

CNN(합성곱) 구현

이미지 분류 경진대회 ILSVRC

- **ILSVRC(ImageNet Large Scale Visual Recognition Challenge)**
 - 이미지 인식(image recognition) 경진대회
 - **ImageNet DB 중 일부를 사용해 이미지 범주를 분류**
 - <http://image-net.org/>
 - 대용량의 이미지셋을 주고 이미지 분류 알고리즘의 성능을 평가하는 대회
 - 2010년에 시작되어, 2017년 종료
- **2010년, 2011년에 우승을 차지한 알고리즘**
 - 얇은 구조(shallow architecture)
- **2012년 CNN 기반 딥러닝 알고리즘 AlexNet이 우승**
 - 깊은 구조(deep architecture), 약 26%였던 인식 오류률을 16%까지 개선
- **2017년 종료**
 - 2015년
 - **사람의 정확도라고 알려진 5%를 추월**
 - 2017년
 - **SENet의 경우 2.3%로 사람의 인식 에러률의 절반도 안됨**

ILSVRC 에러율

우승 알고리즘의 분류 에러율(%)

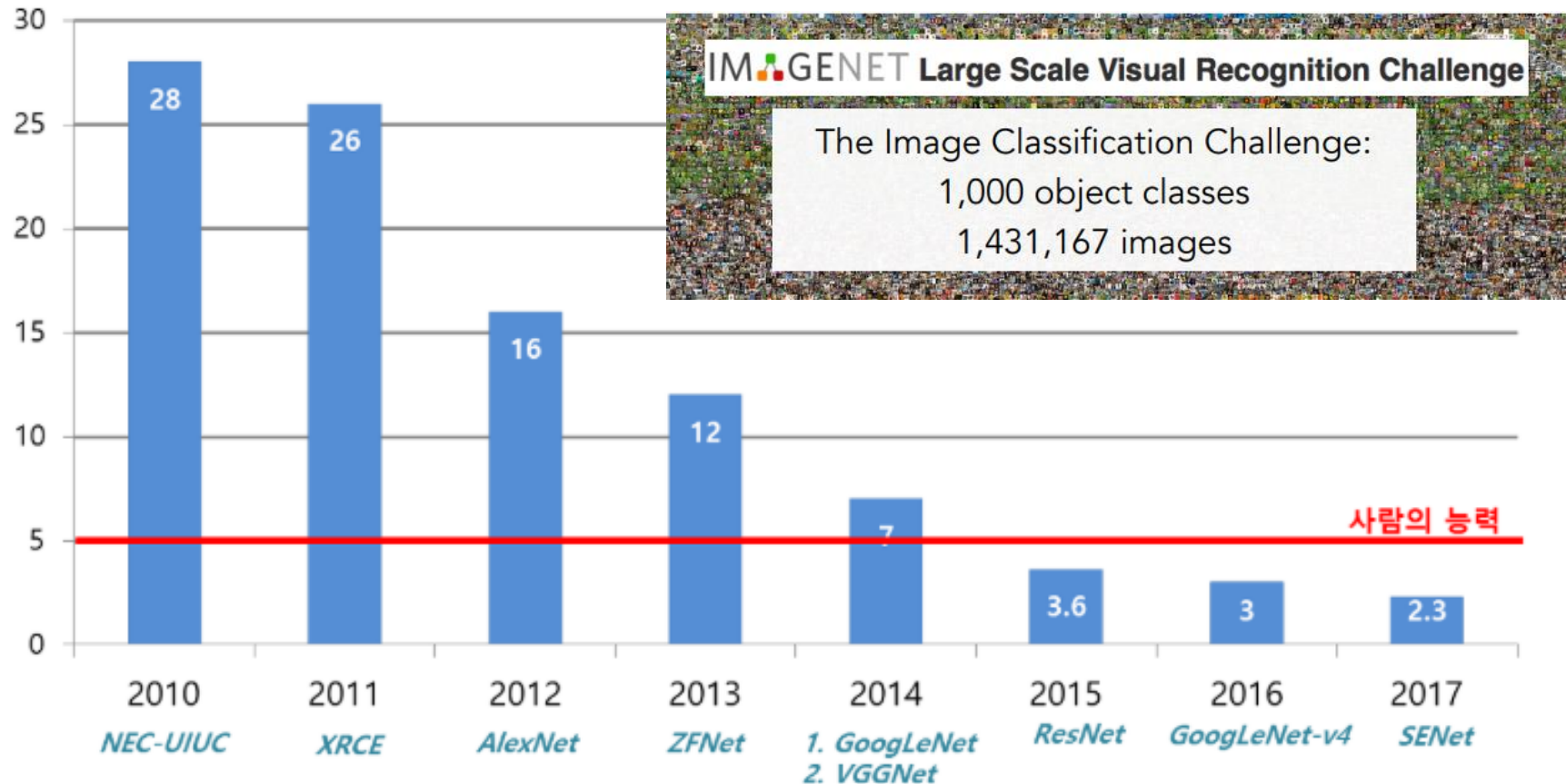
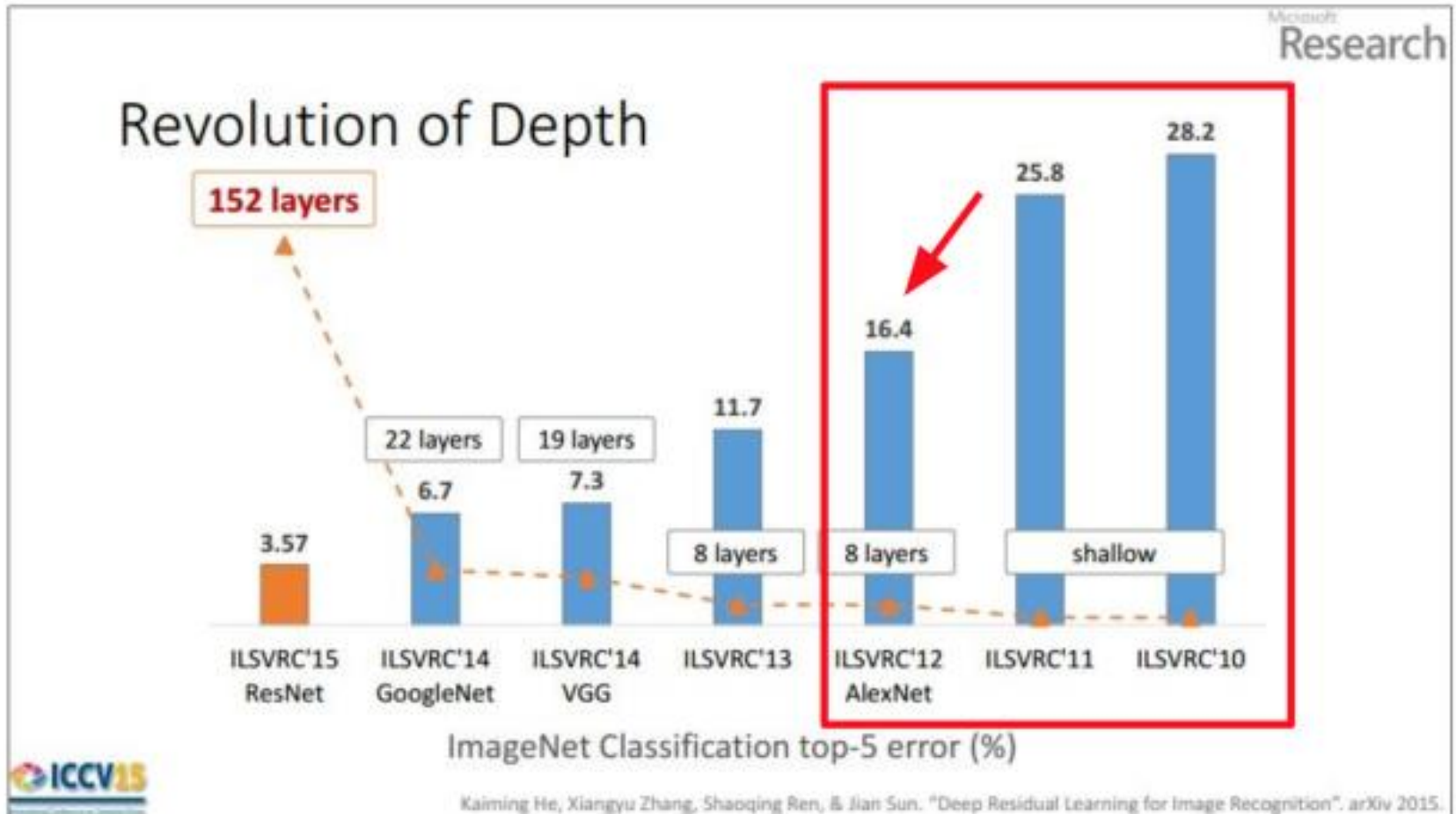


그림1. ILSVRC 대회 역대 우승 알고리즘들과 인식 에러율.

딥러닝 깊이의 증가

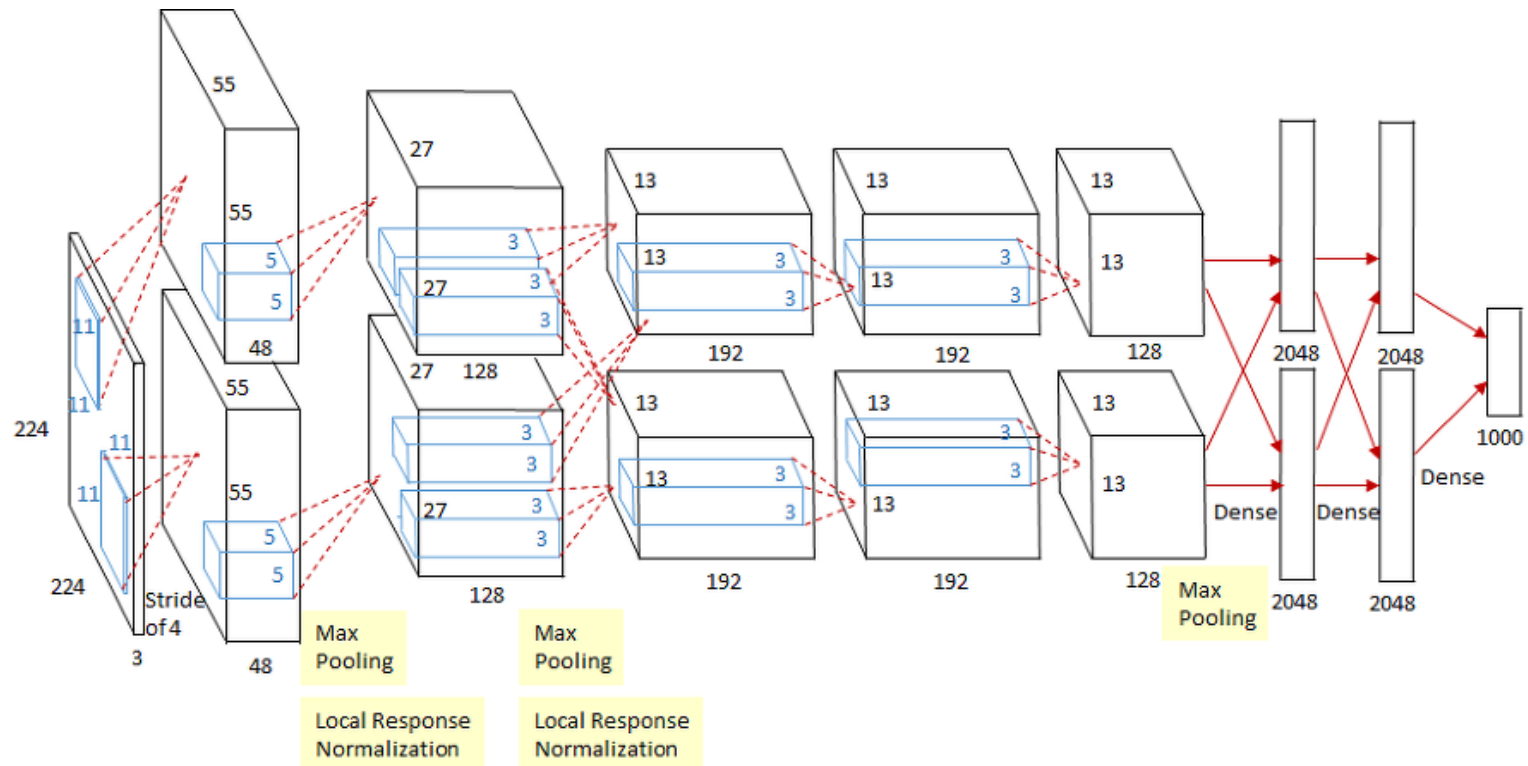


(slide from Kaiming He's recent presentation)

