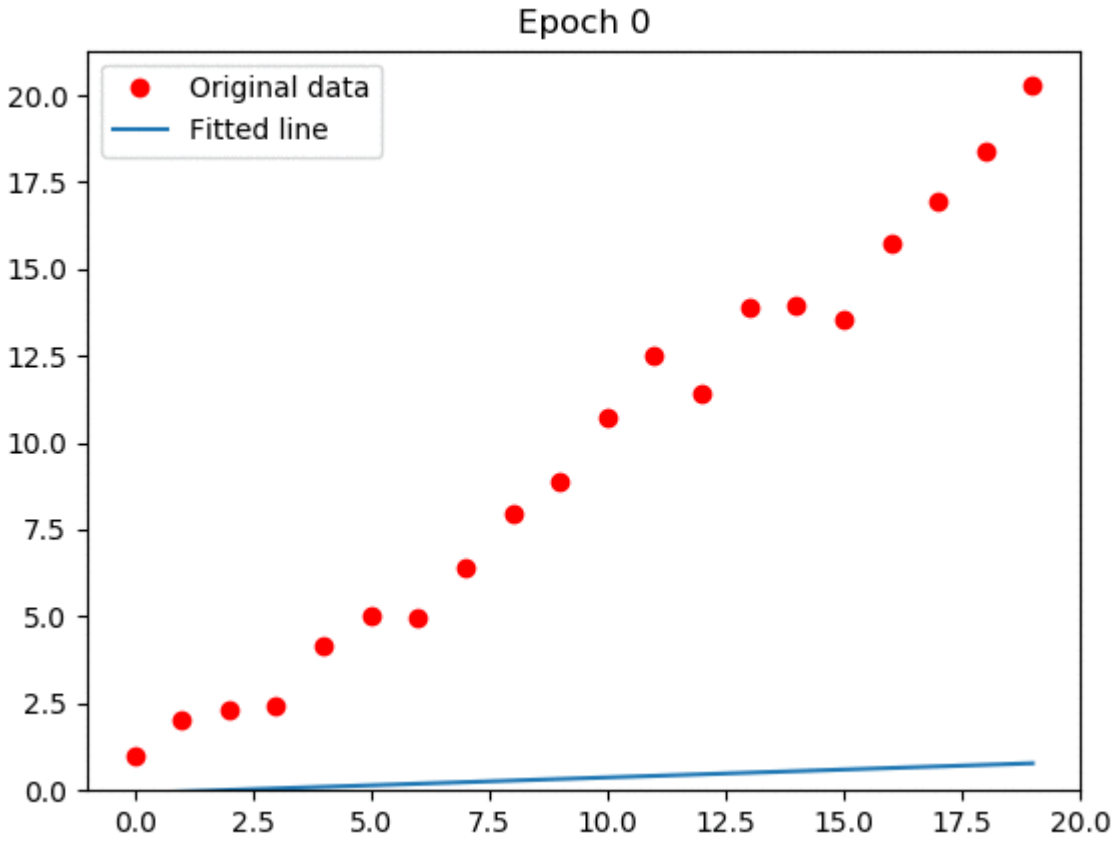
Convolution Neural Networks

1. Linear regression
2. Multilayer perceptron
3. Convolutional neural network
4. Data processing

Linear regression

Linear regression



Math behind:

1: Data $(x_1, y_1), \ldots, (x_n, y_n)$

2: Model $y = \boldsymbol{w}x + \boldsymbol{b}$

3: Loss function: mean square error

$$L(x, \boldsymbol{\theta}) = \sum_{i=1}^{n} |(\boldsymbol{w}x_i + \boldsymbol{b}) - y_i|^2$$

$$\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{b}\}$$

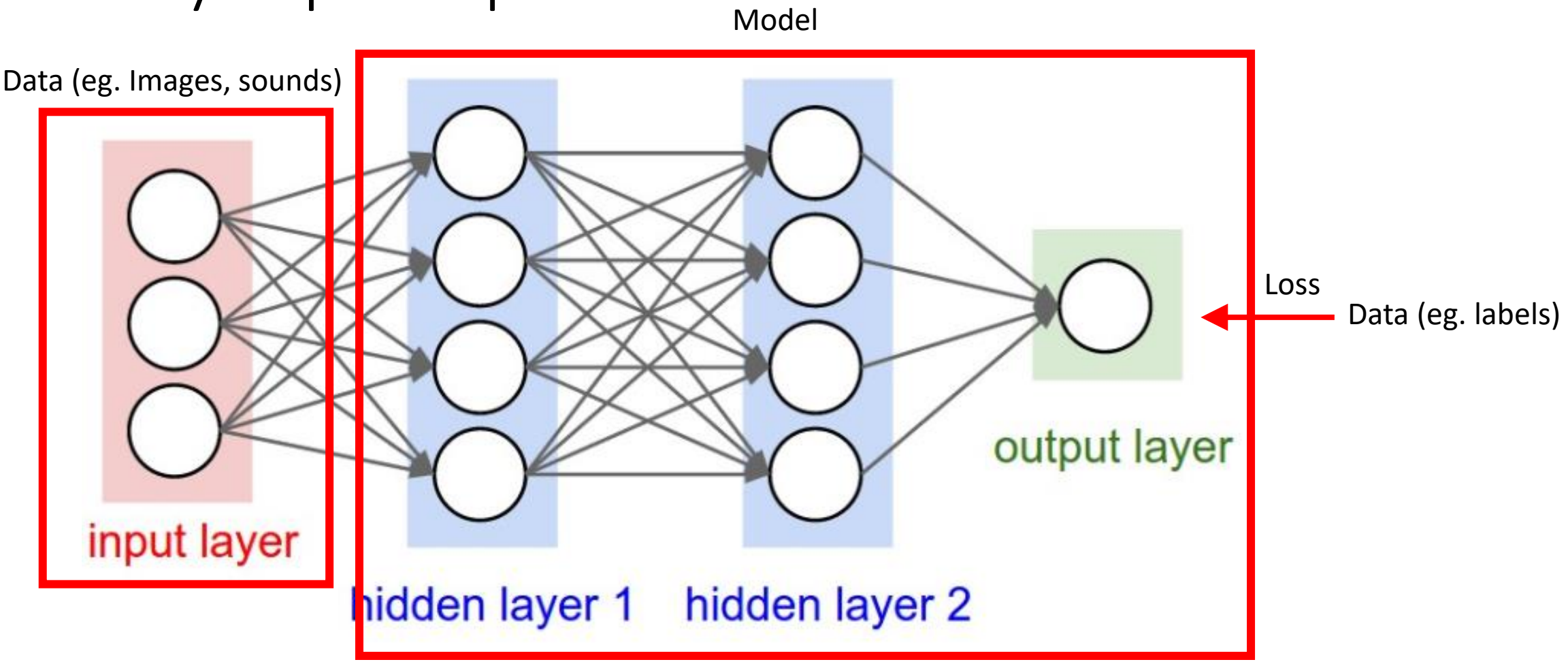
4: Optimization (training): gradient descent

$$\boldsymbol{\theta}^{j+1} = \boldsymbol{\theta}^{j} - \nabla_{\boldsymbol{\theta}} L(x, \boldsymbol{\theta}^{j})$$

