CNN Introduction 1

http://taewan.kim/post/cnn/

CNN의 필요 이유

- ✓ Fully Connected Layer : 입력 데이터는 1차원(배열) 형태
- ✓ 한 장의 컬러 사진은 3차원 데이터입니다.
- ✓ 배치 모드에 사용되는 여러 장의 사진 : 4차원 데이터
- ✓ 사진 데이터로 Fully Connected 신경망을 학습시켜야 할 경우에, 3차원 사진 데이터를 1차원으로 평면화시켜야 함.
  - 평면화 공간 정보 손실 -이미지 공간 정보 유실로 정보 부족발생 - 특징 추출 및 학습이 비효율적이고 정확도를 높이는데 한계가 있음
- ✓ CNN(Convolutional Neural Network)
  - 이미지의 공간 정보를 유지한 상태로 학습이 가능한 모델

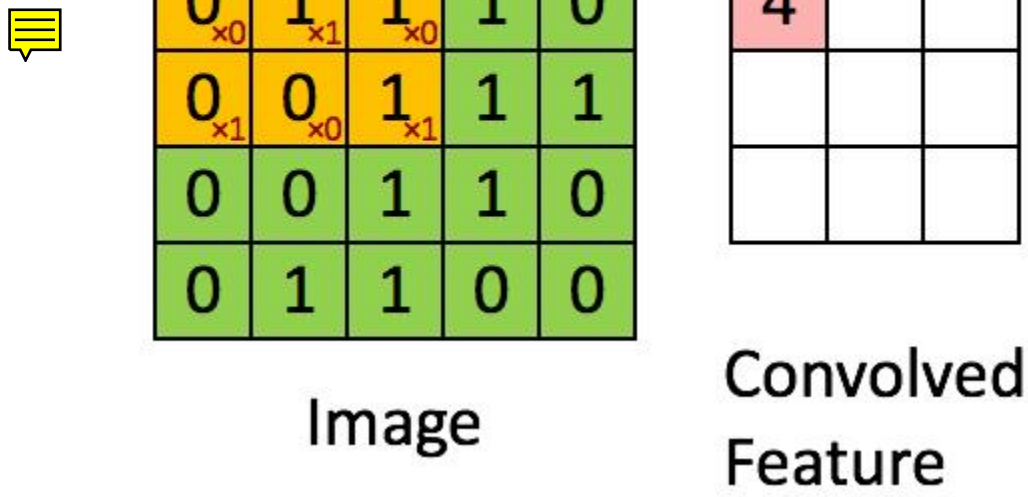
CNN 사용되는 용어

Convolution(합성곱)
채널(Channel)
필터(Filter)
커널(Kernel)
스트라이드(Strid)
패딩(Padding)
피처 맵(Feature Map)
액티베이션 맵(Activation Map)
풀링(Pooling) 레이어

CNN에는 다음과 같은 용어들이 사용됩니다

합성곱 연산은 두 함수 **f, g** 가운데 하나의 함수를 반전**(reverse)**, 전이**(shift)**시킨 다음, 다른 하나의 함수와 곱한 결과를 적분하는 것을 의미한다.
출처: https://ko.wikipedia.org/wiki/%ED%95%A9%EC%84%B1%EA%B3%B1

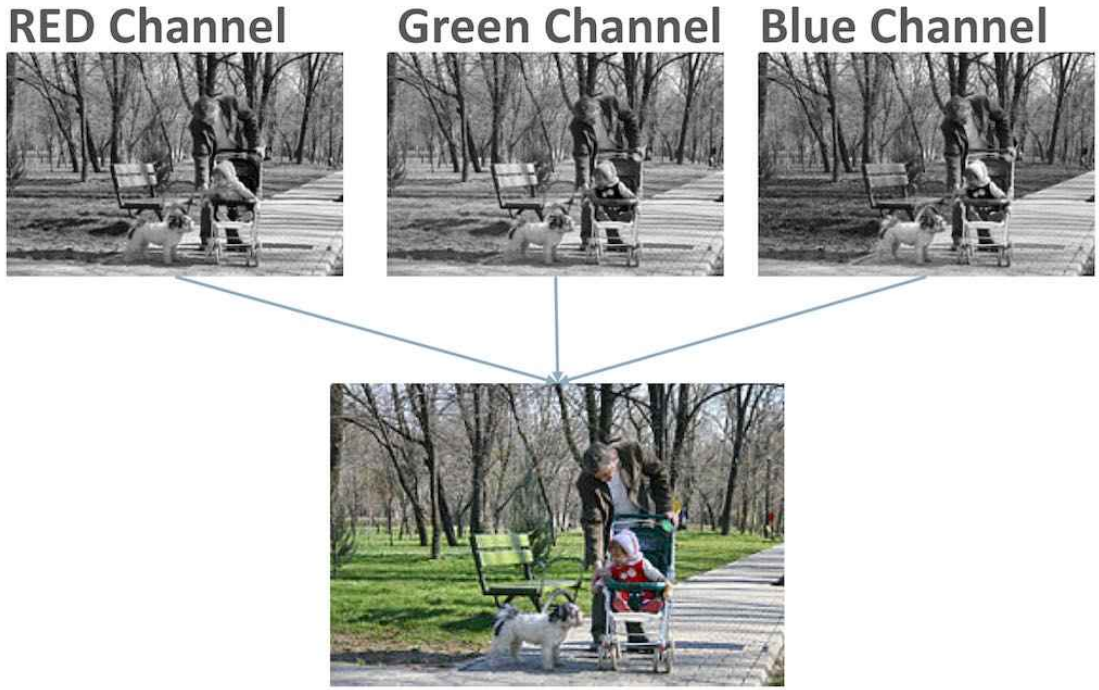
- 2차원 입력데이터(Shape: (5,5))를 1개의 필터로 합성곱 연산 수행
- 합성곱 처리 결과로 부터 Feature Map을 만듬.



http://deeplearning.stanford.edu/wiki/images/6/6c/Convolution_schematic.gif

채널, Channel

이미지 픽셀 하나하나는 실수
컬러 사진은 천연색을 표현하기 위해서, 각 픽셀을 RGB 3개의 실수로 표현한 3차원 데이터
컬러 이미지는 3개의 채널로 구성됨.
흑백 명암만을 표현하는 흑백 사진은 2차원 데이터로 1개 채널로 구성됨.
높이가 39 픽셀이고 폭이 31 픽셀인 컬러 사진 데이터의 shape은 (39, 31, 3)으로 표현됨.
반면에 높이가 39픽셀이고 폭이 31픽셀인 흑백 사진 데이터의 shape은 (39, 31, 1).



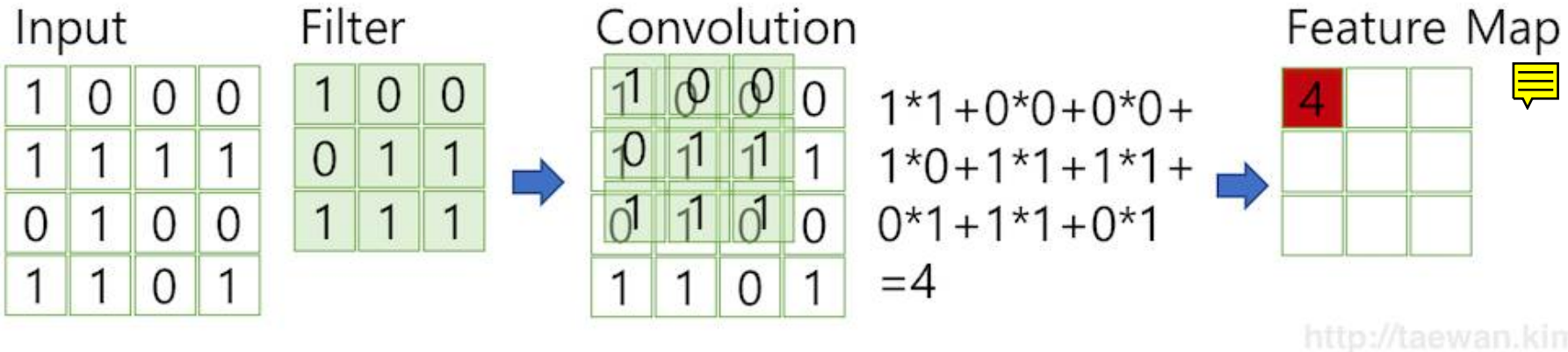
이미지 출처: https://en.wikipedia.org/wiki/Channel_(digital_image)

Convolution Layer에 유입되는 입력 데이터에는 한 개 이상의 필터가 적용됨.
1개 필터는 Feature Map의 채널이 됨.
Convolution Layer에 **n개**의 필터가 적용된다면 출력 데이터는 **n개**의 채널을 갖게 됨.

필터(Filter) & Stride

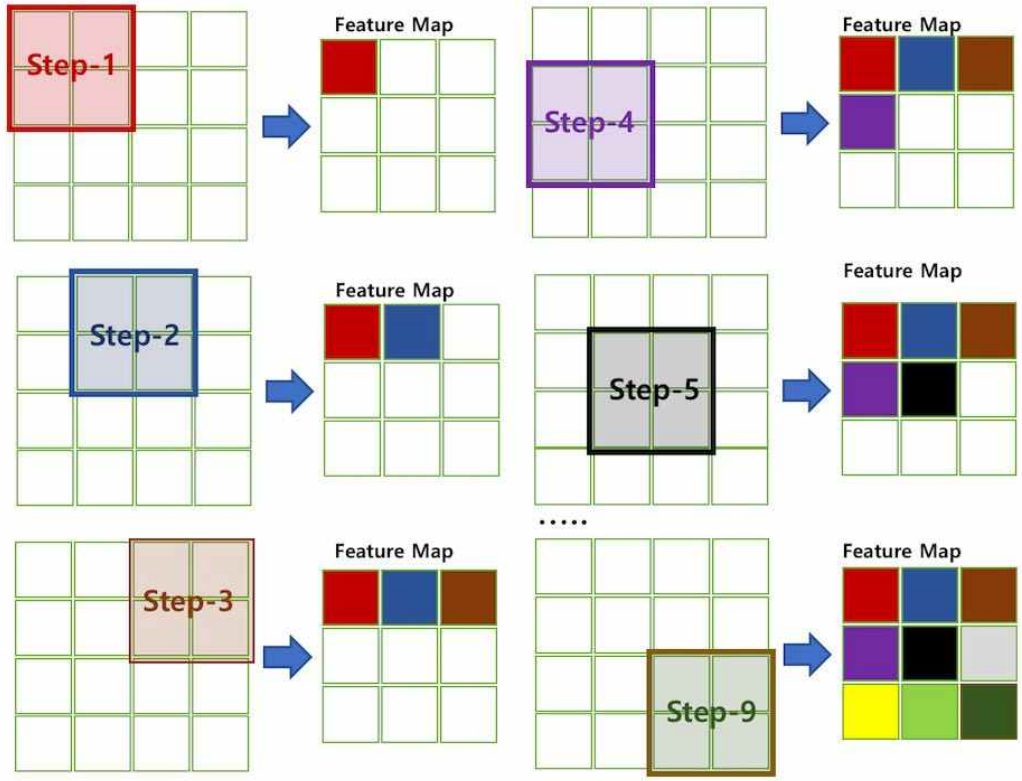
필터 : 이미지의 특징을 찾아내기 위한 공용 파라미터
Filter를 Kernel이라고 하기도 합
CNN에서 Filter와 Kernel은 같은 의미임
필터는 일반적으로 (4, 4)이나 (3, 3)과 같은 정사각 행렬로 정의됨
CNN에서 학습의 대상은 필터 파라미터 임

입력 데이터를 지정된 간격으로 순회하며 채널별로 합성곱을 하고 모든 채널(컬러의 경우 3개)의 합성곱의 합을 Feature Map로 만듬.
필터는 지정된 간격으로 이동하면서 전체 입력데이터와 합성곱하여 Feature Map을 만듬. <그림 3>은 채널이 1개인 입력 데이터를 (3, 3) 크기의 필터로 합성곱하는 과정을 설명합니다.

