국가별 랜드마크 분류 모델 만들기

2020. 06. 18

박 재완, 정 민지, 조 예슬, 최 희경

Agenda

개요

- 프로젝트 방향성

데이터 수집

- 구글 이미지 크롤링
- 데이터 정제
- 데이터 탐색

모델링

- 모델 구성
- 모델 훈련 및 평가
- 모델 성능 개선

예측 및 결과 시각화

- 예측 하기
- 예측 결과 시각화
- 요약

프로젝트 개요

1. 목적

◆ 학습내용을 적용하여 이미지 수집부터 CNN모델 생성 및 모델성능개선 방법을 연습한다.

- 랜드마크 이미지 수집과 각 카테고리 별 분류 모델 만들기
- 모델 성능 개선 활동

2. 자료 내용

- 랜드마크 카테고리 : **17개**

["에펠탑", "구원의 예수상", "개선문", "만리장성", "타지마할", "숭례문", "모아이석상", " 석굴암 본존불 " , " 금문교 " , " 자유의 여신상 " , " 피사의 사탑 " , " 콜로세움 " , " 산티아고 베르나베우 " , " 스핑크스 " , " 부르즈할리파", "런던 브리지", "런던 아이"]

- 최종 이미지 데이터 수 : **4,426**개(최초 이미지 데이터 수집 : 6,800여장)

- Augmentation 후 이미지 데이터 수 : **약 65,000장**

3. 모델링

- Keras 활용

4. 모델 성능개선활동

1) 하이퍼 파라미터 조정
2) 레이어 조정
3) 기타 (image augmentation, VGG-16 적용 시도 / 모델평가방법 변경 : accuracy→상위3개중 정답 있으면 맞춘 것으로 조정)