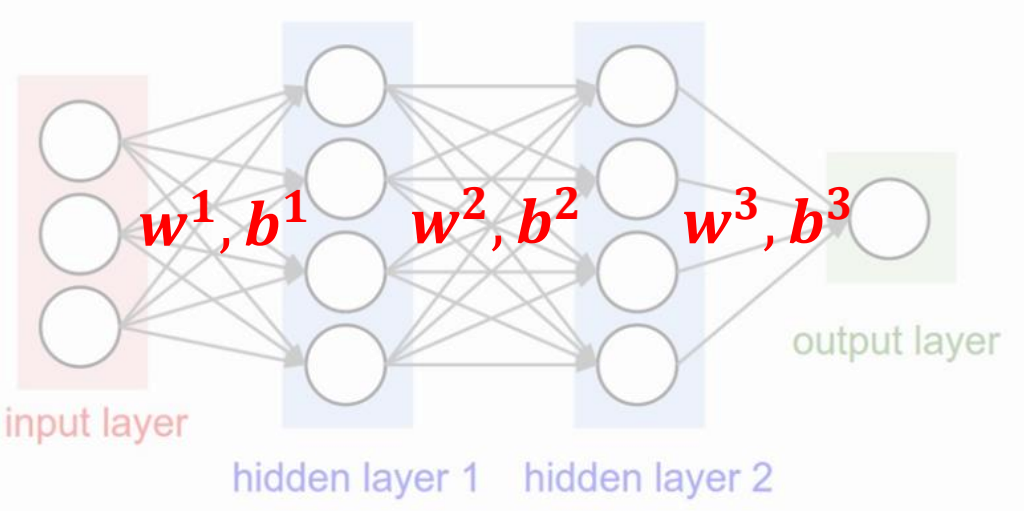
5: Inference (inference)

Multilayer perceptron

Multilayer perceptron



Multilayer perceptron



For classification

$$x^{(1)} = f\left(W^{(1)}x + b^{(1)}\right)$$
$$x^{(2)} = f\left(W^{(2)}x^{(1)} + b^{(2)}\right)$$
$$\cdots\cdots$$
$$x^{(n)} = f\left(W^{(n)}x^{(n-1)} + b^{(n)}\right)$$
$$y_i = \frac{\exp(x_i^{(n)})}{\sum_{j=1}^{L}\exp(x_j^{(n)})} \quad \text{softmax}$$

Optimisation

- Loss function

$$\min_{\theta} L(x,\theta) = -\sum_{i=1}^{L} z_i \log(y_i(x,\theta))$$

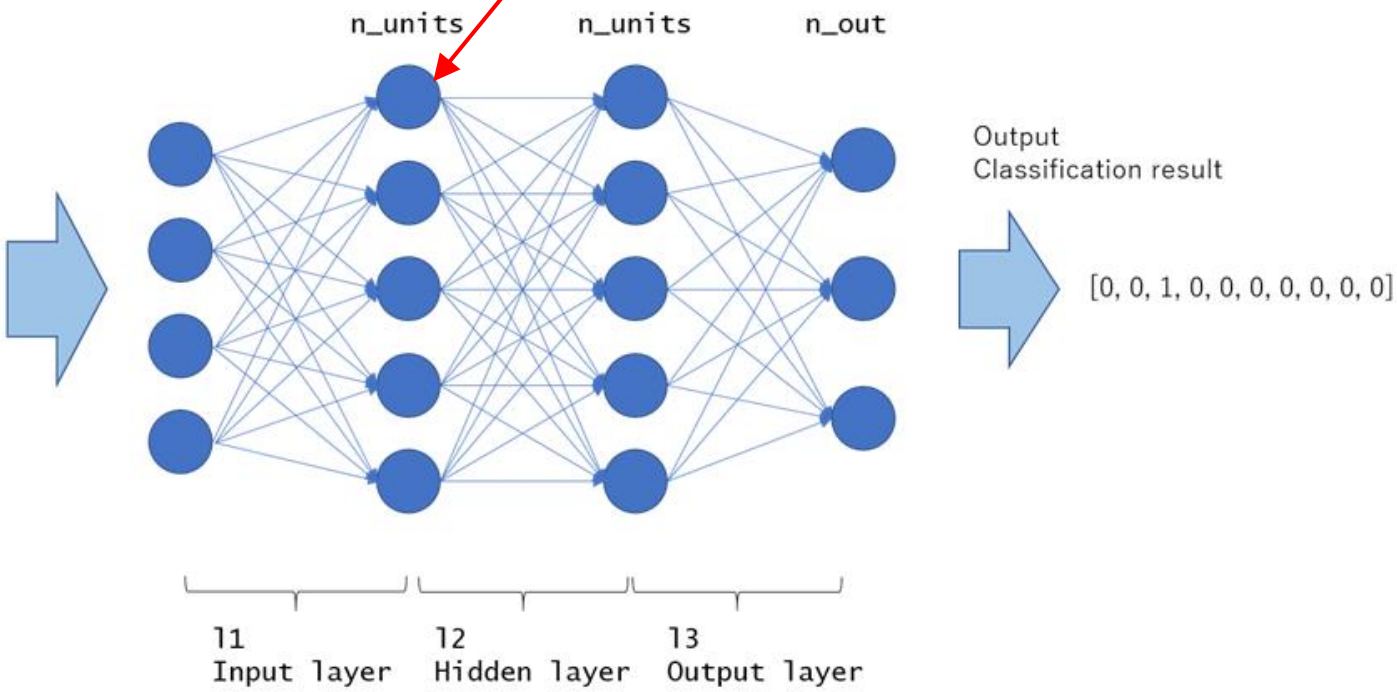
$\theta = \{W^{(i)}, b^{(i)}\}$; z_i: labels; $y_i(x,\theta)$: predicted label map

- Stochastic gradient descent (SGD)

$$\theta^{(n)} = \theta^{(n-1)} - \nabla_{\theta} L(x, \theta^{(n-1)})$$

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

500x500 pixels



Convolutional Neural Network

Dot product and convolution

The dot product of two vectors $\mathbf{a} = [a_1, a_2, \ldots, a_n]$ and $\mathbf{b} = [b_1, b_2, \ldots, b_n]$ is defined as:

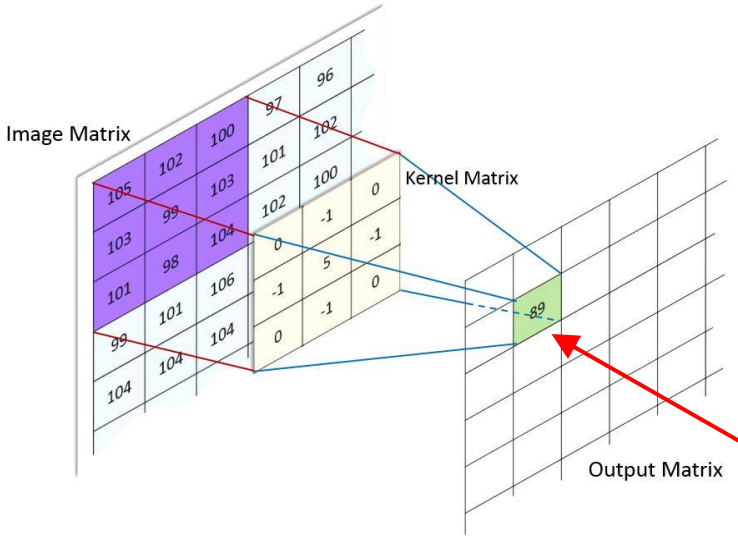
$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

