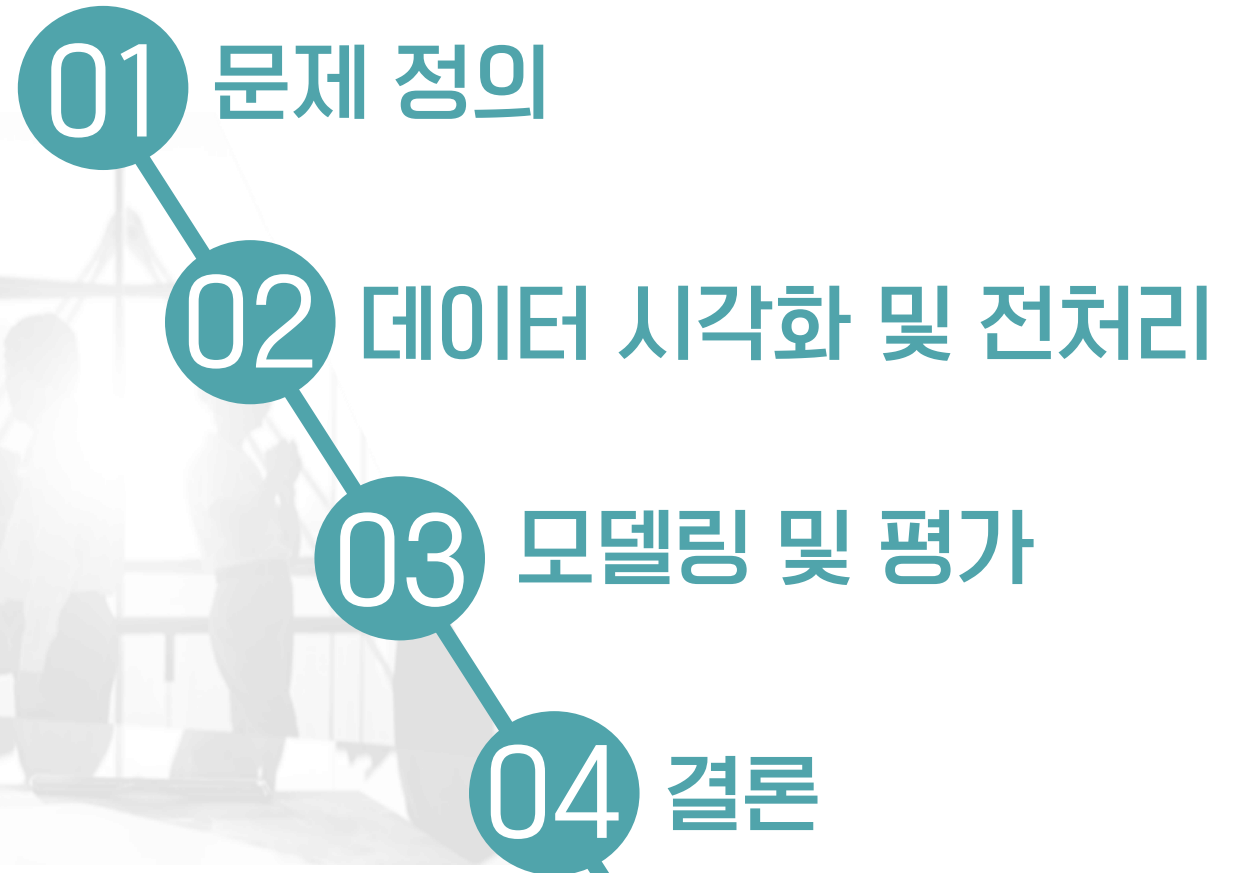


# Final Project Presentation

Team 3

이성준 강유성 양지연

# CONTENTS





# 01

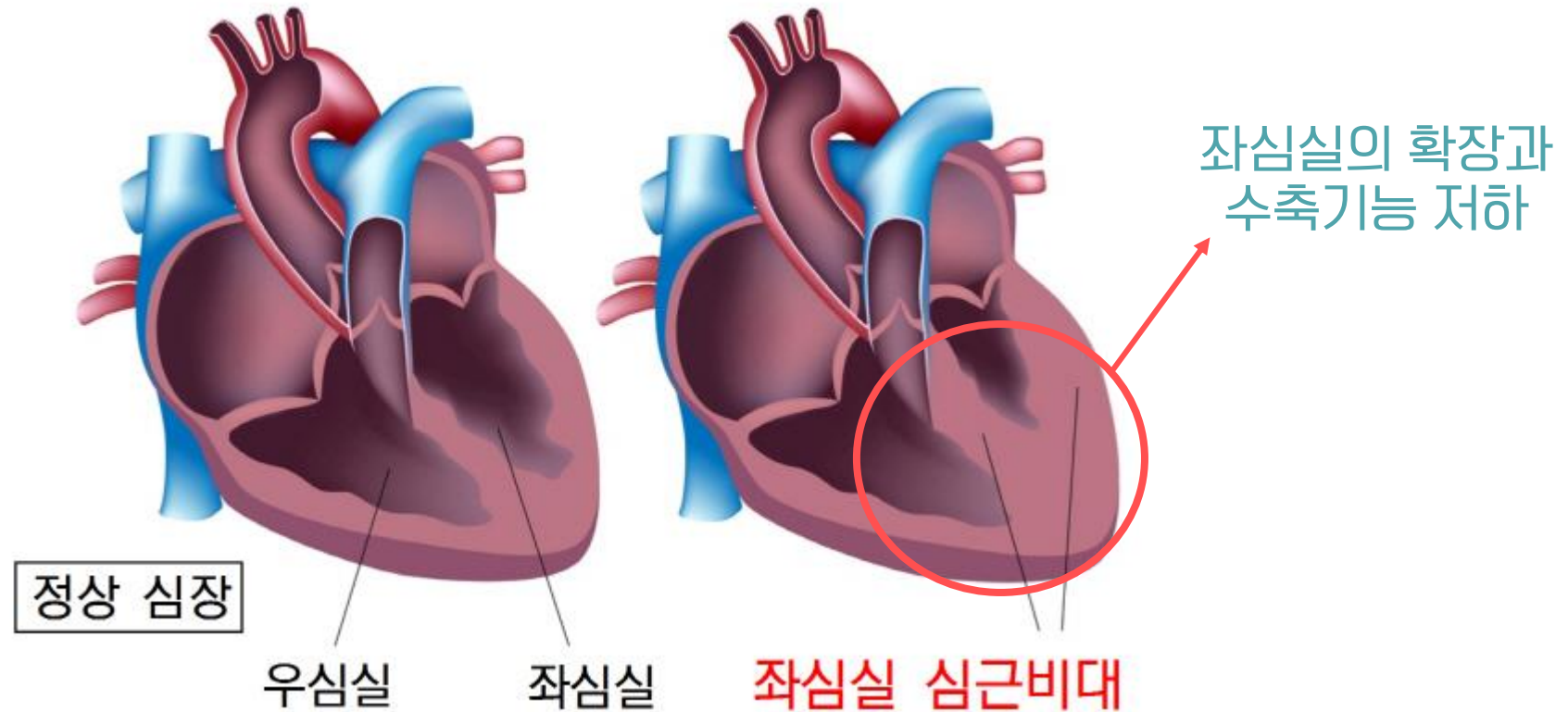
## 문제 정의

좌심실 비대증  
증상 발견의 중요성  
데이터 분포에서의 이점

# 문제 정의 좌심실 비대증

## 좌심실 비대증이란?

심혈관질환 발생 예측의 중요한 전조질환으로, 심전도 검사의 주된 목적 중 하나



# 문제 정의 좌심실 비대증 : 증상 발견의 중요성

## 좌심실 비대와 심전도

### 좌심실비대의 경우



신윤정 외 6명, 「좌심실 비대의 진단방법으로서 심전도」, 『가정의학회지』(2005)

### 더 정확한 '심장 초음파검사'

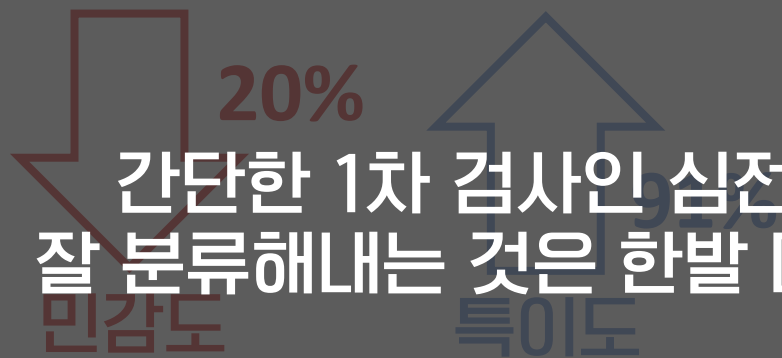


**BUT** 높은 검사비용  
일차 의료에서의 어려움

# 문제 정의 좌심실 비대증 : 증상 발견의 중요성

## 좌심실 비대와 심전도

좌심실비대의 경우



간단한 1차 검사인 심전도검사 데이터에서 좌심실 비대를 잘 분류해내는 것은 한발 더 빠른 의학적 조치를 가능하게 한다.

신윤정 외 6명, 「좌심실 비대의 진단방법으로서 심전도」, 『가정의학회지(2005)』

더 정확한 '심장 초음파검사'



**BUT** 높은 검사비용  
일차 의료에서의 어려움

# 문제 정의 좌심실 비대증 : 데이터 분포에서의 이점

## 데이터 구조(Y label)

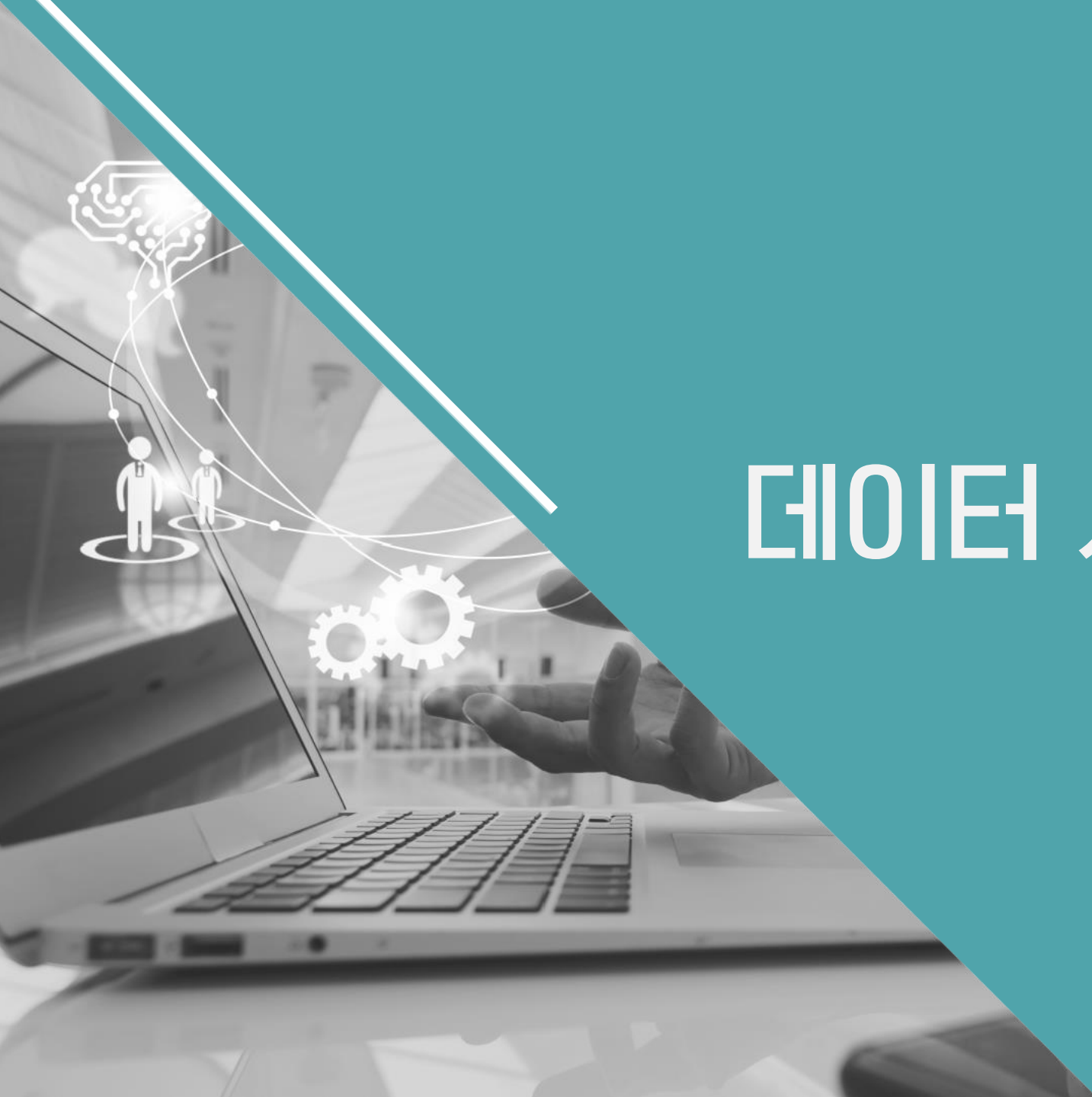
1. Dignosis 리스트 중, 좌심실 비대증 'Left ventricular hypertrophy'를 포함하는 record를 0과 1로 나누어 기록
2. 108:92의 비율로, imbalanced된 데이터가 아니기 때문에 training 시, 하나의 class로 편향된 분류모델이 될 위험성이 낮음

```
record_Y_index=[]
record_N_index=[]
Y_list=[]

for i in range(0,len(records)):
    lists_s=records[i]['dignosis']
    if 'Left ventricular hypertrophy.' in lists_s:
        record_Y_index.append(i)
        Y_list.append(1)
    else:
        record_N_index.append(i)
        Y_list.append(0)
print('record_Y_index의 길이는 ',str(len(record_Y_index)))
print('record_N_index의 길이는 ',str(len(record_N_index)))

record_Y_index의 길이는 108
record_N_index의 길이는 92
```





# 02

## 데이터 시각화 및 전처리

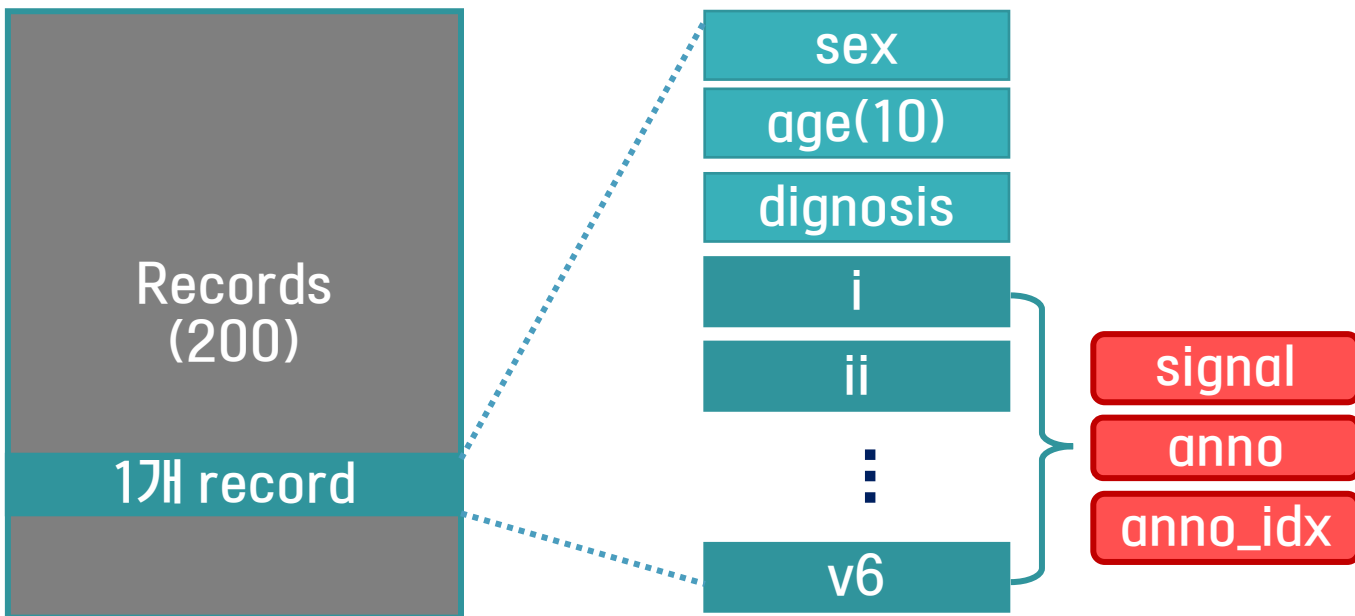
데이터 구조에 대한 설명  
Signal 데이터 Normalization  
Y label에 따른 데이터 시각화  
Anno 데이터 전처리



# 데이터 시각화 및 전처리

## 데이터 구조에 대한 설명

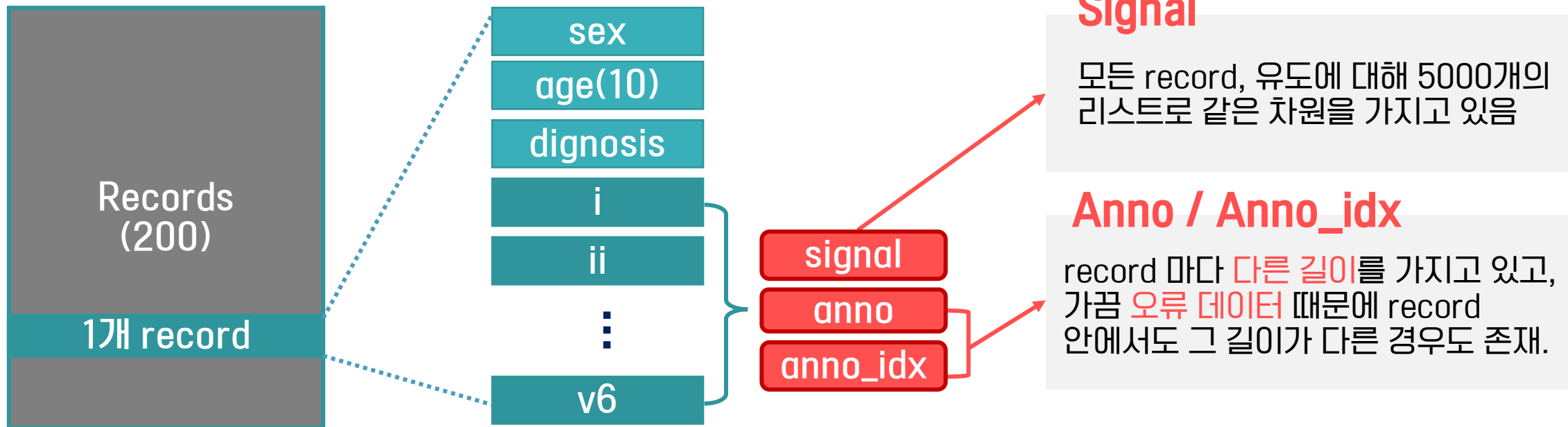
1. 200개의 records로 이루어짐
2. 1개의 record 안에
  - 기본 인적 사항 관련 데이터인 **sex**, **age(10)**은 단순 obs1개
  - **dignosis**는 record 별로 다른 길이의 리스트 형태
  - **i**, **ii**, **iii**, **avr**, **avl**, **avf**, **v1**, **v2**, **v3**, **v4**, **v5**, **v6**은 **signal**, **anno**, **anno\_idx**의 dictionary 형태로 이루어짐



