손동작 분류 경진대회

김다현, 윤태승, 이보원

손동작 분류 경진대회

데이콘 베이직 Basic | 정형 | Accuracy

상금 : 참가시 최소 50 XP, 특별상 데이콘 후드

2022.03.07 ~ 2022.03.18 17:59 + Google Calendar

535명 마감

1. 배경

안녕하세요 여러분! **손동작 분류 경진대회**에 오신 것을 환영합니다.

총 4개의 종류의 손동작 데이터셋을 통해 **데이터 분석 대회에 입문해보세요.**

다른 사람들과 실력을 겨뤄보며 데이터 분석 **대회의 즐거움을 느껴보세요.**

2. 목적

손에 부착된 센서의 데이터를 통해 총 4개의 종류의 손동작을 분류해보세요!

주어진 데이터 **이외의 데이터는 사용 금지!**

1. 평가

- 리더보드
  - 평가 산식 : accuracy score
  - public score : 전체 테스트 데이터 중 50%
  - private score : 전체 테스트 데이터 중 50%

- 리더보드 수상자 (1~3등) 코드 평가
  a. Private 순위 공개 후 코드 제출 기간 내 코드 공유 게시판에 게시
    i. 제목 양식 : 팀 이름, Private 순위와 점수, 모델 이름 (e.g. 데이콘팀, Private 1위, Private 점수 :5.23, RandomForest)
    ii. 내용 : 전처리, 학습, 후처리, 추론 일련의 과정을 담은 코드 및 코드 설명을 게시.
  b. 참가자들의 "좋아요", "댓글" 및 정성 평가를 통해 특별상 수상자 선정

2. 개인 또는 팀 참여 규칙

- 1일 최대 제출 횟수 : 3회
- 개인으로만 참여할 수 있습니다. (팀 구성 X)
- 개인 참가 방법 : 팀 신청 없이, 자유롭게 제출 창에서 제출 가능

1. train.csv : 학습 데이터

- id : 샘플 아이디
- sensor_1 ~ sensor_32 : 센서 데이터
- target : 손동작 class

2. test.csv : 테스트 데이터

- id : 샘플 아이디
- sensor_1 ~ sensor_32 : 센서 데이터

3. sample_submissoin.csv : 제출 양식

- id : 샘플 아이디
- target : 손동작 class

TITLE

EDA

ENSEMBLE

DEEP LEARNING

EDA

Exploratory Data Analysis

EDA

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from google.colab import drive
drive.mount('/content/gdrive')
```

```python
data = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/train.csv')
```

