강아지 vs 고양이 분류하기(2)

학습 내용

- 신경망을 구성 후, 학습을 수행합니다.
- 모델을 저장하는 불러오는 것에 대해 알아봅니다.
- 데이터에 대해 데이터 증식 부분을 추가합니다.

환경

- Google Colab

데이터(강아지 vs 고양이)

- url : https://www.kaggle.com/c/dogs-vs-cats/data (https://www.kaggle.com/c/dogs-vs-cats/data)
- test1.zip : 271.15MB
- train.zip : 543.16MB
- sampleSubmission.csv

목차

01. 데이터 준비하기

목차로 이동하기







- 25,000개의 강아지와 고양이 이미지
- 클래스마다 12,500개를 담고 있다. 압축(543MB크기)
- 클래스마다 1,000개의 샘플로 이루어진 훈련 세트
- 클래스마다 500개의 샘플로 이루어진 검증 세트
- 클래스마다 500개의 샘플로 이루어진 테스트 세트

In [1]:

```
import os, shutil
```

구글 드라이브 연동하기

- 링크가 뜨면 해당 링크로 연결하여 연결 암호 정보를 빈칸에 넣어준다.

In [2]:

```
### 드라이브 마운트
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.
mount("/content/drive", force_remount=True).
```

In [3]:

```
# 데이터 셋 복사 및 확인
!cp -r '/content/drive/My Drive/dataset/cats_dogs'  '/content/'
!ls -ls '/content/cats_dogs'
```

```
total 833952
    88 -rw------- 1 root root     88903 Jul 17 15:18 sampleSubmission.csv
277660 -rw------- 1 root root 284321224 Jul 17 15:18 test1.zip
556204 -rw------- 1 root root 569546721 Jul 17 15:18 train.zip
```

압축풀기를 위한 명령어

```
!rm -rf '/content/datasets/'
!unzip '/content/cats_dogs/cats_and_dogs_test1.zip' -d '/content/datasets/'
!unzip '/content/cats_dogs/cats_and_dogs_train.zip' -d '/content/datasets/'
```

In [5]:

```
!rm -rf '/content/datasets/'
!unzip '/content/cats_dogs/test1.zip' -d '/content/datasets/'
!unzip '/content/cats_dogs/train.zip' -d '/content/datasets/'
```

```
스트리밍 출력 내용이 길어서 마지막 5000줄이 삭제되었습니다.
  inflating: /content/datasets/train/dog.5499.jpg
  inflating: /content/datasets/train/dog.55.jpg
  inflating: /content/datasets/train/dog.550.jpg
  inflating: /content/datasets/train/dog.5500.jpg
  inflating: /content/datasets/train/dog.5501.jpg
  inflating: /content/datasets/train/dog.5502.jpg
  inflating: /content/datasets/train/dog.5503.jpg
  inflating: /content/datasets/train/dog.5504.jpg
  inflating: /content/datasets/train/dog.5505.jpg
  inflating: /content/datasets/train/dog.5506.jpg
  inflating: /content/datasets/train/dog.5507.jpg
  inflating: /content/datasets/train/dog.5508.jpg
  inflating: /content/datasets/train/dog.5509.jpg
  inflating: /content/datasets/train/dog.551.jpg
  inflating: /content/datasets/train/dog.5510.jpg
  inflating: /content/datasets/train/dog.5511.jpg
  inflating: /content/datasets/train/dog.5512.jpg
  inflating: /content/datasets/train/dog.5513.jpg
```

파일 확인

In [6]:

```
!ls -al '/content/datasets/train' | head -5
!ls -l '/content/datasets/train' | grep ^- | wc -l
!ls -al '/content/datasets/test1' | head -5
!ls -l '/content/datasets/test1' | grep ^- | wc -l
```

```
total 609260
drwxr-xr-x 2 root root 770048 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Jul 17 15:19 ..
-rw-r--r-- 1 root root  12414 Sep 20  2013 cat.0.jpg
-rw-r--r-- 1 root root  21944 Sep 20  2013 cat.10000.jpg
25000
total 304260
drwxr-xr-x 2 root root 270336 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Jul 17 15:19 ..
-rw-r--r-- 1 root root  54902 Sep 20  2013 10000.jpg
-rw-r--r-- 1 root root  21671 Sep 20  2013 10001.jpg
12500
```

In [7]:

```
# 원본 데이터셋을 압축 해제한 디렉터리 경로
ori_dataset_dir = './datasets/train'

# 소규모 데이터셋을 저장할 디렉터리
base_dir = './datasets/cats_and_dogs_small'

# 여러번 반복실행을 위해 디렉터리 삭제
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)
os.mkdir(base_dir)

# 훈련, 검증, 테스트 분할을 위한 디렉터리
train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)

val_dir = os.path.join(base_dir, 'validation')
os.mkdir(val_dir)

test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

os.listdir(base_dir)
```

