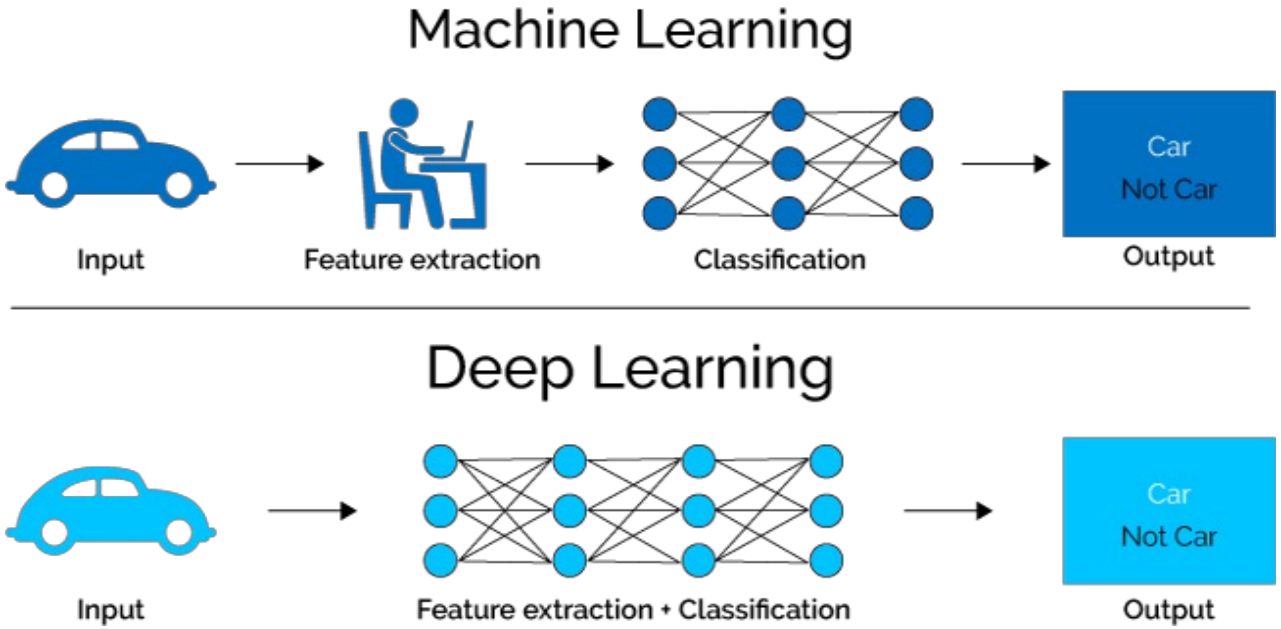
PyTorch 배우는 딥러닝 입문 (2)

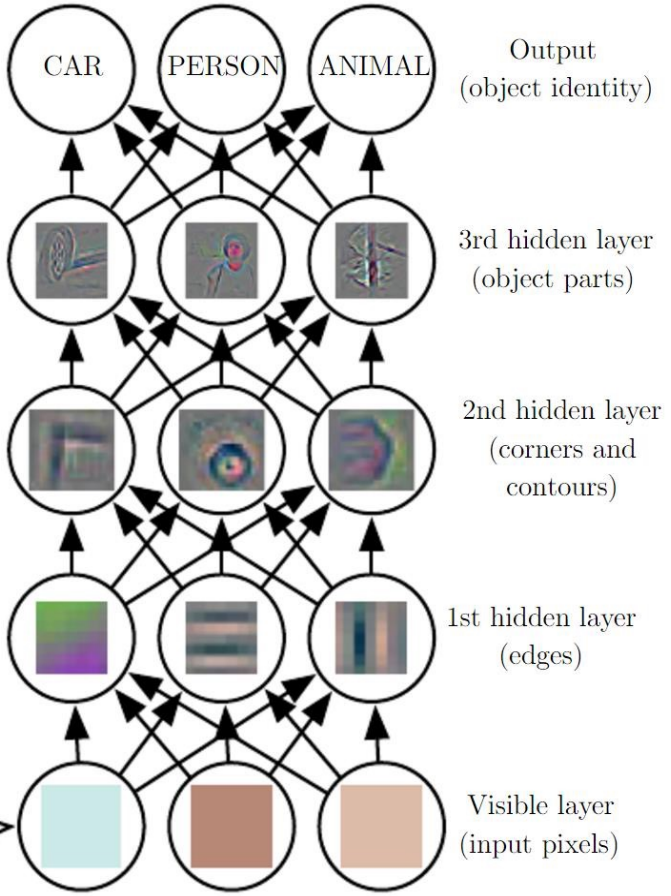
Introduction to Convolutional Neural Network

이영완

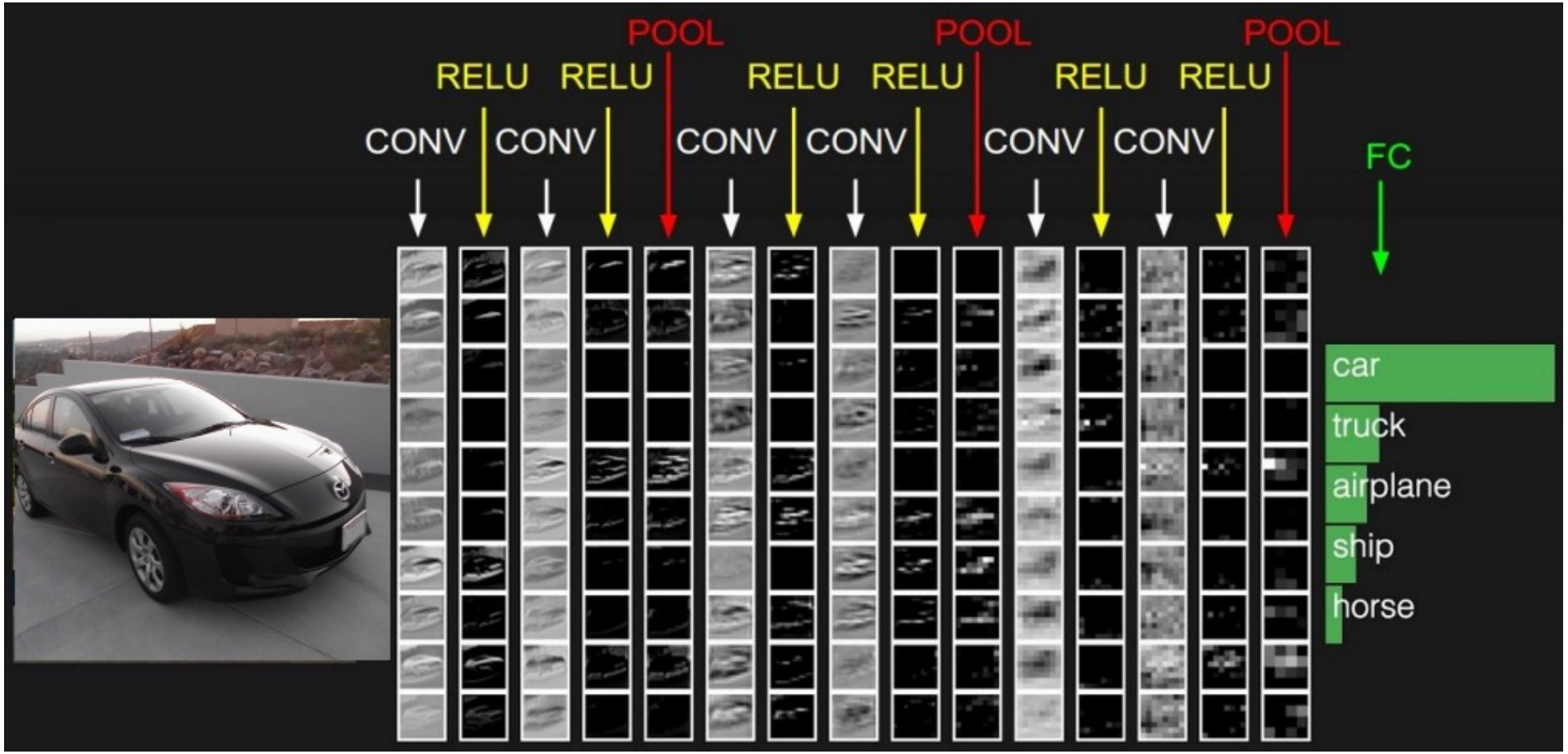
한국전자통신연구원

Deep Learning is Representation Learning
(aka Feature Learning)





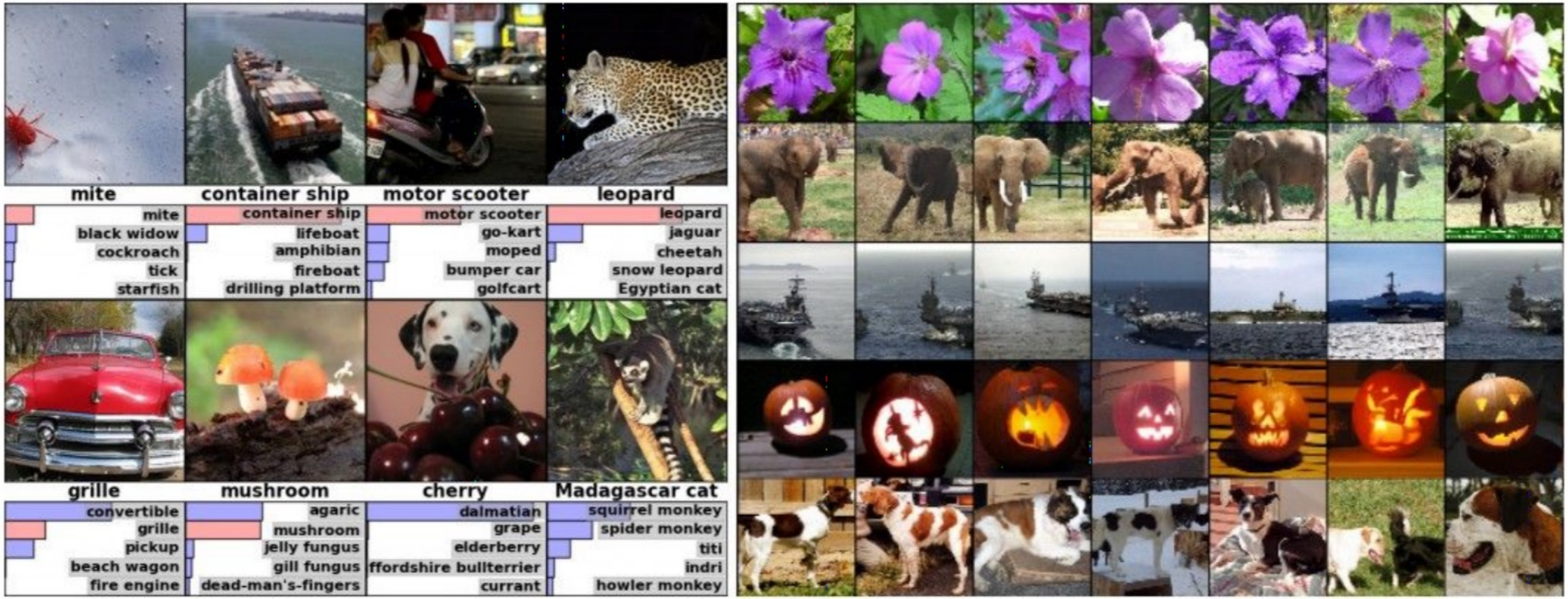
Convolution Neural Network (CNN)



CNNs are everywhere

Classification

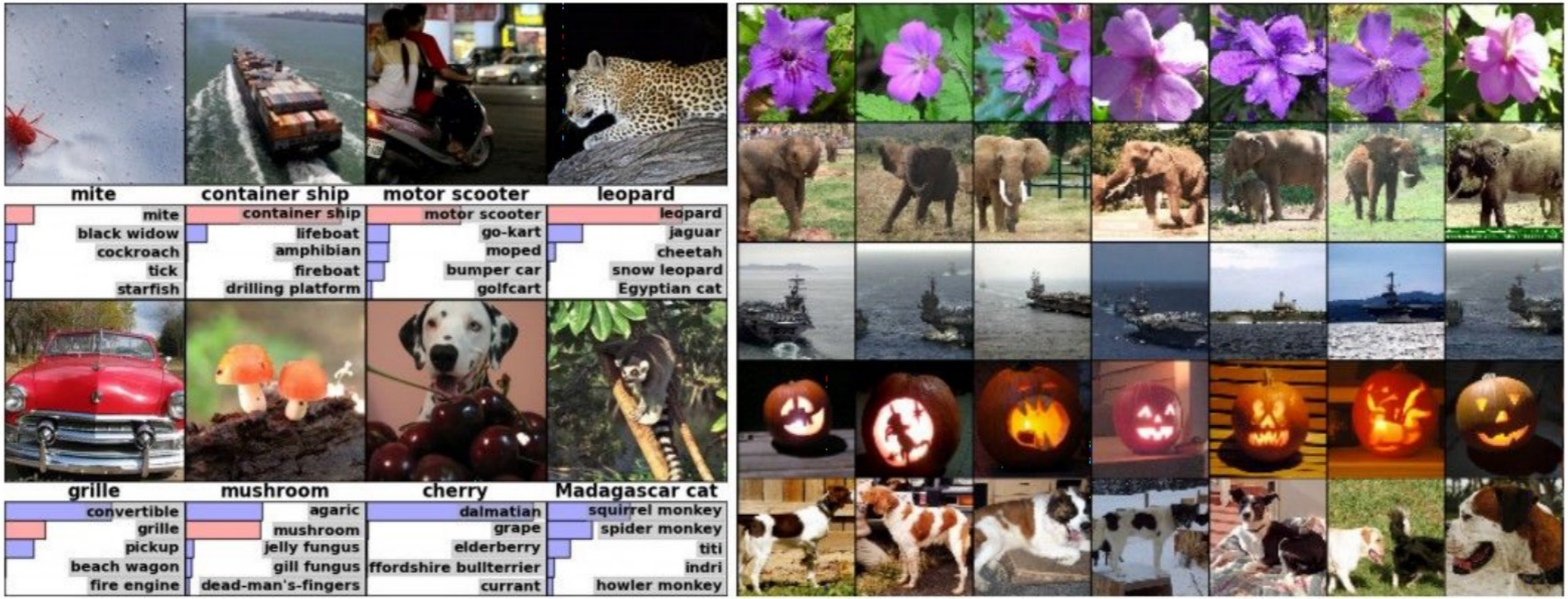
Retrieval



CNNs are everywhere

Classification

Retrieval



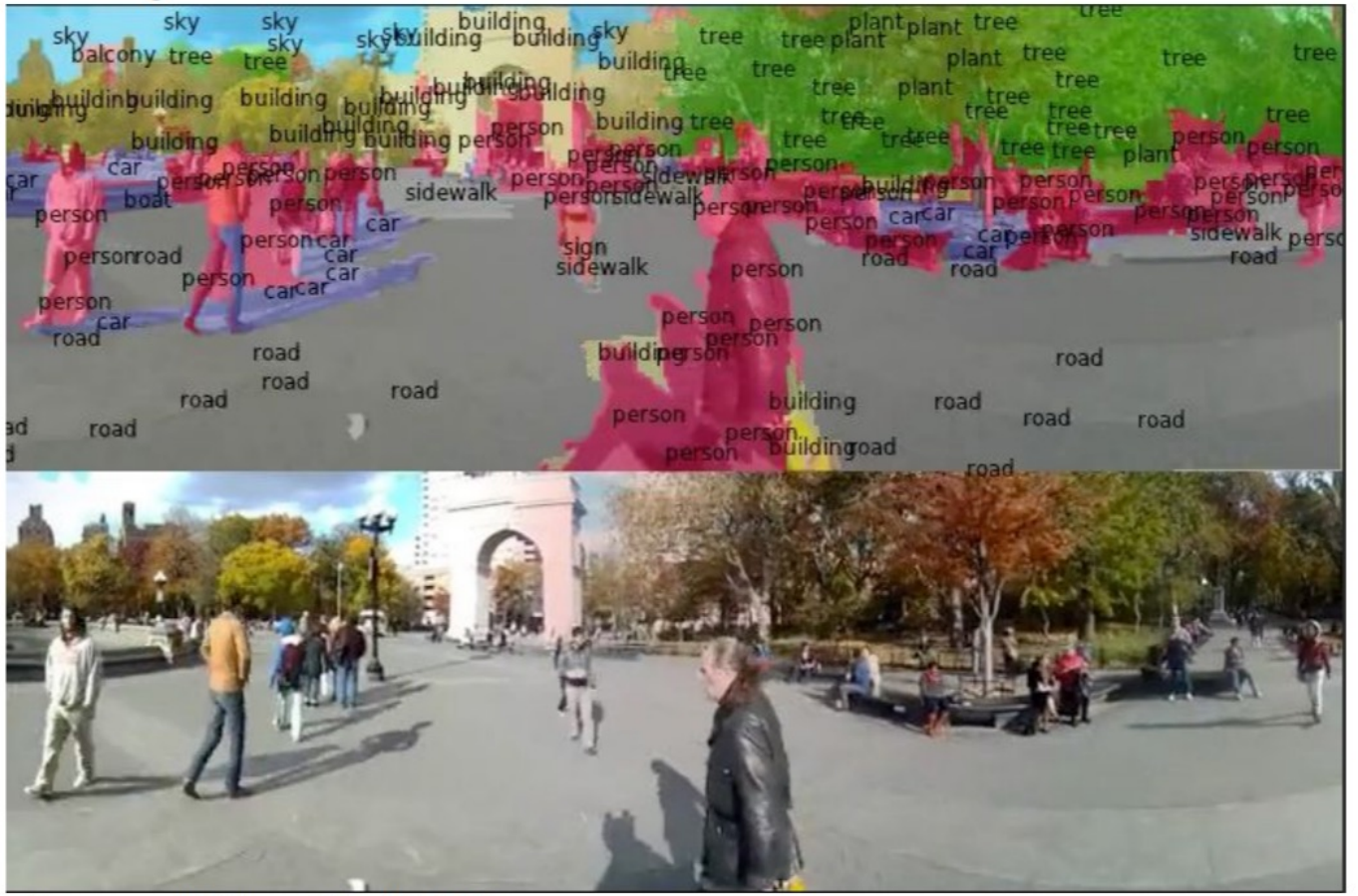
CNNs are everywhere

Detection

Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[Farabet et al., 2012]

CNNs are everywhere

Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line

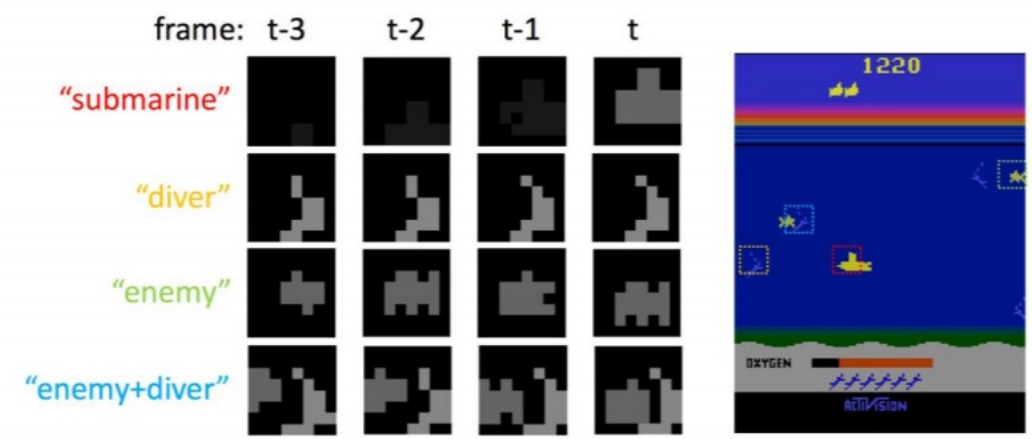
(these are the GPUs on rye01.stanford.edu)

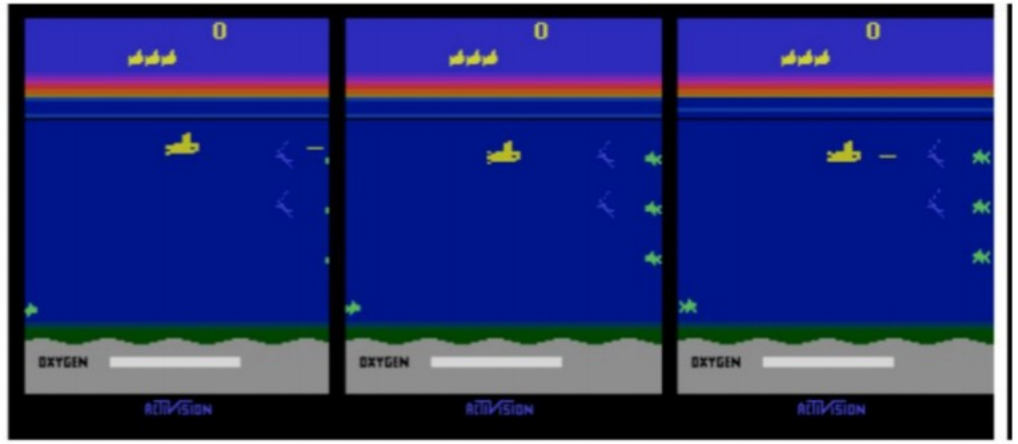
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

