

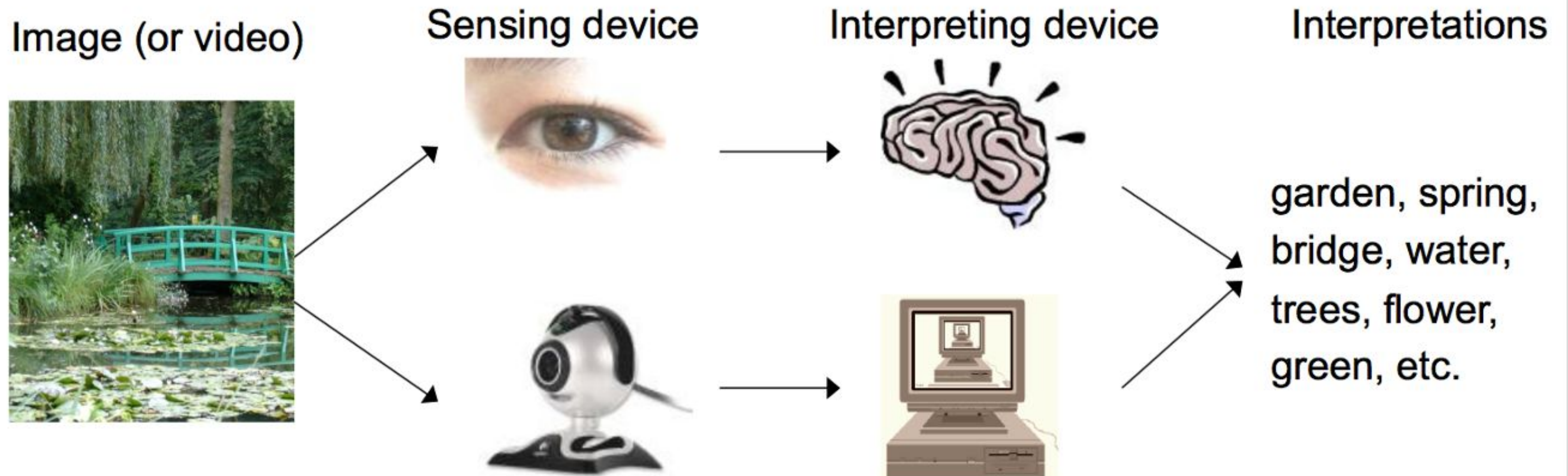
Introduction au Machine Learning

Réseaux de neurones convolutifs

Emmanuel Dellandrea - emmanuel.dellandrea@ec-lyon.fr

Where it all began: Computer Vision

Computer Vision: How to endow computers with the ability to understand digital images and videos ?



Some tasks in Computer Vision

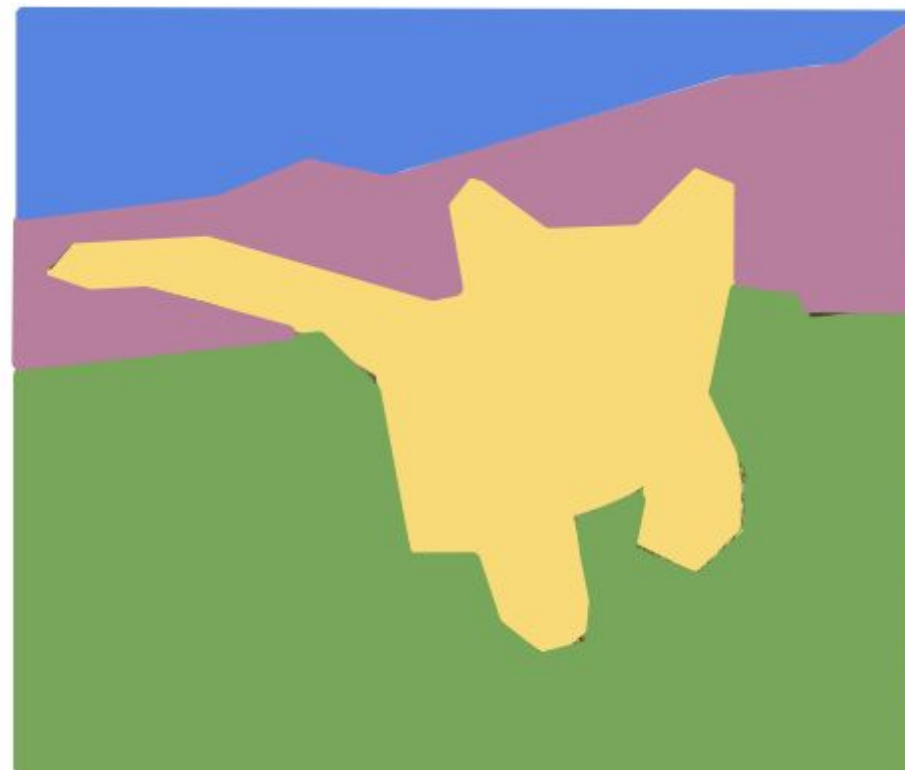
Classification



CAT

No spatial extent

Semantic Segmentation



GRASS, CAT, TREE,
SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Object

Instance Segmentation



DOG, DOG, CAT

[This image is CC0 public domain](#)

A core task in Computer Vision: Image classification

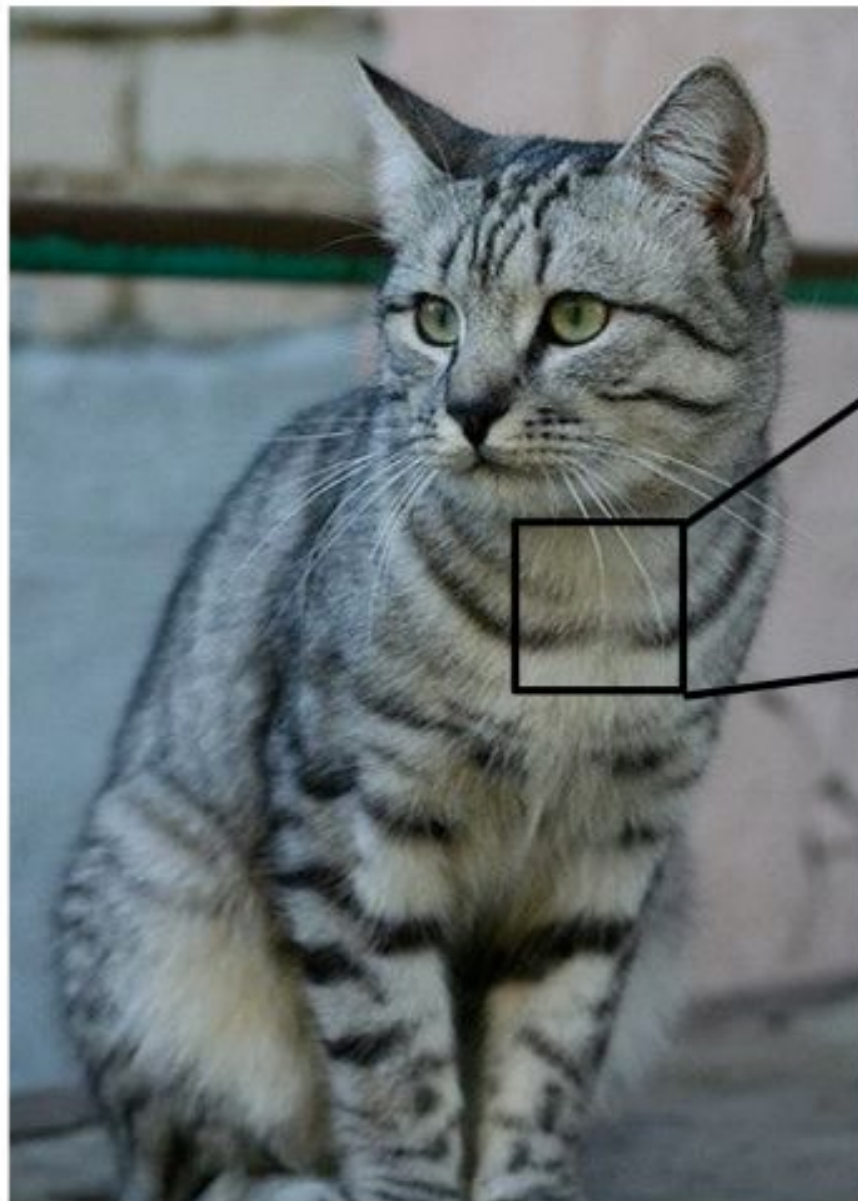


This image by Nikita is
licensed under [CC-BY 2.0](#)

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

The Problem: Semantic Gap



This image by Nikita is
licensed under [CC-BY 2.0](#)

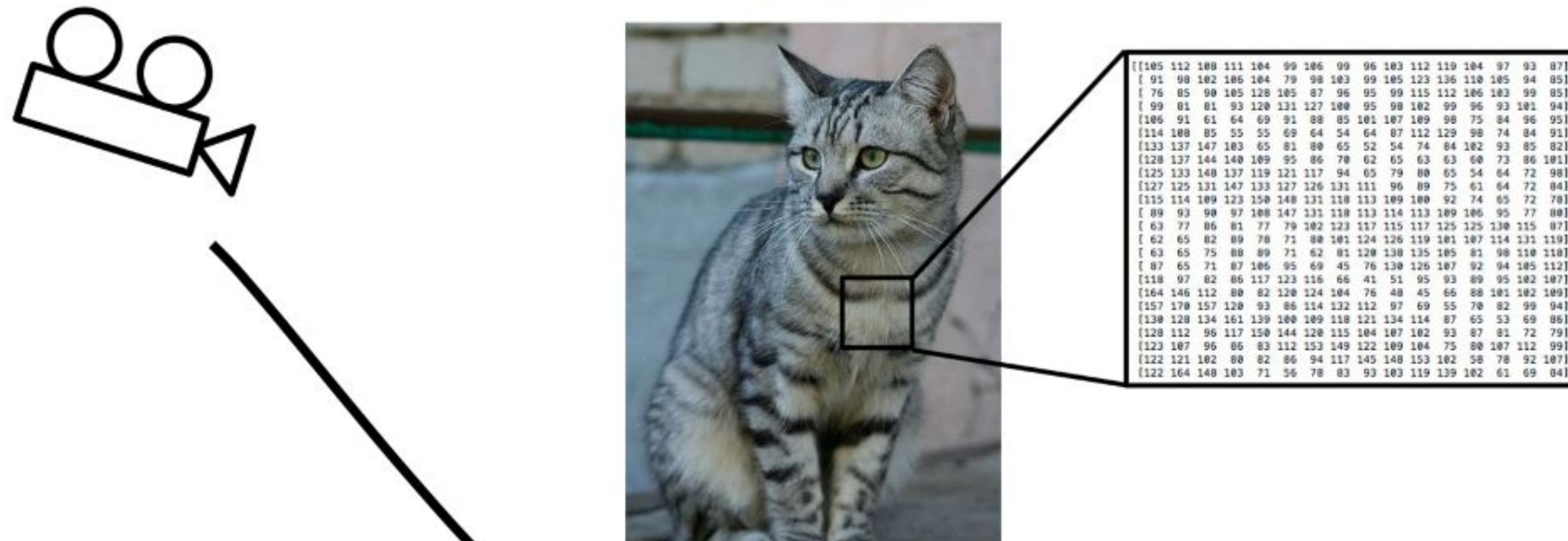
```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]  
[ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]  
[ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]  
[ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]  
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]  
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]  
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]  
[128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]  
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]  
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]  
[115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]  
[ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]  
[ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]  
[ 62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]  
[ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]  
[ 87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]  
[118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]  
[164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]  
[157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]  
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]  
[128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]  
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]  
[122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]  
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]]
```

What the computer sees

An image is just a big grid of
numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint variation



All pixels change when
the camera moves!

Challenges: Illumination



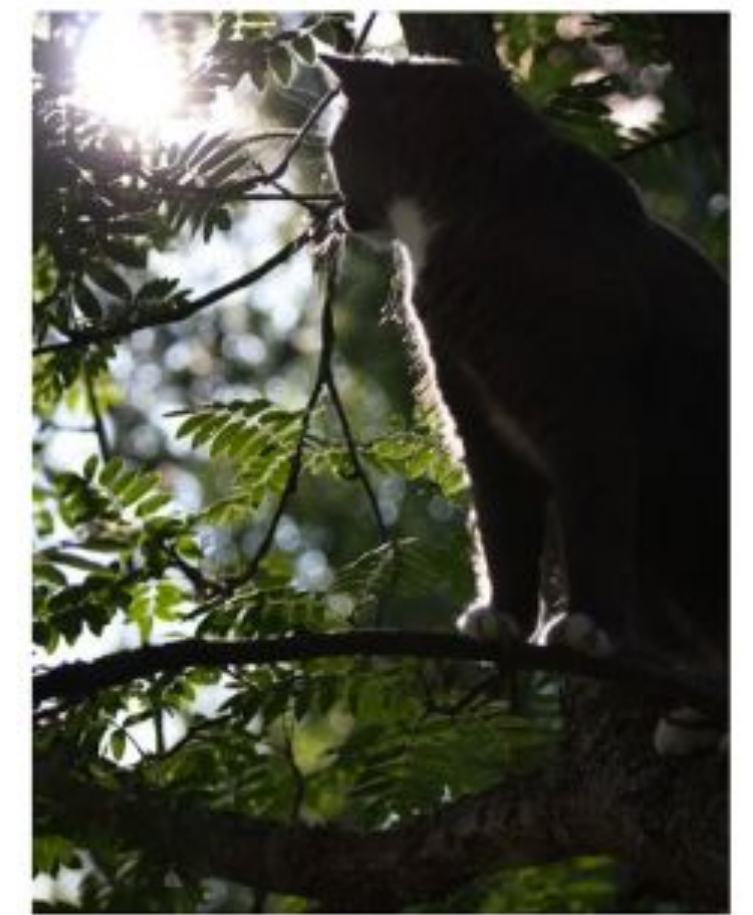
[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

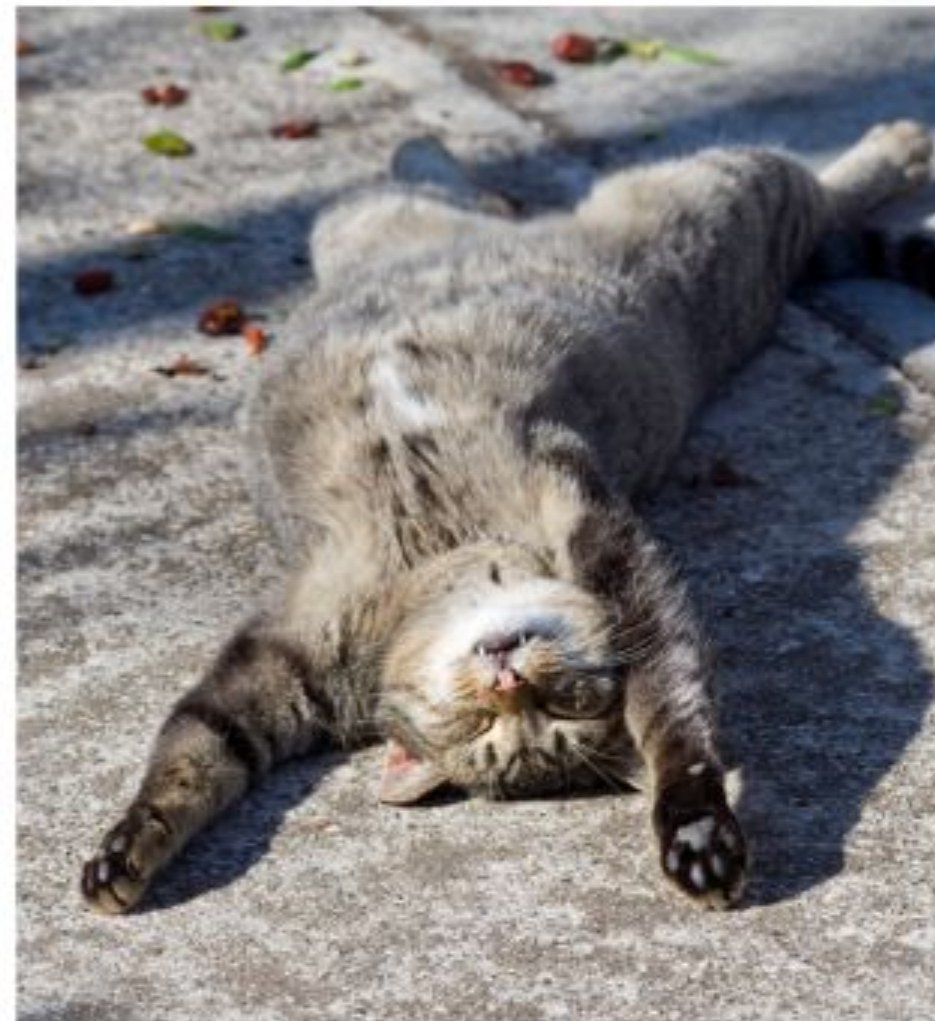


[This image is CC0 1.0 public domain](#)

Challenges: Deformation



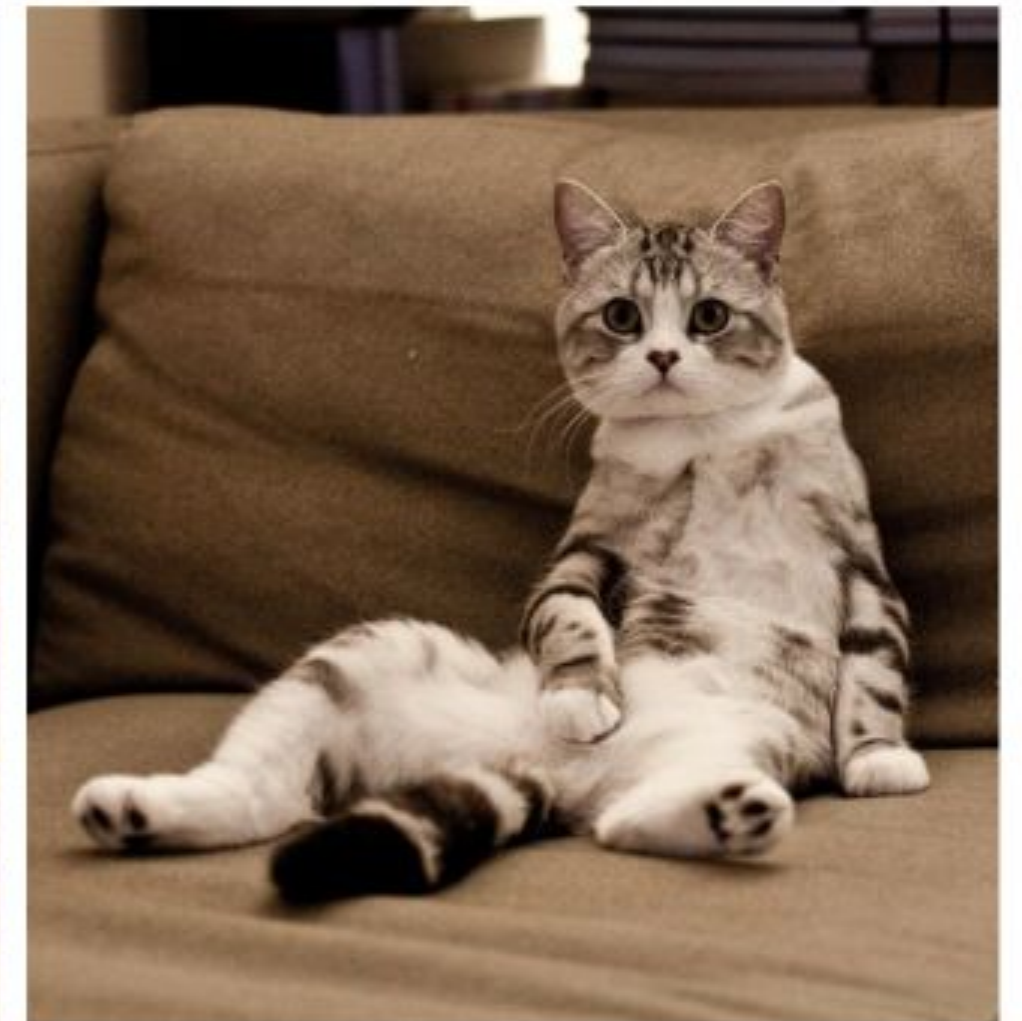
This image by [Umberto Salvagnin](#)
is licensed under [CC-BY 2.0](#)



This image by [Umberto Salvagnin](#)
is licensed under [CC-BY 2.0](#)



This image by [sare bear](#) is
licensed under [CC-BY 2.0](#)



This image by [Tom Thai](#) is
licensed under [CC-BY 2.0](#)

Challenges: Occlusion



[This image](#) is [CC0 1.0](#) public domain



[This image](#) is [CC0 1.0](#) public domain



[This image by jonsson](#) is licensed
under [CC-BY 2.0](#)

Challenges: Background Clutter



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

Challenges: Intraclass variation



This image is [CC0 1.0](#) public domain

An image classifier

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

```
def train(images, labels):  
    # Machine learning!  
    return model
```

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```

airplane



automobile



bird



cat



deer



Image classification

One possible solution (non optimal)

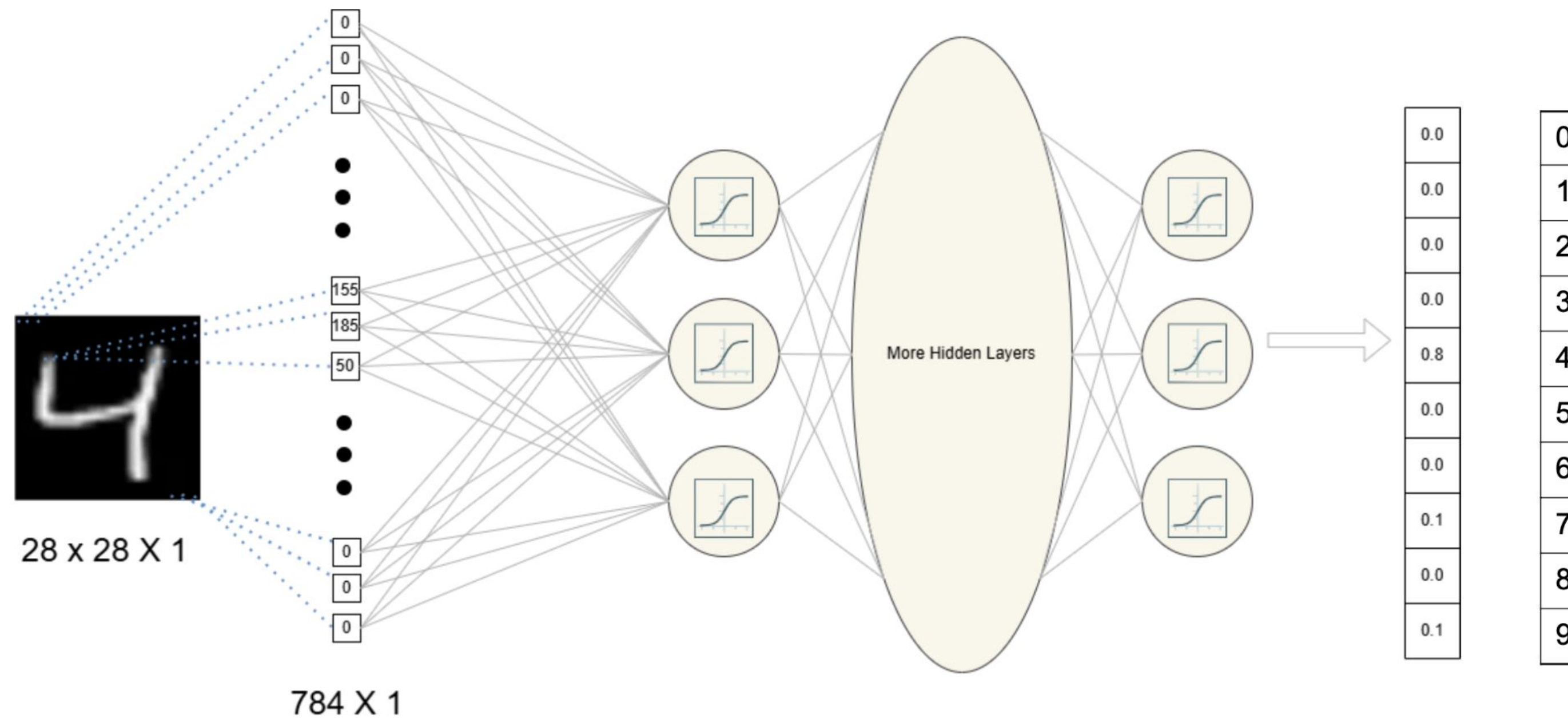
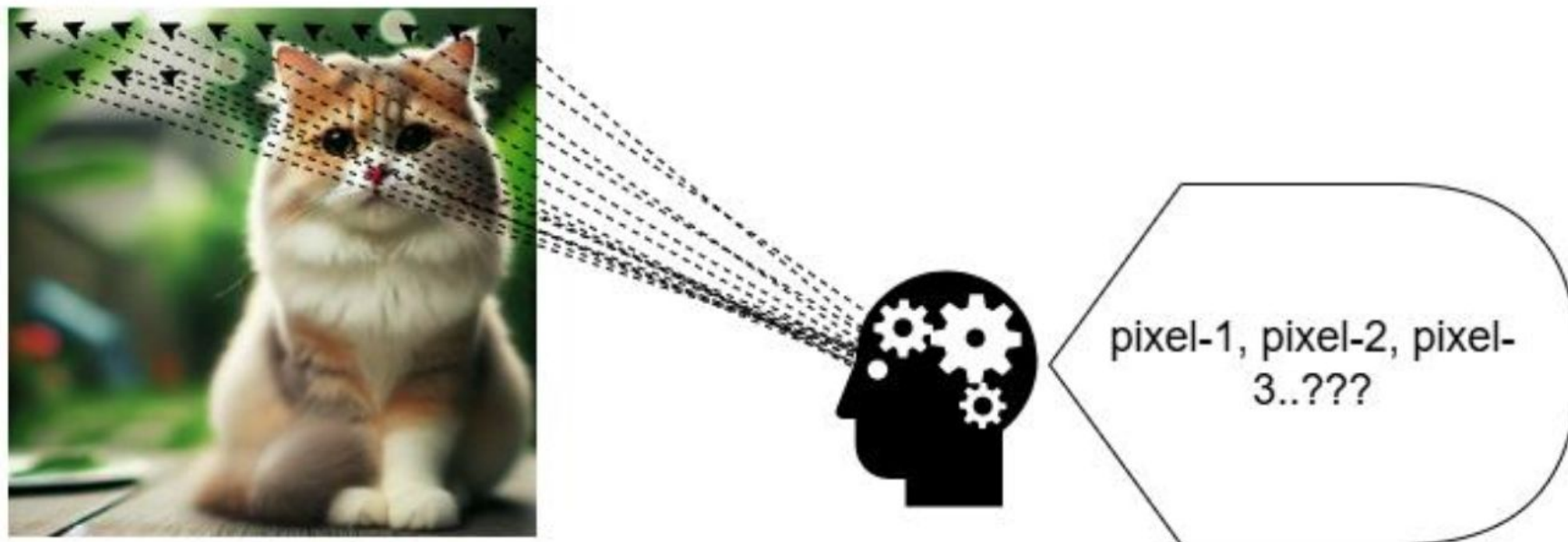


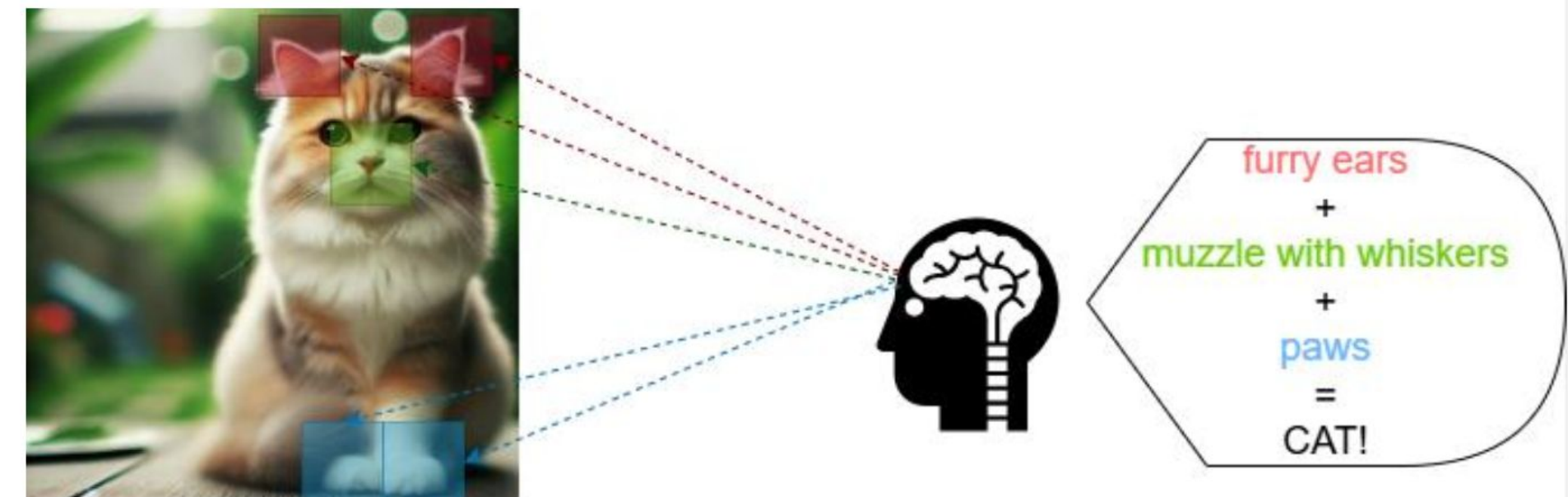
Image classification

How do humans classify images ?