Out[7]:

```
['validation', 'test', 'train']
```

In [9]:

```
import os
for dirname, _, filenames in os.walk(base_dir):
    print(dirname)
```

```
./datasets/cats_and_dogs_small
./datasets/cats_and_dogs_small/validation
./datasets/cats_and_dogs_small/test
./datasets/cats_and_dogs_small/train
```

In [10]:

```
# 훈련용 고양이 사진 디렉터리
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

# 훈련용 강아지 사진 디렉터리
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)

# 검증용 고양이 사진 디렉터리
val_cats_dir = os.path.join(val_dir, 'cats')
os.mkdir(val_cats_dir)

# 검증용 강아지 사진 디렉터리
val_dogs_dir = os.path.join(val_dir, 'dogs')
os.mkdir(val_dogs_dir)

# 테스트용 고양이 사진 디렉터리
test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)

# 테스트용 강아지 사진 디렉터리
test_dogs_dir = os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
```

데이터 준비

In [11]:

```
# 처음 1,000개의 고양이 이미지를 train_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(train_cats_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 고양이 이미지를 validation_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(val_cats_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 고양이 이미지를 test_cats_dir에 복사합니다
fnames = ['cat.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(test_cats_dir, fname)
    shutil.copyfile(src, dst)

# 처음 1,000개의 강아지 이미지를 train_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(train_dogs_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 강아지 이미지를 validation_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1000, 1500)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(val_dogs_dir, fname)
    shutil.copyfile(src, dst)

# 다음 500개 강아지 이미지를 test_dogs_dir에 복사합니다
fnames = ['dog.{}.jpg'.format(i) for i in range(1500, 2000)]
for fname in fnames:
    src = os.path.join(ori_dataset_dir, fname)
    dst = os.path.join(test_dogs_dir, fname)
    shutil.copyfile(src, dst)
```

[훈련/검증/테스트]에 들어있는 사진의 개수를 카운트

In [12]:

```
print('훈련용 고양이 이미지 전체 개수:', len(os.listdir(train_cats_dir)))
print('훈련용 강아지 이미지 전체 개수:', len(os.listdir(train_dogs_dir)))

print('검증용 고양이 이미지 전체 개수:', len(os.listdir(val_cats_dir)))
print('검증용 강아지 이미지 전체 개수:', len(os.listdir(val_dogs_dir)))

print('테스트용 고양이 이미지 전체 개수:', len(os.listdir(test_cats_dir)))
print('테스트용 강아지 이미지 전체 개수:', len(os.listdir(test_dogs_dir)))
```

```
훈련용 고양이 이미지 전체 개수: 1000
훈련용 강아지 이미지 전체 개수: 1000
검증용 고양이 이미지 전체 개수: 500
검증용 강아지 이미지 전체 개수: 500
테스트용 고양이 이미지 전체 개수: 500
테스트용 강아지 이미지 전체 개수: 500
```

- 훈련 이미지 : 2000개
- 검증 이미지 : 1000개
- 테스트 이미지 : 1000개

02. 신경망 구성하기

목차로 이동하기

In [13]:

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [14]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2D  (None, 74, 74, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 128)        0
 2D)

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

- 150 x 150 크기에서 7 x 7 크기의 특성 맵으로 줄어든다.

최적화 알고리즘, 손실 함수 선택

In [16]:

```
from tensorflow.keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/rmsprop.py:130: UserWarnin
g: The `lr` argument is deprecated, use `learning_rate` instead.
  super(RMSprop, self).__init__(name, **kwargs)
```

03. 데이터 전처리

목차로 이동하기

- 사진 파일을 읽기
- JPEG 콘텐츠를 RGB픽셀로 디코딩
- 부동 소수 타입의 텐서로 변환
- 픽셀 값(0~255)의 스케일을 [0,1]사이로 조정

준비된 유틸리티

- keras.preprocessing.image
  - ImageDataGenerator : 디스크에 있는 이미지 파일을 배치 텐서로 변경하는 파이썬 제너레이터 생성

In [17]:

```
from keras.preprocessing.image import ImageDataGenerator

# 모든 이미지를 1/255로 스케일을 조정합니다
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,  # 타깃 디렉터리
        target_size=(150, 150),  # 모든 이미지를 150 × 150 크기로 변경
        batch_size=16,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필요
        class_mode='binary')

val_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(150, 150),
        batch_size=16,
        class_mode='binary')
```

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

제너레이터을 활용하여 데이터와 라벨을 확인해보기

In [18]:

```
for data_batch, labels_batch in train_generator:
    print('배치 데이터 크기:', data_batch.shape)
    print('배치 레이블 크기:', labels_batch.shape)
    break
```

```
배치 데이터 크기: (16, 150, 150, 3)
배치 레이블 크기: (16,)
```

- 제너레이터는 이 배치를 무한정으로 만들어낸다.
- 타깃 폴더에 있는 이미지를 끝없이 반복한다.