CNNs are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]





[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

CNNs are everywhere

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

Somewhat related

A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

Image Captioning

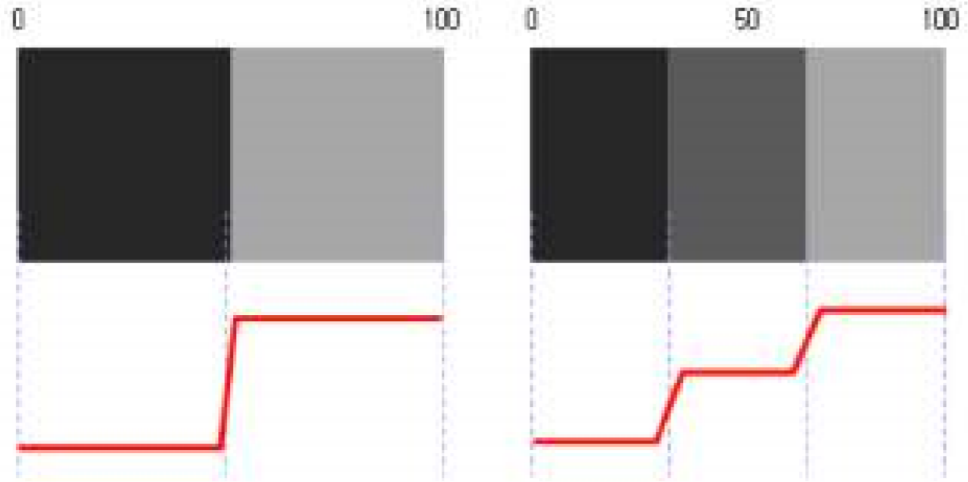
[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

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

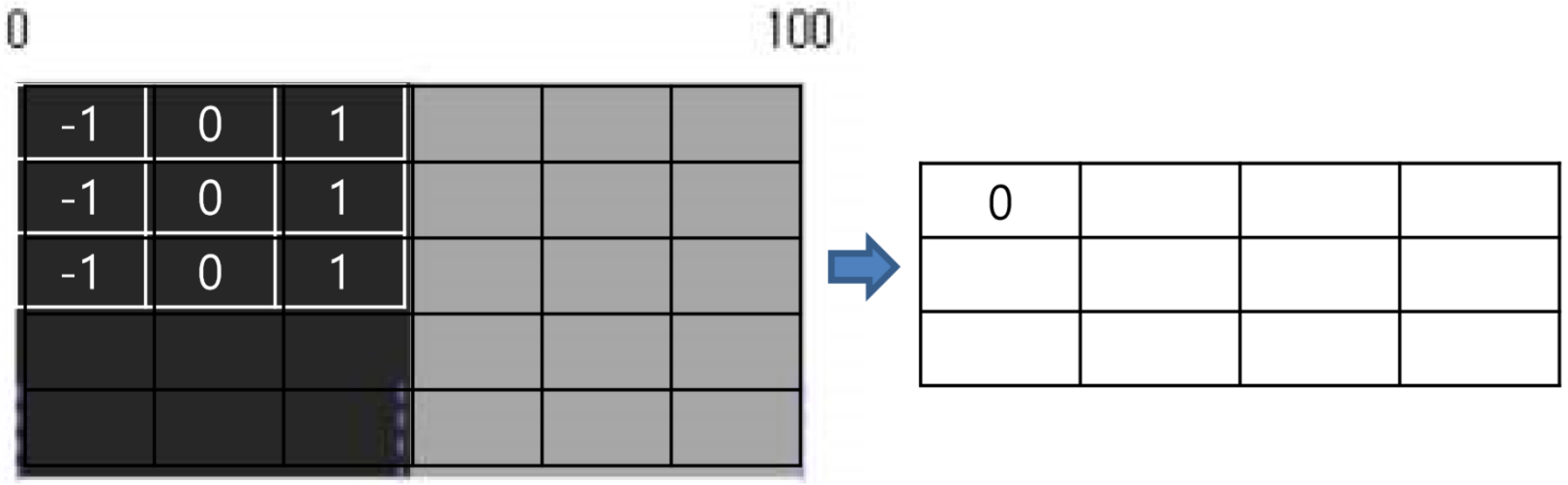
수직 에지를 찾는
컨볼루션 필터



Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

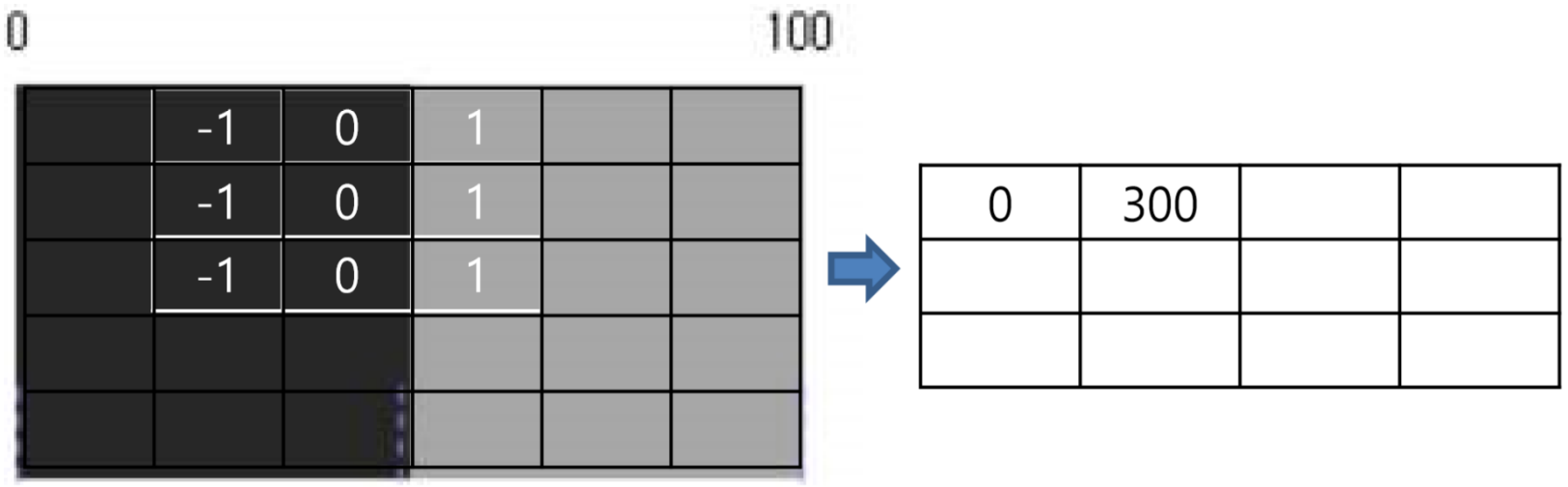
수직 엣지를 찾는
컨볼루션 필터



Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

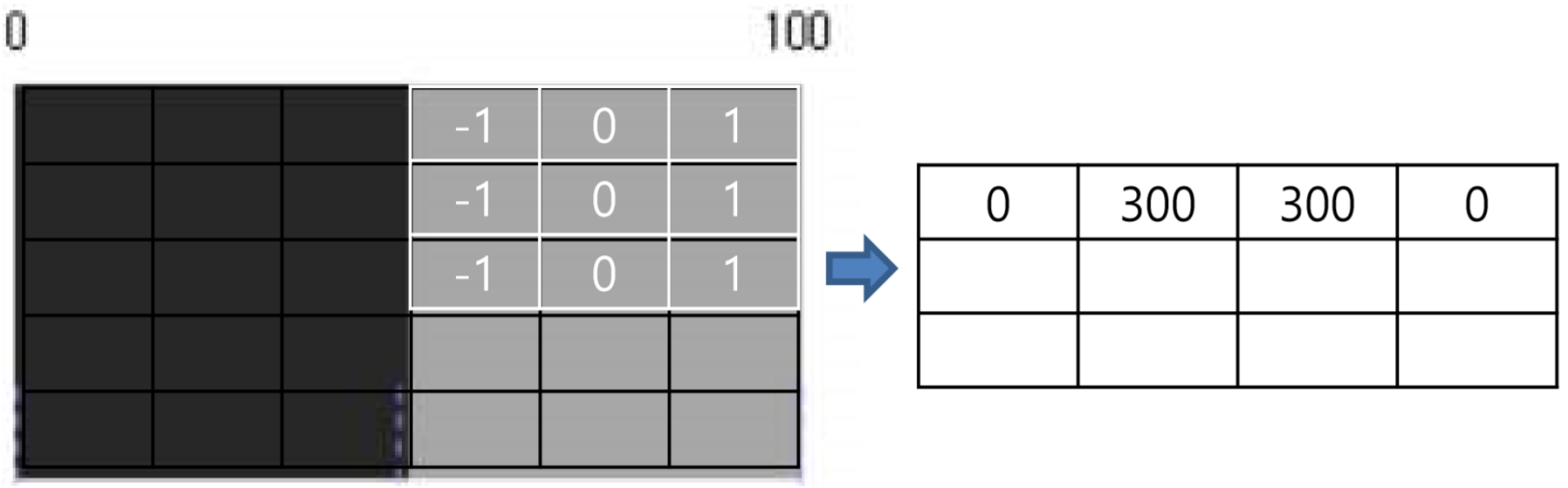
수직 엣지를 찾는
컨볼루션 필터



Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

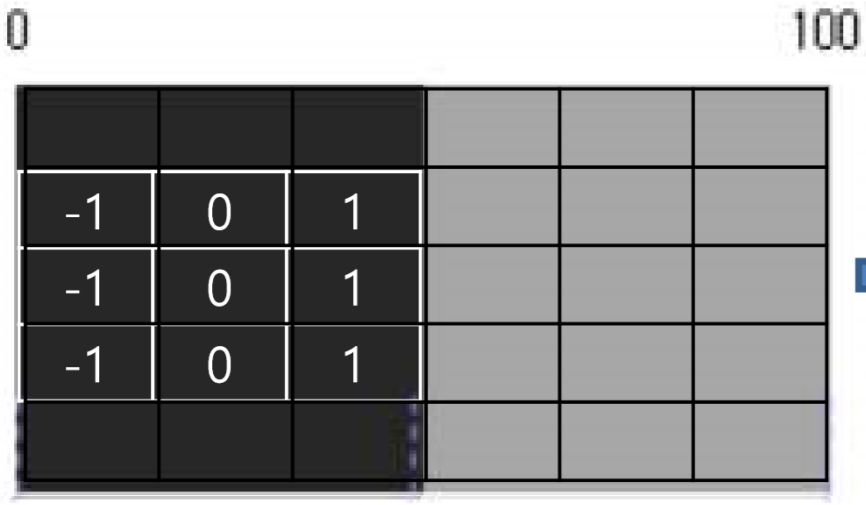
수직 엣지를 찾는
컨볼루션 필터



Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

수직 엣지를 찾는
컨볼루션 필터



0	300	300	0
0			

Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

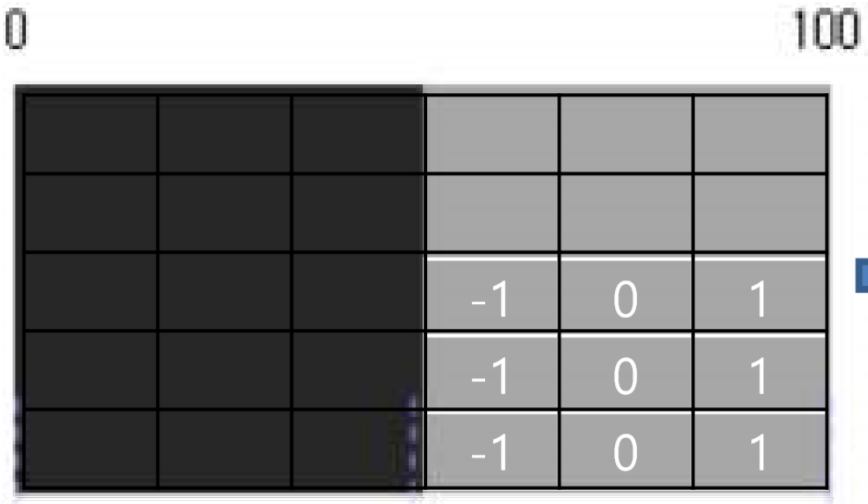
수직 엣지를 찾는
컨볼루션 필터

0	300	300	0
0	300		

Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

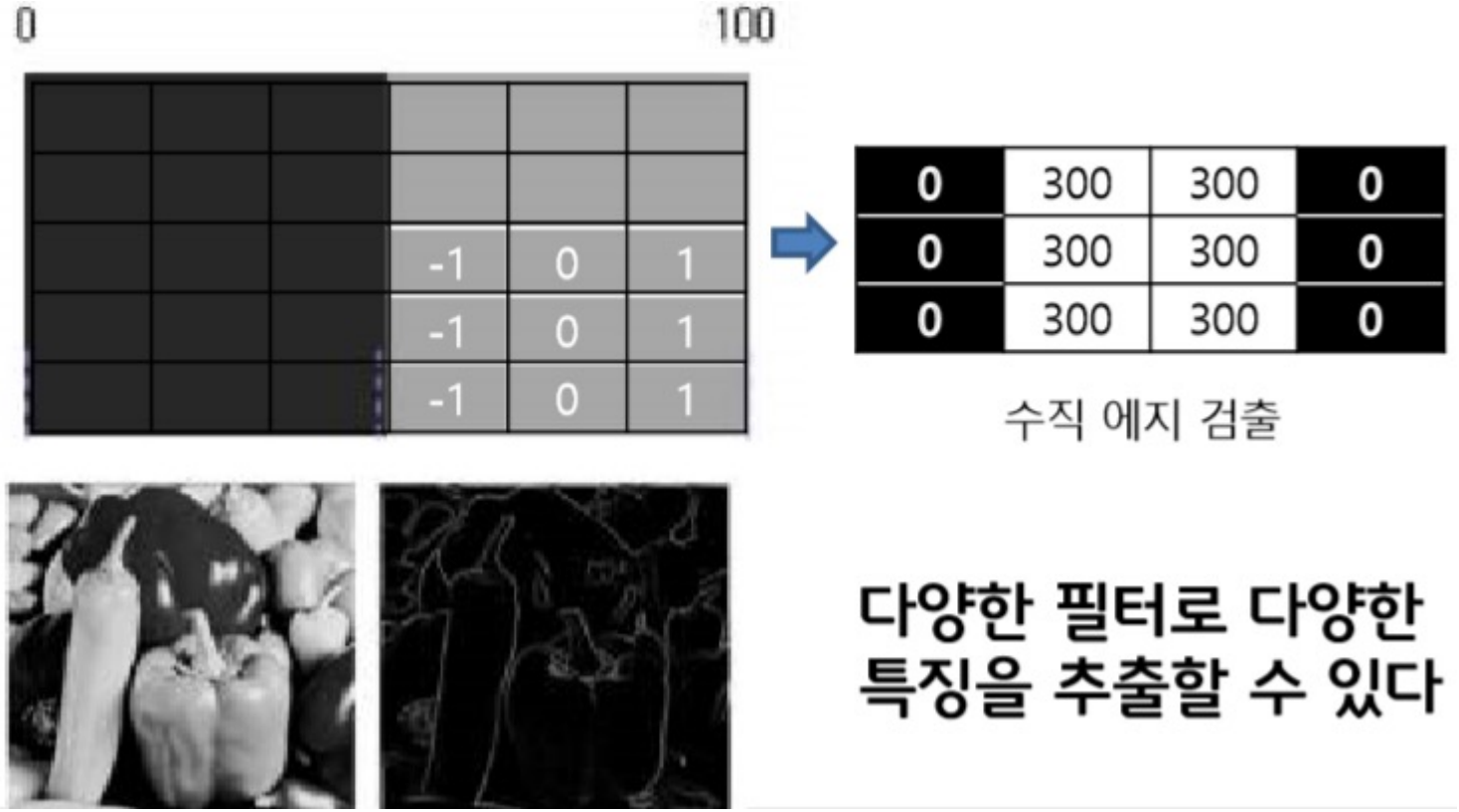
수직 엣지를 찾는
컨볼루션 필터



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

Convolutional Neural Network(CNN)

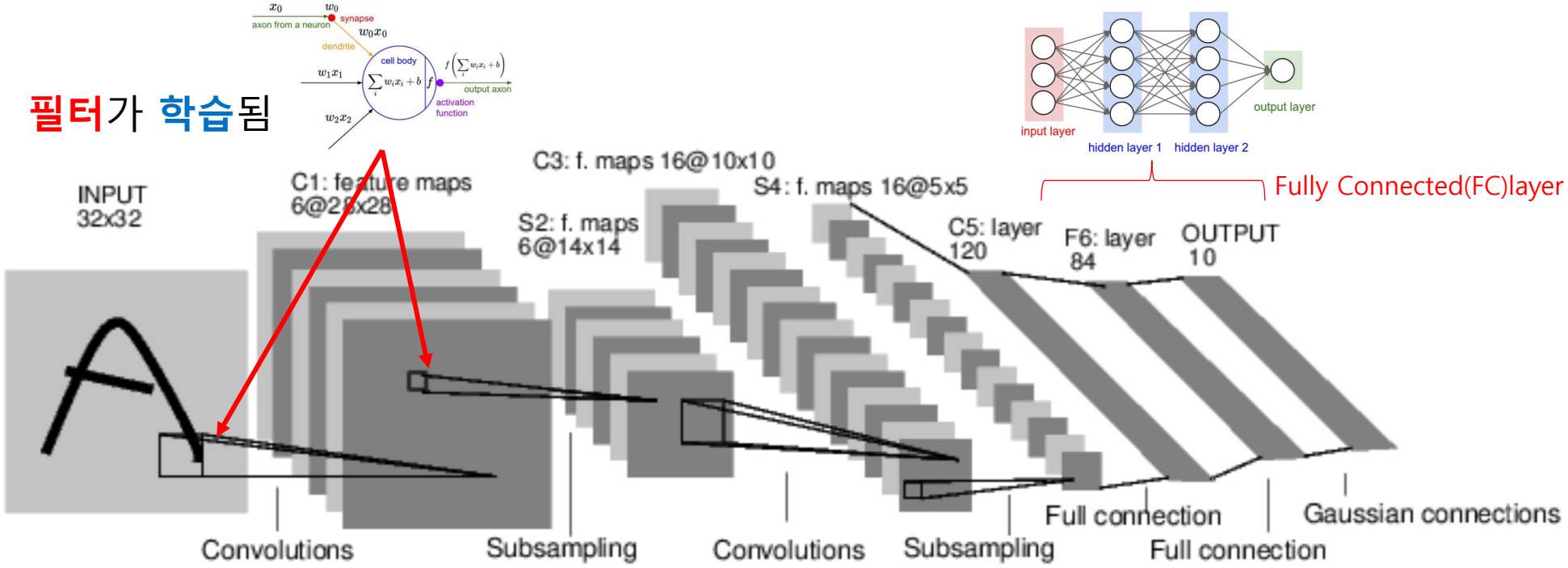
Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.



Convolutional Neural Network(CNN)

Convolution Filter : 이미지의 **특징(feature)**을 추출할 수 있다.

