4, 5장 발표

이상윤, 김태우, 송선영

Start

목차

4 장

4.1 머신 러닝의 네 가지 분류

4.2 머신 러닝 모델 평가

4.3 데이터 전처리, 특성 공학, 특성 학습

4.4 과대적합과 과소적합

4.5 보편적인 머신 러닝 작업 흐름

5 장

5.1 합성곱 신경망 소개

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련하기

5.3 사전 훈련된 컨브넷 사용하기

5.4 컨브넷 학습 시각화

5.5 요약

4.1 머신 러닝의 네가지 분류

1. 지도학습
1. 비지도 학습
1. 자기 지도 학습
1. 강화 학습

4.2 머신 러닝 모델 평가

'단순 홀드아웃 검증'을 활용한 평가

단순히 데이터 세트 중 일부를 테스트 데이터와 검증세트를 떼어놓고 남은 데이터로 훈련하는 방법

단점: 데이터가 편향되어 있을 때 모델의 성능이 크게 변할 수 있음.
(랜덤으로 데이터를 셔플하여 나눌때마다 모델의 성능이 크게 달라지는 경우)

4.2 머신 러닝 모델 평가 ‘K겹 교차 검증’을 활용한 평가 방법

모델을 K개 만들어 검증 점수를 평균하는 방법, 각 모델에 사용하는 데이터는 동일한 크기를 가진 K개의 분할로 나눈 뒤, K -1 개의 분할 데이터로 모델을 훈련하고, 남은 하나의 분할 데이터로 평가한 뒤 각 모델의 검증 점수를 평균하는 방법

특징 :
데이터의 분할에 따라 편차가 클 때 홀드 아웃 검증보다 성능이 좋게 나옴

응용: 셔플링을 활용한 K-겹 교차 검증

K-겹 교차 검증을 동일하게 하되 , 데이터를 K개의 분할로 나누기 전에 매번 데이터를 무작위로 섞는 방법

4.3 데이터 전처리

데이터 전처리
원본 데이터를 신경망에 적용하기 쉽도록 변환하는 것
모델이 수월하게 작업할 수 있는 방식으로 데이터를 표현하는 방법

종류

1. 벡터화

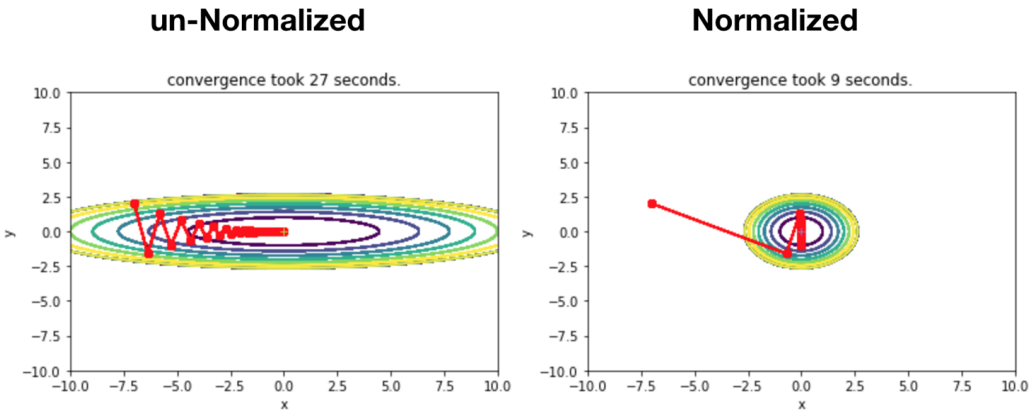
1. 값 정규화

1. 누락된 값 다루기

4.3 데이터 전처리

값 정규화

데이터 이름	데이터 범위
데이터 A	0~ 1
데이터 B	0~ 255
데이터 C	3~ 130



따르면 좋은 규칙

제각각의 범위를 갖는 데이터를 신경망에 넣게되면, 그래디언트의 값이 커지기 때문에, 신경망이 좋은 값을 얻기 어려워집니다.

- 각 특성별로 평균이 0이 되도록 정규화.
- 각 특성별로 표준 편차가 1이 되도록 정규화.

4.3 데이터 전처리

누락된 값 다루기

누락 값으로 0을 입력하기
(0은 정의되지 않은 값)

응용

데이터에 누락된 값이 포함될 가능성이 있다면, 누락된 값이 있는 훈련샘플을 고의적으로 만드는 방법.

예)
훈련 샘플에서 빠질 것 같은 특성에 0을 추가

	0	1	2	3	4
0	94.0800	200.00000	0.035123	0.010869	0.072132
1	5.0580	0.36506	0.009759	100.000000	0.064328
2	NaN	3.96100	0.048617	0.036862	0.080865
3	1955.1000	4.22810	0.156830	0.065750	NaN
4	4.0507	2.04220	0.039377	0.082756	0.032600

4.3 데이터 전처리 - 특성공학

특성공학

알고리즘을 사용하여 데이터를 더 좋은 특성으로 만드는 것

예) 시계 이미지를 입력 받아 시간을 출력하는 모델
-> 초침 분침의 끝 점의 좌표로 표현
-> 초침 분침을 각도로 표현

Raw data: pixel grid		
Better features: clock hands' coordinates	{x1: 0.7, y1: 0.7} {x2: 0.5, y2: 0.0}	{x1: 0.0, y2: 1.0} {x2: -0.38, 2: 0.32}
Even better features: angles of clock hands	theta1: 45 theta2: 0	theta1: 90 theta2: 140

4.4 과대적합과 과소적합

3장에 나온 토픽 분류, 영화 리뷰 예측 , 등 모두 홀드아웃 데이터 모델의 성능이 몇 번의 에포크 후 최고치에 이르렀다가 감소 -> 과대적합 발생

훈련 초기에 훈련 데이터의 손실이 낮아질수록 테스트 데이터의 손실이 낮아지는 상황, 모델의 성능이 계속 발전될 여지가 존재 -> 과소적합 발생

과대적합을 해결하는 가장 좋은 방법 = 더 많은 훈련 데이터를 수집

하지만, 데이터의 수가 정해져 있으며, 모으기 불가능한 상황이 다수.

-> '규제'를 통해 과대적합을 조절

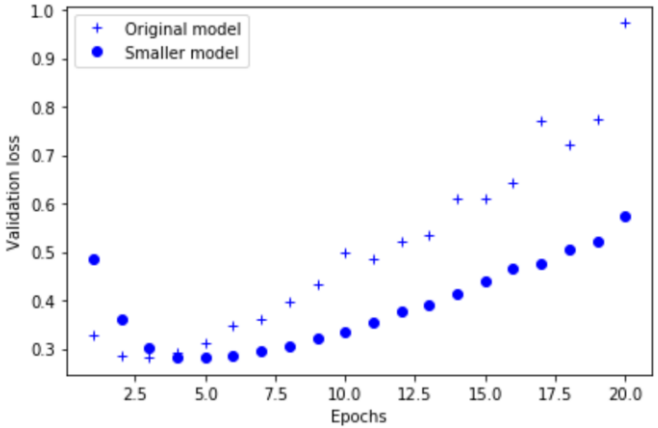
4.4 과대적합과 과소적합 과대적합을 피하는 처리과정

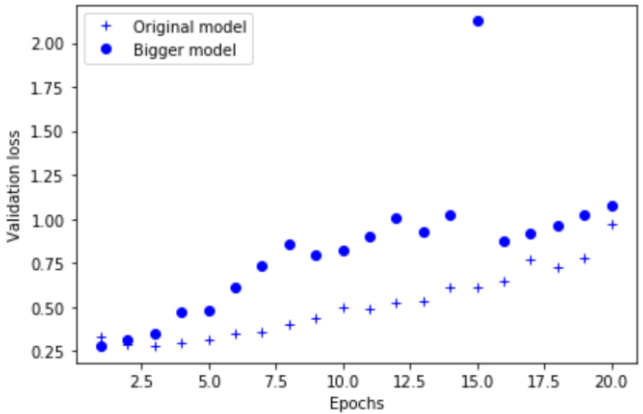
규제의 종류

1. 네트워크 크기 축소

    학습 파라미터의 수를 줄이는 것

적절한 수의 파라미터를 제공하는 것이 일반화 성능을 최고로 유지
하지만 , 층의 수나 층의 유닛 수를 결정하는 방법이 존재하지 않음 -> 검증 손실이 감소되기 시작할 때까지 층과 유닛의 수를 늘리는 방법.





