FastCampus Pytorch

Ch4. Convolutional Neural Networks

HARRY KIM

Lecture Content

1. Convolutional Neural Network
2. Convolution Layer
3. GRAY/RGB
4. Pooling Layer
5. Fully Connected Layer

강의 자료

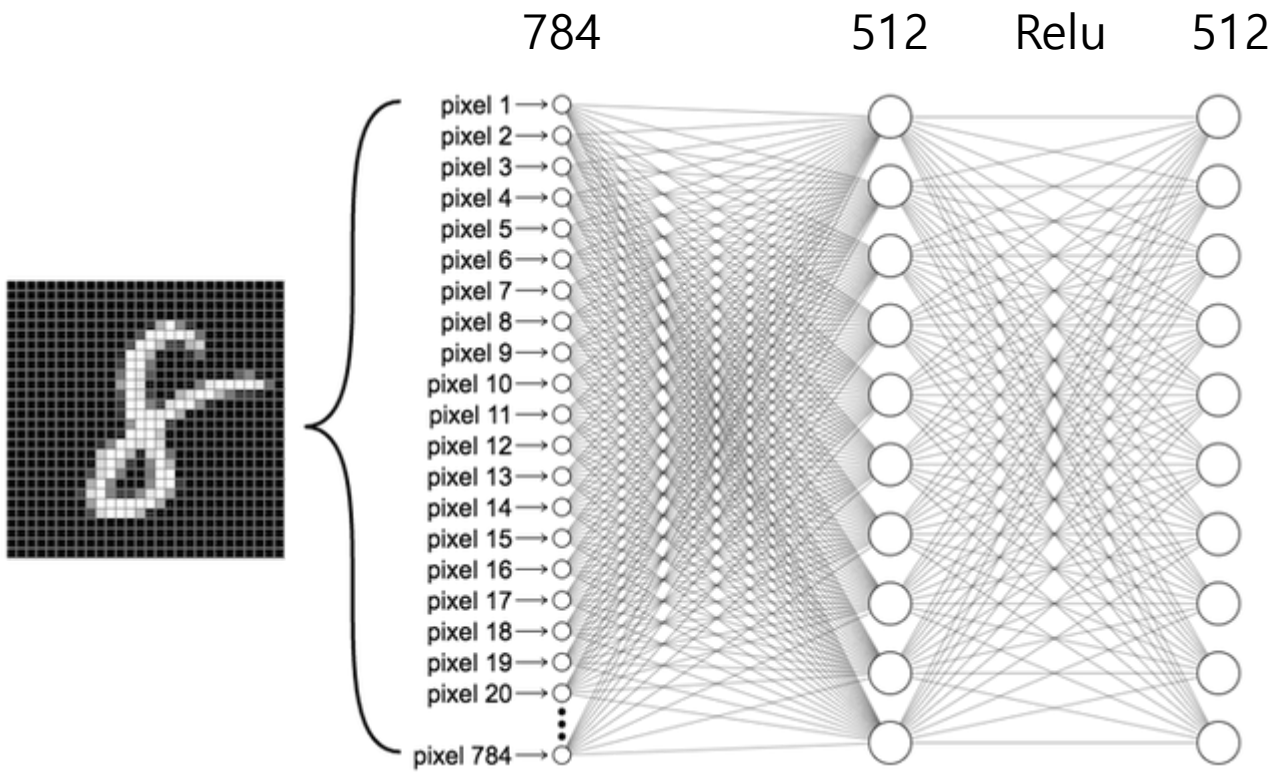
- Books
  - **밑바닥부터 시작하는 딥러닝 [사이토 고키, 2017]**
  - 머신러닝, 딥러닝 실전개발 입문 [쿠지라 히코우즈쿠에, 2017]

- Online
  - UVA DEEP LEARNING COURSE [University of Amsterdam, 2018]
  - **CS231n [http://cs231n.stanford.edu/, 2018]**

1. Convolutional Neural Network

Convolutional Neural Network

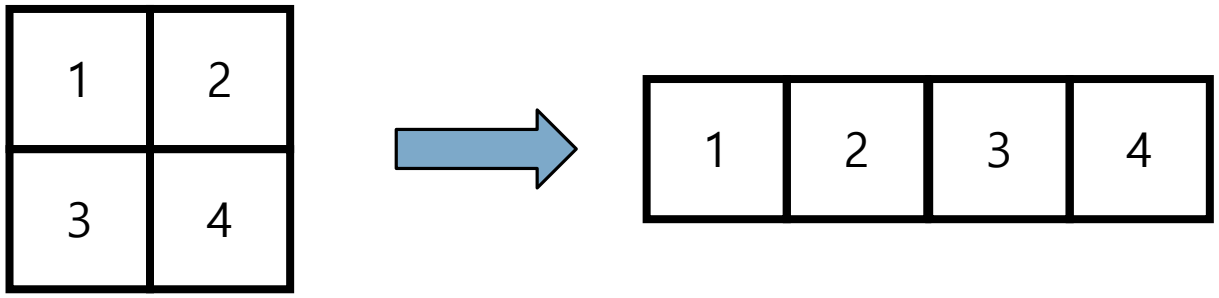
- **합성곱 신경망(CNN, Convolutional Neural Network)**
  - MNIST 학습
  - 각각의 Pixel 값 = 입력값



https://achintavarna.wordpress.com/2017/11/17/keras-tutorial-for-beginners-a-simple-neural-network-to-identify-numbers-mnist-data/

Convolutional Neural Network

- **합성곱 신경망(CNN, Convolutional Neural Network)**
  - MNIST 학습
  - 각각의 Pixel 값 = 입력값
  - 하지만 이미지는 '공간적' 구조를 가지고 있음
    - 기존 데이터는 28*28 = 784의 한 줄로 변환되어 입력
    - 이는 공간적 정보를 잃게함

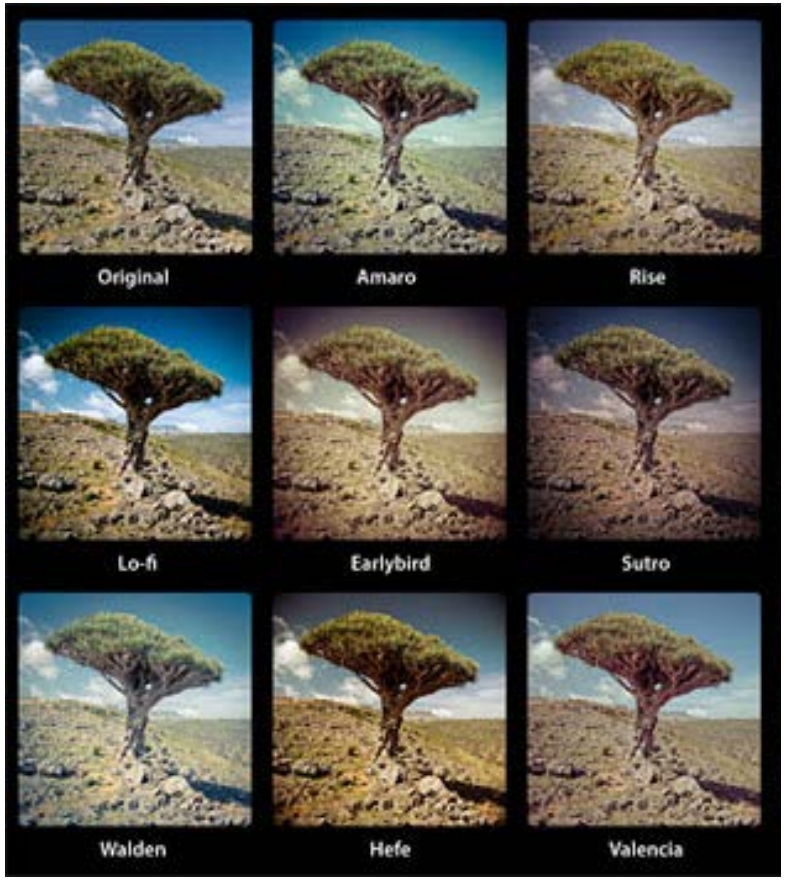


Fast campus

Convolutional Neural Network

- **합성곱 신경망(CNN, Convolutional Neural Network)**
  - 필터(Filler) 연산의 활용

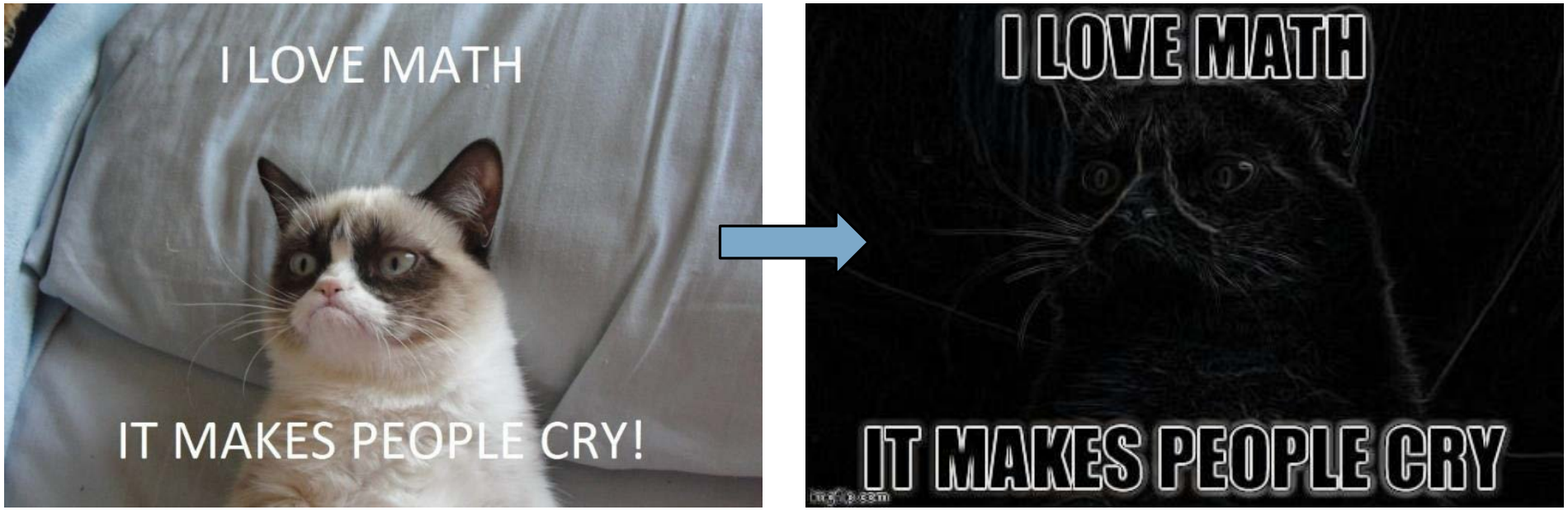
http://www.shellandslate.com/fastmedian.html



http://www.instamify.com/best-7-instagram-filters-to-get-wonderful-images/

Convolutional Neural Network

- **합성곱 신경망(CNN, Convolutional Neural Network)**
  - 필터(Filler) 연산의 활용

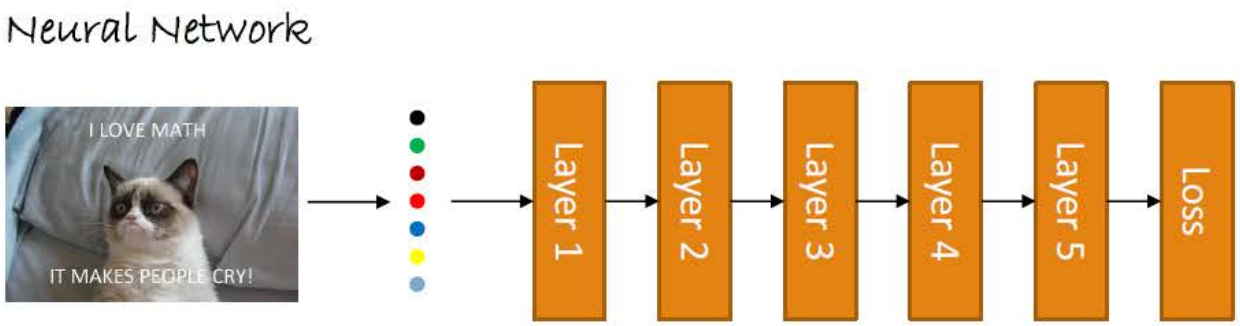


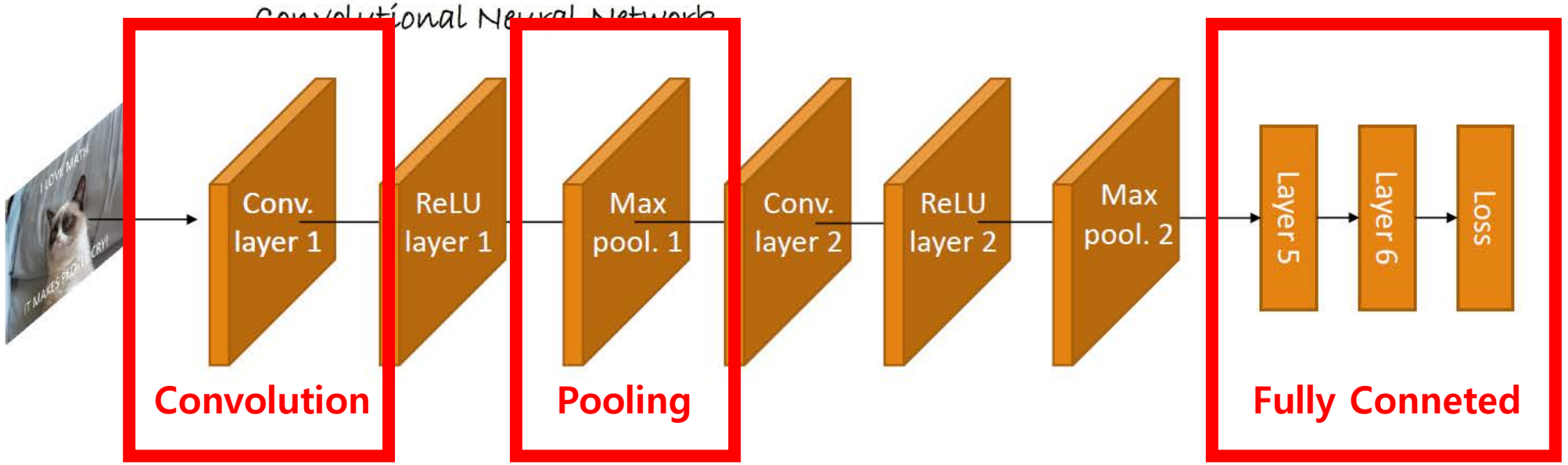
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Fast campus

