위성 관측 데이터 활용 강수량 산출 AI 경진대회

서주원

0. 대회 및 데이터 설명

본 대회는 밝기 온도, 지표 타입, GMI 경도, GMI 위도, DPR 경도, DPR 위도 등의 데이터를 가지고 강수량을 예측하는 대회입니다.

1. 준비

In [1]:
```
import numpy as np
import pandas as pd
import os

import matplotlib.pyplot as plt

import tensorflow as tf

from keras import regularizers
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Add, BatchNormalization, concatenate,Convolution2D, Conv2D
from mlxtend.classifier import EnsembleVoteClassifier
from keras.callbacks import EarlyStopping, LambdaCallback
from keras.applications import ResNet50

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
from sklearn import utils
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.model_selection import KFold, StratifiedKFold
```

```
Using TensorFlow backend.
```

In [2]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only allocate 1GB of memory on the first GPU
  try:
    tf.config.experimental.set_virtual_device_configuration(
        gpus[0],
        [tf.config.experimental.VirtualDeviceConfiguration(memory_limit=1024)])
    logical_gpus = tf.config.experimental.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    # Virtual devices must be set before GPUs have been initialized
    print(e)
```

```
1 Physical GPUs, 1 Logical GPUs
```

- 학습을 몇 번 시도해 봤는데, CPU만을 이용해 학습하니 train set의 데이터가 너무나도 방대해서 epoch당 학습 속도가 너무 좋지 않았습니다. 그래서 tensorflow-gpu를 설치해 다음과 같은 코드를 실행시켜서 GPU로 model을 학습할 수 있게 하는 환경을 구축했습니다.

- Google colab에서는 ram 용량의 제한이 있어서 개인 pc의 anaconda, jupyter notebook을 이용했습니다.

2. 데이터 로드

```
In [3]: submission = pd.read_csv('sample_submission.csv')
        train_files = os.listdir('C:/Users/seozo/OneDrive/바탕 화면/dacon/train')
```

```
In [4]: train = []
        for file in train_files:
            try:
                data = np.load('C:/Users/seozo/OneDrive/바탕 화면/dacon/train/'+file).astype('float32')
                train.append(data)
            except:
                continue
```

```
In [5]: test = []
        for sub_id in submission['id']:
            data = np.load('C:/Users/seozo/OneDrive/바탕 화면/dacon/test/'+'subset_'+sub_id+'.npy').astype('float32')
            test.append(data)
```

- train과 test data들을 로드 했습니다. data들은 40X40 형태의 이미지 파일로 되어있습니다.

```
In [3]: train_df
```

Out[3]:

	temp1	temp2	temp3	temp4	temp5	temp6	temp7	temp8	temp9	type	long_GMI	lat_GMI	long_
0	168.745071	90.459198	193.079697	125.343018	226.008530	215.537094	152.781219	267.132385	237.999283	0	160.967621	40.126064	161.07
1	166.973694	89.801224	193.049561	125.630112	225.944229	215.421158	152.503128	266.770691	238.626785	0	161.015671	40.088432	161.07
2	166.841629	90.226326	192.859787	125.389030	225.677124	216.231277	152.096664	267.000214	238.367294	0	161.064255	40.051228	161.07
3	167.142822	89.574203	193.425949	125.622368	226.249557	215.538849	152.664124	266.966583	237.810135	0	161.113358	40.014450	161.09
4	167.638077	90.122505	192.978592	125.161728	226.883911	216.013626	152.471634	267.064331	237.794601	0	161.162994	39.978111	161.15
...	...	...	...	...	...	...	...	...	...	...	...	...	...
122151995	272.875580	259.872589	281.911682	274.448639	281.595612	282.253876	276.794739	283.406921	281.640350	103	116.462120	29.391375	116.47
122151996	269.418030	253.234665	279.787231	269.509064	280.255035	281.061279	274.059418	282.955505	279.595459	103	116.416550	29.357338	116.42
122151997	265.559967	245.997650	277.532288	266.938690	279.487610	279.573761	272.847809	283.366486	281.000122	103	116.371468	29.322809	116.37
122151998	261.475861	238.112015	273.281372	260.541077	277.068237	276.714783	268.621735	284.484619	281.224976	103	116.326881	29.287781	116.35
122151999	255.189041	227.414917	268.256439	249.050415	272.407532	272.012543	256.546356	280.792480	274.030426	103	116.282814	29.252268	116.35

122152000 rows × 15 columns

