이미지 식별의 어려움 - 기존방식

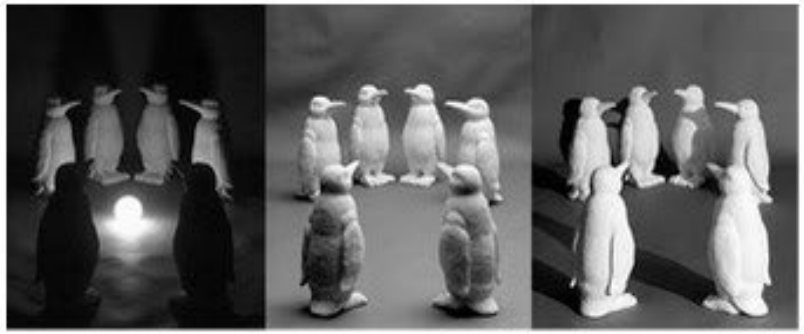
Viewpoint variation

Scale variation

Deformation

Occlusion

Illumination conditions



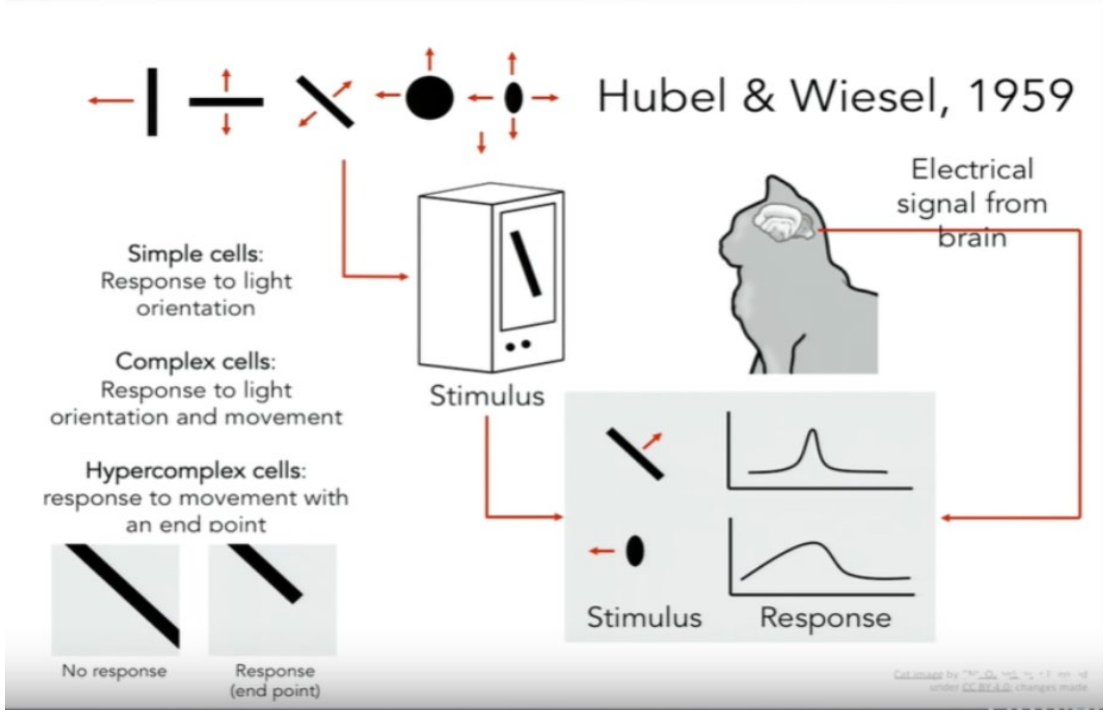
Background clutter

Intra-class variation

https://cs231n.github.io/classification/

시각의 작동원리

- Hubel & Wiesel의 실험(1959)
- 두뇌가 눈에 들어오는 이미지를 분해하여 처리하는 방식을 연구
  - 우리 두뇌는 밖에서 들어온 이미지를 선, 움직임, 색이라는 요소로 분해하여 처리한 후, 다시 재구성하여 물체를 인식함.
  - 단순한 세포는 빛의 방향(수직, 수평 등)에 민감하게 반응
  - 좀 더 복잡한 세포는 빛의 방향과 이동에 민감하게 반응
  - 더 복잡한 세포는 어떤 방향으로 이동하는 물체의 가장자리의 패턴에 민감하게 반응
- 시각 처리는 가장자리의 형태와 같은 가장 작은 구조에서 시작하여 점점 더 복잡한 실체를 인식해나감.

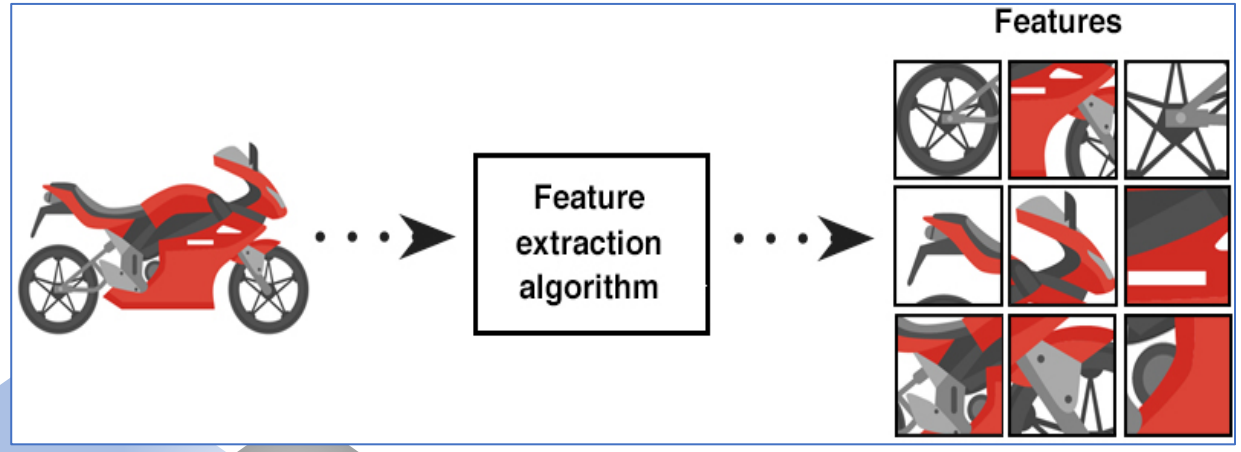


http://cs231n.stanford.edu/

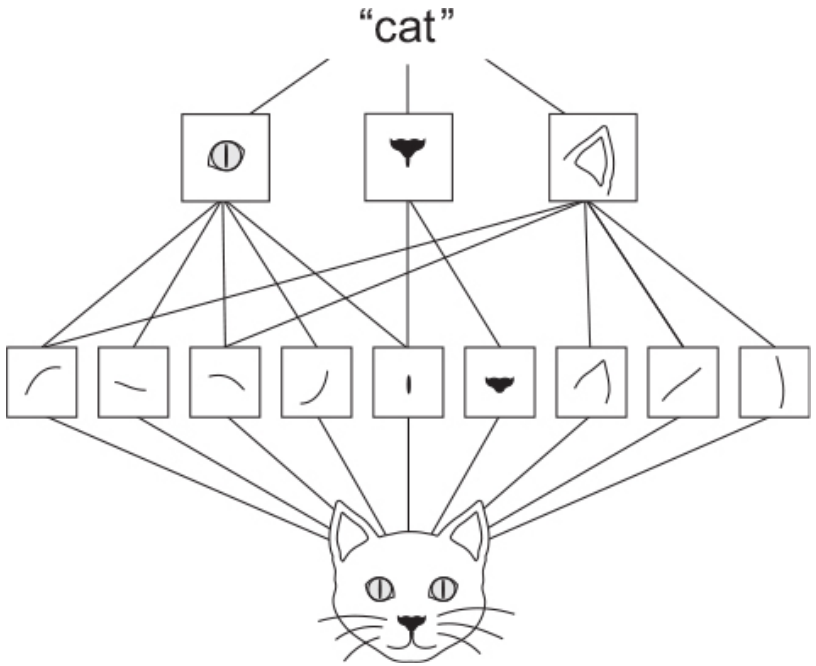
CNN(convolutional Neural Network) 원리

- 기본 신경망은 이미지 입력을 1D로 풀어놓고 학습.
  - 이 구조에서는 이미지를 효율적으로 처리하지 못하고 학습파라미터 수도 매우 크게 됨.
- 기본 딥러닝 구조에 CNN 구조 추가(Convolution(filter), MaxPooling)
- CNN에서는 이미지에 2D 윈도우를 적용하여 지역패턴을 학습.
- 학습된 패턴은 이동불변성을 가짐. 즉 학습된 패턴이 이미지의 어느 구역에 있더라도 인식가능.
- 연속적인 층 구성을 통하여 패턴의 공간적 계층구조(작은 부분에서 시작하여 큰 부분으로)를 학습가능.

부분의 인식

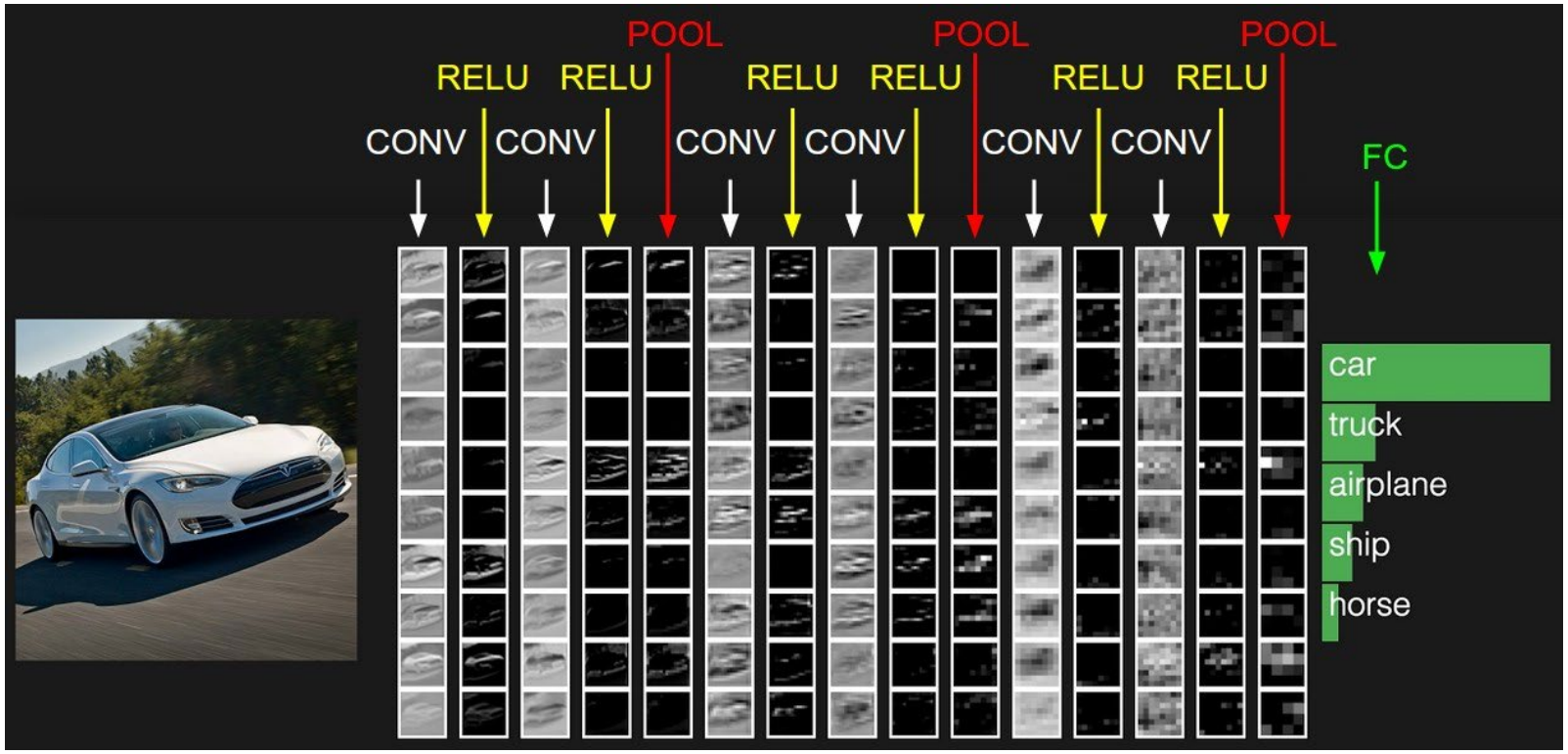


- 공간적 계층구조 인식
  - 아주 작은 지역의 에지를 인식.
  - 에지들을 종합하여 눈, 귀, 코 등 인식.
  - 중간인식된 개체들이 합해져 고양이를 인식.



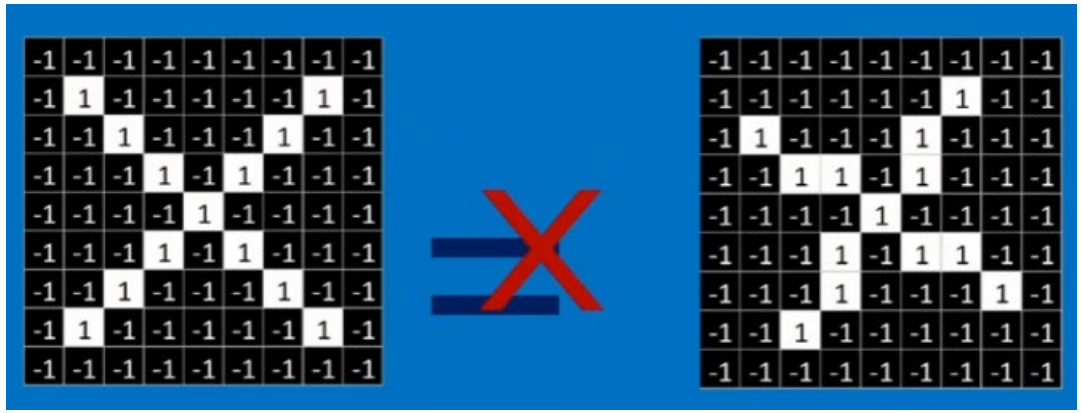
CNN 기본 구조

- 간략한 연산과정
  - 입력에 2D 필터(또는 receptive field) 를 적용하고 통과한 2D 값의 합으로 변환
  - stride간격으로 움직이면서(Convlove) 전체 입력이미지에 대하여 연산 수행
  - MaxPooling 과정을 통해 필터에서 보는 영역을 확대하고 학습 파라미터 양을 줄임.
  - FCN을 통하여 목표값 학습.

