

강아지 vs 고양이 분류하기(3)-VGG16

학습 내용

- 강아지와 고양이 데이터를 캐글로 부터 데이터를 준비
- 사전 훈련 모델에 대해 알아봅니다.
- VGG16 모델을 불러와 사용해 봅니다.
 - 데이터 증식을 사용하지 않은 작은 데이터 셋을 활용함.

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](#)

[02. 사전 훈련된 컨브넷 사용](#)

01. 데이터 준비하기

[목차로 이동하기](#)

In [16]:

```
import os, shutil
```

In [17]:

```
### 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 [18]:

```
# 구글 드라이브에서 데이터 복사
```

```
!cp -r '/content/drive/MyDrive/dataset/cats_dogs' '/content/'
!ls -ls '/content/cats_dogs'
```

```
total 833952
  88 -rw----- 1 root root    88903 Jul 17 16:29 sampleSubmission.csv
277660 -rw----- 1 root root 284321224 Jul 17 16:29 test1.zip
556204 -rw----- 1 root root 569546721 Jul 17 16:29 train.zip
```

In [19]:

```
# 압축을 풀고, 파일 개수 확인
```

```
!rm -rf '/content/datasets/'
!unzip '/content/cats_dogs/test1.zip' -d '/content/datasets/'
!unzip '/content/cats_dogs/train.zip' -d '/content/datasets/'
!ls '/content/datasets/'
!ls -l '/content/datasets/train/' | grep ^- | wc -l
!ls -l '/content/datasets/test1/' | grep ^- | wc -l
```

스트리밍 출력 내용이 길어서 마지막 5000줄이 삭제되었습니다.