→ Convolutional Neural Network는 컨볼루션 필터를 **학습**



[LeNet-5, LeCun 1980]

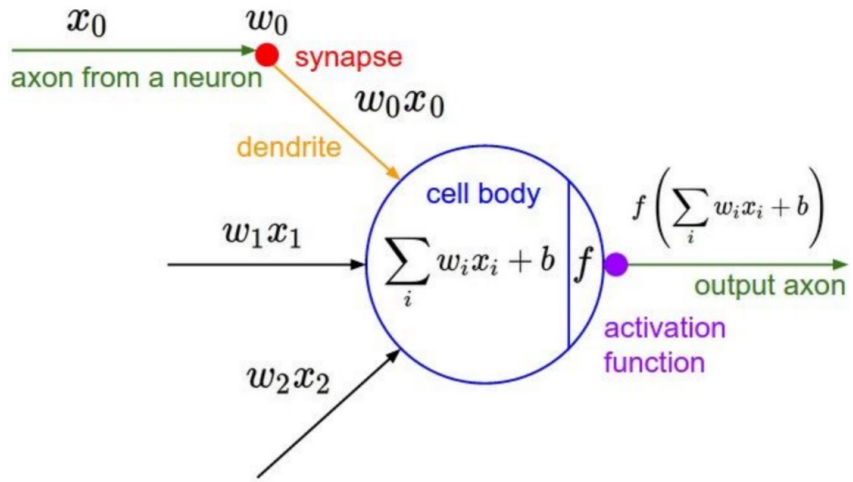
Visualization

https://poloclub.github.io/cnn-explainer/

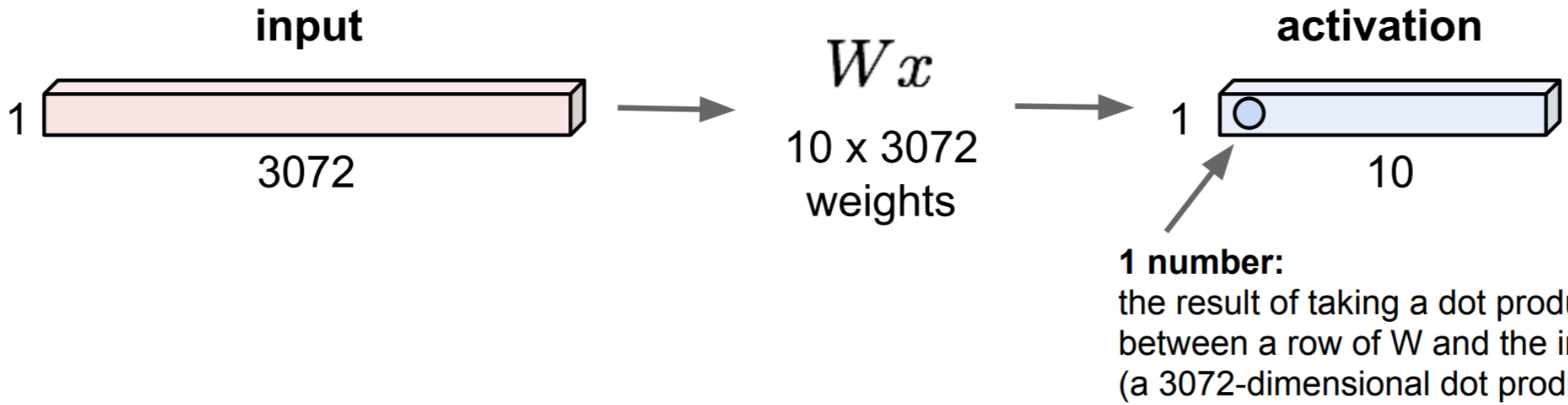
https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Delving deep into CNNs

Fully Connected Layer

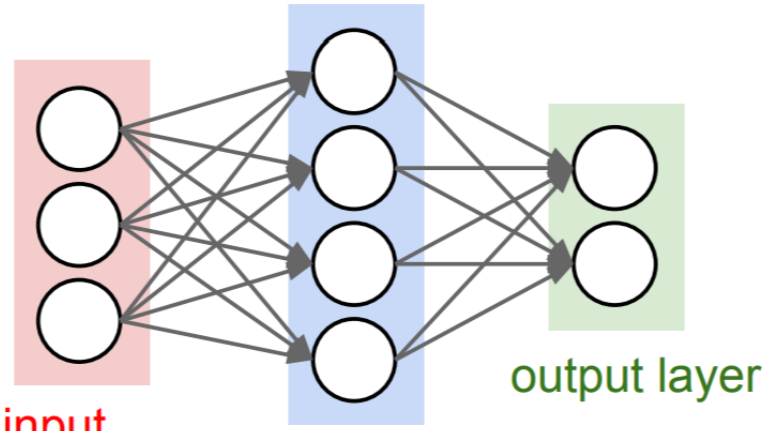


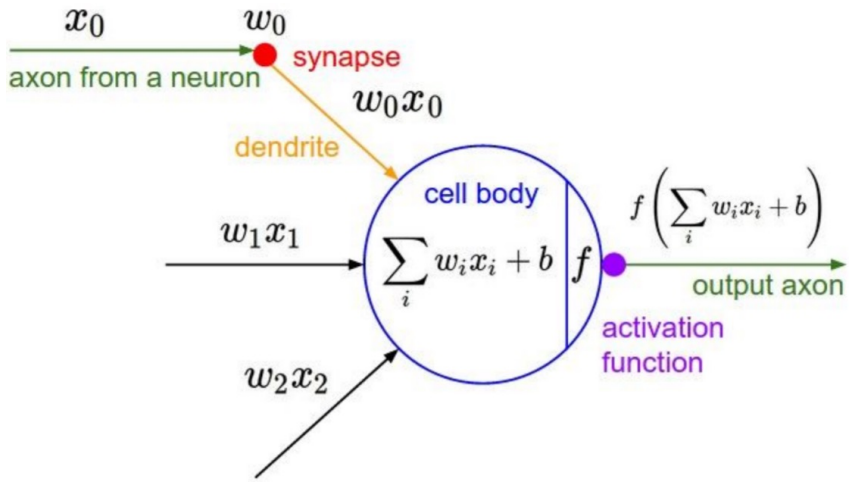
32x32x3 image -> stretch to 3072 x 1



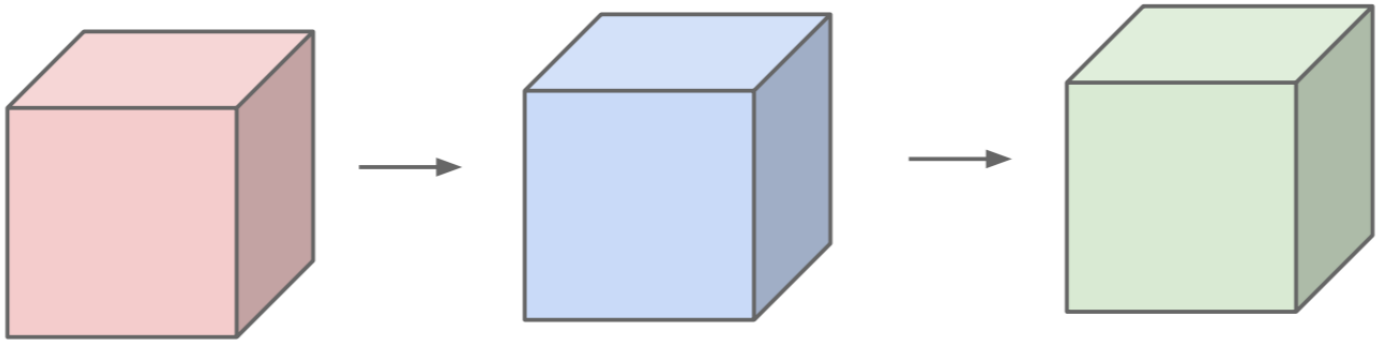
before:

Fully Connected(FC)



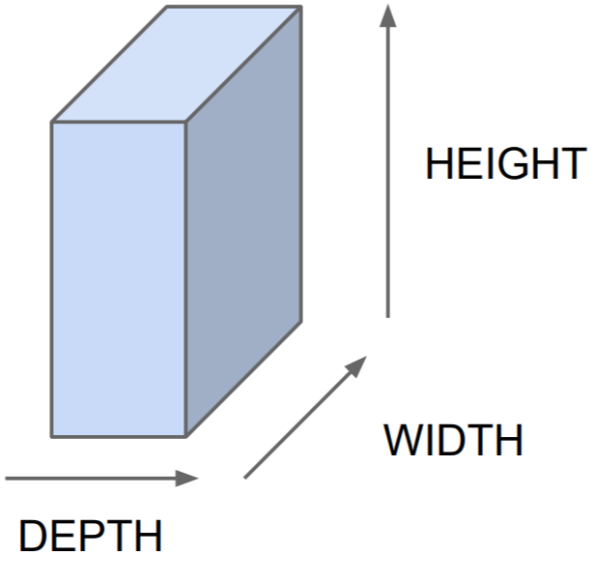


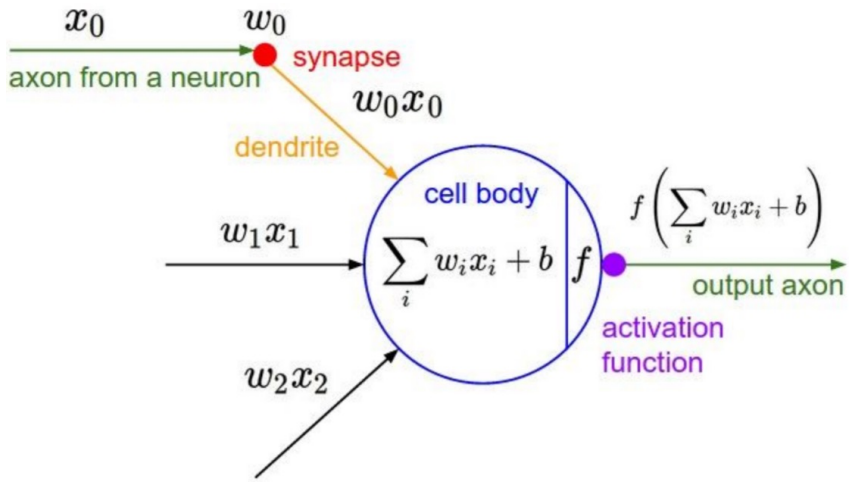
now:



Convolution

All Neural Net activations arranged in **3 dimensions:**



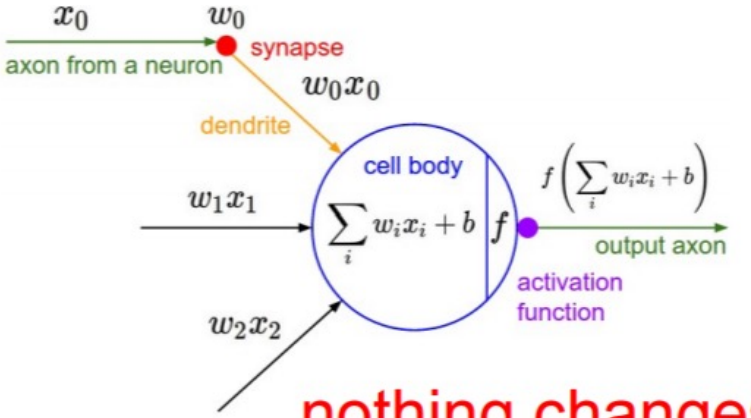


For example, a CIFAR-10 image is a 32x32x3 volume
32 width, 32 height, 3 depth (RGB channels)

pytorch Tensor dimension : N x C x H x W
→ N : batch size , C : Channel(depth)

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes

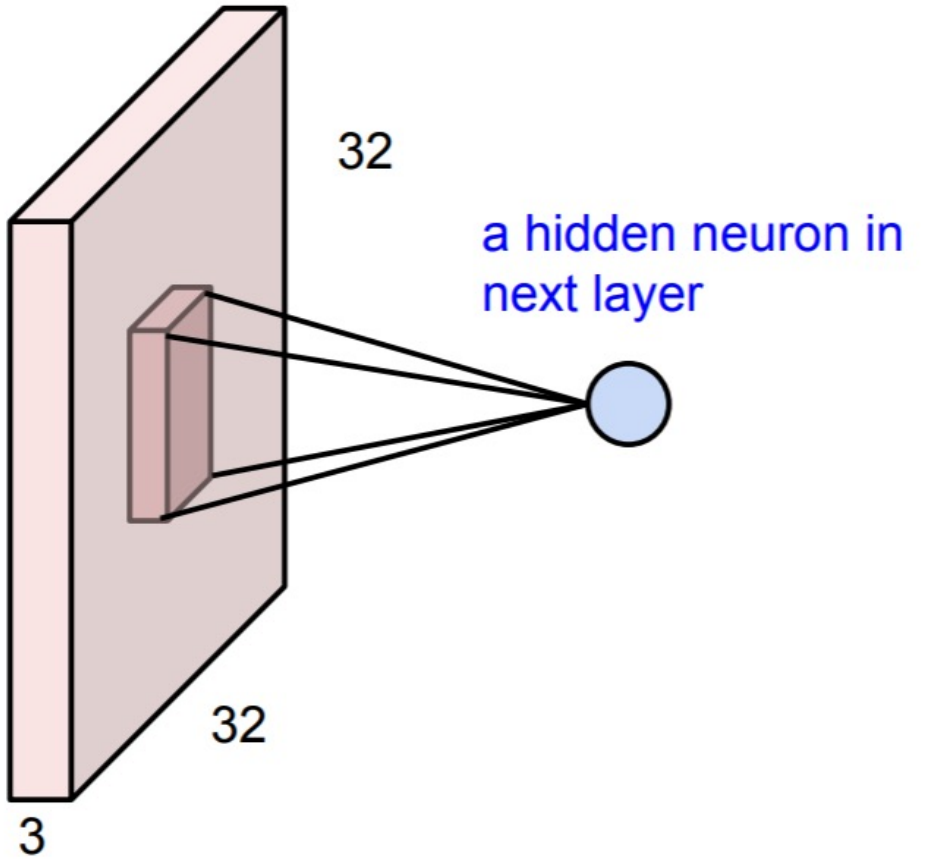


image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

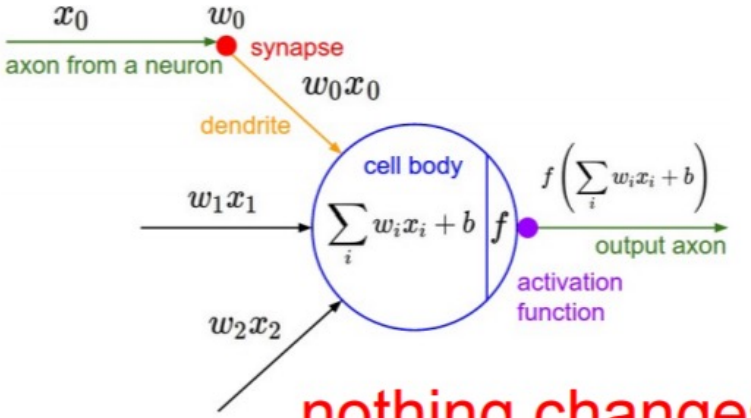
now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

note that connectivity is:

- <u>local in space</u> (5x5 inside 32x32)
- <u>but full in depth</u> (all 3 depth channels)

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes

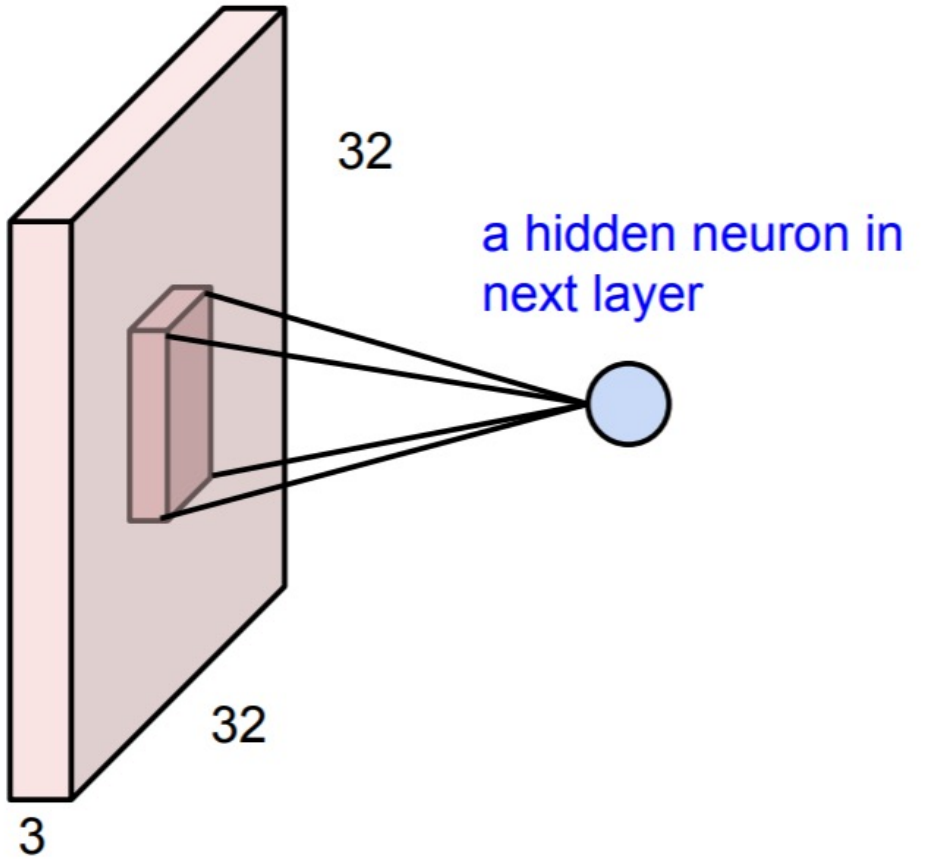


image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

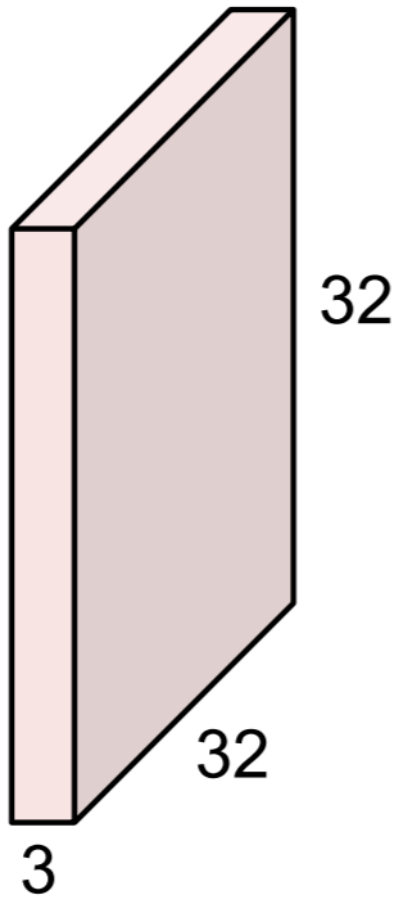
note that connectivity is:

- <u>local in space</u> (5x5 inside 32x32)
- <u>but full in depth</u> (all 3 depth channels)