Strategy 1

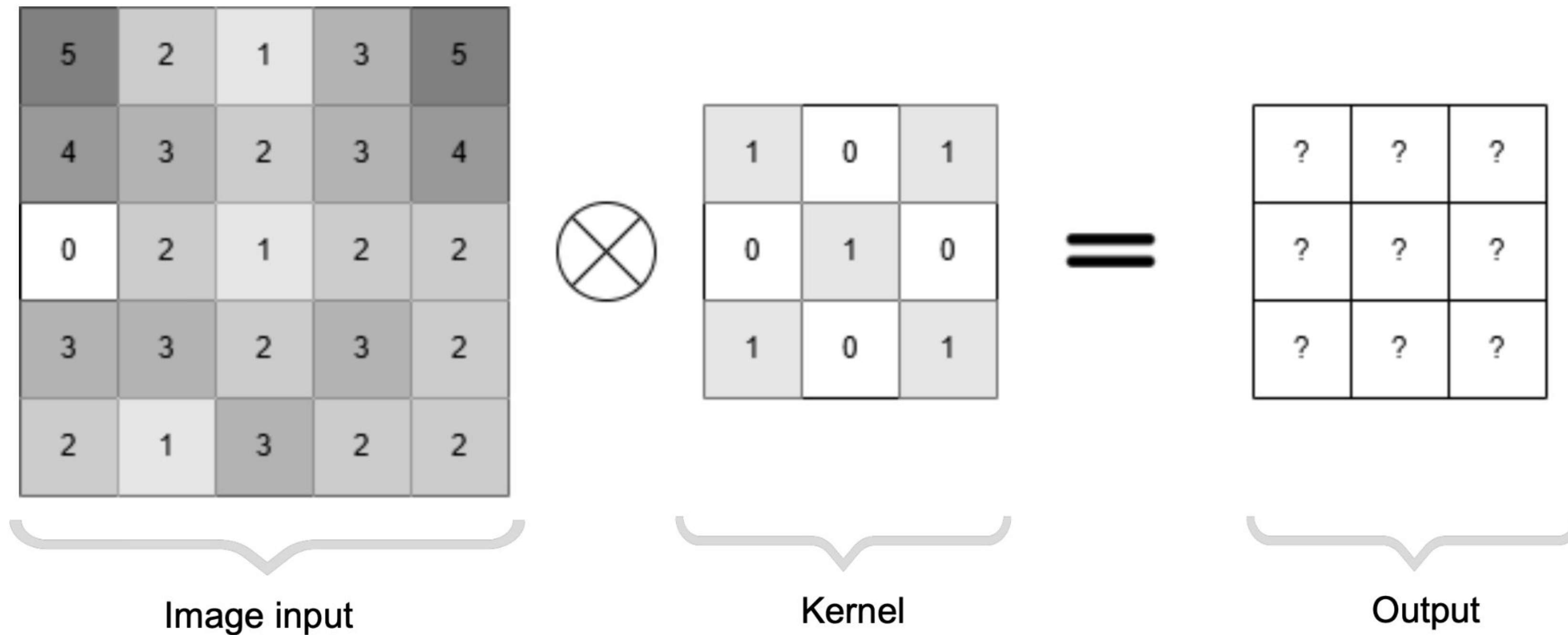


Strategy 2

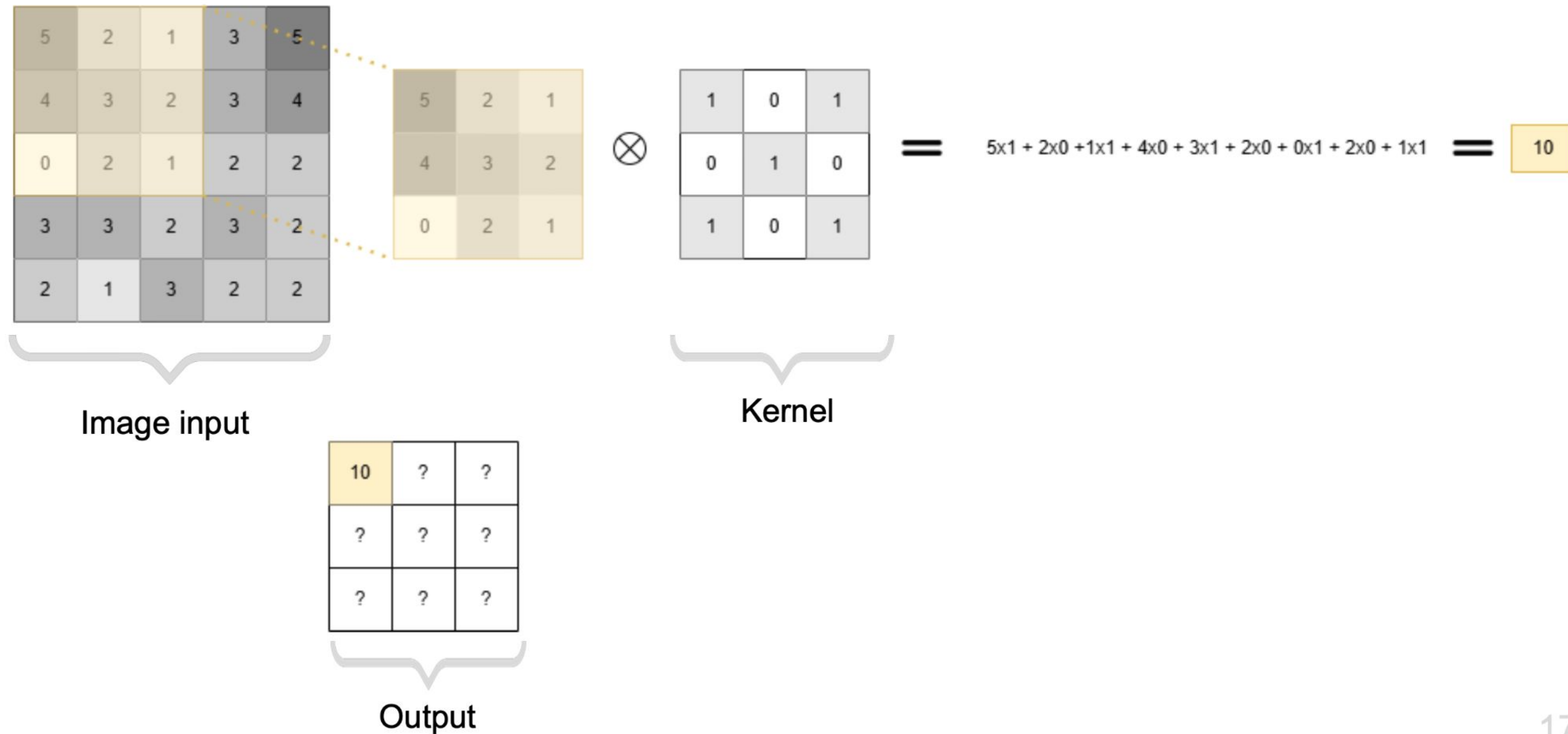


Idea : use **features detectors**

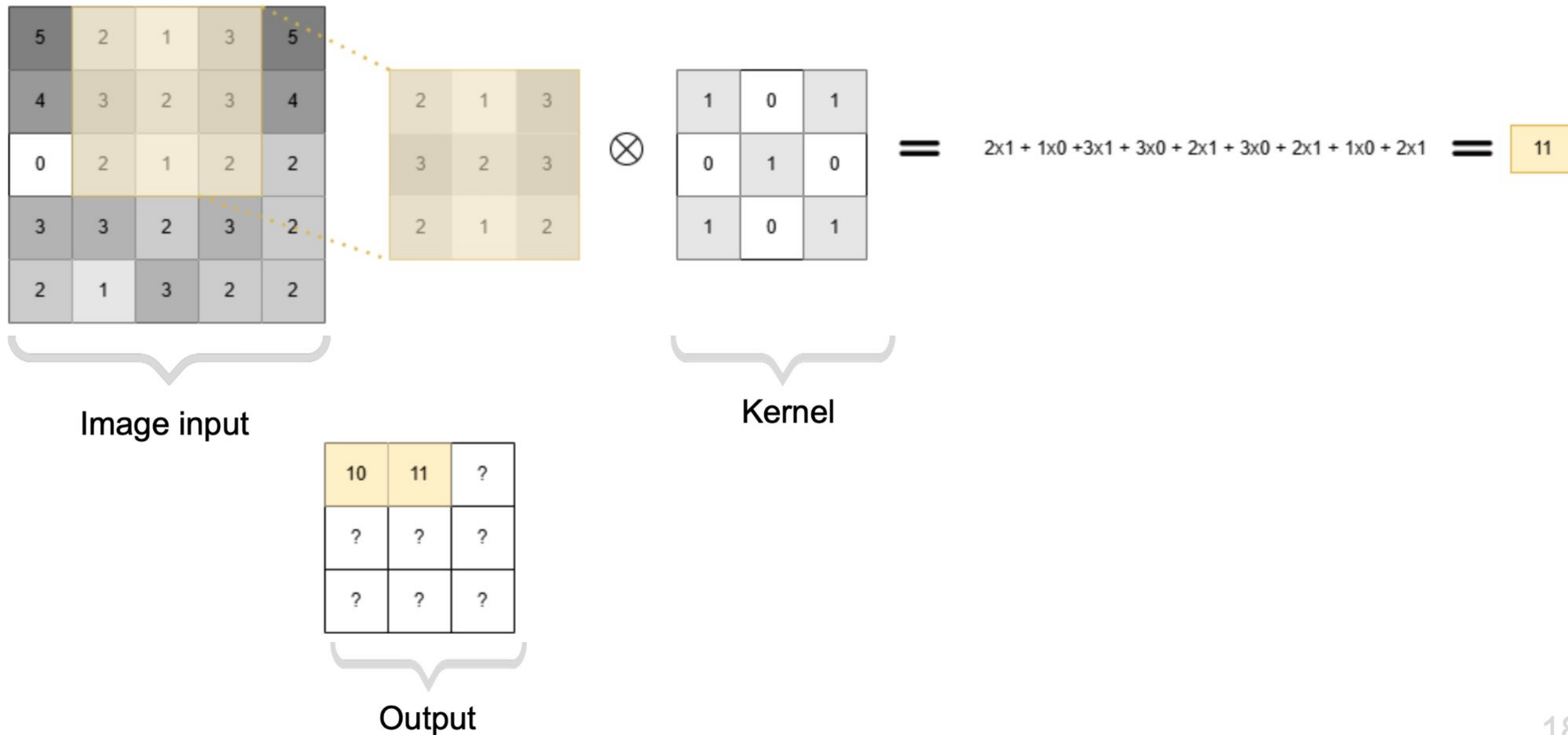
Convolutions



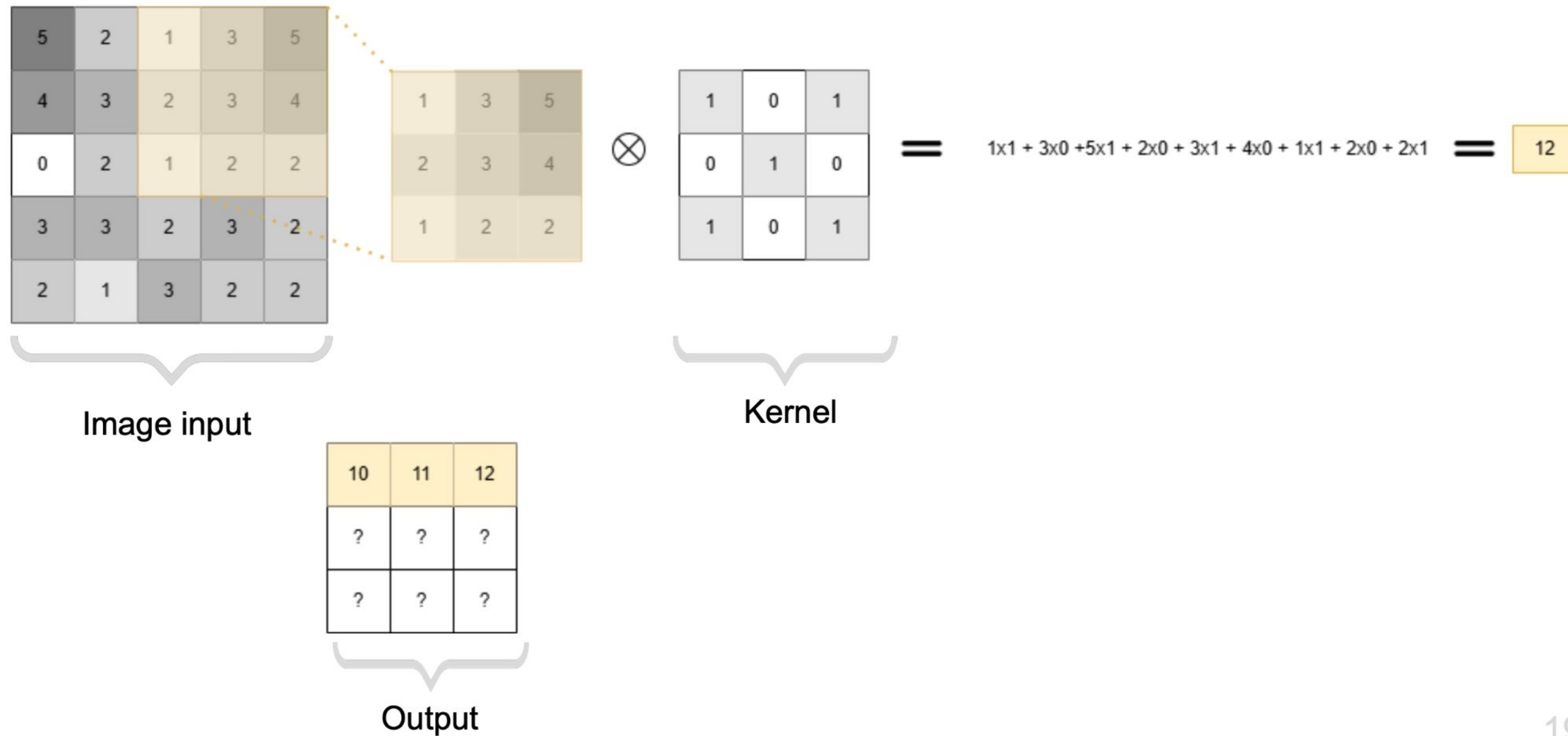
Convolutions



Convolutions



Convolutions



Convolutions

5	2	1	3	5
4	3	2	3	4
0	2	1	2	2
3	3	2	3	2
2	1	3	2	2

Image input



1	0	1
0	1	0
1	0	1

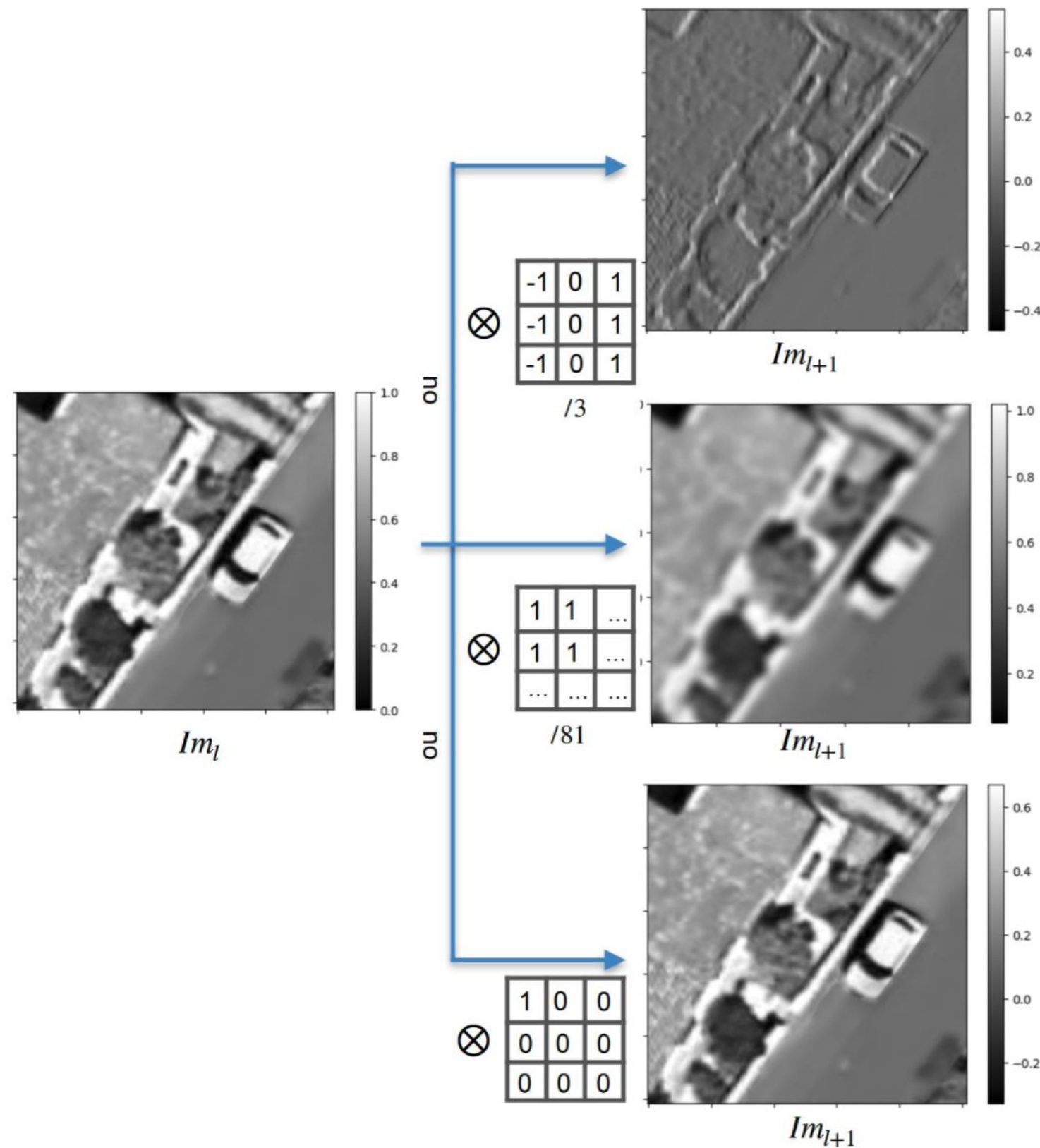
Kernel

=

10	11	12
13	13	12
9	9	11

Output

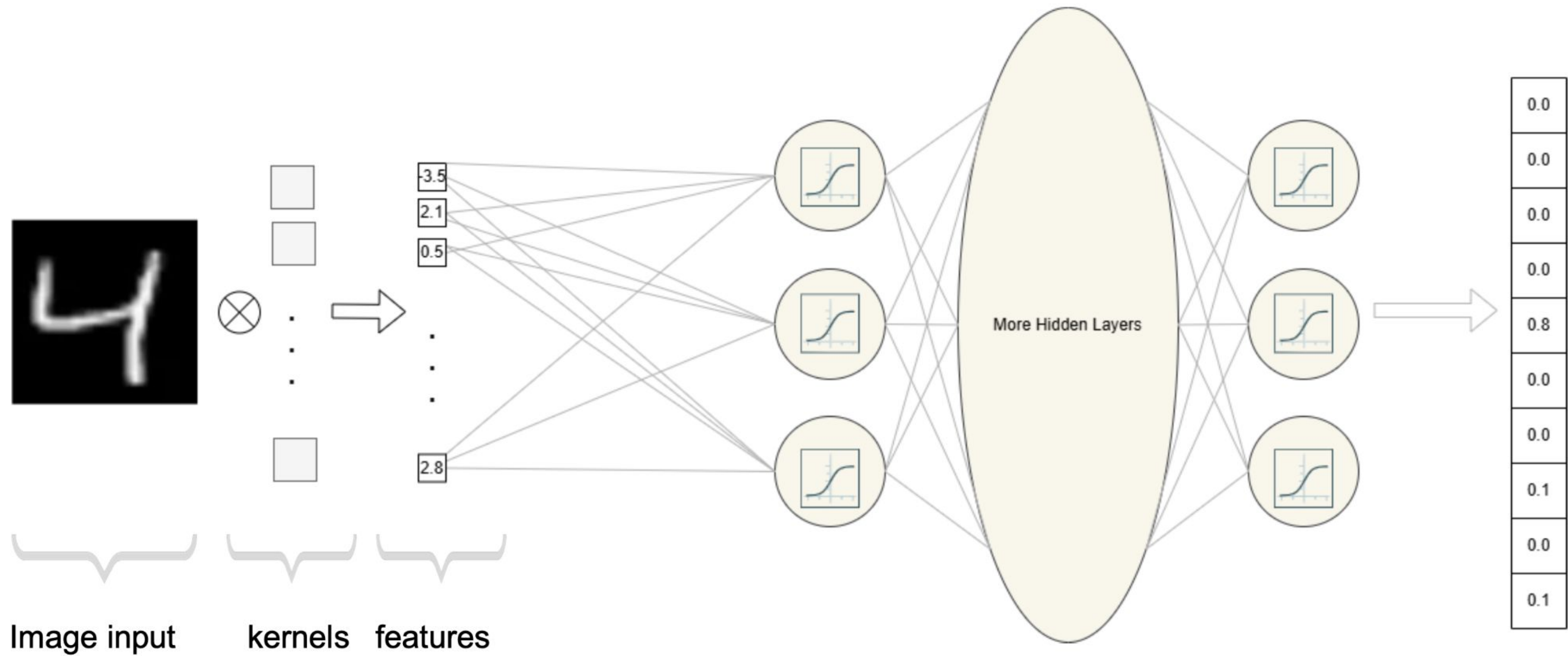
Convolutions



Lot of known kernels :

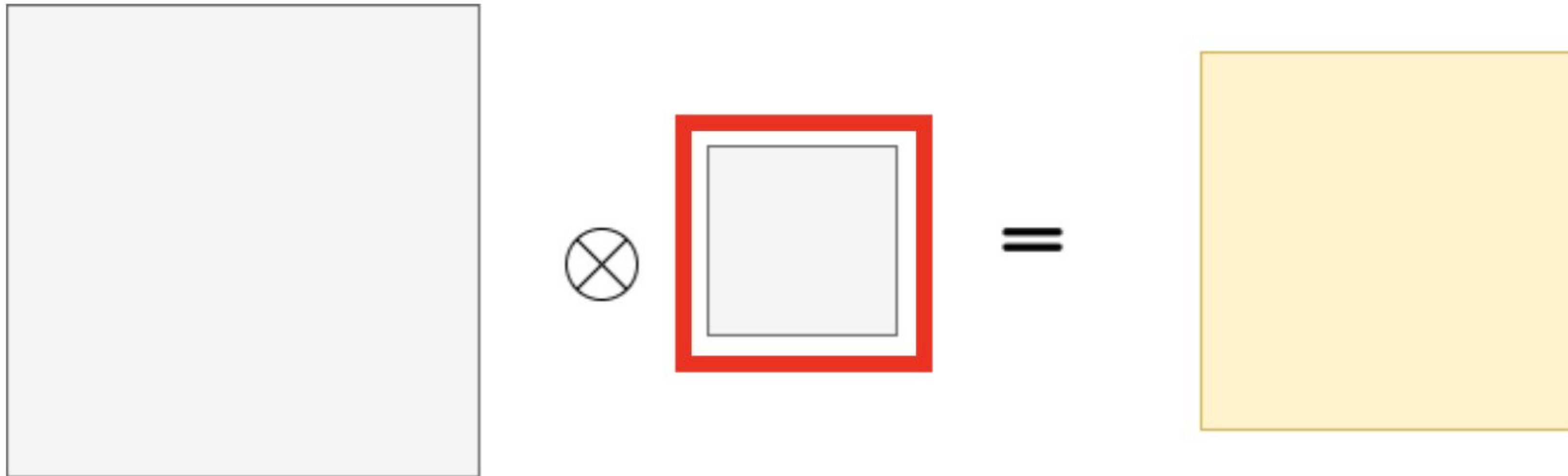
- edge detectors
- shapes : curves, squares...
- textures

Convolutions



Convolutions

Idea behind CNN : Instead of using predefined kernels designed empirically by humans → make the model learn the kernel parameters