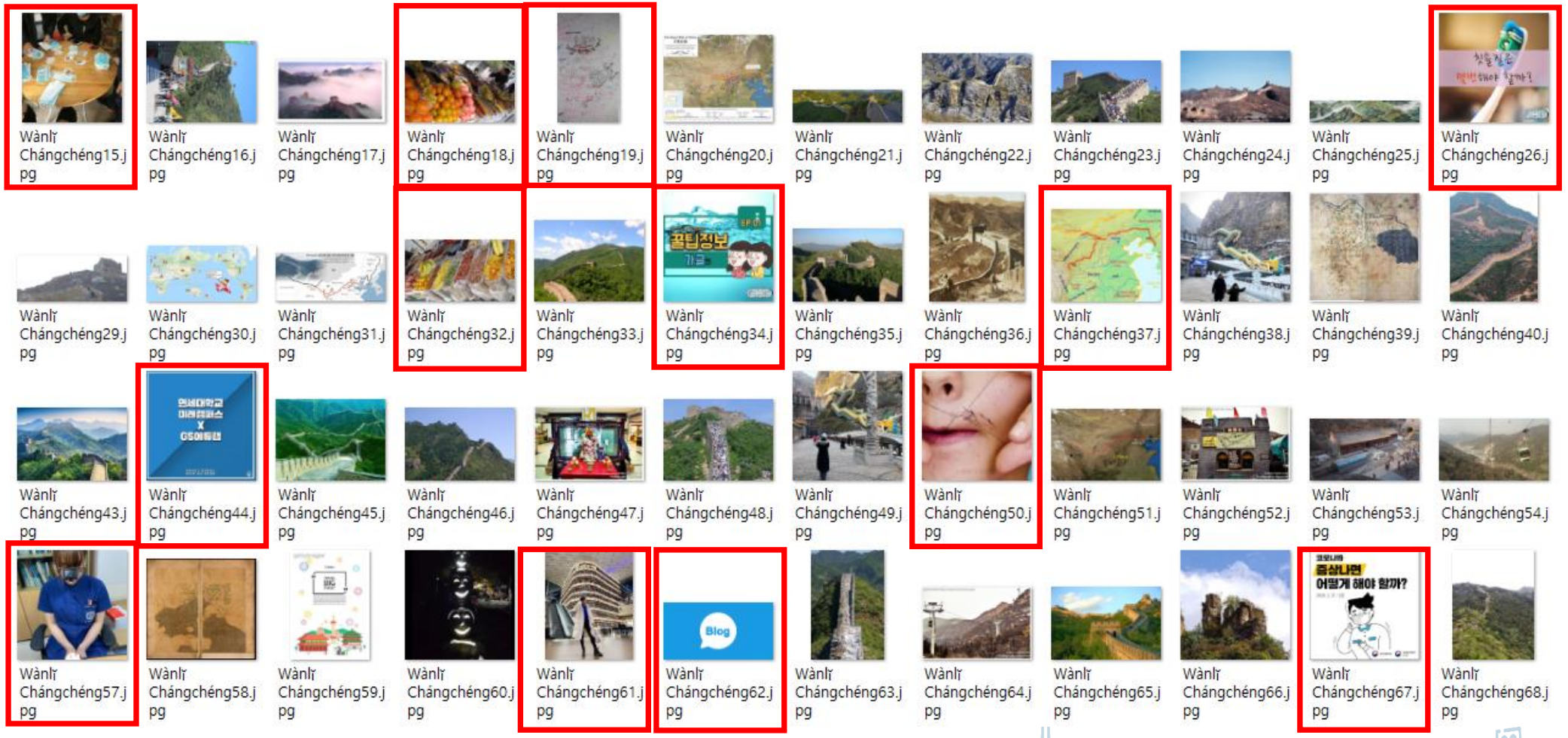
데이터 수집 – 구글 이미지 크롤링

```python
search = [
"Eiffel Tower"
,"Cristo Redentor"
,"Triumphal Arch"
,"Wànlǐ Chángchéng"
,"Taj Mahal"
,"Sungnyemun Gate"
,"Moai"
,"Seokguram buddha"
,"Golden Gate"
,"Statue of Liberty"
,"Torre di Pisa"
,"Colosseum"
,"Santiago Bernabéu"
,"Sphinx"
,"Burj Khalifa"
,"London Eye"
,"London Tower Bridge"
]

driver = webdriver.Chrome()

for i in range(2):
    name = search[i]
    url = "https://www.google.co.in/search?q=" + search[i] + "&tbm=isch"

    driver.implicitly_wait(10) # seconds
    driver.get(url)


    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
    driver.execute_script("window.scrollBy(0,10000)")
    time.sleep(2)
```

```python

    html = driver.page_source
    soup = BeautifulSoup(html)

    img = soup.select('.rg_i.Q4LuWd')
    n = 1
    imgurl = []
    for i in img:
        try:
            imgurl.append(i.attrs["src"])
        except KeyError:
            imgurl.append(i.attrs["data-src"])

    for i in imgurl:
        urlretrieve(i, "E:₩₩Eric_Github₩₩Image_project₩₩" + name + str(n) + ".jpg")
        n += 1
        print(imgurl)
        if (n == 500 ):
            break



driver.close()
```

