2부 실전 딥러닝 - 5장 컴퓨터 비전을 위한 딥러닝

시작일	@2022년 7월 28일
링크	
목표	~~수/0727(27: 면접조율)~~
종료일	@2022년 7월 28일

5장 컴퓨터 비전을 위한 딥러닝

- 합성곱 신경망(컨브넷)의 이해
- 과대적합을 줄이기 위해 데이터 증식 기법 사용
- 특성 추출을 위해 사전 훈련된 컨브넷 사용
- 사전 훈련된 컨브넷을 미세 조정하는 방법
- 컨브넷에서 학습된 것과 컨브넷의 분류 결정 방식을 시각화하는 방법

5.1 합성곱 신경망 소개

- 컨브넷(convnet) : 합성곱 신경망(convolutional neural network)
  - 컴퓨터 비전(computer vision)에 사용
- 기본적인 컨브넷
  - (image_height, image_width, image_channels) 크기의 입력 텐서를 사용 / 배치 텐서 사용 X
  - MNIST 이미지 (28,28,1), 첫 번째 층의 매개변수로 input_shape = (28,28,1)
  - 첫 번째 층의 매개변수로 input_shape = (28,28,1)을 전달

```
# 간단한 컨브넷 만들기
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape = (28, 28, 1)))
```

```
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(64,(3,3), activation = 'relu'))
```

```
model.summary()
✓ 0.1s
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_4 (Conv2D)           (None, 3, 3, 64)          36928

=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

- Conv2D 와 MaxPooling2D 층의 출력은 (height, width, channels) 크기의 3D 텐서

```
# 컨브넷 위에 분류기 추가하기
  # Dense 층을 쌓은 분류기 추가
  # Dense 층은 1D 벡터를 처리
  # 이전 층의 출력이 3D 텐서
    # 3D텐서 -> 1D 텐서로 펼침
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(10, activation = 'softmax')) # 10개의 클래스로 분류
```

```
# MNIST 이미지에 컨브넷 훈련하기
from keras.dataserts import mnist
from keras.utils import to_categorical

(train_images, train_labels, test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000,28,28,1))
train_images= train_images.astype('float32') / 255

test_images = test_images.reshape((10000,28,28,1))
test_images = test_images.astype('float32') /255
```

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer = 'rmsprop',
              loss = 'categorical _crossentropy',
              metrics= ['accuracy'])

model.fit(train_images, train_lables, epochs = 5, batch_size = 64)

test_loss, test_acc = model.evaluate(test_images, test_labels)
```

5.1.1 합성곱 연산

- Dense
  - 입력 특성 공간에 있는 전역 패넡을 학습
- Conv2D
  - 지역 패턴을 학습
  - 2D 윈도우(window)로 입력에서 패턴을 찾음
- MaxPooling2D

컨브넷의 특징

- 학습된 패턴은 `평행 이동 불변성(translation invariant)`를 가짐
  - 평행 이동으로 인해 다르게 인식되지 않음
  - Dense layer - 새로운 위치에 나타나면 새로운 패턴으로 학습해야 함
- 컨브넷은 패턴의 `공간적 계층 구조`를 학습할 수 있음
  1. 첫 번째 합성곱 층이 - 엣지 같은 지역 패턴 학습
  2. 두 번째 합성곱 층 - 첫 번째 층의 특성으로 구성된 더 큰 패턴을 학습
     - 컨브넷 - 매우 복잡하고 추상적인 시각적 개념을 학습

합성곱 연산

- `특성 맵(feature map)` 3D 텐서에 적용됨
  - 높이, 너비, 깊이(채널 축) / RGB: 3, Grayscale : 1

    - 특성 맵에서 작은 패치(patch)들을 추출 → 모든 패치에 같은 변환을 적용 → 출력 특성 맵(output feature map)을 만듦
- `출력 특성 맵(output feature map)`
    - 높이, 너비를 가진 3D 텐서
    - 출력 텐서의 깊이 : 층의 매개변수로 결정
    - 대신 필터(filter)를 의미함
    - `필터(filter)` : 입력 데이터의 어떤 특성을 인코딩함
    - 필터를 적용한 출력 채널 → 응답 맵(response map)
- 핵심 파라미터
    - Conv2D(output_depth, (window_height, window_width)
    - 특성 맵의 출력 깊이 : output_depth
        - 합성곱으로 계산할 필터의 수
        - ex. 32 → 64
    - 입력으로부터 뽑아낼 패치의 크기 : (window_height, window_width)
        - ex. (3 x 3) / (5 x 5)
    - 윈도우가 슬라이딩(sliding)하면서 모든 위치의 3D 특성 패치를 추출
    - 3D 패치 → 1D 벡터로 변환됨
        - 합성곱 커널(convolution kernel): 하나의 학습된 가중치 행렬 * 텐서 곱셈을 통해 변환됨
        - 변환된 모든 벡터는 (height, width, output_depth) 크기의 3D 특성 맵으로 재구성됨
        - 출력 높이와 너비는 입력의 높이, 너비와 다를 수 있음
- 경계 문제와 패딩 이해하기
    - 입력과 동일한 높이 & 너비를 가진 출력 특성 맵 : 패딩(padding)
    - Conv2D 층의 패딩 : padding 매개변수로 설정
        - valid(default) : 패딩을 사용하지 않는다는 뜻
        - same: 입력과 동일한 높이와 너비를 가진 출력을 만들기 위해 패딩함
- 합성곱 스트라이드 이해하기

- 출력 크기에 영향을 미침 : 스트라이드(stride)
- default: 1
- 1> : 스트라이드 합성곱도 가능
  - stride 2 : 특성 맵의 너비 & 높이가 2의 배수로 다운샘플링
  - 특성 맵의 다운샘플링, 스트라이드 대신 최대 풀링(max pooling) 연산 사용하는 경우도 있음

5.1.2. 최대 풀링 연산

- MaxPooling2D: 특성 맵 다운샘플링
  - 입력 특성 맵에서 윈도우에 맞는 패치를 추출
  - 각 채널별로 최댓값을 출력

```
# No max pooling
  # 맥스 풀링을 하지 않을 경우
  # 다운 샘플링이 X
  # 특성 맵의 가중치 개수를 줄이기 위함
    # 다운 샘플링을 하기 위해 stride / average pooling 등을 사용할 수 있음
    # but, max pooling 이 가장 잘 작동하는 편임

model_no_max_pool = models.Sequential()
model_no_max_pool.add(layers.Conv2D(32, (3,3), activation = 'relu', input_shape =
 (28,28,1)))
model_no_max_pool.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model_no_max_pool.add(layers.Conv2D(64, (3,3), activation = 'relu'))

# 모델의 구조
  # 특성의 공간적 계층 구조 학습하는데 도움 X :
  # 마지막 합성곱 층의 특성이 전체 입력에 대한 정보를 가지고 있어야 함
  # 최종 특성 맵
    # 22 x 22 x 64 = 30,976
    # 컨브넷 펼친 후 512 Dense 층과 연결, 약 15,8백만 개의 가중치 파라미터가 생김
    # 작은 모델치고는 너무 많은 가중치고, 심각한 과대적합 발생
model_no_max_pool.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

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

 conv2d_2 (Conv2D)           (None, 22, 22, 64)        36928

=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련하기

- 작은 데이터셋 : 수백 개 ~ 수만 개
- 컴퓨터 비전에서 과대적합을 줄이기 위한 강력한 방법 : 데이터 증식 (data augmentation)
  1. 사전 훈련된 네트워크로 (특성을 추출하는 것)
  2. 사전 훈련된 네트워크를 (세밀하게 튜닝하는 것)
  3. 모델 학습
     a. 처음부터 작은 모델을 훈련
     b. 사전 훈련된 모델을 사용하여 특성 추출하기
     c. 사전 훈련된 모델을 세밀하게 튜닝하기

5.2.1 작은 데이터셋 문제에서 딥러닝의 타당성

5.2.2 데이터 내려받기

- 강아지 vs 고양이 데이터셋 케라스에 포함 X : https://www.kaggle.com/c/dogs-vs-cats/data
- 2만 5천개의 강아지 & 고양이 이미지 & 543MB

- 훈련 세트 : 클래스마다 1,000개의 샘플로 이루어짐
- 검증 세트: 클래스마다 500개
- 테스트 세트 : 클래스마다 500개의 샘플로 이루어짐