AlexNet 구조

AlexNet의 기본구조

- LeNet-5와 크게 다르지 않으며
- 2개의 GPU로 병렬연산을 수행하기 위해서 병렬적인 구조로 설계



이미지 특징 추출 알고리즘

- **Feature extraction**

- 필터(커널)를 사용해 이미지의 주요 특징을 추출

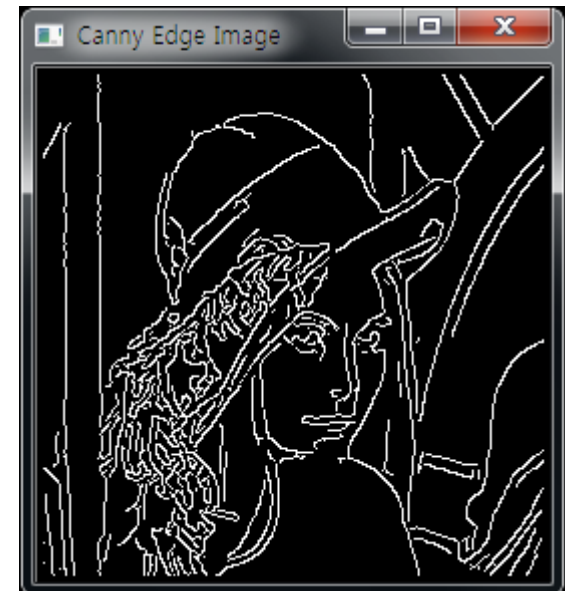
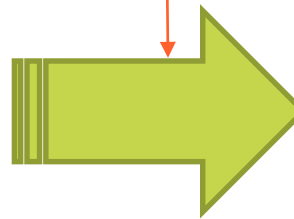
- **외곽선 검출 알고리즘**

0	0	0
-1	1	0
0	0	0

(a) 수직 에지 검출 마스크

0	-1	0
0	1	0
0	0	0

(b) 수평 에지 검출 마스크



CNN(Convolutional Neural Network) 개요

• CNN

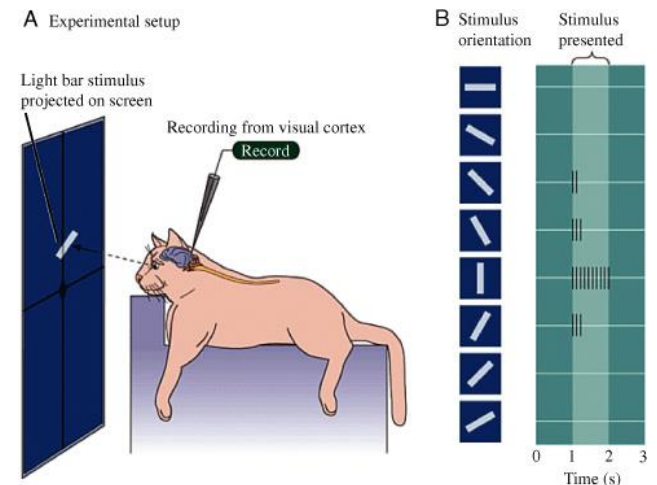
- 이미지의 공간 정보를 유지한 상태로 학습이 가능한 모델
 - 특징 추출기 + 분류기 구성

• 일반 Dense() 층과 비교

- Fully Connected Neural Network와 비교하여 다음과 같은 차별성
 - 각 레이어의 입출력 데이터의 형상 유지
 - 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식
 - 복수의 필터로 이미지의 특징 추출 및 학습
 - 추출한 이미지의 특징을 모으고 강화하는 Pooling 레이어
 - 일반 신경망과 비교하여 학습 패러미터가 매우 적음
 - 필터를 공유 패러미터로 사용하기 때문

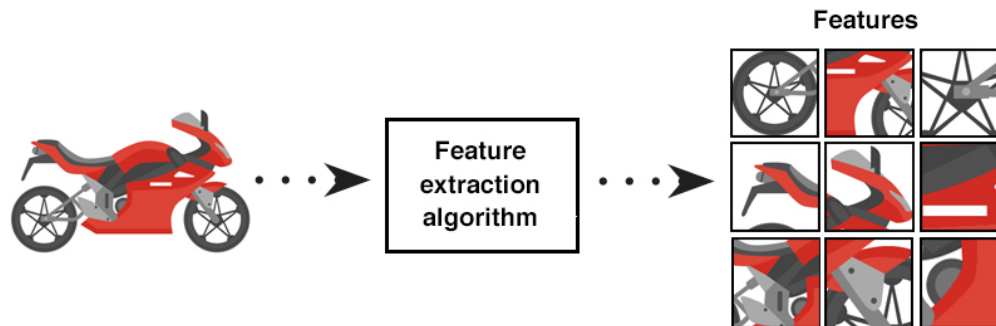
• LeCun 1998년

- LeNet이라는 Network를 1998년에 제안
- 이것이 최초의 CNN



CNN은 컨볼루션 층과 풀링 층, 분류기로 구성

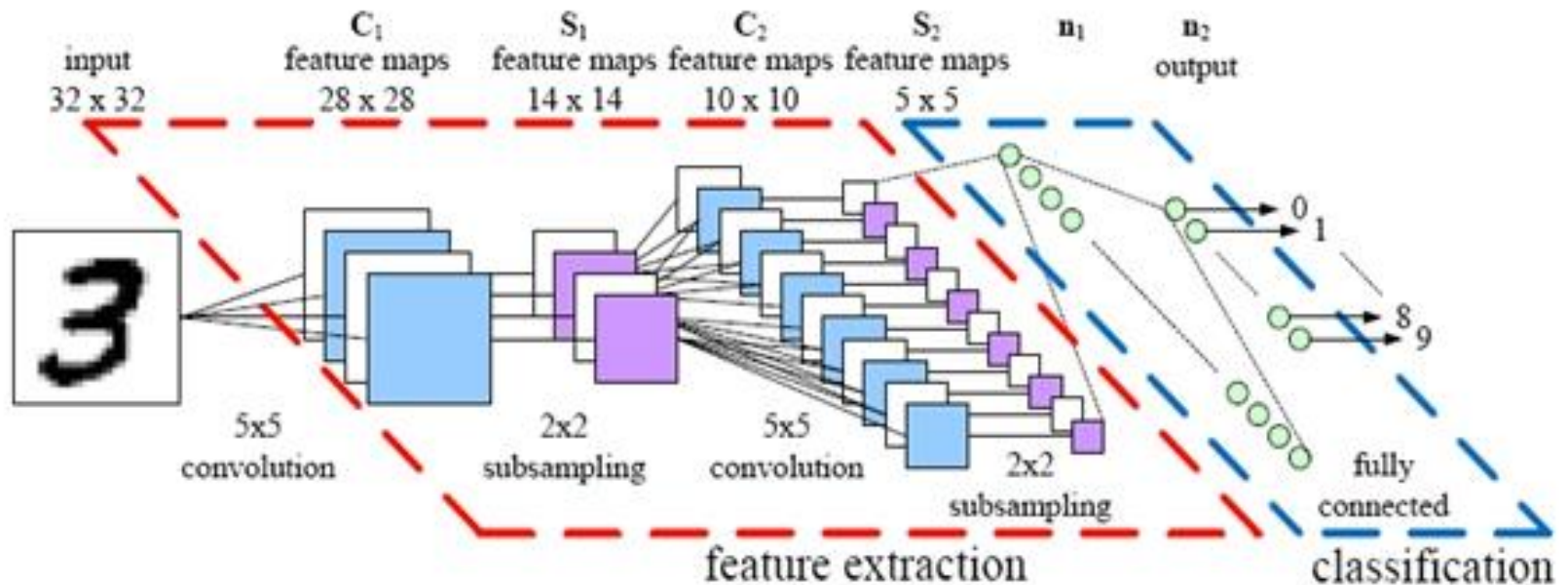
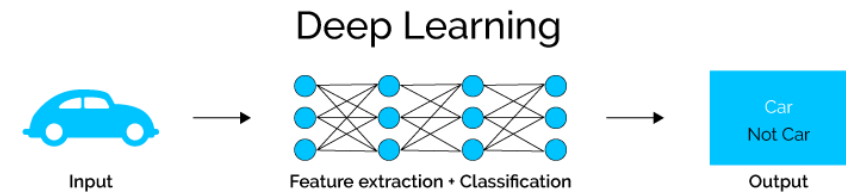
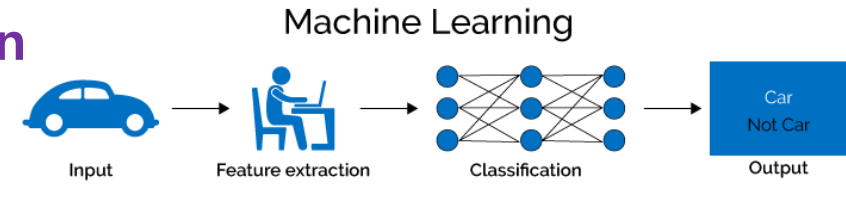
- **Feature Extractor(특징 추출기) + Classifier(분류기)**
 - 이미지의 특징을 추출하는 부분과 클래스를 분류하는 부분으로 나눔
- **특징 추출기:**
 - 자동으로 특징을 추출하는 필터를 생성하는 것이 목적
 - Convolution Layer와 Pooling Layer를 여러 겹 쌓는 형태로 구성
 - Convolution Layer: 입력 데이터에 필터를 적용 후 활성화 함수를 반영하는 필수 요소
 - Pooling Layer: 선택적인 레이어
 - Subsampling 이라고도 부름



- **분류기**
 - CNN 마지막 부분에는 이미지 분류를 위한 Fully Connected 레이어가 추가
 - 처음은 Flatten 레이어
 - 이미지의 특징을 추출하는 부분과 이미지를 분류하는 부분 사이에 이미지 형태의 데이터를 배열 형태로 변환

CNN 구조

- Feature Extraction + Classification



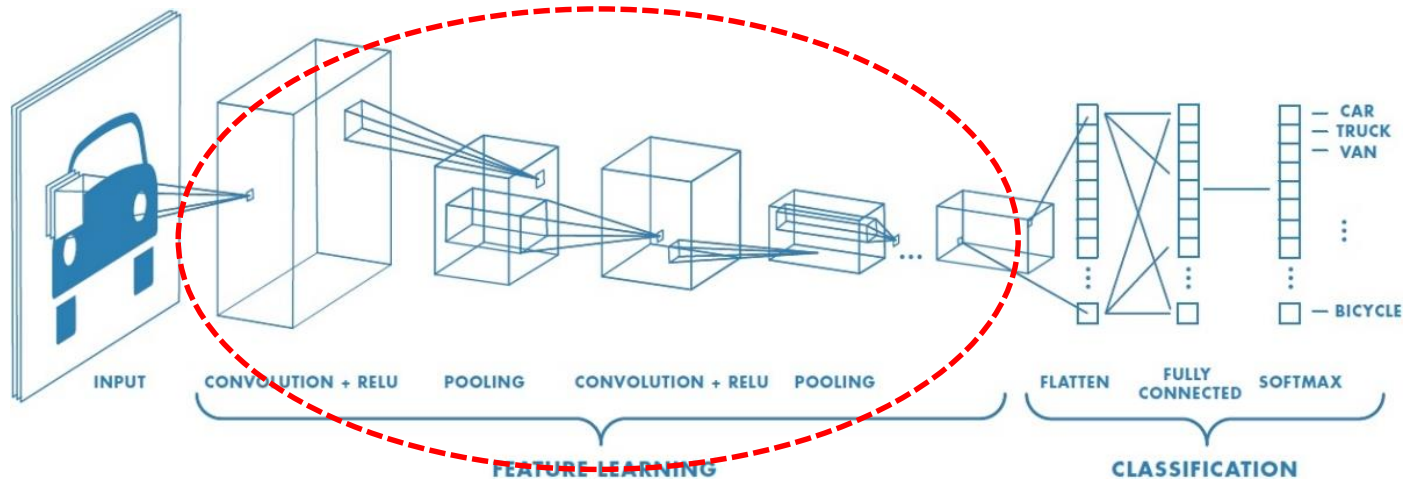
CNN의 컨볼루션

• 컨볼루션 층

- 각 이미지에서 특정 특징을 활성화하는 컨볼루션 필터 집합에 입력 이미지를 통과
- ReLU(Rectified Linear Unit)
 - 음수 값을 0에 매핑하고 양수 값을 유지하여 더 빠르고 효과적인 학습을 가능