```
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
inflating: /content/datasets/train/dog.5514.jpg
inflating: /content/datasets/train/dog.5515.jpg
inflating: /content/datasets/train/dog.5516.jpg
inflating: /content/datasets/train/dog.5517.jpg
inflating: /content/datasets/train/dog.5518.jpg
inflating: /content/datasets/train/dog.5519.jpg
inflating: /content/datasets/train/dog.5520.jpg
inflating: /content/datasets/train/dog.5521.jpg
inflating: /content/datasets/train/dog.5522.jpg
inflating: /content/datasets/train/dog.5523.jpg
inflating: /content/datasets/train/dog.5524.jpg
inflating: /content/datasets/train/dog.5525.jpg
inflating: /content/datasets/train/dog.5526.jpg
inflating: /content/datasets/train/dog.5527.jpg
inflating: /content/datasets/train/dog.5528.jpg
inflating: /content/datasets/train/dog.5529.jpg
inflating: /content/datasets/train/dog.5530.jpg
inflating: /content/datasets/train/dog.5531.jpg
inflating: /content/datasets/train/dog.5532.jpg
inflating: /content/datasets/train/dog.5533.jpg
inflating: /content/datasets/train/dog.5534.jpg
inflating: /content/datasets/train/dog.5535.jpg
inflating: /content/datasets/train/dog.5536.jpg
inflating: /content/datasets/train/dog.5537.jpg
inflating: /content/datasets/train/dog.5538.jpg
inflating: /content/datasets/train/dog.5539.jpg
inflating: /content/datasets/train/dog.5540.jpg
inflating: /content/datasets/train/dog.5541.jpg
inflating: /content/datasets/train/dog.5542.jpg
inflating: /content/datasets/train/dog.5543.jpg
inflating: /content/datasets/train/dog.5544.jpg
inflating: /content/datasets/train/dog.5545.jpg
inflating: /content/datasets/train/dog.5546.jpg
inflating: /content/datasets/train/dog.5547.jpg
inflating: /content/datasets/train/dog.5548.jpg
inflating: /content/datasets/train/dog.5549.jpg
inflating: /content/datasets/train/dog.5550.jpg
inflating: /content/datasets/train/dog.5551.jpg
inflating: /content/datasets/train/dog.5552.jpg
inflating: /content/datasets/train/dog.5553.jpg
inflating: /content/datasets/train/dog.5554.jpg
inflating: /content/datasets/train/dog.5555.jpg
inflating: /content/datasets/train/dog.5556.jpg
inflating: /content/datasets/train/dog.5557.jpg
inflating: /content/datasets/train/dog.5558.jpg
inflating: /content/datasets/train/dog.5559.jpg
inflating: /content/datasets/train/dog.5560.jpg
inflating: /content/datasets/train/dog.5561.jpg
inflating: /content/datasets/train/dog.5562.jpg
inflating: /content/datasets/train/dog.5563.jpg
inflating: /content/datasets/train/dog.5564.jpg
inflating: /content/datasets/train/dog.5565.jpg
inflating: /content/datasets/train/dog.5566.jpg
inflating: /content/datasets/train/dog.5567.jpg
inflating: /content/datasets/train/dog.5568.jpg
inflating: /content/datasets/train/dog.5569.jpg
inflating: /content/datasets/train/dog.5570.jpg
inflating: /content/datasets/train/dog.5571.jpg
inflating: /content/datasets/train/dog.5572.jpg
inflating: /content/datasets/train/dog.5573.jpg
inflating: /content/datasets/train/dog.5574.jpg
inflating: /content/datasets/train/dog.5575.jpg
inflating: /content/datasets/train/dog.5576.jpg
inflating: /content/datasets/train/dog.5577.jpg
inflating: /content/datasets/train/dog.5578.jpg
inflating: /content/datasets/train/dog.5579.jpg
inflating: /content/datasets/train/dog.5580.jpg
inflating: /content/datasets/train/dog.5581.jpg
inflating: /content/datasets/train/dog.5582.jpg
inflating: /content/datasets/train/dog.5583.jpg
inflating: /content/datasets/train/dog.5584.jpg
inflating: /content/datasets/train/dog.5585.jpg
inflating: /content/datasets/train/dog.5586.jpg
inflating: /content/datasets/train/dog.5587.jpg
inflating: /content/datasets/train/dog.5588.jpg
inflating: /content/datasets/train/dog.5589.jpg
inflating: /content/datasets/train/dog.5590.jpg
inflating: /content/datasets/train/dog.5591.jpg
inflating: /content/datasets/train/dog.5592.jpg
inflating: /content/datasets/train/dog.5593.jpg
inflating: /content/datasets/train/dog.5594.jpg
inflating: /content/datasets/train/dog.5595.jpg
inflating: /content/datasets/train/dog.5596.jpg
inflating: /content/datasets/train/dog.5597.jpg
inflating: /content/datasets/train/dog.5598.jpg
inflating: /content/datasets/train/dog.5599.jpg
inflating: /content/datasets/train/dog.5600.jpg
```

In [20]:

```
# 학습용 25000개, 테스트용 12500
!ls -ltr '/content/datasets/' | head -10
!ls -ltr '/content/datasets/train' | tail -10
!ls -ltr '/content/datasets/test1' | tail -10
```

