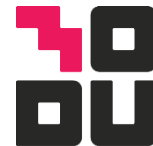


Convolutional Neural Networks

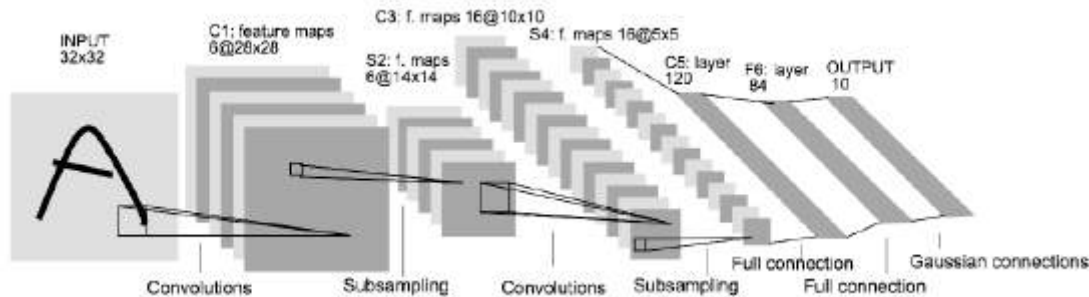
모두의연구소
박은수 책임연구원

큰 그림

큰 그림 : 구조



- LeNet (1998년)



전형적인 구조

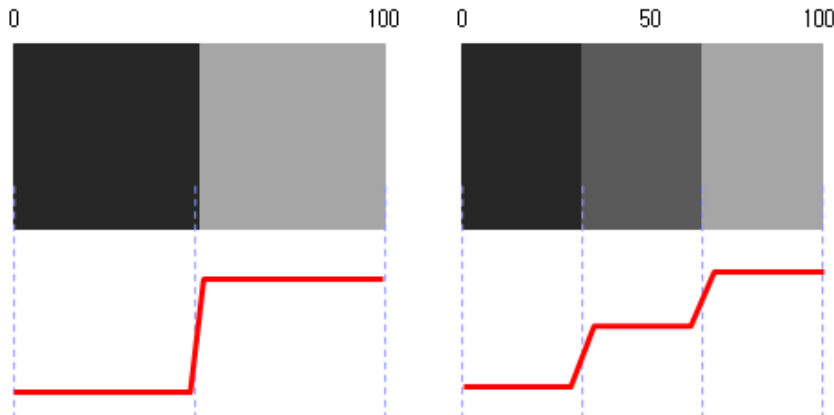
CONV POOL FC

큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

-1	0	1
-1	0	1
-1	0	1

수직 에지를 찾는
컨볼루션 필터



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

수직 에지를 찾는
컨볼루션 필터

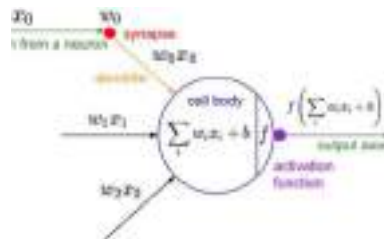
0

100

-1	0	1			
-1	0	1			
-1	0	1			



0			



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

수직 에지를 찾는
컨볼루션 필터

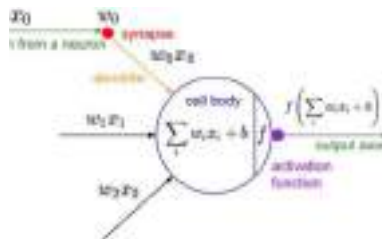
0

100

	-1	0	1		
	-1	0	1		
	-1	0	1		



0	300		



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

수직 에지를 찾는
컨볼루션 필터

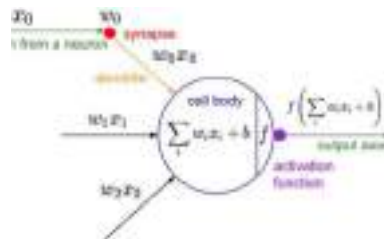
0

100

			-1	0	1
			-1	0	1
			-1	0	1



0	300	300	0



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

수직 에지를 찾는
컨볼루션 필터

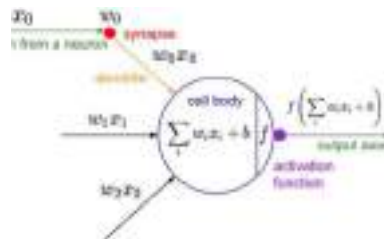
0

100

-1	0	1			
-1	0	1			
-1	0	1			



0	300	300	0
0			



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

수직 에지를 찾는
컨볼루션 필터

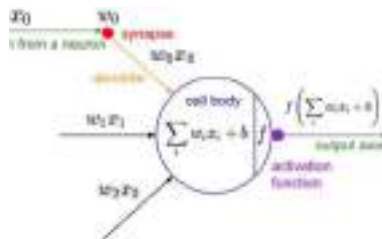
0

100

	-1	0	1		
	-1	0	1		
	-1	0	1		



0	300	300	0
0	300		



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

수직 에지를 찾는
컨볼루션 필터

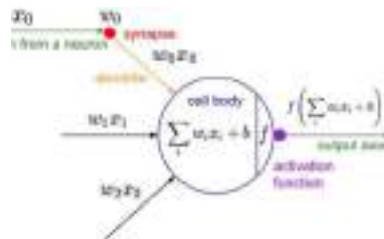
0

100

		-1	0	1	
		-1	0	1	
		-1	0	1	



0	300	300	0
0	300	300	



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

수직 에지를 찾는
컨볼루션 필터

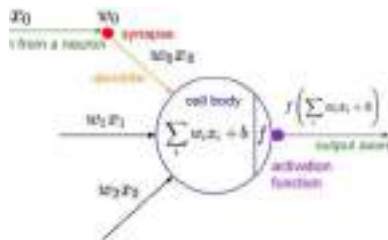
0

100

			-1	0	1
			-1	0	1
			-1	0	1



0	300	300	0
0	300	300	0
0	300	300	0



큰 그림 : 컨볼루션은 ...

컨볼루션 필터 : 이미지의 특징을 추출할 수 있다

0

100

			-1	0	1
			-1	0	1
			-1	0	1



0	300	300	0
0	300	300	0
0	300	300	0

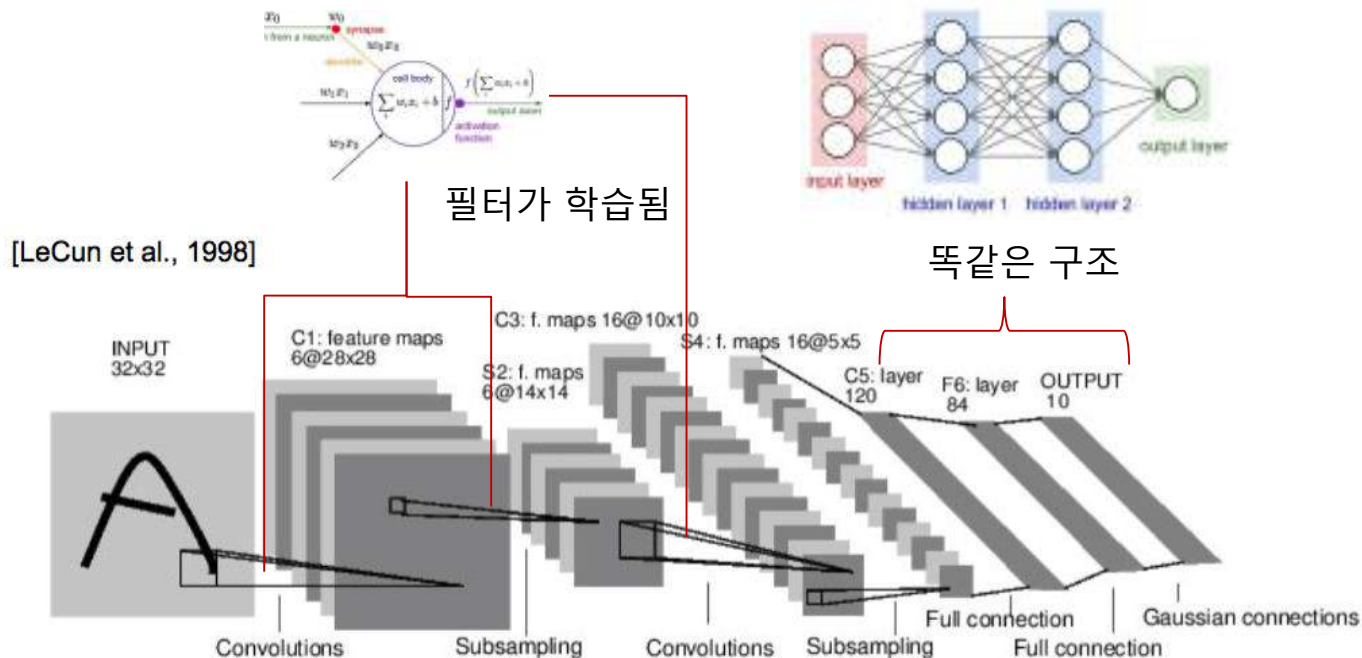
수직 에지 검출



다양한 필터로 다양한
특징을 추출할 수 있다

컨볼루션 뉴럴넷 (Convolutional Neural Network)

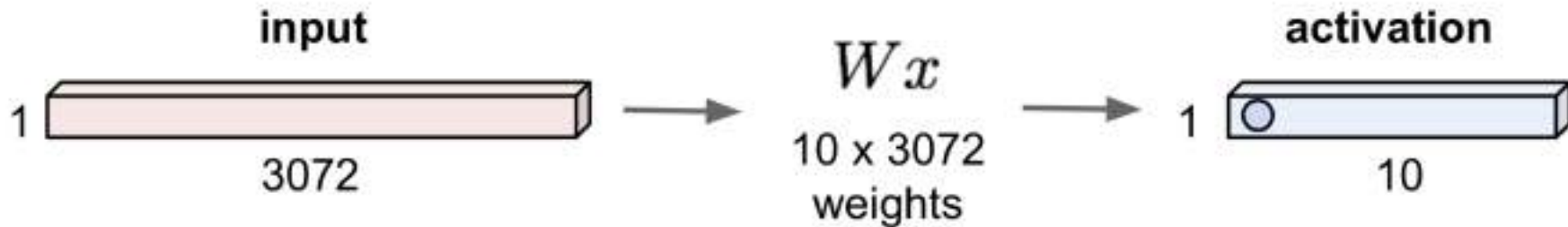
- 필터를 학습하는 구조다



이제부터 자세히 살펴봅시다

Fully Connected Layer

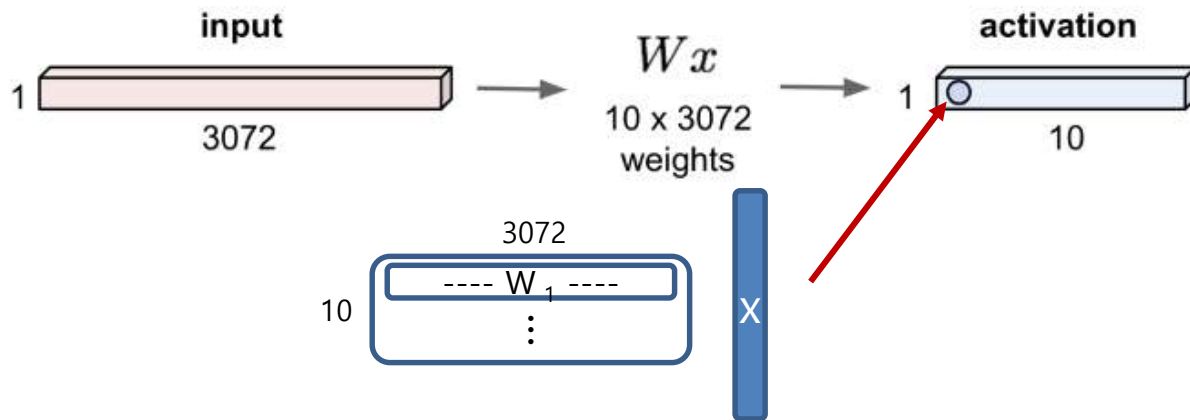
32x32x3 image -> stretch to 3072 x 1



지금까지 살펴봤떠봤던 것 입니다

Fully Connected Layer

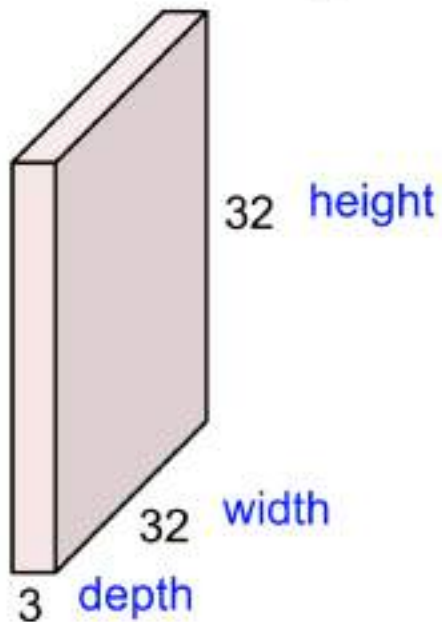
32x32x3 image -> stretch to 3072 x 1



- 1) 전체 영상 각 픽셀에 W_1 의 weighted sum으로 계산한 score
- 2) Activation 적용

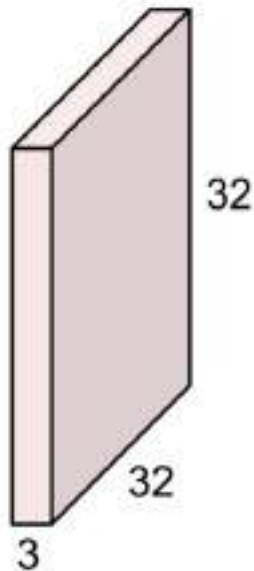
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

