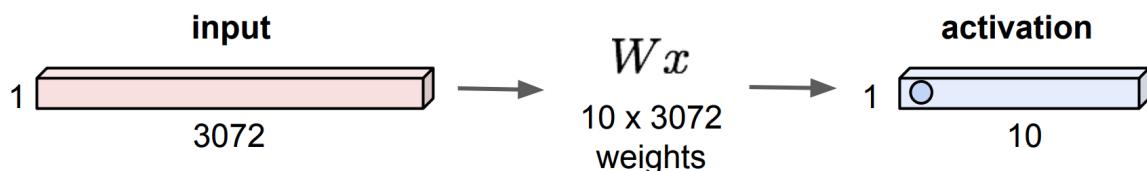


# Lecture 5 | Convolutional Neural Networks

[앞부분에서는 역사에 대해 설명함.]

## Fully Connected Layer

아래의 그림처럼 Fully Connected Layer(FC)는  $1 \times 3072$  길이의 입력이 들어오면  $W$ (weights)와 곱해져서  $1 \times 10$  의 결과를 얻게 된다. 과거에는 이런 방식으로 탐지를 했지만 상당히 나쁜 정확도를 가지고 있다.

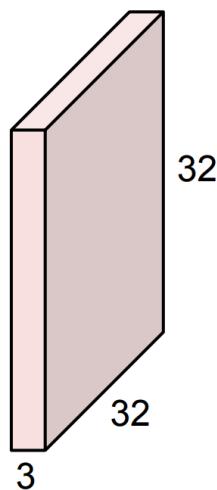


## Convolution Layer

Fully Connected Layer 대신 더 나은 방법을 선택한 것이 바로 Convolution Layer이다.

입력 이미지의 depth는 그대로 보존하면서 크기는 줄여 필터를 만든다. (여기서 depth는 RGB를 의미) 우리는 필터를 이용하여 이미지의 특징을 뽑아낼 것이다.