```
total 1020
drwxr-xr-x 2 root root 770048 Sep 20 2013 train
drwxr-xr-x 2 root root 270336 Sep 20 2013 test1
-rw-r--r-- 1 root root 25671 Sep 20 2013 cat.9937.jpg
-rw-r--r-- 1 root root 3744 Sep 20 2013 cat.9936.jpg
-rw-r--r-- 1 root root 27700 Sep 20 2013 cat.9935.jpg
-rw-r--r-- 1 root root 32621 Sep 20 2013 cat.9999.jpg
-rw-r--r-- 1 root root 9937 Sep 20 2013 cat.9998.jpg
-rw-r--r-- 1 root root 18575 Sep 20 2013 cat.9997.jpg
-rw-r--r-- 1 root root 16855 Sep 20 2013 cat.9996.jpg
-rw-r--r-- 1 root root 13785 Sep 20 2013 cat.9995.jpg
-rw-r--r-- 1 root root 30029 Sep 20 2013 cat.9994.jpg
-rw-r--r-- 1 root root 29133 Sep 20 2013 cat.9993.jpg
-rw-r--r-- 1 root root 5286 Sep 20 2013 12440.jpg
-rw-r--r-- 1 root root 16612 Sep 20 2013 12439.jpg
-rw-r--r-- 1 root root 42306 Sep 20 2013 12438.jpg
-rw-r--r-- 1 root root 20759 Sep 20 2013 12437.jpg
-rw-r--r-- 1 root root 16850 Sep 20 2013 12436.jpg
-rw-r--r-- 1 root root 26283 Sep 20 2013 12435.jpg
-rw-r--r-- 1 root root 29062 Sep 20 2013 12434.jpg
-rw-r--r-- 1 root root 23917 Sep 20 2013 12500.jpg
-rw-r--r-- 1 root root 24012 Sep 20 2013 12499.jpg
-rw-r--r-- 1 root root 21471 Sep 20 2013 12498.jpg
```

In [21]:

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

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

```
total 609260
drwxr-xr-x 2 root root 770048 Sep 20 2013 .
drwxr-xr-x 4 root root 4096 Jul 17 16:29 ..
-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
12500
12500
total 304264
drwxr-xr-x 2 root root 270336 Sep 20 2013 .
drwxr-xr-x 4 root root 4096 Jul 17 16:29 ..
-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
12500
```

데이터 준비

- 데이터를 저장할 디렉터리 준비

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

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

```
# 학습, 검증, 테스트 분할을 위한 디렉터리 생성
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)

print(train_dir, val_dir, test_dir, sep="\\n")
```

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

In [25]:

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

# 학습용 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)
```

데이터를 복사

In [26]:

```
# 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 500개의 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 500개의 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)
```

In [27]:

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

학습용 고양이 이미지 전체 개수: 1000
학습용 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

02. 사전 훈련된 컨브넷 사용

[목차로 이동하기](#)

- 작은 이미지 데이터 셋에 딥러닝을 적용하는 일반적이고 매우 효과적인 방법은 **사전 훈련된 네트워크를 사용하는 것**.
- 사전 훈련된 네트워크(pretrained network)는 일반적으로 대규모 이미지 분류 문제를 위해 대량의 데이터셋에서 미리 훈련되어 저장된 네트워크를 사용.
- ImageNet : 1400만개의 레이블된 이미지와 1000개의 클래스를 가진 데이터 셋.
- VGG16 : 캐런 시몬연(Keren Simonyan)과 앤드류 지서먼(Andrew Zisserman)이 2014년 개발
- 이외에도 VGG, ResNet, Inception, Inception-ResNet, Xception, MobileNet(2017), SeNet 등