# 데이터 시각화 및 전처리

## 데이터 구조에 대한 설명

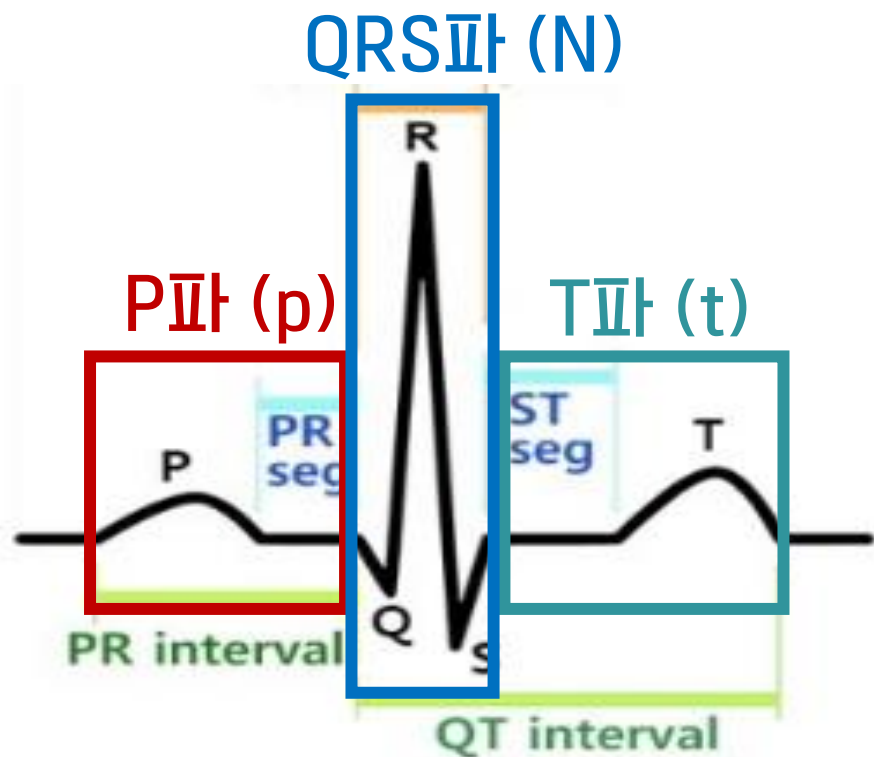
1. 200개의 records로 이루어짐
2. 1개의 record 안에
  - 기본 인적 사항 관련 데이터인 **sex**, **age(10)**은 단순 obs1개
  - **dignosis**는 record 별로 다른 길이의 리스트 형태
  - **i**, **ii**, **iii**, **avr**, **avl**, **avf**, **v1**, **v2**, **v3**, **v4**, **v5**, **v6**은 **signal**, **anno**, **anno\_idx**의 dictionary 형태로 이루어짐



# 데이터 시각화 및 전처리

## 데이터 구조에 대한 설명

- ✓ Anno, anno\_idx는 signal 리스트 데이터 중 N, t, p 의 위치에 해당되는 인덱스에 대한 정보를 가지고 있는 데이터
- ✓ 5000의 차원을 가지고 있는 signal 데이터에서 핵심적인 정보를 뽑아낼 수 있도록 하는 잠재적인 변수라고 판단



	anno	anno_idx_i	anno_idx_ii	anno_idx_iii	anno_idx_avr	anno_idx_avl
0	(	641	644	633	642	646
(N)	N	664	662	666	650	665
2	)	690	682	682	689	681
3	(	773	776	782	775	783
(t)	t	840	843	834	841	838
5	)	887	878	867	876	880
6	(	1252	1250	1271	1251	1257
(p)	p	1282	1278	1288	1280	1285
8	)	1301	1302	1309	1304	1302
9	(	1324	1324	1323	1319	1324
(N)	N	1344	1342	1335	1329	1345
11	)	1374	1374	1362	1362	1359
12	(	1457	1458	1460	1456	1455
(t)	t	1519	1524	1517	1521	1518
14	)	1567	1572	1550	1567	1564

### 딥러닝 모델 학습 전, 데이터 Normalization의 필요성

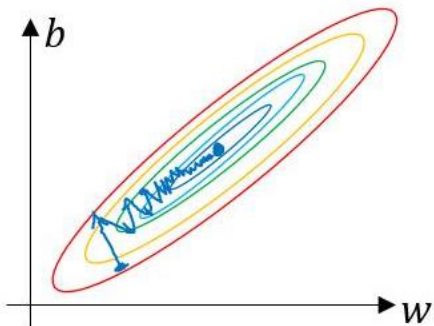
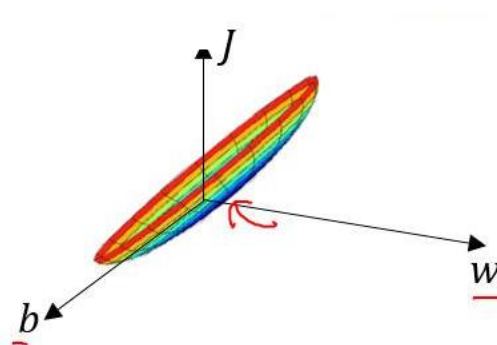
모델의 효율적인 학습

- 데이터의 범위 차이가 심하면 수치계산 비효율적
- 학습을 더 빨리할 수 있는 효과
- Local minimize / maximize 상태에 빠지게 될 가능성을 줄이는 효과

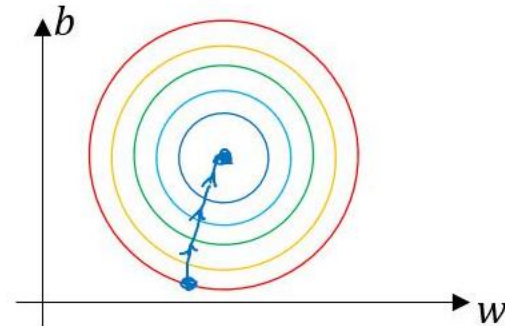
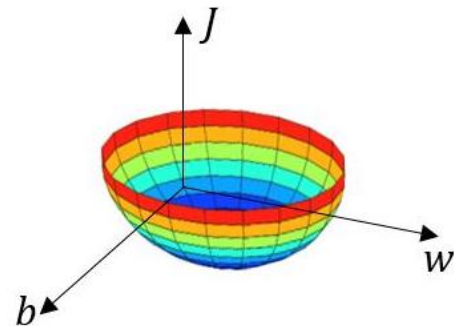


Signal 값들에 대한 적절한 Normalize 전처리가 필요

Unnormalized

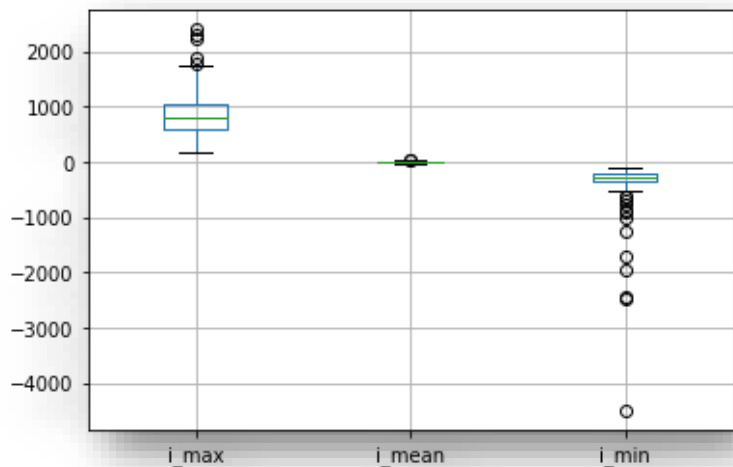


Normalized

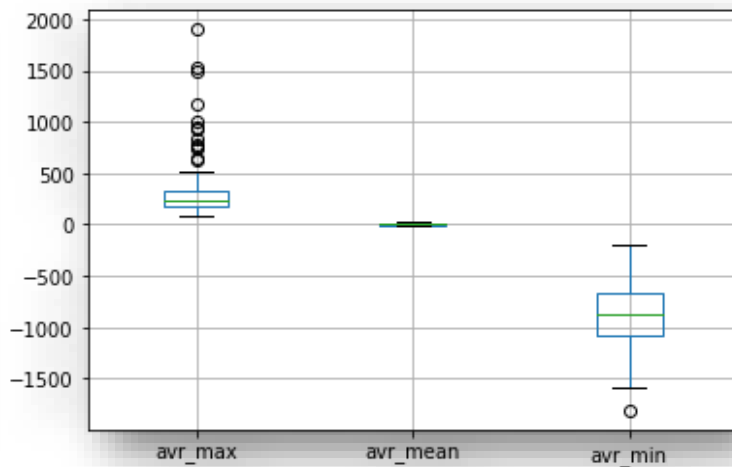




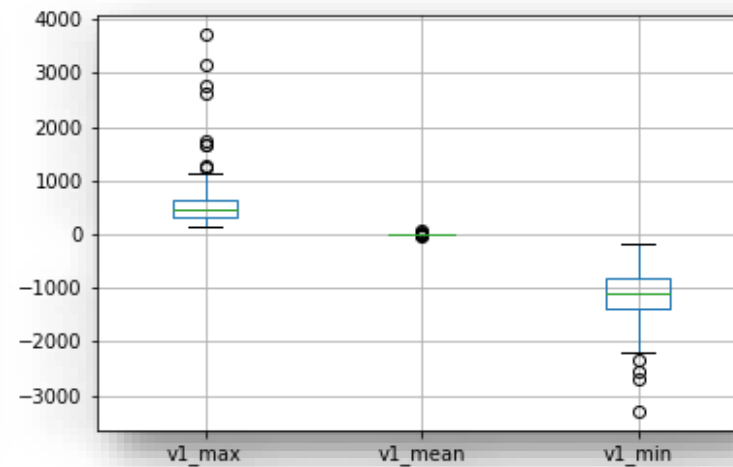
Signal 데이터는 record마다, 유도마다 scale이 모두 천차만별



유도 i의 record 별 min mean max



유도 avr의 record 별 min mean max

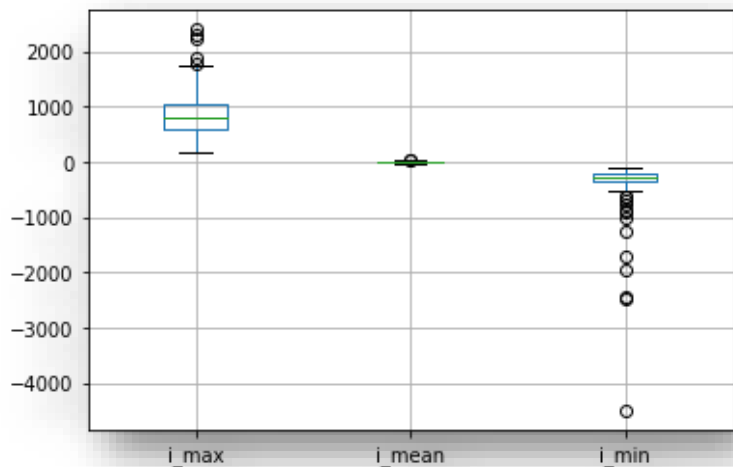


유도 v1의 record 별 min mean max

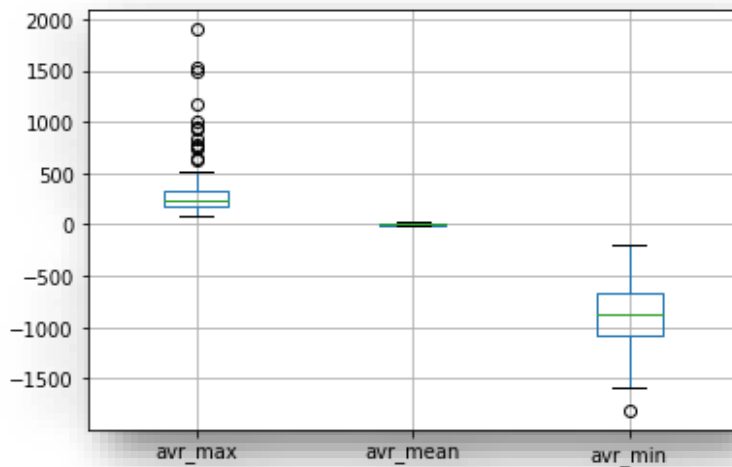
- mean은 거의 0에 가까운 값으로, record, 유도마다 큰 차이 X
- min, max 값들은 같은 유도 안에서도 record들끼리 큰 차이가 나는 scale을 가지고 있음



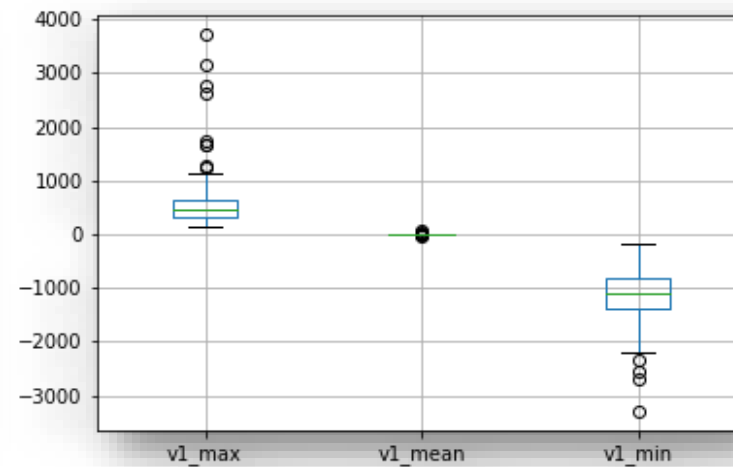
Signal 데이터는 record마다, 유도마다 scale이 모두 천차만별



유도 i의 record 별 min mean max



유도 avr의 record 별 min mean max



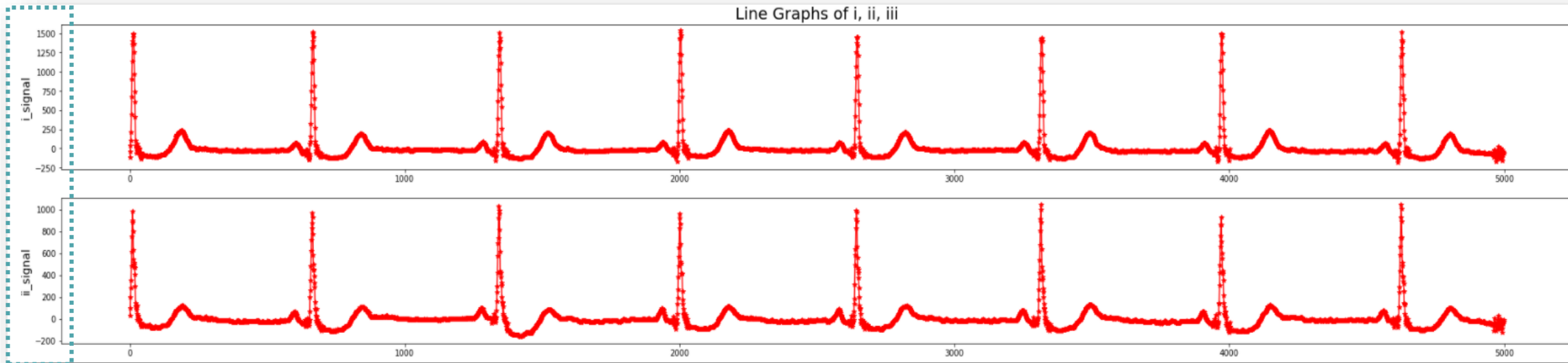
유도 v1의 record 별 min mean max

- 만약 전체 signal 값의 min max를 활용하여 각 record들의 signal을 normalize 할 경우, 각 record 유도 안에서의 변동폭을 제대로 고려하지 못할 것이라고 생각
  - min, max 값들은 같은 유도 안에서도 record를 끼리 큰 차이가 나는 scale을 가지고 있음
- 각 record의 유도 signal 하나의 list마다 min-max scaling을 통해 진행하기로 결정