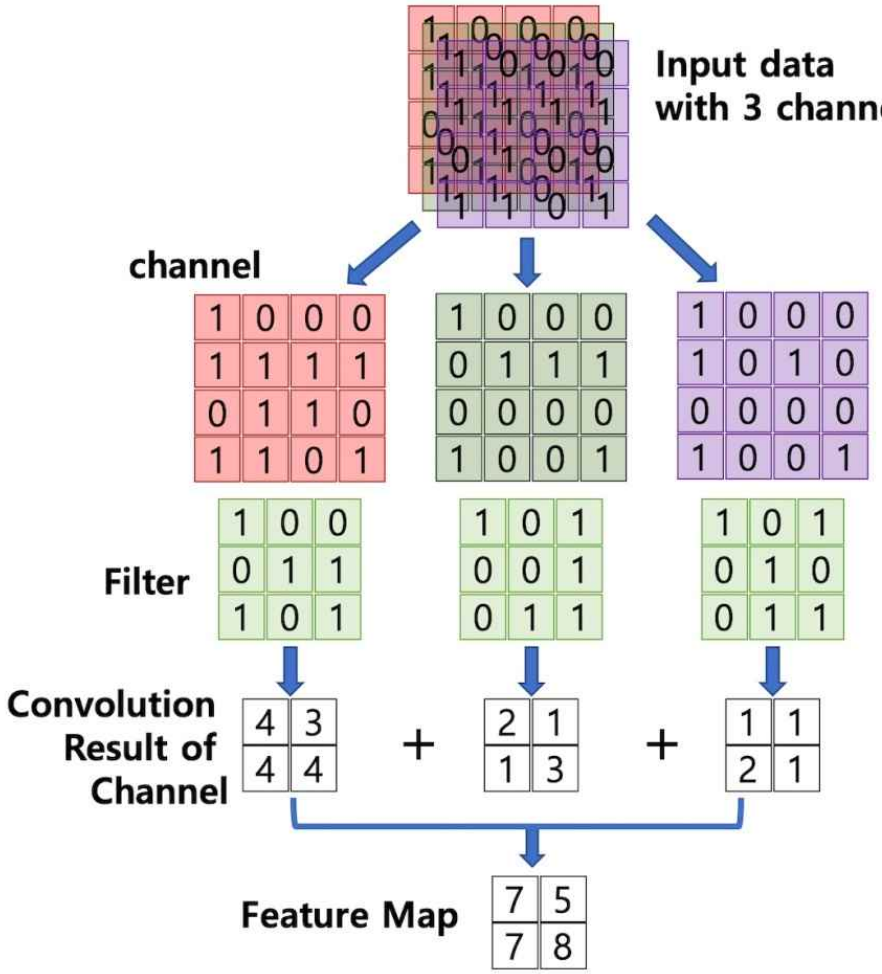


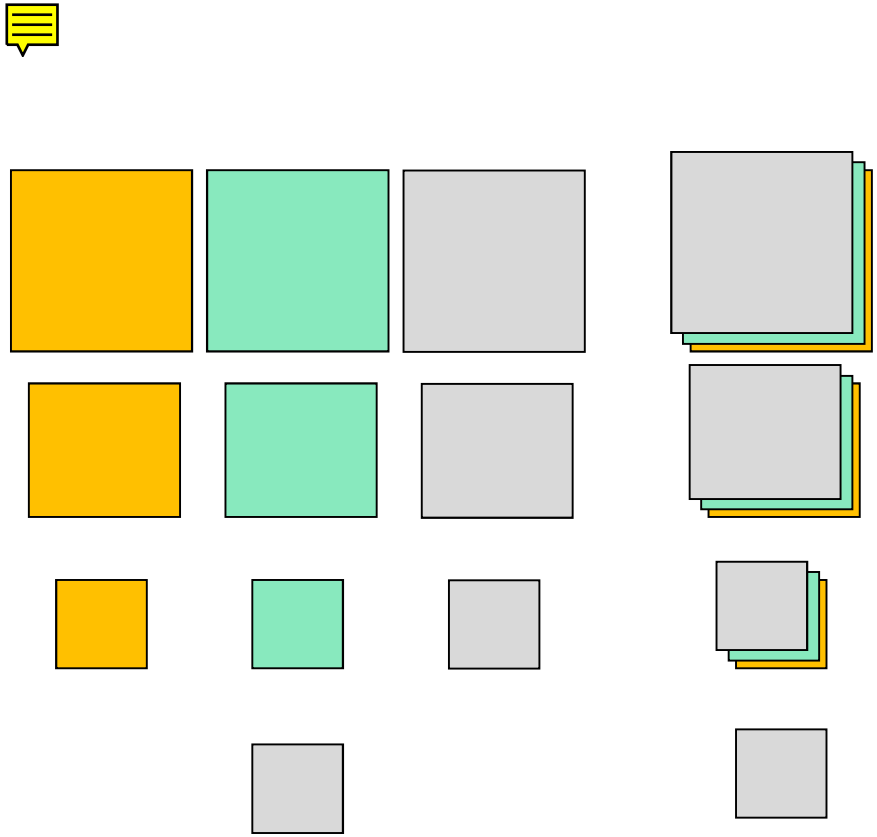
필터(Filter) & Stride

필터는 입력 데이터를 지정한 간격으로 순회하면서 합성곱을 계산합니다. 여기서 지정된 간격으로 필터를 순회하는 간격을 Stride라고 합니다. <그림 4>는 strid가 1로 필터를 입력 데이터에 순회하는 예제입니다. strid가 2로 설정되면 필터는 2칸씩 이동하면서 합성곱을 계산합니다.



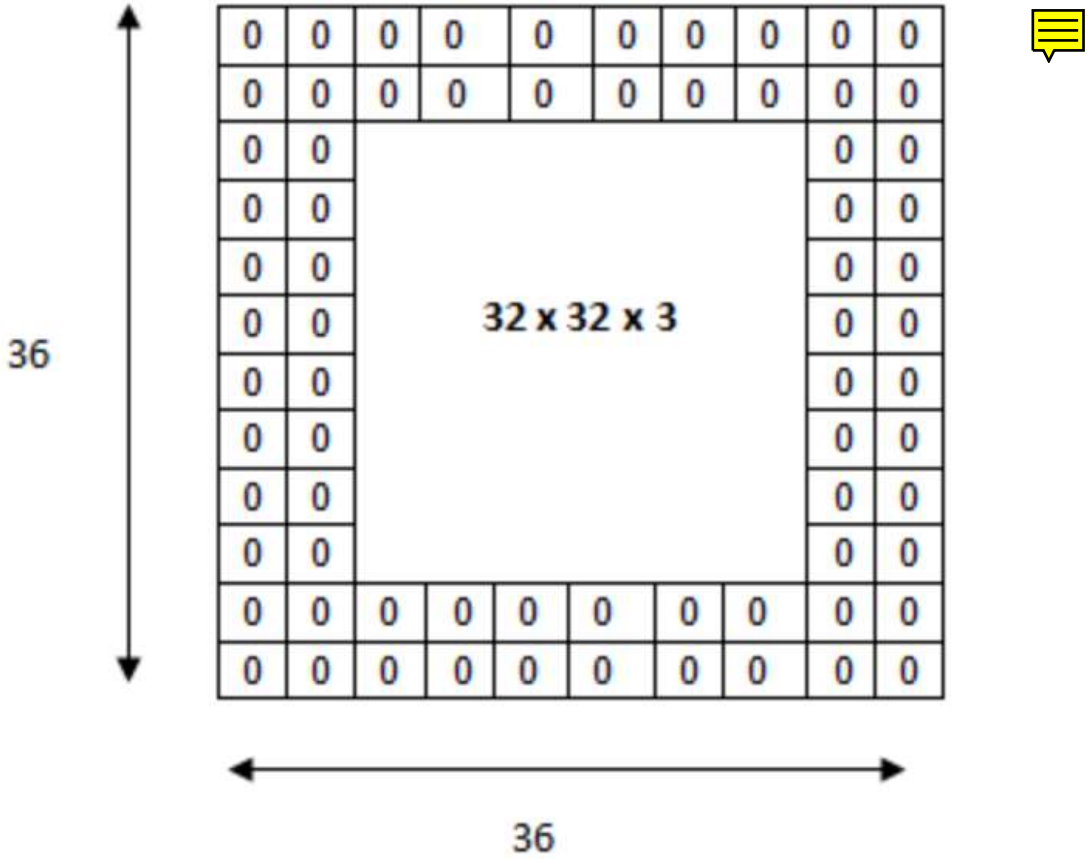
입력 데이터가 여러 채널을 갖을 경우 필터는 각 채널을 순회하며 합성곱을 계산한 후, 채널별 피처 맵을 만듬
각 채널의 피처 맵을 합산하여 최종 피처 맵으로 반환
입력 데이터는 채널 수와 상관없이 필터 별로 1개의 피처 맵이 만들어 짐





패딩(Padding)

