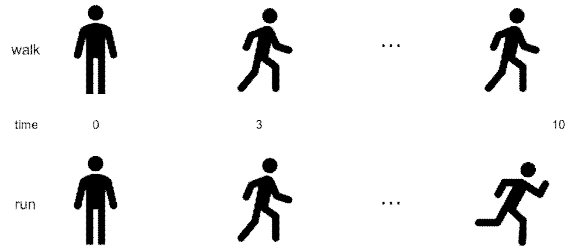


데이터셋 프로토콜

- 1행은 100Hz 단위로 기록(1초는 100행)되며, 손목의 가속도 센서 값만을 사용하므로 54열 중 Timestamp, Activity ID, 3D Acceleration 열 만을 사용한다.

b-1. 데이터 특성 파악

- 시계열 데이터를 학습시키기 위해서는 데이터를 같은 크기의 하나의 배치로 잘라야 한다.
- 데이터를 몇 초 단위로 나눌 것인지는 매우 중요한 문제이다. 나뉘진 데이터는 행동의 특성을 잘 반영할 수 있어야 한다.



걷기/달리기 구분의 문제점

- 예를 들어 걷기와 달리기 데이터를 구분한다고 하였을 때, 맨 처음 걷기 시작할 때와 달리기 시작할 때의 속력은 비슷하기 때문에 맨 처음 3초만 봐서는 달리기인지 걷기인지 구분이 어렵다.
- 이 외에도 피실험자마다 움직임이 다르며, 여러 가지 동작을 같은 단위로 잘랐을 때 동작 특성을 반영하기 어렵다는 문제점이 있다.
- 따라서 데이터를 분석하여 적절한 단위로 자르기 위해 그래프로 그려보았다.
- 아래는 그래프를 그리기 위한 코드이다.

```
import matplotlib.pyplot as plt

#파일 불러오기 -> 파일경로만 다르게 바꾸면 한번에 처리가능
f = open("subject101.dat의 경로를 입력")

#Open subject101's data separated with space(공백) / 데이터 프레임으로
df = pd.read_table(f, sep=" ", header=None)

#time stamp, activity_id, 3D-acceleration data (ms-2), scale: ±16g, resolution: 13-bit
#필요 없는 정보 제거
wanted_data = df[[1,4,5,6]]

#column이름 변경
wanted_data.columns = ['Activity_ID', 'X', 'Y', 'Z']

# activity num ( 1~24 ) : num 0번은 버리는 정보
activity_ids = range(1,25)

# activity num별로 1~24개의 dataframe(?) 생성 (file명 : file_activity_ids)
import sys
mod = sys.modules[__name__]

for i in activity_ids:
    Activity_num = wanted_data['Activity_ID'] == i
    setattr(mod, 'file_{}'.format(i), wanted_data[Activity_num])
```

```

#원하는 시간입력
time = 3000

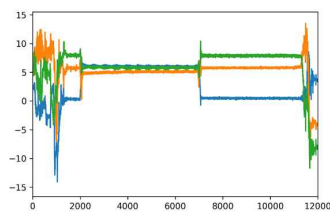
for j in activity_ids:

    #file_j 파일 불러와서 current file에 저장
    current_file = getattr(mod, 'file_{}'.format(j))
    current_file = current_file[['X', 'Y', 'Z']]
    current_file = current_file.reset_index(drop=True)

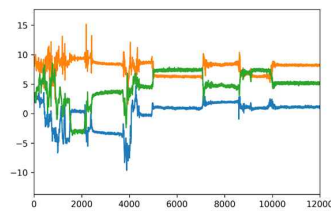
    #draw data
    fig, ax = plt.subplots(1, 1)
    ax.plot(current_file)
    ax.set_xlim([0, 4*time]) #0~12000행 즉, 2분의 데이터

    #그래프 저장
    dst = './'+str(j) + '.png'
    plt.savefig(dst, dpi=300)

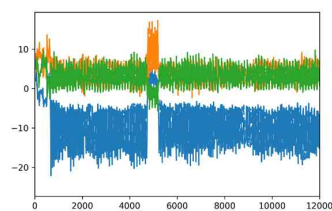
```



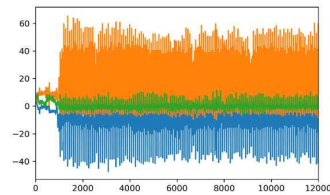
1. Lying



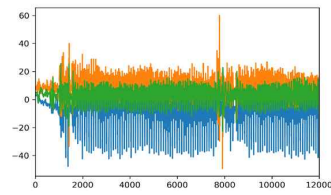
2. Sitting



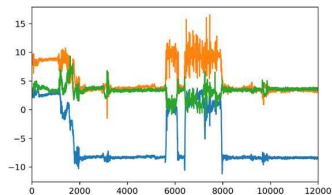
3. Walking



4. Running



5. Rope Jumping

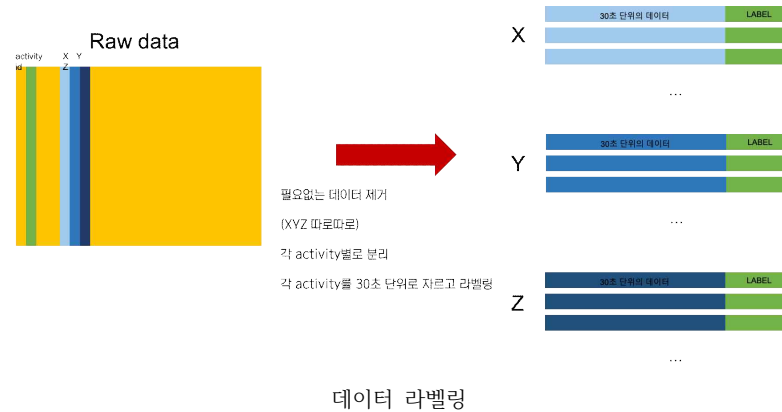


6. Standing

데이터 그래프

- 데이터를 통해 각 행동의 그래프의 개형을 파악하고 특성을 숙지하여 데이터를 자르는 것이 중요하다.
- 위 그래프를 통해 30초 단위로 자르면 동작 분류가 가능하다고 판단하였다.

b-2. 데이터 라벨링



- 라벨링이란 사용자가 원하는 정보를 붙여서 정리하는 것이다.
- 여기서 ActivityID는 필수이며, 그 외에 subject number, timestamp 등 그 외 추가 기록 할만한 사항들을 기록한다.

```
#Activity_ids ,활동 번호 / 데이터프레임으로 x 제작
activity_list = {"ID" : [0,1,2,3,4,5,6,7,9,10,11,12,13,16,17,18,19,20,24],
                 "Activity" : ["other","lying","sitting","standing","walking","running","cycling","Nordic walking","watching
TV","computer work","car driving","ascending stairs","descending stairs","vacuum cleaning","ironing","folding
laundry","house cleaning","playing soccer","rope jumping"]}
```

```
act_ids=pd.DataFrame(activity_list)
```

- activity_id를 이용하여 activity_name을 라벨링하기 위해 activity_id와 activity_name을 데이터 프레임으로 만들어 준다.
- 54열의 데이터 중 손목데이터를 제외한 데이터를 제거하고 dataframe df에서 time stamp / activity _ id / 3축 가속도 센서(X,Y,Z) 부분. 총 5개의 열만 남겨 wanted_data라는 새로운 데이터프레임을 만들었다.