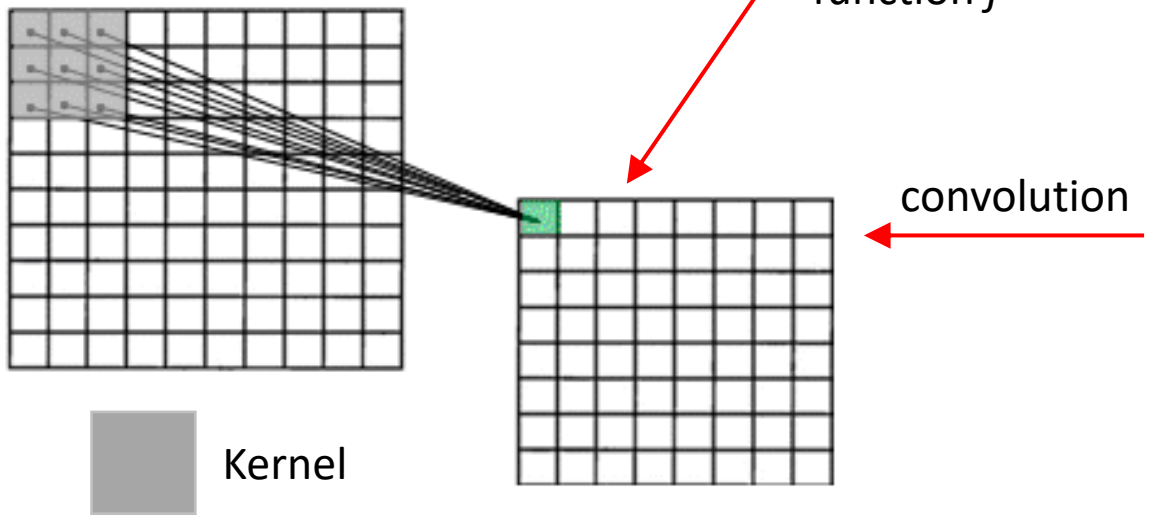
The convolution between an image x and a kernel $\boldsymbol{w}$ is given as

$$G = \boldsymbol{w} * x \qquad G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \boldsymbol{w}[u,v] x[i-u, j-v]$$

Where u, v are indices in the kernel grid and i, j are indices in the image grid. k denotes the radius of the kernel.

Convolutional Neural Network

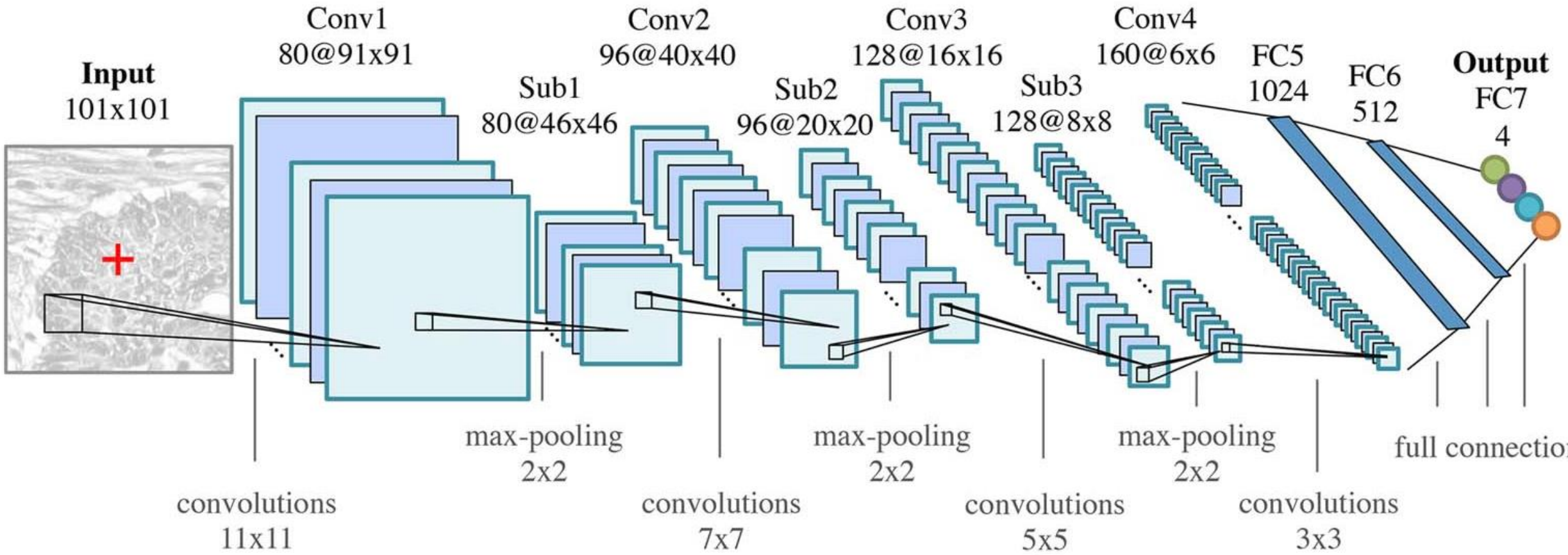




Depending on values, a kernel can cause a wide range of effects

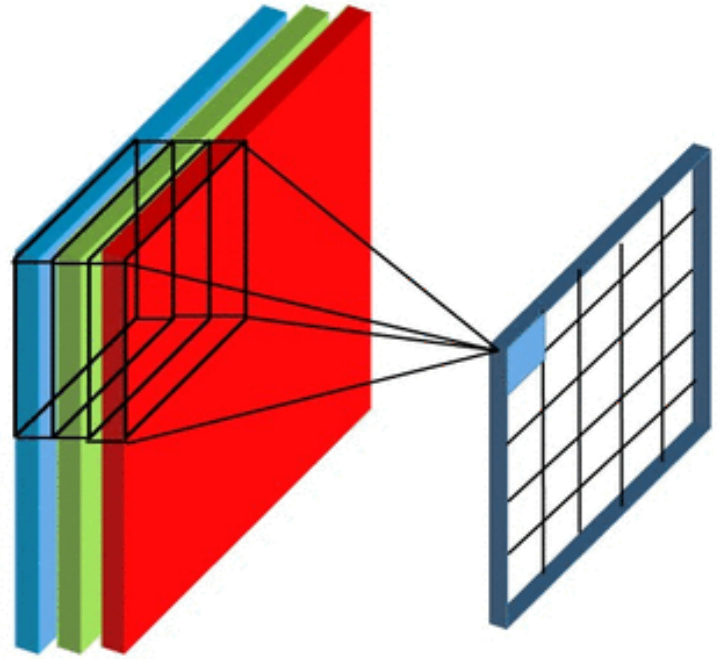
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
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)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolutional Neural Network

