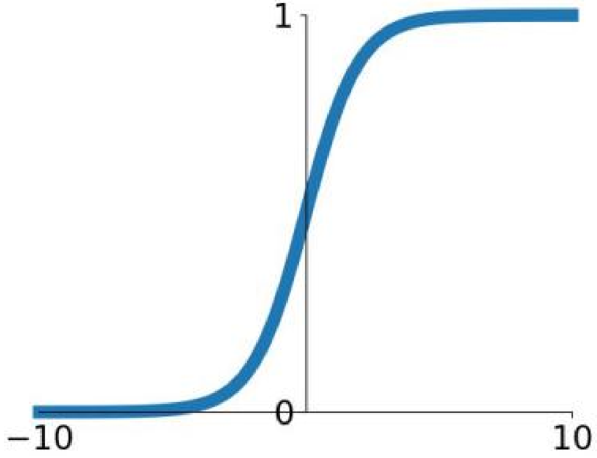
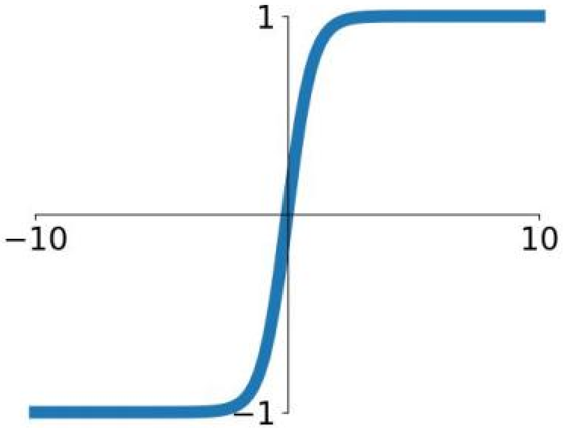
Deep Learning – II

Activation Functions



Sigmoid

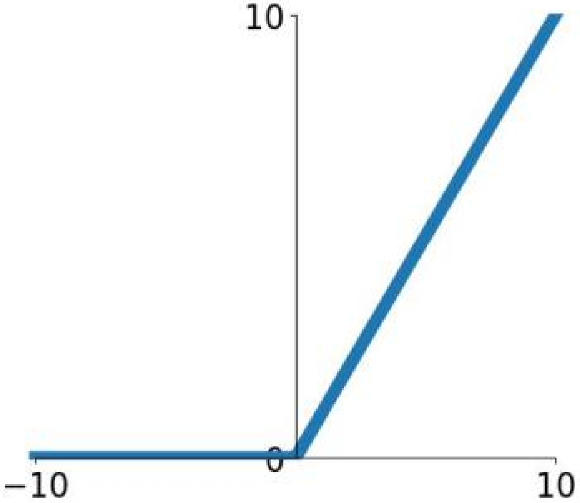
$f(x) = 1/(1+e^{-x})$



tanh

$f(x) = tanh(x)$

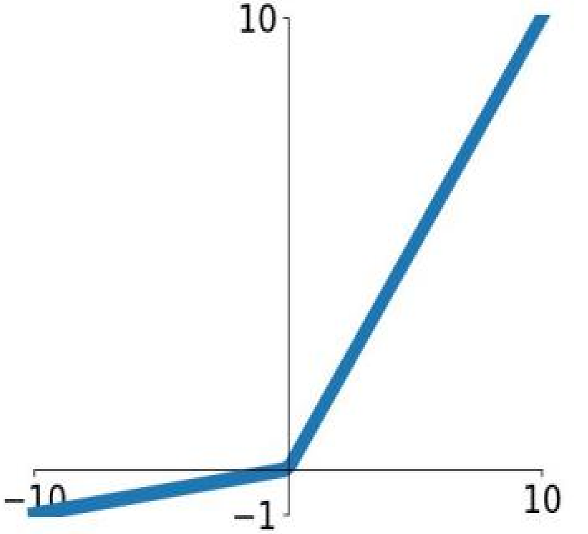
$f(x) = tanh(x)$



ReLU

$f(x) = max(0, x)$

$f(x) = max(0, x)$



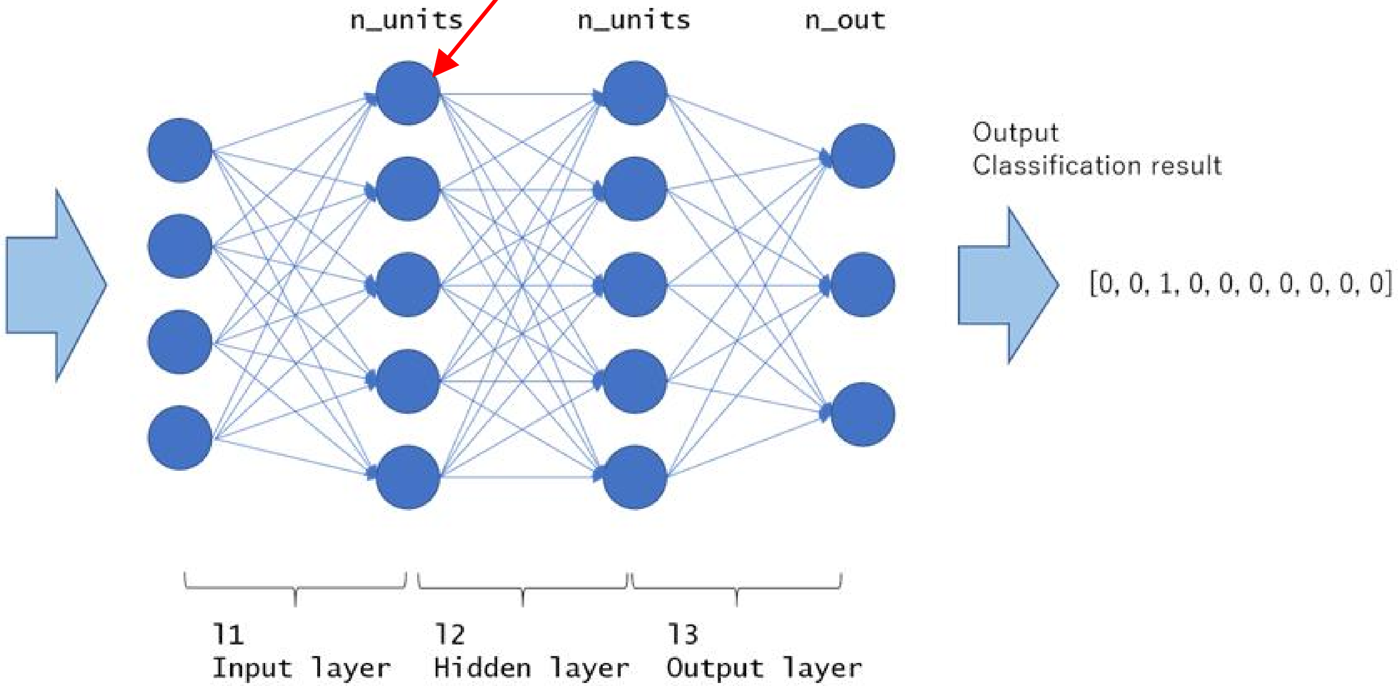
Leaky ReLU

$f(x) = max(0.01x, x)$

$f(x) = max(0.01x, x)$

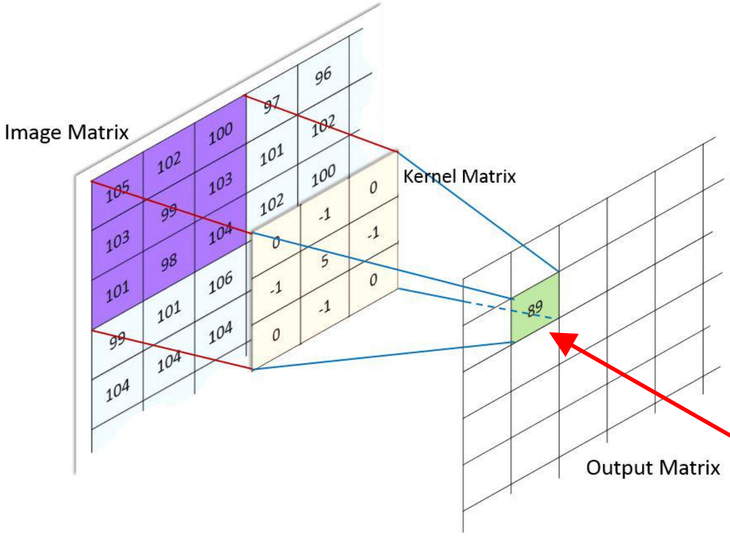
500x500 pixels

We need 250k number of weights/parameters to map the input to a single neuron. Too expensive!



Convolutional Neural Network

Depending on values, a kernel can cause a wide range of effects



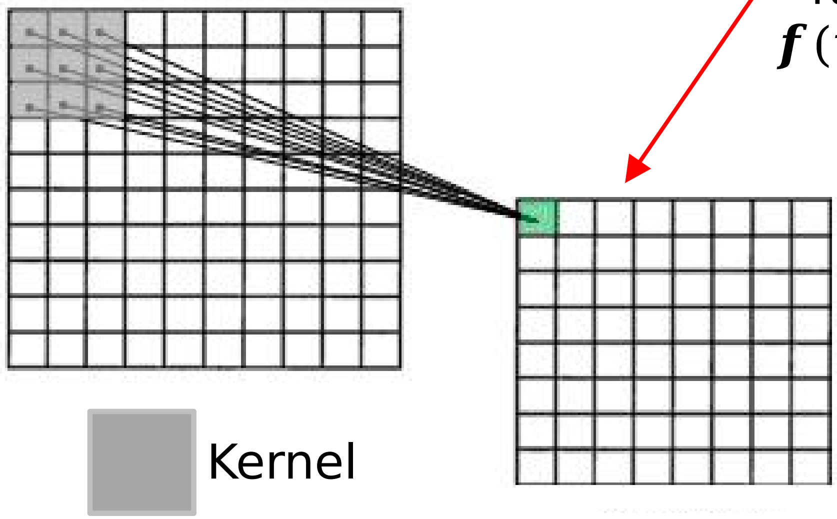
dot product

followed by activation function f

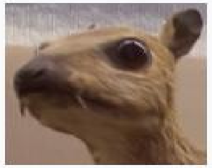
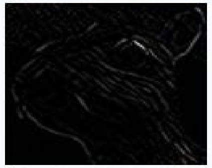
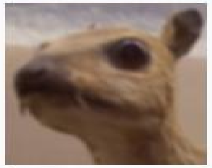
$f(\boldsymbol{w} * x + b)$