7 wanted_data

	TimeStamp	Activity_ID	X	Y	Z
0	8.38	0	2.37223	8.60074	3.51048
1	8.39	0	2.18837	8.56560	3.66179
2	8.40	0	2.37357	8.60107	3.54898
3	8.41	0	2.07473	8.52853	3.66021
4	8.42	0	2.22936	8.83122	3.70000
...
376412	3772.50	0	2.02477	7.29553	5.74194
376413	3772.51	0	2.10836	7.86504	5.85674
376414	3772.52	0	2.07163	8.39581	5.77742
376415	3772.53	0	2.19569	8.77634	6.00892
376416	3772.54	0	2.14774	8.66047	5.73918

376417 rows x 5 columns

TimeStamp Activity_ID X Y Z

2928	37.66	1	2.21530	8.27915	5.58753
2929	37.67	1	2.29196	7.67288	5.74467
2930	37.68	1	2.29090	7.14240	5.82342
2931	37.69	1	2.21800	7.14365	5.89930
2932	37.70	1	2.30106	7.25857	6.09259
...
30110	309.48	1	2.12997	9.24242	2.92737
30111	309.49	1	2.16723	9.16625	2.92819
30112	309.50	1	2.27471	9.27752	2.73698
30113	309.51	1	2.20261	9.20323	2.85162
30114	309.52	1	2.20260	9.39259	2.81249
...
348888	3497.26	24	3.06119	8.38022	4.90709
348889	3497.27	24	2.88107	8.49751	5.13492
348890	3497.28	24	2.75112	8.22912	4.71051
348891	3497.29	24	2.75115	7.09295	4.94527
348892	3497.30	24	2.94035	7.69752	4.83060
...
361795	3626.33	24	2.51550	7.02650	5.78869
361796	3626.34	24	2.50643	6.30465	5.67552
361797	3626.35	24	2.54102	5.84908	5.67758
361798	3626.36	24	2.65866	5.88715	5.79468
361799	3626.37	24	2.51044	6.11629	5.83017

필요한 데이터만 자르기

- 이후 ActivityID별로 데이터를 저장하고 이것을 30초 단위로 자른다.

b-3. XYZ 데이터 만들기

```
#X축을 30초씩 끊어서 저장할 dataframe생성
data_X = pd.DataFrame()

#file_activity_ids를 file_1~file_24까지
for j in activity_ids:

    #file_j 파일 불러와서 current_file에 저장
    current_file = getattr(mod,'file_{}'.format(j))

    #file의 길이가 0이면 해당 동작을 수행하지 않았으므로 제외
    if current_file.index.size!=0:

        #시작 인덱스 = 0, 끝 인덱스 = 길이 -1
        start = 0
        end = current_file.index.size-1

        i = start

        while i < end:

            #30초 단위로 잘라 thirty_sec에 저장
            #100Hz이므로 30초면 3000개
            s = i
            f = s+3000

            #3000개 행(30초), X열(2열) 추출
            thirty_sec = current_file.iloc[s:f,2]

            #index 번호 초기화
            thirty_sec=thirty_sec.reset_index(drop=True)

            #행을 열로 변경해서 3000행-> 3000열을 가진 1개의 행으로 변경
            thirty_sec=thirty_sec.T

            #라벨링 label

            #추가정보 1 : activity_id
            thirty_sec['activity_id'] = j

            #추가정보 2 : 해당 activity_id가 무슨 동작인지 미리 만들어진 표에서 가져옴
            need = act_ids.loc[act_ids['ID']==j]
            thirty_sec['activity_name'] = need.iloc[0,1]

            #피실험자
            thirty_sec['subject_no'] = f_name

            #108번이 왼손잡이일때 넣어줌
            #if f_name == "subject108":
            #    thirty_sec['extra_info'] = "left_handed"

            #thirty_sec을 빈 데이터 프레임 data_X에 저장

            #저장할 dataframe에 행 추가
            data_X = data_X.append(thirty_sec, ignore_index=True)

            #끝 index를 시작 index로
            i = f

            #새로운 시작 index가 file_j의 맨 마지막 index보다 커지면 종료
            if end<i:
                break
```

- 저장해 둔 데이터를 불러와 30초 단위로 자르고, 라벨을 ActivityID, ActivityName, Subject, Number로 넣는다.
- 위 코드를 실행하면 X, Y, Z를 따로따로 만들어 낼 수 있다.

value 0~2999 까지 3000개										추가정보들		
2991	2992	2993	2994	2995	2996	2997	2998	2999	activity_id	activity_name	subject_no	
1.171500	6.169630	6.20475	6.202070	6.09191	6.054920	6.16589	6.08389	6.053320	1.0	lying	subject101	
1.826580	6.128090	6.01606	6.010450	5.97587	6.052780	5.90884	6.00724	5.938880	1.0	lying	subject101	
1.506389	0.385248	0.35147	0.610377	0.43186	0.506102	0.57687	0.46243	0.349866	1.0	lying	subject101	
1.346190	3.531130	3.53567	3.494670	3.64262	3.611790	3.61714	3.54102	3.579350	1.0	lying	subject101	
1.728360	3.726490	3.73023	3.760800	3.68924	3.694320	3.76776	3.87310	3.798320	1.0	lying	subject101	
...	
1.096950	4.313270	3.65320	2.280490	NaN	NaN	NaN	-1.88317	-1.560450	24.0	rope jumping	subject101	
1.153100	-25.643700	-34.20930	-38.023300	-32.43230	-27.054100	-24.60230	-22.48620	-20.170500	24.0	rope jumping	subject101	
1.098690	-10.595100	-11.54580	-11.440700	-11.21940	-10.790600	-10.47010	-10.23500	-9.006110	24.0	rope jumping	subject101	
1.939920	-5.612920	-6.54761	-7.912660	-9.18929	-10.961700	-13.09260	-13.65970	-12.085200	24.0	rope jumping	subject101	
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	24.0	rope jumping	subject101	

데이터 만들기 결과


```

#파일명
file_out_X='/' + f_name + '_protocol_X.csv'
file_out_Y='/' + f_name + '_protocol_Y.csv'
file_out_Z='/' + f_name + '_protocol_Z.csv'

#subject101의 X축 데이터 csv로 저장
data_X.to_csv(file_out_X,header=False, index=False)

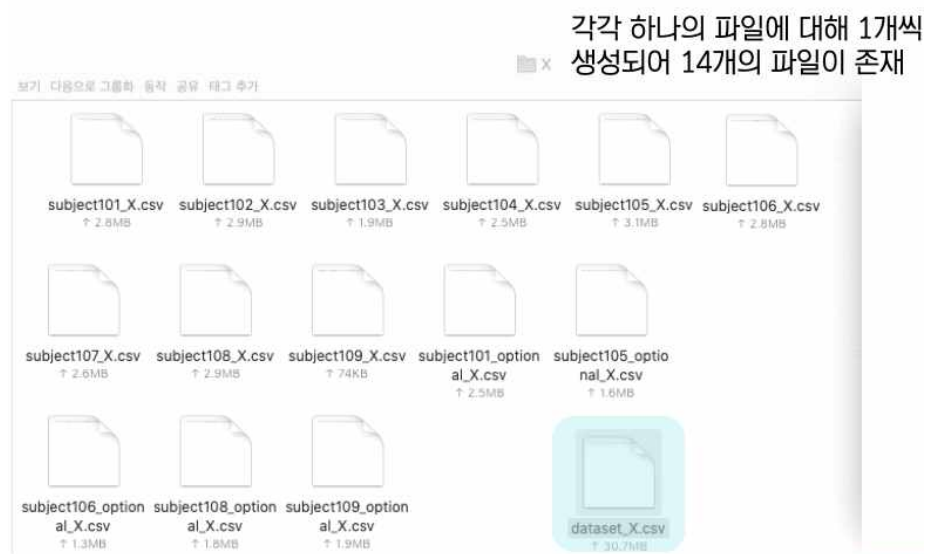
#subject101의 Y축 데이터 csv로 저장
data_Y.to_csv(file_out_Y,header=False, index=False)

#subject101의 Z축 데이터 csv로 저장
data_Z.to_csv(file_out_Z,header=False, index=False)

```

X/Y/Z csv파일 불러오기

- protocol version 9개 / optional version 6개로 따로따로 만들어진 csv파일을 1개의 파일로 합친다.



csv파일 합치기

- 전처리 결과 960행 3000열(+3 Label 열) 의 X/Y/Z 데이터를 얻을 수 있다.