```
# 훈련, 검증 , 테스트 폴더로 이미지 복사하기
import os, shutill

#-------------------------------------------------------------
# original_dataset_dir - 원본 데이터셋을 압축 해제한 디렉터리 경로
# base_dir - 소규모 데이터셋을 저장할 디렉터리
original_datasert_dir = './datasets/dogs_and_cats/train' # 원본 데이터셋을 압축 해제한 디렉토
리 경로

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

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

test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)
#-------------------------------------------------------------
# 훈련용/검증용/테스트용 고양이 사진 디렉터리
train_cats_dir = os.path.join(train_dir, 'cats')
os.mkdir(train_cats_dir)

validation_cats_dir = os.path.join(validation_dir, 'cats')
os.mkdir(validation_cats_dir)

test_cats_dir = os.path.join(test_dir, 'cats')
os.mkdir(test_cats_dir)
#-------------------------------------------------------------
# 훈련용/검증용/테스트용 강아지 사진 디렉터리
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.mkdir(train_dogs_dir)

validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.mkdir(validation_dogs_dir)

test_dogs_dir= os.path.join(test_dir, 'dogs')
os.mkdir(test_dogs_dir)
#-------------------------------------------------------------
# 처음 1,000개의 고양이 이미지를 train_cats_dir에 복사
# 다음 500개의 고양이 이미지를 validation_cats_dir 에 복사
# 다음 500개의 고양이 이미지를 test_cats_dir에 복사
fnames = ['cat'.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
  src = os.path.join(original_dataset_dir, fname)
  dst = os.path.join(train_cats_dir, fname)
  shutil.copyfile(src,dst)
```

```
fnames = ['cat'.{}.jpg'.format(i) for i in range(1000,1500)]
for fname in fnames:
  src = os.path.join(original_dataset_dir, fname)
  dst = os.path.join(validation_cats_dir, fname)
  shutil.copyfile(src,dst)

fnames = ['cat'.{}.jpg'.format(i) for i in range(1500,2000)]
for fname in fnames:
  src = os.path.join(original_dataset_dir, fname)
  dst = os.path.join(test_cats_dir, fname)
  shutil.copyfile(src,dst)
#------------------------------------------------------------
# 처음 1,000개의 강아지 이미지를 train_dogs_dir에 복사
# 다음 500개의 강아지 이미지를 validation_dogs_dir 에 복사
# 다음 500개의 강아지 이미지를 test_dogs_dir에 복사
fnames = ['dog'.{}.jpg'.format(i) for i in range(1000)]
for fname in fnames:
  src = os.path.join(original_dataset_dir, fname)
  dst = os.path.join(train_cats_dir, fname)
  shutil.copyfile(src,dst)

fnames = ['dog'.{}.jpg'.format(i) for i in range(1000,1500)]
for fname in fnames:
  src = os.path.join(original_dataset_dir, fname)
  dst = os.path.join(validation_cats_dir, fname)
  shutil.copyfile(src,dst)

fnames = ['dog'.{}.jpg'.format(i) for i in range(1500,2000)]
for fname in fnames:
  src = os.path.join(original_dataset_dir, fname)
  dst = os.path.join(test_cats_dir, fname)
  shutil.copyfile(src,dst)
#------------------------------------------------------------
# 최종 확인
  # 2,000개의 훈련 이미지
  # 1,000개의 검증 이미지
  # 1,000개의 테스트 이미지
  # 각 데이터는 클래스마다 동일한 개수의 샘플을 포함함
print('훈련용 고양이 이미지 전체 개수', len(os.listdir(train_cats_dir))  # 1,000
print('검증용 고양이 이미지 전체 개수', len(os.listdir(validation_cats_dir)) # 500
print('테스트용 고양이 이미지 전체 개수', len(os.listdir(ttest_cats_dir)) # 500

print('훈련용 강아지 이미지 전체 개수', len(os.listdir(train_dogs_dir)) # 1,000
print('검증용 강아지 이미지 전체 개수', len(os.listdir(validation_dogs_dir)) # 500
print('테스트용 강아지 이미지 전체 개수', len(os.listdir(test_dogs_dir)) # 500
```

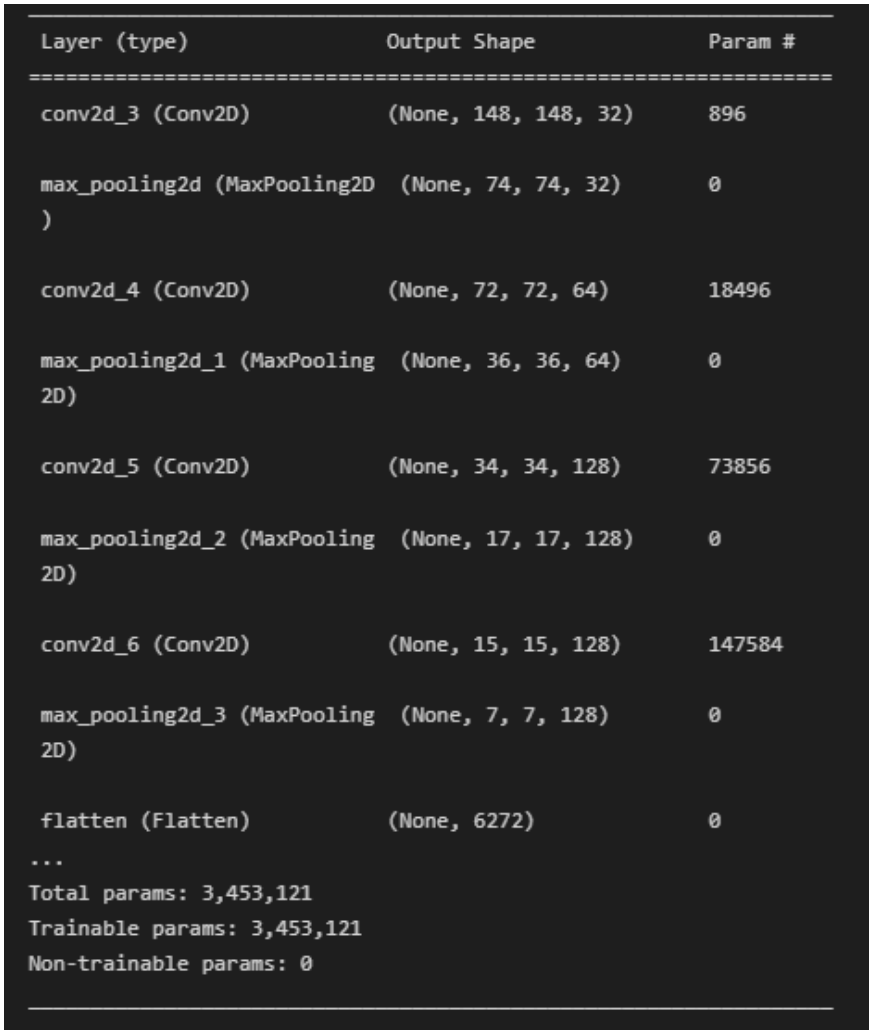
5.2.3 네트워크 구성하기

일반적인 컨브넷

- Conv2D(relu 활성화 함수 사용) + MaxPooling2D 층
- Conv2D + MaxPooling2D 단계를 추가하면

- → 네트워크 용량 늘림
- → Flatten 층의 크기가 너무 커지지 않도록
- → 특성 맵의 크리를 줄일 수 있음
- 150 x 150 크기 입력으로 시작해 Flatten 층 이전에 7 x 7 크기의 특성 맵으로 줄어듦
  - `특성 맵의 깊이 → 네트워크에서 점진적으로 증가`
  - `특성 맵의 크기 → 감소`