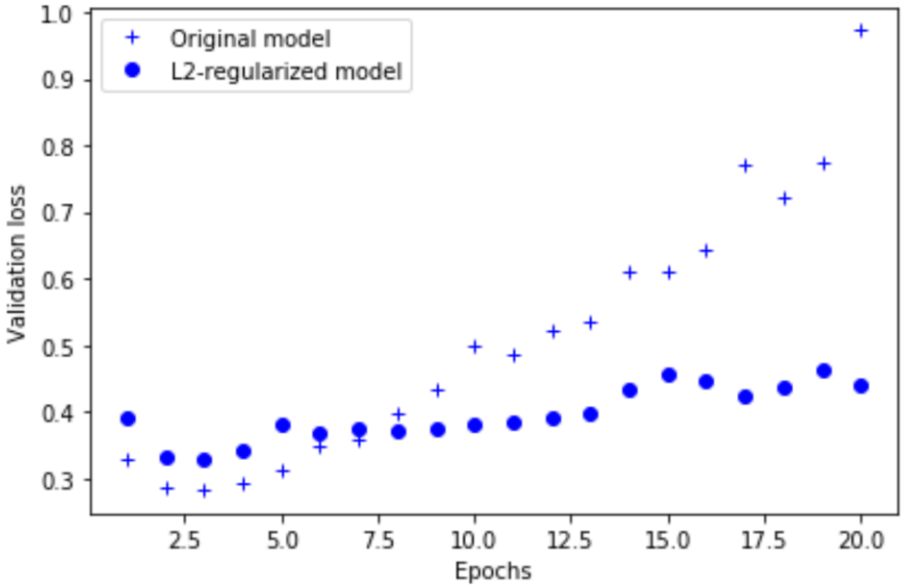
4.4 과대적합과 과소적합 과대적합을 피하는 처리과정

2. 가중치 규제 추가

네트워크 가중치가 더 작은 값을 가지도록 강제하는 것

L1 규제 : 가중치의 절댓값에 비례하는 비용이 추가

L2 규제 : 가중치의 제곱에 비례하는 비용이 추가
(가중치 감쇠)

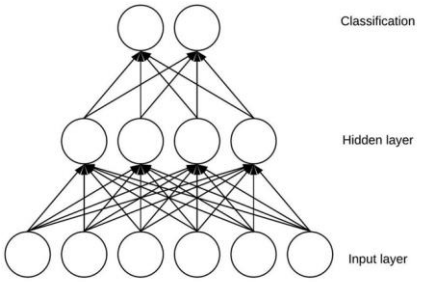


4.4 과대적합과 과소적합 과대적합을 피하는 처리과정

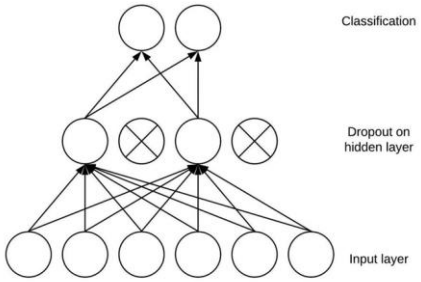
3. 드롭아웃 추가

무작위로 층의 일부 출력을 제외시키는 방법 (0으로 변환)

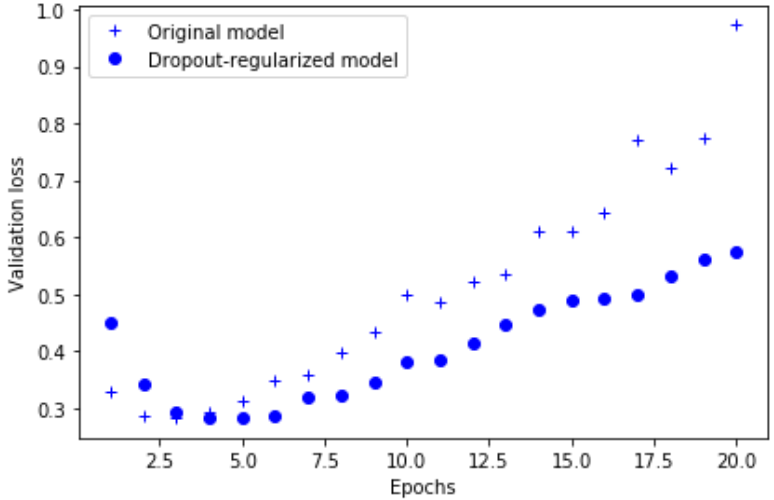
예) 출력 벡터 [0.2 ,0.5 ,1.3 ,0.8 ,1.1] -> [0 ,0.5 ,1.3 ,0 ,1.1]



Without Dropout



With Dropout



4.5 보편적인 머신 러닝 작업 흐름

1. 문제 정의와 데이터셋 수집
1. 성공 지표 선택
1. 평가 방법 선택
1. 데이터 준비
1. 기본보다 나은 모델 준비하기
1. 과대적합 모델 구축
1. 모델 규제, 하이퍼파라미터 튜닝