#	DIX	DJY	DIZ	DKA	DKB	DKC	DKD	DKE	DKF	DKG	DKH	DKI	DKJ	DKK	DKL	DKM
9	1.27014	1.43000	1.43000	1.20000	1.30000	1.30000	1.30000	1.30000	1.30000	1.30000	1.30000	1.30000	1.30000	1.30000	1.30000	1.30000
10															1 lying	subject101
11	-3.43425	-3.40555	-3.29646	-3.33532	-3.33131	-3.29379	-3.25519	-3.28443	-3.25519	-3.37284	-3.29138	-3.37044	-3.25974		2 sitting	subject101
12	0.980775	0.78969	0.943252	0.938974	0.863395	0.860454	1.08879	0.938441	0.942185	0.863395	0.826143	0.823468	0.865802		2 sitting	subject101
13	1.04779	1.27319	1.19601	1.07782	1.12069	1.15554	0.888076	1.04351	1.19814	1.07916	1.03763	1.23058	1.04886		2 sitting	subject101
14	1.18054	1.18215	0.87181	1.21967	1.17787	1.13821	1.06798	1.33544	1.10737	1.14088	1.25773	1.17814	1.26414		2 sitting	subject101
15	0.782248	0.776899	0.815489	0.821108	0.85649	0.701857	0.628419	0.741784	0.857294	0.895987	0.810411	0.933676	0.975207		2 sitting	subject101
16	-7.87104	-7.72844	-7.7582	-8.02674	-7.83164	-7.86809	-7.83298	-7.83779	-7.87264	-7.82656	-7.8271	-7.83218	-7.8295		2 sitting	subject101
17	-7.8083	-7.91819	-7.80563	-7.84208	-7.87425	-7.95224	-7.87425	-7.87238	-7.87398	-7.91605	-7.88094	-7.76677	-7.91579		2 sitting	subject101
18															2 sitting	subject101
19	-7.76597	-7.61054	-7.65126	-7.61508	-7.58504	-7.6534	-7.69814	-7.76918	-7.84529	-7.88441	-7.76944	-7.77345	-7.81579		3 standing	subject101
20	-0.0871689	-0.05461	-0.287331	-0.313357	-0.450091	-0.183425	0.0215604	0.288224	0.316928	0.585478	0.654173	0.655724	0.917051		3 standing	subject101
21	-8.42683	-8.16952	-8.27888	-8.16043	-8.28263	-8.16338	-8.23949	-8.20464	-8.20972	-8.23494	-8.2738	-8.35526	-8.20838		3 standing	subject101
22	-8.50053	-8.58787	-8.53912	-8.46488	-8.3065	-8.26952	-8.38556	-8.5016	-8.34242	-8.38368	-8.39278	-8.27379	-8.45953		3 standing	subject101
23	3.17968	3.10223	3.02745	3.18155	3.03146	3.07139	3.18797	3.26488	3.03093	3.1475	3.07487	3.1049	3.17861		3 standing	subject101
24	-7.0308	-7.02625	-7.25646	-7.36823	-7.43498	-7.40254	-7.44006	-7.44808	-7.63515	-7.63355	-7.51991	-7.53061	-7.52714		3 standing	subject101
25	1.84306	1.9575	1.84145	1.76587	1.99689	1.77443	1.92238	1.80446	1.80714	1.84653	1.84605	1.84466	1.80661		3 standing	subject101
26															3 standing	subject101
27	-7.64615	-7.23475	-6.82094	-6.42612	-6.23985	-6.09831	-5.69306	-5.39341	-5.3189	-5.32746	-5.08869	-5.1329	-5.02113		4 walking	subject101
28	-4.22512	-4.22512	-4.13618	-4.13778	-4.16675	-4.2769	-4.46291	-4.83892	-4.97831	-5.0789	-6.06029	-7.02785	-8.03934		4 walking	subject101
29															4 walking	subject101
30	-13.3997	-12.8034	-12.0875	-11.1998	-10.3716	-9.7407	-9.11673	-8.36254	-7.98706	-7.72976	-7.42745	-7.13155	-7.05784		4 walking	subject101
31	-3.12365	-3.29148	-3.64746	-4.19854	-5.00238	-6.17395	-7.53607	-8.85665	-9.58999	-10.2445	-10.8025	-11.6326	-12.5701		4 walking	subject101
32	-8.82497	-9.41597	-9.00644	-8.52535	-8.26403	-8.04293	-7.78617	-7.47531	-7.28957	-6.88378	-6.71795	-6.58574	-6.36649		4 walking	subject101
33	-14.8085	-15.0843	-15.1652	-15.1692	-14.9101	-14.4686	-14.2536	-13.872	-13.4233	-13.2009	-12.6474	-11.8238	-10.9192		4 walking	subject101
34															4 walking	subject101
35	-8.96762	-8.85721	-9.09768	-9.5418	-9.97817	-10.5429	-10.7072	-10.3917	-11.2348	-13.6575	-17.8654	-26.5863	-37.6843		5 running	subject101
36	5.00825	5.70863	6.33369	6.96409	7.41624	7.67462	7.79012	7.37791	7.23182	7.05383	7.01924	6.95622	6.54775		5 running	subject101
37	-10.3567	-11.1388	-11.6796	-12.2687	-12.6914	-13.7064	-17.3829	-27.8567	-34.6957	-32.7887	-31.3417	-27.5265	-21.3289		5 running	subject101
38	-12.0407	-12.9724	-13.5315	-14.4735	-15.4363	-17.9054	-23.4317	-33.0827	-41.4212	-39.237	-37.0138	-31.191	-21.64		5 running	subject101
39	-10.4709	-10.5717	-10.6744	-10.6672	-11.0159	-11.632	-12.4706	-13.9455	-17.637	-25.1018	-32.8619	-32.2495	-30.0481		5 running	subject101
40																

데이터 전처리

b-4. 결측값 처리

- 데이터 전처리 중 데이터가 측정되지 않아 빈칸으로 채워진 데이터가 있음을 알게 되었다.
- 결측값 처리에는 결측값을 삭제하는 방법, 주변값의 평균치를 채우는 방법 등 여러 가지 방법이 있지만, 이번 프로젝트에서는 최솟값을 모든 데이터에 더해주어 최솟값을 0으로 만들어주는 방법을 사용하였다.

-4.13618	-4.13778	-4.16675	-4.2769	-4.46291	-4.83892	-4.97831	-5.50789	-6.06029	-7.02785	-8.03934	4 walking	subject101			
-7.23051			-9.39931	-10.2886	-11.34	-12.1285		-11.9114	-11.5503	-11.4152	4 walking	subject101			
-12.0875	-11.1998	-10.3716	-9.7407	-9.11673	-8.36254	-7.98706	-7.72976	-7.42745	-7.13155	-7.05784	4 walking	subject101			
-3.64746	-4.19854	-5.00238	-6.17395	-7.53607	-8.85665	-9.58999	-10.2445	-10.8025	-11.6326	-12.5701	4 walking	subject101			
33	-14.8085	-15.0843	-15.1652	-15.1692	-14.9101	-14.4686	-14.2536	-13.872	-13.4233	-13.2009	-12.6474	-11.8238	-10.9192	4 walking	subject101
34														4 walking	subject101
	-8.96762	-8.85721	-9.09768	-9.5418	-9.97817	-10.5429	-10.7072	-10.3917	-11.2348	-13.6575	-17.8654	-26.5863	-37.6843	5 running	subject101

↓ 결측값 처리



결측값 처리 과정

#파일 불러오기

```
dataset_X= pd.read_csv("dataset_X.csv", header=None)
dataset_Y= pd.read_csv("dataset_Y.csv", header=None)
dataset_Z= pd.read_csv("dataset_Z.csv", header=None)
```

데이터 불러오기

- XYZ를 전부 불러와서 데이터 프레임으로 만들어 준다.

```
#각 열에서 최솟값만을 구해서구해 Series로 구성
min_set_X = pd.Series(dataset_X.min())
min_set_Y = pd.Series(dataset_Y.min())
min_set_Z = pd.Series(dataset_Z.min())
```