32x32x3 image

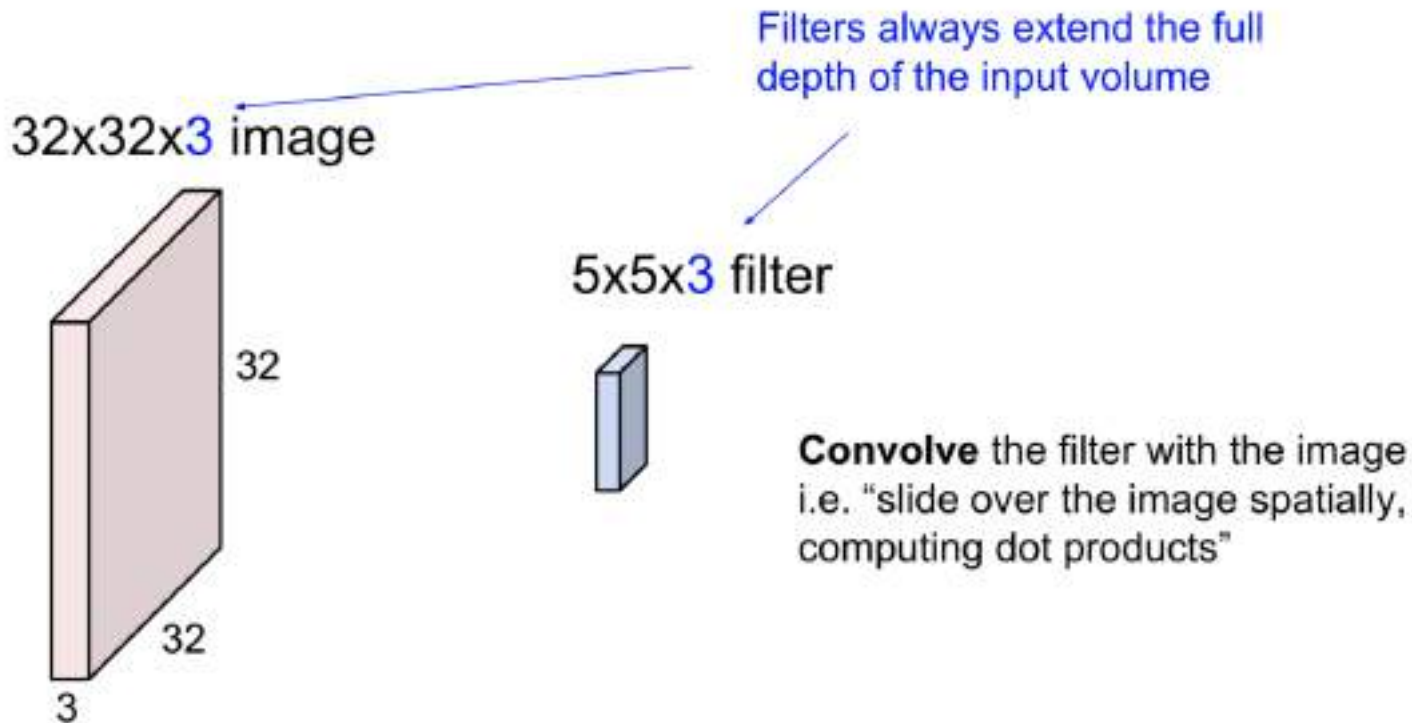


5x5x3 filter

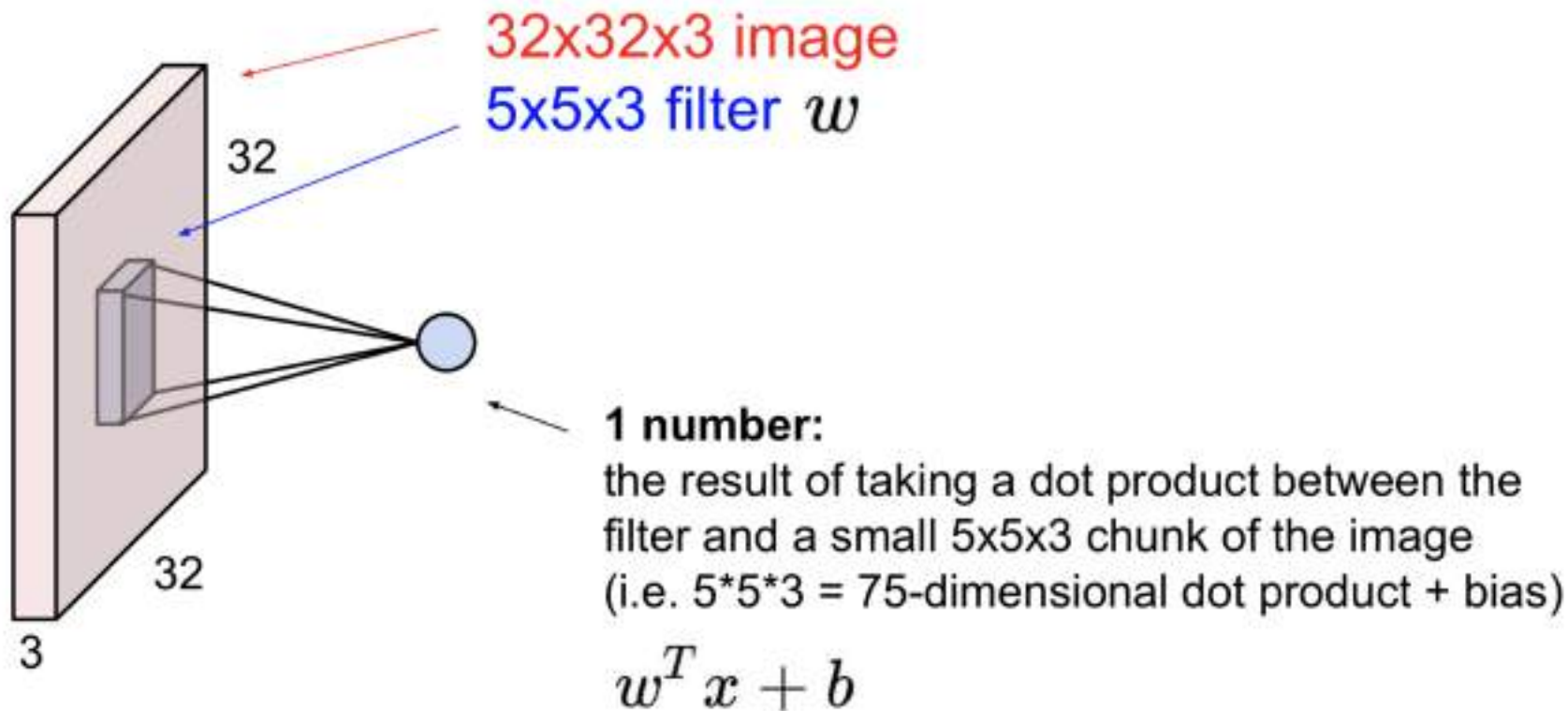


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

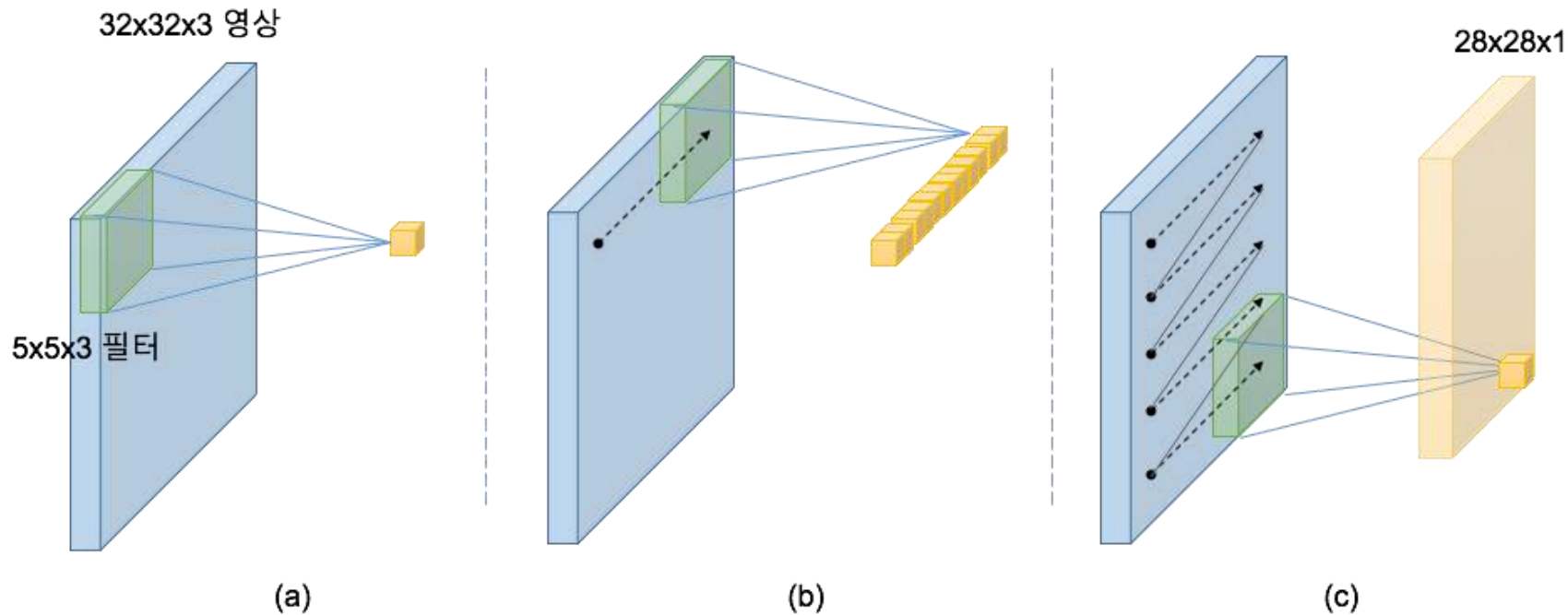
Convolution Layer



Convolution Layer

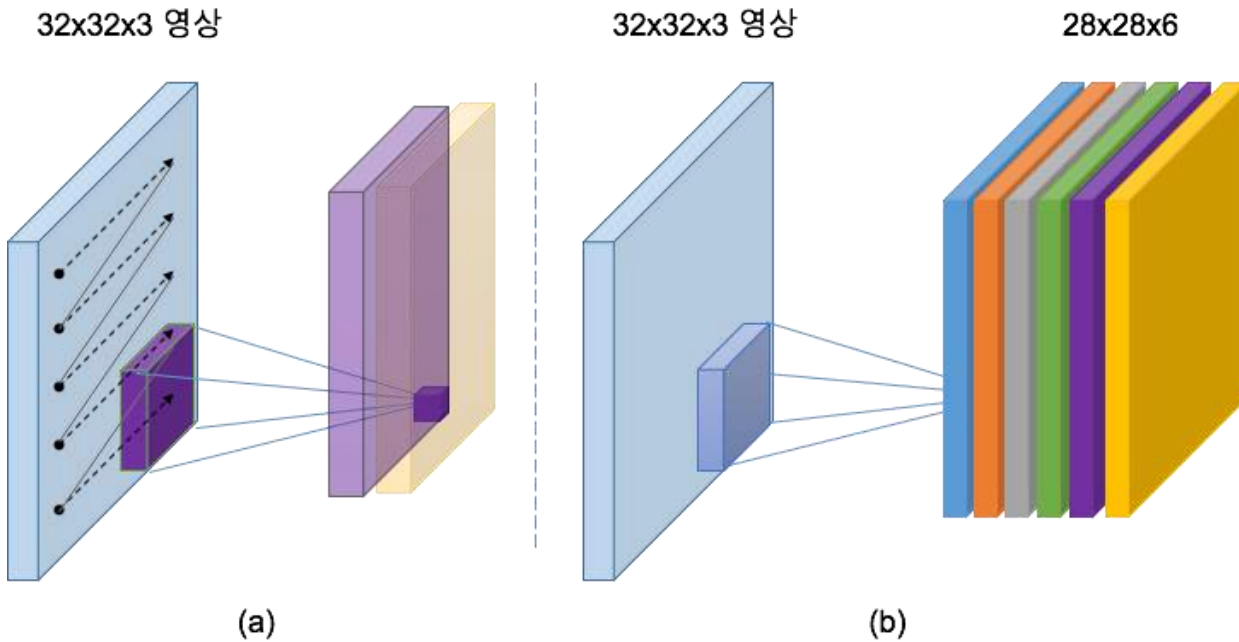


Convolution Layer

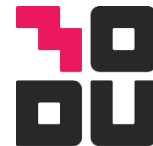


Convolution Layer

똑같은 크기의 필터 6개를 더 만들어 봅시다

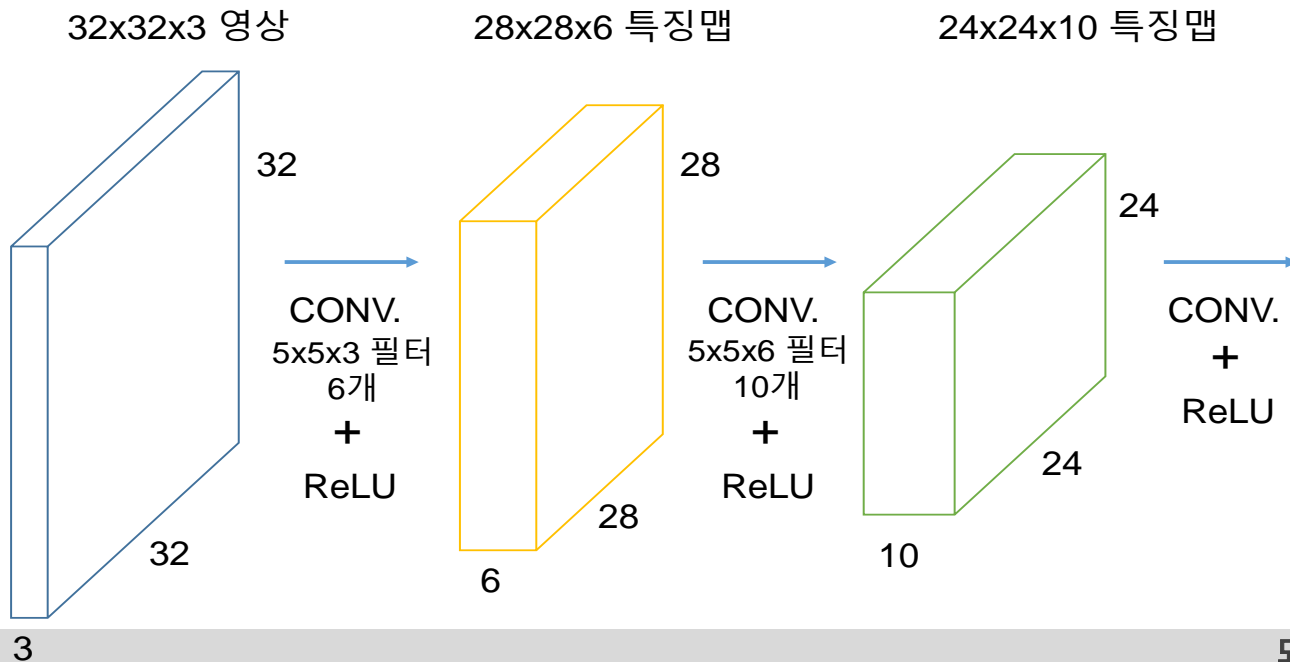


Convolution Layer



모두의연구소

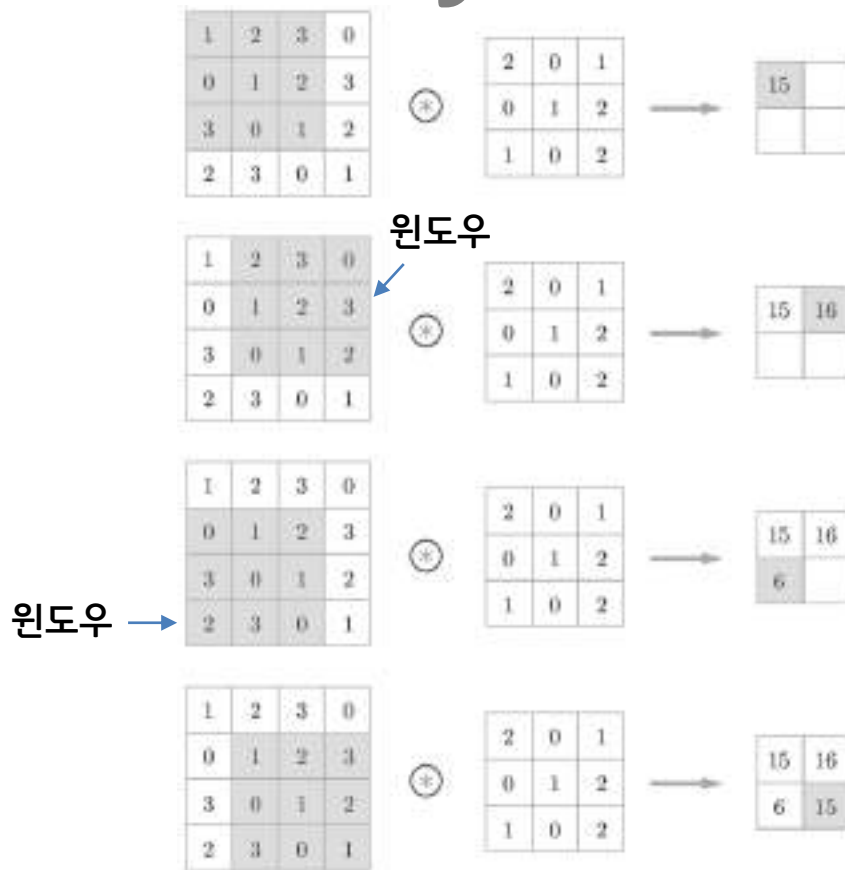
컨볼루션 네트워크는 활성화 함수를 포함한 컨볼루션
레이어의 연결입니다



Convolution Layer

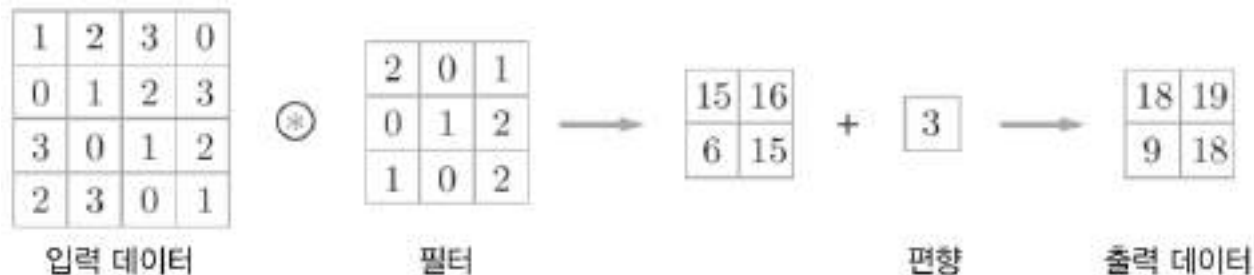
- 컨볼루션 연산

- 1) 윈도우를 일정 간격으로 이동해가며 입력 데이터에 적용
- 2) 입력과 필터에 대응하는 원소끼리 곱한 후 그 총합을 함 (단일 곱셈-누산 (fused multiply-add, FMA))
- Bias 파라미터 존재함



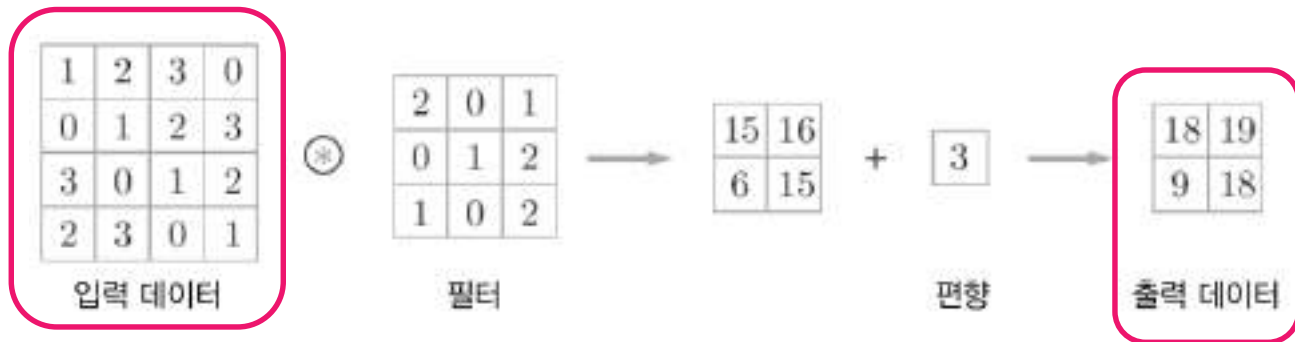
Convolution Layer

- 컨볼루션 연산



Convolution Layer

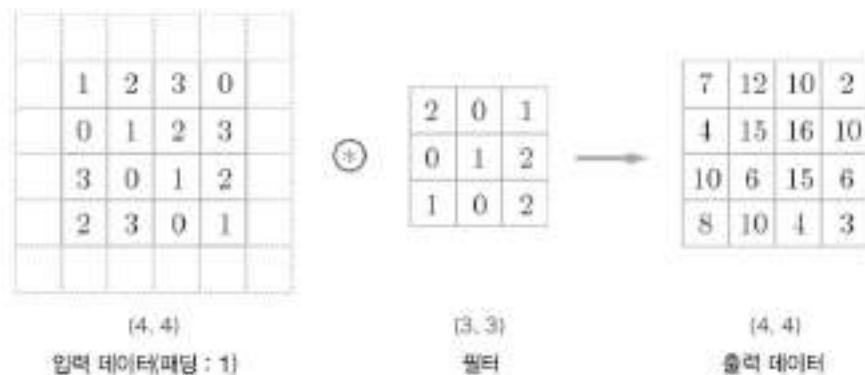
- 컨볼루션 연산



크기가 줄어드는 군요

Convolution Layer

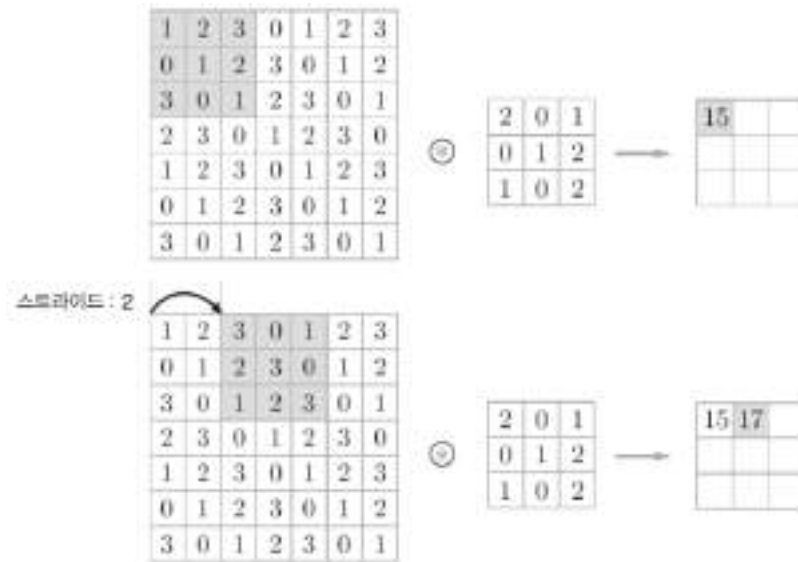
- 패딩 (padding)
 - 컨볼루션 연산의 패딩 처리 : 입력 데이터 주위에 0을 채운다 (패딩은 점선으로 표시했으며 그 안의 값 '0'은 생략함)



- 패딩은 출력(특징맵(feature map))의 크기를 유지 시키고자 할때 주로 사용함

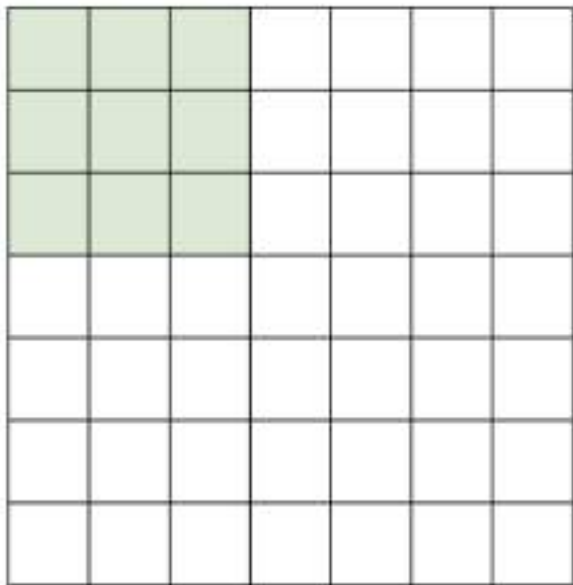
Convolution Layer

- 스트라이드 (stride)
 - 필터 적용하는 위치의 간격
 - 스트라이드를 2로 하면 필터를 적용하는 윈도우가 두 칸씩 이동함
- 스트라이드를 2로 하니 출력
이 3x3이 됨



특징맵의 크기변화

7

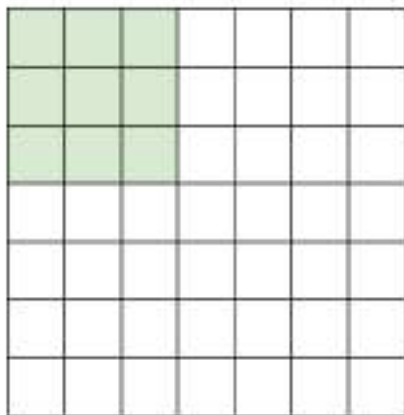


7x7 input (spatially)
assume 3x3 filter

7

특징맵의 크기변화

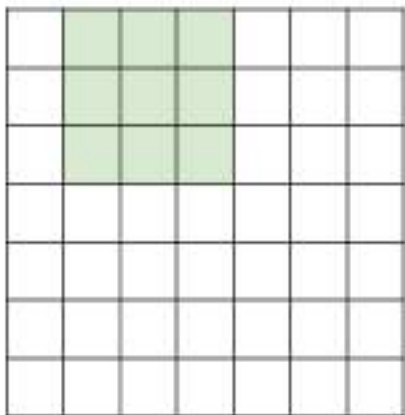
Replicate this column of hidden neurons across space, with some **stride**.



7x7 input
assume 3x3 connectivity, stride 1

특징맵의 크기변화

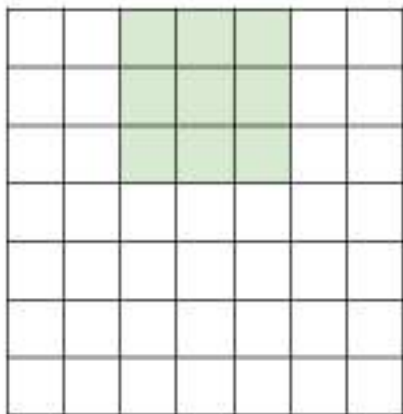
Replicate this column of hidden neurons across space, with some **stride**.



7x7 input
assume 3x3 connectivity, stride 1

특징맵의 크기변화

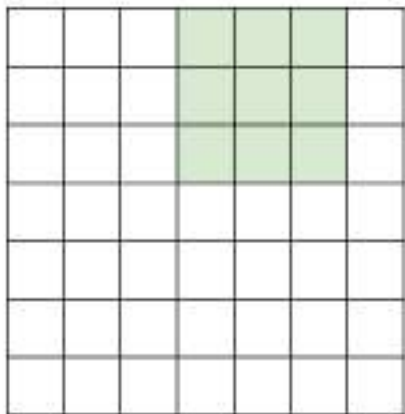
Replicate this column of hidden neurons across space, with some **stride**.



7x7 input
assume 3x3 connectivity, stride 1

특징맵의 크기변화

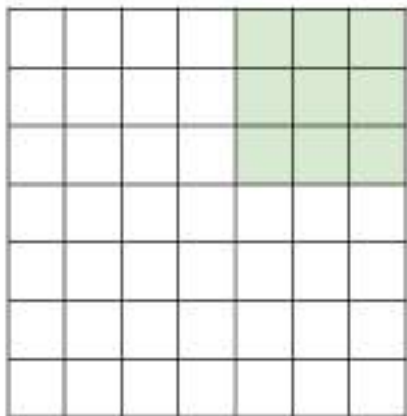
Replicate this column of hidden neurons across space, with some **stride**.



7x7 input
assume 3x3 connectivity, stride 1

특징맵의 크기변화

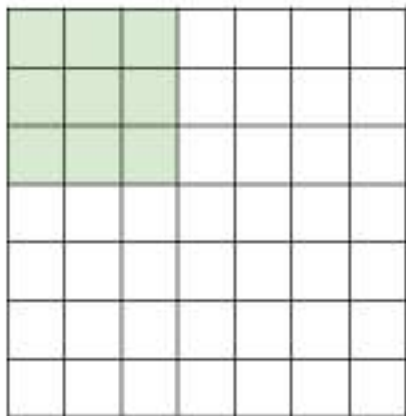
Replicate this column of hidden neurons across space, with some **stride**.



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

특징맵의 크기변화

Replicate this column of hidden neurons across space, with some **stride**.

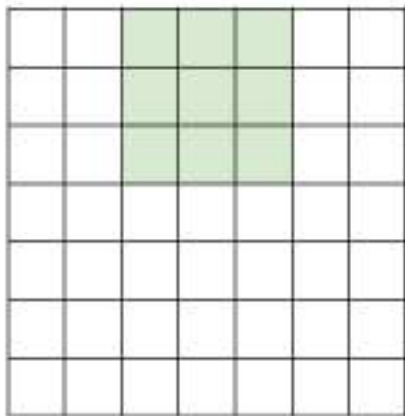


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

특징맵의 크기변화

Replicate this column of hidden neurons across space, with some **stride**.

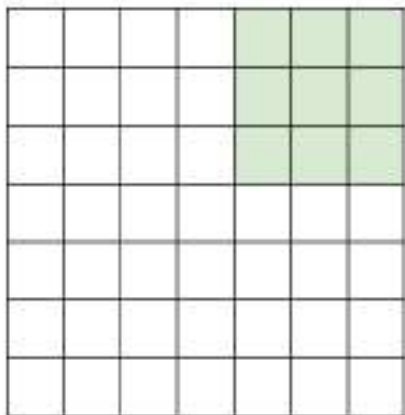


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

특징맵의 크기변화

Replicate this column of hidden neurons across space, with some **stride**.

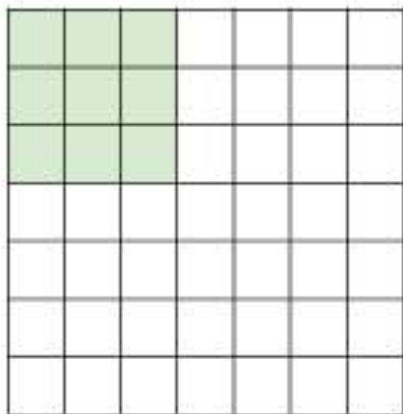


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

특징맵의 크기변화

Replicate this column of hidden neurons across space, with some **stride**.

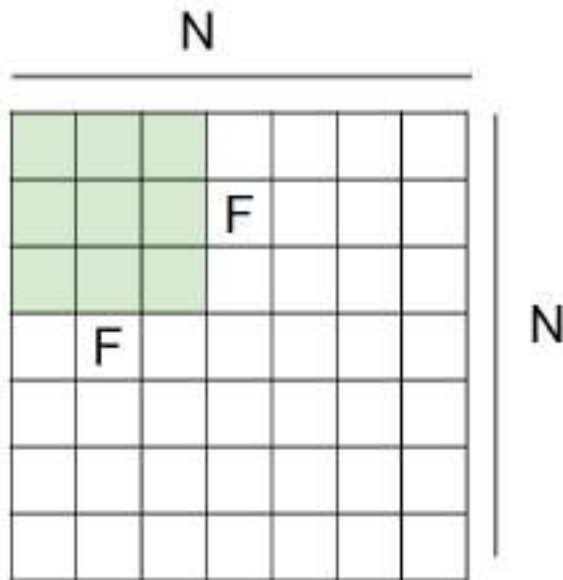


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

what about stride 3? **Cannot.**

특징맵의 크기변화



Output size:
 $(N - F) / \text{stride} + 1$

e.g. $N = 7, F = 3$:
stride 1 $\Rightarrow (7 - 3) / 1 + 1 = 5$
stride 2 $\Rightarrow (7 - 3) / 2 + 1 = 3$
stride 3 $\Rightarrow (7 - 3) / 3 + 1 = \dots \backslash$

특징맵의 크기변화

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

특징맵의 크기변화

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

특징맵의 크기변화

- 출력크기 계산해 보기

- 입력크기 : (H, W)
- 필터크기 : (FH, FW)
- 출력크기 : (OH, OW)
- 패딩 : P
- 스트라이드 : S

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

특징맵의 크기변화

- 출력크기 계산해 보기

입력 : (4,4), 패딩 : 1 , 스트라이드 : 1, 필터: (3,3)

$$OH = \frac{4 + 2 \cdot 1 - 3}{1} + 1 = 4$$

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

	1	2	3	0	
	0	1	2	3	
	3	0	1	2	
	2	3	0	1	

(4, 4)

입력 데이터(패딩 : 1)



2	0	1
0	1	2
1	0	2

(3, 3)

필터



7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

(4, 4)

출력 데이터

특징맵의 크기변화

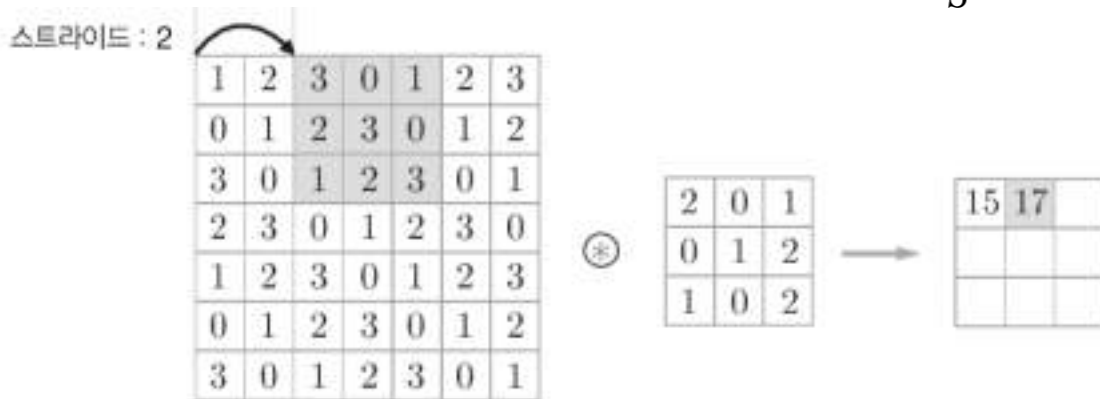
- 출력크기 계산해 보기

입력 : (7,7), 패딩 : 0 , 스트라이드 : 2, 필터: (3,3)

$$OH = \frac{7 + 2 \cdot 0 - 3}{2} + 1 = 3$$

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$



특징맵의 크기변화

- 출력크기 계산해 보기

입력 : (28, 31), 패딩 : 2 , 스트라이드 : 3, 필터: (5, 5)

$OH = ?$

$$OH = \frac{H + 2P - FH}{S} + 1$$

$OW = ?$

$$OW = \frac{W + 2P - FW}{S} + 1$$

특징맵의 크기변화

- 출력크기 계산해 보기

입력 : (28, 31), 패딩 : 2 , 스트라이드 : 3, 필터: (5, 5)

$$OH = \frac{28 + 2 \cdot 2 - 5}{3} + 1 = 10$$

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

$$OW = \frac{31 + 2 \cdot 2 - 5}{3} + 1 = 11$$

- OH , OW 가 정수가 아니면?**

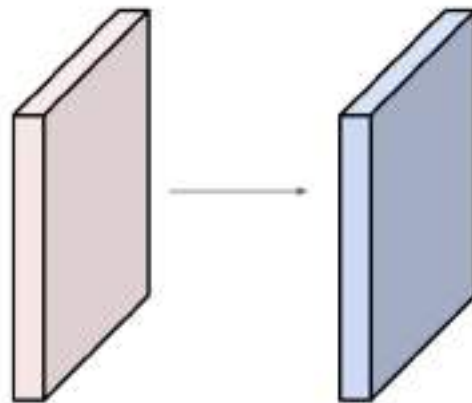
- 출력이 안나오는 것임. 오류를 내는 등의 대응 필요
- 딥러닝 프레임웍은 가까운 정수로 내림 하는 등, 특별히 에러를 내지않고 진행되도록 구현되는 경우가 많음

특징맵의 크기변화

Examples time:

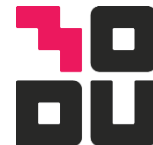
Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

특징맵의 크기변화

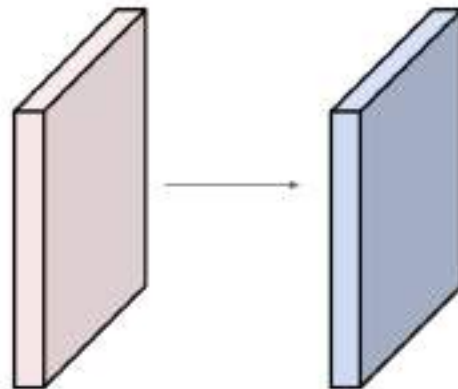


모두의연구소

Examples time:

Input volume: $32 \times 32 \times 3$

10 5×5 filters with stride 1, pad 2



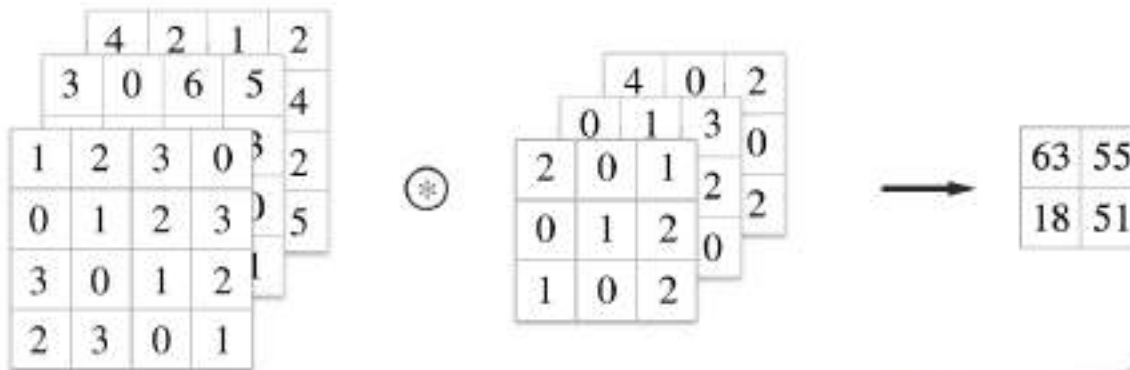
Number of parameters in this layer?

each filter has $5 \times 5 \times 3 + 1 = 76$ params (+1 for bias)

$\Rightarrow 76 \times 10 = 760$

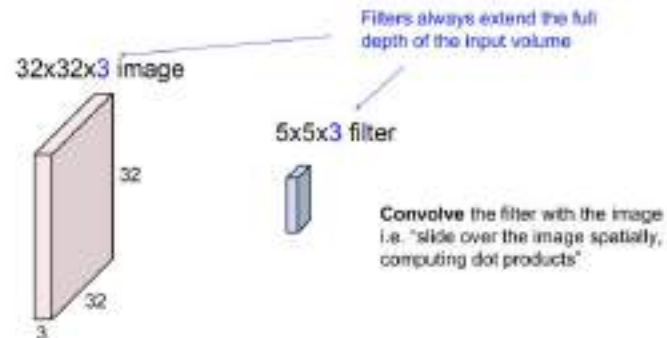
Convolution Layer

- 필터의 채널 수는 입력되는 특징맵의 채널 수와 일치해야 함을 잊지 마세요

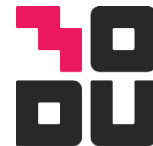


입력 데이터

필터

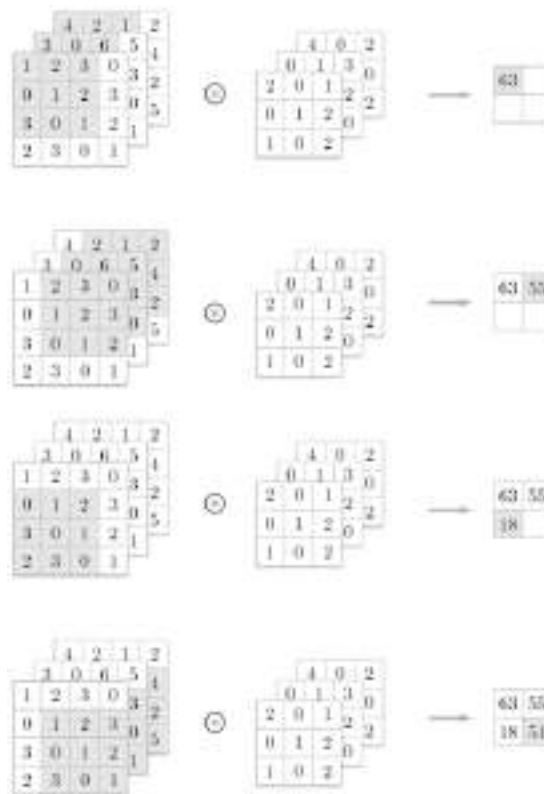


Convolution Layer



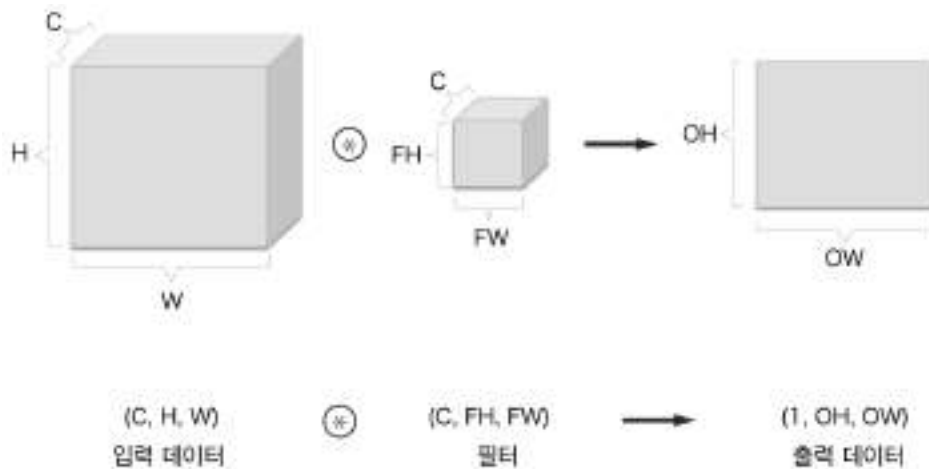
모두의연구소

- 3차원 데이터의 컨볼루션
 - 주의할 점은 입력데이터의 채널 수와 필터의 채널 수가 같아야 한다는 점
 - 각 필터의 채널크기는 같아야 함



Convolution Layer

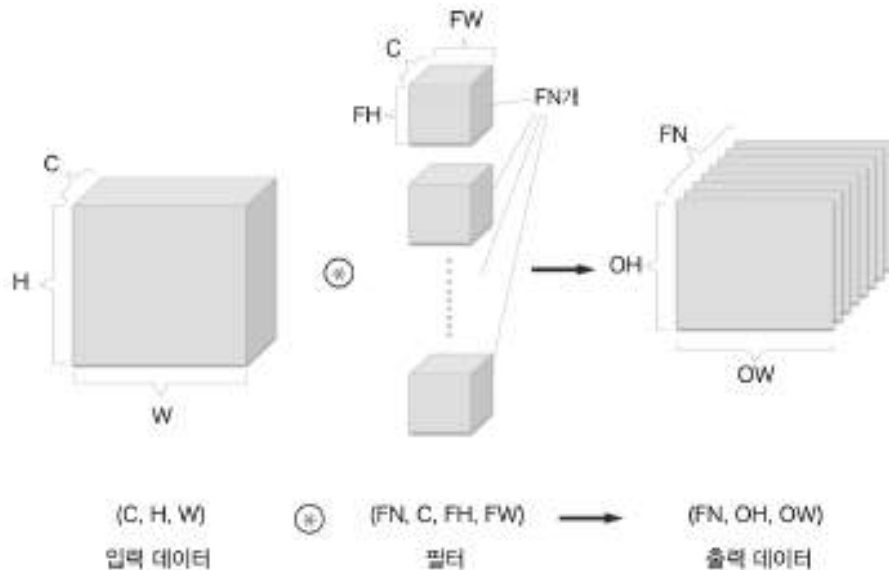
- 블록으로 생각하기



- 출력 데이터는 한장의 특징맵
- 한장이 아니라 여러장, 즉 다수의 채널을 내보내려면?