Convolutional Neural Network

- 합성곱 신경망(CNN, Convolutional Neural Network)

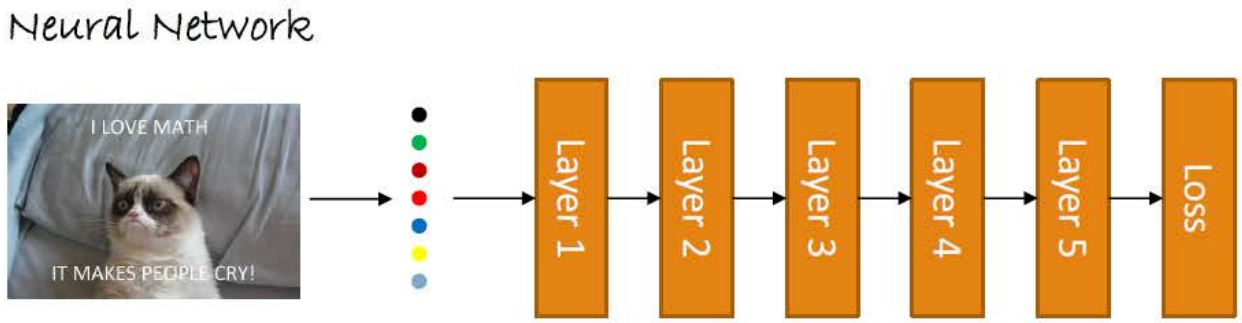


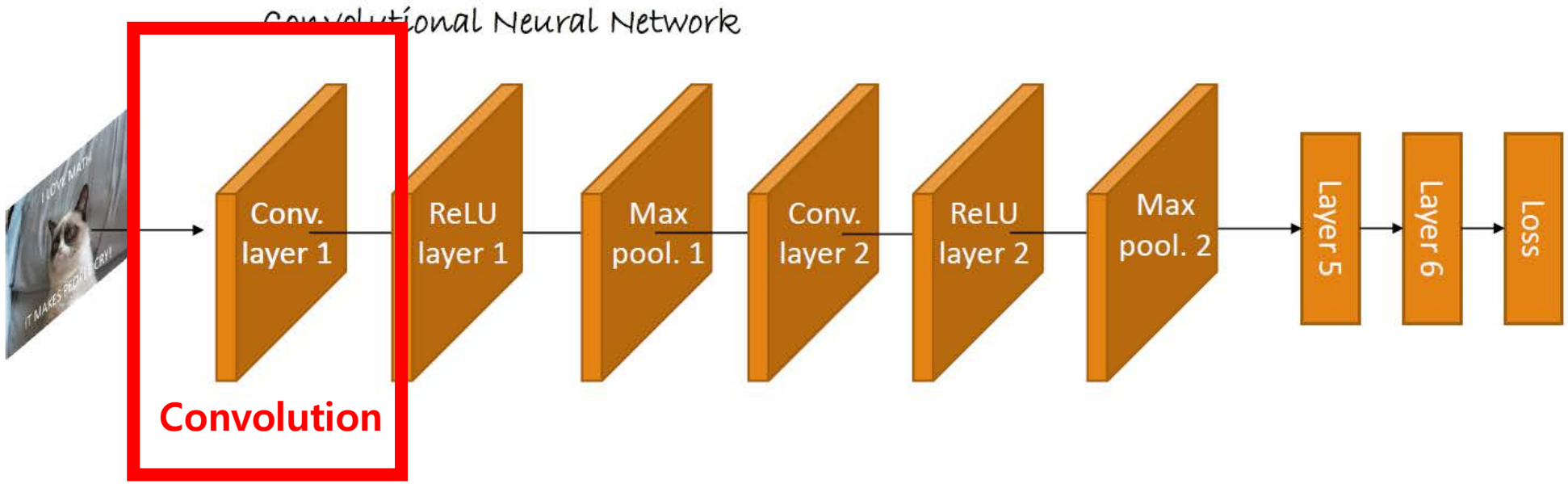


2. Convolutional Layer

Convolution Layer

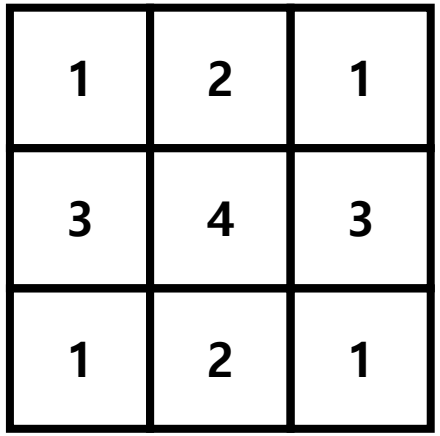
- 합성곱 신경망(CNN, Convolutional Neural Network)



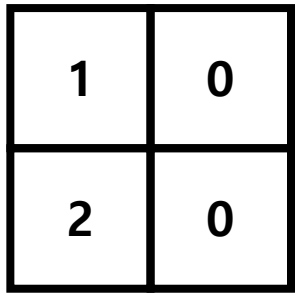


Convolution Layer

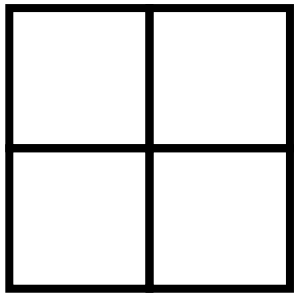
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초



<Input>



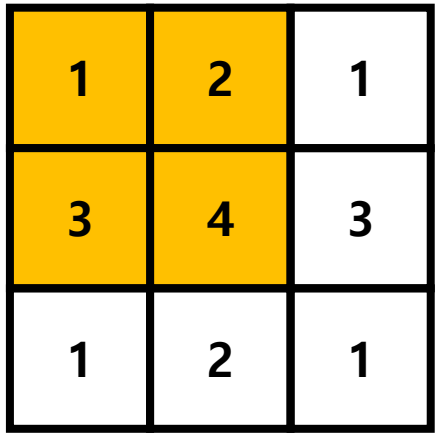
<Filter>



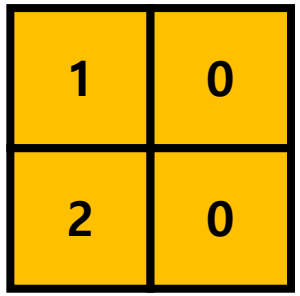
<Output>

Convolution Layer

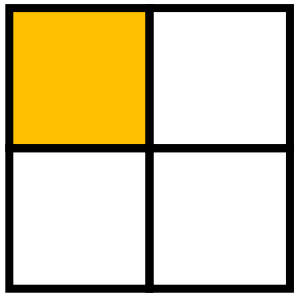
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초



<Input>



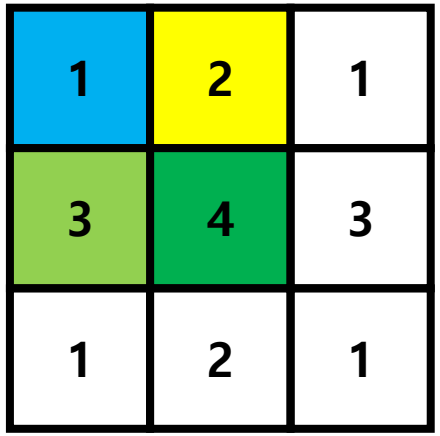
<Filter>



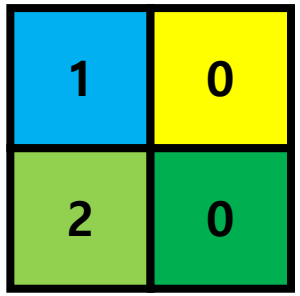
<Output>

Convolution Layer

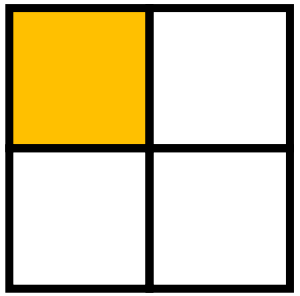
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초



<Input>



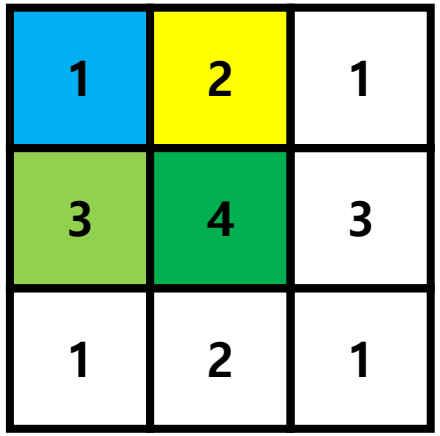
<Filter>



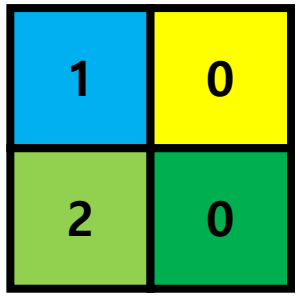
<Output>

Convolution Layer

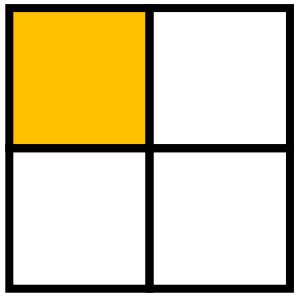
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초



<Input>



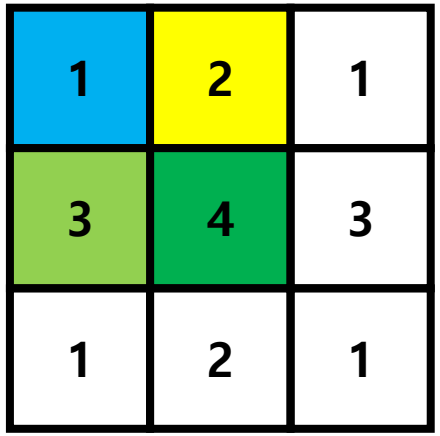
<Filter>



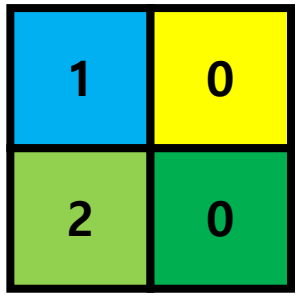
<Output>

Convolution Layer

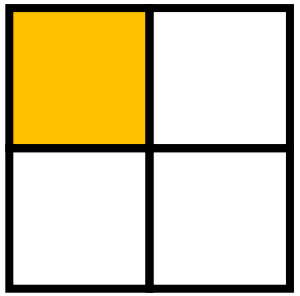
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초