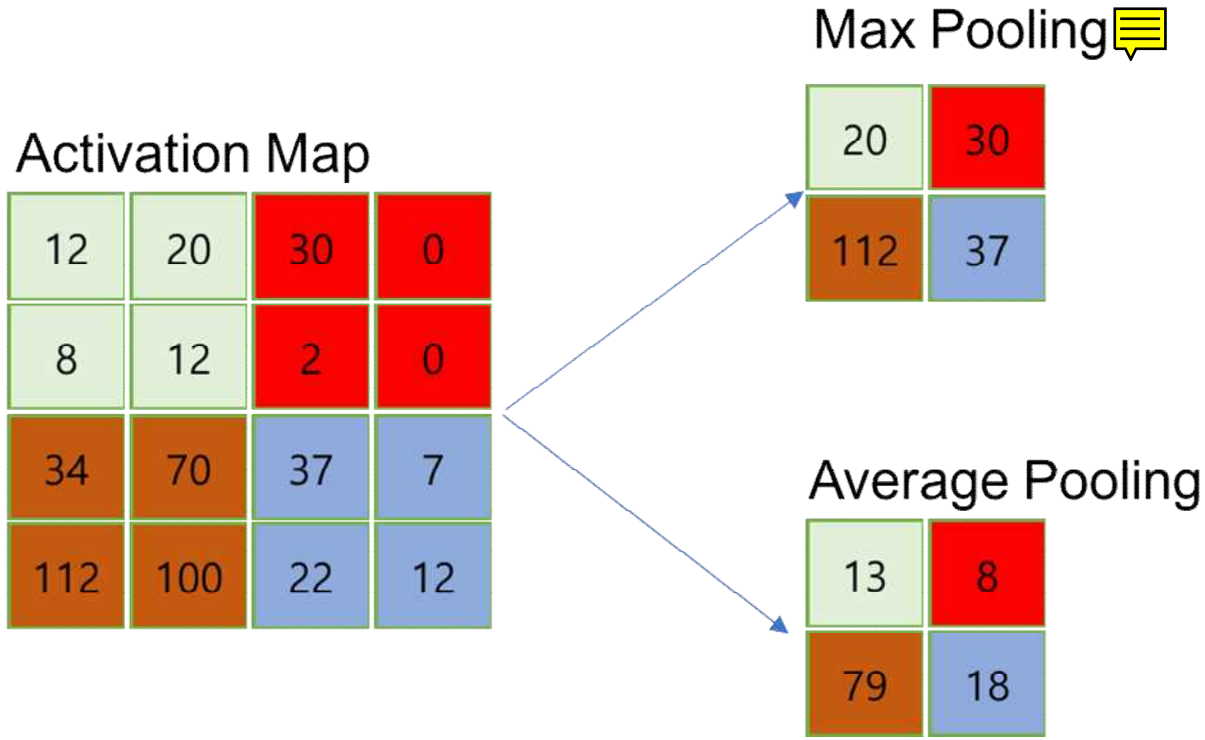
Convolution 레이어에서 Filter와 Stride에 작용으로 Feature Map 크기는 입력데이터 보다 작습니다. Convolution 레이어의 출력 데이터가 줄어드는 것을 방지하는 방법이 패딩입니다. 패딩은 입력 데이터의 외각에 지정된 픽셀만큼 특정 값으로 채워 넣는 것을 의미합니다. 보통 패딩 값으로 0으로 채워 넣습니다.



<그림 6>은 (32, 32, 3) 데이터를 외각에 2 pixel을 추가하여 (36, 36, 3) 행렬을 만드는 예제입니다. Padding을 통해서 Convolution 레이어의 출력 데이터의 사이즈를 조절하는 기능이 외에, 외각을 "0"값으로 둘러싸는 특징으로 부터 인공 신경망이 이미지의 외각을 인식하는 학습 효과도 있습니다.

Pooling 레이어

풀링 레이어는 컨볼류션 레이어의 출력 데이터를 입력으로 받아서 출력 데이터(Activation Map)의 크기를 줄이거나 특정 데이터를 강조하는 용도로 사용됨
처리하는 방법으로는 Max Pooling과 Average Pooning, Min Pooling이 있음
정사각 행렬의 특정 영역 안에 값의 최댓값을 모으거나 특정 영역의 평균을 구하는 방식으로 동작함
Pooing 크기와 Stride를 같은 크기로 설정하여 모든 원소가 한 번씩 처리 되도록 설정함

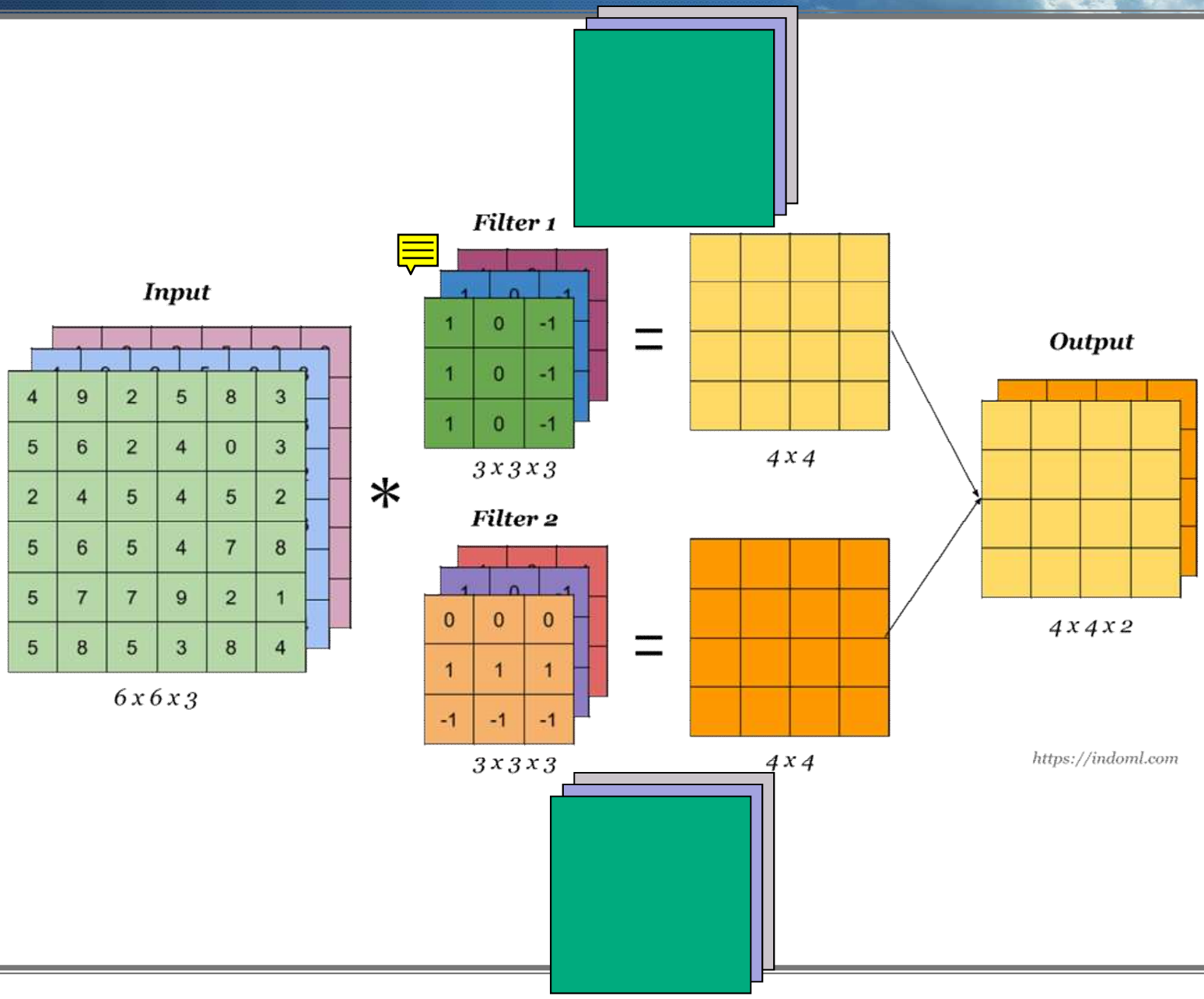


Pooling 레이어는 Convolution 레이어와 비교하여 다음과 같은 특징이 있음
- 학습대상 파라미터가 없음
- Pooling 레이어를 통과하면 행렬의 크기 감소
- Pooling 레이어를 통해서 채널 수 변경 없음
- CNN에서는 주로 Max Pooling을 사용합니다.

CNN Introduction 2

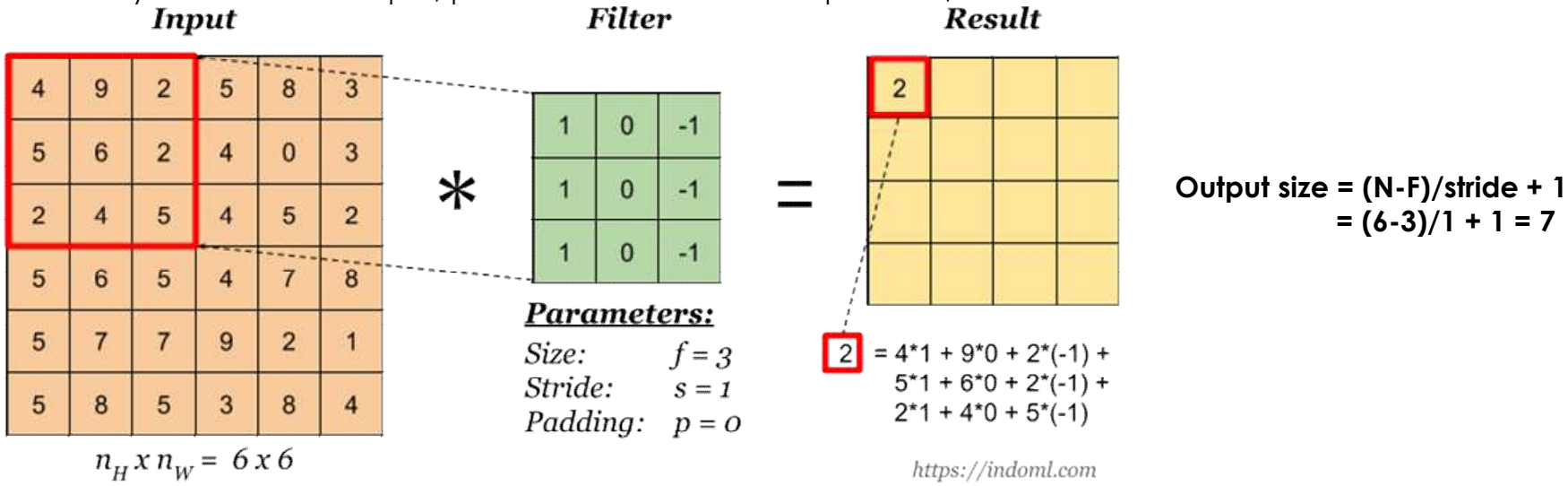
https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