Convolutional Neural Networks

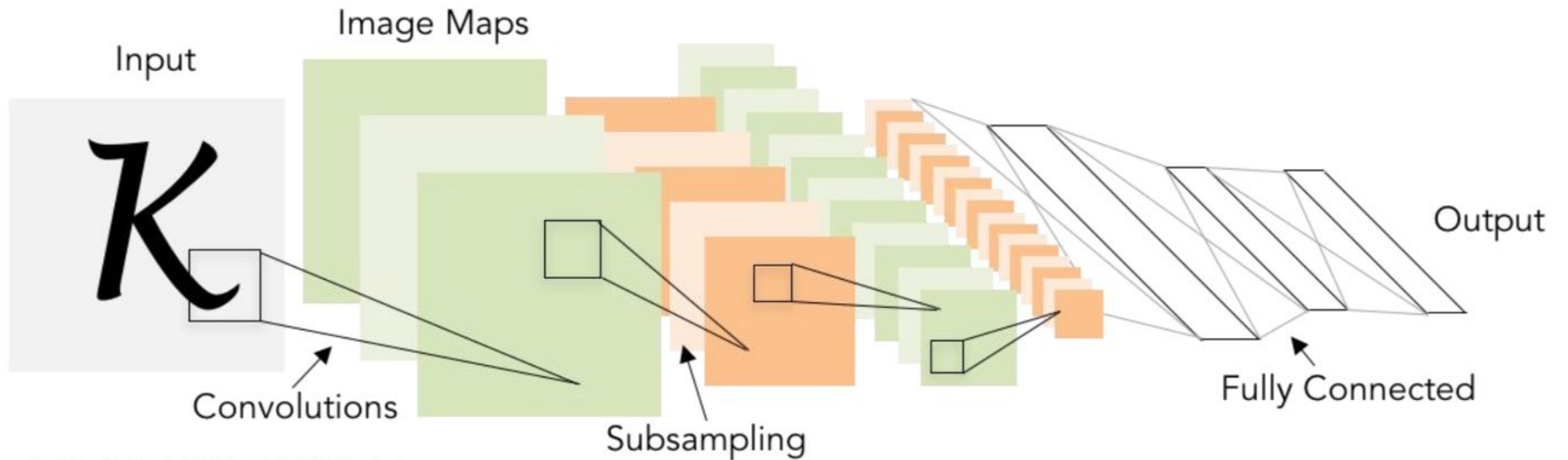
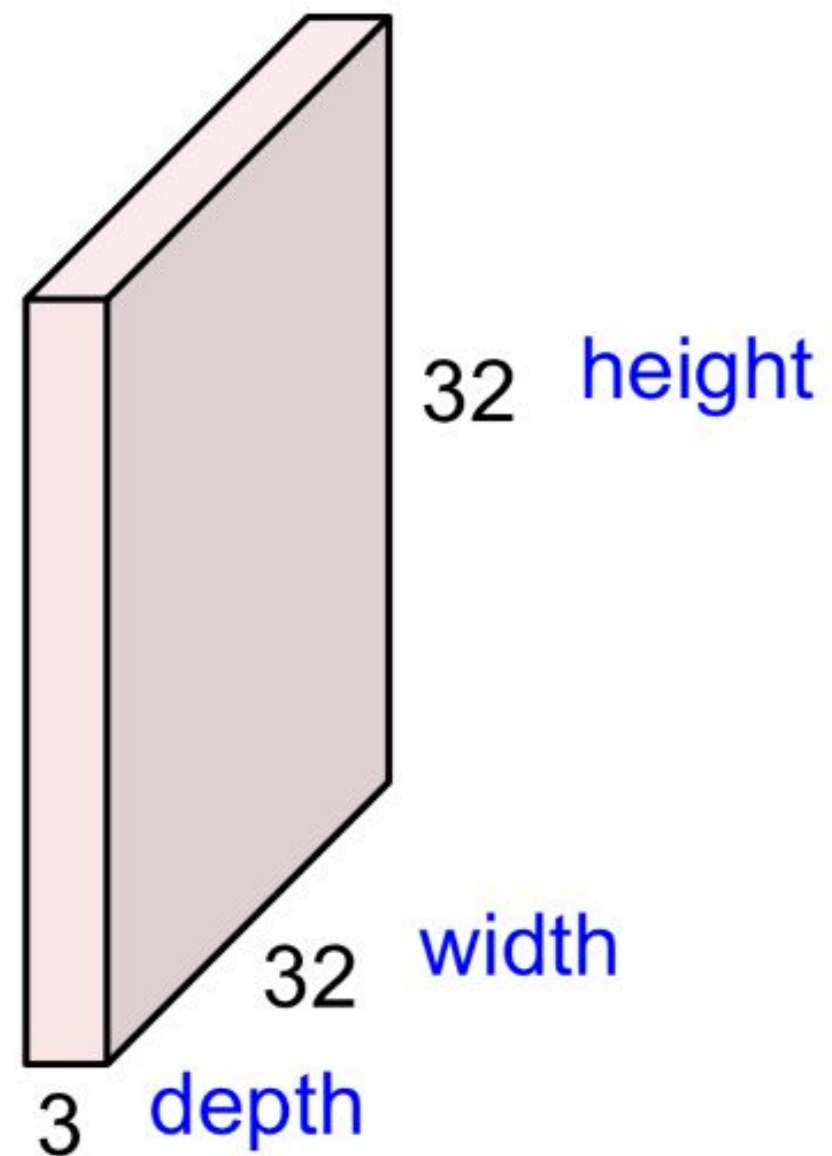


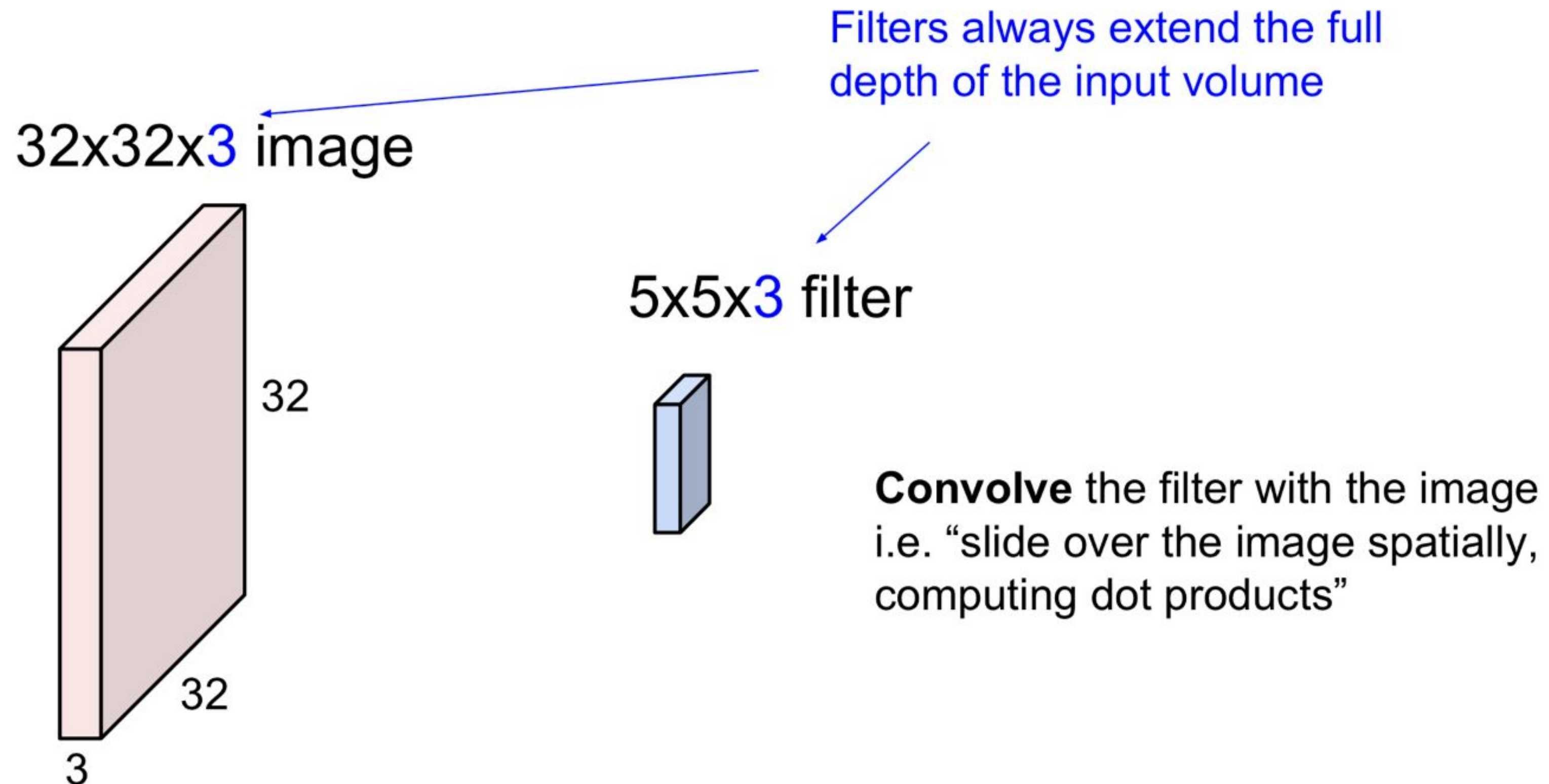
Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Convolutional Layer

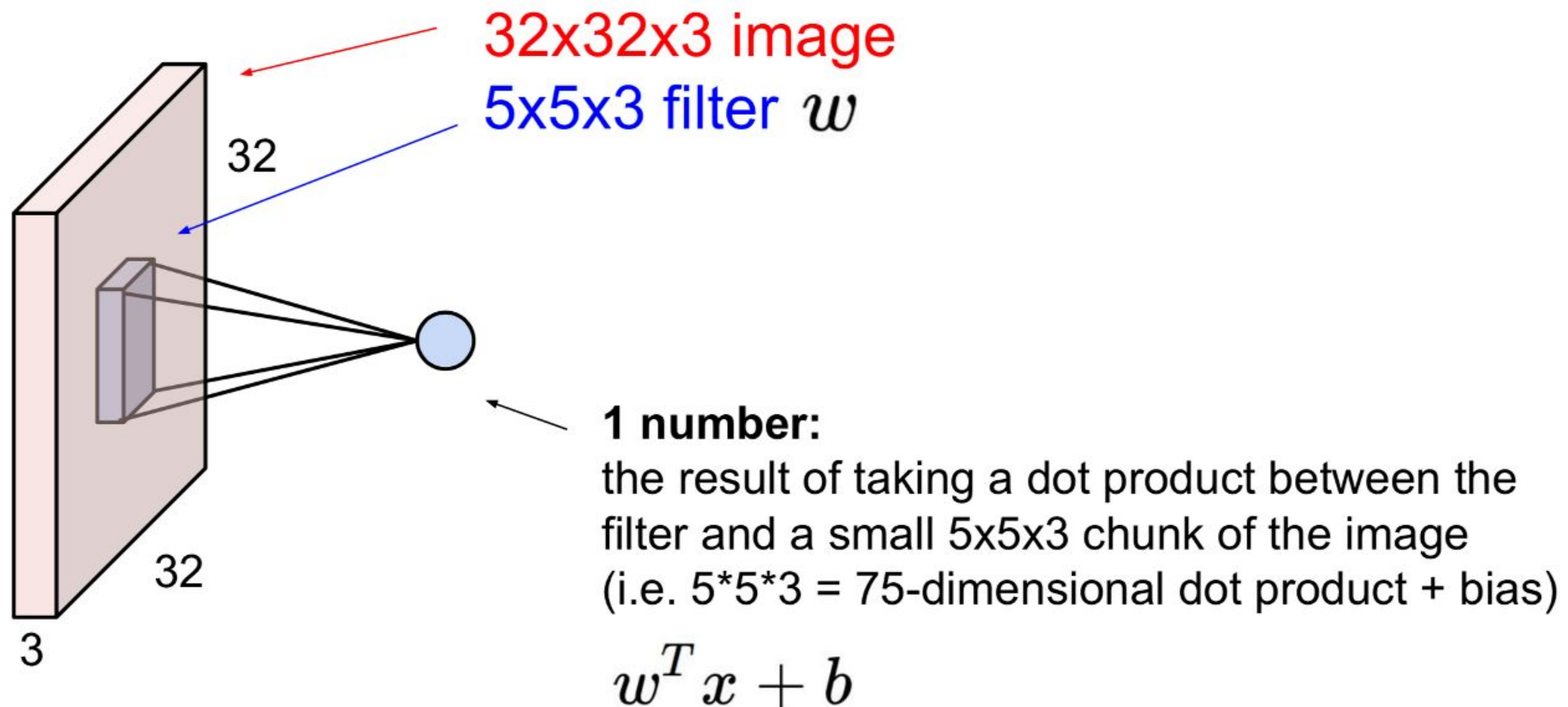
32x32x3 image -> preserve spatial structure



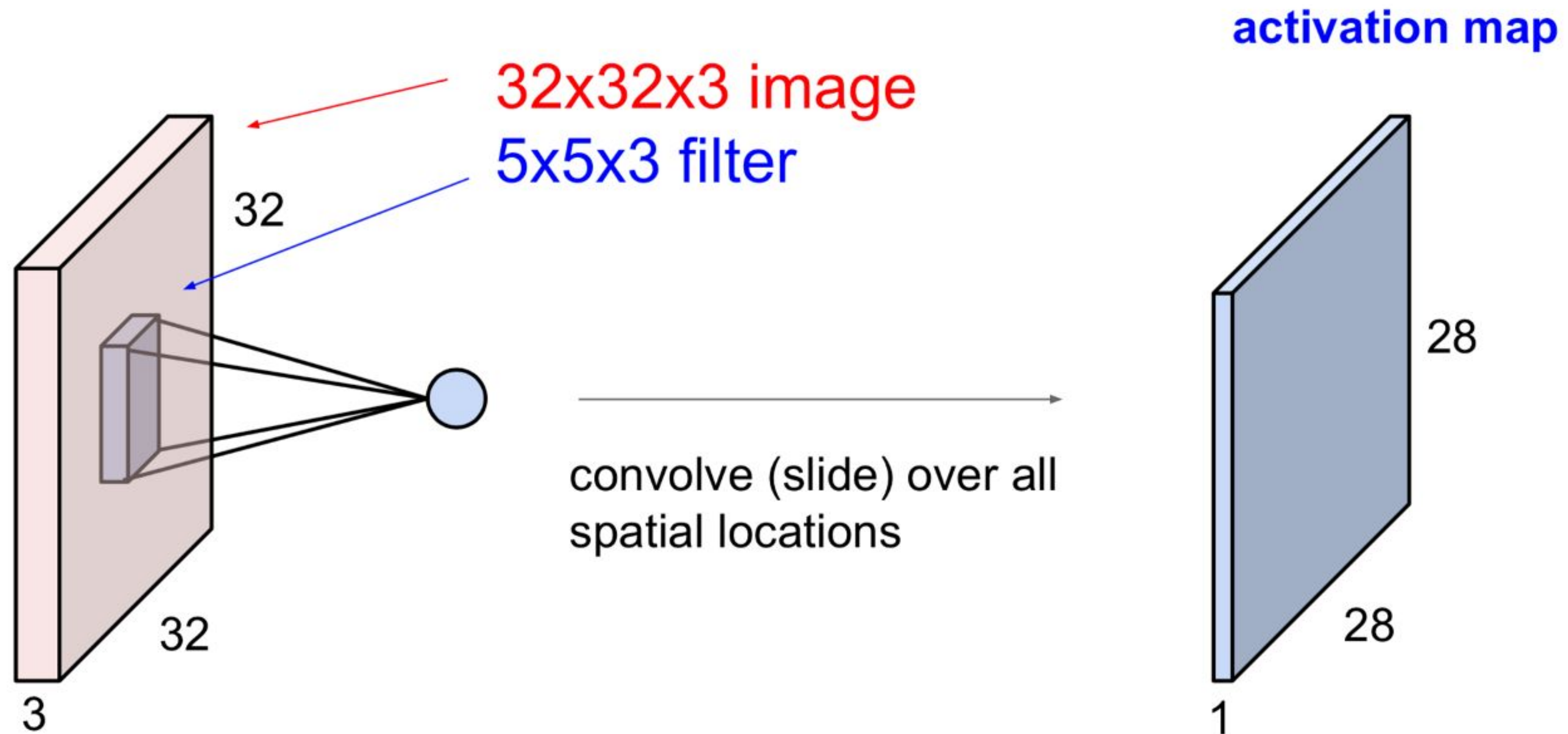
Convolutional Layer



Convolutional Layer

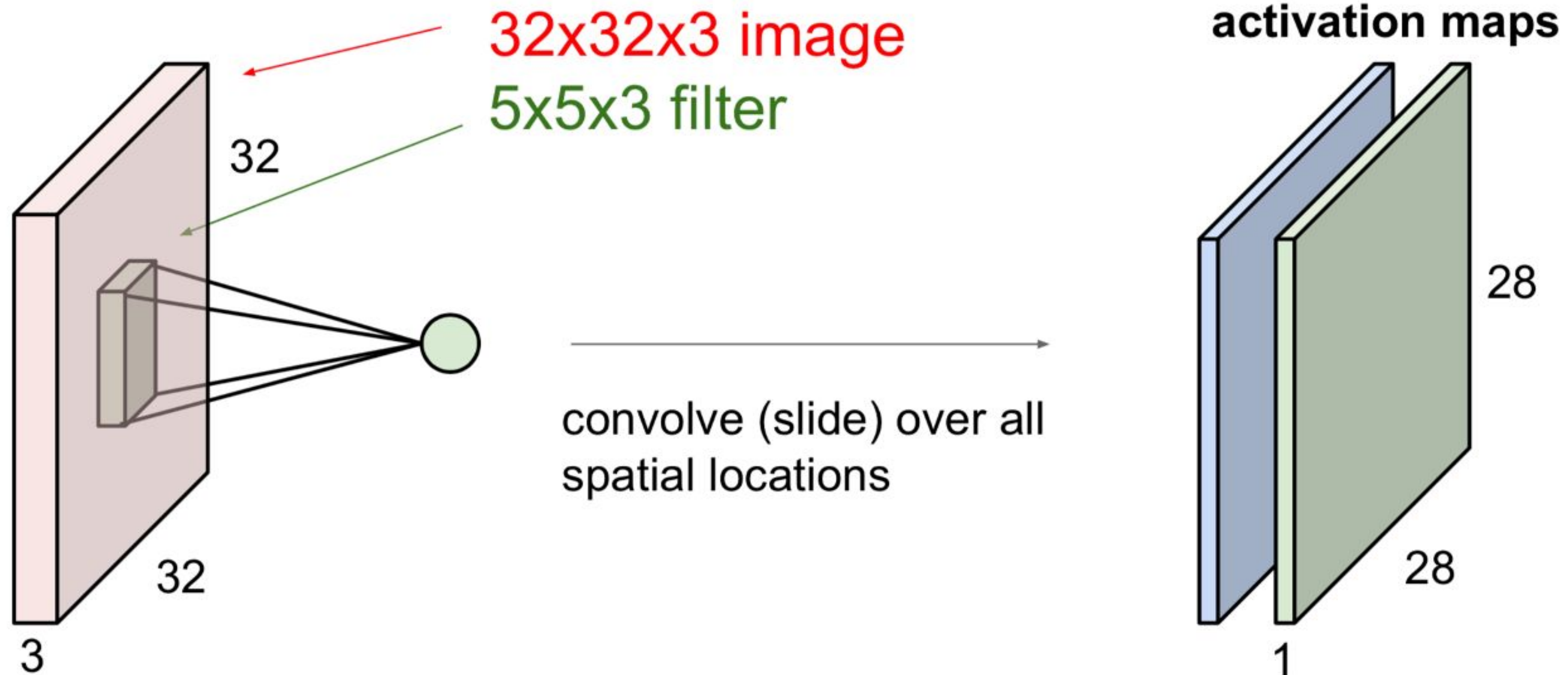


Convolutional Layer



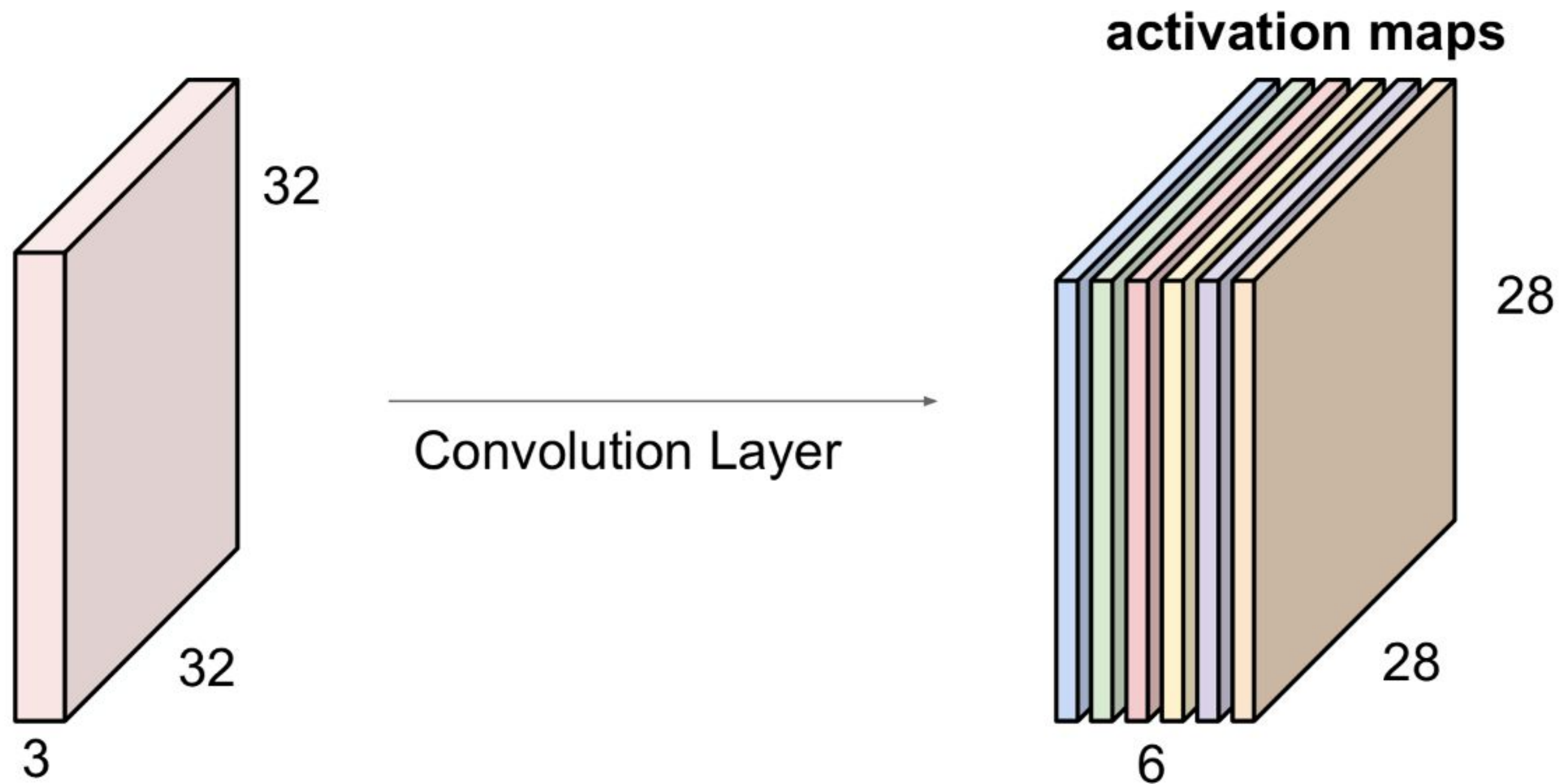
Convolutional Layer

consider a second, **green** filter



Convolutional Layer

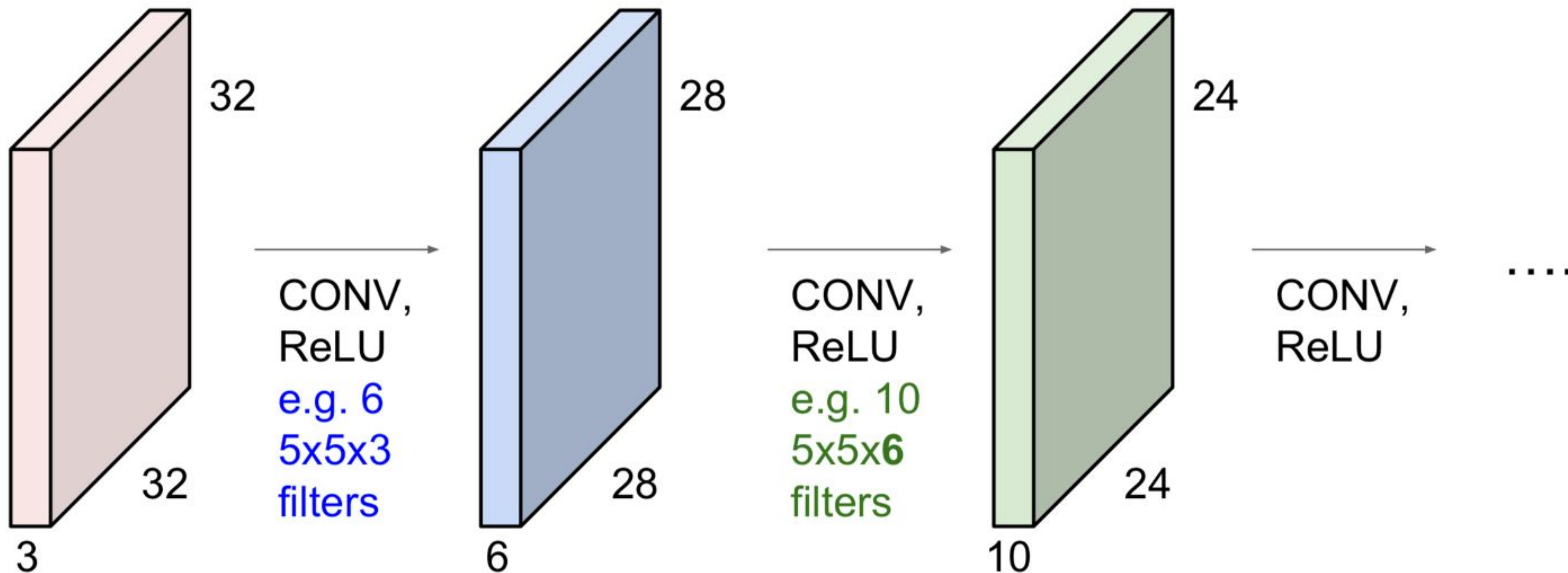
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