```python
data = data.drop('id', axis=1) #data에서 필요없는 id column 제거
data
```

	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7	sensor_8	sensor_9	sensor_10	...
0	-6.149463	-0.929714	9.058368	-7.017854	-2.958471	0.179233	-0.956591	-0.972401	5.956213	4.145636	...
1	-2.238836	-1.003511	5.098079	-10.880357	-0.804562	-2.992123	26.972724	-8.900861	-5.968298	-4.060134	...
2	19.087934	-2.092514	0.946750	-21.831788	9.119235	17.853587	-21.069954	-15.933212	-9.016039	-5.975194	...
3	-2.211629	-1.930904	21.888406	-3.067560	-0.240634	2.985056	-29.073369	0.200774	-1.043742	2.099845	...
4	3.953852	2.964892	-36.044802	0.899838	26.930210	11.004409	-21.962423	-11.950189	-20.933785	-4.000506	...
...	...	...	...	...	...	...	...	...	...	...	...
2330	-3.971043	39.913391	16.034626	-19.067697	8.061361	-70.916786	-39.937026	12.834223	-21.937973	14.942994	...
2331	-3.011710	-4.060355	-1.046067	4.178137	-2.003243	-2.895017	-2.766757	-29.099123	-4.208953	-4.793855	...
2332	-9.001824	5.985711	-8.146347	-10.902201	5.102105	8.133692	32.877614	-3.017438	-3.174442	-5.724941	...
2333	-3.987992	3.011460	-11.949323	-3.810885	16.880234	-5.150117	9.182801	4.960190	-21.002525	-1.881519	...
2334	-1.838225	-7.023497	-45.877365	20.026927	4.058551	8.062100	19.083782	-21.881795	-9.106341	-1.056355	...

2335 rows × 33 columns

EDA

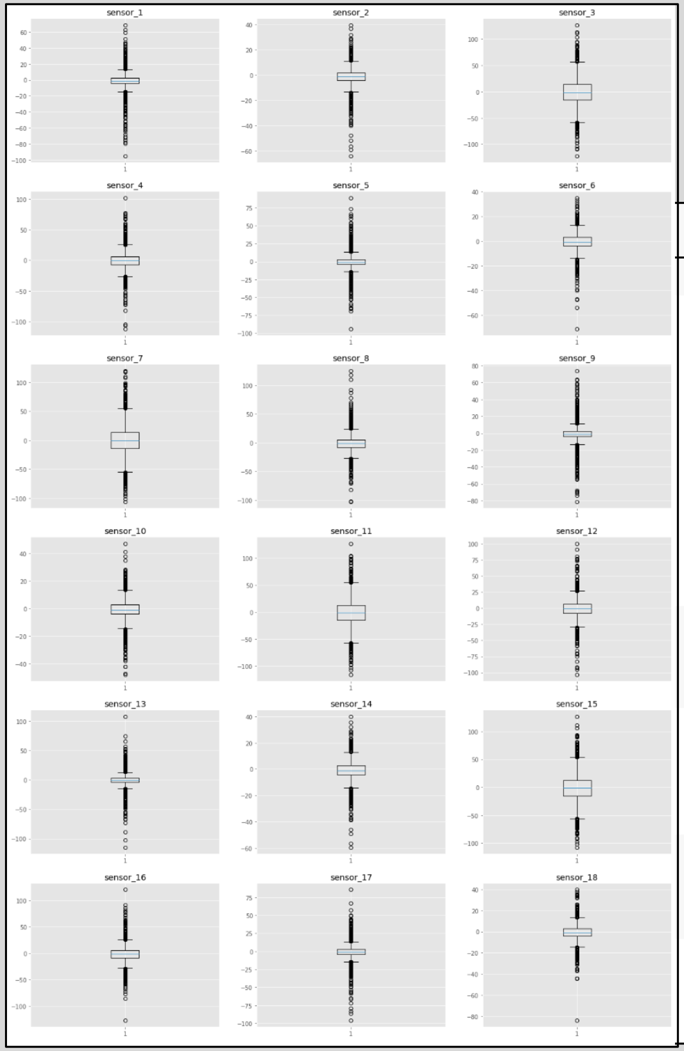
```
# 결측값 유무 확인
data.isnull().sum()
```

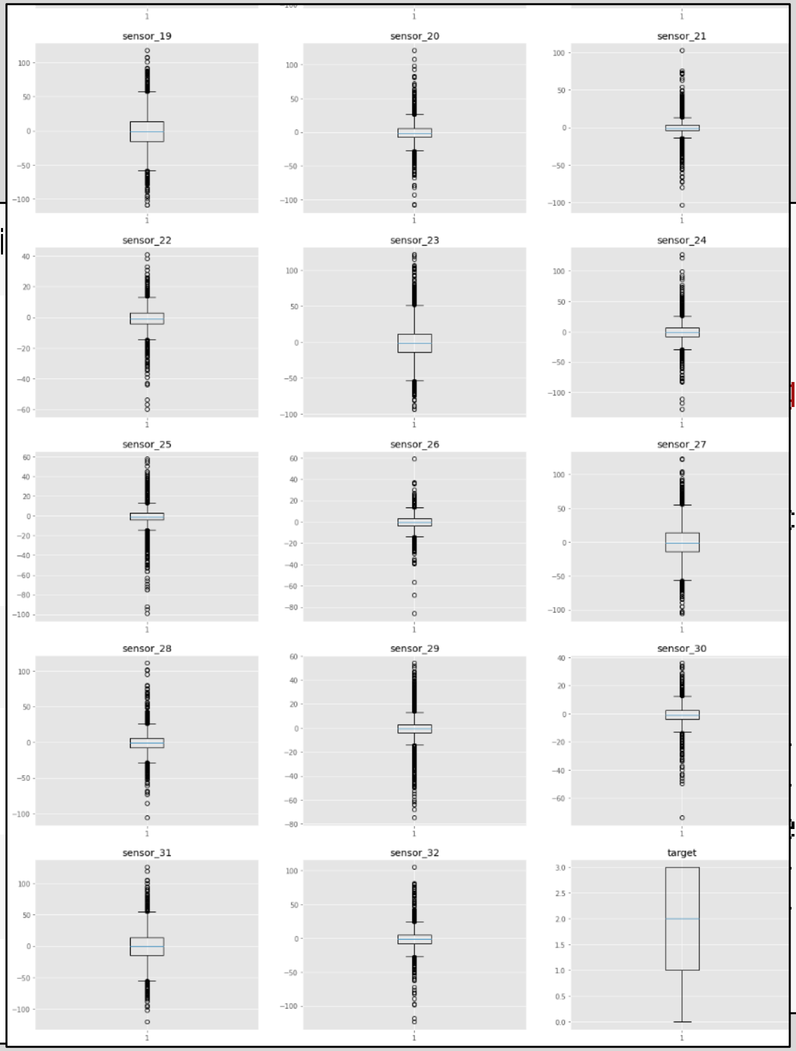
```
sensor_1     0
sensor_2     0
sensor_3     0
sensor_4     0
sensor_5     0
sensor_6     0
sensor_7     0
sensor_8     0
sensor_9     0
sensor_10    0
sensor_11    0
sensor_12    0
sensor_13    0
sensor_14    0
sensor_15    0
sensor_16    0
sensor_17    0
sensor_18    0
sensor_19    0
sensor_20    0
sensor_21    0
sensor_22    0
sensor_23    0
sensor_24    0
sensor_25    0
sensor_26    0
sensor_27    0
sensor_28    0
sensor_29    0
sensor_30    0
sensor_31    0
sensor_32    0
target       0
dtype: int64
```