```
# 강아지 vs 고양이 분류를 위한 소규모 컨브넷 만들기
  # 이진 분류 컨브넷 (convolutional network)
  # 네트워크는 하나의 유닛 -> 크기가 1인 Dense
  # sigmoid 활서오하 함수로 끝
  # 이 유닛 -> 한 클래스에 대한 확률로 인코딩

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'))
```



```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)            (None, 148, 148, 32)      896

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

 conv2d_4 (Conv2D)            (None, 72, 72, 64)        18496

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

 conv2d_5 (Conv2D)            (None, 34, 34, 128)       73856

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

 conv2d_6 (Conv2D)            (None, 15, 15, 128)       147584

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

 flatten (Flatten)            (None, 6272)              0
...
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

```
# 모델의 훈련 설정하기
  # 옵테미아저: RMSprop
  # 네트워크의 마지막이 하나의 시그모이드 유닛
  # 이진 크로스엔트로피(binary crossentropy) 를 손실로 사용
from keras import optimizers

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

5.2.4 데이터 전처리

- JPEG → 부동 소수 타입의 텐서로 적절하게 전처리
  - 사진 파일을 읽음
  - JPEG 콘텐츠를 RGB 픽셀 값으로 디코딩함
  - 그 다음 부동 소수 타입의 텐서로 변환

    - 픽셀 값 (0~255) → [0,1] 사이로 조정함 (신경망은 작은 입력 값을 선호함)
- 헬퍼 유틸리티
    - `keras.preprocessing.image`
    - `ImageDataGenerator()`
        - 디스크에 있는 이미지 파일을 전처리된 배치 텐서로 자동으로 바꾸어 주는 `파이썬 제너레이터`를 만듦

```
# 파이썬 제너레이터 이해하기
  # generator는 반복자(iterator) 처럼 작동하는 객체
  # for... in 연산자에 사용할 수 있음
  # 제너레이터는 yield 연산자를 사용하여 만듦

# 정수를 반환하는 제너레이터
def generator():
  i = 0
  while True:
    i += 1
    yield i

for item in generator():
  print(item)
  if item > 4:
    break

# 출력
# 1
# 2
# 3
# 4
# 5
```

```
# ImageDataGenerator 를 사용하여 디렉터리에서 이미지 읽기
from keras.preprocessing.image import ImageDataGenerator

# 모든 이미지를 1/255로 스케일 조정
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 x 150 크기로 바꿈
          batch_size = 20,
          class_mode = 'binary') # binary_crossentropy 손실을 사용하기 때문에 이진 레이블
이 필요

validation_generator = test_datagen.flow_from_directory(
          validation_dir,
          target_size = (150, 150),
```

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

```
# 제너레이터의 출력
  # 출력: 150 x 150 RGB 이미지 배치 ((20,150,150,3)) 크기 / 이진 레이블 배치 ((20,), 크기)
- # 각 배치의 20개의 샘플(배치 크기)
  # 제너레이터는 이 배치를 무한정 만들어냄
  # 타깃 폴더에 있는 이미지를 끝없이 반복
  # break 문을 사용해야 끝남

for data_batch, labels_batch in train_Generator:
  print("배치 데이터 크기", data_batch.shape) # (20,150,1503)
  print("배치 레이블 크기", labels_batch.shape) # (20,)
  break

# 제너레이터를 사용한 데이터에 모델 훈련
  # fit_generator 메서드
  # fit 메서드와 동일하되, 데이터 제너레이터 사용
  # 첫 번째 매개변수로 입력과 타깃의 배치를 끝없이 반환
  # 데이터가 끝없이 생성됨

    # 케라스 모델에 하나의 에포크를 정의
      # -> 제너레이터로부터 얼마나 많은 샘플을 뽑을지 알려줘야 함
      # -> steps_per_epoch 매개변수에서 이를 설정
      # 제너레이터, steps_per_epoch개의 배치만큼 뽑은 후,
      # stpes_per_epoch 횟수만큼 경사 하강법 단계를 실행한 후
      # 훈련 프로세스는 다음 에포크로 넘어감
        # 여기서 20개의 샘플이 하나의 배치
        # 2,000개의 샘플을 모두 처리할 때까지 100개의 배치를 뽑을 것

  # fit_generator
    # fit과 마찬가지로 validation_data 매개변수로 전달
    # 매개변수에는 데이터 제너레이터도 가능하지만, 넘파이 배열의 튜플도 가능
    # validation_data 로 제너레이터를 전달하면 검증 데이터의 배치가 끝없이 반환됨
    # 따라서, 검증 데이터 제너레이터에서 얼마나 많은 배치를 추출하여 평가할지
      # validation_steps 매개변수에 지정해야 함
      # validation_generator 배치가 20개로 지정되어, 전체 검증 데이터 (1,000)개를 사용
      # validation_steps 는 50으로 설정

# 배치 제너레이터를 사용하여 모델 훈련하기
history = model.fit_generator(
      train_generator,
      steps_per_epoch = 100,
      epochs = 30,
      validation_data = validation_generator,
      valdiation_steps = 50)
```

```
# 훈련 후 모델 저장
  # 모델 저장하기

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

```
# 훈련 데이터 & 검증 데이터에 대한 모델의 손실 & 정확도를 그래프로 그림

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(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.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()
```

- 과대적합됨 (훈련 정확도 : 선형적으로 증라 100% / 검증 정확도 70~72%에 멈춤)
- 비교적 훈련 샘플의 수가 적기 때문에 과대적합이 가장 중요한 문제 / 드롭아웃 & 가중치 감소처럼 과대적합 감소할 수 있는 여러 기법 배움