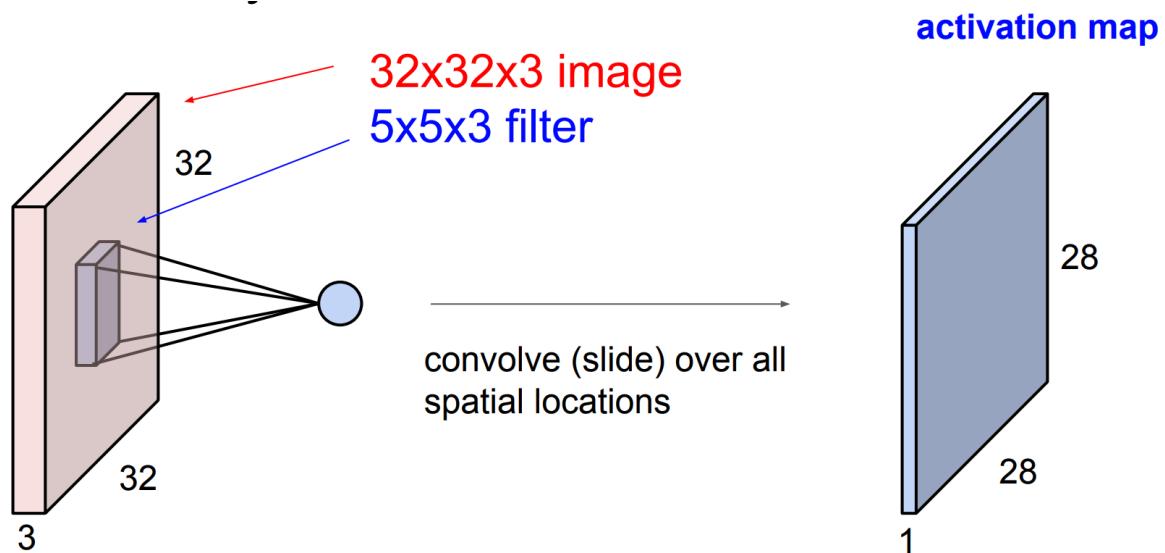
32x32x3 image



5x5x3 filter

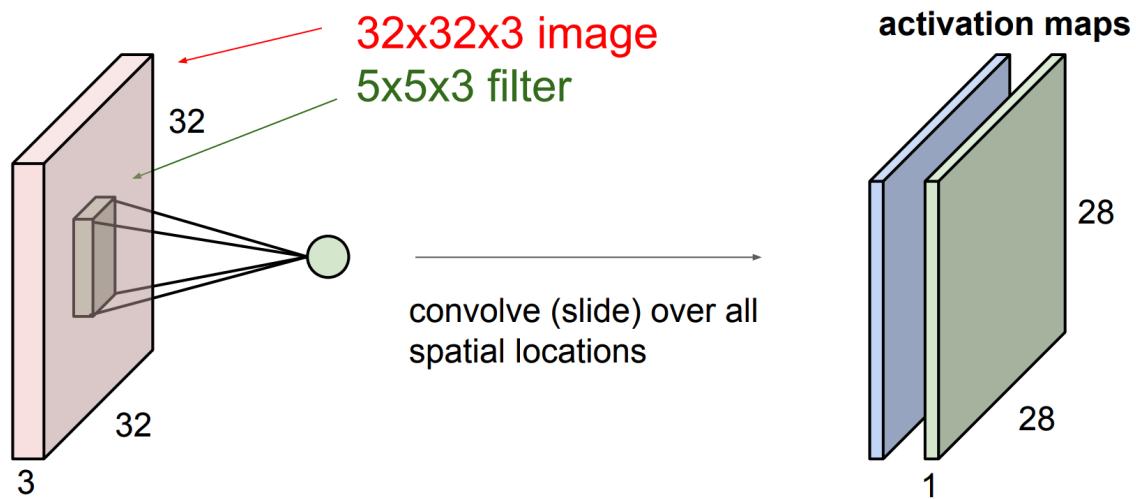


우리는 필터를 이미지의 각각의 특정 부분과 곱하여 특징을 뽑아낼 것이다. 그리고 각 계산마다 하나의 값을 얻어낼 수 있다. 아래의 그림처럼 이미지와 필터가 겹칠 수 있는 모든 영역에서 다 특징을 뽑아내면 우리는  $28 \times 28 \times 1$  의 activation map을 얻을 수 있다. 필터를 왼쪽 위부터 오른쪽 아래까지 계속 이동시키면서 특징을 뽑아낸다고 생각하면 된다. 정확한 설명은, 아래에서 할 것이기 때문에 간단하게 이해하고 넘어가도록 하자.

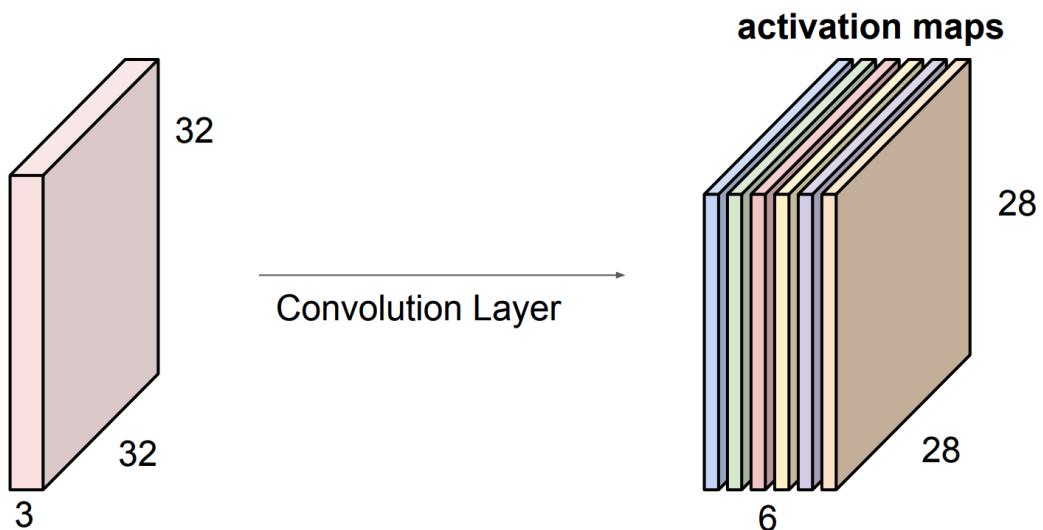


Convolution layer에서 특징을 뽑아낼 때 하나의 필터만 사용하는 것은 아니다. 여러 특징을 뽑아내기 위해 여러개의 필터를 이용해서 반복 실행하면, 여러개의 activation map을 얻을 수 있다. 위에서는 파란색 필터로 특징을 뽑아낸 반면, 아래 그림에서는 초록색 필터를 사용하여 activation map을 얻어내는 것을 볼 수 있다. 이렇게 여러 개의 필터를 이

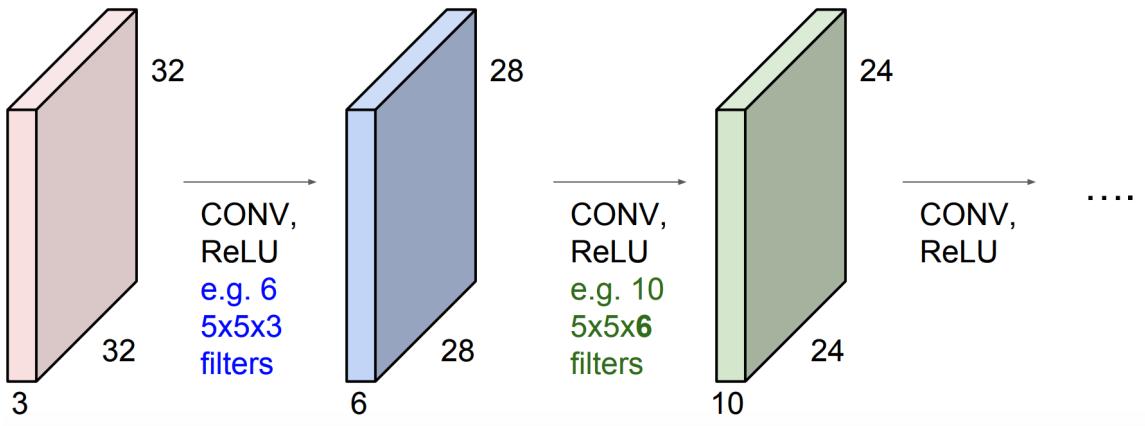
용하여 특정 image의 많은 특징들을 뽑아내어 image classification의 정확도를 높이고자 한다.



