R 데이터 마이닝 기말 프로젝트

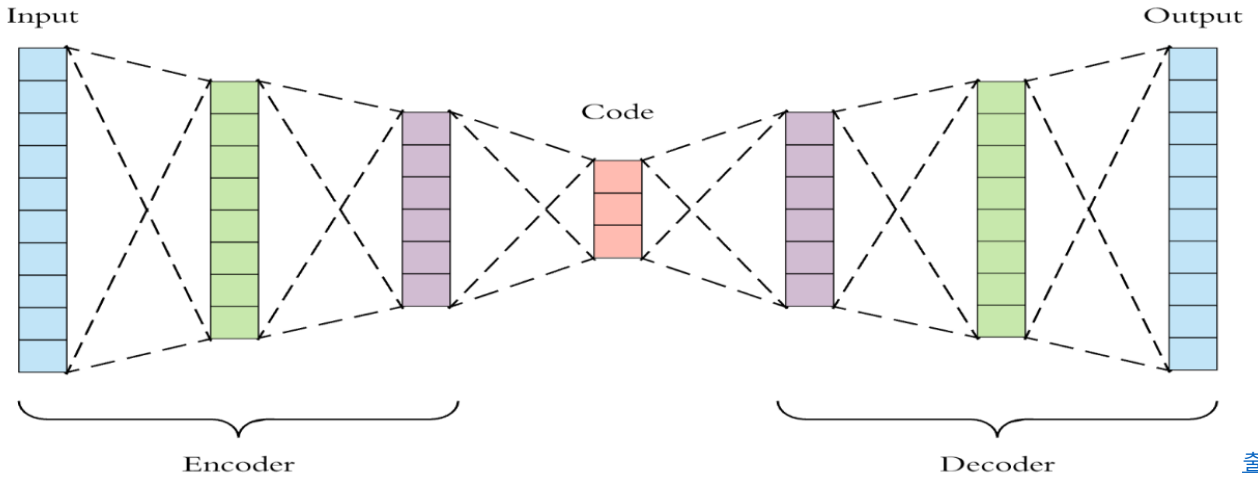
Auto Encoder를 활용한 유사 이미지 탐색

과목명	R 데이터 마이닝
학과	수학교육과
학번	12142821
이름	최영효

목차

1. Auto Encoder

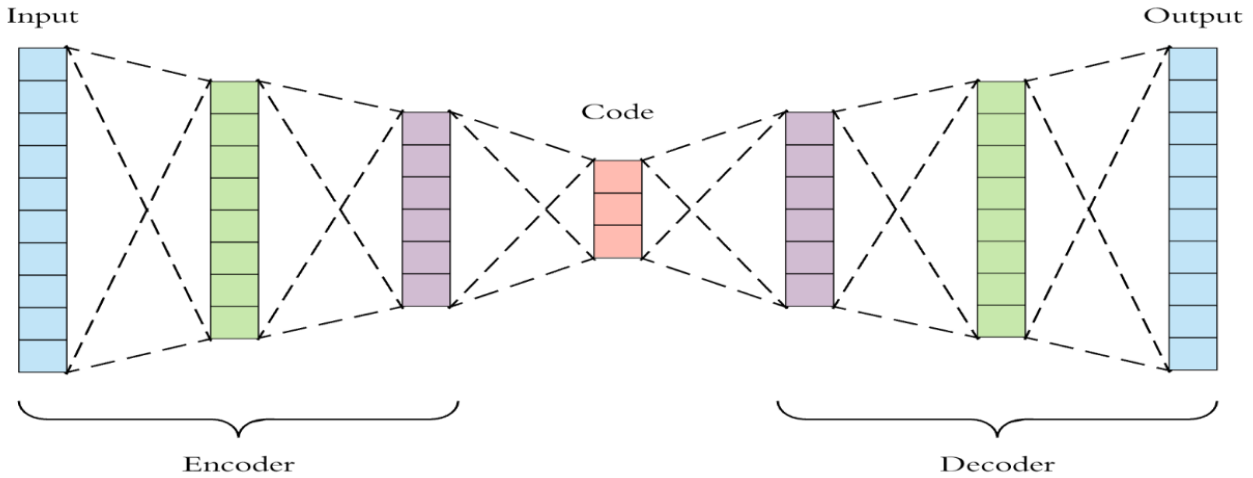
- Auto Encoder란 입력 값과 출력 값이 같도록 하는 모델이다.
- 정답 Label (Target)이 주어지지 않은 Unsupervised Learning 방식이다.
- 인코딩 과정에서 입력된 데이터의 핵심 Feature만 Hidden Layer에서 학습하고, 나머지 정보는 손실된다.
- 디코딩 과정에서 Hidden Layer의 출력 값을 뽑았을 때 완벽한 값 복사가 아닌 **입력 값의 근사치**가 된다.
- 출력 값이 입력 값과 최대한 같아지도록 튜닝함으로써, Feature를 잘 추출할 수 있게 하는 것이 Auto Encoder의 원리이다. (입력 값과 출력 값이 최대한 비슷하게 만드는 가중치를 찾아냄)