- **최초 크롤링 이미지 : 17개 카테고리 6,800여장**

데이터 수집 – 데이터 정제 예시



- **데이터 정제 (최종: 4,426장)**

  : 훈련용 데이터 중 오류가 있는 이미지는 눈으로 확인 후 삭제

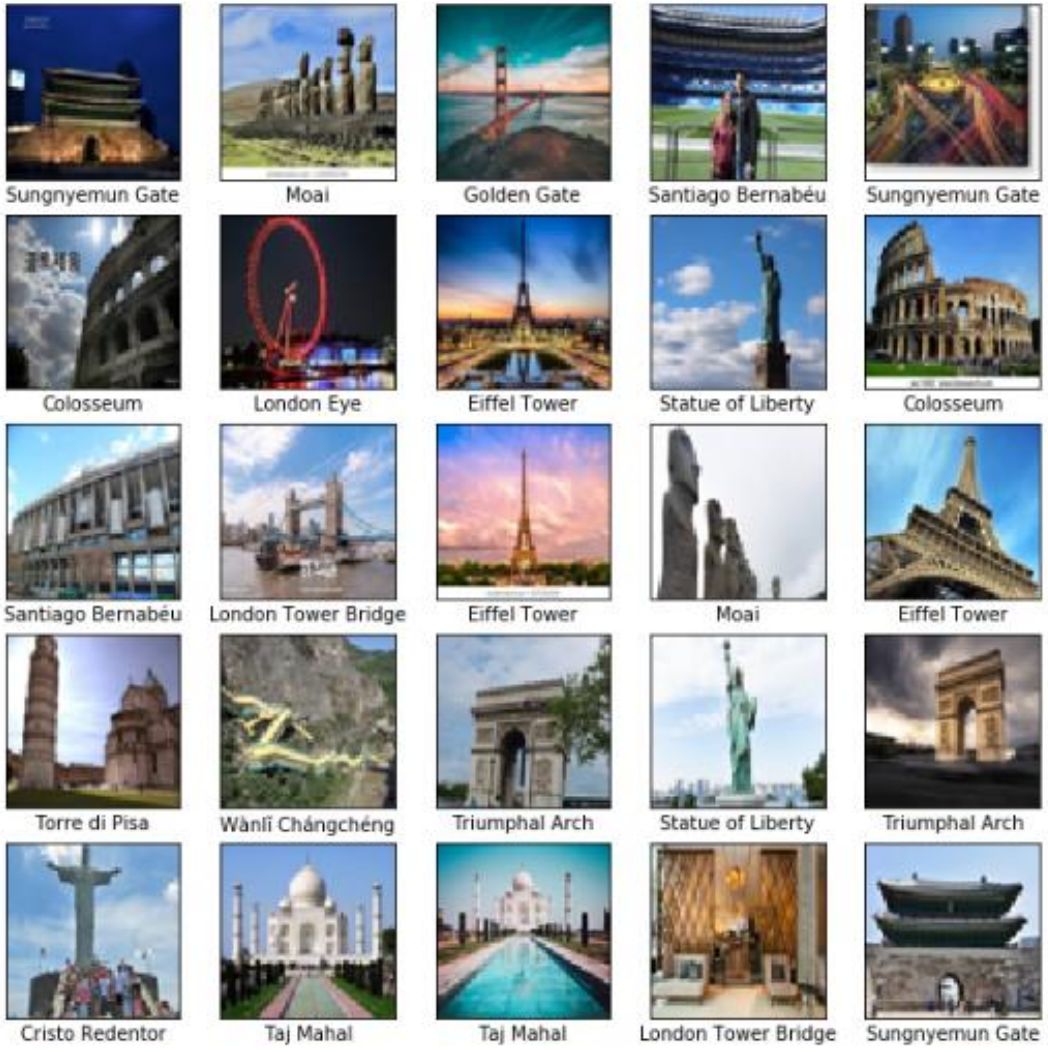
데이터 수집 – 데이터 로딩

```python
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from PIL import Image

import os, glob
import numpy as np

# 현재 경로 확인
print(os.getcwd())

#%%
# 이미지 경로 지정
root_dir = './landmark/'

# 카테고리 정보를 담는 리스트를 선언
categories = [
"Eiffel Tower"
,"Cristo Redentor"
,"Triumphal Arch"
,"Wànlǐ Chángchéng"
,"Taj Mahal"
,"Sungnyemun Gate"
,"Moai"
,"Seokguram buddha"
,"Golden Gate"
,"Statue of Liberty"
,"Torre di Pisa"
,"Colosseum"
,"Santiago Bernabéu"
,"Sphinx"
,"Burj Khalifa"
,"London Eye"
,"London Tower Bridge"
]

nb_classes = len(categories)

image_width = 64
image_height = 64
```

```python
#%%
# 데이터 변수
X = []   # 이미지 데이터
Y = []   # 레이블 데이터

for idx, category in enumerate(categories):   # enumerate : 인덱스, 값 반환
    image_dir = root_dir + category
    files = glob.glob(image_dir + '/' + '*.jpg')   # glob : 지정한 경로에 있는 파일 리스트를 가져옴
    print(image_dir + '/' + '*.jpg')
    print('해당 폴더 파일 갯수 : ',len(files))   # 이미지를 제대로 불러왔는지 확인

    for i, f in enumerate(files):   # 이미지 로딩
        img = Image.open(f)         # 01 이미지 파일 불러오기
        img = img.convert('RGB')    # 02 RGB로 변환
        img = img.resize((image_width, image_height))  # 03 이미지 크기를 resize
        data = np.asarray(img)      # 04 해당 이미지를 숫자 배열 데이터로 변경
        X.append(data)              # 05 변경한 데이터를 X의 리스트에 추가
        Y.append(idx)               # 06 해당 idx(이미지가 속한 범주)에 추가(Y값)

X = np.array(X)
Y = np.array(Y)

print(X.shape)
print(Y.shape)

#%%
X_train, X_test, Y_train, Y_test = train_test_split(X, Y)
xy = (X_train, X_test, Y_train, Y_test)

# 데이터 파일로 저장
np.save(root_dir + 'landmark.npy', xy)
```