Basic Convolution Operation



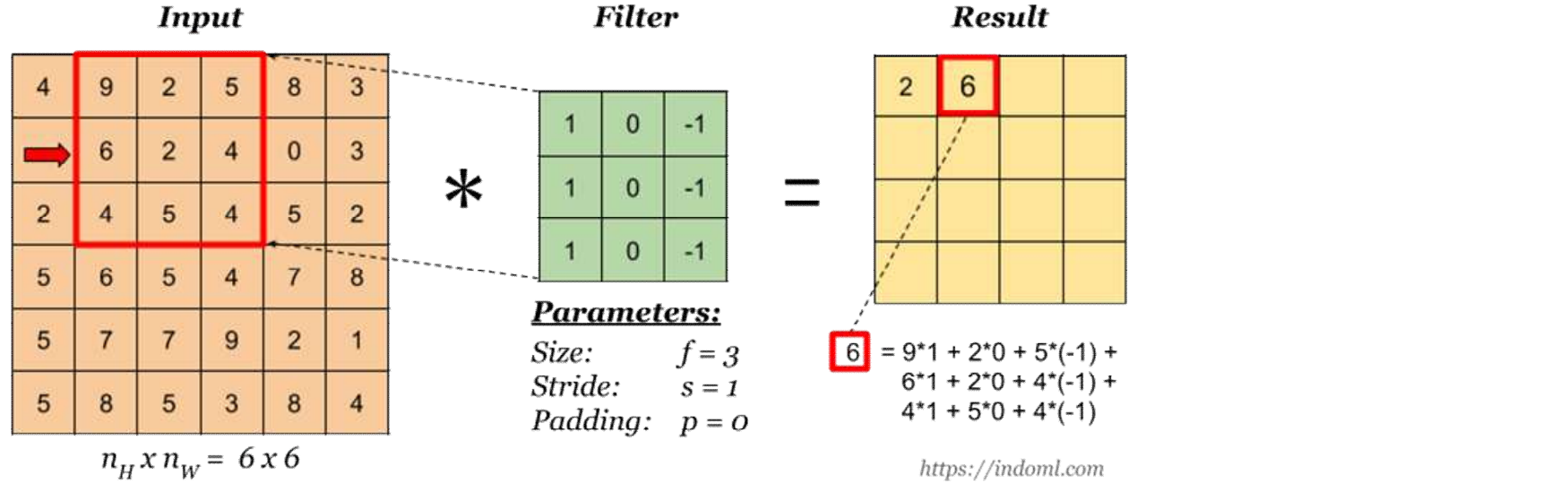
Basic Convolution Operation

Step 1: overlay the filter to the input, perform element wise multiplication, and add the result.



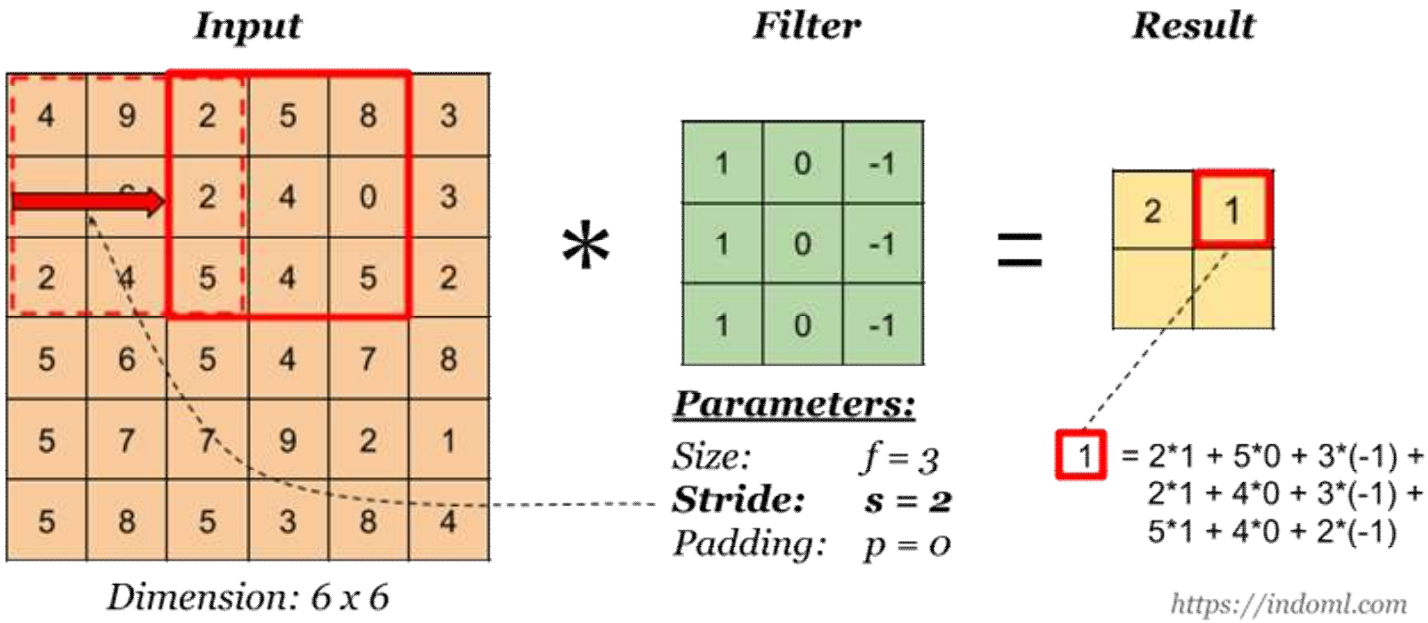
Output size = (N-F)/stride + 1
= (6-3)/1 + 1 = 7

Step 2: move the overlay right one position (or according to the **stride** setting), and do the same calculation above to get the next result. And so on.



Stride

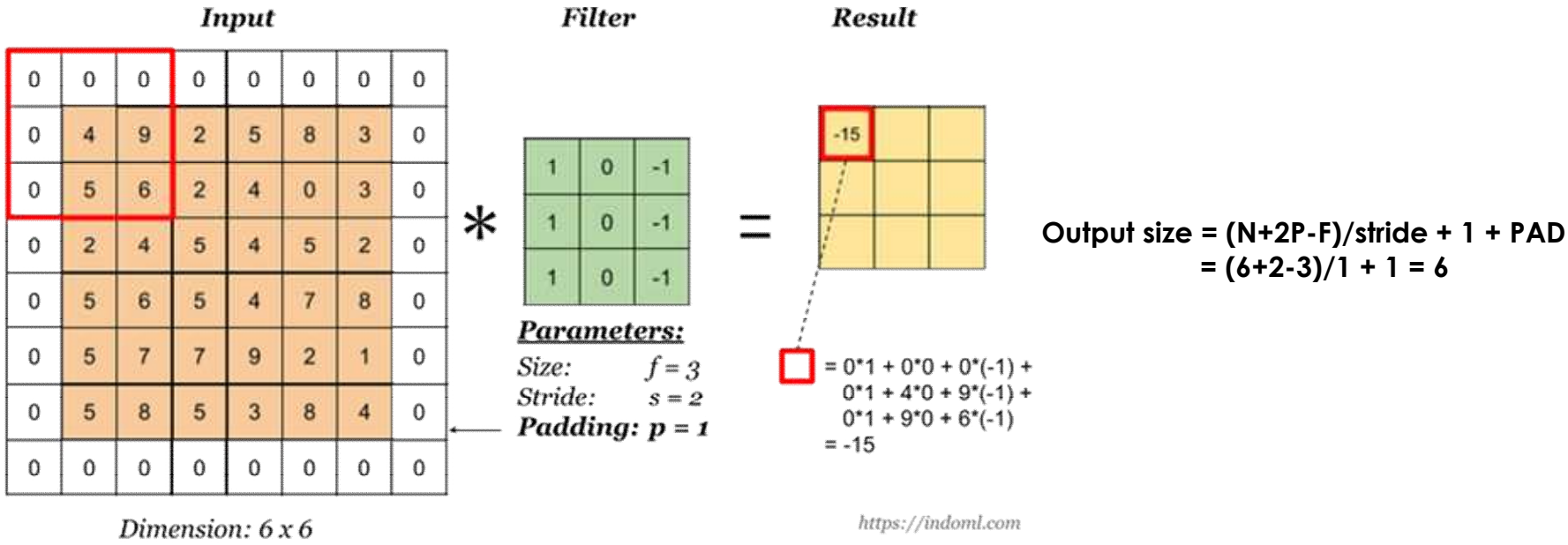
Stride governs how many cells the filter is moved in the input to calculate the next cell in the result.



The total number of multiplications to calculate the result above is (2 x 2) x (3 x 3) = 36.

Padding

Padding has the following benefits:
It allows us to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.
It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.



Notice the the dimension of the result has changed due to padding. See the following section on how to calculate output dimension.
Some padding terminologies:
"**valid**" padding: no padding
"**same**" padding: padding so that the output dimension is the same as the input

Calculating the Output Dimension

The output dimension is calculated with the following formula:

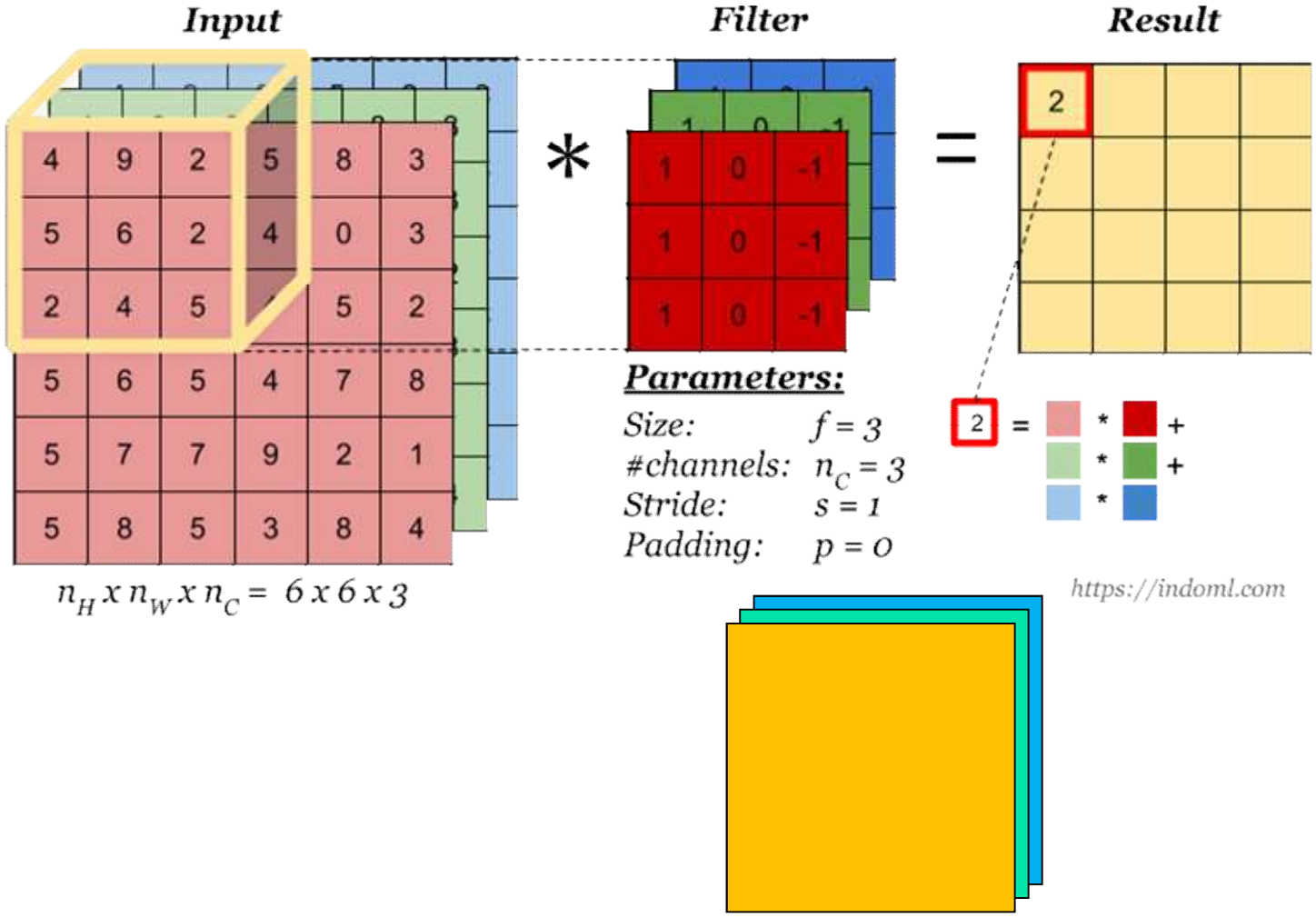
$$n^{[l]} = \lfloor \frac{n^{[l-1]} + 2p^{[l-1]} - f^{[l]}}{s^{[l]}} + 1 \rfloor$$

