강아지 vs 고양이 분류하기(1)

학습 내용

- 구글 드라이브와 연동하는 것을 실습해 봅니다.
- 개와 고양이의 이미지를 준비합니다.
- 신경망을 구축 후, 학습을 수행합니다.
- 모델 학습 후, 이를 저장하는 것을 대해 알아봅니다.

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







- url : https://s3.amazonaws.com/book.keras.io/img/ch5/cats_vs_dogs_samples.jpg (https://s3.amazonaws.com/book.keras.io/img/ch5/cats_vs_dogs_samples.jpg)

- 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 [38]:

```
import os, shutil
```

구글 드라이브 연동하기

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

In [39]:

```
### 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).
```

- 구글 드라이브의 dataset/cats_dogs 의 위치에 데이터를 위치시킵니다.
- cp 명령을 이용하여 내 현재 구글 콜랩으로 복사해 오기.

In [40]:

```
# 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 15:12 sampleSubmission.csv
277660 -rw------- 1 root root 284321224 Jul 17 15:12 test1.zip
556204 -rw------- 1 root root 569546721 Jul 17 15:12 train.zip
```

파일 압축풀기

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

In [41]:

```
!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 [42]:

```
!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 609264
drwxr-xr-x 2 root root 770048 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Jul 17 15:12 ..
-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 304264
drwxr-xr-x 2 root root 270336 Sep 20  2013 .
drwxr-xr-x 4 root root   4096 Jul 17 15:12 ..
-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
```

- train 25000장, test 12500장

데이터 셋 경로 지정

In [43]:

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

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

Out[43]:

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

In [9]:

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

학습, 검증, 테스트 데이터 셋 디렉터리 준비(생성)

- base_dir = './datasets/cats_and_dogs_small'

In [10]:

```
# 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)
```

In [11]:

```
# 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)
```

In [12]:

```
# 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)
```

In [13]:

```
# 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)
```

In [14]:

```
print(train_cats_dir, train_dogs_dir)
print(val_cats_dir, val_dogs_dir)
print(test_cats_dir, test_dogs_dir)
```

```
./datasets/cats_and_dogs_small/train/cats ./datasets/cats_and_dogs_small/train/dogs
./datasets/cats_and_dogs_small/validation/cats ./datasets/cats_and_dogs_small/valida
tion/dogs
./datasets/cats_and_dogs_small/test/cats ./datasets/cats_and_dogs_small/test/dogs
```

In [17]:

```
!ls -al './datasets/cats_and_dogs_small/train' | head -5
!ls -al './datasets/cats_and_dogs_small/validation/' | head -5
!ls -al './datasets/cats_and_dogs_small/test/' | head -5
```

```
total 16
drwxr-xr-x 4 root root 4096 Jul 17 14:48 .
drwxr-xr-x 5 root root 4096 Jul 17 14:48 ..
drwxr-xr-x 2 root root 4096 Jul 17 14:48 cats
drwxr-xr-x 2 root root 4096 Jul 17 14:48 dogs
total 16
drwxr-xr-x 4 root root 4096 Jul 17 14:48 .
drwxr-xr-x 5 root root 4096 Jul 17 14:48 ..
drwxr-xr-x 2 root root 4096 Jul 17 14:48 cats
drwxr-xr-x 2 root root 4096 Jul 17 14:48 dogs
total 16
drwxr-xr-x 4 root root 4096 Jul 17 14:48 .
drwxr-xr-x 5 root root 4096 Jul 17 14:48 ..
drwxr-xr-x 2 root root 4096 Jul 17 14:48 cats
drwxr-xr-x 2 root root 4096 Jul 17 14:48 dogs
```

데이터 준비

- 개와 고양이의 각각의 이미지 준비
  - 학습용 이미지 1000장
  - 검증용 이미지 500장
  - 테스트용 이미지 500장

In [18]:

```
# 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 [19]:

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

```
훈련용 고양이 이미지 전체 개수: 1000
훈련용 강아지 이미지 전체 개수: 1000
```

In [20]:

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

```
검증용 고양이 이미지 전체 개수: 500
검증용 강아지 이미지 전체 개수: 500
```

In [21]:

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

```
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

- Conv2D (3x3) filter:32 - Maxpooling2D (2x2) stride 1
- Conv2D (3x3) filter:64 - Maxpooling2D (2x2) stride 1
- Conv2D (3x3) filter:128 - Maxpooling2D (2x2) stride 1
- Conv2D (3x3) filter:128 - Maxpooling2D (2x2) stride 1

In [23]:

```
from tensorflow.keras import layers
from tensorflow.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 [24]:

```
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 [26]:

```
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 [27]:

```
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=20,
        # 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=20,
        class_mode='binary')
```

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

이미지 제너레이터을 활용하여 데이터와 라벨을 확인해보기

In [29]:

```
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: (20, 150, 150, 3)
배치 레이블 크기: (20,)
```

In [31]:

```
labels_batch[0:3], data_batch[0]
```

Out[31]:

```
(array([1., 1., 1.], dtype=float32),
 array([[[0.41176474, 0.41176474, 0.41176474],
         [0.41176474, 0.41176474, 0.41176474],
         [0.38823533, 0.38823533, 0.38823533],
         ...,
         [0.7725491 , 0.7725491 , 0.7725491 ],
         [0.62352943, 0.62352943, 0.62352943],
         [0.62352943, 0.62352943, 0.62352943]],

        [[0.41176474, 0.41176474, 0.41176474],
         [0.41176474, 0.41176474, 0.41176474],
         [0.38823533, 0.38823533, 0.38823533],
         ...,
         [0.7725491 , 0.7725491 , 0.7725491 ],
         [0.62352943, 0.62352943, 0.62352943],
         [0.62352943, 0.62352943, 0.62352943]],

        [[0.4156863 , 0.4156863 , 0.4156863 ],
         [0.4156863 , 0.4156863 , 0.4156863 ],
         [0.38823533, 0.38823533, 0.38823533],
         ...,
         [0.7843138 , 0.7843138 , 0.7843138 ],
         [0.6392157 , 0.6392157 , 0.6392157 ],
         [0.6392157 , 0.6392157 , 0.6392157 ]],

        ...,

        [[0.50980395, 0.50980395, 0.50980395],
         [0.50980395, 0.50980395, 0.50980395],
         [0.5254902 , 0.5254902 , 0.5254902 ],
         ...,
         [0.6       , 0.6       , 0.6       ],
         [0.6       , 0.6       , 0.6       ],
         [0.6       , 0.6       , 0.6       ]],

        [[0.5529412 , 0.5529412 , 0.5529412 ],
         [0.5529412 , 0.5529412 , 0.5529412 ],
         [0.5647059 , 0.5647059 , 0.5647059 ],
         ...,
         [0.6       , 0.6       , 0.6       ],
         [0.6       , 0.6       , 0.6       ],
         [0.6       , 0.6       , 0.6       ]],

        [[0.5529412 , 0.5529412 , 0.5529412 ],
         [0.5529412 , 0.5529412 , 0.5529412 ],
         [0.5647059 , 0.5647059 , 0.5647059 ],
         ...,
         [0.6       , 0.6       , 0.6       ],
         [0.6       , 0.6       , 0.6       ],
         [0.6       , 0.6       , 0.6       ]]], dtype=float32))
```

- 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 [32]:

```
## 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 = 20
epochs = 30
```