```
# 기초통계량 확인
data.describe()
```

	sensor_1	sensor_2	sensor_3	sensor_4	sensor_5	sensor_6	sensor_7	sensor_8	sensor_9	sensor_10	...
count	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	2335.000000	...
mean	-1.122174	-1.024673	-0.672769	-0.147724	-0.327494	-0.423462	0.676275	-0.936019	-0.797432	-0.704585	...
std	11.486353	7.399859	26.519159	15.551500	11.461970	7.314322	26.869479	15.598104	12.015022	7.384626	...
min	-94.746969	-63.942094	-122.195138	-111.870691	-94.147972	-70.916786	-105.956553	-102.965354	-81.268085	-47.937561	...
25%	-4.036597	-4.031957	-14.878500	-7.116633	-3.968687	-3.957699	-13.937806	-8.053214	-4.031148	-3.983620	...
50%	-0.951398	-1.015582	-0.961088	-0.890469	-0.871690	-0.804810	0.058910	-1.095551	-0.944613	-0.932964	...
75%	2.895540	2.140456	13.974075	6.110973	2.970387	3.006144	13.934438	4.955494	2.235557	2.883284	...
max	68.876142	39.913391	127.124171	102.015561	89.059852	34.923040	120.046277	125.160611	74.101715	47.030119	...

8 rows × 33 columns



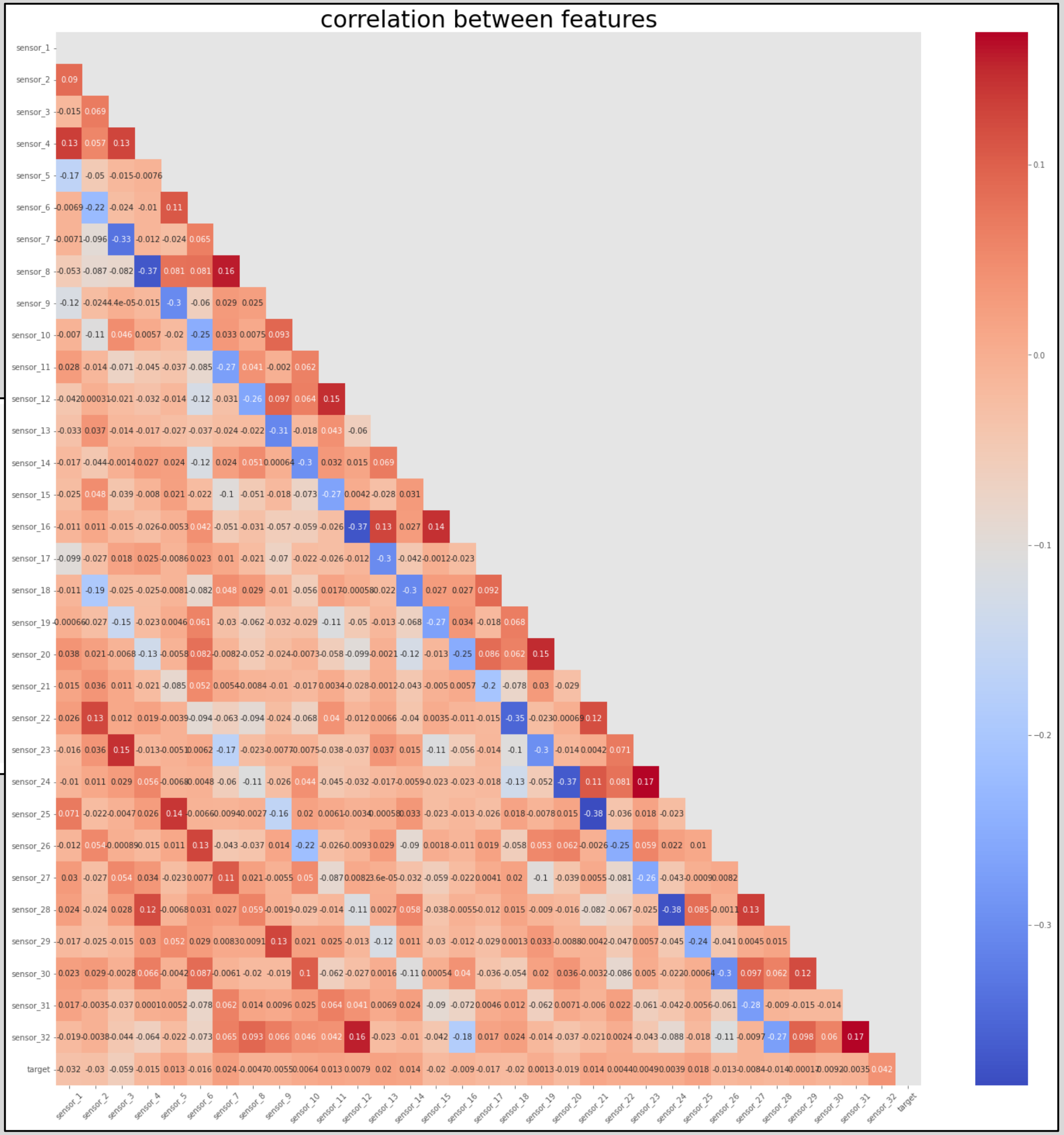