where the $\lfloor \rfloor$ symbols denote *math.floor()* operation.

Convolution Operation on Volume

When the input has more than one channels (e.g. an RGB image), the filter should have matching number of channels. To calculate one output cell, perform convolution on each matching channel, then add the result together.

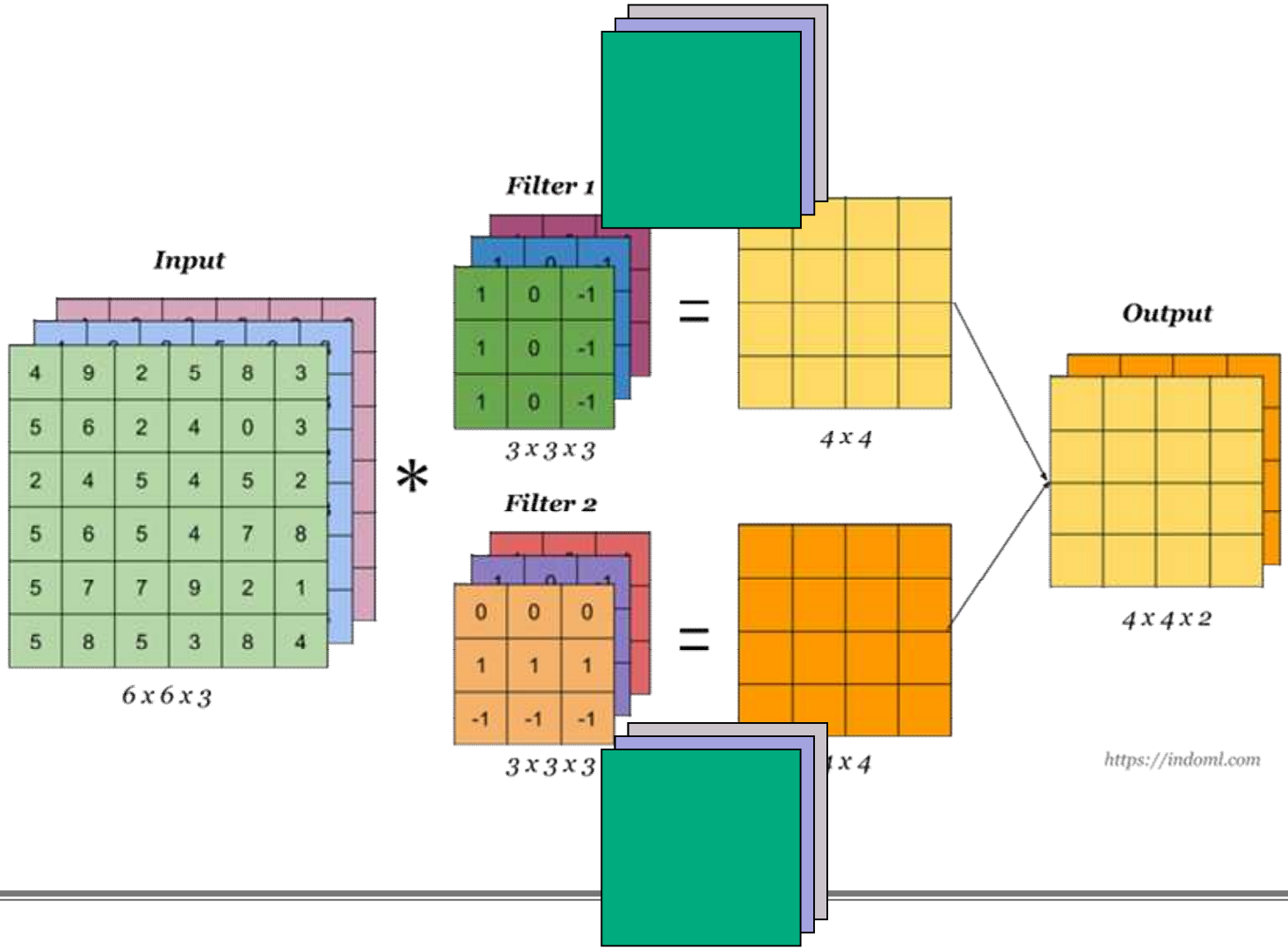
The total number of multiplications to calculate the result is (4 x 4) x (3 x 3 x 3) = 432.



Convolution Operation with Multiple Filters

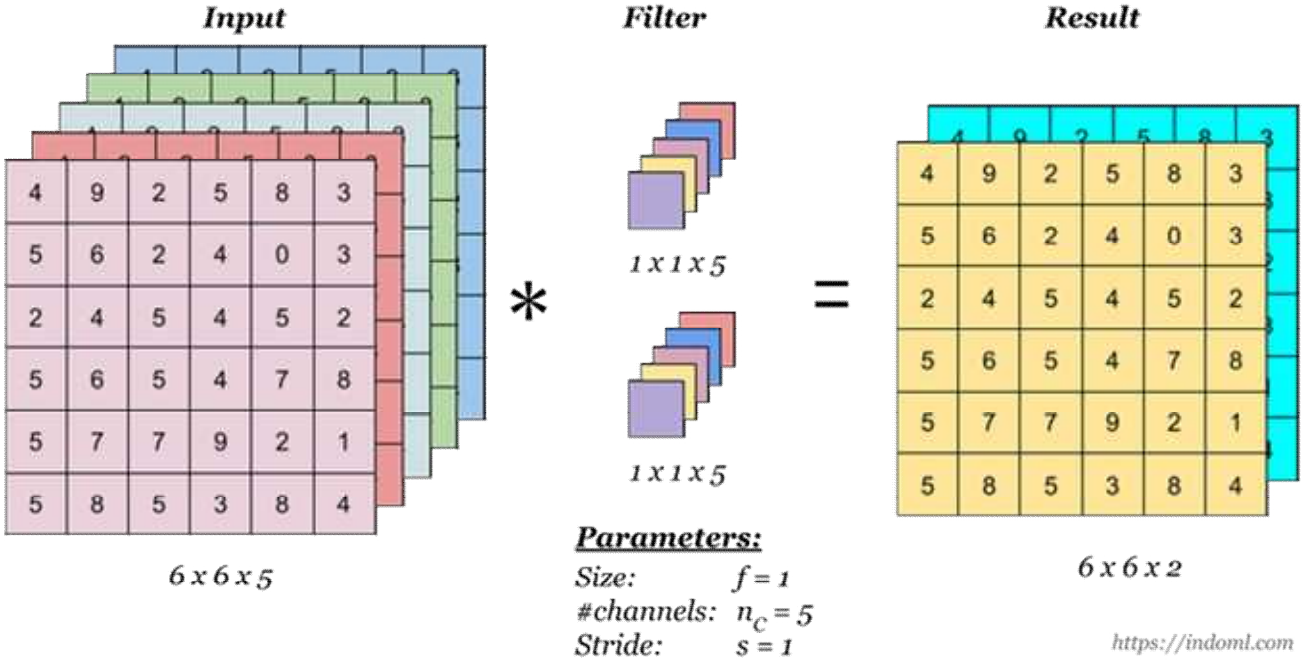
Multiple filters can be used in a convolution layer to detect multiple features. The output of the layer then will have the same number of channels as the number of filters in the layer.

The total number of multiplications to calculate the result is (4 x 4 x 2) x (3 x 3 x 3) = 864



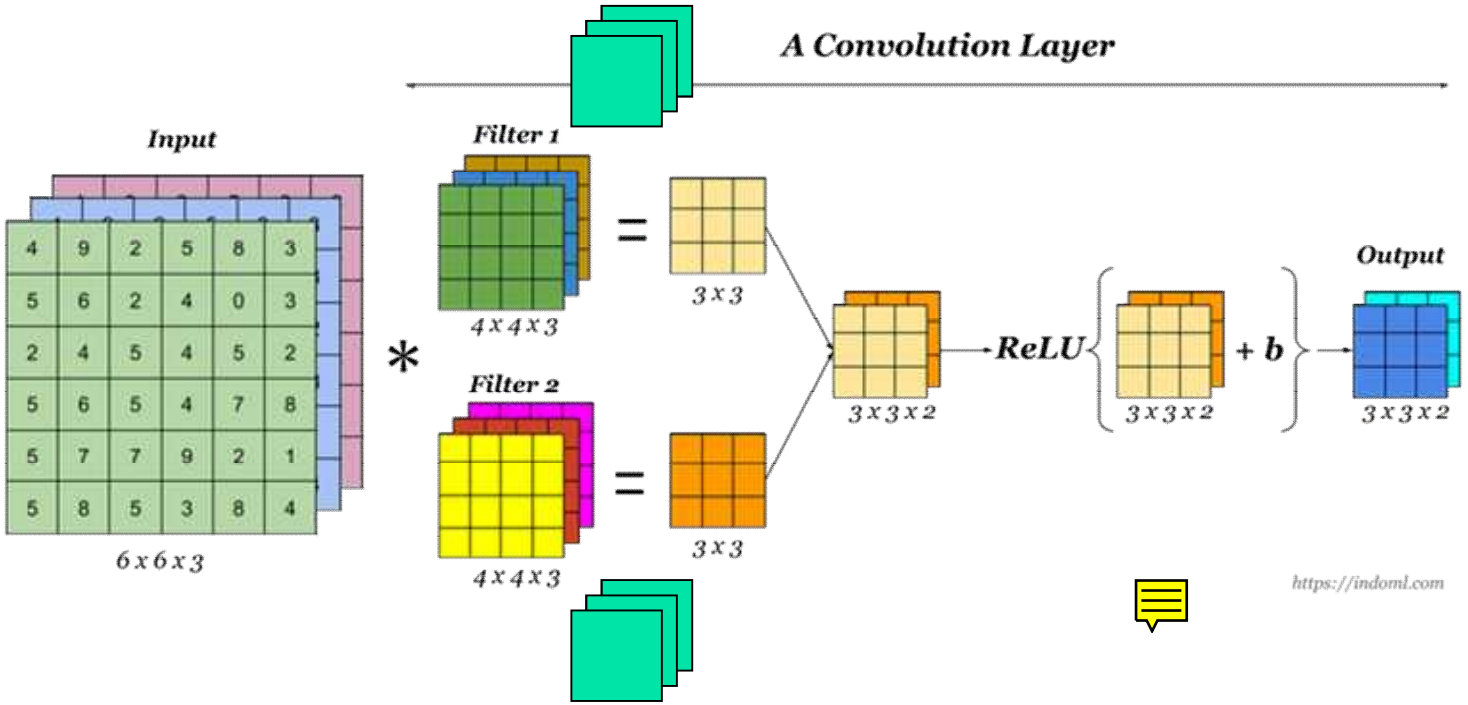
1 x 1 Convolution

This is convolution with 1 x 1 filter. The effect is to flatten or "merge" channels together, which can save computations later in the network:



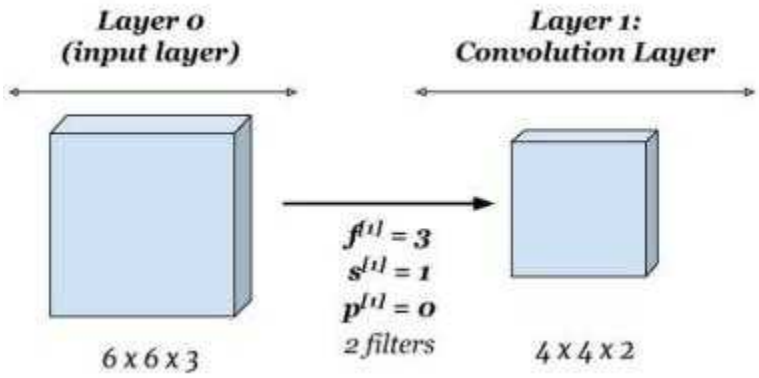
One Convolution Layer

Finally to make up a convolution layer, a bias (ϵ R) is added and an activation function such as **ReLU** or **tanh** is applied.



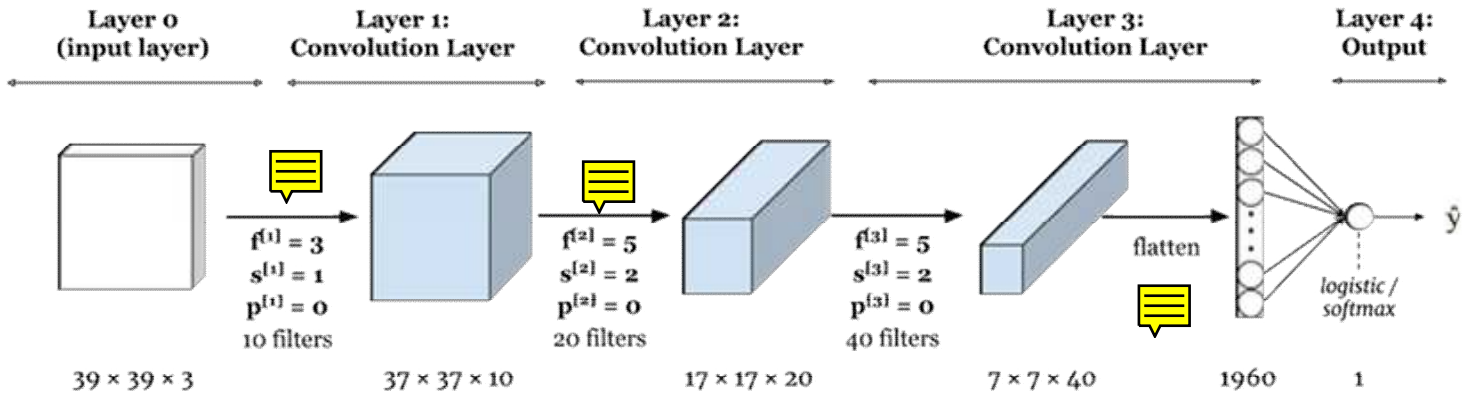
Shorthand Representation

This simpler representation will be used from now on to represent one convolutional layer:



Output size = (N+2P-F)/stride + 1
= (6+0-3)/1 + 1 = 4

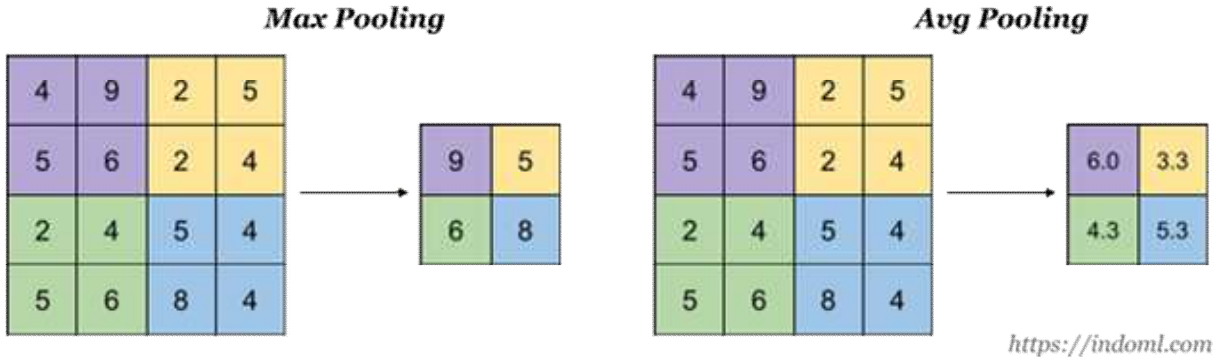
This is a sample network with three convolution layers. At the end of the network, the output of the convolution layer is flattened and is connected to a logistic regression or a softmax output layer.



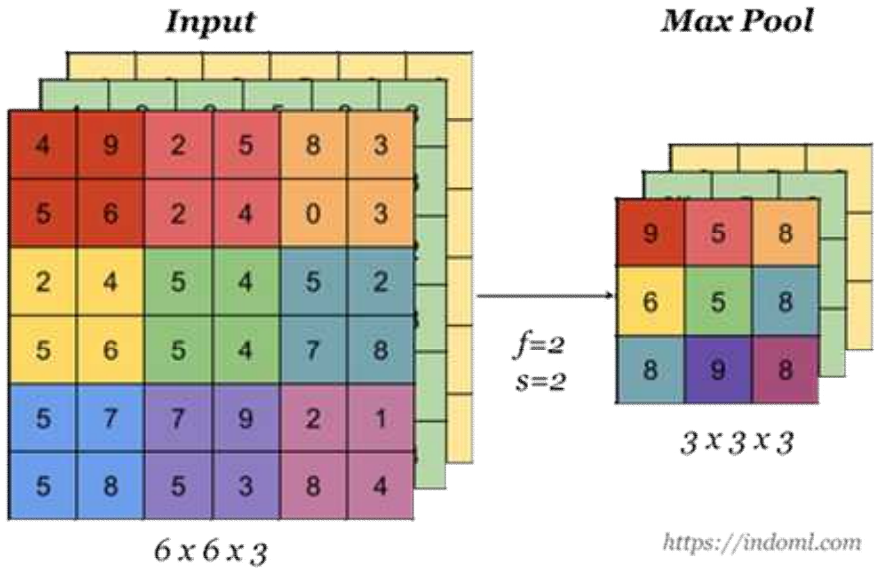
(N+2P-F)/stride + 1
= (39+0-3)/1 + 1 = 37

(N+2P-F)/stride + 1
= (17+0-5)/2 + 1 = 7

Pooling Layer



Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.
Sample types of pooling are **max pooling** and **avg pooling**, but these days max pooling is more common.



it has hyper-parameters:

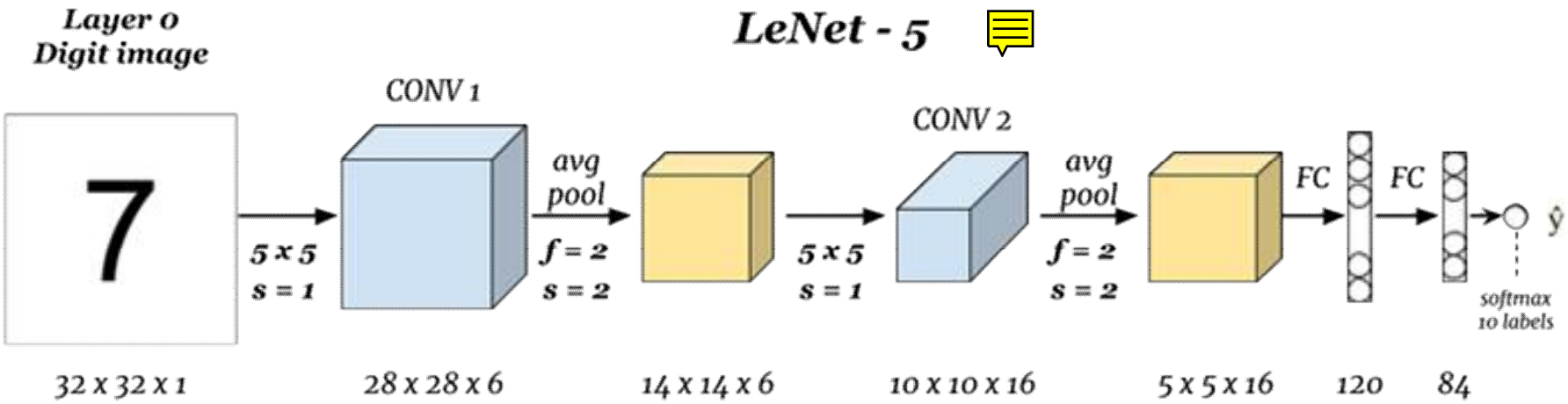
- **size** (***f***)
- **stride** (***s***)
- **type** (max or avg)

but it doesn't have parameter; there's nothing for gradient descent to learn

When done on input with multiple channels, pooling reduces the height and width (nW and nH) but keeps nC unchanged:

Classic Network: LeNet – 5

One example of classic networks is LeNet-5,
from *Gradient-Based Learning Applied to Document Recognition* paper
by Y. Lecun, L. Bottou, Y. Bengio and P. Haffner (1998):