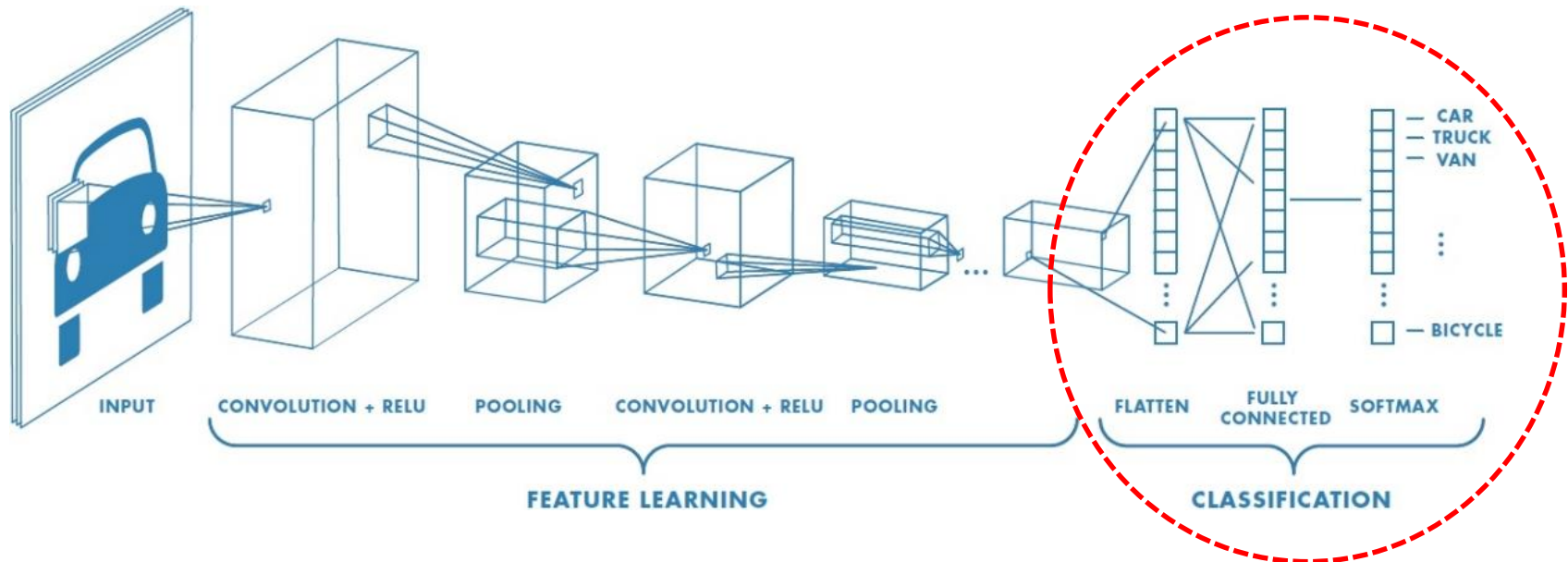
• 풀링(서브샘플링) 층

- 비선형 다운샘플링을 수행
 - 네트워크에서 학습해야 하는 매개 변수 수를 줄여서 출력을 간소화
 - 차원을 축소해 연산량을 감소
 - 이미지의 강한 특징만을 추출하는 특징 선별 효과가 있음



CNN의 분류

- 여러 계층에서 특징을 학습한 다음 분류 단계로 이동
- K 차원의 벡터를 출력하는 완전 연결 계층
 - K는 네트워크가 예측할 수 있는 클래스의 수
 - 이 벡터에는 분류되는 이미지의 각 클래스에 대한 확률
 - 마지막 계층에서는 softmax와 같은 분류 계층을 사용하여 분류 출력을 제공



Convolution 합성곱

- 필터를 통해 합성곱의 결과인 피쳐 맵을 획득

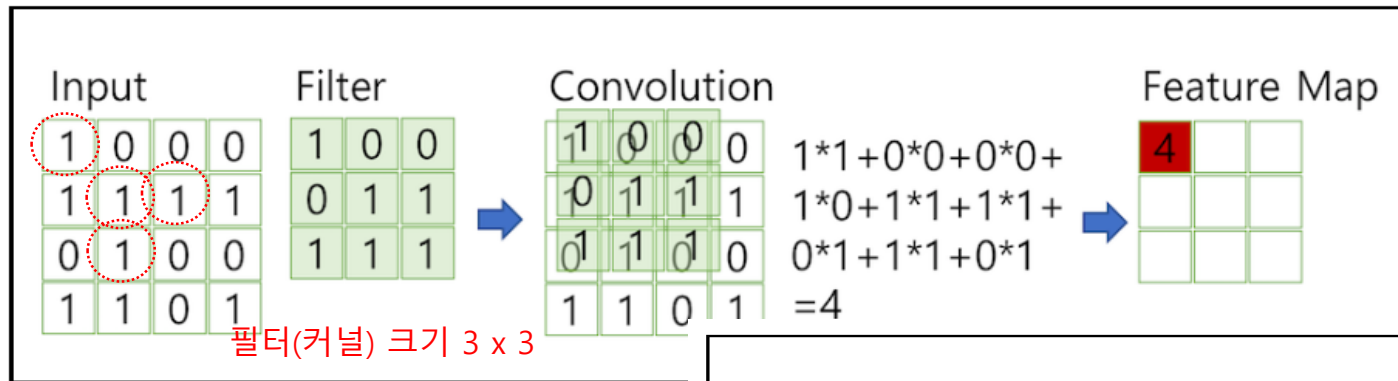


그림3: 합성곱 계산 절차

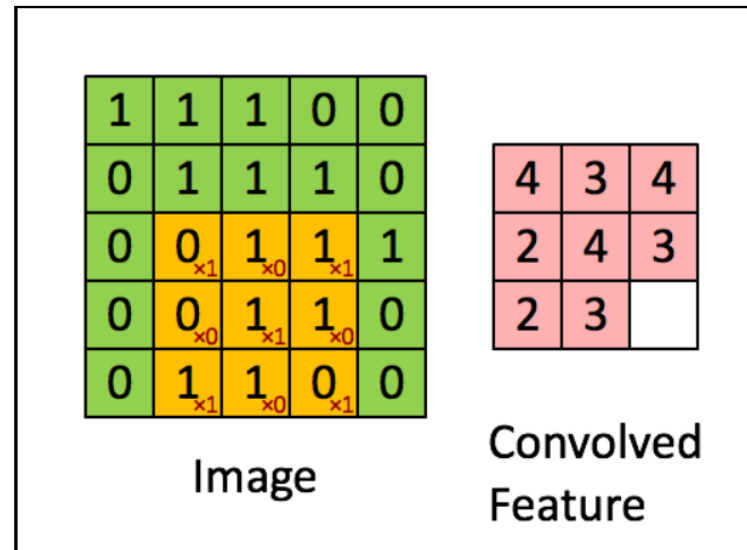


그림1: 합성곱 처리 절차, 출처:

http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

stride

- 보폭 stride가 1로 필터를 입력 데이터에 순회하는 예제
 - strid가 2로 설정되면 필터는 2칸씩 이동하면서 합성곱을 계산

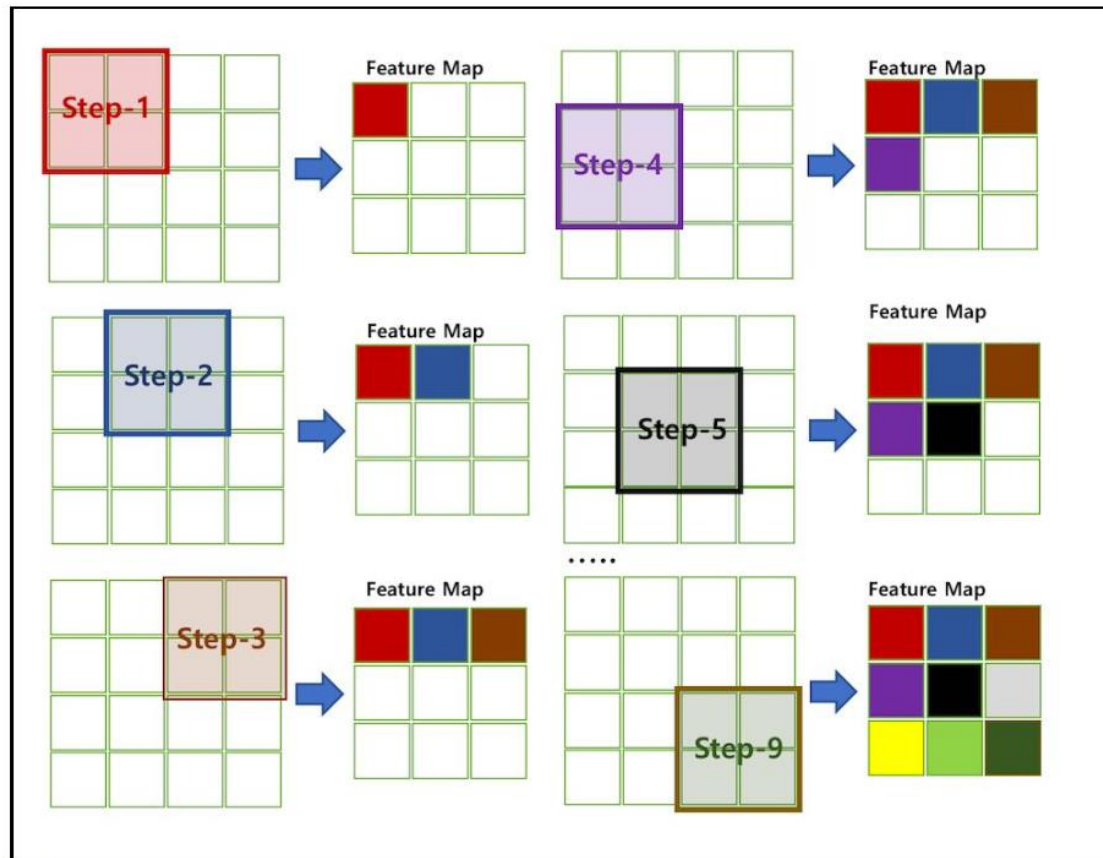


그림 4: Feature Map 과정

칼라인 경우, 채널이 3개

• 입력 데이터가 여러 채널을 갖을 경우

- 필터는 각 채널을 순회하며 합성곱을 계산한 후, 채널별 피쳐 맵을 만듦
- 그리고 각 채널의 피쳐 맵을 합산하여 최종 피쳐 맵으로 반환
- 입력 데이터는 채널 수와 상관없이 필터 별로 1개의 피쳐 맵이 생성

