이미지 파일의 변환

최석재 *lingua@naver.com*

MNIST JPG

MNIST JPG

- MNIST JPG 데이터 처리

  - 앞에서는 Keras에 내장된 MNIST 데이터를 분석, 예측하는 방법을 보았다
  - 여기서는 JPG 이미지를 읽어 처리하는 방법을 살펴본다

  - 여기서는 다음의 과정으로 진행할 것이다

1. 사진 파일을 읽는다
2. JPG 파일을 numpy 배열로 된 RGB 픽셀값으로 변환한다
3. 정수로 된 RGB 픽셀값을 float32 부동소수 타입의 텐서로 변환한다
4. 변환된 픽셀값을 0~1 사이로 스케일링한다

※ colab에 대량의 파일을 올려둘 경우 적정 시간 내에 로드가 안되므로
colab에는 30~100개 정도의 샘플 파일만 올려두었음
(다양한 숫자값으로 구성되어야 함)

MNIST JPG

- 파일 불러오기 테스트
  - 연습을 위해 한 두 개 파일만 불러와 본다
  - Python Image Library를 사용한다

- from google.colab import drive
- drive.mount('/content/gdrive')

- import os, sys
- from IPython.display import display
- from IPython.display import Image as _Imgdis
- from PIL import Image
- import numpy as np

PIL의 Image와 이름이 동일하여 다르게 짓는다

PIL은 pip install pillow로 설치

- folder = '/content/gdrive/MyDrive/pytest_img/mnist_jpg'

해당 폴더 내에서 파일에 해당하는 것들의 목록을 구한다

- files = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]

- print(files)
- print("Working with {0} images".format(len(files)))

```
['img_6_ (5).jpg', 'img_3_ (5).jpg', 'img_2_ (4).jpg', 'img_8_ (2).jpg', 'img_5_ (3).jpg', 'img_1_ (2).jpg',

 Working with 60000 images
```

※ full data는 _original_jpg_backup 폴더에

이미지 확인

처음 두 개만 출력

- for i in range(0, 2):
  print(files[i])
  display(_Imgdis(filename=folder + "/" + files[i], width=30, height=40))

img_0_ (1).jpg

img_0_ (10).jpg

◀ 파일명의 구조를 확인한다

_Imgdis는 이미지 객체를 생성하고,
display 함수는 이미지 객체를 받아서 Jupyter Notebook에 출력한다

이미지 라벨 추출

만약 import가 잘 되지 않으면 앞에 tensorflow를 붙인다
tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img

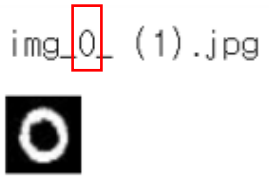
또는 from **keras.utils** import array_to_img, img_to_array, load_img

- from keras.preprocessing.image import array_to_img, img_to_array, load_img

- file_names = []
- file_labels = []

- for _file in files:
  file_names.append(_file)
  label_start = _file.find("_")+1 # 라벨의 시작 위치 (4)
  label_end = _file.find("_ (") # 라벨의 끝 위치 (5)
  file_labels.append(int(_file[label_start:label_end]))

파일이름으로부터 라벨을 추출한다



- print("Files in folder: %d" % len(file_names)) # 60000

PIL 파일을 넘파이 배열로 변환하기

- os.chdir(folder)

한 개 파일을 로드한다
- img = load_img(file_names[0])
- print("Orignal:" ,type(img))

PIL 파일을 넘파이 배열로 변환
- img_array = img_to_array(img)

- print("NumPy array info:")
- print(type(img_array))
- print("type:", img_array.dtype)
- print("shape:", img_array.shape)

- load_img() - 이미지를 PIL(Python Image Library) 파일로 변환
- img_to_array() - PIL 파일을 넘파이 배열로 변환

```
Orignal: <class 'PIL.Image.Image'>
NumPy array info:
<class 'numpy.ndarray'>
type: float32
shape: (28, 28, 3)
```

(높이, 너비, 채널)

이미지 shape 확인

- image_height = img_array.shape[0]
- image_width = img_array.shape[1]
- channels = img_array.shape[2]

- dataset = np.ndarray(shape=(len(file_names), image_height, image_width, channels), dtype =np.int32)
- print(dataset.shape)

```
(60000, 28, 28, 3)
```