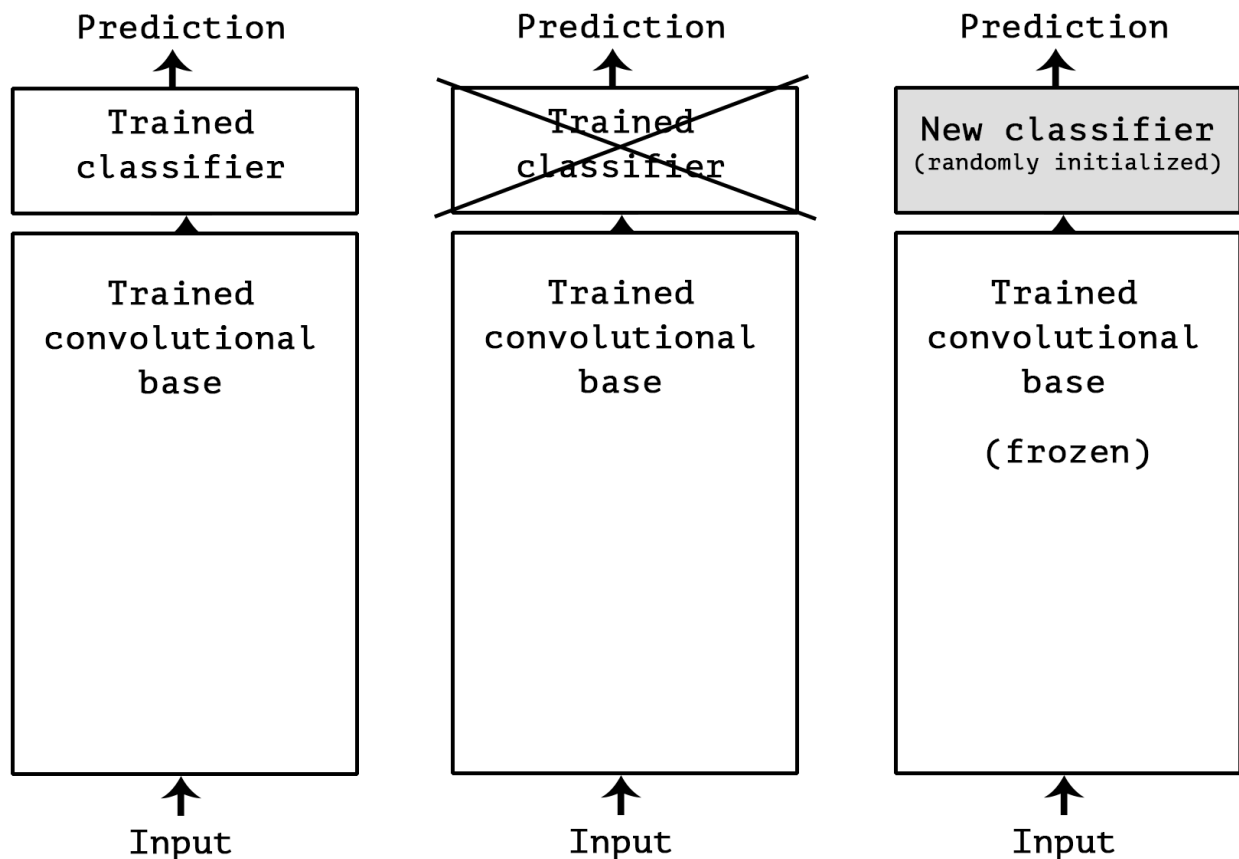
사전 훈련된 네트워크를 사용하는 두가지 방법

- 특성 추출(feature extraction)
- 미세 조정(fine-tuning)

특성 추출

앞서 보았듯이 컨브넷은 이미지 분류를 위해 두 부분으로 구성됩니다.

- 연속된 **합성곱과 풀링 층**으로 시작
- **완전 연결 분류기(fully connected layer)**로 끝납니다.
 - 첫 번째 부분을 모델의 합성곱 기반층(convolutional base)이라고 함.
 - 컨브넷의 경우 특성 추출은 사전에 훈련된 네트워크의 합성곱 기반층을 선택해 새로운 데이터를 통과시키고 그 출력으로 새로운 분류기를 훈련



- 특정 합성곱 층에서 추출한 표현의 일반성(그리고 재사용성)의 수준은 모델에 있는 층의 깊이에 달려 있습니다.
 - 모델의 하위 층은 (에지, 색깔, 질감 등과 같이) **지역적이고 매우 일반적인 특성 맵을 추출**합니다.
 - 반면 **상위 층은 ('강아지 눈'이나 '고양이 귀'와 같이) 좀 더 추상적인 개념을 추출**

만약 새로운 데이터셋이 원본 모델이 훈련한 데이터셋과 많이 다를 경우,

- 전체 합성곱 기반층을 사용하는 것보다는 모델의 하위 층 몇 개만 특성 추출에 사용하는 것이 좋음.
- VGG16 모델은 케라스에 패키지로 포함되어 있습니다.
 - 현재 `tensorflow.keras.applications` 모듈에서 임포트 가능
 - `tensorflow.keras.applications` 모듈에서 사용 가능한 이미지 분류 모델은 다음과 같습니다(모두 ImageNet 데이터셋에서 훈련)