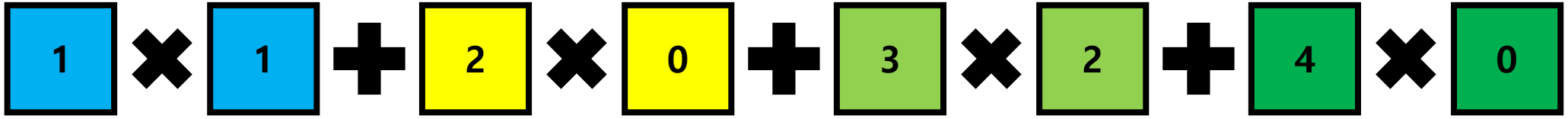
<Input>



<Filter>

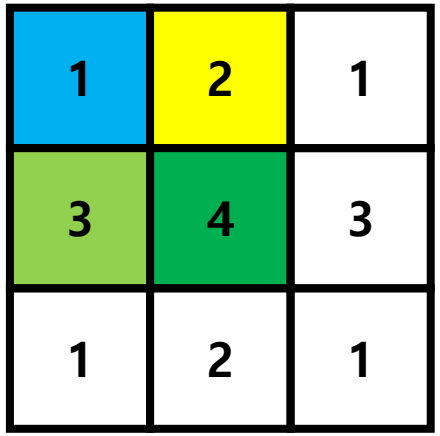


<Output>

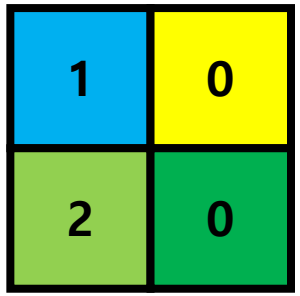


Convolution Layer

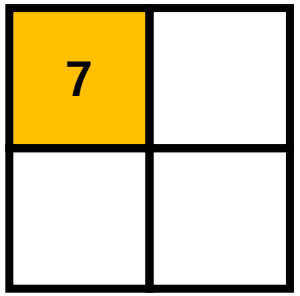
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초



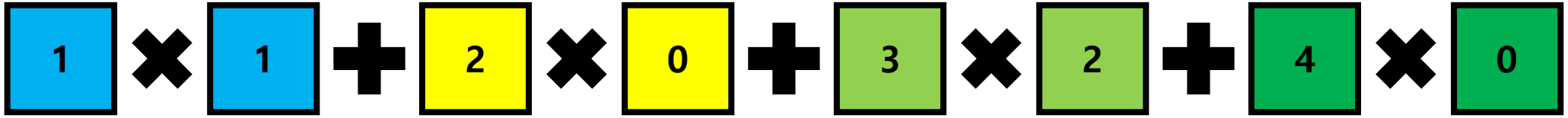
<Input>



<Filter>



<Output>



Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

<Filter>

?		

<Output>

Convolution Layer

- 합성곱(Convolution)
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

<Filter>

?		

<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

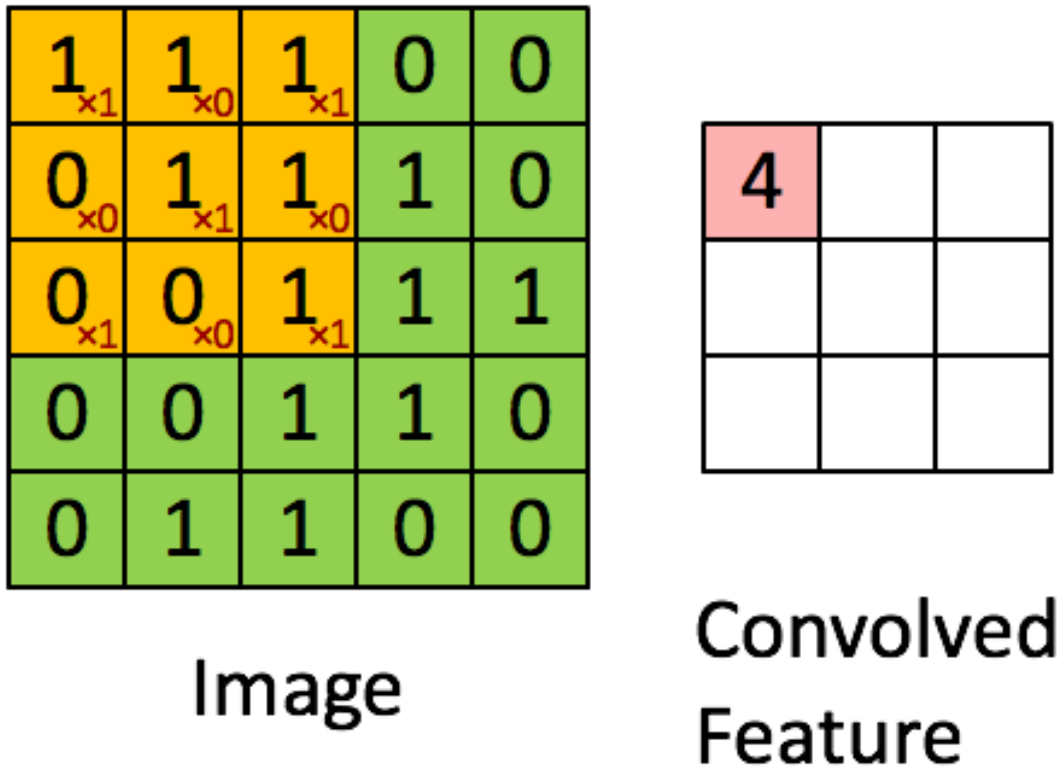
<Filter>

4		

<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초



Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>

?

<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>

?

<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>

?

<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>

?

<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>

?

<Output>

Convolution Layer

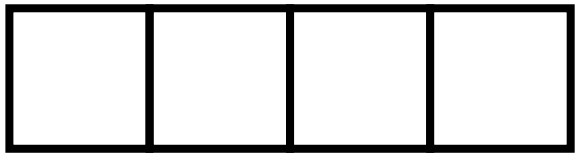
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>



<Output>

Convolution Layer

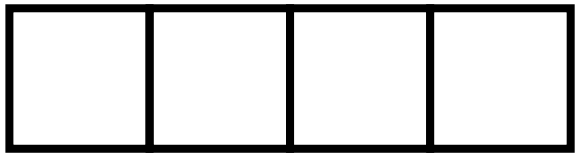
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>



<Output>

Convolution Layer

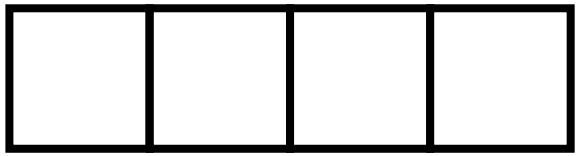
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>



<Output>

Convolution Layer

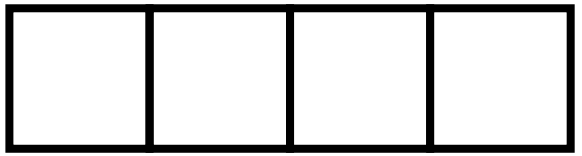
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>



<Output>

Convolution Layer

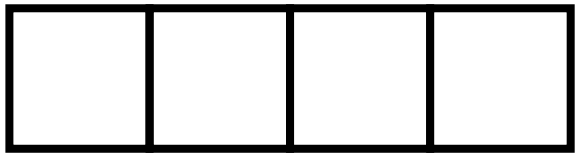
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>



<Output>

Convolution Layer

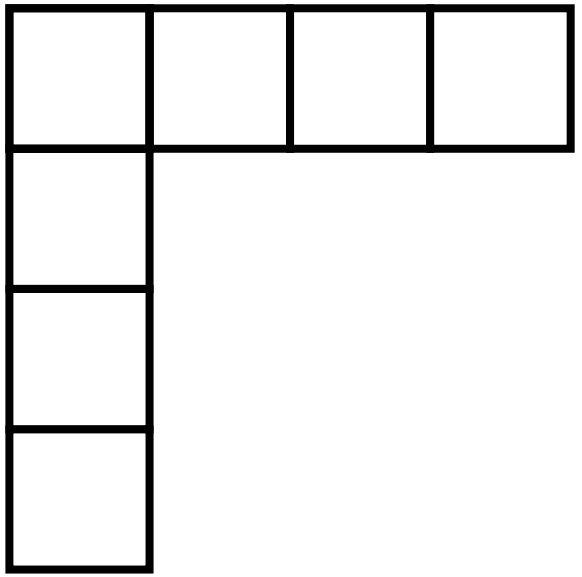
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0
0	1

<Filter>



<Output>

Convolution Layer

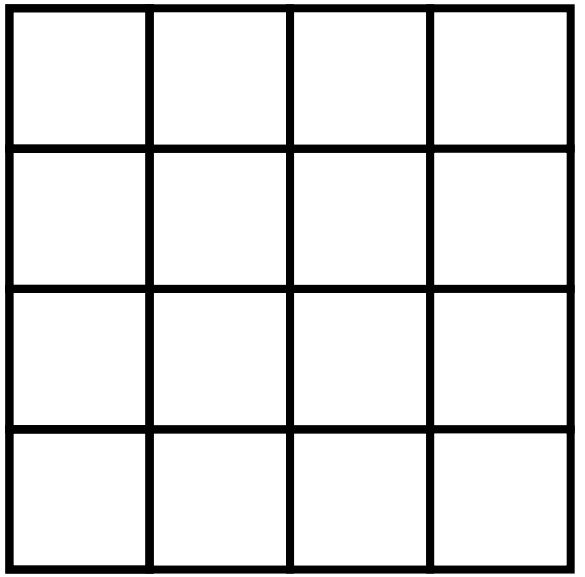
- **합성곱(Convolution)**
  - 필터(Filler) 연산의 기초

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5X5

1	0
0	1

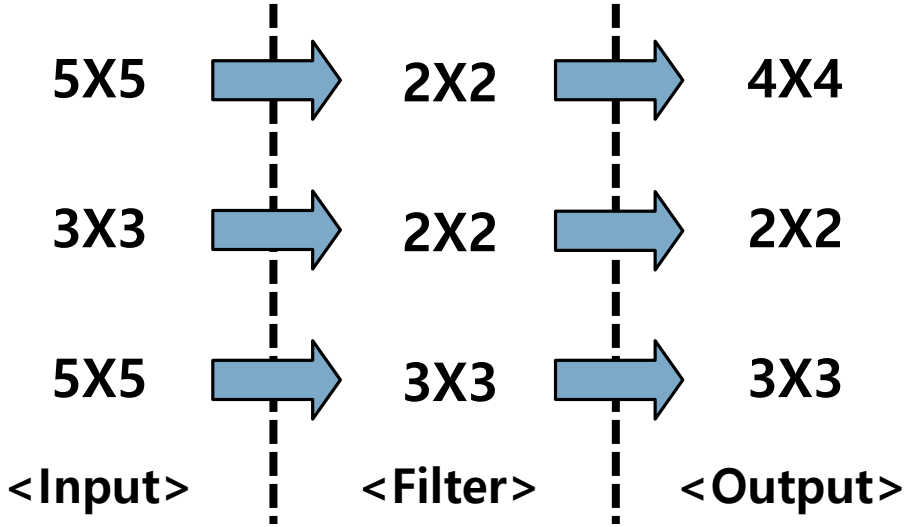
2X2



4X4

Convolution Layer

- **합성곱(Convolution)**
  - 필터(Filler) 연산의 특징 : **필터 적용 시 크기가 작아짐!**
  - 얼마나 줄어드는가?
  - Input : $N \times N$
  - Filter : $F \times F$
  - Output : $(N - F + 1) \times (N - F + 1)$

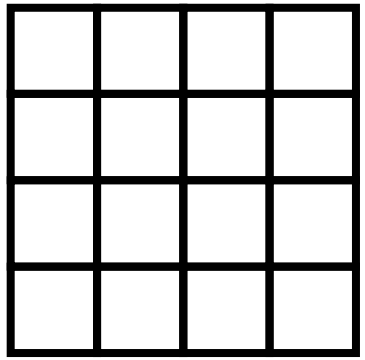


Convolution Layer

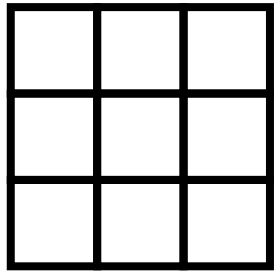
- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입



<Input>

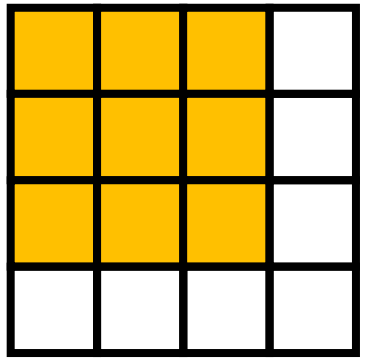


<Filter>

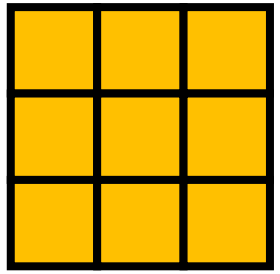
<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입



<Input>

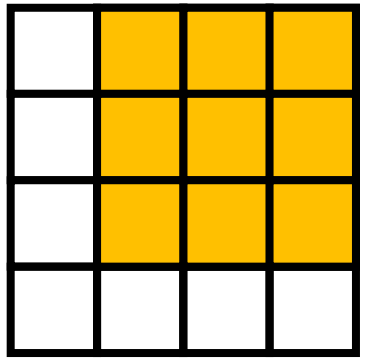


<Filter>

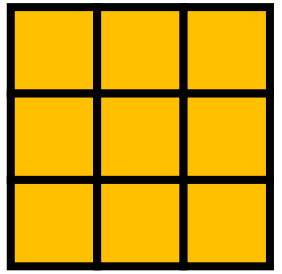
<Output>

Convolution Layer

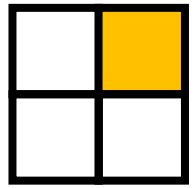
- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입