데이터 최솟값 구하기

- min() 함수를 이용하여 각 열에서의 최솟값을 구할 수 있다.

```
#원본 데이터의 0~2999열에 최솟값의 절댓값을 더 해줌
dataset_X.iloc[:,0:3000] = dataset_X.iloc[:,0:3000] + abs(min_X)
dataset_Y.iloc[:,0:3000] = dataset_Y.iloc[:,0:3000] + abs(min_X)
dataset_Z.iloc[:,0:3000] = dataset_Z.iloc[:,0:3000] + abs(min_X)
```

최솟값 더해주기

- 이제 모든데이터에 라벨을 제외한 부분에 최솟값을 더해준다.
- 위의 최솟값 구하기를 다시 실행했을 때 최솟값이 0이면 성공이다.

```
#결측값 -1로 채우기
dataset_X = dataset_X.fillna(-1)
dataset_Y = dataset_Y.fillna(-1)
dataset_Z = dataset_Z.fillna(-1)
```

결측값 채우기

- 이후 데이터가 측정되지 않아 빈칸으로 채워진 부분을 -1로 채운다.

```
#dataset_?_m.csv로 저장
dataset_X.to_csv("/dataset_X_30sec_m.csv",h
eader=False, index=False)
dataset_Y.to_csv("/dataset_Y_30sec_m.csv",h
eader=False, index=False)
dataset_Z.to_csv("/dataset_Z_30sec_m.csv",h
eader=False, index=False)
```

데이터 저장하기

b-5. 데이터 전처리

```
#dataframe의 길이만큼 반복하기 위해 필요한 변수
length = dataset_X.index.stop

#X, Y에서 label을 제거
dataset_X = dataset_X.iloc[:,0:3000]
dataset_Y = dataset_Y.iloc[:,0:3000]
dataset_Z = dataset_Z.iloc[:,0:3003] #마지막 z는 label이 포함됨
```

데이터셋 다듬기

- X, Y, Z 모두에 붙은 라벨을 Z에 붙은 라벨만 남긴다.

```
#X, Y, Z, label 을 한줄로 데이터 합치기
xyz_data = pd.DataFrame()

for i in range(0,length):
    x = dataset_X.iloc[i,:]
    y = dataset_Y.iloc[i,:]
    z = dataset_Z.iloc[i,:]

    h = pd.concat([x,y],ignore_index=True)
    h = pd.concat([h,z],ignore_index=True)

    xyz_data = xyz_data.append(h,ignore_index=True)
```

한줄씩 한 행으로 합치기



데이터 전처리

- 결측값을 처리하고 데이터 라벨링까지 완료하여 위 그림과 같은 데이터 형태로 처리하였다.
- 데이터 전처리 결과 24가지 행동에 대해 30초 단위로 자른 960개의 데이터가 생성되었다.

- b-4. Training/ Validation/Test Set 나누기



데이터셋 나누기

- 데이터는 학습을 위한 학습데이터, 과적합 방지를 위한 검증데이터, 최종모델 테스트를 위한 테스트 데이터가 필요하다.
- 검증데이터는 테스트 데이터를 잘 대변할 수 있어야하며, 테스트 데이터는 데이터셋 이외의 실제 데이터를 입력하였을 때를 잘 대변할 수 있어야 한다.
- 대부분 학습데이터:검증데이터:테스트 데이터로 5:2:3 비율로 나누는 것을 사용하지만 데이터가 부족하다고 판단하여 6.5:1.5:2 비율로 나누었다.

```
xyz_data_drop = pd.read_csv("방금 만든데이터 가져오기",header=None)

#set column name
xyz_data_drop.rename(columns={4500:'act_id'},inplace=True)
xyz_data_drop.rename(columns={4501:'act_name'},inplace=True)
xyz_data_drop.rename(columns={4502:'sub_no'},inplace=True)
```

데이터 불러오기

```

#각 1~24 행동별로 dataframe 생성
import sys
mod = sys.modules[__name__]

# activity num ( 1~24 ) 까지로 파일 나누기
activity_ids = range(1,25)

for i in activity_ids:
    Activity_num = xyz_data_drop['act_id'] == i
    setattr(mod, 'file_{}'.format(i),xyz_data_drop[Activity_num])

```

Activity_id별로 나누기

```

#빈 데이터 프레임 3개
training_set = pd.DataFrame()
validation_set = pd.DataFrame()
test_set = pd.DataFrame()

for j in activity_ids:
    current_file = getattr(mod, 'file_{}'.format(j))

    #subject를 고르게 분포시키기 위하여 행 섞기
    #101~109까지 순서대로 들어가 학습데이터에는 1010이 전부들어감
    current_file = current_file.sample(frac=1).reset_index(drop=True)

    if current_file.index.size!=0:

        size = current_file.index.size

        #각 set의 비율
        train = math.floor(size*0.65)
        val = math.floor(size*0.15)
        test = math.floor(size - (train+val)) #나머지

        #6.5:1.5:3 비율로 분포시키기
        training_set = training_set.append(current_file.iloc[0:train,:],ignore_index=True)
        validation_set = validation_set.append(current_file.iloc[train:train+val,:],ignore_index=True)
        test_set = test_set.append(current_file.iloc[train+val:train+val+test,:],ignore_index=True)

```

학습데이터/검증데이터/테스트 데이터 나누기

- 인덱스를 사이즈*비율을 곱하여 지정해주는 방법으로 데이터를 나누었다.
- 위 방법을 이용하여 데이터를 나누게 되면 ActivityID별로 정렬되어 있는 데이터가 설정한 비율대로 나뉘지기 때문에 학습에 영향을 주는 것을 피하기 위해 데이터 순서 랜덤화를 진행하였다.

#데이터 섞기

```

random_train = training_set.sample(frac=1).reset_index(drop=True)
random_val = validation_set.sample(frac=1).reset_index(drop=True)
random_test = test_set.sample(frac=1).reset_index(drop=True)

```

데이터 랜덤화

#데이터 csv로 저장

```

random_train.to_csv("/r_training_set_30sec.csv",header=False, index=False)
random_val.to_csv("/r_validation_set_30sec.csv",header=False, index=False)
random_test.to_csv("/r_test_set_30sec.csv",header=False, index=False)

```

데이터 저장하기

b-6. 데이터 전처리_스케일링

```
# standardize train features
scaler = StandardScaler().fit(train.values)
scaled_train = scaler.transform(train.values)
```

데이터 전처리_스케일링

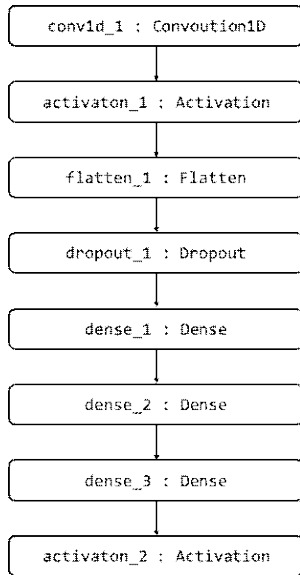
- 스케일링이란 모든 자료에 선형변환을 적용하여 전체 자료의 분포를 평균0, 분산은 1이 되도록 만드는 과정이다.
- 데이터의 값이 너무 크거나 작으면 모델 학습 과정에서 0이나 무한대로 수렴/발산할 확률이 높기 때문에 스케일링 과정을 통해 자료의 overflow, underflow를 방지하고, 최적화 과정에서 안정성과 수렴속도를 향상시킨다.