제너레이터를 사용한 데이터의 모델 훈련

- fit_generator 메서드는 fit 메서드와 동일하고 데이터 제너레이터를 사용 가능.
  - 데이터가 끝없이 생성되기에 케라스 모델에 하나의 에포크를 정의하기 위해 제너레이터로부터 얼마나 많은 샘플을 뽑을지 알려 주어야 한다.(steps_per_epoch 이를 설정)
  - validation_data 매개변수를 사용 가능.
  - validation_steps에서 얼마나 많은 배치를 추출할지 평가할지 validation_steps 매개변수에 지정한다.

In [20]:

```
## 경로에 이미지 데이터의 개수
num_cats_tr = len(os.listdir(train_cats_dir))
num_dogs_tr = len(os.listdir(train_dogs_dir))

num_cats_val = len(os.listdir(val_cats_dir))
num_dogs_val = len(os.listdir(val_dogs_dir))

total_train = num_cats_tr + num_dogs_tr
total_val = num_cats_val + num_dogs_val

print("학습용 데이터 : ", total_train)
print("검증용 데이터 : ", total_val)

batch_size = 16
epochs = 30
```

```
학습용 데이터 :  2000
검증용 데이터 :  1000
```

In [21]:

```
%%time

history = model.fit(   # model.fit_gerator -> model.fit() 으로 최근 추가됨
      train_generator,
      steps_per_epoch= total_train // batch_size,
      epochs=30,
      validation_data=val_generator,
      validation_steps= total_val // batch_size)
```

```
Epoch 1/30
125/125 [==============================] - 23s 72ms/step - loss: 0.6887 - acc: 0.519
0 - val_loss: 0.6880 - val_acc: 0.5343
Epoch 2/30
125/125 [==============================] - 9s 72ms/step - loss: 0.6539 - acc: 0.6285
- val_loss: 0.6303 - val_acc: 0.6431
Epoch 3/30
125/125 [==============================] - 11s 90ms/step - loss: 0.6008 - acc: 0.688
5 - val_loss: 0.6428 - val_acc: 0.6391
Epoch 4/30
125/125 [==============================] - 10s 83ms/step - loss: 0.5650 - acc: 0.713
5 - val_loss: 0.6531 - val_acc: 0.6452
Epoch 5/30
125/125 [==============================] - 13s 103ms/step - loss: 0.5338 - acc: 0.72
75 - val_loss: 0.5749 - val_acc: 0.6865
Epoch 6/30
125/125 [==============================] - 9s 71ms/step - loss: 0.4955 - acc: 0.7635
- val_loss: 0.6248 - val_acc: 0.6623
Epoch 7/30
125/125 [==============================] - 11s 92ms/step - loss: 0.4722 - acc: 0.779
5 - val_loss: 0.5557 - val_acc: 0.7046
Epoch 8/30
125/125 [==============================] - 9s 72ms/step - loss: 0.4507 - acc: 0.7985
- val_loss: 0.5857 - val_acc: 0.7026
Epoch 9/30
125/125 [==============================] - 9s 72ms/step - loss: 0.4127 - acc: 0.8080
- val_loss: 0.5388 - val_acc: 0.7117
Epoch 10/30
125/125 [==============================] - 11s 91ms/step - loss: 0.3891 - acc: 0.834
0 - val_loss: 0.5848 - val_acc: 0.7177
Epoch 11/30
125/125 [==============================] - 11s 91ms/step - loss: 0.3614 - acc: 0.841
5 - val_loss: 0.5745 - val_acc: 0.7208
Epoch 12/30
125/125 [==============================] - 9s 72ms/step - loss: 0.3371 - acc: 0.8585
- val_loss: 0.5489 - val_acc: 0.7500
Epoch 13/30
125/125 [==============================] - 11s 91ms/step - loss: 0.3088 - acc: 0.875
5 - val_loss: 0.5432 - val_acc: 0.7429
Epoch 14/30
125/125 [==============================] - 9s 71ms/step - loss: 0.2892 - acc: 0.8810
- val_loss: 0.5495 - val_acc: 0.7560
Epoch 15/30
125/125 [==============================] - 9s 71ms/step - loss: 0.2637 - acc: 0.8985
- val_loss: 0.6172 - val_acc: 0.7308
Epoch 16/30
125/125 [==============================] - 9s 70ms/step - loss: 0.2452 - acc: 0.9050
- val_loss: 0.5647 - val_acc: 0.7490
Epoch 17/30
125/125 [==============================] - 9s 71ms/step - loss: 0.2159 - acc: 0.9150
```

```
- val_loss: 0.5891 - val_acc: 0.7470
Epoch 18/30
125/125 [==============================] - 11s 91ms/step - loss: 0.2031 - acc: 0.923
5 - val_loss: 0.6163 - val_acc: 0.7510
Epoch 19/30
125/125 [==============================] - 9s 71ms/step - loss: 0.1842 - acc: 0.9305
- val_loss: 0.6634 - val_acc: 0.7359
Epoch 20/30
125/125 [==============================] - 10s 81ms/step - loss: 0.1557 - acc: 0.949
5 - val_loss: 0.6894 - val_acc: 0.7409
Epoch 21/30
125/125 [==============================] - 9s 76ms/step - loss: 0.1384 - acc: 0.9525
- val_loss: 0.7833 - val_acc: 0.7208
Epoch 22/30
125/125 [==============================] - 11s 90ms/step - loss: 0.1268 - acc: 0.962
5 - val_loss: 0.7463 - val_acc: 0.7429
Epoch 23/30
125/125 [==============================] - 10s 78ms/step - loss: 0.1072 - acc: 0.966
0 - val_loss: 0.7518 - val_acc: 0.7470
Epoch 24/30
125/125 [==============================] - 9s 71ms/step - loss: 0.0959 - acc: 0.9715
- val_loss: 0.7750 - val_acc: 0.7450
Epoch 25/30
125/125 [==============================] - 9s 71ms/step - loss: 0.0784 - acc: 0.9760
- val_loss: 0.8603 - val_acc: 0.7369
Epoch 26/30
125/125 [==============================] - 10s 81ms/step - loss: 0.0604 - acc: 0.981
5 - val_loss: 0.9642 - val_acc: 0.7258
Epoch 27/30
125/125 [==============================] - 11s 90ms/step - loss: 0.0562 - acc: 0.983
5 - val_loss: 0.9108 - val_acc: 0.7470
Epoch 28/30
125/125 [==============================] - 9s 71ms/step - loss: 0.0511 - acc: 0.9855
- val_loss: 0.9119 - val_acc: 0.7359
Epoch 29/30
125/125 [==============================] - 10s 82ms/step - loss: 0.0365 - acc: 0.991
5 - val_loss: 0.9879 - val_acc: 0.7389
Epoch 30/30
125/125 [==============================] - 11s 85ms/step - loss: 0.0334 - acc: 0.992
5 - val_loss: 0.9839 - val_acc: 0.7450
CPU times: user 5min 31s, sys: 10.9 s, total: 5min 42s
Wall time: 5min 33s
```