- **train: 3,319장, test: 1,107장**
- 이미지 크기 : thumbnail (image size확대에 한계)

데이터 수집 – 데이터 탐색




```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i], cmap=plt.cm.binary)
    plt.xlabel(categories[Y_train[i]])
plt.show()
```

모델링 – 모델 구성

```
[12] model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 32)      896
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 128, 128, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 64)        18496
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64, 64, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 128)       73856
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 32, 32, 128)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16, 16, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32768)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               16777728
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 17)                8721
_________________________________________________________________
activation_1 (Activation)    (None, 17)                0
=================================================================
Total params: 16,879,697
Trainable params: 16,879,697
Non-trainable params: 0
_________________________________________________________________
```

```
[45] #%% 03 모델 구성 함수 생성(검증방법:정확도)
     def build_model(in_shape):
         model = Sequential()

         model.add(Convolution2D(32,3,3, border_mode='Same', input_shape=in_shape))
         model.add(LeakyReLU(alpha=0.01))
         model.add(MaxPooling2D(pool_size=(2,2)))
         model.add(Dropout(0.25))

         model.add(Convolution2D(64,3,3, border_mode='Same'))
         model.add(LeakyReLU(alpha=0.01))
         model.add(MaxPooling2D(pool_size=(2,2)))
         model.add(Dropout(0.25))

         model.add(Convolution2D(128,3,3, border_mode='Same'))
         model.add(LeakyReLU(alpha=0.01))
         model.add(MaxPooling2D(pool_size=(2,2)))
         model.add(Dropout(0.25))

         model.add(Flatten())
         model.add(Dense(512))
         model.add(LeakyReLU(alpha=0.01))
         model.add(Dropout(0.5))
         model.add(Dense(nb_classes))
         model.add(Activation('softmax'))
```