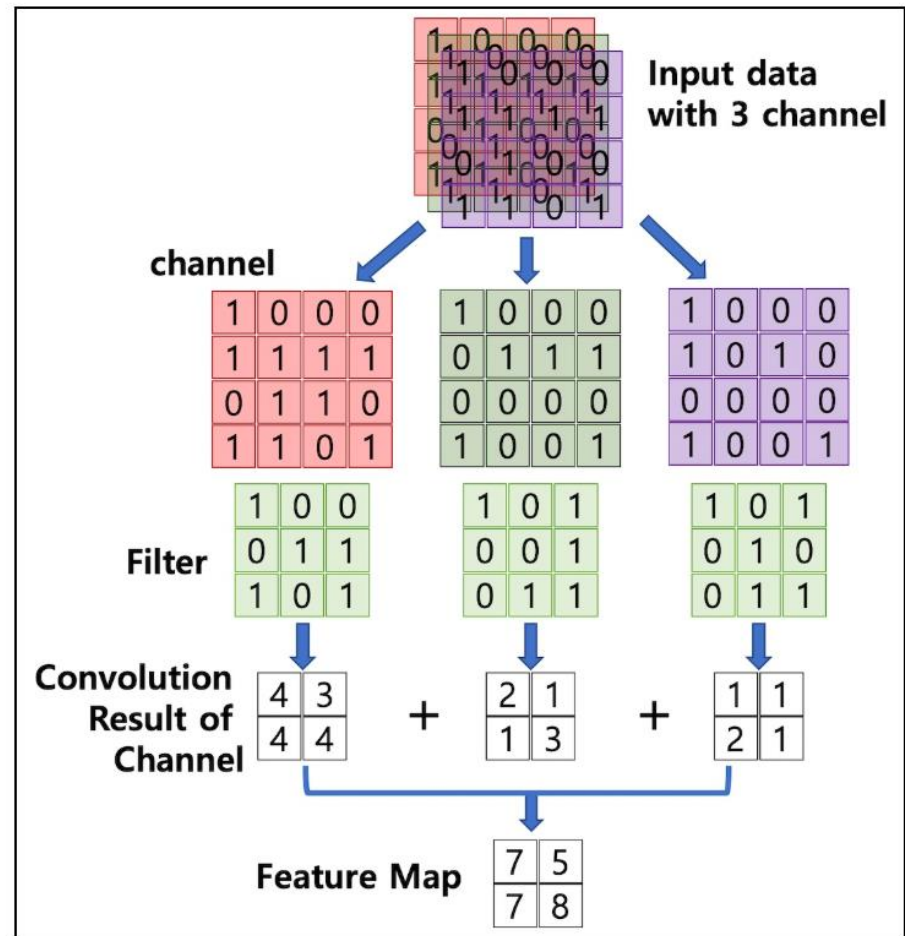
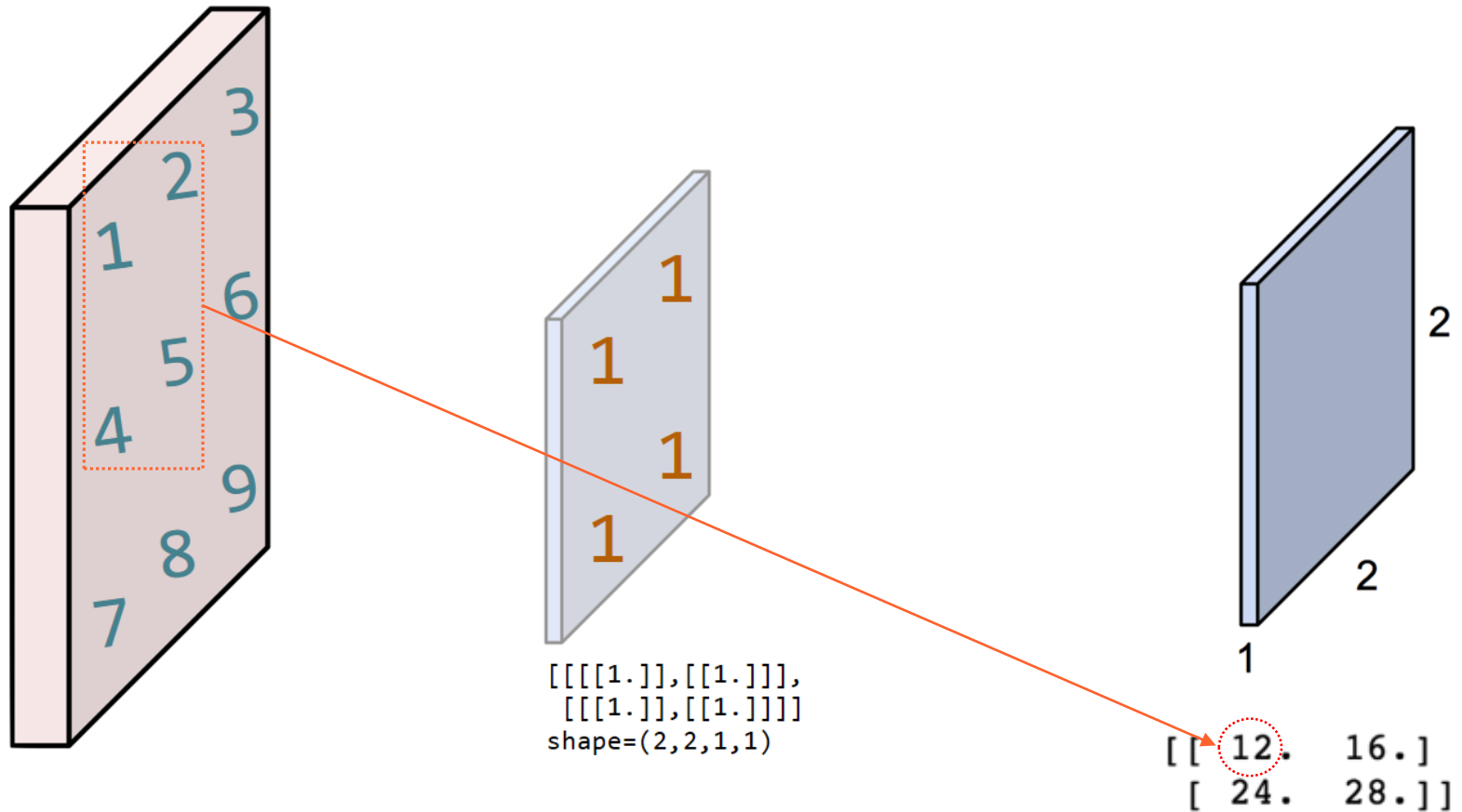


그림 5. 멀티 채널 입력 데이터에 필터를 적용한 합성곱 계산 절차

Simple convolution layer

- Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1



패딩(Padding)

- **합성곱의 결과인 특징 맵**
 - Filter와 Stride에 작용으로 Feature Map 크기는 입력 데이터보다 작음
- **패딩**
 - 입력 데이터의 외각에 지정된 픽셀 만큼 특정 값으로 채워 넣는 것을 의미
 - Convolution 레이어의 출력 데이터가 줄어드는 것을 방지하는 방법
 - 보통 패딩 값으로 0으로 채워 넣음
- **필터의 사이즈가 k**
 - 사방으로 k/2 만큼의 패딩
 - $K=3$
 - $3/2 = 1$

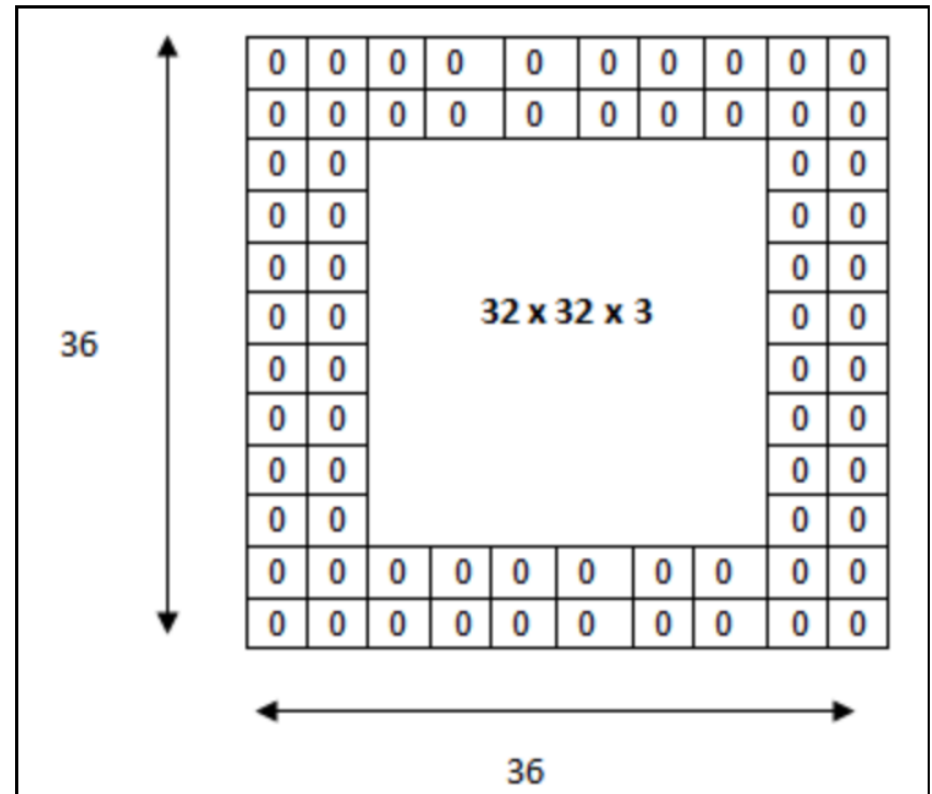


그림 6: padding 예제: 2pixel 추가

Pooling 레이어

- 컨볼루션의 결과를 줄이거나 특정 데이터를 강조

- 출력 데이터를 입력으로 받아서 출력 데이터(Activation Map)의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용
- 일반적으로 Pooling 크기와 Stride를 같은 크기로 설정하여 모든 원소가 한 번씩 처리 되도록 설정

- 풀링의 종류

- Max Pooling
 - 정사각 행렬의 특정 영역 안에 값의 최댓값
- Average Pooling, Min Pooling

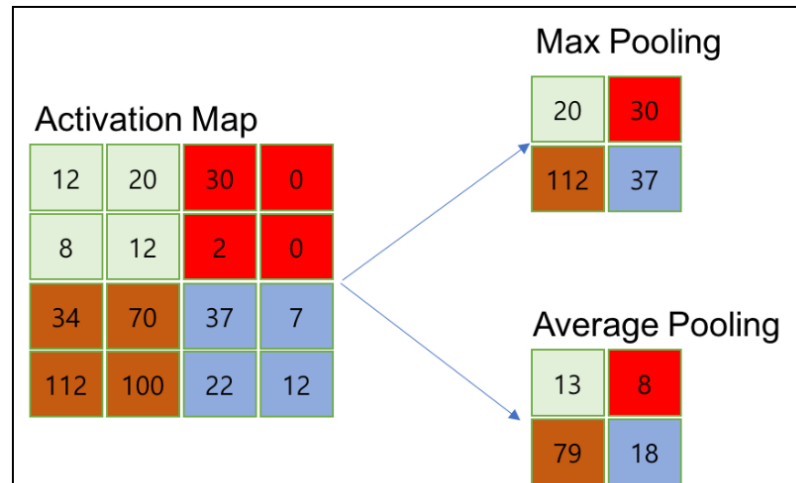
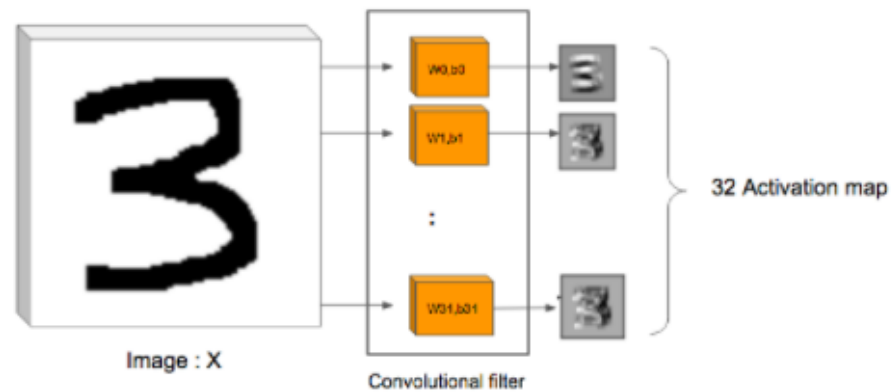
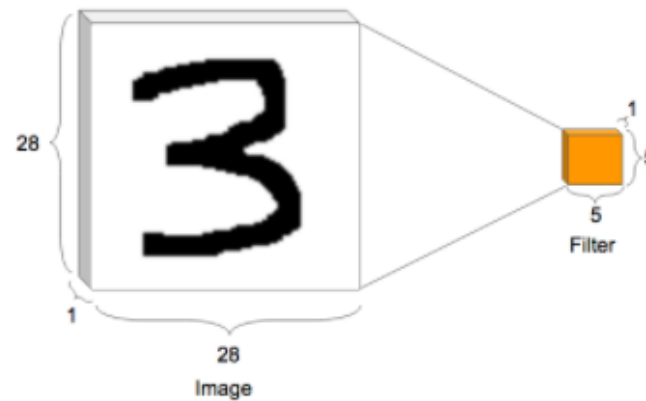
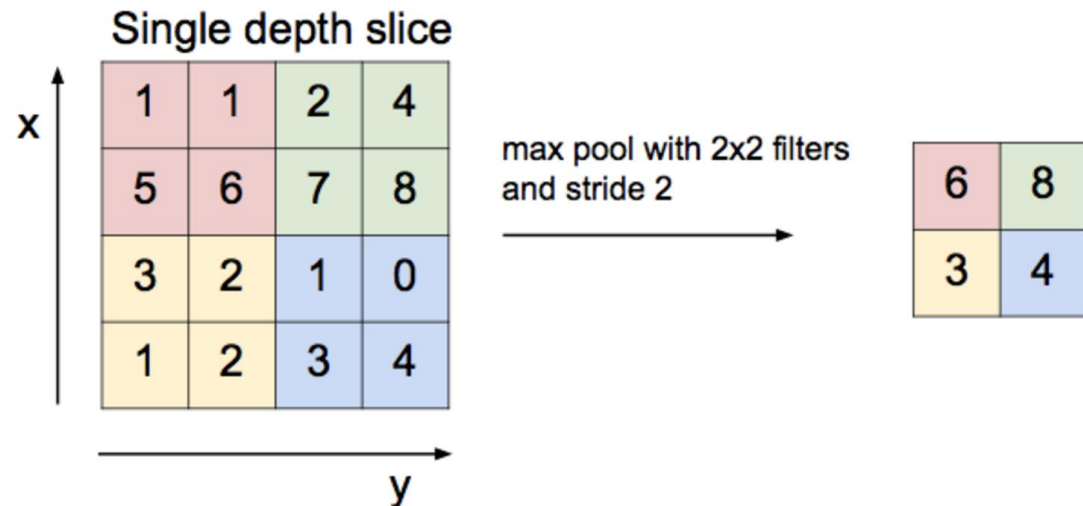
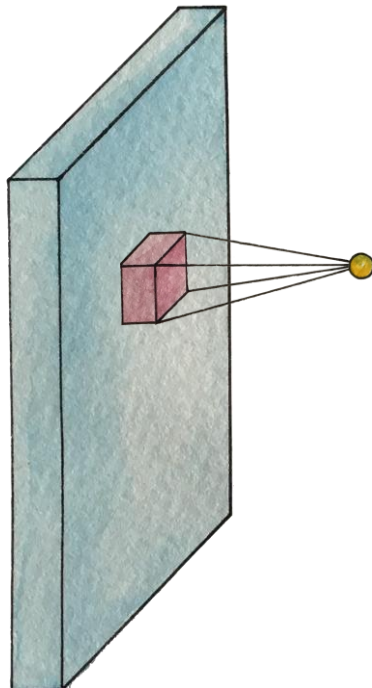