4차원 데이터를 갖는 넘파이 배열을 초기화한다

전체 이미지 파일을 넘파이 배열로 변환

- for count, item in enumerate(file_names):

```
    img = load_img(file_names[count])
    img_array = img_to_array(img)
    img_array = img_array.reshape((28, 28, 3))
    dataset[count] = img_array

    if count % 5000 == 0:
        print("%d images to array" % count)
```

- print("All images to array!")

```
# file_names[count] == item
```

```
0 images to array
5000 images to array
10000 images to array
15000 images to array
20000 images to array
25000 images to array
30000 images to array
35000 images to array
40000 images to array
45000 images to array
50000 images to array
55000 images to array
All images to array!
```

※ 전체는 60,000 (0~59999) 개

이미 shape가 (28, 28, 3) 이므로 img_array = img_array.reshape((28, 28, 3)) 는 안 해줘도 된다
비슷한 상황에서 reshape가 필요한 경우를 위해 남겨놓은 코드

한 개 이미지 파일의 변환 결과 확인

- print(dataset.shape)
- print(file_labels[40])

```
(60000, 28, 28, 3)
6
```

- display(_Imgdis(filename=folder + "/" + files[40], width=30, height=40))

변환 전의 파일로 이미지를 확인

- np.set_printoptions(linewidth=np.inf)
- print(dataset[40, :, :, 0])

변환 후의 np array 출력

```
[[  0   0   0   0   0   0   0   0   1   0  17   5   0   2   0   1   0   4   4   0   9   0   5   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   7   1   0   0   0   6  12   0   1   0  10   0   0   0   5   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  18   0   0  15   0   0  10  12 166 173  11  12   2   0   1   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   6   0   6   8   0   0   2   0 252 246   0   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   5   0  11   3   0  14  14   0 110 253 179  11   8   0  31   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   2   0   3   0   7   0   0  65 236 240  69   5   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  16   0   0  22   0  26 207 245 161  13   0   0  13   0   9   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   3   2   8   0  16 161 255 222  21   0   3   0   0   2   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  16   0  12  70 242 254  55  10   1   1   1   0   0   7   0   0   0   0]
 [  0   0   0   0   0   0   0   0   3   9   0   5  28 210 255 107   3   0   6   0  13   7  11   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   3   1   0 178 247 176   0   0   2   0  10   0   0   0  13   0   0   0   0]
 [  0   0   0   0   0   0   0   0  14   0   0  60 238 249  13   3   0   2   2   4  12   7   3   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  18  18 243 246  98   4   0  18   5  88 139 166  44   0  13   0   0   0   0]
 [  0   0   0   0   0   0   0   0   9   0 152 255 214  13   0  22 108 196 254 255 240 248  74   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 103 255 210  51   0  80 209 251 201  84  76 100 255  83  11   0   0   0   0]
 [  0   0   0   0   0   0   0   0  26 199 250 129   3 189 251 242 122  26   0   0 112 235  60   0   0   0   0   0]
 [  4   0   2   6   0   0   0   3 150 255 203  26 211 255 227  50   0   1  17   0 154 211   0   6   0   0   0   0]
 [  5   0   1   0   0   0   9  42 254 255  99 247 255 185  12   0   7   8   0 152 205  26   0   0   0   0   0   0]
 [  1   0   2   0   0   0  14  72 242 248 255 255 178  17   8   3   0   2 160 255 107   0  16   0   0   0   0   0]
 [  0   0   2   0   5   0   2  66 255 247 252 180  17   7   0   0  81 203 255 159  27   0   8   0   0   0   0   0]
 [  0   1   0   0   6   0   0  46 245 255 255 156  93  86 107 191 234 240 117  23   7   0   0   0   0   0   0   0]
 [  5   0   0   0   1   0   0  31 161 239 255 255 253 255 251 234 164  48   0   0   0   8   0  10   0   0   0   0]
 [  4   0   3   0   0   3   0  14   3   2   0  11   0   0  21   0   0   9   0   0  17   0   0   2   0   0   0   0]
 [  0   0   9   4   0   3   0   0   1   0   0   0  12   4   0   3   1   0  15   6   0   5   1   6   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

훈련 데이터와 테스트 데이터 분리

- from sklearn.model_selection import train_test_split

- train_images, test_images, train_labels, test_labels = train_test_split(dataset, file_labels, test_size=0.2)
- print("Train set size: {0}, Test set size: {1}".format(len(train_images), len(test_images)))