구분	설명
StandardScaler	각 feature의 평균을 0, 분산을 1로 변경. 모든 feature이 같은 스케일을 가짐
RobustScaler	모든 feature이 같은 스케일을 가짐. 평균, 분산이 아닌 median, quartile을 사용
MinMaxScaler	모든 feature이 0,1사이에 위치하게 만듦
Normalizer	각 column의 통계치가 아닌 row를 이용해서 정규화, 유클리드 거리가 1이 되도록 데이터를 조정

스케일링 방법

- StandardScaler는 평균을 제거하고 데이터를 단위분산으로 조정한다. 그러나 이상치가 있다면 평균과 표준편차에 영향을 미쳐 변환된 데이터의 확산은 매우 달라지게 된다.
- 따라서 이상치가 있는 경우 균형잡힌 척도를 보장할 수 없다.
- 결론적으로 모든 스케일러의 처리 전에는 아웃라이어 제거가 선행되어야 하며 데이터 분포 특징에 따라 적절한 스케일러를 적용해주는 것이 좋다.
- StandardScaler는 변환된 결과가 대부분 표준화된 유사형태의 데이터 분포로 반환되기 때문에 이번 프로젝트에서 스케일러로 선택하게 되었다.

3. CNN 모델 기본 구조 파악하기



```

# Keras model with one Convolution1D layer
# unfortunately more number of convolutional layers, filters and filters length
# don't give better accuracy
model = Sequential()
model.add(Convolution1D(nb_filter=512, filter_length=1, input_shape=(nb_features, 3)))
model.add(Activation('relu'))
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(2048, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dense(nb_class))
model.add(Activation('softmax'))

y_train = np_utils.to_categorical(y_train, nb_class)
y_valid = np_utils.to_categorical(y_valid, nb_class)

sgd = SGD(lr=0.01, nesterov=True, decay=1e-6, momentum=0.9)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

nb_epoch = 15
model.fit(X_train_r, y_train, nb_epoch=nb_epoch, validation_data=(X_valid_r, y_valid), batch_size=16)
  
```

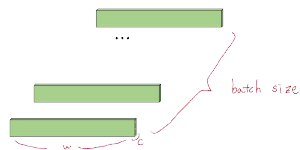
CNN모델 예제 및 구조

- 케라스에서는 add함수를 통해 간단하게 CNN모델을 제작할수 있게 여러 가지 Layer 함수를 제공한다.

a. Conv1d vs Conv2d

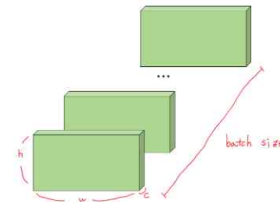
Conv1D

Input Tensor $\geq 3D$ (batch_size, w, c(depth))



Conv2D

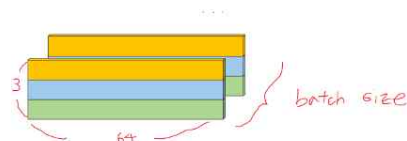
Input Tensor $\geq 4D$ (batch_size, h, w, c(depth))



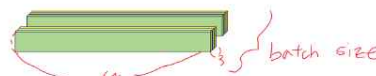
Conv1d 와 Conv2d 비교

- 위의 keras예제는 서로 다른 3개의 데이터가 각각 64열을 가지며, 이를 하나의 데이터 덩어리로 생각하여 학습시킨다.
- conv2d와 conv1d의 형식을 모두 사용할 수 있는데 둘을 비교하면 다음과 같다.

Conv2D



Conv1D



Conv1d 와 Conv2d 비교2

4. 학습에 필요한 변인

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedShuffleSplit
from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten, Convolution1D, Dropout
from keras.optimizers import SGD
from keras.utils import np_utils

#구글 드라이브와 연동
from google.colab import drive
drive.mount('/gdrive')

#데이터 불러오기
train = pd.read_csv('/gdrive/My Drive/데이터/maybe_final/r_training_set_9001.csv', header=None)
val = pd.read_csv('/gdrive/My Drive/데이터/maybe_final/r_validation_set_9001.csv', header=None)
test = pd.read_csv('/gdrive/My Drive/데이터/maybe_final/r_test_set_9001.csv', header=None)

#드롭
train.rename(columns={9000:'act_id'}, inplace=True)
val.rename(columns={9000:'act_id'}, inplace=True)
test.rename(columns={9000:'act_id'}, inplace=True)

def encode(train, val, test):
    label_encoder = LabelEncoder().fit(train.act_id)

    labels = label_encoder.transform(train.act_id)
    # y data
    train_label = train.act_id
    val_label = val.act_id
    test_label = test.act_id
    # x data
    train = train.drop(['act_id'], axis=1)
    val = val.drop(['act_id'], axis=1)
    test = test.drop(['act_id'], axis=1)
    classes = list(label_encoder.classes_)

    return train, val, test, train_label, val_label, test_label, classes

#label / data 분리
train, val, test, train_label, val_label, test_label, classes = encode(train, val, test)

# standardize train features -> 왜 하나요
scaler = StandardScaler().fit(train.values)
scaled_train = scaler.transform(train.values)

nb_features = 3000 # number of features per features type (shape, texture, margin)
nb_class = 25 #1~24|므로 [0,25) 를 사용해야 할 표현가능

# reshape train data
X_train_r = np.zeros((len(train), 3, nb_features)) # 세트수->[행, 열] : 3x30000| 6172#
X_train_r[:, 0, :] = train.iloc[:, 0:nb_features]
X_train_r[:, 1, :] = train.iloc[:, nb_features:nb_features+3000]
X_train_r[:, 2, :] = train.iloc[:, nb_features+3000:nb_features+6000]

# reshape validation data
X_valid_r = np.zeros((len(val), 3, nb_features))
X_valid_r[:, 0, :] = val.iloc[:, 0:nb_features]
X_valid_r[:, 1, :] = val.iloc[:, nb_features:nb_features+3000]
X_valid_r[:, 2, :] = val.iloc[:, nb_features+3000:nb_features+6000]

#
X_test_r = np.zeros((len(test), 3, nb_features))
X_test_r[:, 0, :] = test.iloc[:, 0:nb_features]
X_test_r[:, 1, :] = test.iloc[:, nb_features:nb_features+3000]
X_test_r[:, 2, :] = test.iloc[:, nb_features+3000:nb_features+6000]
```

전체 코드

- 전체 코드에서 본격적인 학습을 하기 전 정확도를 올리기 위해 수정해야 할 부분은 다음과 같다.


```

# Keras model with one Convolution1D layerm
# unfortunately more number of convolutional layers, filters and filters length
# don't give better accuracy
model = Sequential()
model.add(Convolution1D(filters=512, kernel_size=1, padding="same", strides=1, input_shape=(3, nb_features)))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(nb_class))
model.add(Activation('softmax'))

model.summary()

#y_train = np_utils.to_categorical(train_label, nb_class)
#y_valid = np_utils.to_categorical(val_label, nb_class)

sgd = SGD(lr=0.01, nesterov=True, decay=1e-6, momentum=0.9)
model.compile(loss='sparse_categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