Convolution Layer

Filters always extend the full depth of the input volume

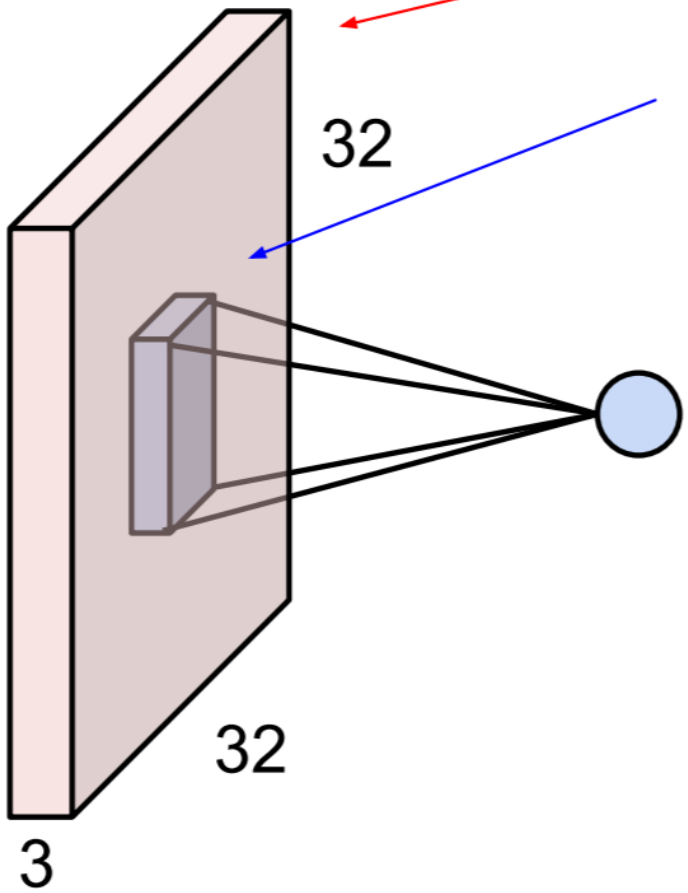
32x32x3 image



5x5x3 filter

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

Convolution Layer



32x32x3 image

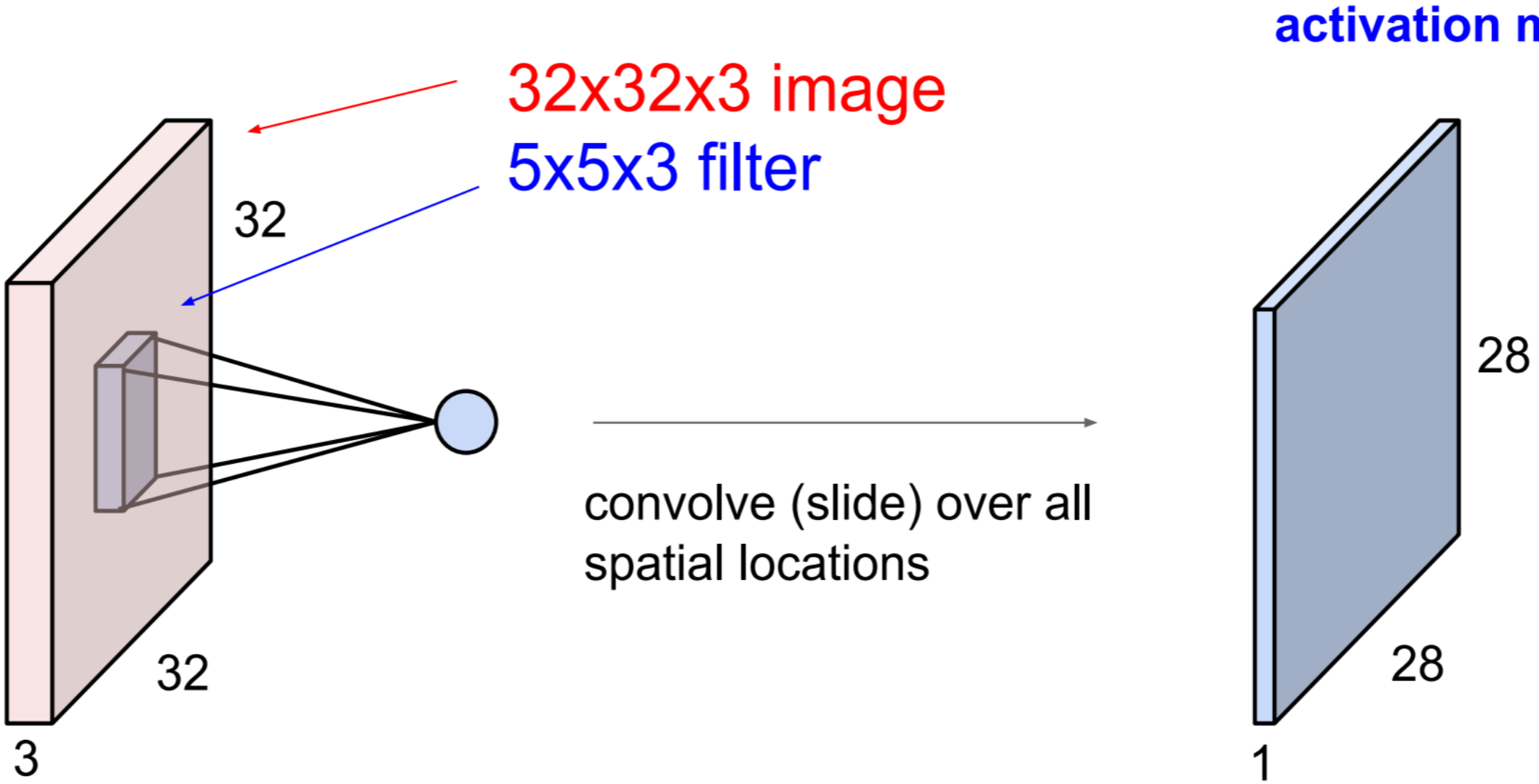
5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

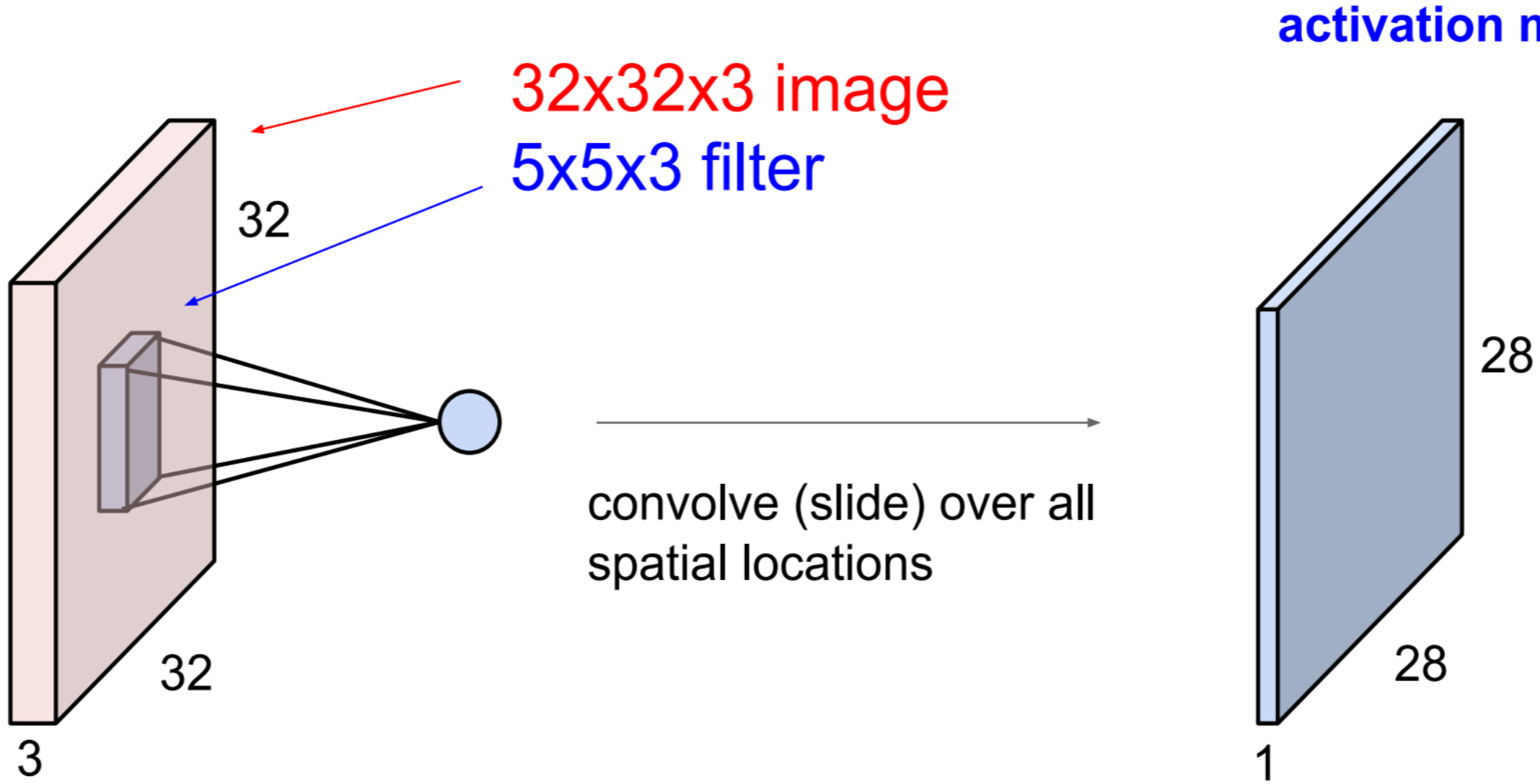
Reference : Stanford Univ. cs231n Lecture note 5

Convolution Layer



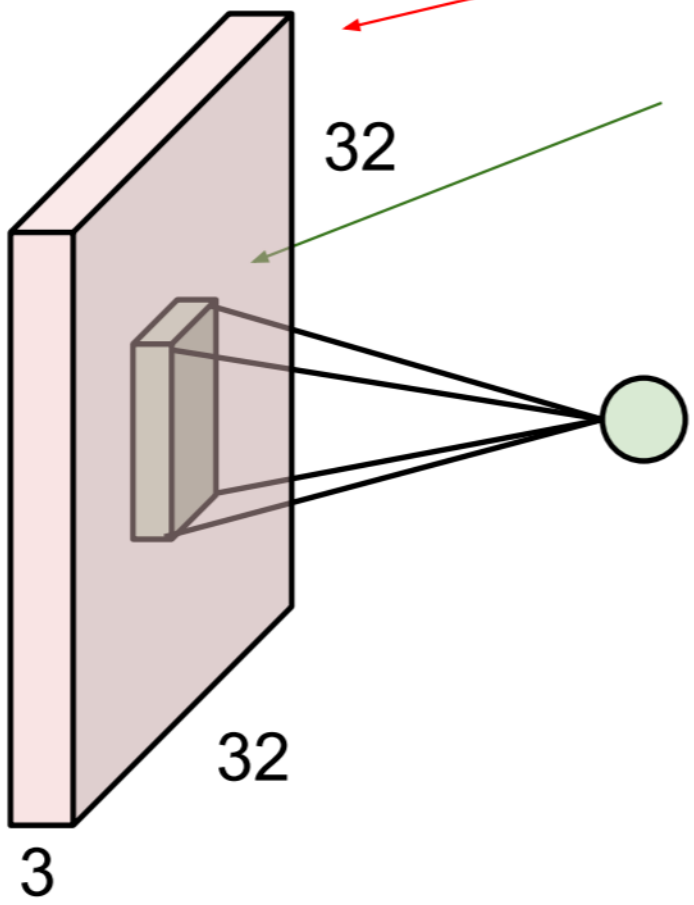
Reference : Stanford Univ. cs231n Lecture note 5

Convolution Layer

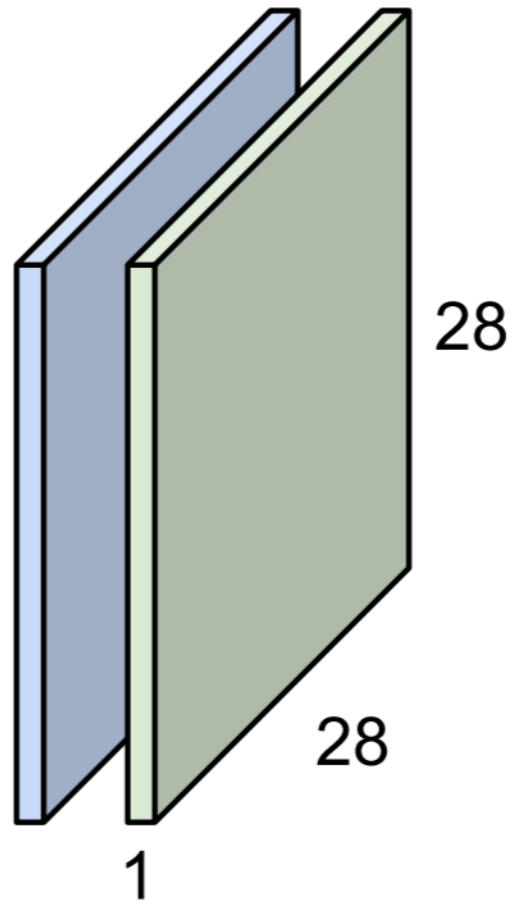


Convolution Layer

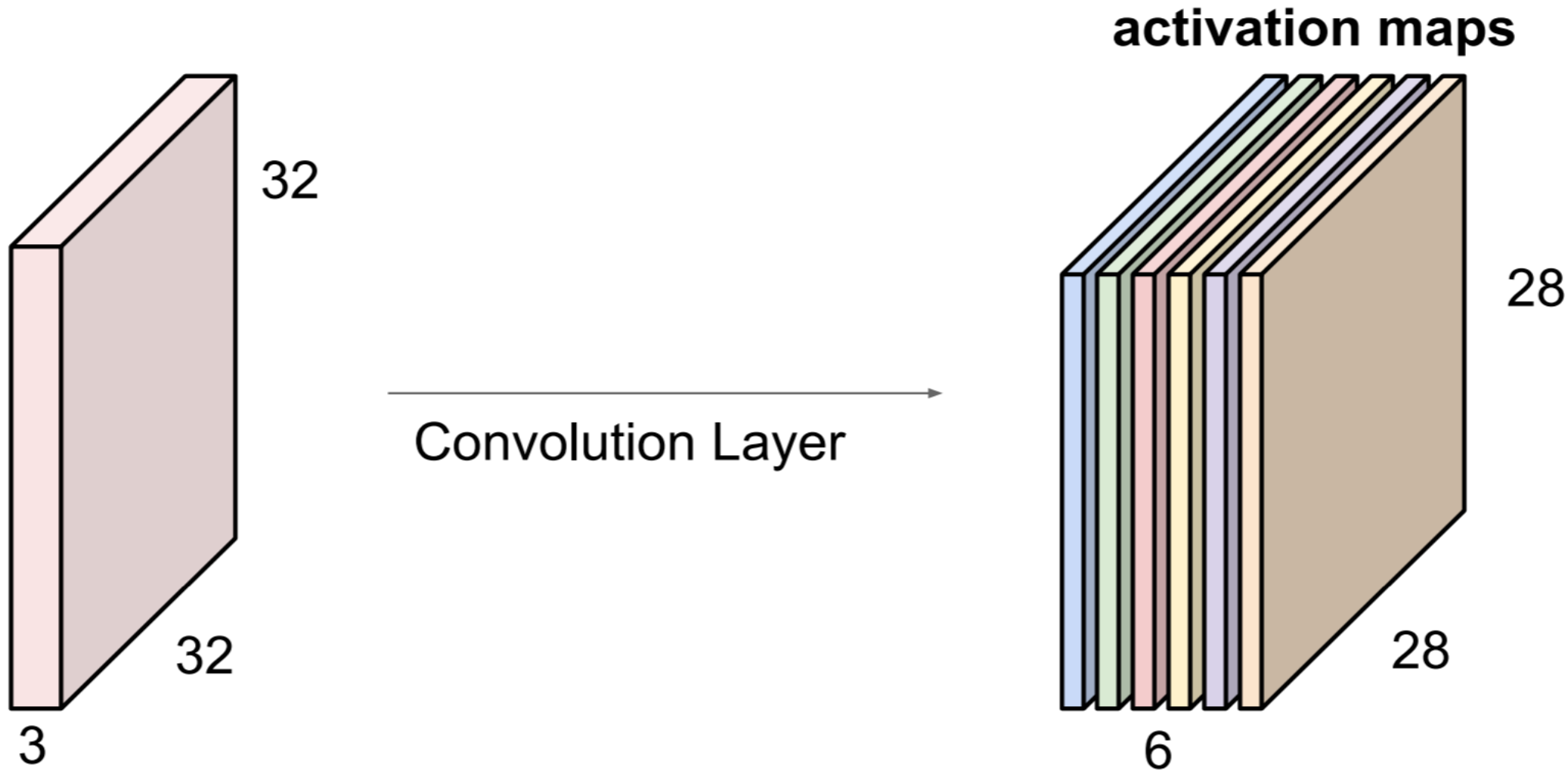
consider a second, green filter



convolve (slide) over all spatial locations



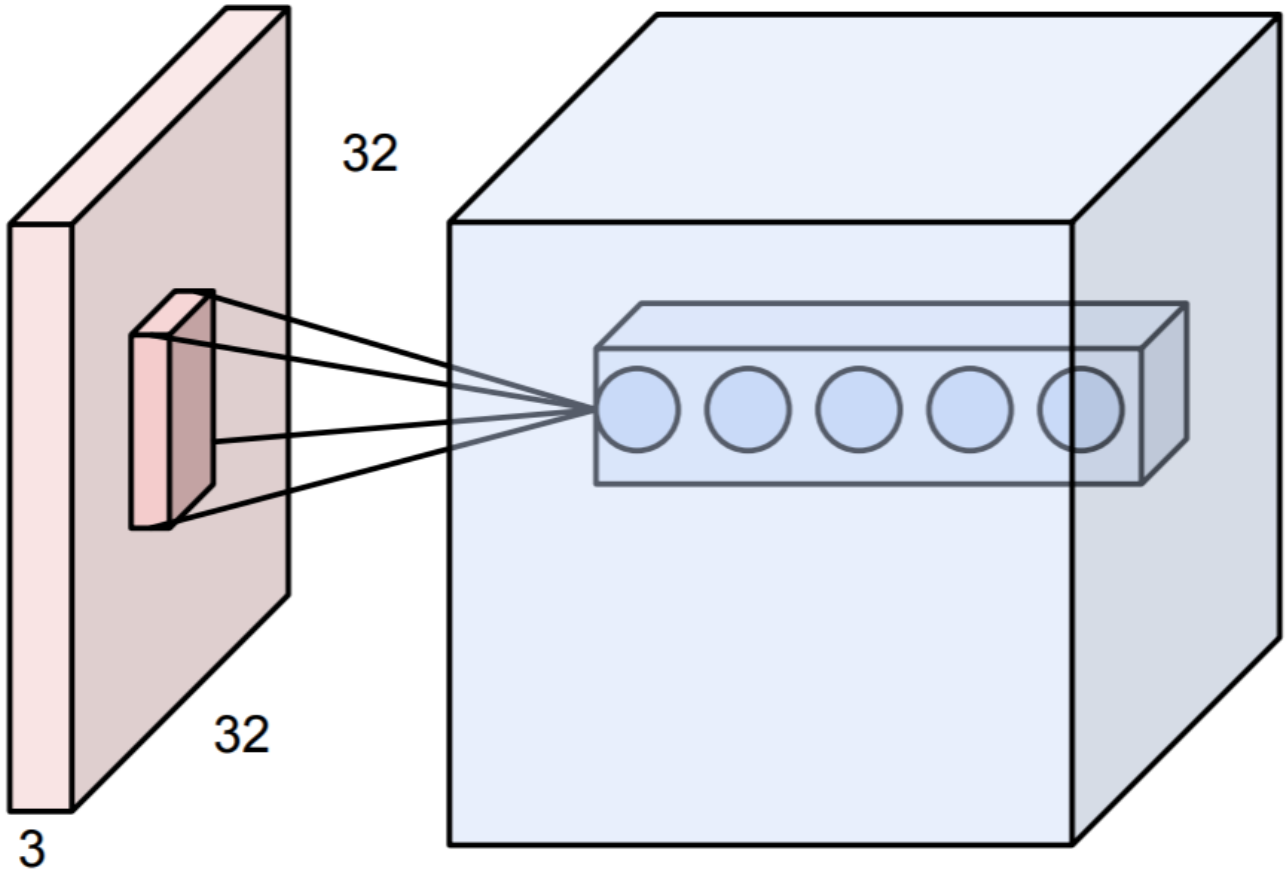
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

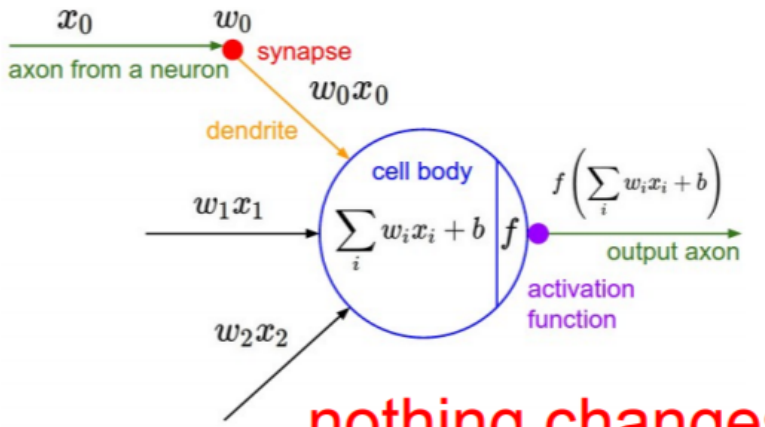


We stack these up to get a "new image" of size 28x28x6!