convolution

Image matrix



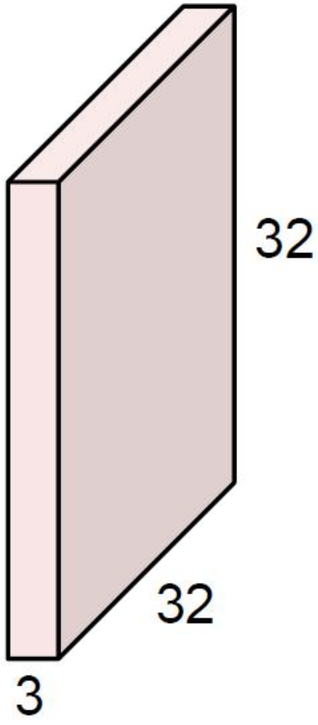
Kernel

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

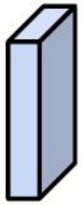
Convolution Layer

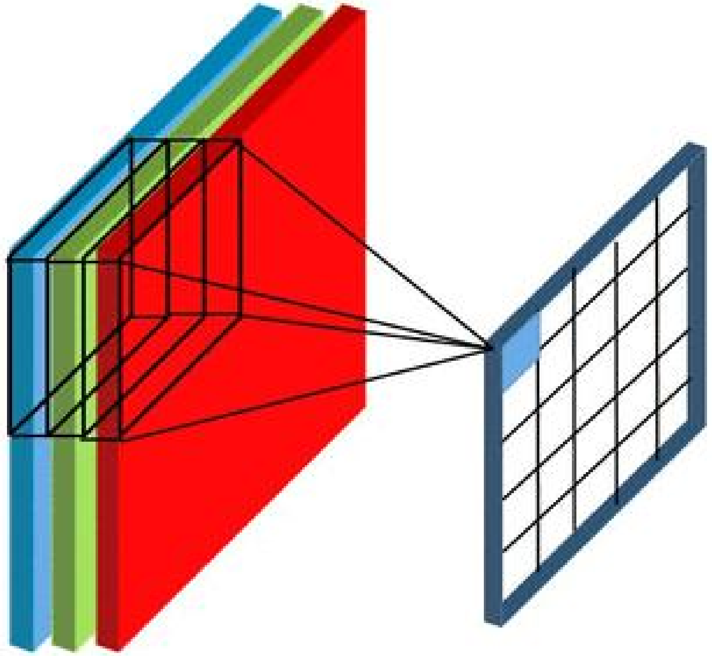
Filters always
depth of the in

32x32x3 image



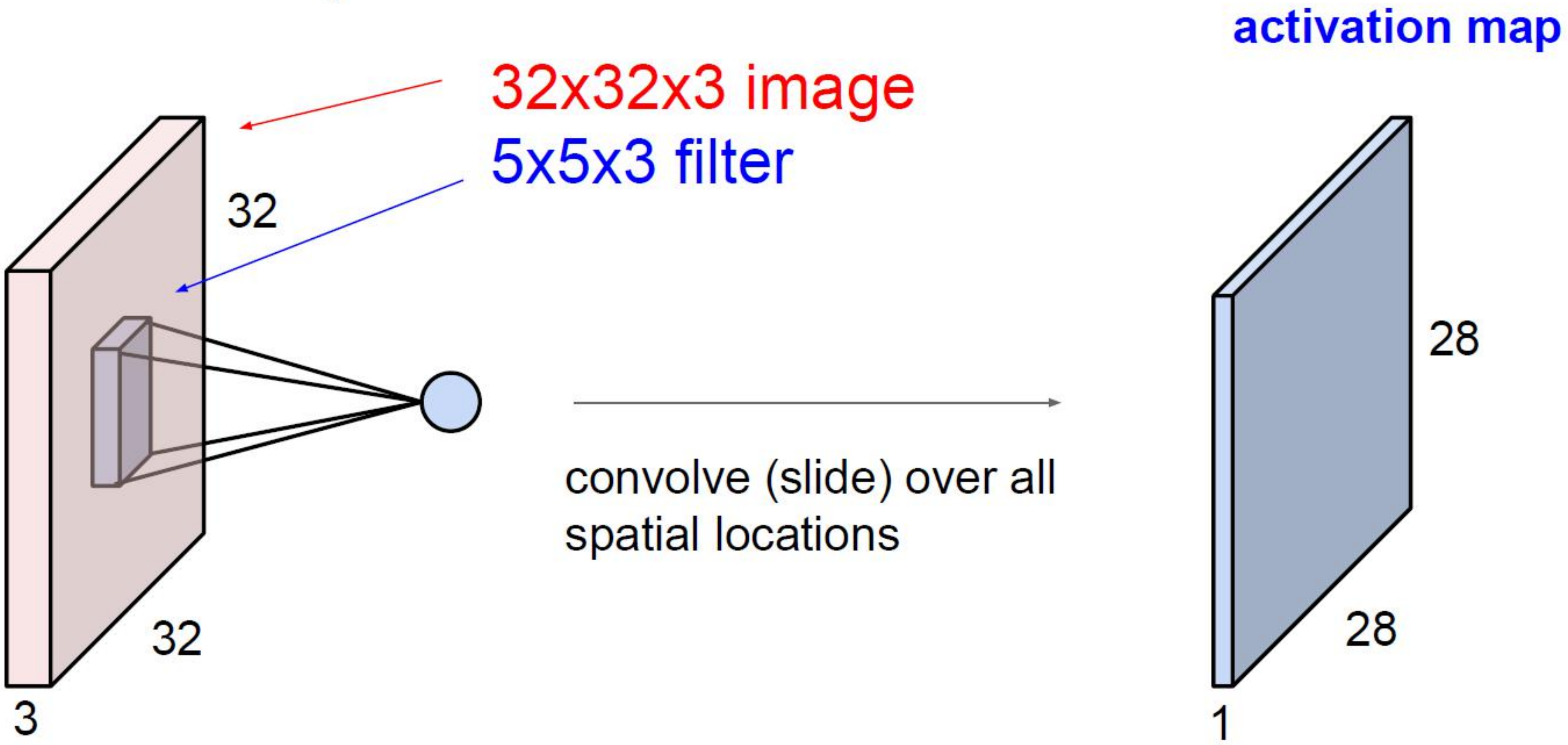
5x5x3 filter





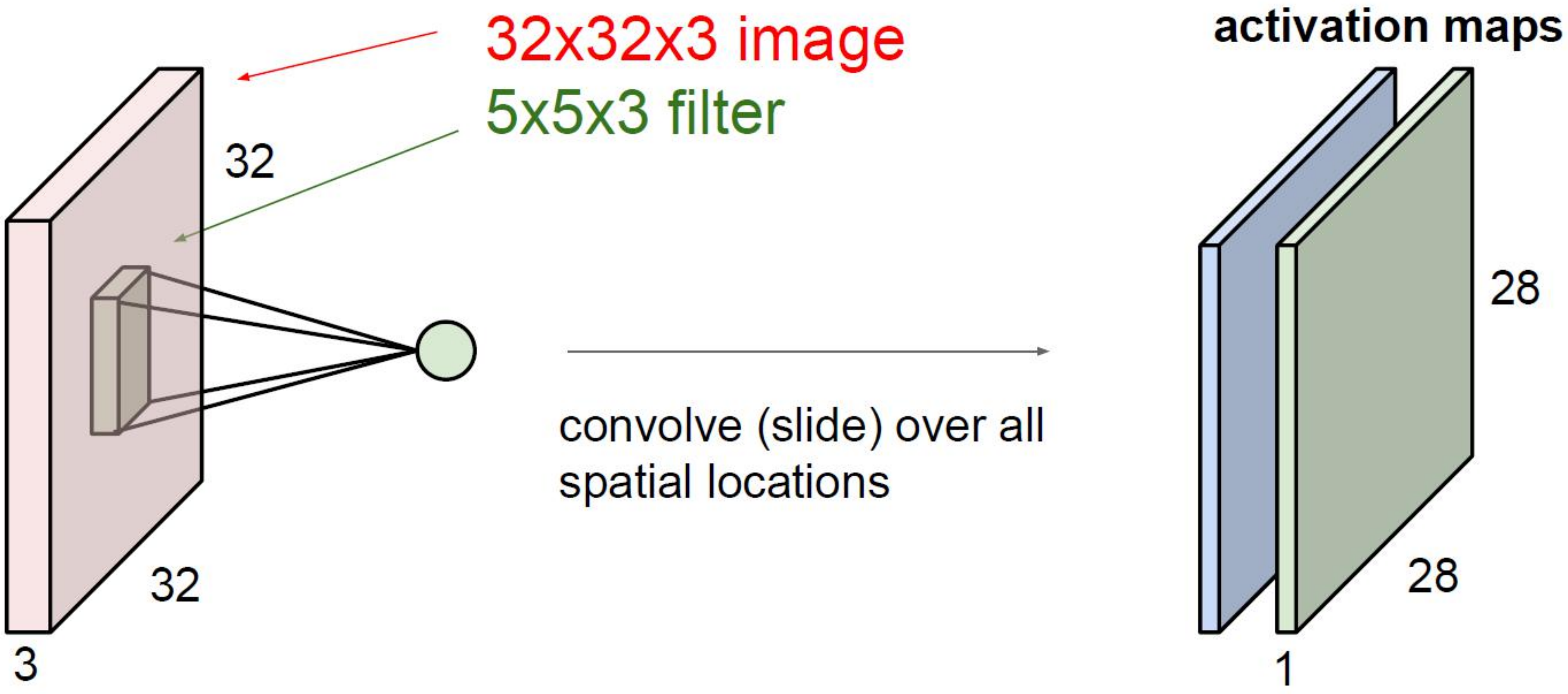
Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

Convolution Layer

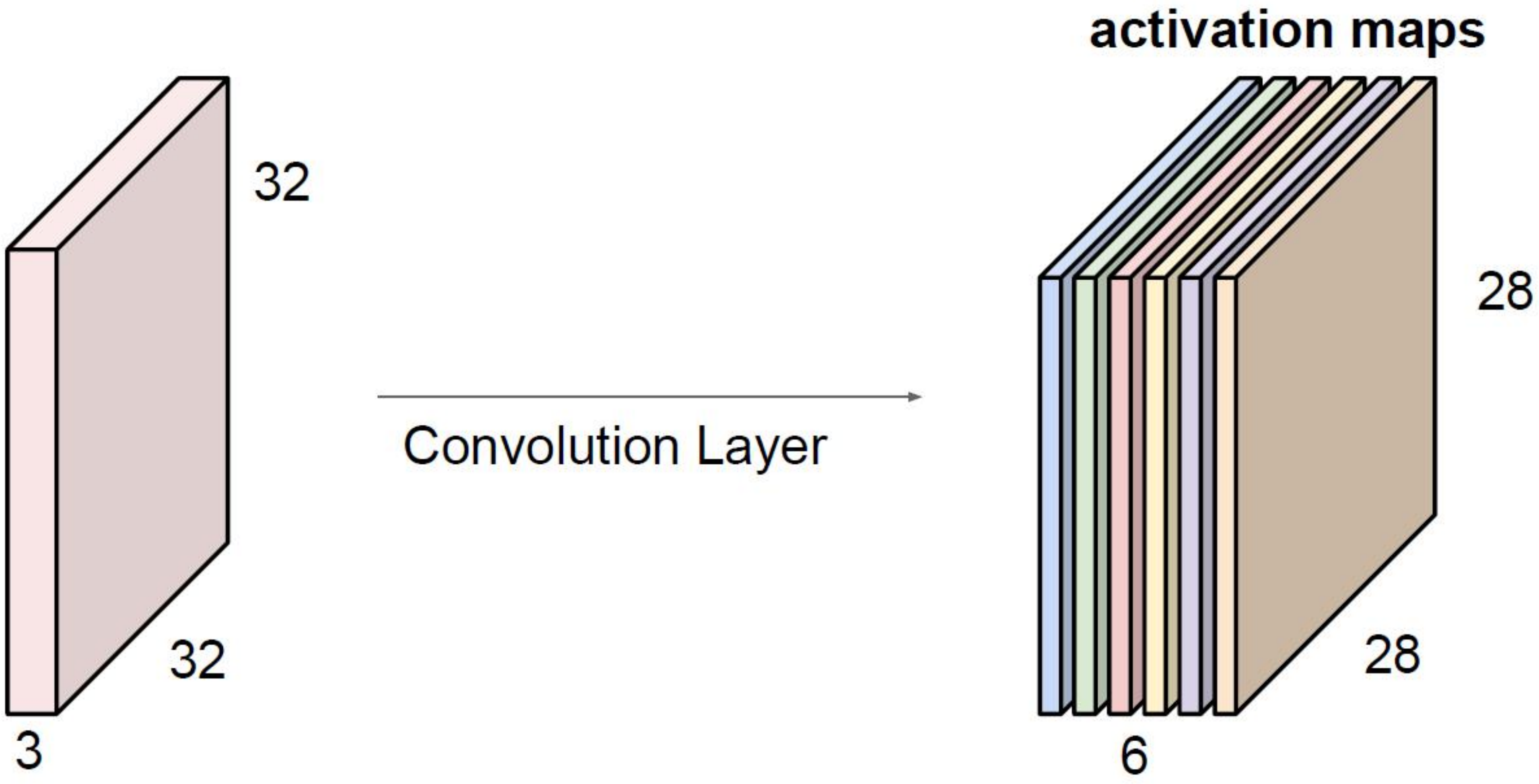


Convolution Layer

consider a second, green filter

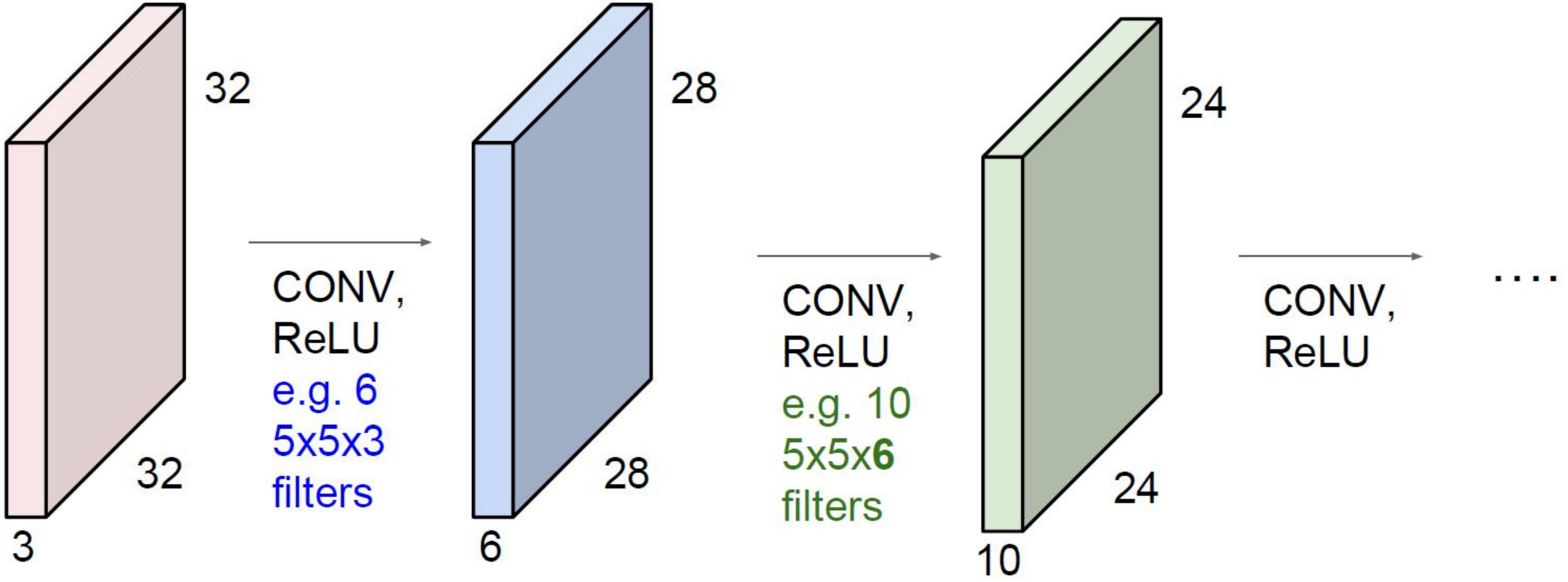


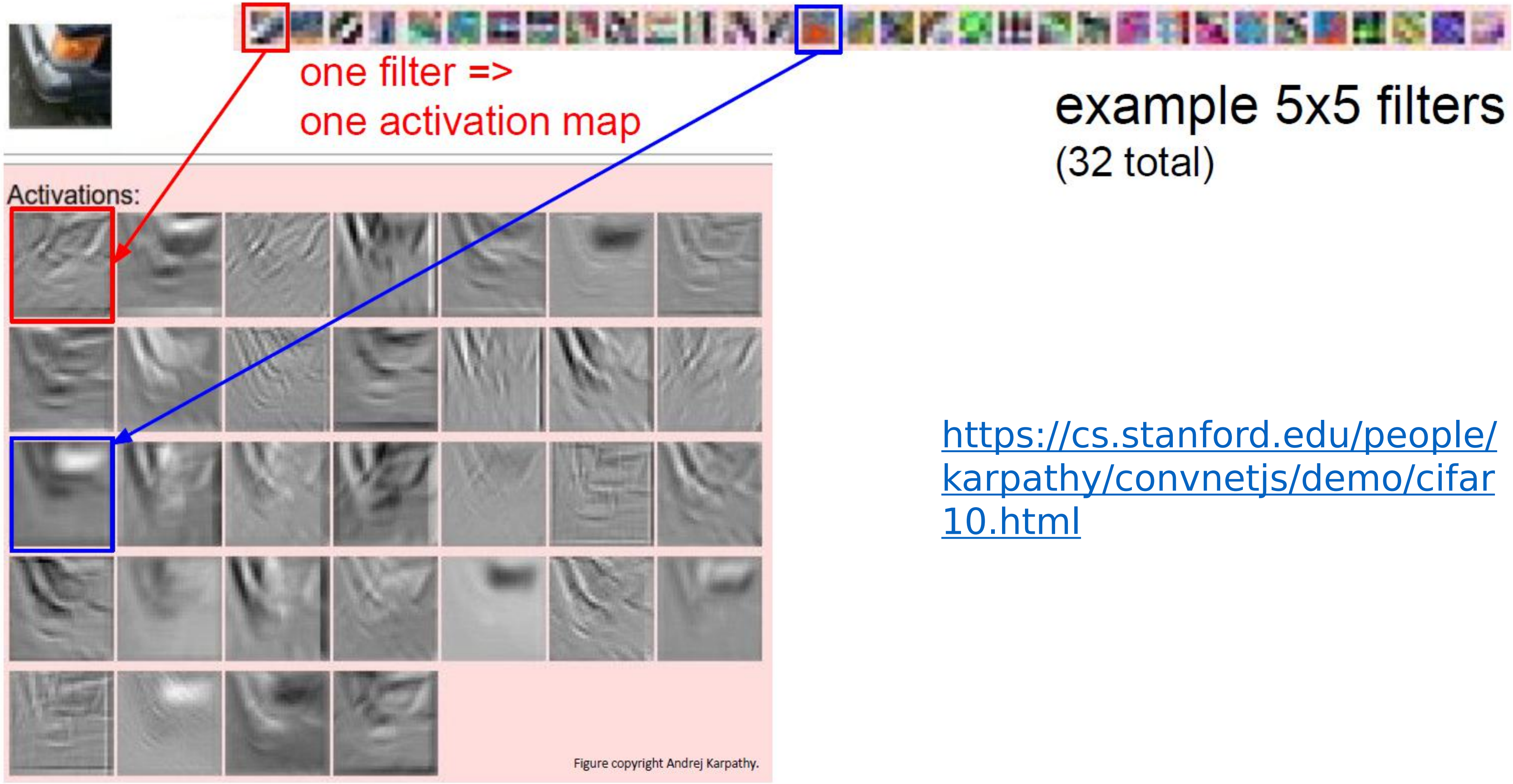
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