Convolutional Layer

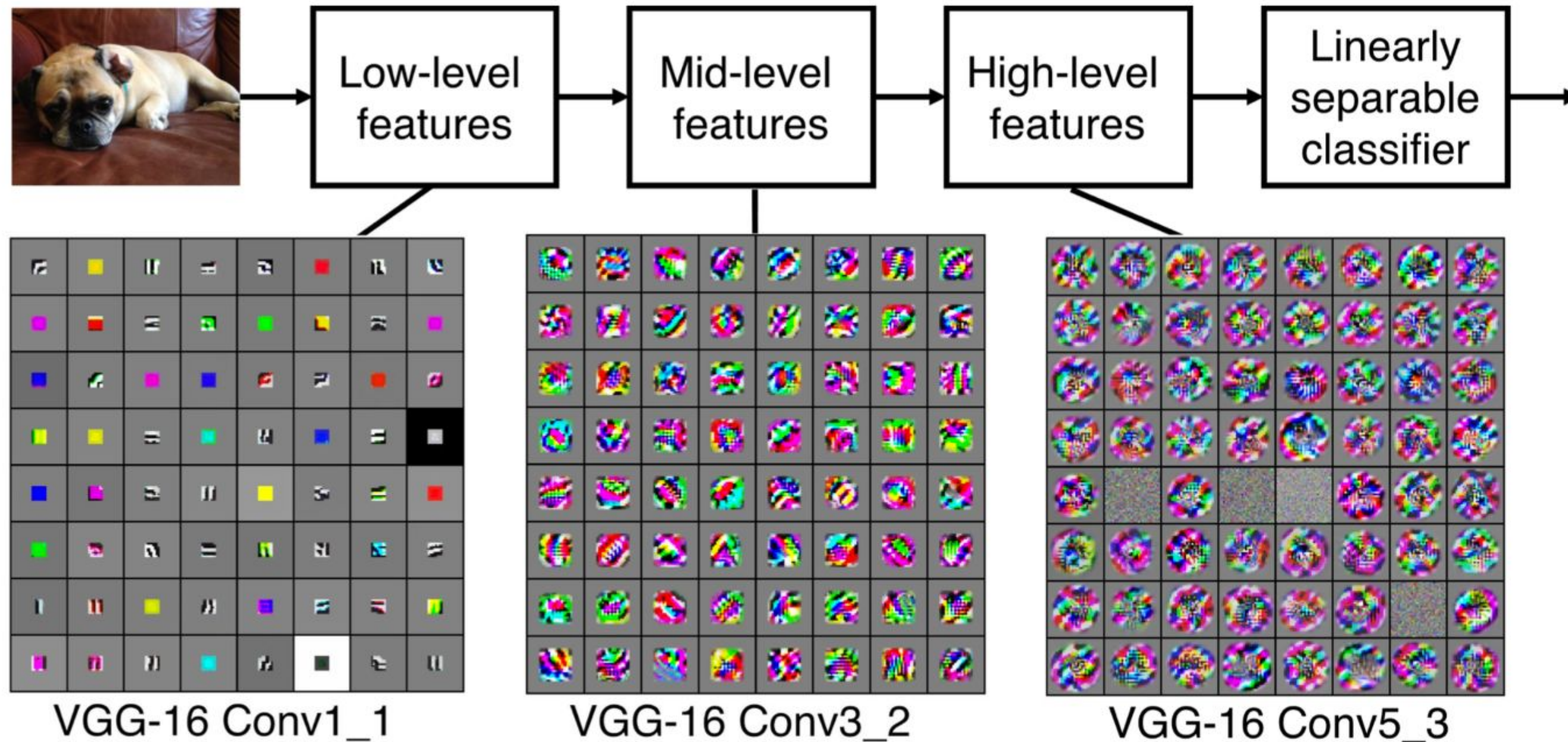
➡ ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



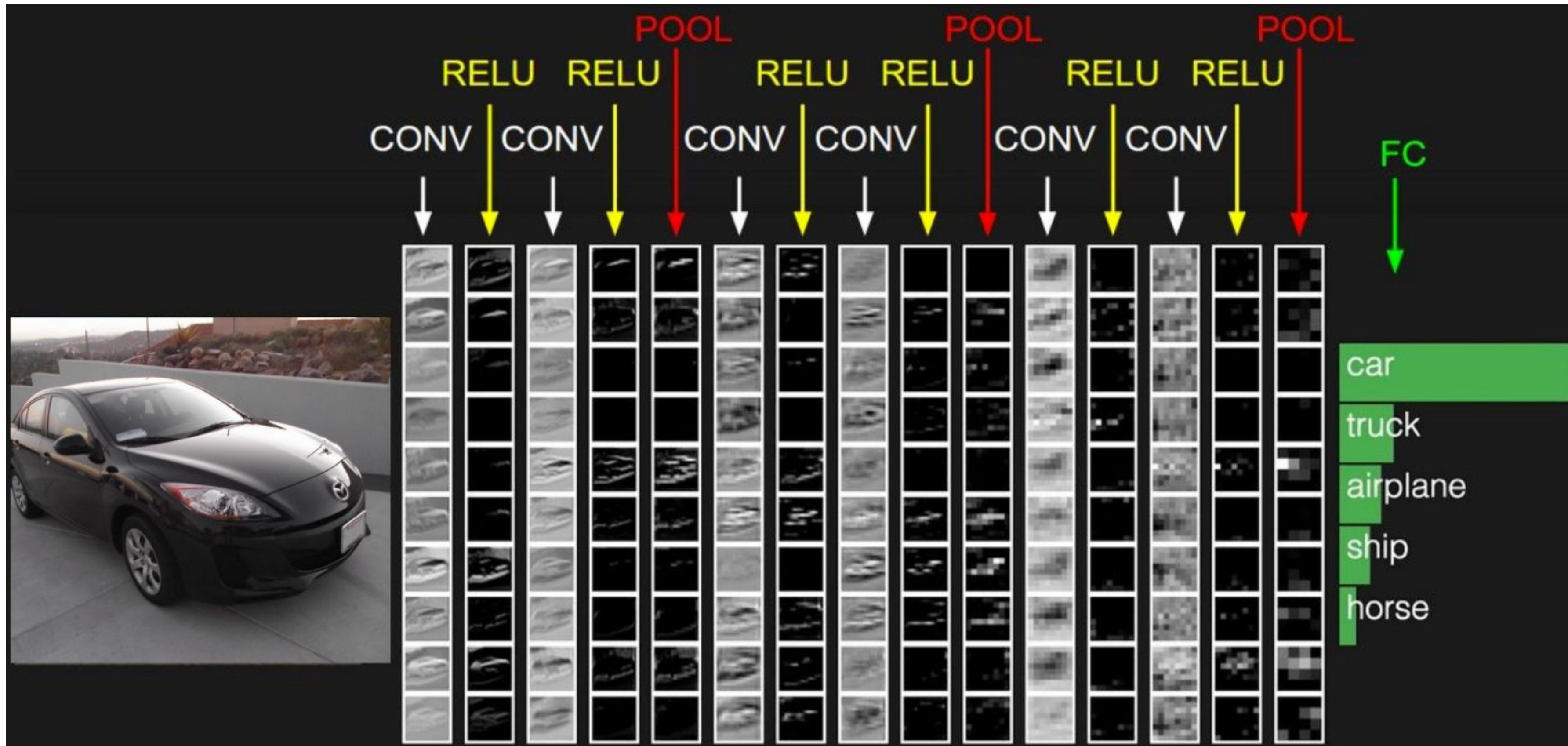
Convolutional Layer

[Zeiler and Fergus 2013]

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

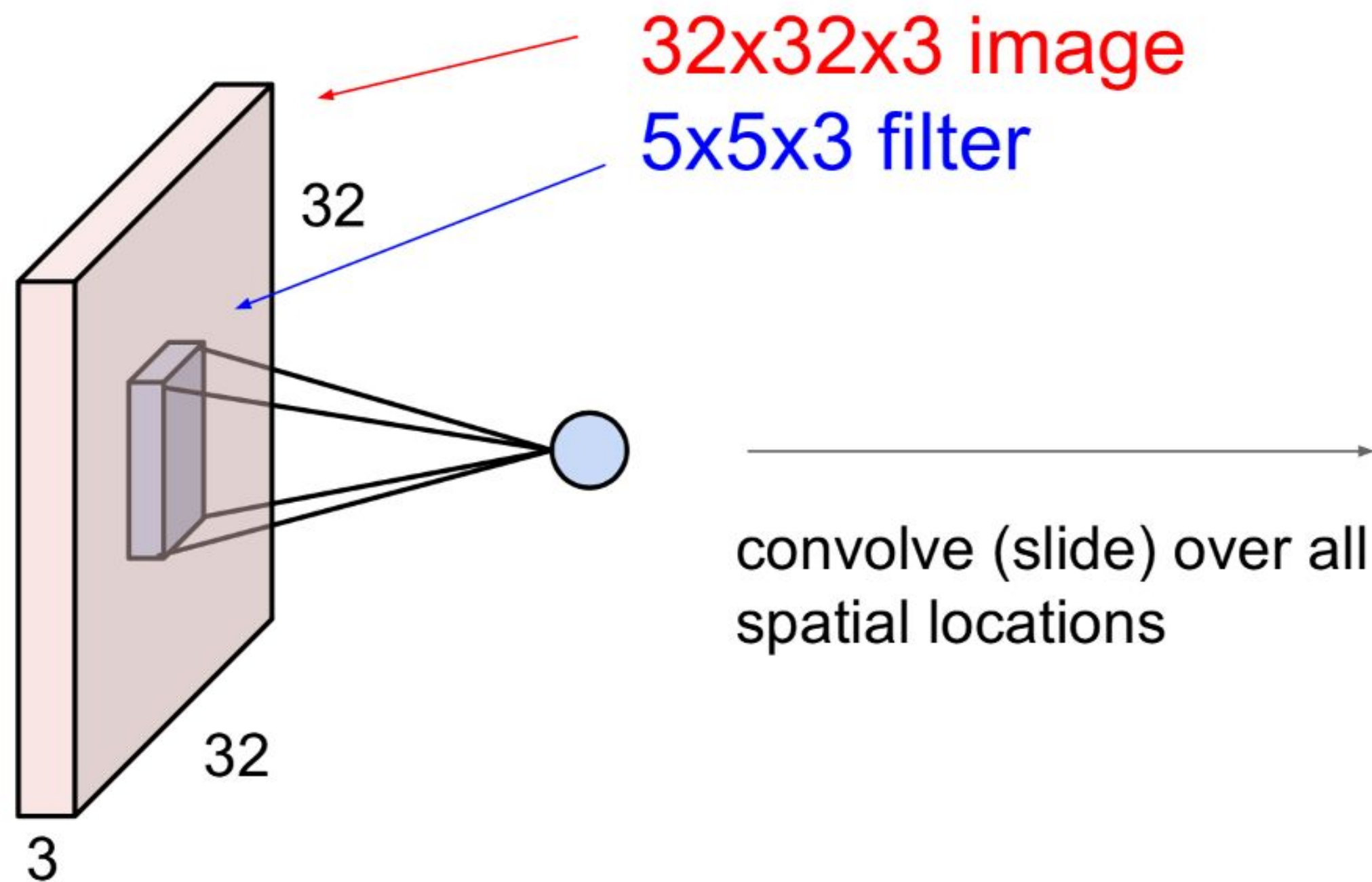


Convolutional Neural Network

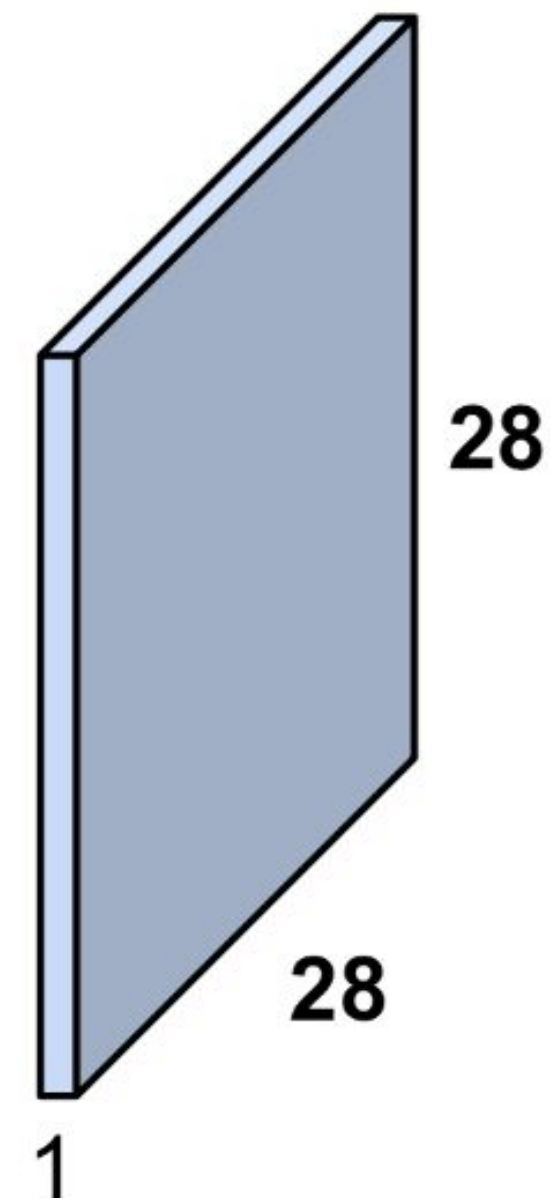


Convolutional Layer

A closer look at spatial dimensions:



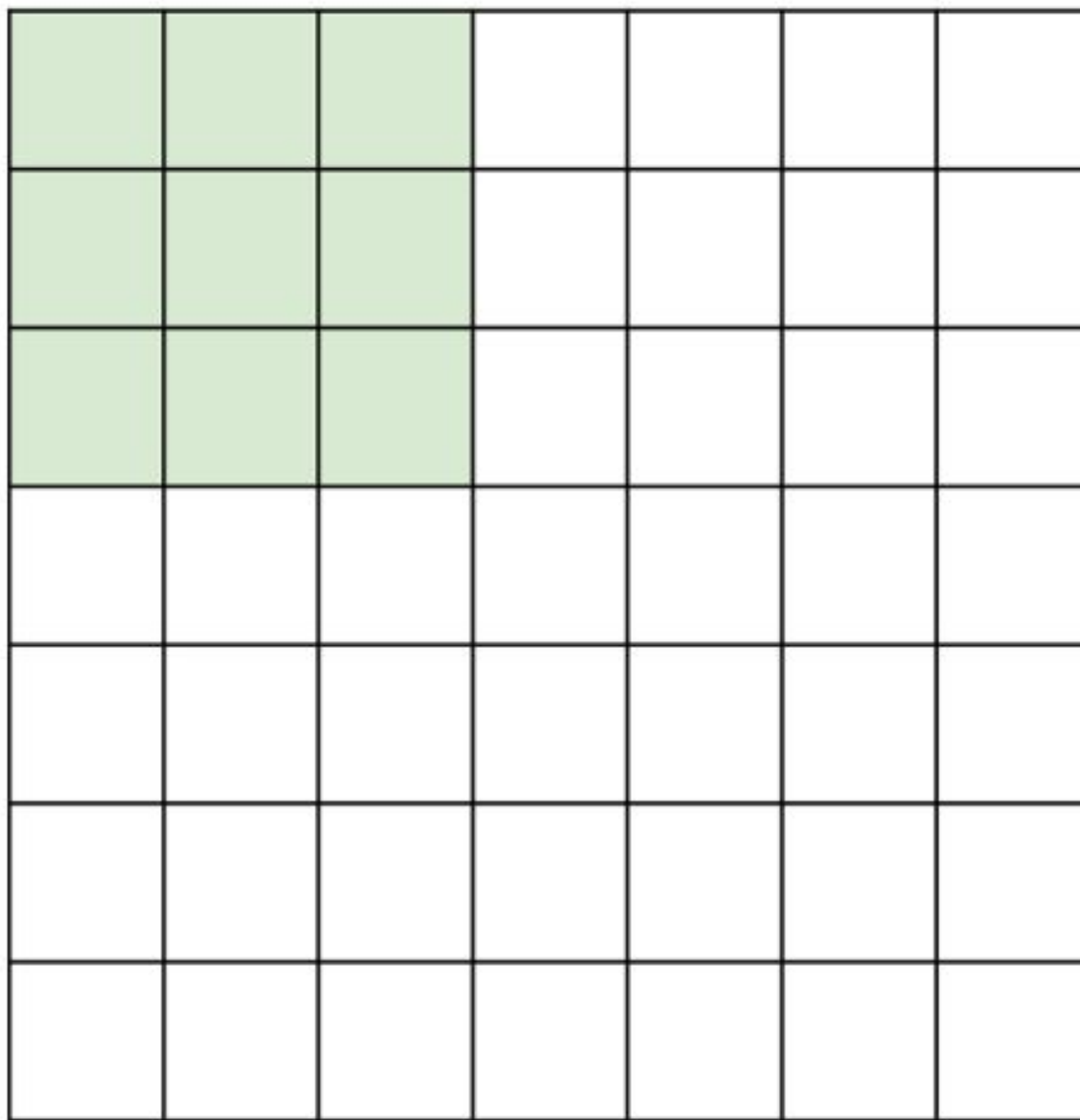
activation map



Convolutional Layer

A closer look at spatial dimensions:

7



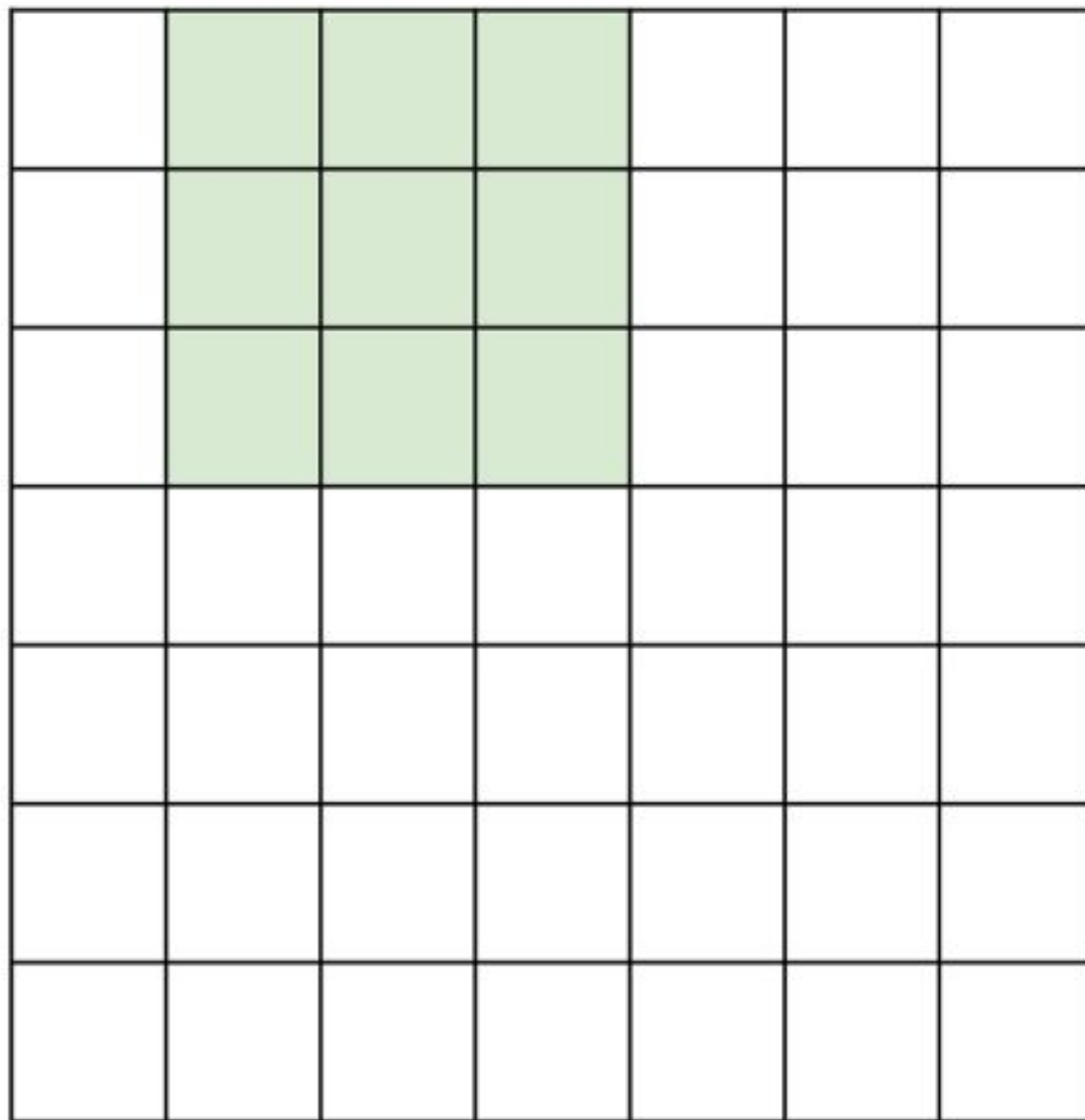
7

7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

A closer look at spatial dimensions:

7



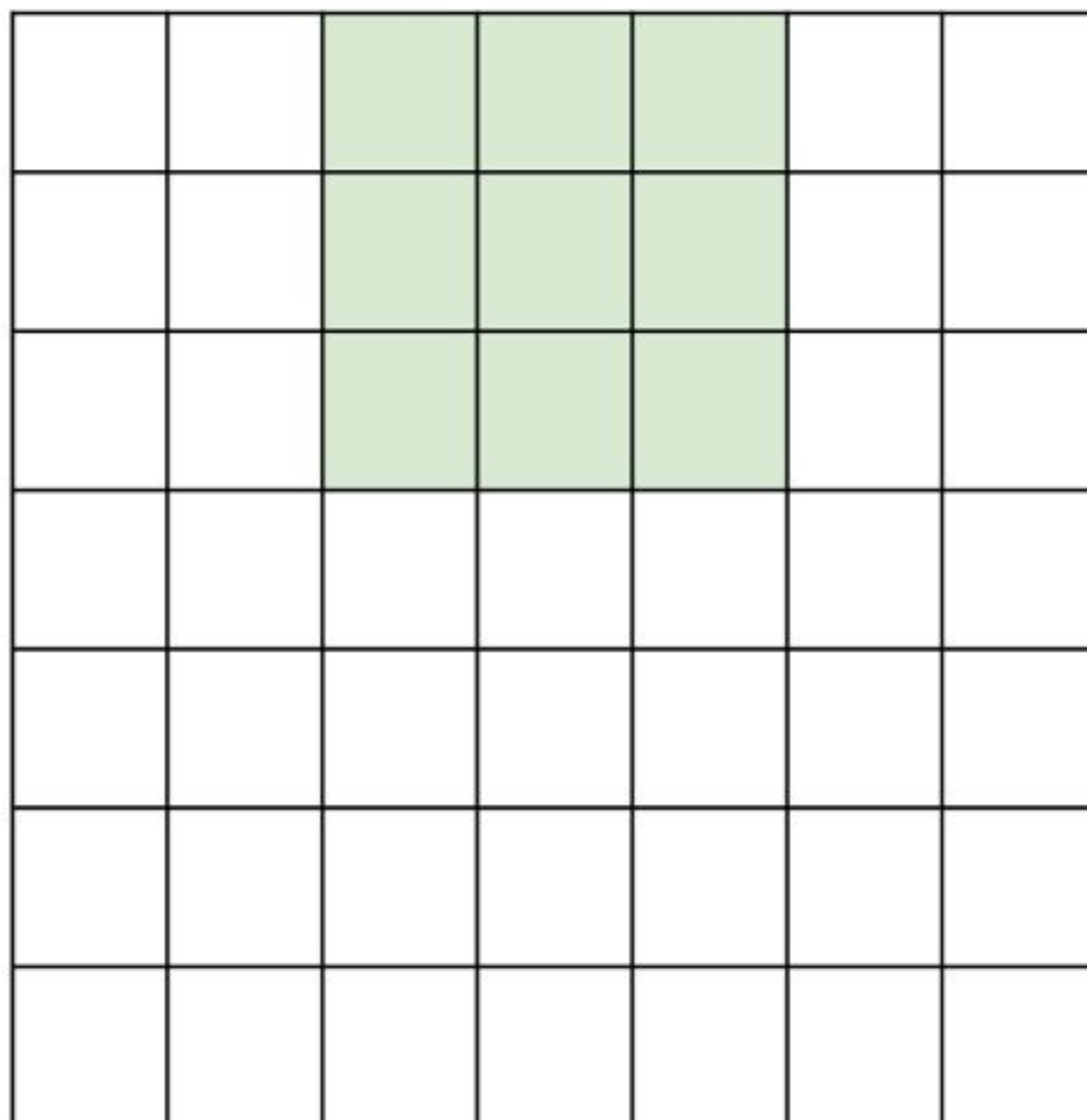
7

7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

A closer look at spatial dimensions:

7



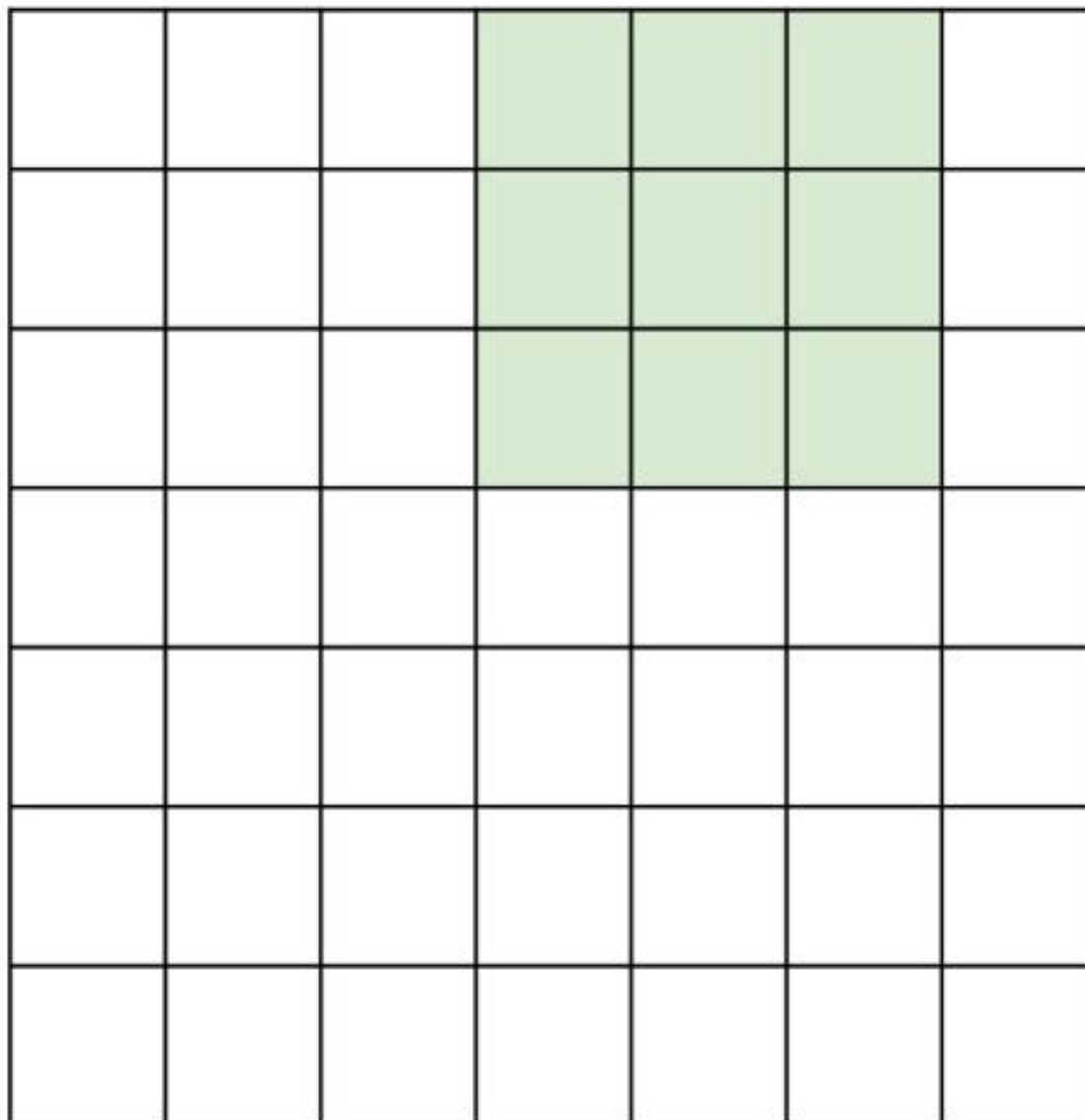
7

7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

A closer look at spatial dimensions:

7



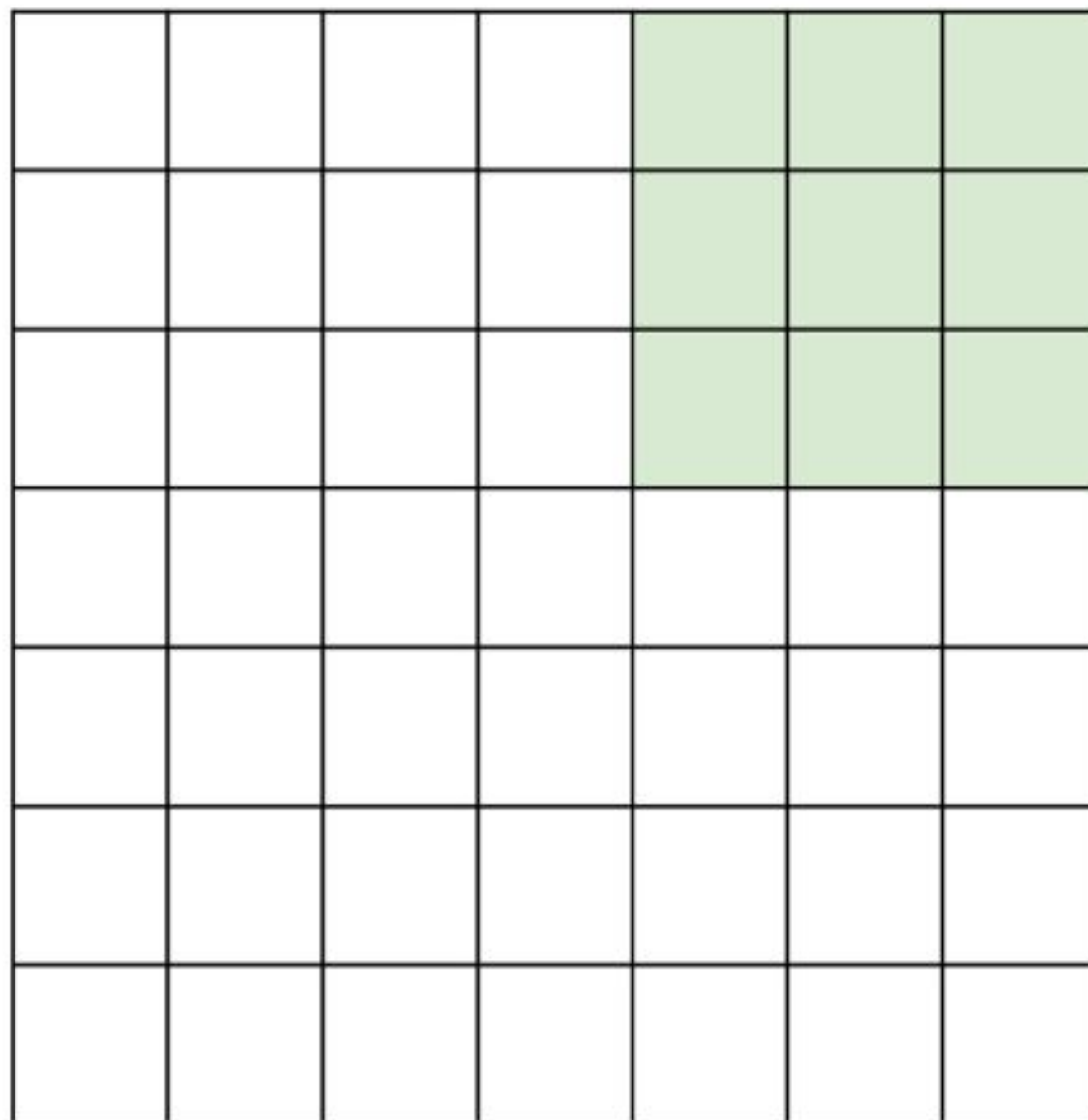
7

7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

A closer look at spatial dimensions:

7



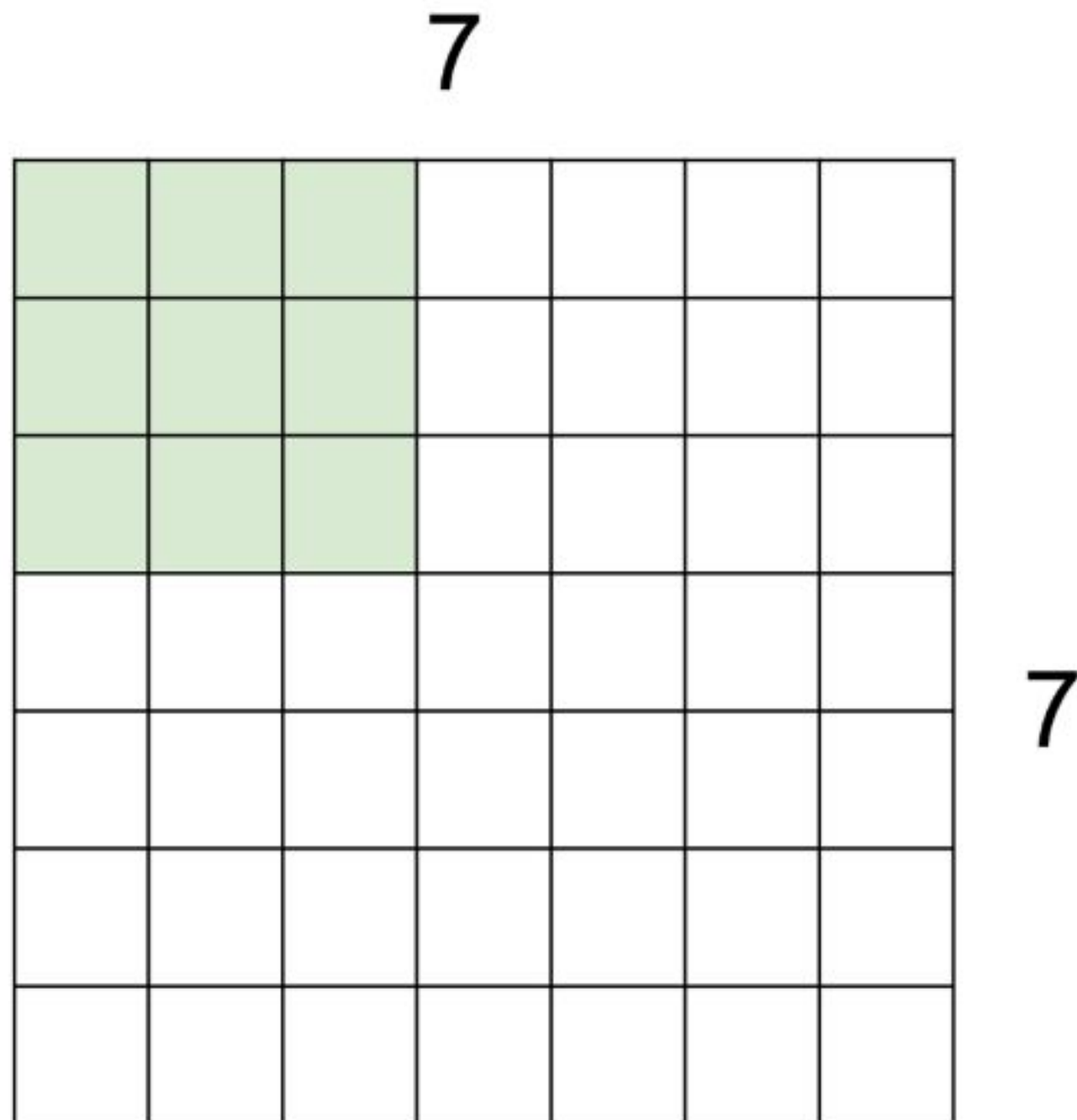
7

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Convolutional Layer

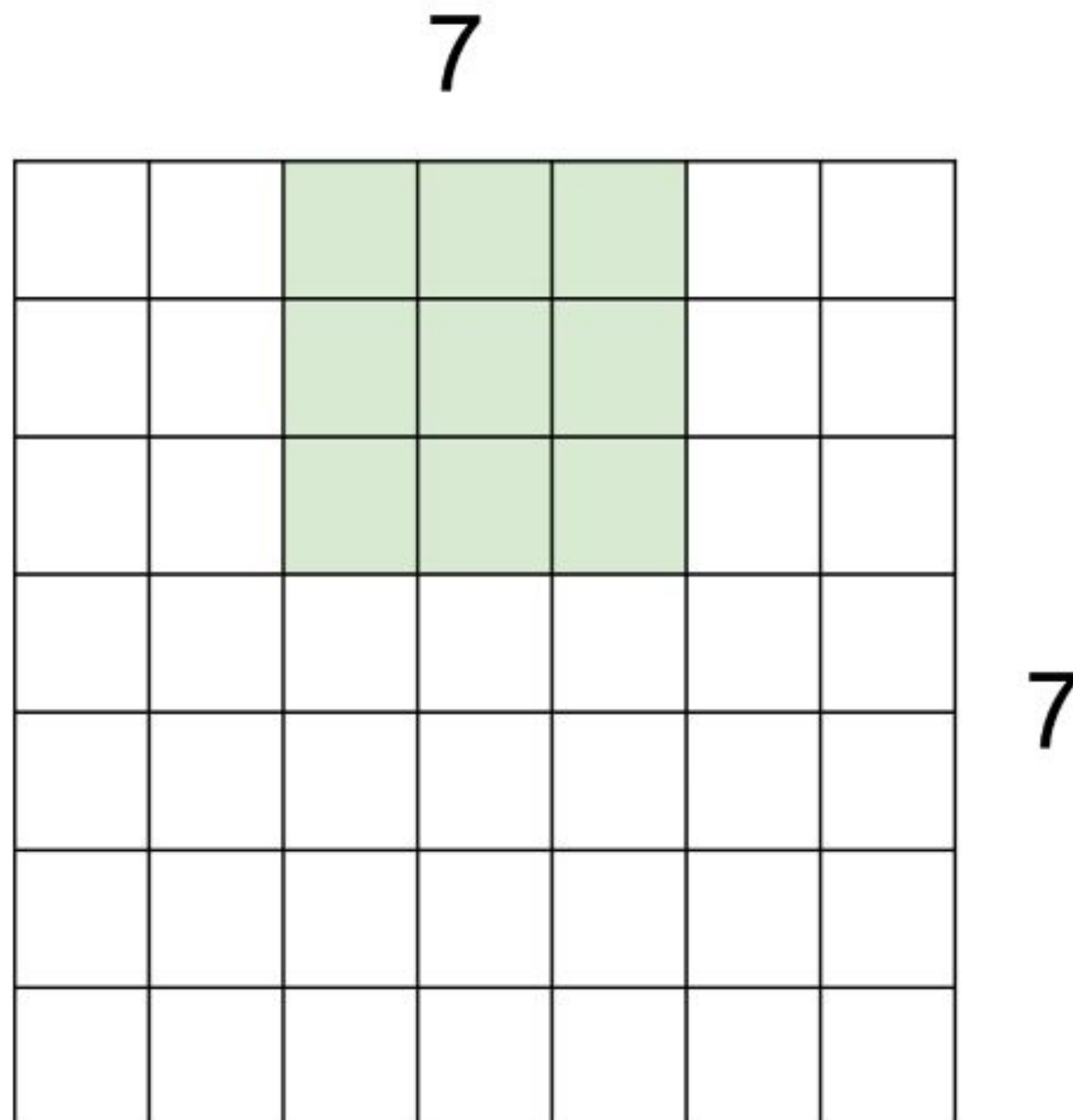
A closer look at spatial dimensions:



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

Convolutional Layer

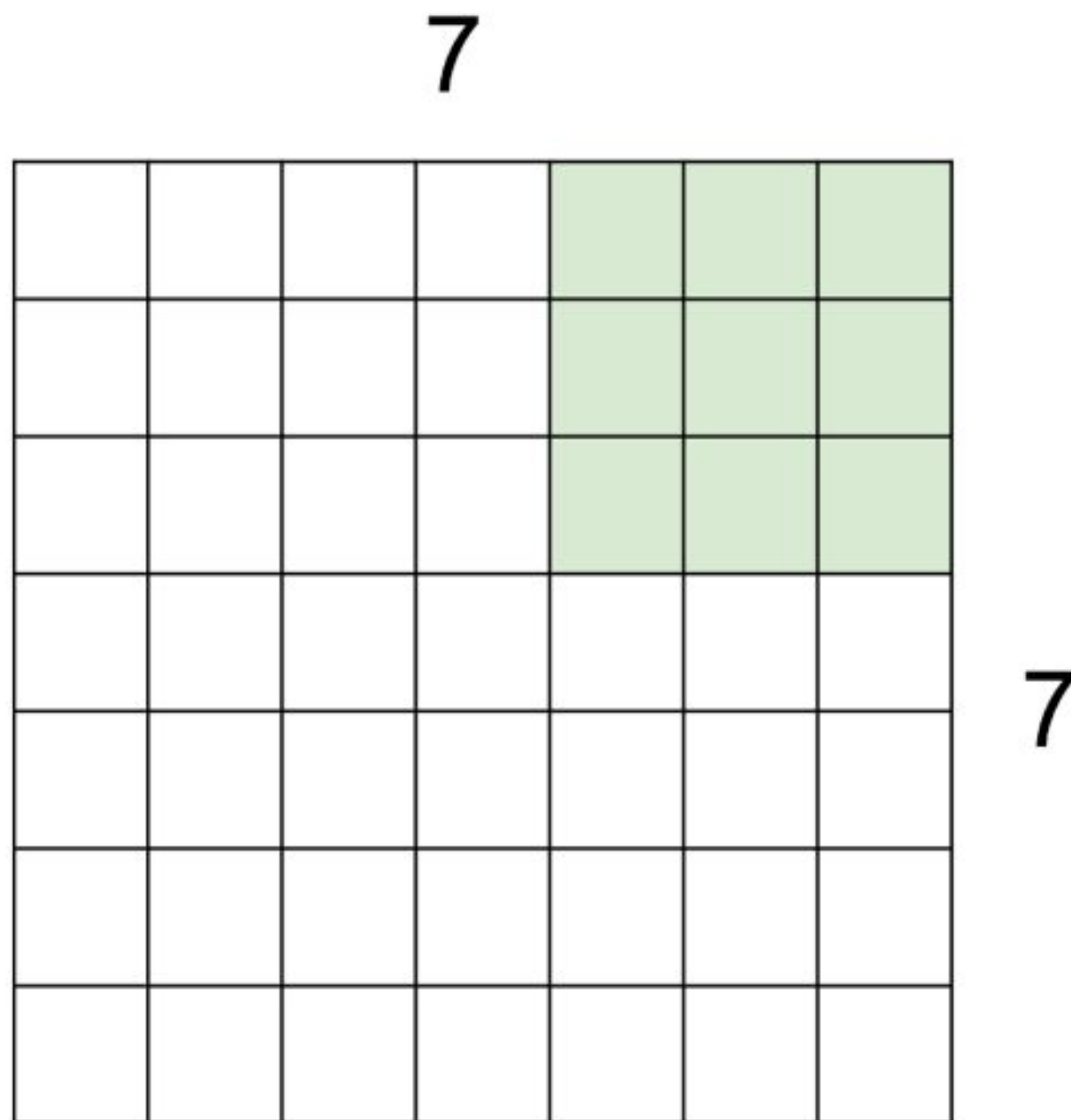
A closer look at spatial dimensions:



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

Convolutional Layer

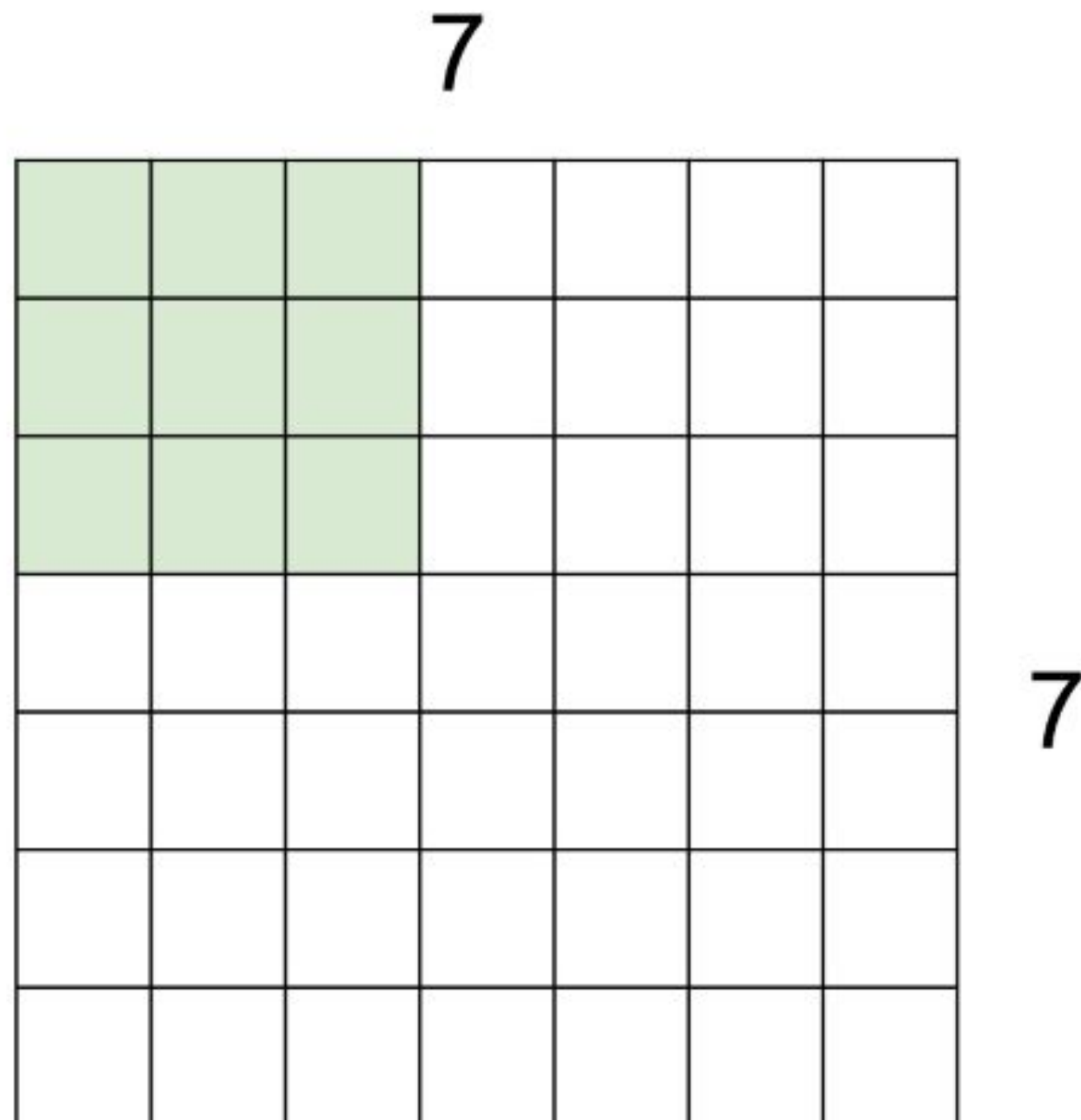
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Convolutional Layer

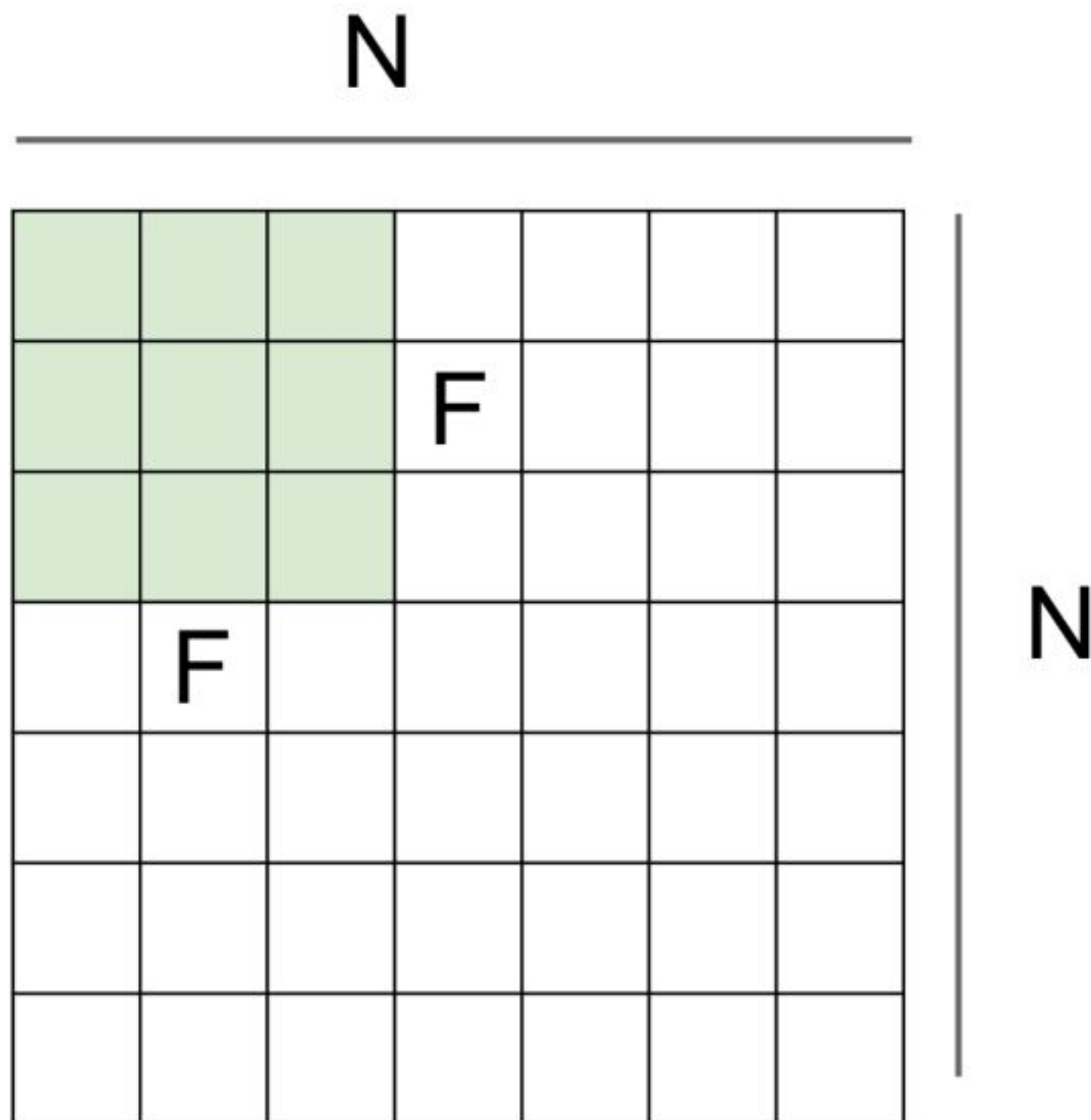
A closer look at spatial dimensions:



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

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolutional Layer



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 $\Rightarrow (7 - 3) / 3 + 1 = 2.33 : \backslash$

Convolutional Layer

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:)

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

Convolutional Layer

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!

Convolutional Layer

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

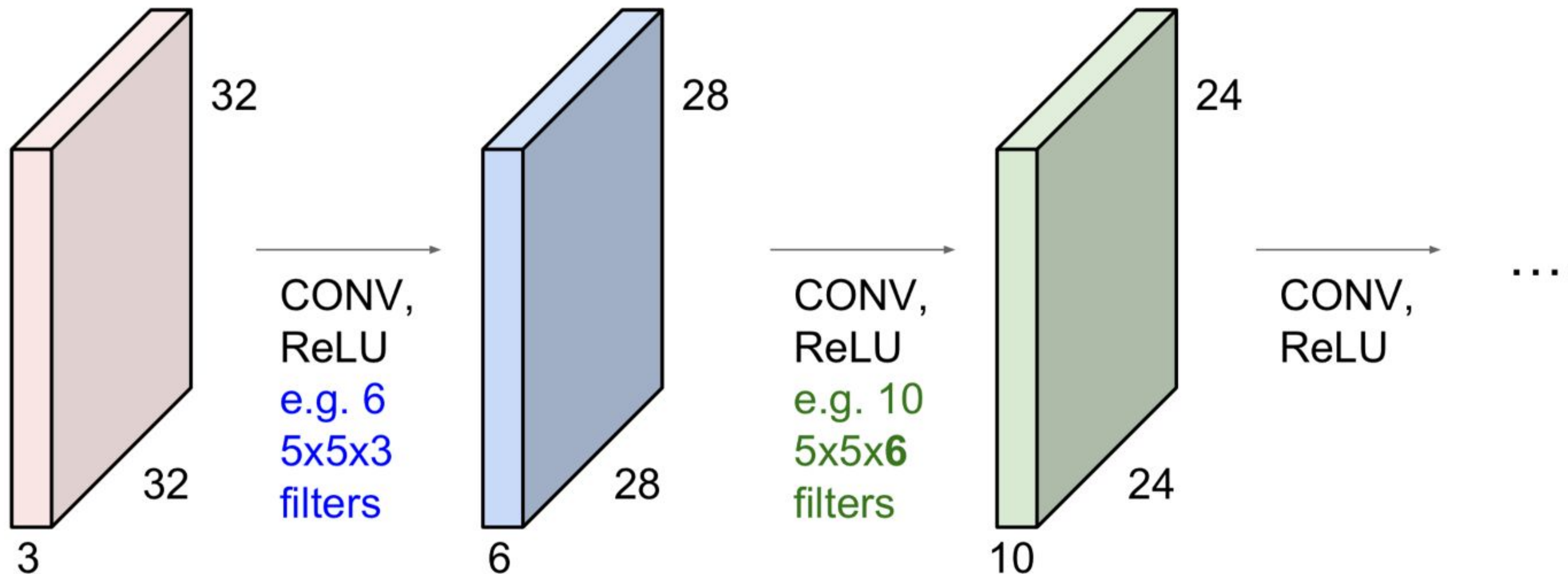
$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Convolutional Layer

Remember back to...