```
In [4]: test_df
```

Out[4]:

	temp1	temp2	temp3	temp4	temp5	temp6	temp7	temp8	temp9	type	long_GMI	lat_GMI	long_DP
0	166.479080	89.160690	187.936172	118.861412	214.203506	214.023972	151.760315	263.783478	233.330048	0	131.104523	40.599606	131.10360
1	166.881927	89.890549	189.061615	119.168121	214.135696	212.979538	150.284988	262.505219	231.299652	0	131.169418	40.618748	131.15570
2	167.336197	89.116234	187.739044	119.040283	214.371033	213.933899	151.608582	263.448883	234.285645	0	131.234039	40.638447	131.20796
3	166.961426	90.114693	188.004745	119.050346	213.354935	213.654694	152.032822	262.841278	232.018250	0	131.298370	40.658699	131.31156
4	166.883255	89.838066	188.203415	119.100990	213.265976	213.429672	152.934494	263.799194	234.420868	0	131.362427	40.679501	131.36265
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3865595	173.905777	94.159363	207.131393	145.359146	248.295395	227.527008	171.421814	278.680511	265.196350	0	164.784195	6.145889	164.78439
3865596	174.443970	93.493919	207.164124	146.399994	248.490982	227.951874	171.581360	279.339722	264.569305	0	164.826538	6.176169	164.82867
3865597	174.390640	94.267021	207.287811	144.315857	247.543961	227.250931	171.539978	279.065399	264.583832	0	164.868530	6.206952	164.85662
3865598	174.346512	93.727180	207.199600	145.618774	247.670929	227.669647	171.560944	279.511566	264.655304	0	164.910156	6.238243	164.85662
3865599	173.615891	93.304138	207.625793	145.431381	247.532852	228.415771	172.077972	280.122284	266.755066	0	164.951401	6.270027	164.85662

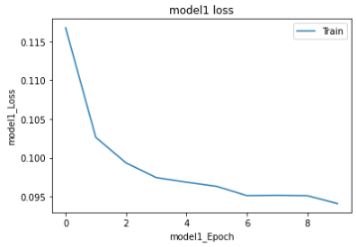
3865600 rows × 14 columns

- ftr 파일로 읽어서 data frame 형식으로 받아 올 수 있었지만 이상하게 제 환경에서는 이렇게 데이터를 받아오면 조그마한 작업을 수행해도 컴퓨터가 멈춰버려서 위의 방법으로 데이터를 받아오게 되었습니다. 아마 개인 pc에서도 ram부족으로 인해서 이런 문제가 발생한 것 같습니다.

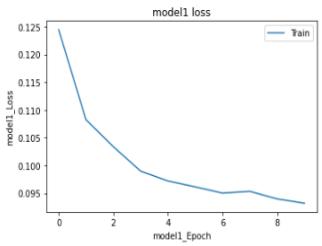
3. 데이터 전처리

```
In [6]: train = np.array(train)
        test = np.array(test)

In [7]: x_train = train[:,:,:,:14]
        y_train = train[:,:,:,14]
        test = test[:,:,:,:14]

In [8]: del train

In [9]: x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.025, random_state=7777)
        x_train.shape, y_train.shape, x_test.shape, y_test.shape
Out[9]: ((74436, 40, 40, 14), (74436, 40, 40), (1909, 40, 40, 14), (1909, 40, 40))
```

```
In [14]: pred = []
         pred = model1.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model1.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('f14_test1.csv', index = False)
tf.Tensor(3.21039, shape=(), dtype=float32)

In [15]: # 트레이닝 Epoch에 따라 Loss의 변화를 그래프로 시각화하는 코드입니다.
         plt.plot(model1_history.history['loss'])
         plt.title('model1 loss')
         plt.ylabel('model1_Loss')
         plt.xlabel('model1_Epoch')
         plt.legend(['Train'],loc='upper right')
         plt.show()
```




```
In [13]: pred = []
         pred = model1.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model1.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('f14_test1.csv', index = False)
tf.Tensor(2.161281, shape=(), dtype=float32)

In [14]: # 트레이닝 Epoch에 따라 Loss의 변화를 그래프로 시각화하는 코드입니다.
         plt.plot(model1_history.history['loss'])
         plt.title('model1 loss')
         plt.ylabel('model1_Loss')
         plt.xlabel('model1_Epoch')
         plt.legend(['Train'],loc='upper right')
         plt.show()
```



- 저는 데이터의 14개의 채널을 모두 이용했습니다. 학습에 필요한 시간이 너무 많이 소모되어서 epoch를 10이라는 작은 수치로 두었지만 제가 만든 모델에 대해서 10개의 채널을 이용한 왼쪽의 정확도 3.210에 비해서 14개의 채널을 이용한 오른쪽의 정확도 2.16128이 좀 더 정확도 높았기 때문입니다.