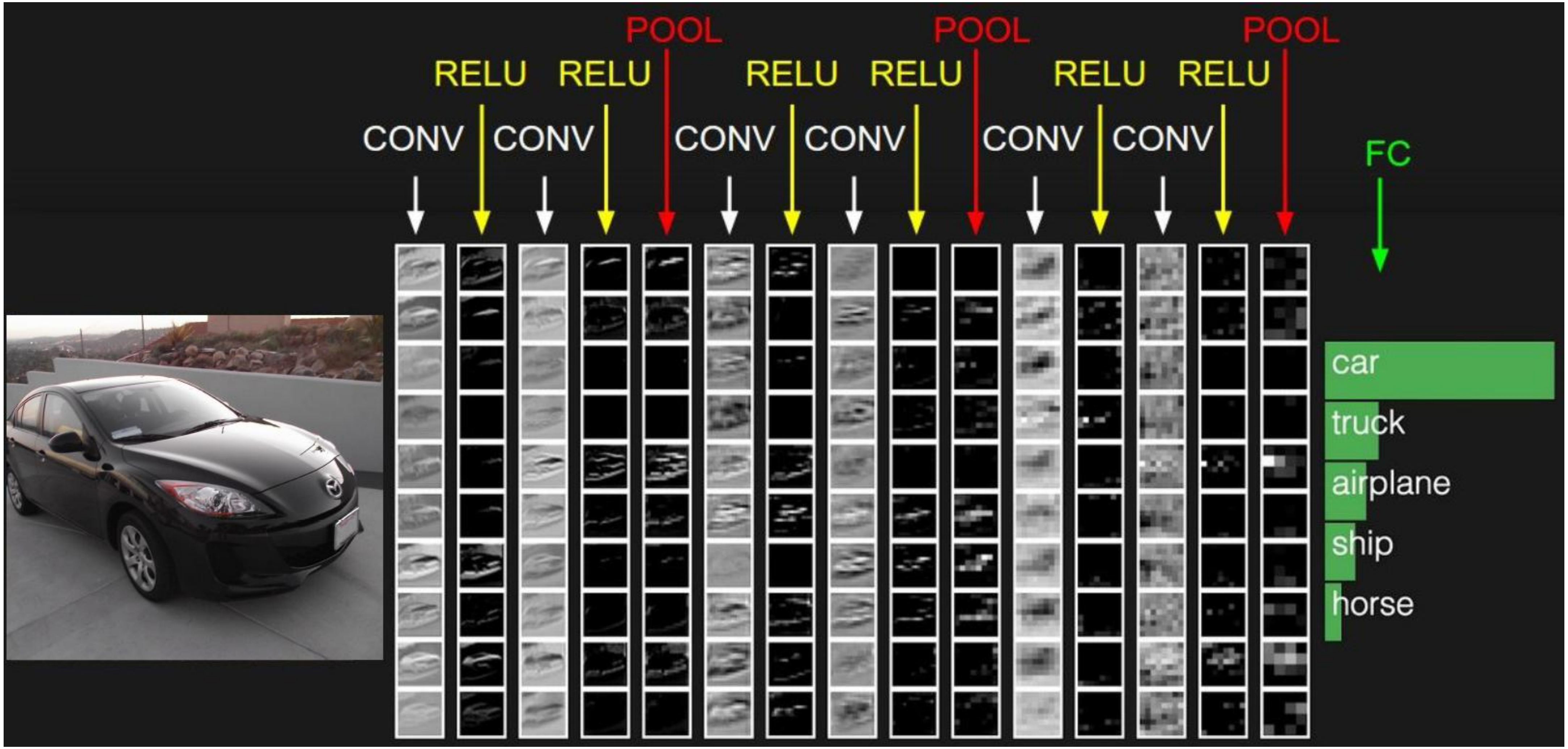
We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

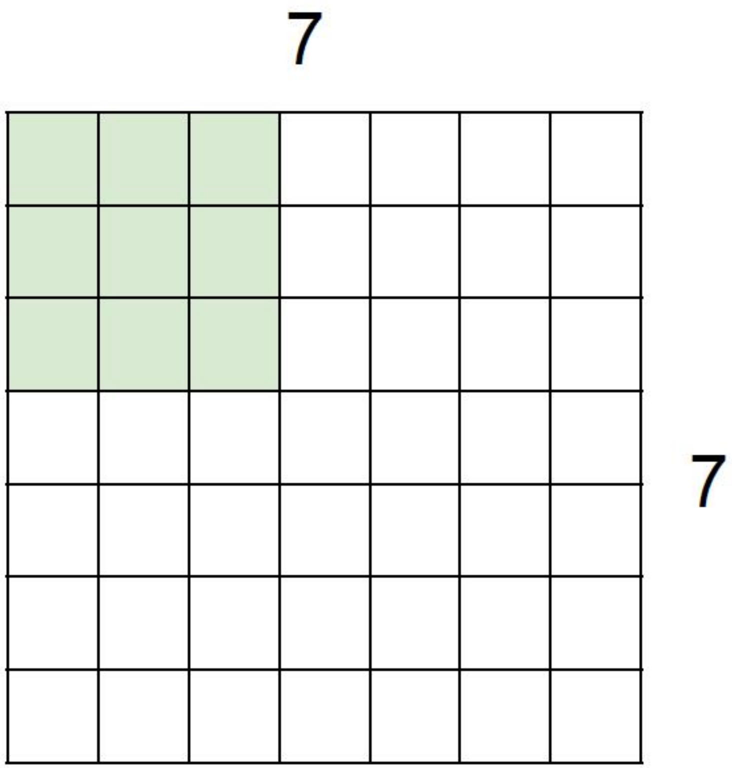




https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

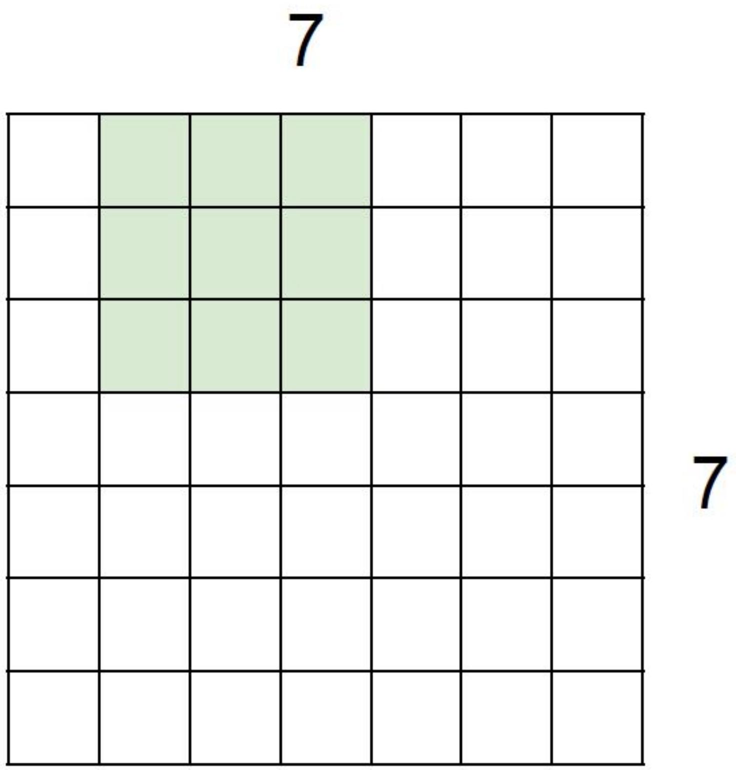

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A closer look at spatial dimensions:



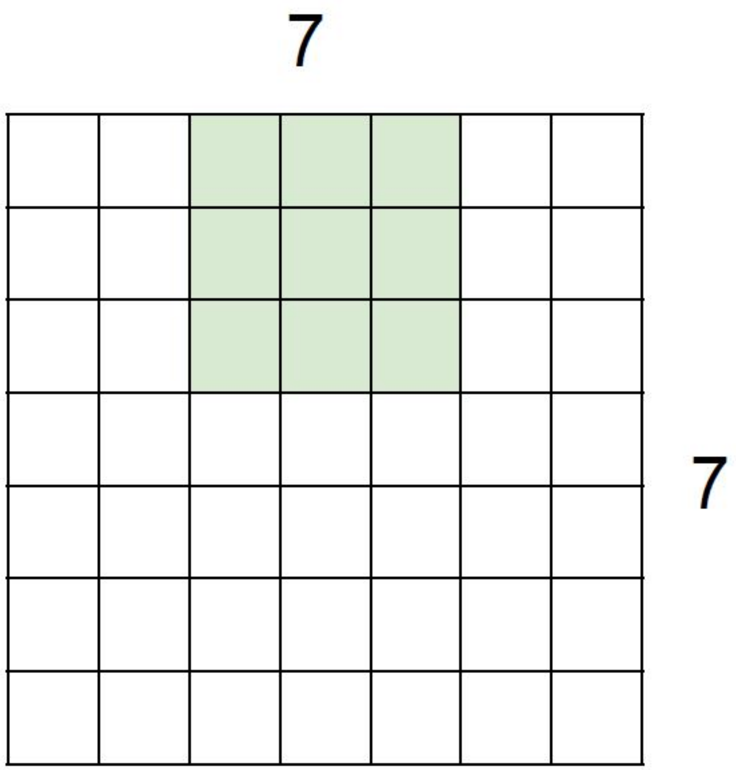
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



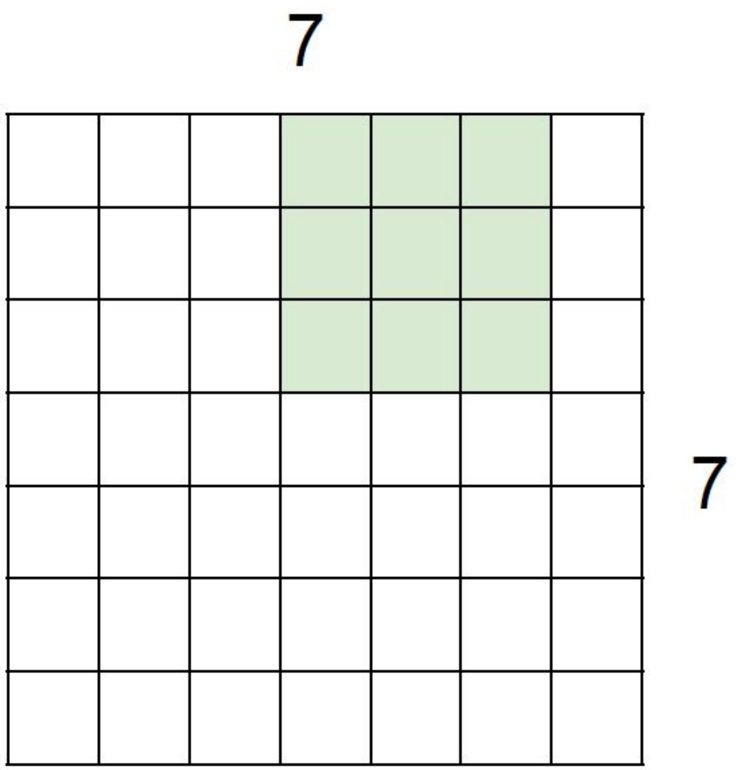
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



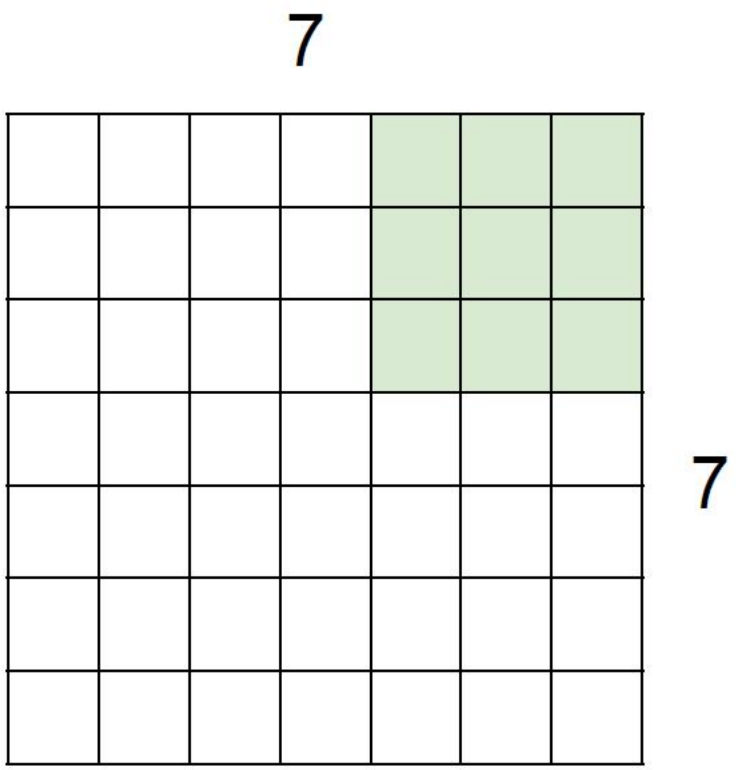
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

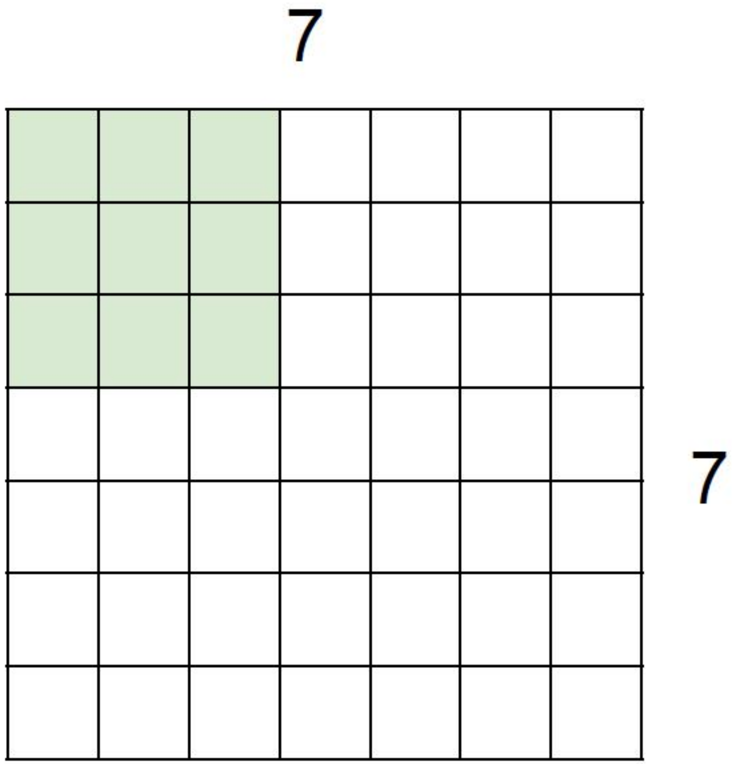
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

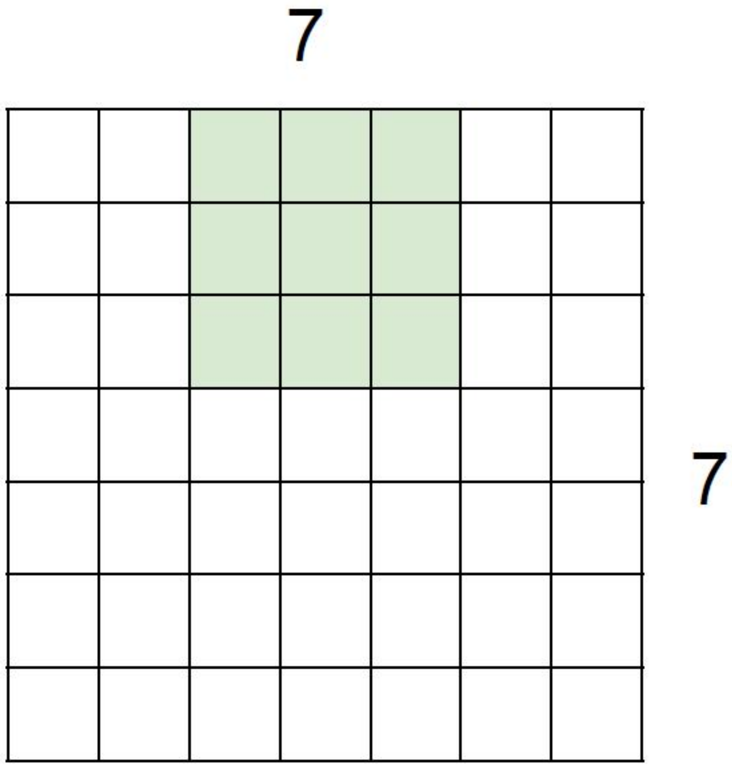
=> 5x5 output

A closer look at spatial dimensions:



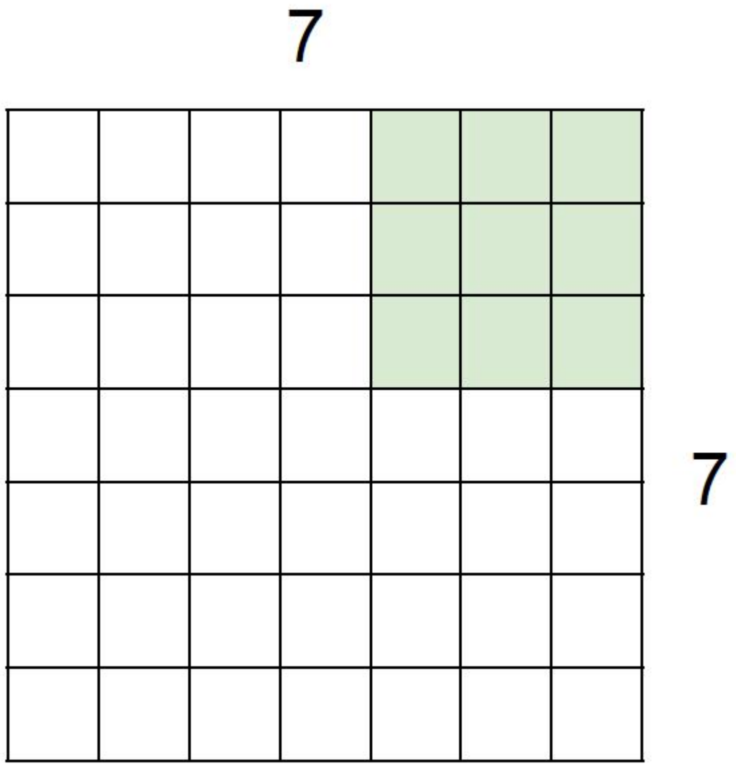
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



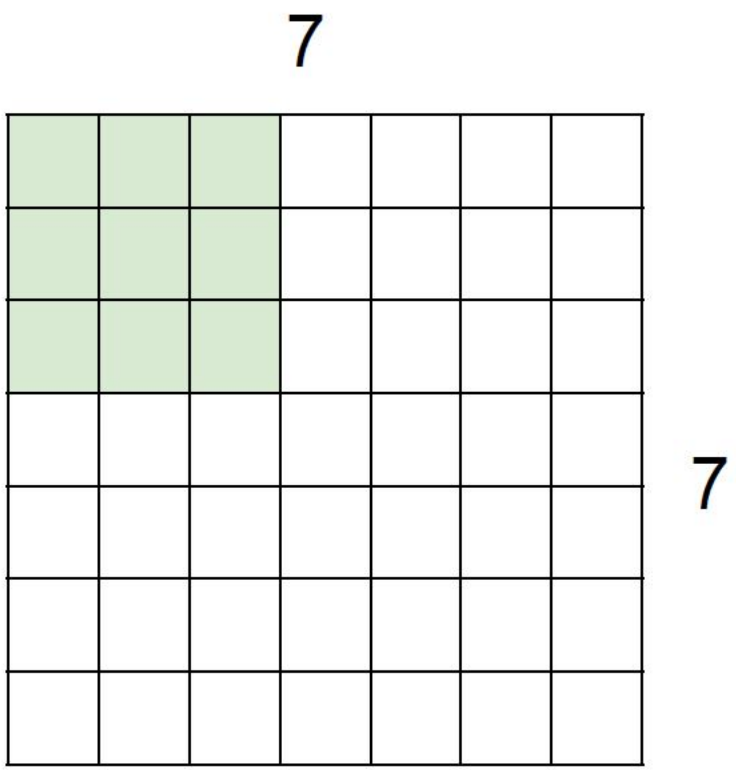
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



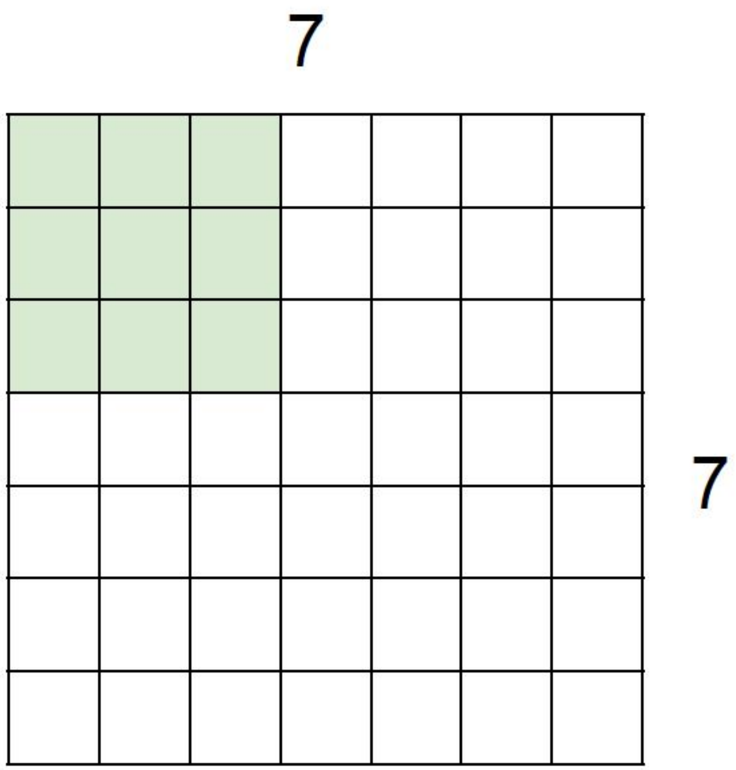
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



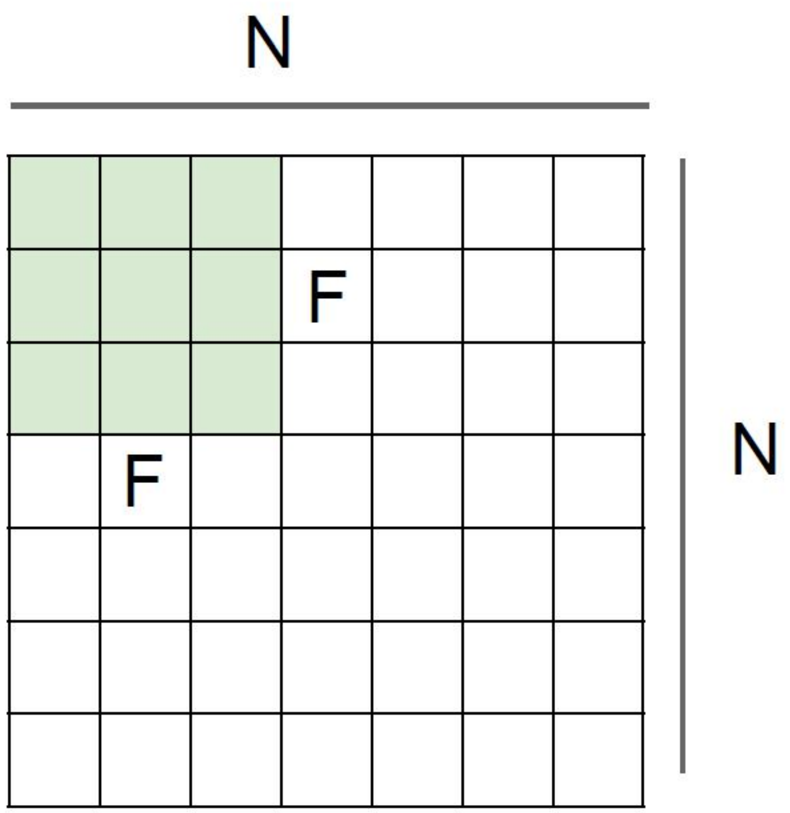
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

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.

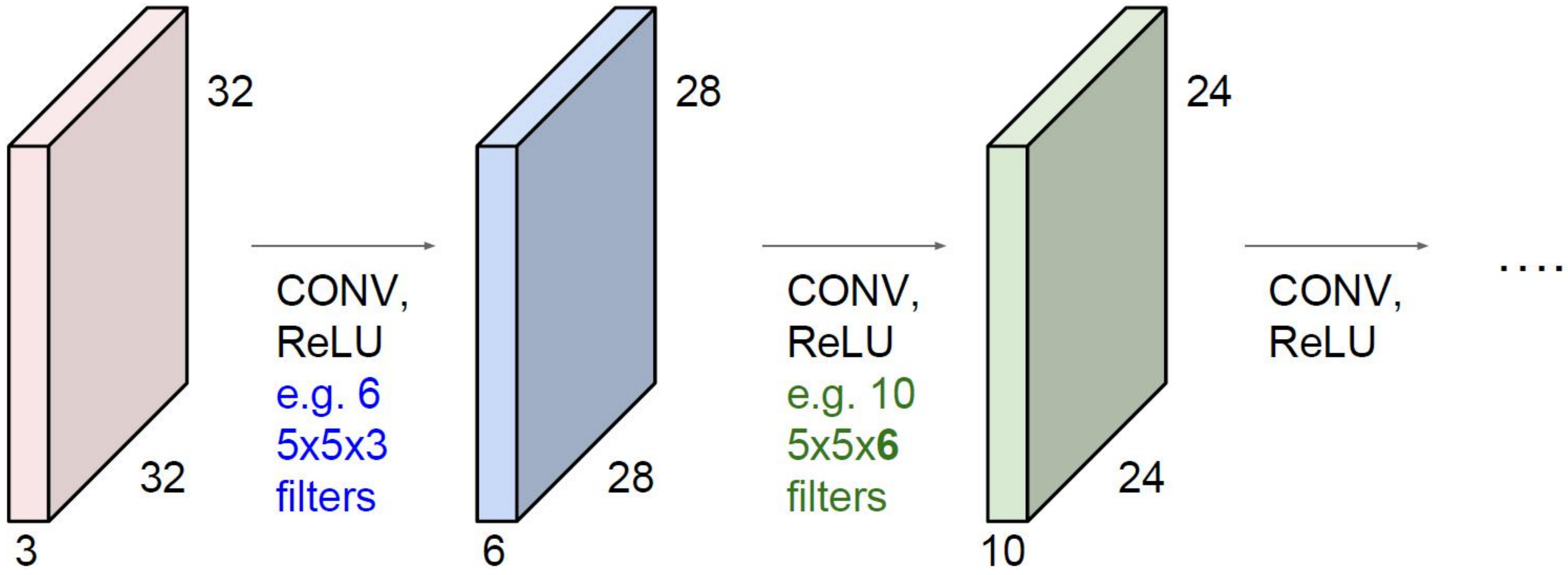


Output size:
(N - F) / stride + 1

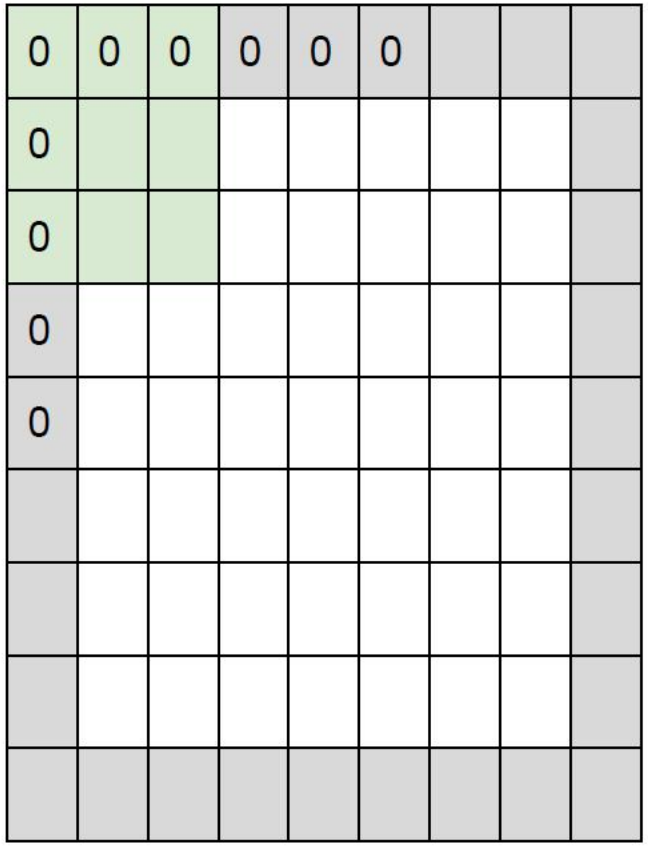
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

Remember back to…

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.



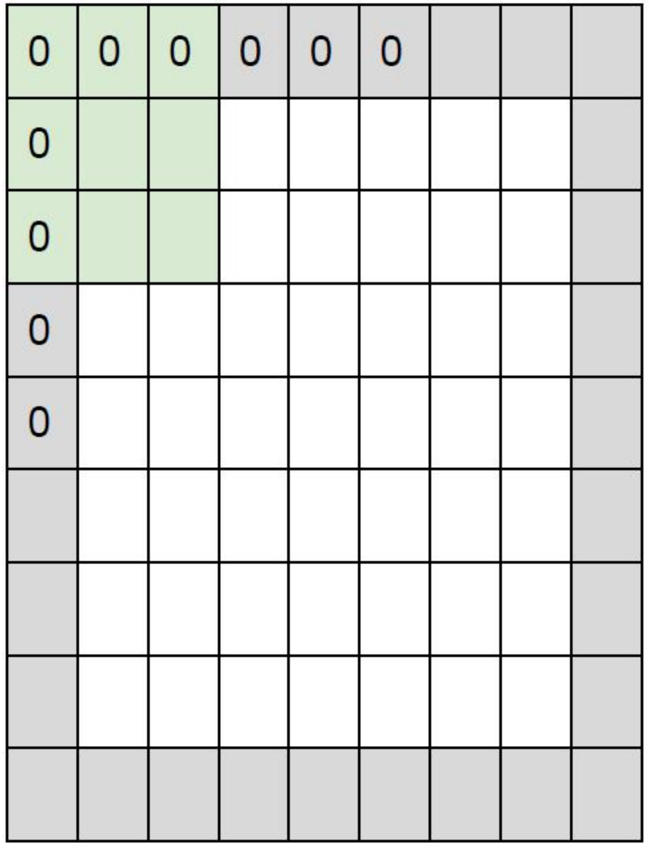
How to keep spatial size: zero paddings



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

How to keep spatial size: zero paddings



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 FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