In [22]:

```
### CPU 30 epochs : 52분 55초
### GPU 30 epochs : 5분 33초
```

훈련 후, 모델 저장

In [26]:

```
model.save('cats_and_dogs_small_1.h5')
```

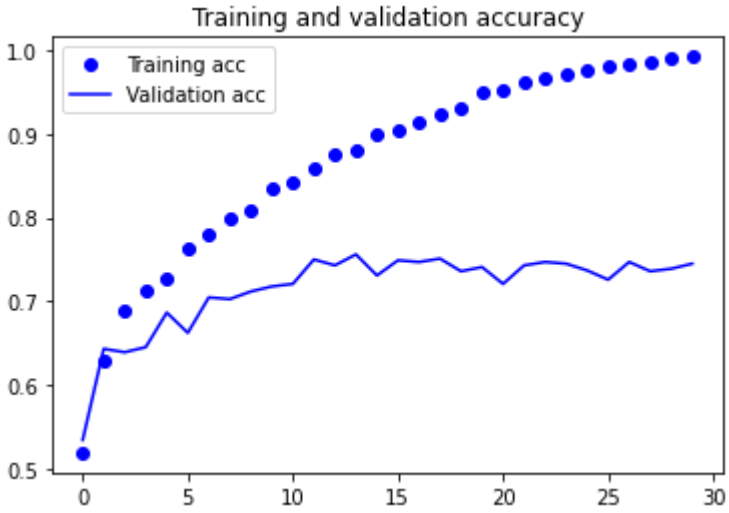
훈련 데이터와 검증 데이터의 모델의 손실과 정확도

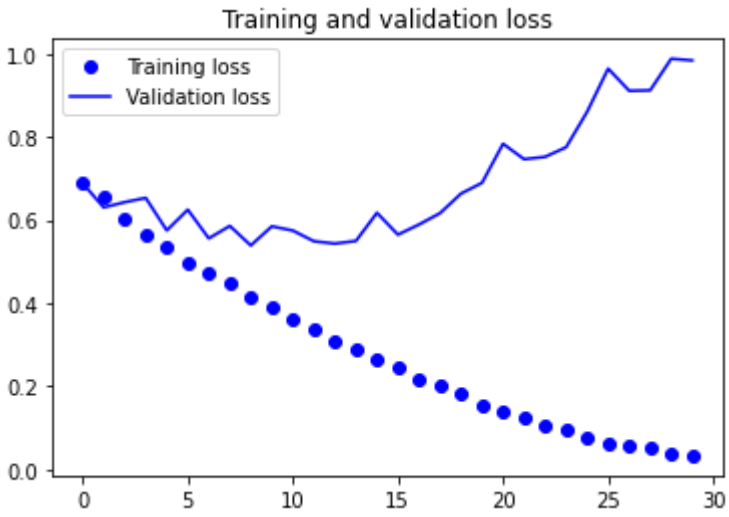
In [27]:

```
import matplotlib.pyplot as plt
```