CNN의 역사

- 2012년 : AlexNet - ILSVRC 우승 모델
- 2013년 : ZFNet - ILSVRC의 우승 모델
- 2014년 : GoLeNet – Going Deeper with Convolutions
 - 22개의 레이어를 사용한 더 Deep한 구조를 채용하여 ImageNet 2014에서 최고 성적
 - VggNet과 다른 점은 VggNet이 Gradient vanishing문제를 pre-training과 weight initialization으로 해결하기 위해 노력했던 것과 달리, 네트워크 구조를 수정하는 것으로 해결.
- 2014년 : VggNet - ImageNet Test(Top-5)에서 93.2%의 성공 2위 - 19개의 Conv filter를 사용

- VGG16
- VGG19
- 2015년 - ResNet – Deep Residual Learning for Image Recognition - 2015년 ILSVRC 우승
 - 152개의 Conv layer를 채용
 - ResNet50
- 2016년 - Xception
 - ImageNet test에서 79%(Top-1), 94.5%(Top-5)를 달성
- 2017년 - MobileNet(MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Application)
- InceptionV3
- 2017년 : DenseNet – Densely Connected Convolutional Networks
 - ResNet의 아이디어를 이어받음.
- 2017년 : ShuffleNet
- 2018년 : NasNet - Learning Transferable Architectures for Scalable Image Recognition
- 2019년 : EfficientNet – EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
 - 84.4%(Top-1)을 달성
 - 기존의 랭커 모델들보다 파라미터 수는 적으면서 정확도를 상회하는 EfficientNet이 등장. NasNet이 82.7%(Top-1)이었던 것에 비해 84.4%를 달성.(20.08.31)기준 88.5%로 달성한 SOTA 모델은 EfficientNet기반.

사전 훈련 네트워크 가져오기

In [28]:

```
from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                    include_top=False, # 완전 분류 연결기 포함/미포함
                    input_shape=(150, 150, 3))
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5)

58892288/58889256 [=====] - 2s 0us/step

58900480/58889256 [=====] - 2s 0us/step

- VGG16 함수에 세 개의 매개변수를 전달
 - weights는 모델을 초기화할 가중치 체크포인트를 지정
 - None, 'imagenet', weights의 path
 - include_top은 네트워크의 최상위 **완전 연결 분류기를 포함할지 안할지**를 지정.
 - 기본값은 ImageNet의 1,000개의 클래스에 대응되는 완전 연결 분류기를 포함.
 - 여기서는 별도의 (강아지와 고양이 두 개의 클래스를 구분하는) 완전 연결 층을 추가할 계획.
 - input_shape은 네트워크에 주입할 **이미지 텐서의 크기**입니다.
 - 이 매개변수는 선택사항입니다. 이 값을 지정하지 않으면 네트워크가 어떤 크기의 입력도 처리할 수 있습니다.

In [29]:

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 150, 150, 3)]	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688

Trainable params: 14,714,688

Non-trainable params: 0

- VGG16은 합성곱 13개, 완전 연결 층(FCL) 3개.
 - include_top이 기본값이 True일 경우 input_shape이 원본 모델과 동일한 (224, 224, 3)이 되어야 한다.
- 다른 모델 VGG19는 합성곱 층 16개, 완전 연결 층 3개로 이루어져 있음.

실습 : 데이터 증식을 사용하지 않는 빠른 특성 추출

In [30]:

```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

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

# 데이터 디렉터리
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

print("데이터 디렉터리 : ", train_dir, validation_dir, test_dir, sep="\n")
```

데이터 디렉터리 :
./datasets/cats_and_dogs_small/train
./datasets/cats_and_dogs_small/validation
./datasets/cats_and_dogs_small/test

이미지 제너레이터를 활용한 데이터 가져오기

In [36]:

```
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 20

def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count))

    print("피쳐, 레이블 : ", features.shape, labels.shape )
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary')

    # 이미지를 20개씩 가져오기(이미지 사이즈 :150, 150)
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        print( "추출된 피쳐 :", type(features_batch), features_batch.shape )

        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch

        i += 1
        if i * batch_size >= sample_count:
            # 제너레이터는 루프 안에서 무한하게 데이터를 만들어내므로
            # 모든 이미지를 한 번씩 처리하고 나면 중지합니다
            break

    return features, labels
```

In [38]:

```
# extract_features(데이터가 있는 디렉터리, 샘플 데이터 개수)
t_features, t_labels = extract_features(train_dir, 100)
print( t_features.shape, t_labels.shape )
```

```
피처, 레이블 : (100, 4, 4, 512) (100,)
Found 2000 images belonging to 2 classes.
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
(100, 4, 4, 512) (100,)
```

In [39]:

```
%%time

train_features, train_labels = extract_features(train_dir, 2000)
validation_features, validation_labels = extract_features(validation_dir, 1000)
test_features, test_labels = extract_features(test_dir, 1000)
```

```
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
피처, 레이블 : (1000, 4, 4, 512) (1000,)
Found 1000 images belonging to 2 classes.
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)

추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
추출된 피처 : <class 'numpy.ndarray'> (20, 4, 4, 512)
```

In [40]:

```
print( train_features.shape, train_labels.shape )
print( validation_features.shape, validation_labels.shape )
print( test_features.shape, test_labels.shape )
```

```
(2000, 4, 4, 512) (2000,)
(1000, 4, 4, 512) (1000,)
(1000, 4, 4, 512) (1000,)
```

- 추출된 특성(데이터)의 크기는 (samples, 4, 4, 512). 완전 연결 분류기에 주입하기 위해 먼저 (samples, 8192)크기로 펼치기

In [41]:

```
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))  
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))  
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

완전 연결 분류기 정의(FCL)

- 완전 연결 분류기(FCL)를 정의
- 신경망의 규제를 위해 드롭아웃을 사용.
- 저장된 데이터와 레이블을 사용해 훈련

In [42]:

```
from tensorflow.keras import models  
from tensorflow.keras import layers  
from tensorflow.keras import optimizers
```

In [43]:

```
%%time

# 완전 연결층
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```

Epoch 1/30

/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/rmsprop.py:130: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.

```
super(RMSprop, self).__init__(name, **kwargs)
```

```
100/100 [=====] - 2s 6ms/step - loss: 0.5977 - acc: 0.6600
- val_loss: 0.4487 - val_acc: 0.8210
```

Epoch 2/30

```
100/100 [=====] - 0s 4ms/step - loss: 0.4312 - acc: 0.8020
- val_loss: 0.3672 - val_acc: 0.8610
```

Epoch 3/30

```
100/100 [=====] - 0s 4ms/step - loss: 0.3550 - acc: 0.8515
- val_loss: 0.3247 - val_acc: 0.8820
```

Epoch 4/30

```
100/100 [=====] - 0s 4ms/step - loss: 0.3121 - acc: 0.8730
- val_loss: 0.3021 - val_acc: 0.8870
```

Epoch 5/30

```
100/100 [=====] - 0s 5ms/step - loss: 0.2865 - acc: 0.8895
- val_loss: 0.2856 - val_acc: 0.8900
```

Epoch 6/30

```
100/100 [=====] - 0s 5ms/step - loss: 0.2615 - acc: 0.8960
- val_loss: 0.2761 - val_acc: 0.8910
```

Epoch 7/30

```
100/100 [=====] - 0s 4ms/step - loss: 0.2496 - acc: 0.9045
- val_loss: 0.2662 - val_acc: 0.8940
```

Epoch 8/30

```
100/100 [=====] - 0s 4ms/step - loss: 0.2347 - acc: 0.9145
- val_loss: 0.2604 - val_acc: 0.8960
```

Epoch 9/30

