# 데이터마이닝 과제

컴퓨터공학과

12161558 김혜윤

제출일 : 05/25

```
import glob
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
import tensorflow as tf
import os
import random
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Conv2DTranspose,
MaxPooling2D, BatchNormalization, Activation, concatenate, Input, GlobalAveragePooling2D
from tensorflow.keras import Model
```

코드 내에서 사용할 package

```
np.random.seed(7)
random.seed(7)
tf.random.set_seed(7)
```

재생산성을 위해서 시드를 고정해준다.

```
def trainGenerator():

    train_path = 'train'
    train_files = sorted(glob.glob(train_path + '/*'))

    for file in train_files:

        dataset = np.load(file)

        target= dataset[:,:,-1].reshape(40,40,1)
        cutoff_labels = np.where(target < 0, 0, target)
        feature = dataset[:,:,:9]

        if (cutoff_labels > 0).sum() < 50:

            continue

        yield (feature, cutoff_labels)

train_dataset = tf.data.Dataset.from_generator(trainGenerator,
                                               (tf.float32, tf.float32), (tf.TensorShape([40,40,9]),tf.TensorShape([40,40,1])))
```

trainGenerator 함수, train 파일 속 파일들을 가져와 합쳐 dataset으로 저장하고, 0~8열 만 사용하면서 강수량이 기록되어 있는 픽셀이 50개 이상인 이미지만 사용한다. yield는 generator로 함수 실행 중간에 빠져나올 수 있는 generator를 만들 때 사용한다. 단순히 값을 내보낼 수 있을 뿐 더러, 넣어 줄 수 도 있다.

```
train_dataset = train_dataset.batch(512).prefetch(1)
```

Batch() 메서드는 한 번에 가져올 데이터의 수를 정한다. Prefetch()는 앞으로 연산에 필요한 data들을 미리 가져오는 것이다. 미리 가져오는 것은 memory latency를 감출 수 있다.

```
test_path = 'test'
test_files = sorted(glob.glob(test_path + '/*'))

X_test = []

for file in tqdm(test_files, desc = 'test'):

    data = np.load(file)

    X_test.append(data[:,:,:9])

X_test = np.array(X_test)
```

: 이번에는 test파일 속 파일들을 가져온다. tqdm은 for문의 상태바를 나타내준다. test파일들을 합쳐서 X_test로 저장한다.

```
def build_model(input_layer, start_neurons):

    # 40 x 40 -> 20 x 20
    conv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(input_layer)
    conv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(conv1)
    pool1 = BatchNormalization()(conv1)
    pool1 = MaxPooling2D((2, 2))(pool1)
    pool1 = Dropout(0.25)(pool1)

    # 20 x 20 -> 10 x 10
    conv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(pool1)
    conv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(conv2)
    pool2 = BatchNormalization()(conv2)
    pool2 = MaxPooling2D((2, 2))(pool2)
    pool2 = Dropout(0.25)(pool2)

    # 10 x 10
    convm = Conv2D(start_neurons * 4, (3, 3), activation="relu", padding="same")(pool2)

    # 10 x 10 -> 20 x 20
    deconv2 = Conv2DTranspose(start_neurons * 2, (3, 3), strides=(2, 2), padding="same")(convm)
    uconv2 = concatenate([deconv2, conv2])
    uconv2 = Dropout(0.25)(uconv2)
    uconv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(uconv2)
    uconv2 = Conv2D(start_neurons * 2, (3, 3), activation="relu", padding="same")(uconv2)
    uconv2 = BatchNormalization()(uconv2)

    # 20 x 20 -> 40 x 40
    deconv1 = Conv2DTranspose(start_neurons * 1, (3, 3), strides=(2, 2), padding="same")(uconv2)
    uconv1 = concatenate([deconv1, conv1])
    uconv1 = Dropout(0.25)(uconv1)
    uconv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(uconv1)
    uconv1 = Conv2D(start_neurons * 1, (3, 3), activation="relu", padding="same")(uconv1)
    uconv1 = BatchNormalization()(uconv1)
    uconv1 = Dropout(0.25)(uconv1)
    output_layer = Conv2D(1, (1,1), padding="same", activation='relu')(uconv1)
```

```
    return output_layer

input_layer = Input((40, 40, 9))
output_layer = build_model(input_layer, 32)
```