5.1 합성곱 신경망 소개

```
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'))

# 3D 텐서를 1D 텐서로 펼침
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu')) # Dense 층 추가
model.add(layers.Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

* **기본적인 컨브넷**
  - 입력: (image_height, image_width, image_channels) 크기의 3D 텐서
  - 출력: (height, width, channels) 크기의 3D 텐서
  - height와 width 차원은 네트워크가 깊어질수록 작아짐
  - channel의 수는 전달된 첫번째 매개변수에 의해 조절

5.1.1 합성곱 연산

완전 연결 층: **전역** 패턴을 학습 (전역패턴 예: MNIST 숫자 이미지에서 모든 픽셀에 걸친 패턴)
합성곱 층: **지역**패턴을 학습 (지역패턴 예: 에지(edge), 질감(texture))

1) 컨브넷의 성질
    1. 학습된 패턴은 평행 이동 불변성을 가짐
        - 오른쪽에서 학습했다면 왼쪽에서도 인식 가능해야함
        - 적은 수의 훈련 샘플을 사용해서 일반화 능력을 가진 표현 학습 가능
        - 이미지를 효율적으로 처리할 수 있음

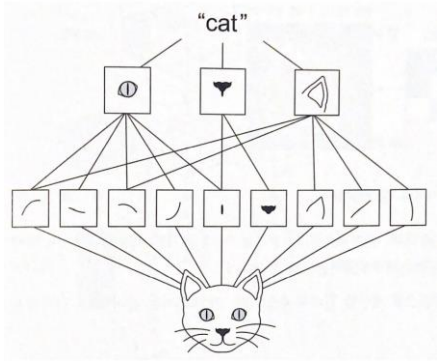
    1. 컨브넷은 패턴의 공간적 계층 구조를 학습할 수 있음
        - 작은 지역 패턴에서 더 큰 지역 패턴으로 학습 가능
        - 복잡하고 추상적인 시각적 개념을 효과적으로 학습할 수 있음

5.1.1 합성곱 연산

2) 합성곱 연산
- **특성 맵**(feature map)이라고 부르는 3D 텐서에 적용
- 입력 특성맵에서 작은 패치들을 추출해 **출력 특성 맵**(output feature map)을 만듦

- 특성 맵과 출력특성 맵 모두 2개의 공간 축(높이와 넓이)과 깊이 축(채널)을 가진 3D 텐서
- 출력특성 맵에서의 깊이 축은 **필터**를 의미
  ex) (28, 28, 1) 크기의 특성 맵 입력 -> (26, 26, 32) 크기의 특성 맵 출력
  -> 32개의 출력 채널 각각은 26x26 크기의 배열 값을 가짐
  (필터의 **응답 맵**(response map) 이라고 함)

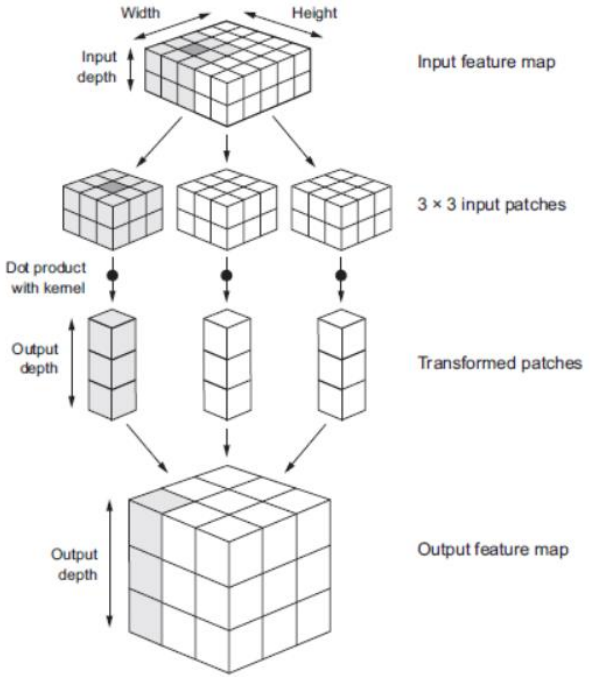
- 필터는 입력데이터의 어떤 특성을 인코딩함 (ex. ' 입력에 얼굴이 있는지 ')



3) 합성곱의 파라미터
- **입력으로부터 뽑아낼 패치의 크기**: 전형적으로 3x3, 5x5 사용
- **특성 맵의 출력 깊이**: 합성곱으로 계산할 필터의 수

5.1.1 합성곱 연산

4) 합성곱 작동 방식

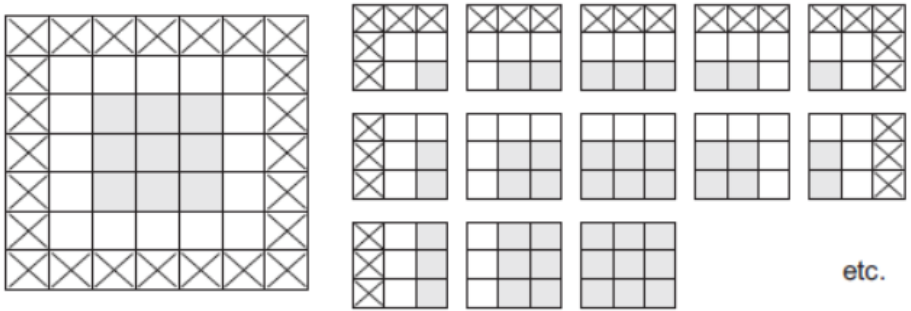


- 3D 입력 특성 맵 위를 3x3 또는 5x5 크기의 윈도우가 슬라이딩하면서 모든 위치에서 3D 특성 패치 ((window_height, window_width, input_depth) 크기)를 추출하는 방식으로 합성곱이 작동함

- 합성곱의 출력 높이와 입력의 높이, 넓이는 다를 수 있음
  - 경계 문제. 입력 특성 맵에 **패딩**을 추가하여 대응할 수 있음
  - **스트라이드**(stride)의 사용 여부에 따라 다름

5.1.1 합성곱 연산

5) 경계 문제와 패딩 이해하기

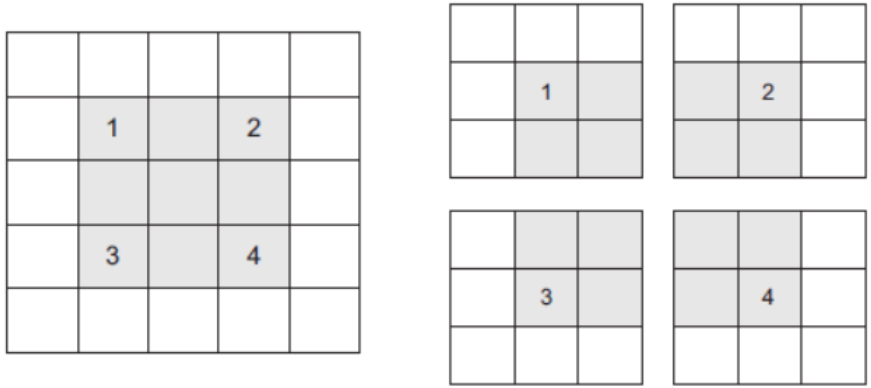


패딩(padding)을 사용하여 입력층과 출력층을 같게 할 수 있음
- 입력 특성 맵의 가장자리에 적절한 개수의 행과 열을 추가하여 모든 입력 타일에 합성곱 윈도우의 중앙을 위치시킬 수 있음

- 패딩 매개변수
  - “valid”: 패딩을 사용하지 않는다는 뜻, 디폴트값
  - “same”: 입력과 동일한 높이와 넓이를 가진 출력을 만들기 위해 패딩한다는 뜻

5.1.1 합성곱 연산

6) 합성곱 스트라이드 이해하기



↑ 2x2 스트라이드를 사용한 3x3 합성곱의 패치

스트라이드

- 두번의 연속적인 윈도우 사이의 거리: 스트라이드라고 불리는 합성곱 파라미터
- 스트라이드 기본값은 1임. 1보다 큰 스트라이드 합성곱도 가능함
- 스트라이드 2를 사용했다는 것은 특성 맵의 넓이와 높이가 2의 배수로 다운샘플링 된 것

5.1.2 최대 풀링 연산

- 스트라이드 합성곱과 매우 비슷하게 강제적으로 특성 맵을 다운샘플링
- 입력 특성 맵에서 윈도우에 맞는 패치를 추출하고 각 채널별로 최댓값을 출력함. 합성곱 개념과 비슷하지만 추출한 패치에 학습된 선형 변환(합성곱 커널)을 적용하는 대신 하드코딩된 최댓값 추출 연산 사용

- 합성곱과 가장 큰 차이점?
  - 합성곱: 3x3 윈도우와 스트라이드 1을 사용
  - 최대 풀링: 보통 2x2 윈도우와 스트라이드 2를 사용하여 특정 맵을 절반 크기로 다운샘플링

- 다운샘플링 하는 이유: 처리할 특성 맵의 가중치 개수를 줄이기 위해

- 최댓값을 취하는 최대 풀링 외에 입력 패치의 채널별 평균값을 계산하여 변환하는 **평균 풀링**(average pooling)으로도 다운샘플링이 가능하지만 최대 풀링이 더 잘 작동함
  - 특성이 특성 맵의 각 타일에서 어떤 패턴이나 개념의 존재 여부를 인코딩하는 경향이 있기 때문

5.1.2 최대 풀링 연산

```
# 합성곱으로만 이루어진 모델
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'))
```

```
model_no_max_pool.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 22, 22, 64)        36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

→ 합성곱만을 이용한 모델의 2가지의 문제

1. 컨브넷에 의해 학습된 고수준 패턴은 초기 입력에 관한 정보가 아주 적어 숫자 분류를 학습하기에 불충분

1. 최종 특성 맵은 22 x 22 x 64 = 30,976 개의 가중치를 가짐. 이 컨브넷을 펼친 후
   512 크기의 Dense 층과 연결한다면 15800개의 가중치 파라미터가 생김.
   -> 작은 모델치고는 너무 많은 가중치, 심각한 과대적합 발생

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

1) 작은 데이터셋 문제에서 딥러닝의 타당성

- 많은 샘플이라는 것은 훈련하려는 네트워크의 크기, 깊이에 상대적
- 딥러닝 모델은 다목적
  -> 대규모 데이터셋에서 훈련시킨 모델을 조금만 변경해서 다른 문제에 재사용 가능
- 작은 데이터셋에 딥러닝을 적용하기 위한 기술
  -> 사전 훈련된 네트워크로 특성을 추출,
  사전 훈련된 네트워크를 세밀하게 튜닝

