이미지 데이터 증강

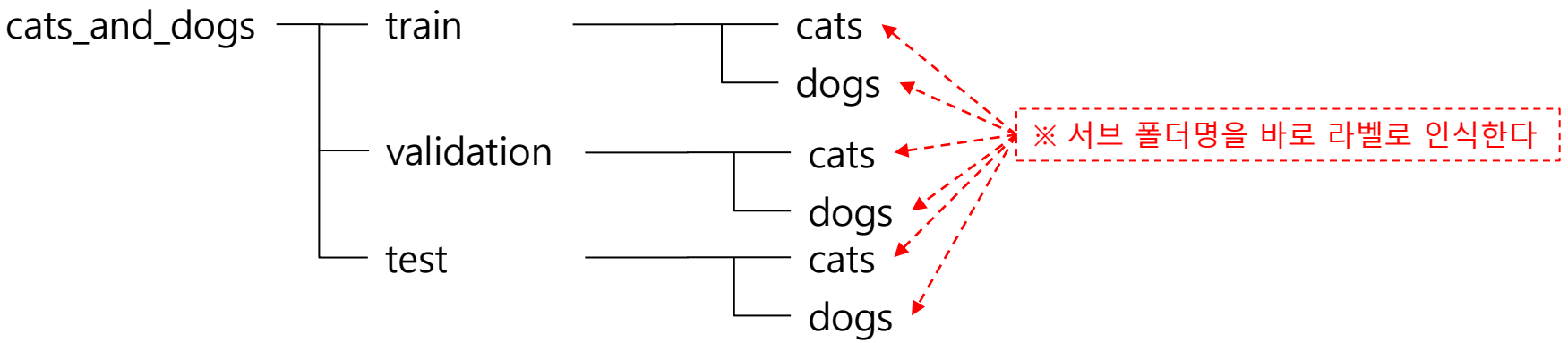
최석재 *lingua@naver.com*

Image Data Generator

ImageDataGenerator

- ImageDataGenerator
  - 이미지 처리를 쉽게 하는 데 쓰이는 Keras 클래스
  - 스케일링, 리사이징, 데이터 증강 등이 이 클래스로 가능함

  - 여러 가지 일을 쉽게 처리하도록 해놓았으나, 사용 방법에 제약이 있음
  - 파일을 읽어들이는데 자주 사용되는 flow_from_directory 메소드는 서브 폴더명을 분류 라벨로 파악하므로 폴더를 이에 맞게 준비해야 함
  - 이 외에 모델 훈련에 사용되는 .fit() 의 사용법도 기존과 다름



구글 드라이브 연결

- 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

객체 생성 및 경로 설정

- from tensorflow.keras.preprocessing.image import ImageDataGenerator

- train_datagen = ImageDataGenerator(rescale=1./255) # 객체를 만들 때 스케일링 수준을 결정
- validation_datagen = ImageDataGenerator(rescale=1./255)
- test_datagen = ImageDataGenerator(rescale=1./255)

- folder = '/content/gdrive/MyDrive/pytest_img/cats_dogs'
- train_dir = folder+"/train"
- validation_dir = folder+"/validation"
- test_dir = folder+"/test"

※ cats_dogs 하위 폴더에 train, test, validation 폴더를 만들어두고,
그 하위에는 다시 cats와 dogs 폴더가 있어야 한다
full data는 _original_jpg_backup 폴더에 있음

데이터 주입

※ target_size: resize될 이미지의 (높이, 너비)
batch_size: 한 번에 훈련될 샘플의 수
batch_size x steps_per_epoch = 전체 데이터 수
class_mode: 이진분류에서는 binary,
다중분류에서는 categorical(원-핫인코딩 포함) 또는
sparse(원-핫인코딩 없는 정수 레이블)
classes: 레이블 0, 1의 순서. ['label1', 'label2']은 차례대로 0과 1.

- train_generator = train_datagen.flow_from_directory(
  train_dir, target_size=(20, 20), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

size 조절이 간단히 이루어진다

- validation_generator = validation_datagen.flow_from_directory(
  validation_dir, target_size=(20, 20), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

train data는 2000건, validation과 test 데이터는 1000건 씩이다

- test_generator = test_datagen.flow_from_directory(
  test_dir, target_size=(20, 20), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

steps_per_epoch: 한 에포크 동안 모델이 학습할 배치의 수

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 및 훈련

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

- history = model.fit(
    train_generator,
    steps_per_epoch=20,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=10)

※ steps_per_epoch: 한 에포크 동안 모델이 학습할 배치의 개수
batch_size x steps_per_epoch ≒ 전체 데이터 수
epochs: 전체 에포크 횟수
validataion_data: 제너레이터로 생성된 검증 데이터
validataion_steps: 검증을 위해 한 에포크에 추출되는 배치의 개수
batch_size x validataion_steps ≒ 전체 데이터 수