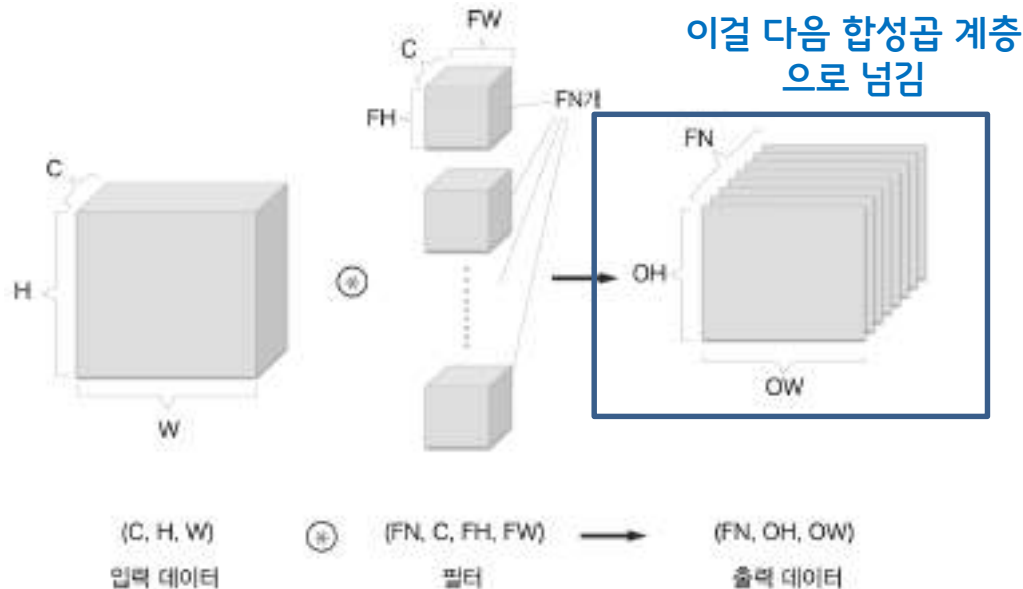
Convolution Layer

- 블록으로 생각하기
 - 한장이 아니라 여러장, 즉 다수의 채널을 내보내려면?
 - 필터를 여러개 사용하면 됨



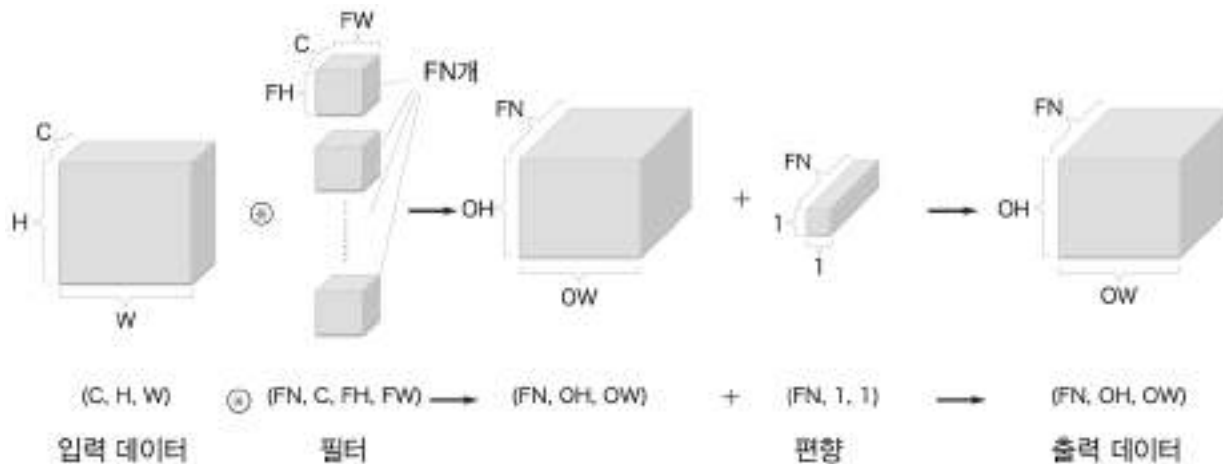
Convolution Layer

- 블록으로 생각하기
 - 한장이 아니라 여러장, 즉 다수의 채널을 내보내려면?
 - 필터를 여러개 사용하면 됨



Convolution Layer

- 편향 (bias)

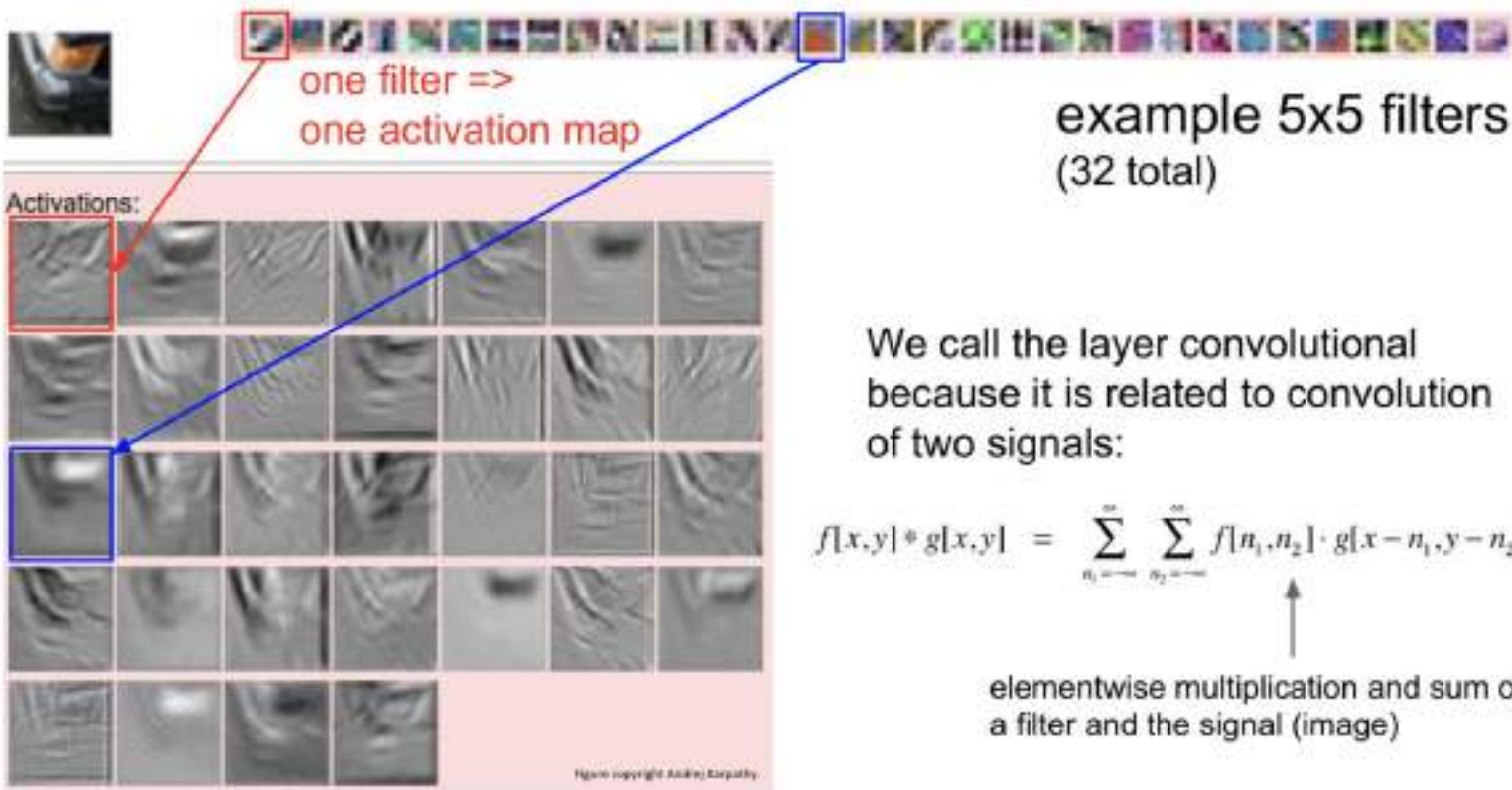


- 필터마다 편향이 하나씩 존재

Convolution Layer

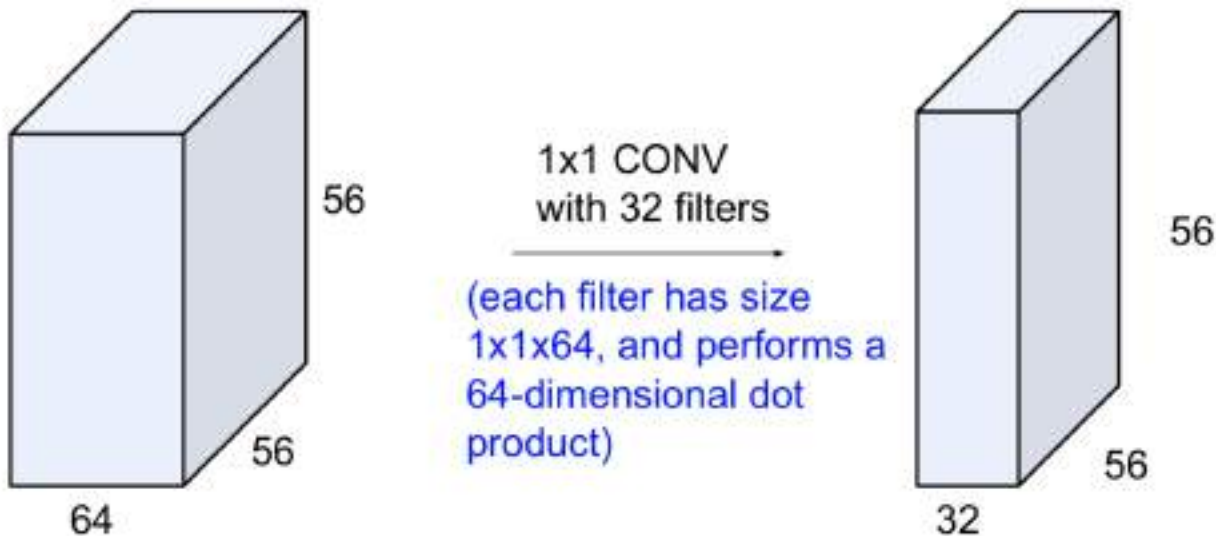


모두의연구소

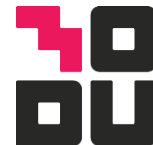


Convolution Layer

(btw, 1x1 convolution layers make perfect sense)



Convolution Layer



모두의연구소

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 of 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 owidth` where

```
width = floor((width + 2*padW - kW) / dW + 1)  
height = floor((height + 2*padH - kH) / dH + 1)
```

Torch is licensed under BSD 3-clause.

Convolution Layer

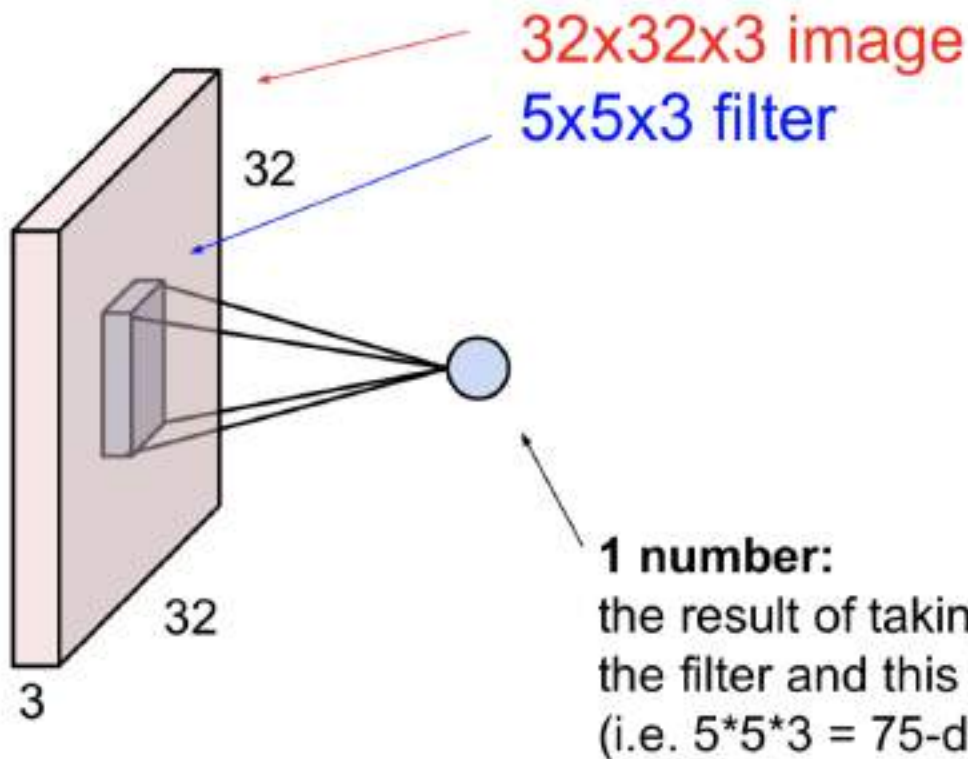
Example: CONV layer in Caffe

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 .

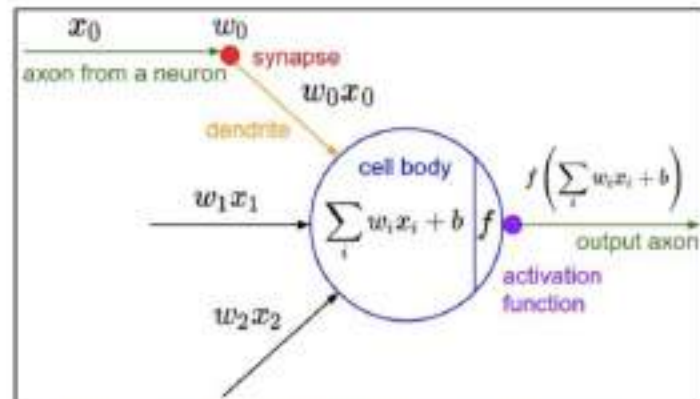
```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 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
    }
  }
}
```

The brain/neuron view of CONV Layer



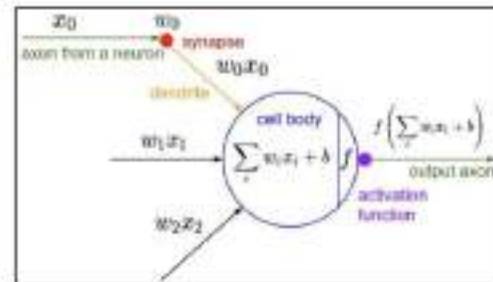
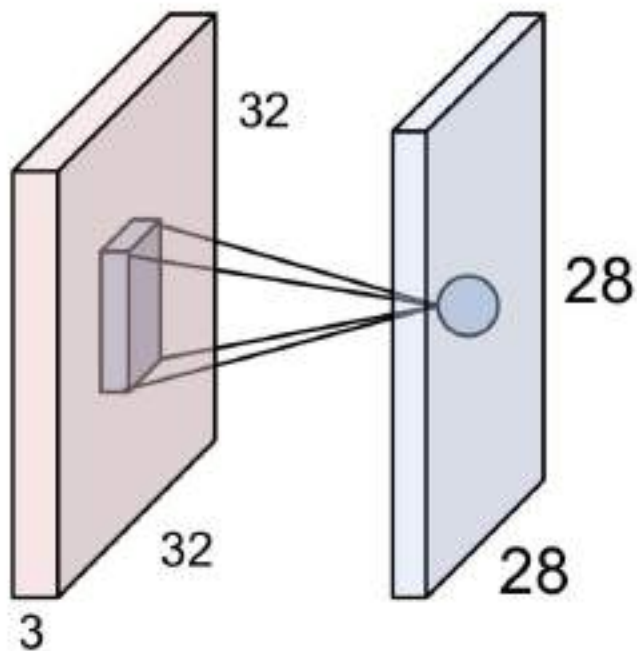
1 number:

the result of taking a dot product between the filter and this part of the image (i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product)



It's just a neuron with local connectivity...

The brain/neuron view of CONV Layer

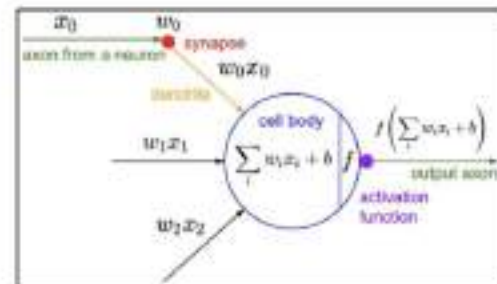
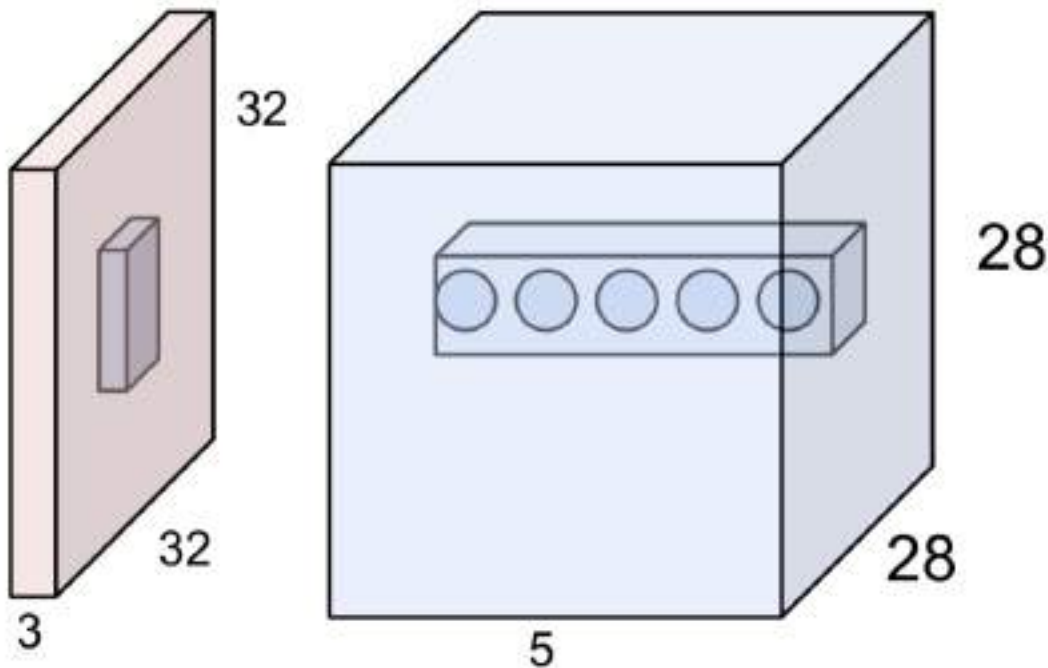


An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”

The brain/neuron view of CONV Layer



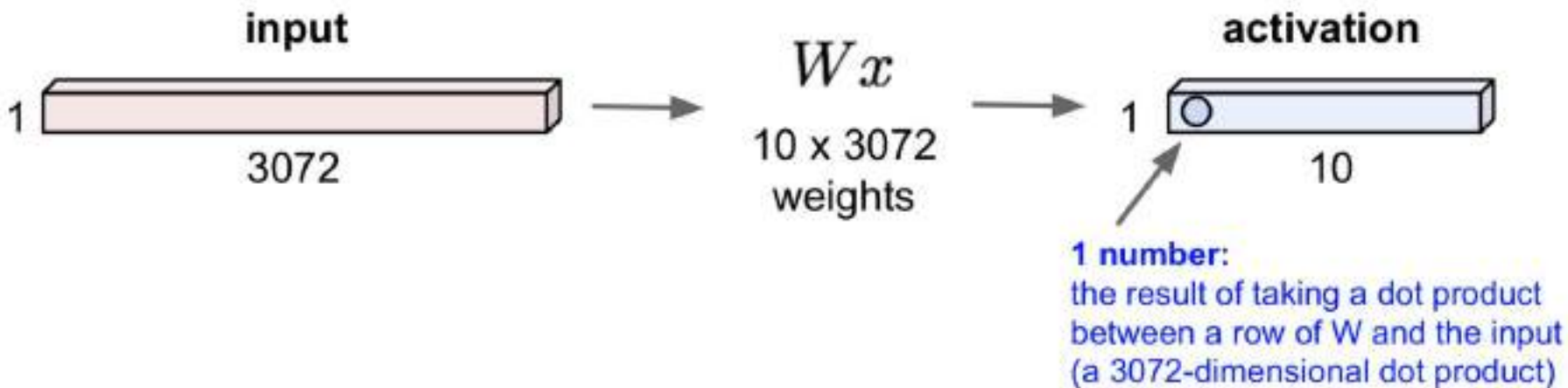
E.g. with 5 filters,
CONV layer consists of
neurons arranged in a 3D grid
(28x28x5)

There will be 5 different
neurons all looking at the same
region in the input volume

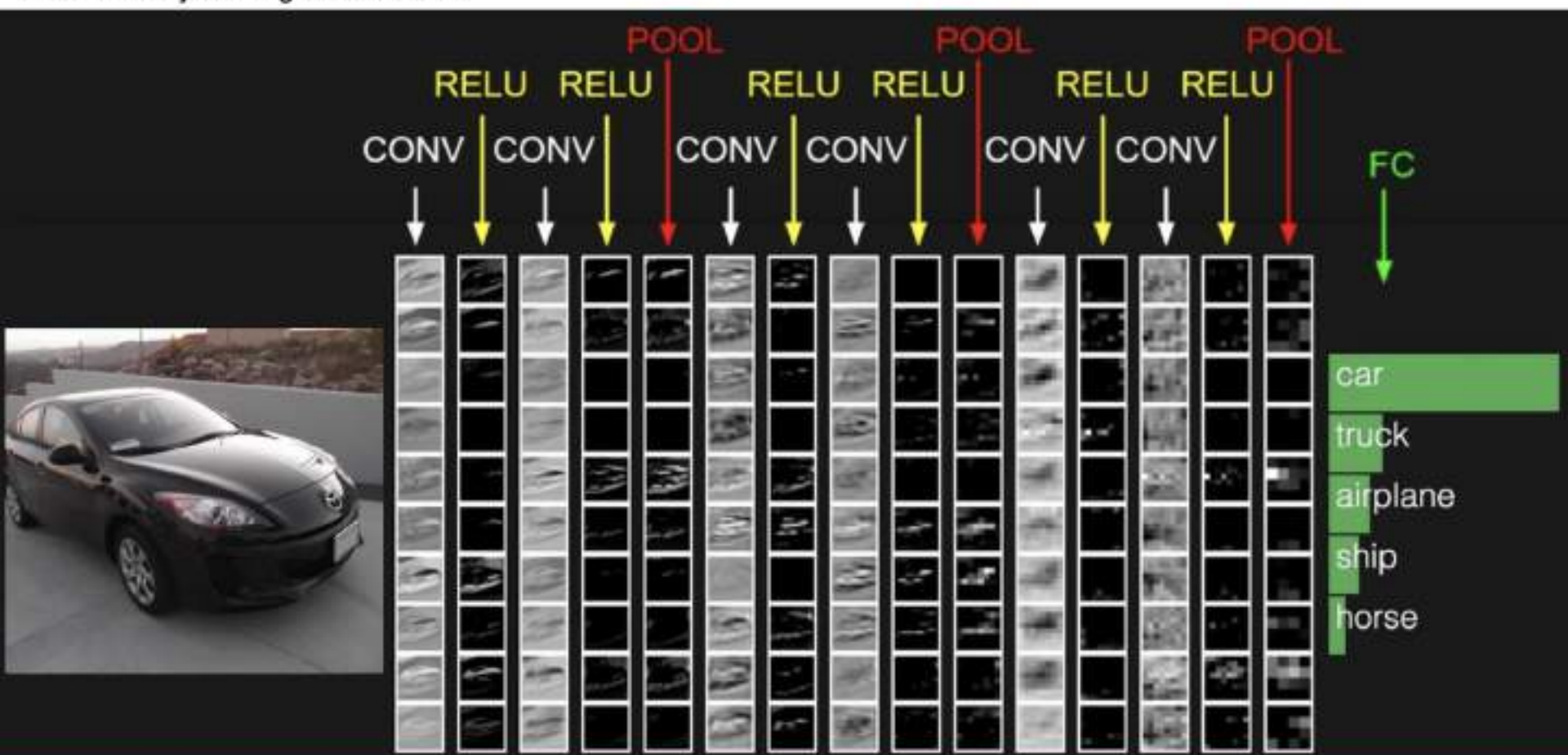
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron
looks at the full
input volume