```
100/100 [=====] - 0s 5ms/step - loss: 0.2232 - acc: 0.9215
- val_loss: 0.2589 - val_acc: 0.8970
```

Epoch 10/30

```
100/100 [=====] - 0s 5ms/step - loss: 0.2056 - acc: 0.9205
- val_loss: 0.2501 - val_acc: 0.9050
```

Epoch 11/30

```
100/100 [=====] - 1s 5ms/step - loss: 0.2013 - acc: 0.9295
- val_loss: 0.2536 - val_acc: 0.8990
```

Epoch 12/30

```
100/100 [=====] - 0s 5ms/step - loss: 0.1810 - acc: 0.9345
- val_loss: 0.2437 - val_acc: 0.9020
```

Epoch 13/30

```

100/100 [=====] - 1s 5ms/step - loss: 0.1761 - acc: 0.9355
- val_loss: 0.2476 - val_acc: 0.9010
Epoch 14/30
100/100 [=====] - 0s 5ms/step - loss: 0.1686 - acc: 0.9370
- val_loss: 0.2438 - val_acc: 0.8960
Epoch 15/30
100/100 [=====] - 0s 5ms/step - loss: 0.1623 - acc: 0.9415
- val_loss: 0.2375 - val_acc: 0.9070
Epoch 16/30
100/100 [=====] - 0s 4ms/step - loss: 0.1614 - acc: 0.9475
- val_loss: 0.2424 - val_acc: 0.9000
Epoch 17/30
100/100 [=====] - 0s 4ms/step - loss: 0.1500 - acc: 0.9520
- val_loss: 0.2398 - val_acc: 0.9040
Epoch 18/30
100/100 [=====] - 0s 4ms/step - loss: 0.1402 - acc: 0.9475
- val_loss: 0.2370 - val_acc: 0.9020
Epoch 19/30
100/100 [=====] - 0s 5ms/step - loss: 0.1382 - acc: 0.9485
- val_loss: 0.2589 - val_acc: 0.8890
Epoch 20/30
100/100 [=====] - 0s 4ms/step - loss: 0.1296 - acc: 0.9535
- val_loss: 0.2363 - val_acc: 0.9050
Epoch 21/30
100/100 [=====] - 0s 5ms/step - loss: 0.1247 - acc: 0.9585
- val_loss: 0.2391 - val_acc: 0.9040
Epoch 22/30
100/100 [=====] - 0s 5ms/step - loss: 0.1204 - acc: 0.9585
- val_loss: 0.2424 - val_acc: 0.9020
Epoch 23/30
100/100 [=====] - 0s 5ms/step - loss: 0.1192 - acc: 0.9610
- val_loss: 0.2361 - val_acc: 0.9020
Epoch 24/30
100/100 [=====] - 0s 5ms/step - loss: 0.1127 - acc: 0.9615
- val_loss: 0.2452 - val_acc: 0.9010
Epoch 25/30
100/100 [=====] - 0s 4ms/step - loss: 0.1063 - acc: 0.9660
- val_loss: 0.2400 - val_acc: 0.8990
Epoch 26/30
100/100 [=====] - 0s 5ms/step - loss: 0.1065 - acc: 0.9655
- val_loss: 0.2424 - val_acc: 0.9000
Epoch 27/30
100/100 [=====] - 0s 4ms/step - loss: 0.1013 - acc: 0.9660
- val_loss: 0.2392 - val_acc: 0.9030
Epoch 28/30
100/100 [=====] - 0s 4ms/step - loss: 0.0983 - acc: 0.9645
- val_loss: 0.2509 - val_acc: 0.9000
Epoch 29/30
100/100 [=====] - 0s 4ms/step - loss: 0.0896 - acc: 0.9715
- val_loss: 0.2401 - val_acc: 0.9050
Epoch 30/30
100/100 [=====] - 0s 5ms/step - loss: 0.0878 - acc: 0.9750
- val_loss: 0.2401 - val_acc: 0.9030
CPU times: user 16.3 s, sys: 1.2 s, total: 17.5 s
Wall time: 15.9 s

```

- FCL(완전 연결층)2개의 Dense 층만 처리하면 되기 때문에 훈련이 빠르다.
 - 합성곱 층의 가중치는 사전 훈련 네트워크 가중치 이용.
 - 데이터가 늘어날 경우 정확도가 올라간다.

훈련 손실과 정확도 곡선

In [44]:

```
import matplotlib.pyplot as plt

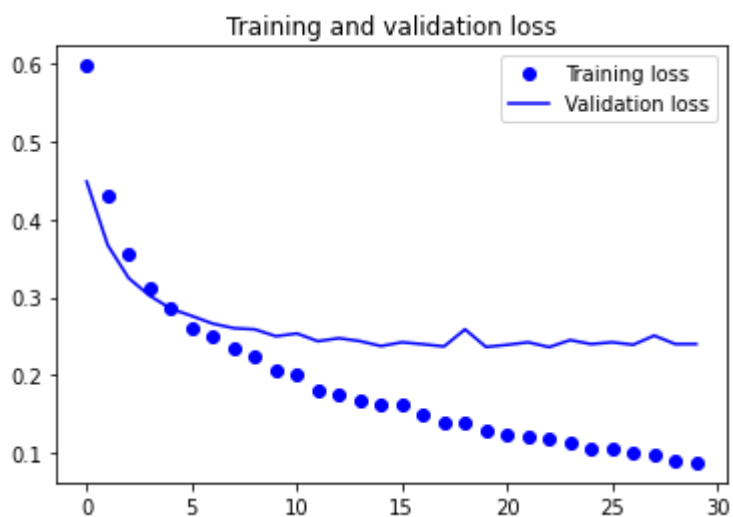
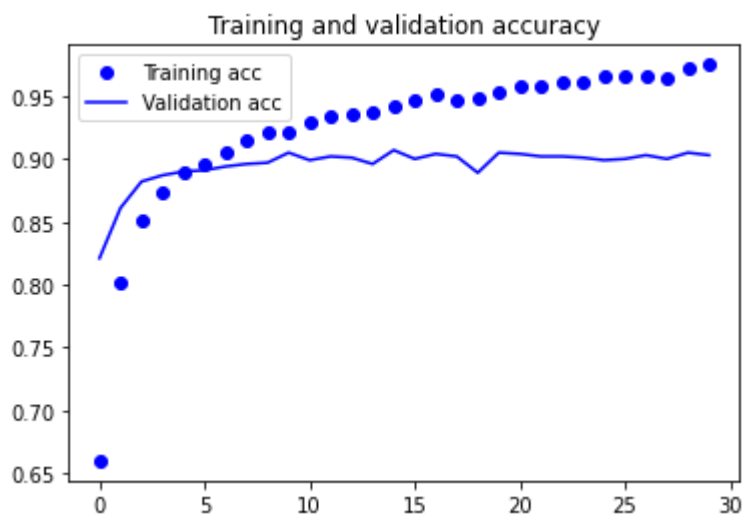
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

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()
```



Summary

- validation 90%의 검증 정확도에 도달
- 앞에서 처음부터 훈련시킨 작은 모델에서 얻은 것보다 훨씬 좋음.
- 드롭아웃을 사용했음에도 불구하고 훈련이 시작하면서 거의 바로 과대적합되고 있다는 것
 - 과대적합의 이유 **작은 이미지 데이터셋**에서는 과대적합을 막기 위해 필수적인 데이터 증식을 사용하지 않음.

REFERENCE

- keras applications : <https://keras.io/api/applications/> (<https://keras.io/api/applications/>)
- CNN의 이야기
 - <https://engineer-mole.tistory.com/43> (<https://engineer-mole.tistory.com/43>)
 - <https://junklee.tistory.com/111> (<https://junklee.tistory.com/111>)