nb_epoch = 30
model.fit(X_train_r, train_label, epochs=nb_epoch, validation_data=(X_valid_r, val_label), batch_size=15)

```

정확도 향상을 위해 수정해봐야 할 부분

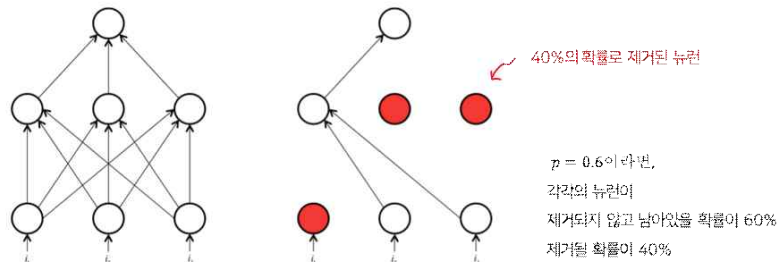
- 정확도 향상을 위해 아래 부분에 대한 수정을 통해 어떤 것을 수정했을 때 정확도가 올라가는지에 대한 실험과 고민이 필요하다.
- Convolution1D를 사용할 것인가? 2D를 사용할 것인가?
- Convolution층은 몇개로 구성할 것인가?
- 각각 층의 필터 사이즈 / 필터 수 / stride 는 어떻게 할 것인가?
- Activation층의 활성화 함수로는 어떤 것을 사용할 것인가?
- Dense층은 몇개로 구성할 것인가? 뉴런의 수는?
- Dropout은 몇으로 설정할 것인가?
- batch size는 몇으로 지정할 것인가?
- epoch는 몇으로 지정할 것인가?
- 더 나아가면 model.compile부분에서 loss함수로는 무엇을 사용할 것인가?
- 답러닝은 실험을 통해 최고의 정확도를 내는 것이기 때문에 정답은 없으며, 하나하나 변인을 통제해가며 실험해야한다.
- 변인은 다음 순서대로 실험하였다.
- Conv1d vs Conv2d -> Conv층수 -> 필터 수 -> 필터사이즈-> 배치사이즈
- 가장 높은 정확도가 나온 것을 중심으로 간격을 좁혀가며 그 부분을 실험하였고, 그 중 가장 높은 정확도가 나오면 그걸 Fix하고 다음 통제로 넘어간다.

a. 필터 수 변경해보기

- 넓은 범위에서 필터 수를 정할 때는 2^n 단위로 증가시키면서 관찰하는 것이 좋다.
- 어느 하나가 가장 좋은 정확도를 보이면 그 근처에서 조금씩 변경하며 적절한 값을 찾아내는 것이 좋다.

b. dropout 값 변경해보기

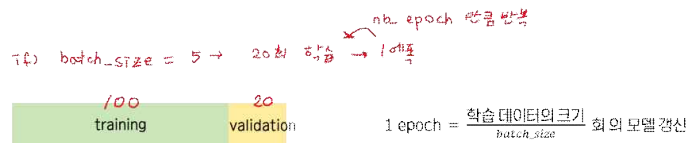
- 과적합을 막기 위한 방법 중 하나로, 어떤 확률 p (하이퍼파라미터, 해당 뉴런이 존재할 확률)로 뉴런의 활성 상태를 유지(1, 즉 출력값이 변동없이 계산된 값 그대로 출력)하거나 0(비활성화, 즉, 출력값이 0임)으로 설정하여 구현.



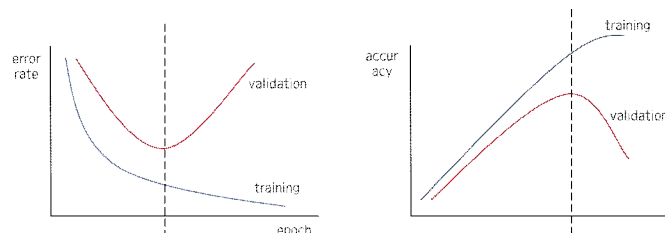
신경망의 각 뉴런을 각각의 미니배치 학습동안 임의의 확률로 비활성화(출력값 0) 함.

- 이러한 방식으로 기하급수적으로 많은 다른 신경망 구조를 간략하게 결합하는 방법을 제공하여 과적합을 방지한다.
- 특정 정보가 없을 때도 망을 정확하게 만들게하고, 망의 뉴런의 어느 하나나 매우 적은 조합에 너무 의존적이 되는 것을 방지한다.
- 보편적으로 0.3~0.5 사이의 값을 사용한다고 한다.

c. epoch 수 조절해보기



에폭수 만큼 반복하여 학습 → 너무 크면 overfitting
너무 적으면 underfitting



학습을 많이 할 수록 정확도↑ 과적합 되니까
→ 학습을 많이 해서 정확도가 높으면서도 과적합이 되지않은, 과적합 직전을 찾아야함

에폭수 조절하기

d. batch size 조절해보기

- batch_size = 1이면, 확률적 경사하강법(stochastic gradient) 이고,
- batch_size = (전체 사이즈) 이면, 경사하강법(batch_gradient)이다. (이쪽이 계산이 복잡하기 때문에 이론상 느림)
- 이에 절충안으로 나온것이 minibatch이다.
- minibatch는 보통 10~1000개로 memory사이즈에 맞추어 32, 64, 128...(2^n)을 사용한다.
- 하지만 데이터가 적을 경우 batch_gradient를 추천한다고 한다.

e. 최종코드

```
# Keras model with one Convolution1D layerm
# unfortunately more number of convolutional layers, filters and filters lenght
# don't give better accuracy
model = Sequential()
model.add(Convolution2D(filters=256, kernel_size=3, padding="valid", strides=1, input_shape=(3, nb_features, 1)))
model.add(Activation('relu'))
model.add(Dropout(0.4))
model.add(Flatten(data_format=None))
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(nb_class))
model.add(Activation('softmax'))

model.summary()

#y_train = np_utils.to_categorical(train_label, nb_class)
#y_valid = np_utils.to_categorical(val_label, nb_class)

sgd = SGD(lr=0.01, nesterov=True, decay=1e-6, momentum=0.9)
adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