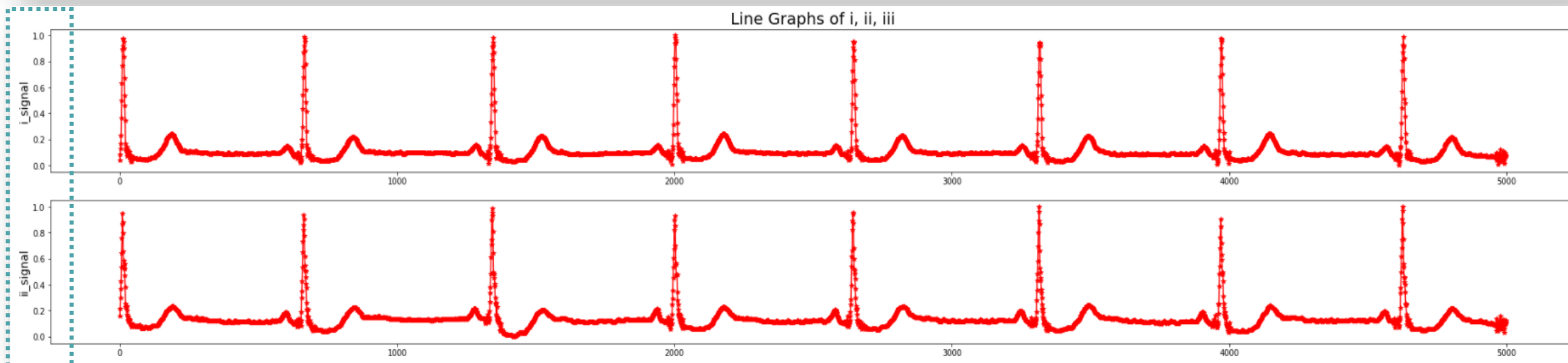
# 데이터 시각화 및 전처리

## Signal 데이터 Normalization

Unnormalized



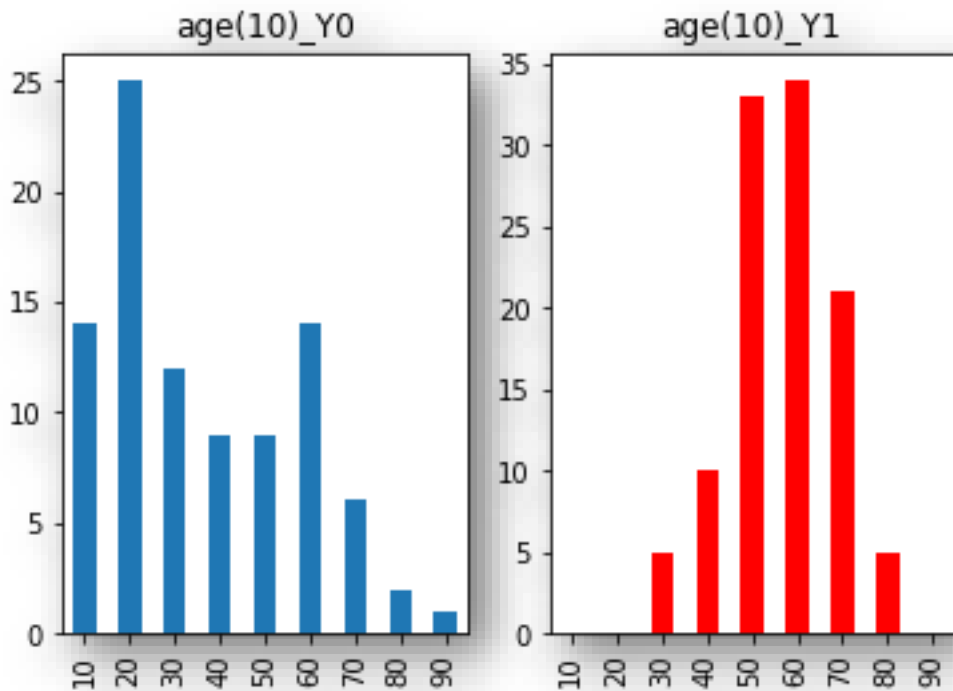
Normalized



### Age(10) / Sex 변수

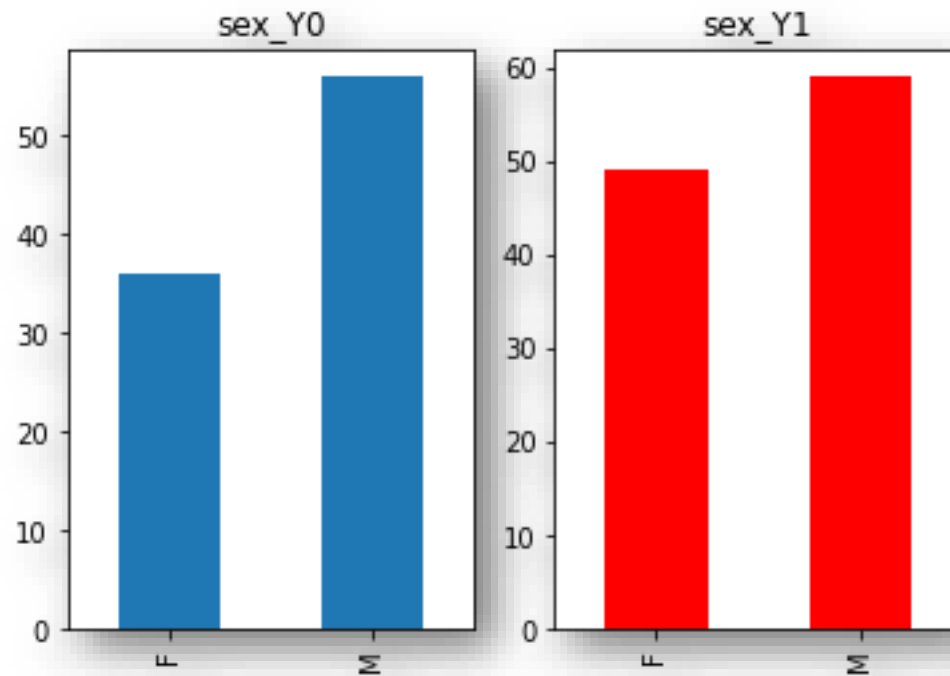
#### Age(10)

증상이 발견된 환자들의 50,60,70대 비율이 확연히 높음.  
증상 유무를 분류하는데 중요한 정보를 가지고 있음.



#### Sex

증상이 발견된 환자들의 여성 비율이 증상이 발견되지 않은 환자들의 여성 비율보다 높은 편.  
뚜렷한 정보를 포함하고 있다고 판단하기 어려움.

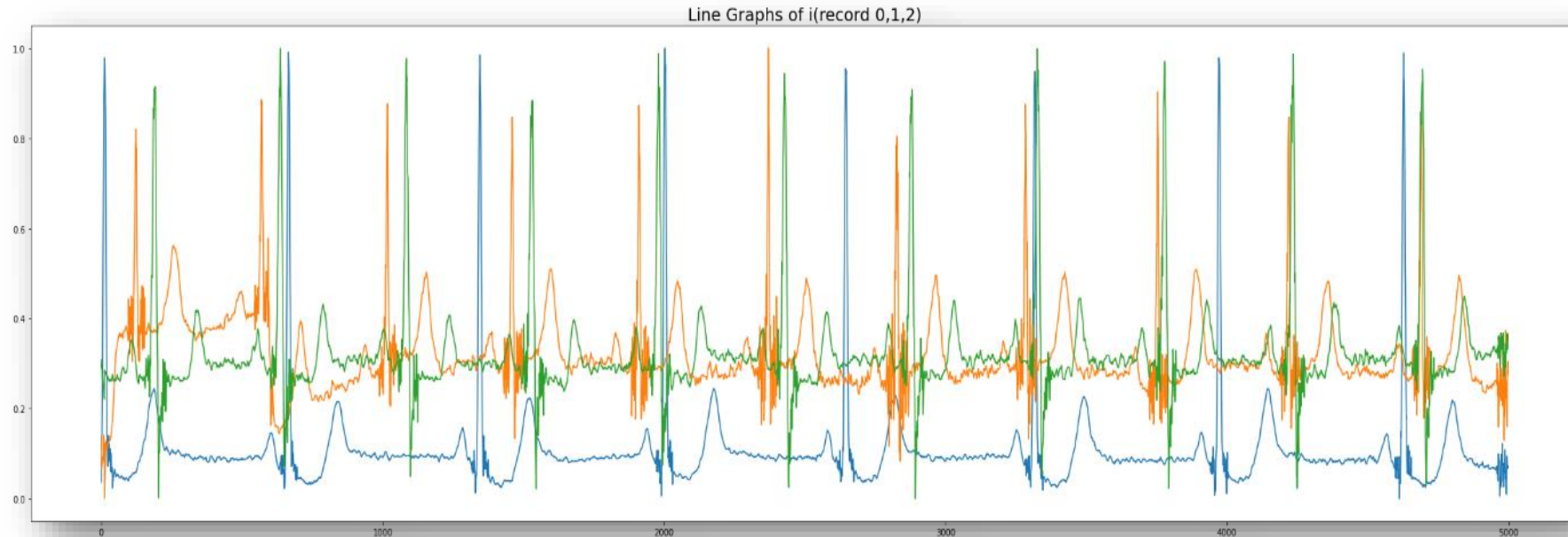




# 데이터 시각화 및 전처리

## Y label에 따른 데이터 시각화

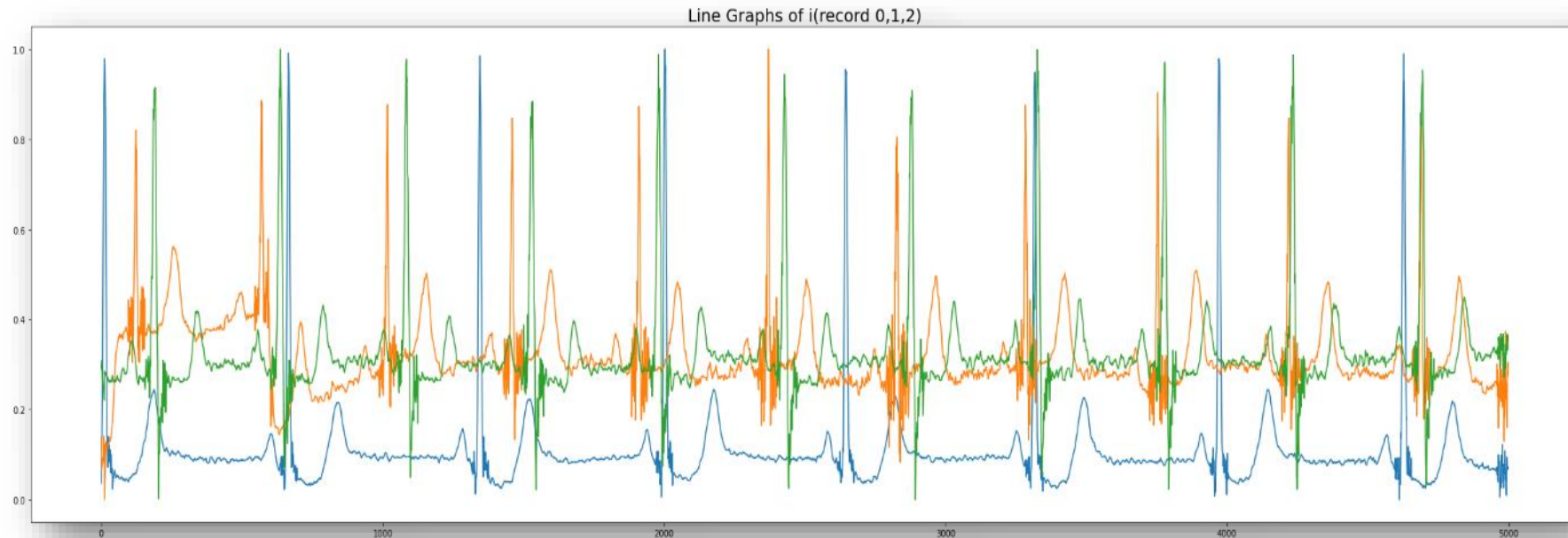
### Signal 변수



i 유도의 record 0,1,2에 대한 signal Plot

- 각 record별로 signal 데이터 파동의 폭과, 시작 지점이 모두 다름
- Signal의 평균적인 값들로는 Y label 별 signal 데이터들의 특성을 보기 어려움
- list 별 평균 값들은 이러한 파동의 특성을 대변할 수 없기 때문

### Signal 변수



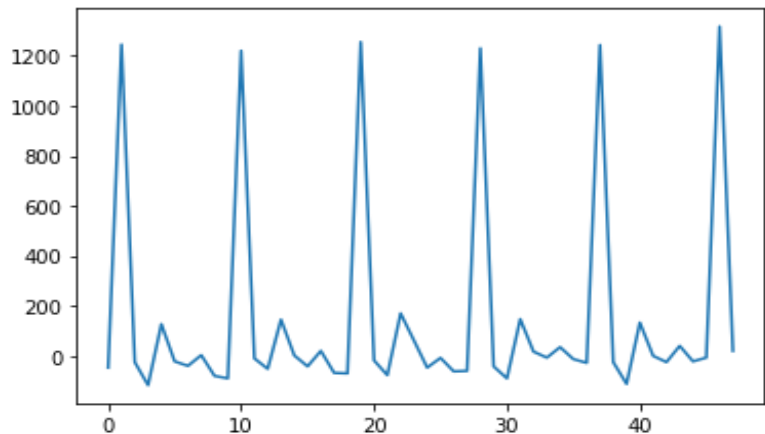
i 유도의 record 0,1,2에 대한 signal Plot

- 각 record별로 signal 데이터 파동의 시작 지점이 모두 다름
- Signal의 평균적인 값들로는 Y label 할 signal 데이터들의 특성을 보기 어려움
- **anno 변수**를 활용, signal의 핵심적인 정보들을 뽑아 내기 위한 전처리 진행

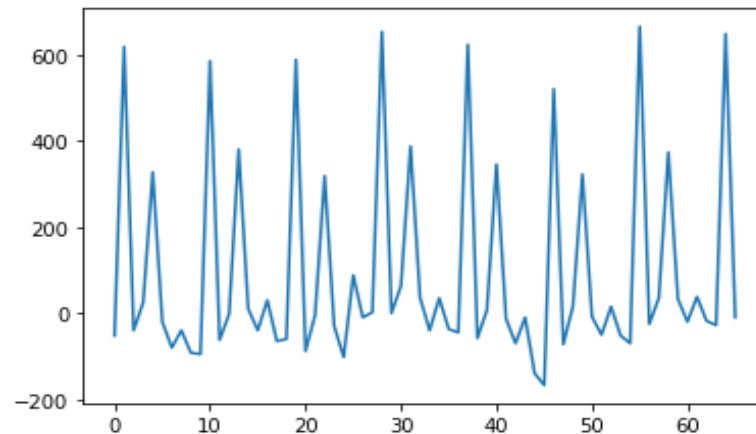
# 데이터 시각화 및 전처리

## Anno 데이터 전처리

앞서 말했 듯, anno, anno\_idx는 signal에서의 데이터들 중 N/t/p 의 위치를 알려주는 데이터.  
문제는 anno, anno\_idx의 길이가 record별로, 유도별로도 차이가 있을 정도로 **형태가 일정 X**



0번째 record의 v6 signal(anno\_idx)



198번째 record의 v2 signal(anno\_idx)

손쉬운 분석을 위해서, 데이터 형태를 일정하게 맞춰줄 수 있는 전처리 방법이 필요

### Signal\_ntp 데이터 생성

대부분의 anno는 (N)(t)(p)의 순서로 반복되는 양상을 띠

→ 반복되는 (N)(t)(p) (9개 리스트) index에 해당되는 **signal 값을 평균한 결과물**을 한 record의 signal 추세를 대변하는 값으로 활용하기로 결정(Signal\_ntp)

```
'v5': {'anno': '(N)(t)(p)(N)(t)(p)(N)(t)(p)(N)(t)(p)(N)(t)(p)(N)(t)(p)(N)(t)(p)(N)',  
      'anno_idx': array([ 506,  530,  564,  617,  673,  712,  927,  950,  972,  993, 1014,  
                        1039, 1105, 1158, 1197, 1401, 1428, 1455, 1472, 1493, 1519, 1583,  
                        1636, 1677, 1871, 1895, 1923, 1939, 1960, 1986, 2047, 2103, 2142,  
                        2347, 2367, 2394, 2411, 2432, 2459, 2523, 2576, 2616, 2819, 2842,  
                        2863, 2886, 2907, 2933, 2998, 3050, 3090, 3293, 3317, 3340, 3360,  
                        3382, 3417, 3470, 3523, 3565, 3768, 3794, 3819, 3837, 3859, 3892,  
                        3948, 4002, 4043, 4251, 4276, 4302, 4320, 4341, 4371]),
```



**Signal\_ntp**  
(9개 차원으로 축소)

```
'signal_ntp': [0.22819643238624224,  
               0.8812715536973175,  
               0.20776810510574661,  
               0.24806779686880093,  
               0.45993873409222685,  
               0.2456530198065126,  
               0.23768959624011962,  
               0.27975936155278175,  
               0.2411458174382763],
```

### Signal\_ntp 데이터 생성

**BUT** (N)(t)(p)가 반복되는 형태로 저장되어 있지 **않은** 데이터들 존재

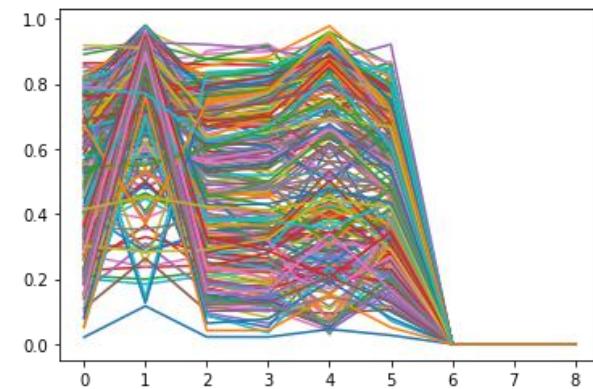
**\*(N)(t)의 형태로만 이루어진 데이터가 있는 경우**

→ 21개의 record

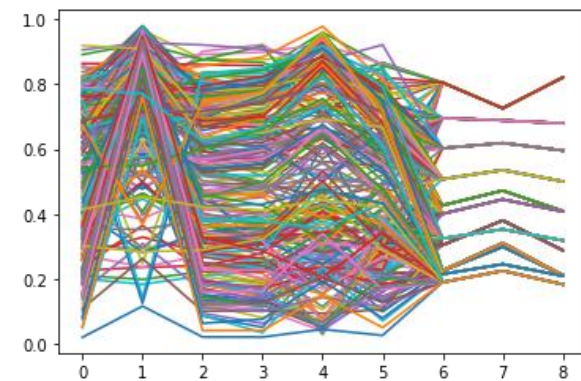
→ 삭제 처리하기에는 record의 개수(200개)가 너무 적음

→ **두 가지 방법**을 활용하여 (p) **부분의 값 처리**를 진행

(21개의 record에 대해서 y label에 대한 분포 확인, 거의 1대 1의 비율)



없는 마지막 3개의 값을 0으로 대체

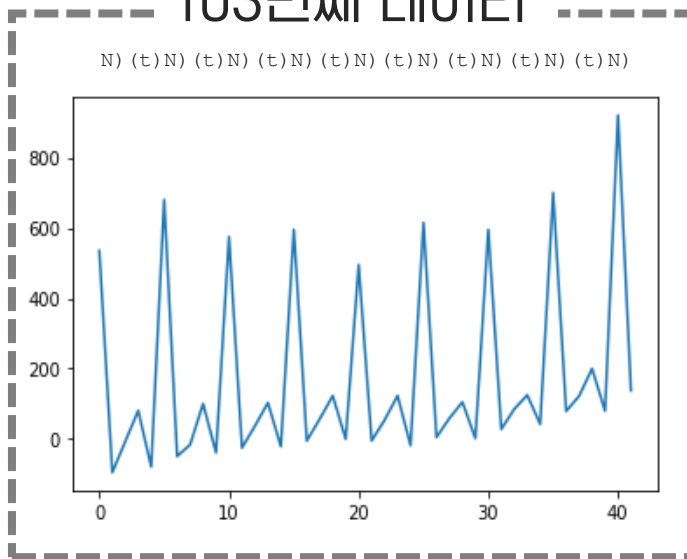


마지막 3개의 값을 각 평균값들로 대체

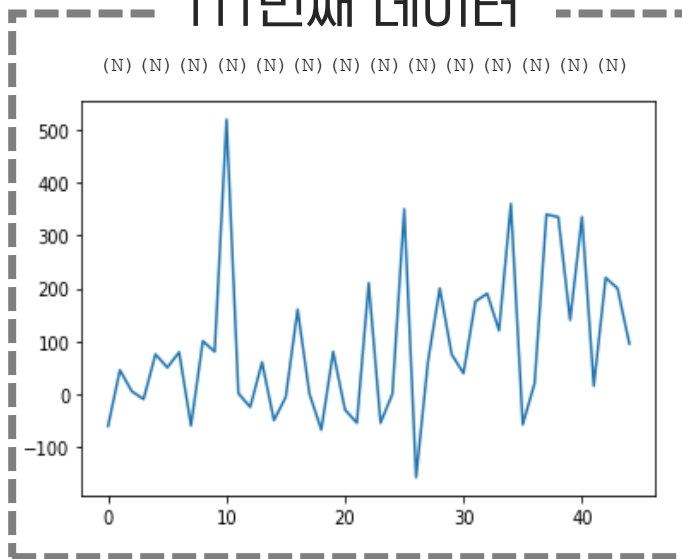
### Signal\_ntp 데이터 생성

하지만 이 외에도 3개 record가 기형적인 데이터를 가지고 있었음.