```
In [10]: y_train_ = y_train.reshape(-1,y_train.shape[1]*y_train.shape[2])

         x_train = np.delete(x_train, np.where(y_train_<0)[0], axis=0)
         y_train = np.delete(y_train, np.where(y_train_<0)[0], axis=0)
         y_train = y_train.reshape(-1, x_train.shape[1], x_train.shape[2],1)
         y_test = y_test.reshape(-1, y_test.shape[1], y_test.shape[2],1)

         y_train_ = np.delete(y_train_, np.where(y_train_<0)[0], axis=0)

         x_train.shape, y_train.shape
Out[10]: ((74061, 40, 40, 14), (74061, 40, 40, 1))
```

- 0보다 작은 데이터 값이 포함된 데이터를 삭제했습니다. Outlier에 대한 조치입니다.

4. loss 함수 정의

In [11]:
```
from sklearn.metrics import f1_score

def mae(y_true, y_pred) :

    y_true, y_pred = np.array(y_true), np.array(y_pred)

    y_true = y_true.reshape(1, -1)[0]

    y_pred = y_pred.reshape(1, -1)[0]

    over_threshold = y_true >= 0.1

    return np.mean(np.abs(y_true[over_threshold] - y_pred[over_threshold]))

def fscore(y_true, y_pred):

    y_true, y_pred = np.array(y_true), np.array(y_pred)

    y_true = y_true.reshape(1, -1)[0]

    y_pred = y_pred.reshape(1, -1)[0]

    remove_NAs = y_true >= 0

    y_true = np.where(y_true[remove_NAs] >= 0.1, 1, 0)

    y_pred = np.where(y_pred[remove_NAs] >= 0.1, 1, 0)

    return(f1_score(y_true, y_pred))

def maeOverFscore(y_true, y_pred):

    return mae(y_true, y_pred) / (fscore(y_true, y_pred) + 1e-07)

def fscore_keras(y_true, y_pred):
    score = tf.py_function(func=fscore, inp=[y_true, y_pred], Tout=tf.float32, name='fscore_keras')
    return score

def maeOverFscore_keras(y_true, y_pred):
    score = tf.py_function(func=maeOverFscore, inp=[y_true, y_pred], Tout=tf.float32,  name='custom_mse')
    return score
```

- loss 함수는 baseline에 구축되어 있는 것을 그대로 사용했습니다.

5. 모델 구축

1) 모델 1

```
def create_model1():
    model = Sequential()
    model.add(Convolution2D(32,kernel_size=(3,3),padding='same', activation='relu',input_shape=(40,40,14)))
    model.add(BatchNormalization())
    model.add(Convolution2D(32,kernel_size=(3,3),padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Convolution2D(64,kernel_size=(3,3),padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Convolution2D(16,kernel_size=(3,3),padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Convolution2D(8,kernel_size=(3,3),padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Convolution2D(2,kernel_size=(3,3),padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(Convolution2D(1,kernel_size=(3,3),padding='same', activation='relu'))

    model.compile(loss="mae", optimizer="adam", metrics=[maeOverFscore_keras, fscore_keras])

    return model

early_stopping = EarlyStopping(patience=5, min_delta=0, monitor='val_loss')
model1 = create_model1()
model1.compile(loss="mae", optimizer="adam", metrics=[maeOverFscore_keras, fscore_keras])
model1_history = model1.fit(x_train, y_train, batch_size=50,epochs = 10, verbose=1)
```

- Baseline과 토론 글 등을 참고해서 만든 모델입니다. Baseline에서는 kernel size를 모두 3으로 주었고 토론 글에서 차원 축소를 하지 않는 모델의 정확도가 좀 더 올라간다고 하여서 차원 축소를 거치지 않는 기본 모델을 제작했습니다.

2) 모델 2 – Resnet 변형