독립변수와 종속변수 대신 generator를 넣는다
하나의 generator는 독립변수와 종속변수를 갖고 있음

steps_per_epoch와 validation_steps는
앞에서 설정한 batch_size와 곱했을 때 전체 데이터 수와 비슷하게 한다

train data는 2000건, validation과 test 데이터는 1000건 씩이다

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)

```
Accuracy of each epoch: [0.5074999928474426, 0.5705000162124634, 0.5989999771118164, 0.6445000171661377, 0.6740000247955322, 0.69499999
28474426, 0.7250000238418579, 0.7415000200271606, 0.7509999871253967, 0.7789999842643738, 0.777999997138977, 0.7994999885559082, 0.812
5, 0.8100000023841858, 0.8379999995231628, 0.8364999890327454, 0.8550000190734863, 0.8690000176429749, 0.8755000233650208, 0.8974999785
423279, 0.8880000114440918, 0.9020000100135803, 0.9075000286102295, 0.9100000262260437, 0.9100000262260437, 0.9304999709129333, 0.93150
00176429749, 0.9315000176429749, 0.9459999799728394, 0.9399999976158142]

Validation Accuracy of each epoch: [0.5, 0.6010000109672546, 0.5, 0.6200000047683716, 0.6729999780654907, 0.6349999904632568, 0.6759999
990463257, 0.6679999828338623, 0.6620000004768372, 0.6959999799728394, 0.6800000071525574, 0.6949999928474426, 0.6179999709129333, 0.67
79999732971191, 0.6869999766349792, 0.6520000100135803, 0.6710000038146973, 0.671999990940094, 0.656000018119812, 0.6729999780654907,
0.6679999828338623, 0.6809999942779541, 0.6420000195503235, 0.6830000281333923, 0.6840000152587891, 0.6830000281333923, 0.6729999780654
907, 0.6790000200271606, 0.6729999780654907, 0.6639999747276306]
```

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: [0.717 0.683 0.668 0.649 0.612 0.606 0.562 0.529 0.516 0.472 0.486 0.439
 0.418 0.416 0.366 0.376 0.34  0.322 0.302 0.273 0.28  0.248 0.235 0.232
 0.228 0.184 0.181 0.188 0.152 0.163]

Validation Loss of each epoch: [0.696 0.675 0.849 0.648 0.628 0.644 0.604 0.627 0.621 0.602 0.607 0.593
 0.71  0.605 0.635 0.668 0.656 0.682 0.684 0.744 0.675 0.686 0.881 0.717
 0.705 0.753 0.807 0.765 0.847 0.824]
```

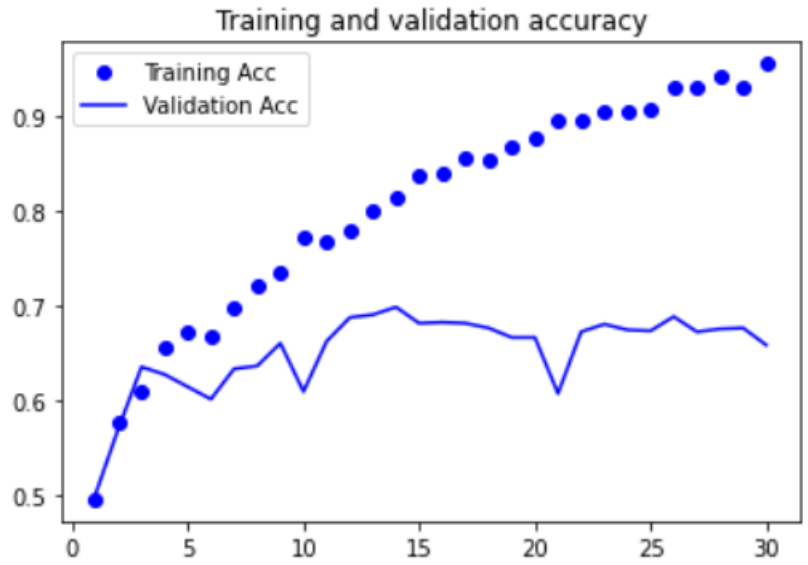
그래프 확인

- 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()

- plt.show()

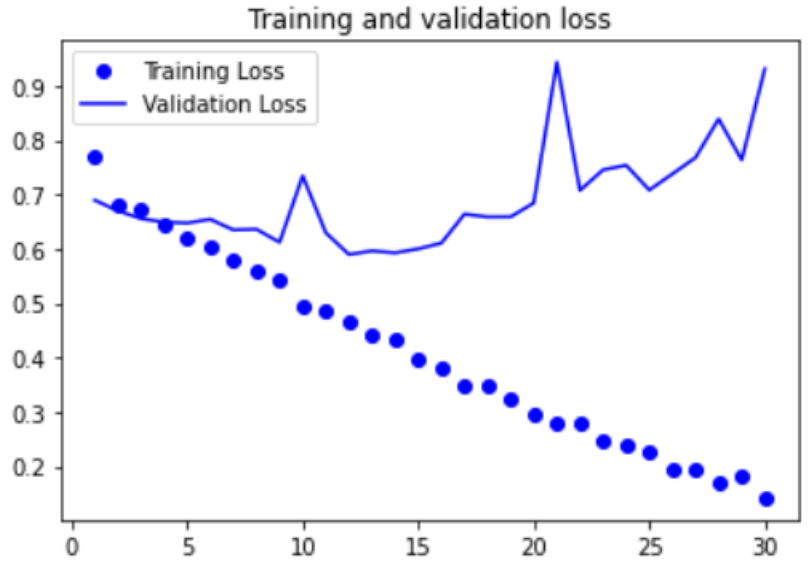


그래프 확인

- 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()

- plt.show()



예측 결과

- model.predict(test_generator)

◀ 제너레이터를 입력한다