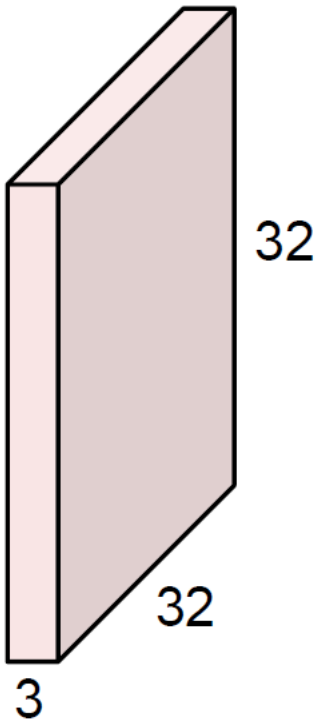


Convolution Layer

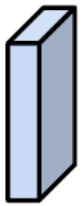
Filters always e
depth of the inp



32x32x3 image

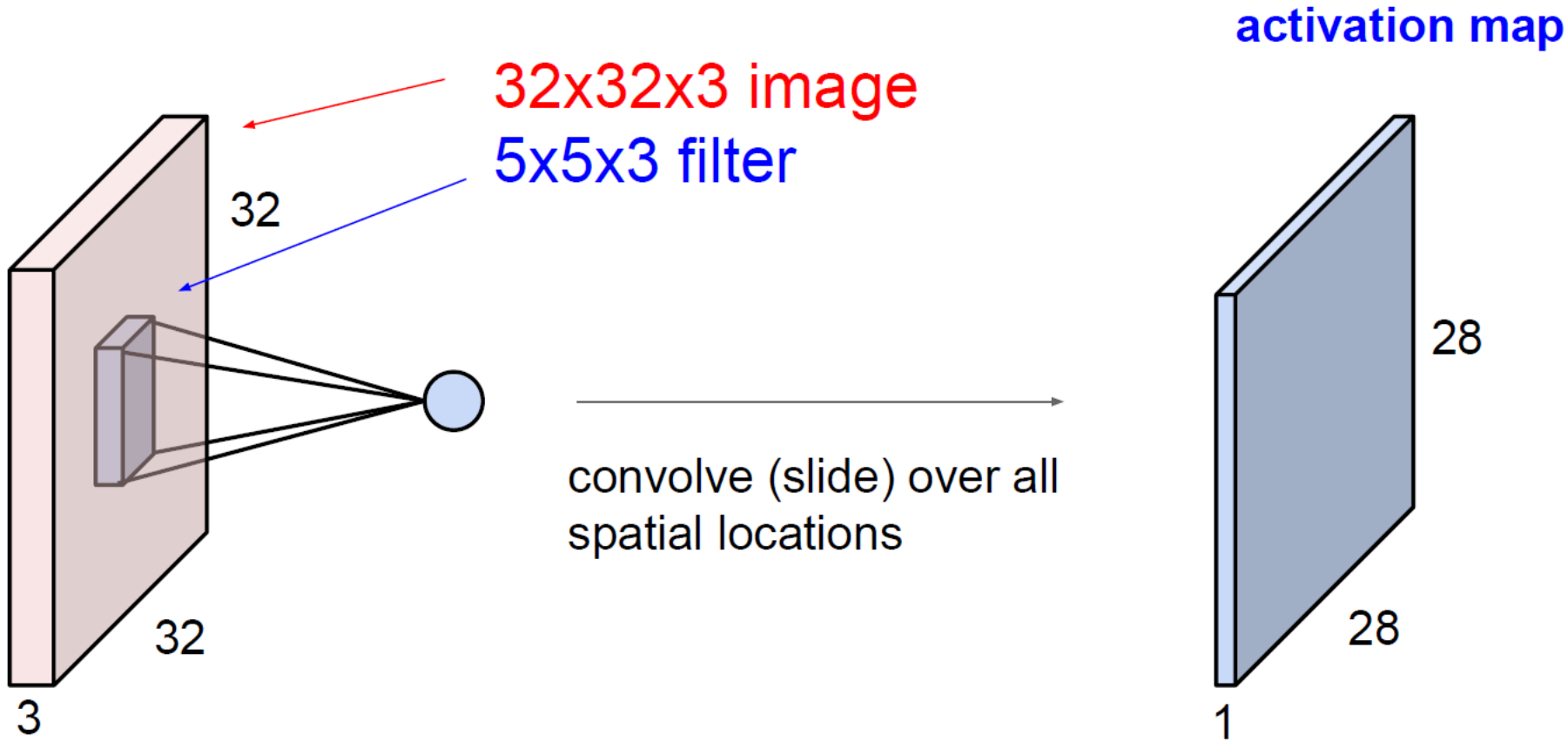


5x5x3 filter



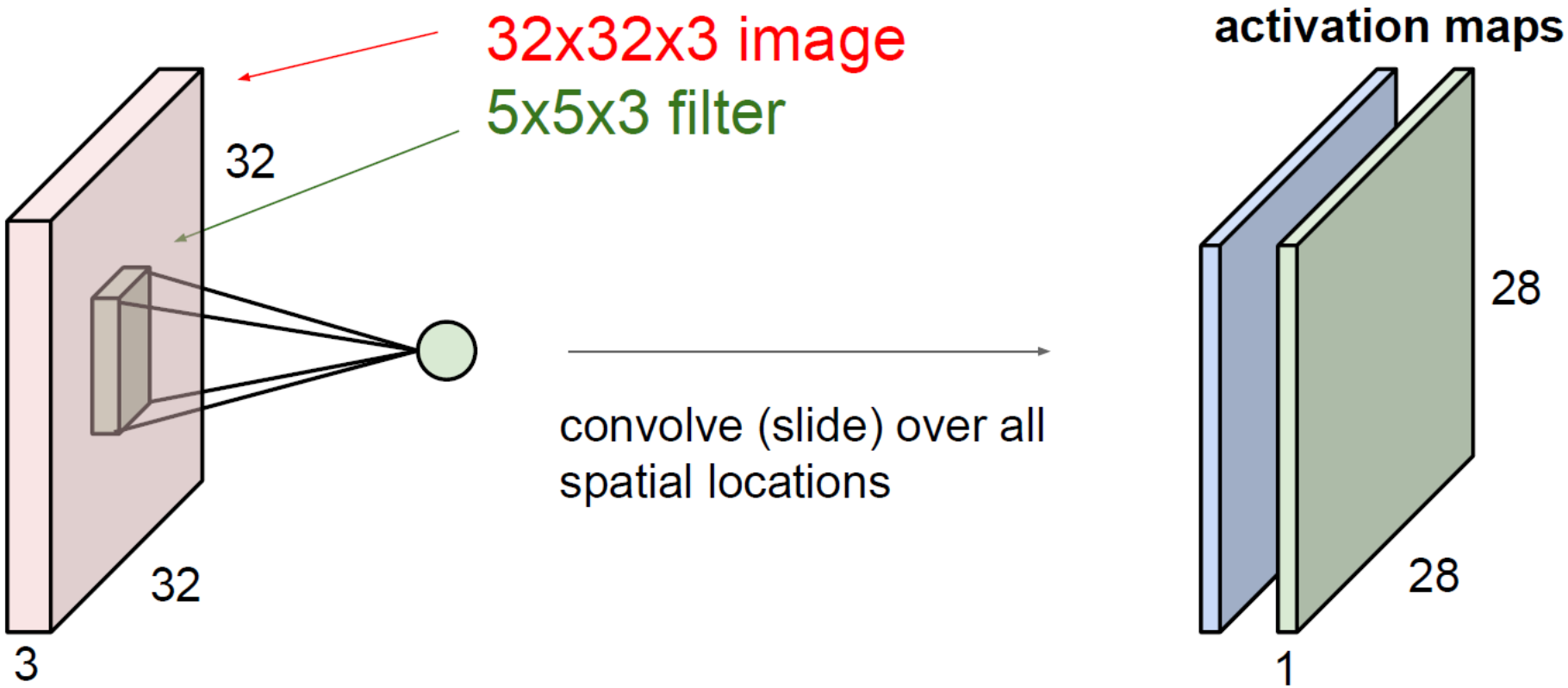
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