5.2.5 데이터 증식 사용하기

- 데이터 증식: 기존 훈련 샘플로부터 더 많은 훈련 데이터를 생성하는 방법
- 그럴듯한 이미지를 생성하도록 여러 가지 랜덤한 변환을 적용하여 샘플을 늘림 : ImageDataGenerator

```
# ImageDataGenerator 를 사용하여 데이터 증식 설정하기
datagen = ImageDataGenerator(
        rotation_range = 20,  # 랜덤하게 사진을 회전시킬 각도 범위 (0~180)
        width_shift_range = 0.1, # 사진을 수평으로 랜덤하게 평행 이동시킬 범위(전체 너비에 대한 비율)
        height_shift_range = 0.1, # 사진을 수직으로 랜덤하게 평행 이동시킬 범위(전체 높이에 대한 비율)
        shear_range = 0.1, # 랜덤하게 전단 변환(shearing transformation)을 적용할 각도 범위
        zoom_range = 0.1, # 랜덤하게 사진을 확대할 범위
        horizontal_flip = True, # 랜덤하게 이미지를 수평으로 뒤집음, 수평 대칭을 가정할 수 있을 때 사용함 (풍경/인물사진)
        fill_mode= 'nearest') # 회전이나 가로/세로 이동으로 인해 새롭게 생성해야 할 픽셀을 채울 전략
```

```
# rotation_range : - rotation_range ~ + rotation_range
# width_shift_range, height_shift_range > 1, 1보다 클 경우 실수/정수일 때 픽셀 값으로 간주됨
# shear_range: rotation_range로 회전할 때, y 축 방향으로 각도를 증가시켜 이미지 변형
# fill_mode = 'nearest' 는 인접 픽셀을 사용
```

```
# 랜덤하게 증식된 훈련 이미지 그리기
from keras.preprocessing import image # 이미지 전처리 유틸리티 모듈

fnames = sorted([os.path.join(train_cats_dir, fname) for
      fname in os.lisdir(train_cats_dir)])

img_path = fnames[3] # 증식할 이미지를 선택합니다

img = image.load_img(img_path, target_size = (150,150)) # 이미지를 읽고 크기를 변경합니다
x = image.img_to_array(img) # (150,150,3)크기의 넘파이 배열로 변환
x = x.reshape((1,) + x.shape) # (1,150,150,3) 크기로 변환

i = 0
for batch in datagen.flow(x, batch_size = 1) :
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    # flow() 메서드는 배치 데이터를 기대하기 때문에 샘플 데이터에 배치 차원을 추가하여 4D 텐서로 만듦
    i += 1
    if i % 4 == 0:
      break

plt.show() # 랜덤한 데이터 증식으로 생성된 고양이 사진
```

```
# 과대적합을 더 억제하기 위해 완전 연결 분류기 직전에 Dropout 층을추가
# 드롭아웃을 포함한 새로운 컨브넷 정의하기
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',
              optimizer = optimizers.RMSprop(lr= le-4),
              metrics = ['acc'])
```

```
# 데이터 증식 & 드롭아웃 사용하여 네트워크 훈련
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,)

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

train_generator = train_datagen.flow_from_directory(
    train_dir, # 타깃 디렉터리
    target_size = (150,150), # 모든 이미지를 150 x 150 크기로 바꿈
    batch_size = 32,
    class_mode = 'binary') # binary_crossentropoy 손실 사용, 이진 레이블 만듦

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size = (150,150),
    batch_size = 32,
    class_mode= 'binary')

history = model.fit_generator(
    train_generator,
    steps_per_epoch = 100,
    epochs = 100,
    validation_d* ata = validation_generator,
    validation_steps = 50)

# 모델 저장
model.save('cats_and_dogs_small_2.h5')
```

- 데이터 증식 & 드롭아웃 → 과대적합 방지
- 다른 규제 기법 & 파라미터 튜닝(ex. 합성곱 층의 필터 수, 네트워크 층 수) → 더 높은 정확도 얻음

5.3 사전 훈련된 컨브넷 사용하기

- 작은 이미지 데이터셋에 딥러닝 적용 → 사전 훈련된 네트워크 많이 사용함
- 사전 훈련된 네트워크(pre-trained network), 대규모 이미지 분류 문제를 위해 대량의 데이터셋에서 미리 훈련되어 저장된 네트워크
- ImageNet (동물/생활 용품) 데이터로 네트워크를 훈련함
- 네트워크를 이미지에서 가구 아이템을 식별하는 등 다른 용도로 사용 가능
- 2014: VGG16 구조 사용 / ImageNet 데이터셋에 널리 사용되는 컨브넷 구조

- ex. VGG16, ResNet, Inception, Inception-ResNet, Xception
- 사전 훈련된 네트워크 사용
  - 특성 추출(feature extraction)
  - 미세 조정(fine-tuning)

5.3.1 특성 추출

- 특성 추출: 사전에 학습된 네트워크의 표현을 사용해 새로운 샘플에서 흥미로운 특성 뽑아내는 것
- 컨브넷 구조 - 1. 합성곱 & 2. 풀링층 & 3. 완전 연결 분류기
- 첫 번째 부분을 모델의 합성곱 기반 층(convolution base)만 사용함 & 분류기를 바꿈
  - 컨브넷의 특성 맵: 사진에 대한 일반적인 콘셉트의 존재 여부를 기록한 맵 & 일반적이기 때문에 다사용 가능
  - 특정 합성곱 층 추출한 표현의 일반성(재사용성) 수준은 모델에 있는 층의 깊이에 달려 있음
    - 모델의 하위 층 (edge, color, texture)
    - 모델의 상위 층('강아지 눈', '고양이 귀') 처럼 더 추상적인 개념 추출
    - 새로운 데이터셋이 원본 모델이 훈련한 데이터셋과 많이 다르면 전체 합성곱 기반 층보다 모델의 하위 층 몇개만 사용하는 것이 좋음
- ImageNet 클래스 집합
  - 여러 고양이 & 강아지
  - 원본 모델의 완전 연결 층에 있는 정보 재사용 가능
  - But, 새로운 문제의 클래스가 원본 모델의 클래스 집합과 겹치지 않는 일반적인 경우를 위해 완전 연결층 사용 X
  - ImageNet 데이터셋에 훈련된 VGG16 네크워트의 합성곱 기반 층 사용해 강아지 & 고양이 특성 추출
- VGG16 모델
  - 케라스 패키지 내장
  - keras.applications 모듈에서 임포트
  - keras.applications 모듈에서 사용 가능한 이미지 분류 모델은
    - Xception

- Inception V3
- ResNet50
- VGG16
  - 합성곱 13개, 완전 연결 층 3개
- VGG19
  - 합성곱 16개, 완전 연결 층 3개
- MobileNet

```
# VGG16 합성곱 기반 층 만들기
  # --------------------------weights--------------------------
    # 모델을 초기화할 가중치 체크포인트 지정
  # --------------------------include_top--------------------------
    # 네트워크의 최상위 완전 연결 분류기를 포함할지 안할지 정함
    # 기본값: ImageNet 클래스 1,000개 대응하는 완전 분류기를 포함함
    # 별도의 강아지 & 고양이 2개이ㅡ 클래스 구분하는 완전 연결 층을 추가 -> 포함시키지 않았음
    # include_top : True 면, 합성곱 층 위에 완전 연결 층이 추가되어 -> input_shape = (224,
224,3)이 되어야 함
  # --------------------------input_shape--------------------------
    # 네트워크에 주입할 이미지 텐서의 크기
    # 이 매개변수: 선택 사항
    # 이 값을 지정하지 않으면, 네트워크가 어떤 크기의 입력도 처리함
from keras.applications import VGG16

conv_base = VGG16(weights = 'imagenet',
                  include_top = False,
                  input_shape = (150,150,3))

conv_base.summary() # 모델 구조 확인
```

```
from tensorflow.keras.applications import VGG16
conv_base = VGG16(weights= 'imagenet',include_top = False,input_shape = (150,150,3))

✓ 7.5s

Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
```

```
conv_base.summary()
✓ 0.4s

Output exceeds the size limit. Open the full output data in a text editor
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
...
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

`최종 특성 맵의 크기 : (4,4,512) → 이 특성 위에 완전 연결 층 놓음`

1. 새로운 데이터셋에서 합성곱 기반 층을 실행 → 출력을 넘파이 배열로 디스크에 저장
    a. 독립된 완전 연결 분류기에 입력으로 사용함
    b. 합성곱 연산은 전체 과장 중 가장 cost
    c. 모든 입력 이미지에 대해 합성곱 기반 층을 한 번만 실행, 빠르고 비용이 적게 듦
    d. 하지만, 이 기법에는 `데이터 증식을 사용할 수 없음`
2. 준비한 모델(conv_base)위에 Dense 층을 쌓아 확장

a. 입력 데이터에서 엔드-투-엔드로 전체 모델을 실행

b. 모델에 노출된 모든 입력 이미지가 매번 합성곱 기반 층을 통과

c. `데이터 증식 사용할 수 있음`

d. 이 방식은 첫번째보다 훨씬 비용이 많이 듦

데이터 증식을 사용하지 않는 빠른 특성 추출

- ImageDataGenerator 를 사용하여 레이블을 넘파이 배열로 추출
- conv_base 모델의 predit 메서드를 호출하여 이미지에서 특성을 추출함