In [28]:

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

epochs = range(len(acc))

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.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()
```





- 검증 손실은 다섯 번의 에포크만에 최소값에 다다른 이후 더 이상 진전이 없음.
- 반면 훈련 손실은 거의 0에 도달할 때까지 선형적으로 계속 감소.
- 비교적 훈련 샘플의 수(2,000)개 적기 때문에 과대 적합이 가장 중요.
    - 드롭아웃이나 가중치 감소(L2규제)와 같은 과대적합을 감소시킬 수 있는 여러가지 기법 학습.

04. 데이터 증식 사용하기

목차로 이동하기

- 과대 적합은 학습할 샘플이 너무 적어 새로운 데이터에 일반화할 수 있는 모델을 훈련시킬 수 없기 때문에 발생합니다.
- 무한히 많은 데이터가 주어지면 데이터 분포의 모든 가능한 측면을 모델이 학습할 수 있을 것입니다.
- 데이터 증식은 기존 훈련 샘플로부터 더 많은 훈련 데이터를 생성하는 방법.

In [29]:

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

- rotation_range : 랜덤하게 사진을 회전시킬 각도 범위(0~180 사이)
- width_shift_range와 height_shift_range는 사진을 수평과 수직으로 랜덤하게 평행 이동 시킬 범위
- shear_range : 랜덤하게 전단 변환(shearing transformation)을 적용할 각도 범위
- zoom_range : 랜덤하게 사진을 확대할 범위
- horizontal_flip : 랜덤하게 이미지를 수평으로 뒤집기. 수평 대칭을 가정할 수 있을 때 사용.(예 풍경/인물 사진)
- fill_mode : 회전이나 가로/세로 이동으로 인해 새롭게 생성해야 할 픽셀을 채울 전략

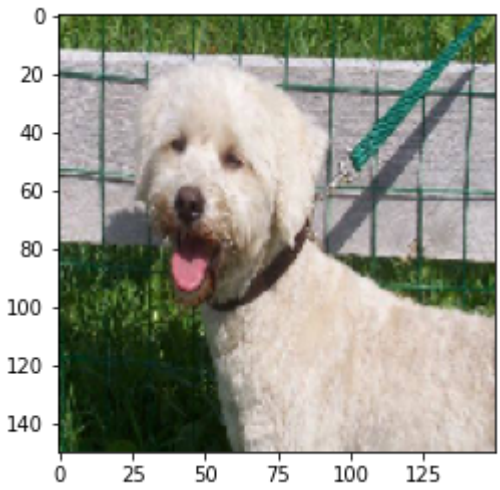
In [31]:

```
# 이미지 전처리 유틸리티 모듈
from keras.preprocessing import image

fnames = sorted([os.path.join(train_dogs_dir, fname) for fname in os.listdir(train_dogs_dir)])
print(fnames[50])