<Input>



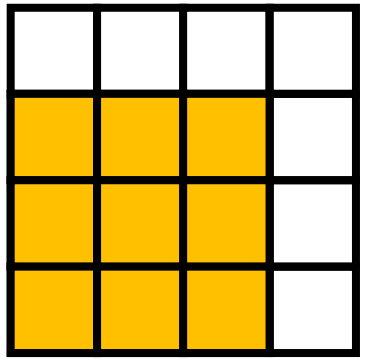
<Filter>



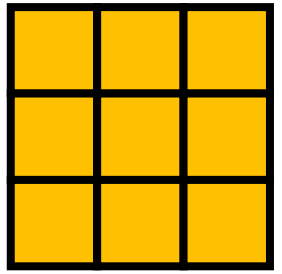
<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입



<Input>

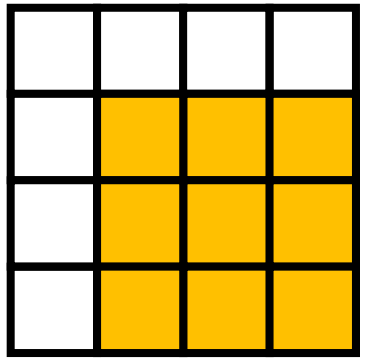


<Filter>

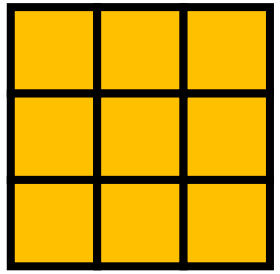
<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입



<Input>

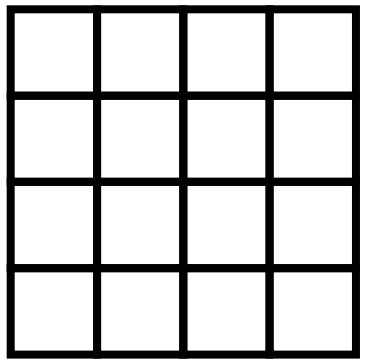


<Filter>

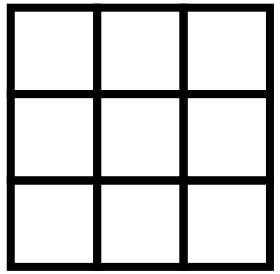
<Output>

Convolution Layer

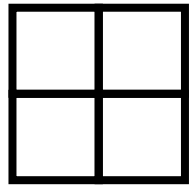
- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입



<Input>



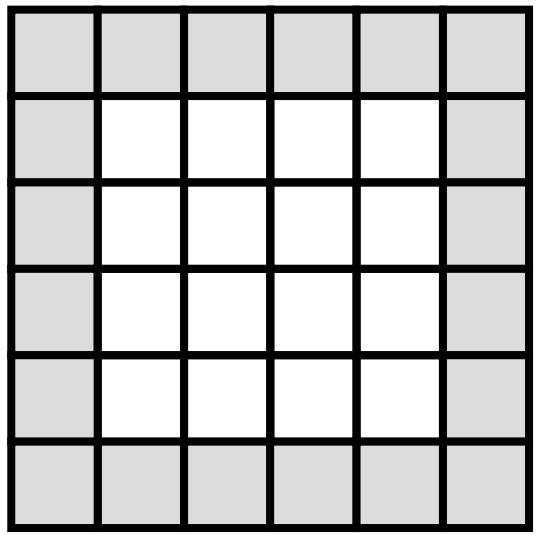
<Filter>



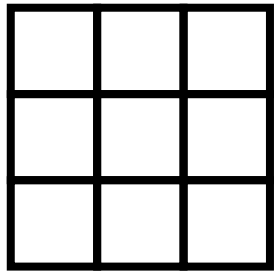
<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입
  - **패딩 1 :**



<Input>

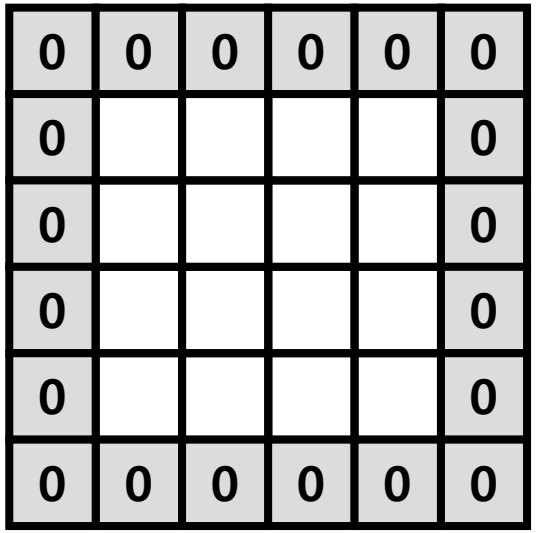


<Filter>

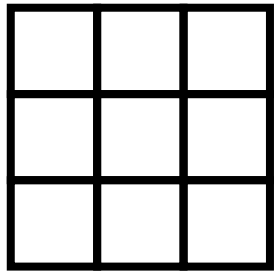
<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입
  - **제로 패딩(Zero Padding)** 1 : 일반적으로 패딩에는 0을 삽입 (Zero Padding)



<Input>

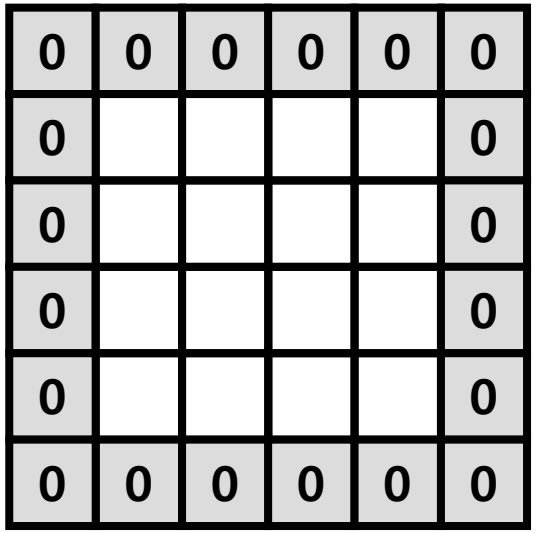


<Filter>

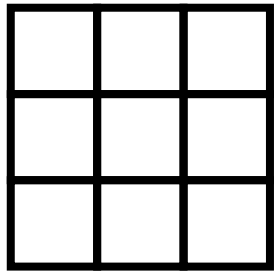
<Output>

Convolution Layer

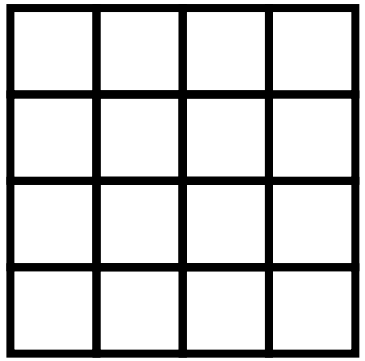
- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입
  - **패딩 : 원래 사이즈를 유지할 수 있음**



<Input>



<Filter>



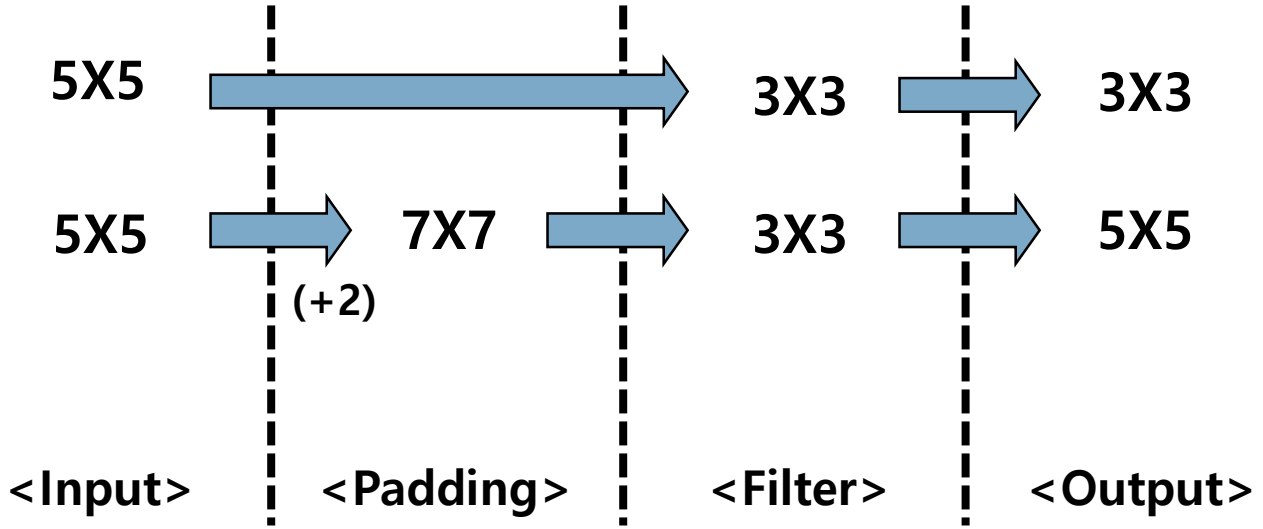
<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 사방에 빈 픽셀을 삽입
  - 패딩의 의의
    - 원래 사이즈를 유지할 수 있음
    - 합성곱 연산으로 인해 필터 적용 후 특징이 많이 사라질 수 있는 것을 방지
    - 오버피팅 방지 (원본 데이터에 0이 추가)

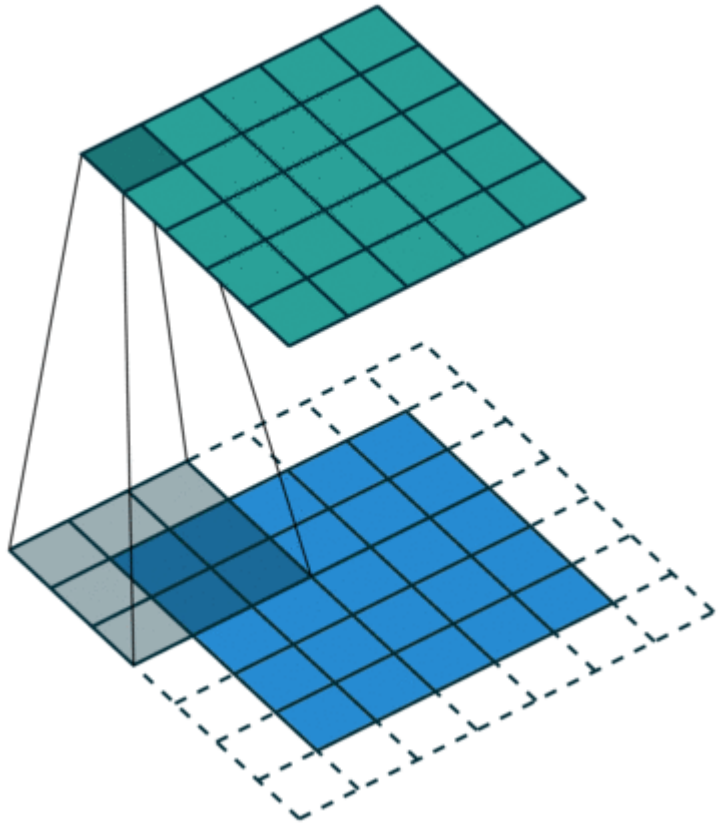
Convolution Layer

- **합성곱(Convolution)**
  - Input : $N \times N$
  - Padding : P
  - Filter : $F \times F$
  - Output : $(N + 2 * P - F + 1) \times (N + 2 * P - F + 1)$