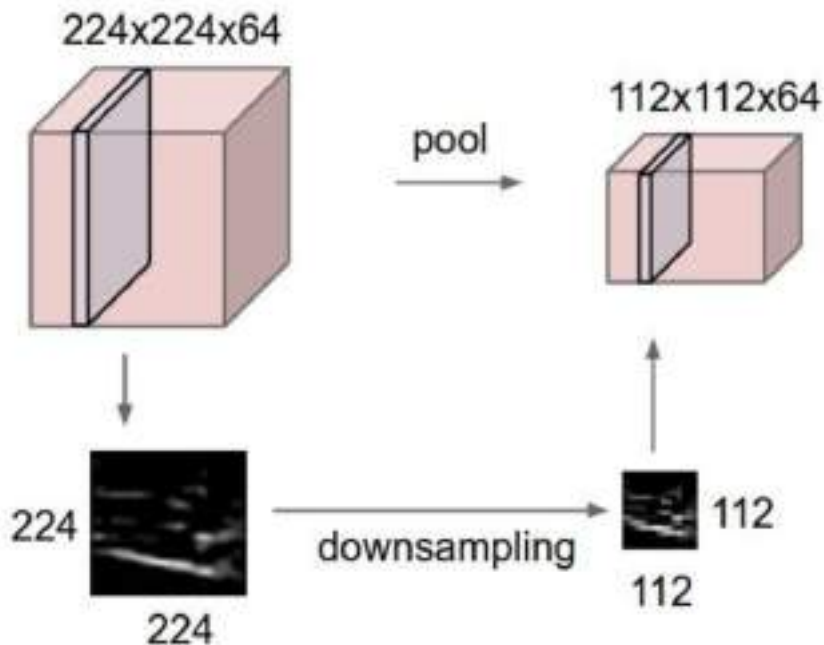


two more layers to go: POOL/FC



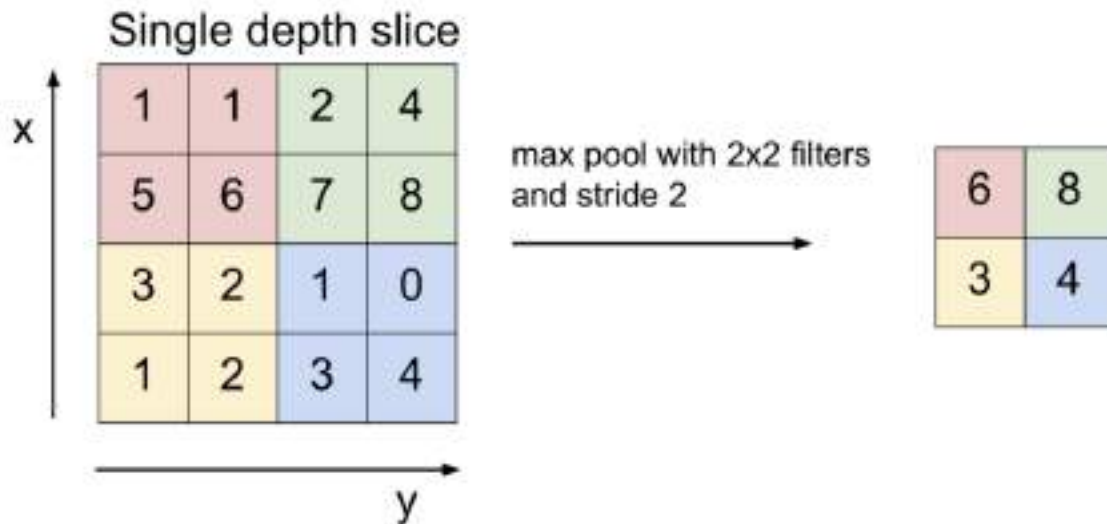
Pooling Layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



Pooling Layer

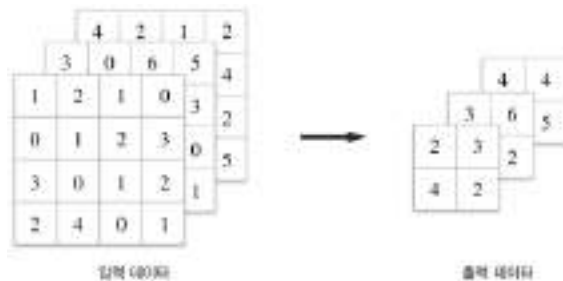
- 세로·가로 방향의 공간을 줄이는 연산
 - 2x2 최대 풀링(max pooling)을 스트라이드 2로



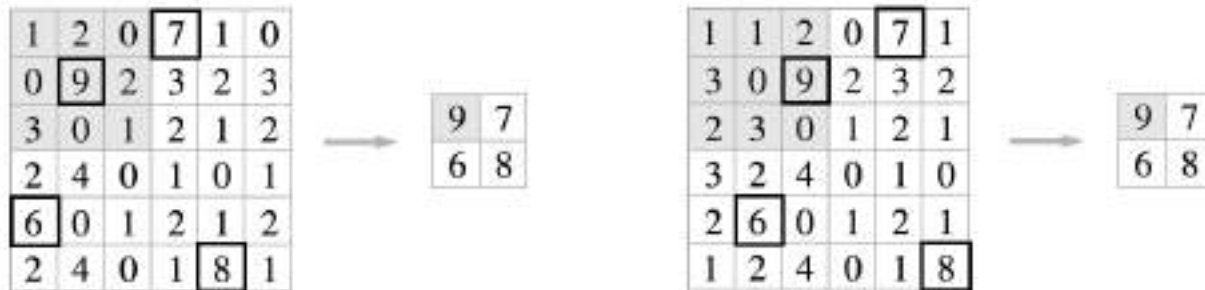
- 평균 풀링(Average pooling)도 있습니다

Pooling Layer

- 풀링 계층의 특징
 - 학습해야 할 매개변수가 없음
 - 채널 수가 변하지 않음
 - 입력의 변화에 영향을 적게 받음 (강건하다)



데이터가 오른쪽으로 1칸씩 이동한 경우



Pooling 후 특징맵 크기 변화

Common settings:

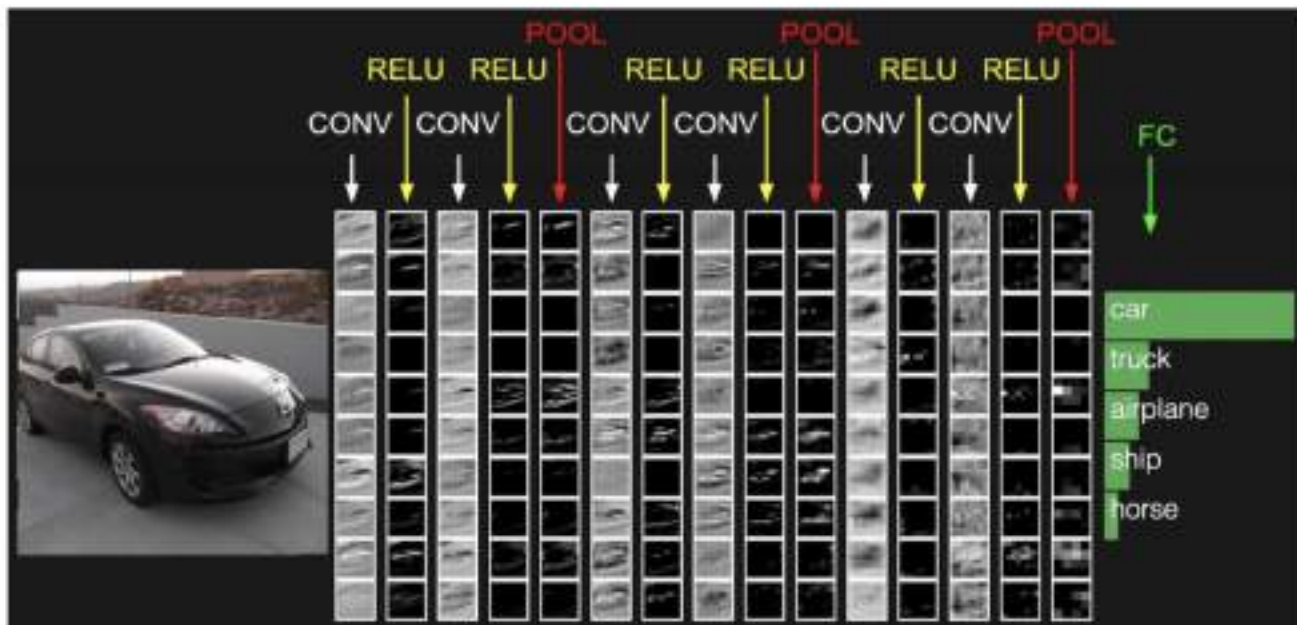
$$F = 2, S = 2$$

$$F = 3, S = 2$$

- 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

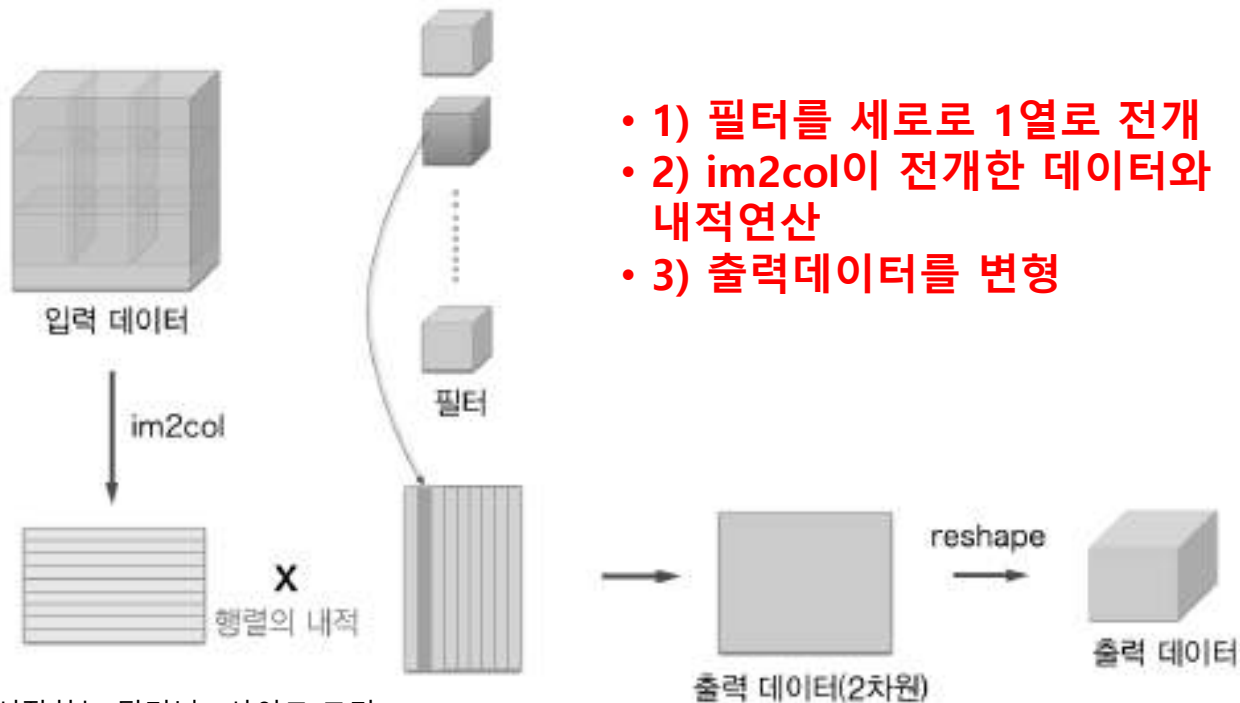
Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

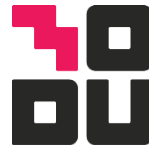


Convolutional Networks 구현

- im2col로 데이터 전개하기
 - 합성곱을 행렬 곱 연산으로 (Affine 계층연산으로)

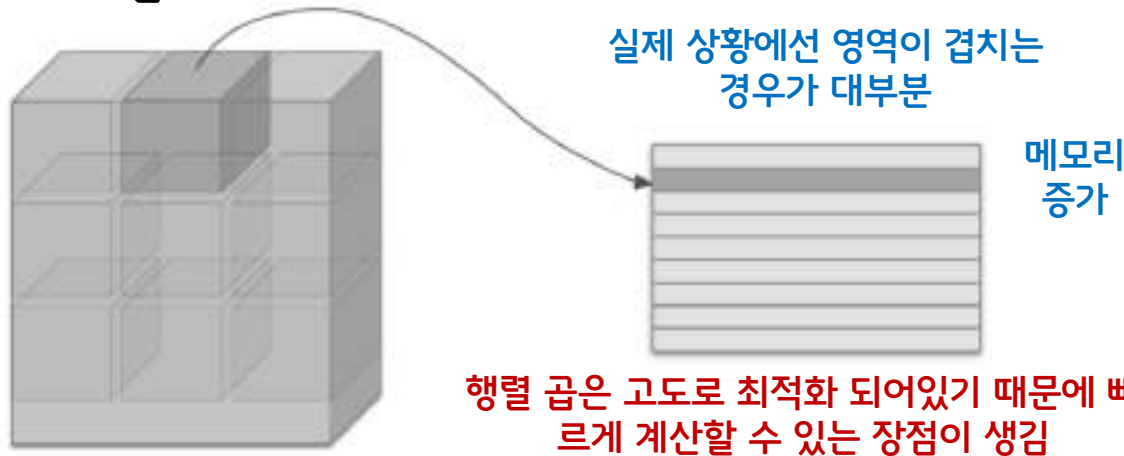


Convolutional Networks 구현



모두의연구소

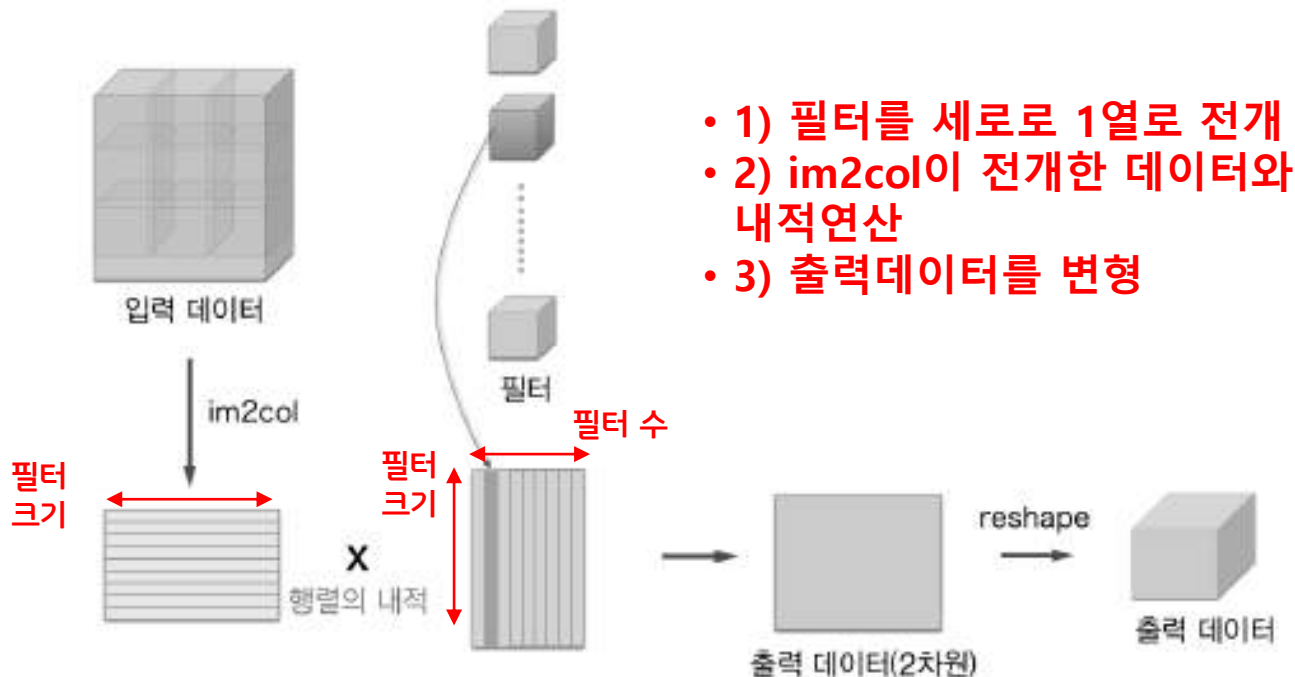
- im2col로 데이터 전개하기
 - 입력데이터에서 **필터를 적용하는 영역(3차원 블록)을 한줄로 늘어 놓습니다**
 - 이 전개를 필터를 적용하는 모든 영역에서 수행하는게 im2col입니다



필터 적용 영역을 앞에서부터 순서대로 1줄로 펼친다

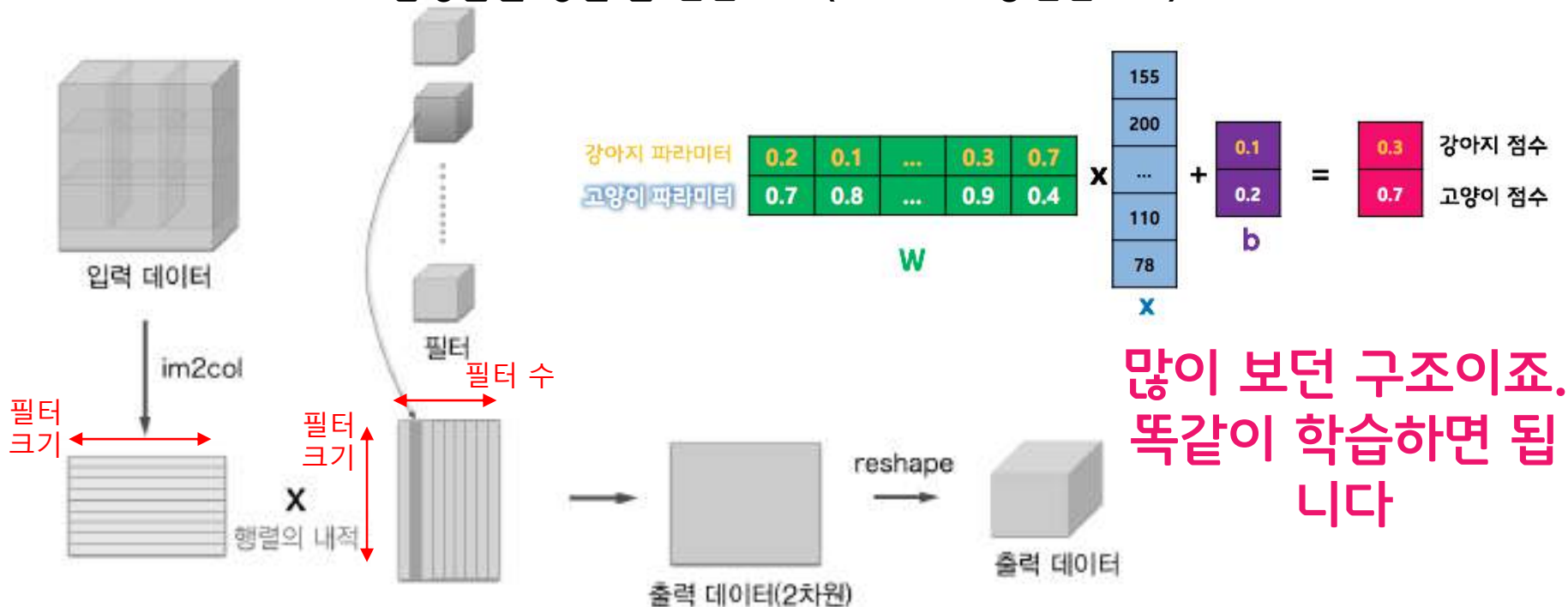
Convolutional Networks 구현

- im2col로 데이터 전개하기
 - 합성곱을 행렬 곱 연산으로 (Affine 계층연산으로)



Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - 합성곱을 행렬 곱 연산으로 (Affine 계층연산으로)



Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - im2col : common/util.py

```
39 def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
40     """다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화).
41
42     Parameters
43     -----
44     input_data : 4차원 배열 형태의 입력 데이터(이미지 수, 채널 수, 높이, 너비)
45     filter_h : 필터의 높이
46     filter_w : 필터의 너비
47     stride : 스트라이드
48     pad : 패딩
49
50     Returns
51     -----
52     col : 2차원 배열
53     """
```

Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - im2col : 3_day/ex_im2col.py

```
1 import sys, os
2 sys.path.append(os.pardir)
3 from common.util import im2col
4
5 x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
6 col1 = im2col(x1, 5, 5, stride=1, pad=0)
7 print(col1.shape) # (9, ?)
8
9 x2 = np.random.rand(10, 3, 7, 7)
10 col2 = im2col(x2, 5, 5, stride=1, pad=0)
11 print(col2.shape) # (?, ?)
```

Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - im2col : 3_day/ex_im2col.py

```
1 import sys, os
2 sys.path.append(os.pardir)
3 from common.util import im2col
4
5 x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
6 col1 = im2col(x1, 5, 5, stride=1, pad=0) ← 5x5 필터
7 print(col1.shape) # (9, ?)
8
9 x2 = np.random.rand(10, 3, 7, 7)
10 col2 = im2col(x2, 5, 5, stride=1, pad=0)
11 print(col2.shape) # (?, ?)
```

데이터 : (1, 3, 7, 7)

5x5 필터

Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - im2col : 3_day/ex_im2col.py

```
1 import sys, os
2 sys.path.append(os.pardir)
3 from common.util import im2col
4
5 x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
6 col1 = im2col(x1, 5, 5, stride=1, pad=0) ← 5x5 필터
7 print(col1.shape) # (9, ?)
8
9 x2 = np.random.rand(10, 3, 7, 7)
10 col2 = im2col(x2, 5, 5, stride=1, pad=0)
11 print(col2.shape) # (?, ?)
```

(9, ?)

Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - im2col : 3_day/ex_im2col.py

```
1 import sys, os
2 sys.path.append(os.pardir)
3 from common.util import im2col
4
5 x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
6 col1 = im2col(x1, 5, 5, stride=1, pad=0) ← 5x5 필터
7 print(col1.shape) # (9, ?)
8
9 x2 = np.random.rand(10, 3, 7, 7)
10 col2 = im2col(x2, 5, 5, stride=1, pad=0)
11 print(col2.shape) # (?, ?)
```

(9, 75)

Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - im2col : 3_day/ex_im2col.py

```
1 import sys, os
2 sys.path.append(os.pardir)
3 from common.util import im2col
4
5 x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
6 col1 = im2col(x1, 5, 5, stride=1, pad=0) ← 5x5 필터
7 print(col1.shape) # (9, ?)
8
9 x2 = np.random.rand(10, 3, 7, 7) ← 데이터 수만 10개로 늘어남
10 col2 = im2col(x2, 5, 5, stride=1, pad=0)
11 print(col2.shape) # (?, ?)
```

(?, ?)

Convolution 레이어 구현하기

- im2col로 데이터 전개하기
 - im2col : 3_day/ex_im2col.py

```
1 import sys, os
2 sys.path.append(os.pardir)
3 from common.util import im2col
4
5 x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
6 col1 = im2col(x1, 5, 5, stride=1, pad=0) ← 5x5 필터
7 print(col1.shape) # (9, ?)
8
9 x2 = np.random.rand(10, 3, 7, 7) ← 데이터 수만 10개로 늘어남
10 col2 = im2col(x2, 5, 5, stride=1, pad=0)
11 print(col2.shape) # (?, ?)
```

(90, 75)

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

?	?	
?	?	

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

?		

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

9	?	