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity





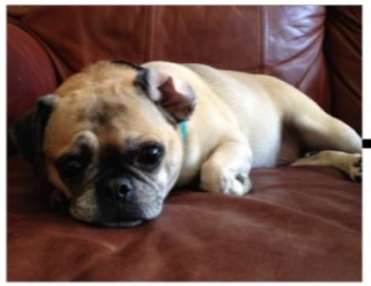
nothing changes

These form a single [1 x 1 x depth] “depth column” in the output volume

Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

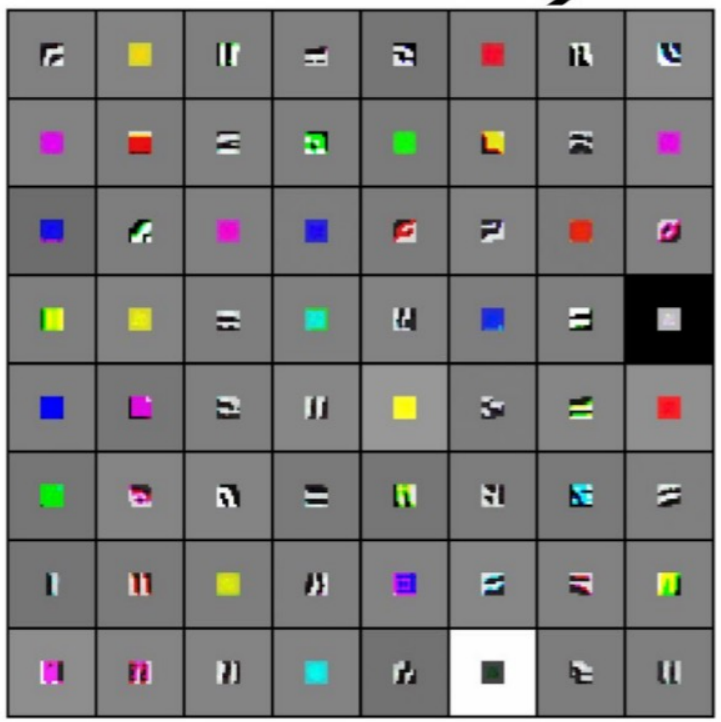


Low-level features

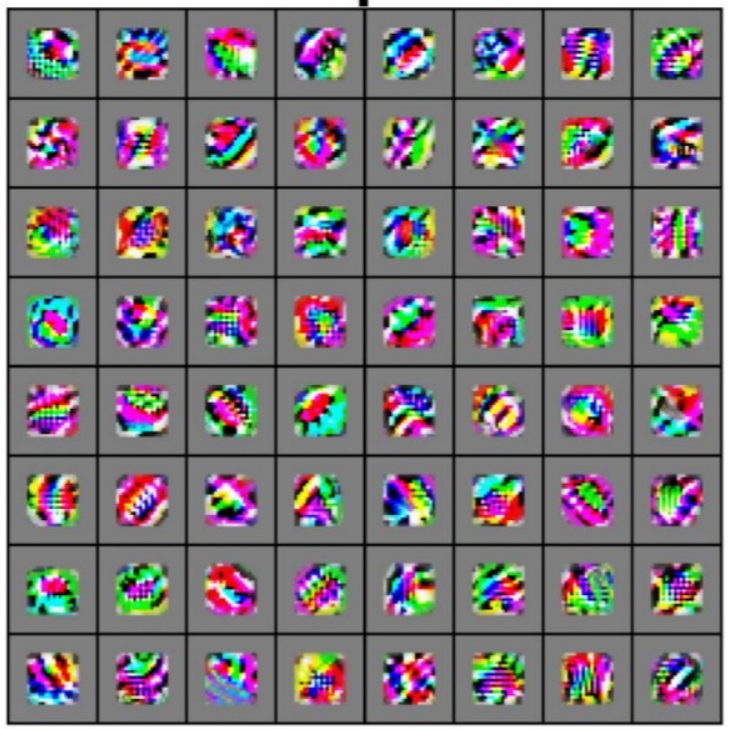
Mid-level features

High-level features

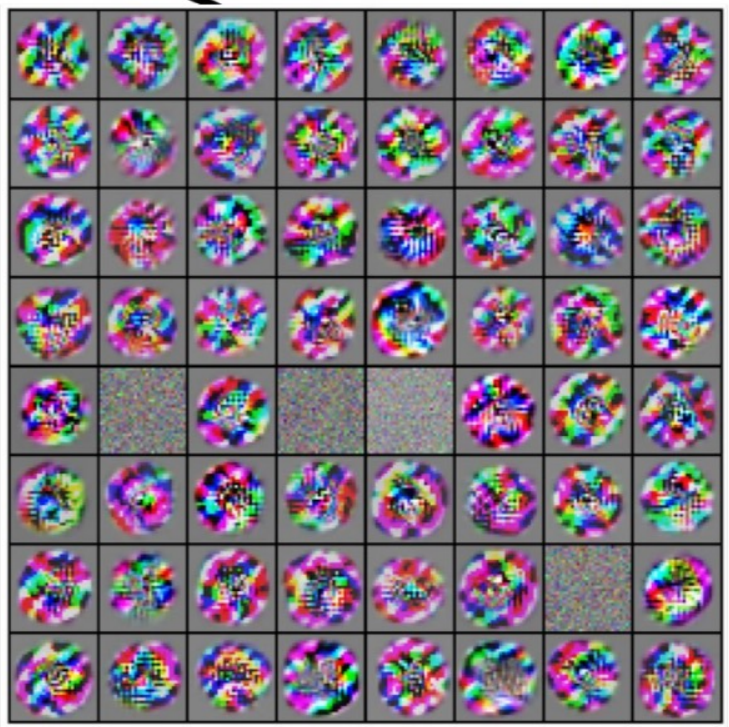
Linearly separable classifier



VGG-16 Conv1_1



VGG-16 Conv3_2

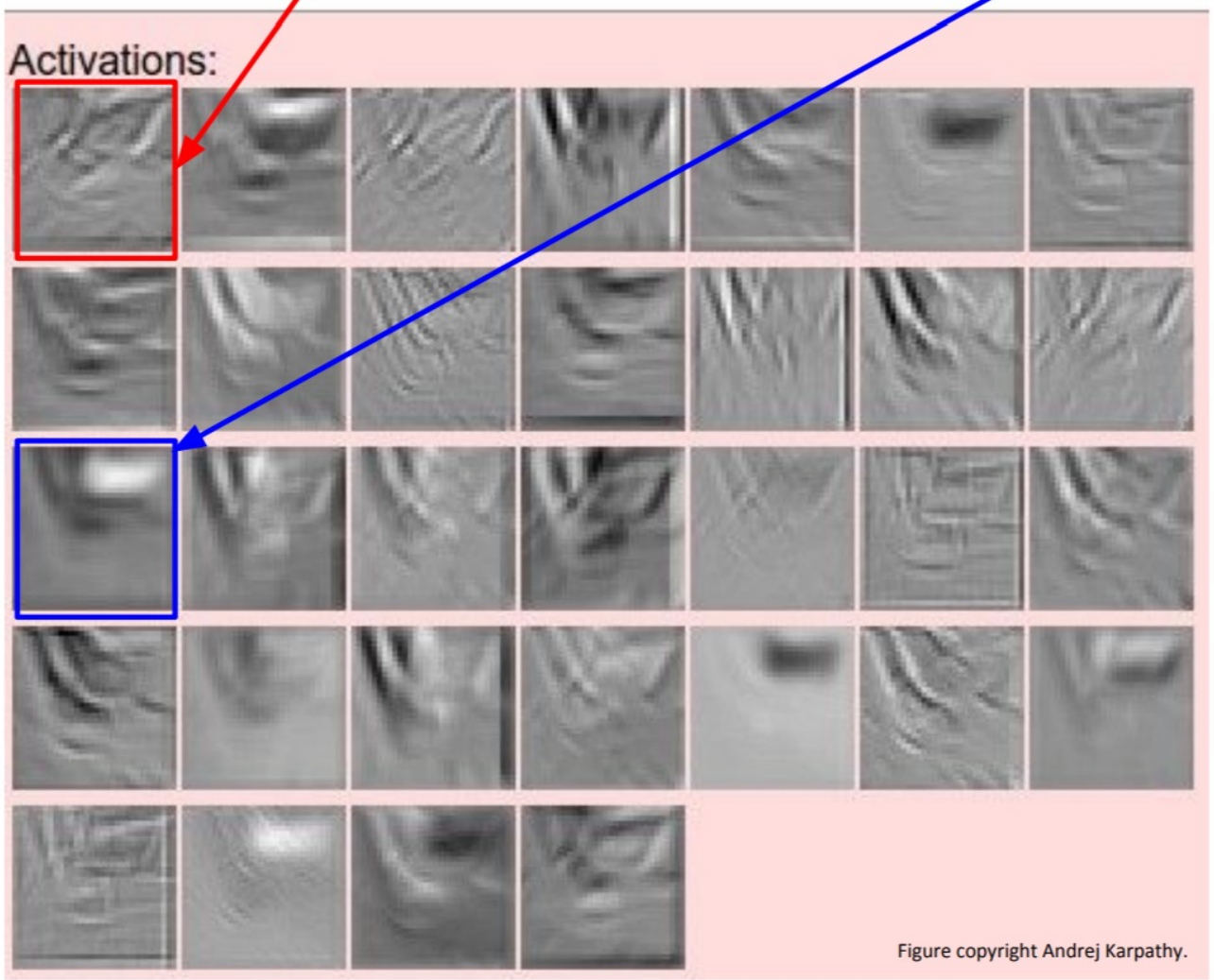


VGG-16 Conv5_3

one filter =>

one activation map

example 5x5 filters
(32 total)

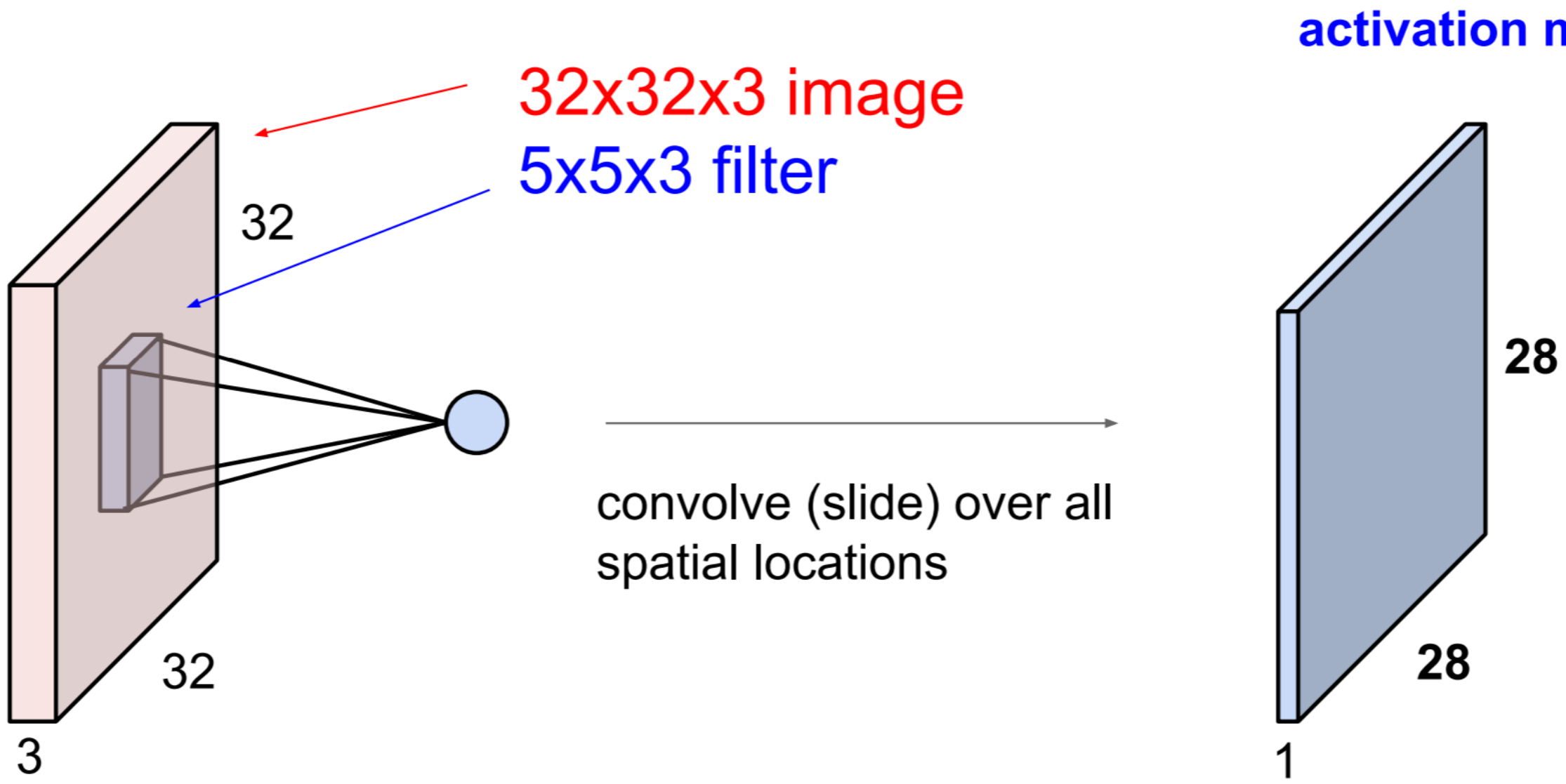


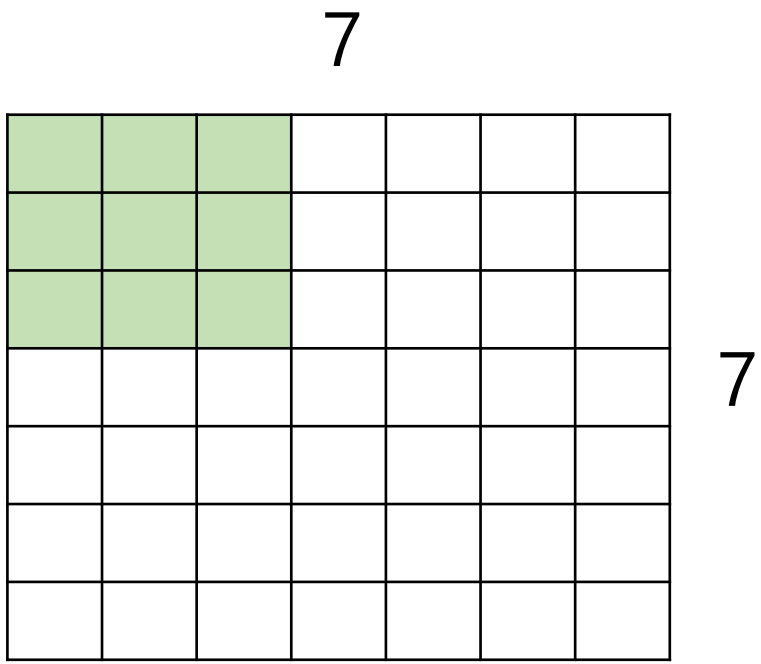
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

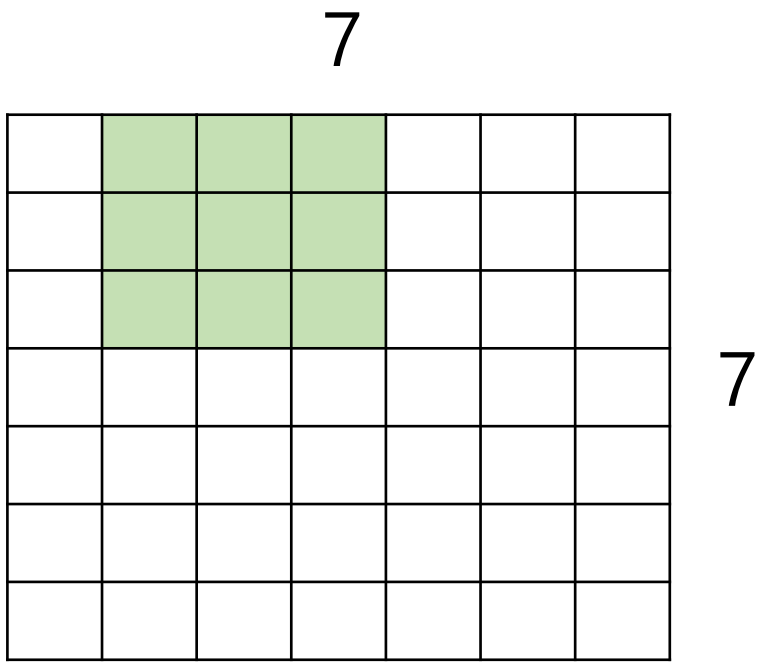
elementwise multiplication and sum of a filter and the signal (image)

A closer look at spatial dimensions:

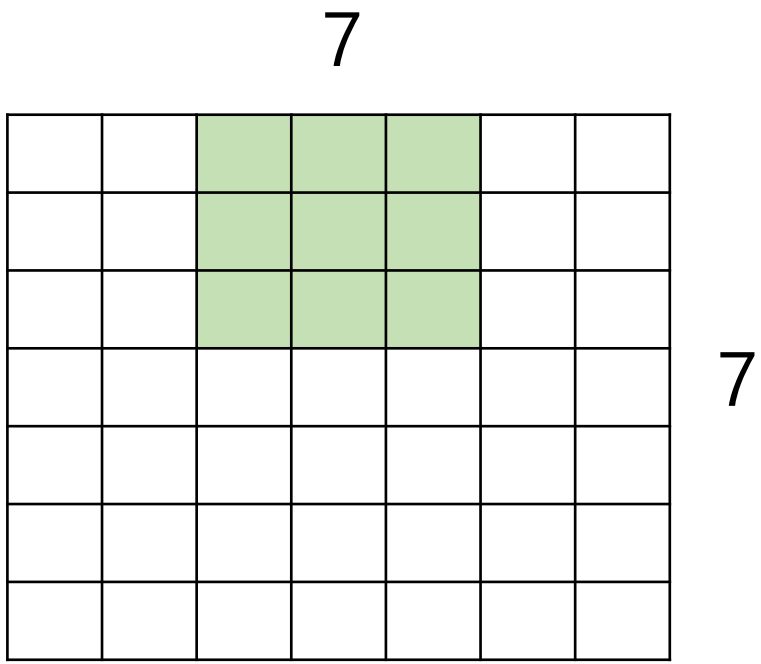




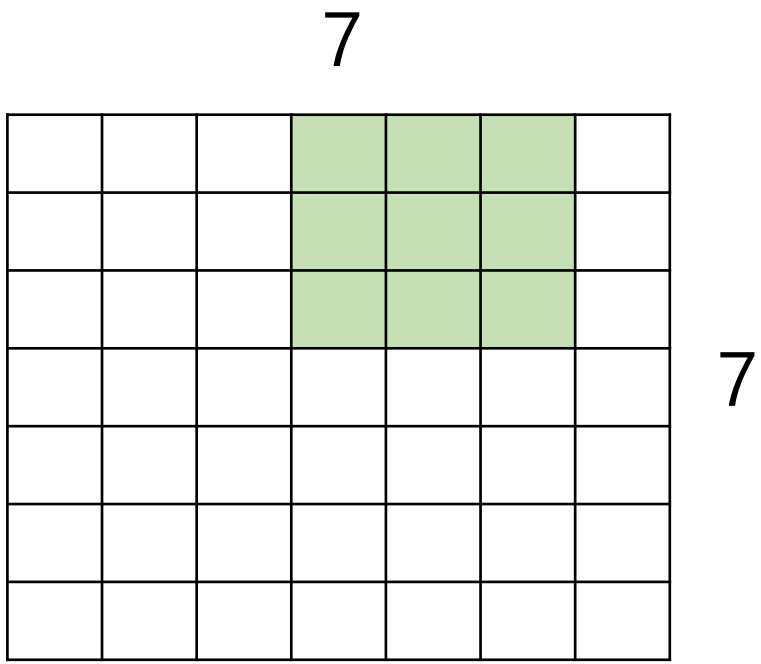
7x7 input (spatially)
assume 3x3 filter



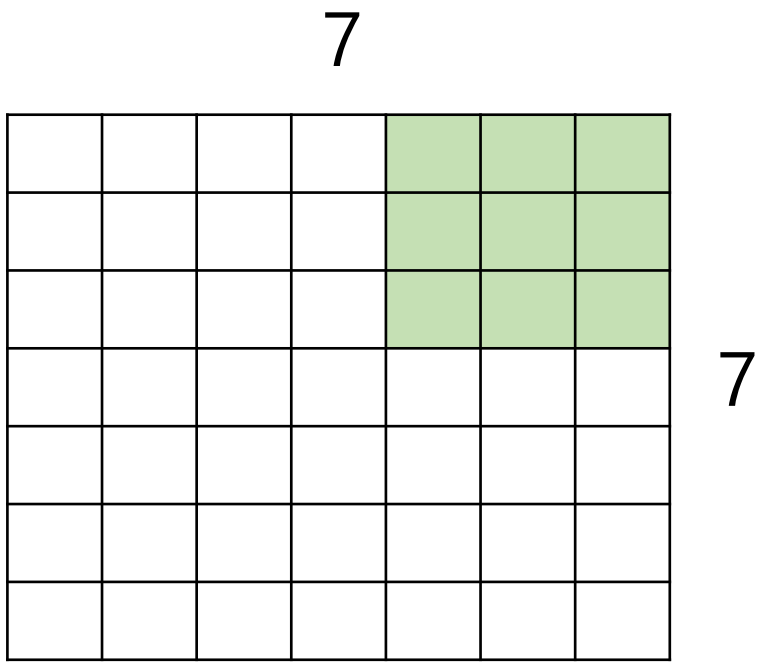
7x7 input (spatially)
assume 3x3 filter



7x7 input (spatially)
assume 3x3 filter

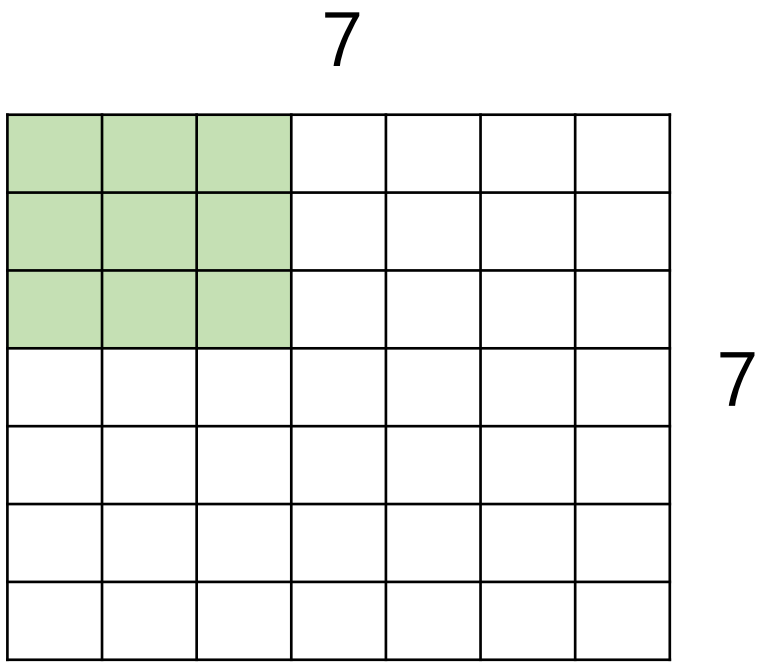


7x7 input (spatially)
assume 3x3 filter

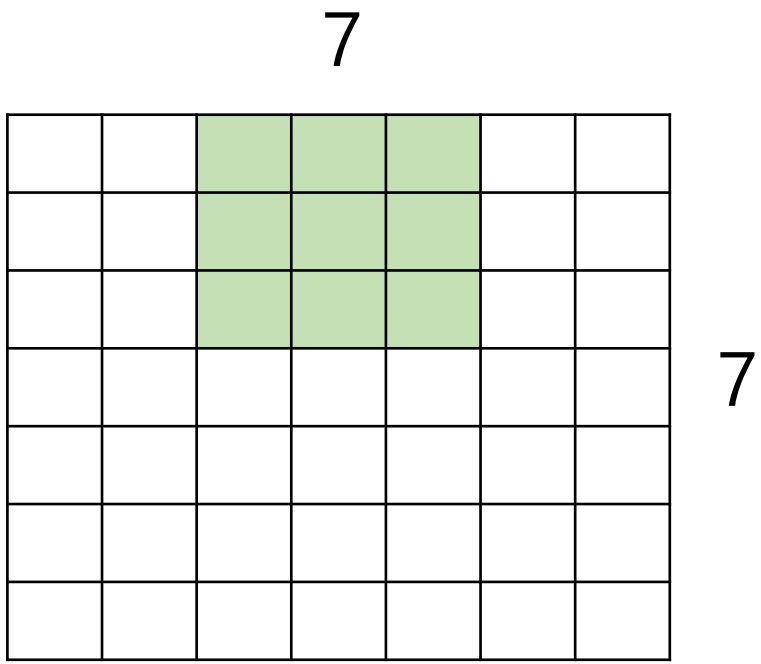


7x7 input (spatially)
assume 3x3 filter

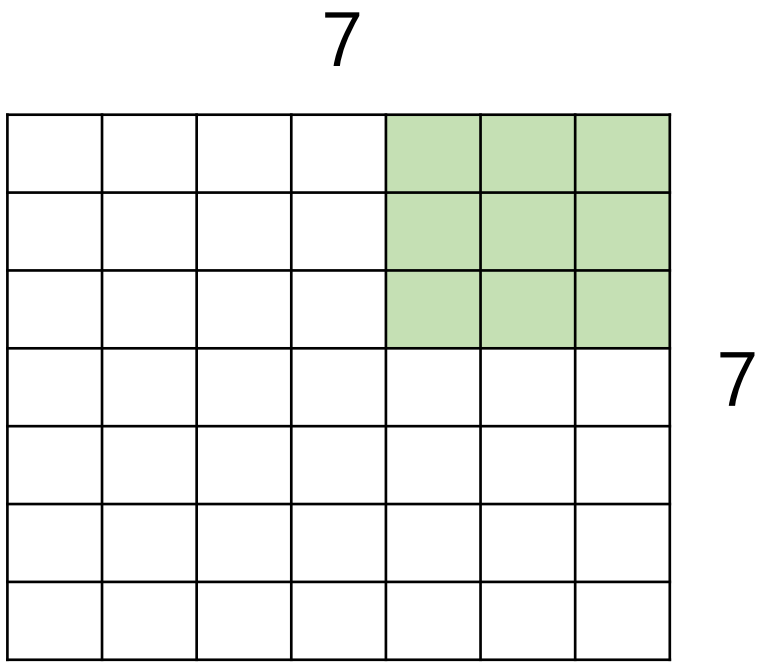
→ **5x5 output**



7x7 input (spatially)
assume 3x3 filter
with **stride 2**

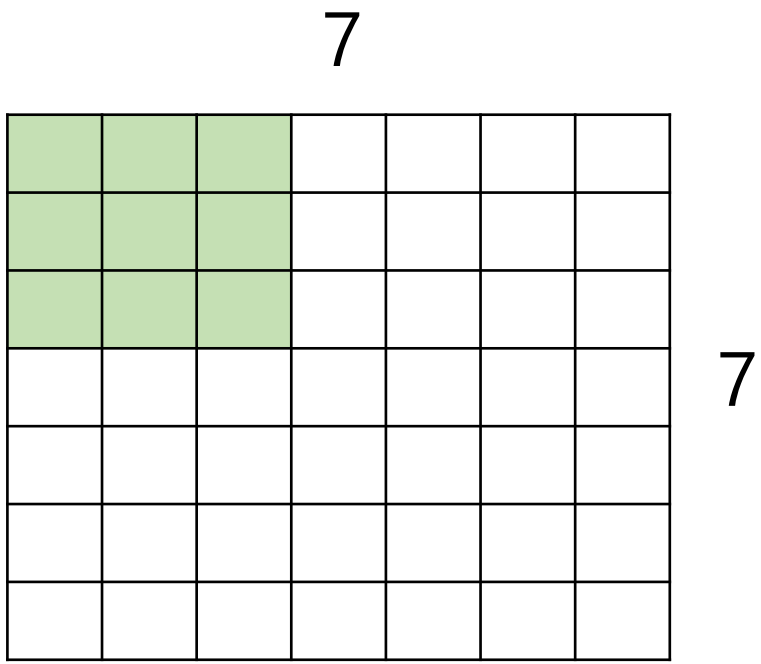


7x7 input (spatially)
assume 3x3 filter
with **stride 2**

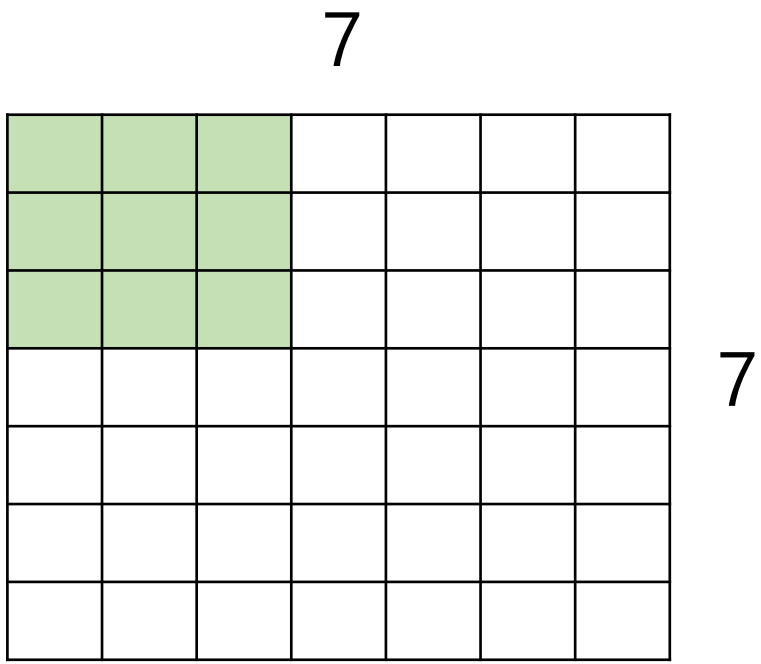


7x7 input (spatially)
assume 3x3 filter
with **stride 2**

→ 3x3 output

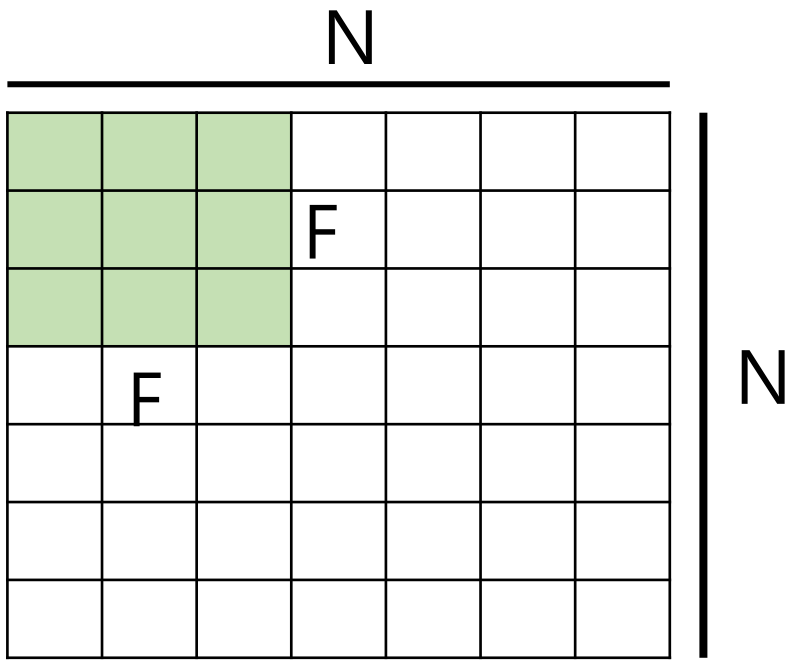


7x7 input (spatially)
assume 3x3 filter
with **stride 3 ?**



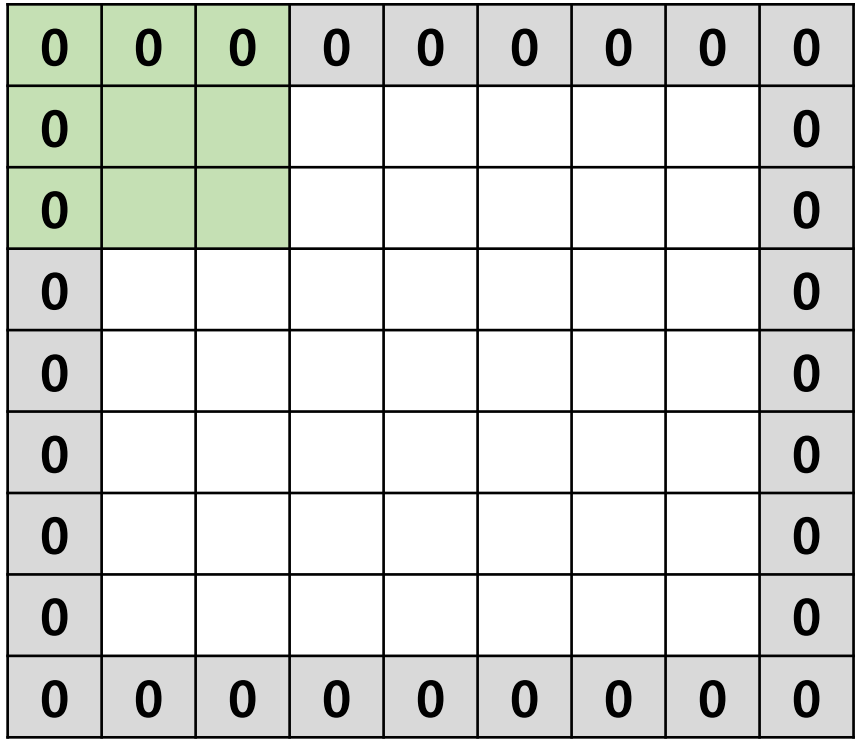
7x7 input (spatially)
assume 3x3 filter
with **stride 3 ?**

→ **Doesn't fit!!**



Output size:
(N - F) / stride + 1

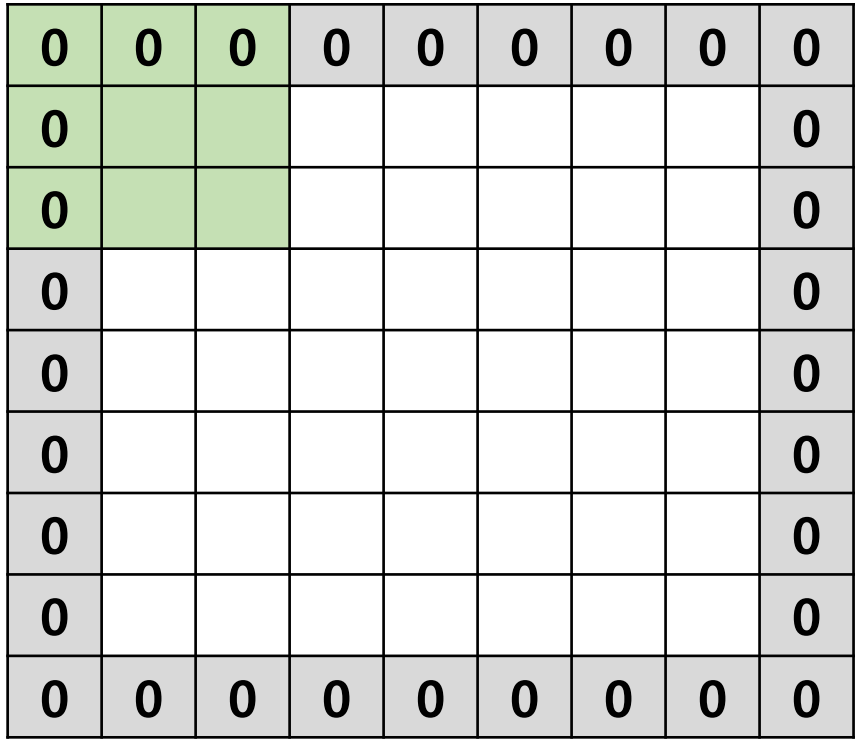
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\



7x7 input (spatially)
assume 3x3 filter with **stride 1** & **pad 1 pixel**

→ What is the output?

(recall:)
(N – F) / stride + 1

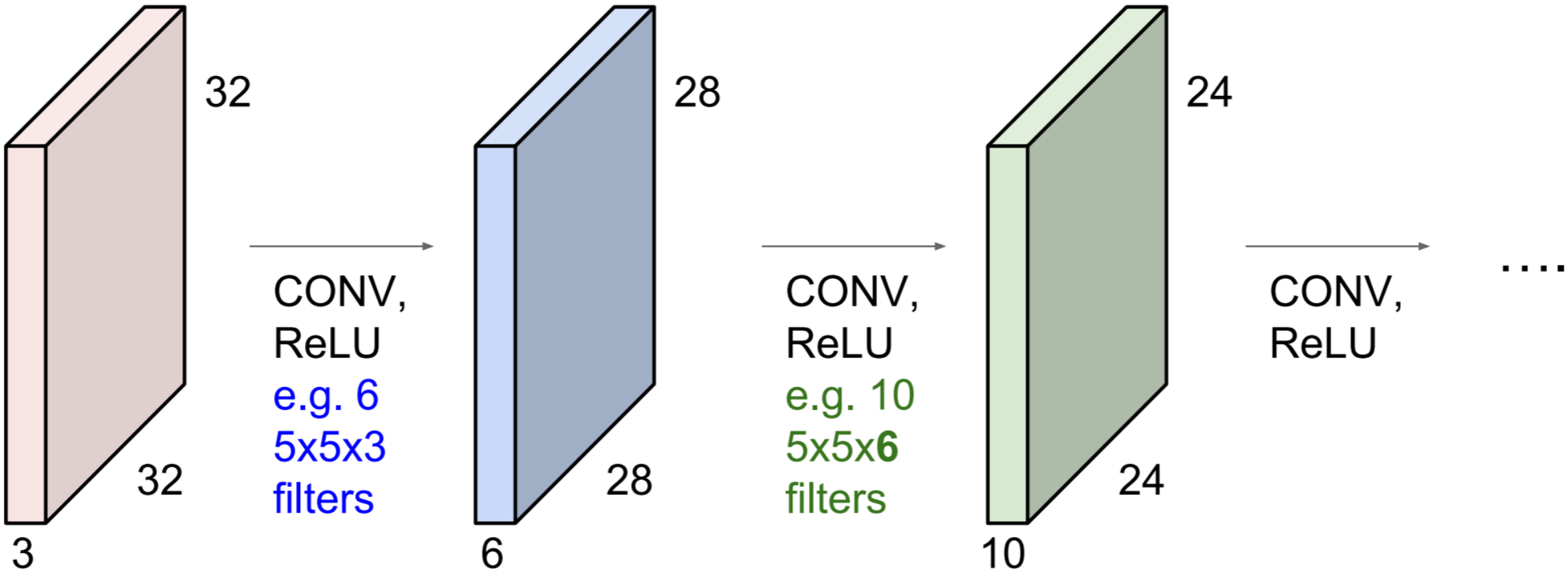


7x7 input (spatially)
assume 3x3 filter with **stride 1** & **pad 1 pixel**

→ What is the output? **7x7 output!**

In general, zero padding with **(F – 1)/2** pixel **preserves** spatial size.
(given filters of size FxF, stride 1)
e.g., F=3 → zero pad with 1
F=5 → zero pad with 2
F=7 → zero pad with 3

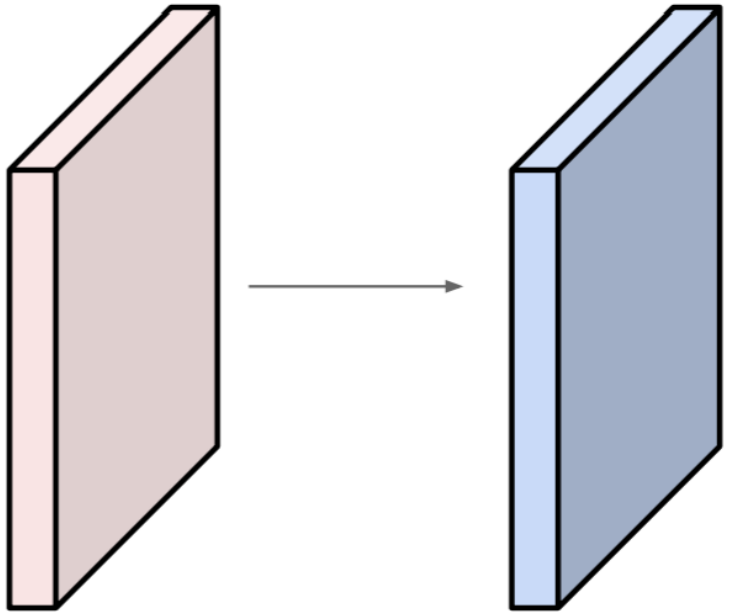
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



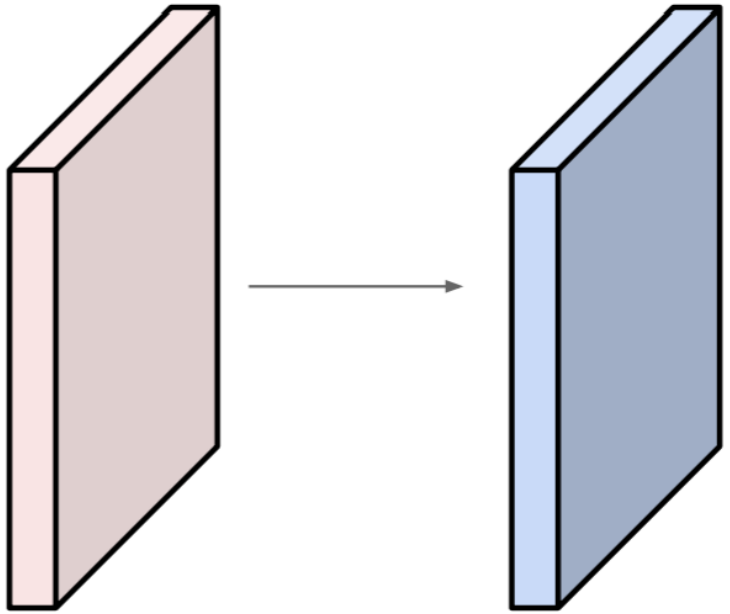
Examples time:

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

Output volume size ??