```
#Resnet
    def create_model3():
        inputs=Input(x_train.shape[1:])

        bn=BatchNormalization()(inputs)
        x=Conv2D(256, kernel_size=3, strides=1, padding='same', activation='relu')(bn)

        bn=BatchNormalization()(x)
        x2=Conv2D(128, kernel_size=3, strides=1, padding='same', activation='relu')(bn)
        concat=concatenate([x, x2], axis=3)

        bn=BatchNormalization()(concat)
        x2=Conv2D(64, kernel_size=3, strides=1, padding='same', activation='relu')(bn)
        concat=concatenate([concat, x2], axis=3)

        bn=BatchNormalization()(concat)
        x2=Conv2D(32, kernel_size=3, strides=1, padding='same', activation='relu')(bn)
        concat=concatenate([concat, x2], axis=3)
        bn=BatchNormalization()(concat)
        x2=Conv2D(32, kernel_size=3, strides=1, padding='same', activation='relu')(bn)
        concat=concatenate([concat, x2], axis=3)
        bn=BatchNormalization()(concat)
        x2=Conv2D(32, kernel_size=3, strides=1, padding='same', activation='relu')(bn)
        concat=concatenate([concat, x2], axis=3)

        bn=BatchNormalization()(concat)
        x=Conv2D(32, kernel_size=3, strides=1, padding='same', activation='relu')(bn)
        concat=concatenate([concat, x], axis=3)
        bn=BatchNormalization()(concat)
        x=Conv2D(32, kernel_size=3, strides=1, padding='same', activation='relu')(bn)
        concat=concatenate([concat, x], axis=3)

        bn=BatchNormalization()(concat)
        outputs=Conv2D(1, kernel_size=3, strides=1, padding='same', activation='relu')(bn)

        model=Model(inputs=inputs, outputs=outputs)

        return model

    early_stopping = EarlyStopping(patience=5, min_delta=0, monitor='val_loss')
    model3 = create_model3()
    model3.compile(loss="mae", optimizer="adam", metrics=[maeOverFscore_keras, fscore_keras])
    model3_history = model3.fit(x_train, y_train, batch_size=10, epochs = 3, verbose=1)
```

- CNN 모델 중 하나인 Resnet을 변형한 모델입니다.

3) 모델 3 – VGG 변형

```
#vgg16 변형
def create_model4():
    inputs=Input(x_train.shape[1:])
    bn=BatchNormalization()(inputs)

    x = Conv2D(64, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=r
    x = Conv2D(64, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=r

    x = Conv2D(128, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=
    x = Conv2D(128, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=

    x = Conv2D(256, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=
    x = Conv2D(256, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=
    x = Conv2D(256, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=

    x = Conv2D(512, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=
    x = Conv2D(512, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=
    x = Conv2D(512, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=

    x = Conv2D(512, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=
    x = Conv2D(512, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=
    x = Conv2D(512, (3, 3), activation='relu', strides=1, padding='same', kernel_initializer='he_normal', kernel_regularizer=

    bn=BatchNormalization()(x)

    outputs=Conv2D(1, (1,1), strides=1, padding='same', activation='relu')(bn)

    model=Model(inputs=inputs, outputs=outputs)

    return model

early_stopping = EarlyStopping(patience=5, min_delta=0, monitor='val_loss')
model4 = create_model4()
model4.compile(loss="mae", optimizer="adam", metrics=[maeOverFscore_keras, fscore_keras])
model4_history = model4.fit(x_train, y_train, batch_size=10, epochs = 3, verbose=1)
```

- CNN 모델 중 하나인 VGG16을 변형한 모델입니다.

4) 모델 4 – Alexnet 변형

```
#alexnet 변형
def create_model5():

    inputs=Input(x_train.shape[1:])
    bn=BatchNormalization()(inputs)

    x=Conv2D(96, (11,11), strides=1, padding='same', activation='relu')(bn)
    x=Conv2D(256, (5,5), strides=1,padding='same', activation='relu')(x)
    bn=BatchNormalization()(x)

    x=Conv2D(384, (3,3), strides=1, padding='same', activation='relu')(bn)
    bn=BatchNormalization()(x)


    x=Conv2D(384, (3,3), strides=1, padding='same', activation='relu')(bn)
    x=Conv2D(384, (3,3), strides=1, padding='same', activation='relu')(x)

    outputs=Conv2D(1, (1,1), strides=1, padding='same', activation='relu')(x)

    model=Model(inputs=inputs, outputs=outputs)