모델을 구축하는 함수다. Conv2D() 는 Convolution 레이어를 만드는데 사용되는데 이 layer는 영상 인식에 주로 사용되며, 필터가 탑재되어 있다. Start_neurons은 convolution 필터의 개수, (,)은 convolution 커널의 (행,열)이다. Activation='relu'는 활성화 함수를 설정하는데 여기서 사용한 relu는 rectifier 함수로 은닉층에 주로 사용된다. 그외에도 디폴트 값인 linear, 이진 분류문제에서 출력층에 주로 사용되는 sigmoid와 다중 클래스 분류 문제에서 출력층에 사용되는 softmax가 있다. padding="same"은 경계 처리 방법을 정의하는데, 여기 사용된 same은 출력 이미지 사이즈가 입력 이미지 사이즈와 동일하도록 설정하는 것이고, valid는 유효한 영역만 출력이 된다. 따라서 출력 이미지 사이즈는 입력 사이즈보다 작다. Conv1은 input_layer에 적용한 후 다시 conv1에 적용한다. 입력층을 지나서 만나게 되는 레이어들의 입력은 normalization이 쉽지않은데 이를 위해 batchnormalization을 사용한다. regularization을 해준다고도 볼 수 있고, 학습속도 개선, 가중치 초기값 선택의 의존성이 낮아지고, 과적합의 위험성을 낮춰준다. 활성화 함수의 활성화값 또는 출력값을 정규화해준다. MaxPoolin2D는 convolution layer의 출력 이미지에서 주요값만 뽑아 크기가 작은 출력 영상을 만드는 역할을 한다. 이것은 지역적인 사소한 변화가 영향을 미치지 않도록 한다. 여기선 MaxPooling2D((2,2))(poo1)은 pool1을 수직, 수평 축소비율이 (2,2)로 출력 영상 크기를 입력 영상 크기의 반으로 줄어든다. Dropout은 over-fitting을 줄이기 위한 regularization 기술이다. Conv2DTranspose()는 conv2d를 역으로 하는데 역행렬이 아니라 전치 합성곱이라 초기값이 그대로 나오지 않는다. 이렇게 레이어를 쌓아 model을 만들어준다.

```
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
```

규칙에 정의된 함수로 y_true : submission.csv 형태의 실제값, y_pred : 예측값이다.

손실 함수 정의로 베이스 라인에 정의된 함수이다.

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

Compile()은 모델을 기계가 이해할 수 있도록 컴파일 한다. 오차 함수와 최적화 방법, 메트릭 함수를 선택할 수 있다. Optimizer은 훈련 과정을 설정하는 옵티마이저를 설정한다. adam이나 sgd같이 문자열로 지정할 수 도 있다. Loss는 훈련 과정에서 사용할 손실 함수를 설정하고 metrics은 훈련을 모니터링 하기 위한 지표를 선택한다.

```
model_history = model.fit(train_dataset, epochs = 2, verbose=1)
```

Fit()은 모델을 학습시킨다. 모델이 오차로부터 매개 변수를 업데이트 시키는 과정을 학습, 훈련 또는 적합이라고 하는데 , 모델이 데이터에 적합해가는 과정이기 때문이다. 그런 의미에서 fit()은 모델의 훈련을 시작한다라는 의미를 가진다. Train_dataset은 훈련 데이터, epohs = 2는 총 훈련 횟수를 정의한다. verbose는 학습 중 출력되는 문구를 설정하는데 0은 출력 없음, 1: 진행도를 보여주는 진행 막대를 보여준다, 2: 미니 배치마다 손실정보를 출력한다.

```
pred = model.predict(X_test)
```

```
submission = pd.read_csv('sample_submission.csv')
```

```
submission.iloc[:,1:]=pred.reshape(-1,1600)
```

```
submission.to_csv('submission.csv',index=False)
```

```
submission
```

: predict() 함수는 모델을 사용하는 함수이다. X_test 데이터로 모델을 사용한다.  결과를 submission.csv로 저장한다.

결과