출처: https://gaussian37.github.io/dl-concept-autoencoder2/

1. Auto Encoder – Image Search 알고리즘

- Auto Encoder를 유사 이미지 탐색에 적용할 수 있다.
- Auto Encoder 모델에서 Encoding 부분까지의 Weight를 사용한다.
- 이미지들을 Encoding 부분까지 적용하여 Feature 값을 추출한 뒤 해당 이미지 값에 대한 kNN 알고리즘을 적용하여 유사 이미지들을 찾을 수 있다.



2. 개발 동기

- 지난 학기에 수강한 '소프트웨어 실습' 과목에서 유사 이미지를 탐색하는 프로그램을 개발했다.

- 이미지들 간에 Hash 값을 통해 비교하는 Hash Matching 알고리즘을 통해 유사 이미지를 탐색했다.

- 데이터 마이닝 수업에서 배운 Deep Learning 및 Auto Encoder를 활용해 Image Search에 적용해보고 싶었다.

3. 개발 과정

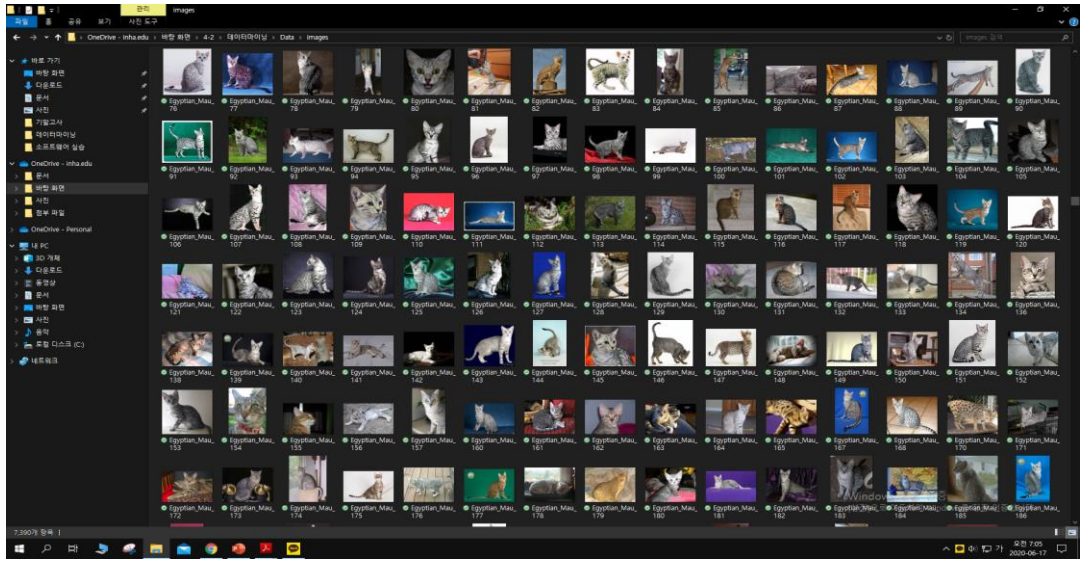
- 고양이/개 이미지 오픈 Data와 소프트웨어 실습 프로젝트에서 사용한 여행 이미지 두가지 데이터셋을 활용했다.