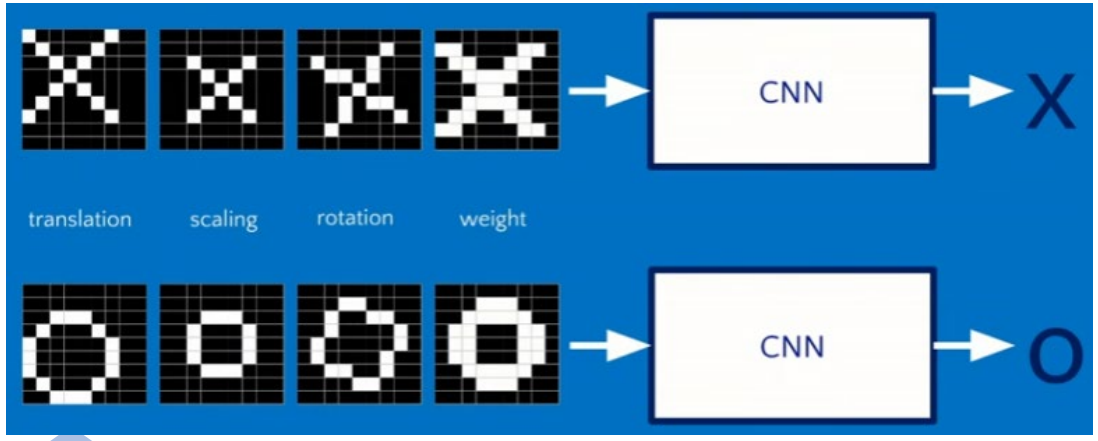


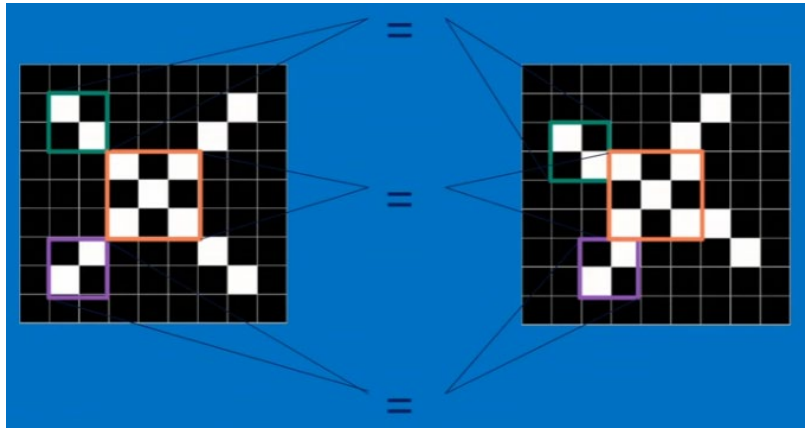
기존방식 vs. CNN

기존 Vision 알고리즘



CNN 알고리즘





https://www.youtube.com/watch?v=JB8T_zN7ZC0

CNN 기본 구조

- (실습)
- MNIST 예제를 기존 FCN(97.8% 정확도) 대신 CNN 사용해서 분류.
- 기존 1차원(28 * 28)입력 대신 2차원 입력(28, 28, 1) 사용
- 입력형식 = (가로픽셀수, 세로 픽셀수, 컬러채널)
- Conv2D, MaxPooling2D : 이미지 처리 기능
- layers.Flatten() : 일반 분류연산을 위해 데이터 펼침
- layers.Dense(10, activation='softmax') : 분류 연산 실행

Conv2D 층

- 대부분의 계산 작업을 수행하는 컨볼루션 네트워크의 핵심층
- 너비, 높이, 깊이(필터 개수)로 구성.
- 입력이미지에서 추출가능한 특성학습정보를 저장하는 역할
- 포워드 패스 동안 입력 이미지의 너비와 높이에 걸쳐 각 필터를 슬라이드하고 입력이미지와 내적을 계산
- 해당 필터의 응답을 제공하는 2차원 활성화 맵 생성
- 앞쪽층은 에지, 색상 등 지역적인 특성을 학습
- 뒤쪽으로 진행하면서 앞쪽 층의 특성으로 구성되고 실제 타겟과 더 밀접한 정보 학습(클래스 분류정보, 개의 품종관련정보 등)
- 층이 깊어질수록 많은 수의 필터 배치.

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

- (28, 28,1)입력을 받아서
- (3, 3) 윈도우로 슬라이딩
- → (26, 26) 크기, 32 개의 출력특성맵(필터) 생성
- 필터 : 입력데이터에 존재하는 특징 인식(예를 들어 코, 귀).

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320

max_pooling2d (MaxPooling2D  (None, 13, 13, 32)        0
)

conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496

max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)          0
2D)

conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928

=================================================================
```

Conv2D 층

- 슬라이딩
  - 입력특성맵 전체(가로, 세로, 깊이)를 정해진 패치(3,3)가 순회
- 패딩
  - 입력의 크기만큼 출력을 얻고 싶을 때 적용
  - 가장자리를 0으로 채운 패딩 추가
  - valid : 패딩 미사용(출력크기가 줄어듬, 가장자리 정보 소실)
  - same : 패딩 사용하여 출력크기를 입력크기와 동일하게 함.
- 스트라이드
  - 필터가 입력이미지를 이동하는 픽셀간격. 기본값은 1.

(5, 5) 크기의 입력에 대하여 (3, 3) 패치를 적용시 패딩의 사용

Figure 5.5. Valid locations of 3 × 3 patches in a 5 × 5 input feature map

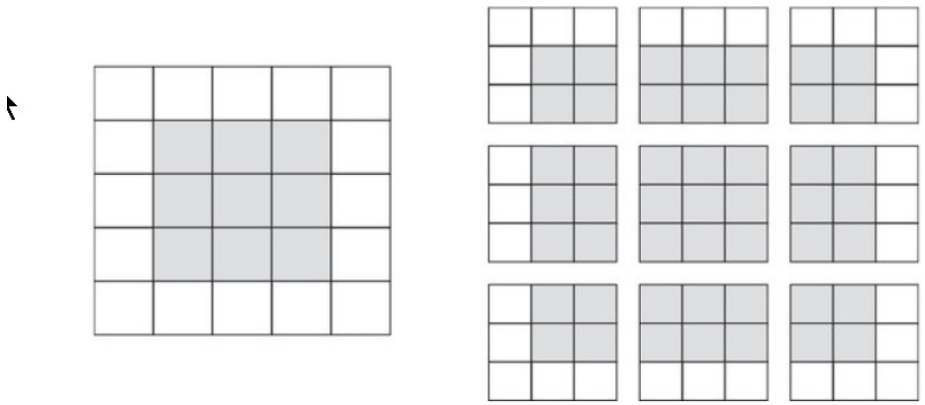
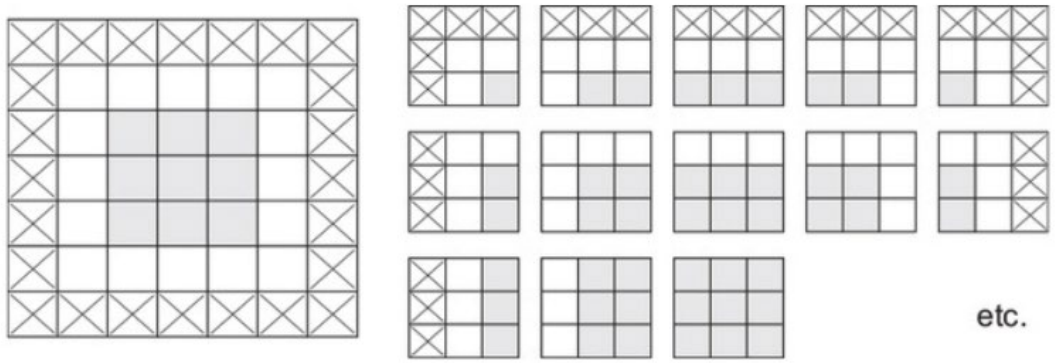
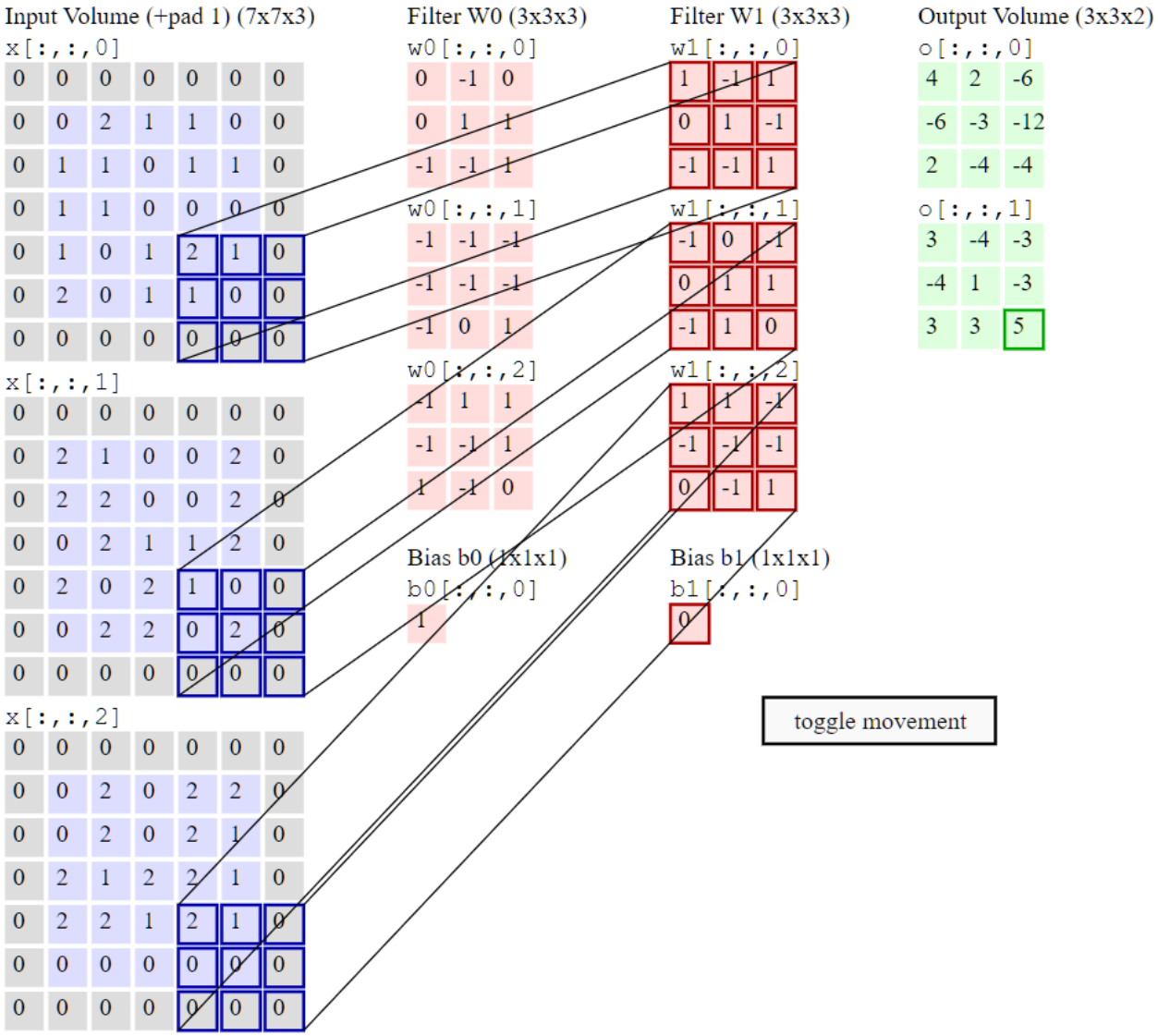


Figure 5.6. Padding a 5 × 5 input in order to extract 25 3 × 3 patches



Conv2D 층

- 입력 : 7x7 이미지, 3 채널
- Padding : 1
- Stride : 2
- 필터(Conv2D) : 2 개, 3 채널
- 출력 : 3x3 이미지.2 개

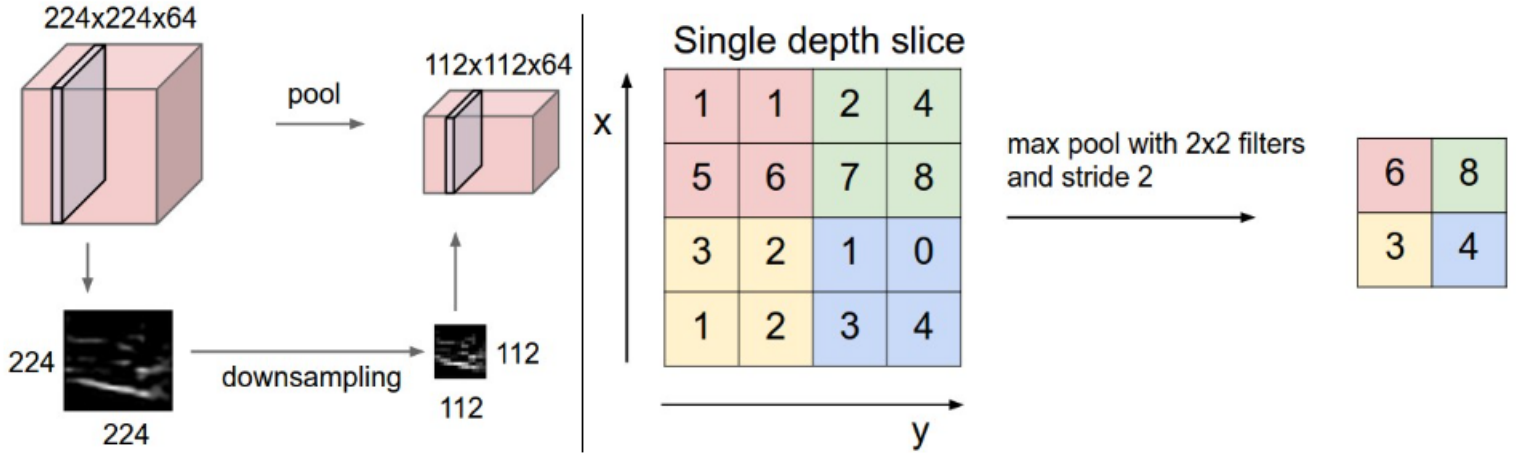


https://cs231n.github.io/convolutional-networks/

MaxPooling2D

- (2, 2) 패치, stride = 2 사용하여 각 채널별로 최대값 출력
- (26, 26, 32) → (13, 13, 32)
- 중요한 정보(신호)를 살리면서 특성맵의 크기를 줄이는 역할
- 표현의 공간 크기를 점진적으로 줄여 네트워크에서 매개변수와 계산의 양을 줄이고 따라서 과적합도 제어
- 연속적인 합성곱 층 연산결과를 커진 윈도우를 통해 보므로 공간적 계층구조를 인식하는데 유리.

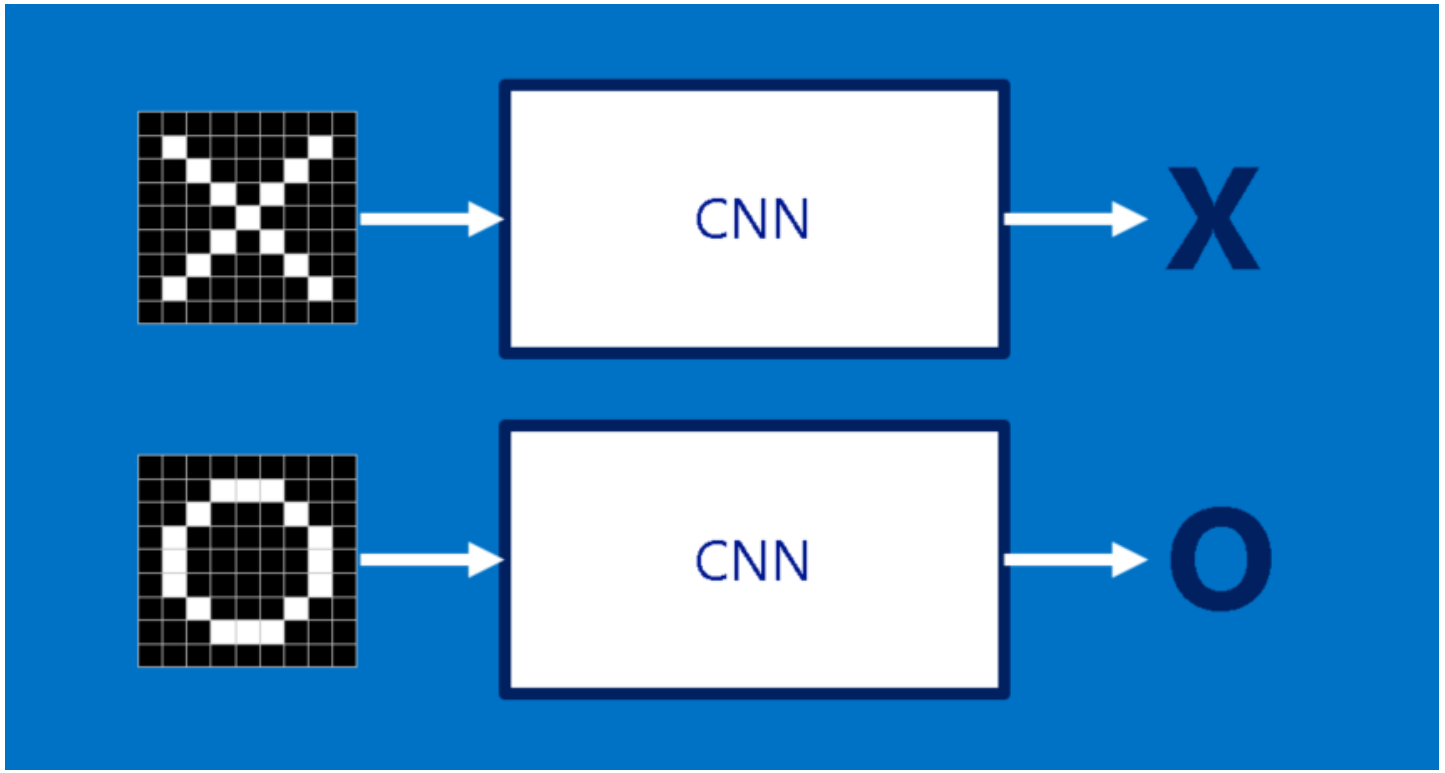
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928

=================================================================
```