```
array([[9.92391646e-01],
       [5.24865031e-01],
       [2.60951538e-02],
       [9.80516374e-01],
       [1.31631106e-01],
       [9.86417770e-01],
       [6.46064460e-01],
       [9.76619184e-01],
       [3.71158242e-01],
       [6.82823122e-01],
       [1.90818265e-01],
       [9.52107430e-01],
       [7.88021505e-01],
       [9.95823622e-01],
       [8.13408613e-01],
       [9.35868979e-01],
       [1.12327449e-02],
       [9.95020986e-01],
       [5.76292038e-01],
```

모델의 성능 평가도 같은 방식으로 진행

model.evaluate(test_generator)

참고 : Image Dataset From Directory

- Tensorflow 2.0에서 이미지 처리를 위한 새로운 도구로 제시
- ImageDataGenerator를 이용하는 것보다 더 빠른 것이 장점

- 그러나 rescaling, 데이터 증강 등 많은 도구들을 지원하지 않는다
- 필요한 기능들은 별도로 구현 또는, 다른 클래스의 기능을 사용해야 한다
  (예: rescaling은 layers.Rescaling, 데이터 증강은 layers.RandomFlip/RandomRotation 등)

- ImageDataGenerator와 동일하게 서브 폴더명을 분류 라벨로 파악하므로
  데이터는 동일한 방식으로 분류되어 있어야 한다
- 데이터 주입 이후에는 ImageDataGenerator와 동일하게 사용하면 된다
  (즉, 모델 설계 부분부터는 동일하게 진행)

데이터 주입

- from tensorflow.keras.utils import image_dataset_from_directory

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

- train_dir = folder+"/train"
- validation_dir = folder+"/validation"
- test_dir = folder+"/test"

- train_dataset = image_dataset_from_directory(directory=train_dir, image_size=(20, 20), batch_size=100)
- validation_dataset = image_dataset_from_directory(directory=validation_dir, image_size=(20, 20), batch_size=100)
- test_dataset = image_dataset_from_directory(directory=test_dir, image_size=(20, 20), batch_size=100)

※ 이후 모델 설계 부분부터는 ImageDataGenerator와 동일하게 진행한다

데이터 증강

데이터 증강

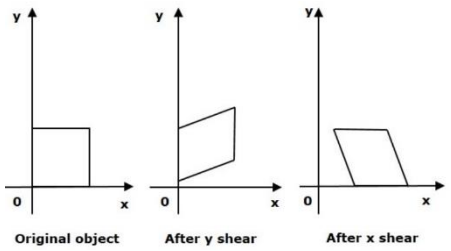
- 데이터 증강
  - 훈련 데이터가 부족할 때 기존의 데이터를 약간 변형하여 늘리는 방법
  - 같은 이미지를 회전시켜 모델이 데이터의 다양한 측면을 학습하게 함
  - 똑같은 데이터가 아니기에 과적합되지 않으면서 일반화를 이룰 수 있음
  - 데이터의 양이 적을 때 사용하면 일반적으로 모델의 성능이 향상됨

  - ImageDataGenerator를 통해 쉽게 구현할 수 있음

한 개 이미지 증강 연습

- import os
- from tensorflow.keras.preprocessing.image import ImageDataGenerator

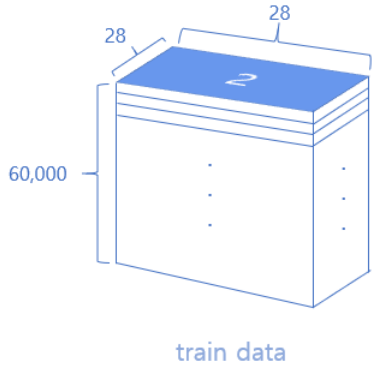
- datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

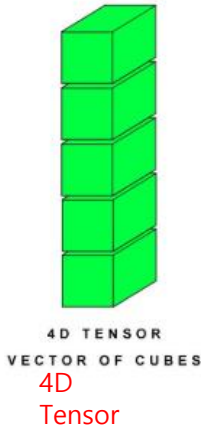


- rescale: 스케일링 수준
- rotation_range: 사진을 회전시킬 각도의 범위
- width_shift_range: 수평으로 평행 이동 시킬 범위 (넓이의 비율)
- height_shift_range: 수직으로 평행 이동 시킬 범위 (높이의 비율)
- shear_range: rotation_range로 회전할 때 증가시킬 y축 방향의 각도
- zoom_range: 사진을 확대할 범위
- horizontal_flip: 좌우 반전

※ 상하반전의 vertical_flip도 있으나, 성능이 좋아지지 않을 가능성이 있다
글씨가 있는 경우에는 좌우반전이나 상하반전을 해서는 안된다

파일 불러오기





- train_cats_dir = folder+"/train/cats/"
- file = os.listdir(train_cats_dir)[0]
- file_path = os.path.join(train_cats_dir, file)
- print(file_path)

- from tensorflow.keras.preprocessing import image
- img = image.load_img(file_path, target_size=(120, 120))
- x = image.img_to_array(img)
- x = x.reshape(1, 120, 120, 3)
- print(x.shape) # (1, 120, 120, 3)

load_img() 함수는 이미지를 PIL(Python Image Library) 파일로 변환한다
img_to_array() 함수는 PIL 파일을 넘파이 배열로 변환한다

앞에 축을 하나 추가한다

1행, 120 x 120 사이즈의 RGB 데이터

flow()를 이용한 데이터 증강

- import matplotlib.pyplot as plt

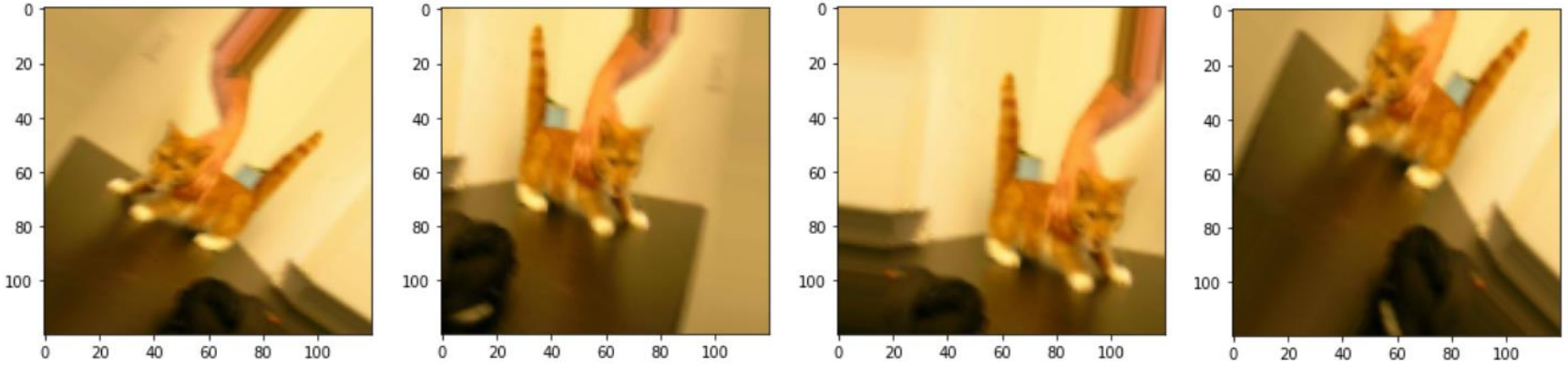
ImagaDataGenerator.flow(x, batch_size=1) 으로 데이터를 하나씩 넣는다
array_to_img() 함수는 넘파이 배열 상태로 되어 있는 것을 다시 PIL 타입 이미지로 변환

- i = 0
- for batch in datagen.flow(x, batch_size=1):