Examples time:



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

Output volume size :
(32 – 5 + 2*2)/1 + 1 = 32 spatially,

→ 32x32x10

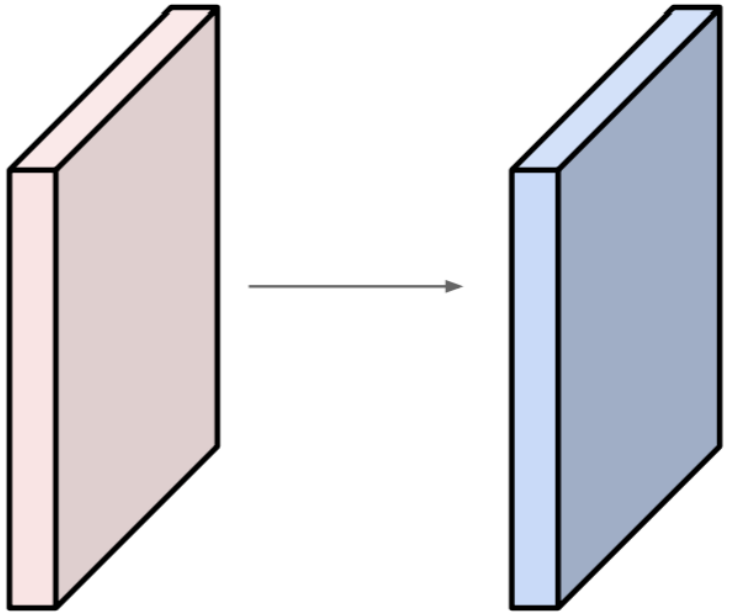
$$W_{output} = \frac{I - K + 2 * P}{S} + 1$$

$I : input$
$K : filter\ size$
$P : pad$

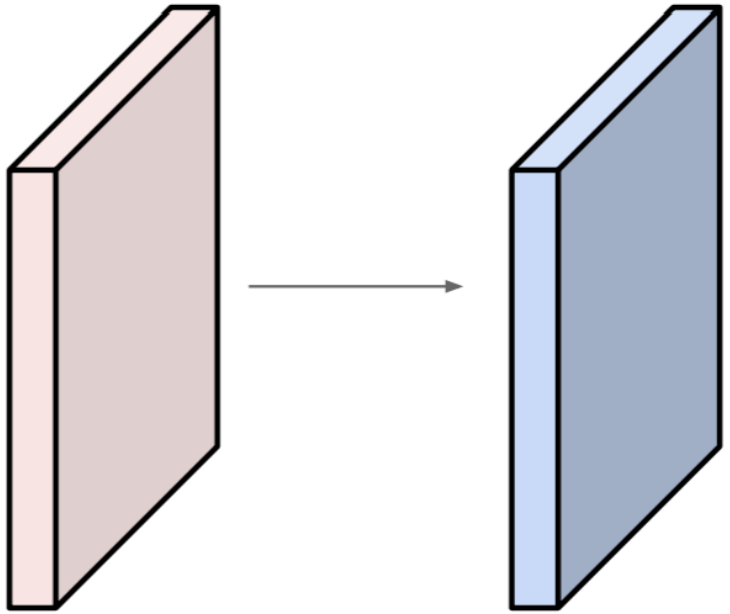
Examples time:

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

Number of parameters in this layer?



Examples time:



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

Number of parameters in this layer?

→ each filter has 5x5x3 + 1 = 76 params (+1 for bias)
→ 76x10 = 760

Summary. To summarize, the Conv Layer:

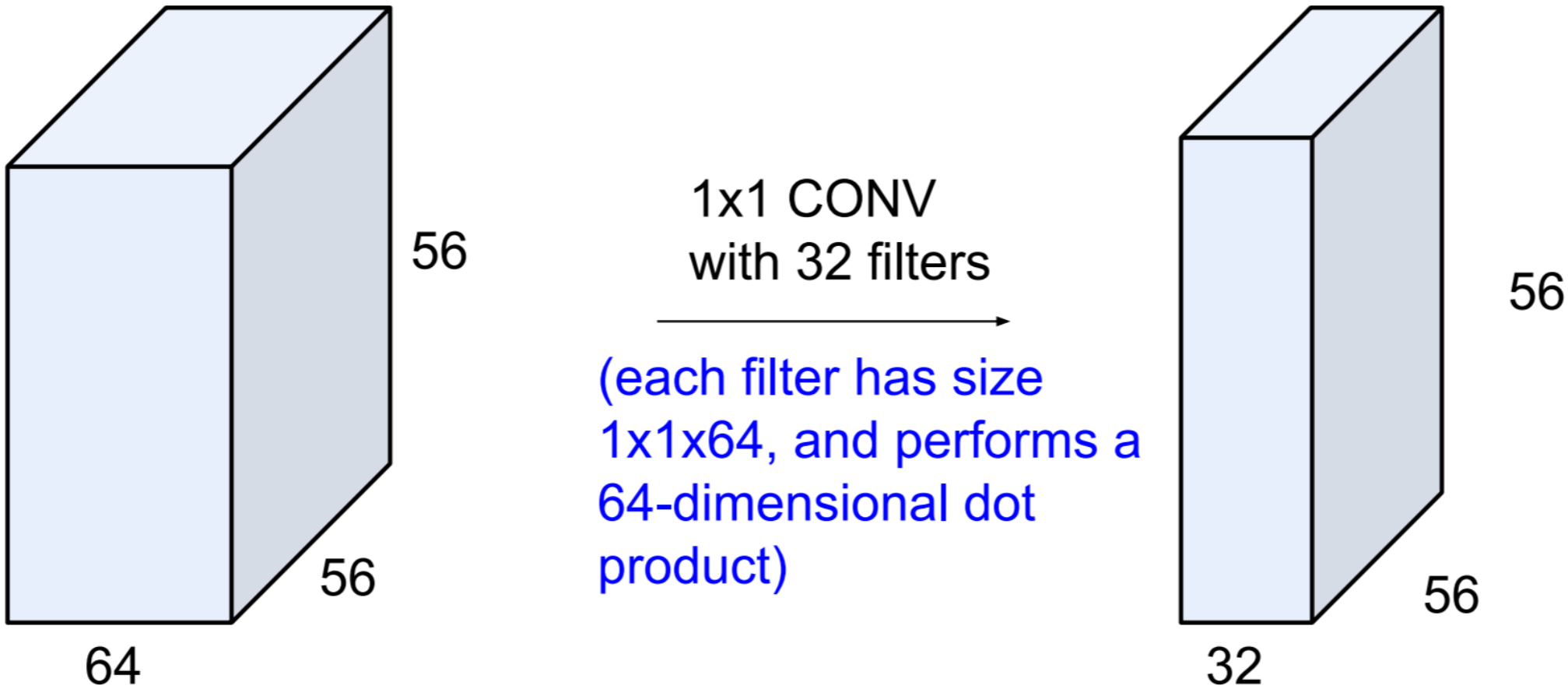
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (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.

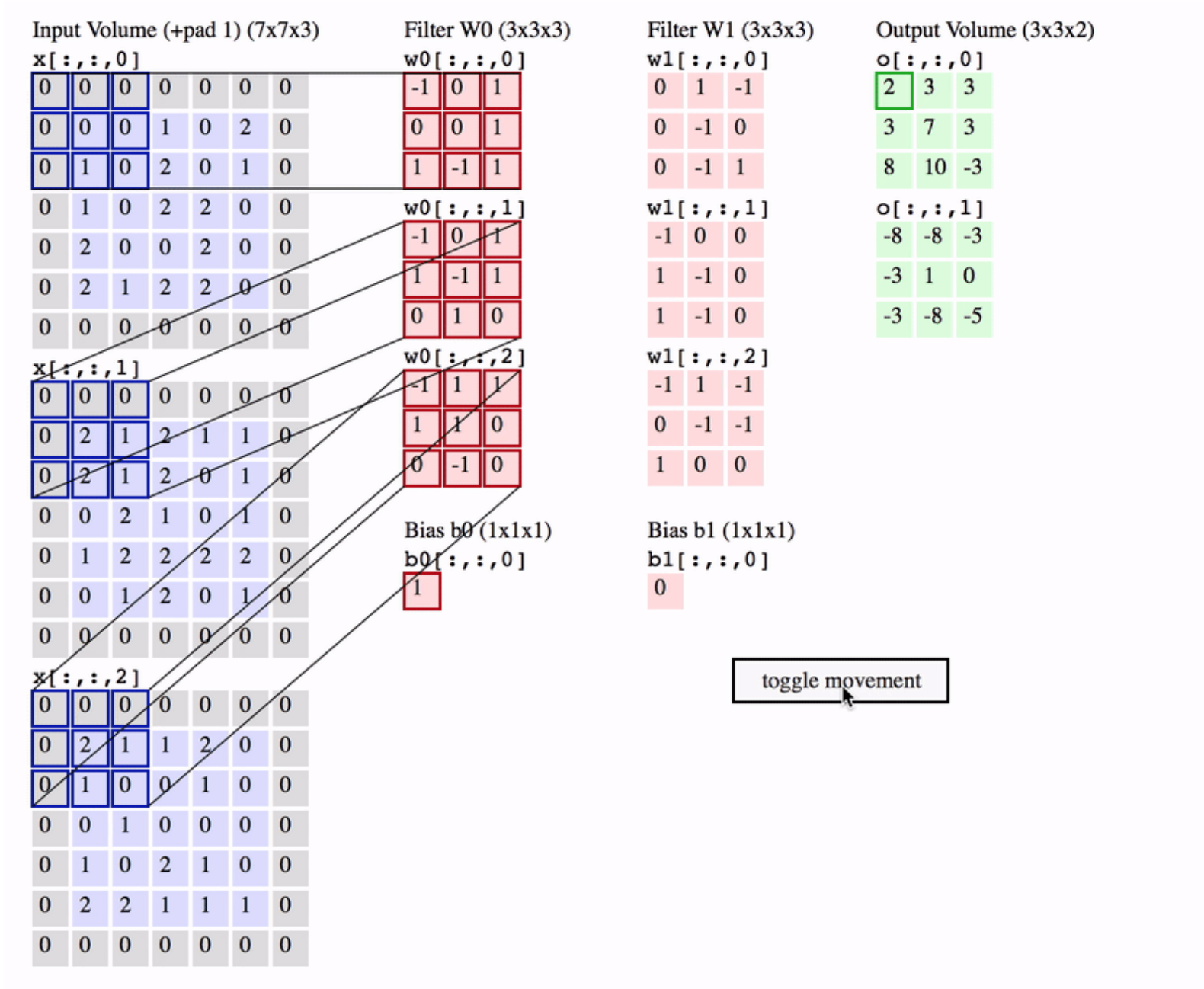
Common settings:

K = (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

(btw, 1x1 convolution layers make perfect sense)





https://cs231n.github.io/convolutional-networks/

Examples CONV layer in PyTorch

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 $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. v
  - $D_2 = K$

• NOTE

Depending of the size of your kernel, several (of the last) columns of the input might be lost, because it is a valid cross-correlation, and not a full cross-correlation. It is up to the user to add proper padding.

• NOTE

When *groups == in_channels* and *out_channels == K * in_channels*, where *K* is a positive integer, this operation is also termed in literature as depthwise convolution.

In other words, for an input of size $(N, C_{in}, H_{in}, W_{in})$, a depthwise convolution with a depthwise multiplier *K*, can be constructed by arguments $(in\_channels = C_{in}, out\_channels = C_{in} \times K, ..., groups = C_{in})$.

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

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

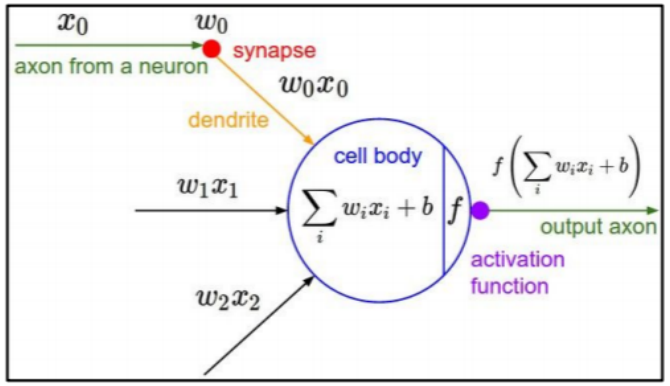
In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

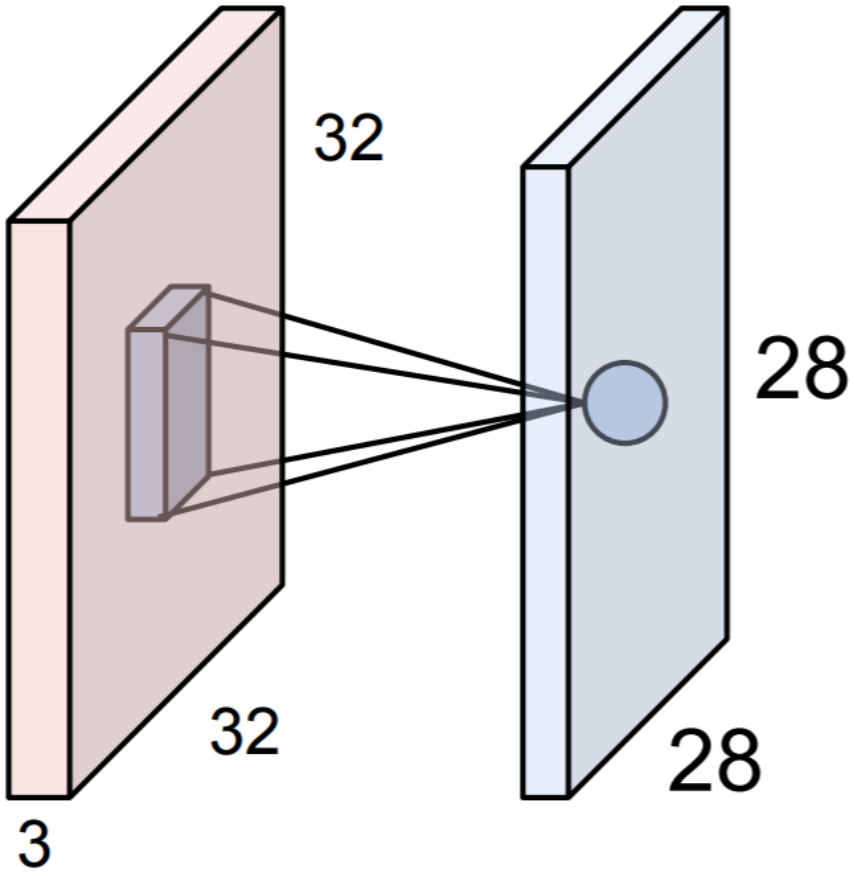
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

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

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{out\_channels}{in\_channels} \right\rfloor$.

The brain/neuron view of CONV Layer



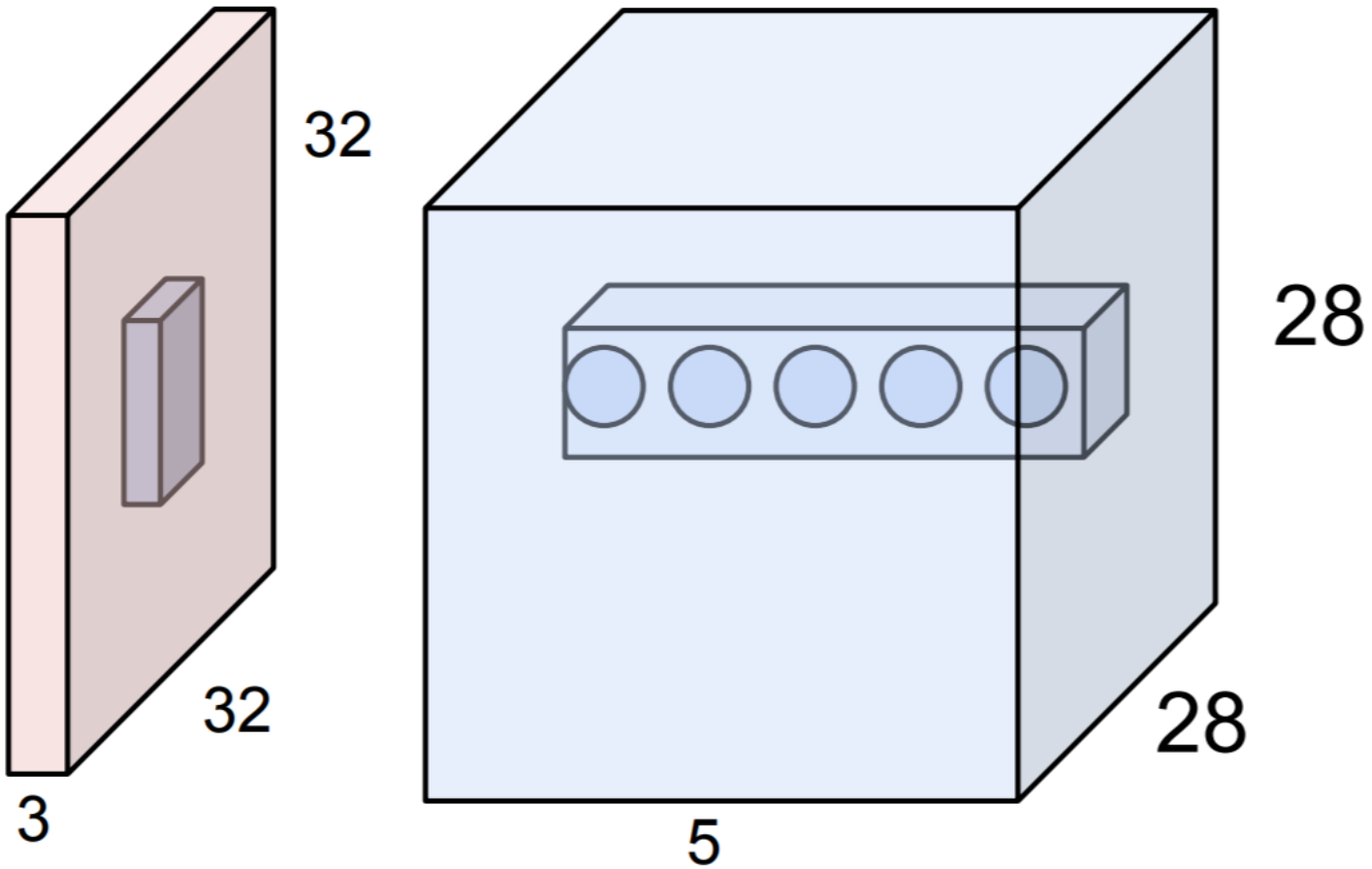


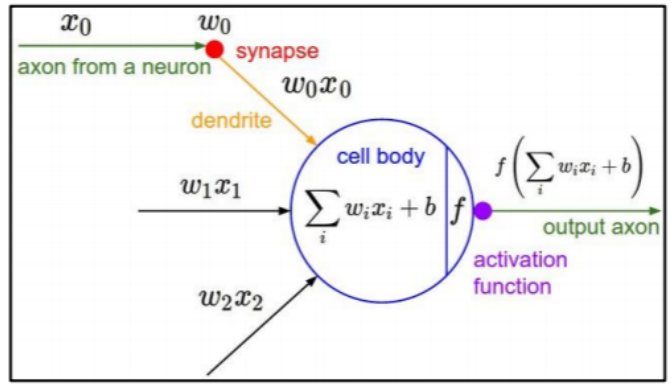
An activation map is a 28x28 sheet of neuron outputs:

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





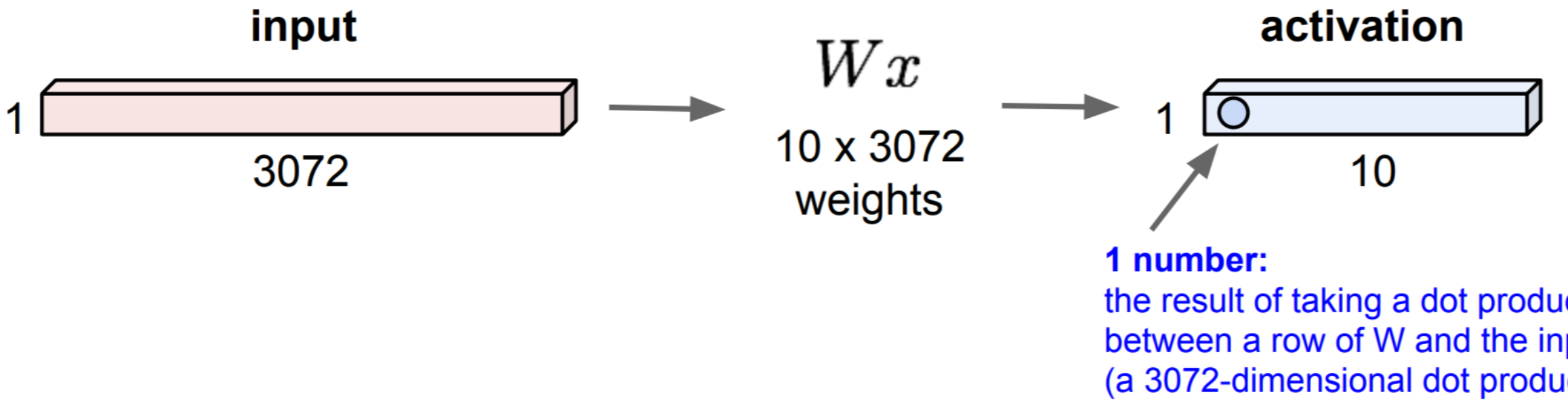
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