```
# 사전 훈련된 합성곱 기반 층을 사용한 특성 추출하기
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

base_dir = './datasets/cats_and_dogs_small'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

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))
  generator = datagen.flow_from_directory(
    directory,
    target_size = (150,150),
    batch_size = batch_size,
    class_mode = 'binary')

  i = 0
  for inputs_batch, labels_batch in generator:
    features_batch = conv_base.predict(inputs_batch)
    feature[i * batch_size : (i+1) * batch_size] = features_batch
    labelse[i * batch_size : (i+1) * batch_size] = labels_batch
    i += 1
    if i * batch_size >= simple_count:
      break # 제너레이터는 루프 안에서 무한하게 데이터를 만들어 내, 모든 이미지 한번씩 처리하고 나면 중지
함
  return features, labels

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

# 추출된 특성의 크기 (samples, 4, 4, 512)
# 완전 연결 분류기에 주입하기 위해 먼저 (samples, 8192)로 펼침
```

```
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))

# 그리고 완전 연결 분류기를 정의(규제 위한 드롭아웃 사용)
# 저장된 데이터 & 라벨 훈련
```

```
# 완전 연결 분류기를 정의하고 훈련하기
from keras import models
from keras import layers
from keras import optimizers

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))

# 2개의 Dense 층만 처리하면 되기 때문에 훈련이 매우 빠름
# CPU 를 사용하더라도 한 에포크 1초 미만
```

```
# 결과 그래프 그리기
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(1, len(acc) + 1)

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

plt.show()

# 90%의 검증 정확도에 도달
# 처음부터 훈련시킨 작은 모델에서 얻은 것보다 훨씬 좋음
# 그래프는 많은 비율로 드롭아웃을 사용했음에도, 훈련 시작하면서 거의 바로 과대적합
# 작은 이미지 데이터셋에서는 -> 과대적합 막기 위해 -> 데이터 증식 필요
```

데이터 증식을 사용한 특성 추출

- 훨씬 느리고 비용이 많이 들지만, 데이터 증식 기법을 사용할 수 있음
- conv_base 모델을 확장하고, 입력 데이터를 사용하여 엔드-투-엔드로 실행

```
# 모델에 층과 동일하게 작동되어, 층 추가하듯 Sequential 모델에 conv_base같은 다른 모델 추가 가능
# 합성곱 기반 층 위에 완전 연결 분류기 추가하기
from keras import models
from keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation ='relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))

model.summray()
```

- VGG16 : 14,714,688 개의 많은 파라미터 가짐
- 합성곱 기반 층 위에 추가한 분류기는 200만 개의 파라미터 가짐
- 모델을 컴파일하고 훈련하기 전에 합성곱 기반 층을 `동결`하는 것이 중요함
  - `동결(freezing) : 훈련하는 동안 가중치가 업데이트되지 않도록 막음`
  - 이렇게 하지 않으면 사전에 학습된 표현이 훈련하는 동안 수정됨
  - 맨 위의 Dense 층은 랜덤하게 초기화, 매우 큰 가중치 업데이트 값이 네트워크에 전파될 것 → 사전 학습된 표현을 훼손함
  - 케라스 → trainable 속성을 False → 네트워크 동결

```
#-----------------------------------------------
# 동결 전
print('conv_base 를 동결하기 전 훈련되는 가중치의 수: ',
      len(model.trainable_weights))
# conv_base를 동결하기 전 훈련되는 가중치의 수 : 30

#-----------------------------------------------
# 동결 후
conv_base.trainable = False
print('conv_base를 동결한 후 훈련되는 가중치의 수: ',
      len(model.trainable_weights))

# conv_base를 동결한 후 훈련되는 가중치의 수 : 4
```

- 동결: 추가한 2개의 Dense 층 가중치만 훈련
- 층마다 2개씩 (가중치 행렬 & 편향 벡터) 총 4개의 텐서가 훈련됨
- 변경 사항 적용: 먼저 모델을 컴파일 해야 함
- 컴파일 단계 후 trainable 속성을 변경하면 반드시 모델을 다시 컴파일행 햐 함
- 그렇지 않으면 변경 사항 적용되지 않음

```
# 동결된 합성곱 기반 층과 함께 모델을 엔드-투-엔드로 훈련하기
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers

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,
    flip_mode= 'nearest')
```

```
test_datagen = ImageDataGenerator(rescale= 1./255) # 검증 데이터 증식 X

train_generator = train_datagen.flow_from_directory(
      train_dir, # 타깃 디렉터리
      target_size = (150, 150), # 모든 이미지의 크기를 150 x 150 으로 변경
      batch_size= 20,
      class_mode= 'binary') # binary_crossentropy 손실을 사용하므로 이진 레이블이 필요함

validation_generator = test_datagen.flow_from_directory(
      validation_dir,
      target_size = (150,150),
      batch_size = 20,
      class_mode= 'binary')

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

history = model.fit_generator(
    train_generator,
    steps_per_epoch = 100,
    epochs = 30,
    validation_data = validation_generator,
    validataion_steps = 50,
    verbose = 2)
```

5.3.2 미세 조정

- 모델 재사용: 또 하나는 특성 추출을 보완하는 `미세 조정(fine-tuning)`
- 미세 조정
  - 특성 추출에 사용했떤 동결 모델의 상위 층 몇 개를 동결해서 해제하고
  - 모델에 새로 추가한 층(완전 연결 분류기) 함께 훈련
- 주어진 문제에 밀접하게 `재사용 모델의 표현을 일부 조정 → 미세 조정`
- 앞서 랜덤하게 초기화된 상단 분류기를 훈련하기 위해 VGG16 합성곱 기반 층을 동결
- 맨 위에 있는 분류기가 훈련된 후에 합성곱 기반의 상위 층을 미세조정할 수 있음
- 분류기가 미리 훈련되지 않으면 훈련되는 동안 머누 큰 오차 신호가 네트워크에 전파됨
- 미세 조정될 층들이 사전에 학습한 표현들을 망가뜨림
- 네트워크를 미세 조정하는 단계는 :
  - 사전에 훈련된 기반 네트워크 위에 새로운 네트워크를 추가함
  - 기반 네트워크를 동결함
  - 새로 추가한 네트워크를 훈련함

    - 기반 네트워크에서 일부 층의 동결을 해제함
    - 결을 해제한 층과 새로 추가한 층을 함께 훈련함
- 처음 세 단계는 특성 추출 할때 이미 완료함
- 네 번째 단계를 진행 , conv_base 동결해제하고 개별 층 동결
- conv_base.summary()

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

- 마지막 3개의 합성곱 층을 미세 조정함

- block4_pool 까지는 모든 층은 동결되고 block5_conv1, block5_conv2, block5_conv3 층은 학습 대상이 됨
- 왜 더 많은 층을 미세 조정하지 않을까요?
  - 왜 전체 합성곱 기반 층을 미세조정 하지 않을까요?
    - 합성곱 기반 층의 하위 층: 좀 더 일반적이고 재사용 가능한 특성들을 인코딩 함
    - 반면 상위 층은 좀 더 특화된 특성을 인코딩함
    - 새로운 문제에 재활용하도록 수정이 필요한 것은 구체적인 특성: 이들을 미세 조정하는 것이 유리함
    - 하위 층으로 갈수록 미세 조정에 대한 효과가 감소함
    - 훈련해야 할 파라미터가 많을수록 과대적합의 위험이 커짐
      - 합성곱 기반 층 1,500만 개의 파라미터 가짐
      - 작은 데이터셋으로 전부 훈련하려면 매우 위험함
      - 그러므로, 합성곱 기반 층에서 최상위 2~3개의 층만 미세 조정하는 것이 좋음