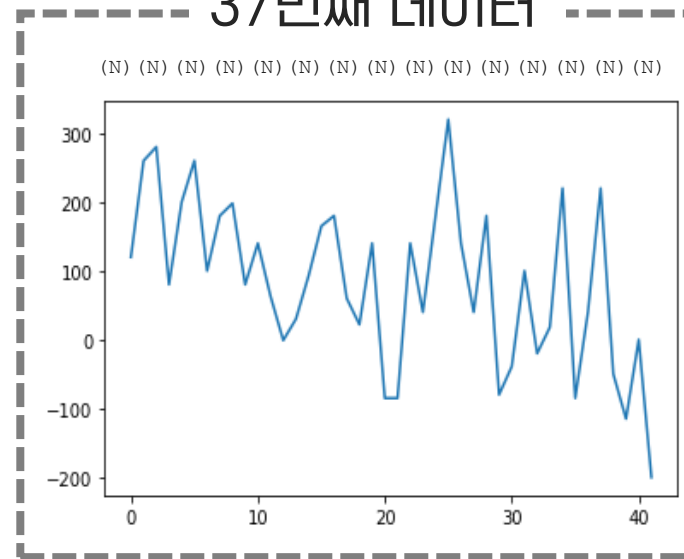
103번째 데이터



111번째 데이터



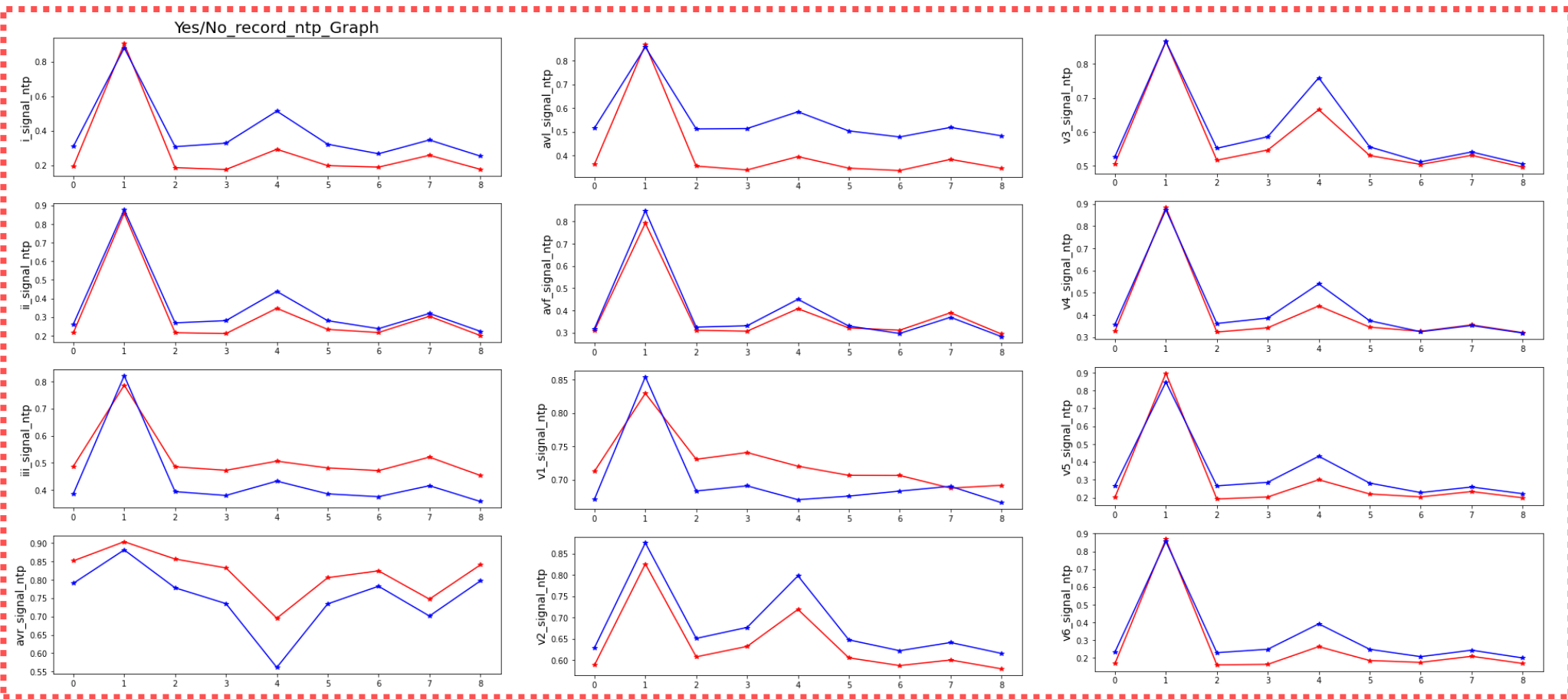
37번째 데이터



이 세 개 record의 경우 많은 부분을 평균적인, 혹은 0의 가짜 데이터로 대체해  
넣어야 하기 때문에 오히려 일관성을 해칠 수 있는 위험성이 클 것으로 판단

→ 삭제

### Y label에 따른 signal\_ntp의 평균적인 값 비교



빨간색이 좌심실 비대증 O 파란색이 좌심실 비대증 X





# 03

## 모델링 및 평가

모델링을 위해 고려한 Framework  
활용한 Python Package

DNN 모델링

1D-CNN 모델링



# 모델링 및 평가

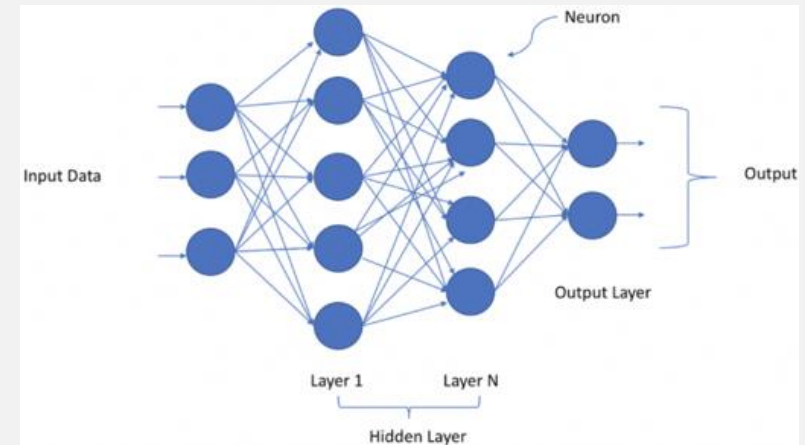
## 모델링을 위해 고려한 Framework

### 1. DNN Framework

(Signal을 Anno를 이용해 9개 list element 값으로 축소시킨 데이터)

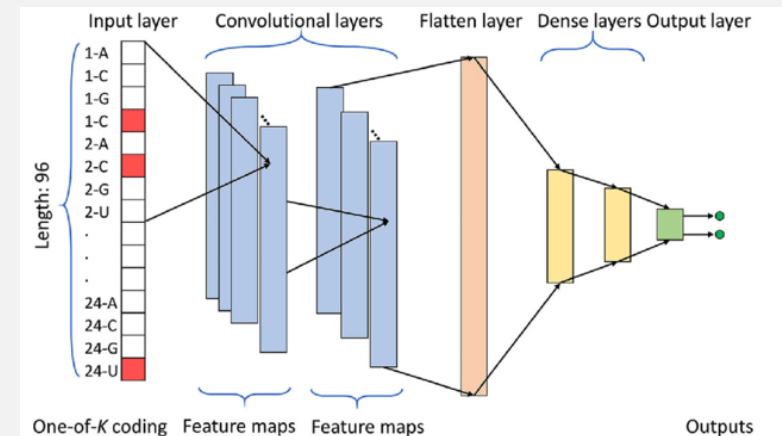
**signal\_ntp** / age(10) / sex

- 압축된 심전도 정보와 환자의 개별 정보 데이터를 하나의 input으로 활용하는 모델



### 2. 1D-CNN Framework

- signal(5000개 list element) 데이터의 **주기적인 파동을 지니고 있는 신호의 특성**을 고려한 모델
- 좌심실 비대증의 진단에 결정적인 정보를 제공하는 유도 signal 활용



### Tensorflow Keras 패키지

#### Tensorflow

- 데이터 흐름 프로그래밍을 위한 오픈소스 소프트웨어
- 머신러닝, 딥러닝을 구현하기 위해 다양한 기능을 제공하는 라이브러리
- 계산 구조와 목표함수만 정의하면 자동으로 미분 계산을 처리

#### Keras

- tensorflow 등 딥러닝 라이브러리를 백엔드로 사용, 다양한 딥러닝 모델을 쉽게 구성할 수 있도록 보조
- 사용하기 쉬운 API
- 거의 모든 종류의 딥러닝 모델을 간편하게 만들고 훈련시킬 수 있는 파이썬 딥러닝 프레임워크



# 모델링 및 평가

## DNN 모델링

### 모델 Input 형태

sex, age 변수를 categorical 변수로 취급하여 one-hot encoding 한 변수들로 변환  
유도별 9개 리스트로 이루어진 signal\_ntp 변수들 추가 하여 총 117개의 차원으로 input 결정

	sex		age								signal_ntp									
	M	10	20	30	40	50	60	70	80	i_ntp_1	i_ntp_2	i_ntp_3	i_ntp_4	i_ntp_5	i_ntp_6	i_ntp_7	i_ntp_8	i_ntp_9	ii_ntp_1	ii_ntp_2
0	0	0	0	0	0	1	0	0	0	0.098254	0.973553	0.086721	0.050011	0.226814	0.107233	0.091968	0.151173	0.083108	0.147099	0.956050
1	1	0	0	0	0	0	1	0	0	0.365911	0.863760	0.383599	0.296796	0.483465	0.281585	0.287038	0.351857	0.275958	0.213630	0.884585
2	1	0	0	0	0	1	0	0	0	0.341046	0.950236	0.308588	0.266144	0.423603	0.304762	0.305259	0.378330	0.288779	0.082858	0.938927
3	1	0	0	0	0	1	0	0	0	0.146269	0.873148	0.175481	0.214295	0.555542	0.181467	0.157315	0.203040	0.142144	0.277911	0.576951
4	1	0	0	0	0	0	1	0	0	0.178577	0.925385	0.220679	0.203841	0.404770	0.189319	0.166674	0.236938	0.160573	0.298758	0.923165
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
192	0	0	0	0	0	0	1	0	0	0.226246	0.852691	0.184084	0.162891	0.392577	0.193409	0.196171	0.328625	0.195729	0.247047	0.867740
193	0	0	0	1	0	0	0	0	0	0.197055	0.937768	0.180894	0.202196	0.362009	0.204503	0.195901	0.262111	0.182887	0.357381	0.950938
194	0	0	0	1	0	0	0	0	0	0.105215	0.918727	0.119861	0.143330	0.380843	0.153565	0.124114	0.199401	0.106789	0.164639	0.944696
195	0	0	0	1	0	0	0	0	0	0.080617	0.932598	0.077626	0.080209	0.293871	0.102396	0.084329	0.183113	0.081338	0.146168	0.925265
196	0	0	0	0	0	1	0	0	0	0.320330	0.900107	0.290330	0.322111	0.500999	0.321667	0.314999	0.402554	0.315998	0.378607	0.925874

## DNN framework에 활용한 기본적인 모델 설명

```
model=Sequential()  
model.add(Dense(64,input_dim=117,activation='relu'))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(8,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))  
  
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])  
  
simple_mod=model.fit(x_train,y_train,epochs=120,validation_data=(x_val,y_val))
```

**Activation function**

**BatchNormalization**

**L2 Regularizer**

**Adam Optimizer**

### DNN framework에 활용한 기본적인 모델 설명

```
model=Sequential()  
model.add(Dense(64,input_dim=117,activation='relu'))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(8,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))  
  
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])  
  
simple_mod=model.fit(x_train,y_train,epochs=120,validation_data=(x_val,y_val))
```

Activation function

BatchNormalization

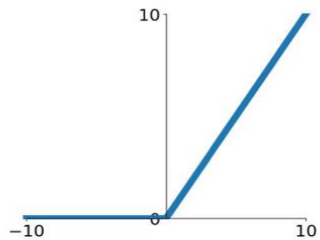
L2 Regularizer

Adam Optimizer

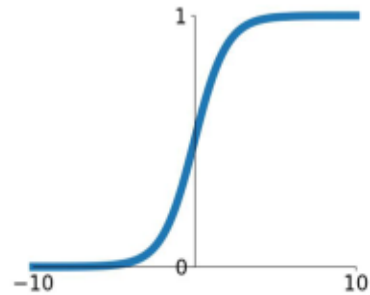
# 모델링 및 평가 DNN 모델링

## DNN framework에 활용한 기본적인 모델 설명

### Activation function

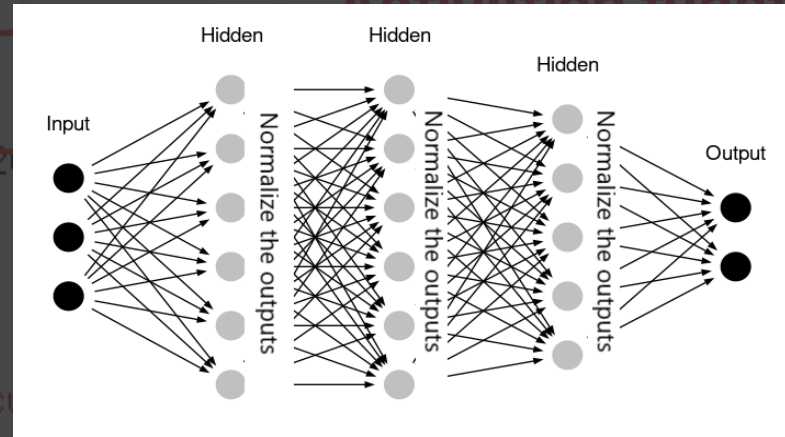


**ReLU**  
(Rectified Linear Unit)



**Sigmoid**

### BatchNormalization



hidden layer에서 활용 이진 분류를 위한 output layer에서 활용 특정 layer에서 학습된 weight의 imbalance를 해결

## DNN framework에 활용한 기본적인 모델 설명

```
model=Sequential()  
model.add(Dense(64,input_dim=117,activation='relu'))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(8,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))  
  
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])  
  
simple_mod=model.fit(x_train,y_train,epochs=120,validation_data=(x_val,y_val))
```

Activation function

BatchNormalization

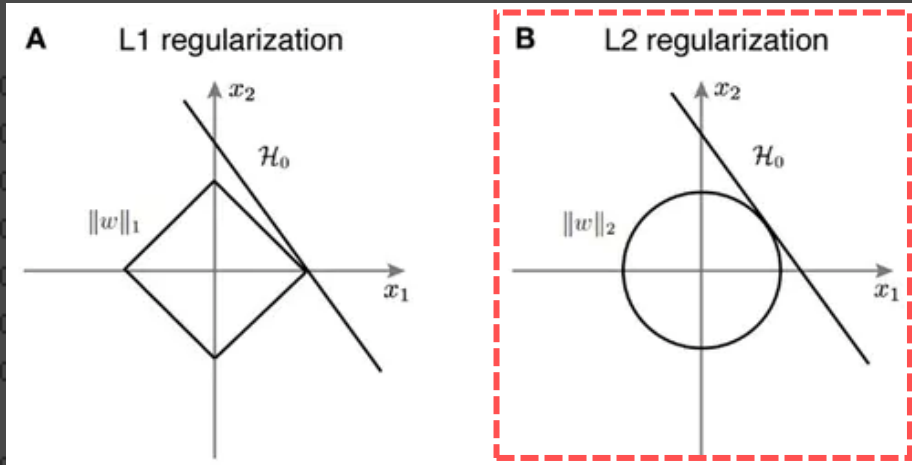
L2 Regularizer

Adam Optimizer

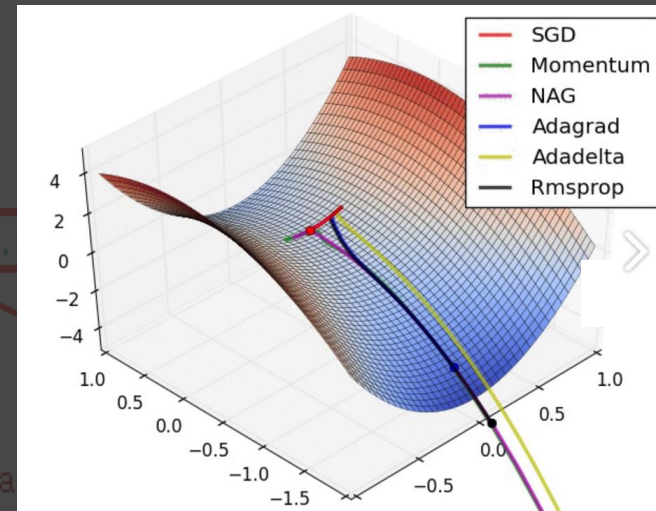
# 모델링 및 평가 DNN 모델링

## DNN framework에 활용한 기본적인 모델 설명

### L2 Regularizer



### Adam Optimizer



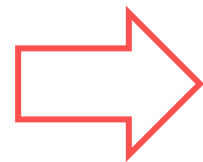
- layer에 최적화를 위한 패널티 부여
- loss 함수에 합해진 패널티는 overfitting을 예방
- 일반화 성능을 높이는데 보조하는 역할