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

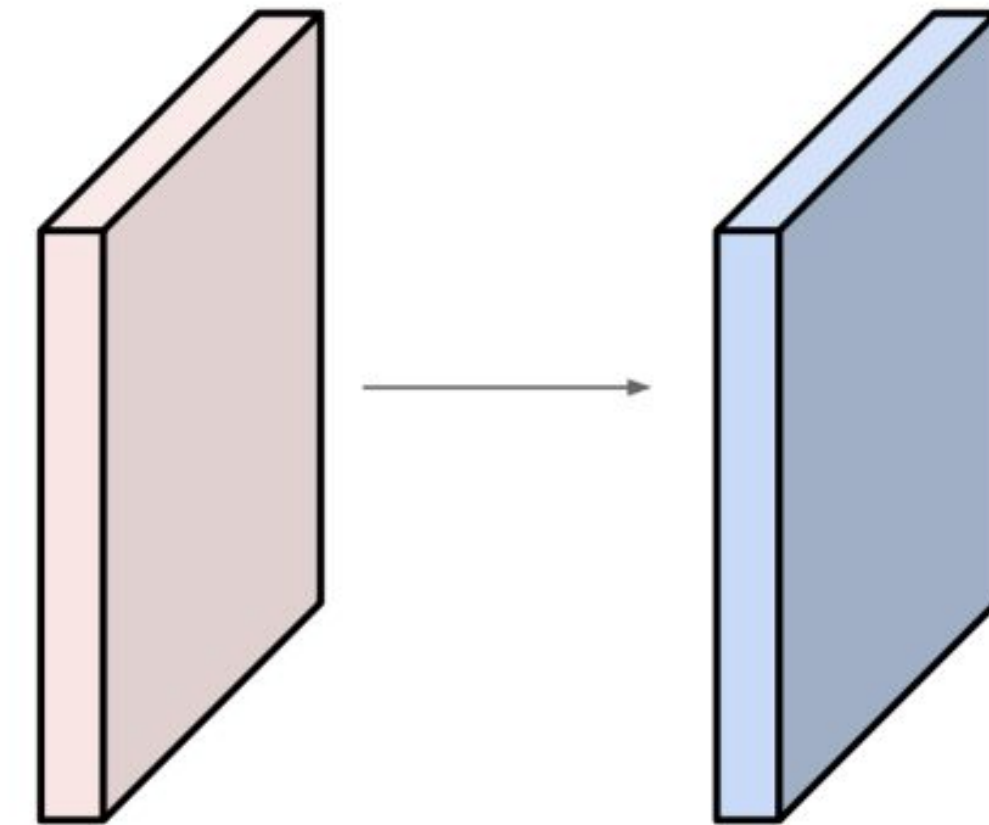


Convolutional Layer

Examples time:

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

Output volume size: ?

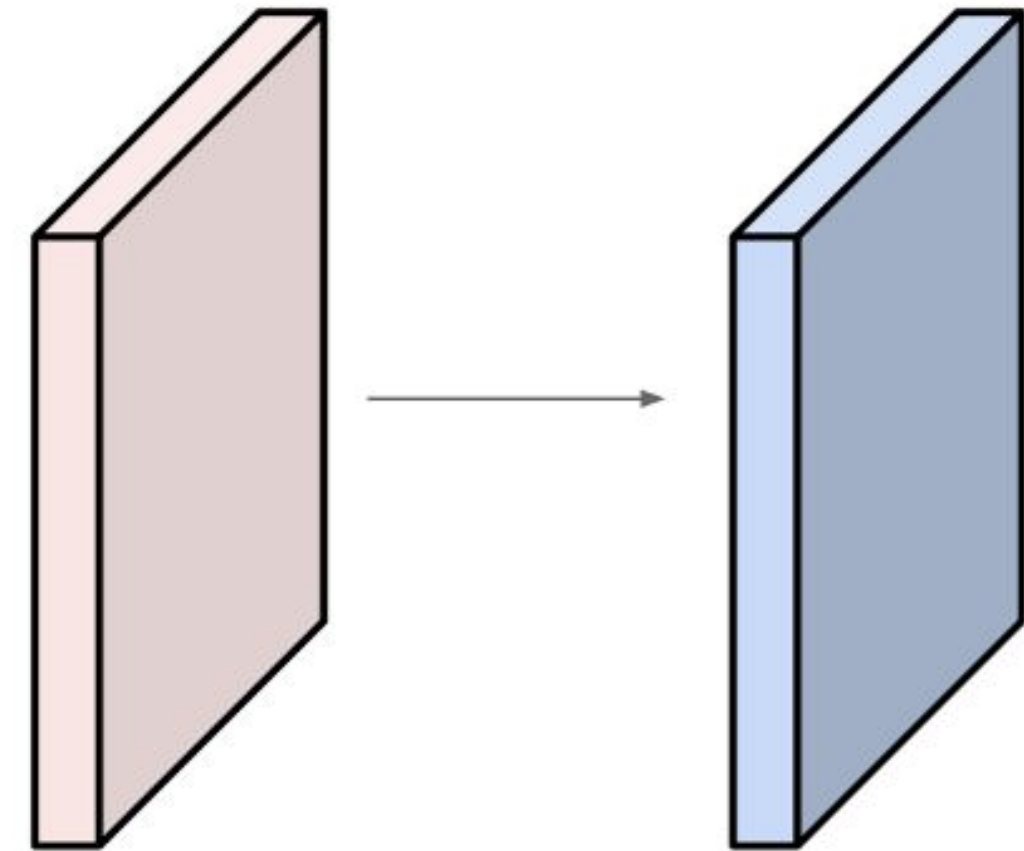


Convolutional Layer

Examples time:

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

Output volume size:
 $(32 + 2 * 2 - 5) / 1 + 1 = 32$ spatially, so
32x32x10

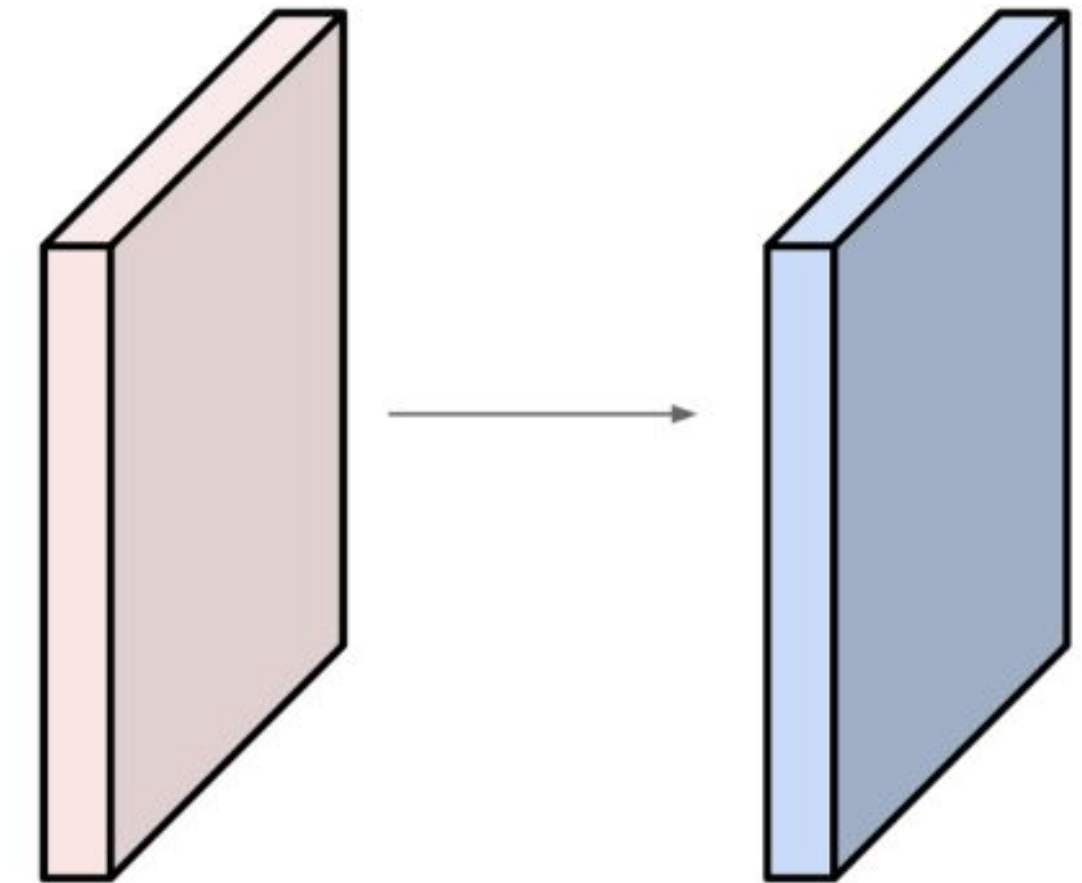


Convolutional Layer

Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2



Number of parameters in this layer?

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

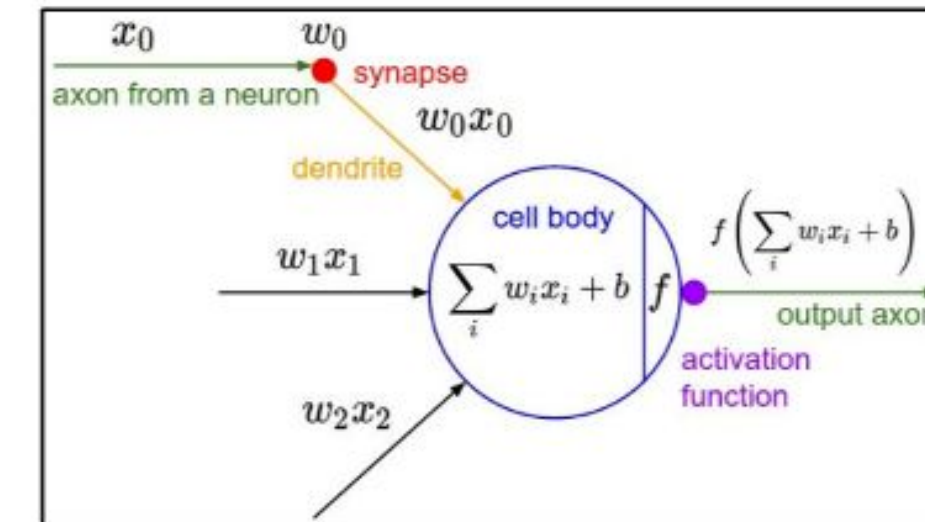
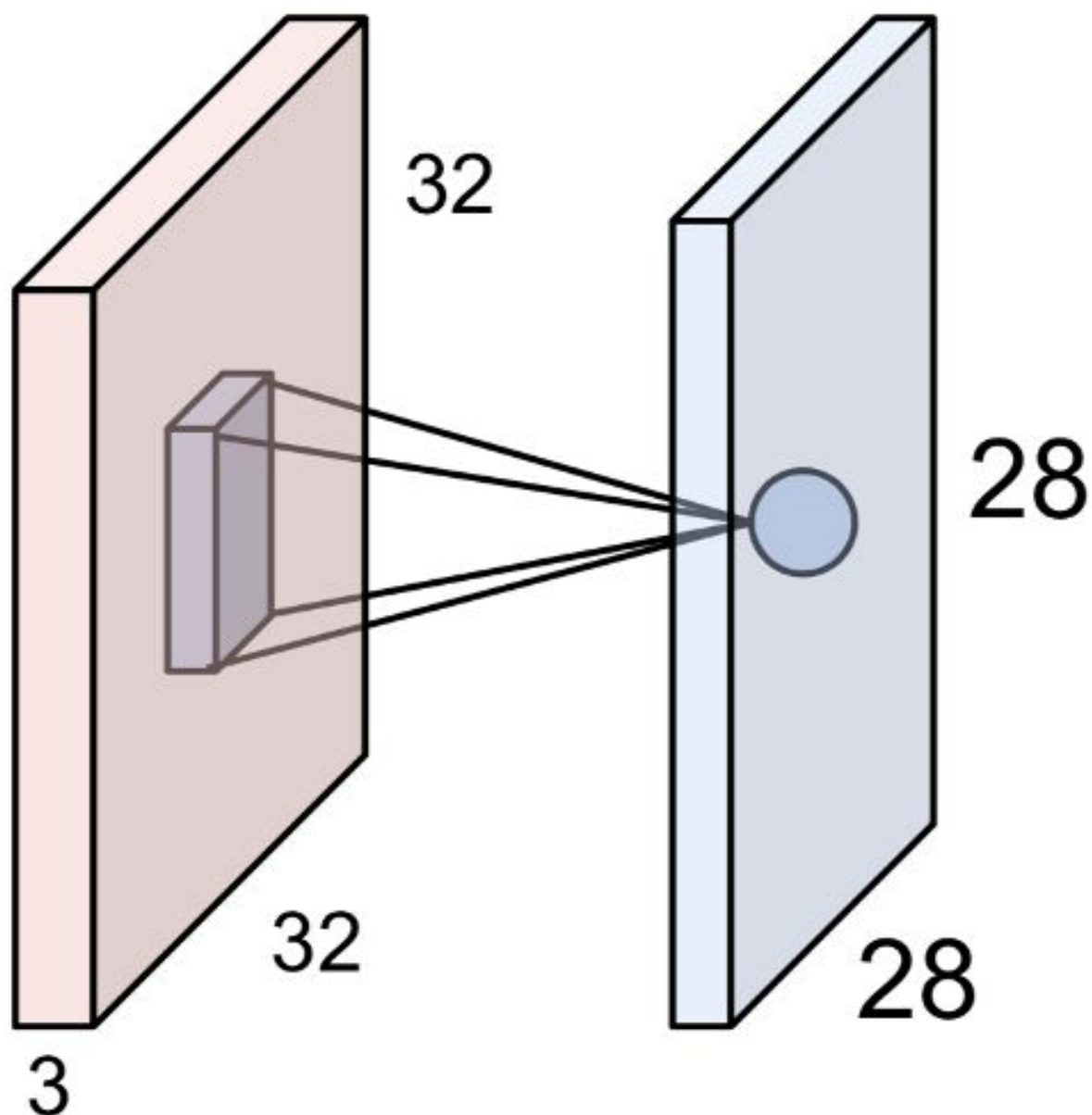
=> $76*10 = 760$

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)
 - $D_2 = K$

Convolutional Layer

The brain/neuron view of CONV Layer



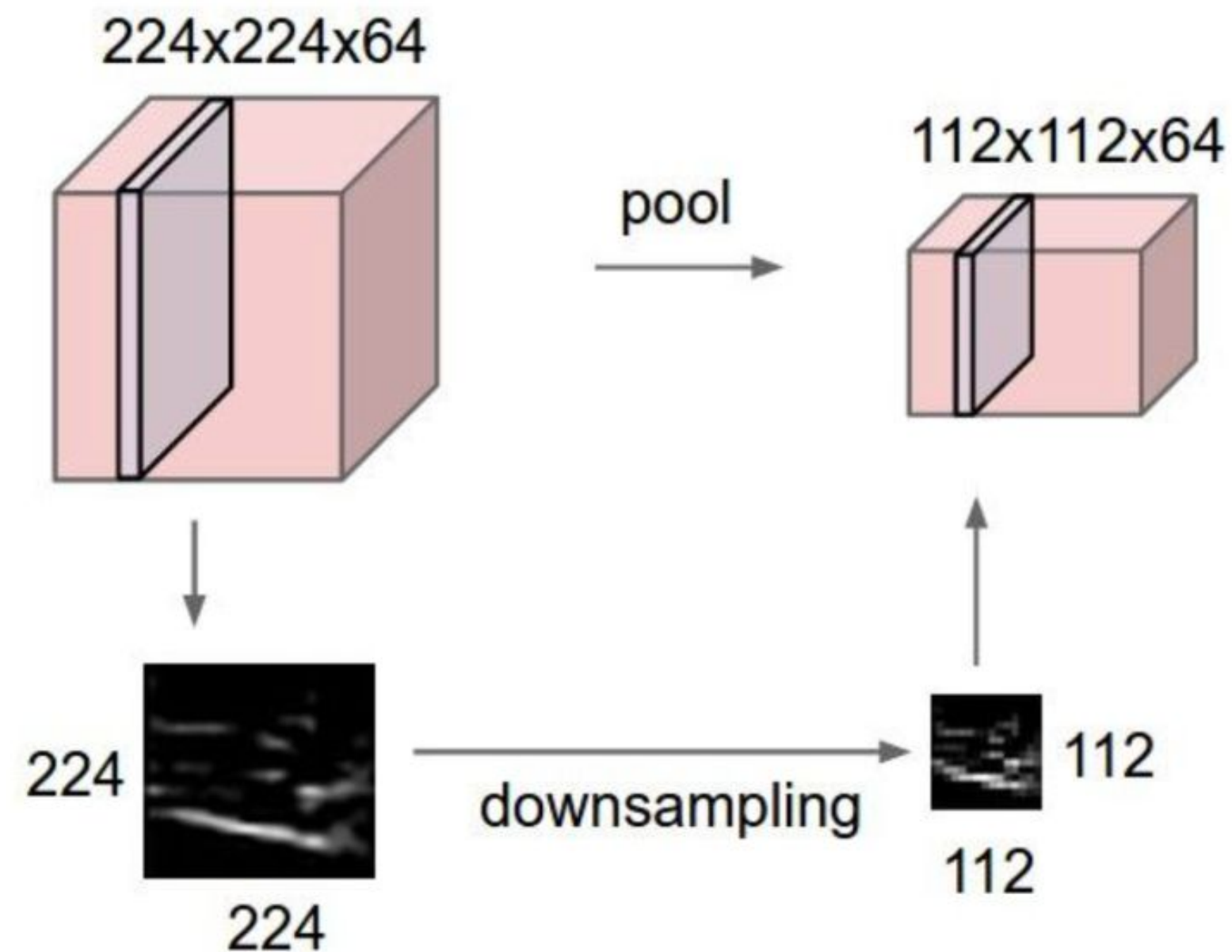
An activation map is a 28x28 sheet of neuron outputs:

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

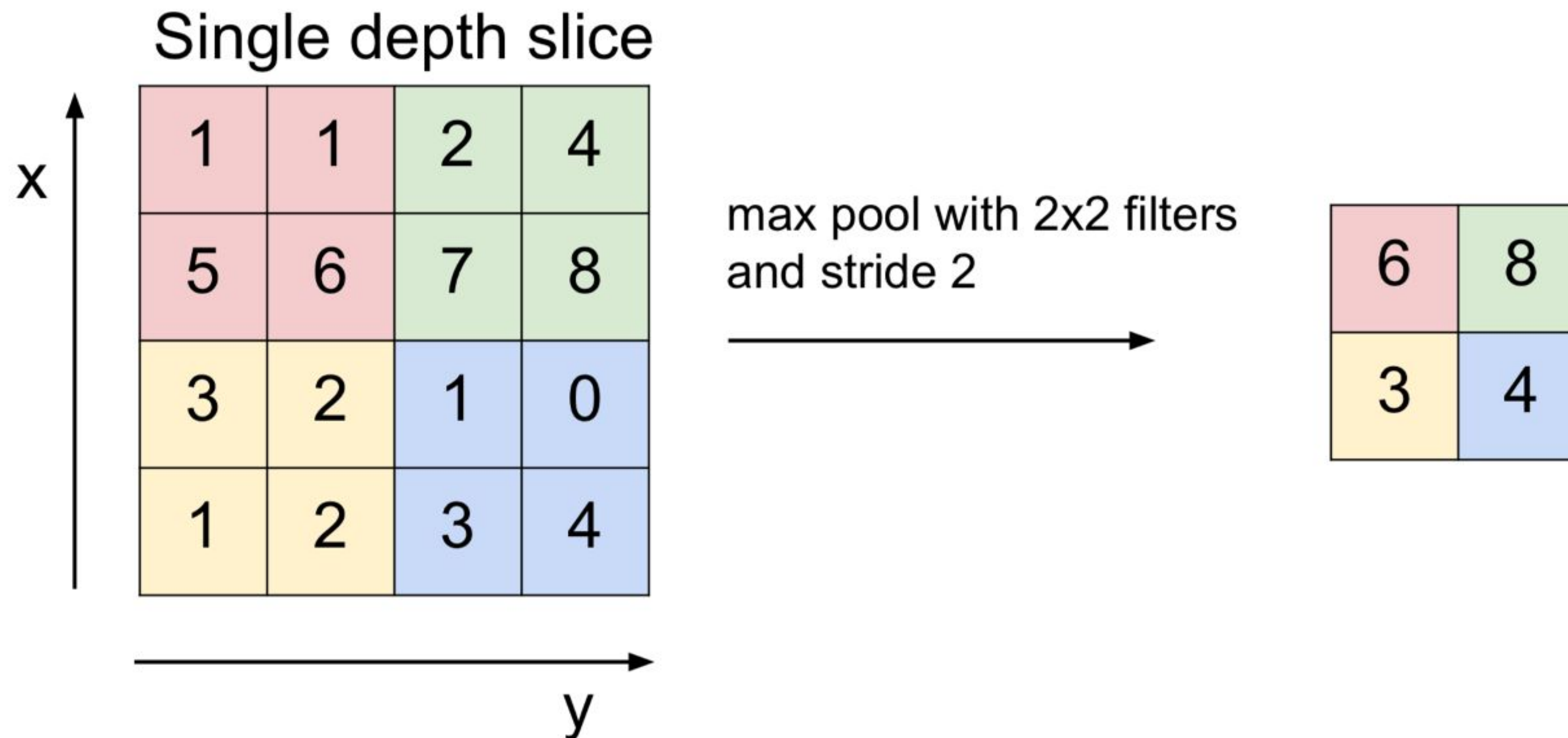
“5x5 filter” -> “5x5 receptive field for each neuron”

Pooling layer

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



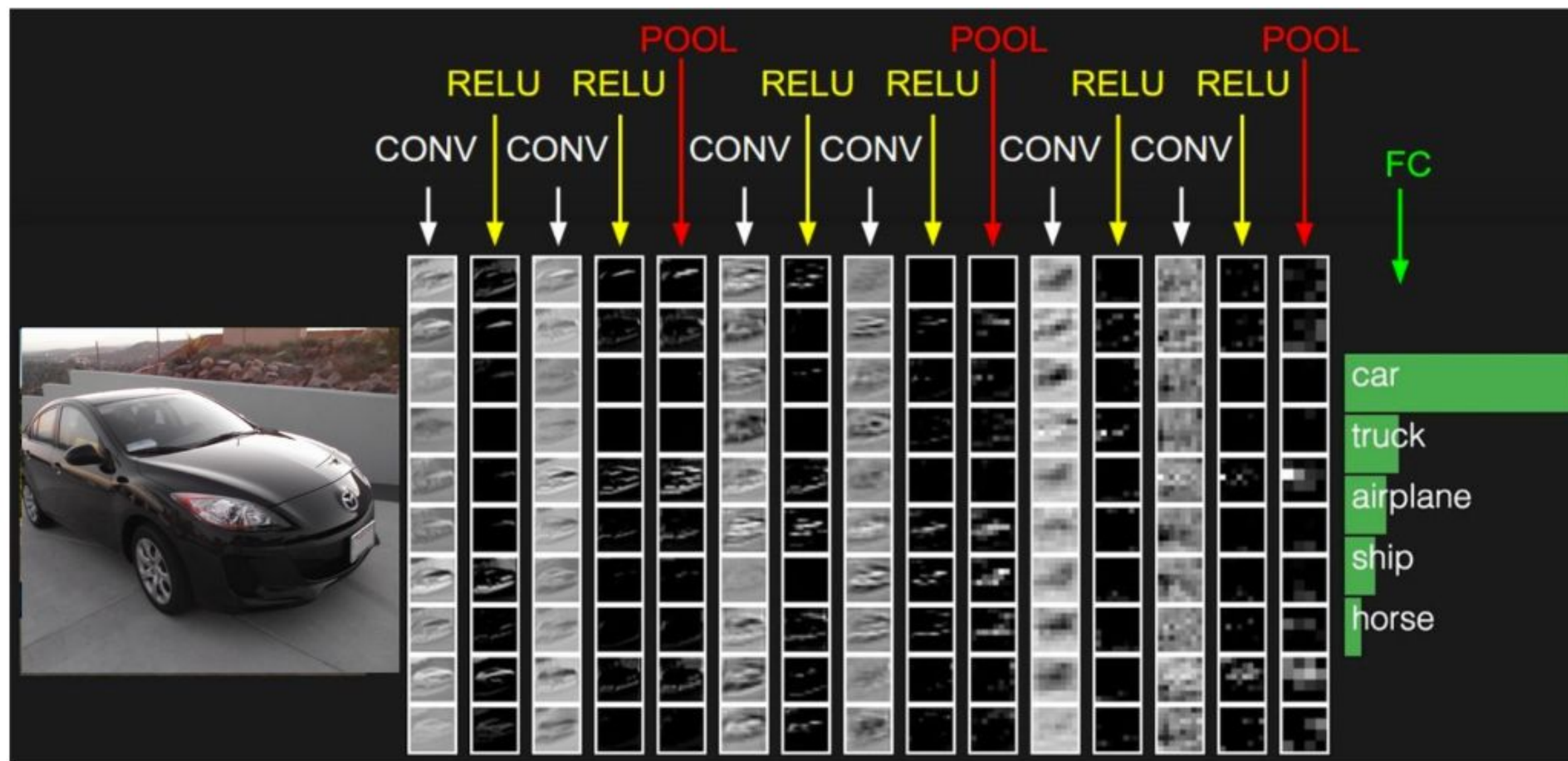
MAX POOLING



Convolutional Neural Network

Fully Connected Layer (FC layer)

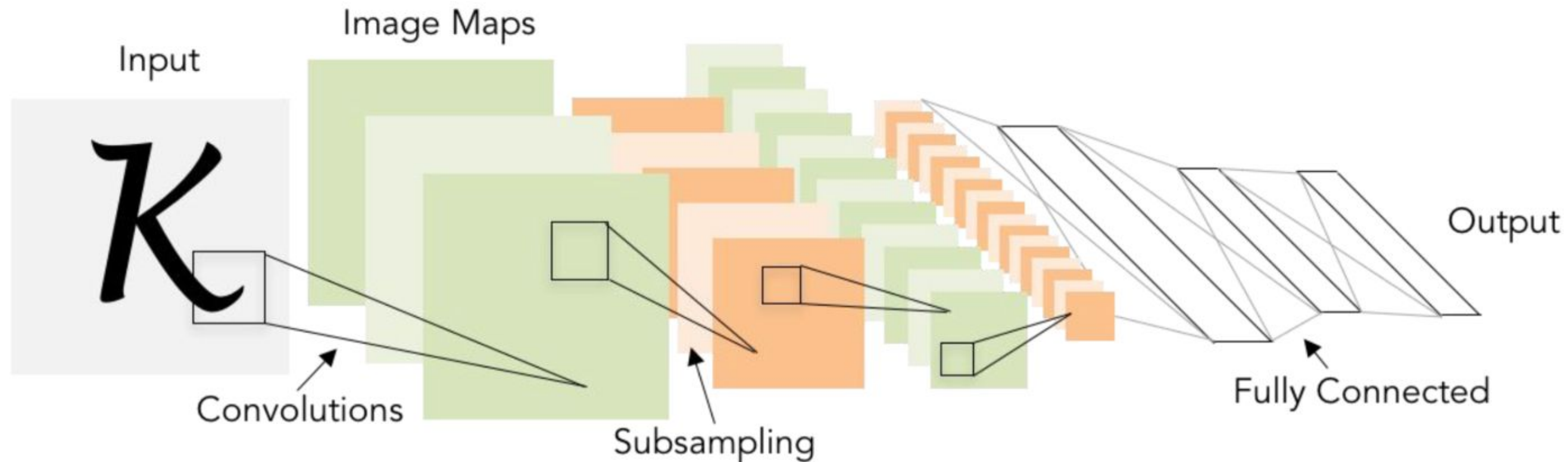
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
 $[(\text{CONV-RELU}) * N - \text{POOL?}] * M - (\text{FC-RELU}) * K, \text{SOFTMAX}$
where N is usually up to ~5, M is large, $0 \leq K \leq 2$.
 - but recent advances such as ResNet/GoogLeNet challenge this paradigm