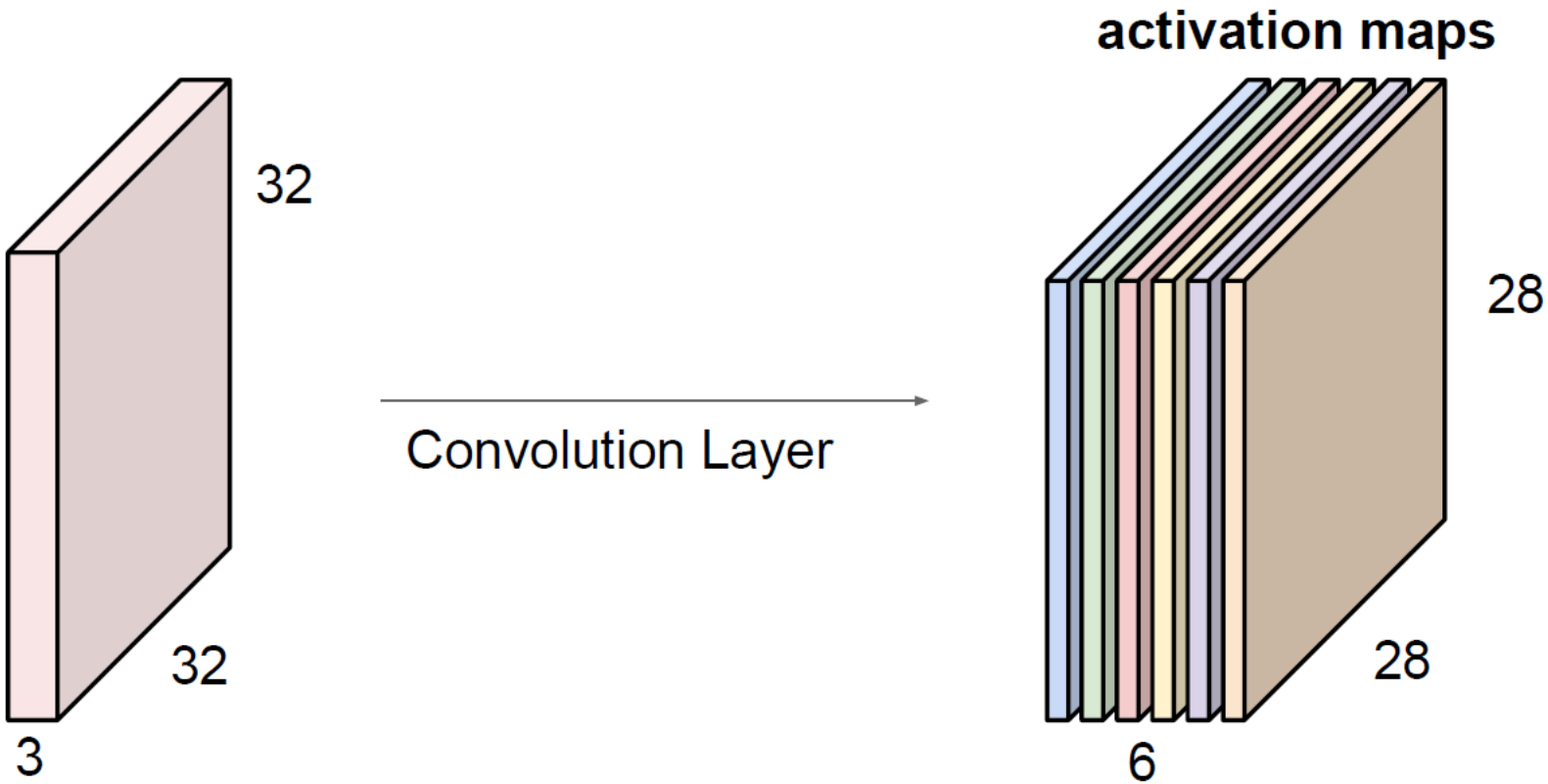


Convolution Layer

consider a second, green filter

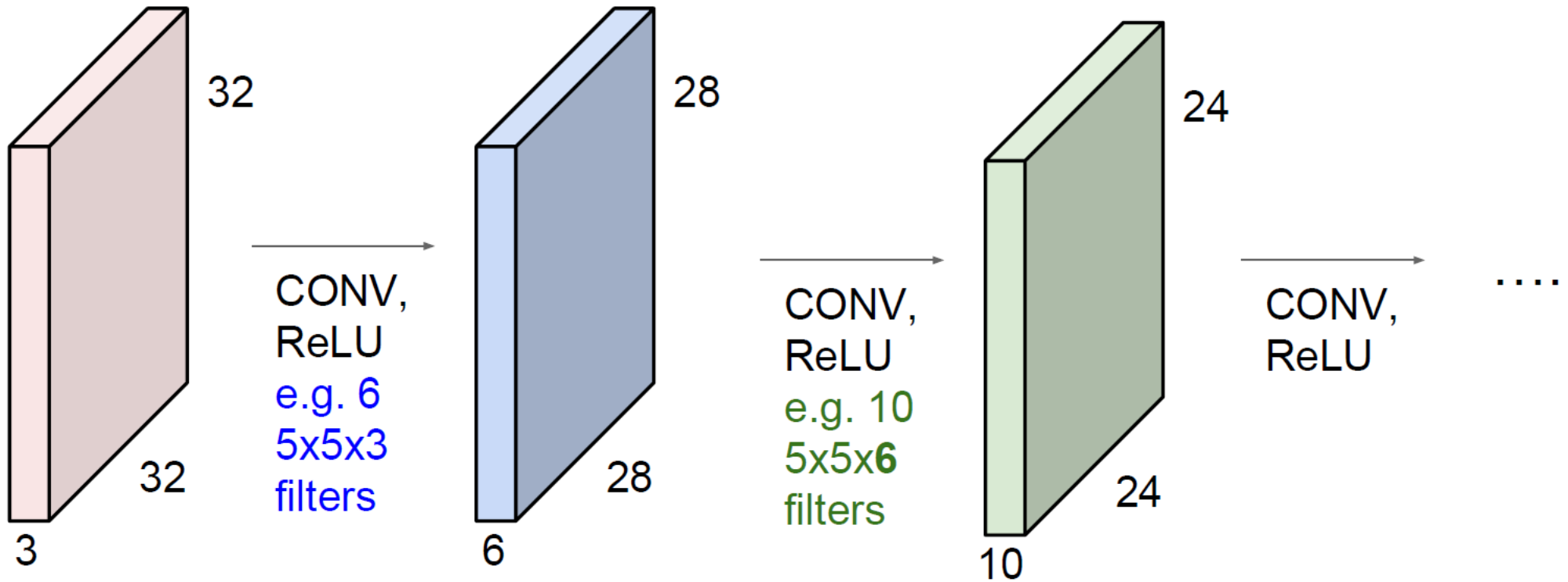


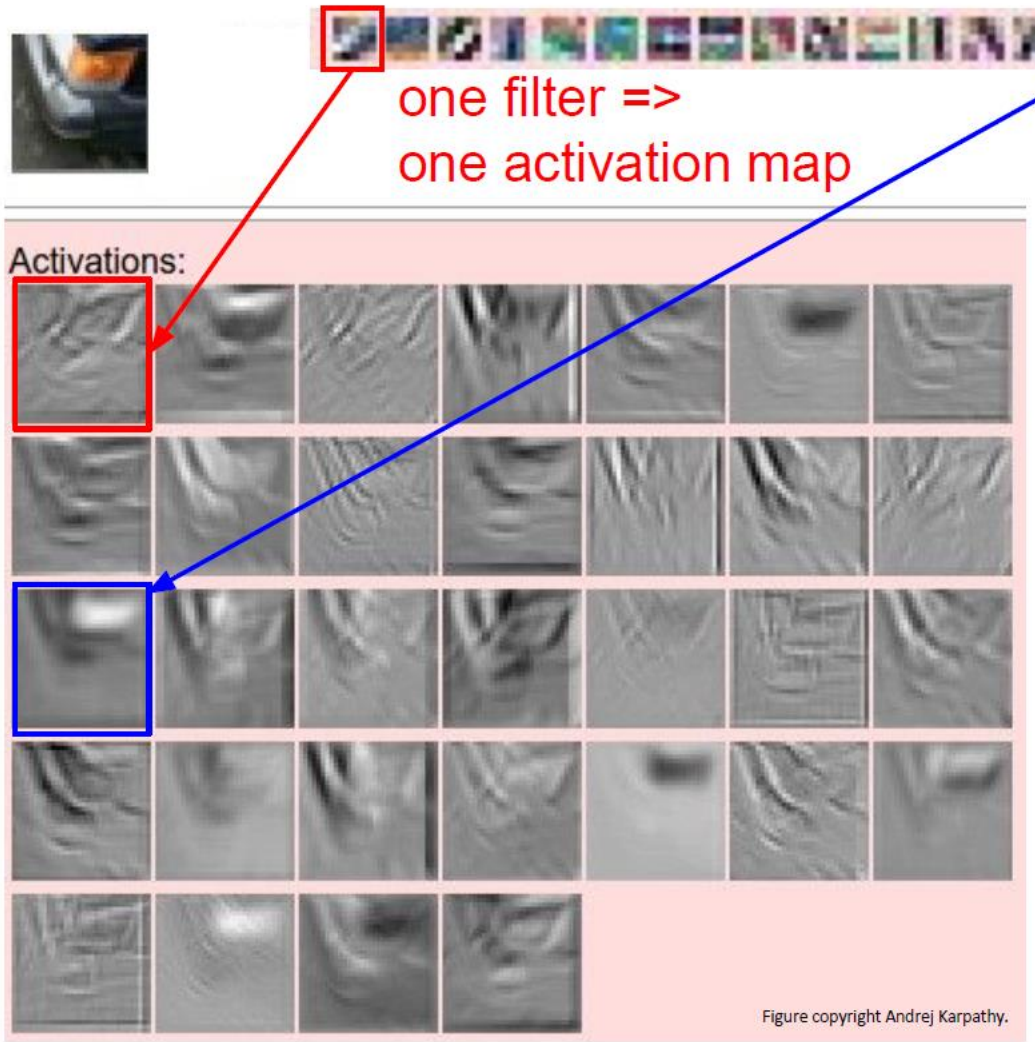