- 모델의 학습과 그 결과에 따른 손실함수의 값을 최소화
- 모델의 학습속도를 빠르고 안정적이게 만드는 것
- momentum과 RMSProp 두 optimizer를 적절하게 조화



## 1. Train/validation model

```
model=Sequential()  
model.add(Dense(64,input_dim=117,activation='relu'))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(8,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))  
  
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])  
  
simple_mod=model.fit(x_train,y_train,epochs=120,validation_data=(x_val,y_val))
```

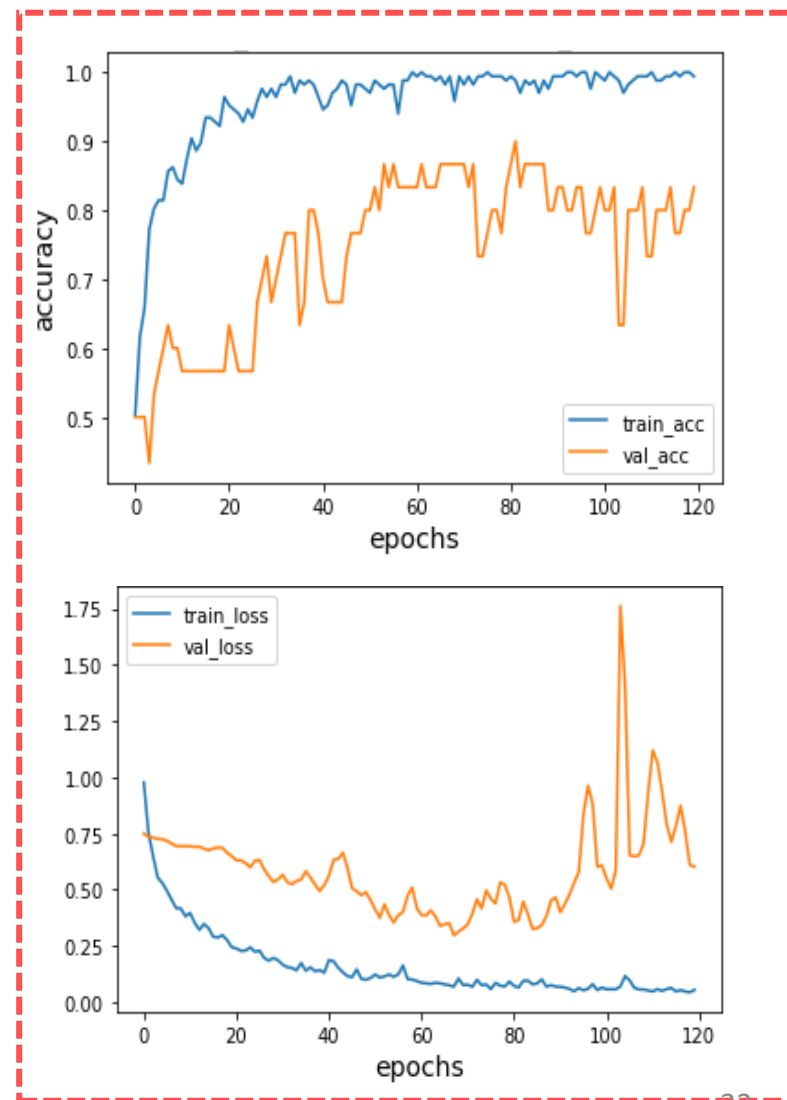


train accuracy: 0.994    validation accuracy: 0.833

그래프를 보면 train accuracy와 loss는 계속해서 개선되는데 비해,  
validation accuracy와 loss는 가면 갈수록 오히려 개선되지 않는 양상



train set에 모델이 overfitting 되고 있다는 증거로,  
해결하기 위한 다른 방법이 필요

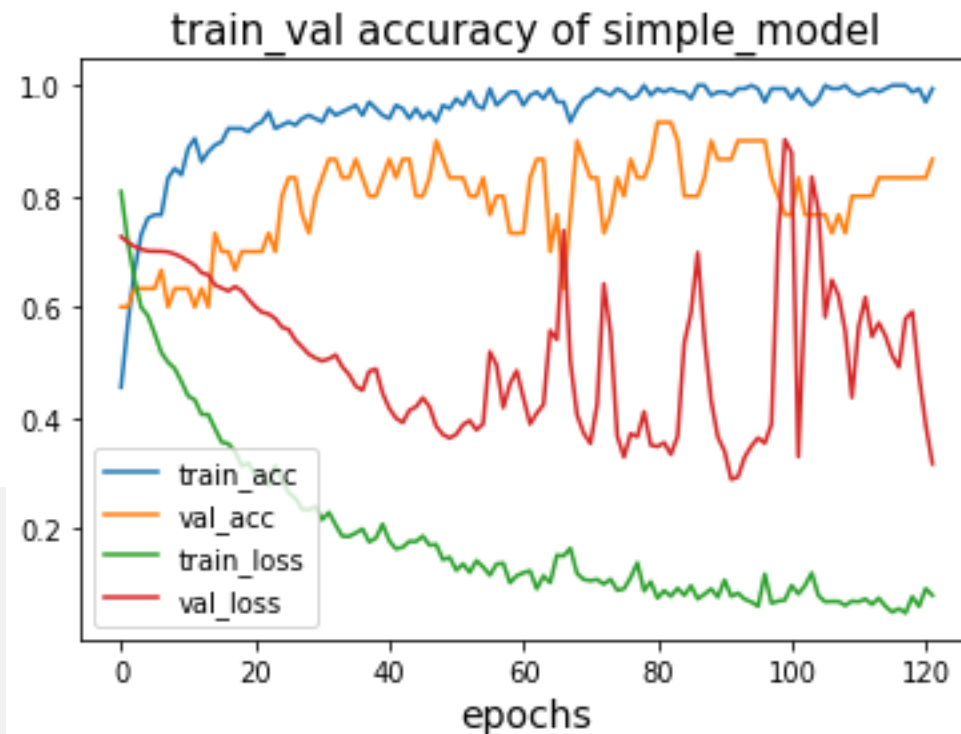


## 2. Train/validation EarlyStopping model

```
model=Sequential()  
model.add(Dense(64,input_dim=117,activation='relu'))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(8,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))  
  
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])  
  
early_stopping = EarlyStopping(patience = 30)  
simple_mod=model.fit(x_train,y_train,epochs=200,validation_data=(x_val,y_val),callbacks=[early_stopping])
```

### EarlyStopping이란?

- validation set의 성능이 더 이상 개선의 여지가 없을 때, 학습을 종료하는 callback 함수
- 너무 많은 epoch으로 인한 과적합을 방지하고, 모델 학습 시간을 효율적으로 절약하기 위함



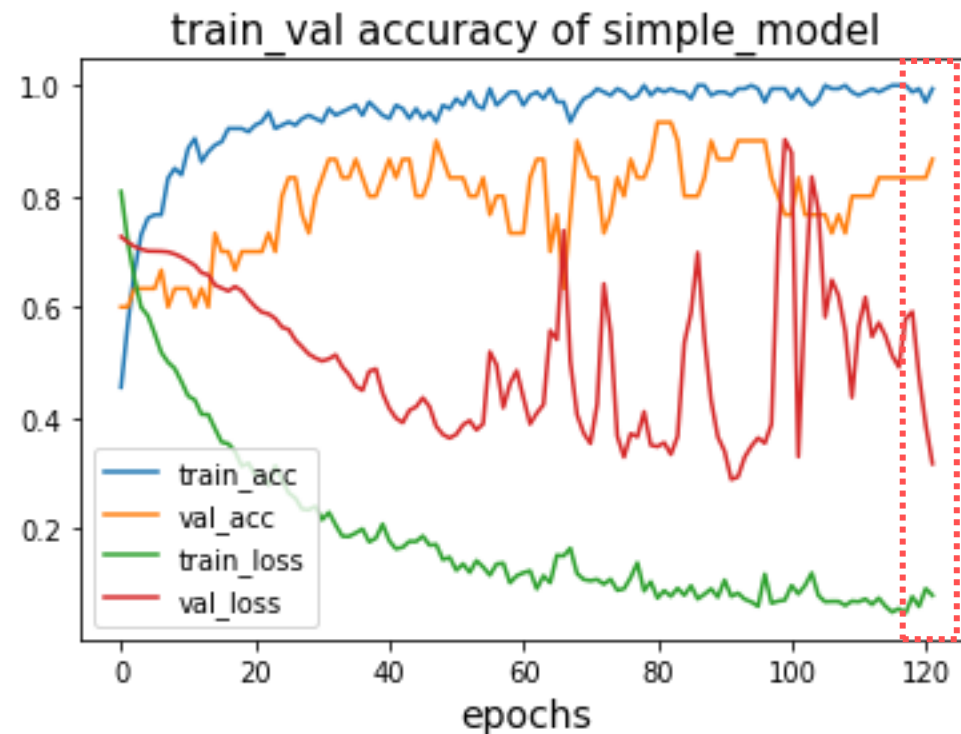
## 2. Train/validation EarlyStopping model

```
model=Sequential()  
model.add(Dense(64,input_dim=117,activation='relu'))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(8,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))  
  
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])  
  
early_stopping = EarlyStopping(patience = 30)  
simple_mod=model.fit(x_train,y_train,epochs=200,validation_data=(x_val,y_val),callbacks=[early_stopping])
```

train accuracy: 0.994   validation accuracy: 0.8667

→ 좀 더 개선된 validation accuracy

200개의 epoch을 돌 수 있도록 설정하였는데,  
early stopping 함수로 인해, 적정선인 122번째 epoch에서 멈춤



## 2. Train/validation EarlyStopping model

```
model=Sequential()  
model.add(Dense(64,input_dim=117,activation='relu'))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))  
model.add(BatchNormalization(axis=1))  
model.add(Dense(8,activation='relu'))  
model.add(Dense(1,activation='sigmoid'))
```

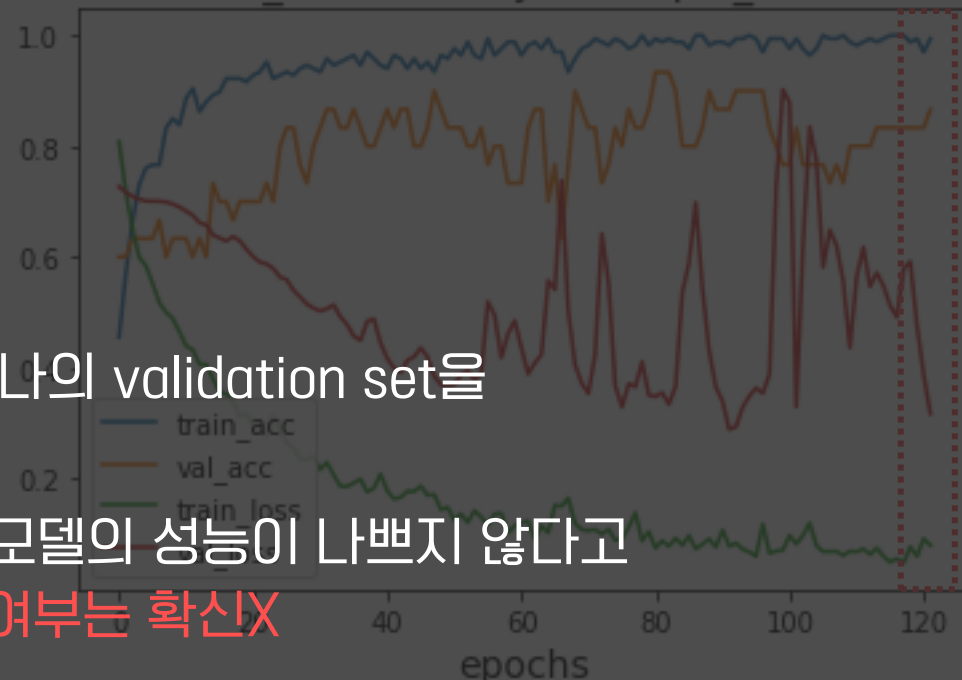
```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
early_stopping = EarlyStopping(patience = 30)
```

```
simple_model=model.fit(x_train,y_train,validation_data=(x_val,y_val),callbacks=[early_stopping],
```



train\_val accuracy of simple\_model



→ 하지만 우리는 현재 random 하게 골라진 단 하나의 validation set을 evaluation에 활용하며 모델을 training 시킴.

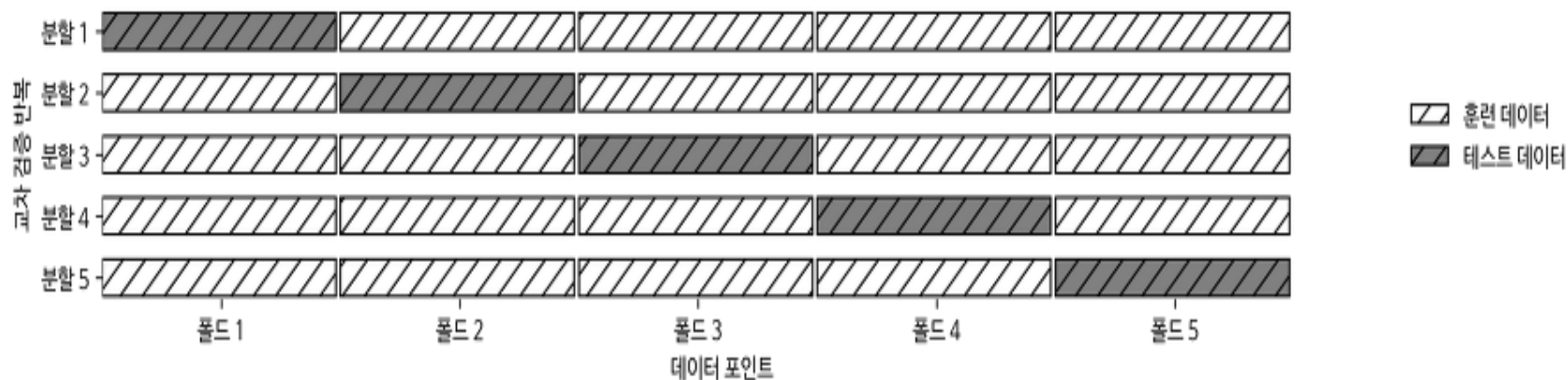
→ 좀 더 개선된 validation accuracy

→ 별도의 test set이 없기 때문에 현재 만들어진 모델의 성능이 나쁘지 않다고 판단 되어도, 실제로 일반화시킬 수 있을 지의 여부는 확신X

200개의 epoch를 돌 수 있도록 설정하였는데,

→ 결과적으로는 validation set에도 overfitting 되었을 가능성을 배제할 수 없기 때문

## k-fold cross validation

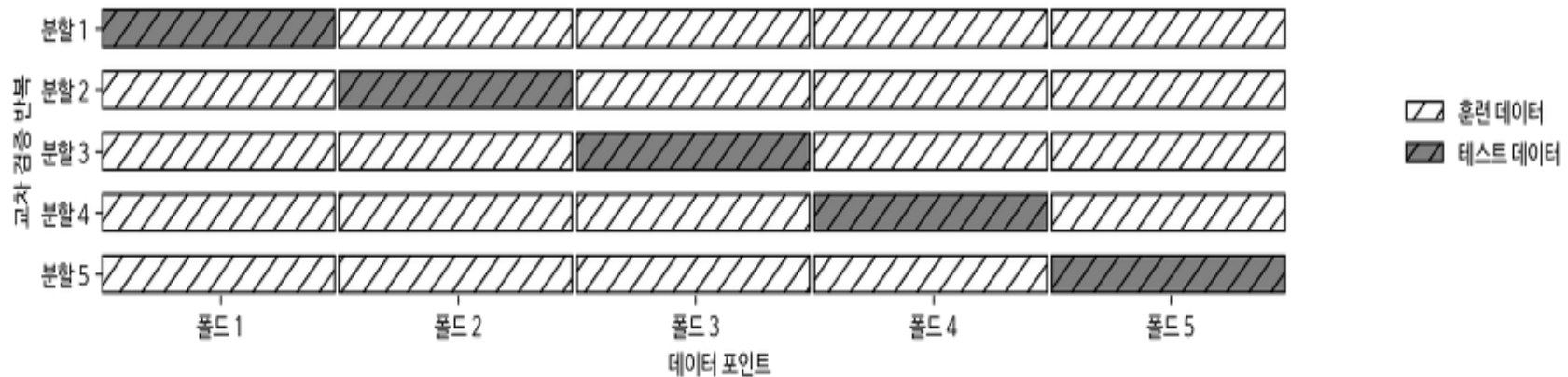


일반화의 성능을 측정하기 위해 하나의 train/validation 세트를 나누는 것보다 더 **안정적인 방법**

→ 데이터를 fold라고 하는 비슷한 크기의 부분 집합으로 여러 번 반복하여 나눔

→ 검증하고자 하는 모델을 나누어진 각 데이터 세트에 대해 학습 후 각 테스트 데이터를 이용해 평가

## k-fold cross validation



일반화의 성능을 측정하기 위해 k-fold cross validation은 반복보다 더 안정적인 방법

- 모든 데이터를 테스트 세트로 한번씩 활용 가능
- train/validation set에 있는 데이터 특성으로 인해 발생할 수 있는 편파된 결과의 가능성 감소
- 특정 validation set에 대한 overfitting 가능성 감소