그림 7: Pooling 예제: Max Pooling, Average Pooling

하나의 이미지에 커널은 여러 개 구성 가능



Convolution layer and max pooling



유명 강의

- 스탠포드 대학의 CNN 강좌
 - CS231n 강의노트 Convolutional Neural Networks
 - <http://aikorea.org/cs231n/convolutional-networks/>
 - 강의 PPT
 - <http://cs231n.stanford.edu/slides/2020/>

컨볼루션 데모

컨볼루션 레이어

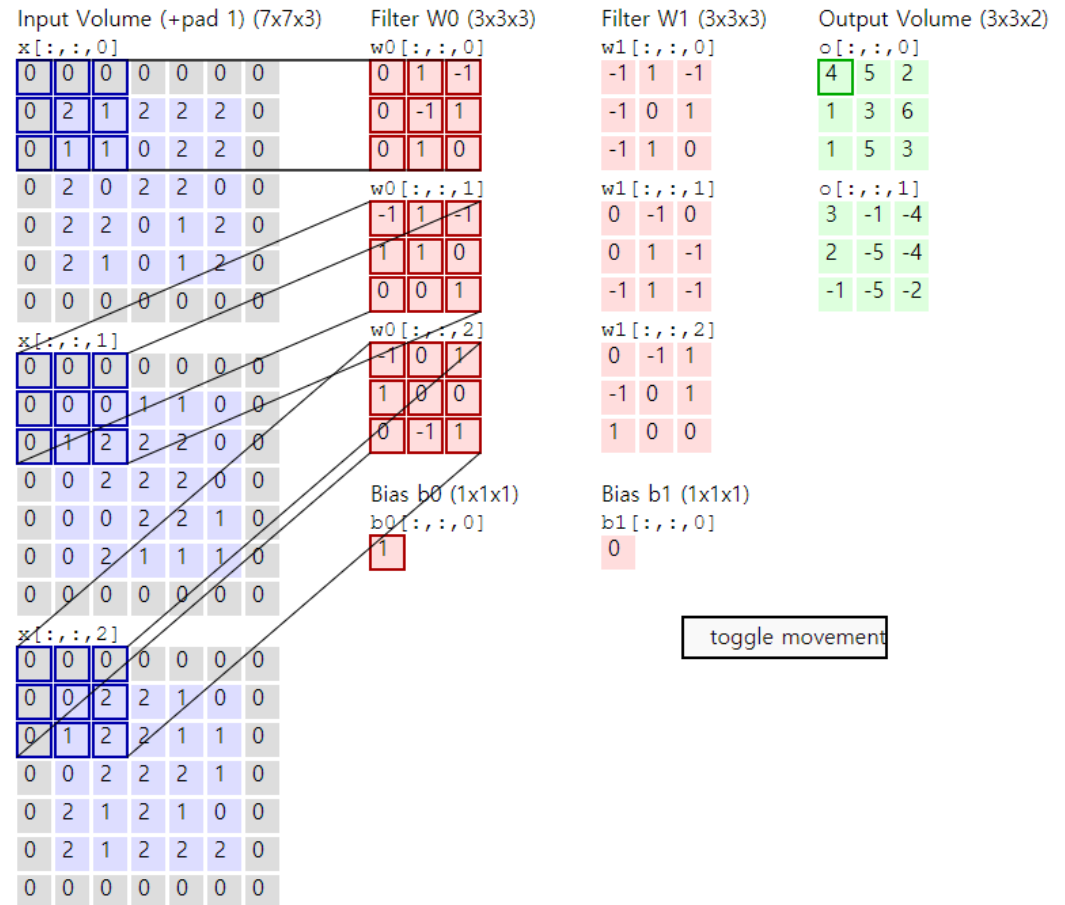
- 행렬(볼륨)의 구성
 - 입력 볼륨(파란색)
 - 가중치 볼륨(빨간색)
 - 출력 볼륨(녹색)

입력

- W1=5, H1=5
- D1=3(채널 수)
 - 제로 패딩 P=1 이 적용되어 입력 볼륨의 가장자리가 모두 0으로 추가

가중치(필터, 커널)

- 컨볼루션 레이어의 파라미터
- 3 * 3 크기의 필터 수 2개
 - 편향도 2개
 - stride 2마다 적용
 - K=2, F=3, S=2, P=1



컨볼루션 데모

• 입력

- W1=5, H1=5
- D1=3(채널 수)
 - 제로 패딩 P=1 이 적용되어 입력 볼륨의 가장자리가 모두 0으로 추가

• 가중치(필터, 커널)

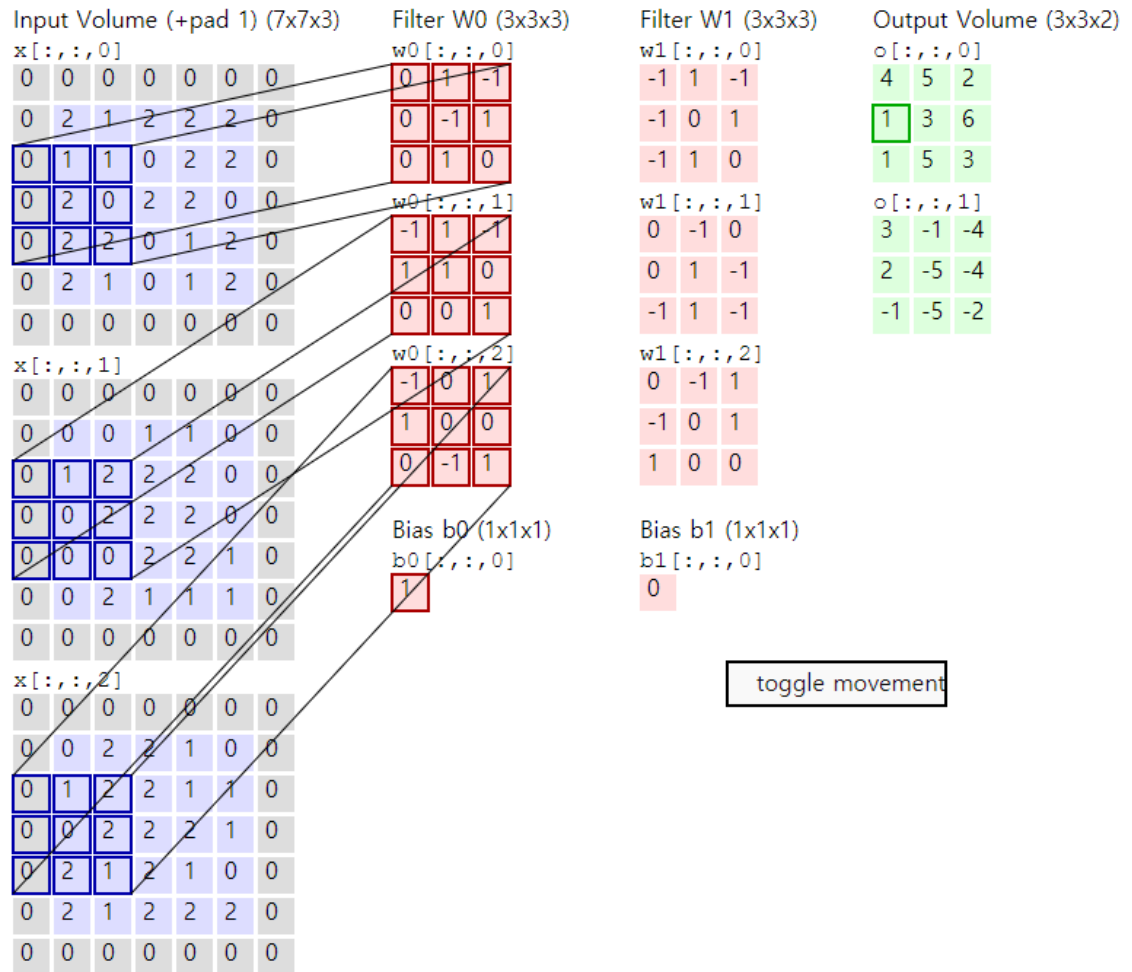
- 컨볼루션 레이어의 파라미터
- 3 * 3 크기의 필터 수 2개
 - 편향도 2개
 - stride 2마다 적용
 - K=2, F=3, S=2, P=1

• 출력 볼륨

- 크기와 계산
 - 가로/세로 각각 $(5 - 3 + 2)/2 + 1 = 3$
 - 입력(파란색)과 필터(빨간색)이 elementwise로 곱해진 뒤 하나로 더해지고
 - Bias가 더해짐
 - 필터 개수만큼 편향 수 필요

• 패러미터의 수: 커널의 원소 수 + 편향 수

- 커널 수(K) * 커널 사이즈(F)² * 채널:색상 수(D) + 커널 수(K)