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

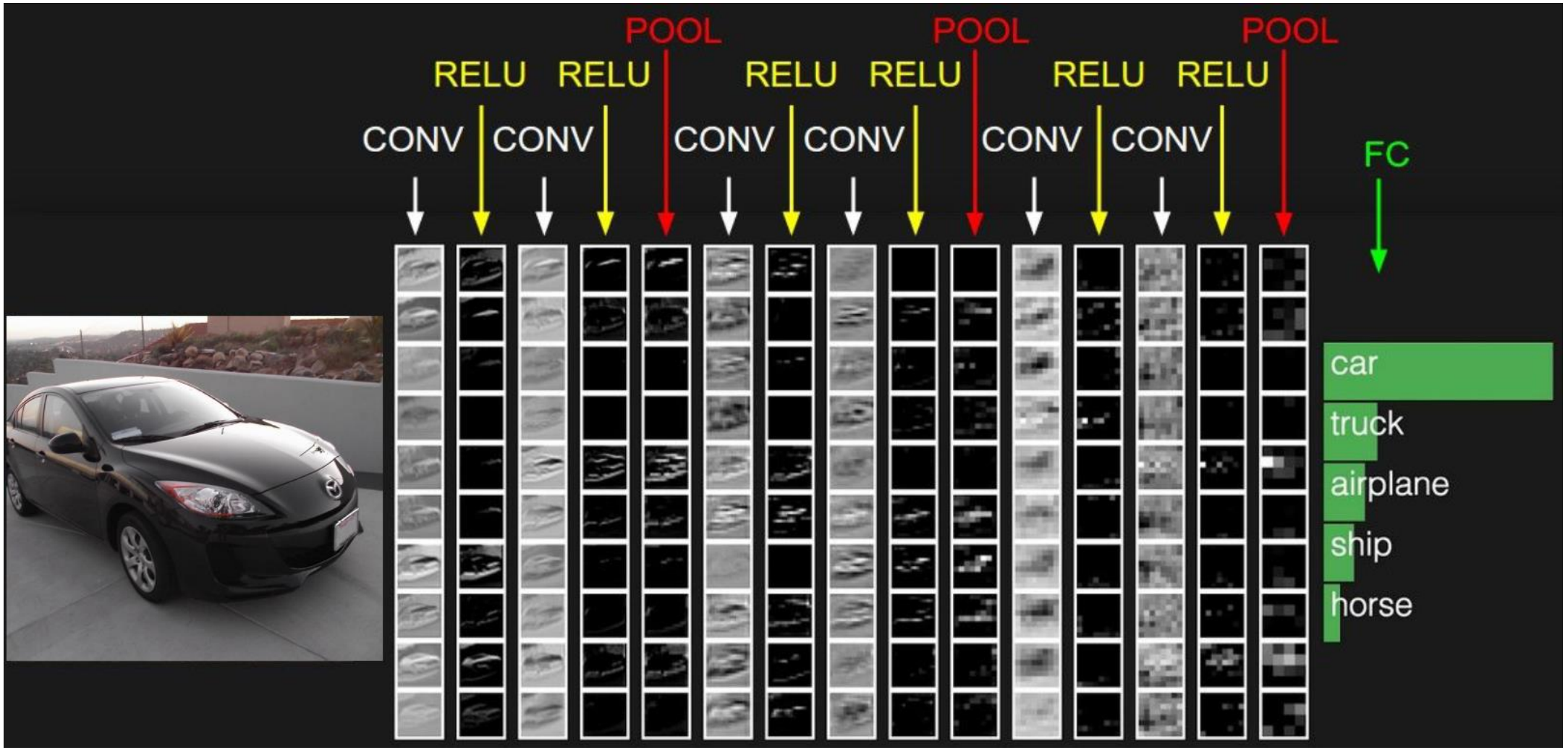
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