```
    plt.figure(i)                                  # i 번째 그림을 그릴 준비를 한다
    plt.imshow(image.array_to_img(batch[0]))   # 이미지 생성
    i += 1
    if i%4 == 0:
        break
```

- plt.show() # 출력

4장씩 만든다
flow() 함수를 이용하면 이와 같이 원하는 양의 데이터 증강이 가능하다



flow_from_directory()를 이용한 데이터 증강

- 대량의 이미지를 처리할 때는 flow_from_directory()를 사용한다

- train_datagen = ImageDataGenerator(
  rescale=1./255,
  rotation_range=40,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True)

train 데이터만 증강

※ validation과 test 데이터는 증강을 하면 안된다
※ 여기서는 flow_from_directory()를 이용할 것이며,
한 batch 마다 하나의 증강된 이미지가 생성된다 (batch_size 100인 경우, 100장 당 1장)
따라서 2000 data / 100 batch_size = 20 aug. images
100 epoch면, 20 aug. images x 100 epochs = 2000 aug. images

- validation_datagen = ImageDataGenerator(rescale=1./255)
- test_datagen = ImageDataGenerator(rescale=1./255)

데이터 주입

- train_dir = folder+"/train/"
- test_dir = folder+"/test/"
- validation_dir = folder+"/validation/"

- train_generator = train_datagen.flow_from_directory(
  train_dir, target_size=(20, 20), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

이진분류에서는 binary
다중분류에서는 categorical

- validation_generator = validation_datagen.flow_from_directory(
  validation_dir, target_size=(20, 20), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

- test_generator = test_datagen.flow_from_directory(
  test_dir, target_size=(20, 20), batch_size=100, class_mode='binary',
  classes=['cats', 'dogs'])

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

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.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 및 훈련

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

- history = model.fit(
    train_generator,
    steps_per_epoch=20,
    epochs=100,
    validation_data=validation_generator,
    validation_steps=10)

```
Epoch 1/100
20/20 [==============================] - 11s 417ms/step - loss: 0.7293 - acc: 0.5040 - val_loss: 0.6895 - val_acc: 0.5060
Epoch 2/100
20/20 [==============================] - 9s 432ms/step - loss: 0.6937 - acc: 0.5165 - val_loss: 0.6832 - val_acc: 0.6090
Epoch 3/100
20/20 [==============================] - 8s 410ms/step - loss: 0.6906 - acc: 0.5355 - val_loss: 0.6834 - val_acc: 0.5390
Epoch 4/100
20/20 [==============================] - 9s 432ms/step - loss: 0.6820 - acc: 0.5730 - val_loss: 0.6574 - val_acc: 0.6210
Epoch 5/100
20/20 [==============================] - 8s 412ms/step - loss: 0.6804 - acc: 0.5735 - val_loss: 0.6699 - val_acc: 0.5490
Epoch 6/100
20/20 [==============================] - 8s 408ms/step - loss: 0.6641 - acc: 0.6040 - val_loss: 0.6402 - val_acc: 0.6360
Epoch 7/100
```

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)