Convolution Layer

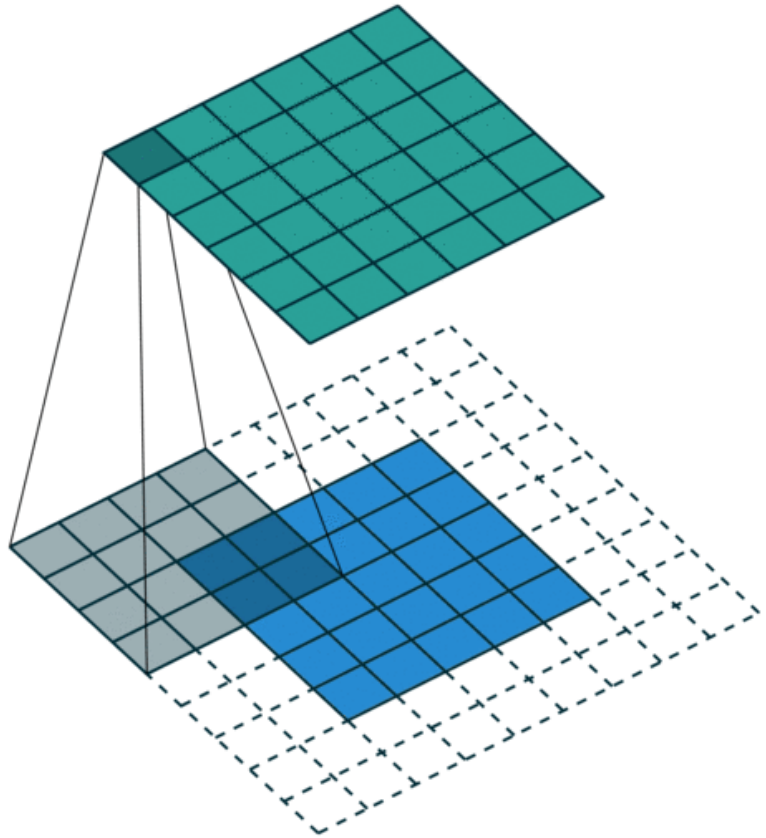
- **합성곱(Convolution)**
  - 패딩(Padding) : 1



https://tensorflow.blog/a-guide-to-convolution-arithmetic-for-deep-learning/

Convolution Layer

- **합성곱(Convolution)**
  - 패딩(Padding) : 2



https://tensorflow.blog/a-guide-to-convolution-arithmetic-for-deep-learning/

Convolution Layer

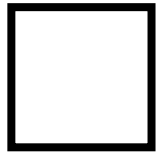
- **합성곱(Convolution)**
  - 스트라이드(Stride) : 지정한 개수만큼 건너뛰어 연산

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

<Filter>



<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 스트라이드(Stride) : 지정한 개수만큼 건너뛰어 연산

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

<Filter>

<Output>

Convolution Layer

- **합성곱(Convolution)**
  - 스트라이드(Stride) : 지정한 개수만큼 건너뛰어 연산

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

<Filter>

<Output>

Convolution Layer

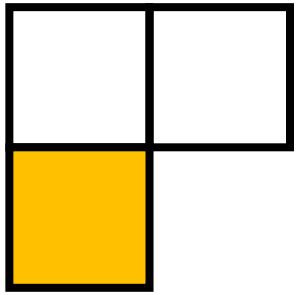
- **합성곱(Convolution)**
  - 스트라이드(Stride) : 지정한 개수만큼 건너뛰어 연산

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

<Filter>



<Output>

Convolution Layer

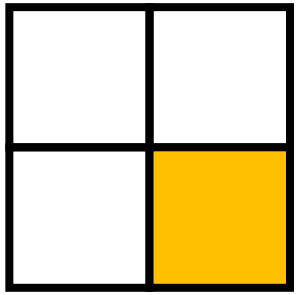
- **합성곱(Convolution)**
  - 스트라이드(Stride) : 지정한 개수만큼 건너뛰어 연산

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

<Input>

1	0	1
0	1	0
1	0	1

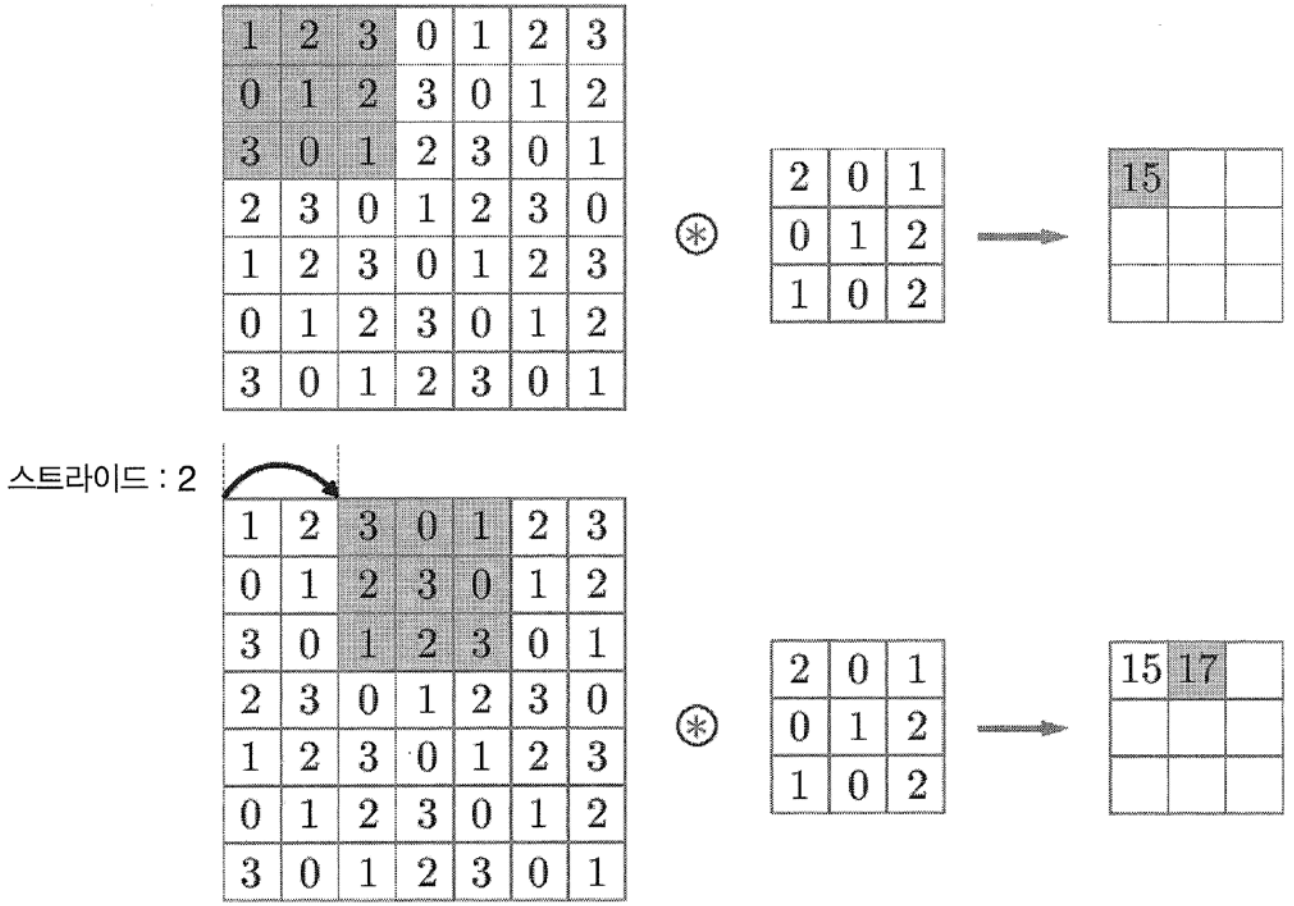
<Filter>



<Output>

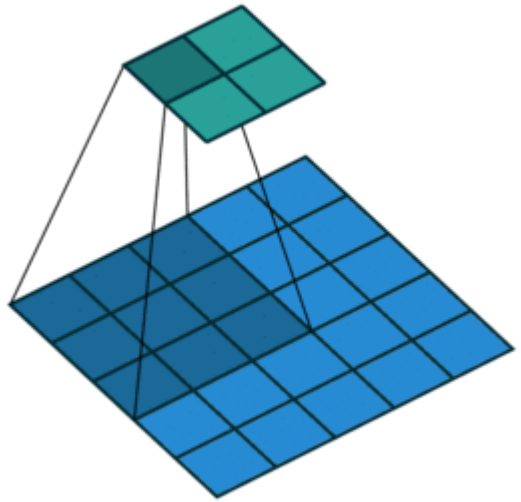
Convolution Layer

- **합성곱(Convolution)**
  - 스트라이드(Stride) : 2



Convolution Layer

- **합성곱(Convolution)**
  - 스트라이드(Stride) : 2



https://tensorflow.blog/a-guide-to-convolution-arithmetic-for-deep-learning/

Convolution Layer

- **합성곱(Convolution)**
  - Input : $N \times N$
  - Padding : P
  - Stride : S
  - Filter : $F \times F$
  - Output : $\left(\frac{N+2P-F}{S}+1\right) \times \left(\frac{N+2P-F}{S}+1\right)$

Convolution Layer

- **합성곱(Convolution)**
  - Input : $N \times N$
  - Padding : P
  - Stride : S
  - Filter : $F \times F$
  - Output : $\left(\frac{N+2P-F}{S}+1\right) \times \left(\frac{N+2P-F}{S}+1\right)$
  - 실습
    - [예제1] Input : (4,4), Padding : 1, Stride : 1, Filter : (3, 3)
    - [예제2] Input : (7,7), Padding : 0, Stride : 2, Filter : (3, 3)
    - [예제3] Input : (28, 28), Padding : 2, Stride : 3, Filter : (5,5)

Convolution Layer

- **합성곱(Convolution)**
  - Input : $N \times N$
  - Padding : P
  - Stride : S
  - Filter : $F \times F$
  - Output : $\left(\frac{N+2P-F}{S}+1\right) \times \left(\frac{N+2P-F}{S}+1\right)$
  - 실습
    - [예제1] Input : (4,4), Padding : 1, Stride : 1, Filter : (3, 3)
    - **[답1] (4,4)**
    - [예제2] Input : (7,7), Padding : 0, Stride : 2, Filter : (3, 3)
    - **[답2] (3,3)**
    - [예제3] Input : (28, 28), Padding : 2, Stride : 3, Filter : (5,5)
    - **[답3] (10,10)**

Convolution Layer

- 합성곱(Convolution)
  - Conv2d
    - 이미지 처리
    - Conv3d : 영상 처리
  - Kenel_size
  - Stride
  - Padding
  - ~~Dilation/Groups~~
  - Bias

class `torch.nn.Conv2d`(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True*) [source]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{out}, H_{out}, W_{out})$ can be precisely described as:

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k),$$

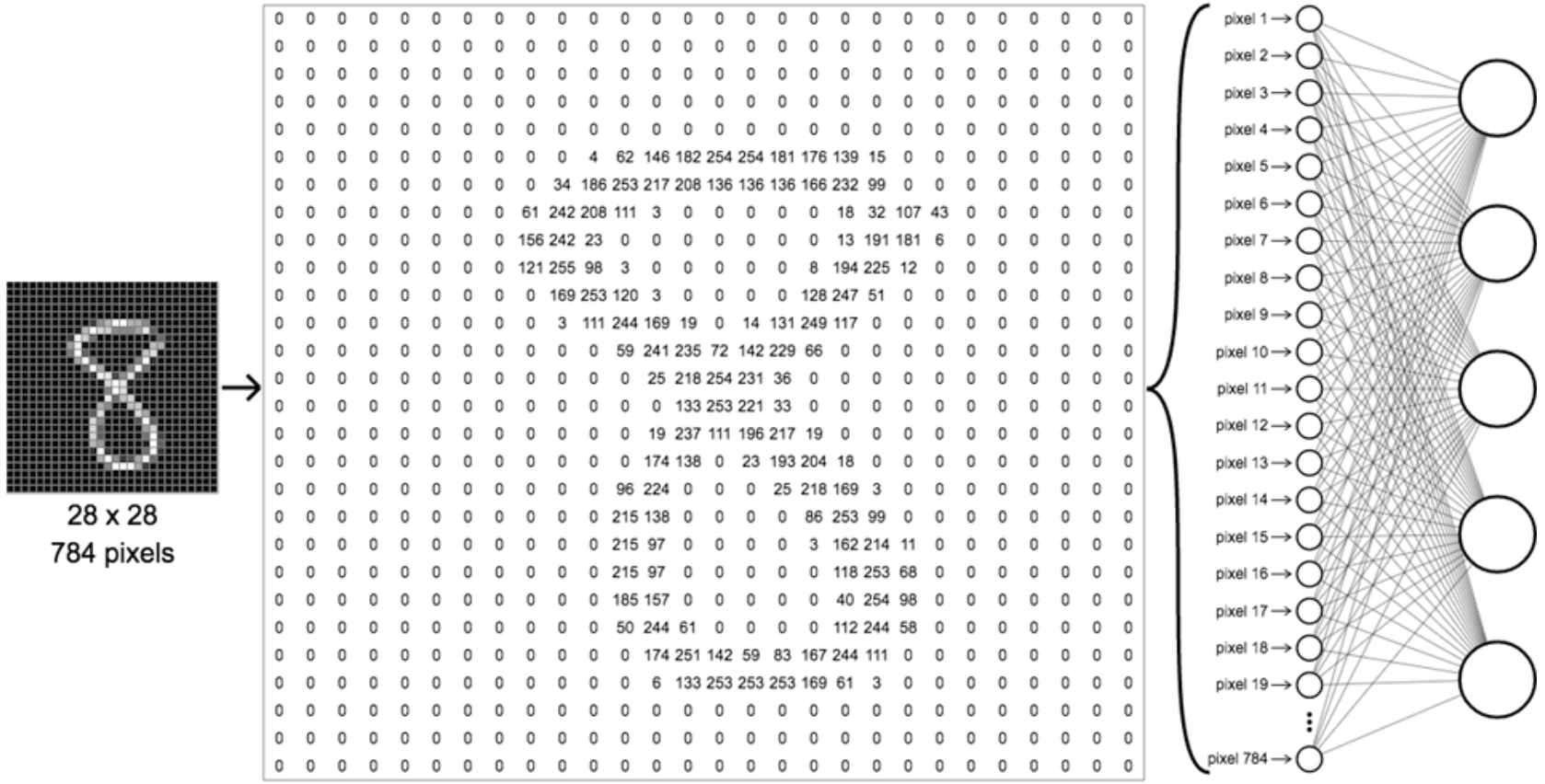
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,