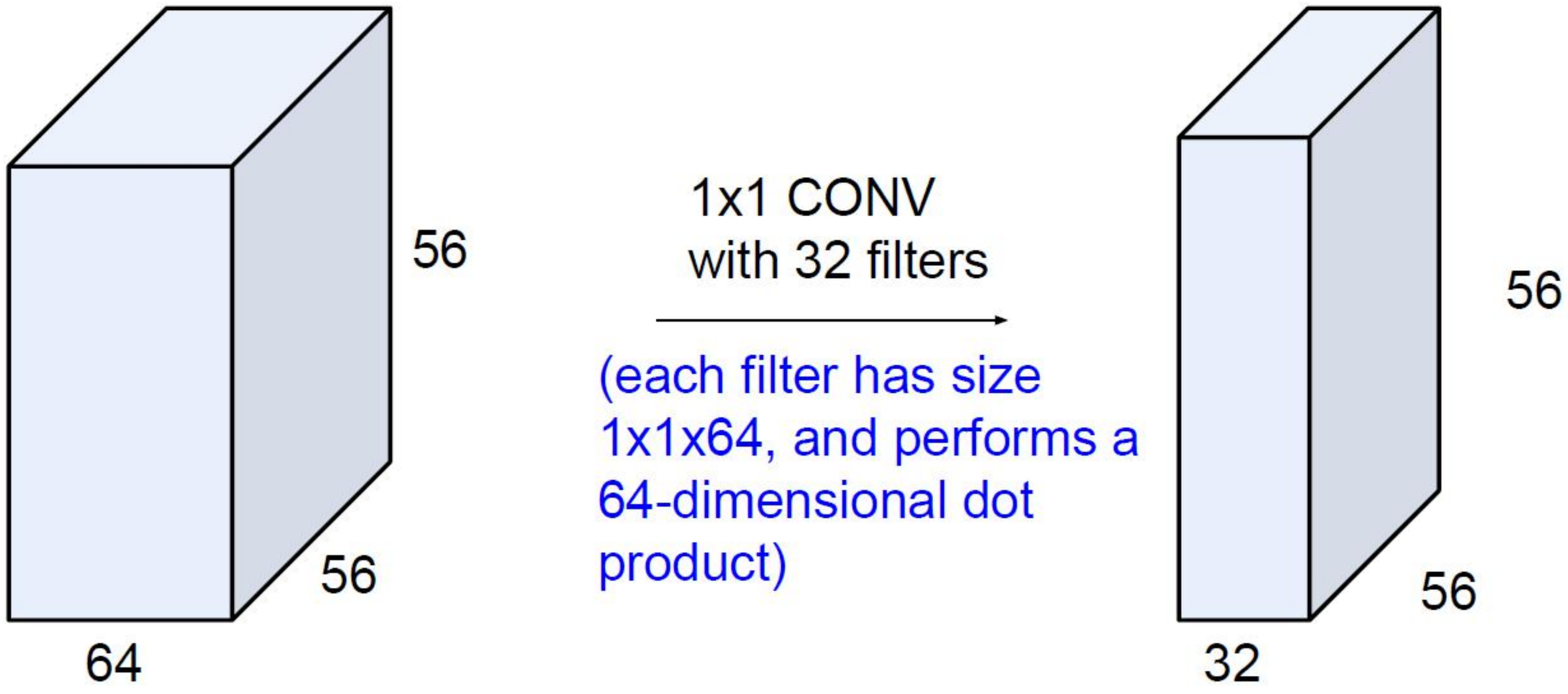
Magic formula (output volume size)

- Input a volume of size $W_1 \times H_1 \times D_1$, convoluted with
  1. Number of filters K
  2. Kernel of size F
  3. Stride S
  4. Number of zero-padding P
- Output a volume of size $W_2 \times H_2 \times D_2$
  1. $W_2 = \frac{W_1 - F + 2P}{S} + 1$
  2. $H_2 = \frac{H_1 - F + 2P}{S} + 1$
  3. $D_2 = K$

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)



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: $\left\lfloor \frac{out\_channels}{in\_channels} \right\rfloor$.

https://pytorch.org/docs/stable/nn.htm

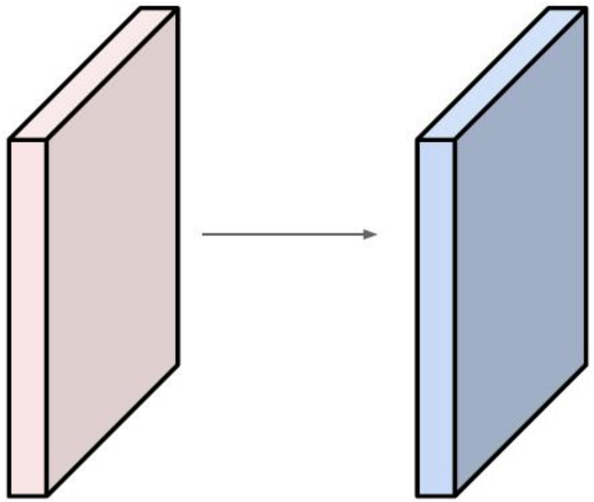
Magic formula (number of parameters)

- Input a volume of size $\boldsymbol{W_1} \times \boldsymbol{H_1} \times \boldsymbol{D_1}$, convoluted with
  1. Number of filters $\boldsymbol{K}$
  2. Kernel of size $\boldsymbol{F}$
  3. Stride $\boldsymbol{S}$
  4. Number of zero-padding $\boldsymbol{P}$
- Produces number of parameters
  1. $\boldsymbol{F} \times \boldsymbol{F} \times \boldsymbol{D_1}$ per filter
  2. $\boldsymbol{F} \times \boldsymbol{F} \times \boldsymbol{D_1} \times \boldsymbol{K} + \boldsymbol{K}$ (total number of parameters: weights + basis)

Examples time:

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

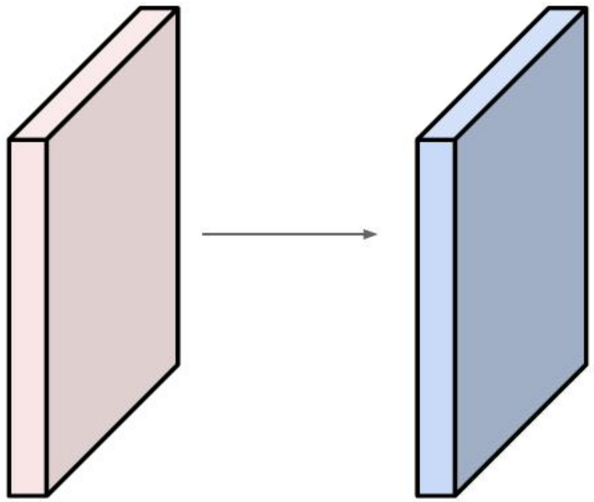
Output volume size: ?



Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Output volume size:

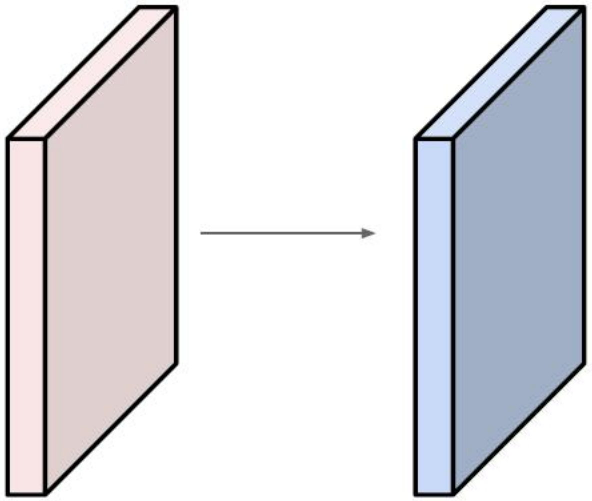
(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

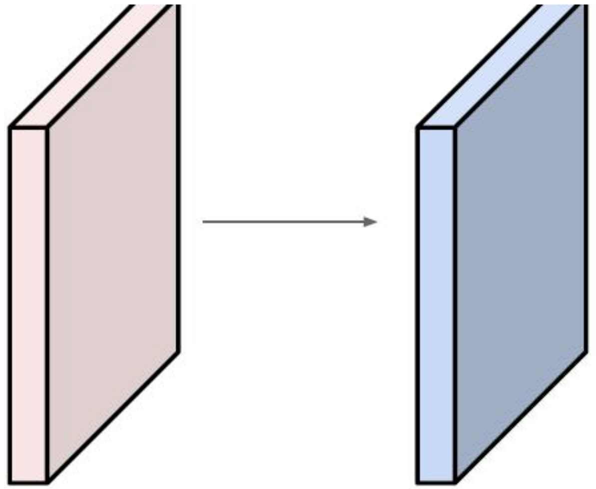
Examples time:

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

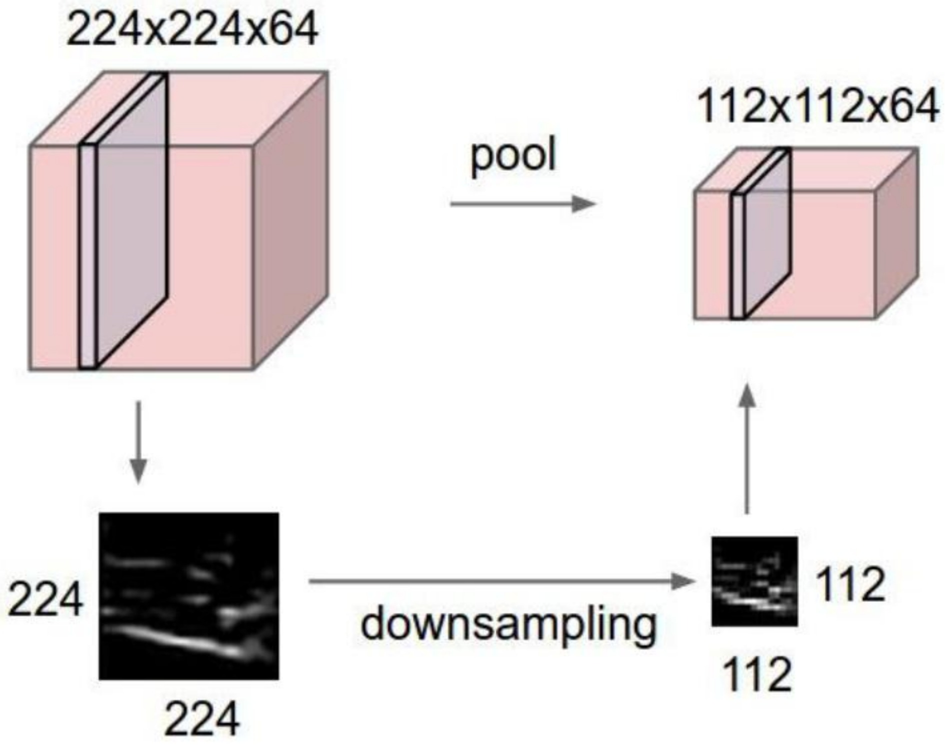
Number of parameters in this layer?

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

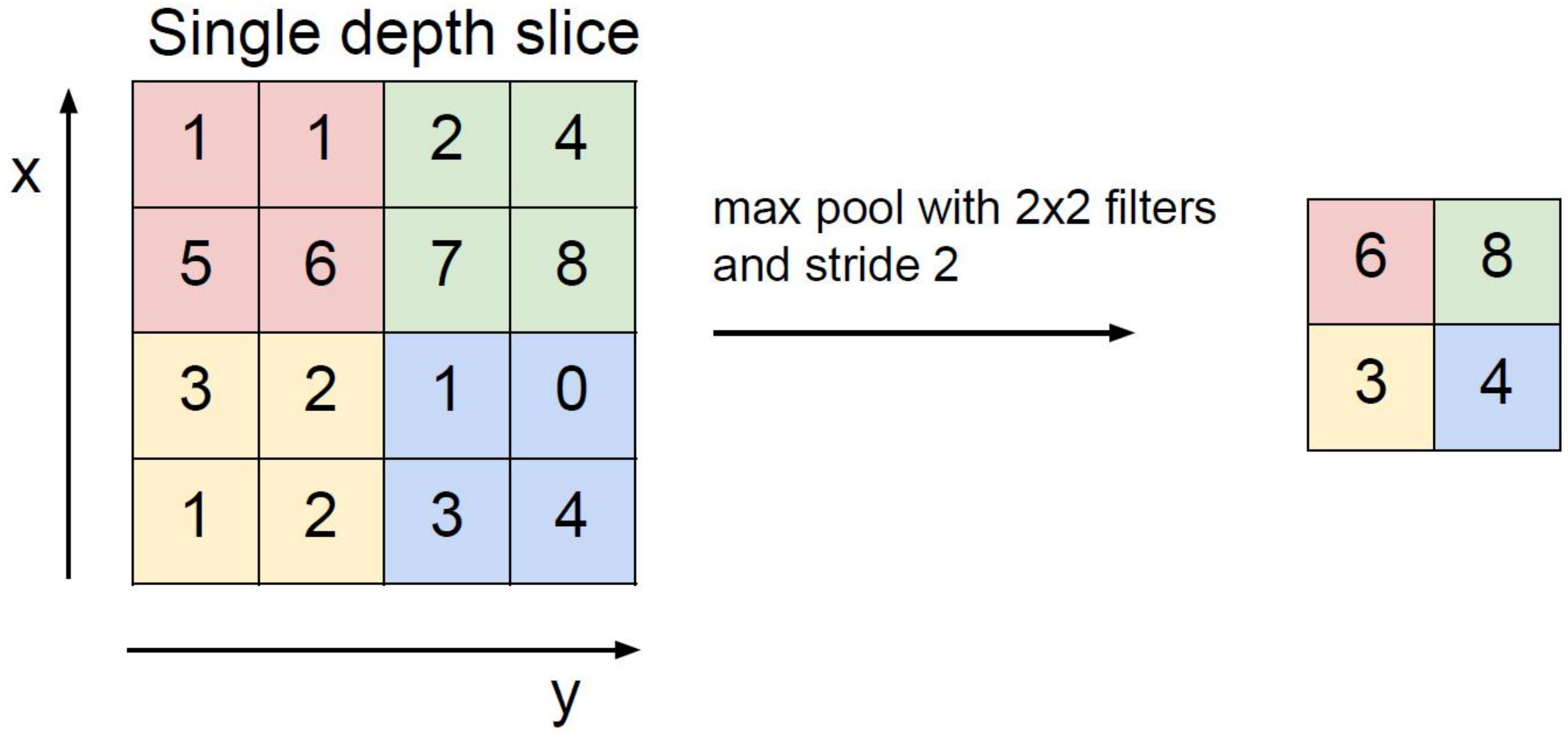
=> 76*10 = **760**

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

Magic formula (output volume size after pooling)

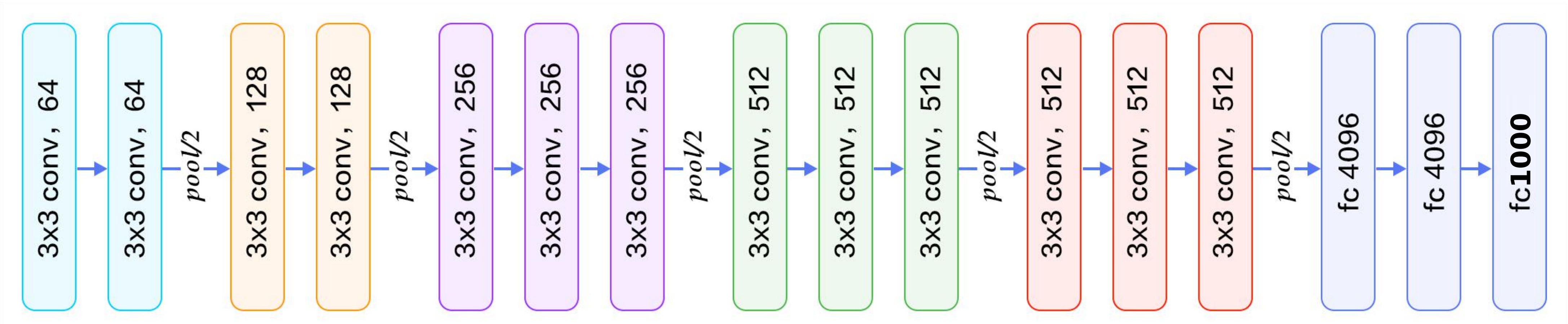
- Input a volume of size $\boldsymbol{W_1} \times \boldsymbol{H_1} \times \boldsymbol{D_1}$, convoluted with
  1. Kernel of size $\boldsymbol{F}$
  2. Stride $\boldsymbol{S}$
- Output a volume of size $\boldsymbol{W_2} \times \boldsymbol{H_2} \times \boldsymbol{D_2}$
  1. $\boldsymbol{W_2 = \frac{W_1 - F}{S} + 1}$
  2. $\boldsymbol{H_2 = \frac{H_1 - F}{S} + 1}$
  3. $\boldsymbol{D_2 = D_1}$