```
Train set size: 48000, Test set size: 12000
```

모델 설계

- from keras import models
- from keras import layers
- import numpy as np

- model = models.Sequential()
- model.add(layers.Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 3), activation='relu'))
- model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
- model.add(layers.MaxPooling2D(pool_size=2))
- model.add(layers.Dropout(0.25))
- model.add(layers.Flatten())
- model.add(layers.Dense(128, activation='relu'))
- model.add(layers.Dropout(0.25))
- model.add(layers.Dense(10, activation='softmax'))

- model.summary()

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        896

conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496

max_pooling2d (MaxPooling2D  (None, 12, 12, 64)        0
)

dropout (Dropout)            (None, 12, 12, 64)        0

flatten (Flatten)            (None, 9216)              0

dense (Dense)                (None, 128)               1179776

dropout_1 (Dropout)          (None, 128)               0

dense_1 (Dense)              (None, 10)                1290

=================================================================
Total params: 1,200,458
Trainable params: 1,200,458
Non-trainable params: 0
_________________________________________________________________
```

모델 컴파일

- model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc'])

데이터 reshape 및 스케일링

- print("before:", train_images.shape) # (48000, 28, 28, 3)

- train_images = train_images.reshape((len(train_images), 28, 28, 3))
- train_images = train_images.astype('float32')/255

- test_images = test_images.reshape((len(test_images), 28, 28, 3))
- test_images = test_images.astype('float32')/255

- print("after:", train_images.shape) # (48000, 28, 28, 3)

※ 현재 train_images의 shape가 이미 (48000, 28, 28, 3) 이므로 reshape를 하지 않아도 무방하다 (사용 방법 기록용)

라벨 원-핫 인코딩

- from tensorflow.keras.utils import to_categorical
- train_labels = to_categorical(train_labels)
- test_labels = to_categorical(test_labels)

모델 훈련

- history = model.fit(train_images, train_labels, epochs=5, batch_size=32, validation_data=(test_images, test_labels))

```
Train on 33600 samples, validate on 8400 samples
Epoch 1/5
33600/33600 [==============================] - 8s 226us/sample - loss: 0.2249 - acc: 0.9309 - val_loss: 0.0641 - val_acc: 0.9807
Epoch 2/5
33600/33600 [==============================] - 5s 155us/sample - loss: 0.0672 - acc: 0.9796 - val_loss: 0.0566 - val_acc: 0.9829
Epoch 3/5
33600/33600 [==============================] - 5s 154us/sample - loss: 0.0478 - acc: 0.9859 - val_loss: 0.0484 - val_acc: 0.9854
Epoch 4/5
33600/33600 [==============================] - 5s 155us/sample - loss: 0.0346 - acc: 0.9893 - val_loss: 0.0428 - val_acc: 0.9875
Epoch 5/5
33600/33600 [==============================] - 5s 152us/sample - loss: 0.0268 - acc: 0.9919 - val_loss: 0.0453 - val_acc: 0.9879
```

성능 평가

- test_loss, test_acc = model.evaluate(test_images, test_labels)
- print('test_acc:', test_acc)

```
8400/8400 [==============================] - 1s 128us/sample - loss: 0.0453 - acc: 0.9879
test_acc: 0.98785716
```

- acc = history.history['acc']
- val_acc = history.history['val_acc']
- loss = history.history['loss']
- val_loss = history.history['val_loss’]

- print('Validation Accuracy of each epoch:', val_acc)

```
Accuracy of each epoch: [0.9309226, 0.97964287, 0.9858631, 0.98925596, 0.99193454]
```