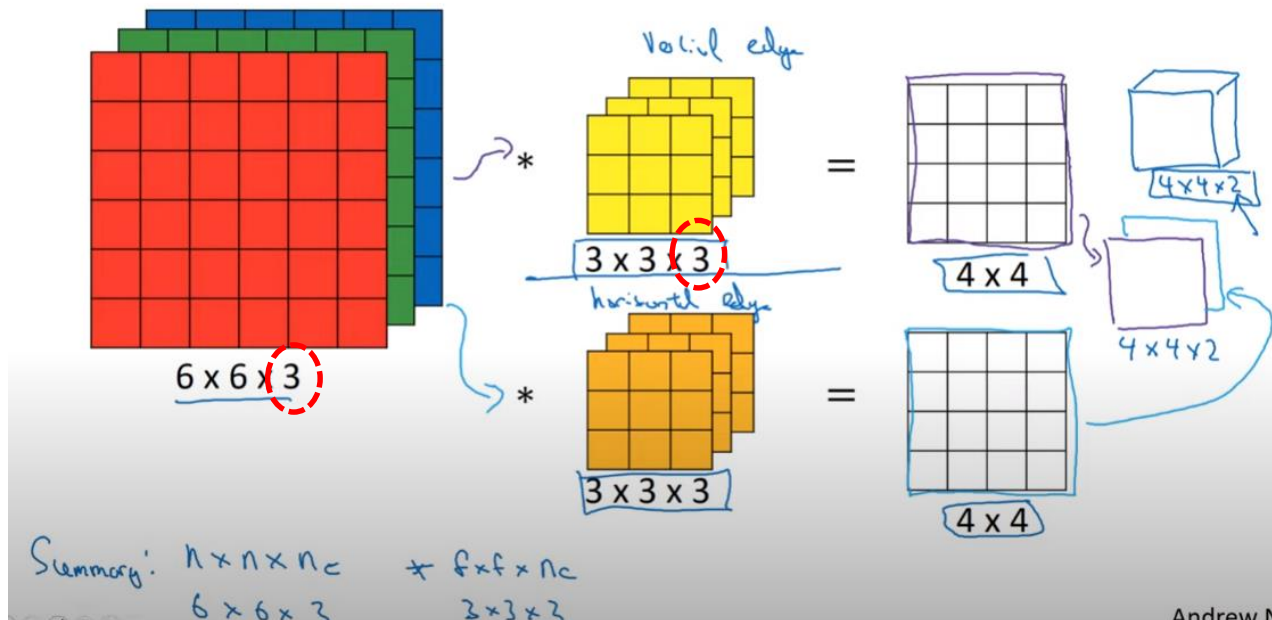


필터 수 2

- 채널 3
 - 커널 크기 3×3
- 결과
 - $4 \times 4 \times 2$
 - 4×4 가 2개

<https://www.deeplearning.ai/>

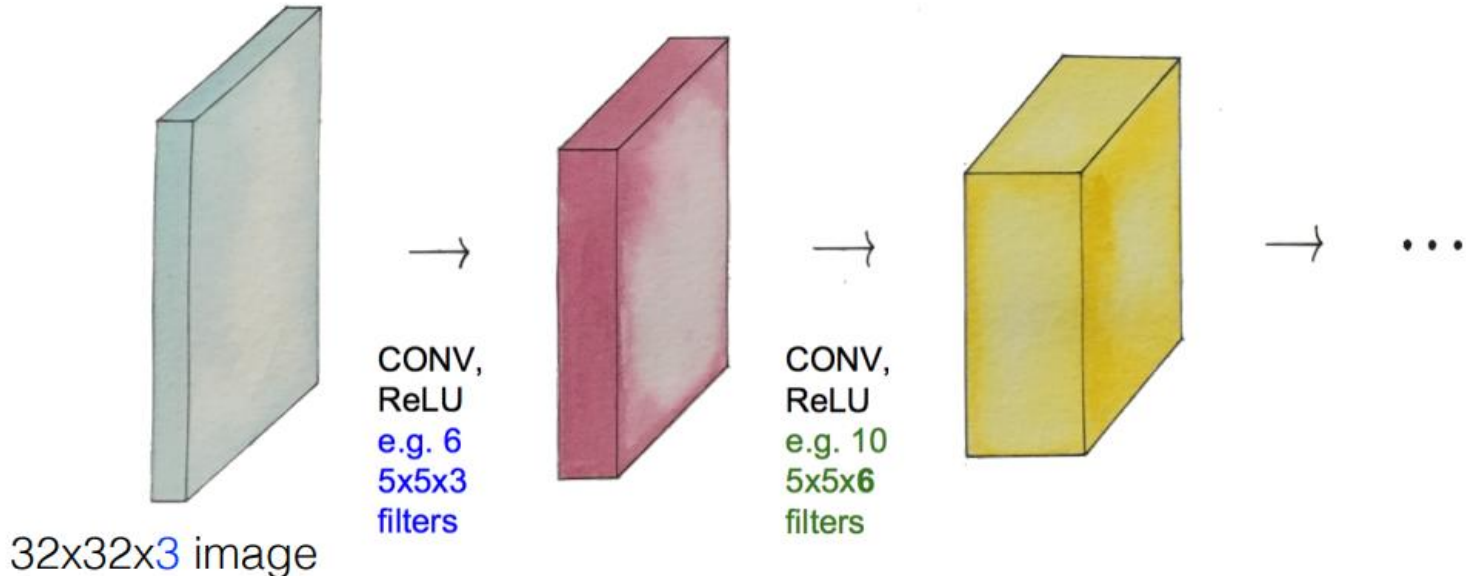
Multiple filters



Andrew N

각 컨볼루션 단계의 패러미터 수

- **패러미터의 수: 커널의 원소 수 + 편향 수**
 - 커널 수(K) * 커널 사이즈(F)² * 채널:색상 수(D) + 커널 수(K)
- **커널 5 * 5의 3개 인 층**
 - $3 * (5*5) * 3 + 3$
- **커널 5 * 5의 3개 인 층**
 - $6 * (5*5) * 3 + 6$



파일

- `cnn_basic.ipynb`

채널을 추가한 이미지 그리기

- 3 x 3 이미지 하나

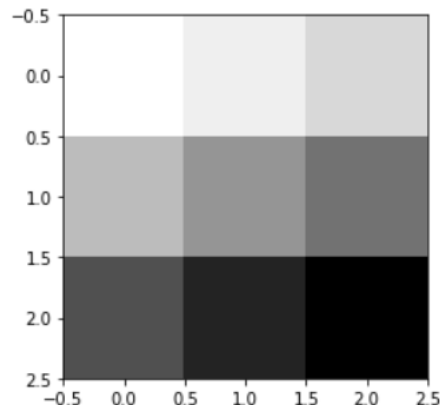
- (1, 3, 3)

- 채널 추가

- (1, 3, 3, 1)

```
[3] 1 image = np.array([[[[1],[2],[3]],
2          [[4],[5],[6]],
3          [[7],[8],[9]]]], dtype=np.float32)
4 print(image.shape)
5 plt.imshow(image.reshape(3,3), cmap='Greys')
```

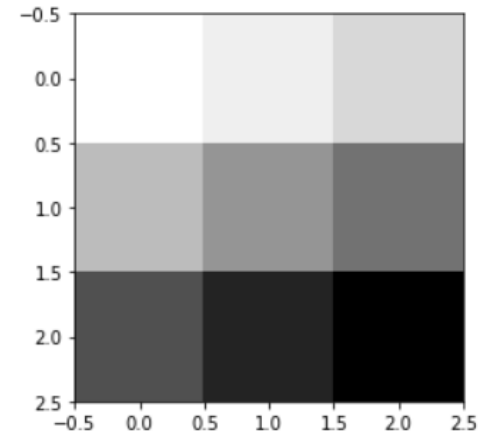
↗ (1, 3, 3, 1)
<matplotlib.image.AxesImage at 0x7f9001b1dba8>



```
[1] 1 # %matplotlib inline
2 import numpy as np
3 import tensorflow as tf
4 import matplotlib.pyplot as plt
```

```
[2] 1 image = np.array([[[[1,2,3],
2          [4,5,6],
3          [7,8,9]]], dtype=np.float32)
4 print(image.shape)
5 image = image.reshape(-1, 3, 3, 1)
6 '''
7 image = np.array([[[[1],[2],[3]],
8          [[4],[5],[6]],
9          [[7],[8],[9]]]], dtype=np.float32)
10 '''
11 print(image.shape)
12 plt.imshow(image.reshape(3,3), cmap='Greys')
```

↗ (1, 3, 3)
(1, 3, 3, 1)
<matplotlib.image.AxesImage at 0x7f9001b7ff98>



1 filter (2,2,1,1) with padding: VALID

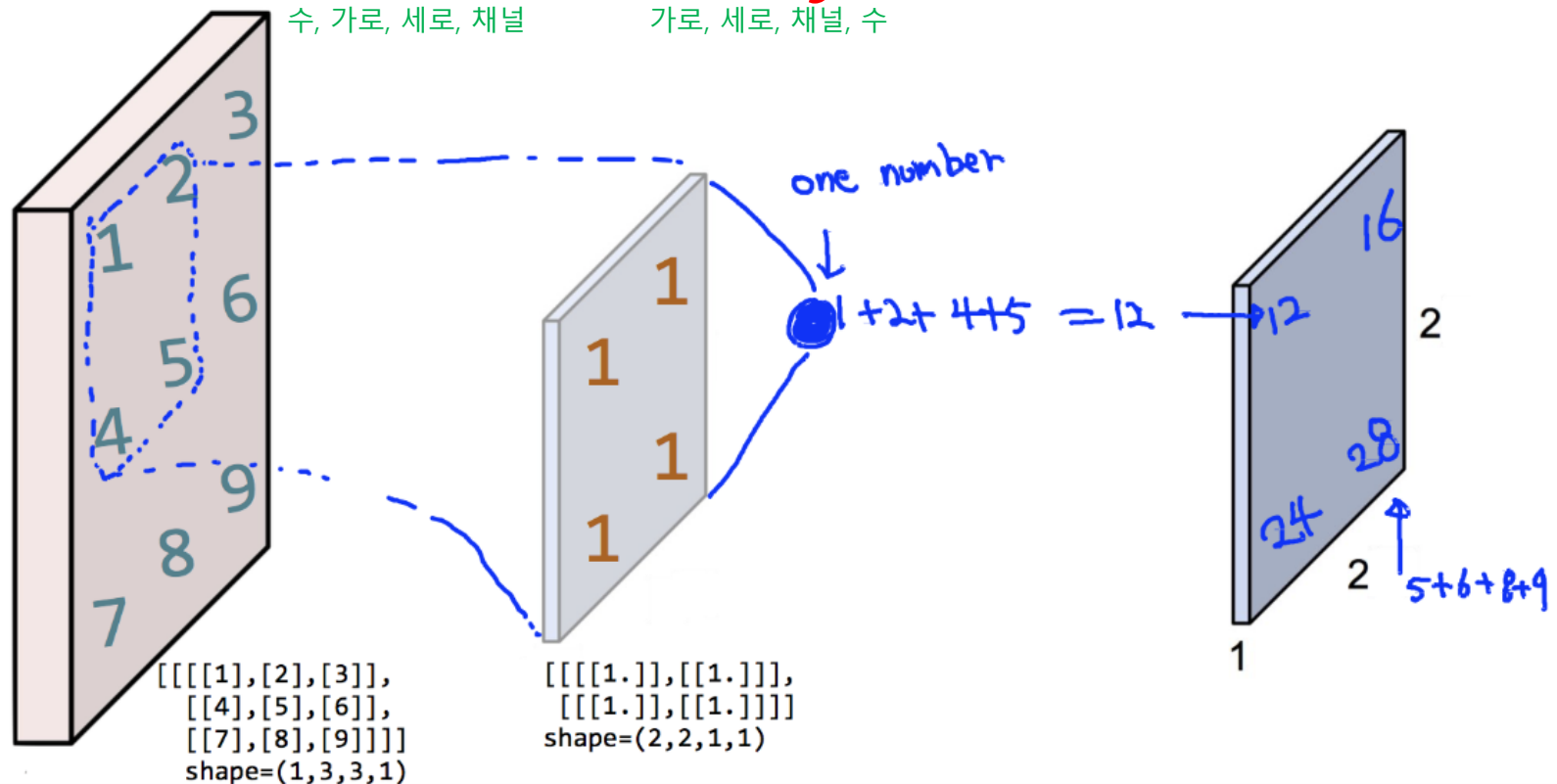
weight.shape = 1 filter (2, 2, 1, 1)

Simple convolution layer

Image: 1,3,3,1 image, Filter: 2,2,1,1, Stride: 1x1, Padding: VALID

수, 가로, 세로, 채널

가로, 세로, 채널, 수



tf.nn.conv2d

```

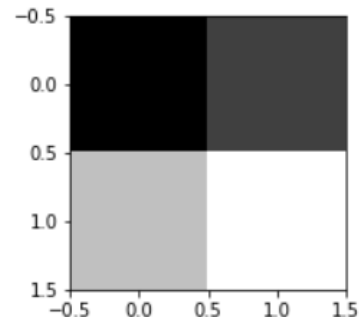
1 # print("imag:\n", image)
2 print("image.shape", image.shape)
3 # weight가 커널
4 weight = tf.constant([[[[1.]], [[1.]]],
5                       [[[1.]], [[1.]]]])
6 print("weight.shape", weight.shape)
7
8 conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='VALID')
9 print("conv2d.shape", conv2d.shape)
10
11 conv2d_img = np.swapaxes(conv2d, 0, 3)
12
13 for i, one_img in enumerate(conv2d_img):
14     print(one_img.reshape(2,2))
15     plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(2,2), cmap='gray')

```

```

image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 1)
conv2d.shape (1, 2, 2, 1)
[[12. 16.]
 [24. 28.]]

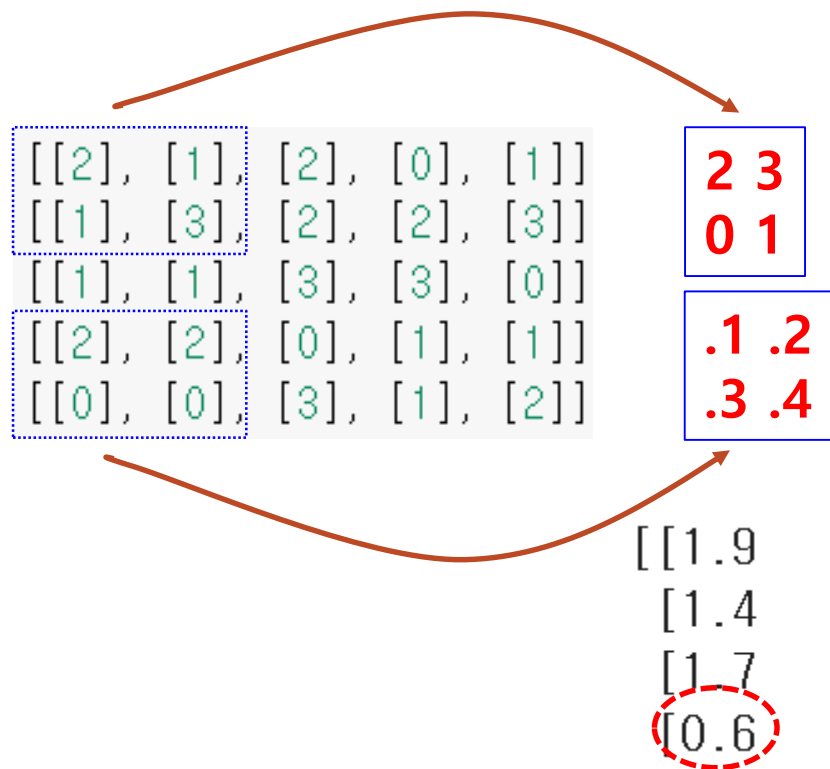
```