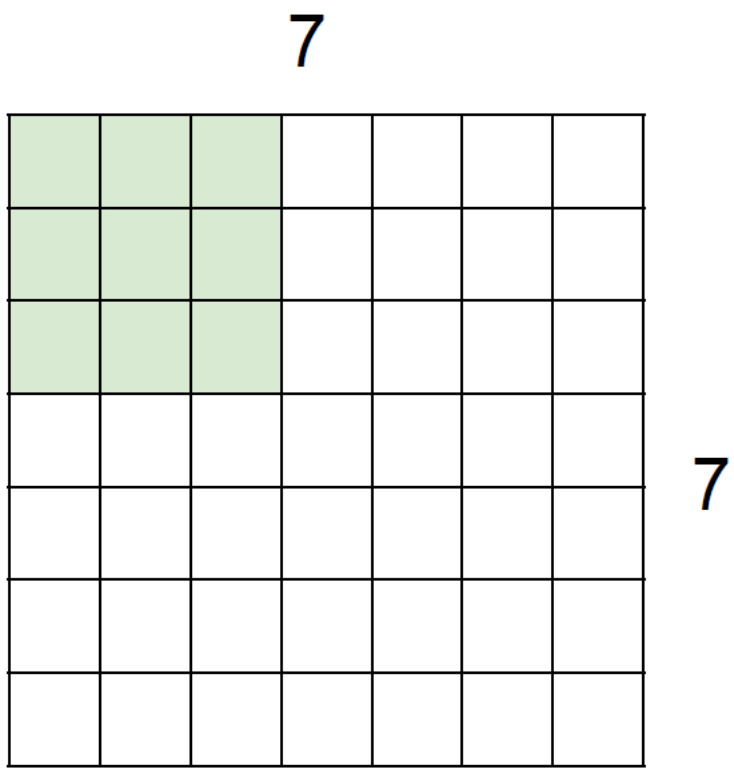


example 5x5 filters
(32 total)

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

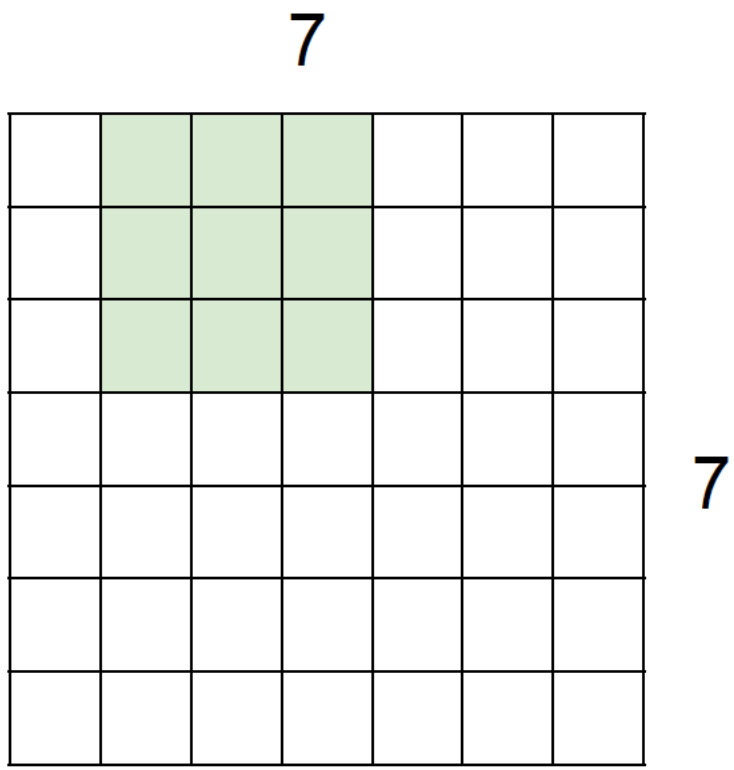

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A closer look at spatial dimensions:



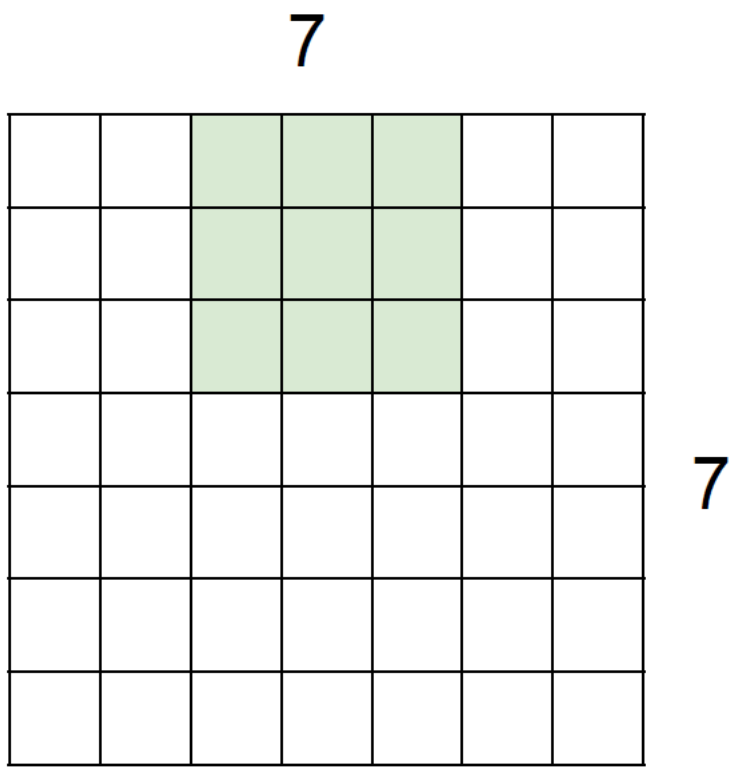
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



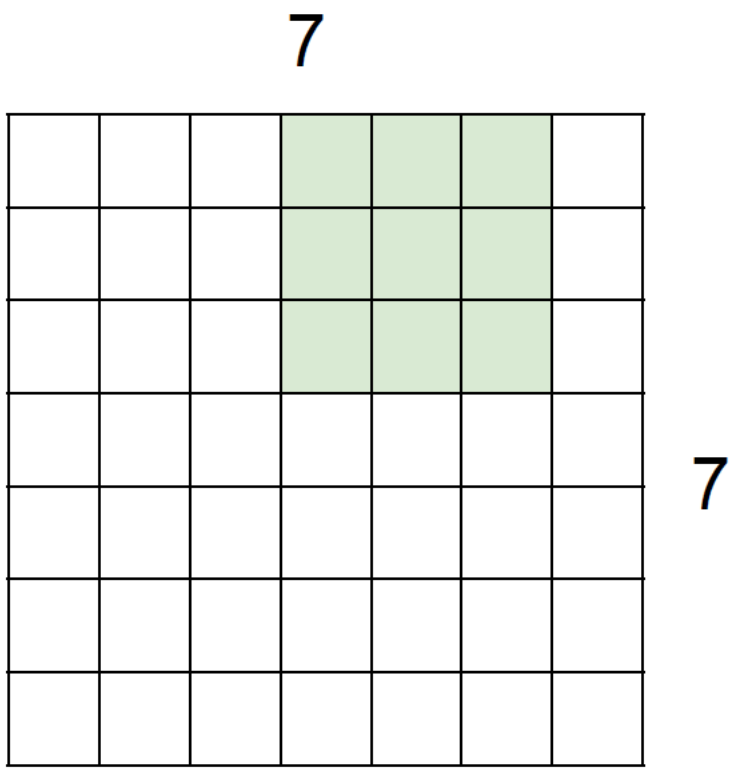
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