O = (N+2P-F)/stride + 1
= (32+0-5)/1 + 1 = 28

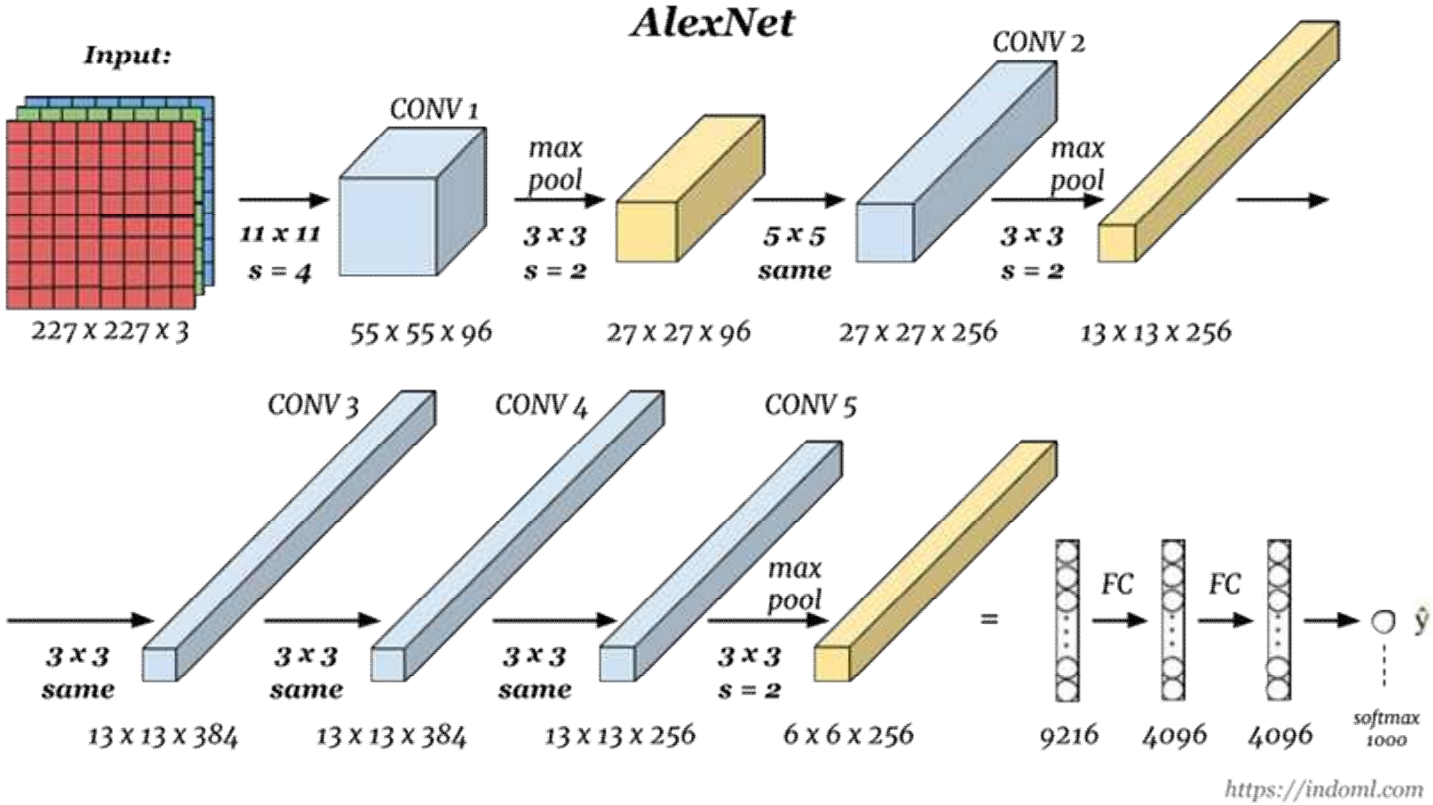
O = (N+2P-F)/stride + 1
= (14+0-5)/1 + 1 = 10

Number of parameters: ~ 60 thousands.

Classic Network: AlexNet

AlexNet is another classic CNN architecture from *ImageNet Classification with Deep Convolutional Neural Networks* paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever (2012).

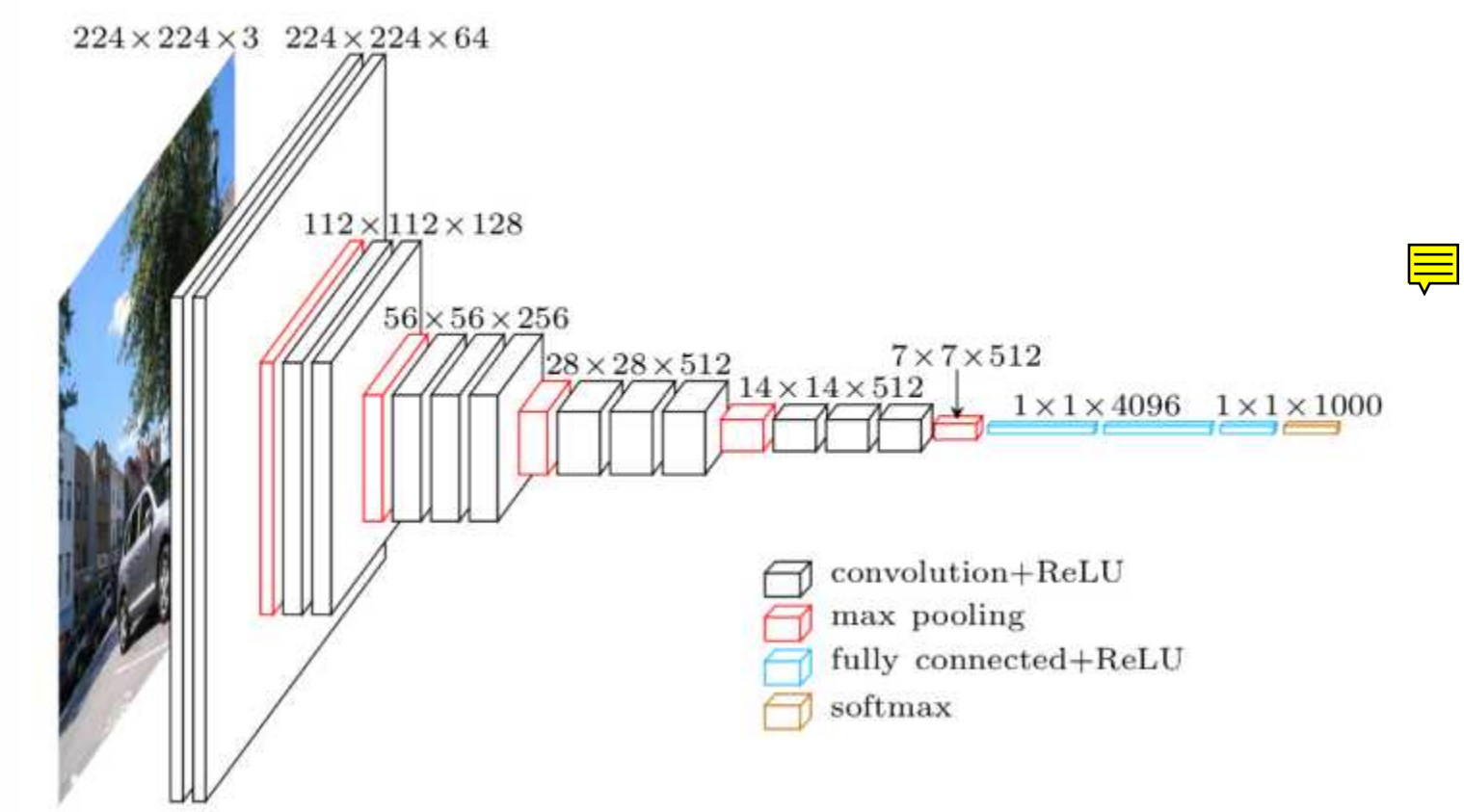
O = (N+2P-F)/stride + 1
= (227+0-11)/4 + 1 = 55



Number of parameters: ~ 60 millions.

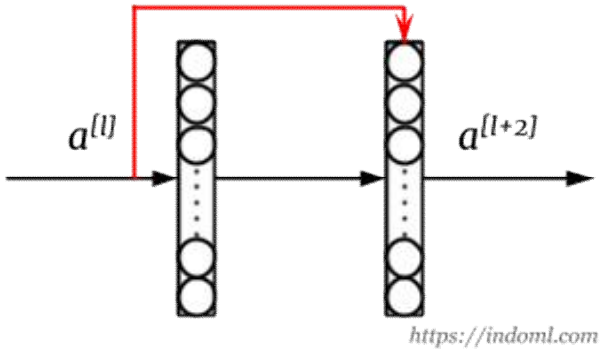
Classic Network: VGG-16

VGG-16 from *Very Deep Convolutional Networks for Large-Scale Image Recognition* paper by Karen Simonyan and Andrew Zisserman (2014). The number 16 refers to the fact that the network has 16 trainable layers (i.e. layers that have weights).

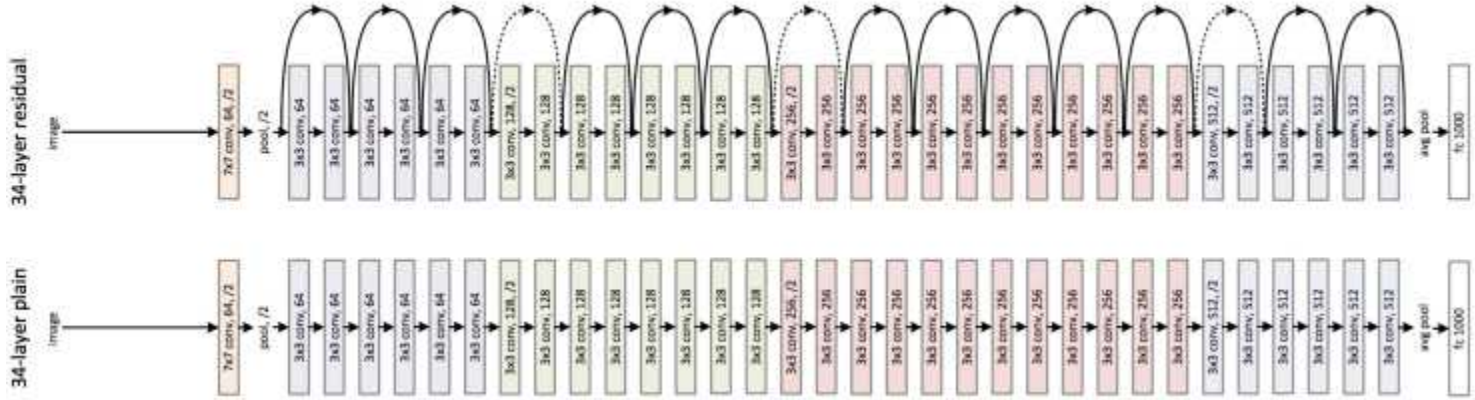


Number of parameters: ~ 138 millions.
The strength is in the simplicity: the dimension is halved and the depth is increased on every step (or stack of layers)