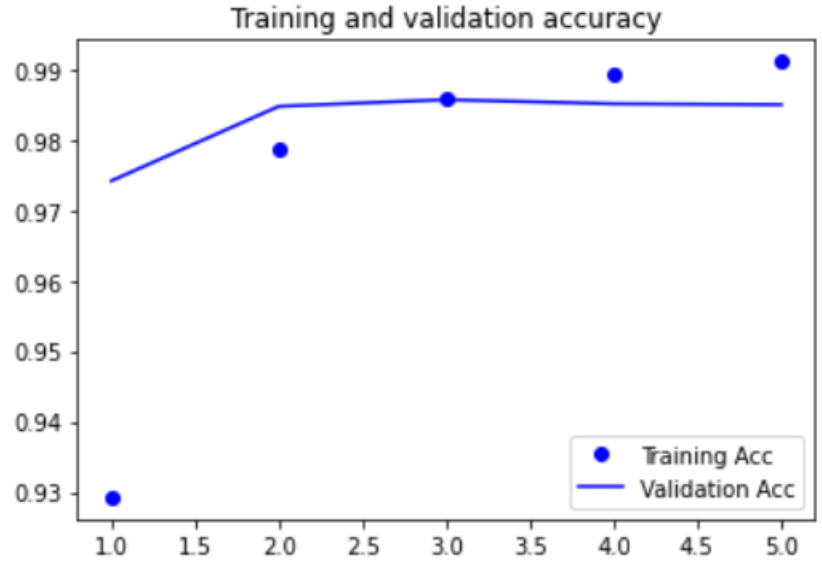
그래프 확인

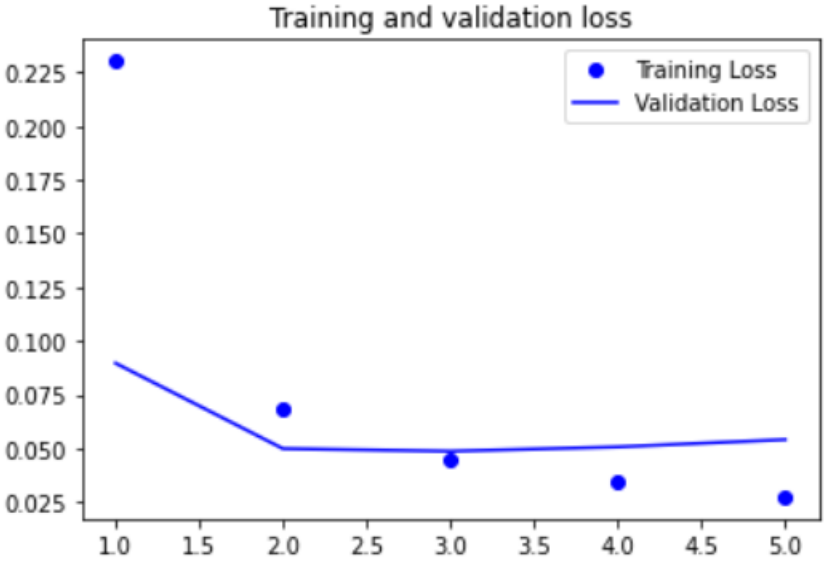
- import matplotlib.pyplot as plt
- epochs = range(1, len(val_acc) +1)

- plt.plot(epochs, acc, 'bo', label='Training Acc')
- plt.plot(epochs, val_acc, 'b', label='Validation Acc')
- plt.title('Training and validation accuracy')
- plt.legend()
- plt.show()

- plt.figure()

- plt.plot(epochs, loss, 'bo', label='Training Loss')
- plt.plot(epochs, val_loss, 'b', label='Validation Loss')
- plt.title('Training and validation loss')
- plt.legend()
- plt.show()





Cats and Dogs

cats and dogs

- folder = '/content/gdrive/MyDrive/pytest_img/cats_dogs'
- files = [f for f in os.listdir(folder) if os.path.isfile(os.path.join(folder, f))]
- print("Working with {0} images".format(len(files)))

리스트 컴프리헨션 해석

```
Working with 25000 images
```

※ full data는 _original_jpg_backup 폴더에
Colab으로는 small data를 이용하므로 학습이 완전하게 이루어지지 않는다

이미지 확인

처음 두 개만 출력

- for i in range(0, 2):
  print(files[i])
  display(_Imgdis(filename=folder + "/" + files[i], width=120, height=160))

cat.0.jpg

cat.1.jpg

파일 이름 패턴을 확인한다

cat.0.jpg
cat.1.jpg

이 부분이 label

이미지 라벨 추출

- from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img
- file_names = []
- file_labels = []

- for _file in files:
    file_names.append(_file)
    label_start = 0 # 라벨의 시작 위치
    label_end = _file.find(".") # 라벨의 끝 위치
    label_str = _file[label_start:label_end]

    if label_str == "cat":
        file_labels.append(0)
    elif label_str == "dog":
        file_labels.append(1)
    else:
        print("Error!")