3. GRAY/RGB

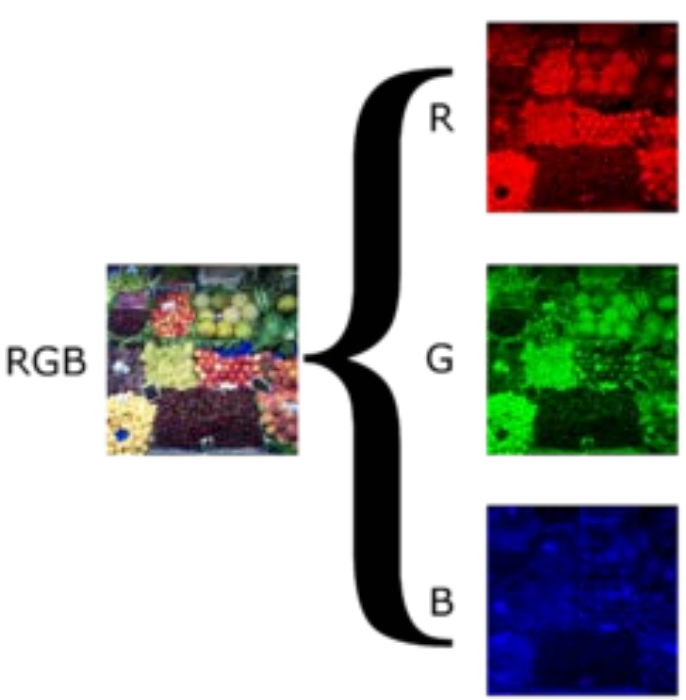
GRAY/RGB

- 합성곱(Convolution)
  - **GRAY**/RGB

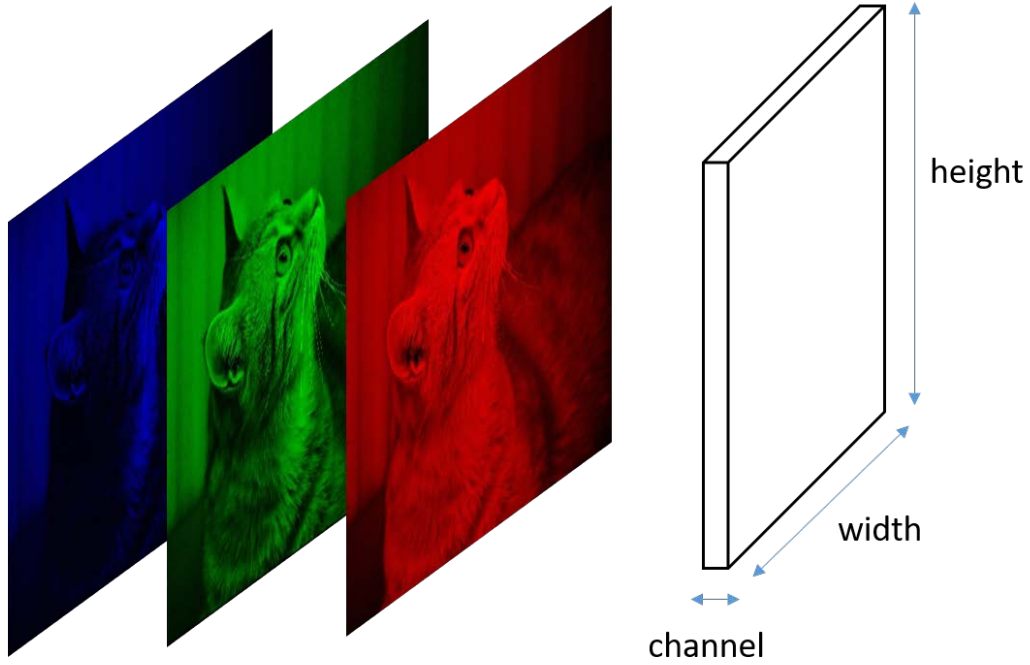


GRAY/RGB

- **합성곱(Convolution)**
  - GRAY**/RGB**



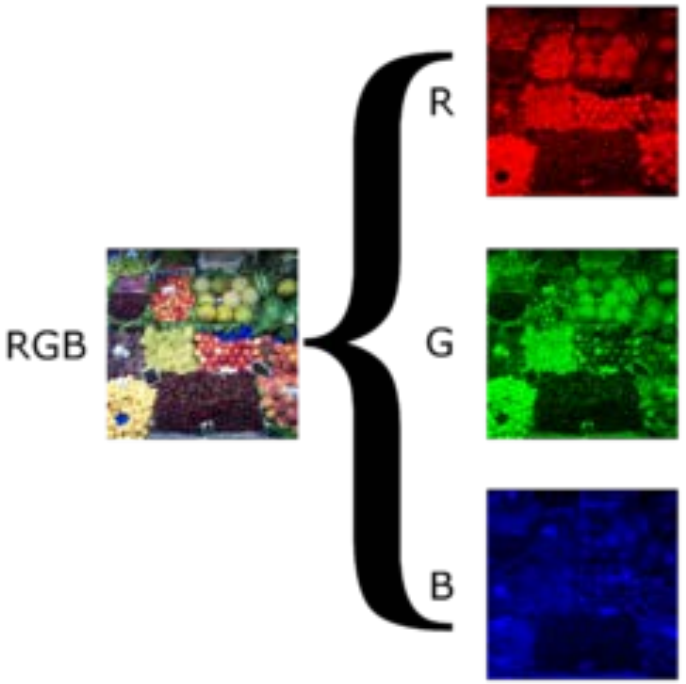
https://en.wikipedia.org/wiki/Grayscale



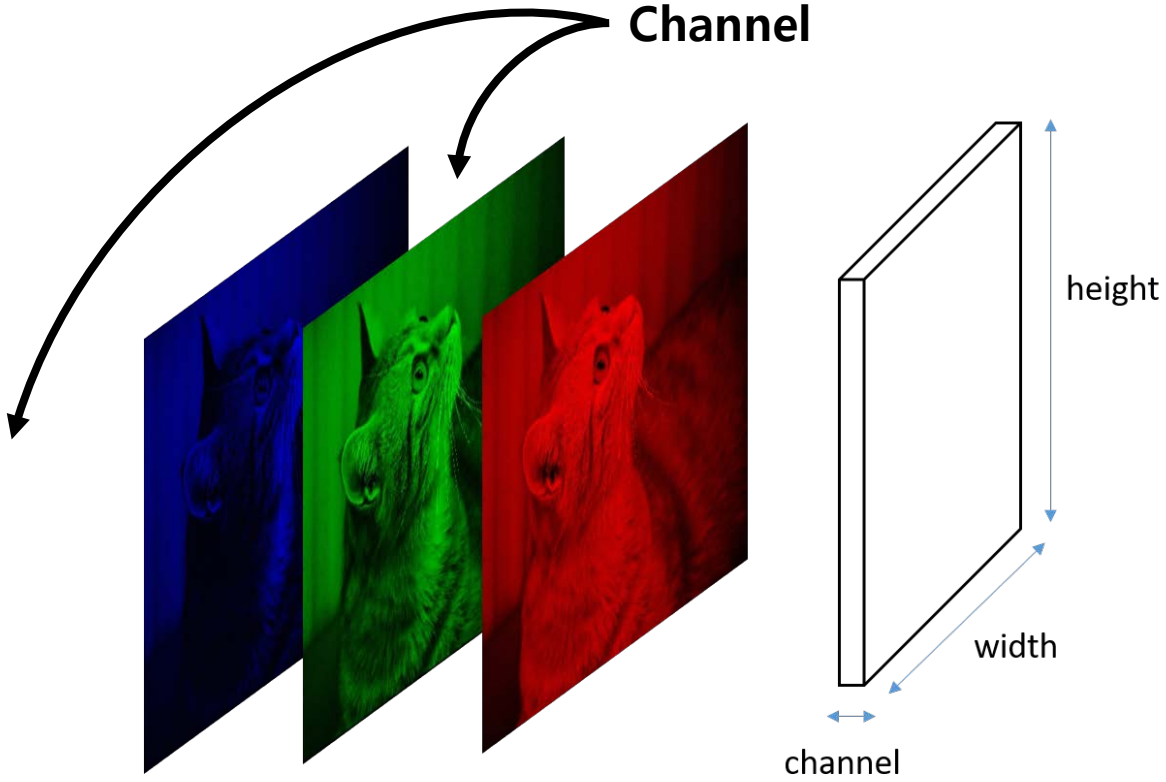
http://corochann.com/understanding-convolutional-layer-1227.html