2 x 2 커널 2개 적용

• Strides 1

– 결과 4 x 4



```
[[10. 10. 6. 6.]
 [12. 15. 13. 13.]
 [ 7. 11. 16. 7.]
 [10. 7. 4. 7.]]
```

$$= 2*2 + 1*3 + 1*0 + 3*1 = 10$$

$$= 2*.1 + 2*.2 + 0*.3 + 0*.4 = .6$$

```
[[1.9 2.2 1.6 2.]
 [1.4 2.2 2.7 1.7]
 [1.7 1.3000001 1.3 1.]
 [0.6 1.4000001 1.5 1.4]]
```

2 x 2 커널 2개 적용 컨볼루션

```
x_in = np.array([[
    [[2], [1], [2], [0], [1]],
    [[1], [3], [2], [2], [3]],
    [[1], [1], [3], [3], [0]],
    [[2], [2], [0], [1], [1]],
    [[0], [0], [3], [1], [2]], ]])
x = tf.constant(x_in, dtype=tf.float32)

# 2 x 2 커널 2개 적용
kernel_in = np.array([
    [[2, 0.1]], [[3, 0.2]] ],
    [[0, 0.3]], [[1, 0.4]] ], ])
```

```
kernel = tf.constant(kernel_in, dtype=tf.float32)

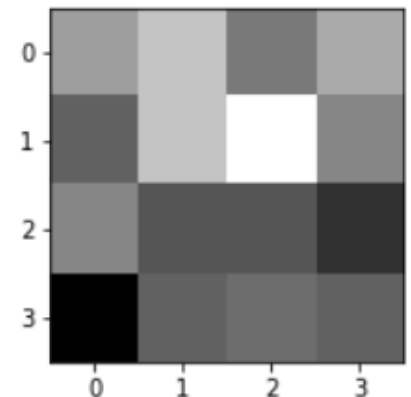
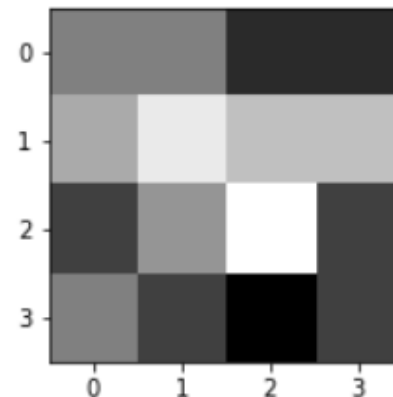
conv2d = tf.nn.conv2d(x, kernel, strides=[1, 1, 1, 1], padding='VALID')
print("conv2d.shape", conv2d.shape)
```

```
conv2d_img = np.swapaxes(conv2d, 0, 3)
for i, one_img in enumerate(conv2d_img):
    print(one_img.reshape(4,4))
    plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(4,4), cmap='gray')
```

```
conv2d.shape (1, 4, 4, 2)
```

```
[[10. 10.  6.  6.]
 [12. 15. 13. 13.]
 [ 7. 11. 16.  7.]
 [10.  7.  4.  7.]]
```

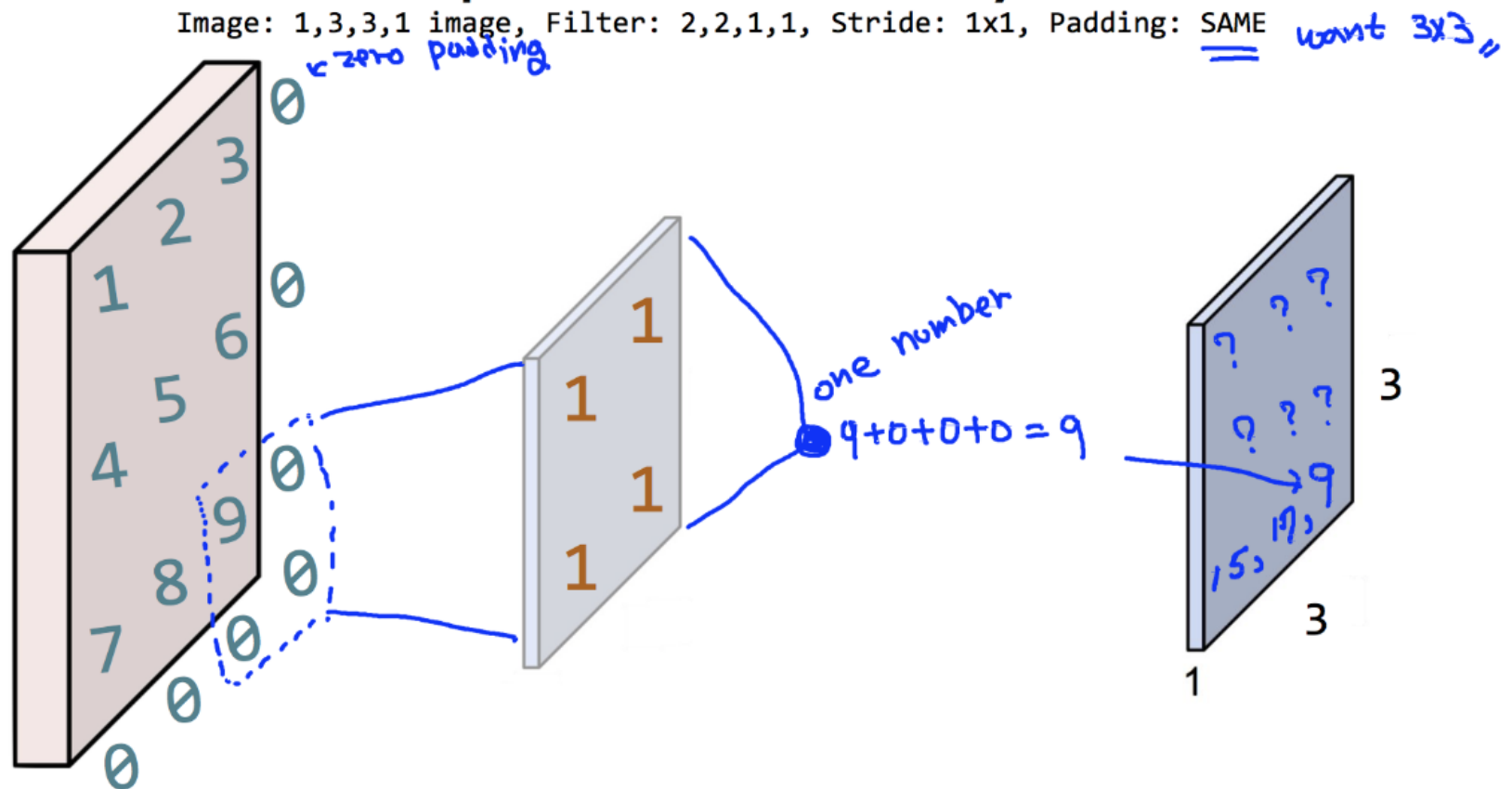
```
[[1.9      2.2      1.6      2.      ]
 [1.4      2.2      2.7      1.7      ]
 [1.7      1.3000001 1.3      1.      ]
 [0.6      1.4000001 1.5      1.4      ]]
```



`tf.nn.conv2d(..., padding='SAME')`

1 filter (2,2,1,1) with padding:SAME

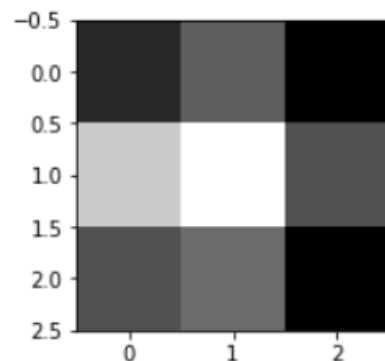
Simple convolution layer



컨볼루션 결과 크기도 원본과 동일

```
[7] 1 print("image.shape", image.shape)
    2
    3 weight = tf.constant([[[[1.]], [[1.]]],
    4                       [[[1.]], [[1.]]]])
    5 print("weight.shape", weight.shape)
    6
    7 conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
    8 print("conv2d.shape", conv2d.shape)
    9
   10 conv2d_img = np.swapaxes(conv2d, 0, 3)
   11 for i, one_img in enumerate(conv2d_img):
   12     print(one_img.reshape(3,3))
   13     plt.subplot(1,2,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

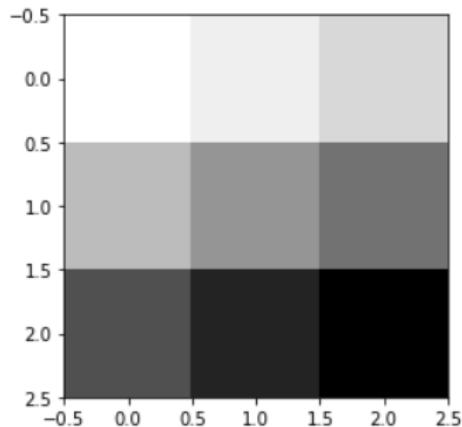
```
↳ image.shape (1, 3, 3, 1)
   weight.shape (2, 2, 1, 1)
   conv2d.shape (1, 3, 3, 1)
   [[12. 16.  9.]
    [24. 28. 15.]
    [15. 17.  9.]
```



필터가 3개

```
1 image = np.array([[[[1],[2],[3]],
2                   [[4],[5],[6]],
3                   [[7],[8],[9]]]], dtype=np.float32)
4 print(image.shape)
5 plt.imshow(image.reshape(3,3), cmap='Greys')
```

```
(1, 3, 3, 1)
<matplotlib.image.AxesImage at 0x7f9001b1dba8>
```



1	1
1	1

```
[[12. 16. 9.]
 [24. 28. 15.]
 [15. 17. 9.]]
```

10	10
10	10

```
[[120. 160. 90.]
 [240. 280. 150.]
 [150. 170. 90.]]
```

-1	-1
-1	-1

```
[[ -12.  -16.  -9.]
 [-24.  -28. -15.]
 [-15.  -17.  -9.]]
```

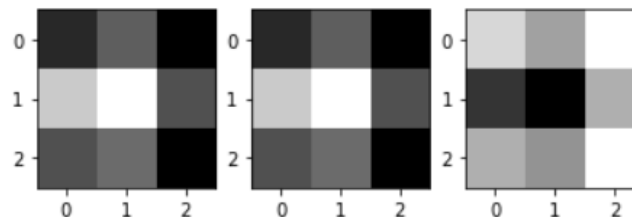
```
weight = tf.constant([[[[1., 10., -1.]], [[1., 10., -1.]]],
                      [[[1., 10., -1.]], [[1., 10., -1.]]]])
```

(2,2,1,3)

2행 2열의 채널 1개, 3개 filters (2,2,1,3)

```
[8] 1 # print("imag:₩n", image)
2 print("image.shape", image.shape)
3
4 weight = tf.constant([[[[1.,10.,-1.],[[1.,10.,-1.]],
5                        [[1.,10.,-1.],[[1.,10.,-1.]]]])
6 print("weight.shape", weight.shape)
7
8 conv2d = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1], padding='SAME')
9 print("conv2d.shape", conv2d.shape)
10
11 conv2d_img = np.swapaxes(conv2d, 0, 3)
12 for i, one_img in enumerate(conv2d_img):
13     print(one_img.reshape(3,3))
14     plt.subplot(1,3,i+1), plt.imshow(one_img.reshape(3,3), cmap='gray')
```

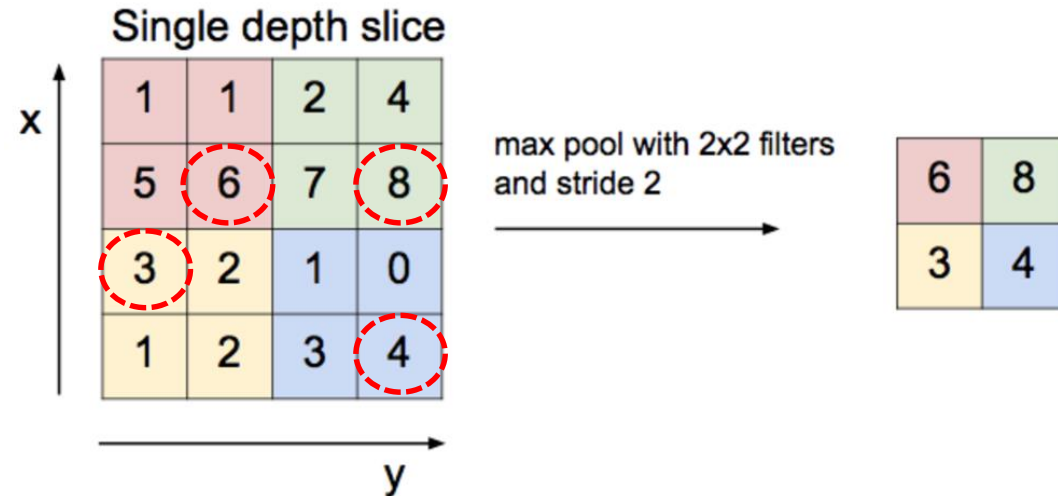
```
↳ image.shape (1, 3, 3, 1)
weight.shape (2, 2, 1, 3)
conv2d.shape (1, 3, 3, 3)
[[12. 16.  9.]
 [24. 28. 15.]
 [15. 17.  9.]]
[[120. 160.  90.]
 [240. 280. 150.]
 [150. 170.  90.]]
[[-12. -16.  -9.]
 [-24. -28. -15.]
 [-15. -17.  -9.]]
```



커널이 2, 이미지가 2 x 2인 맥스풀링

- padding='VALID'

MAX POOLING



```

1 image = np.array([[[[4],[3]],
2                   [[2],[1]]], dtype=np.float32)
3 pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
4                       strides=[1, 1, 1, 1], padding='VALID')
5 print(pool.shape)
6 print(pool.numpy())

```

```

(1, 1, 1, 1)
[[[4.]]]