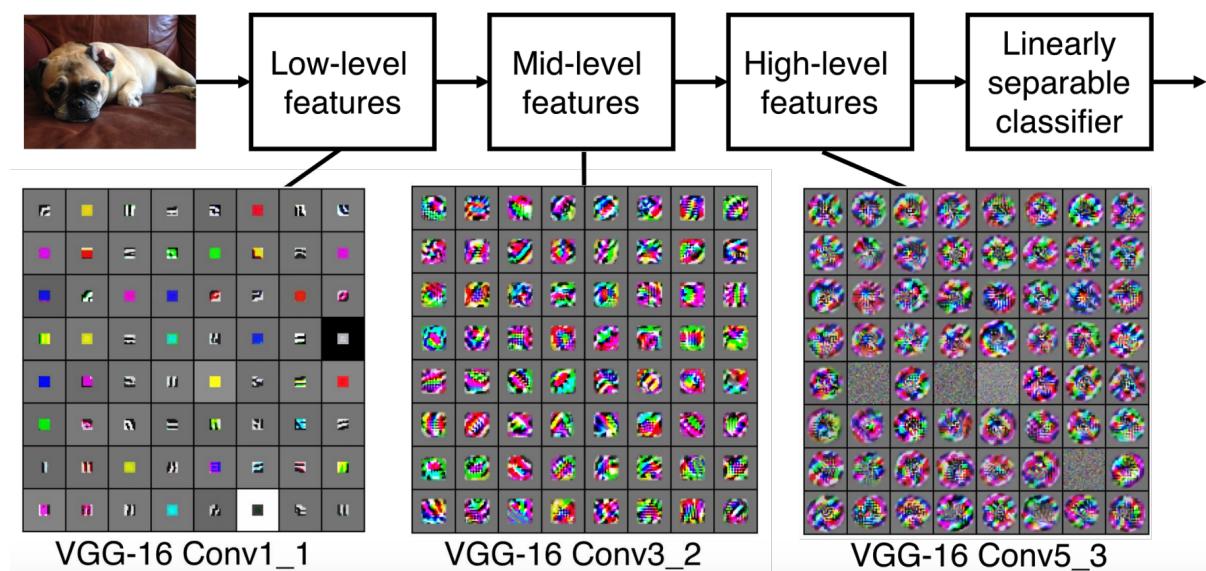
예를 들어, 우리가 6개의  $5 \times 5 \times 3$  필터를 가지고 특징을 뽑아냈다면 결과적으로 6개의  $28 \times 28 \times 1$  activation map, 즉  $28 \times 28 \times 6$  의 activation map을 얻을 수 있다.



위 작업을 반복하면, 필터를 이용하여 계속 새로운 행렬(특징들)을 얻어낼 수 있다. 여기서 알 수 있는 것은, 필터의 크기가  $1 \times 1$ 이 아닌 이상, 특징을 뽑아낼수록 가지고 있는 행렬의 크기가 감소한다는 것이다.



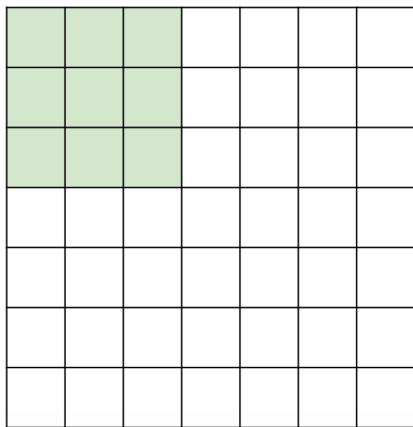
위처럼 진행하게 되면 초반에는 low-level 특징들을 얻게 되고, 점점 여러 필터들을 거칠 수록 High-level 특징들에 대한 데이터 셋을 얻을 수 있게 된다. 이는 뒤에서 더 자세히 살펴 볼 것이다.



앞서 우리는 필터를 거치면 activation map이 나온다고 했는데 정확히 어떤 매커니즘을 통해 특징을 뽑아내고, activation map을 만들어 내는지 알아보자.