## 3. 7-fold cross validation

```
kfold=KFold(n_splits=7, shuffle=True, random_state=123)
acc_per_train=[]
acc_per_fold=[]
loss_per_fold=[]
precision_per_fold=[]
recall_per_fold=[]

fold_no=1
for train, val in kfold.split(X_train, Y_train):
    model=Sequential()
    model.add(Dense(64, input_dim=117, activation='relu'))
    model.add(BatchNormalization(axis=1))
    model.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.001)))
    model.add(BatchNormalization(axis=1))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

    # Generate a print
    print('-----')
    print(f"Training for fold {fold_no} ...")

    # Fit data to model
    history = model.fit(X_train[train], Y_train[train], epochs=100, verbose=1)
    acc_per_train.append(history.history['accuracy'][-1])

    # Generate generalization metrics
    scores = model.evaluate(X_train[val], Y_train[val], verbose=0)
    pred_1 = model.predict(X_train[val])
    #print(pred_1)
    pred_idx_list=[]
    for pred in pred_1:
        if pred>0.5:
            pred_idx_list.append(1)
        else:
            pred_idx_list.append(0)
    #print(pred_idx_list)
    pred_idx_arr = np.array(pred_idx_list, dtype=np.float32)
    precision_score=precision_score(Y_train[val], pred_idx_arr)
    acc=accuracy_score(Y_train[val], pred_idx_arr)
    #print(acc)
    recall_score=recall_score(Y_train[val], pred_idx_arr)
    precision_per_fold.append(precision_score)
    recall_per_fold.append(recall_score)

    print(f"Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}: {model.metrics_names[1]} of {scores[1]*100}%")
    acc_per_fold.append(scores[1] * 100)
    loss_per_fold.append(scores[0])

    # Increase fold number
    fold_no = fold_no + 1
```

Score per fold

```
> Fold 1 - Loss: 0.5633392930030823 - Accuracy: 79.31034564971924% - Precision: 0.8333333333333334 - Recall: 0.7142857142857143
> Fold 2 - Loss: 0.537846565246582 - Accuracy: 89.28571343421936% - Precision: 0.8666666666666667 - Recall: 0.9285714285714286
> Fold 3 - Loss: 0.7385379076004028 - Accuracy: 82.14285969734192% - Precision: 0.75 - Recall: 1.0
> Fold 4 - Loss: 1.256752610206604 - Accuracy: 64.28571343421936% - Precision: 1.0 - Recall: 0.4444444444444444
> Fold 5 - Loss: 0.3527984619140625 - Accuracy: 89.28571343421936% - Precision: 0.9375 - Recall: 0.8823529411764706
> Fold 6 - Loss: 0.7832733392715454 - Accuracy: 71.42857313156128% - Precision: 0.7142857142857143 - Recall: 0.45454545454545453
> Fold 7 - Loss: 0.9370770454406738 - Accuracy: 82.14285969734192% - Precision: 0.8 - Recall: 0.9411764705882353

Average scores for all folds:
> Accuracy: 79.6973969255175 (+- 8.467378131396762)
> Loss: 0.7385178889547076
```

최종적으로 accuracy 79.6퍼센트의 모델 성능 확인

Fold에 따른 모델의 성능차가 나는 것으로 보아, 모델이 validation, train set의 형태, 특성에 따라 영향을 많이 받고 있는 것으로 추측

## 3. 7-fold cross validation

```
kfold=KFold(n_splits=7,shuffle=True,random_state=123)
acc_per_train=[]
acc_per_fold=[]
loss_per_fold=[]
precision_per_fold=[]
recall_per_fold=[]

fold_no=1
for train, val in kfold.split(X_train,Y_train):
    model=Sequential()
    model.add(Dense(64,input_dim=117,activation='relu'))
    model.add(BatchNormalization(axis=1))
    model.add(Dense(32,activation='relu',kernel_regularizer=regularizers.l2(0.001)))
    model.add(BatchNormalization(axis=1))
    model.add(Dense(8,activation='relu'))
    model.add(Dense(1,activation='sigmoid'))

    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

    # Generate a print
    print('-----')
    print('Training for fold {fold_no} ...')


    # Fit data to model
    history = model.fit(X_train[train], Y_train[train],epochs=100,verbose=1)
    acc_per_train.append(history.history['accuracy'][-1])

    # Generate generalization metrics
    scores = model.evaluate(X_train[val], Y_train[val], verbose=0)
    pred_1 = model.predict(X_train[val])
    #print(pred_1)
    pred_idx_list=[]
    for pred in pred_1:
        if pred>0.5:
            pred_idx_list.append(1)
        else:
            pred_idx_list.append(0)
    #print(pred_idx_list)
    pred_idx_arr = np.array(pred_idx_list, dtype=np.float32)
    precision_score=precision_score(Y_train[val], pred_idx_arr)
    acc=accuracy_score(Y_train[val], pred_idx_arr)
    #print(acc)
    recall_score=recall_score(Y_train[val], pred_idx_arr)
    precision_per_fold.append(precision_score)
    recall_per_fold.append(recall_score)

    print('Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
    acc_per_fold.append(scores[1] + 100)
    loss_per_fold.append(scores[0])

    # Increase fold number
    fold_no = fold_no + 1
```

Score per fold

 **DNN Framework의 한계**

> Fold 1 - Loss: 0.56333978823 - Accuracy: 79.6973969255175 - Precision: 0.8666666666666667 - Recall: 0.7142857142857143  
> Fold 2 - Loss: 0.537889541554 - Accuracy: 89.28571343421936% - Precision: 0.8666666666666667 - Recall: 0.9285714285714286  
> Fold 3 - Loss: 0.7385379076004028 - Accuracy: 82.14285969734192% - Precision: 0.75 - Recall: 1.0  
> Fold 4 - Loss: 0.3527984619140625 - Accuracy: 89.28571343421936% - Precision: 0.9375 - Recall: 0.8823529411764706  
> Fold 5 - Loss: 0.4970770454405738 - Accuracy: 87.14285969734192% - Precision: 0.8 - Recall: 0.9411764705882353  
> Fold 6 - Loss: 0.4545454545454545  
> Fold 7 - Loss: 0.4545454545454545

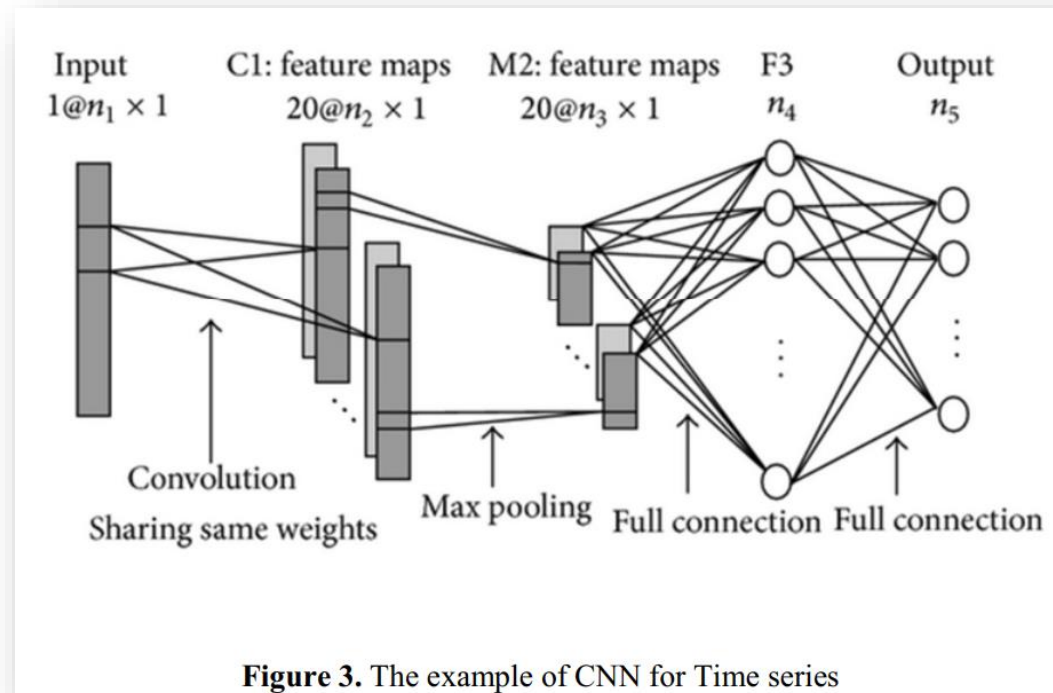
Average scores over 7 folds  
> Accuracy: 79.6973969255175 (+- 8.467378131396762)  
> Loss: 0.7385178889547076

5000개의 signal 데이터를 9개로 축소하는 과정에서  
데이터의 손실이 많이 발생해, 모델의 성능 저하로  
이어진 것으로 예상

signal의 데이터가 특정한 주기에 따라 반복되는 시퀀스  
데이터(시계열성)인데도 불구하고, 해당 DNN 모델에서는  
이러한 주기적인 정보들을 고려하지 못하는 모델이기  
때문에 추가적으로 정보가 손실된 것으로 판단.



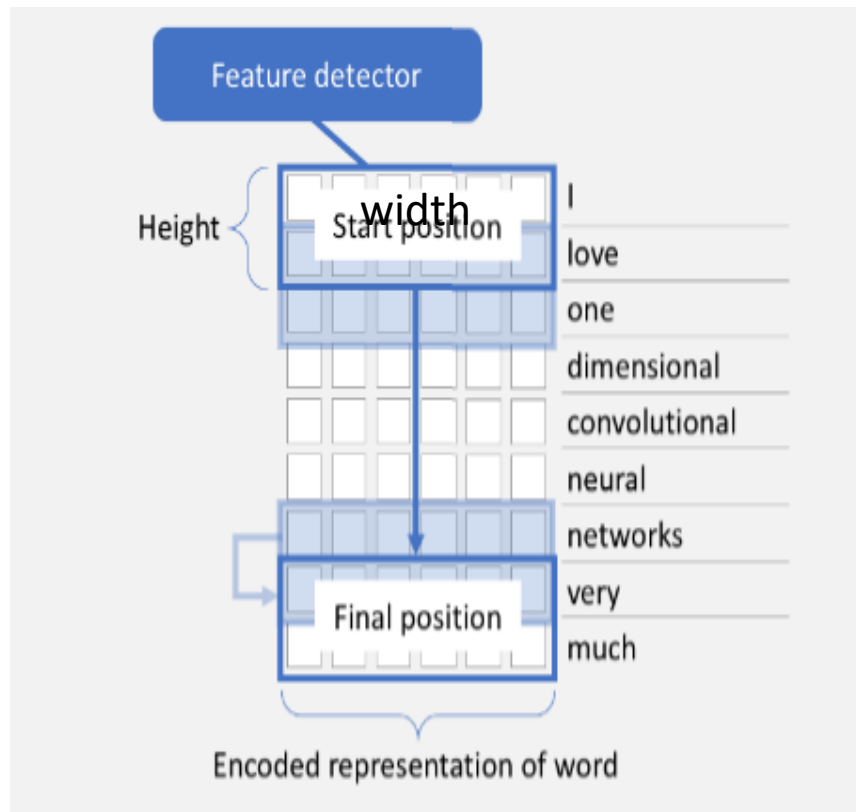
## ECG Classification을 위한 1D-CNN 논문 연구



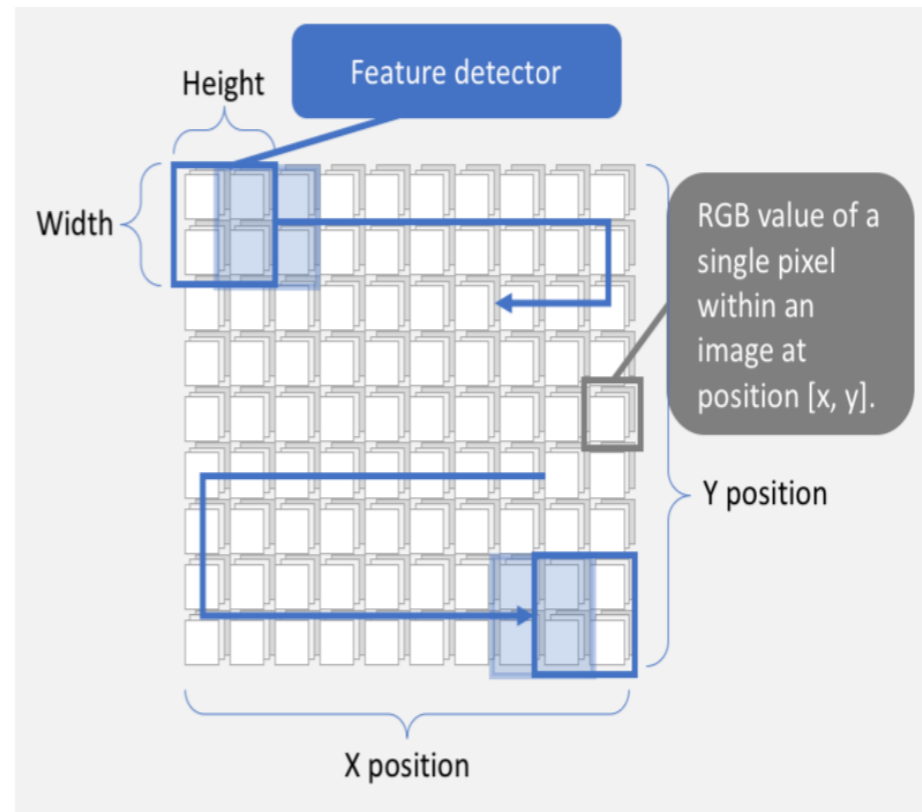
Deep Learning for ECG Classification  
B Pyakillya et al 2017 J. Phys.: Conf. Ser. 913 012004

## 1D-CNN 모델에 대한 설명

1-Dimensional CNN

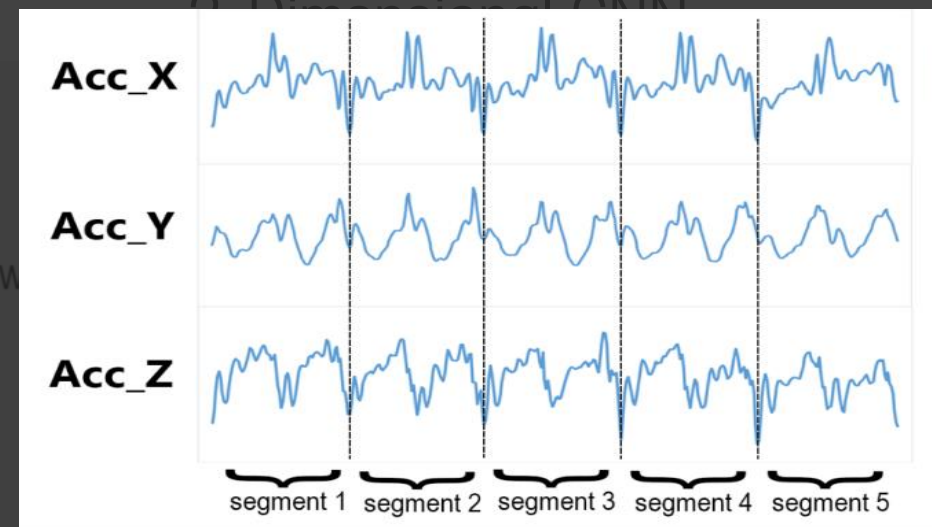
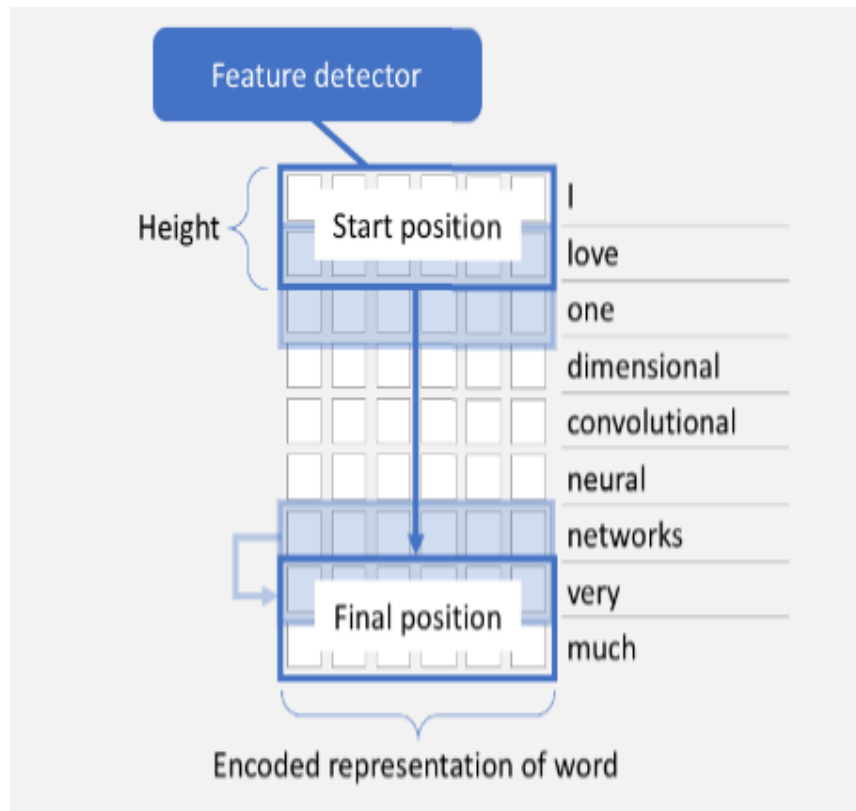


2-Dimensional CNN



## 1D-CNN 모델에 대한 설명

### 1-Dimensional CNN



모든 환자의 심전도 데이터가

→ 5000개의 심전도 전압(voltage)로 고정

→ 12가지의 시그널 태그

**1D-CNN 모델을 활용하는 것이 적합!**

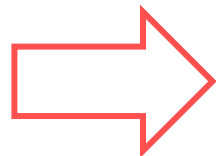
### Input 데이터 전처리

```
### X 데이터 ###
df=pd.DataFrame(x_records)
avl=np.array(df['avl'].tolist())
v1=np.array(df['v1'].tolist())
v5=np.array(df['v5'].tolist())
v6=np.array(df['v6'].tolist())

sum_x = []
for i in range(0,200) :
    one=(avl[i],v1[i],v5[i],v6[i])
    sum_x.append(one)

pre_x=np.array(sum_x)
trans_x=np.transpose(pre_x,(0,2,1))
x=trans_x[:, :, :, np.newaxis]
x.shape
```

(200, 5000, 4, 1)



좌심실 비대증을 판별하는데 관련성이 높은 정보들을 담고 있는 4개 신호를 input으로 활용

1. avl
2. v1
3. v5
4. v6

최종 input 형태: ( 200 , 5000 , 4 , 1 )

record 개수 / signal list 길이 / 신호의 종류

# 모델링 및 평가 1D-CNN 모델링

## 1D-CNN framework에 활용한 기본적인 모델 설명

### 가장 기본이 되는 모델 ###

```
import tensorflow as tf
def return_model():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(5,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation="sigmoid")(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
    print(model.summary())
    return model
```