Common settings:

F = 2, S = 2

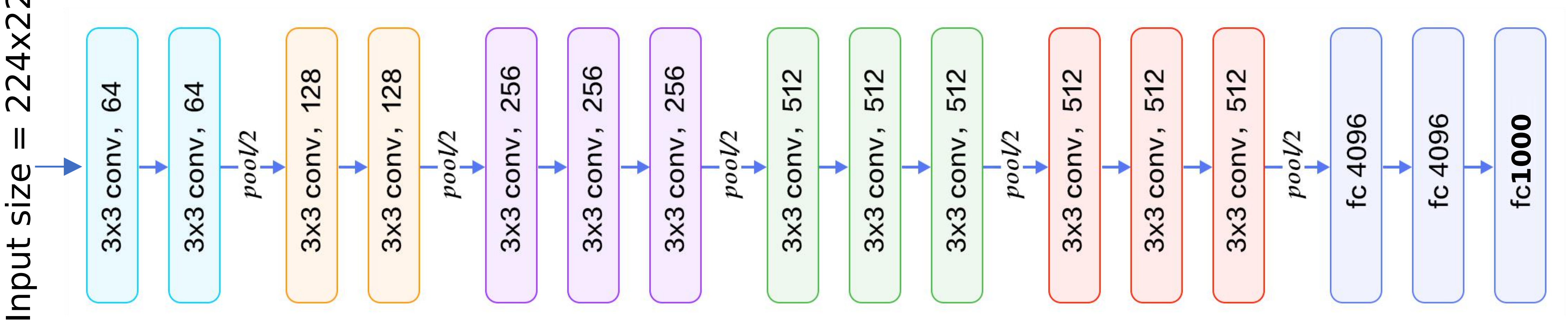
Case study: VGG16 for ImageNet classification