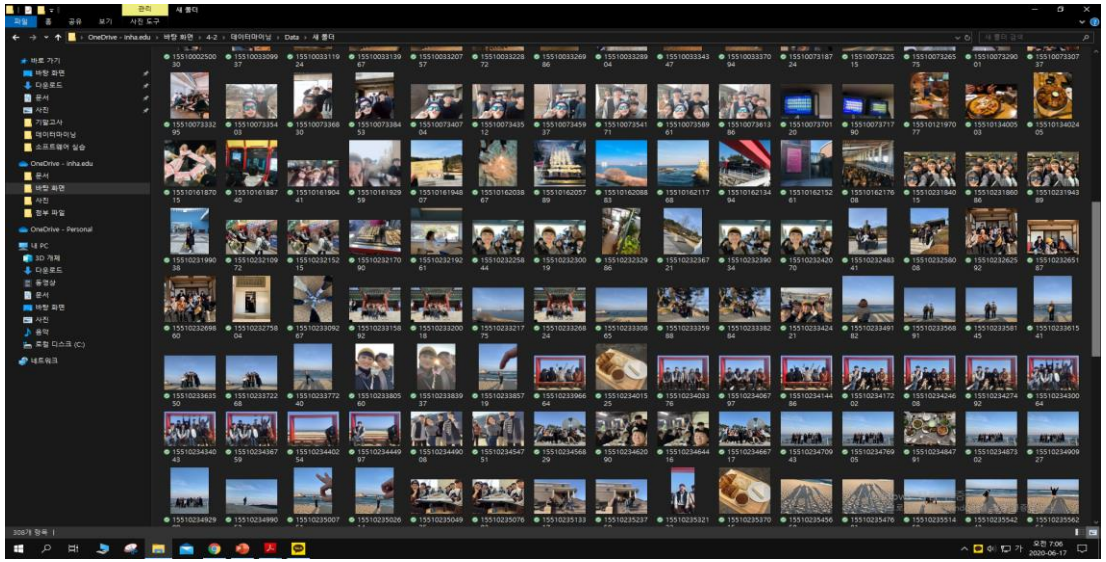
- Keras를 활용해 Auto Encoder 모델을 세워 Weight를 학습시켰다.

- Encoding 부분을 통해 찾고자 하는 이미지에 대한 값을 얻은 뒤, kNN 알고리즘을 활용해 유사 이미지들을 찾았다.
- 관련 코드는 https://github.com/choi-yh/DataMining에서 확인할 수 있다.

3. 개발 과정 – Data Set

- http://www.robots.ox.ac.uk/~vgg/data/pets/ 에서 The Oxford-IIIT Pet Dataset을 다운받아 활용했다.
- 37종의 개와 고양이에 대한 7390장으로 이루어져 있다.
- 소프트웨어 실습 프로젝트에서 사용한 300장 정도의 여행 사진을 활용했다.





3. 개발 과정 – Data Set

- 이미지를 Numpy array 형태인 .npy 파일로 만들어 저장했다.
- The Oxford-IIIT Pet Dataset 은 메모리 문제로 인해 160 * 160 형태로 resize 하여 저장했다.
- 여행 이미지는 600 * 600 형태로 resize 하여 저장했다.
- npy 불러온 뒤 Train, Test set으로 나누어 사용했다.

```
import numpy as np
from PIL import Image
```

```
import os
path = 'C:₩₩Users₩₩yhcho₩₩OneDrive₩₩바탕 화면₩₩새 폴더'

os.chdir(path) # 지정된 경로에 현재 작업 디렉토리 변경
os.getcwd() # 현재 작업 디렉토리 반환

files = os.listdir(path)
files.sort()

print(files[32], len(files))
```

```
1551003313967.jpg 308
```

```
def image_to_array(path):
    image = Image.open(path).convert('RGB') # load image
    image = image.resize((600, 600)) # resize image
    data = np.asarray(image)
    return data
```

```
X = []

for file_name in files:
    file_path = path + '/' + file_name
    data = image_to_array(file_path)
    X.append(data)
```

```
X = np.array(X)
```

```
X.shape
```

```
(308, 600, 600, 3)
```

```
np.save('C:₩₩Users₩₩yhcho₩₩OneDrive₩₩바탕 화면₩₩MyData', X)
```