아래와 같이 입력의 크기가  $7 \times 7$ 이고 필터의 크기가  $3 \times 3$ 인 상황을 가정해보자.

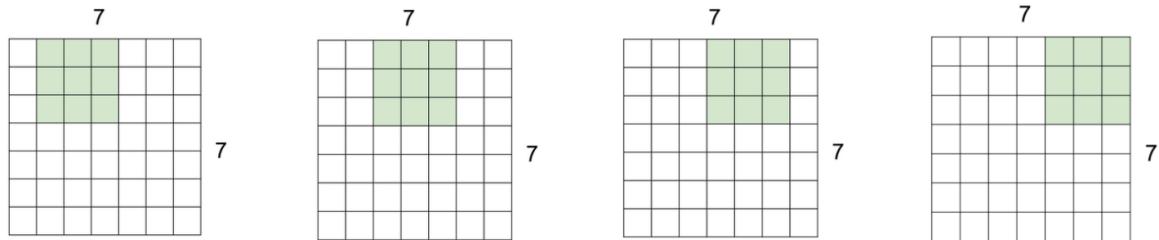
7



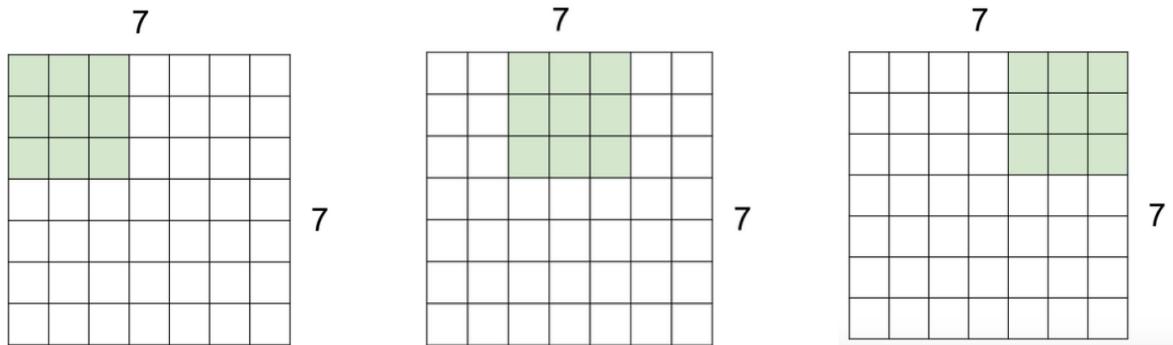
7x7 input (spatially)  
assume 3x3 filter

7

우선 필터를 한칸씩 이동시켜 가면서 데이터를 뽑아내면 우리는  $5 \times 5$  activation map을 얻을 수 있다.



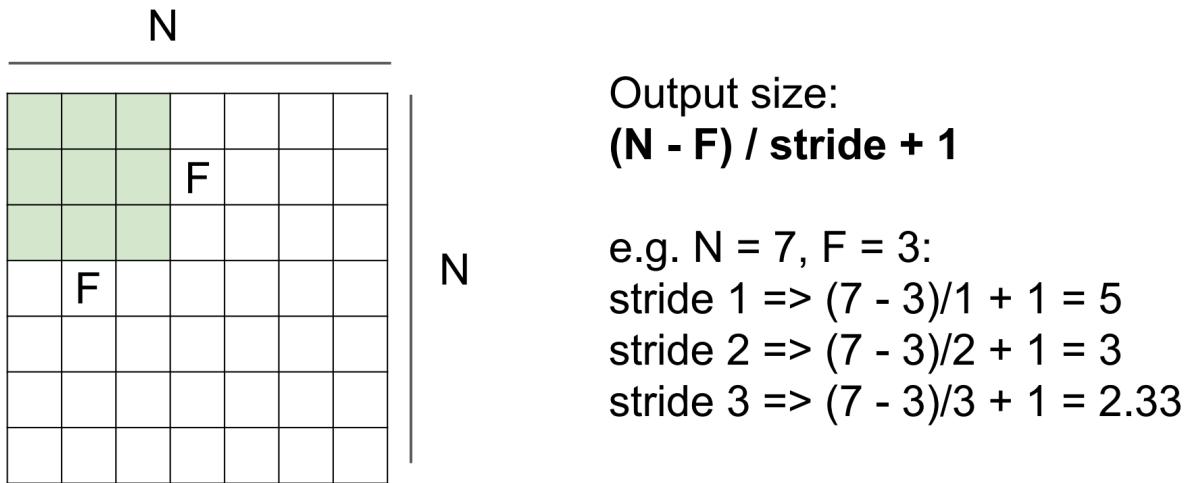
또, 필터를 2칸씩 이동시켜 간다면 한 방향으로 2번 밖에 이동하지 못하기 때문에  $3 \times 3$  activation map을 얻는다.



만약 3칸씩 이동하려 한다면 어떻게 될까? 개수가 딱 들어맞지 않기 때문에 정상적으로 작동하지 못한다.