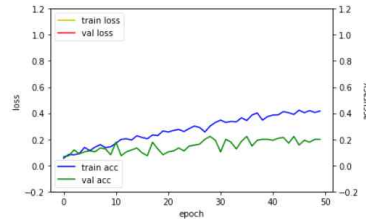
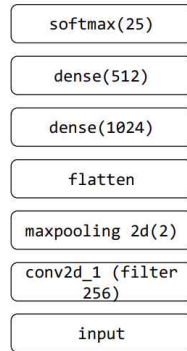
nb_epoch = 40
mhistory = model.fit(X_train_r, train_label, epochs=nb_epoch, validation_data=(X_valid_r, val_label), batch_size=128)
```

최종코드

5. 학습결과

1) Conv2d 층수 조절 결과

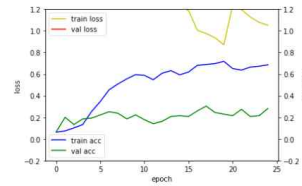
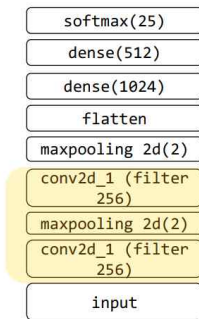
Conv2d_1(filter 256 / size 3)



Accuracy : 24.42%
time : 2s

Conv2d 층수 1개

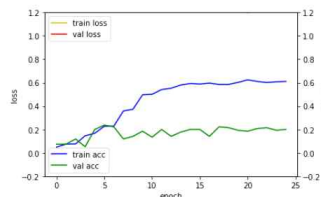
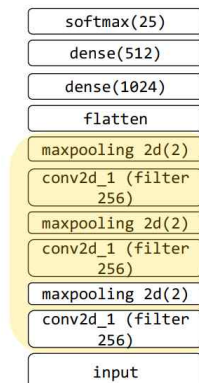
Conv2d_1 / 2(filter 256 / size 3)



Accuracy : 20.28%
time : 42s

Conv2d 층수 2개

Conv2d_1 / 2 / 3(filter 256 / size 3)



Accuracy : 19.32%
time : 28s

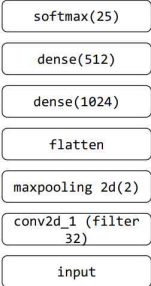
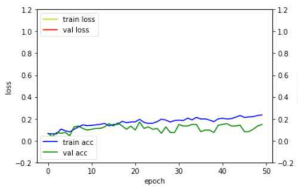
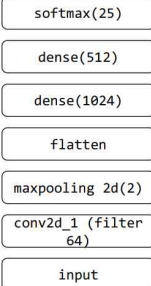
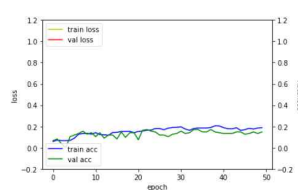

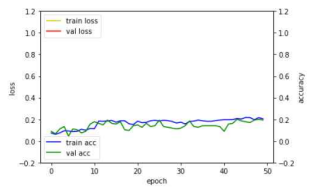
Conv2d 층수 3개

Conv 총 1개 vs 2개 vs 3개 비교

	1개	2개	3개
Accuracy(average)	24.42%	20.28%	19.32%
time(MAX)	2s 400ms	42s 8s/step	28s 6s/step

Conv2d 층수 조절 비교결과

2) 필터 수 조절 결과

Filter 32개   Accuracy : 17.86% time : 7s	Filter 64개   Accuracy : 10.67% time : 13s
Filter 128개   Accuracy : 21.13% time : 25s	Filter 512개 <p>RAM 용량을 초과하여 세션 강제 종료로 인해 실험 불가능</p>

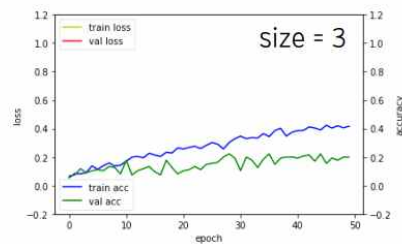
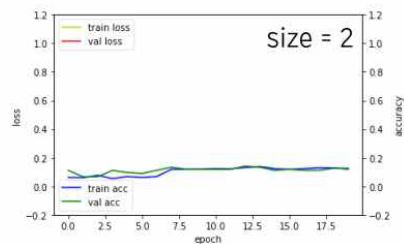
Filter 수 32 vs 64 vs 128 vs 256 vs 512 비교

	32개	64개	128개	256개	512개
Accuracy	17.86%	10.67%	21.13%	24.42%	x
time	7s	13s	26s	2s	x

3) 필터 크기 조절 결과

filter size : 1 vs 2 vs 3

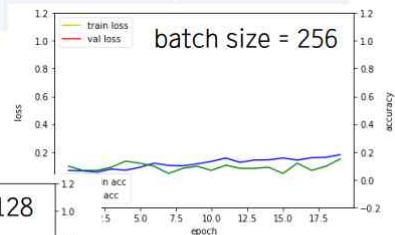
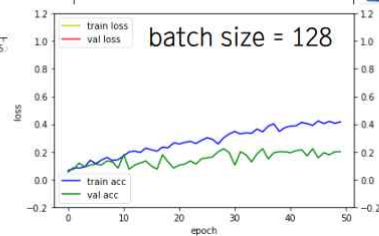
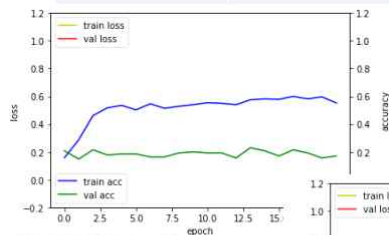
	Size 1	Size 2	Size 3
Accuracy	RAM	11.05%	24.42%
time	RAM	56s	2s



4) 배치 크기 조절 결과

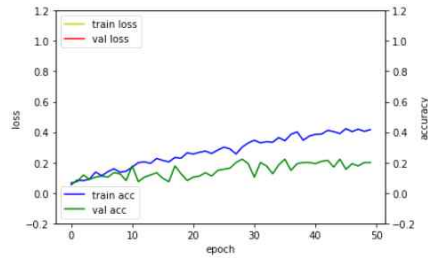
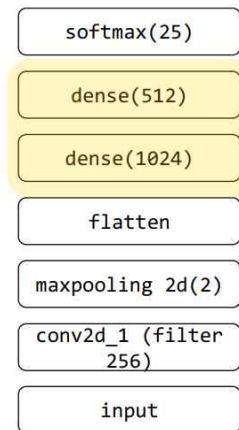
batch size 1 vs 128 vs 256 vs 617 비교

	1 확률적 경사하강법	128 미니 배치 경사하강법	256 경사하강법	617 경사하강법
Accuracy	20.51%	24.42%	18.91%	RAM
time	52s	2s	32s	RAM



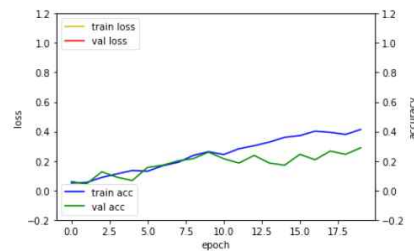
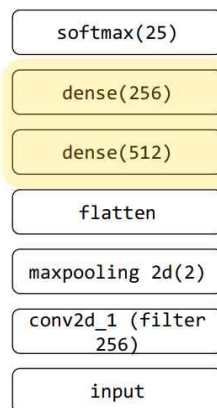
5) dense 층 조절 결과

dense(1024) dense(512)



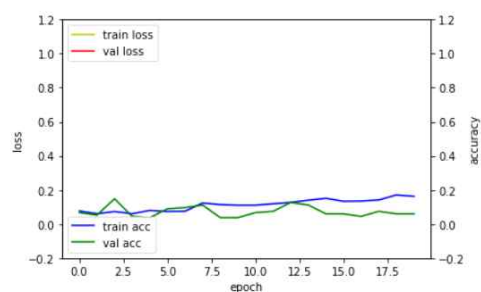
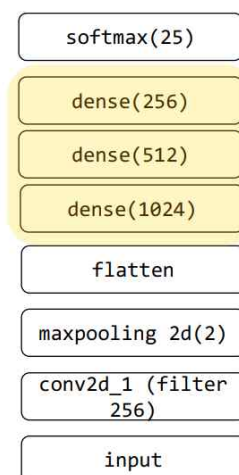
Accuracy : 24.42%
time : 2s

dense(512) dense(256)



Accuracy : 23.07%
time : 1s

dense(1024) dense(512) dense(256)



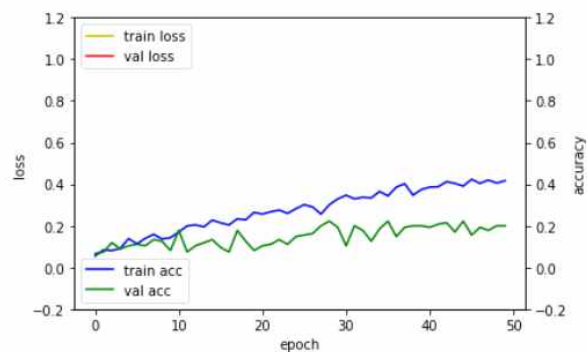
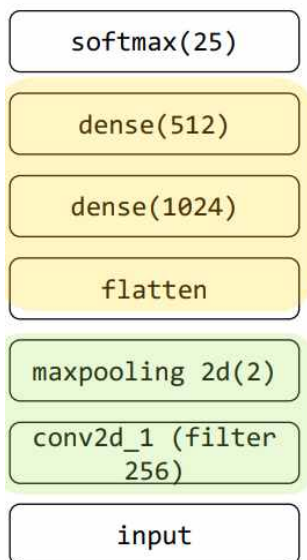
Accuracy : 13.63%
time : 1s

6) 실험결과 정리

N	Accuracy(%)	1batch_time (MAX)	drop out	Conv2_1/ filter	Conv2_1 / kernel size	Conv2_1 /stride	maxpool_2/ pool_size	maxpool_2/ stride	dropout
1	20.67%	2s 400ms		256	3	1	2	None	
2	19.23%	2s 351ms		256	3	1	2	None	
3	26.92%	2s 372ms		256	3	1	2	None	
4	24.52%	2s 401ms		256	3	1	2	None	
5	30.77%	2s 400ms		256	3	1	2	None	
	24.42%			256	3	1	2	None	

dropout	dense1	dense2	dropout	dense3	epoch	batchsize
0.4	1024	512	0.4		50	128
0.4	1024	512	0.4		50	128
0.4	1024	512	0.4		100	128
0.4	1024	512	0.4		40	128
0.4	1024	512	0.4		40	128
0.4	1024	512	0.4		50	128

실험결과 정리_1



Accuracy : 24.42%
time : 2s

실험결과 정리_2

6. 실패원인 예측

- 1) 데이터를 나눈 기준(30초)가 유의미 하지 않을 수 있음
- 2) 데이터 수가 충분하지 않음
 - 데이터 수가 총 960개 / 클래스가 24개로 한 클래스 당 40개 정도의 데이터 뿐임.
- 3) 클래스당 데이터 수의 편향이 심함
 - 1:Lying이 압도적으로 많고 24:Rope Jumping의 데이터수는 부족함
- 4) 결측값(-1)이 분석을 방해함.

9896	163.33922	163.10895	163.18439	163.14746	163.06887	163.41757	163.03048	162.99198	1
8337	157.17939	157.41929	157.46137	157.08115	157.00873	156.27849	155.73814	155.392224	12
-1	-1	-1	-1	-1	-1	-1	-1	-1	12
2841	153.902207	153.26706	154.656888	153.30457	154.851395	157.54607	159.94405	159.7978	5
-1	-1	-1	-1	-1	-1	-1	-1	-1	4
-1	-1	-1	-1	-1	-1	-1	-1	-1	2
3589	163.17184	163.1348	163.33467	163.64389	164.025	163.86796	163.59718	163.56158	17
2904	157.19366	157.386	157.30917	157.19653	157.11945	157.272	157.27525	157.24245	19
5985	164.0985	164.17959	164.0662	164.18115	164.14064	164.2532	164.10019	164.29366	10
5626	159.49988	159.76315	159.67938	159.40468	159.40458	159.10108	158.75986	158.95617	7
2873	161.96122	162.03526	162.03669	162.27028	162.54484	162.5047	162.5414	162.42546	17
0781	163.24054	163.23834	163.1617	163.27867	163.12238	163.16234	163.39242	163.35423	10
7614	163.8147	163.62248	163.77626	163.92851	163.66163	163.85352	163.69654	163.6603	2
9848	157.39796	157.59252	157.55413	157.51506	157.66827	157.47801	157.36084	157.55344	10
1033	155.523425	155.83243	156.21774	156.48502	156.75261	157.0206	156.75277	156.71453	12
6177	157.70475	157.81766	157.97645	157.74731	157.54018	157.23356	156.53308	156.32807	7
-1	-1	-1	-1	-1	-1	-1	-1	-1	13
-1	-1	-1	-1	-1	-1	-1	-1	-1	18
8125	155.98542	155.73733	155.453789	155.088874	154.992353	154.856067	155.149339	156.35439	7
6077	162.85835	161.45026	161.85045	162.8572	163.69463	163.83954	163.79627	162.68278	6
8256	163.15785	162.69744	162.45819	162.07275	161.72733	162.11491	161.73002	161.76514	10
-1	-1	-1	-1	-1	-1	-1	-1	-1	17
8526	154.29519	154.123568	154.373406	154.915364	155.032264	155.231936	155.118862	155.440257	16
5415	158.8703	159.10992	158.35177	157.54149	157.23603	157.09598	156.44518	155.71659	7
7441	158.73929	158.16051	158.00581	157.75778	157.77265	157.7757	157.74577	157.75514	13
7264	155.85944	155.7423	155.81943	155.78238	155.89765	155.85927	155.97264	155.85806	9
7735	166.3138	165.6289	165.1777	164.7257	164.53772	164.19075	164.34336	164.6901	2
-1	-1	-1	-1	-1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1	7

5) 하드웨어적 문제

- 지나치게 많은 파라미터로 인해 RAM / GPU 용량 문제로 인한 에러들이 발생

6) 공부가 부족해서 발생한 문제

- 적절하지 못한 최적화 방법의 사용 / 정규화의 부재 등

7. 실패원인 개선해보기

1) 데이터를 나눈 기준(30초)이 유의미 하지 않을 수 있음

2) 데이터의 수가 충분하지 않음.

=> 데이터 나누는 기준을 30초가 아닌 15초로 변경하여, 1500열로 끝는다.

=> 데이터의 수가 대략 2배가 된다.

3) 결측값(-1)이 분석을 방해함.

=> 행마다 “결측값의 개수”를 계산하여 -1(결측값)이 지나치게 많은 행을 제거