https://cs231n.github.io/convolutional-networks/

OX 예측시 작동 사례

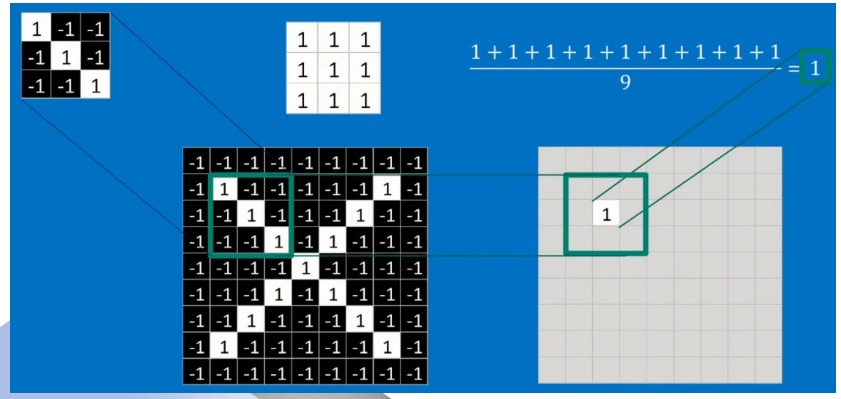
- 입력이미지를 필터 크기로 순회하면서 학습된 필터에 정의된 특징들을 찾음.
- 필터는 학습 과정에서 훈련데이타에서 공통된 특징들을 찾음.
- OX 문제에서 X를 대상으로는 대각선, x 형태의 패턴을 학습할 것임.

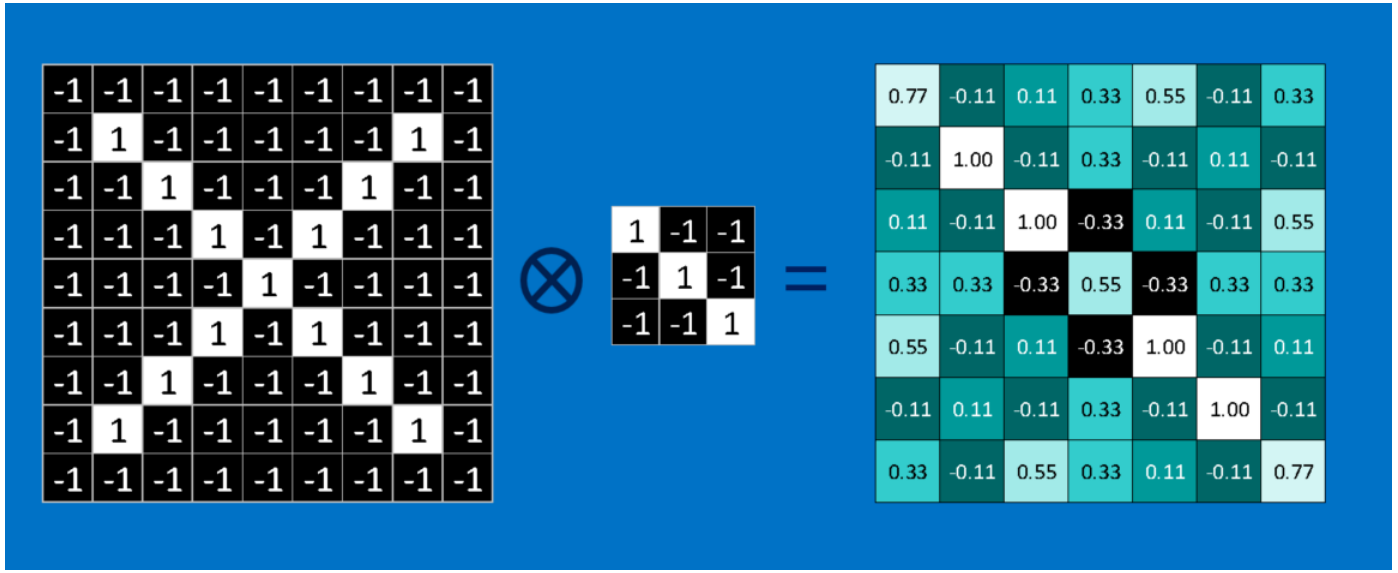


https://www.kdnuggets.com/2016/08/brohrer-convolutional-neural-networks-explanation.html

Conv layer에서 일어나는 작업

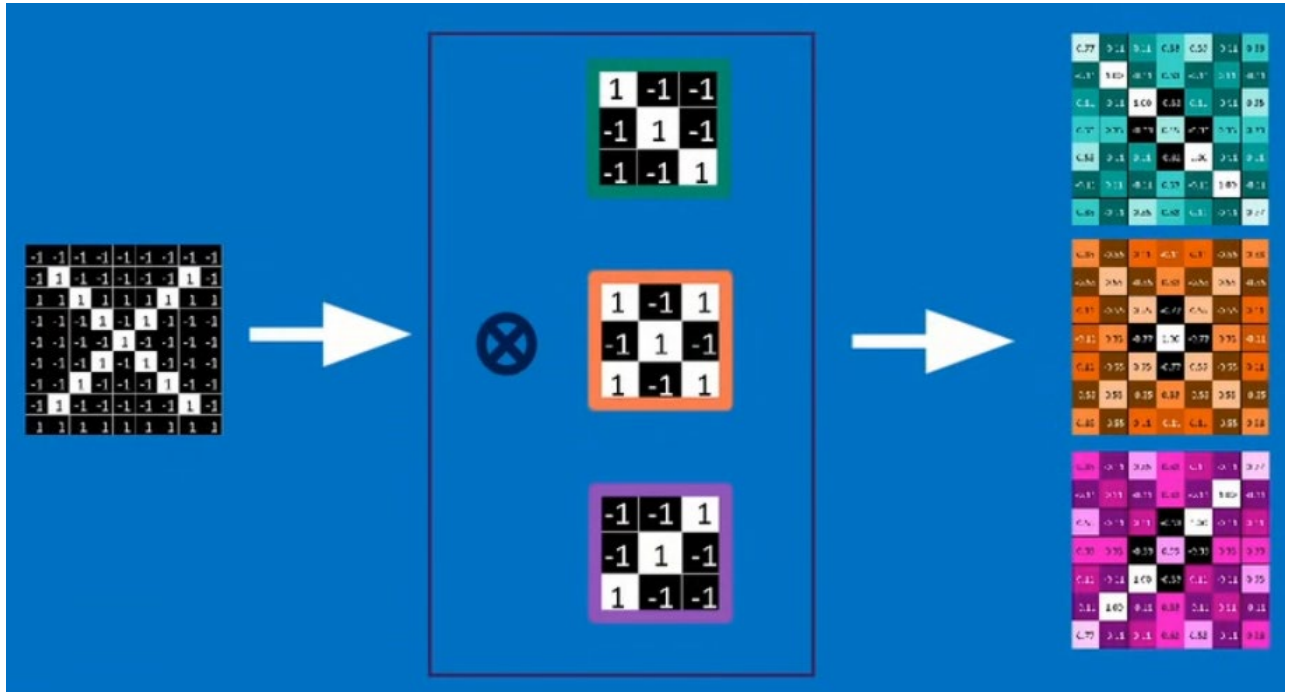
- 예측시에 입력이미지를 대상으로 'convolve' 하면서 대각선, 'X' 패턴을 모든 위치에서 찾음.
- 찾는 방법
  - 필터(3x3)와 입력에서 임의의 위치의 윈도우(3x3) 를 대상으로 픽셀별로 비교
  - 1 값은 흰색(또는 강한 신호), -1(혹은 낮은) 값은 검은색(또는 약한 신호)
  - 필터의 1값 위치에 1(또는 높은 값)이 있으면 픽셀별로 곱했을 때 큰 값 산출
  - 필터의 -1값 위치에 -1(또는 낮은 값)이 있으면 픽셀별로 곱했을 때 큰 값 산출
  - 반대의 경우 음수 산출
  - 곱한 값의 절대값이 클 수록 필터의 패턴이 (정상 또는 역상으로) 존재하는 것임.
  - 이미지 전체를 convolve 하면서 연산.
- (해석)
  - 1에 가까운 값 : 입력 이미지에서 필터가 위치하는 위치 알려줌.
  - -1에 가까운 값 : 필터의 역상위치 알려줌.
  - 0에 가까운 값 : 필터 패턴 찾지 못함.





Conv layer에서 일어나는 작업 예

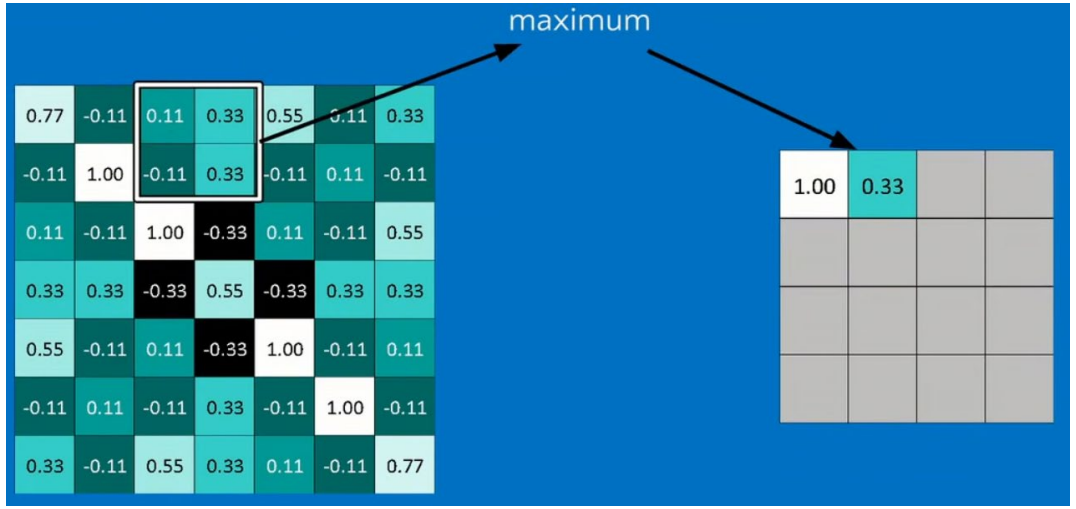
- 모든 필터에 대해 수행

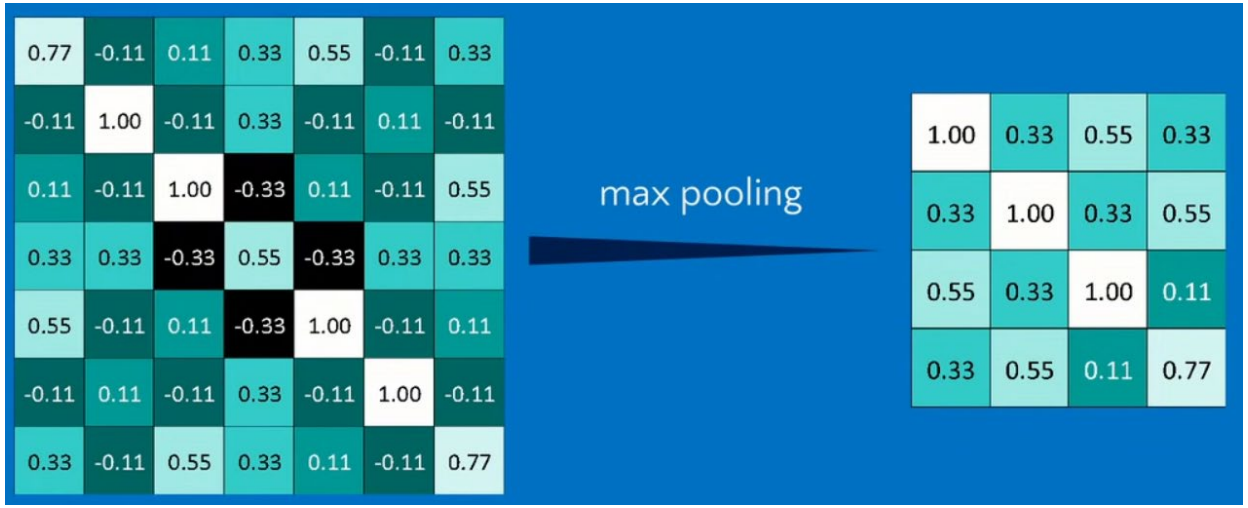


Pooling layer에서 일어나는 작업 예

- (보통) 2x2 윈도우, stride=2 를 적용하여 필터의 출력에 대하여 윈도우에서 가장 큰 값만 취함.
- Pooling 후 크기는 약 ¼ 로 줄어듬.
- 윈도우별로 가장 큰 값(필터에 강한 신호를 보인 위치)을 취하므로 **필터 매칭정보는 유지하고 있음(**다만 위치정보는 희미해짐).
- 전체 이미지가 축소되어 인코딩되는 효과.