zero-padding size = 1



Case study: VGG16 for ImageNet classification

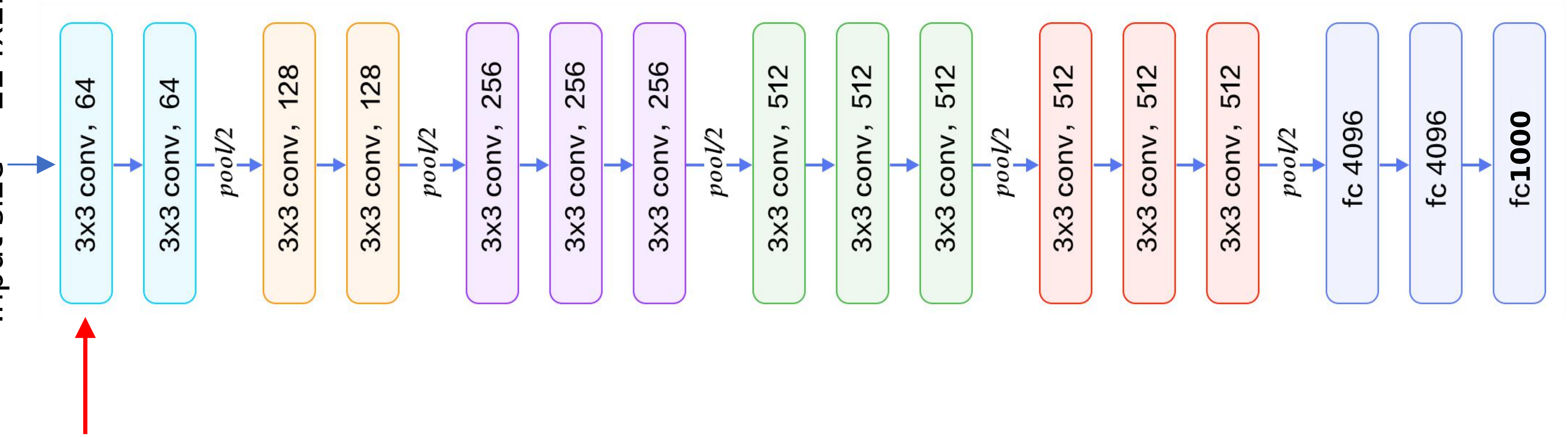
zero-padding size = 1



Case study: VGG16 for ImageNet classification

zero-padding size = 1

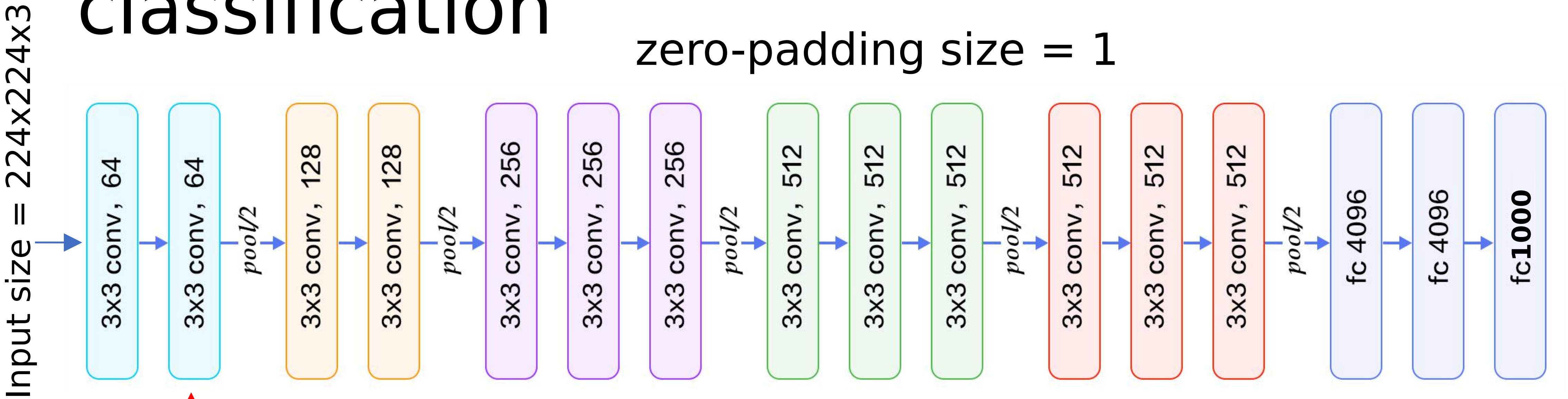
Input size = 224x224x3



224x224x64
(3x3x3)x64+64

Case study: VGG16 for ImageNet classification

zero-padding size = 1



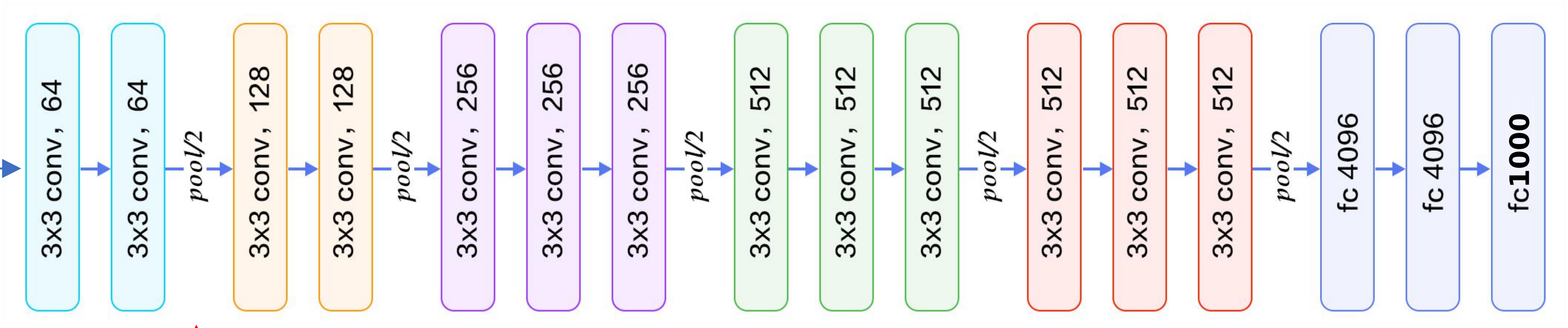
224x224x64

(3x3x64)x64+64

Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3

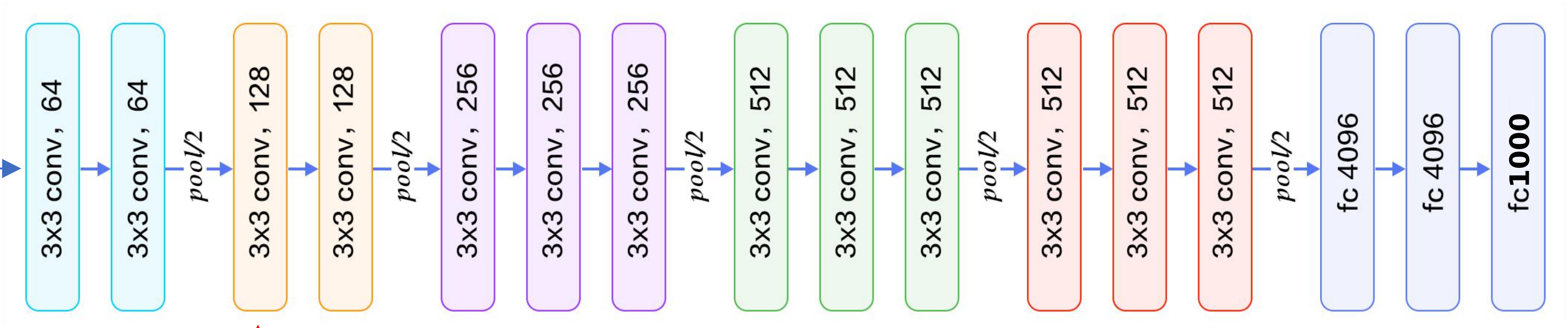


112x112x64
0

Case study: VGG166 for ImageNet classification

Input size = 224x224x3

zero-padding size = 1



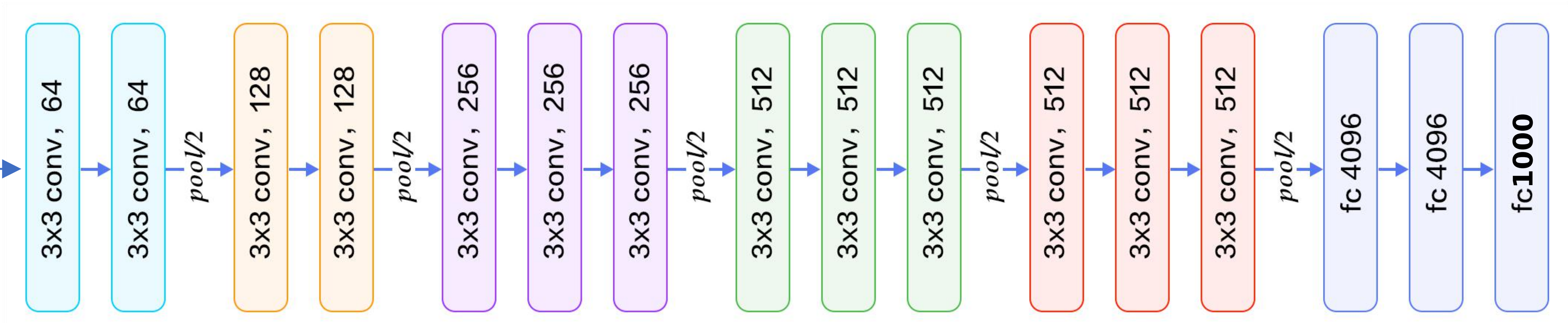
112x112x128

(3x3x64)x128+128

Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3



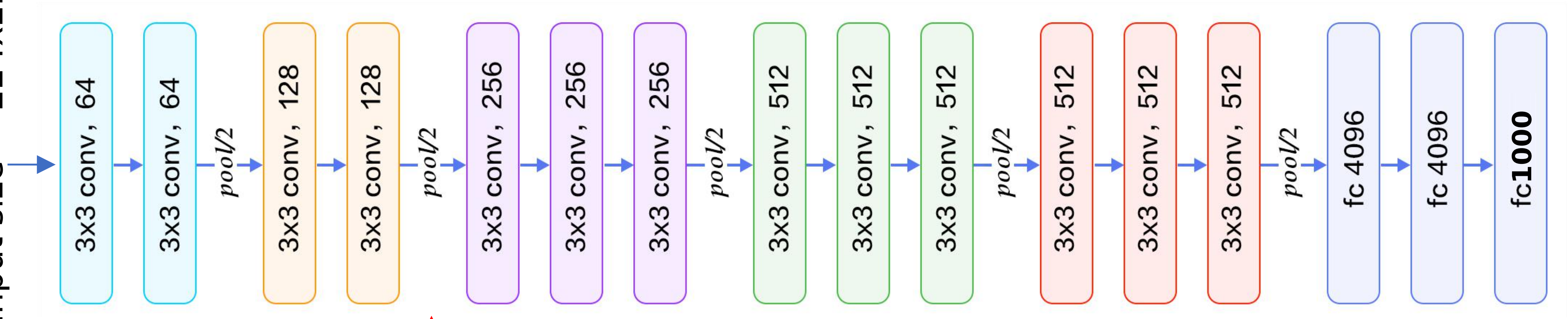
112x112x128

(3x3x128)x128+128

Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3

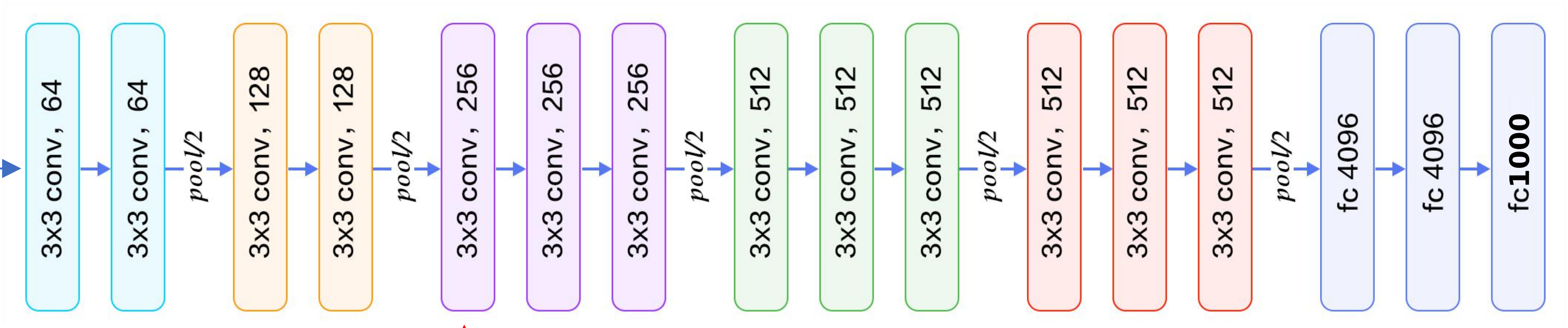


64x64x128
0

Case study: VGG166 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3



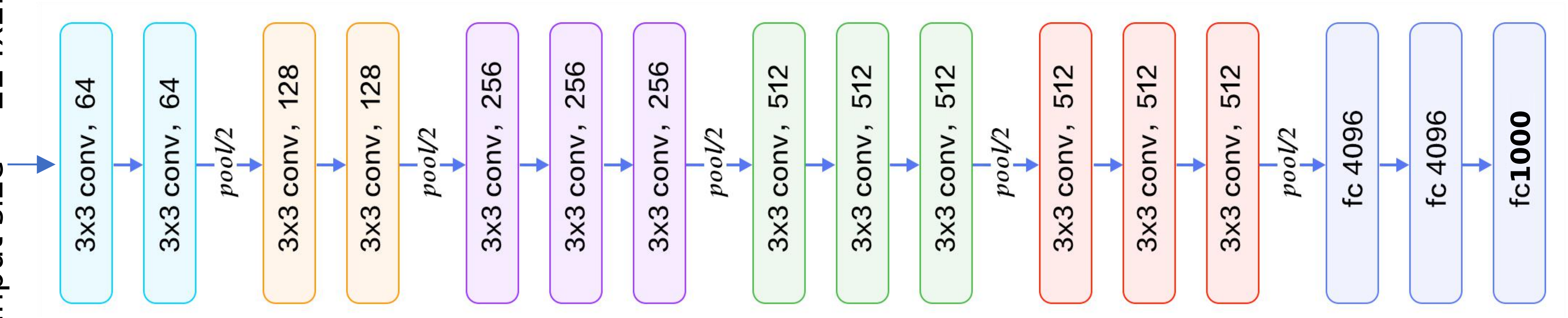
56x56x256

(3x3x128)x256+256

Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3

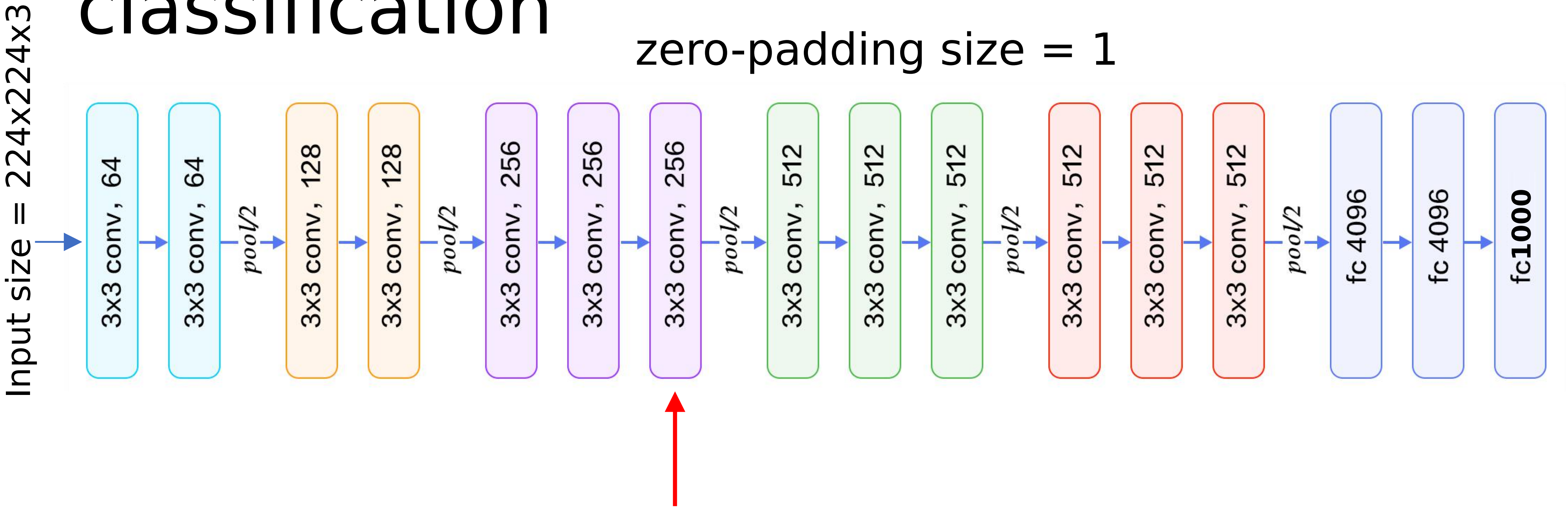


56x56x256

(3x3x256)x256+256

Case study: VGG16 for ImageNet classification

zero-padding size = 1



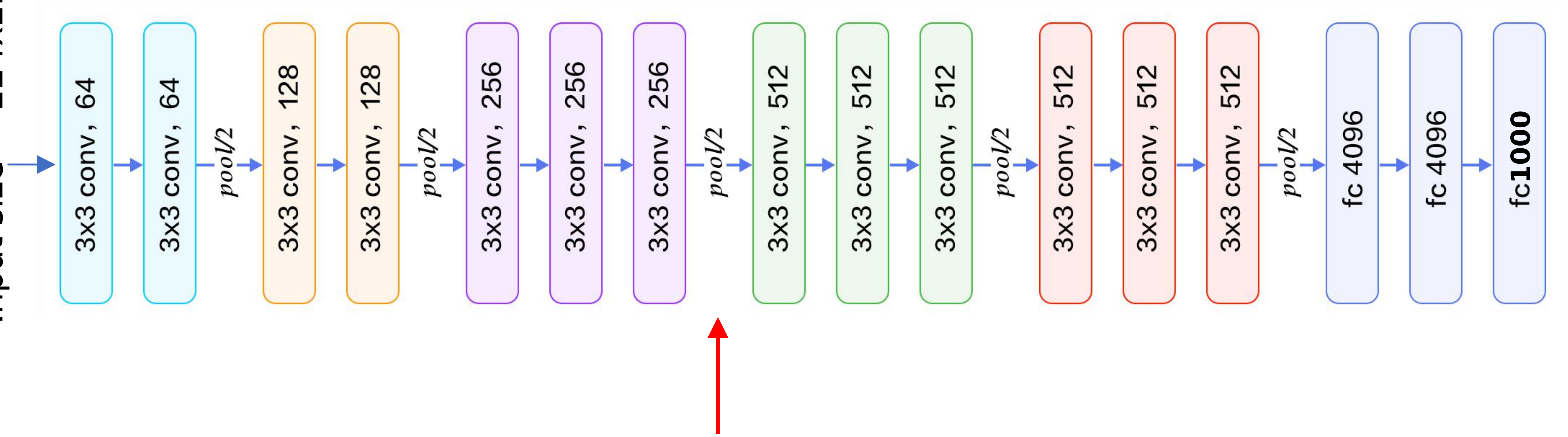
56x56x256

(3x3x256)x256+256

Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3

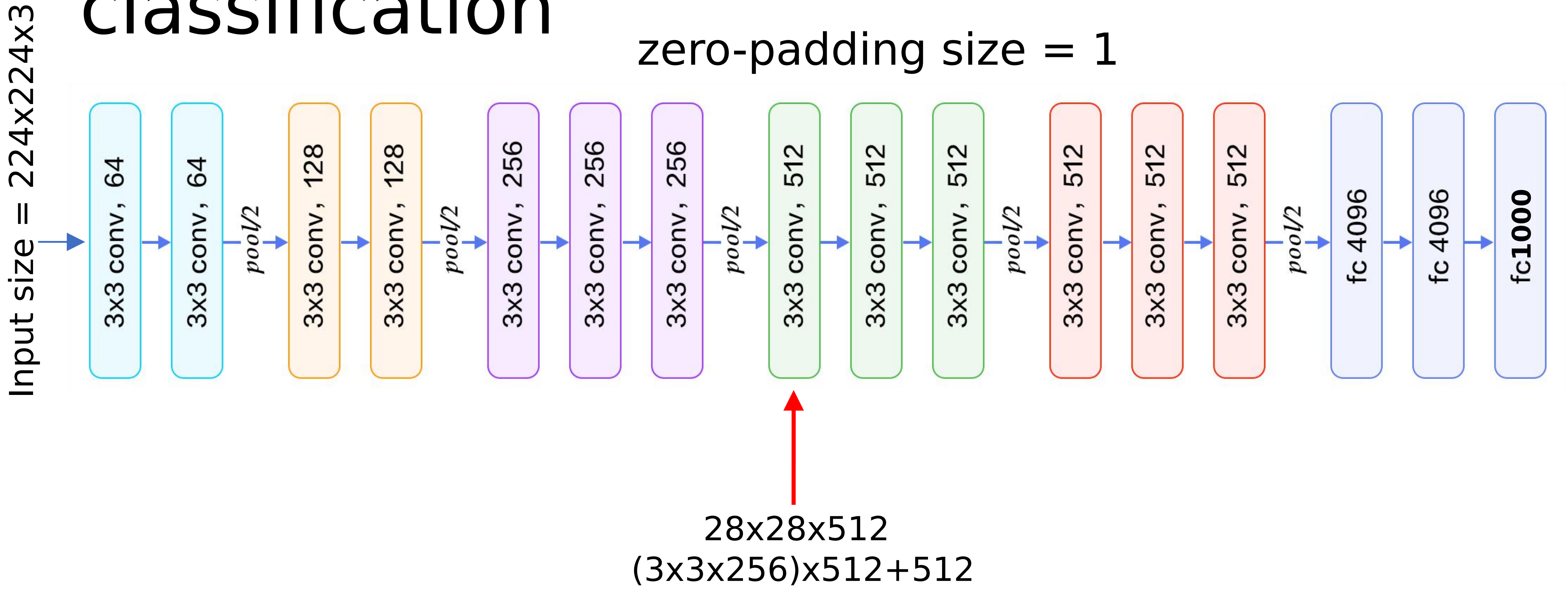


28x28x256

0

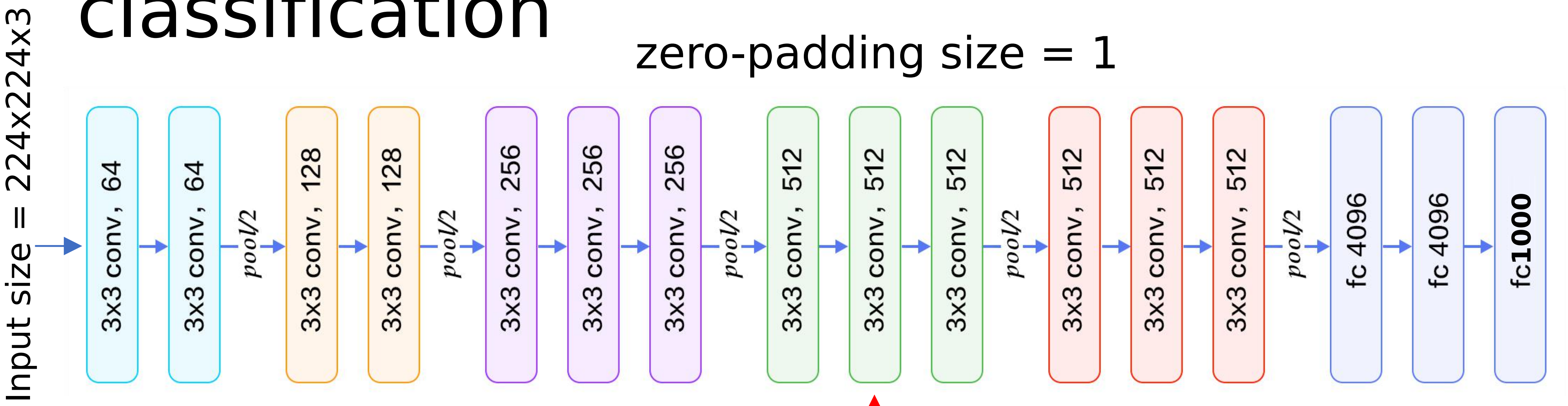
Case study: VGG16 for ImageNet classification