```
b.pyplot as plt
ne
gplot")

olumns
ze=(20,60))

en(feature)):
(11,3,i+1)
ature[i])
(data[feature[i]])
```

2023-05-28

EDA

```
plt.figure(figsize=(25,25))

heat_table = data.corr()
mask = np.zeros_like(heat_table)
mask[np.triu_indices_from(mask)] = True
heatmap_ax = sns.heatmap(heat_table, annot=True, mask = mask, cmap='coolwarm')
heatmap_ax.set_xticklabels(heatmap_ax.get_xticklabels(), fontsize=10, rotation=45)
heatmap_ax.set_yticklabels(heatmap_ax.get_yticklabels(), fontsize=10)
plt.title('correlation between features', fontsize=30)
plt.show()
```



2023-05-28

ENSEMBLE

Ensemble

```
# 모델 훈련을 위해 트레인 셋과 테스트 셋 분리
from sklearn.model_selection import train_te

x_train, x_test, y_train, y_test = train_tes
print(x_train.shape, x_test.shape, y_train.s

(1868, 32) (467, 32) (1868,) (467,)

from sklearn.ensemble import Rando
from sklearn.metrics import accura

rf_clf = RandomForestClassifier(n_
rf_clf.fit(x_train, y_train)

predict = rf_clf.predict(x_test)
print(accuracy_score(y_test,predic

0.7708779443254818
```

```
# 그리드 서치를 통해서 하이퍼 파라미터 탐색
from sklearn.model_selection import GridSearchCV

params = { 'n_estimators' : [100, 120, 140],
           'max_depth' : [16, 20, 24],
           'min_samples_leaf' : [2, 4],
           'min_samples_split' : [2, 4]
          }

= -1)

_samples_split': 4, 'n_estimators': 140}
```

```
# 그리드 서치 결과에 따른 하이퍼 파라미터 입력 후 정확도 측정
rf_clf = RandomForestClassifier(n_estimators=140,
                                max_depth=24,
                                min_samples_leaf=2,
                                min_samples_split=4,
                                random_state=0, n_jobs = -1)
rf_clf.fit(x_train, y_train)

predict = rf_clf.predict(x_test)
print(accuracy_score(y_test,predict))