```
8 xyz_data['Count'] = (xyz_data.iloc[:,1:] == -1).sum(axis=1)
9 drop_nan_xyz = xyz_data.loc[xyz_data['Count']>=200,:].index
10
11
1 xyz_data_drop = xyz_data.drop(drop_nan_xyz)
```

결측값의 수가 4500열중 200열 이상이면
해당 행을 제거

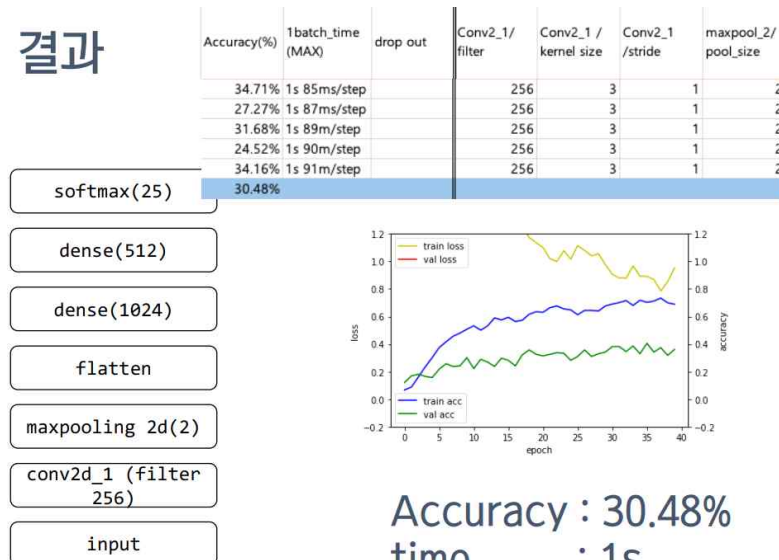
행 당 결측값의 갯수

4492	4493	4494	4495	4496	4497	4498	4499	act_id	act_name	sub_no	Count
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	9.0	watching TV	subject101	1065
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	11.0	car driving	subject101	2943
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	18.0	folding laundry	subject101	4158
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	19.0	house cleaning	subject101	4233
-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0	lying	subject101	3939

결측값 제거

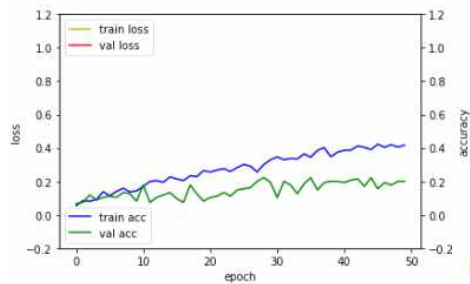
4) 개선 결과

결과



8. 결과 비교

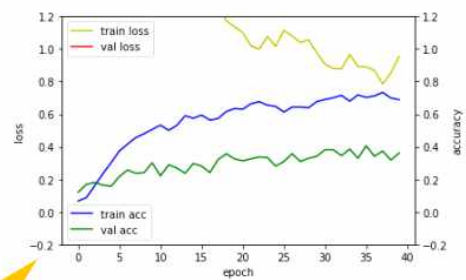
30초 단위 (960개)
결측값 매우 많음



Accuracy : 24.42%

time : 2s

15초 단위 (1722개)
결측값 대다수 제거



Accuracy : 30.48%

time : 1s

6.06%
성장했다!!

개선 전 후 정확도 비교

9. Reference

데이터셋

- archive.ics.uci.edu/ml/datasets/PAMAP2+Physical+Activity+Monitoring

그래프그리기

- data-make.tistory.com/137

학습데이터 나누기

- iagreebut.tistory.com/9?category=794421

CNN 학습

- tykimos.github.io/2017/01/27/CNN_Layer_Talk/

StratifiedShuffleSplit참고

- rfriend.tistory.com/520

데이터의 마지막 전처리 및 기본구조 잡기

- www.kaggle.com/alexanderlazarev/simple-keras-1d-cnn-features-split/notebook
- wdprogrammer.tistory.com/33

에폭수(Epoch)

- 3months.tistory.com/424

batch size

- goodtogreate.tistory.com/entry/Batch-크기의-결정-방법
- blog.lunit.io/2018/08/03/batch-size-in-deep-learning/

loss함수

- wikidocs.net/32105