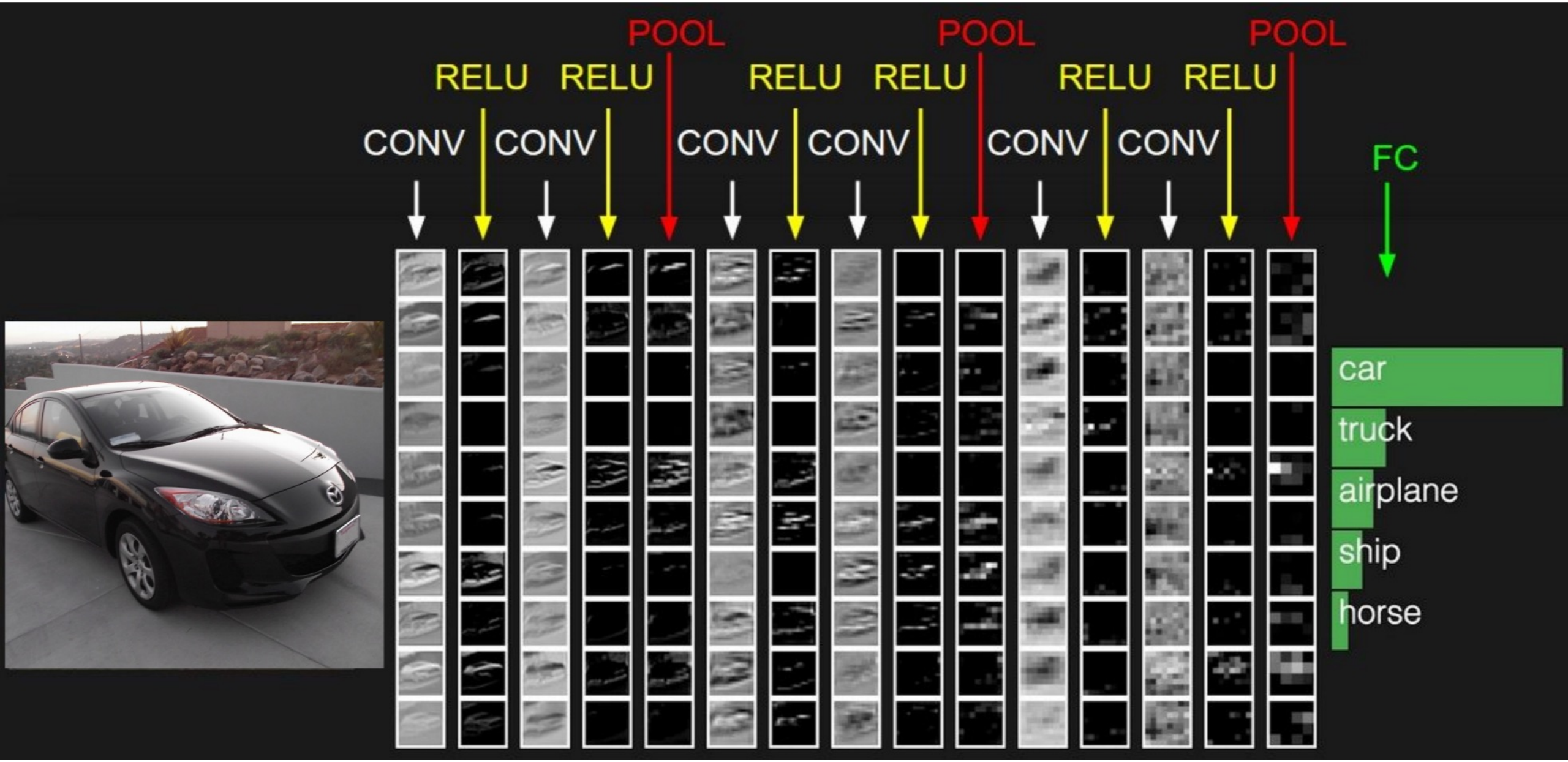
There will be 5 different neurons all looking at the same region in the input volume

Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

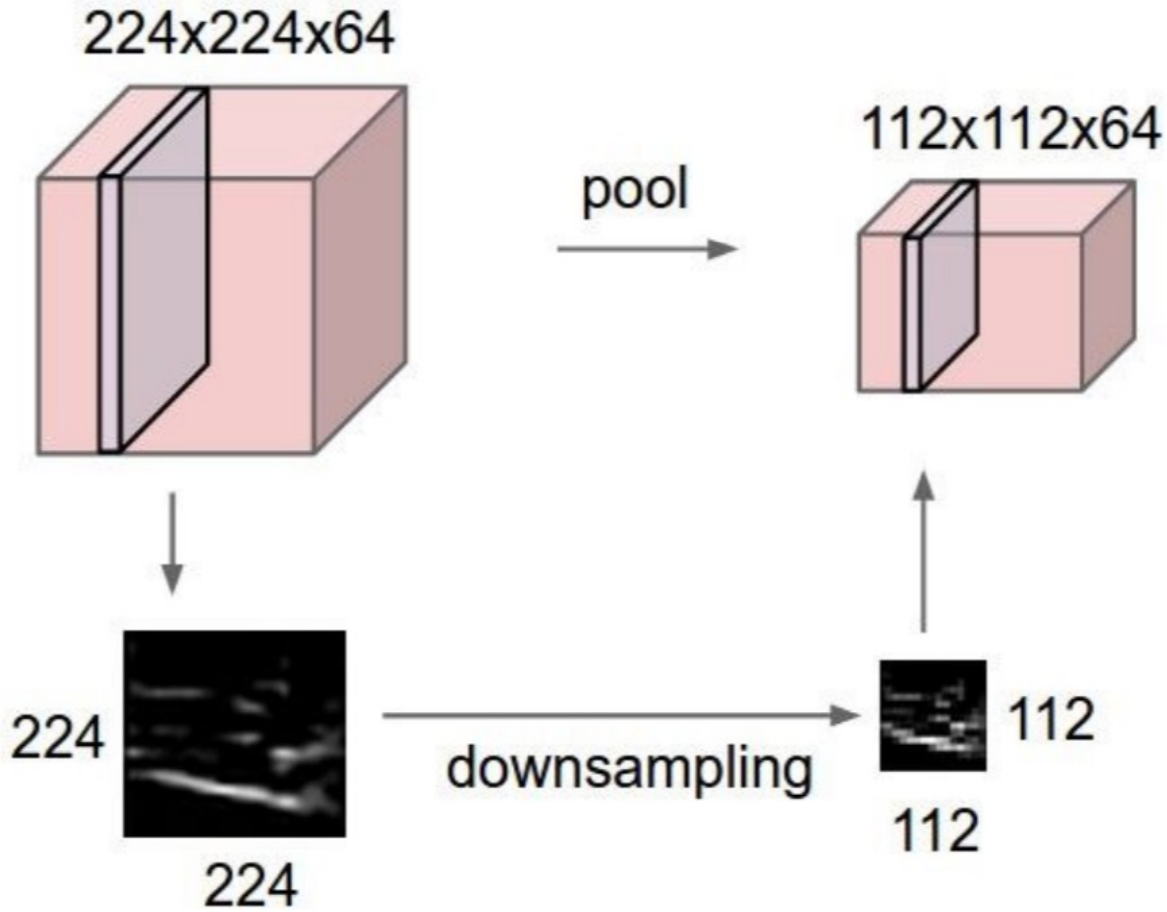
Each neuron looks at the full input volume



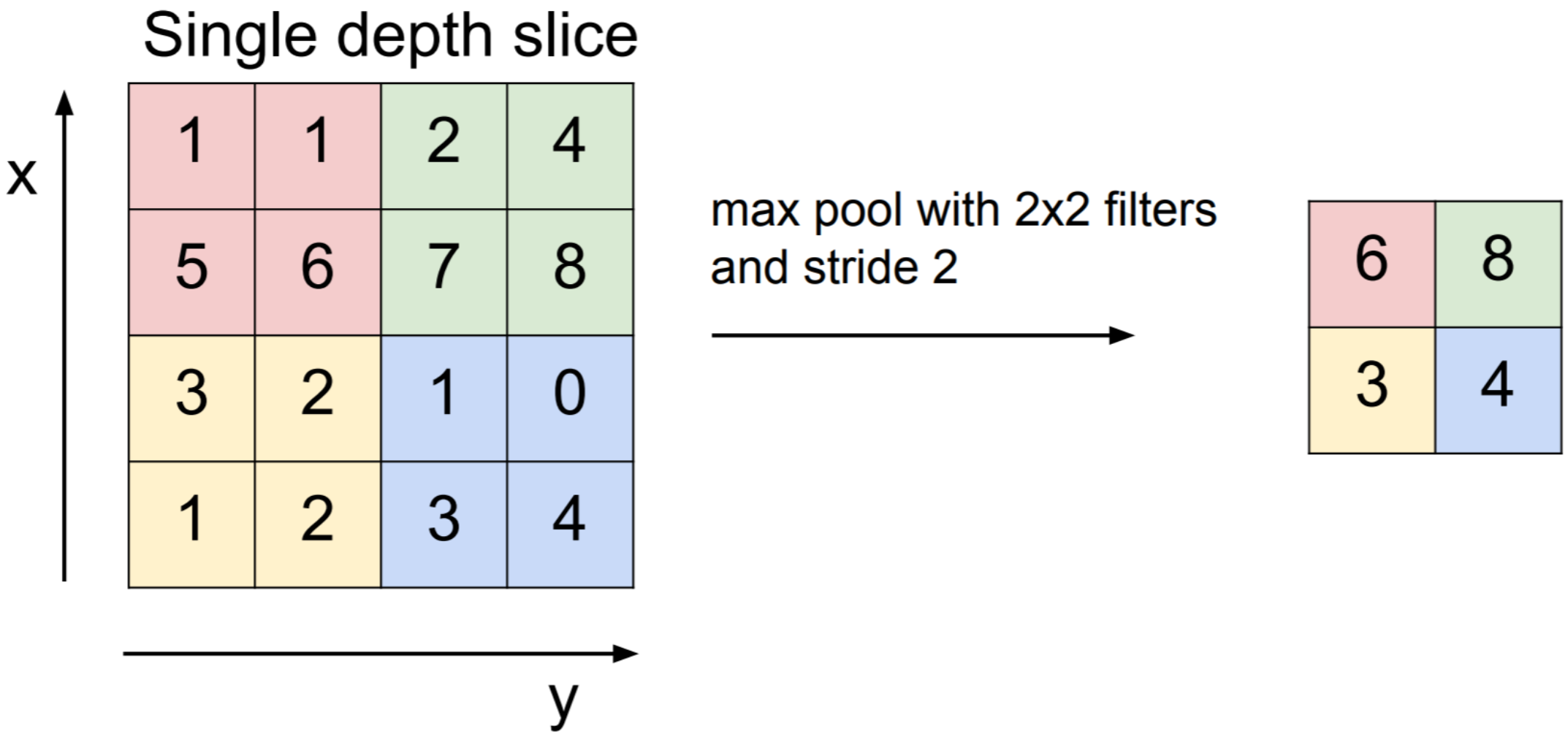

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Pooling layer

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



MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

- 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

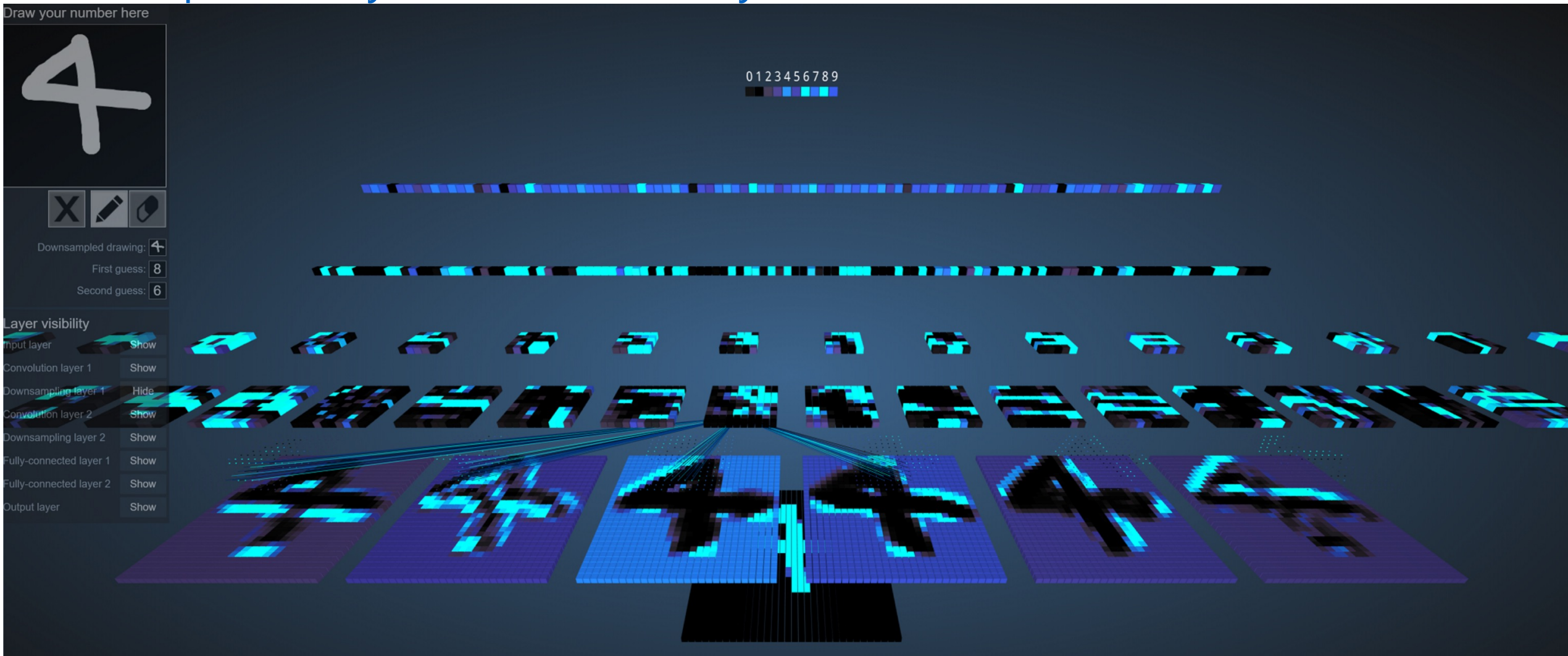
Common settings:

F = 2, S = 2

F = 3, S = 2

Visualization

https://scs.ryerson.ca/~aharley/vis/conv/



Visualization

https://poloclub.github.io/cnn-explainer/

Visualization

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

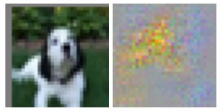
By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

Network Visualization

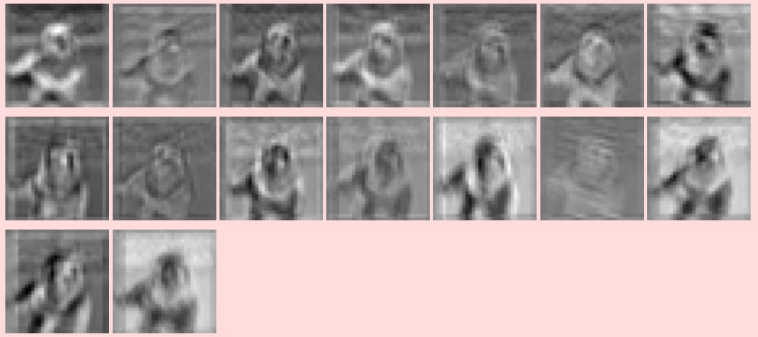
input (32x32x3)
max activation: 0.5, min: -0.48824
max gradient: 0.04221, min: -0.06669

Activations:

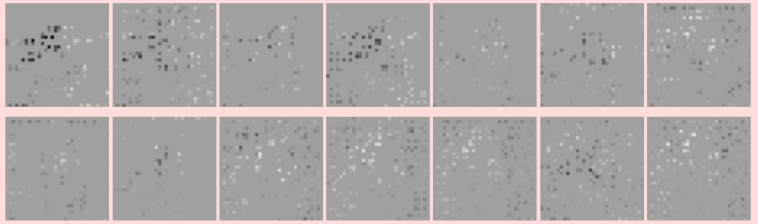


conv (32x32x16)
filter size 5x5x3, stride 1
max activation: 1.12859, min: -1.01916
max gradient: 0.0212, min: -0.03665
parameters: 16x5x5x3+16 = 1216

Activations:

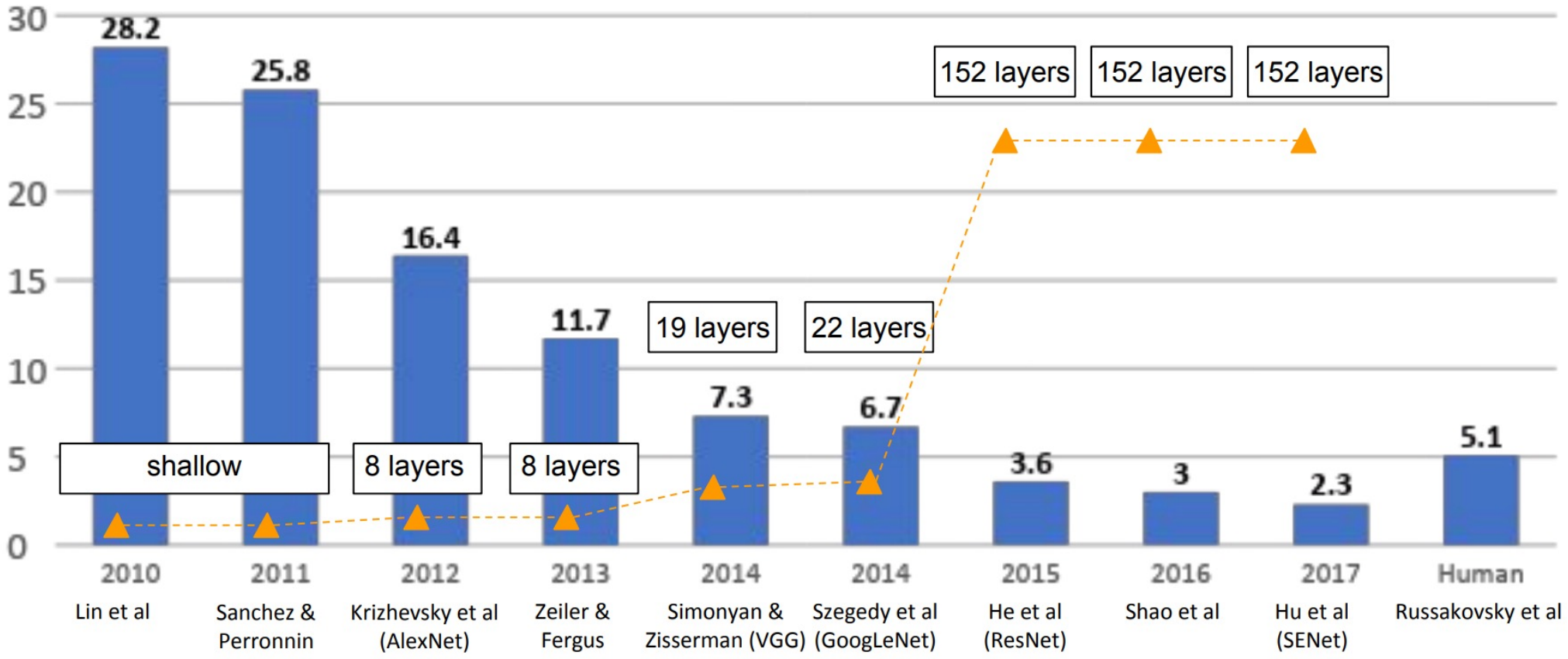


Activation Gradients:



Takeaway

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



References

- http://cs231n.github.io/convolutional-networks/
- http://cs231n.stanford.edu/
- https://deeplearningzerotoall.github.io/season2/lec_pytorch.html