0.7558886509635975
```

Ensemble

```python
test = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/test.csv')
X_test = test.drop(['id'], axis=1)
```

```python
#데이터를 0~1 값으로 정규화
def get_preprocessed_data(train, test):
    train = np.array(X/255.0, dtype=np.float32)
    test = np.array(X_test/255.0, dtype=np.float32)

    return train, test

X, X_test =  get_preprocessed_data(X, X_test)
```

```python
# - 값이 있어서 MinMaxScalar를 이용해 한번더 정규화
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X = scaler.fit_transform(X)
X_test = scaler.transform(X_test)
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier

from sklearn.model_selection import GridSearchCV

# 모델들을 할당할 리스트를 생성
clfs = []

# estimators 리스트에 모델들을 추가
rf = RandomForestClassifier()
clfs.append(rf)

gbc = GradientBoostingClassifier()
clfs.append(gbc)

etc = ExtraTreesClassifier()
clfs.append(etc)

# 모들의 파라미터들을 할당할 리스트를 생성
params = []

# params 리스트에 성능을 비교하고자하는 파라미터들 추가
params_rf = {'n_estimators' : [90, 100, 110, 120],
            'min_samples_split' : [2,3,4]}
params.append(params_rf)

params_gbc = {'learning_rate':[0.05,0.06,0.07,0.08,0.09,0.1,0.11,0.12,0.13,0.14,0.15],
             'n_estimators':[60,70,80,90,100,110,120,130,140,150]}
params.append(params_gbc)

params_etc = {'n_estimators' : [50,60,70,80,90,100,110,120,130,140,150]}
params.append(params_etc)
```

Ensemble

```python
# 루프 진행정도를 한줄로 출력해서 보기 위함
# GridSearchCV를 이용해 모델 최적화
from tqdm.auto import tqdm

def gridSearchCV(models,par
    best_models=[]
    for i in tqdm(range(0,le
        model_grid = GridSe
        model_grid.fit(X, Y
        best_models.append(
    return best_models

best_model_list = gridSearc
```

```
100%

Fitting 5 folds for each of
Fitting 5 folds for each of
Fitting 5 folds for each of
```

```python
# 최적화된 모델들 확인
best_model_list
```

```
[RandomForestClassifier(n_es
 GradientBoostingClassifier
 ExtraTreesClassifier(n_est imators=140)]
```

```python
# 최적화된 모델들을 사용
best_models = [
    ('rf', RandomForestClassifier(n_estimators=120)),
    ('GBR',GradientBoostingClassifier(learning_rate=0.14, n_estimators=150)),
    ('ET', ExtraTreesClassifier(n_estimators=140))
]

# 앙상블 기법을 위한 패키지를 호출
from sklearn.ensemble import VotingClassifier

# 앙상블 모델을 학습
voting_clf = VotingClassifier(estimators=best_models,voting='soft') # voting 파라미터를 'soft' 로 설정하는 경우 모델의 output의 평균을 이용해 라벨을 예측
voting_clf.fit(X, Y_obj)
```

```
VotingClassifier(estimators=[('rf', RandomForestClassifier(n_estimators=120)),
                             ('GBR',
                              GradientBoostingClassifier(learning_rate=0.14,
                                                         n_estimators=150)),
                             ('ET', ExtraTreesClassifier(n_estimators=140))],
                 voting='soft')
```

```python
pred = voting_clf.predict(X_test)
```

DEEP LEARNING

CNN

Deep Learning

```
dataset = data.values
X = dataset[:, 0:32].astype(float)
Y_obj = dataset[:,32]
```

```
# softmax 활용을 위한 원핫 인코딩
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(Y_obj)
Y_encoded = to_categorical(encoder.transform(Y_obj))

Y_encoded
```

```
array([[0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [1., 0., 0., 0.],
       ...,
       [0., 0., 0., 1.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.]], dtype=float32)
```

Deep Learning

```
#1차 시도
from tensorflow.keras.optimizers impor
from tensorflow.keras.losses import Ca
from tensorflow.keras.metrics import A
from keras.models import Sequential
from keras.layers import Dense, Activa

model = Sequential()
model.add(Dense(16, activation='relu')
model.add(Dense(4, activation='relu'))
model.add(Dense(4, activation='softmax

model.compile(optimizer=Adam(0.001), l
```

```
history = model.fit(X, Y_encoded, epochs=10, batch_size=1)