```
Accuracy of each epoch: [0.5040000081062317, 0.5164999961853027, 0.5354999899864197, 0.5730000138282776, 0.5734999775886536, 0.6039999723434448, 0.6134999990463257, 0.6154999732971191, 0.621999979019165, 0.6340000033378601, 0.6395000219345093, 0.6399999856948853, 0.6560000181198112, 0.637499988079071, 0.6549999713897705, 0.6464999914169312, 0.6504999995231628, 0.6875, 0.6729999780654907, 0.6690000295639038, 0.6700000166893005, 0.6759999990463257, 0.6629999876022339, 0.6735000014305115, 0.6725000143051147, 0.6740000247955322, 0.6850000023841858, 0.684499979019165, 0.6654999852180481, 0.6869999766349792, 0.6875, 0.6970000267028809, 0.684499979019165, 0.6859999895095825, 0.7020000219345093, 0.6905000209808350, 0.7089999914169312, 0.7055000066757202, 0.6955000162124634, 0.6880000233650208, 0.6984999775886536, 0.7064999938011169, 0.6924999952316284, 0.7049999833106995, 0.6949999928474426, 0.7049999833106995, 0.7110000252723694, 0.7195000052452087, 0.7080000042915344, 0.7179999947547913, 0.7174999713897705, 0.6899999976158142, 0.7080000042915344, 0.7095000147819519, 0.7070000171661377, 0.7145000100135803, 0.7214999794960022, 0.7105000019073486, 0.7049999833106995, 0.7174999713897705, 0.7225000262260437, 0.7185000181198112, 0.7235000133514404, 0.7264999747276306, 0.7210000157356262, 0.7179999947547913, 0.7160000205039978, 0.7340000271797188, 0.7260000109672546, 0.7185000181198112, 0.7220000028610230, 0.7379999756813049, 0.7099999785423279, 0.7264999747276306, 0.7225000262260437, 0.7264999747276306, 0.7325000166893005, 0.7185000181198112, 0.7315000295639038, 0.7455000281333923, 0.7325000166893005, 0.7335000038146973, 0.7264999747276306, 0.7425000071525574, 0.7294999957084656, 0.7329999804496765, 0.7450000047683716, 0.7400000095367432, 0.7315000295639038, 0.7404999732971191, 0.7429999709129333, 0.7419999837875366, 0.7484999895095825, 0.7365000247955322, 0.7304999828338623, 0.7455000281333923, 0.7465000152587891, 0.7505000233650208, 0.7329999804496765, 0.7509999871253967]