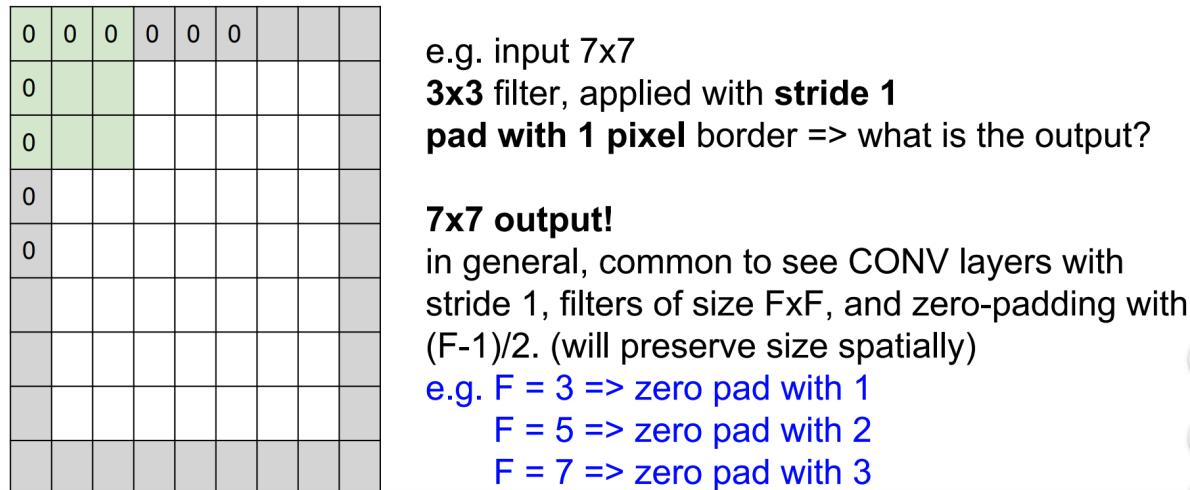
결국, input 크기가  $N \times N$ , 필터 크기가  $F \times F$ , 이동 폭을 stride라고 하면

Output size는  $(N - F) / \text{stride} + 1$ 이 된다.



이처럼 개수가 딱 들어맞지 않는 경우가 생길 때, 우리는 0-pad를 사용한다. 가장자리에 0으로 된 데이터를 적절한 개수 만큼 넣으면 stride 1, 2, 3인 경우 모두 개수가 맞아 떨어져서 위의 작업을 할 수 있게 된다.

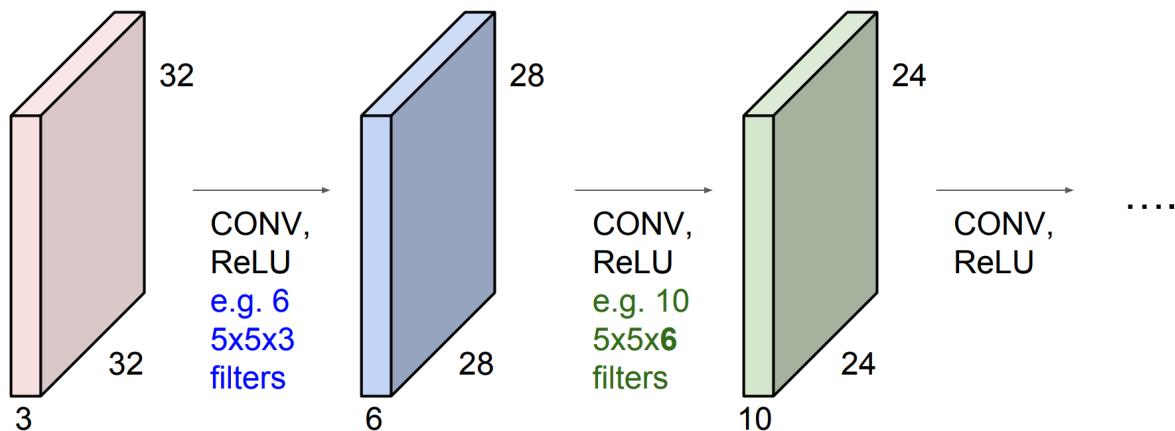
즉 원하는 stride가 있는데 개수가 안 맞아 떨어질 경우 0-pad를 이용하여 이를 해결할 수 있다.



또한 0-pad를 이용하면 input의 크기와 동일한 output을 얻어낼 수 있다.

아래 그림을 보면 아무런 0-pad 없이 진행하기 때문에 한번 필터를 지나칠 때마다 사이즈가 4씩 감소한다. 물론, 빠르게 데이터의 크기를 감소시켜 결과를 얻는다는 입장에서 좋을 수는 있다. 하지만 그러면 정확성이 떨어질 가능성이 높다. 특히 가장자리에 있는 데이터의 손실이 일어나기 때문에 각 단계를 거칠수록 정보의 손실이 생기는 것이다.