# 증식할 이미지 선택합니다
img_path = fnames[50]

# 이미지를 읽고 크기를 변경합니다
img = image.load_img(img_path, target_size=(150, 150))
plt.imshow(img)
```

./datasets/cats_and_dogs_small/train/dogs/dog.143.jpg

Out[31]:

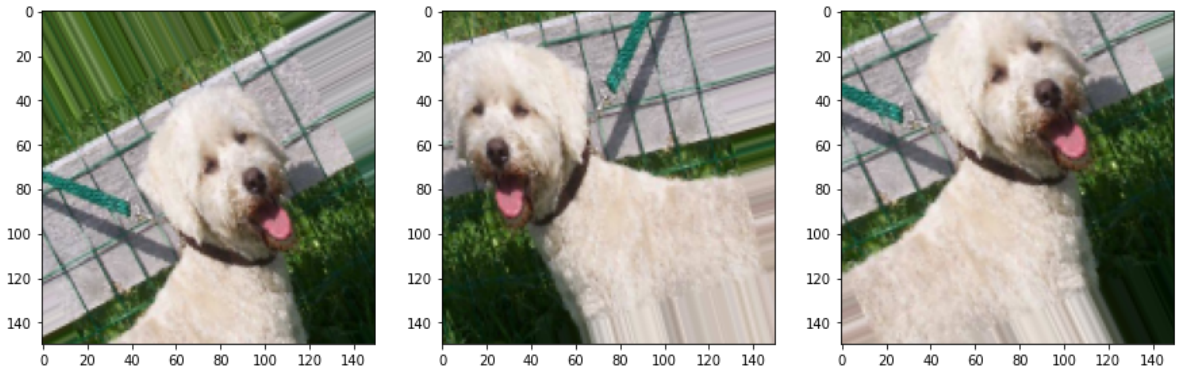
<matplotlib.image.AxesImage at 0x7f25c9d0c790>

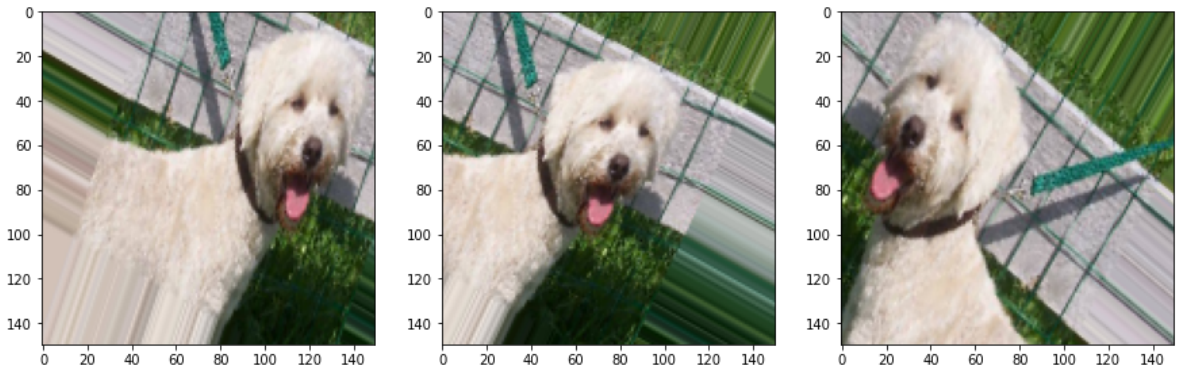


In [36]:

```
x = image.img_to_array(img)

# (1, 150, 150, 3) 크기로 변환합니다
x = x.reshape((1,) + x.shape)

# flow() 메서드는 랜덤하게 변환된 이미지의 배치를 생성합니다.
# 무한 반복되기 때문에 어느 지점에서 중지해야 합니다!
i = 0
plt.figure(figsize=(15, 15))
for batch in datagen.flow(x, batch_size=1):
    print(batch[0].shape)
    plt.subplot(2,3, i+1)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 6 == 0:
        break

plt.show()
```

```
(150, 150, 3)
(150, 150, 3)
(150, 150, 3)
(150, 150, 3)
(150, 150, 3)
(150, 150, 3)
```





신경망 구현하기

In [37]:

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              metrics=['acc'])
```

In [38]:

```
## 경로에 이미지 데이터의 개수
num_cats_tr = len(os.listdir(train_cats_dir))
num_dogs_tr = len(os.listdir(train_dogs_dir))

num_cats_val = len(os.listdir(val_cats_dir))
num_dogs_val = len(os.listdir(val_dogs_dir))

total_train = num_cats_tr + num_dogs_tr
total_val = num_cats_val + num_dogs_val

print("학습용 데이터 : ", total_train)
print("검증용 데이터 : ", total_val)

batch_size = 16
epochs = 15
```

```
학습용 데이터 :  2000
검증용 데이터 :  1000
```

In [39]:

```
%%time

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,)

# 검증 데이터는 증식되어서는 안된다.
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # 타깃 디렉터리
        train_dir,
        target_size=(150, 150),  # 모든 이미지를 150 × 150 크기로 변경
        batch_size=32,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블을 만들어야 합니다
        class_mode='binary')

val_generator = test_datagen.flow_from_directory(
        val_dir,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

batch_size = 32

history = model.fit_generator(
      train_generator,
      steps_per_epoch= total_train // batch_size,
      epochs=50,
      validation_data= val_generator,
      validation_steps= total_val // batch_size)
```

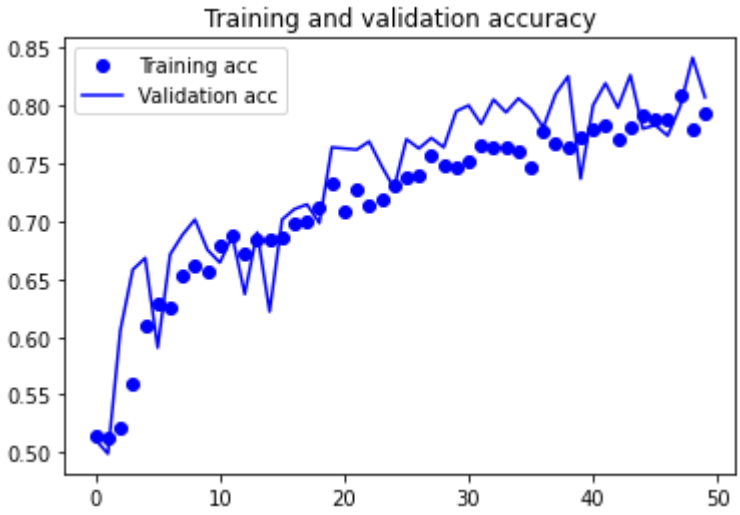
```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:35: UserWarning: `Mod
el.fit_generator` is deprecated and will be removed in a future version. Please us
e `Model.fit`, which supports generators.
```