Validation Accuracy of each epoch: [0.5059999823570251, 0.609000027179718, 0.5389999747276306, 0.6209999918937683, 0.5490000247955322, 0.6359999775886536, 0.5709999799728394, 0.6610000133514404, 0.6200000047683716, 0.6650000214576721, 0.6740000247955322, 0.6729999780654907, 0.6179999709129333, 0.621999979019165, 0.5960000157356262, 0.5749999880790710, 0.6549999713897705, 0.6589999794960022, 0.6610000133514404, 0.6880000233650208, 0.5870000123977661, 0.5720000267028809, 0.6200000047683716, 0.6579999923706055, 0.6380000114440918, 0.6919999718666077, 0.6629999876022339, 0.6169999837875366, 0.5680000185966492, 0.5970000028610230, 0.6359999775886536, 0.621999979019165, 0.6389999985694885, 0.6700000166893005, 0.7089999914169312, 0.6809999942779541, 0.6029999852180481, 0.6430000066757202, 0.6539999842643738, 0.7080000042915344, 0.6729999780654907, 0.6669999957084656, 0.6970000267028809, 0.6119999885559082, 0.6029999852180481, 0.6690000295639038, 0.7120000123977661, 0.6880000233650208, 0.6869999766349792, 0.6309999823570251, 0.6620000004768372, 0.6840000152587891, 0.6179999709129333, 0.6079999804496765, 0.7070000171661377, 0.6489999890327454, 0.6539999842643738, 0.6949999928474426, 0.6259999871253967, 0.6660000085830688, 0.6259999871253967, 0.6629999876022339, 0.6980000138282776, 0.6039999723434448, 0.6029999852180481, 0.6740000247955322, 0.6719999940940094, 0.6639999747276306, 0.6520000100135803, 0.7310000061988831, 0.6769999861717224, 0.5830000042915344, 0.7179999947547913, 0.6520000100135803, 0.6669999957084656, 0.6899999976158142, 0.7110000252723694, 0.6539999842643738, 0.7110000252723694, 0.6669999957084656, 0.6909999847412109, 0.7120000123977661, 0.6690000295639038, 0.6830000281333923, 0.6209999918937683, 0.6470000147819519, 0.6890000104904175, 0.6639999747276306, 0.7160000205039978, 0.6729999780654907, 0.7129999995231628, 0.6830000281333923, 0.6380000114440918, 0.7229999899864197, 0.6719999940940094, 0.7080000042915344, 0.6320000290870667, 0.7129999995231628, 0.7179999947547913, 0.7030000090599606]
```

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: [0.729 0.694 0.691 0.682 0.68  0.664 0.665 0.66  0.646 0.644 0.644 0.636
 0.63  0.631 0.616 0.624 0.623 0.601 0.605 0.605 0.606 0.598 0.609 0.601
 0.6   0.583 0.595 0.595 0.601 0.585 0.586 0.585 0.583 0.588 0.574 0.578
 0.573 0.578 0.579 0.581 0.578 0.575 0.576 0.566 0.571 0.573 0.561 0.558
 0.567 0.551 0.563 0.568 0.562 0.556 0.556 0.563 0.552 0.551 0.56  0.55
 0.554 0.545 0.555 0.542 0.548 0.553 0.546 0.536 0.54  0.543 0.548 0.524
 0.554 0.539 0.545 0.541 0.538 0.54  0.536 0.527 0.53  0.53  0.537 0.514
 0.53  0.537 0.527 0.523 0.526 0.521 0.515 0.514 0.511 0.517 0.54  0.516
 0.514 0.519 0.547 0.501]

Validation Loss of each epoch: [0.689 0.683 0.683 0.657 0.67  0.64  0.655 0.626 0.654 0.616 0.614 0.614
 0.635 0.637 0.657 0.683 0.611 0.606 0.612 0.601 0.699 0.76  0.648 0.609
 0.649 0.596 0.626 0.703 0.879 0.743 0.67  0.757 0.679 0.606 0.595 0.618
 0.701 0.642 0.636 0.577 0.606 0.618 0.611 0.749 0.763 0.634 0.593 0.608
 0.614 0.687 0.67  0.612 0.713 0.704 0.585 0.652 0.658 0.603 0.711 0.667
 0.702 0.651 0.619 0.729 0.78  0.642 0.669 0.703 0.689 0.579 0.661 0.867
 0.57  0.718 0.636 0.659 0.571 0.69  0.572 0.733 0.656 0.577 0.691 0.633
 0.695 0.764 0.641 0.679 0.613 0.666 0.606 0.656 0.773 0.598 0.656 0.632
 0.798 0.603 0.586 0.645]
```

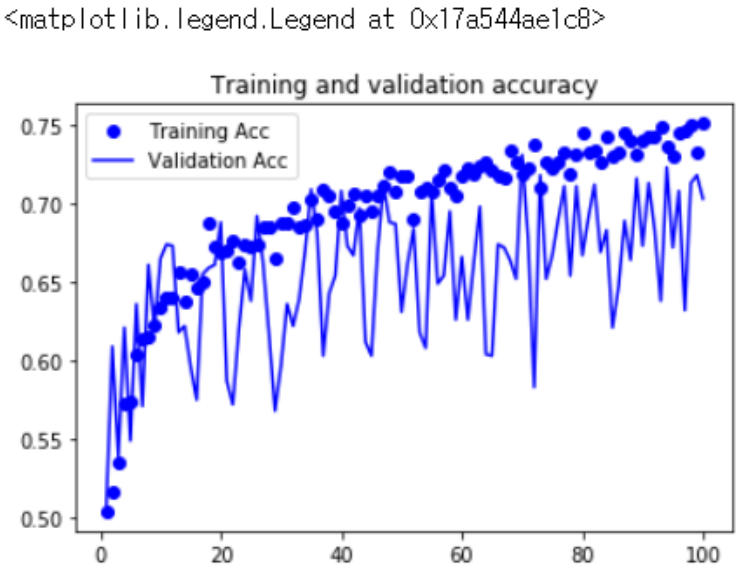
그래프 확인

- 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()

- plt.show()

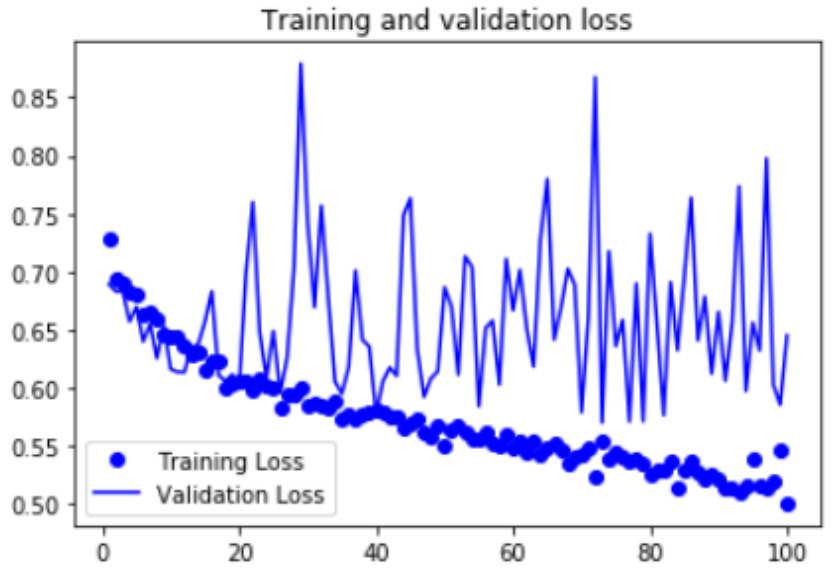


그래프 확인

- 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()