```
# 특정 층까지 모든 층 동결하기
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
  if layer.name == 'block5_conv1':
    set_trainable = True
  if set_trainable:
    layer.trainable = True
  else:
    layer.trainable = False
```

  - 네트워크 미세 조정
    - 학습률을 낮춘 RMSProp 옵티마이저 사용
    - 학습률을 낮추는 이유: 미세조정하는 3개의 층에서 학습된 표현을 조금씩 수정하기 위해
    - 변경량이 너무 크면 학습된 표현에 나쁜 영향을 미칠 수 있음

```
# 모델 미세 조정
mobile.compile(loss = 'binary_crossentropy' ,
               optimzier = optimizers.RMSprop(lr = 1e-5),
               metrics= ['acc'])
```

```
history = model.fit_generator(
        train_generator,
        steps_per_epoch = 100,
        epochs = 100,
        validation_data = validation_generator,
        valdiation_steps = 50)
```

```
  # 부드러운 그래프 그리기
def smooth_curve(points, factor = 0.8):
  smoothed_points = []
  for point in points:
    if smoothed_points:
      previous = smoothed_points[-1]
      smoothed_points.append(previous * factor + point* (1-factor)) # 전 point *
 0.8 + 현재 * 0.2
    else:
      smoothed_points.append(point)
  return smoothed_points

plt.plot(epochs, smooth_curve(acc), 'bo', label = 'Smoothed training acc')
plt.plot(epochs, smooth_curve(val_acc), 'b', label = 'Smoothed validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, smooth_curve(loss), 'bo', label = 'Smoothed training loss')
plt.plot(epochs, smooth_curve(val_loss), 'b', label = 'Smoothed validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
  # 마지막으로 테스트 데이터에서 이 모델을 평가

test_generator = test_datagen.flow_from_directory(
  test_dir,
  target_size = (150,150),
  batch_size= 20,
  class_mode= 'binary')

test_loss, test_acc = model.evluate_generator(test_generator, steps = 50)
print('test acc', test_acc)
```

5.3.3 정리

- `컨브넷:` 컴퓨터 비전 작업에서 가장 뛰어난 머신 러닝 모델
  - 아주 작은 데이터셋에서도 처음부터 훈련해서 괜찮은 성능을 낼 수 있음

- 작은 데이터셋에서 `과대 적합 문제` 발생
  - `데이터 증식`은 이미지 데이터를 다룰 때 과대적합을 막을 수 있는 강력한 방법
- `특성 추출 방식`으로 새로운 데이터셋에 기존 컨브넷을 쉽게 재사용할 수 있음
  - 작은 이미지 데이터셋으로 작업할 때 효과적
- 특성 추출을 보완하기 위해 `미세 조정`을 사용할 수 있음
  - 미세 조정은 기존 모델에서 사전에 학습한 표현의 일부를 새로운 문제에 적응시킴
  - 이 기법은 조금 더 성능을 끌어올림

5.4 컨브넷 학습 시각화

- 컨브넷: 시각적 개념을 학습한 것이기 때문에 시각화 가능
- 컨브넷 중간층의 출력(중간층에 있는 활성화)를 시각화하기
  - 연속된 컨브넷 층이 입력을 어떻게 변형시키는지 이해하고 개별적인 컨브넷 필터의 의미 파악
- 컨브넷 필터를 시각화하기
  - 컨브넷 필터가 찾으려는 시각적인 패턴과 개념이 무엇인지 상세하게 이해하도록 도움
- 클래스 활성화에 대한 히트맵(heatmap)을 이미지에 시각화하기
  - 이미지의 어느 부분이 주어진 클래스에 속하는 데 기여했는지 이해하고 이미지에 객체 위치를 추정(localization)하는데 도움

5.4.1 중간층의 활성화 시각화하기

- 어떤 입력이 주어졌을 때 네트워크에 있는 여러 합성곱과 풀링 층이 출력하는 특성 맵을 그리는 것
  - 층의 출력이 활성화 함수의 출력이라서 종종 활성화(activation)이라고 부름
  - 네트워크에 의해 학습된 필터들이 어떻게 입력을 분해하는지 보여 줌
  - 너비, 높이, 깊이(채널) 3가지 차원에 대해 특성 맵을 시각화하는 것이 좋음
  - 각 채널은 비교적 독립적인 특성을 인코딩
  - 특성 맵의 각 채널 내용을 독립적인 2D 이미지로 그리는 것이 괜찮은 방법

```
# 5.2 절에 저장했던 모델을 로드해 다시 시작
from keras.models import load_model
model = load_model('cats_and_dogs_small_2.h5')
model.summary() # 모델 구조 출력

# conv2d_5 - maxpooling2d_5
# - conv2d_6 - maxpooling2d_6
# - conv2d_7 - maxpooling2d_7
# - conv2d_8 - maxpooling2d_8
# - flatten - dropoout_1 - dense_3 - dense_4
```

```
# 개별 이미지 전처리하기
img_path ='./datasets/cats_and_dogs_small/test/cats/cat.1700.jpg'

from keras.preprocessing import image
import numpy as np

img = image.load_img(img_path, target_size = (150,150))
img_tensor = image.img_to_array(img)
img_tensor = np.expand_dims(img_tensor, axis = 0) # 이미지를 4D 텐서로 변경
img_tensor /= 255. # 모델이 훈련될 때 적용한 전처리 방식을 동일하게 사용함

print(img_tensor.shape) # 이미지 텐서의 크기 (1,150,150,3)
```

```
# 테스트 사진 출력
import matplotlib.pyplot as plt

plt.imshow(img_tensor[0])
plt.show()
```

```
# 입력 텐서와 출력 텐서의 리스트로 모델 객체 만들기
    # 확인하고 싶은 특성 맵 추출을 위해
    # 이미지 배치를 입력으로 받아 "모든 합성곱 & 풀링층 출력하는 케라스 모델"
    # 케라스 Model 클래스 사용
    # 모델 객체 - 2개의 매개변수 필요
      # 1. 입력 텐서(입력 텐서의 리스트)
      # 2. 출력 텐서(출력 텐서의 리스트)
      # 반환되는 객체는 Sequential과 같은 케라스 모델이지만
      # 특정 입력과 특정 출력을 매핑함
      # Model클래스 사용 - Sequential과 달리 여러 개의 출력을 가진 모델을 만들 수 있음

from keras import models

layer_outputs = [layer.output for layer in model.layers[:8] # 상위 8개 층의 출력을 추출
함
activation_model = models.Model(inputs = model.input, outputs =  layer_outputs)  #
입력에 대한 8개의 층의 출력을 반환하는 모델

# 입력 이미지가 주입될 때, 이 모델은 원본 모델의 활성화 값을 반환함
```

```
# 이 모델이 이 책에서는 처음 나오는 다중 출력 모델
# 지금까지 모델은 정확히 하나의 입력 & 하나의 출력만을 가짐
# 일반적으로 모델은 몇 개의 입력과 출력이라도 가질 수 있음
# 이 모델은 하나의 입력과 층의 활성화마다 하나씩 총 8개의 출력을 가짐
```

```
# 예측 모드로 모델 수행하기
activations= activation_model.predict(img_tensor) # 층의 활성화마다 하나씩 8개의 넘파이 배열로 이루어진 리스트를 반환
```

```
# ex. 고양이 이미지에 대한 첫 번째 합성곱 층의 활성화 값
first_layer_activation = activations[0]
print(first_layer_activation.shape) # (1,148,148,32)
```

```
# 32개의 채널을 가진 148 x 148 크기의 특성 맵
# 원본 모델의 첫 번째 층 활성화 중 20번째 채널
# 20번째 채널 시각화하기
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0, :,:,19], cmap = 'viridis')