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

04. 모델 학습

목차로 이동하기

In [33]:

```
%%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
100/100 [==============================] - 22s 95ms/step - loss: 0.6885 - acc: 0.542
5 - val_loss: 0.6659 - val_acc: 0.5980
Epoch 2/30
100/100 [==============================] - 9s 91ms/step - loss: 0.6600 - acc: 0.6000
- val_loss: 0.6413 - val_acc: 0.6480
Epoch 3/30
100/100 [==============================] - 9s 90ms/step - loss: 0.6127 - acc: 0.6595
- val_loss: 0.6707 - val_acc: 0.6000
Epoch 4/30
100/100 [==============================] - 9s 92ms/step - loss: 0.5690 - acc: 0.7040
- val_loss: 0.6294 - val_acc: 0.6430
Epoch 5/30
100/100 [==============================] - 9s 92ms/step - loss: 0.5425 - acc: 0.7350
- val_loss: 0.5958 - val_acc: 0.6720
Epoch 6/30
100/100 [==============================] - 10s 97ms/step - loss: 0.5097 - acc: 0.744
5 - val_loss: 0.5911 - val_acc: 0.6770
Epoch 7/30
100/100 [==============================] - 10s 101ms/step - loss: 0.4833 - acc: 0.76
80 - val_loss: 0.5830 - val_acc: 0.6910
Epoch 8/30
100/100 [==============================] - 9s 89ms/step - loss: 0.4580 - acc: 0.7745
- val_loss: 0.5821 - val_acc: 0.6950
Epoch 9/30
100/100 [==============================] - 9s 88ms/step - loss: 0.4348 - acc: 0.7995
- val_loss: 0.5498 - val_acc: 0.7160
Epoch 10/30
100/100 [==============================] - 9s 89ms/step - loss: 0.4090 - acc: 0.8220
- val_loss: 0.5551 - val_acc: 0.7250
Epoch 11/30
100/100 [==============================] - 11s 108ms/step - loss: 0.3889 - acc: 0.81
75 - val_loss: 0.5439 - val_acc: 0.7190
Epoch 12/30
100/100 [==============================] - 11s 114ms/step - loss: 0.3583 - acc: 0.83
75 - val_loss: 0.7095 - val_acc: 0.6670
Epoch 13/30
100/100 [==============================] - 11s 107ms/step - loss: 0.3361 - acc: 0.85
50 - val_loss: 0.5533 - val_acc: 0.7320
Epoch 14/30
100/100 [==============================] - 9s 89ms/step - loss: 0.3150 - acc: 0.8690
- val_loss: 0.5743 - val_acc: 0.7380
Epoch 15/30
100/100 [==============================] - 9s 88ms/step - loss: 0.2876 - acc: 0.8820
- val_loss: 0.5726 - val_acc: 0.7410
Epoch 16/30
100/100 [==============================] - 9s 90ms/step - loss: 0.2677 - acc: 0.8965
- val_loss: 0.5922 - val_acc: 0.7270
Epoch 17/30
100/100 [==============================] - 9s 91ms/step - loss: 0.2427 - acc: 0.9060
```

```
- val_loss: 0.6538 - val_acc: 0.7290
Epoch 18/30
100/100 [==============================] - 9s 91ms/step - loss: 0.2132 - acc: 0.9230
- val_loss: 0.5955 - val_acc: 0.7460
Epoch 19/30
100/100 [==============================] - 9s 91ms/step - loss: 0.2031 - acc: 0.9265
- val_loss: 0.6357 - val_acc: 0.7430
Epoch 20/30
100/100 [==============================] - 9s 90ms/step - loss: 0.1750 - acc: 0.9390
- val_loss: 0.6690 - val_acc: 0.7310
Epoch 21/30
100/100 [==============================] - 9s 91ms/step - loss: 0.1626 - acc: 0.9435
- val_loss: 0.6602 - val_acc: 0.7370
Epoch 22/30
100/100 [==============================] - 9s 91ms/step - loss: 0.1415 - acc: 0.9550
- val_loss: 0.7833 - val_acc: 0.7360
Epoch 23/30
100/100 [==============================] - 9s 91ms/step - loss: 0.1310 - acc: 0.9590
- val_loss: 0.7083 - val_acc: 0.7320
Epoch 24/30
100/100 [==============================] - 9s 91ms/step - loss: 0.1106 - acc: 0.9655
- val_loss: 0.7745 - val_acc: 0.7310
Epoch 25/30
100/100 [==============================] - 9s 91ms/step - loss: 0.0927 - acc: 0.9730
- val_loss: 0.7833 - val_acc: 0.7350
Epoch 26/30
100/100 [==============================] - 9s 92ms/step - loss: 0.0821 - acc: 0.9770
- val_loss: 0.9688 - val_acc: 0.7190
Epoch 27/30
100/100 [==============================] - 9s 91ms/step - loss: 0.0725 - acc: 0.9775
- val_loss: 1.0292 - val_acc: 0.7150
Epoch 28/30
100/100 [==============================] - 9s 92ms/step - loss: 0.0643 - acc: 0.9810
- val_loss: 1.0052 - val_acc: 0.7080
Epoch 29/30
100/100 [==============================] - 10s 104ms/step - loss: 0.0546 - acc: 0.98
40 - val_loss: 0.9281 - val_acc: 0.7370
Epoch 30/30
100/100 [==============================] - 9s 91ms/step - loss: 0.0443 - acc: 0.9890
- val_loss: 1.0565 - val_acc: 0.7250
CPU times: user 5min 35s, sys: 10.7 s, total: 5min 45s
Wall time: 5min 3s
```