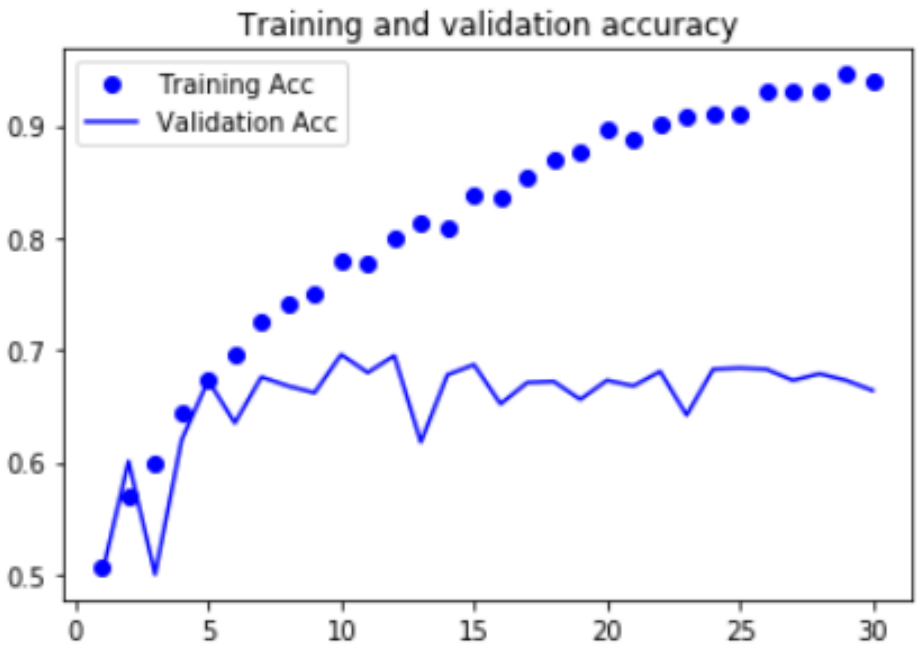
- plt.show()



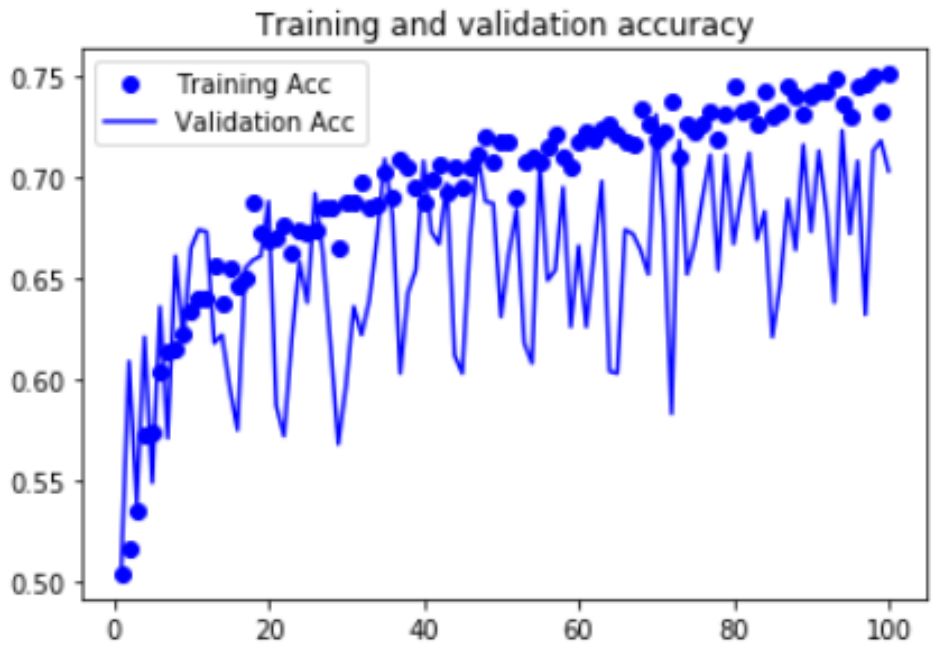
그래프 비교

- 데이터 증강 전과 후 비교

데이터 증강 전



데이터 증강 후



validation 데이터 기준으로 epoch 5 이후에도 성능이 발전 추세에 있다
그러나 에포크에 따른 편차가 크다

데이터 증강

- epoch=100
  - epoch마다 20 images가 만들어지므로 계속 성능이 향상되는 모습을 보여줌
  - batch_size를 줄여 한 epoch마다 만들어지는 이미지의 양을 늘릴 수도 있음
    (현재 20 augmentated images x 100 epochs = 2000 aug. images)

테스트 데이터 예측

- model.predict(test_generator)

```
array([[0.86577374],
       [0.41578248],
       [0.78163946],
       [0.6870457 ],
       [0.78610504],
       [0.48902673],
       [0.4969166 ],
       [0.9113179 ],
       [0.51031196],
       [0.7385125 ],
       [0.7231395 ],
       [0.40316844],
       [0.8306495 ]
```