2) 데이터 내려받기

```
import os, shutil
```

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

# 소규모 데이터셋을 저장할 디렉터리
base_dir = './datasets/cats_and_dogs_small'
if os.path.exists(base_dir):  # 반복적인 실행을 위해 디렉토리를 삭제합니다.
    shutil.rmtree(base_dir)   # 이 코드는 책에 포함되어 있지 않습니다.
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_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)

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

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

```
훈련용 고양이 이미지 전체 개수: 1000
```

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

3) 네트워크 구성하기

```
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'))
```

```
from keras import optimizers

# 네트워크의 마지막이 sigmoid 이기 때문에 binary crossentropy를 손실로 사용
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

4) 데이터 전처리

- 네트워크에 주입되기 전에 부동 소수 타입의 텐서로 전처리되어 있어야 함
- ImageDataGenerator 클래스: 디스크에 있는 이미지 파일을 전처리된 배치 텐서로 자동으로 바꾸어주는 파이썬 generator를 만들어 줌.

* 파이썬 제너레이터: 반복자(iterator)처럼 작동하는 개체

```
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(
        # 타깃 디렉터리
        train_dir,
        # 모든 이미지를 150 × 150 크기로 바꿉니다
        target_size=(150, 150),
        batch_size=20,
        # binary_crossentropy 손실을 사용하기 때문에 이진 레이블이 필요합니다
        class_mode='binary')

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

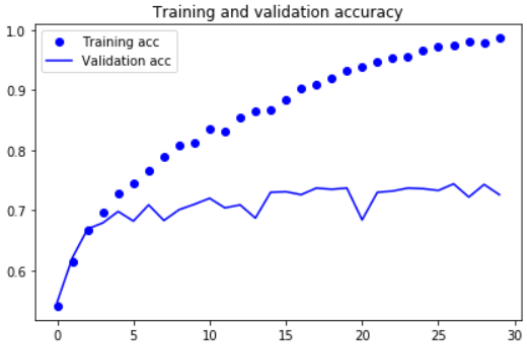
```
for data_batch, labels_batch in train_generator:
    print('배치 데이터 크기:', data_batch.shape)
    print('배치 레이블 크기:', labels_batch.shape)
    break
```

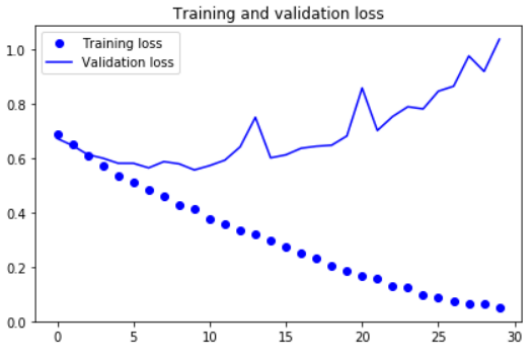
```
배치 데이터 크기: (20, 150, 150, 3)
배치 레이블 크기: (20,)
```

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

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

```
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()
```





→ **과대적합**

- 훈련 정확도가 선형적으로 증가하여 거의 100%에 도달
- 반면에 검증 정확도는 70~72%에서 멈춤
- 검증 손실은 다섯 번의 에포크만에 최솟값에 다다른 이후 더이상 진전되지 않음
- 반면에 훈련 손실은 거의 0에 도달할때까지 선형적으로 계속 감소

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

5) 데이터 증식 사용하기

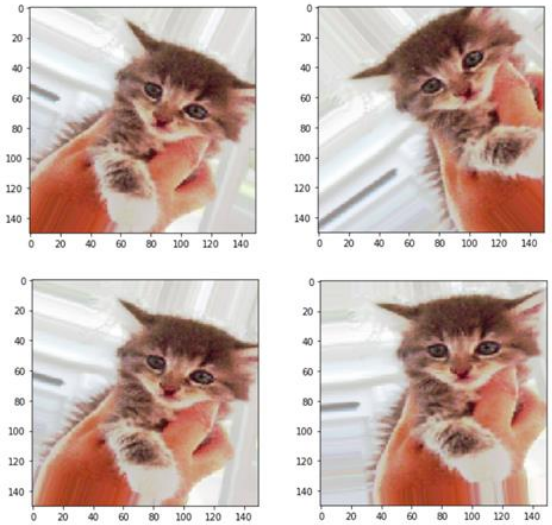
- 데이터 증식을 이용하여 과대적합을 줄일 수 있음, 일반화에 도움됨
- 데이터 증식: 기존 훈련 샘플로부터 더 많은 훈련 데이터를 생성하는 방법
  - 그럴듯한 이미지를 생성하도록 여러 가지 랜덤한 변환을 적용하여 샘플을 늘려 훈련할 때 모델이 정확히 같은 데이터를 두 번 만나지 않도록 하는 것이 목표

```
# imageDataGenerator를 사용하여 데이터 증식 설정하기
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_range, height_shift_range: 사진을 수평과 수직으로 랜덤하게 평행 이동시킬 범위 (전체 넓이와 높이에 대한 비율)
- shear_range: 랜덤하게 전단 변환을 적용할 각도 범위
- zoom_range: 랜덤하게 사진을 확대할 범위
- horizontal_flip: 랜덤하게 이미지를 수평으로 뒤집음. 수평 대칭을 가정할 수 있을때 사용 ex) 풍경/인물 사진
- fill_mode: 회전이나 가로/세로 이동으로 인해 새롭게 생성해야 할 픽셀을 채울 전략

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

```
# 랜덤하게 증식된 훈련 이미지 그리기

# 이미지 전처리 유틸리티 모듈
from keras.preprocessing import image

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

# 증식할 이미지 선택합니다
img_path = fnames[3]

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

# (150, 150, 3) 크기의 넘파이 배열로 변환합니다
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
for batch in datagen.flow(x, batch_size=1):
    plt.figure(i)
    imgplot = plt.imshow(image.array_to_img(batch[0]))
    i += 1
    if i % 4 == 0:
        break

plt.show()
```



새로운 정보가 아닌 기존 정보를 재조합한 데이터이기 때문에 과대적합을 제거하기에 충분하지 않을 수 있음
-> 완전 연결 분류기 직전에 Dropout 층 추가

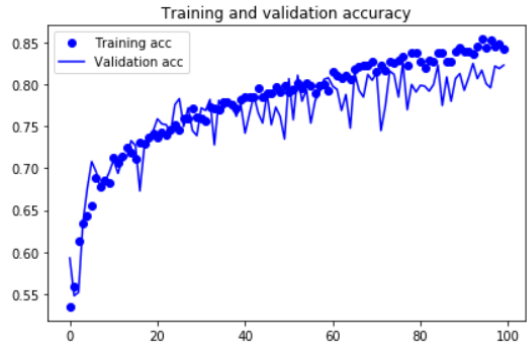
5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

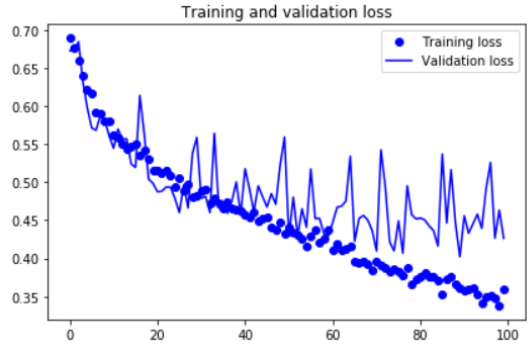
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (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)) # Dropout 층 추가
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

```
# 데이터 증식 제너레이터를 사용하여 컨브넷 훈련하기
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 안 됩니다!
test_datagen = ImageDataGenerator(rescale=1./255)

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

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_data=validation_generator,
      validation_steps=50)
```

5.2 소규모 데이터셋에서 밑바닥부터 컨브넷 훈련