# result
  # 이 채널은 대각선 에지를 감지하도록 인코딩 된 것 같으
  # 16번째 채널을 그림 (5-26)
  # 합성곱 층이 학습한 필터는 결정적이지 않기 때문에 채널 이미지가 책과 다를 수 있음
plt.matshow(first_layer_activation[0, :,:15], cmap = 'viridis')
```

```
# 네트워크의 모든 활성화를 시각화 (5-27)
# 8개의 활성화 맵에서 추출한 모든 채널을 그리기 위해 하나의 큰 이미지 텐서에 추출한 결과 나란히 놓음

# 중간층의 모든 활성화에 있는 채널 시각화하기
layer_names = [] # 층의 이름을 그래프 제목으로 사용
for layer in model.layers[:8]:
  layer_names.append(layer.name)

images_per_row = 16

for layer_name, layer_activation in zip(layer_names, activations): # 특성 맵을 그림
  n_features = layer_activation.shape[-1] # 특성 맵에 있는 특성의 수

  size = layer_activation.shape[1] # 특성 맵의 크기 (1, size, size, n_features)

  n_cols= n_features // images_per_row # 활성화 채널을 위한 그리드 크기를 구함
  display_grid = np.zeros((size * n_cols , image_per_row * size ))

  for col in range(n_cols): # 각 활성화를 하나의 큰 그리드에 채움
    for row in range(images_per_row):
      channel_image = layer_activation[0, :, :, col * images_per_row + row]
      channel_image -= channel_image.mean() # 그래프를 나타내기 좋게 특성을 처리
      channel_image /= channel_image.std()
```

```
        channel_image *= 64
        channel_image *= 128
        channel_image = np.clip(channel_image, 0, 255).astype('uint8')
        display_grid[col * size : (col+1) * size, # 그리드를 출력
                     row * size : (row+1) * size ] = channel_image

scale = 1./size
plt.figure(figsize = (scale * display_grid.shape[1],
                      scale * display_grid.shape[0]))

plt.title(layer_name)
plt.grid(False)
plt.imshow(display_grid, aspect = 'auto', cmap = 'viridis')

plt.show()
```

- 층의 깊이에 따라 점점 더 추상적으로 변함 (높은 층의 활성화 함수일수록)

5.4.2 컨브넷 필터 시각화하기

- 각 필터가 반응하는 시각적 패턴을 그러보는 것
- 빈 입력 이미지에서 시작해서 특정 필터의 응답을 최대화하기 위해 컨브넷 입력 이미지에 경사 상승법을 적용
  - `경사 상승법` : 손실 함수의 값이 커지는 방향으로 그래디언트를 업데이트하기 때문에 경사 하강법과 반대
  - 입력 이미지는 선택된 필터가 최대로 응답하는 이미지가 됨
  - 전체 과정
    - 특성 합성곱 층의 **한 필터 값을 최대화하는 손실 함수** 정의
    - 이 활성화 값을 최대화하기 위해 입력 이미지를 변경하도록 확률적 경사 상승법 사용
    - ex. ImageNet (VGG16) → block3_con1 층 필터 0번의 활성화를 손실로 정의

```
# 필터 시각화를 위한 손실 텐서 정의하기
from keras.applications import VGG16
from keras import backend as K

model = VGG16(weights= 'imagenet',
              include_top = False)

layer_name = 'block3_conv1'
filter_index = 0

layer_output = model.get_layer(layer_name).output
```

```
loss = K.mean(layer_output[:, :, :, filter_index])

# 경사 상승법을 구현하기 위해 모델의 입력에 대한 손실의 그래디언트가 필요
# 이를 위해 케라스 backend 모듈의 gradients 함수를 사용
```

```
# 입력에 대한 손실의 그래디언트 구하기
grads= K.gradients(loss, model.input)[0] # gradients 함수가 반환하는 텐서 리스트(여기서는
 크기가 1인 리스트)에서 첫 번째 텐서를 추출
```

```
# 경사 상승법 과정을 부드럽게 하기 위해
  # 그래디언트 텐서를 L2노름(텐서에 있는 값을 제곱한 합의 제곱근)으로 나누어 정규화
  # 입력 이미지에 적용할 수정량의 크기를 항상 일정 범위 안에 놓을 수 있음

# 그래디언트 정규화하기
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5) # 0 나눗셈을 방지하기 위해 1e-5 를 더
함
```

```
# 입력 이미지에 대한 손실 텐서와 그래디언트 텐서를 계산
# 케라스 백엔드 함수를 사용하여 처리
# iterate 넘파이 텐서(크기가 1인 텐서의 리스트)를 입력으로 받아 손실과 그래디언트 2개의 넘파이 텐서
를 반환

# 입력값에 대한 넘파이 출력 값 출력
iterate = K.function([model.input], [loss, grads])

import numpy as np
loss_value, grads_value = iterate([np.zeros(1,150,150,3)])
```

```
# 파이썬 루프를 만들어 확률적 경사 상승법을 구성
# 확률적 경사 상승법을 사용한 손실 최대화하기
input_img_data = np.random.random((1,150,150,3)) * 30 + 128. # 잡음이 섞인 회색 이미지
로 시작

step = 1. # 업데이트할 그래디언트의 크기
for i in range(40): # 경사 상승법 40회 실행
  lsos_value, grads_value = iterate([input_img_data]) # 손실과 그래디엍느 계산

  input_img_data += grads_value * step # 손실을 최대화하는 방향으로 입력 이미지를 수정
```

```
# 결과 이미지 텐서: (1,150,150,3) 크기의 부동 소수 텐서
  # -> [0,255] 사이의 정수 X
  # -> 출력 가능한 이미지로 변경하기 위해 후처리 필요

def deprocess_image(x):
    # 텐서의 평균이 0, 표준 편차가 0.1이 되도록 정규화합니다
    x -= x.mean()
```

```
    x /= (x.std() + 1e-5)
    x *= 0.1

    # [0, 1]로 클리핑합니다
    x += 0.5
    x = np.clip(x, 0, 1)

    # RGB 배열로 변환합니다
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

```
# 필터 번호를 입력으로 받는 함수를 만듦
  # 함수는 필터 활성화를 최대화하는 패턴을 이미지 텐서로 출력
  # 필터 시각화 이미지를 만드는 함수

def generate_pattern(layer_name, filter_index, size = 150):
  layer_output = model.get_layer(layer_name).output def generate_pattern(layer_nam
e, filter_index, size=150):
    # 주어진 층과 필터의 활성화를 최대화하기 위한 손실 함수를 정의합니다
    layer_output = model.get_layer(layer_name).output
    loss = K.mean(layer_output[:, :, :, filter_index])

    # 손실에 대한 입력 이미지의 그래디언트를 계산합니다
    grads = K.gradients(loss, model.input)[0]

    # 그래디언트 정규화
    grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)

    # 입력 이미지에 대한 손실과 그래디언트를 반환합니다
    iterate = K.function([model.input], [loss, grads])

    # 잡음이 섞인 회색 이미지로 시작합니다
    input_img_data = np.random.random((1, size, size, 3)) * 20 + 128.

    # 경사 상승법을 40 단계 실행합니다
    step = 1.
    for i in range(40):
        loss_value, grads_value = iterate([input_img_data])
        input_img_data += grads_value * step

    img = input_img_data[0]
    return deprocess_image(img)
```

```
# 코드 실행
plt.imshow(generate_pattern('block3_conv1', 0))
plt.show()

# block3_conv1 층의 필터 0은 물방울 패턴에 반응하는 것 같습니다.
```

```
# 모든 층에 있는 필터를 시각화
  # 간단하게 만들기 위해 각 층에서 처음 64개의 필터만 사용하겠습니다
  # 각 합성곱 블럭의 첫 번째 층만 살펴보겠습니다(block1_conv1, block2_conv1, block3_conv1,
 block4_conv1, block5_conv1).
  # 여기서 얻은 출력을 64 × 64 필터 패턴의 8 × 8 그리드로 정렬합니다.
  # 각 필터 패턴 사이에 검은 색 마진을 약간 둡니다.

for layer_name in ['block1_conv1', 'block2_conv1', 'block3_conv1', 'block4_conv
1']:
    size = 64
    margin = 5

    # 결과를 담을 빈 (검은) 이미지
    results = np.zeros((8 * size + 7 * margin, 8 * size + 7 * margin, 3), dtype='u
int8')

    for i in range(8):  # results 그리드의 행을 반복합니다
        for j in range(8):  # results 그리드의 열을 반복합니다
            # layer_name에 있는 i + (j * 8)번째 필터에 대한 패턴 생성합니다
            filter_img = generate_pattern(layer_name, i + (j * 8), size=size)

            # results 그리드의 (i, j) 번째 위치에 저장합니다
            horizontal_start = i * size + i * margin
            horizontal_end = horizontal_start + size
            vertical_start = j * size + j * margin
            vertical_end = vertical_start + size
            results[horizontal_start: horizontal_end, vertical_start: vertical_en
d, :] = filter_img

    # results 그리드를 그립니다
    plt.figure(figsize=(20, 20))
    plt.imshow(results)
    plt.show()
```