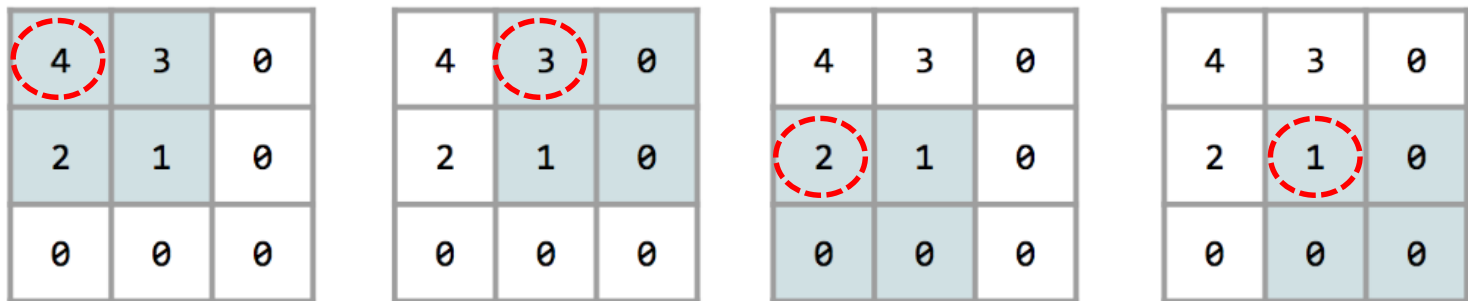
```



옵션 padding

- padding='SAME'

▼ SAME: Zero paddings



```
[10] 1 image = np.array([[[[4],[3]],
2           [[2],[1]]]], dtype=np.float32)
3 pool = tf.nn.max_pool(image, ksize=[1, 2, 2, 1],
4           strides=[1, 1, 1, 1], padding='SAME')
5 print(pool.shape)
6 print(pool.numpy())
```

```
↳ (1, 2, 2, 1)
   [[[4.]
    [3.]]
   [[2.]
    [1.]]]]
```

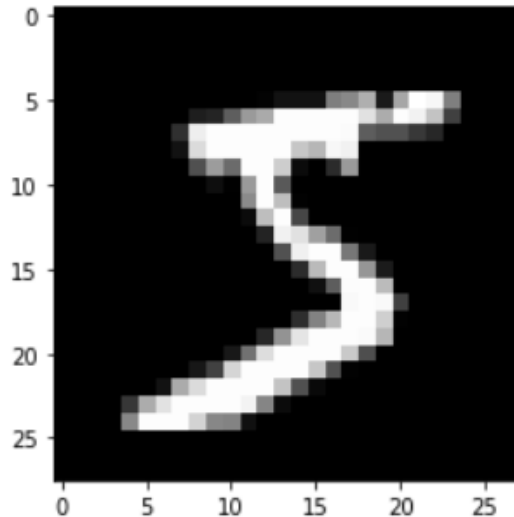
학습 데이터

- 첫 이미지

```
[11] 1 import tensorflow as tf  
      2 mnist = tf.keras.datasets.mnist  
      3  
      4 mnist = mnist.load_data()
```

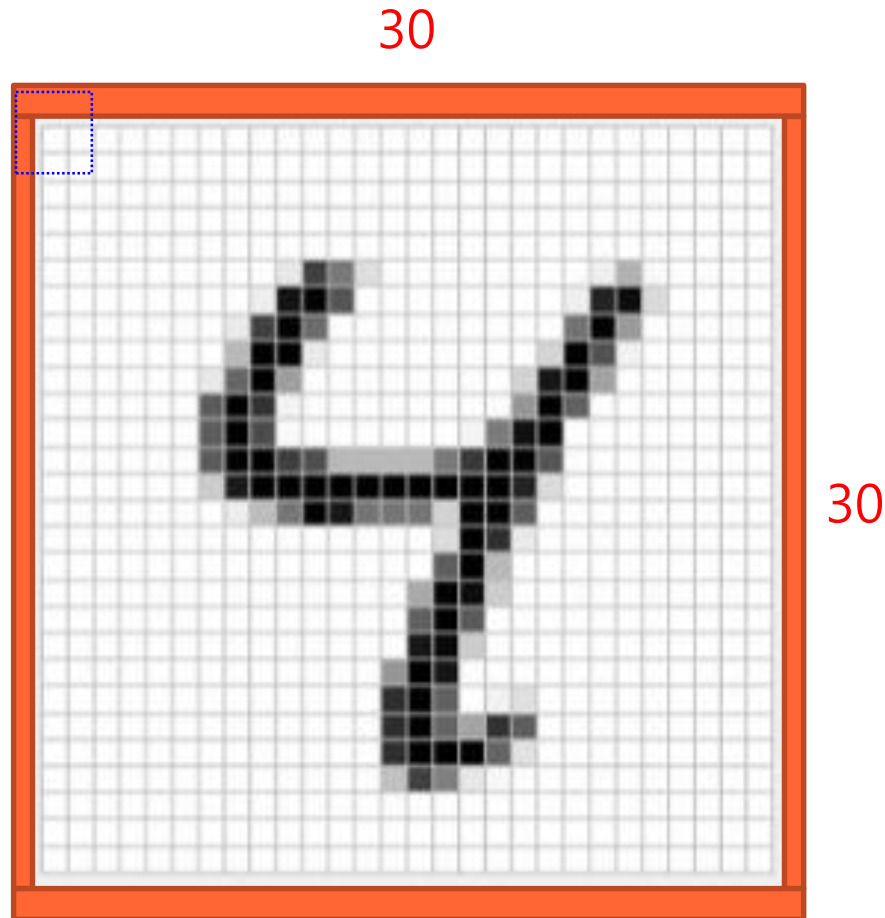
```
[12] 1 img = mnist[0][0][0].reshape(28,28)  
      2 plt.imshow(img, cmap='gray')  
      3 print(img.shape)
```

☞ (28, 28)



필터 크기가 3이고 strides 2

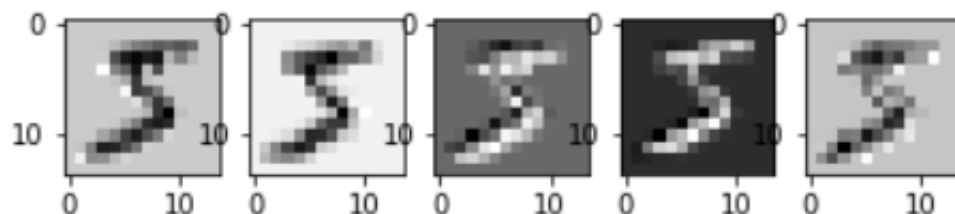
- 필터의 사이즈가 k
 - 사방으로 $k/2$ 만큼의 패딩
 - $K=3$
 - $3/2 = 1$
- Strides 2인 경우
 - 결과
 - 14×14



필터 크기가 3이고 **strides** 2, 필터 수가 5

```
[13] 1 # 4차원으로 확장
      2 img = img.reshape(-1,28,28,1)
      3 # 3 x 3의 필터 5개 생성
      4 W1 = tf.Variable(tf.random.normal([3, 3, 1, 5], stddev=0.01))
      5
      6 conv2d = tf.nn.conv2d(img, W1, strides=[1, 2, 2, 1], padding='SAME')
      7 print(conv2d.shape)
      8
      9 conv2d_img = np.swapaxes(conv2d, 0, 3)
     10 for i, one_img in enumerate(conv2d_img):
     11     plt.subplot(1,5,i+1)
     12     plt.imshow(one_img.reshape(14,14), cmap='gray')
     13
```

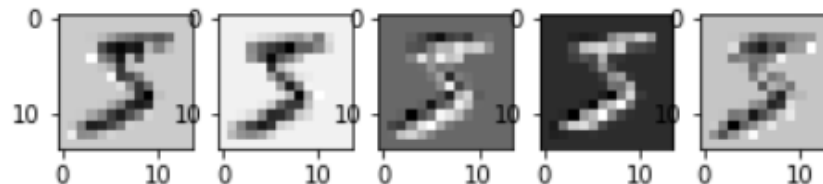
↳ (1, 14, 14, 5)



필터 크기가 2이고 strides 2 적용

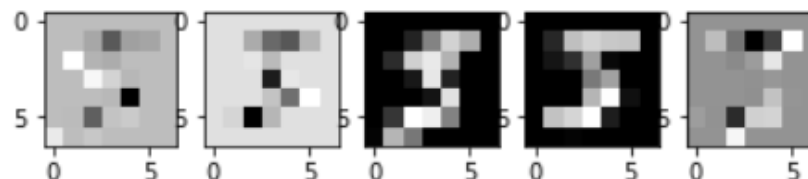
• 이미지의 크기가 반으로 줄

↳ (1, 14, 14, 5)



```
[15] 1 pool = tf.nn.max_pool(conv2d, ksize=[1, 2, 2, 1],
2                                strides=[1, 2, 2, 1], padding='SAME')
3 print(pool.shape)
4
5 pool_img = np.swapaxes(pool, 0, 3)
6 for i, one_img in enumerate(pool_img):
7     plt.subplot(1,5,i+1)
8     plt.imshow(one_img.reshape(7, 7), cmap='gray')
```

↳ (1, 7, 7, 5)



이미지를 위한 `keras.layers.Conv2D`

- `keras.layers.Conv2D()`

- `filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, activation=None, use_bias=True, ...`

- **주요 인자**

- `filters`: 정수, 컨볼루션의 아웃풋 필터의 수
- `kernel_size`: 정수 혹은 단일 정수의 튜플/리스트, 커널 크기(가로, 세로)
- `Strides`: 정수 혹은 단일 정수의 튜플/리스트. 컨볼루션의 보폭 길이를 특정
- `padding`: "valid", 혹은 "same" (대소문자 무시). "valid"는 "패딩 없음"을 의미
 - "same"은 아웃풋이 원래 인풋과 동일한 길이를 갖도록 인풋을 패딩
 - 제로 패딩을 사용해 아웃풋이 원래의 인풋과 같은 길이를 갖도록
- `data_format`: 문자열, "channels_last" (디폴트 값) 혹은 "channels_first" 중 하나
 - **입력 차원의 순서를 표현**
 - "channels_last"는 (이미지수, 높이, 너비, channels) 형태
 - "channels_first"는 (이미지수, channels, 높이, 너비) 형태

- **입력 형태**

- `data_format`이 "channels_last"이면 (batch, rows, cols, channels) 형태의 4D 텐서
 - `data_format`이 "channels_first"이면 (batch, channels, rows, cols) 4D 텐서

- **출력 형태**

- `data_format`이 "channels_last"이면 (batch, new_rows, new_cols, filters) 형태의 4D 텐서
 - `data_format`이 "channels_first"이면 (batch, filters, new_rows, new_cols) 4D 텐서

MNIST 입력 모양

- 흑백 이미지, 60000개 $28 * 28$, 기본 형태, 채널라스트
 - (6000, 28, 28, 1)