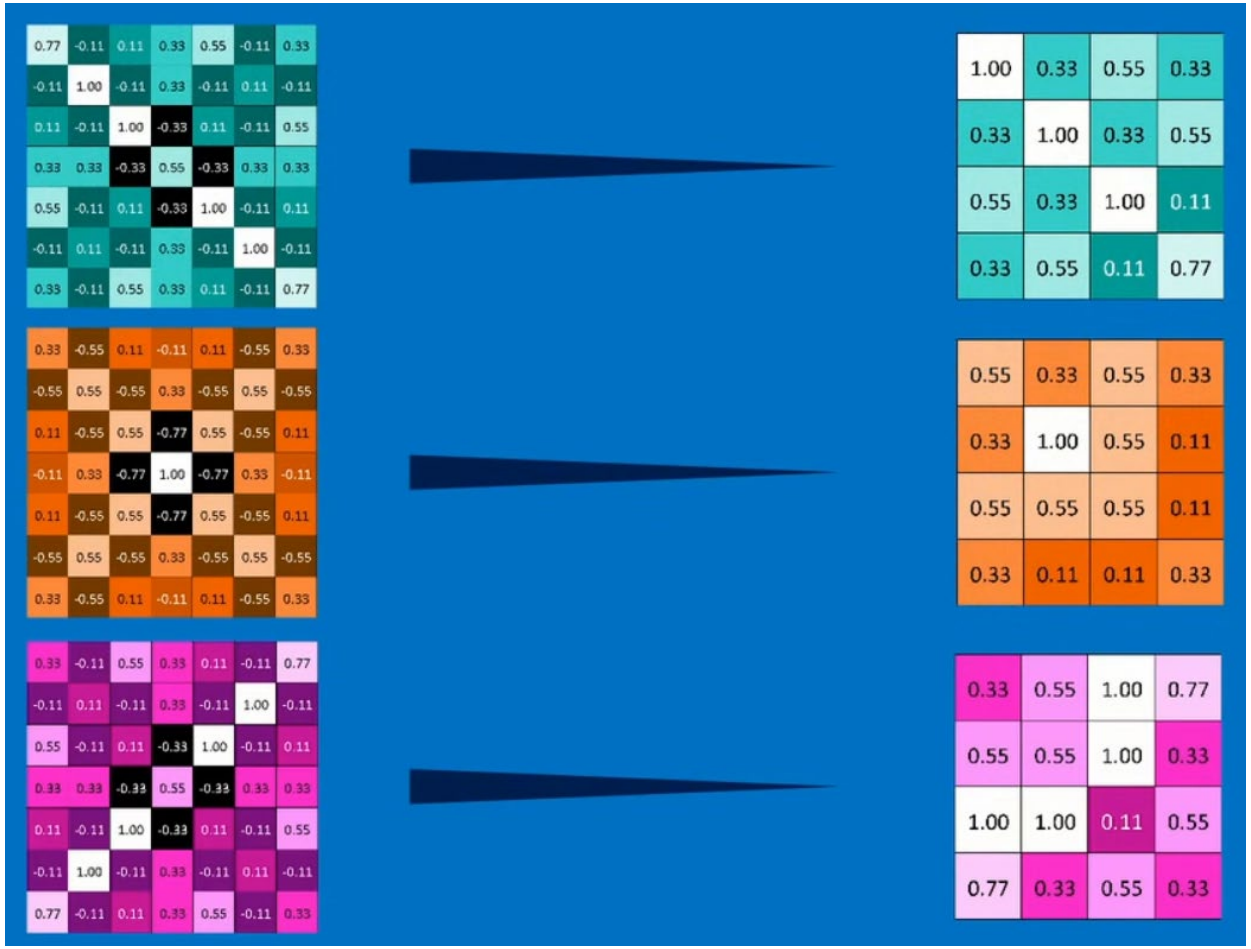
2x2 window, stride:2
7 x 7 image → 4 x 4





Pooling layer에서 일어나는 작업 예

- 모든 필터출력에 대하여 MaxPooling 수행
- 학습은 일어나지 않고 크기가 약 ¼로 줄어듬.

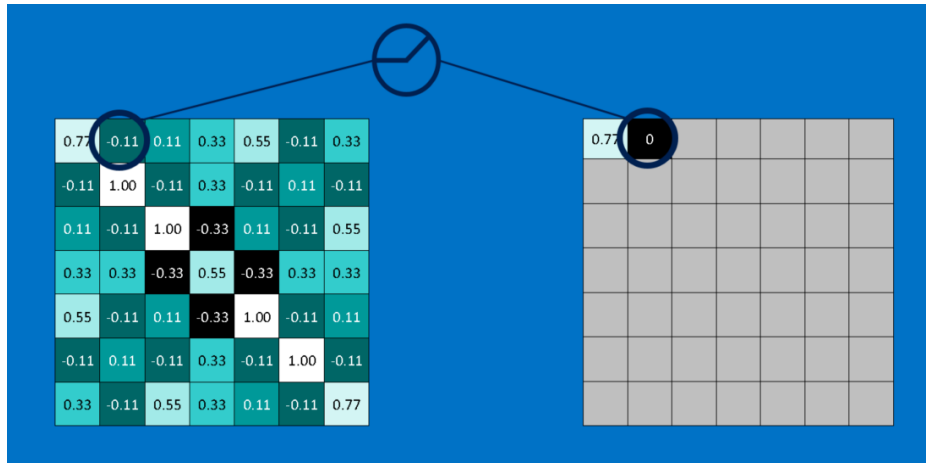


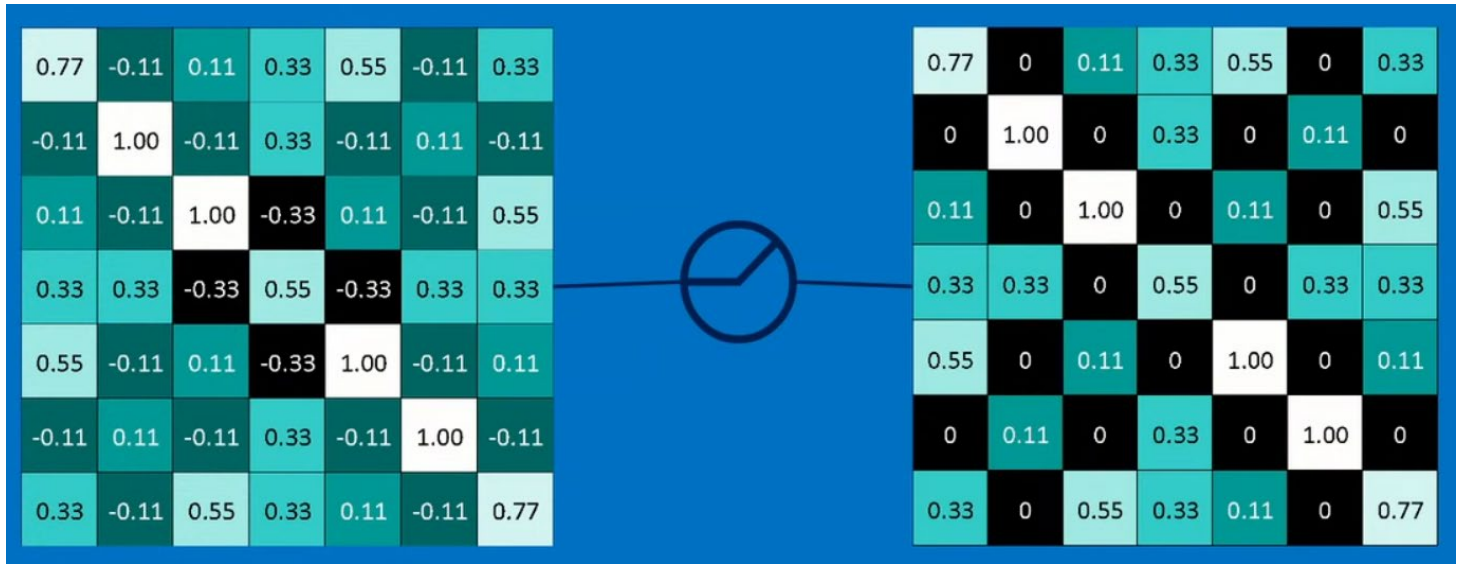
Activation Layer에서 일어나는 작업 예

- ReLU(Rectified Linear Unit) 적용
- 음수인 값을 모두 0으로 대체. 학습과정에 비선형성 부여.
- Conv -> ReLU 가 반복될 때 주로 적용됨.