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

9	13	

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

9	13	
21	25	

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

	17	
9		
21	25	

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

	17	25
9		
21	25	

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

결과를 채워
봅시다

	17	25
9	41	49
21	25	

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

2x2 필터 2개 : W bias 2개 : b

- Stride = 1
- Padding = 0

최종결과

		17	25
		41	49
9	13		
21	25		

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

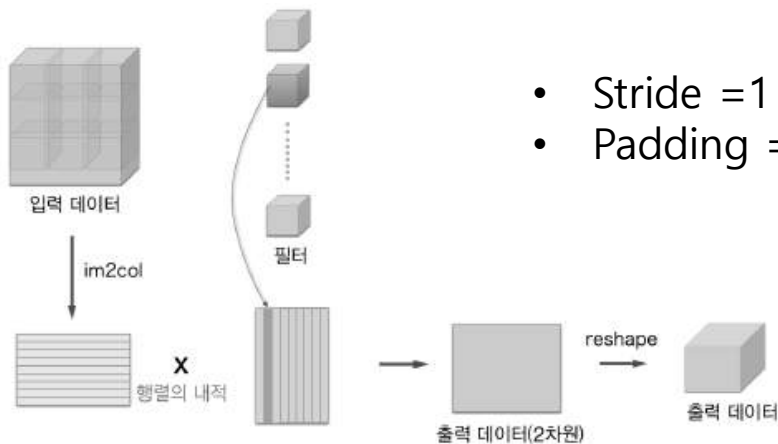
3x3 입력 데이터 : x

*

1	1		1
1	1		
2	2		1
2	2		

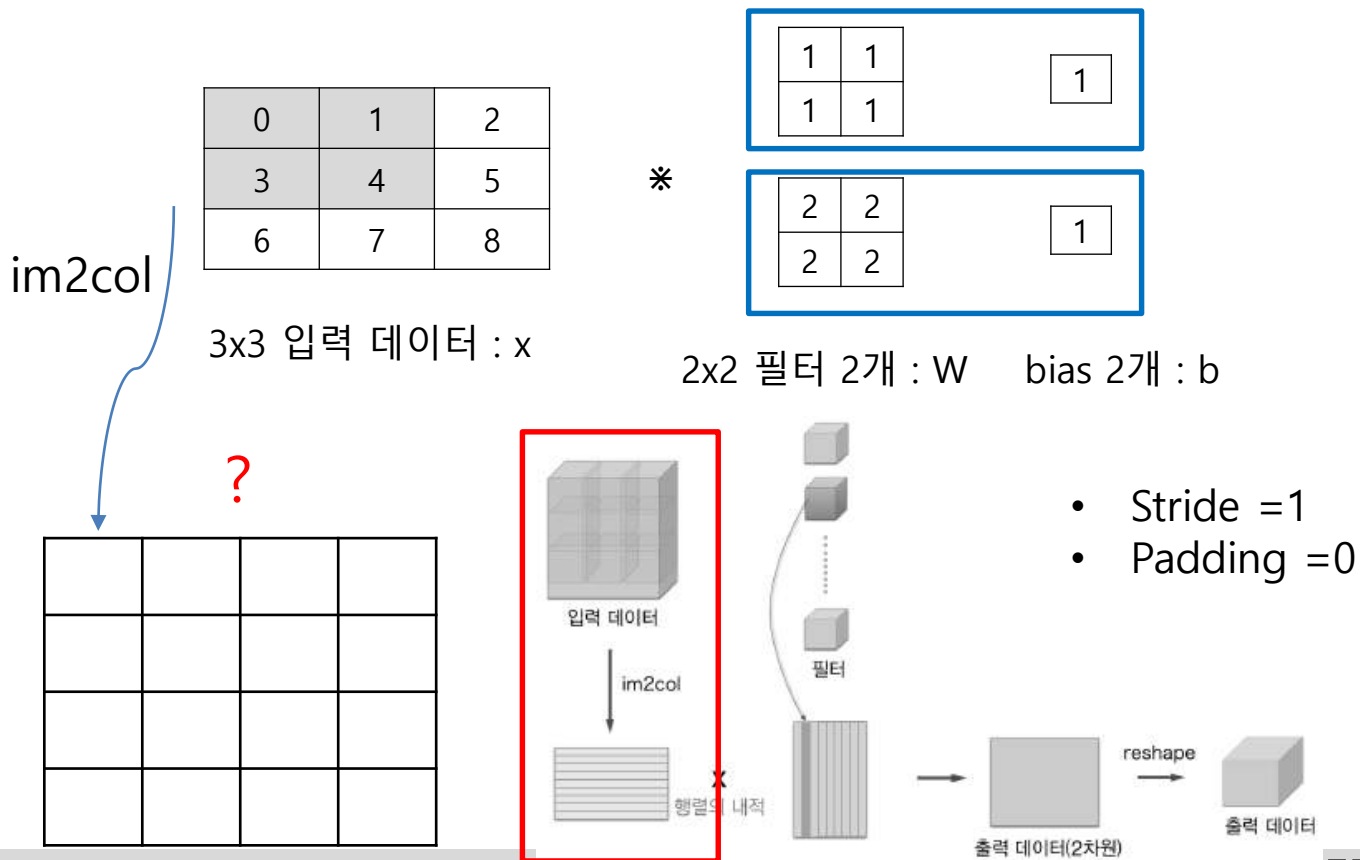
2x2 필터 2개 : W bias 2개 : b

이 방식으로
해봅시다

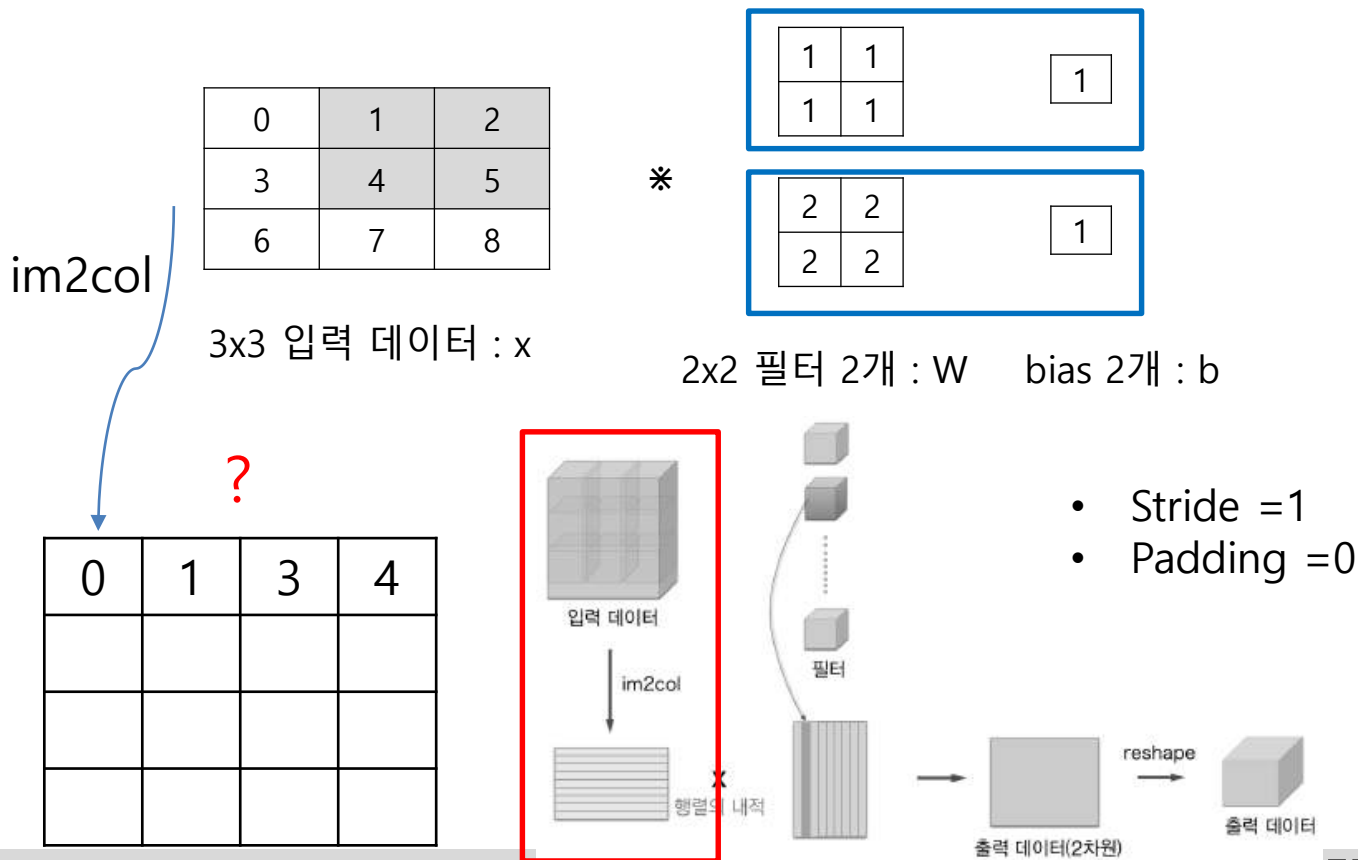


- Stride = 1
- Padding = 0

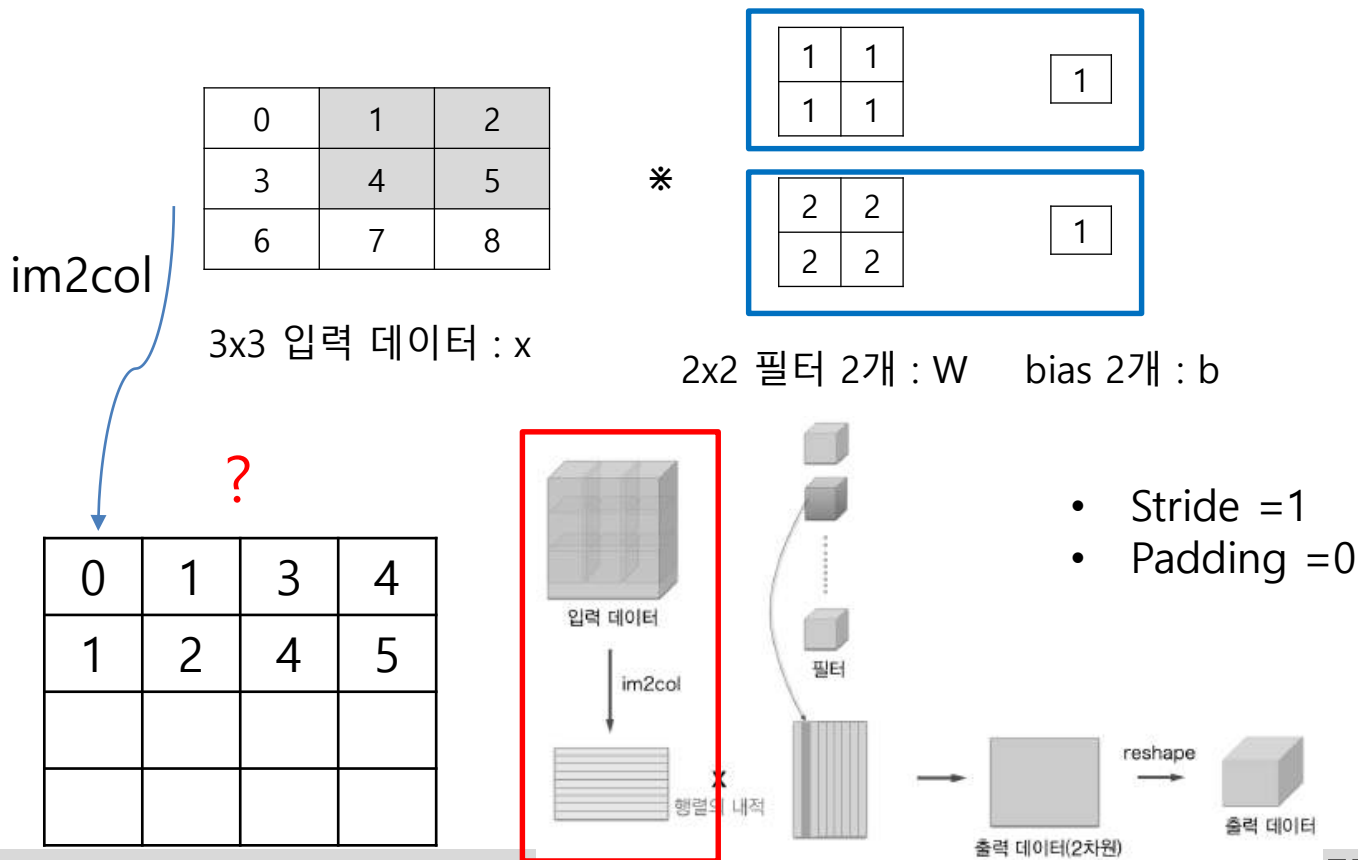
완전 단순 Conv 구현해보기



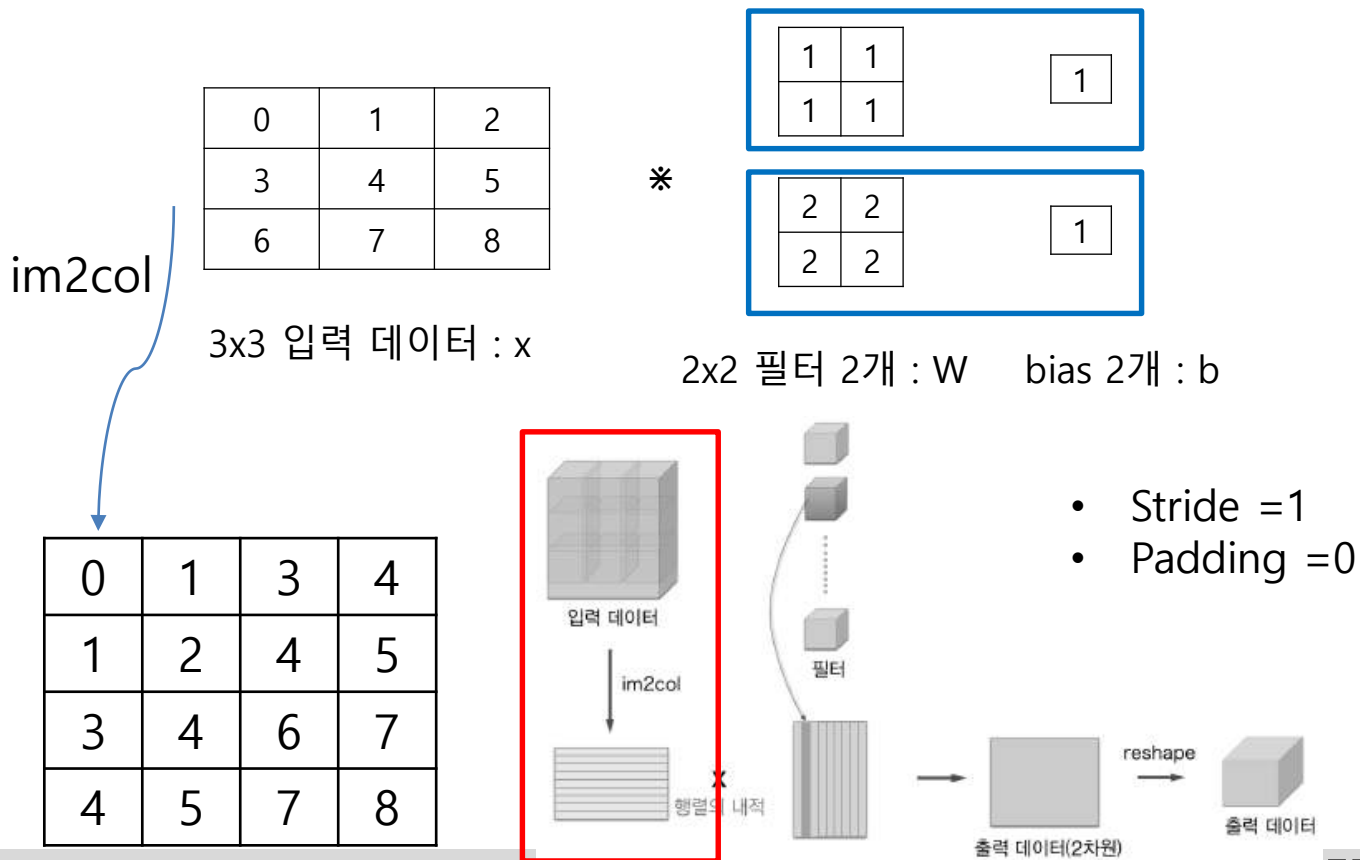
완전 단순 Conv 구현해보기



완전 단순 Conv 구현해보기



완전 단순 Conv 구현해보기



완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

*

1	1		1
1	1		

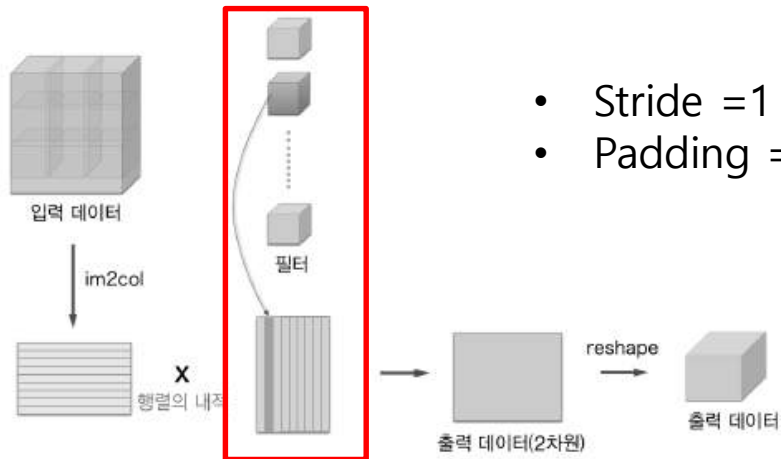
2	2		1
2	2		

3x3 입력 데이터 : x

Bias 일단 무시

2x2 필터 2개 : W bias 2개 : b

?



- Stride = 1
- Padding = 0

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

*

1	1		1
1	1		
2	2		1
2	2		

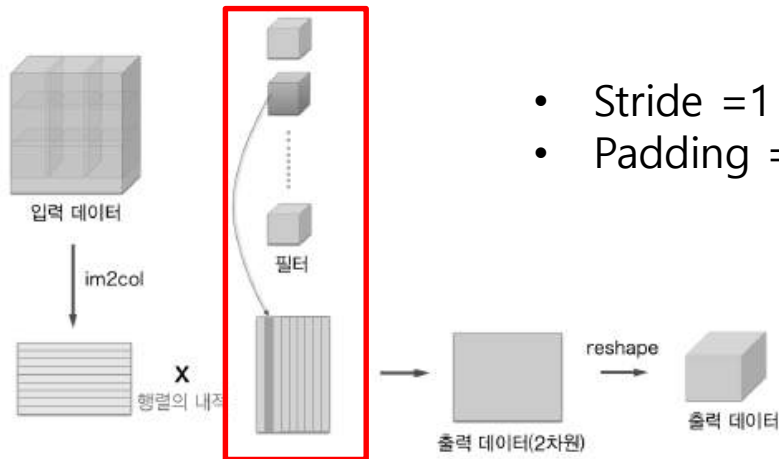
3x3 입력 데이터 : x

Bias 일단 무시

2x2 필터 2개 : W bias 2개 : b

?

1	
1	
1	
1	



- Stride = 1
- Padding = 0

완전 단순 Conv 구현해보기

0	1	2
3	4	5
6	7	8

*

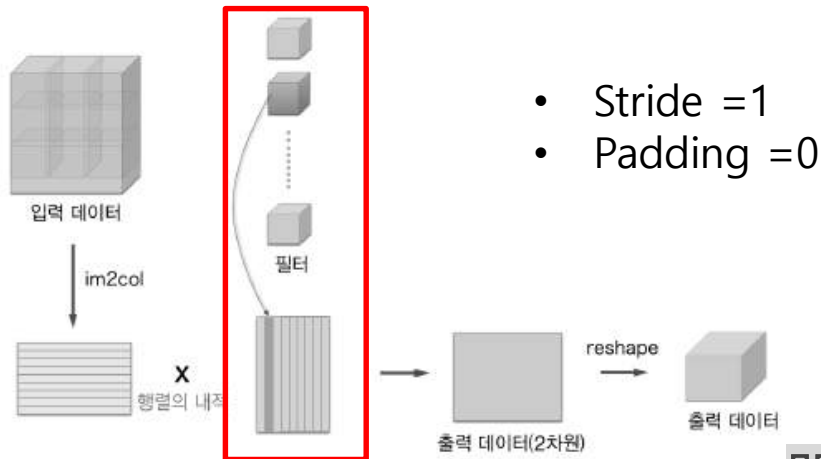
1	1		1
1	1		
2	2		1
2	2		

3x3 입력 데이터 : x

Bias 일단 무시

2x2 필터 2개 : W bias 2개 : b

1	2
1	2
1	2
1	2



완전 단순 Conv 구현해보기



모두의연구소

- 이제 Affine 계층을 이용해서 구현해 봅시다
 - 3_day/prac1_toyConv.py

```
6  ### 초기값 설정
7  x = list(range(1*3*3))
8  x = np.array(x).reshape(1, 1, 3, 3) # (데이터 수, 채널 수, 높이, 너비)
9  W = np.ones((2, 1, 2, 2)) # (필터 수, 필터 채널, 필터 높이, 필터 너비)
10 W[1] = W[1]*2
11 b = np.ones((2,)) # (바이어스 수)
12 pad = 0
13 stride = 1
14 ###
15 N_, C_, H_, W_ = x.shape
16 FN, C, FH, FW = W.shape
17 ###
18 out_h = 1 + int((H_ + 2*pad - FH)/stride)
19 out_w = 1 + int((W_ + 2*pad - FW)/stride)
20 ###
21 col = im2col(x, FH, FW, stride, pad) # im2col 함수 활용
22 #####
23 ##### your code #####
24 #####
25 col_W = None # W의 형태를 변형하시오.
26 out = None # Affine 연산 후 바이어스를 더합시다.
27 #####
28 out_final = out.reshape(N_, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

0	1	2
3	4	5
6	7	8

3x3 입력 데이터 : x

*

1	1	1
1	1	1

2	2	1
2	2	1

2x2 필터 2개 : W bias 2개 : b

완전 단순 Conv 구현해보기

- 2_day/prac1_toyConv.py



모두의연구소

```
6  """ 초기값 설정
7  x = list(range(1*3*3))
8  x = np.array(x).reshape(1, 1, 3, 3) # (데이터 수, 채널 수, 높이, 너비)
9  W = np.ones((2, 1, 2, 2)) # (필터 수, 필터 채널, 필터높이, 필터 너비)
10 W[1] = W[1]*2
11 b = np.ones((2,)) # (바이어스 수)
12 pad = 0
13 stride = 1
14 """
15 N_, C_, H_, W_ = x.shape
16 FN, C, FH, FW = W.shape
17 """
18 out_h = 1 + int((H_ + 2*pad - FH)/stride)
19 out_w = 1 + int((W_ + 2*pad - FW)/stride)
20 """
21 col = im2col(x, FH, FW, stride, pad) # im2col 함수 활용
22 #####
23 ##### your code #####
24 #####
25 col_W = None # W의 형태를 변형하시오.
26 out = None # Affine 연산 후 바이어스를 더합니다.
27 #####
28 out_final = out.reshape(N_, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

완전 단순 Conv 구현해보기

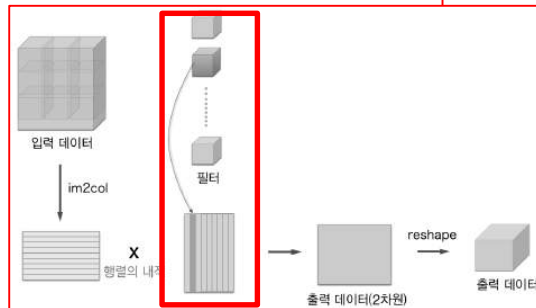


모두의연구소

- 2_day/prac1_toyConv.py

```
6  ### 초기값 설정
7  x = list(range(1*3*3))
8  x = np.array(x).reshape(1, 1, 3, 3) # (데이터 수, 채널 수, 높이, 너비)
9  W = np.ones((3, 4, 5, 2)) # (채널, 필터높이, 필터 너비)
10 W[1] = W[1]*2
11 b = np.ones((3, 4, 5, 2))
12 pad = 0
13 stride = 1
14 ###
15 N_, C_, H_, W_ = x.shape
16 FN, C, FH, FW = W.shape
17 ###
18 out_h = 1 + (H_ - FH) // stride
19 out_w = 1 + (W_ - FW) // stride
20 ###
21 col = im2col(x, FH, FW, stride, pad) # im2col 함수 활용
22 #####
23 ##### your code #####
24 #####
25 col_W = None # W의 형태를 변형하시오.
26 out = None # Affine 연산 후 바이어스를 더합니다.
27 #####
28 out_final = out.reshape(N_, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

힌트



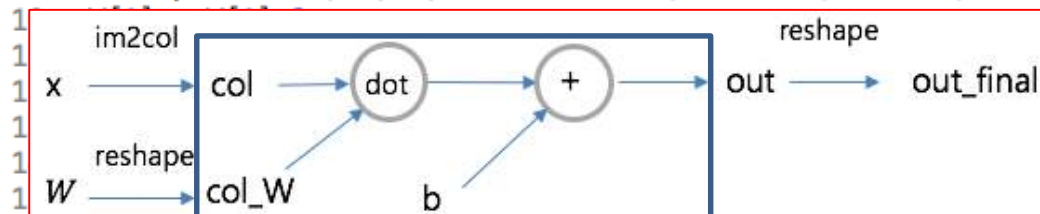
완전 단순 Conv 구현해보기

- 2_day/prac1_toyConv.py

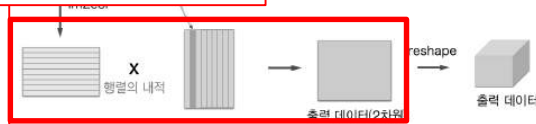


모두의연구소

```
6  ### 초기값 설정
7  x = list(range(1*3*3))
8  x = np.array(x).reshape(1, 1, 3, 3) # (데이터 수, 채널 수, 높이, 너비)
9  W = np.ones((2, 1, 2, 2)) # (필터 수, 필터 채널, 필터높이, 필터 너비)
```

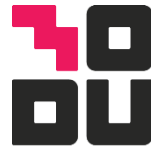


```
16 FN, C, FH, FW = W.shape
17 ###
18 out_h = 1 + int((H_ + 2*pad - FH)/stride)
19 out_w = 1 + int((W_ + 2*pad - FW)/stride)
```



```
20 ###
21 col = im2col(x, FH, FW, stride, pad) # im2col 함수 활용
22 #####
23 ##### your code #####
24 #####
25 col_W = None # W의 형태를 변형하시오.
26 out = None # Affine 연산 후 바이어스를 더합니다.
27 #####
28 out_final = out.reshape(N_, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

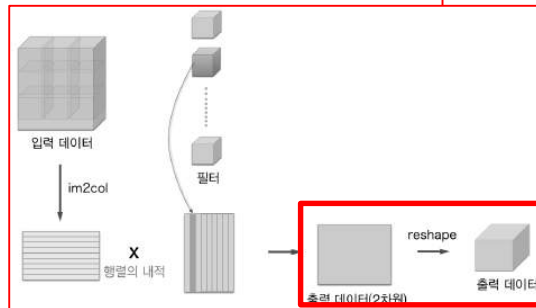
완전 단순 Conv 구현해보기



모두의연구소

- 2_day/prac1_toyConv.py

```
6  ### 초기값 설정
7  x = list(range(1*3*3))
8  x = np.array(x).reshape(1, 1, 3, 3) # (데이터 수, 채널 수, 높이, 너비)
9  W = np.ones((2, 1, 2, 2)) # (필터 수, 필터 채널, 필터높이, 필터 너비)
10 W[1] = W[1]*2
11 b = np.ones((2,)) # (바이어스 수)
12 pad = 0
13 stride = 1
14 ###
15 N_, C_, H_, W_ = x.shape
16 FN, C, FH, FW = W.shape
17 ###
18 out_h = 1 + int((H_ + 2*pad - FH)/stride)
19 out_w = 1 + int((W_ + 2*pad - FW)/stride)
20 ###
21 col = im2col(x, FH, FW, stride, pad) # im2col 함수 활용
22 #####
23 ##### your code #####
24 #####
25 col_W = None # W의 형태를 변형하십시오.
26 out = None # Affine 연산 후 바이어스를 더합니다.
27 #####
28 out_final = out.reshape(N_, out_h, out_w, -1).transpose(0, 3, 1, 2)
```



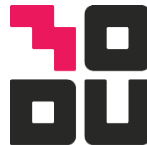
		17	25
		41	49
9	13		
21	25		

완전 단순 Conv 구현해보기

- 2_day/prac1_toyConv.py
 - 백 프로파게이션은 어떻게해요?



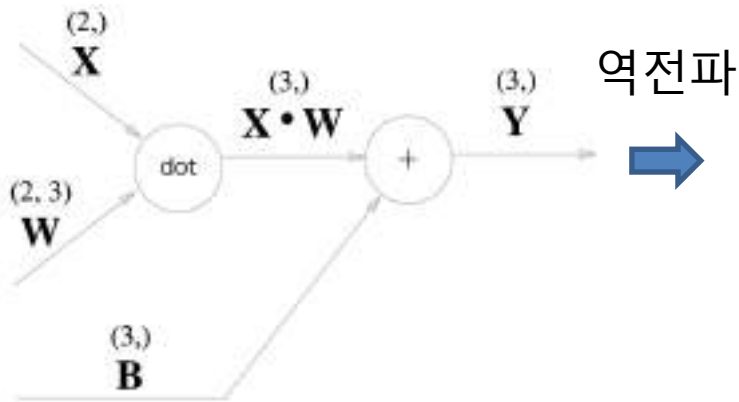
완전 단순 Conv 구현해보기



모두의연구소

- 2_day/prac1_toyConv.py

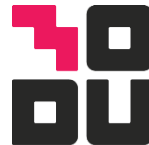
- 백 프로파게이션은 어떻게해요?
 - 순방향을 구현해 보니 Affine 계층 했던 것 하면 될 것 같다는 생각이 뇌리에 스칩니다



$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^T$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

완전 단순 Conv 구현해보기

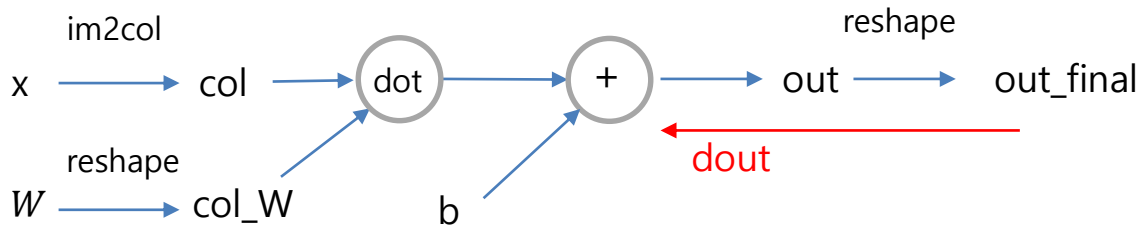


모두의연구소

- 2_day/prac1_toyConv.py

- 백 프로파게이션은 어떻게해요?

```
34 ### backward
35 dout = np.ones_like(out_final)
36 dout = dout.transpose(0,2,3,1).reshape(-1, FN)
37
38 #####
39 ##### your code #####
40 #####
41 db = None
42 dcol_W = None
43 dW = dcol_W.transpose(1, 0).reshape(FN, C, FH, FW)
44 dcol = None
45 #####
46 dx = col2im(dcol, x.shape, FH, FW, stride, pad)
```



완전 단순 Conv 구현해보기

- 2_day/prac1_toyConv.py



모두의연구소

- 백 프로파게이션은 어떻게해요?