3. 개발 과정 – The Oxford-IIIT Pet Dataset

- 5542장의 Train 이미지와 1848장의 Test 이미지로 분류해 학습했다.
- batch_size = 10, epoch = 10으로 설정했다.
- padding = 'same', optimizer='adam', loss='mse'를 적용했다.
- Layer를 다양하게 쌓으며 시도를 해봤고, loss가 가장 작은 모델을 사용했다.

```
batch_size = 10
```

```
input_img = layers.Input(shape=(160, 160, 3))
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPool2D((2, 2), padding='same')(x)
x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPool2D((2, 2), padding='same')(x)
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPool2D((2, 2), padding='same')(x)
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
x = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPool2D((2, 2), padding='same', name='encoder')(x)
```

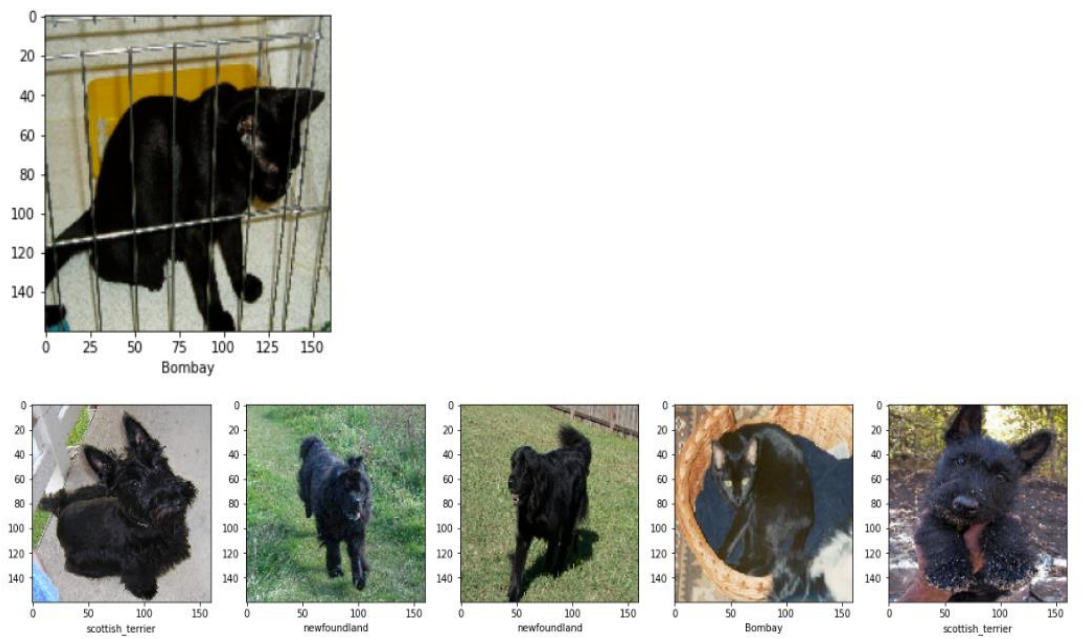
```
Total params: 11,361,027
Trainable params: 11,361,027
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
555/555 [==============================] - 95s 171ms/step - loss: 0.0283
Epoch 2/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0147
Epoch 3/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0117
Epoch 4/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0105
Epoch 5/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0099
Epoch 6/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0094
Epoch 7/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0090
Epoch 8/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0088
Epoch 9/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0087
Epoch 10/10
555/555 [==============================] - 93s 167ms/step - loss: 0.0084
<tensorflow.python.keras.callbacks.History at 0x7f054049b518>
```