    return model

early_stopping = EarlyStopping(patience=5, min_delta=0, monitor='val_loss')
model5 = create_model5()
model5.compile(loss="mae", optimizer="adam", metrics=[maeOverFscore_keras, fscore_keras])
model5_history = model5.fit(x_train, y_train, batch_size=10, epochs = 3, verbose=1)
```

- CNN 모델 중 하나인 VGG16을 변형한 모델입니다.

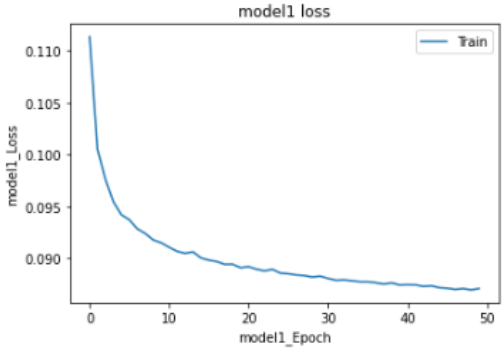
- 각 모델에는 EarlyStopping을 적용시켜서 loss값이 0보다 더 좋아지는 경우가 없는 것이 5번 진행되면 그만 수행하도록 설정했습니다.

6. 성능 평가

1) model1

```
In [14]: pred = []
         pred = model1.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model1.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('dacon_test4.csv', index = False)

         tf.Tensor(1.7414199, shape=(), dtype=float32)
```

```
In [21]: # 트레이닝 Epoch에 따라 Loss의 변화를 그래프로 시각화하는 코드입니다.
         plt.plot(model1_history.history['loss'])
         plt.title('model1 loss')
         plt.ylabel('model1_Loss')
         plt.xlabel('model1_Epoch')
         plt.legend(['Train'],loc='upper right')
         plt.show()
```

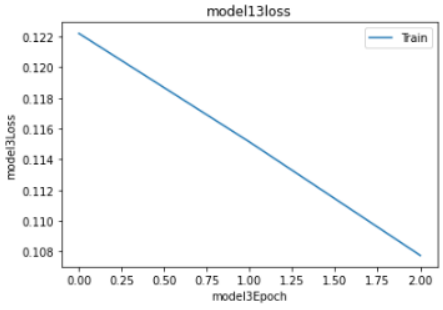


- Model1을 epoch을 50으로 두고 batch를 64로 주었을 때의 결과입니다. Test에 대해서 1.7414점이 나왔지만, 제출 결과는 1.90148점이었습니다.

2) model2 – Resnet 변형

```
In [16]: pred = []
         pred = model3.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model3.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('f14_test2.csv', index = False)

         tf.Tensor(2.3090997, shape=(), dtype=float32)
```

```
In [17]: # 트레이닝 Epoch에 따라 Loss의 변화를 그래프로 시각화하는 코드
         plt.plot(model3_history.history['loss'])
         plt.title('model13loss')
         plt.ylabel('model3Loss')
         plt.xlabel('model3Epoch')
         plt.legend(['Train'],loc='upper right')
         plt.show()
```



- Resnet을 변형한 모델입니다. Batch를 10으로 주어야만 작동했고 그에 따라서 epoch당 학습 속도가 너무 많이 소모되어 epoch를 3으로 주고 실험했습니다. Test에 대한 결과는 2.3090입니다.

3) model3 – VGG 변형

```
In [16]: pred = []
         pred = model4.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model4.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('f14_test3.csv', index = False)

         tf.Tensor(20960372.0, shape=(), dtype=float32)
```

- VGG를 변형한 모델입니다. 마찬가지로 batch를 10으로 주어야만 작동합니다. Resnet에 비해서 epoch당 학습하는데 걸리는 시간이 3배정도 많이 들었습니다. GPU 메모리 문제 탓인지 epoch가 7이 넘어가면 메모리 부족으로 학습이 중단되었습니다. 1번 학습 했을 때, 209600372로 상당히 좋지 않은 점수를 보여줍니다. 학습을 1번만 해서 정확하지 않은 점도 있으나 타 모델들이 1번 학습했을 때의 점수가 기본 5~100점대인 것을 감안하면 좋은 성능은 아닌 것 같습니다.

4) model4 – Alexnet 변형