전형적인 층쌓기 패턴

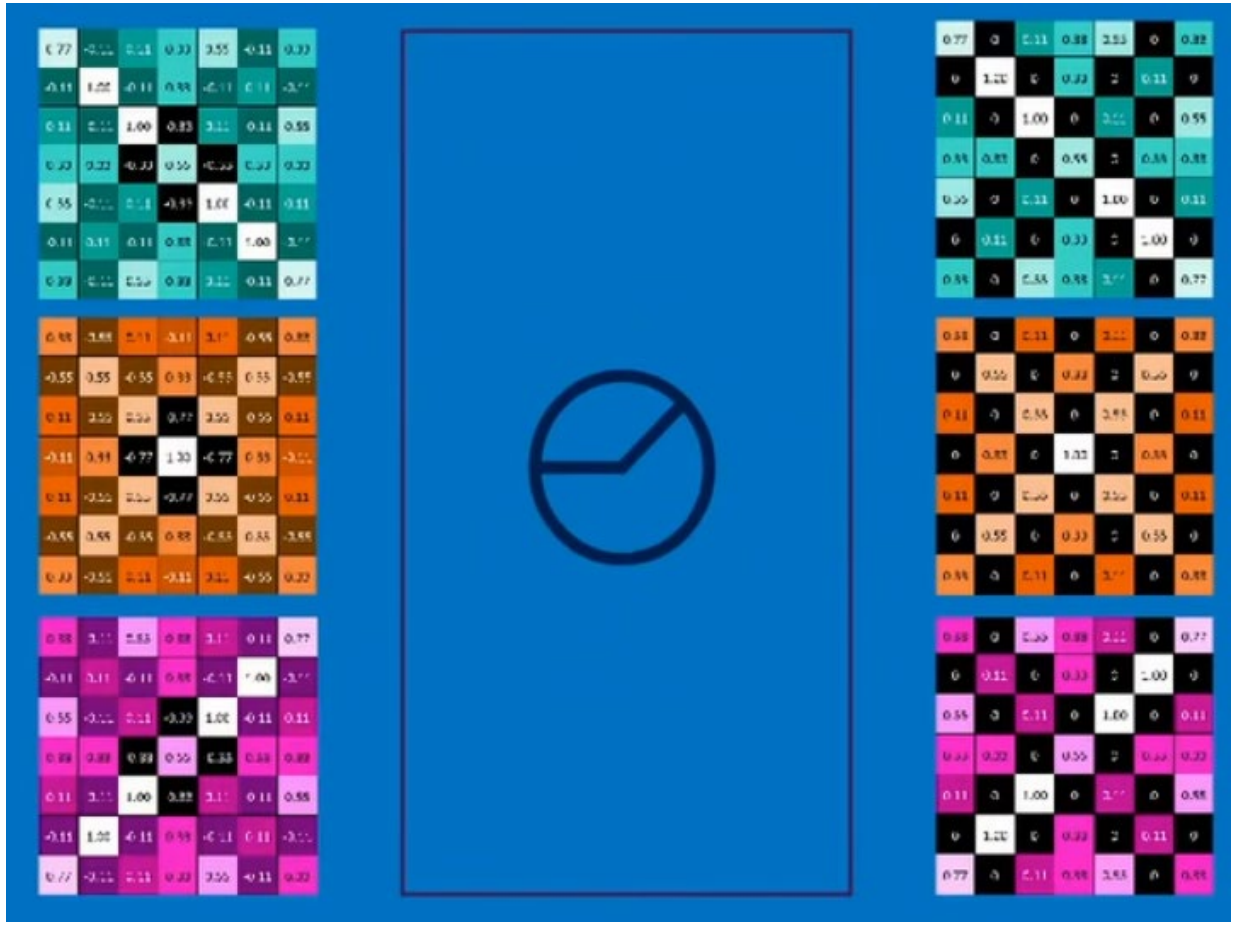
```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```





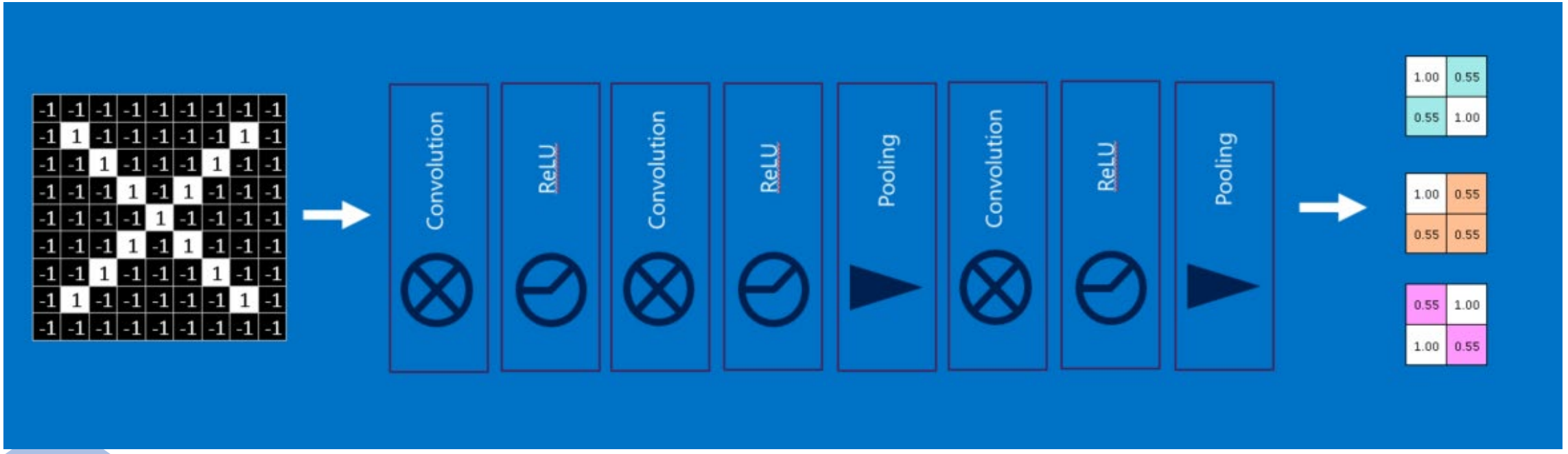
Activation Layer에서 일어나는 작업 예

- 모든 pooling layer 출력에 대해 실행.



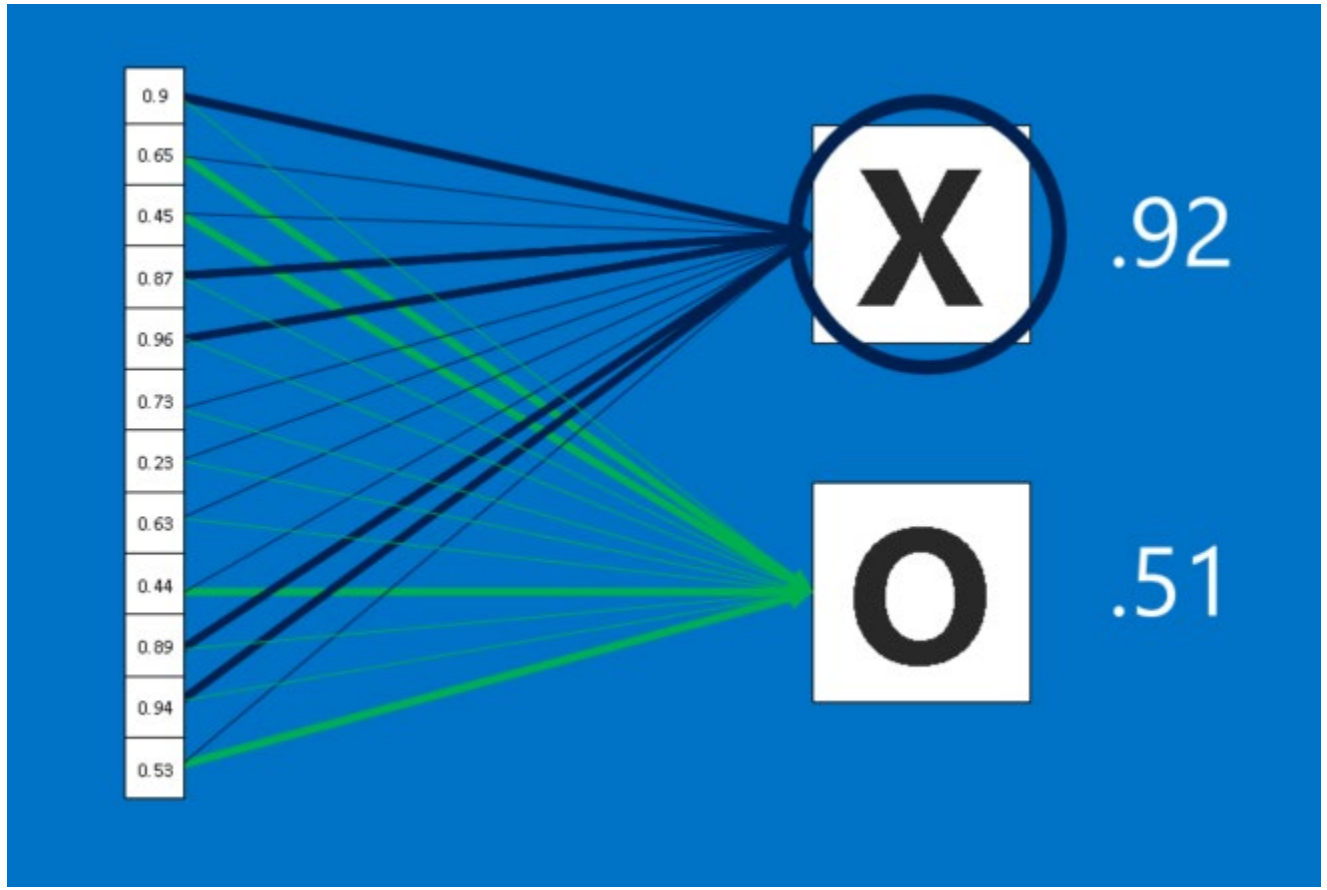
Convolution-ReLU-Pooling

- Conv – ReLU – MaxPooling 과정의 반복적인 연산을 통하여 원본 이미지를 필터 패턴을 학습하면서 크기는 줄어드는 변환을 함.
- 위의 과정을 연속적으로 수행 가능
- 층이 깊어지면서 필터는 점점 더 크고 복잡한 특성을 학습하게 됨.
- 초기 층은 윤곽선, 밝은 객체 등 기본적인 패턴을 학습하고 층이 깊어지면서 특정 모양, 패턴 등 복잡하고 모양을 인식가능한 패턴을 학습.

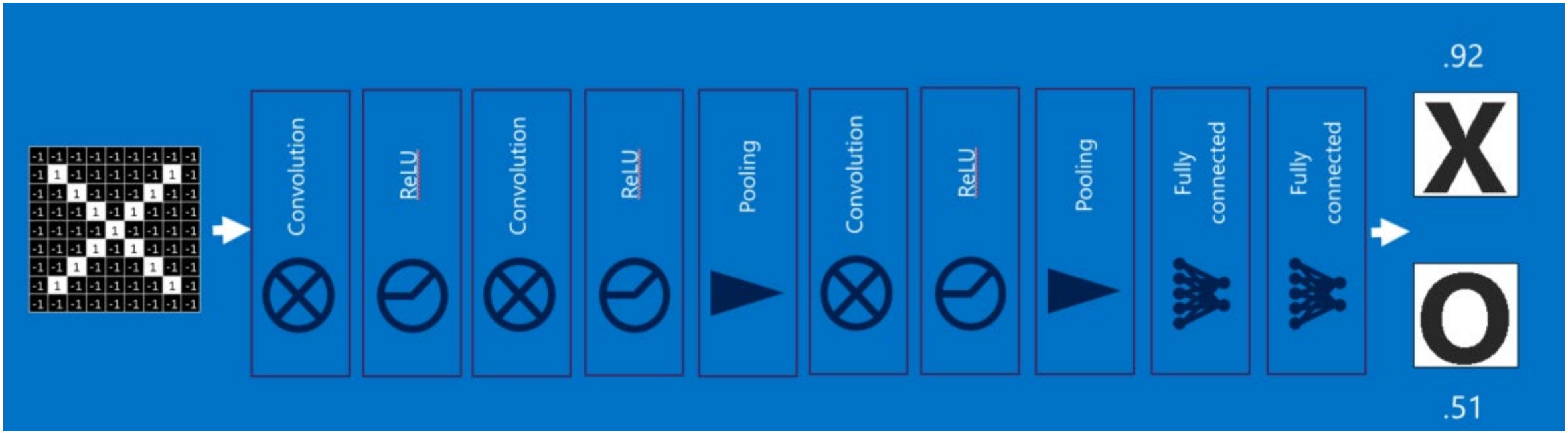


Fully Connected Layer

- 필터의 출력을 목표값으로 변환하는 단계
- Dense layer 이며 목표값을 결정하기 위한 가중치와 절편을 학습함.

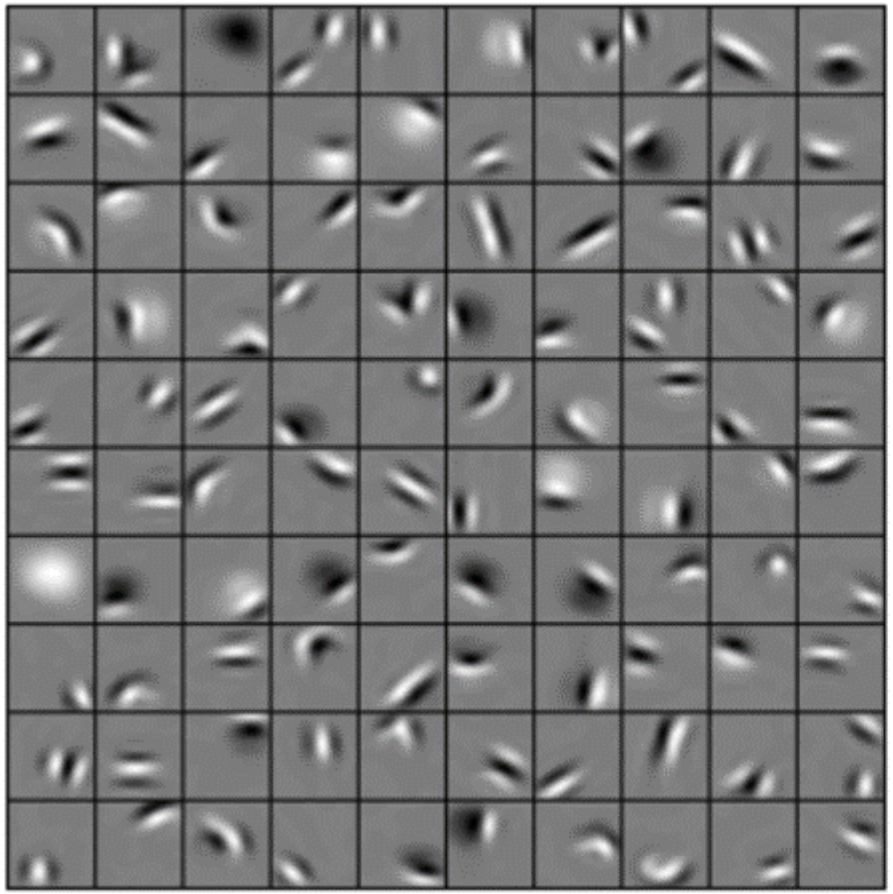


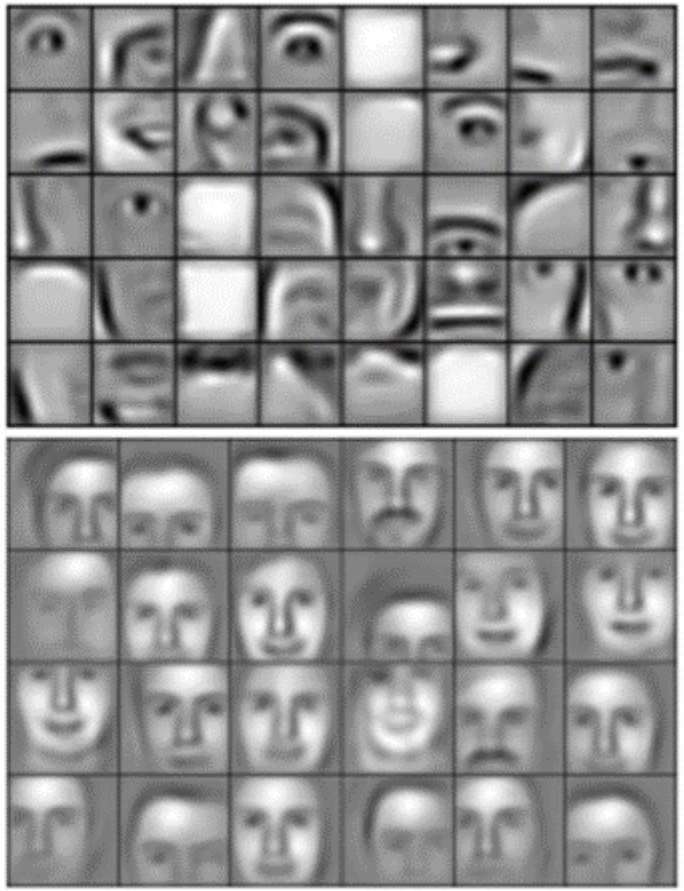
예측 전체 과정


Convolution
ReLU
Convolution
ReLU
Pooling
Convolution
ReLU
Pooling
Fully connected
Fully connected
.92
X
O
.51

얼굴을 대상으로 학습한 필터출력의 예

- 얼굴을 대상으로 학습했을 때 층이 깊어짐에 따라 필터의 출력이 얼굴 형태를 보임.
- Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, Honglak Lee, Roger Grosse, Rajesh Ranganath, Andrew Y. Ng





고양이-개 분류하기

- 데이타셋
  - 2013년 kaggle에서 컴퓨터비전 경연대회를 위해 만들어진 데이타셋
  - Kaggle 경연에서는 컨브넷 사용자가 95% 정확도로 우승함.

- (실 습)
  - 전체 데이타중 일부만 사용
  - 각각 1000 개의 샘플로 이루어진 훈련세트
  - 각각 500개의 샘플로 이루어진 검증 세트
  - 각각 500개의 샘플로 이루어진 테스트 세트

- 방법 1
  - 작은 컨브넷 모델로 예측
- 방법 2
  - 데이터 증식기법으로 학습데이타를 보강한 후 예측
- 방법 3
  - 사전훈련된 네트워크를 이용한 예측

Colab 사용

- 1. 구글 계정 필요함.
  - 1. https://colab.research.google.com/ -> 구글 계정 로그인 > 파일/노트열기 > Github >

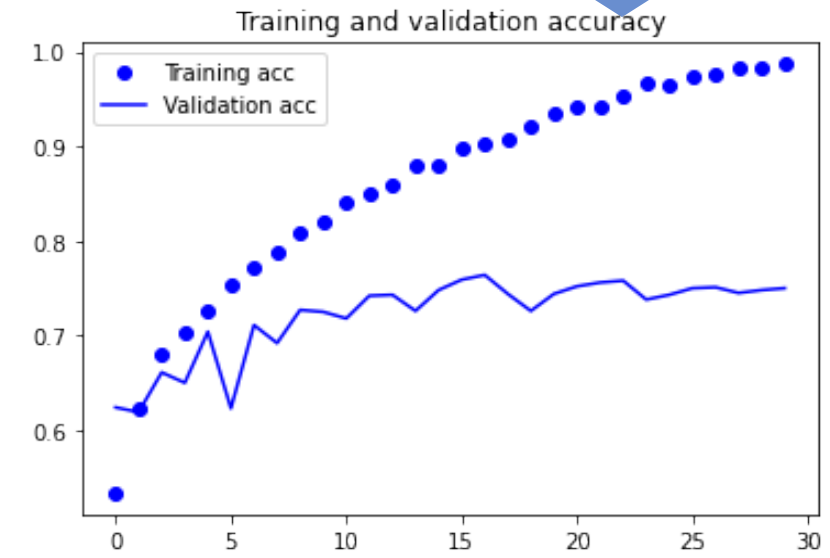
- 2. 저장소 : gbmax/open > 7.1 ...파일 선택
  - 인식할 수 없는 런타임 '...'이(가) 기본값인 'python3'(으)로 설정됩니다. > 확인

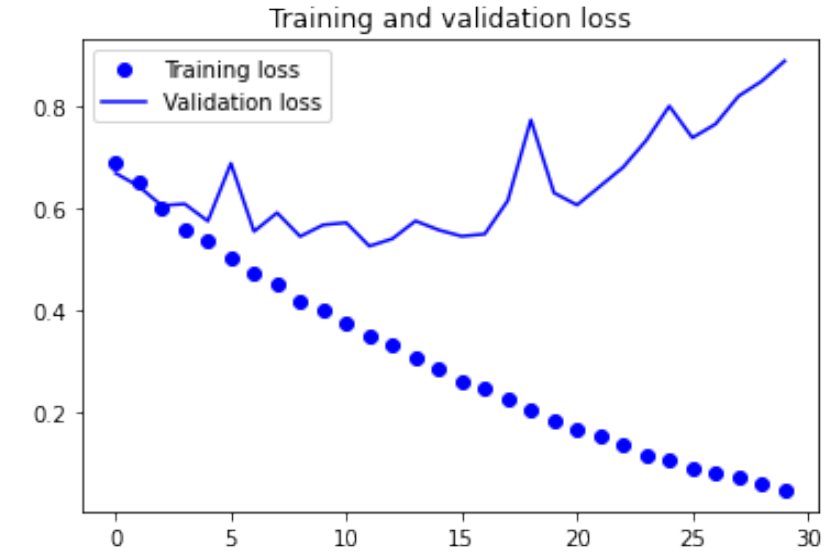
- 3. Colab HW 사양 확인 방법

```
from tensorflow.python.client import device_lib
device_lib.list_local_devices()

import platform
platform.platform()