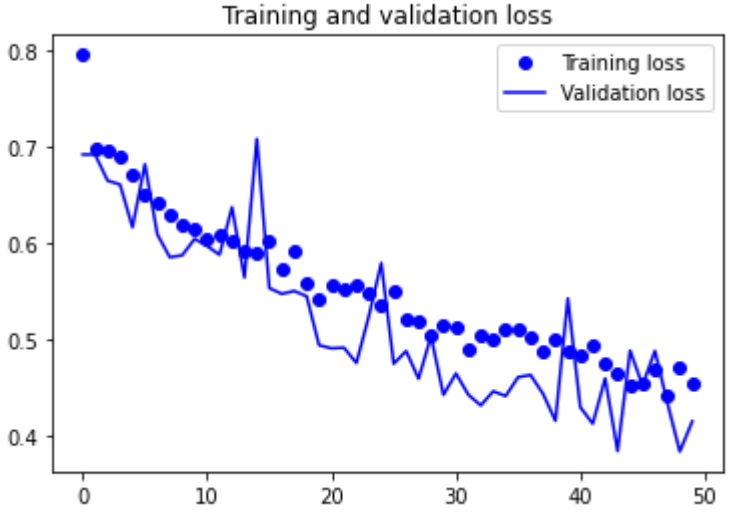
In [45]:

```
model.save('cats_and_dogs_small_3_50epoch.h5')
```

In [46]:

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

epochs = range(len(acc))

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.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()
```





05. 모델 학습 시, 콜백 함수 사용하기

목차로 이동하기

- ReduceLROnPlateau : 모델의 개선이 없을 경우, Learing Rate를 조절해 모델의 개선을 유도하는 콜백함수.

- monitor : ReduceLROnPlateau의 기준이 되는 값을 입력.
- factor : Learning rate를 얼마나 감소시킬 지 정하는 인자값.
  - 현재 lr이 0.01이고 factor가 0.8일때, 콜백함수가 실행되면 lr은 0.008이다.
  - 현재 lr이 0.3이고 factor가 0.1일때, 콜백함수가 실행되면 lr은 0.03이다.
- patience : monitor되는 값의 개선이 없을 경우, 최소 몇번의 epoch을 진행하고, learning rate를 조절할지.
- verbose : 0 또는 1, 1일경우 EarlyStopping이 적용될 때, 화면이 적용. 0일경우 화면에 나타냄이 없이 종료.

In [47]:

```
from tensorflow.keras.callbacks import LearningRateScheduler,ReduceLROnPlateau
from tensorflow import keras
```

In [54]:

```
lr_scheduler = ReduceLROnPlateau(monitor='val_accuracy',factor=0.8, patience=5, verbose=1)
save_best = keras.callbacks.ModelCheckpoint("Model.h5",
                                    monitor='val_accuracy',
                                    save_best_only=True, verbose=1)
```

In [61]:

```
# 모델을 불러온다.
load_model = keras.models.load_model('cats_and_dogs_small_3_50epoch.h5')
load_model
```

Out[61]:

```
<keras.engine.sequential.Sequential at 0x7f25c80f7190>
```

In [62]:

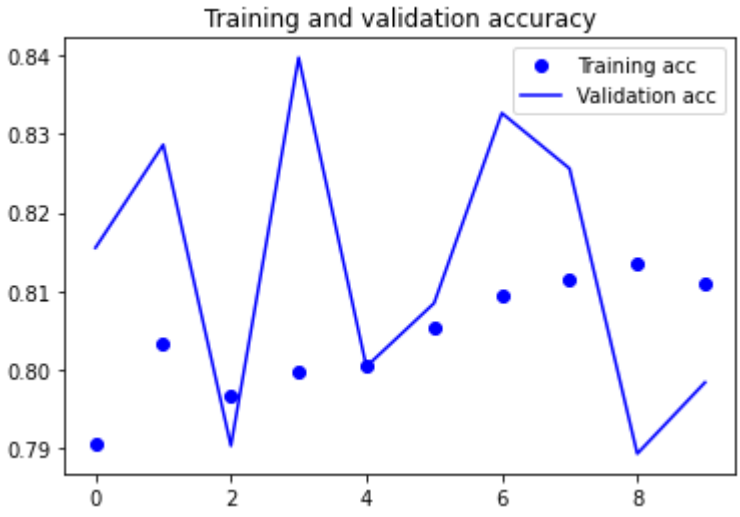
```
history = load_model.fit_generator(
      train_generator,
      steps_per_epoch= total_train // batch_size,
      epochs=10,
      validation_data= val_generator,
      validation_steps= total_val // batch_size,
      callbacks=[save_best])
```

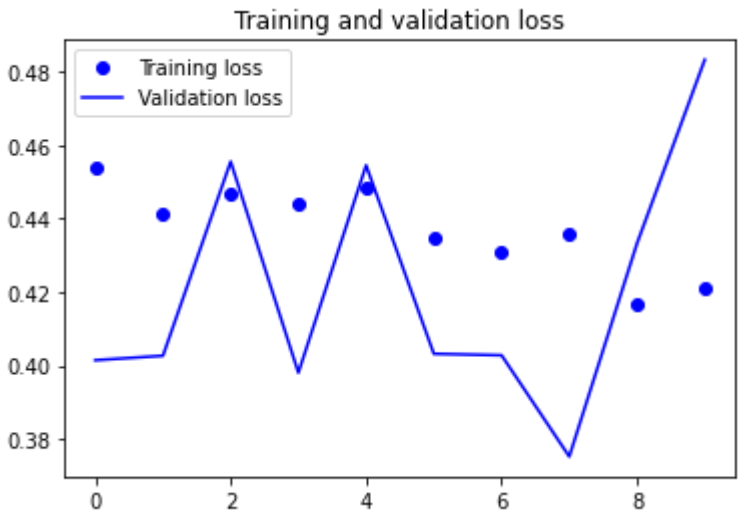
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning: `Model.
fit_generator` is deprecated and will be removed in a future version. Please use `Mo
del.fit`, which supports generators.
  import sys

Epoch 1/10
62/62 [==============================] - ETA: 0s - loss: 0.4539 - acc: 0.7907WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 17s 271ms/step - loss: 0.4539 - acc: 0.7907
- val_loss: 0.4014 - val_acc: 0.8155
Epoch 2/10
62/62 [==============================] - ETA: 0s - loss: 0.4414 - acc: 0.8034WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 18s 287ms/step - loss: 0.4414 - acc: 0.8034
- val_loss: 0.4027 - val_acc: 0.8286
Epoch 3/10
62/62 [==============================] - ETA: 0s - loss: 0.4467 - acc: 0.7967WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 17s 274ms/step - loss: 0.4467 - acc: 0.7967
- val_loss: 0.4556 - val_acc: 0.7903
Epoch 4/10
62/62 [==============================] - ETA: 0s - loss: 0.4439 - acc: 0.7998WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 16s 266ms/step - loss: 0.4439 - acc: 0.7998
- val_loss: 0.3980 - val_acc: 0.8397
Epoch 5/10
62/62 [==============================] - ETA: 0s - loss: 0.4485 - acc: 0.8004WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 16s 266ms/step - loss: 0.4485 - acc: 0.8004
- val_loss: 0.4545 - val_acc: 0.8004
Epoch 6/10
62/62 [==============================] - ETA: 0s - loss: 0.4345 - acc: 0.8054WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 16s 266ms/step - loss: 0.4345 - acc: 0.8054
- val_loss: 0.4032 - val_acc: 0.8085
Epoch 7/10
62/62 [==============================] - ETA: 0s - loss: 0.4312 - acc: 0.8095WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 17s 267ms/step - loss: 0.4312 - acc: 0.8095
- val_loss: 0.4028 - val_acc: 0.8327
Epoch 8/10
62/62 [==============================] - ETA: 0s - loss: 0.4356 - acc: 0.8115WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 19s 314ms/step - loss: 0.4356 - acc: 0.8115
- val_loss: 0.3752 - val_acc: 0.8256
Epoch 9/10
62/62 [==============================] - ETA: 0s - loss: 0.4167 - acc: 0.8135WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 17s 266ms/step - loss: 0.4167 - acc: 0.8135
- val_loss: 0.4333 - val_acc: 0.7893
Epoch 10/10
```

```
62/62 [==============================] - ETA: 0s - loss: 0.4212 - acc: 0.8110WARNIN
G:tensorflow:Can save best model only with val_accuracy available, skipping.
62/62 [==============================] - 17s 268ms/step - loss: 0.4212 - acc: 0.8110
- val_loss: 0.4833 - val_acc: 0.7984
```

In [63]:

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

epochs = range(len(acc))

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.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()
```





확인

- 다른 규제 기법을 더 사용하고 네트워크의 파라미터를 튜닝하면 (합성곱 층의 필터 수나 네트워크 층의 수 등) 86%, 87%정도까지 더 높은 정확도를 얻을 수도 있다.
- 데이터가 적기 때문에 컨브넷을 처음부터 훈련해서 더 높은 정확도를 달성하기는 어렵다.
- 이런 상황에서 정확도를 높이기 위한 다음 단계는 사전 훈련된 모델을 사용하는 것이다.