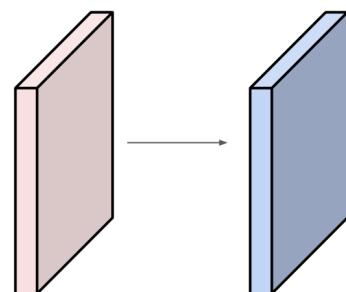
그렇기 때문에 우리는 0-pad를 이용하여 사이즈를 유지할 수 있고 가장자리의 손실도 없게끔 할 수 있다.



예를 들어 보자. 만약 아래의 그림처럼 input의 크기가  $32 \times 32 \times 3$ 이고 사이즈가  $5 \times 5$ 인 필터 10개가 있으면 최종 output의 사이즈는 어떻게 될까?

Examples time:

Input volume: **32x32x3**  
10  $5 \times 5$  filters with stride 1, pad 2



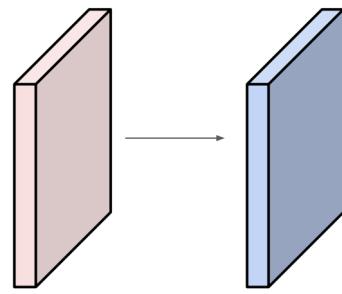
Output volume size: ?

우선 pad가 2이기 때문에 전체 크기가 32가 아닌  $32 + 2 * 2$  해서 36이 된다. 그리고 필터 사이즈가 5이기 때문에 5를 빼면 31이 되고 stride가 1이기 때문에 그대로 31이 된다. 최종적으로 1을 더하면 32가 되고, 필터가 10개이기 때문에 크기는  $32 * 32 * 10$ 이 된다.

두번 째로, 동일한 상황에서 parameter의 개수는 몇개일까?

Examples time:

Input volume: **32x32x3**  
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

필터 사이즈가  $5 \times 5 \times 3$ 이고, 각 필터마다 bias가 있기 때문에 총  $5 * 5 * 3 + 1 = 76$  개의 parameter들이 있다. 또, 필터가 총 10개 이기 때문에 전체 parameter의 개수는 760개가 된다.

아래는 위의 내용들을 요약한 부분이다.

우리는 주로 필터의 개수를 2의 배수(32, 64, 128, 256 ...)으로 한다.

### Common settings:

**Summary.** To summarize, the Conv Layer:

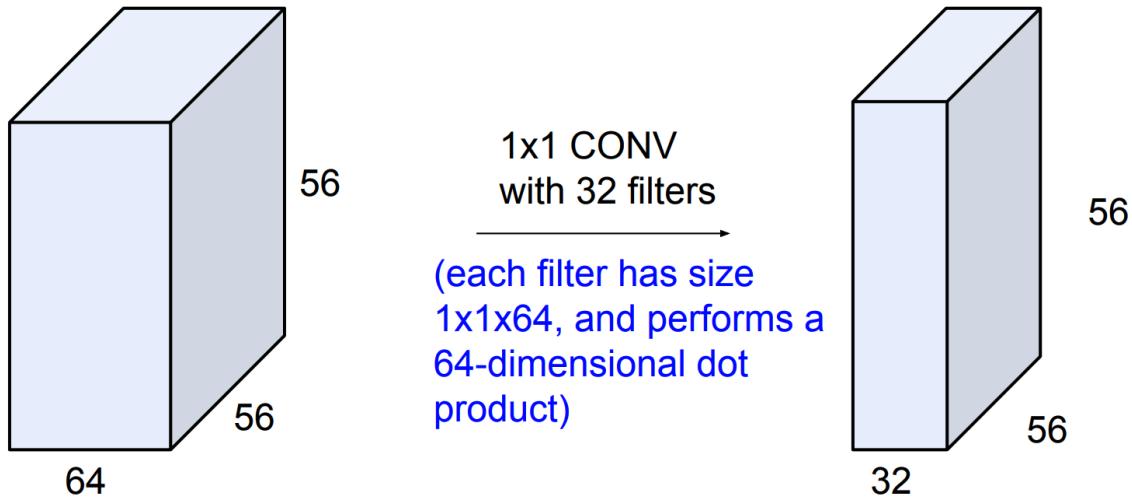
- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces  $F \cdot F \cdot D_1$  weights per filter, for a total of  $(F \cdot F \cdot D_1) \cdot K$  weights and  $K$  biases.
- In the output volume, the  $d$ -th depth slice (of size  $W_2 \times H_2$ ) is the result of performing a valid convolution of the  $d$ -th filter over the input volume with a stride of  $S$ , and then offset by  $d$ -th bias.

$K = (\text{powers of 2, e.g. } 32, 64, 128, 512)$

- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$  (whatever fits)
- $F = 1, S = 1, P = 0$

필터의 크기를  $1 \times 1$ 로 하는 경우도 존재한다.  $1 \times 1$ 이면 input과 output의 사이즈가 동일하지 않은가라는 생각을 할 수 있는데, 이 때는 depth 사이즈를 변경할 수 있다.