GRAY/RGB

- **합성곱(Convolution)**
  - GRAY**/RGB**



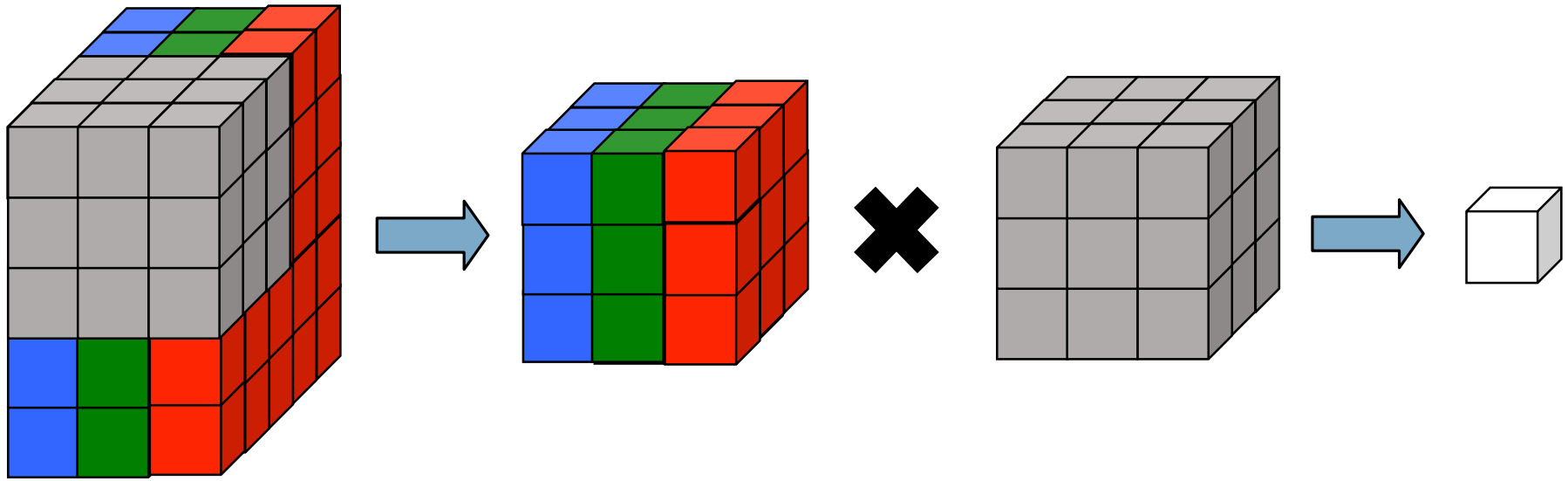
https://en.wikipedia.org/wiki/Grayscale



http://corochann.com/understanding-convolutional-layer-1227.html

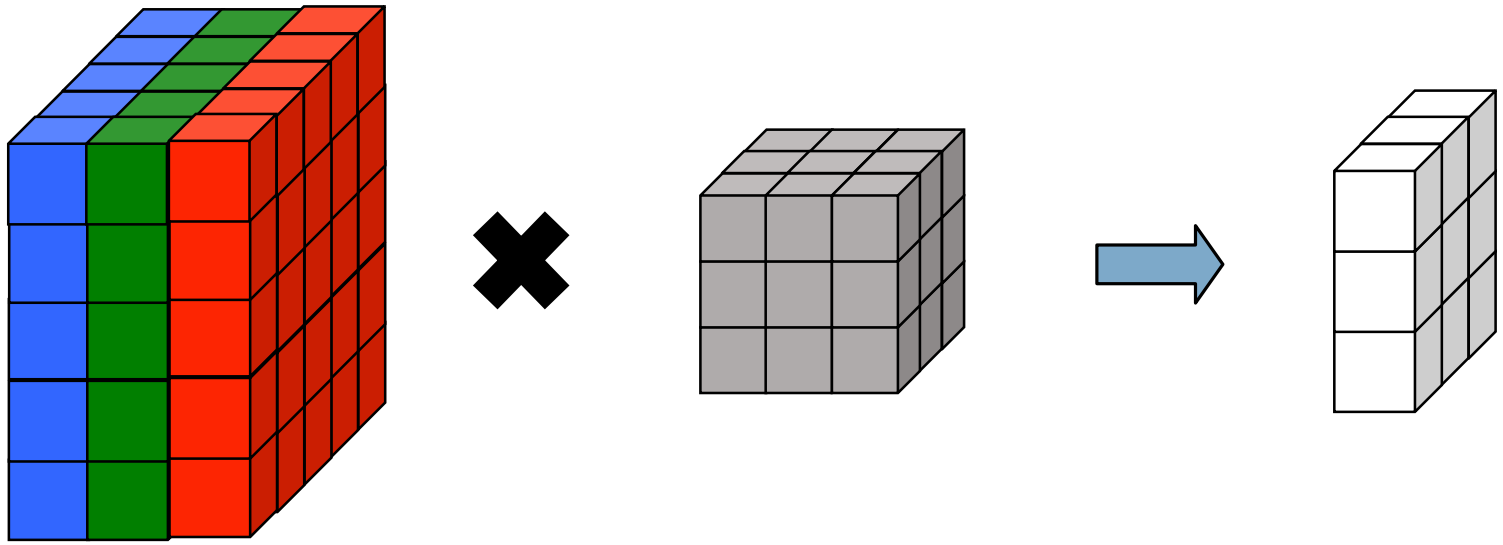
GRAY/RGB

- 합성곱(Convolution)
  - GRAY/**RGB**



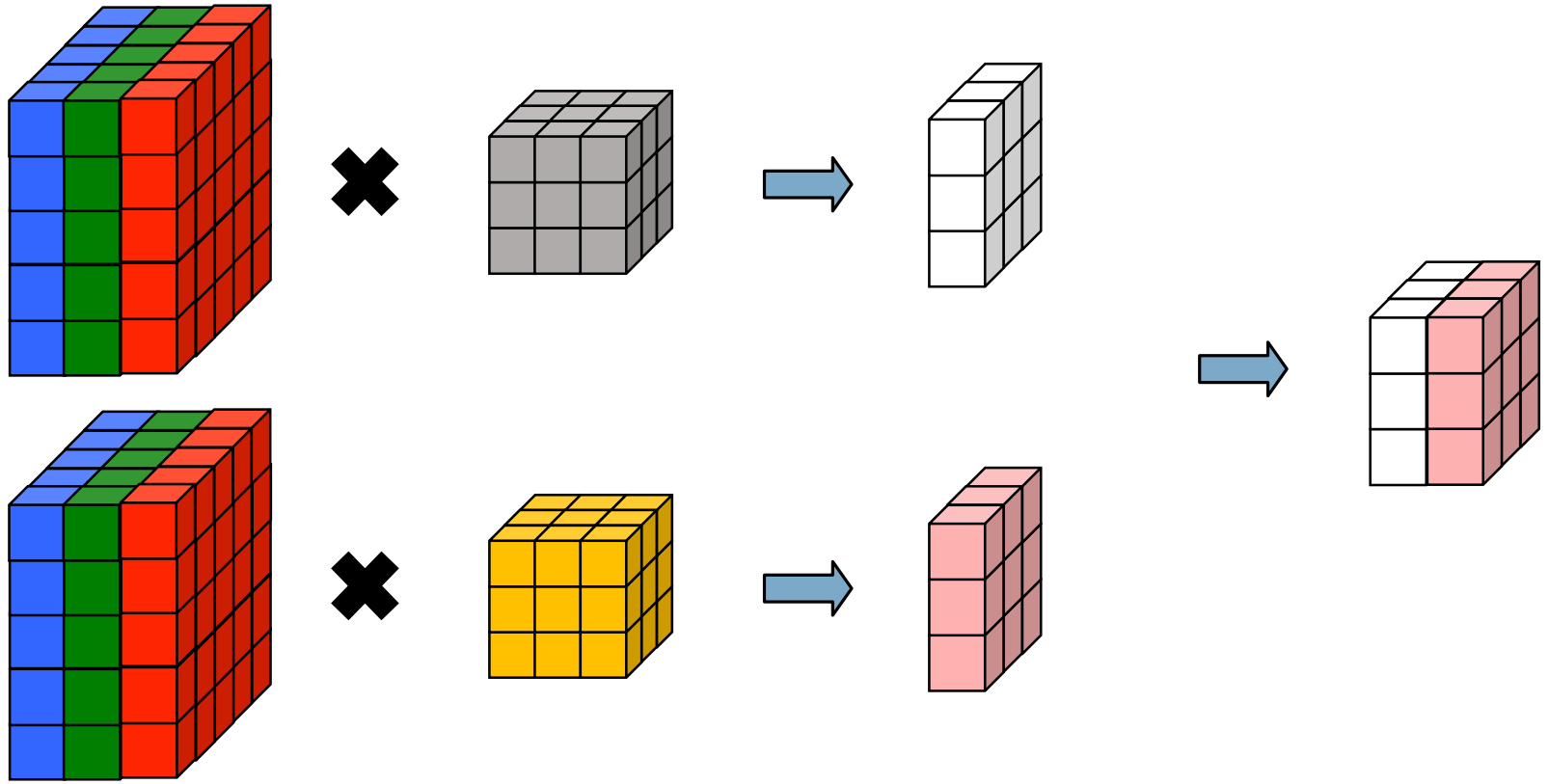
GRAY/RGB

- 합성곱(Convolution)
  - GRAY/**RGB**



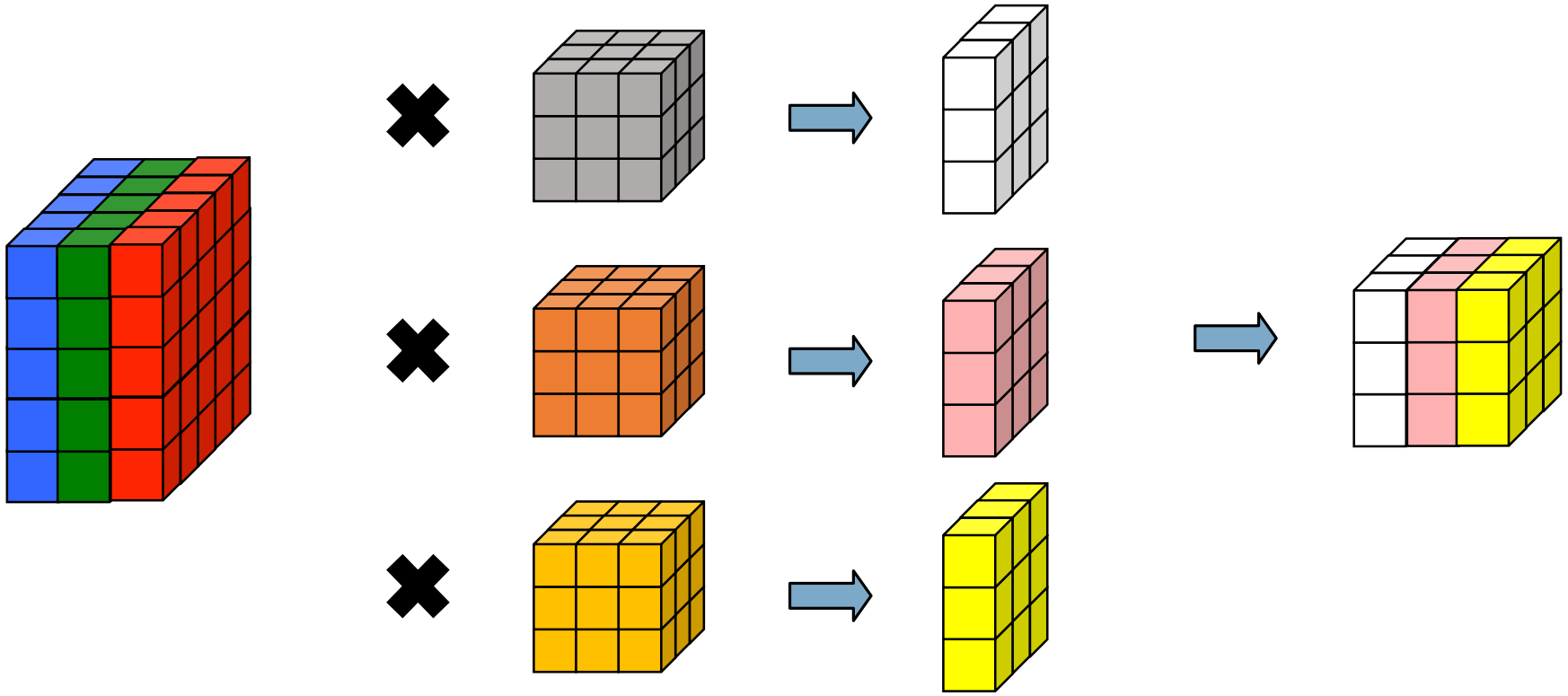
GRAY/RGB

- 합성곱(Convolution)
  - GRAY/**RGB**



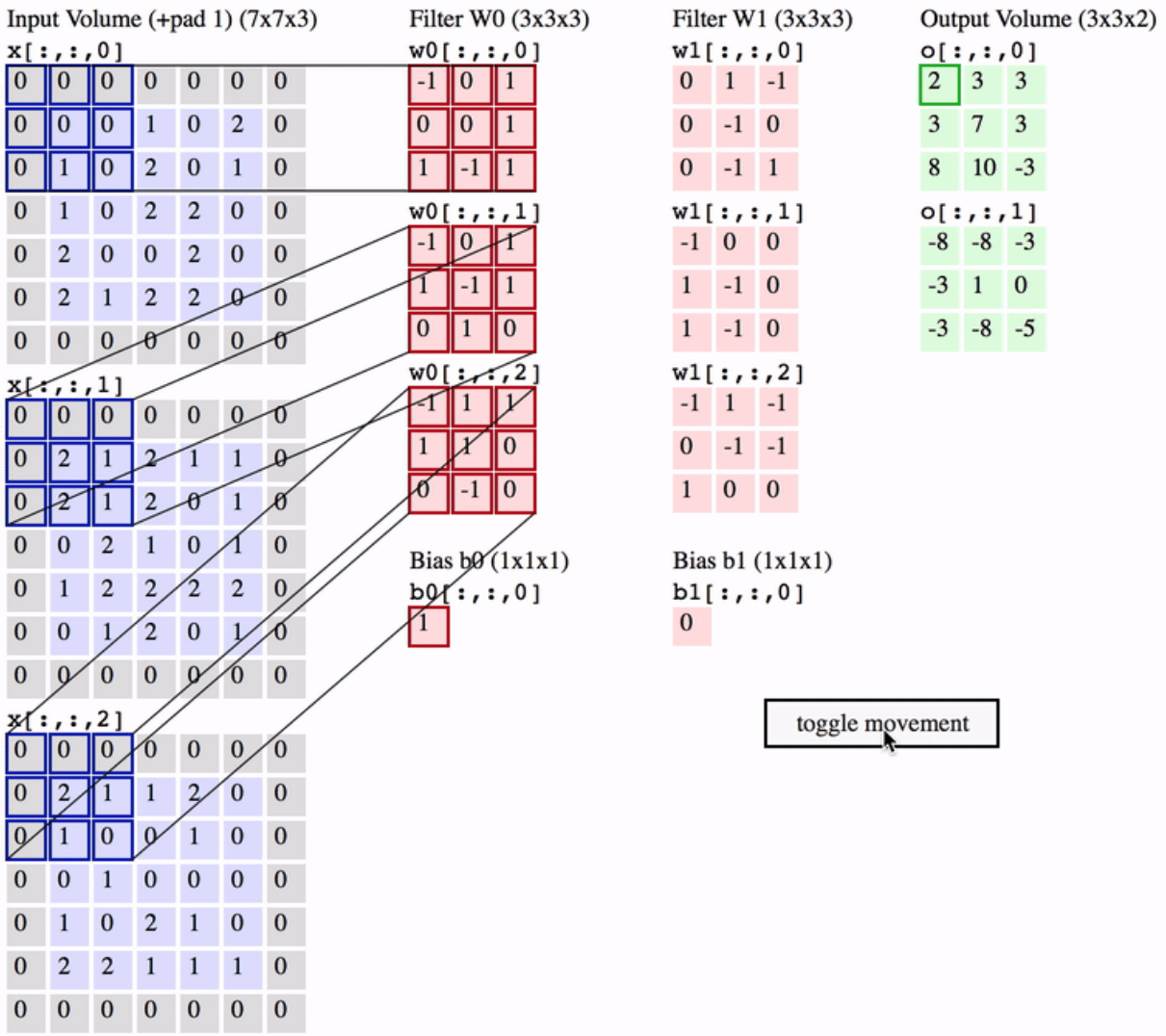
GRAY/RGB

- 합성곱(Convolution)
  - GRAY/**RGB**



GRAY/RGB

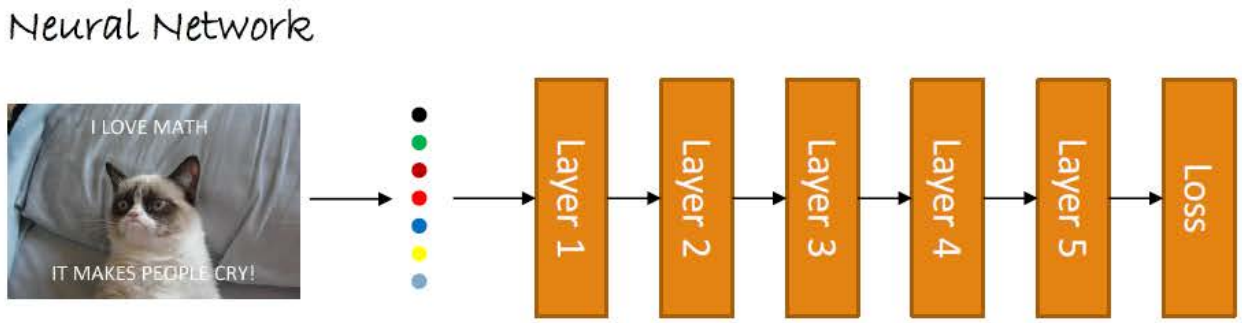
- 합성곱(Convolution)