```
In [22]: pred = []
         pred = model5.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model5.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('f14_test4.csv', index = False)

         tf.Tensor(20960372.0, shape=(), dtype=float32)
```

- Alexnet을 변형한 모델입니다. 마찬가지로 batch를 10으로 주어야만 작동합니다. Resnet에 비해서 epoch당 학습하는데 걸리는 시간이 1.5배 정도 많이 들었습니다. test 점수가 209600372로 상당히 좋지 않은 점수를 보여줍니다. 학습을 1번만 해서 정확하지 않은 점도 있으나 타 모델들이 1번 학습했을 때의 점수가 기본 5~100점대인 것을 감안하면 좋은 성능은 아닌 것 같습니다.

- VGG나 Alexnet을 변형한 모델들은 데이터가 제대로 들어가지 않는 다던가, 차원 문제에 대해서 한 번 확인해 봐야할 것 같습니다.

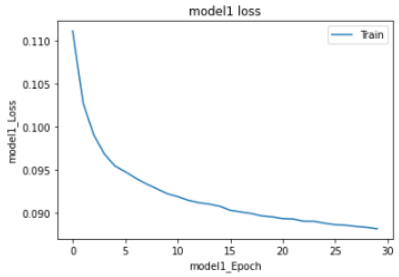
5) 모델 선택

성능이 가장 좋다고 판단되는 2개의 모델1과 모델2에 대해서 epoch를 크게 주어서 실험했습니다.

모델 1같은 경우에는 epoch를 기존 50에서 100으로 늘려서 학습한 후 제출했을 때, 오히려 점수가 2.066점으로 악화되었습니다.

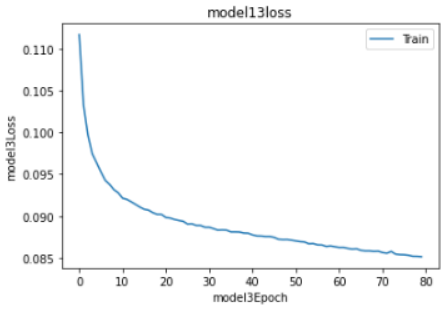
모델 2(Resnet) 같은 경우 epoch를 30 그리고 80까지 늘려서 학습한 결과

```
In [13]: pred = []
         pred = model3.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model3.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('dacon_test5.csv', index = False)

         tf.Tensor(1.6354055, shape=(), dtype=float32)

In [14]: # 트레이닝 Epoch에 따라 Loss의 변화를 그래프로 시각화하는 코드입니다.
         plt.plot(model3_history.history['loss'])
         plt.title('model13loss')
         plt.ylabel('model3Loss')
         plt.xlabel('model3Epoch')
         plt.legend(['Train'],loc='upper right')
         plt.show()
```



30만큼 늘렸을 때는 test에서 1.63 제출시, 1.66으로 제일 성능이 좋았습니다.

```
In [13]: pred = []
         pred = model3.predict(x_test)
         print(maeOverFscore_keras(y_test, pred))

         pred = model3.predict(test)
         submission.iloc[:,1:] = pred.reshape(-1, 1600)
         submission.to_csv('f14_test2.csv', index = False)

         tf.Tensor(1.5797597, shape=(), dtype=float32)

In [14]: # 트레이닝 Epoch에 따라 Loss의 변화를 그래프로 시각화하는 코드입니다.
         plt.plot(model3_history.history['loss'])
         plt.title('model13loss')
         plt.ylabel('model3Loss')
         plt.xlabel('model3Epoch')
         plt.legend(['Train'],loc='upper right')
         plt.show()
```



80만큼 늘렸을 때는 test에서 1.57로 제일 좋았지만, 막상 제출해 보니 1.67점을 받았습니다.

7. 의견

- 컴퓨터의 램 문제 때문에 ftr 파일의 data frame형태로 받아와서 작업하지 못했던 것이 아쉽습니다.

- 이번 대회는 여러 CNN 모델들에 대해서 알아보고 직접 모델도 만들어 본 좋은 기회였습니다.

- Sklearn의 K-Fold나 xg-boost, AdaboostClassifier 등을 이용해서 여러 모델에 대해서 stacking ensemble을 구현하고 싶었는데, 이것을 완성하지 못해 아쉽습니다.

- GPU로 model을 학습시키는 방법을 좀 더 빨리 알았으면 hyper parameter에 대한 정확도에 대해서 여러 모델로 더 많이 비교 분석 해볼 수 있었을 텐데, 이 점이 상당히 아쉽습니다.

- GPU로 model을 학습시켜도 어떤 모델에 대해서는 batch를 크게 주어도 잘 동작하는 반면, 어떤 모델에서는 batch를 작게 주어야만 작동했습니다. Batch 작게 주어야만 모델을 학습시킬 수 있었기 때문에 여러 model에 대해 epoch를 크게 주어 학습시키지 못했던 것이 아쉽습니다.