즉, 우리는 Convolution layer을 거치면서 해당 이미지의 특징들을 low-level 부터 High-level까지 추출할 수 있고, 추후에 이를 이용하여 image classification을 할 것이다. 각각의 Conv layer마다 적합한 필터 및 그 개수, 패딩을 얼마나 할 것인지 등의 정보를 정하고 이를 반복해 나갈 것이다.



Torch 와 Caffe에서는 아래와 같이 layer을 만들어 놓았다고 한다.

## Example: CONV layer in Torch

**Summary.** To summarize, the Conv Layer:

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .

### SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, KH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `KH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is  $(kW-1)/2$  .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is  $(KH-1)/2$  .

Note that depending on the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidht` where

$$\text{owidht} = \text{floor}((\text{width} + 2*\text{padW} - \text{kW}) / \text{dW} + 1)$$

$$\text{oheight} = \text{floor}((\text{height} + 2*\text{padH} - \text{KH}) / \text{dH} + 1)$$

Torch is licensed under BSD 3-clause.

## Example: CONV layer in Caffe

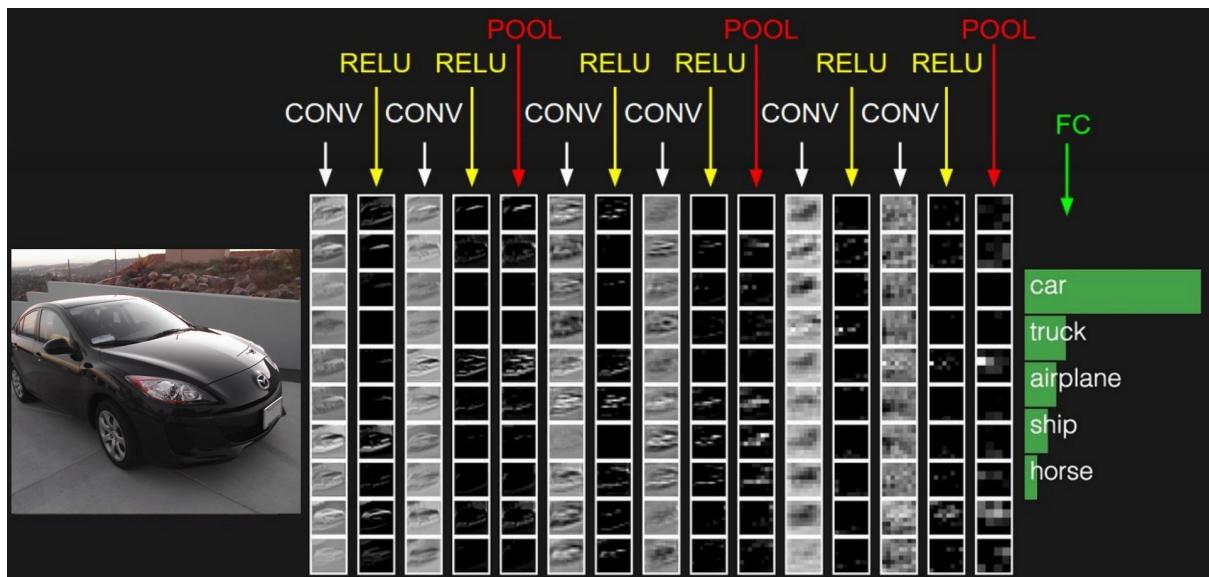
```
layer {
    name: "conv1"
    type: "Convolution"
    bottom: "data"
    top: "conv1"
    # learning rate and decay multipliers for the filters
    param { lr_mult: 1 decay_mult: 1 }
    # learning rate and decay multipliers for the biases
    param { lr_mult: 2 decay_mult: 0 }
    convolution_param {
        num_output: 96      # learn 96 filters
        kernel_size: 11     # each filter is 11x11
        stride: 4           # step 4 pixels between each filter application
        weight_filler {
            type: "gaussian" # initialize the filters from a Gaussian
            std: 0.01          # distribution with stdev 0.01 (default mean: 0)
        }
        bias_filler {
            type: "constant" # initialize the biases to zero (0)
            value: 0
        }
    }
}
```