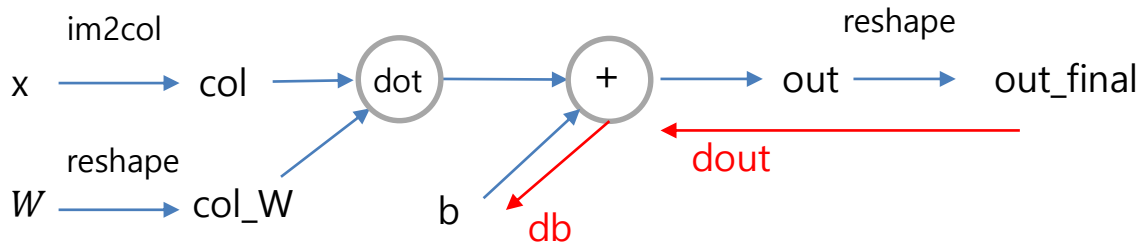
```
34 ### backward
35 dout = np.ones_like(out_final)
36 dout = dout.transpose(0,2,3,1).reshape(-1)
37
38 #####
39 ##### your code #####
40 #####
41 db = None
42 dcol_W = None
43 dW = dcol_W.transpose(1, 0).reshape(FN)
44 dcol = None
45 #####
46 dx = col2im(dcol, x.shape, FH, FW, stride, pad)
```

Affine / Softmax 계층 구현하기

- 배치용 Affine 계층
- 데이터가 2개 일 경우(N=2)의 면향은 계산된 각각의 결과에 더해집니다

여전파의 경우

```
In [23]: dY = np.array([[1,2,3],[4,5,6]])
In [24]: dY
Out[24]:
array([[1, 2, 3],
       [4, 5, 6]])
In [25]: dE = np.sum(dY, axis=0)
In [26]: dE
Out[26]: array([5, 7, 9])
```



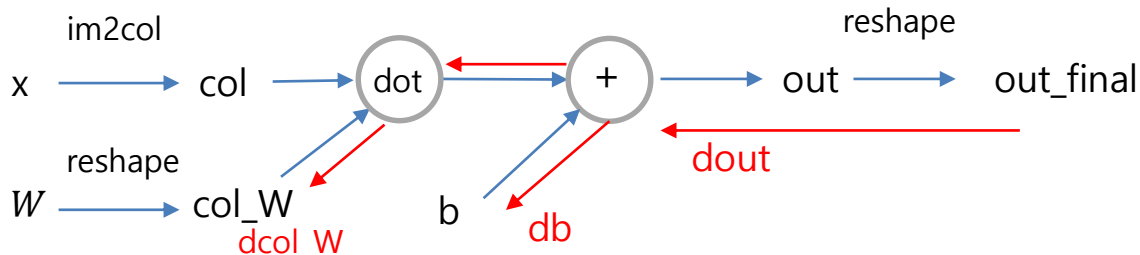
완전 단순 Conv 구현해보기

- 3_day/prac1_toyConv.py
- 백 프로파게이션은 어떻게해요?

```
34 ### backward
35 dout = np.ones_like(out_final)
36 dout = dout.transpose(0,2,3,1).reshape(-1, FN)
37
38 #####
39 ##### your code #####
40 #####
41 db = None
42 dcol_W = None
43 dW = dcol_W.transpose(1, 0).reshape(FN, C, FH, FW)
44 dcol = None
45 #####
46 dx = col2im(dcol, x.shape, FH, FW, stride, pad)
```

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^T$$

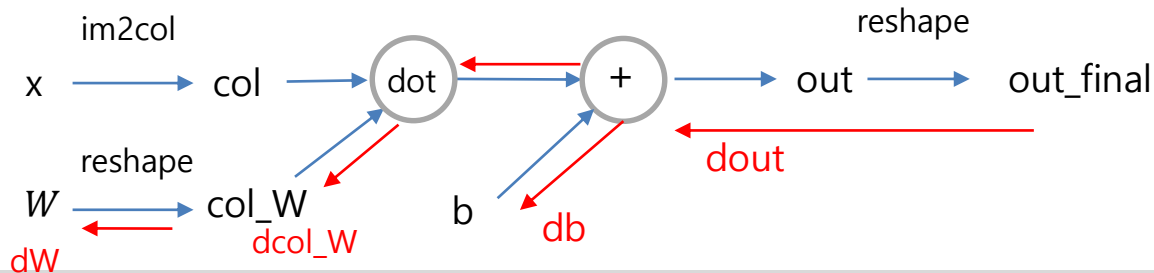
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$



완전 단순 Conv 구현해보기

- 3_day/prac1_toyConv.py
- 백 프로파게이션은 어떻게해요?

```
34 ### backward
35 dout = np.ones_like(out_final)
36 dout = dout.transpose(0,2,3,1).reshape(-1, FN)
37
38 #####
39 ##### your code #####
40 #####
41 db = None
42 dcol_W = None
43 dW = dcol_W.transpose(1, 0).reshape(FN, C, FH, FW)
44 dcol = None
45 #####
46 dx = col2im(dcol, x.shape, FH, FW, stride, pad)
```



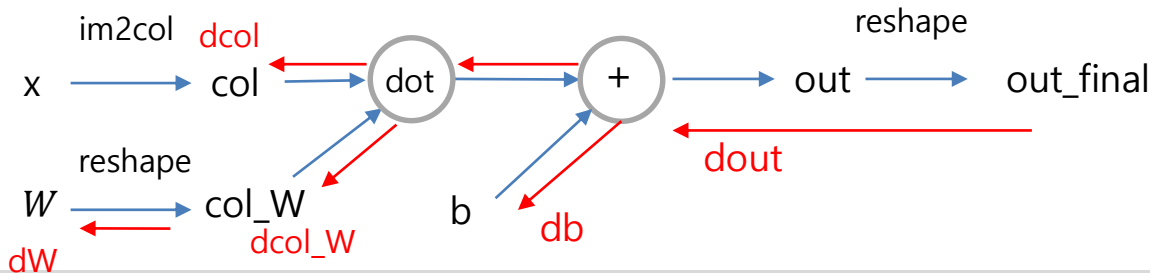
완전 단순 Conv 구현해보기

- 3_day/prac1_toyConv.py
- 백 프로파게이션은 어떻게해요?

```
34 ## backward
35 dout = np.ones_like(out_final)
36 dout = dout.transpose(0,2,3,1).reshape(-1, FN)
37
38 #####
39 ##### your code #####
40 #####
41 db = None
42 dcol_W = None
43 dW = dcol_W.transpose(1, 0).reshape(FN, C, FH, FW)
44 dcol = None
45 #####
46 dx = col2im(dcol, x.shape, FH, FW, stride, pad)
```

$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^T$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$



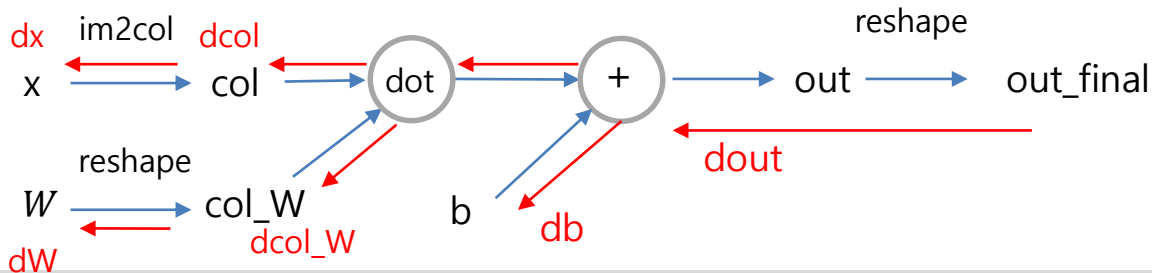
완전 단순 Conv 구현해보기



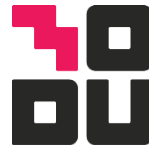
모두의연구소

- 3_day/prac1_toyConv.py
- 백 프로파게이션은 어떻게해요?

```
34 ### backward
35 dout = np.ones_like(out_final)
36 dout = dout.transpose(0,2,3,1).reshape(-1, FN)
37
38 #####
39 ##### your code #####
40 #####
41 db = None
42 dcol_W = None
43 dW = dcol_W.transpose(1, 0).reshape(FN, C, FH, FW)
44 dcol = None
45 #####
46 dx = col2im(dcol, x.shape, FH, FW, stride, pad)
```



완전 단순 Conv 구현해보기



모두의연구소

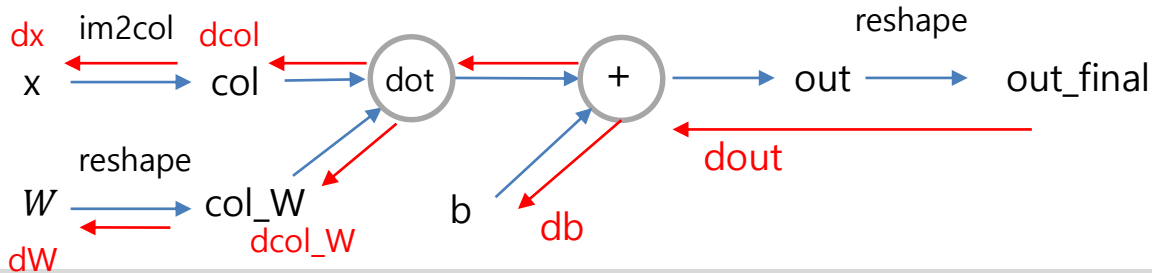
- 3_day/prac1_toyConv.py

- 백 프로파게이션은 어떻게해요?

```
34 ### backward
35 dout = np.ones_like(out_final)
36 dout = dout.transpose(0,2,3,1).reshape(-1,
37
38 #####
39 ##### your code #####
40 #####
41 db = None
42 dcol_W = None
43 dW = dcol_W.transpose(1, 0).reshape(FN, C, fn, fw)
44 dcol = None
45 #####
46 dx = col2im(dcol, x.shape, FH, FW, stride, pad)
```

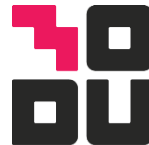
```
dW :
[[[ 8. 12.]
 [ 20. 24.]]]

[[[ 8. 12.]
 [ 20. 24.]]]]
db :
[ 4.  4.]
dx :
[[[ 3.  6.  3.]
 [ 6. 12.  6.]
 [ 3.  6.  3.]]]]
```



모듈화된 형태로 살펴봅시다

Convolution 레이어 구현하기



모두의연구소

– common/layers.py

```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
213
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
```


Convolution 레이어 구현하기



모두의연구소

– common/layers.py

```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
213
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
```

- FN : 필터 수
- C : 필터 채널 수
- FH : 필터 높이
- FW : 필터 너비

Convolution 레이어 구현하기



모두의연구소

```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
213
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
```

기억하시나요?

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

Convolution 레이어 구현하기



모두의연구소

```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
213
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
```

im2col

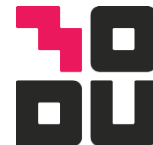
Convolution 레이어 구현하기



모두의연구소

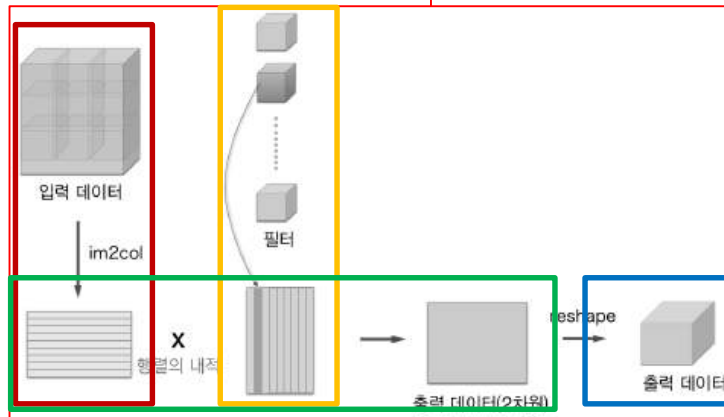
```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
213
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
```

Convolution 레이어 구현하기

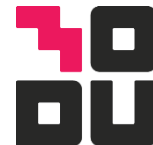


모두의연구소

```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
213
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
```

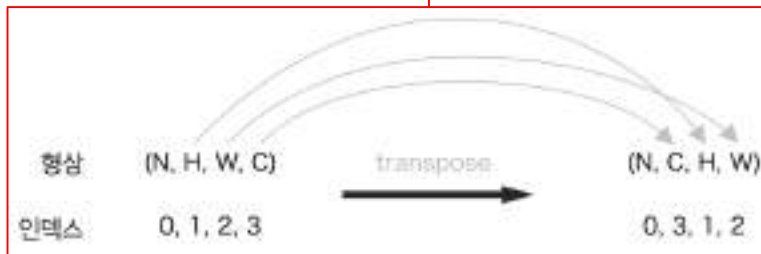


Convolution 레이어 구현하기



모두의연구소

```
198 class Convolution:
199     def __init__(self, W, b, stride=1, pad=0):
200         self.W = W
201         self.b = b
202         self.stride = stride
203         self.pad = pad
204
205         # 중간 데이터 (backward 시 사용)
206         self.x = None
207         self.col = None
208         self.col_W = None
209
210         # 가중치와 편향 매개변수의 기울기
211         self.dW = None
212         self.db = None
213
214     def forward(self, x):
215         FN, C, FH, FW = self.W.shape
216         N, C, H, W = x.shape
217         out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
218         out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
219
220         col = im2col(x, FH, FW, self.stride, self.pad)
221         col_W = self.W.reshape(FN, -1).T
222
223         out = np.dot(col, col_W) + self.b
224         out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
225
226         self.x = x
227         self.col = col
228         self.col_W = col_W
229
230         return out
```



```
In [40]: x.shape
Out[40]: (3, 4, 5, 2)

In [41]: x.transpose(0,1,3,2).shape
Out[41]: (3, 4, 2, 5)

In [42]: x.transpose(1,2,3,0).shape
Out[42]: (4, 5, 2, 3)

In [43]: x.transpose(3,2,1,0).shape
Out[43]: (2, 5, 4, 3)
```


Convolution 레이어 구현하기

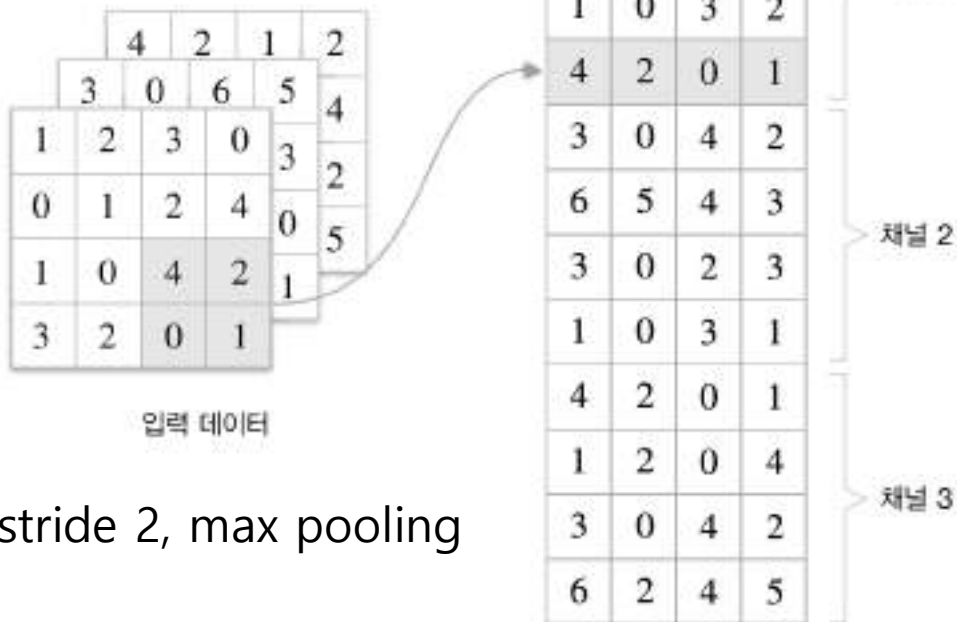
– common/layers.py

```
232     def backward(self, dout):
233         FN, C, FH, FW = self.W.shape
234         dout = dout.transpose(0,2,3,1).reshape(-1, FN)
235
236         self.db = np.sum(dout, axis=0)
237         self.dW = np.dot(self.col.T, dout)
238         self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
239
240         dcol = np.dot(dout, self.col_W.T)
241         dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)
242
243         return dx
```

col2im을 제외하면 Affine 계층과 같습니다

Pooling 레이어 구현하기

- im2col을 사용한 후 변형



2x2, stride 2, max pooling

Pooling 레이어 구현하기

- im2col을 사용합니다

- 1) 입력 데이터 전개
- 2) 형렬 최대값을 구함
- 3) 적절한 모양으로 성형



Pooling 레이어 구현하기

- 디버그 하며 살펴보도록 합시다
- 3_day/prac2_toyPooling.py

```
1 import sys, os
2 sys.path.append(os.pardir)
3 import numpy as np
4 from common.util import im2col, col2im
5
6 ### 초기값 설정
7 x = list(range(2*3*3))
8 x = np.array(x).reshape(1, 2, 3, 3) # (데이터 수, 채널 수, 높이, 너비)
9 pool_h = 2
10 pool_w = 2
11 stride = 1
12 pad = 0
13 ###
14 print("x :\n", x)
15 N_, C_, H_, W_ = x.shape
16
17 out_h = int(1 + (H_ - pool_h) / stride)
18 out_w = int(1 + (W_ - pool_w) / stride)
19
20 col_old = im2col(x, pool_h, pool_w, stride, pad) # 출력해보세오
21 col = col_old.reshape(-1, pool_h*pool_w) # 위의 결과와 비교해보세오
22
23 arg_max = np.argmax(col, axis=1) # max의 위치가여 (np.argmax)
24 out = np.max(col, axis=1) # 출력값
25 out_final = out.reshape(N_, out_h, out_w, C_).transpose(0, 3, 1, 2)
26 print("out_final :\n", out_final)
27
28 #####
29 ### backward
30 dout = np.ones_like(out_final)
31 dout = dout.transpose(0, 2, 3, 1)
32
33 pool_size = pool_h * pool_w
34
35 # 결과를 저장하기 위한 공간 (pooling size만큼 크기를 넓혀 줍니다.)
36 dmax = np.zeros((dout.size, pool_size))
37
38 # 기억해 두었던 위치에 값을 채워줍니다.
39 dmax[np.arange(arg_max.size), arg_max.flatten()] = dout.flatten()
40
41 # 원래의 모양대로 복원합니다.
42 dmax = dmax.reshape(dout.shape + (pool_size,))
43
44 dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
45 dx = col2im(dcol, x.shape, pool_h, pool_w, stride, pad)
46 print("dx :\n", dx)
```

풀링레이어 클래스

- /common/layers.py

```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.s = None
        self.arg_max = None

    def forward(self, s):
        N, C, H, W = s.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        out = np.zeros((N, out_h, out_w, C))
        col = col2im(s, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)

        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        self.s = s
        self.arg_max = arg_max

        return out

    def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
        dmax = np.zeros((dout.size, pool_size))
        dmax[np.arange(self.arg_max.size, self.arg_max.flatten())] = dout.flatten()
        dmax = dmax.reshape(dout.shape + (pool_size,))

        dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
        dx = col2im(dcol, self.s.shape, self.pool_h, self.pool_w, self.stride, self.pad)

        return dx
```

Pooling 레이어 구현하기



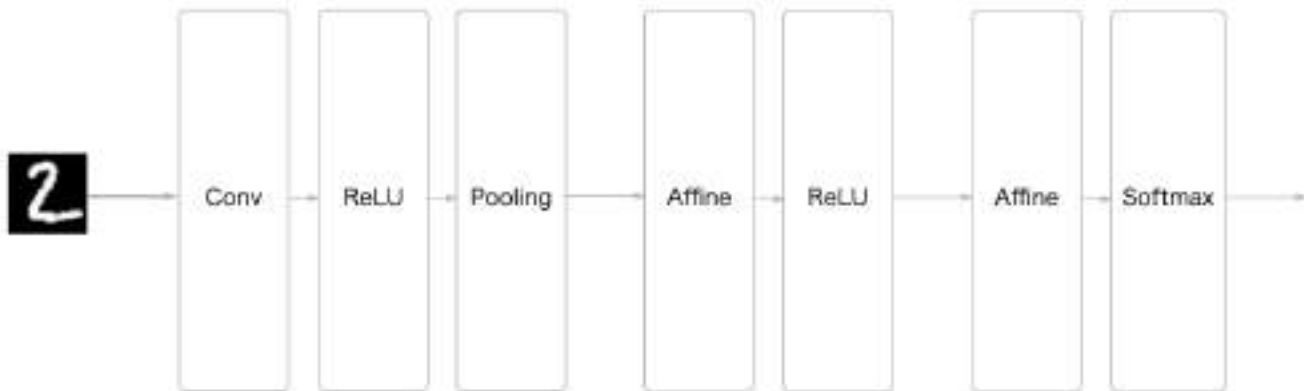
모두의연구소

- 풀링 계층 구현하기 : /common/layers.py

```
246 class Pooling:
247     def __init__(self, pool_h, pool_w, stride=1, pad=0):
248         self.pool_h = pool_h
249         self.pool_w = pool_w
250         self.stride = stride
251         self.pad = pad
252
253         self.x = None
254         self.arg_max = None
255
256     def forward(self, x):
257         N, C, H, W = x.shape
258         out_h = int(1 + (H - self.pool_h) / self.stride)
259         out_w = int(1 + (W - self.pool_w) / self.stride)
260
261         col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
262         col = col.reshape(-1, self.pool_h*self.pool_w)
263
264         arg_max = np.argmax(col, axis=1)
265         out = np.max(col, axis=1)
266         out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)
267
268         self.x = x
269         self.arg_max = arg_max
270
271         return out
272
273     def backward(self, dout):
274         dout = dout.transpose(0, 2, 3, 1)
275
276         pool_size = self.pool_h * self.pool_w
277         dmax = np.zeros((dout.size, pool_size))
278         dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flatten()
279         dmax = dmax.reshape(dout.shape + (pool_size,))
280
281         dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
282         dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)
283
284         return dx
```

CNN 돌려보기

- 아래의 그림과 같은 CNN을 돌려봅니다



- 지금 `3_day/train_convnet.py` 를 실행하세요

Conv. → ReLU → Pooling → Affine → ReLU → Affine → Softmax

- 필터 수:30
- 필터 크기:5x5
- 패딩:0
- 스트라이드:1
- 필터크기:2x2
- 스트라이드:2
- 뉴런 수:100
- 뉴런 수:10

CNN 돌려보기

- 3_day/train_convnet.py

```
1 # coding: utf-8
2 import sys, os
3 sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from dataset.mnist import load_mnist
7 from simple_convnet import SimpleConvNet
8 from common.trainer import Trainer
9
10 # 데이터 읽기
11 (x_train, t_train), (x_test, t_test) = load_mnist(flatten=False)
12
13 # 시간이 오래 걸릴 경우 데이터를 줄인다.
14 x_train, t_train = x_train[:5000], t_train[:5000]
15 x_test, t_test = x_test[:1000], t_test[:1000]
16
17 max_epochs = 20
18
19 network = SimpleConvNet(input_dim=(1,28,28),
20                          conv_param = {'filter_num': 30, 'filter_size': 5,
21                                         'pad': 0, 'stride': 1},
22                          hidden_size=100, output_size=10, weight_init_std=0.01)
23
24 trainer = Trainer(network, x_train, t_train, x_test, t_test,
25                  epochs=max_epochs, mini_batch_size=100,
26                  optimizer='Adam', optimizer_param={'lr': 0.001},
27                  evaluate_sample_num_per_epoch=1000)
28 trainer.train()
29
30 # 매개변수 보존
31 network.save_params("params.pkl")
32 print("Saved Network Parameters!")
33
34 # 그래프 그리기
35 markers = {'train': 'o', 'test': 's'}
36 x = np.arange(max_epochs)
37 plt.plot(x, trainer.train_acc_list, marker='o', label='train', markevery=2)
38 plt.plot(x, trainer.test_acc_list, marker='s', label='test', markevery=2)
39 plt.xlabel("epochs")
40 plt.ylabel("accuracy")
41 plt.ylim(0, 1.0)
42 plt.legend(loc='lower right')
43 plt.show()
```

오래걸리니 조금만 돌리기

Trainer 설정

네트워크 구성 설정

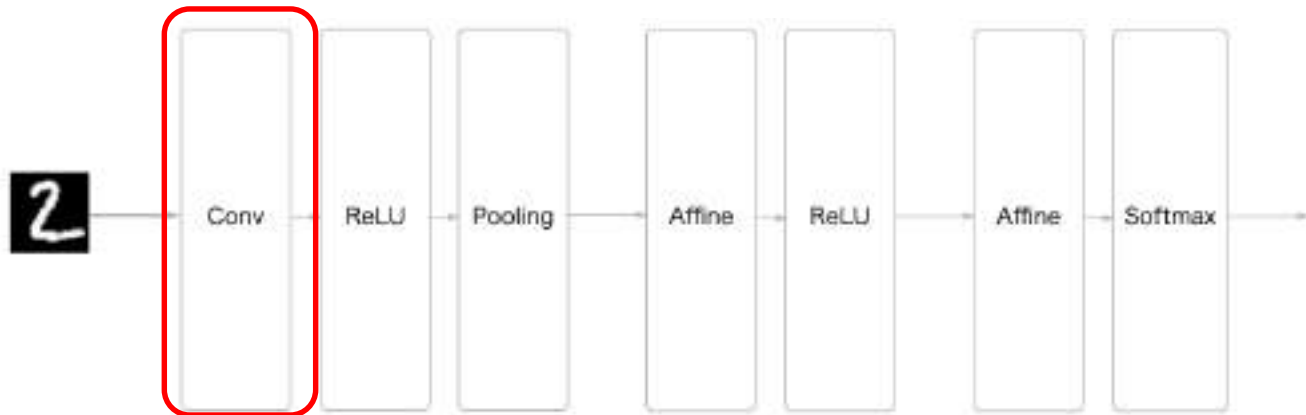
- 3_day/simple_convnet.py

파라미터 저장

출력

CNN 시각화 하기

- 1번째 층의 가중치 시각화



- 지금 `3_day/train_convnet.py` 를 실행하세요

Conv. → ReLU → Pooling → Affine → ReLU → Affine → Softmax

- 필터 수:30
- 필터 크기:5x5
- 패딩:0
- 스트라이드:1

- 필터크기:2x2
- 스트라이드:2
- 뉴런 수:100

- 뉴런 수:10

형상 : (30, 1, 5, 5) → 출력해서 볼 수 있음

CNN 시각화 하기

- 1번째 층의 가중치 시각화
 - 3_day/visualize_filter.py

```
1 # coding: utf-8
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from simple_convnet import SimpleConvNet
5
6 def filter_show(filters, nx=8, margin=3, scale=10):
7     """
8     c.f. https://gist.github.com/aidiary/07d530d5e08011832b12#file-draw\_weight-py
9     """
10    FN, C, FH, FW = filters.shape
11    ny = int(np.ceil(FN / nx))
12
13    fig = plt.figure()
14    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)
15
16    for i in range(FN):
17        ax = fig.add_subplot(ny, nx, i+1, xticks=[], yticks=[])
18        ax.imshow(filters[i, 0], cmap=plt.cm.gray_r, interpolation='nearest')
19    plt.show()
20
21
22 network = SimpleConvNet()
23 # 무작위(랜덤) 초기화 후의 가중치
24 filter_show(network.params['W1'])
25
26 # 학습된 가중치
27 network.load_params("params.pkl")
28 filter_show(network.params['W1'])
```