```
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()
```





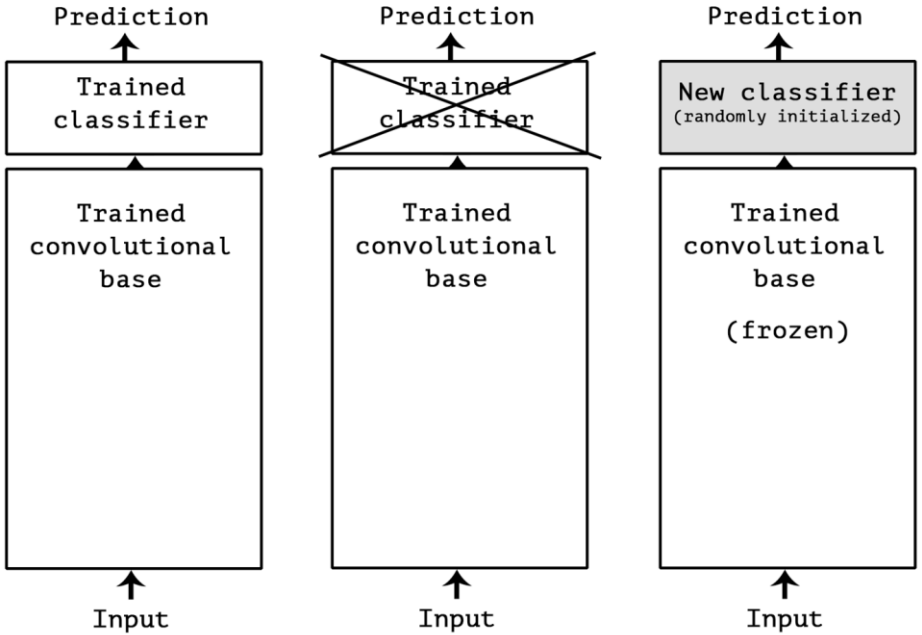
→ **과대적합 x**

- 훈련 곡선이 검증 곡선에 가깝게 따라가고 있음
- 검증 데이터에서 82% 정확도를 달성
- 규제하지 않은 모델과 비교했을 때 15% 정도 향상
- 데이터가 적기 때문에 처음부터 훈련해서 더 높은 정확도를 달성하기는 어려움 -> 사전 훈련된 모델 사용

5.3 사전 훈련된 컨브넷 사용하기

- ImageNet 데이터 셋을 통해 훈련된 네트워크를 활용하는 방법
- ImageNet : 1,400만 개의 레이블된 이미지와 1,000개의 클래스로 이루어진 데이터 셋
- 강아지와 고양이 분류하기
- 방법 :
  - 특성 추출
  - 미세 조정

5.3 사전 훈련된 컨브넷 사용하기 - 특성추출



- 특성 추출 : 사전에 학습된 네트워크의 표현을 사용해 새로운 샘플에서 흥미로운 특성을 뽑아내는 것
- 합성곱 기반 층 :
  - 일반적이다.
  - 객체의 위치 정보를 고려한다.
- 완전 연결 분류기 :
  - 모델이 훈련된 클래스 집합에 특화됨.
  - 위치 정보를 고려하지 않음.
- 일반성?
  - 모델에 있는 층의 깊이에 따라..
  - 하위층 : 색, 질감 등 지역적이고 일반적인 특성 맵
  - 상위 층 : '강아지 눈' 과 같은 추상적인 개념을 추출

5.3 사전 훈련된 컨브넷 사용하기 - 특성추출

```
from keras.applications import VGG16

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

이 후에는?

1. conv_base (합성곱 기반 층) 의 출력을 한 번만 실행하고 독립된 완전 연결 분류기에 입력으로 사용

1. conv_base 위에 Dense 층을 쌓아 확장

- 합성 곱 연산은 비용이 많이 들기에, 1번은 빠르지만 데이터 증식을 사용할 수 없고,
- 2번은 느리지만 데이터 증식을 사용할 수 있다.

```
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

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

5.3 사전 훈련된 컨브넷 사용하기 - 특성추출

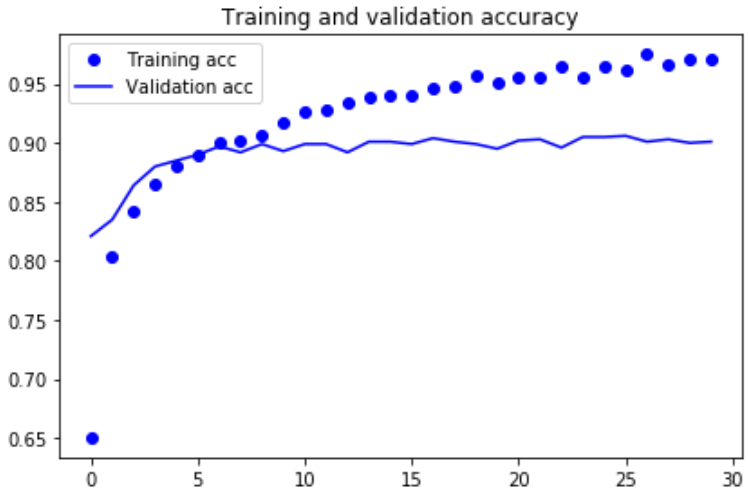
```
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator

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)
        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

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)
```

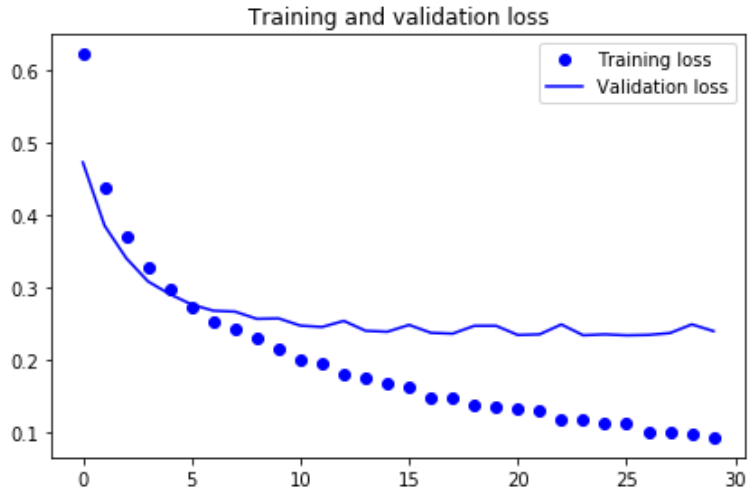
```
train_features = np.reshape(train_features, (2000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1000, 4 * 4 * 512))
test_features = np.reshape(test_features, (1000, 4 * 4 * 512))
```

```
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))
```

합성 곱 연산은 한 번만 수행,
이 후에는 2개의 Dense 층만 처리

5.3 사전 훈련된 컨브넷 사용하기 - 특성추출





- 결과 :
  90% 가량의 검증 정확도, 그러나 과적합이 눈에 띔.

5.3 사전 훈련된 컨브넷 사용하기 - 특성추출

```
from keras.applications import VGG16

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

이 후에는?