Epoch 1/10
2335/2335 [==============================] - 4s 1ms/step - loss: 2.2130 - accuracy: 0.2557
Epoch 2/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.3397 - accuracy: 0.2946
Epoch 3/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.3157 - accuracy: 0.3345
Epoch 4/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.2826 - accuracy: 0.3675
Epoch 5/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.2133 - accuracy: 0.3919
Epoch 6/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.1845 - accuracy: 0.4206
Epoch 7/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.1096 - accuracy: 0.4287
Epoch 8/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.0531 - accuracy: 0.4831
Epoch 9/10
2335/2335 [==============================] - 3s 1ms/step - loss: 1.0029 - accuracy: 0.5105
Epoch 10/10
2335/2335 [==============================] - 3s 1ms/step - loss: 0.9506 - accuracy: 0.5400
```

Deep Learning

```
수를 추가하여 모델학습 강화 및 하이퍼 파리미터 조정
crossentropy손실함수 적용
ics=['accuracy'])
```

```
Epoch 1/20
2335/2335 [==============================] - 4s 2ms/step - loss: 1.4346 - accuracy: 0.4561
Epoch 2/20
2335/2335 [=========
Epoch 3/20
2335/2335 [=========
Epoch 4/20
2335/2335 [=========
Epoch 5/20
2335/2335 [=========
Epoch 6/20
2335/2335 [=========
Epoch 7/20
2335/2335 [=========
Epoch 8/20
2335/2335 [=========
Epoch 9/20
2335/2335 [=========
Epoch 10/20
2335/2335 [=========
Epoch 11/20
2335/2335 [=========
```

```
Epoch 12/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.2698 - accuracy: 0.9182
Epoch 13/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.2477 - accuracy: 0.9178
Epoch 14/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.2856 - accuracy: 0.9126
Epoch 15/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.1988 - accuracy: 0.9379
Epoch 16/20
2335/2335 [==============================] - 6s 2ms/step - loss: 0.2384 - accuracy: 0.9366
Epoch 17/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.1803 - accuracy: 0.9460
Epoch 18/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.2353 - accuracy: 0.9302
Epoch 19/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.2119 - accuracy: 0.9383
Epoch 20/20
2335/2335 [==============================] - 4s 2ms/step - loss: 0.2014 - accuracy: 0.9400
```

Deep Learning

```
# 제출 데이터 평가 결과 자체 성능 평가에 비해 분
# 과적합 문제 완화를 위한 Dropout 활용 및 하이퍼
from keras.layers import Dropout

model = Sequential()
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

model.compile(optimizer=Adam(0.001), loss='spars
```

```
history = model.fit(X, Y_obj, epochs=100, batch_size=8)
```

```
Epoch 1/100
292/292 [===========
Epoch 2/100
292/292 [===========
Epoch 3/100
292/292 [===========
Epoch 4/100
292/292 [===========
Epoch 5/100
292/292 [===========
Epoch 6/100
292/292 [===========
Epoch 7/100
292/292 [===========
Epoch 8/100
292/292 [===========
Epoch 9/100
292/292 [===========
Epoch 10/100
292/292 [===========
Epoch 11/100
292/292 [===========
Epoch 12/100
292/292 [===========
Epoch 13/100
292/292 [===========
Epoch 14/100
292/292 [===========
Epoch 15/100
292/292 [===========
```

```
Epoch 84/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7877 - accuracy: 0.6694
Epoch 85/100
292/292 [==============================] - 1s 2ms/step - loss: 0.8119 - accuracy: 0.6608
Epoch 86/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7910 - accuracy: 0.6887
Epoch 87/100
292/292 [==============================] - 1s 2ms/step - loss: 0.8290 - accuracy: 0.6668
Epoch 88/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7858 - accuracy: 0.6749
Epoch 89/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7764 - accuracy: 0.6848
Epoch 90/100
292/292 [==============================] - 1s 2ms/step - loss: 0.8170 - accuracy: 0.6767
Epoch 91/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7690 - accuracy: 0.6938
Epoch 92/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7785 - accuracy: 0.6857
Epoch 93/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7896 - accuracy: 0.6762
Epoch 94/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7862 - accuracy: 0.6934
Epoch 95/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7780 - accuracy: 0.7006
Epoch 96/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7667 - accuracy: 0.6882
Epoch 97/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7666 - accuracy: 0.6904
Epoch 98/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7548 - accuracy: 0.6861
Epoch 99/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7948 - accuracy: 0.6916
Epoch 100/100
292/292 [==============================] - 1s 2ms/step - loss: 0.7515 - accuracy: 0.6985
```

Deep Learning