파일 이름이 cat면 0, dog면 1로 라벨링

- print("Files in folder: %d" % len(file_names)) # 25000

shape 확인

- PIL 파일을 넘파이 배열로 변환하기

load_img() → img_to_array()
PIL np.array

```
os.chdir(folder)

# 한 개 파일을 로드한다
img = load_img(file_names[0])
print("Orignal:" ,type(img))

# PIL 파일을 넘파이 배열로 변환
img_array = img_to_array(img)

print("NumPy array info:")
print(type(img_array))
print("type:", img_array.dtype)
print("shape:", img_array.shape)
```

```
Orignal: <class 'PIL.JpegImagePlugin.JpegImageFile'>
NumPy array info:
<class 'numpy.ndarray'>
type: float32
shape: (374, 500, 3)
```

※ 이미지마다 shape가 다르다

```
# 다른 한 개 파일을 로드한다
img = load_img(file_names[1])
print("Orignal:" ,type(img))

# PIL 파일을 넘파이 배열로 변환
img_array = img_to_array(img)

print("NumPy array info:")
print(type(img_array))
print("type:", img_array.dtype)
print("shape:", img_array.shape)
```

```
Orignal: <class 'PIL.JpegImagePlugin.JpegImageFile'>
NumPy array info:
<class 'numpy.ndarray'>
type: float32
shape: (280, 300, 3)
```

이미지 shape 통일

모든 이미지의 size를 20 x 20 에 맞추기로 한다

- image_height = 20
- image_width = 20
- channels = 3

- dataset = np.ndarray(shape=(len(file_names), image_height, image_width, channels), dtype=np.int32) (25000, 20, 20, 3)을 담을 수 있는 dataset 변수 준비

- print(dataset.shape) # (25000, 20, 20, 3)

새 디렉토리 생성

사이즈가 조절된 이미지를 저장할 디렉토리를 생성한다

- import os

- if not os.path.exists(os.path.join(folder, 'new/')):
    os.makedirs(os.path.join(folder, 'new/'))

모든 파일 크기 조절

- 파일의 크기를 조절한 뒤, 이제까지의 과정을 저장한다

- from PIL import Image

- for count, item in enumerate(file_names):

```
    img = load_img(file_names[count])
    img = img.resize((20, 20), Image.Resampling.LANCZOS)
    img.save(fp=os.path.join(folder, 'new/')+file_names[count])
```

◀ (너비, 높이) 를 20 x 20 으로 맞춤

※ Image.Resampling.LANCZOS :
LANCZOS 함수를 이용하여 이미지의 해상도를 낮추면서 생기는 계단 현상 등의 문제를 보정
이 외에 NEAREST, BILINEAR, BICUBIC, HAMMING, BOX 등이 있음

opencv

- 이미지 축소에 opencv 패키지(cv2)를 사용할 수도 있다
  - 기본적으로 아래 명령어로 설치할 수 있다
    - pip install opencv-python
    - https://pypi.org/project/opencv-python/
    - 품질이 더 좋지만, opencv에 대해 다루는 다음 과정에서 진행한다

```
import cv2

for count, item in enumerate(file_names):
    img = load_img(file_names[count])
    img_array = img_to_array(img)
    img_array = cv2.resize(img_array,(20,20))
    dataset[count] = img_array

    if count % 500 == 0:
        print("%d images to array" % count)
print("All images to array!")
```

◀ opencv 패키지를 이용한 resize

넘파이 배열로 변환

- 크기 조절된 파일을 넘파이 배열로 변환한다

load_img() → img_to_array()
PIL np.array

- for count, item in enumerate(file_names):

```
    img = load_img(os.path.join(folder, 'new/')+file_names[count])
    img_array = img_to_array(img)
    dataset[count] = img_array

    if count % 500 == 0:
        print(f"{count} images to array")
```

◀ 20 x 20 x 3으로 조절된 데이터를 같은 형상의 변수에 넣음

- print("All images to array!")

```
0 images to array
500 images to array
1000 images to array
1500 images to array
2000 images to array
2500 images to array
3000 images to array
3500 images to array
4000 images to array
4500 images to array
5000 images to array
```

변환 결과 이미지 확인

- 한 개 이미지 파일을 열어 변환 결과를 확인한다

- print(dataset.shape) # (25000, 20, 20, 3)
- print(file_labels[30]) # 0 (cat)