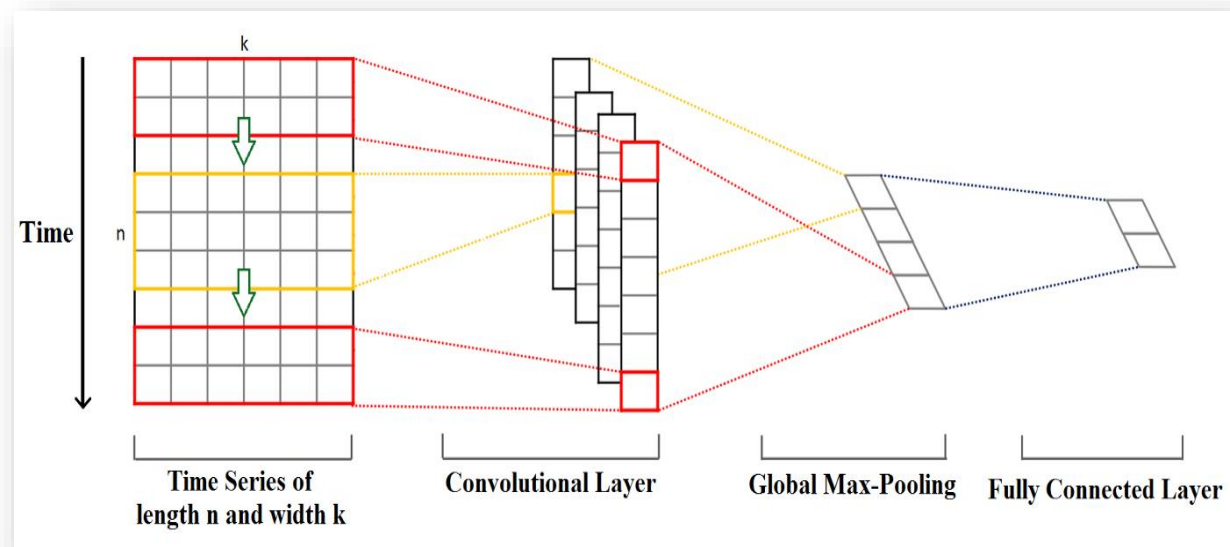
Conv-1

Conv-2

Conv-3

Conv-4

Conv-5  
+ GlobalAvgPooling



# 모델링 및 평가 1D-CNN 모델링

## 1D-CNN framework에 활용한 기본적인 모델 설명

```
### 가장 기본이 되는 모델 ###
import tensorflow as tf
def return_model():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(5,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation="sigmoid")(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
    print(model.summary())
    return model
```

Convolution layer

Dropout

GlobalAveragePooling

# 모델링 및 평가 1D-CNN 모델링

## 1D-CNN framework에 활용한 기본적인 모델 설명

```
### 가장 기본이 되는 모델 ###  
import tensorflow as tf  
def return_model():  
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(5,1),padding='valid')(input_tens)  
    x = tf.keras.layers.BatchNormalization()(x)  
    x = tf.keras.layers.ReLU()(x)  
    x = tf.keras.layers.Dropout(rate=0.5)(x)  
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)  
    x = tf.keras.layers.BatchNormalization()(x)  
    x = tf.keras.layers.ReLU()(x)  
    x = tf.keras.layers.Dropout(rate=0.5)(x)  
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)  
    x = tf.keras.layers.BatchNormalization()(x)  
    x = tf.keras.layers.ReLU()(x)  
    x = tf.keras.layers.Dropout(rate=0.5)(x)  
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)  
    x = tf.keras.layers.BatchNormalization()(x)  
    x = tf.keras.layers.ReLU()(x)  
    x = tf.keras.layers.Dropout(rate=0.5)(x)  
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)  
    x = tf.keras.layers.BatchNormalization()(x)  
    x = tf.keras.layers.ReLU()(x)  
    x = tf.keras.layers.GlobalAveragePooling2D()(x)  
    x = tf.keras.layers.Dense(1, activation="sigmoid")(x)  
    model = tf.keras.Model(inputs=input_tens, outputs=x)  
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])  
    print(model.summary())  
    return model
```

Convolution layer

Dropout

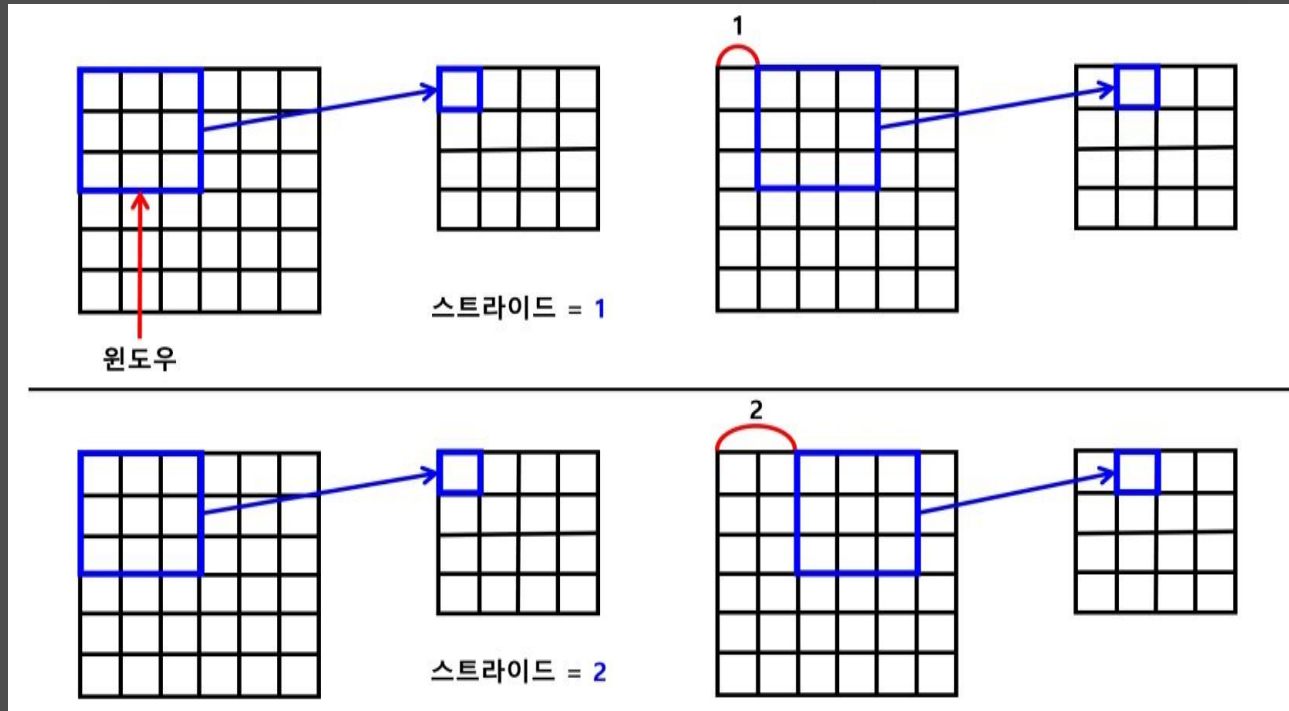
GlobalAveragePooling

# 모델링 및 평가 1D-CNN 모델링

## 1D-CNN framework에 활용한 기본적인 모델 설명

### Convolution layer

### 가장 기본이 되는 모델 ###  
`import tensorflow as tf`



```
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(inputs=input_tens, outputs=x)
model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
print(model.summary())
return model
```

### Kernel\_size

지역 패턴 학습을 위한  
윈도우 사이즈 parameter

### Stride

입력 데이터에 kernel을  
적용할 때 이동할 간격을  
조정하는 parameter

첫 Parameter 기준점

- **kernel\_size : (250,4)**
- **strides : (5,1)**



# 모델링 및 평가 1D-CNN 모델링

## 1D-CNN framework에 활용한 기본적인 모델 설명

```
### 가장 기본이 되는 모델 ###
import tensorflow as tf
def return_model1():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(5,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation="sigmoid")(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
    print(model.summary())
    return model
```

Convolution layer

Dropout

GlobalAveragePooling

# 모델링 및 평가 1D-CNN 모델링

## 1D-CNN framework에 활용한 기본적인 모델 설명

### Dropout

### 가장 기본이 되는 모델 ###

```
import tensorflow as tf
```

```
def return_model():
```

```
    input_tens = tf.keras.Input(shape=(5000, 4, 1))
```

```
    x = tf.keras.layers.Conv2D(256, kernel_size=(3, 3),
```

```
    x = tf.keras.layers.BatchNormalization(x)
```

```
    x = tf.keras.layers.ReLU(x)
```

```
    x = tf.keras.layers.Dropout(rate=0.5)(x)
```

```
    x = tf.keras.layers.Conv2D(512, kernel_size=(3, 3),
```

```
    x = tf.keras.layers.BatchNormalization(x)
```

```
    x = tf.keras.layers.ReLU(x)
```

```
    x = tf.keras.layers.Dropout(rate=0.5)(x)
```

```
    x = tf.keras.layers.Conv2D(128, kernel_size=(3, 3),
```

```
    x = tf.keras.layers.BatchNormalization(x)
```

```
    x = tf.keras.layers.ReLU(x)
```

```
    x = tf.keras.layers.Dropout(rate=0.5)(x)
```

```
    x = tf.keras.layers.Conv2D(64, kernel_size=(3, 3),
```

```
    x = tf.keras.layers.BatchNormalization(x)
```

```
    x = tf.keras.layers.ReLU(x)
```

```
    x = tf.keras.layers.GlobalAveragePooling2D(x)
```

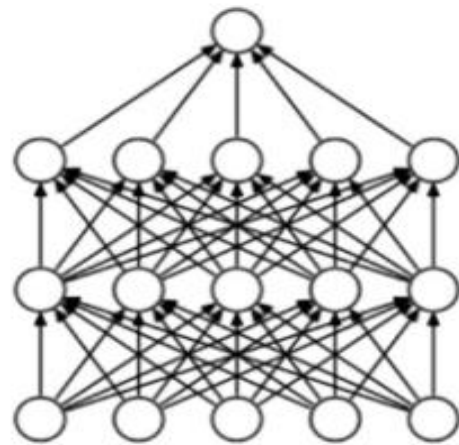
```
    x = tf.keras.layers.Dense(10)(x)
```

```
    model = tf.keras.Model(inputs=input_tens, outputs=x)
```

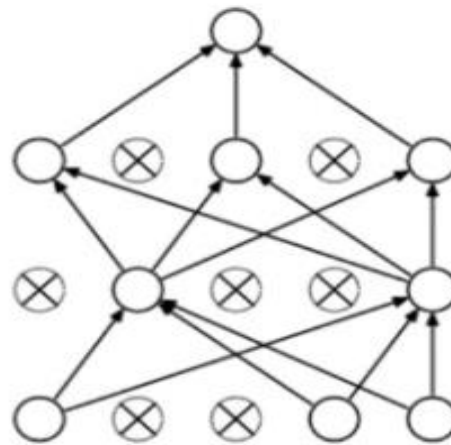
```
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
```

```
    print(model.summary())
```

```
    return model
```



(a) Standard Neural Net



(b) After applying dropout.

인공 신경망의 뉴런을 확률적으로 사용하지 않음으로써 과적합을 방지

# 모델링 및 평가 1D-CNN 모델링

## 1D-CNN framework에 활용한 기본적인 모델 설명

```
### 가장 기본이 되는 모델 ###
import tensorflow as tf
def return_model1():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(5,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    tf.keras.layers.GlobalAveragePooling2D()(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
    print(model.summary())
    return model
```

Convolution layer

Dropout

GlobalAveragePooling

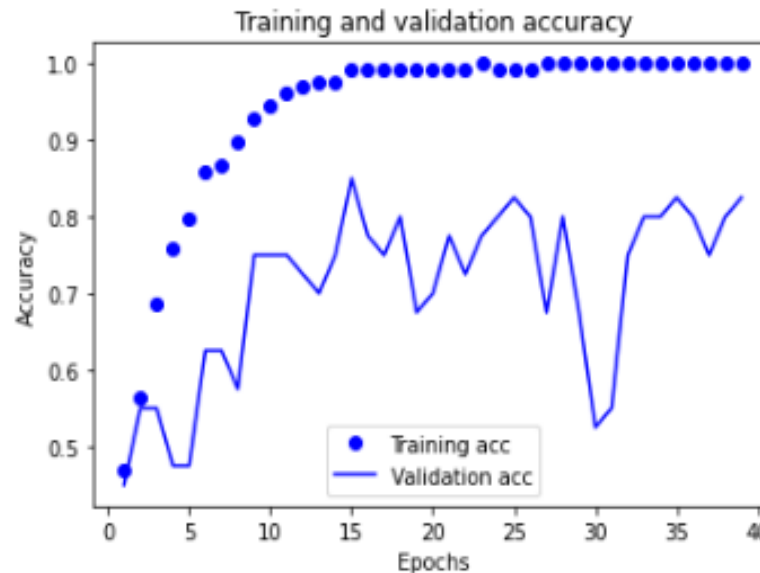
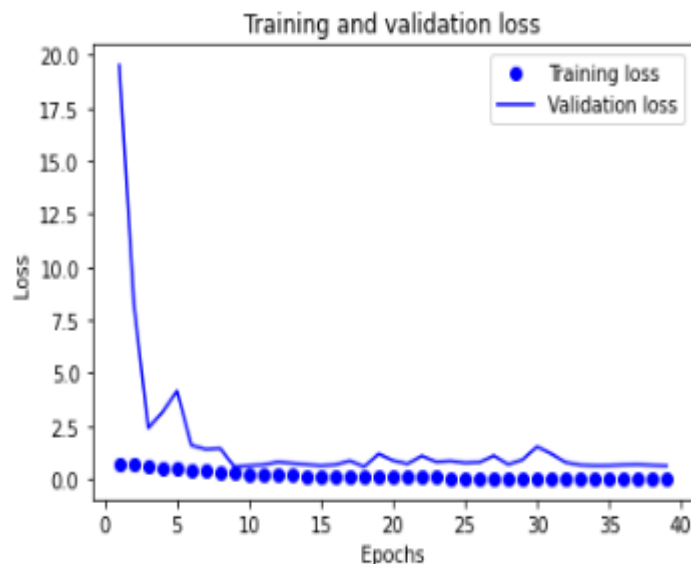


# 모델링 및 평가 1D-CNN 모델링

## Train, Validation, Test

```
[ ] ### 테스트 셋과 트레이닝 셋 ###  
from sklearn.model_selection import train_test_split  
x_train_all, x_test, y_train_all, y_test = train_test_split(x, y, stratify=y, test_size=.2, random_state=0)  
  
[ ] ### 트레이닝 셋과 validation 셋 ###  
x_train, x_val, y_train, y_val = train_test_split(x_train_all, y_train_all,  
                                                  stratify=y_train_all,  
                                                  test_size=.2,  
                                                  random_state=0)
```

train / test : 0.8: 0.2 로 분할  
train / validation: 0.8 : 0.2로 분할



test accuracy: 0.82  
precision score: 0.857  
recall score: 0.818

# 모델링 및 평가 1D-CNN 모델링

## Train, Validation, Test

```
[ ] ### 테스트 셋과 트레이닝 셋 ###
from sklearn.model_selection import train_test_split
x_train_all, x_test, y_train_all, y_test = train_test_split(x, y, stratify=y, test_size=.2, random_state=0)

[ ] ### 트레이닝 셋과 validation 셋 ###
x_train, x_val, y_train, y_val = train_test_split(x_train_all, y_train_all,
                                                  stratify=y_train_all,
                                                  test_size=.2,
                                                  random_state=0)
```

train / test : 0.8: 0.2 로 분할  
train / validation: 0.8 : 0.2로 분할

Score per fold

```
> Fold 1 - Loss: 0.634103536605835 - Accuracy: 75.86206793785095% Precision: 0.75 - Recall: 0.8823529411764706-AP : 0.7307
> Fold 2 - Loss: 0.13012497127056122 - Accuracy: 89.65517282485962% Precision: 0.85 - Recall: 1.0-AP : 0.85
> Fold 3 - Loss: 0.11483889818191528 - Accuracy: 96.55172228813171% Precision: 0.9375 - Recall: 1.0-AP : 0.9375
> Fold 4 - Loss: 0.12708641588687897 - Accuracy: 96.55172228813171% Precision: 0.9444444444444444 - Recall: 1.0-AP : 0.9444
> Fold 5 - Loss: 0.0196569561958313 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 6 - Loss: 0.028140777722001076 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 7 - Loss: 0.016056489199399948 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
```

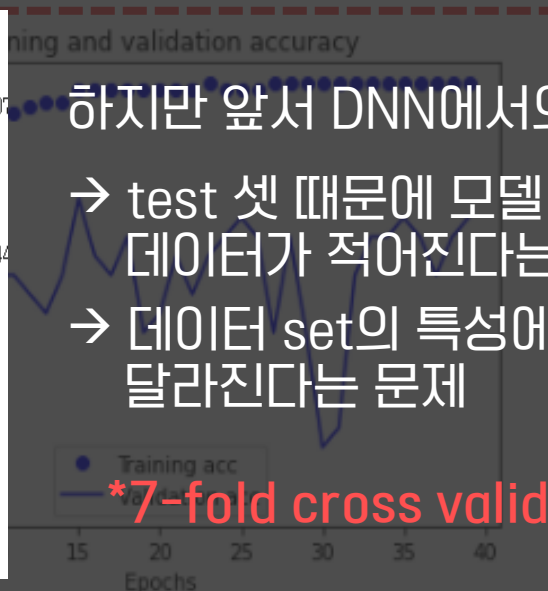
```
Average scores for all folds:
> Accuracy: 94.08866933413914 (+- 8.18386601088431)
> Loss: 0.15285829215177468
> AP: 0.9232392382240252
```

하지만 앞서 DNN에서의 문제와 마찬가지로,

→ test 셋 때문에 모델 train에 사용되는 데이터가 적어진다는 문제

→ 데이터 set의 특성에 따라 모델 성능이 달라진다는 문제

\*7-fold cross validation 활용하기로 결정



# 모델링 및 평가 1D-CNN 모델링

## 모델의 성능 개선을 위한 parameter 조정: kernel\_size

```
### 커널 사이즈 (500,4) ###
import tensorflow as tf
def return_model2():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(500,4), strides=(5,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation="sigmoid")(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
    print(model.summary())
    return model
```