```
train = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/train.csv")
test = pd.read_csv('/content/gdrive/My Drive/Colab Notebooks/test.csv')
```

```
x_train = train.drop(['id','target'], axis=1)
y_train = train.target
x_test = test.drop(['id'], axis=1)
```

```
def get_preprocessed_data(train, test):
    train = np.array(x_train/255.0, dtype=np.float32)
    test = np.array(x_test/255.0, dtype=np.float32)

    return train, test

x_train, x_test =  get_preprocessed_data(x_train, x_test)
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
x_train = np.array(x_train).reshape(-1, 8, 4, 1)
x_test = np.array(x_test).reshape(-1, 8, 4, 1)
```

Deep Learning

```python
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense , Conv2D , Dropout , Flatten , Activation, MaxPooli
from tensorflow.keras.optimizers import Adam
```

```python
input_tensor = Input(shape=(8, 4, 1))

x = Conv2D(filters=128, kernel_size=(3, 2), padding='same', activation='relu')(input_tensor)
x = Conv2D(filters=128, kernel_size=(3, 2), padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=(2, 1))(x)
```

```python
x = Conv2D(filters=256, kernel_size=(3, 2), padding='same', activation='relu')(x)
x = Conv2D(filters=256, kernel_size=(3, 2), padding='same', activation='relu')(x)
x = MaxPooling2D(pool_size=2)(x)
```

```python
x = Flatten(name='flatten')(x)
x = Dropout(rate=0.5)(x)
x = Dense(300, activation='relu', name='fc1')(x)
x = Dropout(rate=0.3)(x)
output = Dense(10, activation='softmax', name='output')(x)
```

```python
model = Model(inputs=input_tensor, outputs=output)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 8, 4, 1)]         0

 conv2d (Conv2D)             (None, 8, 4, 128)         896

 conv2d_1 (Conv2D)           (None, 8, 4, 128)         98432

 max_pooling2d (MaxPooling2D  (None, 4, 4, 128)        0
 )

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         196864

 conv2d_3 (Conv2D)           (None, 4, 4, 256)         393472

 max_pooling2d_1 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 flatten (Flatten)           (None, 1024)              0

 dropout_3 (Dropout)         (None, 1024)              0

 fc1 (Dense)                 (None, 300)               307500

 dropout_4 (Dropout)         (None, 300)               0

 output (Dense)              (None, 10)                3010

=================================================================
Total params: 1,000,174
Trainable params: 1,000,174
Non-trainable params: 0
_________________________________________________________________
```

Deep Learning

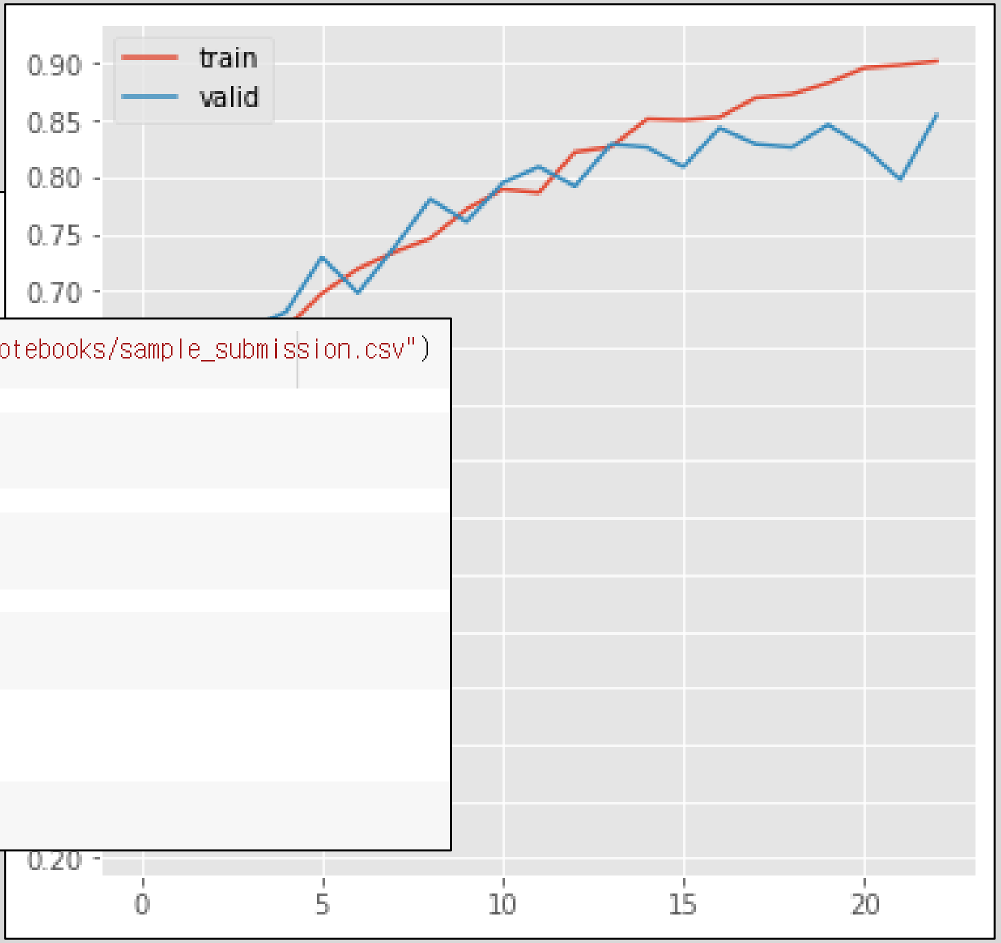
```
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x=x_train, y=y_train, batch_size=40, epochs=23, validation_split=0.15)
```