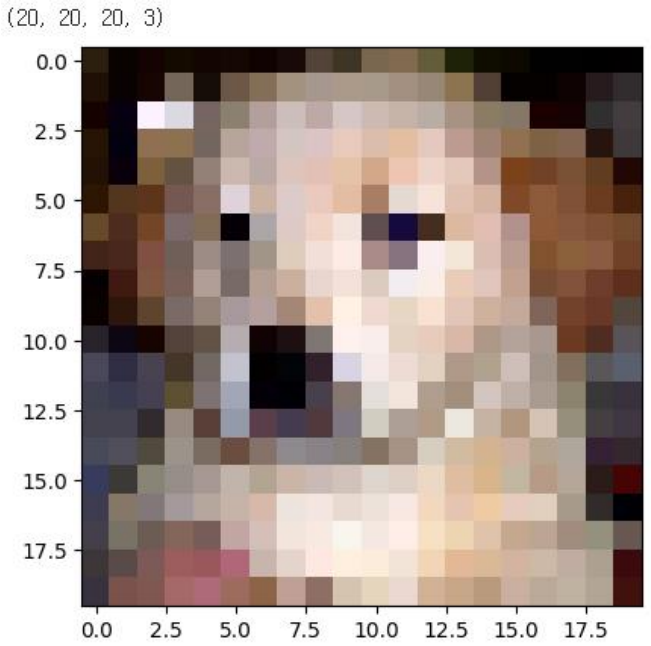
모델의 성능 평가도 같은 방식으로 진행

model.evaluate(test_generator)

참고: ImageDataGenerator 이미지 접근

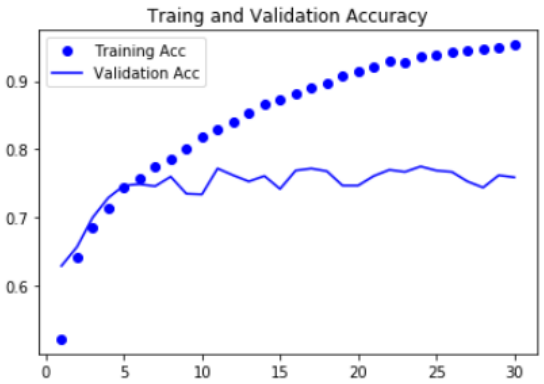
- import matplotlib.pyplot as plt

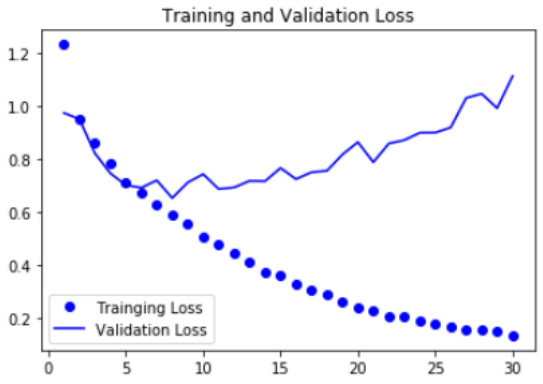
- for _ in range(len(test_generator)):
  img, label = test_generator.next()
  print(img.shape)
  plt.imshow(img[0]) # batch의 첫 번째
  plt.show()



연습문제 1

- 자연 풍경 이미지 분류하기
  - pytest_img > _original_jpg_backup > Intel_Image_Classification 폴더에는 자연 풍경 이미지 압축파일이 있다
    - buildings, forest, glacier, mountain, sea, street 이미지가 폴더명을 라벨로 하여 분류된다
  - 압축 파일을 풀어 seg_train 데이터로 모델을 만들고, seg_test 데이터로 성능 그래프 출력
    - seg_test 데이터를 validation data와 test data 모두로 사용한다





※ ImageDataGenerator.flow_from_directory() 사용
※ 데이터 증강은 하지 않음
※ 파일이 많으므로 Local PC에서 진행
단, pytest_img > Intel_image_Classification 폴더에 소량의 데이터가 있음

연습문제 2

- 자연 풍경 이미지 분류하기

buildings_2
buildings_5
buildings_9
buildings_11
buildings_13
buildings_14
buildings_20
buildings_25
buildings_53
buildings_58
buildings_60
buildings_63
buildings_68
buildings_70
buildings_80

street

mountain

buildings

- seg_pred 폴더에는 분류가 되어 있지 않은 파일들이 있다

① 각각의 파일을 읽고, 예측하여 "seg_new" 라는 폴더를 만들어 예측된 분류명을 파일명으로 하여 이미지 파일을 생성하시오 (예: buildings_2.jpg)

② 예측된 분류명과 이미지를 10개 출력하시오.
단, 전처리 과정에서 이미지의 사이즈가 줄어드니, 출력 이미지는 seg_pred 폴더의 이미지로 하여 잘 보이게 하시오

※ ImageDataGenerator.flow_from_directory()가 아닌, load_img()를 사용
※ 데이터 증강은 하지 않음
※ 파일이 많으므로 Local PC에서 진행
단, pytest_img > Intel_image_Classification 폴더에 소량의 데이터가 있음