- Dropout 추가
- layer 추가(convolution , pooling 1층 추가)
- Activation Function 수정

모델링 – 모델 컴파일

- Optimizer : RmsProp → adam

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

- 모델 평가 방법 : accuracy → top3

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=[tf.keras.metrics.TopKCategoricalAccuracy(k=3,
                                                                name='top_k_categorical_accuracy',
                                                                dtype=None)])
```

모델링 – 모델 훈련 및 평가

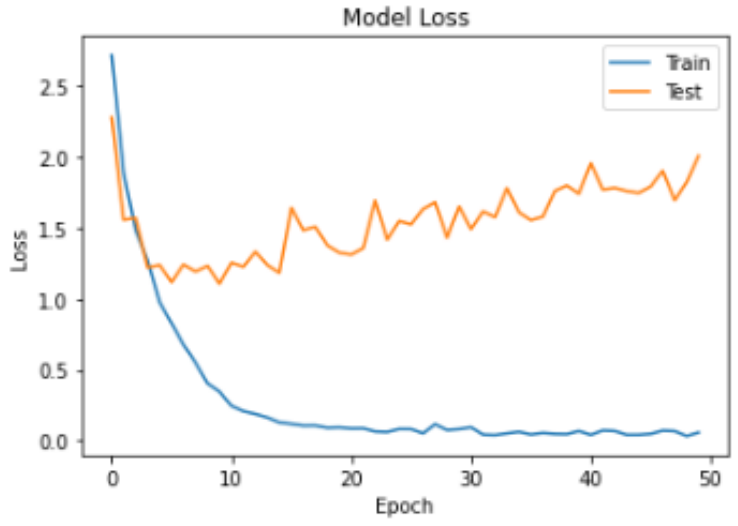
```
[7] from keras.callbacks import EarlyStopping

    epochs=300

    early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)

    def model_train(x, y):
        print(x.shape[1:])
        model = build_model(x.shape[1:])
        model.fit(x, y, batch_size=32, epochs=epochs, callbacks=[early_stopping])

        return model
```

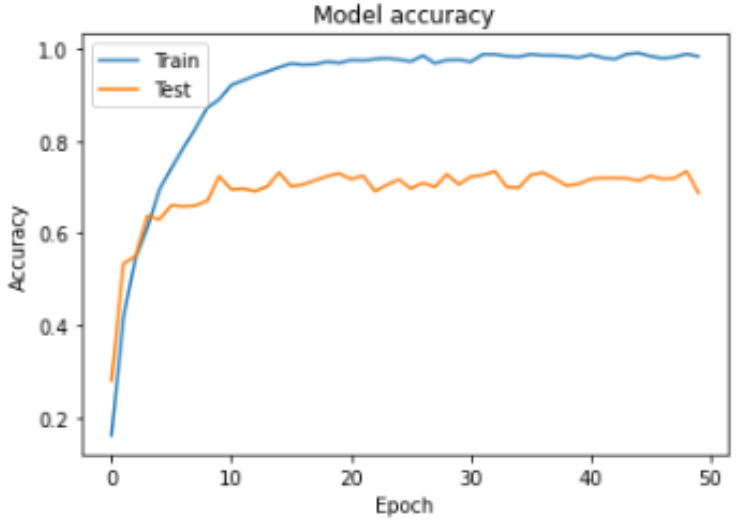
```
print(x_train.shape, y_train.shape, x_test.shape,  y_test.shape)