**kernel\_size : (500,4)**

```
### 커널 사이즈 (400,4) ###
import tensorflow as tf
def return_model3():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(400,4), strides=(5,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation="sigmoid")(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])
    print(model.summary())
    return model
```

**kernel\_size : (400,4)**

# 모델링 및 평가 1D-CNN 모델링

## 모델의 성능 개선을 위한 parameter 조정: kernel\_size

```
## 커널 사이즈 (500,4) ##  
import tensorflow as tf  
def return_model2():  
    back_bone = tf.keras.layers.Conv1D(500, 4, 1)
```

**kernel\_size : (500,4)**

Score per fold

```
> Fold 1 - Loss: 0.9738743305206299 - Accuracy: 82.75862336158752% Precision: 0.7727272727272727 - Recall: 1.0-AP : 0.7727272727272727  
> Fold 2 - Loss: 0.110985666513443 - Accuracy: 93.1034505367279% Precision: 0.8947368421052632 - Recall: 1.0-AP : 0.8947368421052632  
> Fold 3 - Loss: 0.48077720403671265 - Accuracy: 75.86206793785095% Precision: 0.6818181818181818 - Recall: 1.0-AP : 0.6818181818181818  
> Fold 4 - Loss: 0.43120673298835754 - Accuracy: 89.65517282485962% Precision: 0.85 - Recall: 1.0-AP : 0.85  
> Fold 5 - Loss: 0.15697050094604492 - Accuracy: 92.85714030265808% Precision: 0.875 - Recall: 1.0-AP : 0.875  
> Fold 6 - Loss: 0.07603410631418228 - Accuracy: 92.85714030265808% Precision: 0.875 - Recall: 1.0-AP : 0.875  
> Fold 7 - Loss: 0.05199926346540451 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333
```

Average scores for all folds:

```
> Accuracy: 89.07459548541478 (+/- 6.6946397421966815)  
> Loss: 0.3259782578263964  
> AP: 0.840373661426293
```

**Evaluation Result**

**Accuracy: 89.07**

**AP: 0.84**

```
x = tf.keras.layers.ReLU()(x)  
x = tf.keras.layers.Dropout(rate=0.5)(x)  
x = tf.keras.layers.Conv1D(500, 4, 1)(x)  
x = tf.keras.layers.BatchNormalization()(x)  
x = tf.keras.layers.ReLU()(x)  
x = tf.keras.layers.GlobalMaxPooling1D()(x)  
x = tf.keras.layers.Dense(10, activation='softmax')(x)  
model = tf.keras.Model(inputs=x.inputs[0], outputs=x.outputs[0])  
model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])  
print(model.summary())  
return model
```

```
## 커널 사이즈 (400,4) ##  
import tensorflow as tf  
def return_model3():
```

**kernel\_size : (400,4)**

Score per fold

```
> Fold 1 - Loss: 0.9607096910476685 - Accuracy: 75.86206793785095% Precision: 0.75 - Recall: 0.8823529411764706-AP : 0.7307692307692308  
> Fold 2 - Loss: 0.045207489281892776 - Accuracy: 96.55172228813171% Precision: 0.9444444444444444 - Recall: 1.0-AP : 0.9444444444444444  
> Fold 3 - Loss: 0.09420929104089737 - Accuracy: 96.55172228813171% Precision: 0.9375 - Recall: 1.0-AP : 0.9375  
> Fold 4 - Loss: 0.06458041071891785 - Accuracy: 96.55172228813171% Precision: 1.0 - Recall: 0.9411764705882353-AP : 0.975609756097561  
> Fold 5 - Loss: 0.12348861247301102 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333  
> Fold 6 - Loss: 0.002559548243880272 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0  
> Fold 7 - Loss: 0.054409485310316086 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333
```

Average scores for all folds:

```
> Accuracy: 94.05348300933838 (+/- 7.524192897229637)  
> Loss: 0.19216636115951197  
> AP: 0.9221429376348241
```

**Evaluation Result**

**Accuracy: 94.05**

**AP: 0.92**

```
x = tf.keras.layers.ReLU()(x)  
x = tf.keras.layers.Dropout(rate=0.5)(x)  
x = tf.keras.layers.Conv1D(400, 4, 1)(x)  
x = tf.keras.layers.BatchNormalization()(x)  
x = tf.keras.layers.ReLU()(x)  
x = tf.keras.layers.GlobalMaxPooling1D()(x)  
x = tf.keras.layers.Dense(10, activation='softmax')(x)  
model = tf.keras.Model(inputs=x.inputs[0], outputs=x.outputs[0])  
model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=["accuracy"])  
print(model.summary())  
return model
```



# 모델링 및 평가 1D-CNN 모델링

## 모델의 성능 개선을 위한 parameter 조정: kernel\_size

**kernel\_size : (250,4)**

```
-----
Score per fold
-----
> Fold 1 - Loss: 0.634103536605835 - Accuracy: 75.86206793785095% Precision: 0.75 - Recall: 0.8823529411764706-AP : 0.7307
-----
> Fold 2 - Loss: 0.13012497127056122 - Accuracy: 89.65517282485962% Precision: 0.85 - Recall: 1.0-AP : 0.85
-----
> Fold 3 - Loss: 0.11483889818191528 - Accuracy: 96.55172228813171% Precision: 0.9375 - Recall: 1.0-AP : 0.9375
-----
> Fold 4 - Loss: 0.12708641588687897 - Accuracy: 96.55172228813171% Precision: 0.9444444444444444 - Recall: 1.0-AP : 0.9444
-----
> Fold 5 - Loss: 0.0196569561958313 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
-----
> Fold 6 - Loss: 0.028140777722001076 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
-----
> Fold 7 - Loss: 0.016056489199399948 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
-----

Average scores for all folds:
> Accuracy: 94.08866933413914 (+- 8.18386601088431)
> Loss: 0.15285829215177468
> AP: 0.9232392382240252
-----
```

	250	500	400
Accuracy	94	89	94
Ap	92	64	92

# 모델링 및 평가 1D-CNN 모델링

## 모델의 성능 개선을 위한 parameter 조정: stride

### stride : (10,1)

```
## 스트라이드 사이즈 (10,1) ##
import tensorflow as tf
def return_model4():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(10,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=['accuracy'])
    print(model.summary())
    return model
```

### stride : (50,1)

```
## 스트라이드 사이즈 (50,1) ##
import tensorflow as tf
def return_model5():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(50,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=['accuracy'])
    print(model.summary())
    return model
```

### stride : (125,1)

```
## 스트라이드 사이즈 (125,1) ##
import tensorflow as tf
def return_model7():
    input_tens = tf.keras.Input(shape=(5000,4,1))
    x = tf.keras.layers.Conv2D(256, kernel_size=(250,4), strides=(125,1),padding='valid')(input_tens)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(512, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(128, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.Conv2D(64, kernel_size=(5,1), padding='valid')(x)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.ReLU()(x)
    x = tf.keras.layers.Dropout(rate=0.5)(x)
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
    model = tf.keras.Model(inputs=input_tens, outputs=x)
    model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=['accuracy'])
    print(model.summary())
    return model
```

# 모델링 및 평가 1D-CNN 모델링

모델의 성능 개선을 위한 parameter 조정: stride

stride : (10,1)

```
## 스트라이드 사이즈 (10,1) ##
import tensorflow as tf
def return_model(4):
    input_tens = tf.keras.Input(shape=(5000, 4, 1))

Score per fold
> Fold 1 - Loss: 0.7109609246253967 - Accuracy: 82.75862336158752% Precision: 0.8333333333333334 - Recall: 0.8823529411764706-AP : 0.8042596348884382
> Fold 2 - Loss: 0.15787300128936768 - Accuracy: 96.55172238913171% Precision: 0.9444444444444444 - Recall: 1.0-AP : 0.9444444444444444
> Fold 3 - Loss: 0.05475236398112831 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 4 - Loss: 0.029198916628956795 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 5 - Loss: 0.0816337913274765 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333
> Fold 6 - Loss: 0.1008332895252727 - Accuracy: 96.42857313156128% Precision: 1.0 - Recall: 0.9285714285714286-AP : 0.9642857142857143
> Fold 7 - Loss: 0.11702698469161987 - Accuracy: 96.42857313156128% Precision: 1.0 - Recall: 0.9285714285714286-AP : 0.9642857142857143

Average scores for all folds:
> Accuracy: 95.51372357777187 (+/- 5.4318403121791405)
> Loss: 0.17886847018131188
> AP: 0.9443726918353778
```

Evaluation Result

Accuracy: 95.51  
AP: 0.944

```
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.Conv1D(128, kernel_size=5, padding='valid')(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.Conv1D(64, kernel_size=5, padding='valid')(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.GlobalAveragePooling1D()(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(inputs=input_tens, outputs=x)
model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=['accuracy'])
print(model.summary())
return model
```

stride : (50,1)

```
## 스트라이드 사이즈 (50,1) ##
import tensorflow as tf
def return_model(5):
    input_tens = tf.keras.Input(shape=(5000, 4, 1))

Score per fold
> Fold 1 - Loss: 0.9910569058189392 - Accuracy: 75.86206793789509% Precision: 0.75 - Recall: 0.8823529411764706-AP : 0.7307302231237323
> Fold 2 - Loss: 0.039236653596162796 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 3 - Loss: 2.321249009178711 - Accuracy: 62.068953050842295% Precision: 1.0 - Recall: 0.26666666666666666-AP : 0.6459770114942529
> Fold 4 - Loss: 0.04439346023583412 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 5 - Loss: 0.015340731479227543 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 6 - Loss: 0.0068758102133870125 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 7 - Loss: 0.19388794898968616 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333

Average scores for all folds:
> Accuracy: 90.62280058860779 (+/- 14.235947138668744)
> Loss: 0.5160050740731614
> AP: 0.9014343668501884
```

Evaluation Result

Accuracy: 90.6  
AP: 0.901

```
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.Conv1D(128, kernel_size=5, padding='valid')(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.Conv1D(64, kernel_size=5, padding='valid')(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.GlobalAveragePooling1D()(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(inputs=input_tens, outputs=x)
model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=['accuracy'])
print(model.summary())
return model
```

stride : (125,1)

```
## 스트라이드 사이즈 (125,1) ##
import tensorflow as tf
def return_model(7):
    input_tens = tf.keras.Input(shape=(5000, 4, 1))

Score per fold
> Fold 1 - Loss: 1.3473494052886963 - Accuracy: 79.31034564971924% Precision: 0.8235294117647058 - Recall: 0.8235294117647058-AP : 0.7816489679035914
> Fold 2 - Loss: 0.5899624824523326 - Accuracy: 82.75862336158752% Precision: 0.9285714285714286 - Recall: 0.7647058823529411-AP : 0.8480150680962041
> Fold 3 - Loss: 0.07945370674133301 - Accuracy: 96.55172238913171% Precision: 0.9375 - Recall: 1.0-AP : 0.9375
> Fold 4 - Loss: 0.012656772503217262 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 5 - Loss: 0.20868688940412604 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333
> Fold 6 - Loss: 0.008039540611207485 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 7 - Loss: 0.002150089770218134 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0

Average scores for all folds:
> Accuracy: 93.57848634726568 (+/- 8.118056711568965)
> Loss: 0.3211862710304558
> AP: 0.9286424813333041
```

Evaluation Result

Accuracy: 93.57  
AP: 0.928

```
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.Conv1D(128, kernel_size=5, padding='valid')(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.Conv1D(64, kernel_size=5, padding='valid')(x)
x = tf.keras.layers.ReLU()(x)
x = tf.keras.layers.Dropout(rate=0.5)(x)
x = tf.keras.layers.GlobalAveragePooling1D()(x)
x = tf.keras.layers.Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(inputs=input_tens, outputs=x)
model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.binary_crossentropy, metrics=['accuracy'])
print(model.summary())
return model
```

# 모델링 및 평가 1D-CNN 모델링

모델의 성능 개선을 위한 parameter 조정: stride

stride : (10,1)

```
## 스트라이드 사이즈 (10,1) ##
import tensorflow as tf
def return_model4():
    input_tens = tf.keras.Input(shape=(5000, 4, 1))

Score per fold
> Fold 1 - Loss: 0.7109609246253967 - Accuracy: 82.75862336158752% Precision: 0.8333333333333334 - Recall: 0.8823529411764706-AP : 0.8042596348884382
> Fold 2 - Loss: 0.15787300128936788 - Accuracy: 96.55172228813171% Precision: 0.9444444444444444 - Recall: 1.0-AP : 0.9444444444444444
> Fold 3 - Loss: 0.05475238398112831 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 4 - Loss: 0.029198916628956795 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 5 - Loss: 0.0816337913274765 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333
> Fold 6 - Loss: 0.1008332895252727 - Accuracy: 96.42857313156128% Precision: 1.0 - Recall: 0.9285714285714286-AP : 0.9642857142857143
> Fold 7 - Loss: 0.11702698469161987 - Accuracy: 96.42857313156128% Precision: 1.0 - Recall: 0.9285714285714286-AP : 0.9642857142857143

Average scores for all folds:
> Accuracy: 95.51372357777187 (+- 5.4318403121791405)
> Loss: 0.17886847018131188
> AP: 0.9445725918353778
```

Evaluation Result

Accuracy: 95.51  
AP: 0.944

stride : (50,1)

```
## 스트라이드 사이즈 (50,1) ##
import tensorflow as tf
def return_model5():
    input_tens = tf.keras.Input(shape=(5000, 4, 1))

Score per fold
> Fold 1 - Loss: 0.9910559058189392 - Accuracy: 75.86206793785095% Precision: 0.75 - Recall: 0.8823529411764706-AP : 0.7307302231237323
> Fold 2 - Loss: 0.039236653596162796 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 3 - Loss: 2.32124900178711 - Accuracy: 62.068953050842295% Precision: 1.0 - Recall: 0.2666666666666666-AP : 0.6459770114942529
> Fold 4 - Loss: 0.04439346023583412 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 5 - Loss: 0.015340731479227543 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 6 - Loss: 0.0068758102133870125 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 7 - Loss: 0.1938879489896816 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333

Average scores for all folds:
> Accuracy: 90.62280058860779 (+- 14.235947138668744)
> Loss: 0.5160050740731614
> AP: 0.9014343668501884
```

Evaluation Result

Accuracy: 90.6  
AP: 0.901

stride : (125,1)

```
## 스트라이드 사이즈 (125,1) ##
import tensorflow as tf
def return_model7():
    input_tens = tf.keras.Input(shape=(5000, 4, 1))

Score per fold
> Fold 1 - Loss: 1.3473494052886963 - Accuracy: 79.31034564971924% Precision: 0.8235294117647058 - Recall: 0.8235294117647058-AP : 0.7816489679035914
> Fold 2 - Loss: 0.5899624824523326 - Accuracy: 82.75862336158752% Precision: 0.9285714285714286 - Recall: 0.7647058823529411-AP : 0.8480150680962041
> Fold 3 - Loss: 0.07945370674133301 - Accuracy: 96.55172228813171% Precision: 0.9375 - Recall: 1.0-AP : 0.9375
> Fold 4 - Loss: 0.012656772503217262 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 5 - Loss: 0.2086868894012604 - Accuracy: 96.42857313156128% Precision: 0.9333333333333333 - Recall: 1.0-AP : 0.9333333333333333
> Fold 6 - Loss: 0.008039540611207485 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0
> Fold 7 - Loss: 0.002150089770218134 - Accuracy: 100.0% Precision: 1.0 - Recall: 1.0-AP : 1.0

Average scores for all folds:
> Accuracy: 93.5784634728568 (+- 8.118056711568965)
> Loss: 0.3211862710304558
> AP: 0.9286424813333041
```

Evaluation Result

Accuracy: 93.57  
AP: 0.928

# 모델링 및 평가

## 1D-CNN 모델링

### 최종 모델 선정

Model: "functional\_3"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 5000, 4, 1)]	0
conv2d_5 (Conv2D)	(None, 951, 1, 256)	256256
batch_normalization_5 (Batch Normalization)	(None, 951, 1, 256)	1024
re_lu_5 (ReLU)	(None, 951, 1, 256)	0
conv2d_6 (Conv2D)	(None, 947, 1, 512)	655872
batch_normalization_6 (Batch Normalization)	(None, 947, 1, 512)	2048
re_lu_6 (ReLU)	(None, 947, 1, 512)	0
conv2d_7 (Conv2D)	(None, 943, 1, 512)	1311232
batch_normalization_7 (Batch Normalization)	(None, 943, 1, 512)	2048
re_lu_7 (ReLU)	(None, 943, 1, 512)	0
conv2d_8 (Conv2D)	(None, 939, 1, 128)	327808
batch_normalization_8 (Batch Normalization)	(None, 939, 1, 128)	512
re_lu_8 (ReLU)	(None, 939, 1, 128)	0
dropout_1 (Dropout)	(None, 939, 1, 128)	0
conv2d_9 (Conv2D)	(None, 935, 1, 64)	41024
batch_normalization_9 (Batch Normalization)	(None, 935, 1, 64)	256
re_lu_9 (ReLU)	(None, 935, 1, 64)	0
global_average_pooling2d_1 (Global Average Pooling)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 2,598,145  
Trainable params: 2,595,201  
Non-trainable params: 2,944

None

### 1D-CNN Model

**kernel\_size: (250,4)**  
**stride: (10,1)**

**Accuracy: 95.51**  
**AP: 0.944**



# 04 결론

결론 및 의의  
한계점 및 개선 방향성

### 결론 및 의의

1. 데이터를 다방면으로 활용해 전처리와 시각화에 많은 노력
2. 다양한 input data 형태를 활용하여 걸맞는 모델에 각각 적용
3. 95퍼센트의 Accuracy로 높은 성능을 가진 모델 최종 선정
4. K-fold 를 사용하는 등, 적은 양의 데이터를 활용하며 과적합의 문제를 고려하기 위해 노력

## 한계점 및 개선 방향성

1. 모델의 파라미터 조정을 할 수 있는 시간이 더 많았다면 kernel, stride에 대한 복합적인 경우의 수도 고려 가능
2. 데이터 개수가 더 컸다면 모델의 training이 더 잘되지 않았을까
3. 좌심실 비대증 유무 판별에 결정적인 기준이 되어주는 절대적인 값이 Normalize 과정으로 적용이 어려워 짐  
→ 절대적인 값 정보를 반영해주는 변수를 추가했다면 더 나은 성능 기대
4. 다양한 input 데이터 형태의 경우의 수를 고려하지 못함





# THANK YOU

감사합니다