Review: LeNet-5

[LeCun et al., 1998]




Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

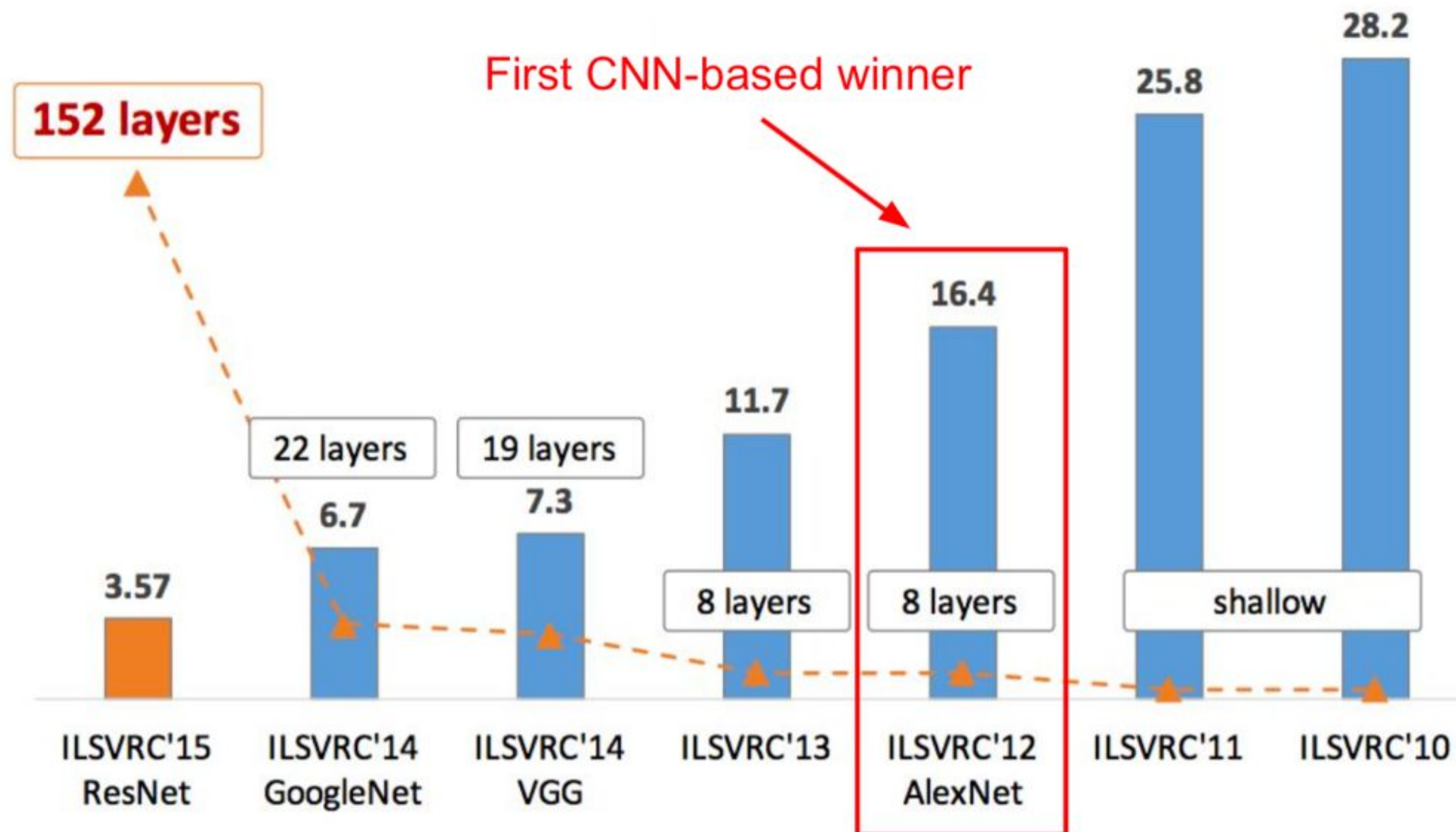
The Image Classification Challenge:
1,000 object classes
1,431,167 images



Output: Scale T-shirt <u>Steel drum</u> Drumstick Mud turtle	✓	Output: Scale T-shirt Giant panda Drumstick Mud turtle	✗
--	---	--	---

Russakovsky et al. arXiv, 2014

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

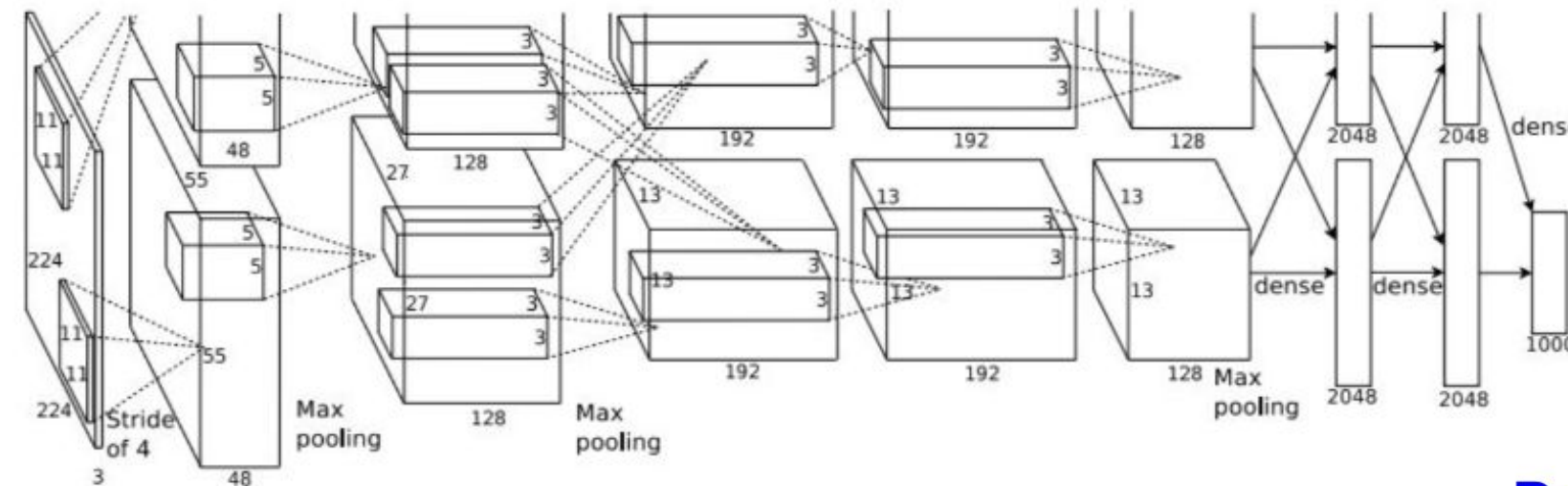
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)

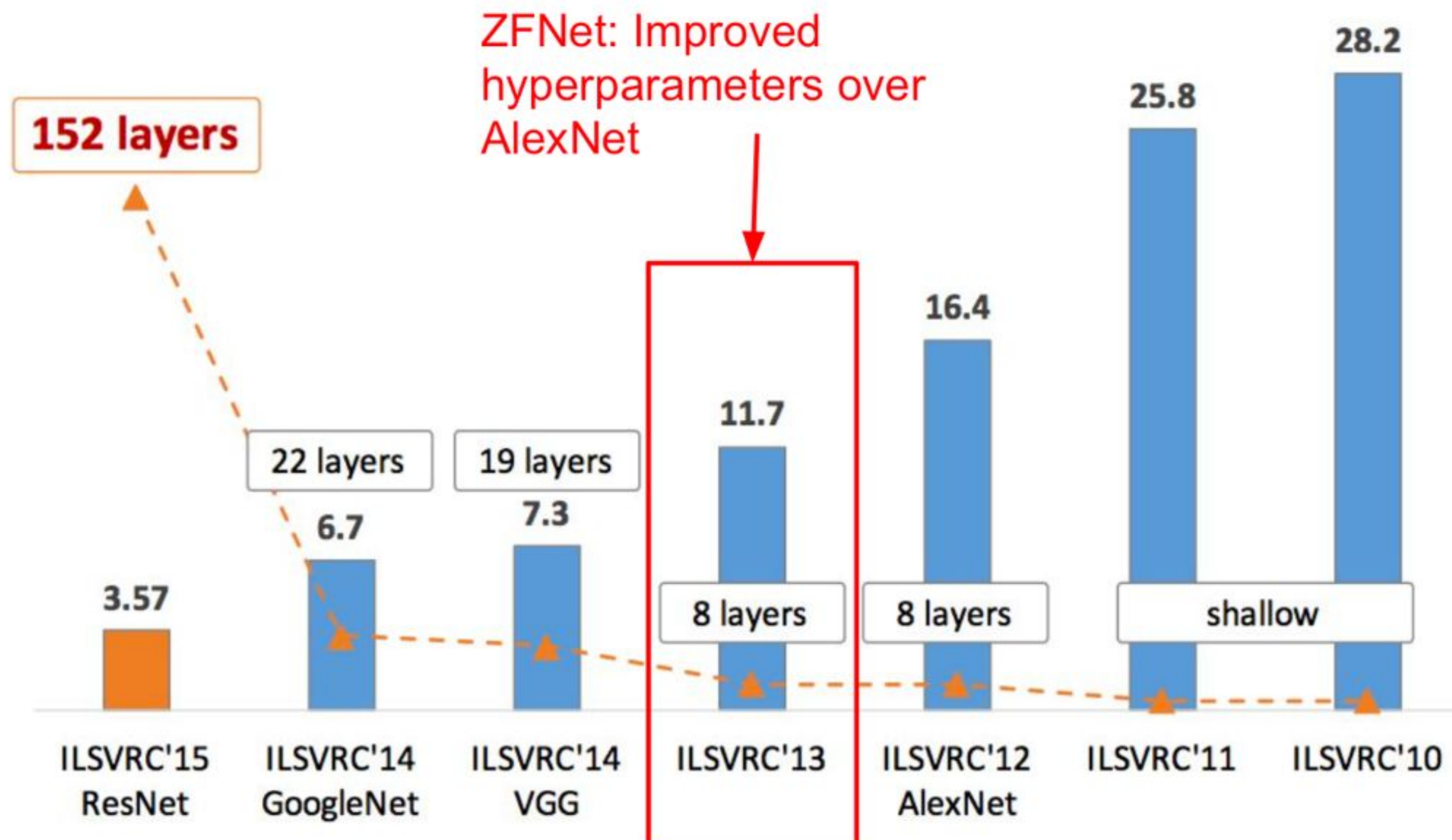


Details/Retrospectives:

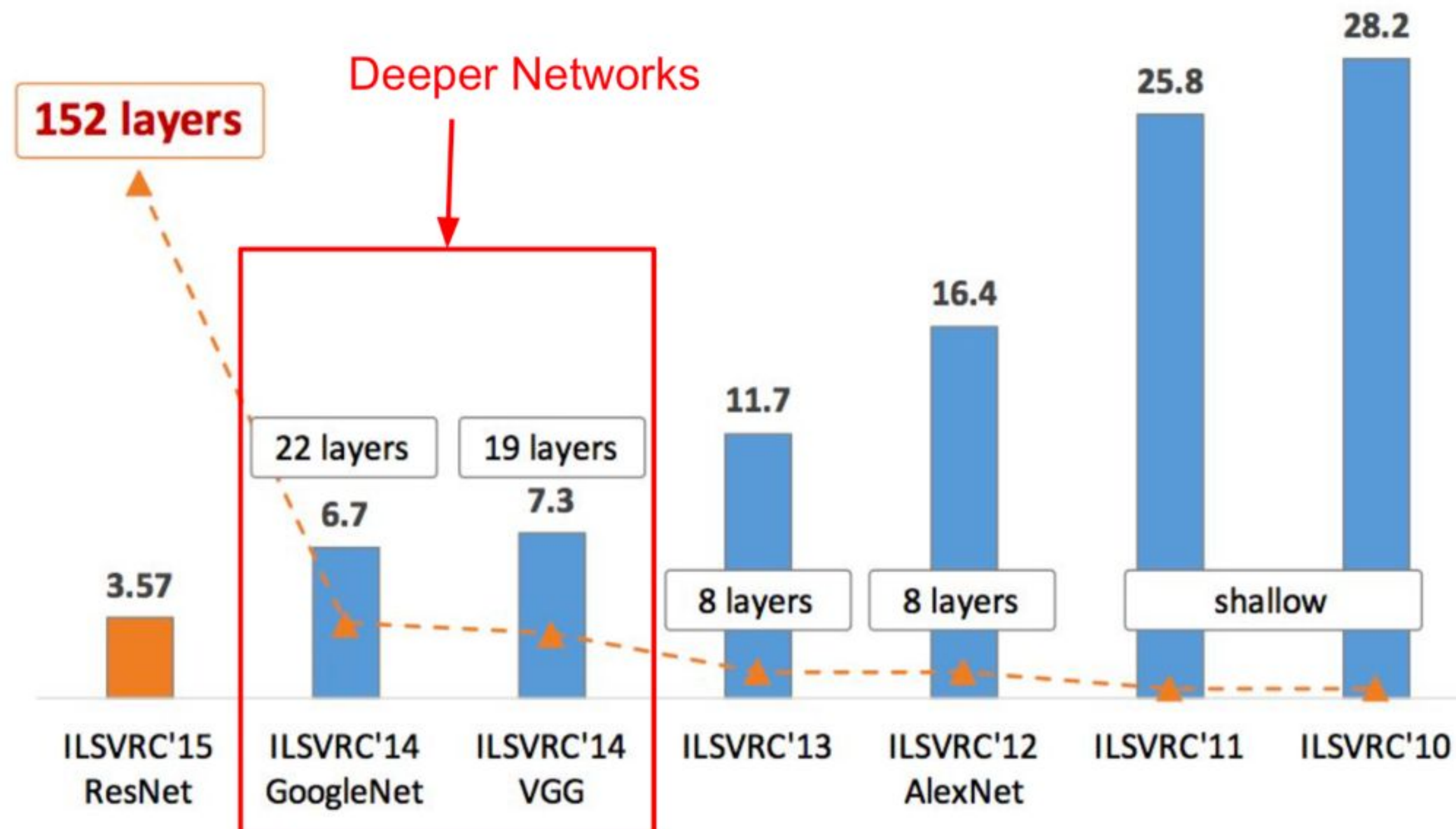
- first use of ReLU
- used LRN layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

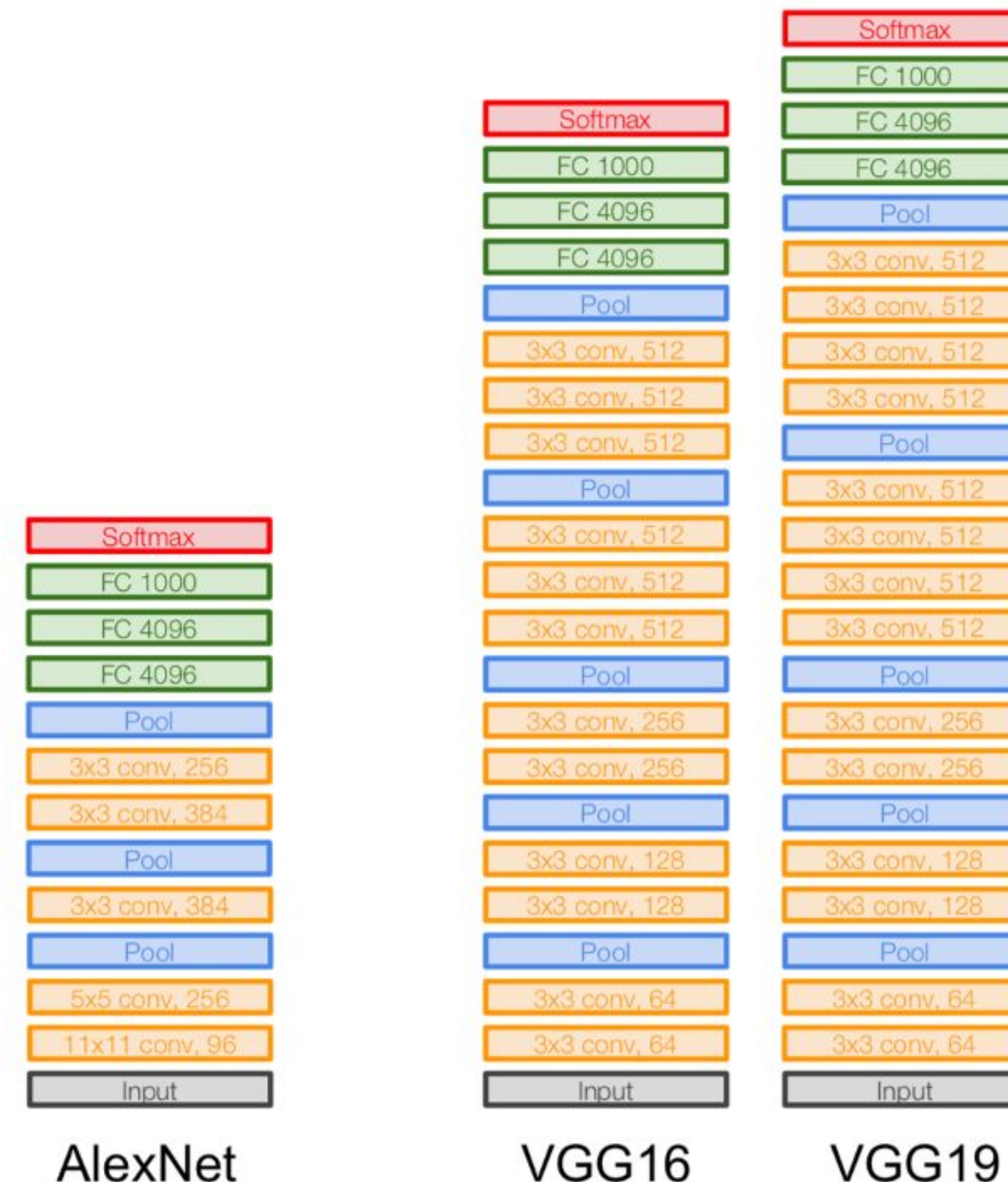
8 layers (AlexNet)

-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)

-> 7.3% top 5 error in ILSVRC'14



VGGNet

INPUT: [224x224x3] memory: $224*224*3=150\text{K}$ params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*3)*64 = 1,728$

CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*64)*64 = 36,864$

POOL2: [112x112x64] memory: $112*112*64=800\text{K}$ params: 0

CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*64)*128 = 73,728$

CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*128)*128 = 147,456$

POOL2: [56x56x128] memory: $56*56*128=400\text{K}$ params: 0

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*128)*256 = 294,912$

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$

CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$

POOL2: [28x28x256] memory: $28*28*256=200\text{K}$ params: 0

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*256)*512 = 1,179,648$

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [14x14x512] memory: $14*14*512=100\text{K}$ params: 0

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

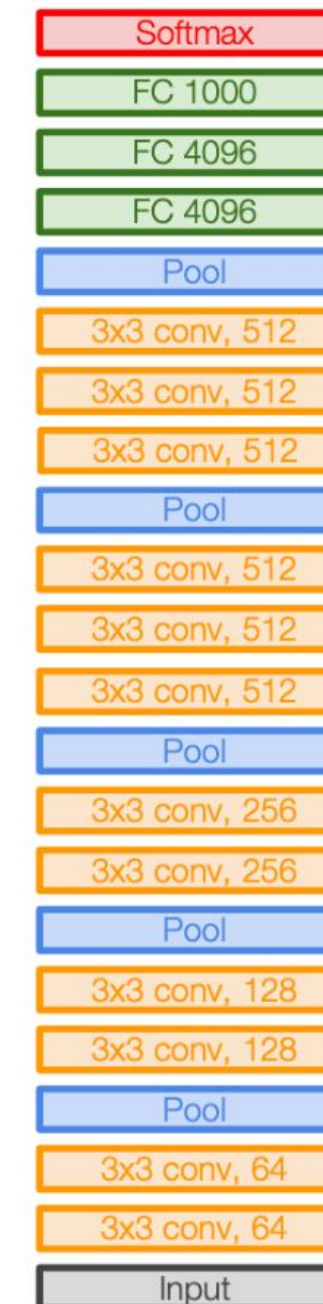
CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$

POOL2: [7x7x512] memory: $7*7*512=25\text{K}$ params: 0

FC: [1x1x4096] memory: 4096 params: $7*7*512*4096 = 102,760,448$

FC: [1x1x4096] memory: 4096 params: $4096*4096 = 16,777,216$

FC: [1x1x1000] memory: 1000 params: $4096*1000 = 4,096,000$



VGG16

TOTAL memory: $24M * 4 \text{ bytes} \approx 96MB$ / image (for a forward pass)

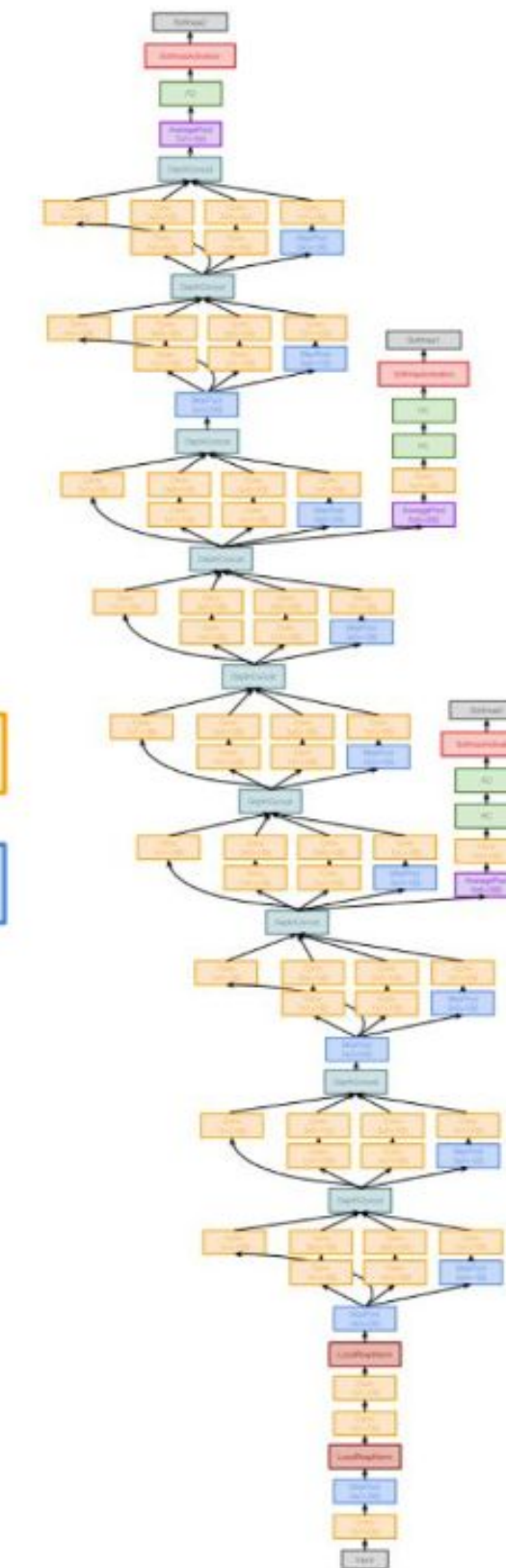
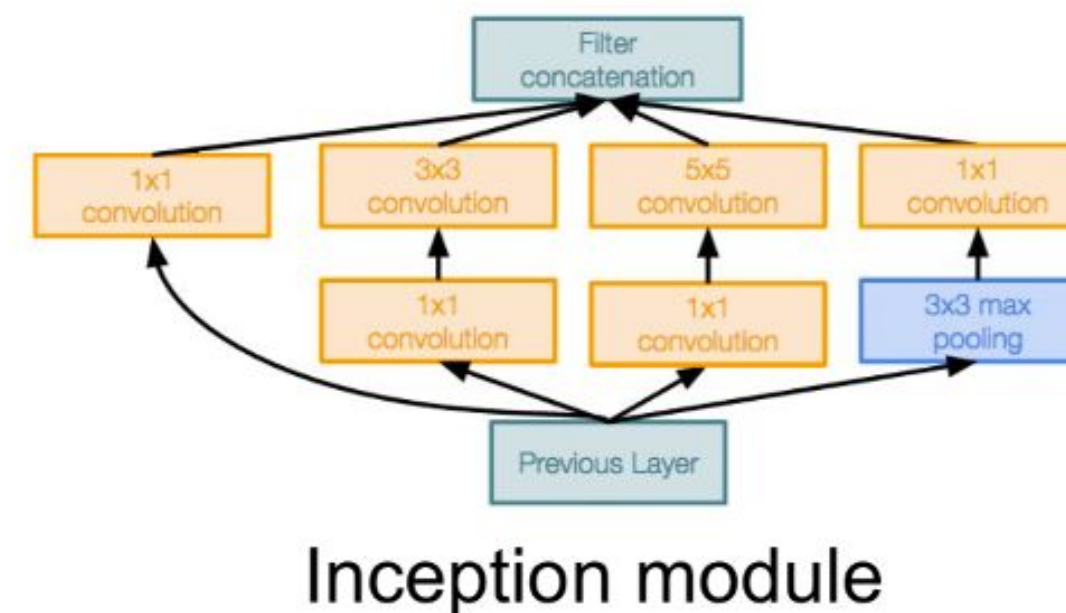
TOTAL params: 138M parameters

Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

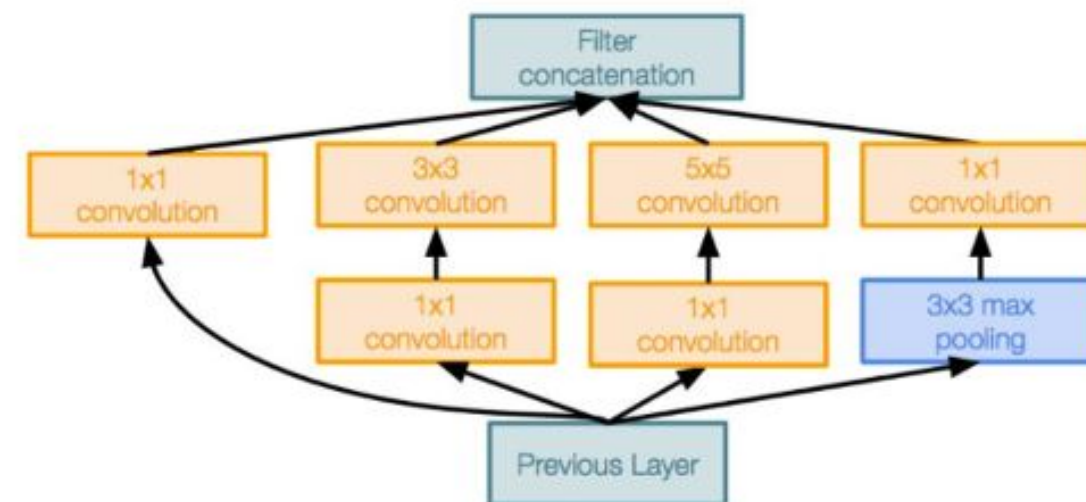
- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
12x less than AlexNet
27x less than VGG-16
- Efficient “Inception” module
- No FC layers