- display(_Imgdis(filename=os.path.join(folder, 'new/') + "/" + files[30], width=100, height=100))

cat

변환된 PIL 파일의 모습
변환된 모습을 보려면 width=20, height=20으로 보아야 하나
작아서 (100, 100)으로 늘림

변환 결과 넘파이 배열 확인

- 한 개 이미지 파일의 변환 결과 확인

- np.set_printoptions(linewidth=np.inf)
- print(dataset[30, :, :, 0])

np.array 30번 파일의 첫 번째 channel

```
[[ 19  20  51  67  80  63  86 111 107 110 110  65  68  55  51  47  50  53  47  33]
 [ 39  37  54  55  62  49  79 105 102  97  90  53  56  40  42  44  41  47  46  35]
 [ 42  46  66  59  68  59  82  96  98  89  72  51  57  46  49  52  47  53  50  40]
 [ 40  46  62  53  65  62  81  83  74  74  53  46  61  59  59  50  60  64  60  45]
 [ 49  43  42  21  34  41  66  69  55  74  53  58  71  76  72  51  52  62  62  53]
 [ 42  36  35  16  29  32  57  64  48  79  58  74  74  71  69  52  41  57  67  65]
 [ 45  46  55  39  43  36  62  78  57  89  71  99  77  52  64  76  65  78  87  81]
 [ 78  74  75  50  40  29  67  97 101 129 113 151 114  72  96 136 105 111 107  91]
 [103 109 111  91  58  72 101  75  46  48 188 195 104  60  76  89  98 111 113  97]
 [108 112 123 114  94  93  84  62  73 142 211 193 115  45  69  62  58  79  95  93]
 [104  96 111 104 104  93  51  48 109 225 222 204 145  32  56  38  34  60  83  91]
 [115 106 123 113 123 105  45  66 125 206 187 201 162  36  60  54  34  54  73  80]
 [ 93  94 114 107 122 108  54  89 207 180 168 197 146  28  48  68  27  39  55  63]
 [100 103 111 108 140 158 143 177 207 219 239 153  93 130 156  78  16  28  44  57]
 [ 90  98  90  97 147 187 194 201 180 131 181 138  55  32  33   7  11  18  30  42]
 [ 92 109 104 125 179 212 203 169  49  46 138 116  50  32  25   0   6   6   9  14]
 [133 115 155 196 196 188 127  31  69 134 108  34  30  48  26   3   8   6   5   5]
 [ 88 153 194 199 200 144  57  34  78  76  39  20  42  45  15   0   7   5   3   4]
 [ 36 127 172 152 137  77  15  45  57  34  22  36  44  30  12  11  10   8   6   6]
 [ 17  49 116 109  63  46  29  36  26  28  49  60  40  26  22  14  16  14  12  12]]
```

훈련 데이터와 테스트 데이터 분리

- from sklearn.model_selection import train_test_split

- train_images, test_images, train_labels, test_labels = train_test_split(dataset, file_labels, test_size=0.2)
- print("Train set size: {0}, Test set size: {1}".format(len(train_images), len(test_images)))

```
Train set size: 20000, Test set size: 5000
```

모델 설계

- from tensorflow.keras import models
- from tensorflow.keras import layers
- import numpy as np

- model = models.Sequential()
- model.add(layers.Conv2D(32, kernel_size=(3, 3), input_shape=(20, 20, 3), activation='relu'))
- model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
- model.add(layers.MaxPooling2D(pool_size=2))
- model.add(layers.Dropout(0.25))
- model.add(layers.Flatten())
- model.add(layers.Dense(128, activation='relu'))
- model.add(layers.Dropout(0.25))
- model.add(layers.Dense(1, activation='sigmoid'))

- model.summary()

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 18, 18, 32)        896
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496
max_pooling2d (MaxPooling2D  (None, 8, 8, 64)          0
)
dropout (Dropout)            (None, 8, 8, 64)          0
flatten (Flatten)            (None, 4096)              0
dense (Dense)                (None, 128)               524416
dropout_1 (Dropout)          (None, 128)               0
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 543,937
Trainable params: 543,937
Non-trainable params: 0
_________________________________________________________________
```

모델 컴파일

- model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['acc'])

데이터 reshape 및 스케일링

- print("before:", train_images.shape)

- train_images = train_images.reshape((len(train_images), 20, 20, 3))
- train_images = train_images.astype('float32')/255

- test_images = test_images.reshape((len(test_images), 20, 20, 3))
- test_images = test_images.astype('float32')/255

- print("after:", train_images.shape)

resize 과정을 통해 shape가 맞추어져 있으므로 reshape는 해주지 않아도 된다

데이터 확인

- train 데이터 확인

```
array([[[[0.18431373, 0.02745098, 0.        ],
         [0.23921569, 0.08627451, 0.05882353],
         [0.18431373, 0.03137255, 0.00392157],
         ...,
         [0.11372549, 0.09803922, 0.08627451],
         [0.07843138, 0.0627451 , 0.05098039],
         [0.05098039, 0.03529412, 0.02352941]],