# 모델 학습
model = model_train(x_train, y_train)
```

```
Epoch 11/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.1916 - accuracy: 0.9409
Epoch 12/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.1578 - accuracy: 0.9491
Epoch 13/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.1810 - accuracy: 0.9425
Epoch 14/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.1214 - accuracy: 0.9623
Epoch 15/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.0958 - accuracy: 0.9684
Epoch 16/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.0747 - accuracy: 0.9783
Epoch 17/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.0623 - accuracy: 0.9846
Epoch 18/50
3319/3319 [==============================] - 4s 1ms/step - loss: 0.1180 - accuracy: 0.9654
```

- EarlyStopping : 300적용 → stop 되지 않고 300까지 진행

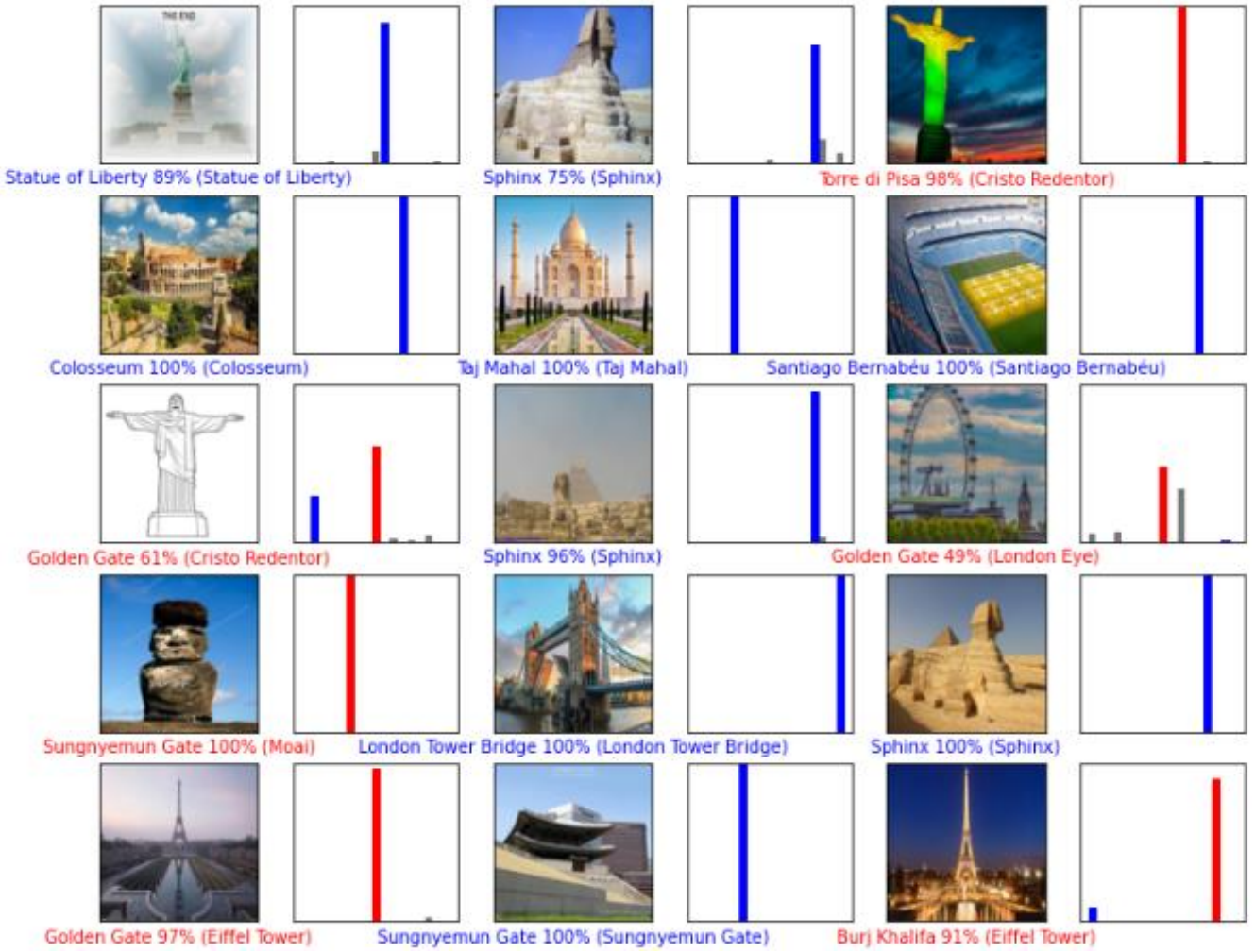
모델링 – 모델 훈련 및 평가

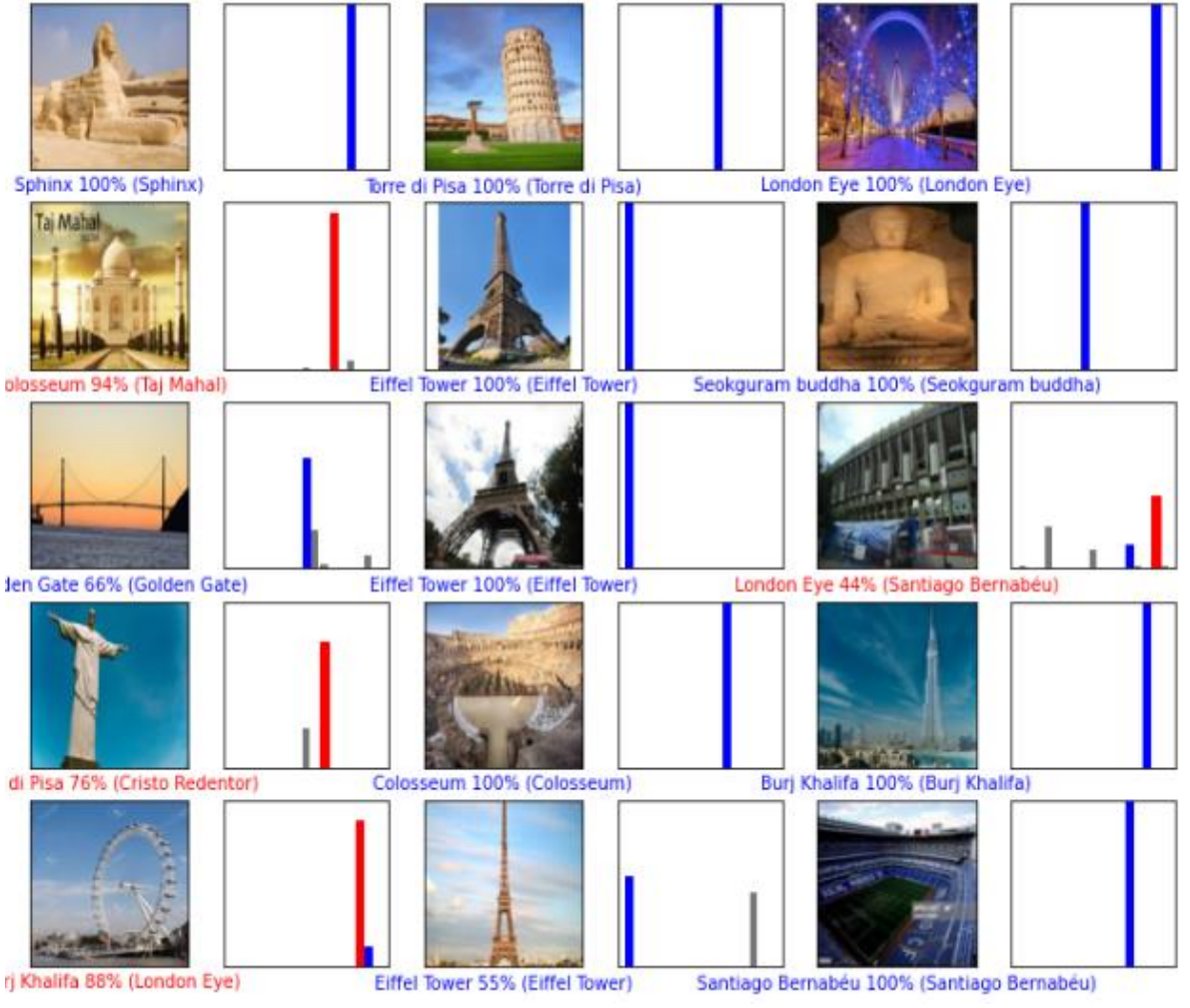