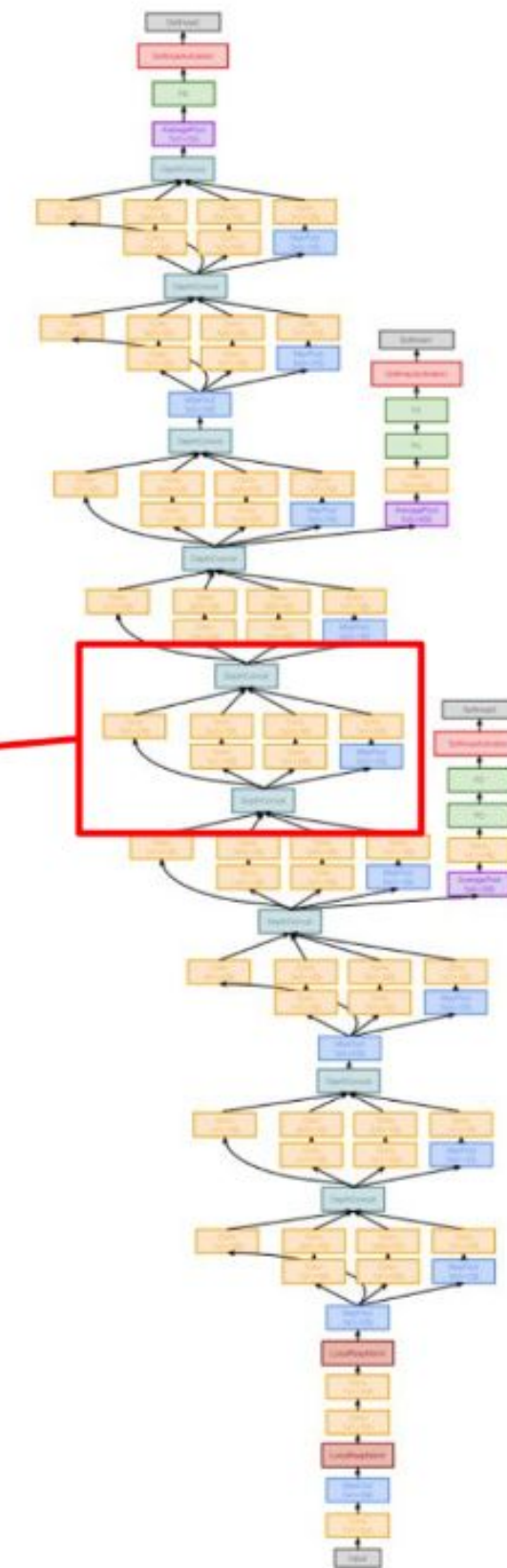
Case Study: GoogLeNet

[Szegedy et al., 2014]

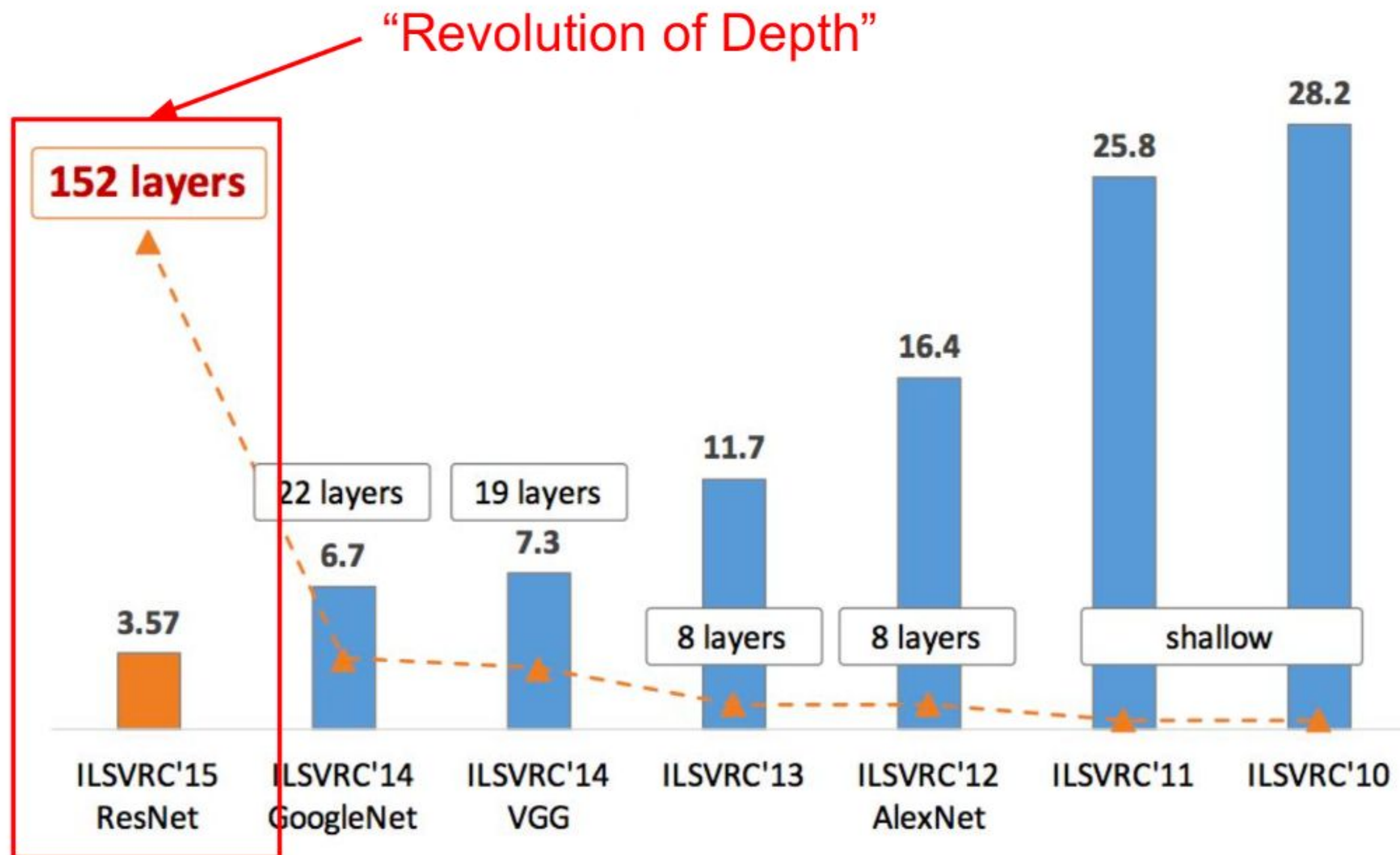
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other



Inception module



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



CNN architectures

Training ResNet in practice:

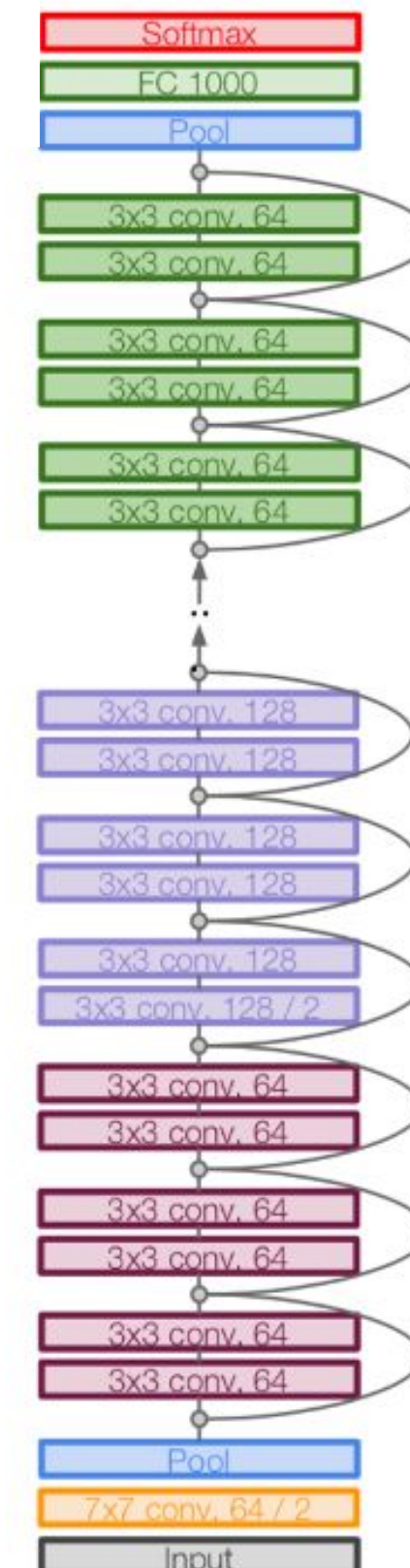
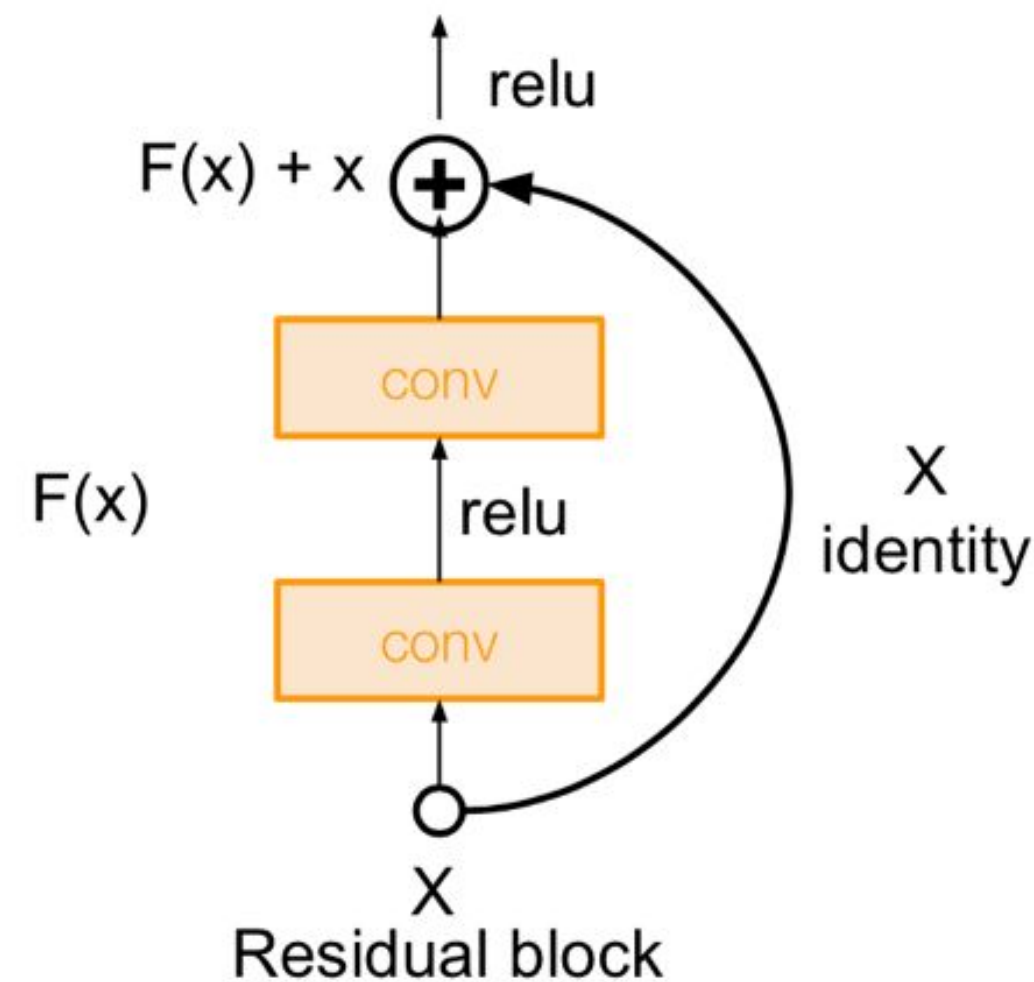
- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of $1e-5$
- No dropout used

Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



AlexNet showed that you can use CNNs to train Computer Vision models.

VGG shows that bigger networks work better

ResNet showed us how to train extremely deep networks

- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to other topics:

- Efficient Networks: **MobileNet, ShuffleNet**
- **Neural Architecture Search** can now automate architecture design

Transfer Learning

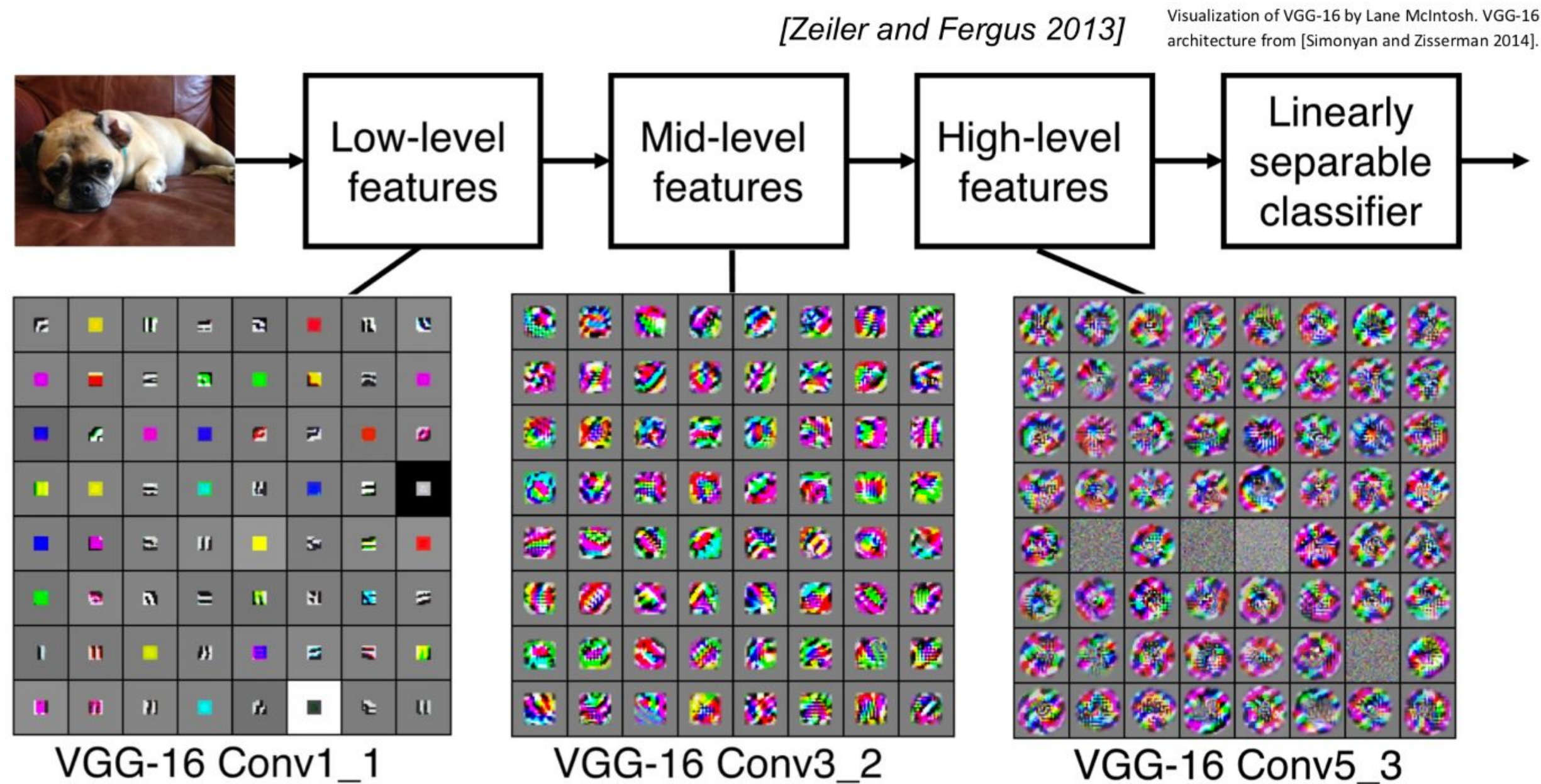


Should we train from scratch such big networks for each new Computer Vision problem ?

Transfer Learning

Should we train from scratch such big networks for each new Computer Vision problem ?

→ No: use transfer learning (fine tuning)

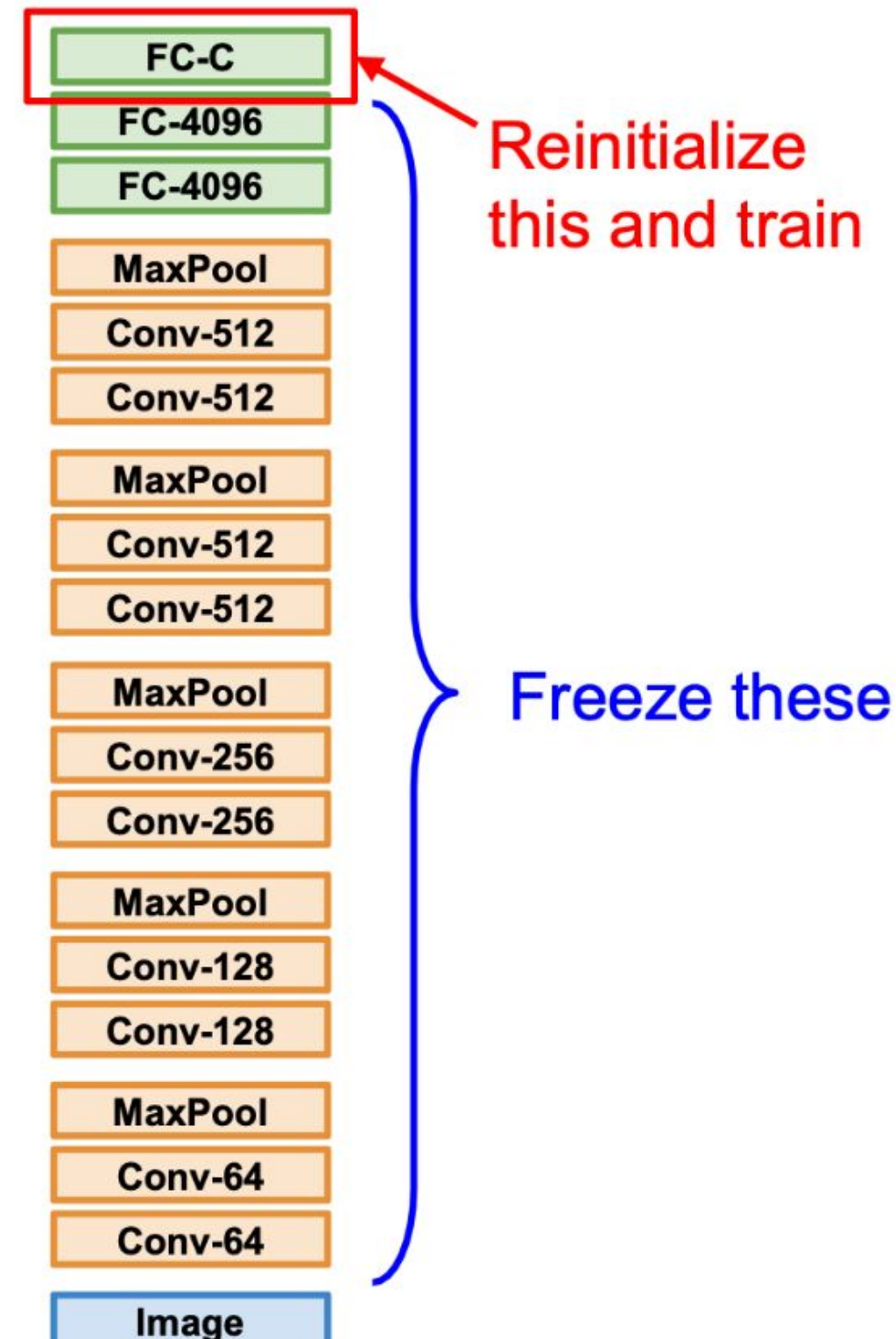


Transfer Learning

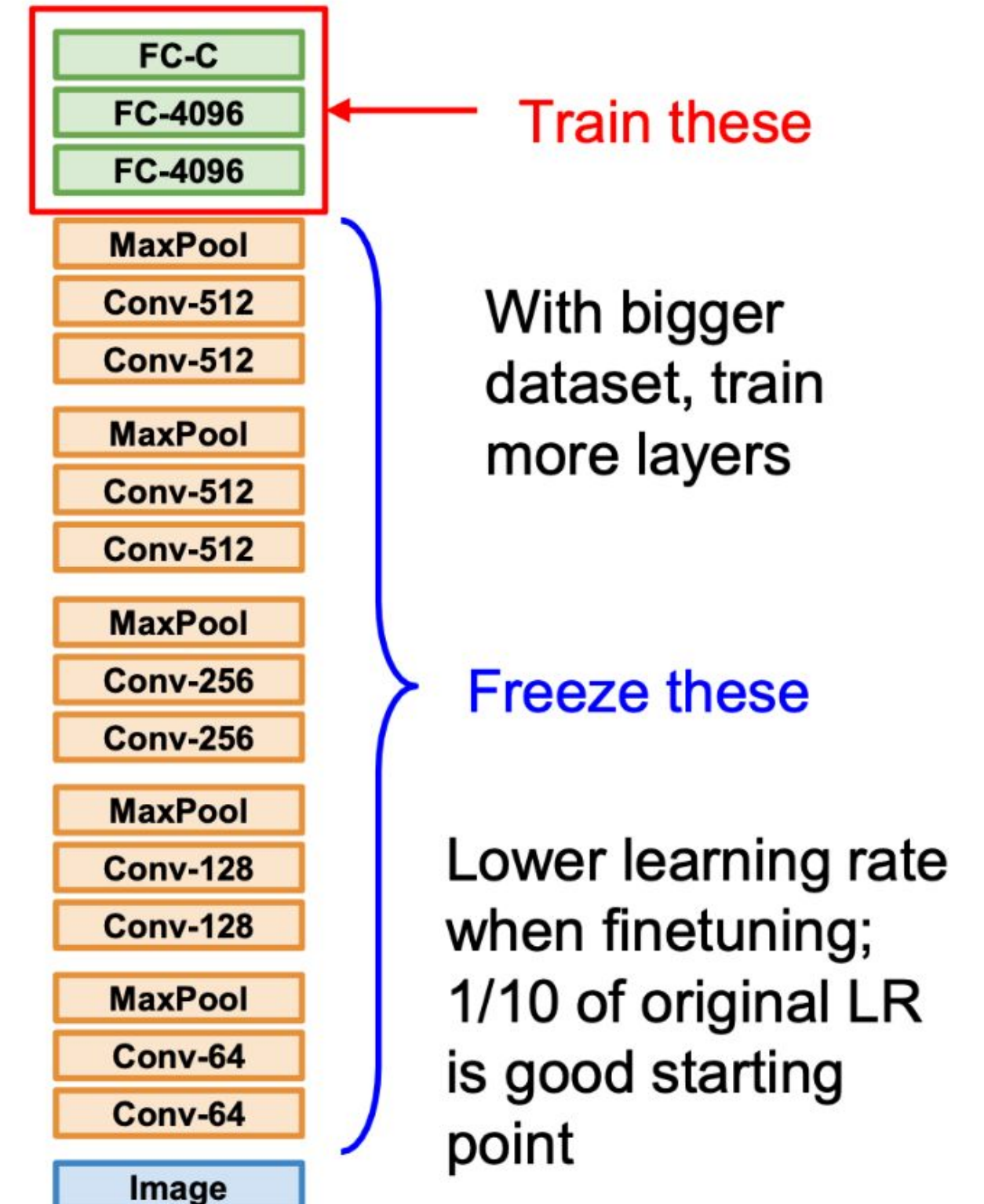
1. Train on Imagenet



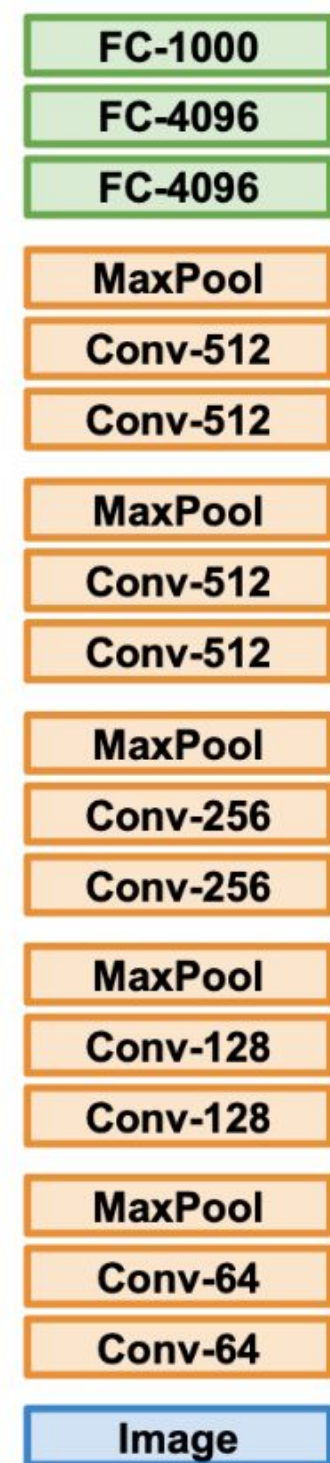
2. Small Dataset (C classes)



3. Bigger dataset



Transfer Learning



More specific

More generic

	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers or start from scratch!

Have some dataset of interest but it has $< \sim 1\text{M}$ images?

1. Find a very large dataset that has similar data, train a big model there
2. Transfer learn to your dataset

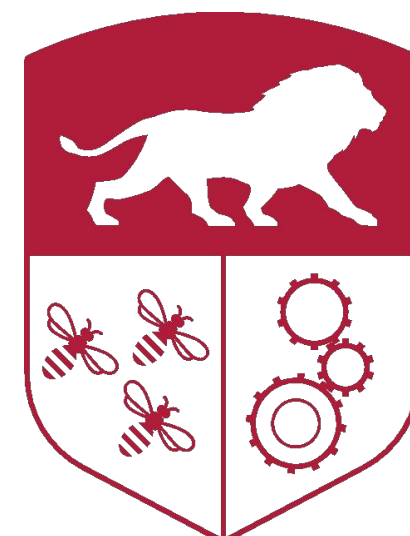
Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

PyTorch models: <https://pytorch.org/vision/0.9/models.html>

PyTorch datasets: <https://pytorch.org/vision/main/datasets.html>

Some useful references

- CS231n: Convolutional Neural Networks for Visual Recognition (<http://cs231n.stanford.edu>)
- Fidle - Deep Learning Introduction (<https://www.fidle.cnrs.fr/w3/>)
- Neural Networks and Deep Learning (<http://neuralnetworksanddeeplearning.com>)
- Deep Learning (<http://www.deeplearningbook.org>)
- PyTorch (<http://pytorch.org>)



**CENTRALE
LYON**

36, avenue Guy de Collongue 69130 Écully
www.ec-lyon.fr | [@centralelyon](https://twitter.com/centralelyon)