        [[0.12156863, 0.        , 0.        ],
         [0.20784314, 0.05490196, 0.02745098],
         [0.16470589, 0.01960784, 0.        ],
         ...,
         [0.12156863, 0.10588235, 0.09411765],
         [0.05882353, 0.04313726, 0.03137255],
         [0.01960784, 0.00392157, 0.        ]],

        [[0.16470589, 0.01176471, 0.        ],
         [0.24705882, 0.09411765, 0.06666667],
         [0.18039216, 0.03529412, 0.00392157],
```

넘파이 배열로 변환

- 넘파이 배열로 변환
  - 이진 분류이므로 종속변수에 대해 원-핫 인코딩 없이 리스트를 넘파이 배열로만 변환해준다

- train_labels = np.array(train_labels)
- test_labels = np.array(test_labels)

모델 훈련

- history = model.fit(train_images, train_labels, epochs=50, batch_size=128, validation_data=(test_images, test_labels))

```
Epoch 1/50
157/157 [==============================] - 10s 18ms/step - loss: 0.6668 - acc: 0.6083 - val_loss: 0.6223 - val_acc: 0.6856
Epoch 2/50
157/157 [==============================] - 2s 15ms/step - loss: 0.5867 - acc: 0.6919 - val_loss: 0.5545 - val_acc: 0.7174
Epoch 3/50
157/157 [==============================] - 2s 15ms/step - loss: 0.5454 - acc: 0.7239 - val_loss: 0.5568 - val_acc: 0.7208
Epoch 4/50
157/157 [==============================] - 2s 15ms/step - loss: 0.5171 - acc: 0.7433 - val_loss: 0.5368 - val_acc: 0.7248
Epoch 5/50
157/157 [==============================] - 2s 15ms/step - loss: 0.4961 - acc: 0.7566 - val_loss: 0.4966 - val_acc: 0.7550
Epoch 6/50
157/157 [==============================] - 2s 15ms/step - loss: 0.4715 - acc: 0.7750 - val_loss: 0.5096 - val_acc: 0.7500
Epoch 7/50
157/157 [==============================] - 2s 15ms/step - loss: 0.4551 - acc: 0.7829 - val_loss: 0.6455 - val_acc: 0.6856
```

성능 평가

- print("Test Data Accuracy:", model.evaluate(test_images, test_labels))

```
157/157 [==============================] - 1s 5ms/step - loss: 1.0671 - acc: 0.7730
Test Data Accuracy: [1.0671008825302124, 0.7730000019073486]
```

예측 결과

- model.predict(test_images)

```
array([[9.9890733e-01],
       [4.9766142e-05],
       [4.3592080e-03],
       ...,
       [9.9291819e-01],
       [1.9573957e-02],
       [2.0116819e-02]], dtype=float32)
```

정확도 확인

- acc = history.history['acc']
- val_acc = history.history['val_acc']
- loss = history.history['loss']
- val_loss = history.history['val_loss']

- print('Accuracy of each epoch:', acc)
- print()
- print('Validation Accuracy of each epoch:', val_acc)

※ 70%가 채 되지 않았다

```
Accuracy of each epoch: [0.608299970626831, 0.6918500065803528, 0.7239000201225281, 0.7433000206947327, 0.756600022315979, 0.7749500274
658203, 0.7829499840736389, 0.7954999804496765, 0.8088499903678894, 0.8169000148773193, 0.827750027179718, 0.8410500288009644, 0.851100
0275611877, 0.8640499711036682, 0.8694999814033508, 0.8835999965667725, 0.8919500112533569, 0.9014999866485596, 0.907800018787384, 0.91
75500273704529, 0.9204000234603882, 0.9284499883651733, 0.9359999895095825, 0.9411500096321106, 0.9451500177383423, 0.9475499987602234,
0.9547500014305115, 0.9563000202178955, 0.9564999938011169, 0.9606000185012817, 0.9609500169754028, 0.9625499844551086, 0.9664499759674
072, 0.9660000205039978, 0.9684000015258789, 0.968500018119812, 0.9692999720573425, 0.9704999923706055, 0.9696499705314636, 0.970749974
2507935, 0.9731000065803528, 0.972000002861023, 0.9745000004768372, 0.972599983215332, 0.9739000201225281, 0.973550021648407, 0.9746999
740600586, 0.9742000102996826, 0.9719499945640564, 0.9722999930381775]