1. conv_base (합성곱 기반 층) 의 출력을 한 번만 실행하고 독립된 완전 연결 분류기에 입력으로 사용

1. conv_base 위에 Dense 층을 쌓아 확장

데이터 증식

- 합성 곱 연산은 비용이 많이 들기에, 1번은 빠르지만 데이터 증식을 사용할 수 없고,
- 2번은 느리지만 데이터 증식을 사용할 수 있다.

```
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

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

5.3 사전 훈련된 컨브넷 사용하기 - 특성추출

```
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.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
_________________________________________________________________
```

conv_base 위에 dense층을 2개 추가하여 네트워크를 만든 모습

```
print('conv_base를 동결하기 전 훈련되는 가중치의 수:',
      len(model.trainable_weights))
```

```
conv_base를 동결하기 전 훈련되는 가중치의 수: 30
```

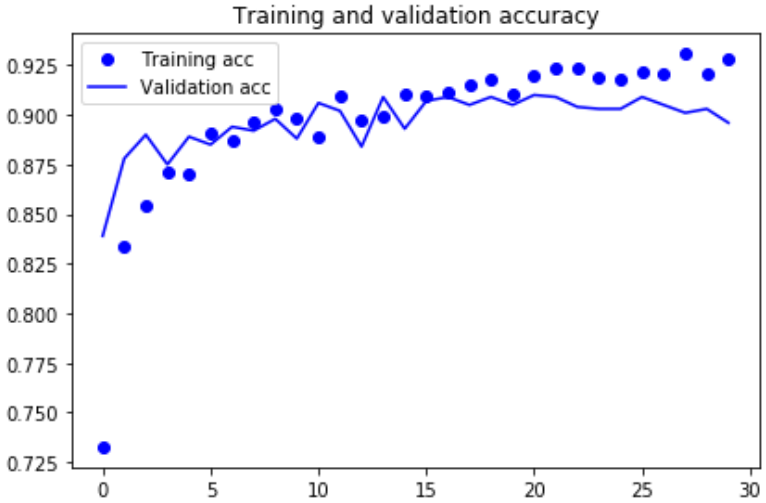
```
conv_base.trainable = False
```

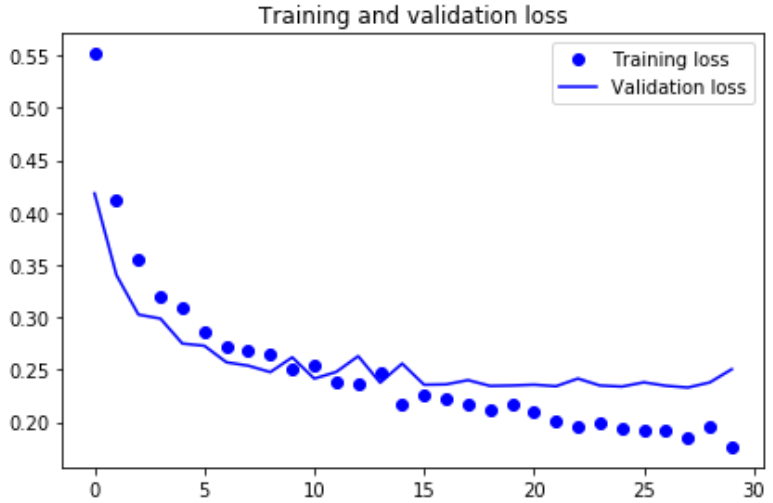
```
print('conv_base를 동결한 후 훈련되는 가중치의 수:',
      len(model.trainable_weights))
```

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

합성곱 기반 층은 동결,
Dense층만 가중치가 업데이트 되도록 설정

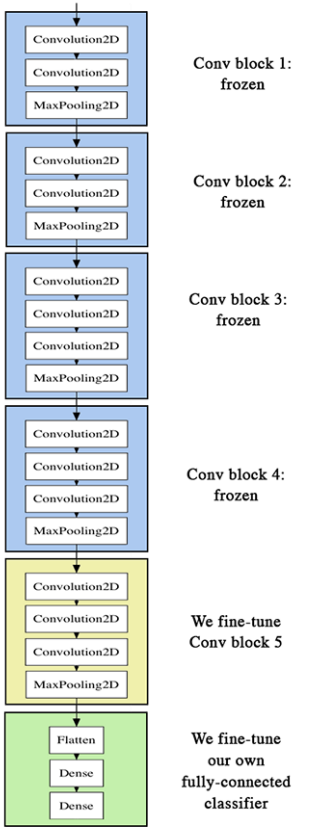
5.3 사전 훈련된 컨브넷 사용하기 - 특성추출





- 결과 :
  정확도는 비슷하지만, 과대적합이 많이 줄어들었다.

5.3 사전 훈련된 컨브넷 사용하기 - 미세 조정



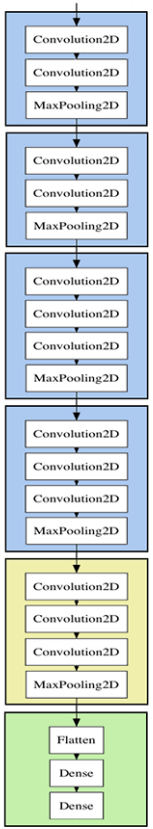
미세 조정 : 동결 모델의 상위 층 몇 개를 동결에서 해제하고 모델에 새로 추가한 층과 함께 훈련하는 것

```
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
```

- 진행 단계

1. 사전에 훈련된 기반 네트워크 위에 새로운 네트워크를 추가
2. 기반 네트워크를 동결
3. 새로 추가한 네트워크를 훈련
4. 기반 네트워크에서 일부 층의 동결을 해제
5. 동결을 해제한 층과 새로 추가한 층을 함께 훈련

5.3 사전 훈련된 컨브넷 사용하기 - 미세 조정



미세 조정 : 동결 모델의 상위 층 몇 개를 동결에서 해제하고 모델에 새로 추가한 층과 함께 훈련하는 것

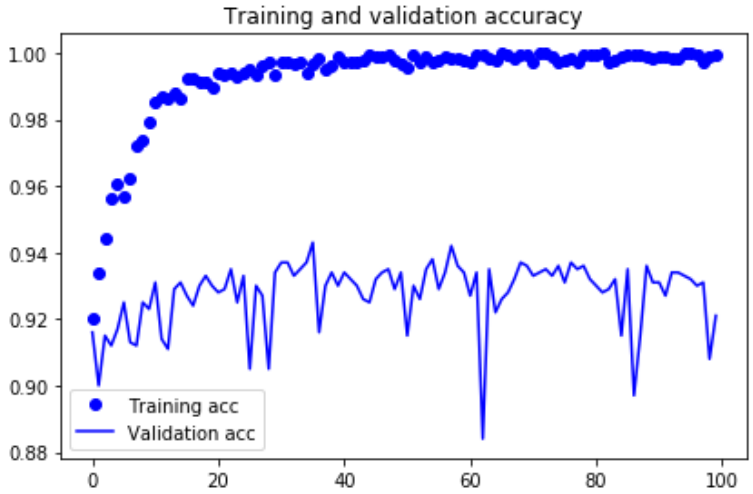
```
conv_base.trainable = True