```
Epoch 11/23
50/50 [==============================] - 6s 125ms/step - loss: 0.5488 - accuracy: 0.7893 - val_loss: 0.5429 - val_accuracy: 0.7949
Epoch 12/23
50/50 [==============================] - 6s 124ms/step - loss: 0.5413 - accuracy: 0.7863 - val_loss: 0.5266 - val_accuracy: 0.8091
Epoch 13/23
50/50 [==============================] - 6s 124ms/step - loss: 0.4798 - accuracy: 0.8221 - val_loss: 0.5689 - val_accuracy: 0.7920
Epoch 14/23
50/50 [==============================] - 6s 124ms/step - loss: 0.4444 - accuracy: 0.8266 - val_loss: 0.4844 - val_accuracy: 0.8291
Epoch 15/23
50/50 [==============================] - 7s 124ms/step - loss: 0.3956 - accuracy: 0.8513 - val_loss: 0.4917 - val_accuracy: 0.8262
Epoch 16/23
50/50 [==============================] - 6s 124ms/step - loss: 0.3874 - accuracy: 0.8503 - val_loss: 0.5227 - val_accuracy: 0.8091
Epoch 17/23
50/50 [==============================] - 6s 123ms/step - loss: 0.3850 - accuracy: 0.8528 - val_loss: 0.4637 - val_accuracy: 0.8433
Epoch 18/23
50/50 [==============================] - 6s 123ms/step - loss: 0.3531 - accuracy: 0.8700 - val_loss: 0.4599 - val_accuracy: 0.8291
Epoch 19/23
50/50 [==============================] - 6s 124ms/step - loss: 0.3273 - accuracy: 0.8730 - val_loss: 0.5020 - val_accuracy: 0.8262
Epoch 20/23
50/50 [==============================] - 6s 125ms/step - loss: 0.3077 - accuracy: 0.8831 - val_loss: 0.4396 - val_accuracy: 0.8462
Epoch 21/23
50/50 [==============================] - 6s 124ms/step - loss: 0.2966 - accuracy: 0.8962 - val_loss: 0.5329 - val_accuracy: 0.8262
Epoch 22/23
50/50 [==============================] - 6s 124ms/step - loss: 0.2739 - accuracy: 0.8987 - val_loss: 0.5494 - val_accuracy: 0.7977
Epoch 23/23
50/50 [==============================] - 6s 125ms/step - loss: 0.2531 - accuracy: 0.9022 - val_loss: 0.4818 - val_accuracy: 0.8547
```

Deep Learning

```
import matplotlib.pyplot as plt

def show_history(
  plt.figure(figs
  plt.yticks(np.a
  plt.plot(histor
  plt.plot(histor
  plt.legend()

show_history(hist
```

```
submission = pd.read_csv("/content/gdrive/My Drive/Colab Notebooks/sample_submission.csv")

pred = model.predict(x_test)

predicted = pred.argmax(axis=-1)

predicted
array([0, 0, 1, ..., 0, 0, 3])

submission['target'] = predicted
```



감사합니다