!cat /etc/issue.net
!cat /proc/meminfo
!cat /proc/cpuinfo
!nvidia-smi
!python --version
!ls
```

고양이-개 분류하기 - 작은 컨브넷으로 예측하기

- MNIST 보다 이미지가 크고 복잡하기 때문에 좀 더 큰 모델 생성
  - 특성맵은 층이 깊어지면서 크기가 줄어듬(150 x 150, 처음 이미지 크기) → (7, 7)
  - 특성맵의 깊이(필터의 개수)는 층이 깊어지면서 많아짐(32개 → 128개)
- 이미지데이타 전처리
  - 1. 파일 읽어서 RGB 값으로 변환
  - 2. 보동소숫점 유형 텐서로 변환
  - 3. 픽셀값을 0 ~ 255 → 0 ~ 1 로 변환
- ImageDatagenerator() ← yield()
  - 위 전처리를 자동으로 수행
  - 데이터 공급을 무한반복
- fit_generator()
  - fit() 역할을 하면서 학습데이타로 generator 이용
- 과대적합모델임
  - 학습 데이타수가 비교적 적음
  - 훈련정확도 : 100% 도달, 검증정확도 : 70% ~ 72% 수준.

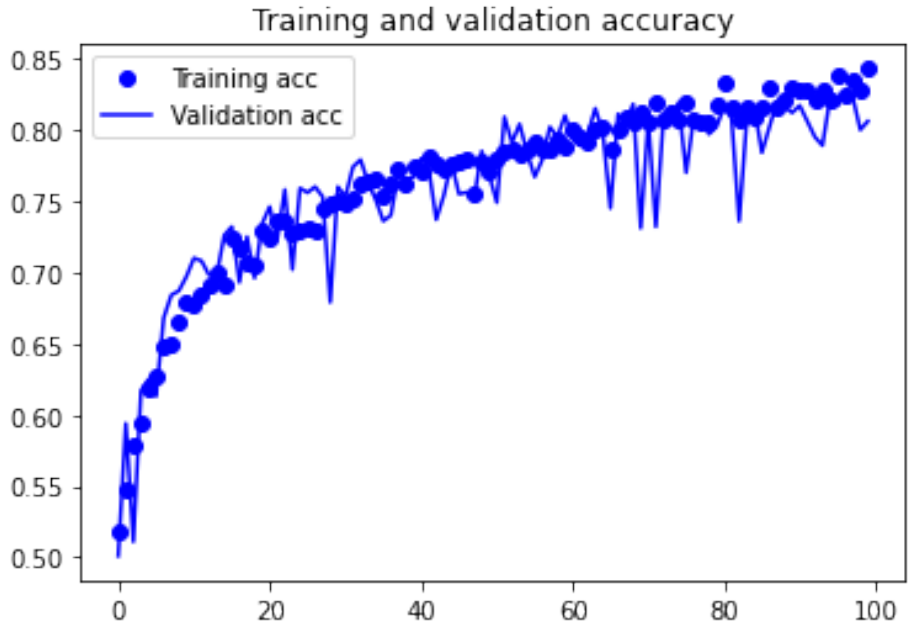




고양이-개 분류하기 – 데이터 증식 사용

- 과대적합을 피하기 위해 많은 데이터 필요함.
- 기존 훈련데이타를 이용하여 추가적인 학습데이타 생성
  - ImageDataGenerator() 이용
  - 위 전처리를 자동으로 수행
  - 데이터 공급을 무한반복
- 과대적합 줄이는 dropout 적용
- ImageDataGenerator()
  - rotation_range : 사진을 회전시킬 각도(0-180도).
  - width_shift_range / height_shift_range : 사진을 수평과 수직으로 랜덤하게 평행 이동.
  - shear_range : 전단 변환 적용할 각도.
  - zoom_range : 사진을 확대할 범위.
  - horizontal_flip : 이미지를 수평으로 뒤집음 (예, 풍경/인물 사진).
  - fill_mode : 회전이나 가로/세로 이동으로 인해 새롭게 생성할 픽셀을 채울 전략

```
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')
```



- 정확도 : 82%
- 데이터 증식, 드롭아웃으로 과대적합 억제.
- 훈련곡선과 검증곡선이 비슷.
- (모델파일 다운로드 → 시각화에서 사용함)

고양이-개 분류하기 - 사전훈련된 컨브넷 사용

- Pretrained network
  - 대규모 이미지 데이타셋에서 미리 훈련되어 저장된 네트워크
  - 사전훈련된 네트워크에 의해 학습된 특성의 계층구조는 일반적인 모델의 역할을 할 수 있음
  - 새로운 문제가 원래작업과 다른 클래스 분류이더라도 활용 가능.
  - ImageNet 데이타셋으로 훈련된 모델은 유사한 특징을 가지는 이미지의 분류에 사용가능.
  - 2014년 개발된 VGG16 이용

- 이용방법
  - 특성 추출
  - 미세조정

- VGG16(2014)
  - 간단하고 ImageNet 데이터셋에 널리 사용되는 컨브넷 구조.
  - 조금 오래되었고 최고 수준의 성능에는 못미치며 최근의 다른 모델보다는 조금 무거움.
  - 새로운 개념을 도입하지 않고 이해하기 쉬움.
- 이외 활용할 수 있는 모델로는 ResNet, Inception, Inception-ResNet, Xception, MobileNet 등이 있음.

고양이-개 분류하기 - 사전훈련된 컨브넷 사용

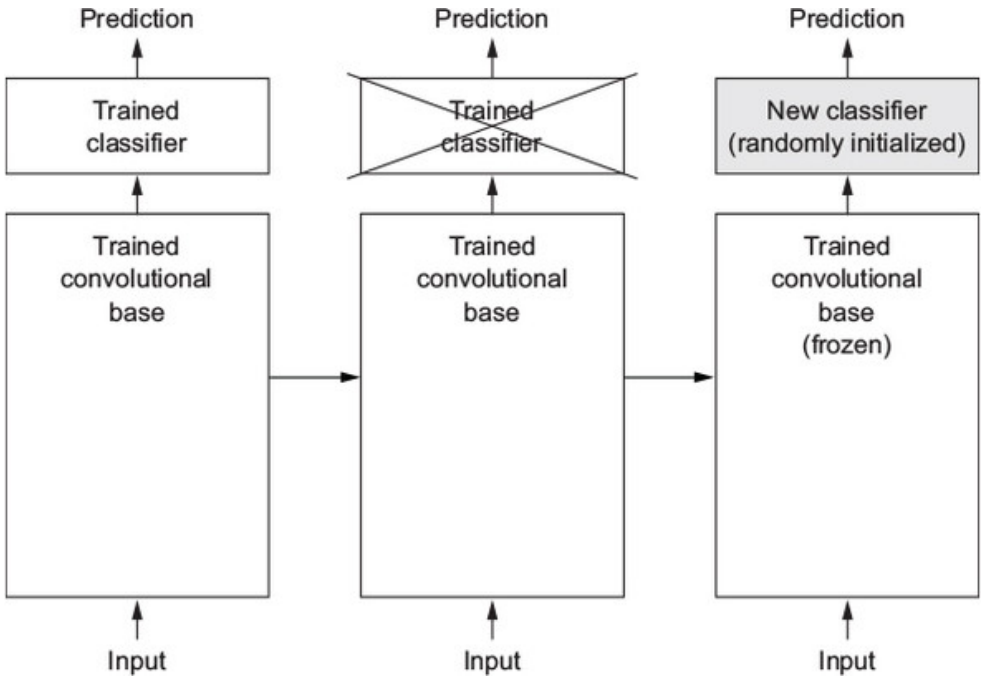
- VGG16의 기반층만 사용사용하여 목표데이타(개/고양이 데이터)를 통과시키고
- 그 출력으로 새로운 분류기(개/고양이 분류기)를 훈련

- 기존 합성곱층 :
  - 일반적인 이미지 특성을 학습하였기 때문에 유용함.
  - 하위층은 에지, 색깔, 질감 등 일반적이고 기본적인 특성을 학습함.
  - 상위층은 강아지의 눈, 고양이의 귀 처럼 추상적이고 종합적인 특성을 학습함.
- 기존 분류기 :
  - 이미지에 관한 정보는 없고 (기존 1000 클래스 문제의) 출력을 분류하는데 필요한 정보만을 담고 있음.

- 방법 1(빠른 특성추출 방법)
  - 개/고양이 데이터를 기존 합성곱층을 통과시키고(conv_base.predict())
  - 그 결과를 기존 1000 개 분류기 대신 새로운 분류기 학습모델의 입력으로 사용

- 방법 2(conv_base 포함 새로운 모델 학습)
  - - 기존 합성곱층을 포함하여 새로운 모델을 구성

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



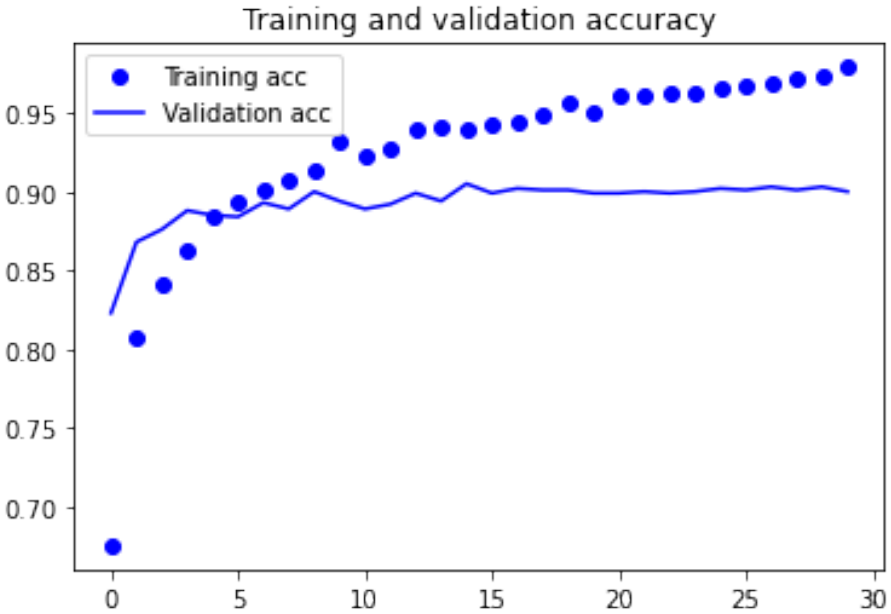
사전훈련된 컨브넷 사용 - 빠른 특성추출 방법

- conv_base 이용
  - 기존 합성곱층을 개/고양이 데이터의 출력용도로 사용(conv_base.predict())
  - 기존 1000 클래스 학습모델에서 개/고양이 정보에만 집중하는 효과.
  - 그 결과를 (기존 1000 개 분류기 대신) 새로운 분류기(개/고양이만 분류하는) 학습모델의 입력으로 사용.

- 새 분류모델 구현
  - 개/고양이 학습이 추가된 모델을 입력으로 새로운 분류 모델 구현

```
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'))
```

- 결과
  - 90% 검증정확도
  - 빠른 epoch에서 과대적합 보임(적은 데이타셋 때문)
- 합성곱 연산을 하지 않으므로 빠른 대신 이미지 증식을 활용할 수 없음.



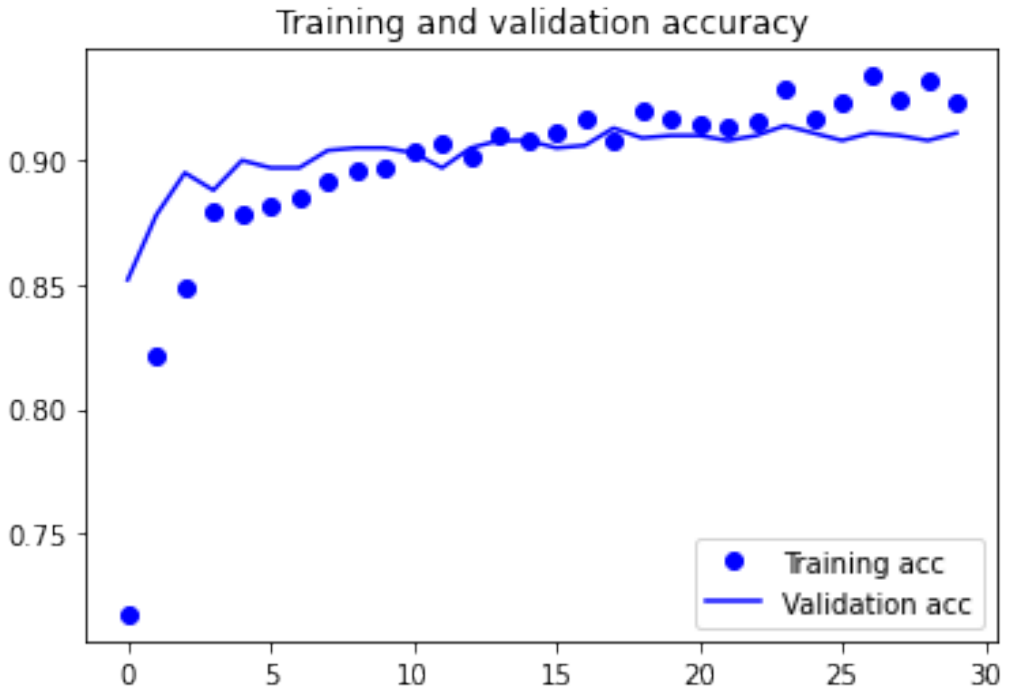
사전훈련된 컨브넷 사용 - conv_base 포함 새로운 모델 사용

- 기존 conv_base를 포함하고 개/고양이 분류에 촛점을 맞춘 새로운 모델 구축