**Summary.** To summarize, the Conv Layer:

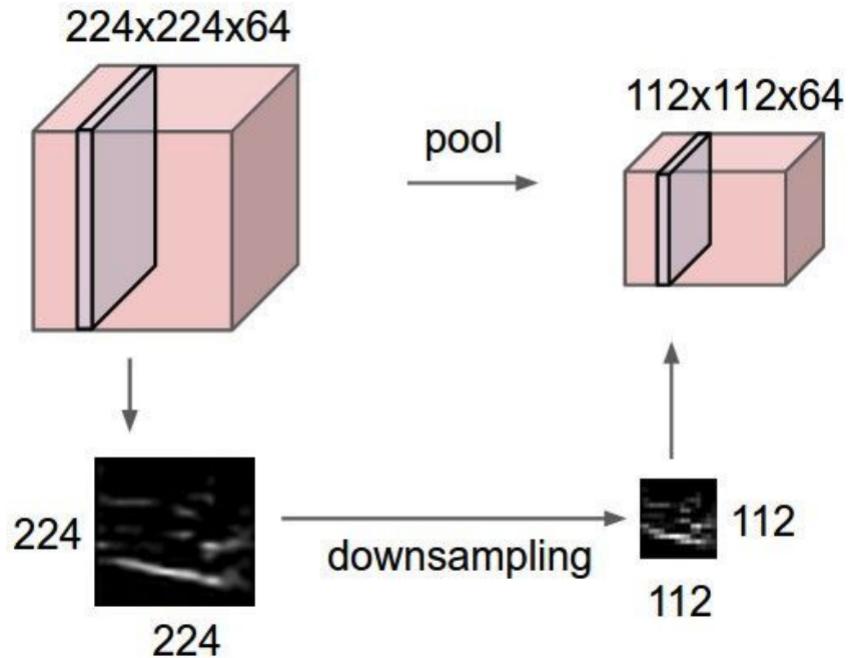
- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .

Caffe is licensed under [BSD 2-Clause](#).

layer에는 Conv layer만 있는 것이 아니라 pooling layer도 존재한다. 이번에는 pooling layer에 대해 알아보자.

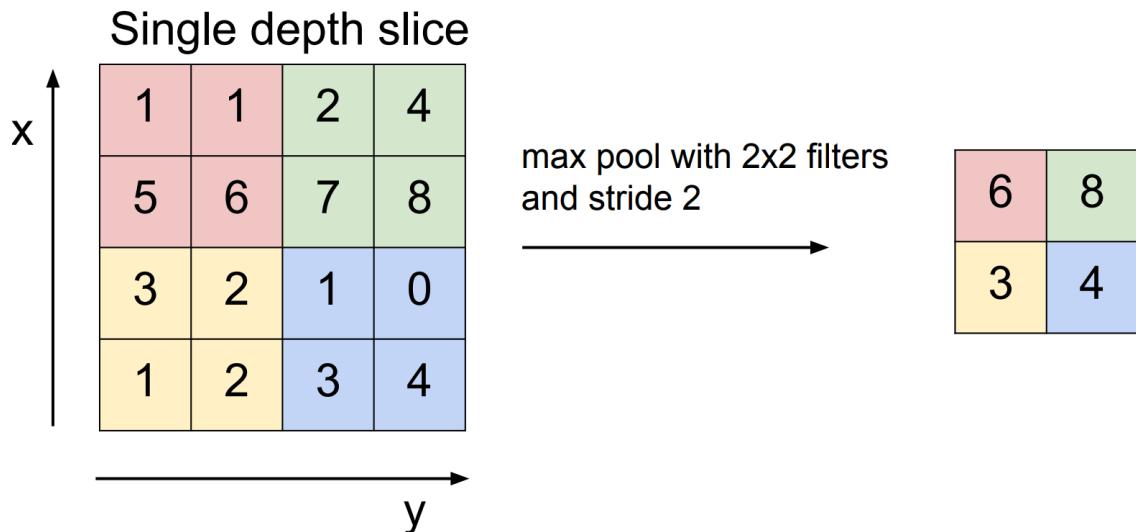


pooling layer에서는 사이즈를 축소하는 역할을 한다. 여기서는 depth에는 변화가 없고, 가로와 세로 사이즈를 줄이는 동작이 일어난다.



pooling 방법 중 하나가 max pooling이다. 만약 아래 그림처럼  $4 \times 4$  사이즈의 입력이 주어지고  $2 \times 2$  필터를 이용한다고 해보자. 그렇다면 필터의 사이즈별로 input을 쪼개고, 각 칸마다 최댓값만을 들고 와서 새로운 데이터 셋으로 output을 만드는 것이다.

## MAX POOLING



즉 크기가  $F$ , stride 가  $S$ 인 필터를 이용하여 pooling을 한다고 해보자. 기존의 사이즈가  $W_1 * H_1 * D_1$ 이라고 하면

$W_2 = (W_1 - F)/S + 1$  ,  $H_2 = (H_1 - F)/S + 1$  ,  $D_2 = D_1$  으로 변하게 된다.

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

## Fully Connected Layer (FC layer)

앞에서 배운 Conv, RELU, pooling layer들을 여러개 겹친 후, 마지막 Fully Connected Layer를 통해 값을 분류하게 된다. 본래의 이미지를 바로 Fully Connected Layer의 input으로 넣어 그 사진을 분류하는 것 보다 특징들을 선별하여 그 값을 입력으로 넣는 것 이 더 높은 정확도를 보인다는 사실을 알 수 있다.