3. 개발 과정 – The Oxford-IIIT Pet Dataset

- 모델을 학습시킨 뒤 Test dataset에 대한 Encoding을 적용한 뒤 kNN 알고리즘을 적용했다.
- 비슷한 이미지를 찾았지만 완전히 같은 품종을 찾기는 어려웠다.

```
from sklearn.neighbors import NearestNeighbors
n_neigh = 5

codes = codes.reshape(-1, 10*10*512)
print('codes.shape: ', codes.shape)
query_code = query_code.reshape(1, 10*10*512)
print('query_code.shape: ', query_code.shape)

nbrs = NearestNeighbors(n_neighbors=n_neigh).fit(codes)

distances, indices = nbrs.kneighbors(np.array(query_code))

closest_images = X_test[indices]

closest_images = closest_images.reshape(-1, 160, 160, 3)
print('closest_image.shape: ', closest_images.shape)
print('')
# print('distance: ', distances)
print('indices: ', indices)
```

```
codes.shape:  (1848, 51200)
query_code.shape:  (1, 51200)
closest_image.shape:  (5, 160, 160, 3)

indices:  [[1802 1113 1615  938  236]]
```



3. 개발 과정 – 여행 이미지

- 소프트웨어 실습에서 활용한 주제인 '여행지에서 찍은 사진 중 유사한 이미지를 찾기'를 시도했다.
- 300장 정도의 이미지를 231장의 Train set, 77장의 Test set으로 분류하여 학습했다.
- epoch=10, batch_size=5, padding='same', optimizer='adam', loss='mse'를 적용했다.
- Pet dataset에서 사용한 모델을 좀 더 단순화시켜 Weight를 학습했다.

```
input_img = layers.Input(shape=(600, 600, 3))
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPool2D((2, 2), padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPool2D((2, 2), padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPool2D((2, 2), padding='same', name='encoder')(x)

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = models.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.summary()

autoencoder.fit(X_train, X_train, epochs=10, batch_size=5, callbacks=None)
```

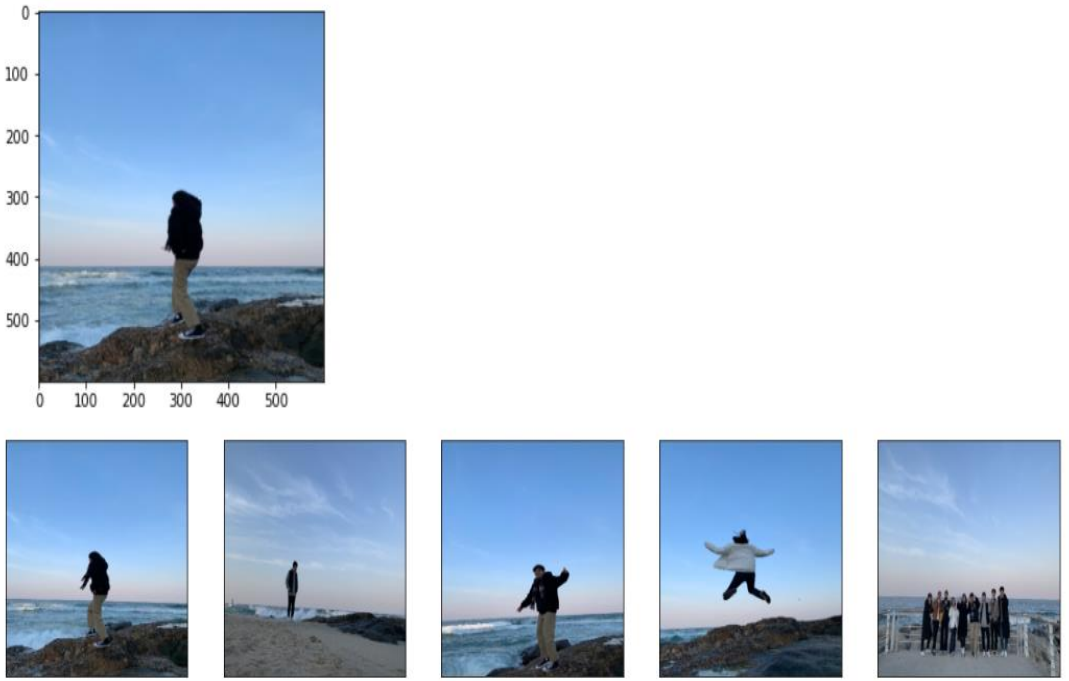
```
Total params: 72,883
Trainable params: 72,883
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
47/47 [==============================] - 15s 323ms/step - loss: 0.0427
Epoch 2/10
47/47 [==============================] - 14s 300ms/step - loss: 0.0178
Epoch 3/10
47/47 [==============================] - 14s 301ms/step - loss: 0.0146
Epoch 4/10
47/47 [==============================] - 14s 300ms/step - loss: 0.0119
Epoch 5/10
47/47 [==============================] - 14s 301ms/step - loss: 0.0101
Epoch 6/10
47/47 [==============================] - 14s 302ms/step - loss: 0.0097
Epoch 7/10
47/47 [==============================] - 14s 302ms/step - loss: 0.0097
Epoch 8/10
47/47 [==============================] - 14s 302ms/step - loss: 0.0093
Epoch 9/10
47/47 [==============================] - 14s 300ms/step - loss: 0.0089
Epoch 10/10
47/47 [==============================] - 14s 303ms/step - loss: 0.0088
<tensorflow.python.keras.callbacks.History at 0x7faaa0570780>
```