zero-padding size = 1



Case study: VGG16 for ImageNet classification

zero-padding size = 1

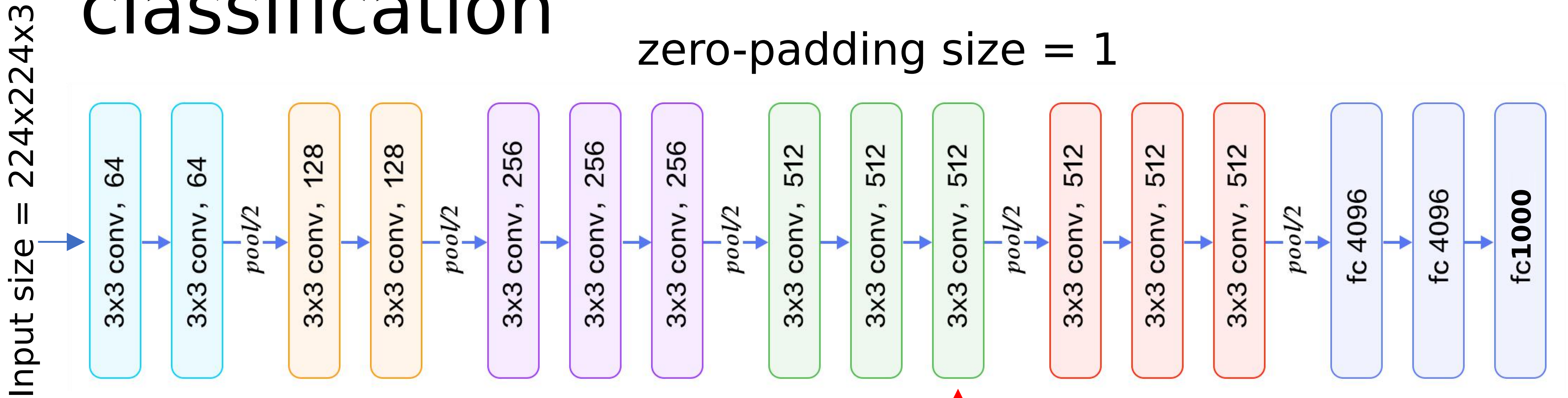


28x28x512

(3x3x512)x512+512

Case study: VGG16 for ImageNet classification

zero-padding size = 1



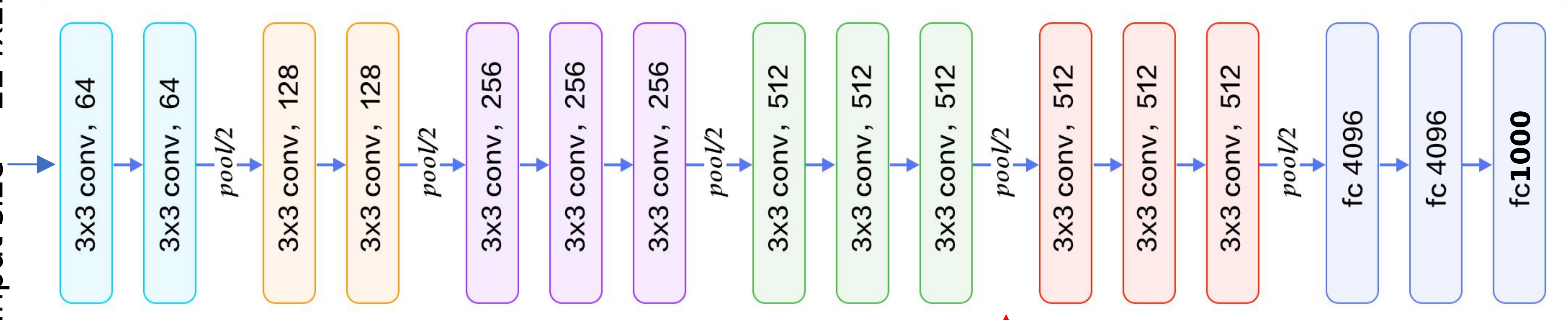
28x28x512

(3x3x512)x512+512

Case study: VGG16 for ImageNet classification

zero-padding size = 1

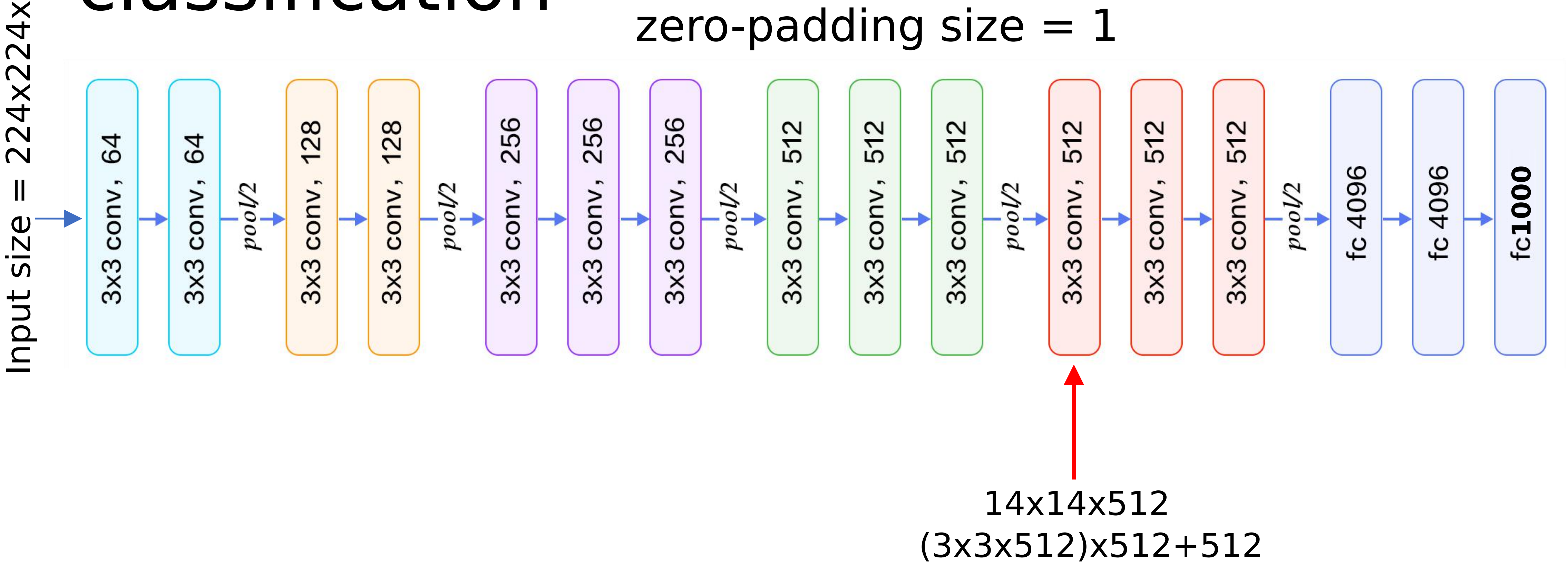
Input size = 224x224x3



14x14x512

0

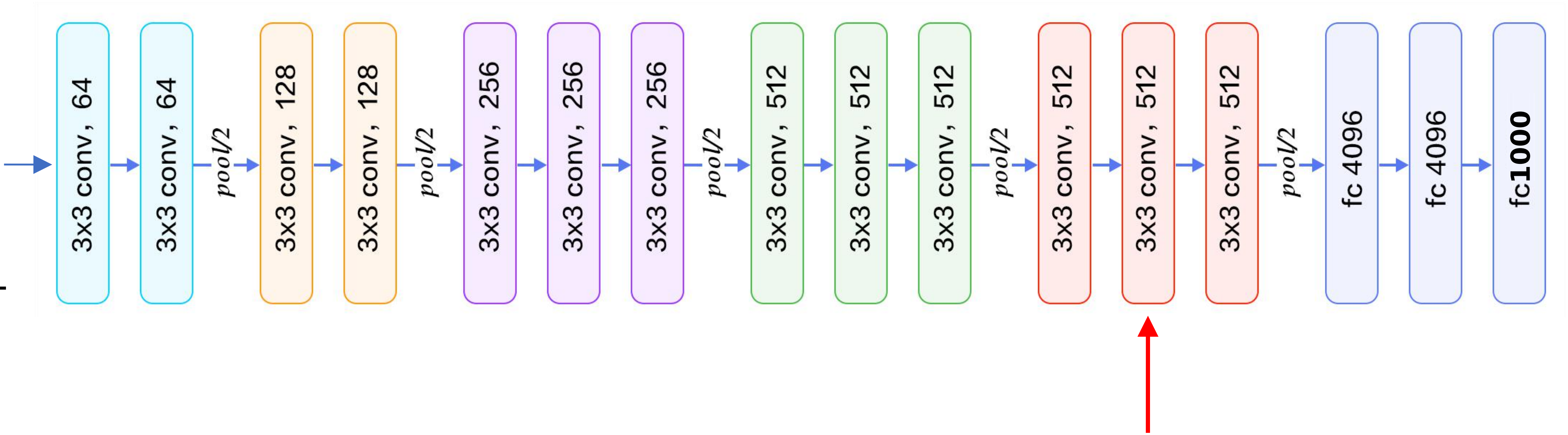
Case study: VGG16 for ImageNet classification



Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3

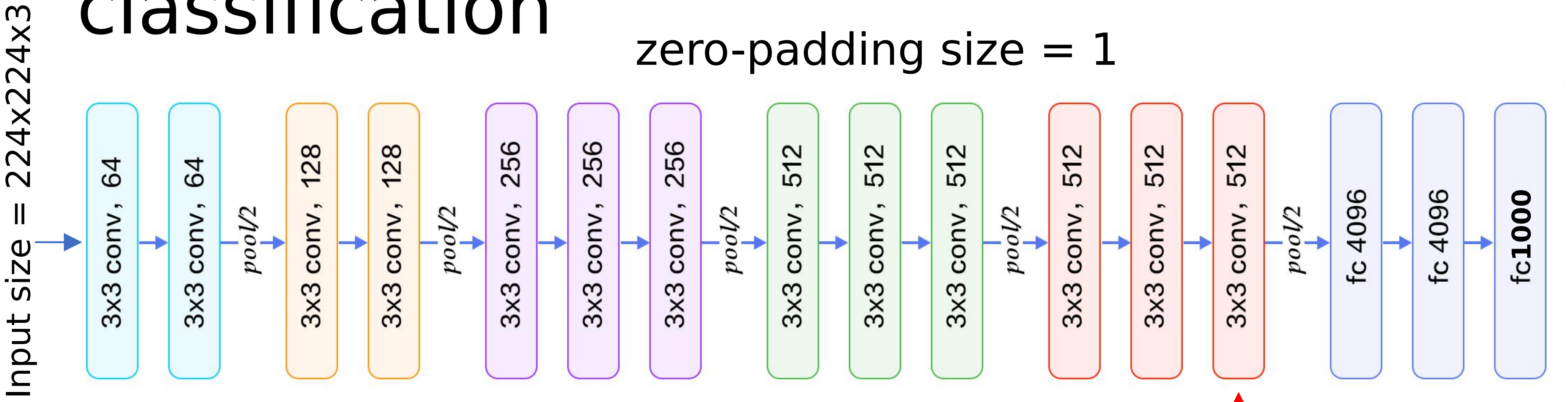


14x14x512

(3x3x512)x512+512

Case study: VGG16 for ImageNet classification

zero-padding size = 1



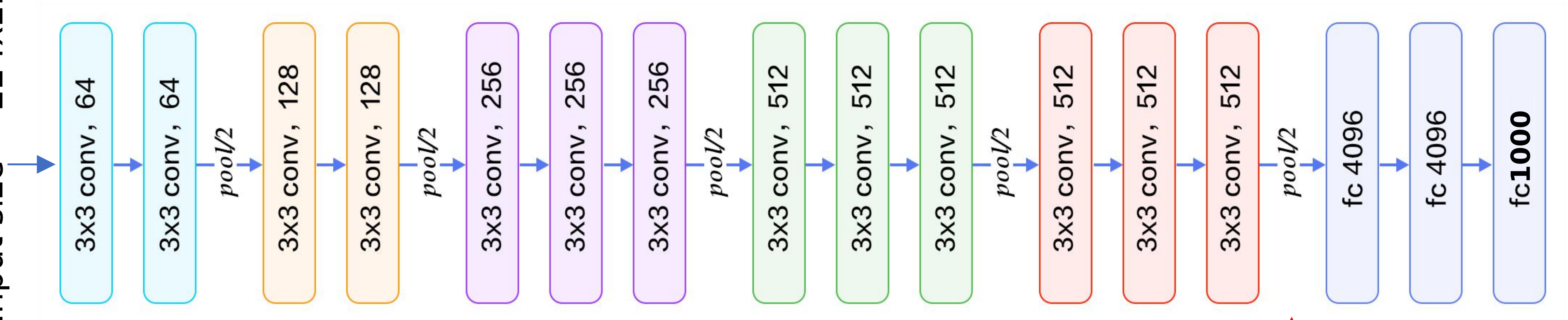
14x14x512

(3x3x512)x512+512

Case study: VGG16 for ImageNet classification

Input size = 224x224x3

zero-padding size = 1

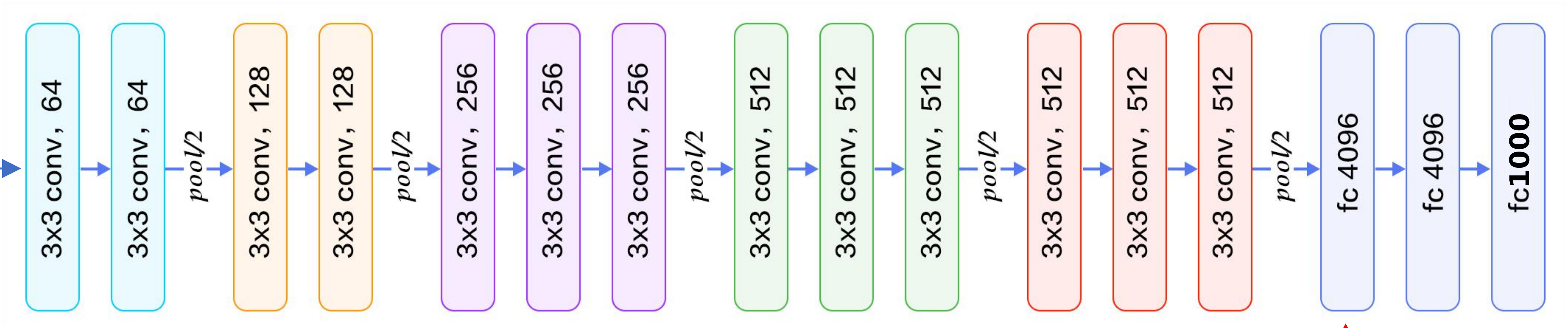


7x7x512
0

Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3



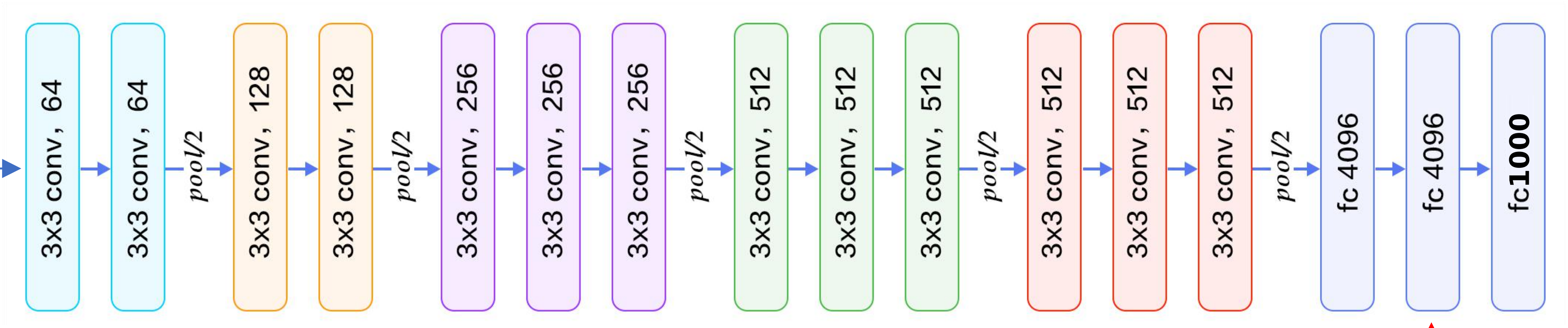
1x1x4096

(7x7x512)x4096+4096

Case study: VGG16 for ImageNet classification

zero-padding size = 1

Input size = 224x224x3

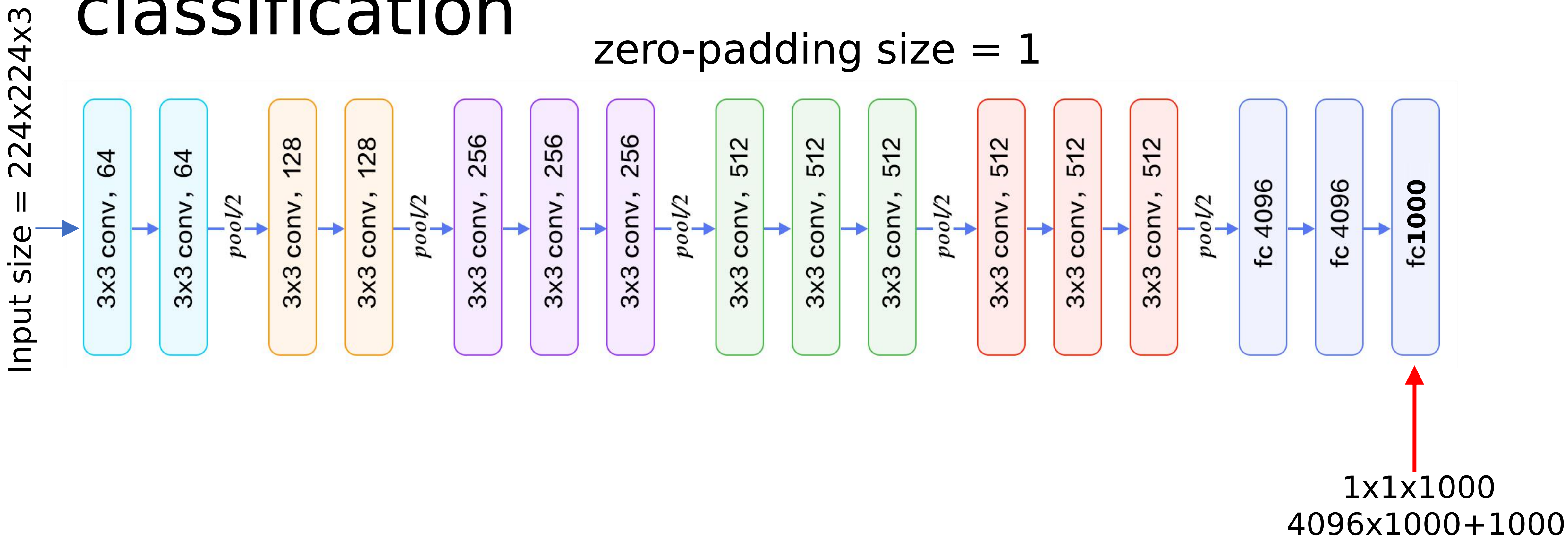


1x1x4096

4096x4096+4096

Case study: VGG16 for ImageNet classification

zero-padding size = 1



Total parameters: 138M

Summary

- Convolutional networks stack CONV, POOL, FC layers
- Trend towards getting rid of POOL/FC layers (just CONV)
- [(CONV-ReLU)*N-POOL]*M-(FC-ReLU)*K,SOFTMAX for classification