```
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'))
```

- 입력에 이미지증식기능을 적용할 수 있음.
- 합성곱기반층은 기존 가중치를 사용하므로 학습되지 않도록 해야 함.
- 입력데이타에 대하여 합성곱 연산부터 분류까지 학습.
- 합성곱 연산을 하기 때문에 이미지 증식을 활용할 수 있음.

- 결과
  - ➢ 과대적합에 강함

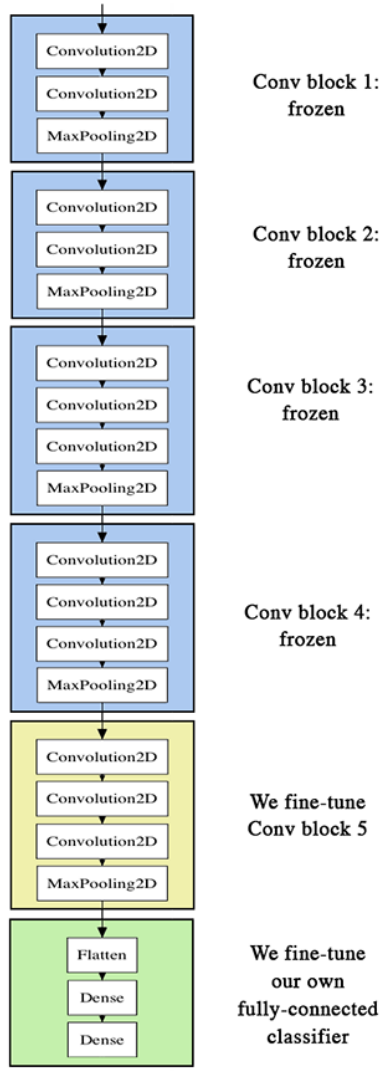


사전훈련된 컨브넷 사용 - 미세조정

- 훈련된 재사용 모델의 상위층 몇 개를 동결해제
- 모델에 새로 추가한 층(완전연결 분류기)를 포함하여 훈련
- 주어진 문제에 조금 더 집중하여 재사용 모델의 표현 일부를 조정하는 기법
  - 최상위 분류기가 훈련된 후에 합성곱 기반층의 상위층이 미세조정되어야 함.
  - 최상위 분류기가 훈련되지 않으면 합성곱 기반층의 상위층에 전파되는 오차신호가 매우 커 미세조정 층의 최적화가 어렵게 됨.
  - 학습률은 보통보다 낮게 설정(이미 학습된 네트워크를 미세조정하므로)

- 더 많은 층을 미세조정하지 않는 이유
  - 하위 층들은 좀 더 일반적인 특징들을 학습(기본 구성요소들)한 반면 상위층들은 목표에 좀 더 가까운 특징(1000 개의 클래스)들을 학습.
  - 따라서 개/고양이 분류에 집중하려면 마지막 분류 층의 학습을 (1000 개 클래스 학습에서) 개/고양이 만 분류하도록 학습하는 것이 유리함.
  - 데이타셋의 양이 적으므로 많은 층을 학습하면(용량과다) 쉽게 과대적합이 됨.

- 과정
  - 사전훈련된 기반모델 위에 새로운 네트워크(여기서는 분류기)를 추가
  - 기반네트워크를 동결
  - 새로 추가한 네트워크를 훈련
  - 기반 네트워크 상위층 동결해제
  - 전체 모델 다시 훈련



사전훈련된 컨브넷 사용 - 미세조정

- (실습)
  - 마지막 3개의 합성곱층(Conv2D)을 미세조정
  - 학습률은 평소보다 낮게(lr=1e-5)
  - 변동이 크면 학습된 표현을 흔들 수 있음.

- 결과
  - 손실은 epoch 당 증가, 정확도는 변동 없음.
  - 손실값은 epoch 당 개별 손실의 평균임.
  - 손실값은 대리손실함수(surrogate loss function), 여기서는 cross-entropy
  - 정답에 해당하는 손실과 정확하게 비례하지 않음.

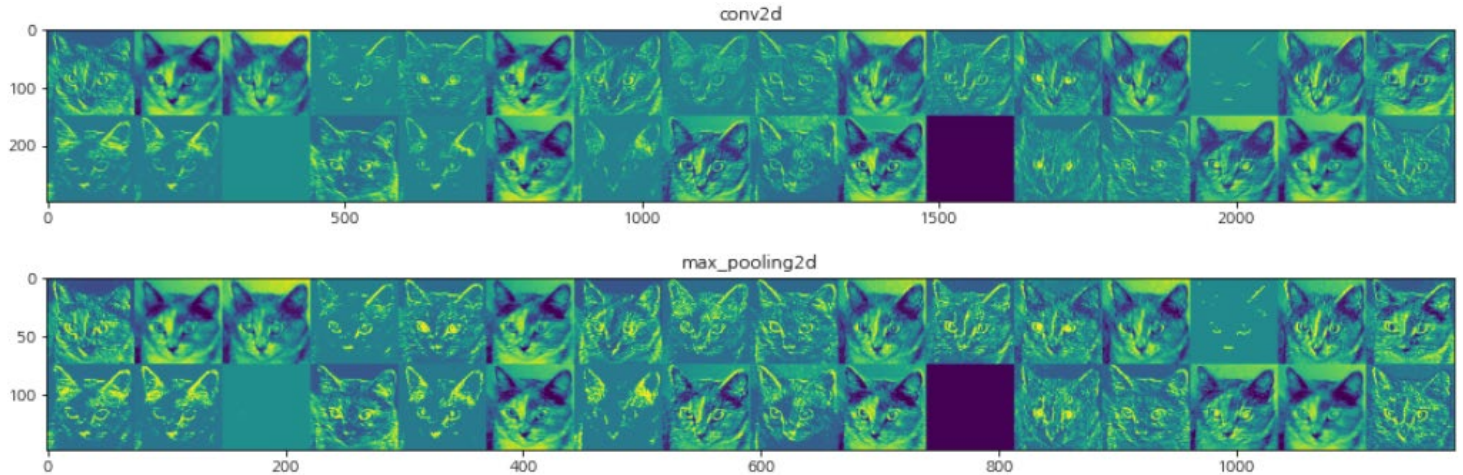
학습과정 시각화

- 컨브넷은 시각적인 개념을 학습한 것이기 때문에 시각화에 유리함
- 중간활성화층 시각화
  - 필터를 통과한 결과 이미지
  - 계속되는 컨브넷층이 입력을 어떻게 변형시키는지 이해하고 개별적인 컨브넷 필터의 의미를 파악하는데 도움.
- 컨브넷 필터 시각화
  - 필터가 찾으려는 시각적 패턴을 확인할 수 있음.

- 클래스 활성화에 대한 히트맵
  - 이미지의 어느 부분이 클래스 판단에 기여했는지 표시.

학습과정 시각화 - 중간활성화층 시각화

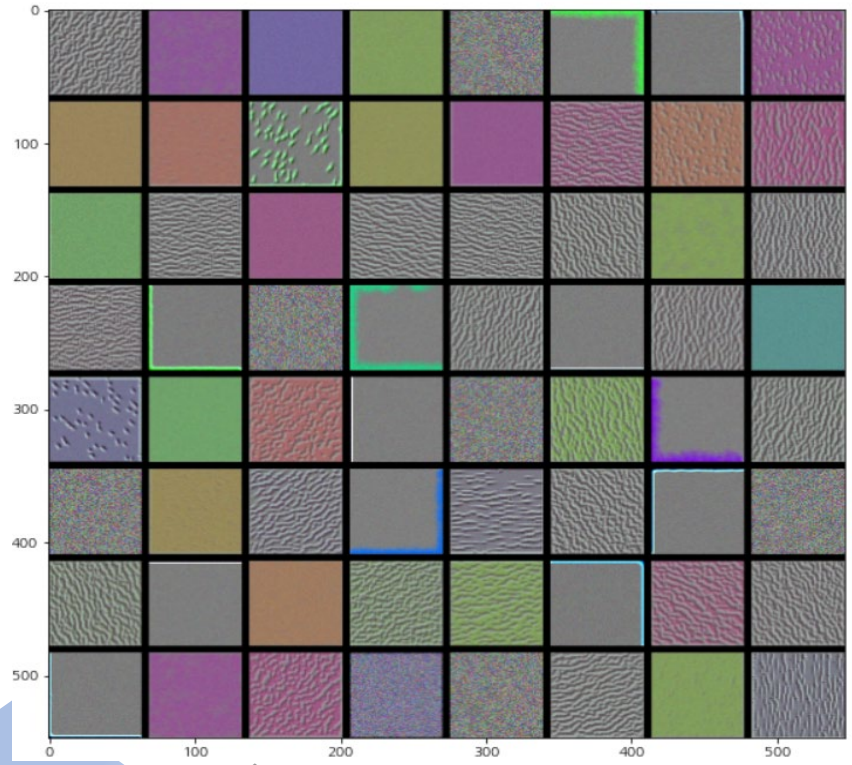
- 필터를 통과한 결과 이미지
- 합성곱층, 풀링 층이 출력하는 특성 이미지
- 학습된 필터들이 입력을 어떻게 분해하는지 보여줌
- 첫번째 층
  - 여러 종류의 에지감지기. 처음 사진의 거의 모든 정보가 유지됨.
- 상위층
  - 점점 더 추상적이 됨. 시각적으로 이해하기 어려워짐.
  - 고양이의 귀, 눈 같은 고수준(큰 범위) 개념을 인코딩함.
  - 이미지 자체의 정보는 줄어들고 클래스에 관한 정보가 커짐.
- 비어있는 활성층 증가
  - 필터가 학습한 패턴이 입력에 나타나지 않은 경우
  - 작은 부분(초기층)은 대부분의 입력이미지에 존재, 큰 범위의 부분일수록 입력이미지에 없는 경우도 생김.



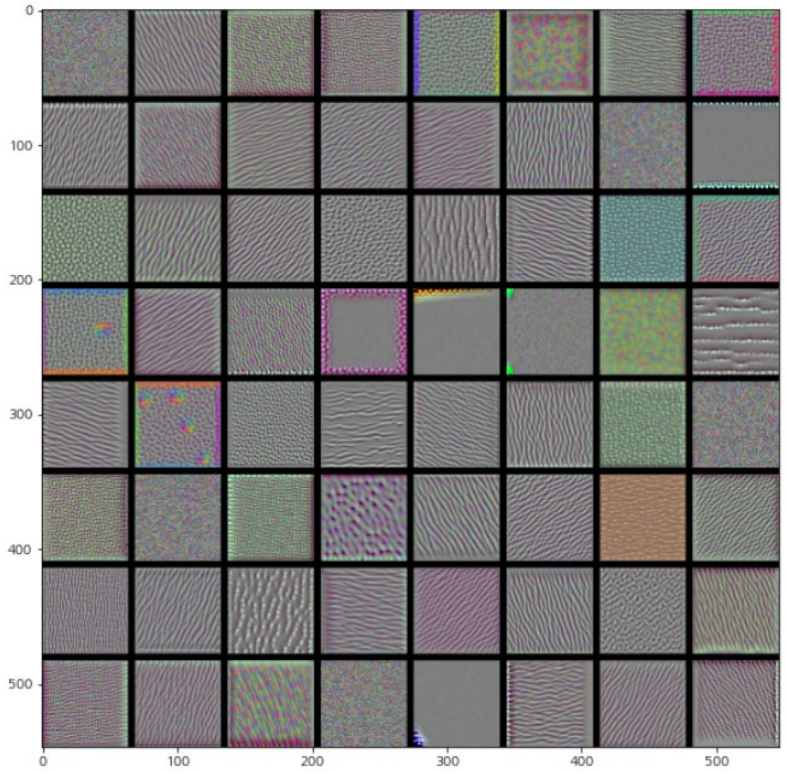
학습과정 시각화 – 필터 시각화

- 컨브넷 필터 시각화
  - 필터가 찾으려는 시각적인 패턴과 개념이 무었인지 확인.
  - 하위층은 단순한 패턴, 상위층으로 갈수록 복잡한 패턴을 학습함

VGG16의 'block1_conv1' 필터

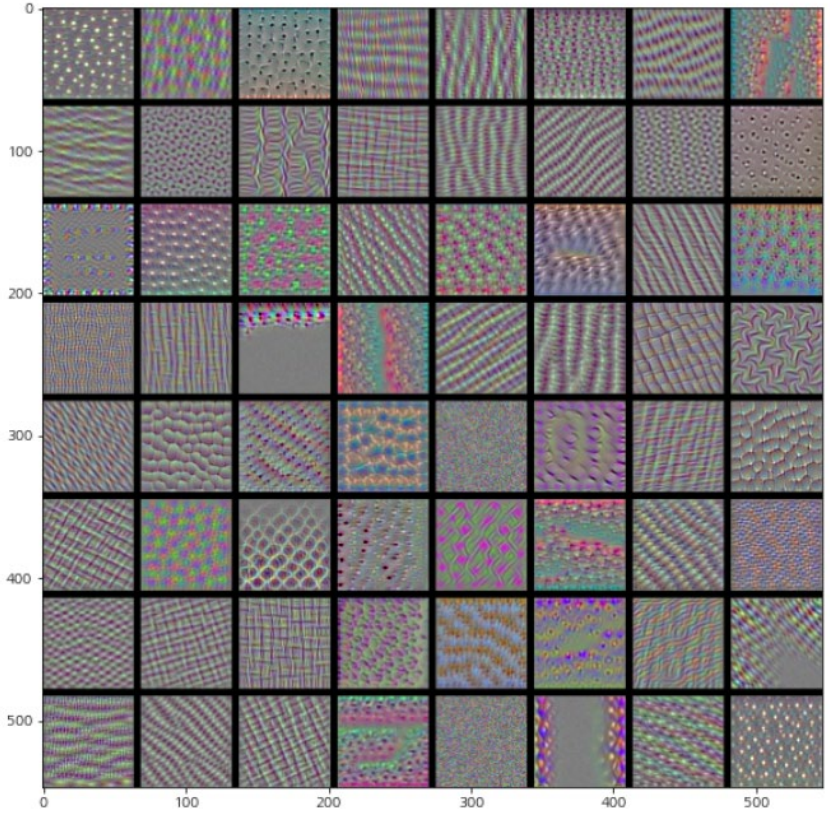


VGG16의 'block2_conv1' 필터

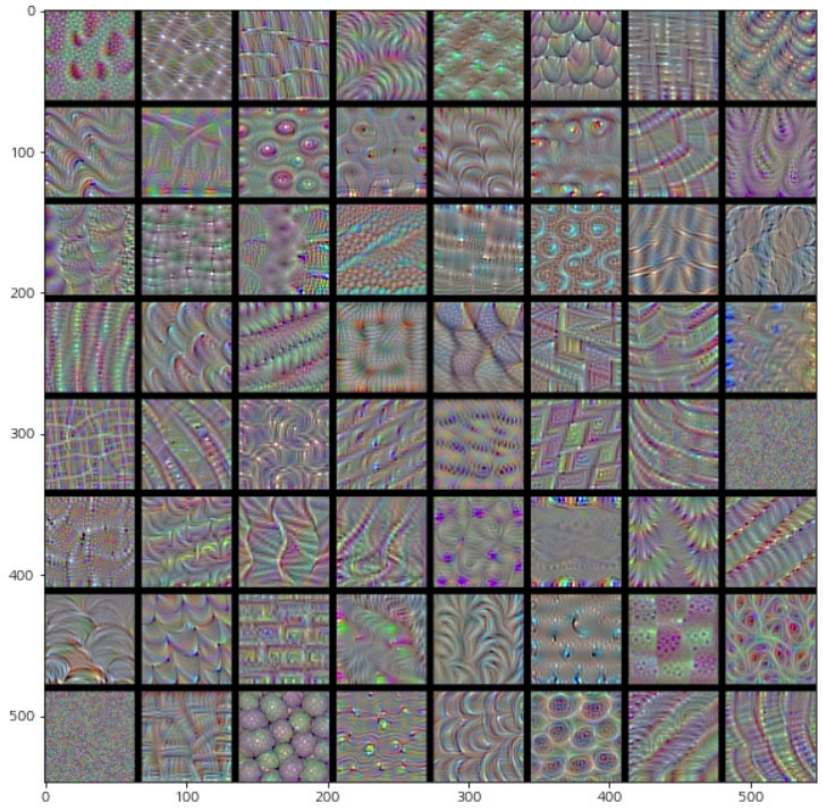


학습과정 시각화

VGG16의 'block3_conv1' 필터



VGG16의 'block4_conv1' 필터



실습 : 7.4-visualize-convnets