3. 개발 과정 – 여행 이미지

- 이전과 마찬가지로 학습 후 Test set에 Encoding을 적용한 뒤 kNN 알고리즘을 적용했다.
- 데이터가 적었지만 많이 있는 비슷한 이미지의 경우 어느정도 탐색이 되었다.

```
from sklearn.neighbors import NearestNeighbors
n_neigh = 5

codes = codes.reshape(-1, 75*75*16)
print('codes.shape: ', codes.shape)
query_code = query_code.reshape(1, 75*75*16)
print('query_code.shape: ', query_code.shape)

nbrs = NearestNeighbors(n_neighbors=n_neigh).fit(codes)

distances, indices = nbrs.kneighbors(np.array(query_code))

closest_images = X_test[indices]

closest_images = closest_images.reshape(-1, 600, 600, 3)
print('closest_image.shape: ', closest_images.shape)
print('')
# print('distance: ', distances)
print('indices: ', indices)
```

```
codes.shape:  (77, 90000)
query_code.shape:  (1, 90000)
closest_image.shape:  (5, 600, 600, 3)

indices:  [[60 44 14 25 32]]
```



4. 한계점

- **알고리즘의 문제**
  - The Oxford-IIIT Pet Dataset에 관심이 생겨 시도했지만 Auto Encoder를 통한 Image Search 에는 적절하지 않은 Dataset이었다.
  - Auto Encoder 개념으로는 속성이 다른 이미지를 찾을 순 있지만 세부적으로 비교하는 것은 어렵다는 사실을 알았다. (ex) 사람, 개, 비행기 중 사람을 찾을 순 있지만 사람 중에서 '최영효'라는 인물을 찾기는 어렵다.)
  - CBIR(Content Based Image Retrieval)에 대한 이해 및 다른 알고리즘을 적용해야 한다.
- **메모리 문제**
  - 이미지를 Numpy array로 변환할 때 resize를 했는데, 원래 사이즈로 변환했을 때 메모리 문제로 인해 데이터를 불러오는 것이 불가능했다. 이로 인해 이미지 사이즈를 줄여서 사용해야 했다.
- **데이터의 부족**
  - 여행 이미지의 경우 실제 여행에서 촬영한 이미지를 활용했기 때문에 데이터가 부족했고, 유사 이미지로 판별할 수 있는 테스트 데이터도 부족했다.

5. 느낀 점

- **Deep Learning에 실제 데이터를 적용한 경험**
  - Deep Learning에 대한 이론 공부를 했고, Mnist, Fashion Mnist와 같이 교과서처럼 사용되는 Dataset을 활용해 연습을 했지만 실제 Dataset을 구하고 가공해 모델을 세워 결과를 얻은 경험은 처음이었다.
  - 모델을 세울 때 기존에는 예제를 보고 따라했지만 새롭게 세우려다 보니 많은 시도가 필요했다.
  - 시도를 하면서 어떤 식으로 학습이 진행되는 지에 대한 공부가 많이 되었다.
- **빅데이터 처리의 중요성**
  - 공부를 하면서 메모리가 부족해 문제가 생긴 것은 처음이었다. 이러한 경험을 통해 빅데이터에 처리에 대한 중요성을 느꼈고, 데이터 엔지니어링에 대한 공부가 필요하다는 생각이 들었다.