```python
def plt_show_loss(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train','Test'], loc=0)

def plt_show_acc(history):
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train','Test'], loc=0)

plt_show_loss(history)
plt.show()

plt_show_acc(history)
plt.show()
```

예측 및 결과 시각화


Statue of Liberty 89% (Statue of Liberty)
Sphinx 75% (Sphinx)
Torre di Pisa 98% (Cristo Redentor)
Colosseum 100% (Colosseum)
Taj Mahal 100% (Taj Mahal)
Santiago Bernabéu 100% (Santiago Bernabéu)
Golden Gate 61% (Cristo Redentor)
Sphinx 96% (Sphinx)
Golden Gate 49% (London Eye)
Sungnyemun Gate 100% (Moai)
London Tower Bridge 100% (London Tower Bridge)
Sphinx 100% (Sphinx)
Golden Gate 97% (Eiffel Tower)
Sungnyemun Gate 100% (Sungnyemun Gate)
Burj Khalifa 91% (Eiffel Tower)


Sphinx 100% (Sphinx)
Torre di Pisa 100% (Torre di Pisa)
London Eye 100% (London Eye)
olosseum 94% (Taj Mahal)
Eiffel Tower 100% (Eiffel Tower)
Seokguram buddha 100% (Seokguram buddha)
den Gate 66% (Golden Gate)
Eiffel Tower 100% (Eiffel Tower)
London Eye 44% (Santiago Bernabéu)
di Pisa 76% (Cristo Redentor)
Colosseum 100% (Colosseum)
Burj Khalifa 100% (Burj Khalifa)
rj Khalifa 88% (London Eye)
Eiffel Tower 55% (Eiffel Tower)
Santiago Bernabéu 100% (Santiago Bernabéu)

프로젝트 결과 요약

구 분	내 용	이 슈
이미지 사이즈 조정	32 → 64 → 128 → (256)	• 원본 이미지 사이즈 자체가 작음 • 256으로 늘렸을 때 속도 저하
Dropout 추가	0.25 / 0.25 / 0.5	• 현 모델에서 정확도에 큰 영향을 미치지는 않았음
Layer 추가	Convolution, pooling 추가	• 정확도 상승
GPU/TPU 사용	Google colab 활용	• Epoch당 소요시간 급감(32초 → 6초)
Activation Function	Relu → Leakyrelu(alpha 0.01)	• 정확도 상승
Optimizer	RmsProp → adam	• 정확도 상승
모델 평가 방법 변경	Accuracy → top3 categorical accuracy	• 정확도 상승
VGG 모델 활용	VGG 16, VGG13 수정 적용	• 정확도 급감 • → 향후 fine tune 적용해 볼 예정
Image augmentation	ImageDataGenerator(30배 증폭)	• 65,507개 이미지(총 용량 3G)로 증폭 • 정확도 **0.8553**으로 크게 상승

Appendix.01

	Loss	Accuracy	# of layers	Image_size	Epoch	Batchsize	1st_Conv2d	1st_Activation	Max_Pooling2D	Dropout	2nd or 3rd_Activation	Model.copile _optimizer
try_1	1.6263	0.6468	5	32	10	64	(32,3,3)	Relu	(2,2)		-	rmsprop
try_2	5.8495	0.6134	5	64	50	32	(32,3,3)	Relu	(2,2)	0.25	-	rmsprop
try_3	2.2813	0.6486	5	128	30	32	(32,3,3)	Relu	(2,2)	0.25	-	rmsprop
try_4	2.4084	0.6748	5	256	30	32	(32,3,3)	Relu	(2,2)	0	-	rmsprop
try_5	2.3220	0.6757	7	256	30	32	(32,3,3)	Relu	(2,2)	0.25	Relu	rmsprop
try_6	1.3419	0.7200	7	256	30	32	(32,3,3)	Relu	(2,2)	0.25	Relu	Adam
try_7	1.3986	0.7254	7	256	30	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_8	2.5792	0.6480	7	256	30	32	(32,3,3)	Elu	(2,2)	0.25	Elu	rmsprop
try_9	1.747	0.6820	7	128	30	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_12	2.2183	0.6775	7	256	30	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_10(top3)	1.5888	0.9437	7	128	30	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_11(top3)	4.5057	0.9941	7	128	300	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_12	2.2183	0.6775	7	256	30	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_13	4.8315	0.7299	7	128	300	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_14	2.173	0.6893	7	128	50	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_15(=14)	1.799	0.72	7	128	50	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_16(=14)	1.8641	0.7103	7	128	50	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_16(top3)	2.3362	0.9703	7	128	50	32	(32,3,3)	LeakyRelu(alpha=0.01)	(2,2)	0.25	LeakyRelu(alpha=0.01)	Adam
try_17(VGG)	10.6	0.4336	13	128	50	32	(64,3,3)	Relu	(2,2)	0	Relu	Adam
try_18(VGG)	5.823	0.4435	13	128	50	32	(64,3,3)	Relu	(2,2)	0.25	Relu	Adam

감사합니다