네트워크 초기화

파라미터 로드

CNN 시각화 하기

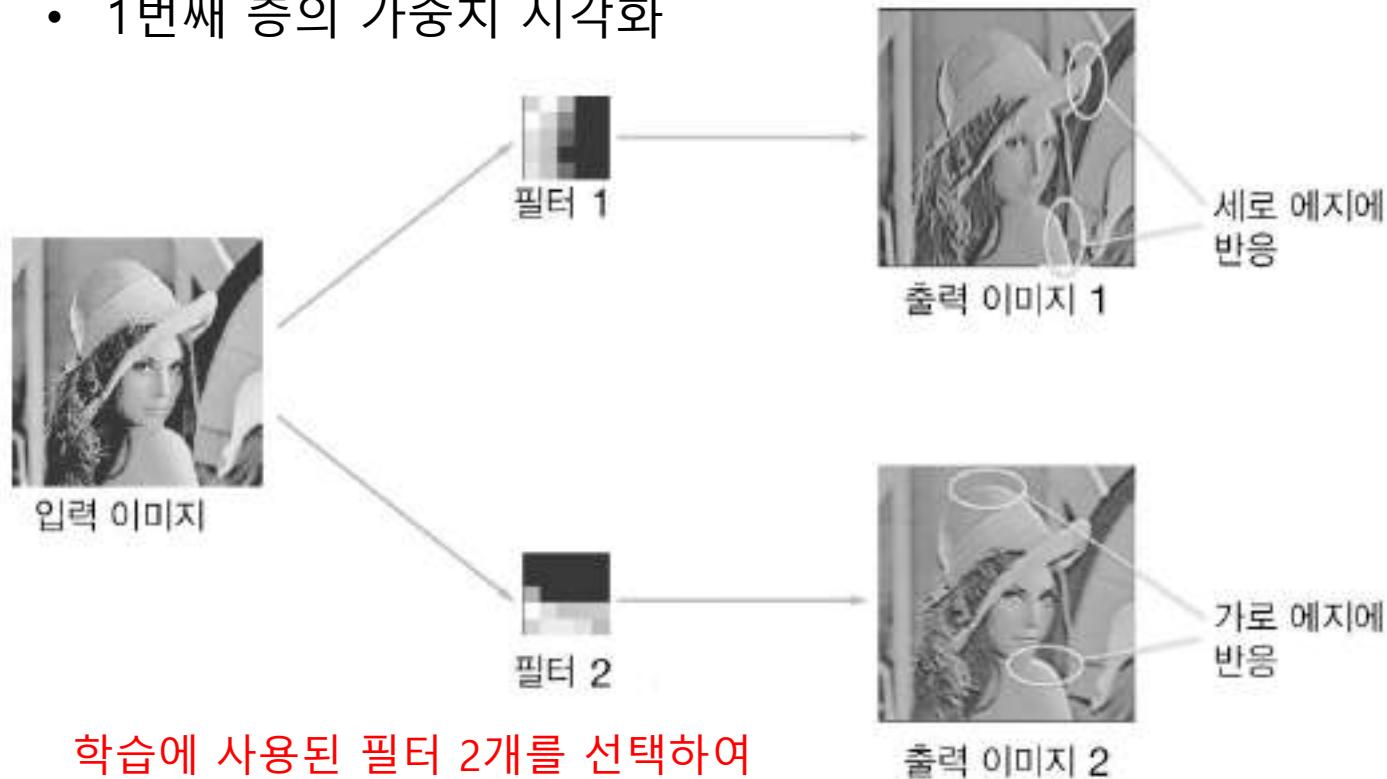
- 1번째 층의 가중치 시각화
 - 학습 전과 후의 1번째 층 가중치 비교 : 가장 작은 값 0으로(검은색), 가장 큰 값(255)은 흰색으로 표시함



- 학습 후 필터는 에지(색상이 바뀌는 경계선)와 블랍(국소적으로 덩어려진 영역 등을 보고 있음)

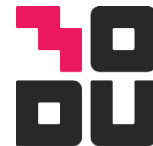
CNN 시각화 하기

- 1번째 층의 가중치 시각화



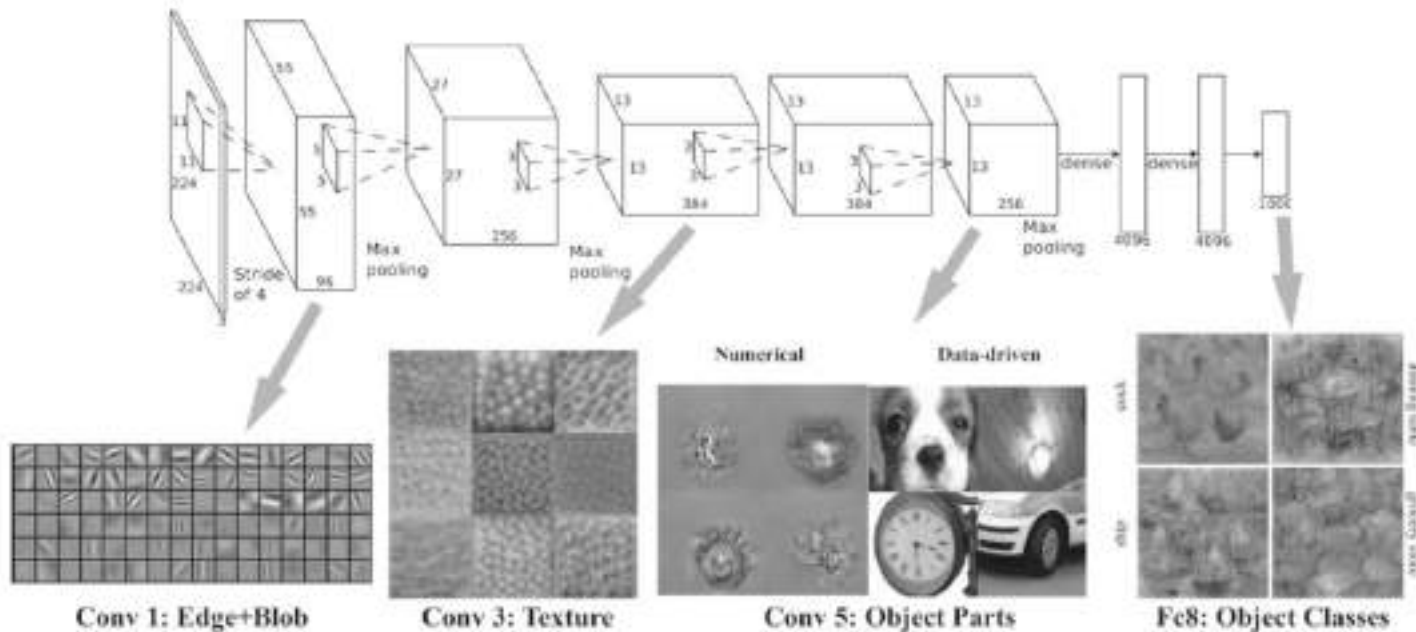
학습에 사용된 필터 2개를 선택하여
적용해봄 : 1층은 원시적 정보

CNN 시각화 하기



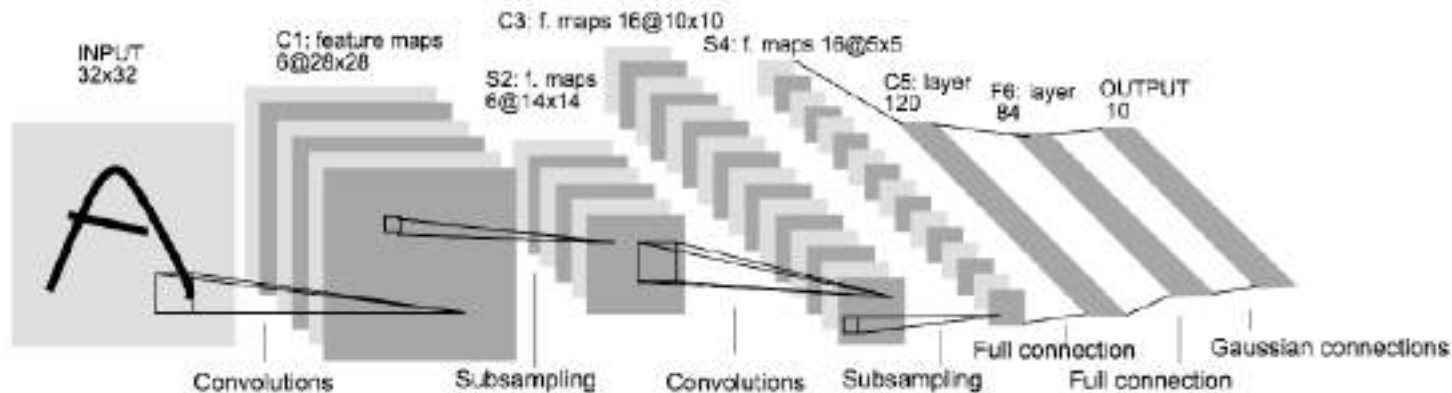
모두의연구소

- 층 깊이에 따른 추출정보 변화
 - 1번째 층은 에지와 블랍, 3번째 층은 텍스처, 5번째 층은 사물의 일부, 마지막 완전연결 계층은 사물의 클래스에 뉴런이 반응한다
 - 층이 깊어질 수록 뉴런이 반응하는 대상이 단순한 모양에서 고급 정보로 변해감



대표적 CNN

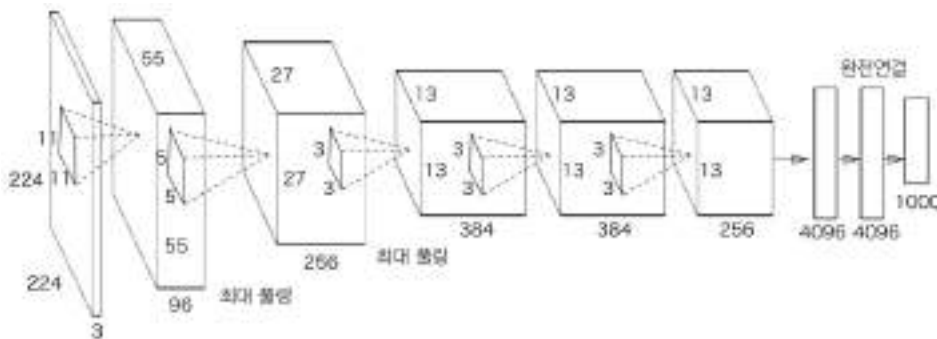
- LeNet (1998년)



- 현재의 CNN과의 가장 큰 차이 : 시그모이드
 - 현재는 주로 ReLU

대표적 CNN

- AlexNet (2012년)



- LeNet과의 차이
 - 활성화 함수로 ReLU 적용
 - LRN (Local Response Normalization)이라는 국소적 정규화를 실시하는 계층 적용 (현재 거의 안씀)
 - 드롭아웃 적용
- GPU를 활용하였음. 빅데이터와 GPU가 딥러닝 발전의 큰 원동력임

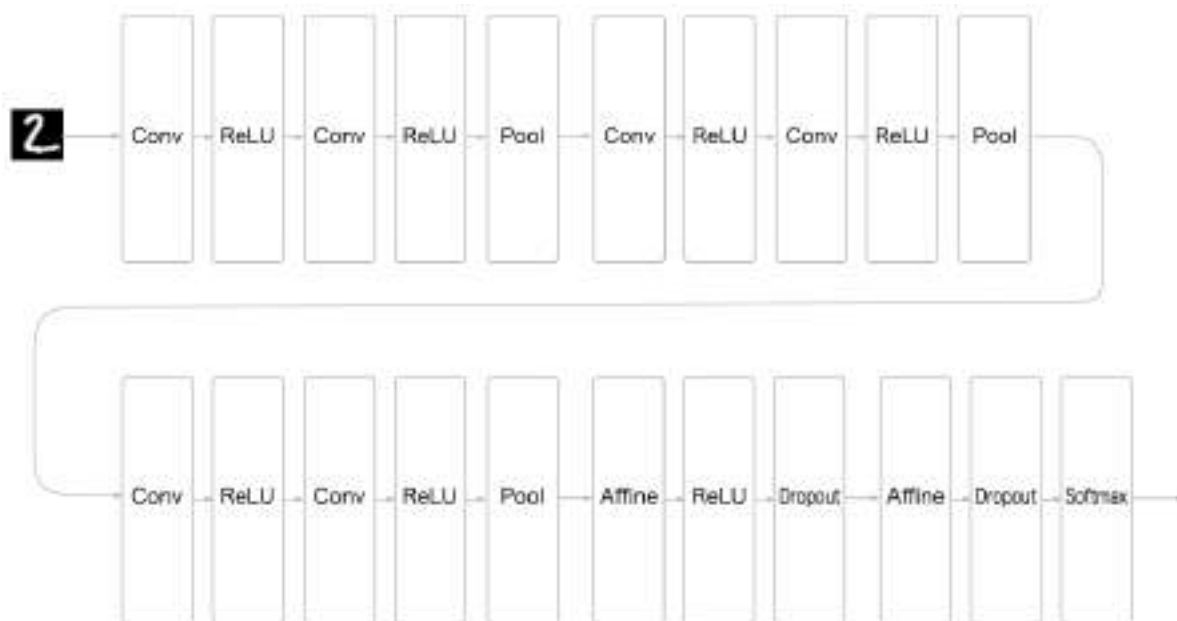
정리

- CNN은 지금까지 완전연결 계층(fully connected 네트워크에 합성곱 계층과 풀링 계층을 새로 추가
- 합성곱 계층과 풀링계층은 im2col(이미지를 행렬로전개하는 함수)를 이용하면 간단하고 효율적으로 구현할 수 있음
- CNN을 시각화해보면 계층이 깊어질 수록 고급 정보가 추출되는 모습을 확인할 수 있음
- 대표적인 CNN에는 LeNet과 AlexNet이 있음
- 딥러닝의 발전에는 빅데이터와 GPU가 크게 기여했음

딥러닝

더 깊게

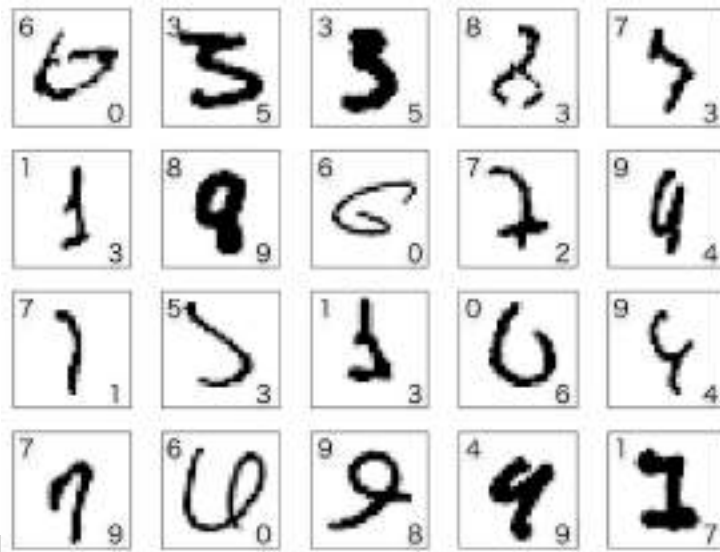
- 더 깊은 신경망으로
 - 3_day/deep_convnet.py



더 깊게

- 더 깊은 신경망으로
 - 학습 완료된 네트워크 파라미터를 이용해서 돌려봅시다
 - 3_day/misclassified_mnist.py
 - 정확도 : 99.38% 즉 오인식율 0.62%

- 왼쪽위 : 정답
- 오른쪽 아래 : 추론



더 깊게

- 정확도를 높이려면
 - What is the class of this image? (웹사이트)
 - http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

MNIST
who is the best in MNIST ?

1 1 5 4 3
7 5 3 5 3
5 5 9 0 6
3 6 2 0 0

MNIST 10,000 images
Unit: error %
Classify handwritten digits. Some additional results are available on the original dataset page.

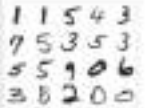
Result	Method	Venue	Details
0.21%	Regularization of Neural Networks using DropConnect	ICML 2013	
0.23%	Multi-column Deep Neural Networks for Image Classification	CVPR 2012	
0.23%	APWC: Augmented Pattern Classification with Neural Networks	arXiv 2015	
0.24%	Batch-normalized Maxout Network in Network	arXiv 2015	Details
0.28%	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree	AISTATS 2016	paper
0.31%	Recurrent Convolutional Neural Network for Object Recognition	CVPR 2015	
0.31%	On the Importance of Normalization Layers in Deep Learning with Piecewise Linear Activation Units	arXiv 2015	
0.32%	Fractional Max-Pooling	arXiv 2015	Details

- 목록들이 사용하는 네트워크가 그다지 깊지않음(합성곱 계층 2개에 완전연결 계층 2개 정도인 신경망)
- 손글씨 숫자라는 문제가 비교적 단순해서 신경망의 표현력을 극한까지 높일 필요는 없기 때문이라고 생각합니다. 그래서 층을 깊게 해도 혜택이 적다고 할 수 있죠. (저자)

더 깊게

- 정확도를 높이려면
 - What is the class of this image? (웹사이트)
 - http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

MNIST
who is the best in MNIST ?



MNIST 10,000 images
Unit: error %
Classify handwritten digits. Some additional results are available on the original dataset page.

Result	Method	Venue	Details
0.21%	Regularization of Neural Networks using DropConnect	ICML 2013	
0.23%	Multi-column Deep Neural Networks for Image Classification	CVPR 2012	
0.23%	APWC: Augmented Pattern Classification with Neural Networks	arXiv 2015	
0.24%	Batch-normalized Maxout Network in Network	arXiv 2015	Details
0.28%	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree	AISTATS 2016	paper
0.31%	Recurrent Convolutional Neural Network for Object Recognition	CVPR 2015	
0.31%	On the Importance of Normalization Layers in Deep Learning with Piecewise Linear Activation Units	arXiv 2015	
0.32%	Fractional Max-Pooling	arXiv 2015	Details

- 목록들이 사용하는 네트워크가 그다지 깊지않음(합성곱 계층 2개에 완전연결 계층 2개 정도인 신경망)
- 앙상블, 학습률 감소, 데이터 확장 등이 정확도 향상 등이 기여

더 깊게

- 정확도를 높이려면
 - 데이터 확장 (data augmentation)
 - 입력 이미지를 회전하거나 세로로 이동하는 등 미세한 변화를 주어 이미지의 개수를 늘림
 - 데이터가 몇개 없을때 효과적



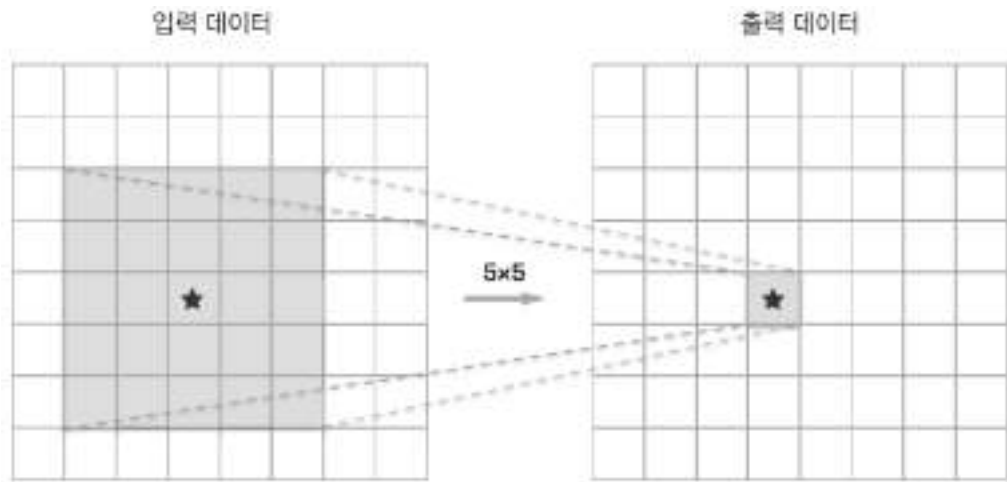
- 이미지 crop이나 flip도 있음
- 밝기 등의 외형변화나 확대/축소 등의 스케일 변화도 있음

더 깊게

- 깊게하는 이유
 - ILSVRC(이미지 인식대회 이름)에서 층을 깊게할 수록 정확도가 높아지는 경향이 나타남
- 깊게하는 이점
 - 신경망의 매개변수가 줄어 듦. 깊게한 경우 깊지 않은 경우보다 적은 매개변수로 같은 수준의 표현력을 달성할 수있음

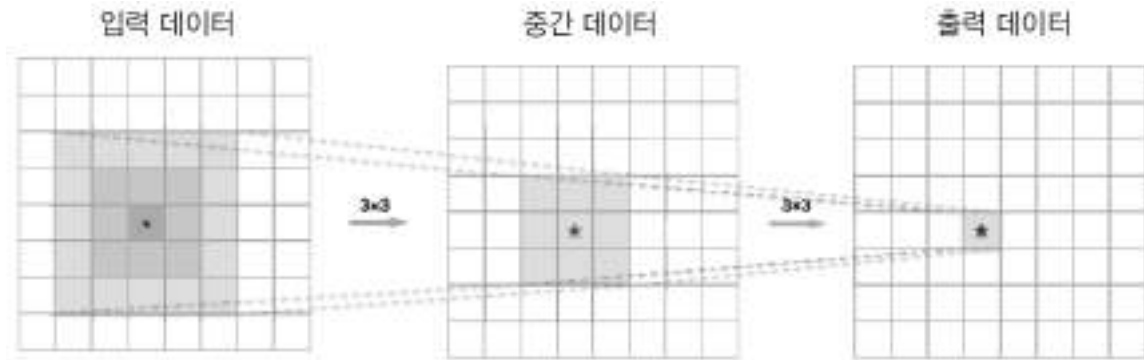
더 깊게

- 깊게하는 이점
 - 5x5 한번과 3x3 두번의 비교



더 깊게

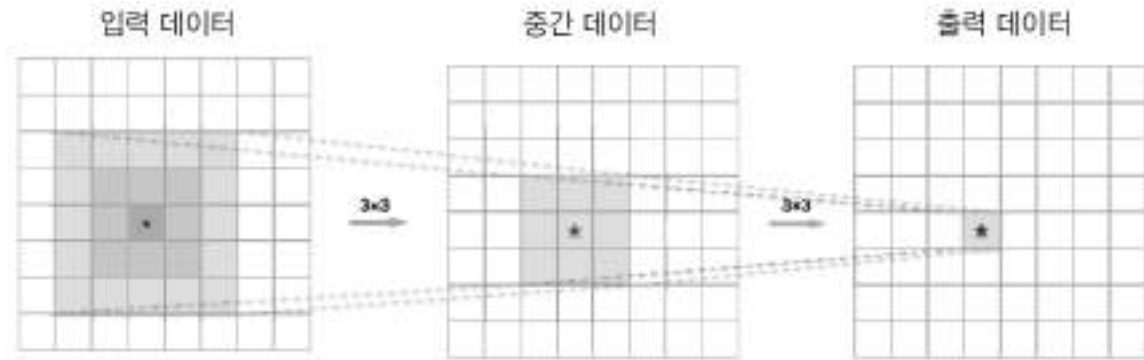
- 깊게하는 이점
 - 5x5 한번과 3x3 두번의 비교



- 5x5와 같은 크기의 영역을 처리 (receptive field)
- 층이 깊어지기에 ReLU와 같은 비선형성 추가로 표현력이 개선. 비선형 함수가 겹쳐지면 더 복잡한것도 표현 가능

더 깊게

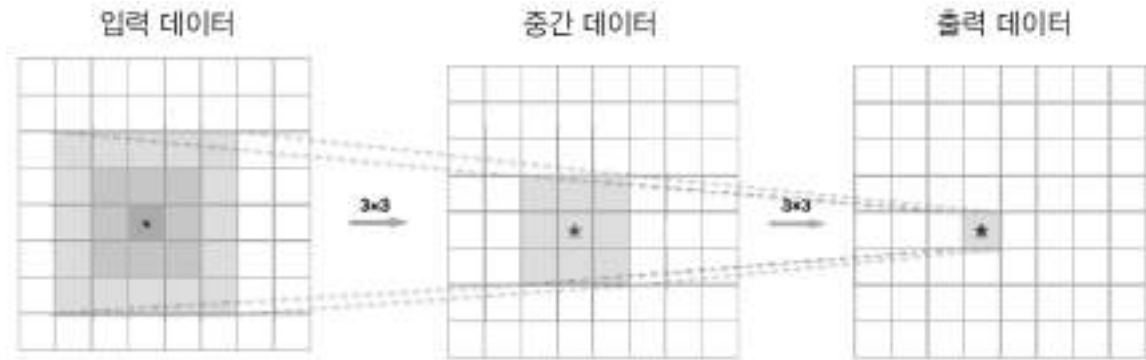
- 깊게하는 이점
 - 5x5 한번과 3x3 두번의 비교



- 매개변수 수 비교
 - 5x5 필터 1개 : 25개
 - **3x3 필터 2개 : $(3 \times 3) \times 2 = 18$ 개**
 - 7x7 필터 1개 : 49개
 - **3x3 필터 3개 : $(3 \times 3) \times 3 = 27$ 개**

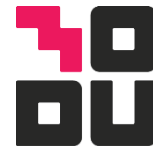
더 깊게

- 깊게하는 이점
 - 5x5 한번과 3x3 두번의 비교



- 매개변수 수 비교
 - 5x5 필터 1개 : 25개
 - **3x3 필터 2개 : $(3 \times 3) \times 2 = 18$ 개**
 - 7x7 필터 1개 : 49개
 - **3x3 필터 3개 : $(3 \times 3) \times 3 = 27$ 개**
- 학습 데이터가 줄면 학습이 빨라짐