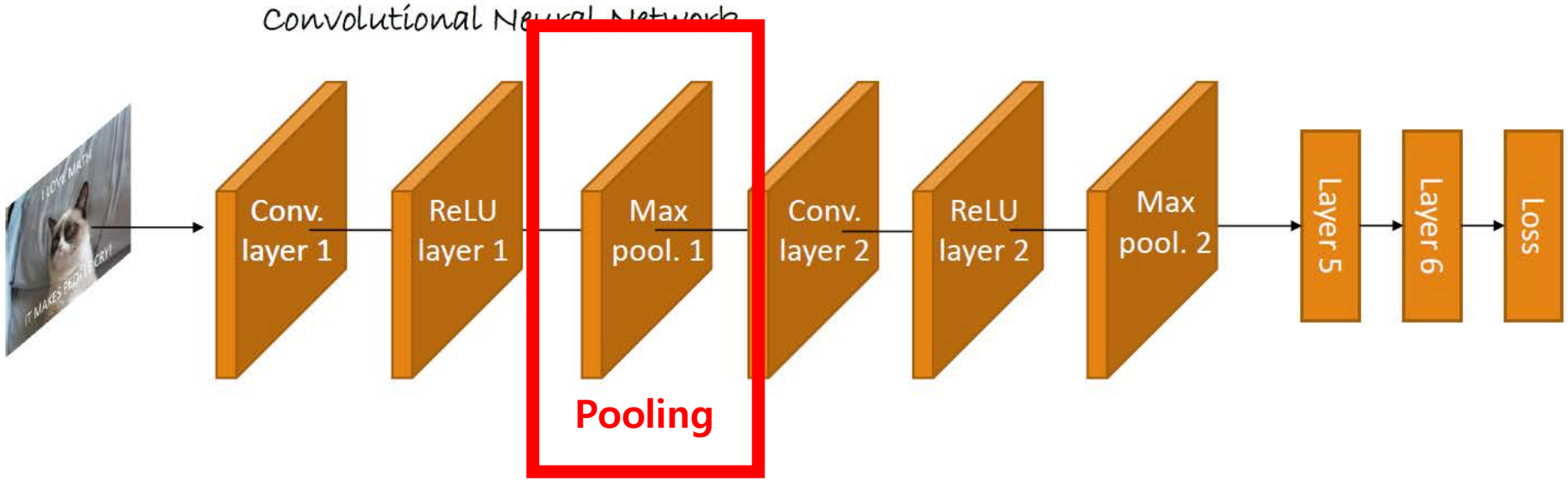


4. Pooling Layer

Pooling Layer

- 풀링(Pooling)





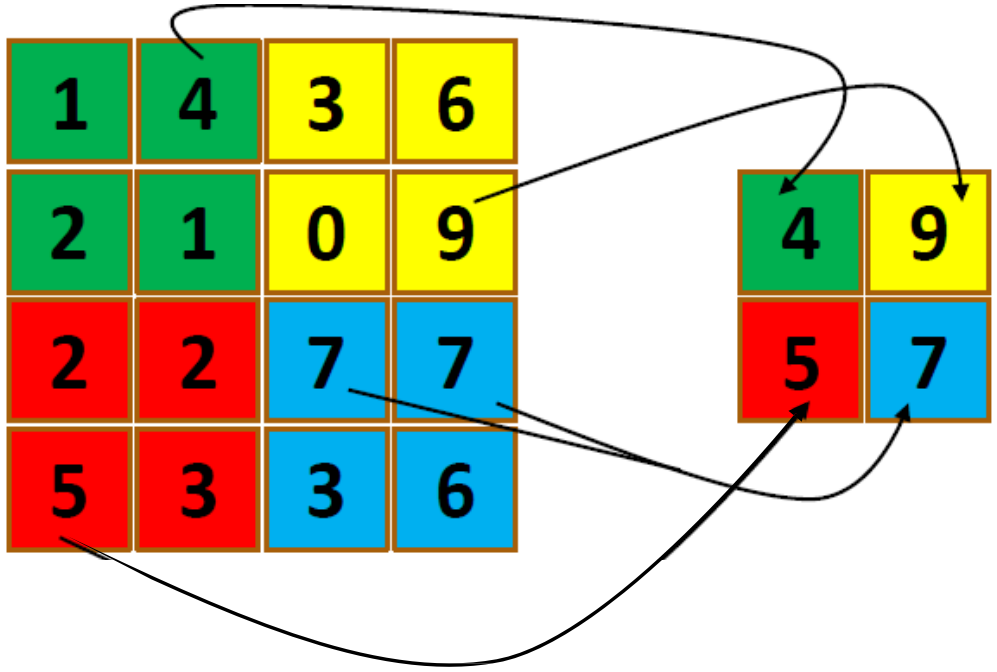
Fast campus

Pooling Layer

- **풀링(Pooling)**
  - 풀링(Pooling) : 여러 개의 변수로 하나로 묶는 작업
  - 풀링의 종류
    - **Max pooling**
    - Average pooling

Pooling Layer

- 풀링(Pooling)
  - 맥스 풀링(Max Pooling) : 가장 최대값을 추출
    - Size : 2x2, Stride : 2



Fast campus

Pooling Layer

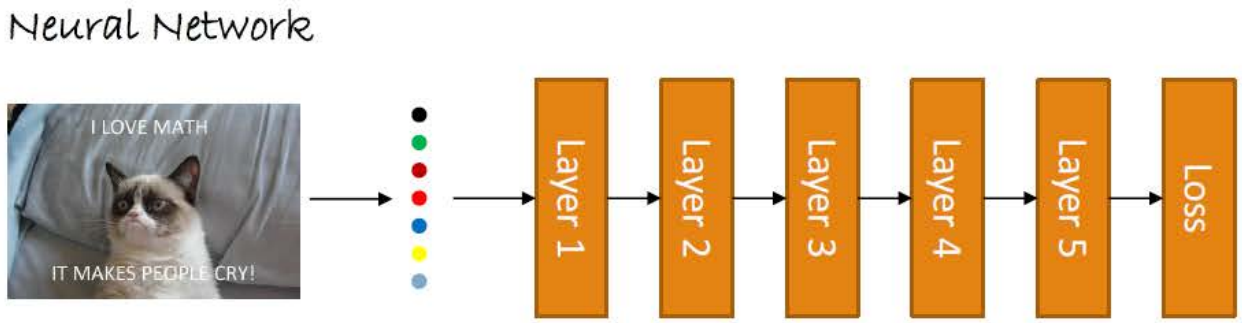
- **풀링(Pooling)**
  - 풀링의 의의
    - 작은 변화에 대해 불변
    - Layer의 사이즈를 줄여주므로 빠른 계산 가능
    - 가장 중요한 특성을 다음 특성으로 넘김

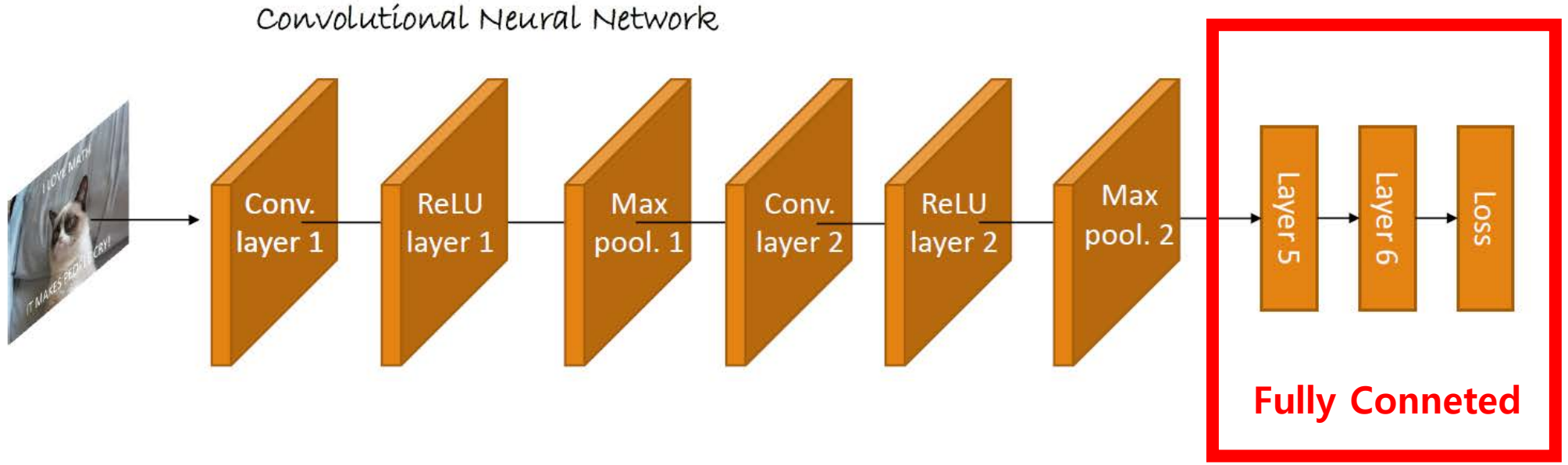
5. Fully Connected Layer

Fast campus

Fully Connected Layer

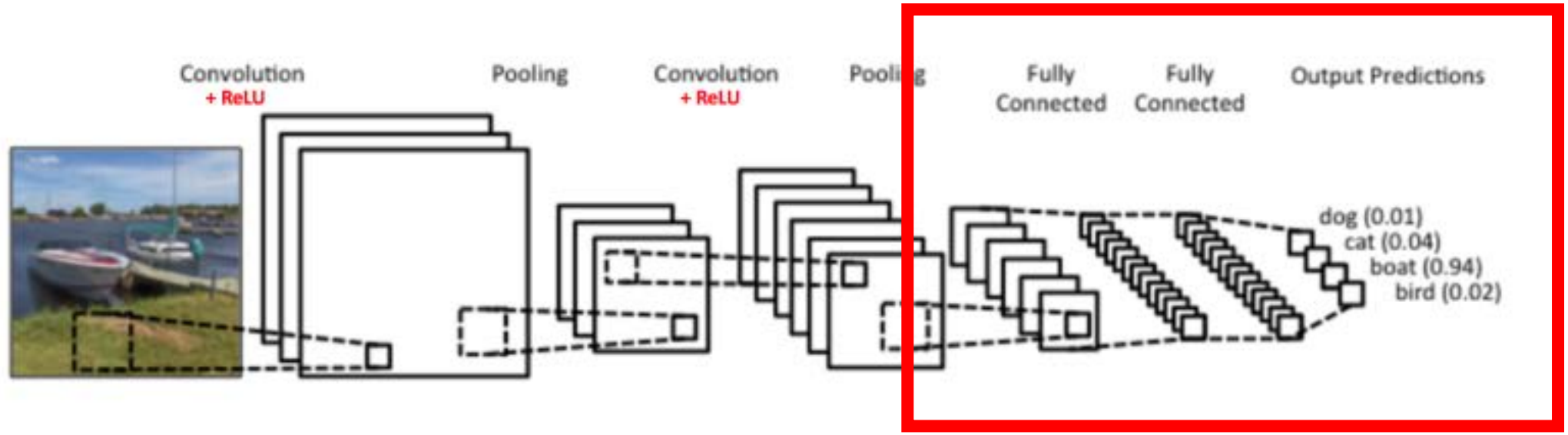
- **Fully Connect Layer (FC Layer)**





Fully Connected Layer

- **Fully Connect Layer (FC Layer)**
  - 맨 마지막 계층을 쫙 편 뒤에, NN을 사용하여 분류



  - 예를 들어, 맨 마지막에 Size : 3x3, Filter : 64의 결과가 나왔다면,
  - Linear(64*3*3, 100)을 활용하여 100개의 Fully Connected로 피게 됨.

실 습