Validation Accuracy of each epoch: [0.6855999827384949, 0.7174000144004822, 0.72079998254776, 0.7247999906539917, 0.7549999952316284,
0.75, 0.6855999827384949, 0.7522000074386597, 0.7558000087738037, 0.7576000094413757, 0.7580000162124634, 0.7595999836921692, 0.7743999
95803833, 0.7598000168800354, 0.7675999999046326, 0.7590000033378601, 0.7581999897956848, 0.7639999985694885, 0.7681999802589417, 0.756
7999958992004, 0.7580000162124634, 0.756600022315979, 0.7509999871253967, 0.7699999809265137, 0.7710000276565552, 0.7630000114440918,
0.7721999883651733, 0.7717999815940857, 0.7634000182151794, 0.7699999809265137, 0.7458000183105469, 0.7681999802589417, 0.7743999958038
33, 0.7638000249862671, 0.7689999938011169, 0.7728000283241272, 0.7631999850273132, 0.7441999912261963, 0.7590000033378601, 0.766600012
7792358, 0.7698000073432922, 0.7731999754905701, 0.7631999850273132, 0.7692000269889832, 0.7630000114440918, 0.7562000155448914, 0.7689
999938011169, 0.7749999761581421, 0.7644000053405762, 0.7731999754905701]
```

손실값 확인

- print('Loss of each epoch:', np.round(loss, 3))
- print()
- print('Validation Loss of each epoch:', np.round(val_loss, 3))

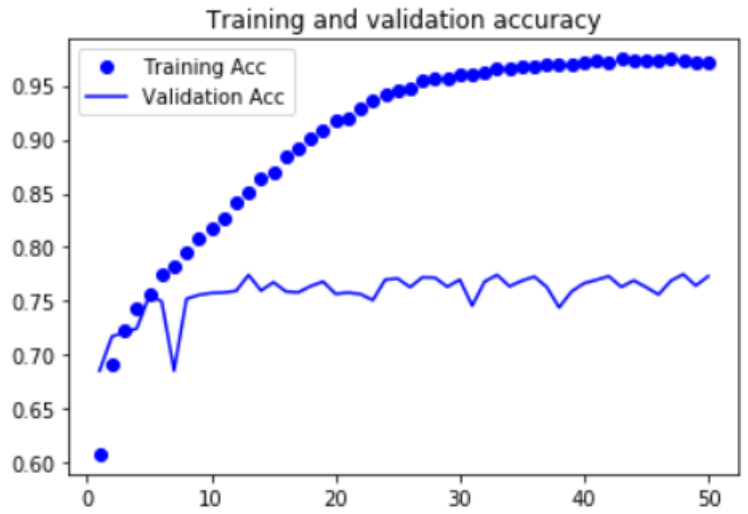
```
Loss of each epoch: [0.667 0.587 0.545 0.517 0.496 0.471 0.455 0.44  0.415 0.401 0.385 0.363 0.341 0.317 0.304 0.281 0.261 0.242 0.226
0.209 0.196 0.183 0.163 0.156 0.148 0.14  0.125 0.12  0.117 0.111 0.104 0.104 0.094 0.097 0.089 0.088 0.088 0.086 0.085 0.085 0.078 0.0
79 0.075 0.081 0.076 0.077 0.077 0.082 0.08  0.08 ]

Validation Loss of each epoch: [0.622 0.555 0.557 0.537 0.497 0.51  0.645 0.505 0.514 0.517 0.512 0.525 0.503 0.523 0.516 0.535 0.588
0.565 0.578 0.556 0.679 0.638 0.733 0.7   0.703 0.672 0.698 0.692 0.655 0.79  0.778 0.866 0.788 0.858 0.905 0.953 0.709 0.669 0.911 0.8
34 0.853 1.04  0.726 1.134 0.988 1.057 0.897 1.24  1.013 0.95 ]
```

그래프 확인

- import matplotlib.pyplot as plt

- epochs = range(1, len(acc) +1)

- plt.plot(epochs, acc, 'bo', label='Training Acc')
- plt.plot(epochs, val_acc, 'b', label='Validation Acc')
- plt.title('Training and validation accuracy')
- plt.legend()



그래프 확인

- plt.figure()

- plt.plot(epochs, loss, 'bo', label='Training Loss')
- plt.plot(epochs, val_loss, 'b', label='Validation Loss')
- plt.title('Training and validation loss')
- plt.legend()