더 깊게



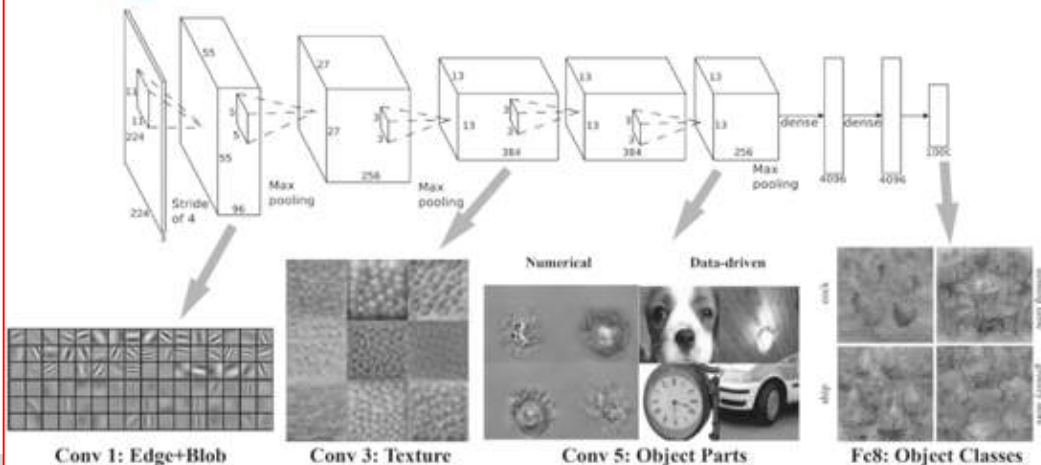
모두의연구소

- 깊게하는 이점

CNN 시각화 하기



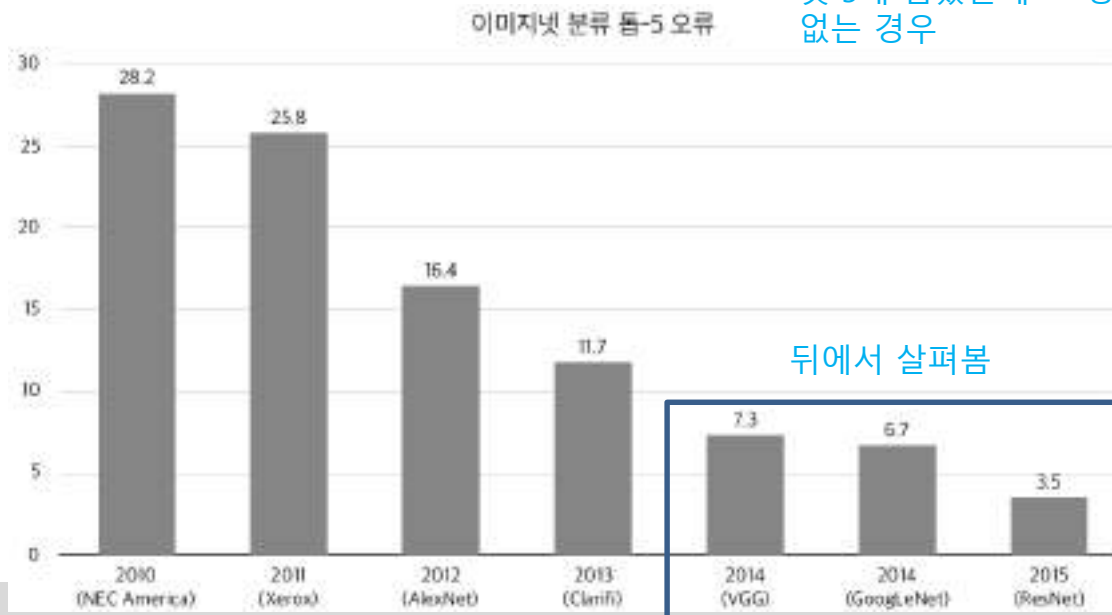
- 층 깊이에 따른 추출정보 변화
 - 1번째 층은 에지와 블랍, 3번째 층은 텍스처, 5번째 층은 사물의 일부, 마지막 완전연결 계층은 사물의 클래스에 뉴런이 반응한다
 - 층이 깊어질 수록 뉴런이 반응하는 대상이 단순한 모양에서 고급 정보로 변해감



딥러닝의 초기 역사

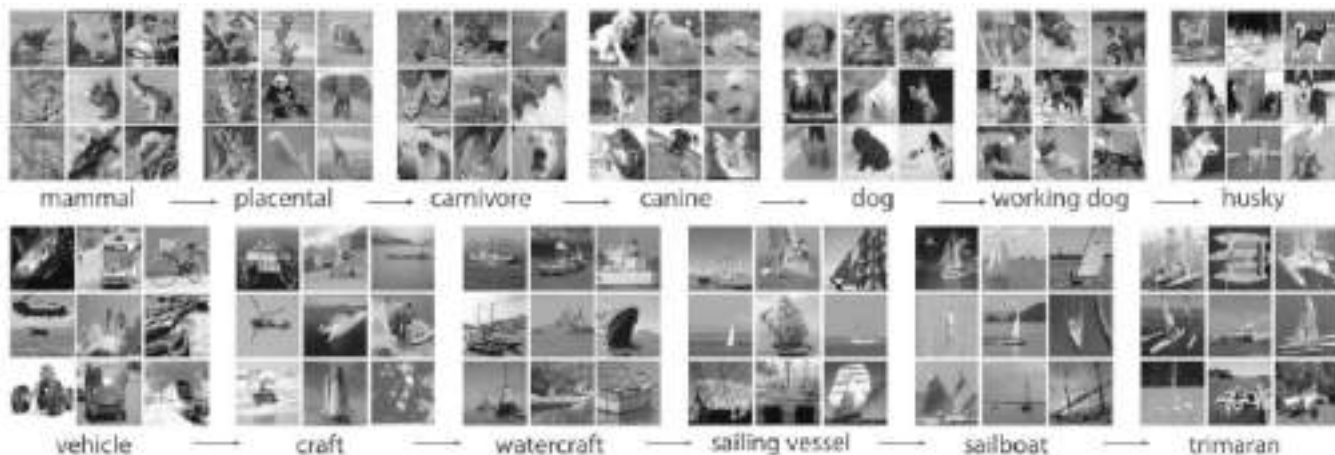
- 딥러닝 주목의 계기 : ILSVRC (ImageNet Large Scale Visual Recognition Challenge) 2012 대회
AlexNet의 높은 성능

Top-5 오류 : 확률이 가장 높은
것 5개 뽑았을때 그 중 정답이
없는 경우



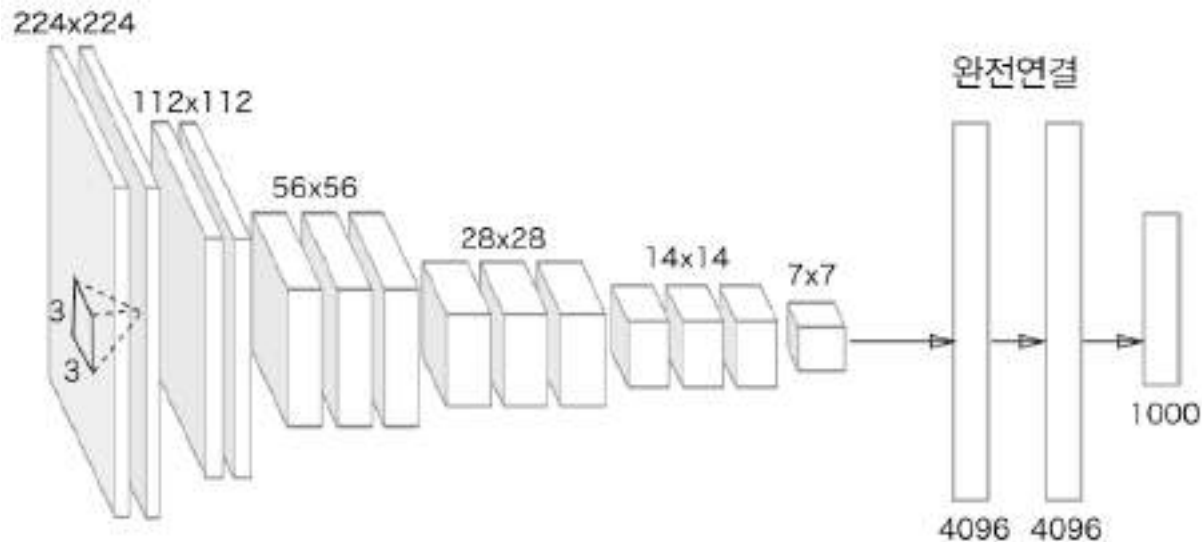
딥러닝의 초기 역사

- 이미지넷 (ImageNet)
 - 100만 장이 넘는 이미지를 담고 있음
 - 매년 열리는 ILSVRC 대회에서 사용되는 데이터
 - ILSVRC : 시험항목이 여러가지(classification, detection 등). 분류에서는 1000개의 클래스를 분류



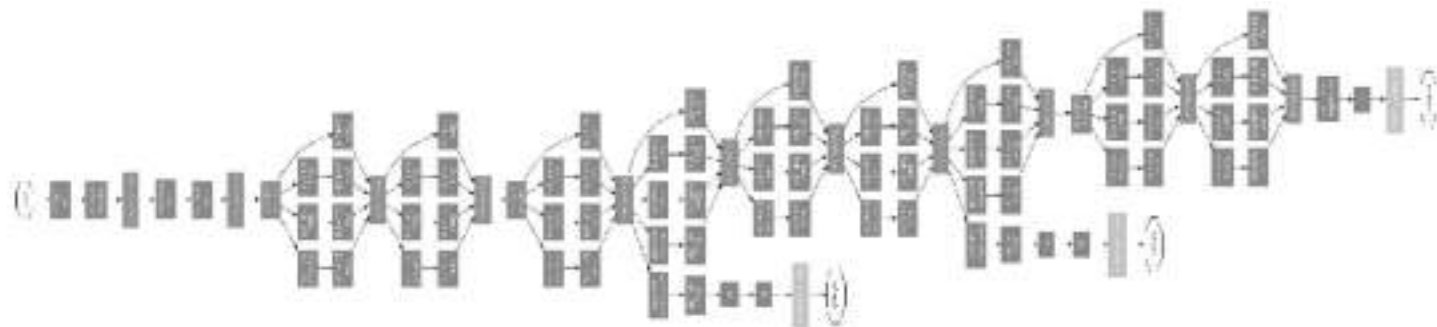
더 깊게

- VGG
 - 3x3필터의 연속
 - 16층과 19층이 성능이 좋았음(VGG-16, VGG-19)

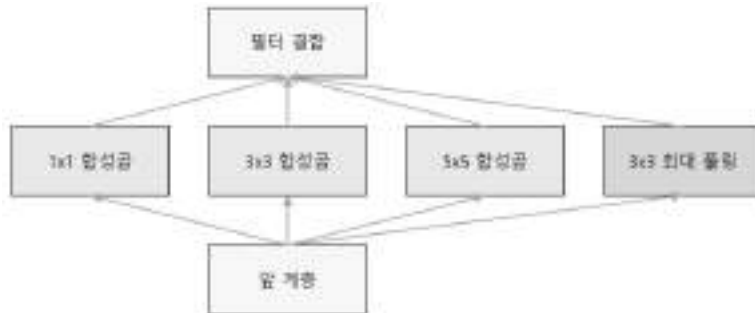


더 깊게

- GoogLeNet

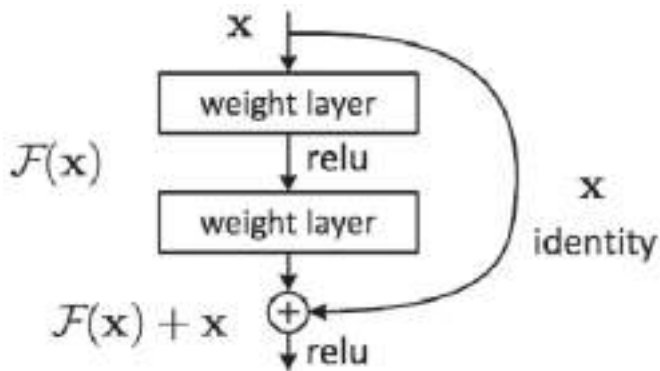
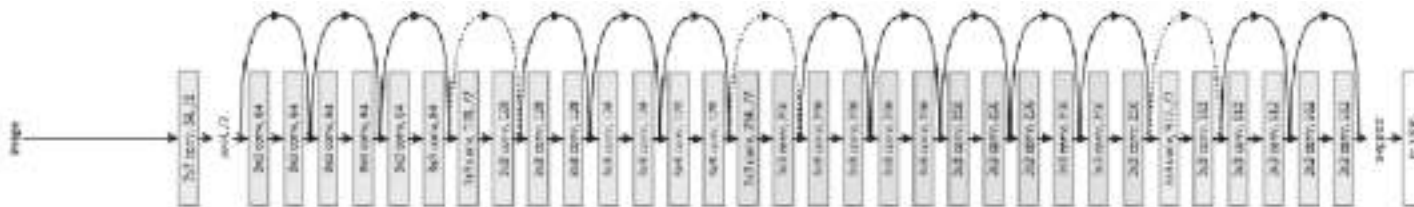


Inception 구조



더 깊게

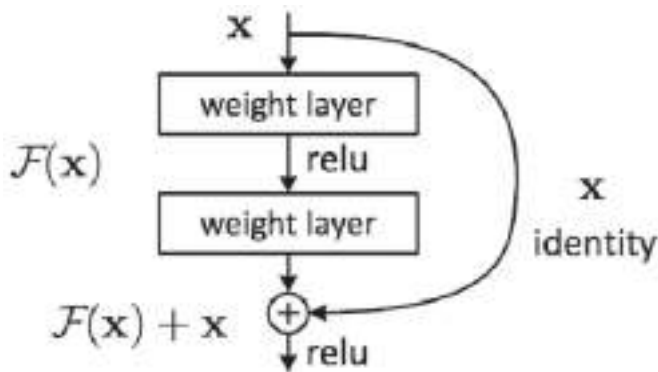
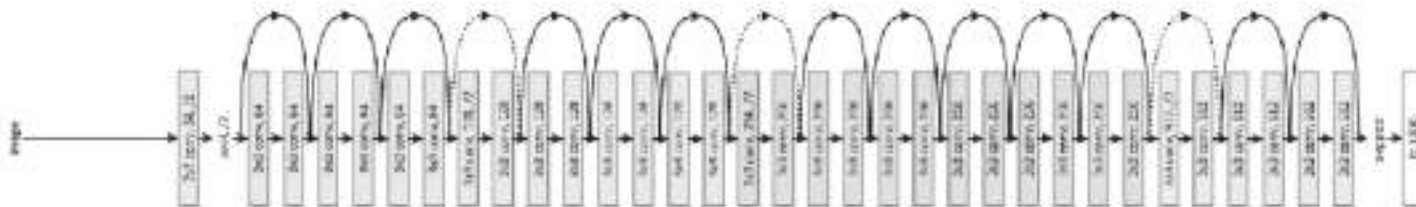
- ResNet



- 150층 이상으로 해도 정확도가 오름
 - Top-5 오류 : 3.5%

더 깊게

- ResNet

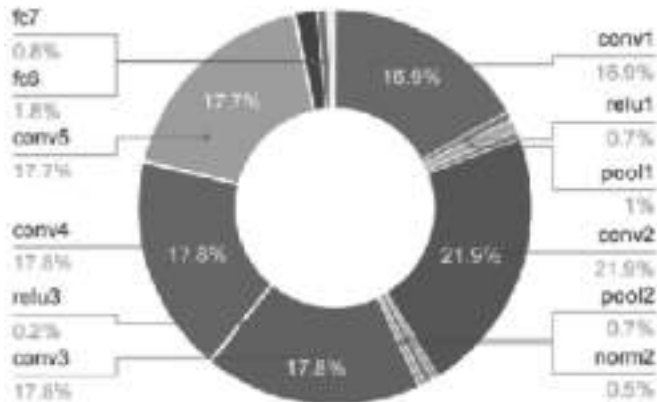


- 스킵 연결 (skip connection)
 - 입력 데이터를 그대로 흘리는 것으로 역전파 때도 상류의 기울기를 '그대로' 하류로 보냄
 - 층을 깊게 할 수록 기울기가 작아지는 소실 문제를 줄여 줌

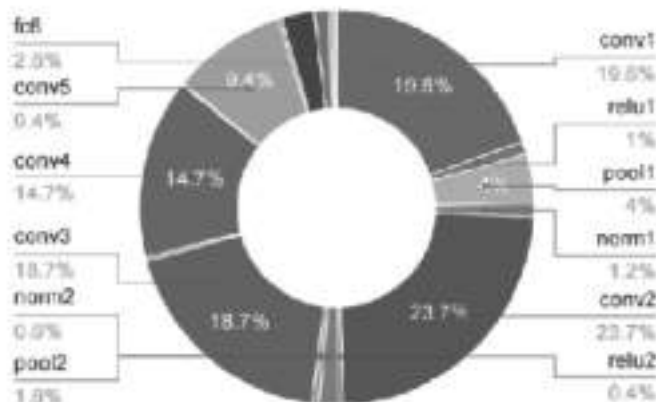
더 빠르게 (딥러닝 고속화)

- 풀어야 할 숙제
 - AlexNet 순전파 속도

GPU Forward Time Distribution



CPU Forward Time Distribution

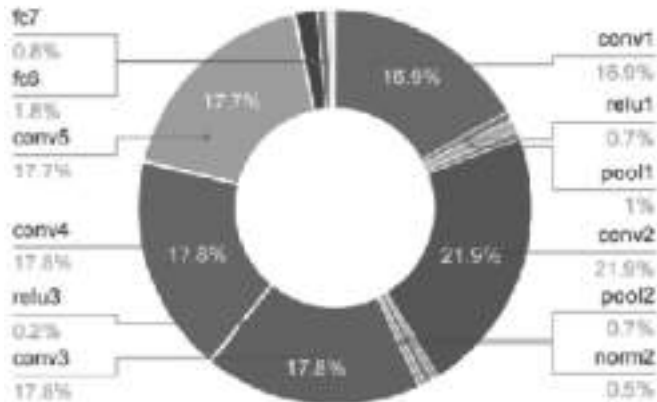


- 합성곱(Convolution) 계층에서 오랜 시간 소요
 - GPU에서 95%, CPU에서 89%
 - 학습 때도 마찬가지 임

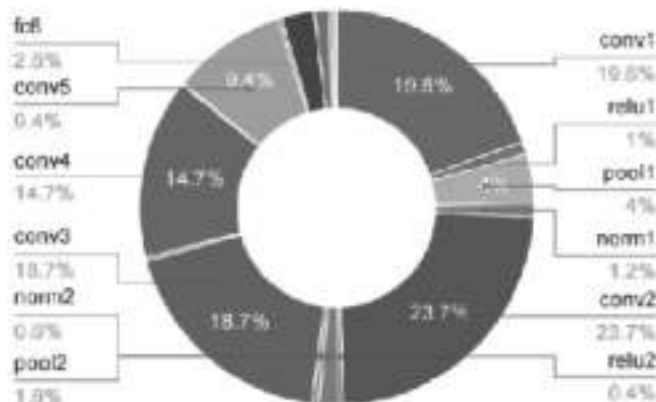
더 빠르게 (딥러닝 고속화)

- 풀어야 할 숙제
 - AlexNet 순전파 속도

GPU Forward Time Distribution



CPU Forward Time Distribution

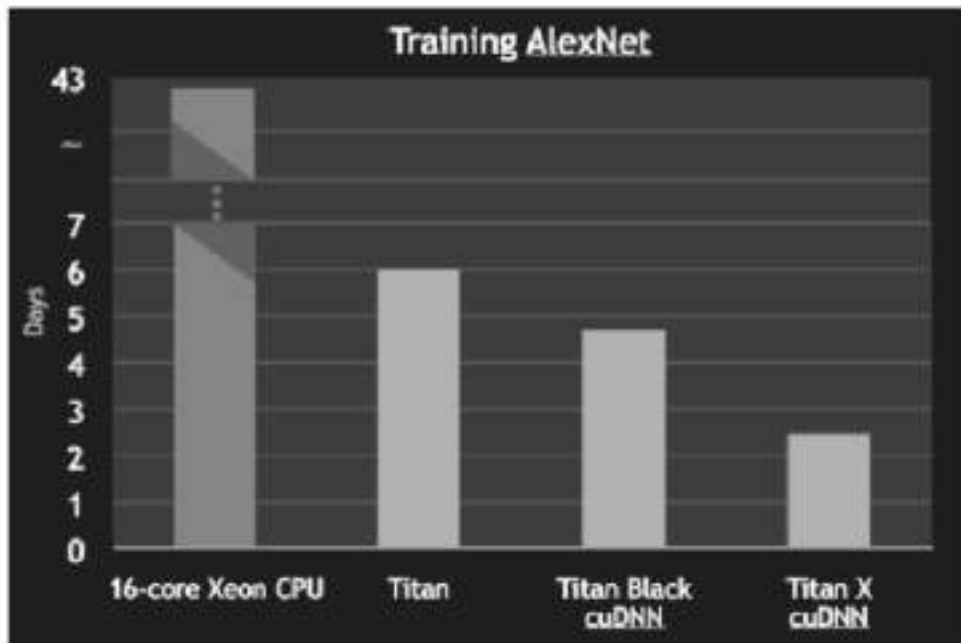


- 매트릭스 곱 (단일 곱셈-누산)연산이 빨라야 함

더 빠르게 (딥러닝 가속화)

- GPU를 활용한 가속화

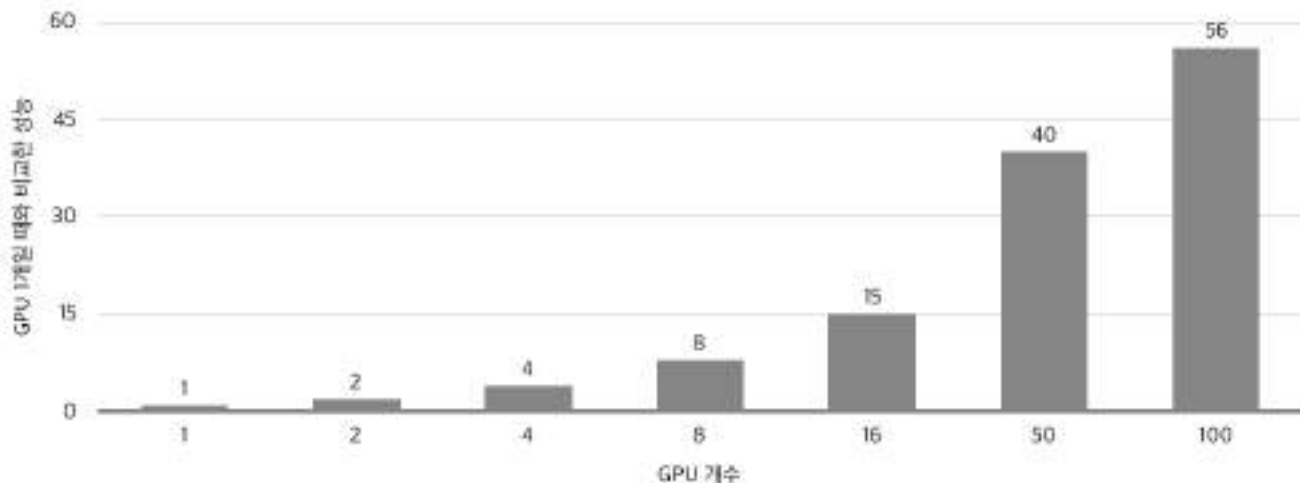
- 엔비디아 GPU 컴퓨팅 통합개발 환경 : CUDA
- 엔비디아는 cuDNN 라이브러리 제공



AlexNet의 학습 시간을 16코어 제온 CPU와 엔비디아 Titan X와 비교

더 빠르게 (딥러닝 가속화)

- 분산학습



텐서플로우 분산학습 성능

56배가 빨라지면 7일짜리 작업이 불과 3시간만에 끝남

더 빠르게 (딥러닝 고속화)

- 연산정밀도와 비트 수 줄이기
 - 메모리 용량과 버스 대역폭 등이 딥러닝 고속화에 병목이 될 수 있음
 - 학습 시 대량의 가중치 매개변수와 중간 데이터를 메모리에 저장해야 함
 - GPU의 버스를 흐르는 데이터가 많으면 병목발생
 - 따라서 데이터를 작게 저장하면 좋음
 - 실수 표현시 64비트와 32비트인데 많은 비트를 사용하면 계산 오차를 줄이지만 메모리사용량 증가
- 딥러닝은 높은 수치 정밀도를 요구하지 않음
 - 신경망에 노이즈가 조금 섞여도 출력결과가 잘 달라지지 않음

더 빠르게 (딥러닝 고속화)

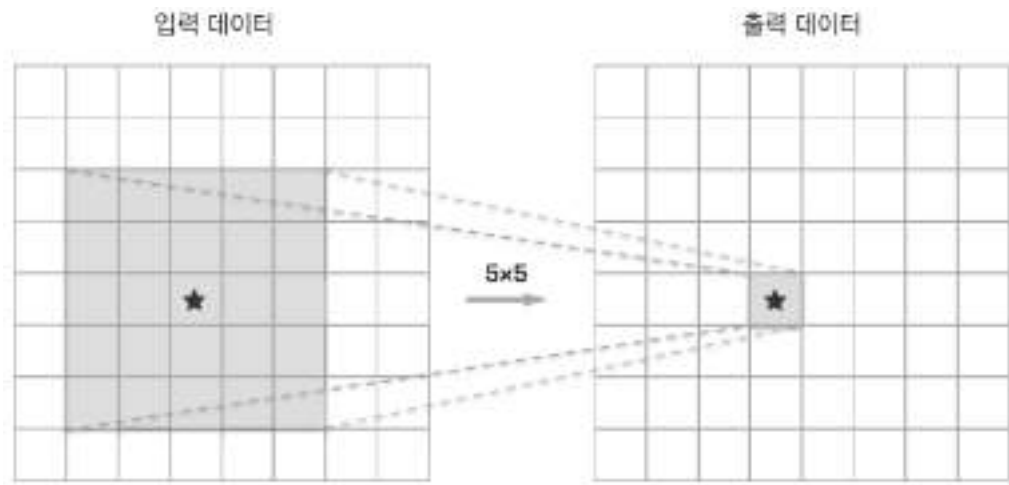
- 연산정밀도와 비트 수 줄이기
 - 16비트 반정밀도(half-precision) 부동 소수점 사용해도 학습에 큰 문제가 없다고 함
 - 엔비디아 파스칼(Pascal) 아키텍처 GPU는 16비트 반정밀도를 지원
 - 반정밀도 부동소수점이 표준적으로 이용될 것으로 예상함
- 넘파이 16비트 반정밀도 부동 소수점 지원함
 - 스토리지로서 16비트이지 연산자체는 16비트를 수행하지 않음
 - 엔비디아 맥스웰 아키텍처 GPU도 마찬가지
 - 3_day/half_float_network.py
- 1비트로 하는 연구도 있음

정리

- 수 많은 문제에서 신경망을 더 깊게 하여 성능을 개선할 수 있다
- 이미지 인식 기술대회인 ILSVRC에서는 최근 딥러닝 기반 기법이 상위권을 독점하고 있으며, 그 깊이도 더 깊어지는 추세이다
- 유명한 신경망으로는 VGG, GoogLeNet, ResNet이 있다
- GPU와 분산학습, 비트 정밀도 감소 등으로 딥러닝을 고속화 할 수 있다

더 작은 필터

- 작은 필터로 더 깊게
 - 5x5 한번과 3x3 두번의 비교



요약



모두의연구소

- ConvNet은 CONV, POOL, FC 레이어로 구성
- 작은 필터로 더 깊은 네트워크를 구성하는게 트렌드
- FC레이어를 제거하는게 트렌드
 - 추후 영상처리 부분에서 다루게 됨
- 전형적 구조
 - $[(\text{CONV-RELU}) * N - \text{POOL}] * M - (\text{FC-RELU}) * K, \text{SOFTMAX}$
 - N : 보통 ~5, M : 크게, $0 \leq K \leq 2$
- 최근엔 전혀적인 구조 보다 다양한 CNN 모델 변형들이 많아짐



박은수 Research Director

E-mail : es.park@modulabs.co.kr