In [34]:

```
### CPU 30 epochs : 52분 55초
### GPU 30 epochs : 4분 21초
```

05. 훈련 후, 모델 저장

목차로 이동하기

In [35]:

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

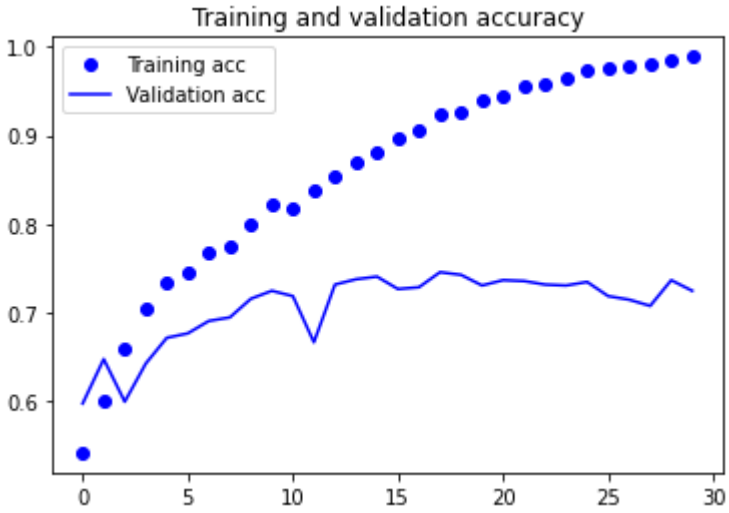
훈련 데이터와 검증 데이터의 모델의 손실과 정확도

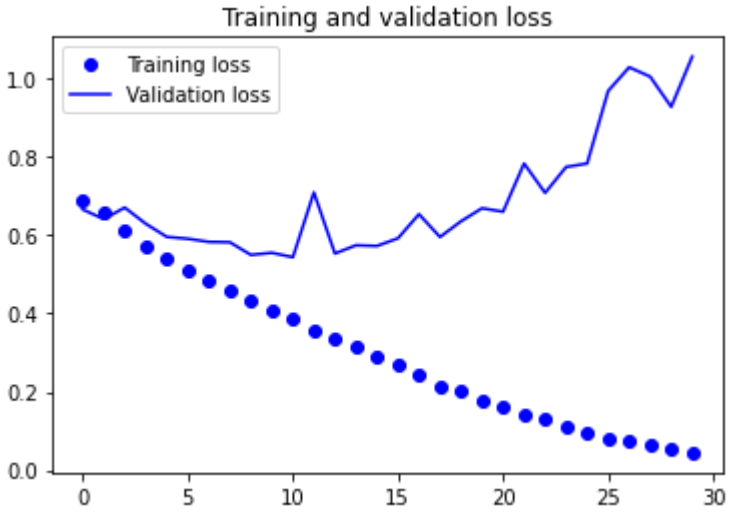
In [36]:

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

In [37]:

```
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.

실습해 보기

- 다양한 방법을 이용해서 성능을 개선시키는 것을 해 보기

- history
  - 2022/07 code 실행 및 확인