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
```

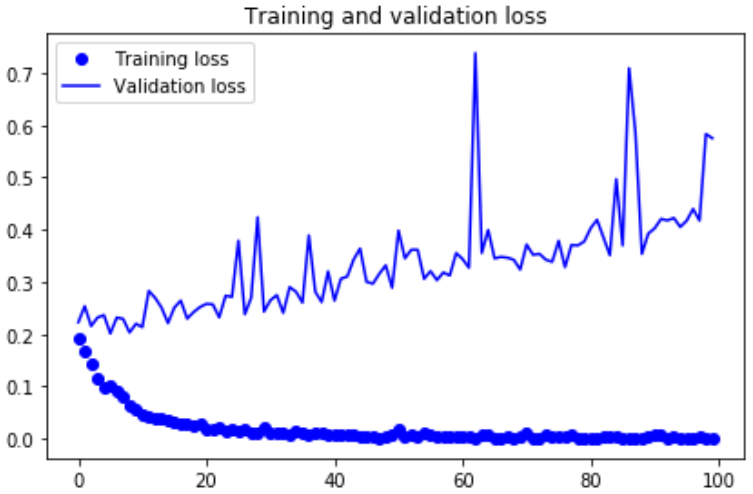
- 상위층만 미세 조정하는 이유?

합성곱 기반 층에 있는 하위 층들은 일반적이고 재사용가능한 특성들을 인코딩하고 상위 층들은 특화된 특성을 인코딩함. => 미세 조정에 대한 효과

훈련해야 할 파라미터가 많을수록 과대적합의 위험이 커진다.

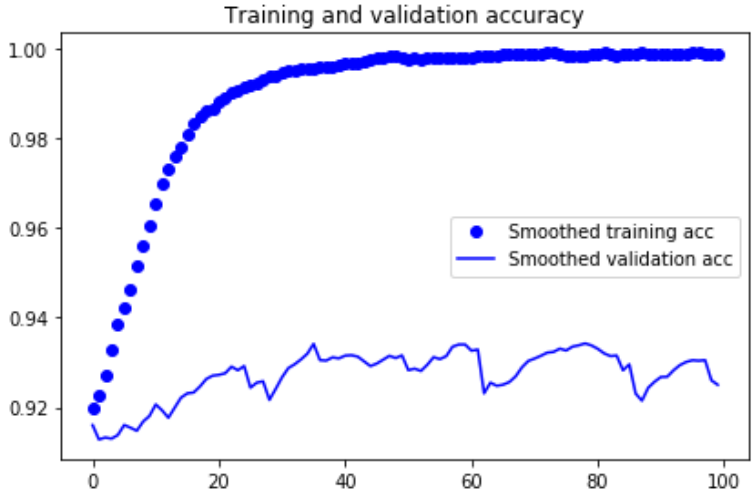
5.3 사전 훈련된 컨브넷 사용하기 - 미세 조정

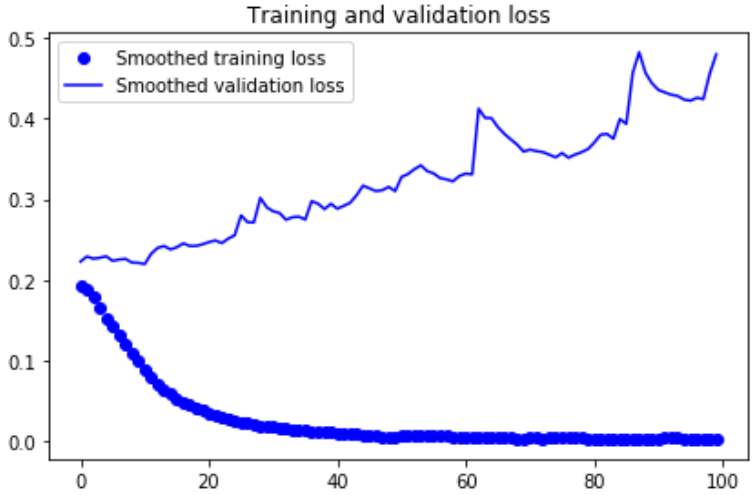




- 그래프가 불규칙하게 보임
  지수 이동 평균으로 정확도와 손실 값을 부드럽게 표현

5.3 사전 훈련된 컨브넷 사용하기 - 미세 조정





- 결과
  정확도 약 1-2%향상됨
  손실도는 악화되었으나, 이는 개별적인 손실 값의 평균을 그린 것이므로 큰 의미가 없다.

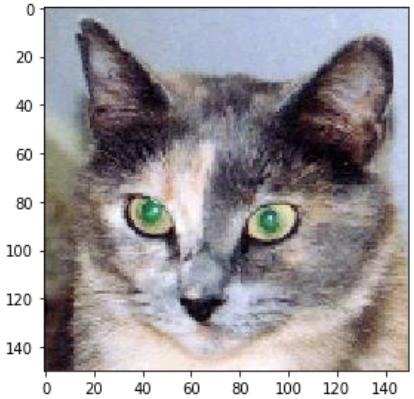
5.3 사전 훈련된 컨브넷 사용하기 - 정리

- 컨브넷은 컴퓨터 비전 작업에 가장 뛰어난 머신 러닝 모델입니다. 아주 작은 데이터셋에서도 처음부터 훈련해서 괜찮은 성능을 낼 수 있습니다.
- 작은 데이터셋에서는 과대적합이 큰 문제입니다. 데이터 증식은 이미지 데이터를 다룰 때 과대적합을 막을 수 있는 강력한 방법입니다.
- 특성 추출 방식으로 새로운 데이터셋에 기존의 컨브넷을 쉽게 재사용할 수 있습니다. 작은 이미지 데이터셋으로 작업할 때 효과적인 기법입니다.
- 특성 추출을 보완하기 위해 미세 조정을 사용할 수 있습니다. 미세 조정은 기존 모델에서 사전에 학습한 표현의 일부를 새로운 문제에 적응시킵니다. 이 기법은 조금 더 성능을 끌어올립니다.

5.4 컨브넷 학습 시각화

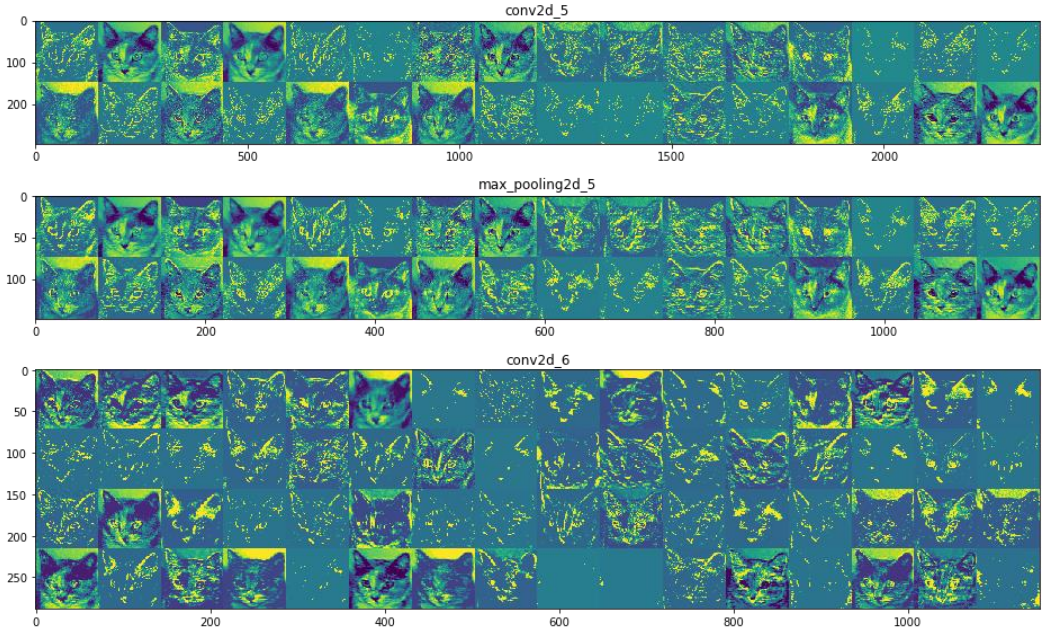
1. 컨브넷 중간층의 출력을 시각화하기
1. 컨브넷 필터를 시각화하기
1. 클래스 활성화에 대한 히트맵을 이미지에 시각화하기

5.4.1 중간층의 활성화 시각화하기

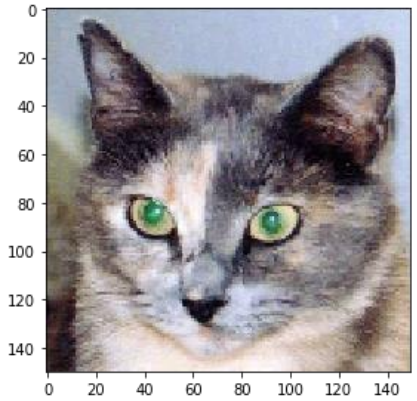


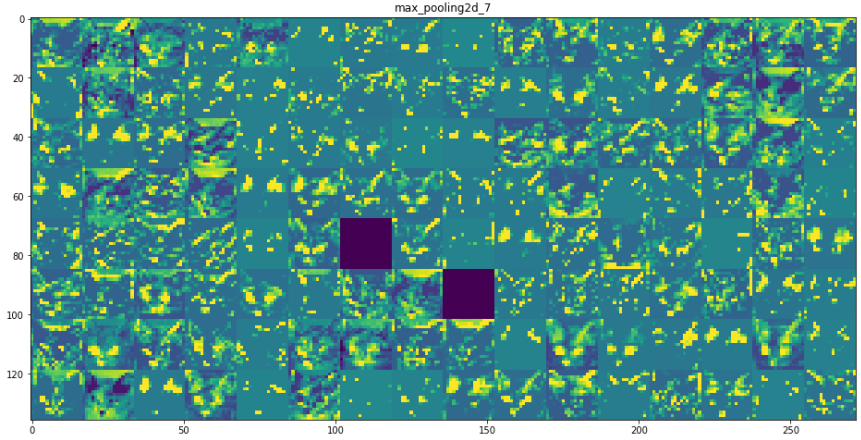
- 하위층

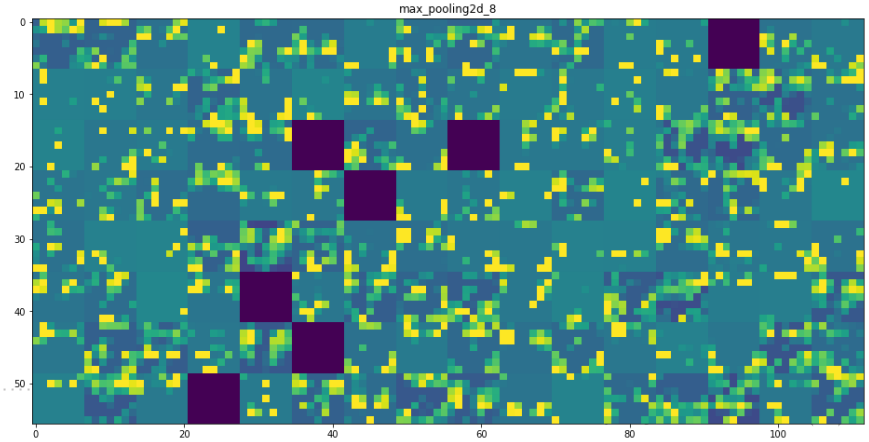
비교적 고양이의 특성이 잘 나타난다.