ResNet



The problem with deeper neural networks are they are harder to train and once the number of layers reach certain number, the training error starts to raise again. Deep networks are also harder to train due to exploding and vanishing gradients problem. ResNet (Residual Network), proposed by He at all in *Deep Residual Learning for Image Recognition paper* (2015), solves these problems by implementing skip connection where output from one layer is fed to layer deeper in the network:



In the image above, the skip connection is depicted by the red line. The activation $\boldsymbol{a}^{[l+2]}$ is then calculated as:

$\boldsymbol{z}^{[l+2]} = \boldsymbol{W}^{[l+2]}\,\boldsymbol{a}^{[l+1]} + \boldsymbol{b}^{[l+2]}$

$\boldsymbol{a}^{[l+2]} = \boldsymbol{g}^{[l+2]}(\boldsymbol{z}^{[l+2]} + \boldsymbol{a}^{[l]})$

The advantages of ResNets are:

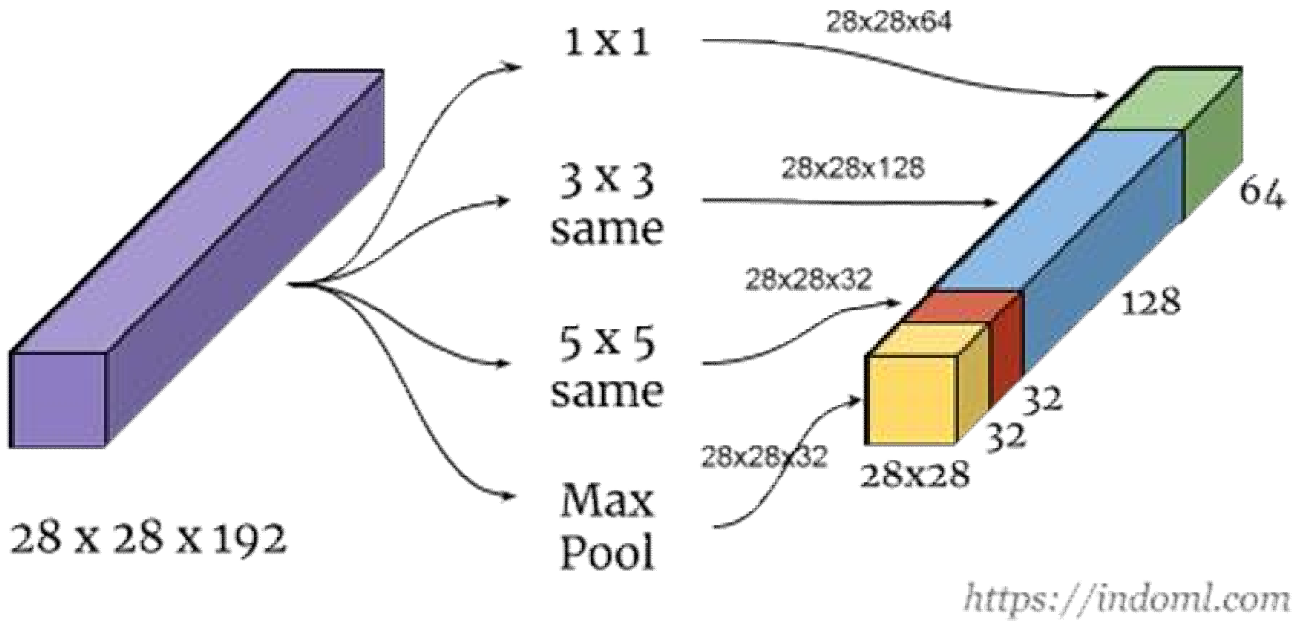
- performance doesn't degrade with very deep network
- cheaper to compute ability to train very very deep network

ResNet works because:

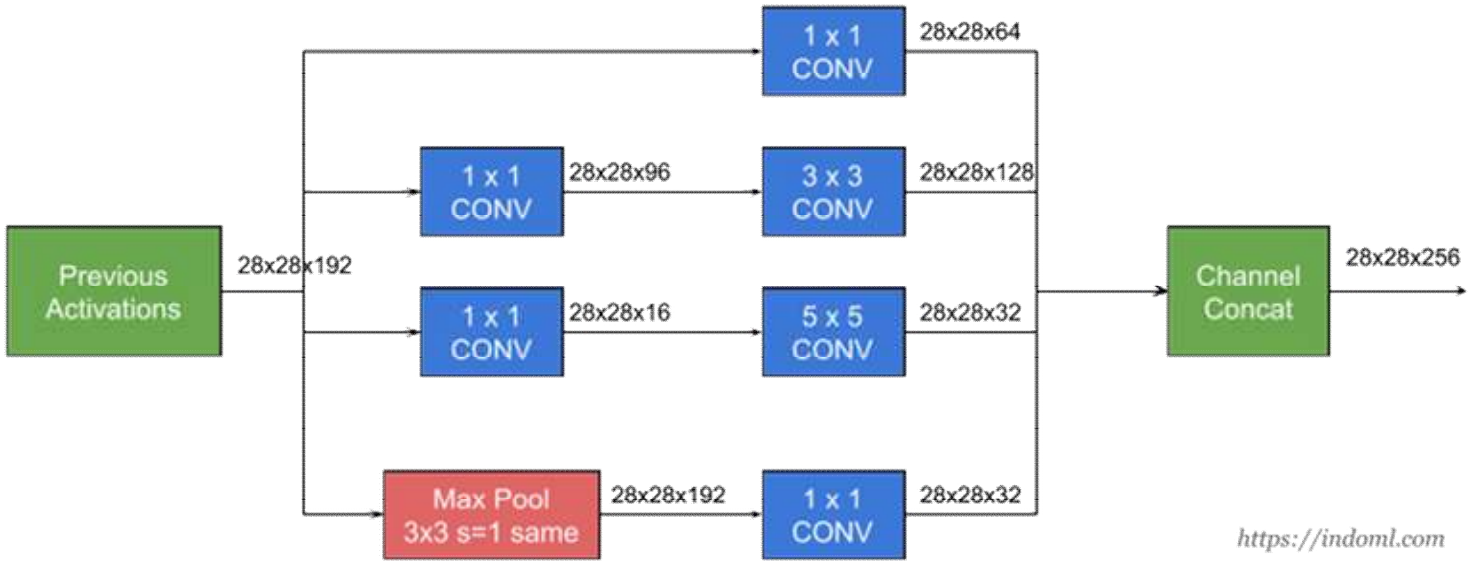
- identify function is easy for residual block to learn
- using a skip-connection helps the gradient to back-propagate and thus helps you to train deeper networks

Inception

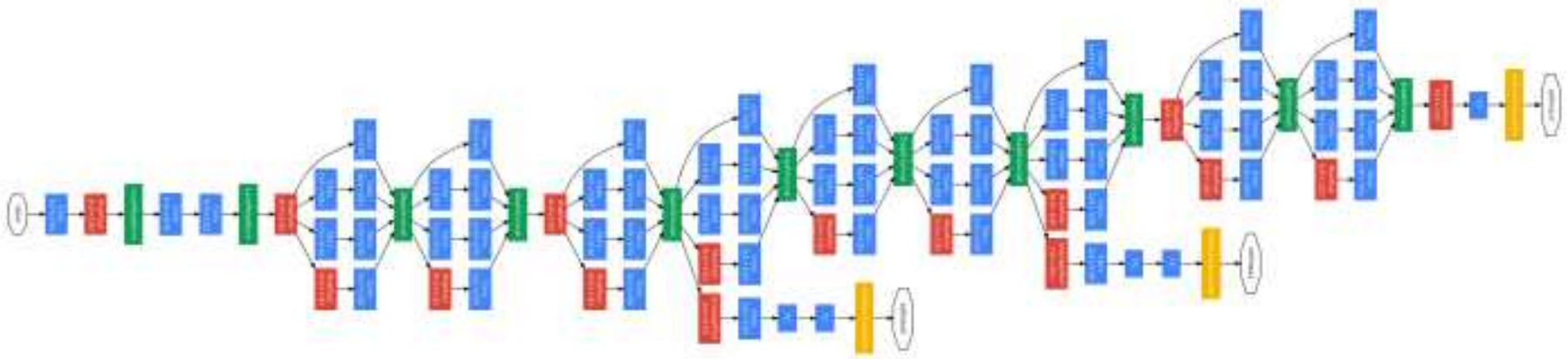
The motivation of the inception network is, rather than requiring us to pick the filter size manually, let the network decide what is best to put in a layer. We give it choices and hopefully it will pick up what is best to use in that layer:



The problem with the above network is computation cost (e.g. for the 5 x 5 filter only, the computation cost is (28 x 28 x 32) x (5 x 5 x 192) = ~ 120 millions).
Using 1 x 1 convolution will reduce the computation to about 1/10 of that. With this idea, an inception module will look like this:



Below is an inception network called **GoogLeNet**, described in *Going Deeper with Convolutions paper* by Szegedy et all (2014), which has 9 inception modules:



conv2d 비교

TF1

```
conv2d_img = tf.nn.conv2d(image, weight, strides=[1, 1, 1, 1],
                                                    padding='VALID')
```

Pytorch

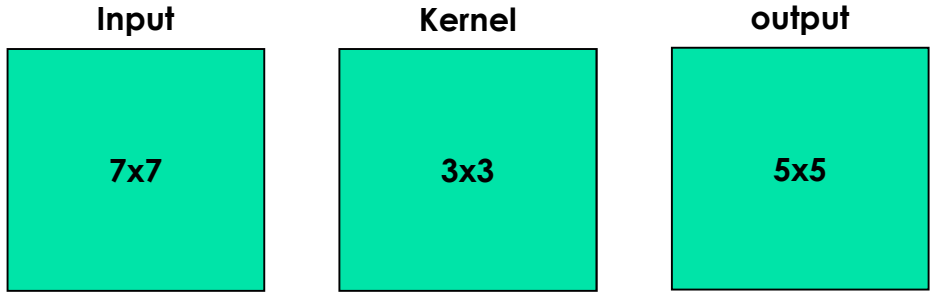
```
conv1 = torch.nn.Conv2d(3, 1, 3)
conv1.weight.data.fill_(10.0)
conv1.bias.data.fill_(10)

input = torch.randn(1, 3, 10, 10)
output = conv1(input)
```

TF2

```
conv2d_img = tf.keras.layers.Conv2D(1, 2, kernel_initializer=my_init,
                                                    bias_initializer='zeros')(x)
```

Stride 예제



Stride 조정
Padding 조정

Output size = (N+2P-F)/stride + 1
= (6+2-3)/1 + 1 = 6