학습과정 시각화

- 클래스 활성화에 대한 히트맵을 이미지에 시각화
  - 이미지의 어느부분이 주어진 클래스에 속하는데 기여했는지 이해.
  - 이미지에서 객체의 위치를 추정하는데 도움.

아프리카 코끼리(92.5%)
코끼리(tusker)(7%)
인도코끼리(0.4%)
귀의 차이를 판별근거로 삼음.

인도코끼리

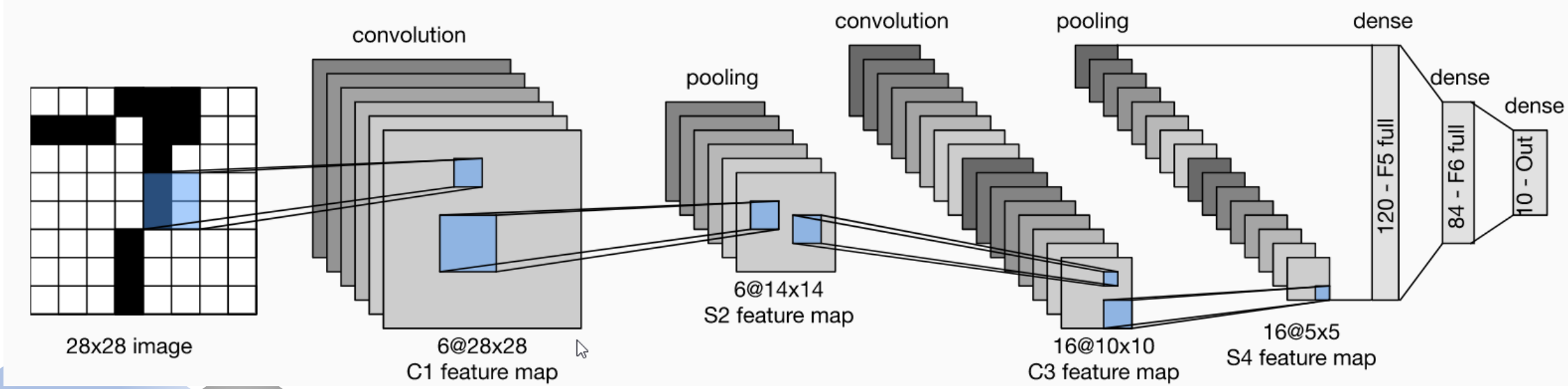
tusker(어금니)

아프리카 코끼리

LeNet – LeCun et al., 1998

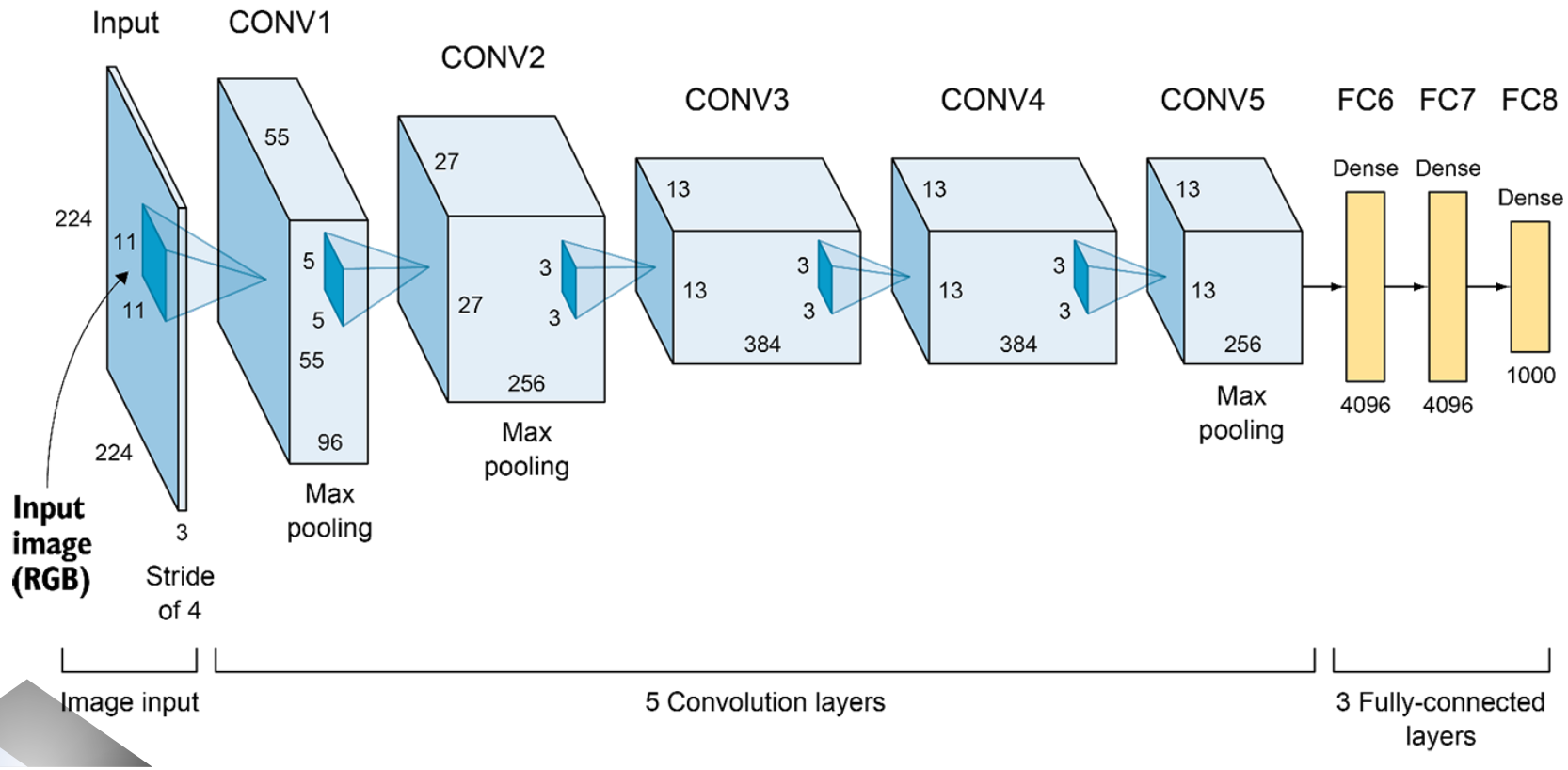
- 28 x 28 MNIST 숫자 데이타셋 대상
- 파라미터 : 60,000개
- 연산량 : 34만개
- convolution layer : 2개
  - 5 x 5 kernel, padding = 2, sigmoid activation
- sub-sampling layer : 2개
- fully-connected layer : 3개

- 모델 특징
  - 네트워크 깊어질수록 높이와 폭이 줄어듬, 채널수는 증가



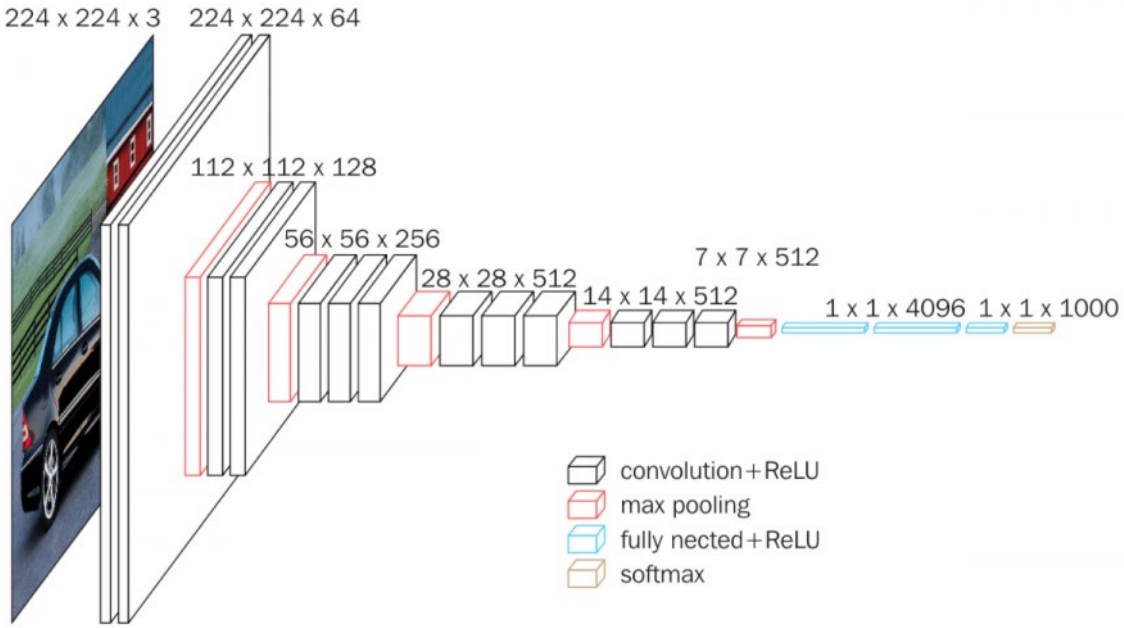
AlexNet – Krizhevsky et al., 2012

- 파라미터 : 약 6200만개
- convolution layer : 5개(pooling layer : 3개)
- fully-connected layer : 3개
- 모델 특징
  - 학습 최적화 위해 ReLU 활성화 함수, 2개의 GPU 사용
  - 과적합 방지 위해 Data Augmentation, Dropout 사용



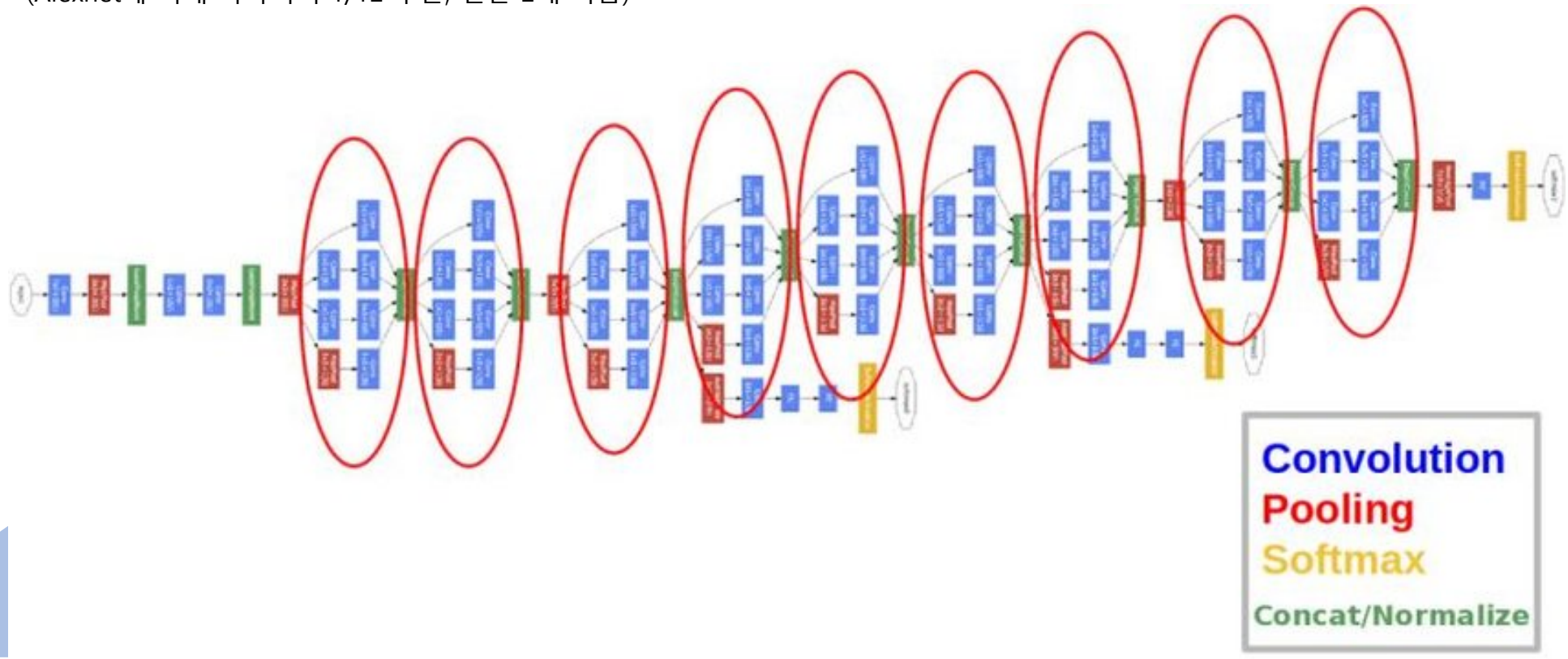
VGG16 - Karen Simonyan & Andrew Zisserman. 2014

- convolution layer : 13개(pooling layer : 5개)
- fully-connected layer : 3개
- 총 layer : 16개
- 파라미터 : 1억 3800만개
- 모델 특징
  - 더 깊은 네트워크를 형성하기 위해, 단순하고 작은 필터 적용.
  - -> 모든 convolution layer에 "3x3 (stride 1, pad 1)" 필터 적용
  - 필터 갯수는 2배로 계속 규칙적 상승



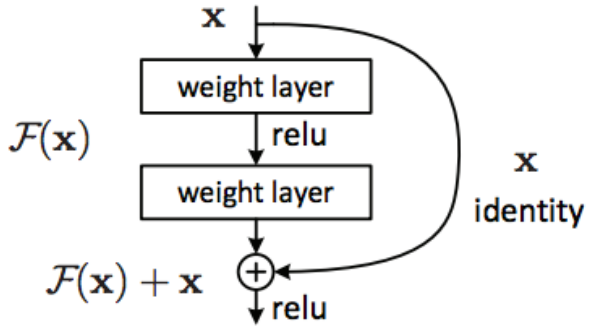
GoogleNet – team at Google, 2014

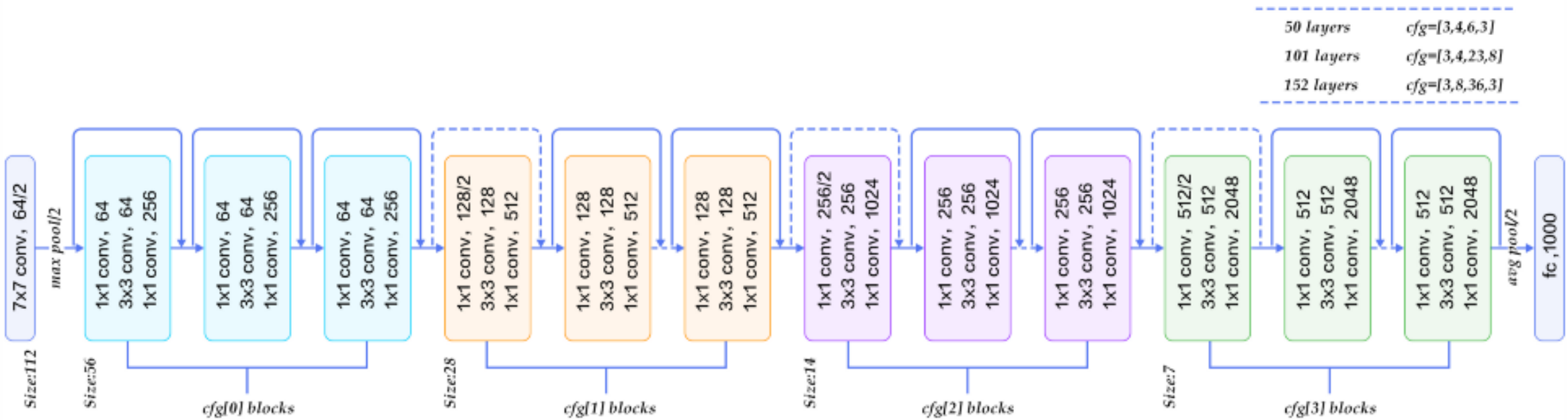
- 파라미터 : 약 500만개
- 총 layer : 22개
- 모델 특징
  - 더 깊고 넓은 네트워크(파라미터 증가)를 형성하면 overfiting, 연산량(연산비용) 증가 등의 문제 발생.
  - => 이를 막기 위해 Inception(1x1사용) 개념을 적용하여 연산비용 감소
  - (Alexnet에 비해 파라미터 1/12 수준, 연산 2배 빠름)



ResNet - Kaiming He et al., 2015

- Skip connection 적용
  - 출력에 입력을 더함.
  - 더하기 연산이므로 연산량 증가 없음.
  - 역전파시 추가된 입력의 미분값 1 이 추가되므로 기울기 소실문제를 방지할 수 있음.
- 152 layer 구현
  - Cfg0 block :3번, cfg1 block : 8번, cfg2 block : 36번, cfg3 block : 3 번 반복 구조 + avg pool + fcn





모델 설계 및 활용 방안

- 입력 이미지의 크기
  - 일반적으로 층이 깊어짐에 따라 ¼ 로 줄어들므로(2x2 window, stride 2 MaxPooling) 2의 배수인 이미지가 바람직.
  - 224 x 224 : ImageNet 샘플 크기이므로 ImageNet에서 학습된 모델 사용시 최적.
  - 또는 384 x 384, 512 x 512
- Conv layer
  - 일반적으로 3 x 3 필터, padding = 1, stride = 1 사용.
  - 5 x 5, 7 x 7 사용도 가능함.
- Pooling layer
  - 2 x 2 window, stride = 2 사용
- 모델설계 중요 결정사항
  - Conv layer, ReLU, Pooling layer 의 설계 및 반복
  - Fully Connected Layer의 설계 및 반복
  - 각 층별 중요 인자 결정
- Don’t be a hero.
  - 기존에 잘 학습된 모델을 기반으로 목적에 맞도록 튜닝하여 사용하는 것이 효율적임.