초기 사진에 있는 거의 모든 정보가 유지된다.



5.4.1 중간층의 활성화 시각화하기







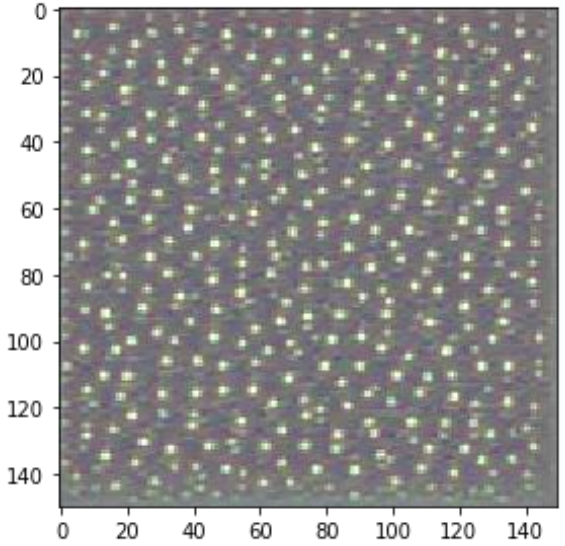
- 상위층

활성화는 점점 더 추상적으로 되고 시각적으로 이해하기 어려워진다. '고양이 귀'와 '고양이 눈'과 같이 고수준의 개념을 인코딩하기 시작한다.
상위 층의 표현은 이미지의 시각적 콘텐츠에 관한 정보가 점점 줄어들고 이미지의 클래스에 관한 정보가 점점 증가합니다.

5.4.2 컨브넷 필터를 시각화하기

컨브넷의 각 필터가 반응하는 시각적 패턴을 그려보자.

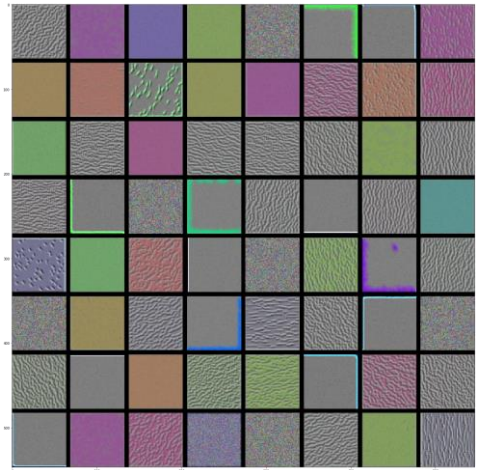
- 특성 합성곱 층의 한 필터 값을 최대화하는 손실 함수를 정의하고, 활성화 값을 최대화하기 위해 입력 이미지를 변경하도록 확률적 경사 상습법을 사용.

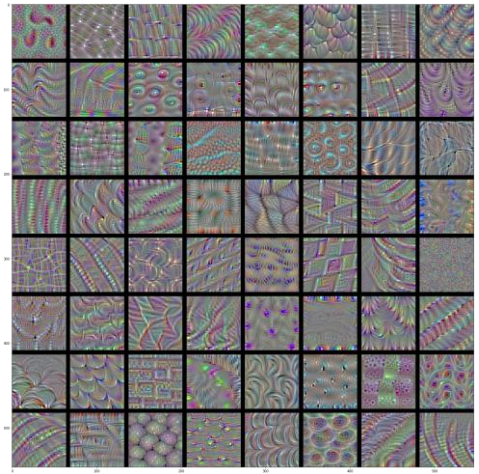


- block3_conv1 층의 0번째 채널이 최대로 반응하는 패턴

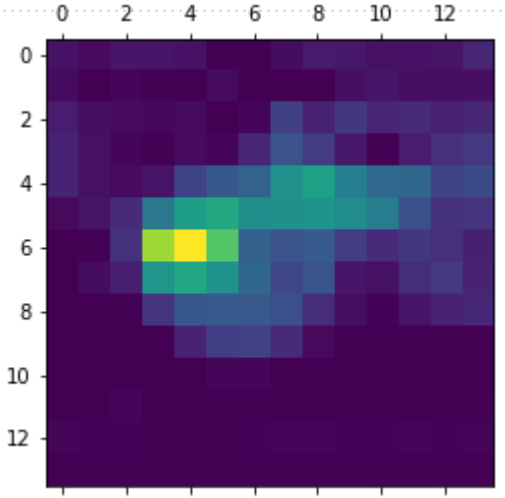
5.4.2 컨브넷 필터를 시각화하기

- 컨브넷의 각 층은 필터의 조합으로 입력을 표현할 수 있는 일련의 필터를 학습한다.
- 첫 번째 층의 필터는 (왼쪽 그림) 간단한 대각선 방향의 에지와 색깔을 인코딩하고,
  상위 층의 필터는 (오른쪽 그림) 깃털, 눈, 나뭇잎 등 자연적인 이미지에서 찾을 수 있는 질감을 닮아 가기 시작한다.





5.4.3 클래스 활성화에 대한 히트맵 이미지에 시각화



테스트 사진에 대한 아프리카 코끼리 클래스 활성화 히트맵

- 이미지의 어떤 부분이 컨브넷의 최종 분류 결정에 기여하는가?
- 클래스 활성화 맵 : 특성 출력 클래스에 대해 입력 이미지의 모든 위치를 계산한 2D 점수 그리드
- 입력 이미지가 주어지면 합성곱 층에 있는 특성 맵의 출력을 추출, 특성 맵의 모든 채널 출력에 채널에 대한 클래스의 그래디언트 평균을 곱함.

5.4.3 클래스 활성화에 대한 히트맵 이미지에 시각화

- 입력 이미지가 각 채널을 활성화하는 정도에 대한 공간적인 맵을 클래스에 대한 각 채널의 중요도로 가중치를 부여하여 입력 이미지가 클래스를 활성화하는 정도에 대한 공간적인 맵을 만듬.

- 네트워크가 아프리카 코끼리라고 생각하는 이유와 사진 어디에 코끼리가 위치하는지 볼 수 있다.

원본 이미지에 클래스 활성화 히트맵을 겹친 이미지