5.4.3 클래스 활성화의 히트맵 시각화하기

- 이미지에 어느 부분이 컨브넷의 최종 분류 결정에 기여하는지 이해하는 데 유용
- 컨브넷의 결정 과정을 디버깅하는 데 도움이 됨
  - 이미지에 특정 물체가 있는 위치를 파악하는데 사용함
  - 일반적으로 클래스 활성화 맵(CAM) 시각화라고 부름
  - 입력 이미지에 대한 클래스 활성화의 히트맵을 만듦
    - 클래스 활성화 히트맵: 특정 출력 클래스에 대해 입력 이미지의 모든 위치에 대해 계산된 2D 점수 그리드
    - 클래스에 대해 각 위치가 얼마나 중요한지 알려줌
  - Grad-CAM: 입력 이미지가 주어지면 합성곱 층에 있는 특성 맵의 출력을 추출

- 특성 맵의 모든 채널의 출력에 채널에 대한 클래스의 그래디언트 평균을 곱함
- '입력 이미지가 각 채널을 활성화하는 정도'에 대한 공간적인 맵을
- '클래스에 대한 각 채널의 중요도'로 가중치를 부여하여
- '입력 이미지가 클래스를 활성화하는 정도'에 대한 공간적인 맵을 만드는 것

```
# 사전 훈련된 가중치로 VGG16 네트워크 로드하기
from keras.applications.vgg16 import VGG16

K.clear_session()

# 이전 모든 예제에서는 최상단의 완전 연결 분류기 제외, 여기서는 포함
model = VGG16(weights = 'imagenet')
```

```
# VGG16을 위해 입력 이미지 전처리
# VGG16 모델이 인식할 수 있도록 변환
  # 224 x 224 크기의 이미지에서 훈련
  # keras.applications.vgg16.preprocess_input 함수
    # 1. 이미지를 로드
    # 2. 224 x 224 크기로 변경
    # 3. 넘파이 float32 텐서로 바꿈
    # 4. 전처리 함수를 적용

# VGG 모델은 카페(Caffe) 딥러닝 라이브러리에서 훈련되어 정규화 방식이 다름
  # 입력 데이터의 이미지 채널 RGB -> BGR로 바꾸고
  # ImageNet 데이터셋에서 구한 채널별 평균값 [103.939, 116.779, 123.68]을 뺌

from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np

# 이미지 경로
img_path = './datasets/creative_commons_elephant.jpg'

# 224 × 224 크기의 파이썬 이미징 라이브러리(PIL) 객체로 반환됩니다
img = image.load_img(img_path, target_size=(224, 224))

# (224, 224, 3) 크기의 넘파이 float32 배열
x = image.img_to_array(img)

# 차원을 추가하여 (1, 224, 224, 3) 크기의 배치로 배열을 변환합니다
x = np.expand_dims(x, axis=0)

# 데이터를 전처리합니다(채널별 컬러 정규화를 수행합니다)
x = preprocess_input(x)
```

```
# 이미지에서 사전 훈련된 네트워크를 실행 & 예측 벡터를 디코딩
preds= model.predict(x)
print('Predicted', decode_predictions(preds, top = 3)[0])
```

```
# decode_predictions() 함수는
# ImageNet 데이터셋에 대한 예측 결과에서
# top 매개변수에 지정된 수만큼 최상위 항목을 반환해 줌
```

- np.argmax(preds[0]) # 386

```
# Grad-CAM 알고리즘 설정하기

# 예측 벡터의 '아프리카 코끼리' 항목
african_elephant_output = model.output[:, 386]

# VGG16의 마지막 합성곱 층인 block5_conv3 층의 특성 맵
last_conv_layer = model.get_layer('block5_conv3')

# block5_conv3의 특성 맵 출력에 대한 '아프리카 코끼리' 클래스의 그래디언트
grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]

# 특성 맵 채널별 그래디언트 평균 값이 담긴 (512,) 크기의 벡터
pooled_grads = K.mean(grads, axis=(0, 1, 2))

# 샘플 이미지가 주어졌을 때 방금 전 정의한 pooled_grads와 block5_conv3의 특성 맵 출력을 구합니다
iterate = K.function([model.input], [pooled_grads, last_conv_layer.output[0]])

# 두 마리 코끼리가 있는 샘플 이미지를 주입하고 두 개의 넘파이 배열을 얻습니다
pooled_grads_value, conv_layer_output_value = iterate([x])

# "아프리카 코끼리" 클래스에 대한 "채널의 중요도"를 특성 맵 배열의 채널에 곱합니다
for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]

# 만들어진 특성 맵에서 채널 축을 따라 평균한 값이 클래스 활성화의 히트맵입니다
heatmap = np.mean(conv_layer_output_value, axis=-1)
```

```
# 히트맵 후처리하기
# 시각화를 위해 히트맵을 0~1 사이로 정규화
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
plt.show()
```

```
# 원본 이미지에 히트맵 덧붙이기
# OpenCV를 사용해 앞에서 얻은 히트맵에 원본 이미지를 겹친 이미지
import cv2

# cv2 모듈을 사용해 원본 이미지를 로드합니다
img = cv2.imread(img_path)

# heatmap을 원본 이미지 크기에 맞게 변경합니다
heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
```

```
# heatmap을 RGB 포맷으로 변환합니다
heatmap = np.uint8(255 * heatmap)

# 히트맵으로 변환합니다
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

# 0.4는 히트맵의 강도입니다
superimposed_img = heatmap * 0.4 + img

# 디스크에 이미지를 저장합니다
cv2.imwrite('./datasets/elephant_cam.jpg', superimposed_img)
```

5.5 요약

- 컨브넷은 시각적인 분류 문제를 다루는 데 최상의 도구
- 컨브넷 = 패턴의 계층 구조와 개념을 학습
- 학습된 표현은 쉽게 분석할 수 있음 (conv - not black box)
- 이미지 분류 문제를 풀기 위해 자신만의 컨브넷 처음부터 훈련 가능
- 과대적합을 줄이기 위해 데이터 증식하는 방법도 있음
- 사전 훈련된 컨브넷을 사용하여 특성 추출 & 미세 조정하는 법
- 클래스 활성화 히트맵을 포함하여 컨브넷이 학습한 필터를 시각화