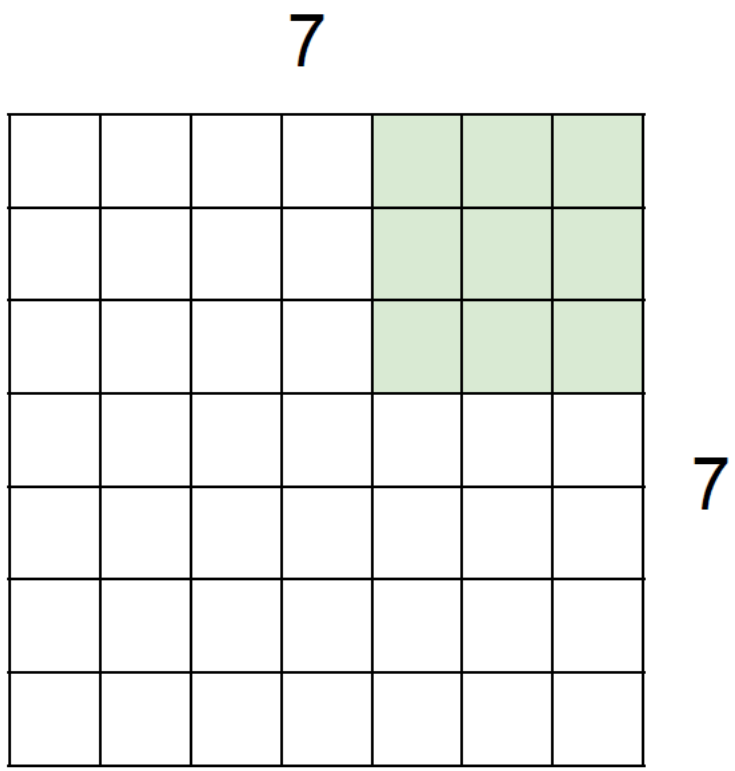
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

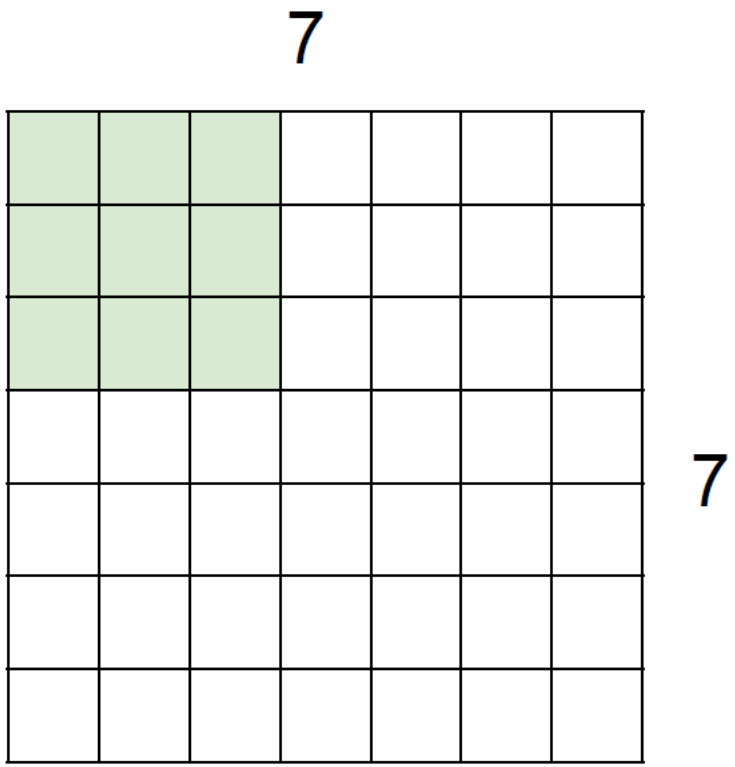
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

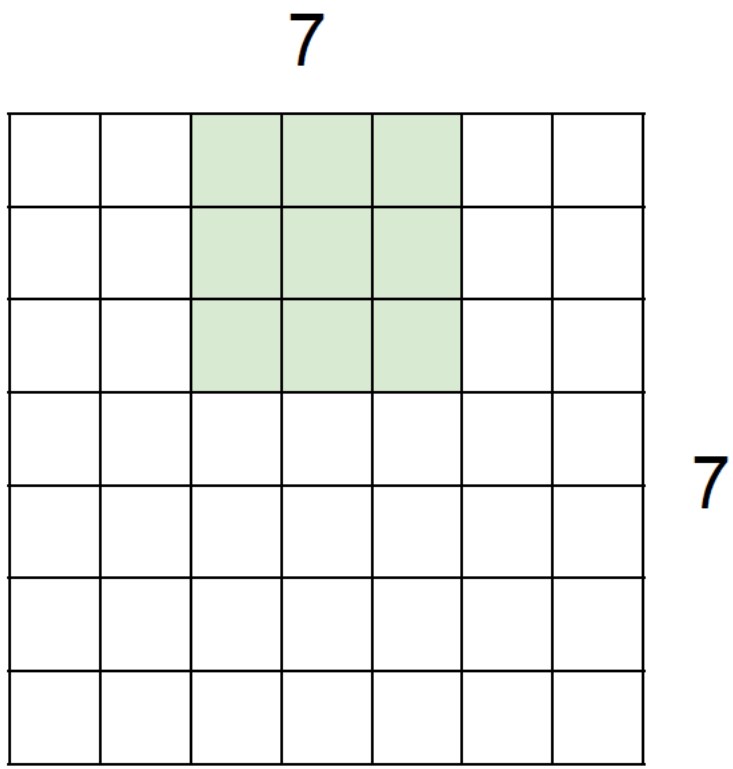
=> 5x5 output

A closer look at spatial dimensions:



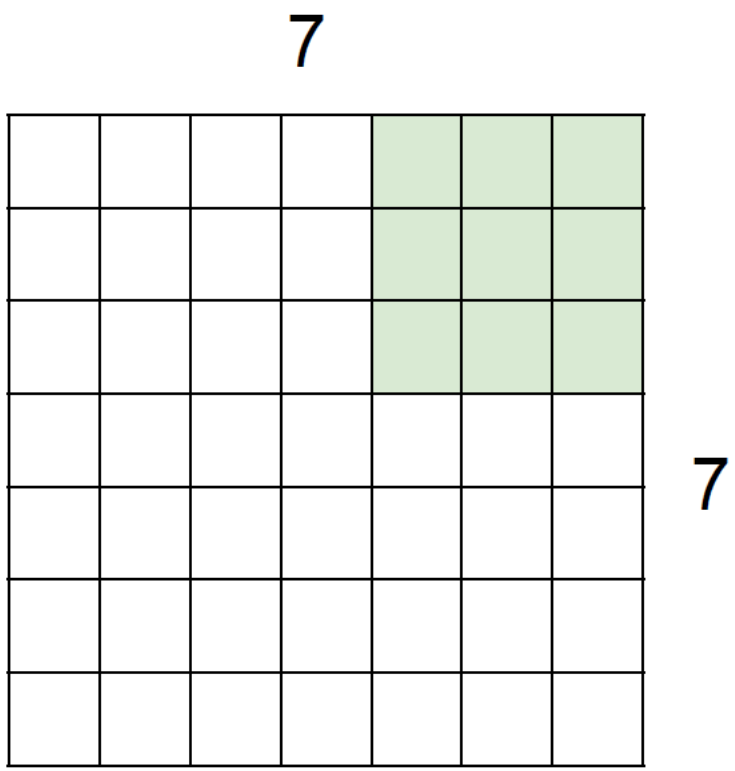
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



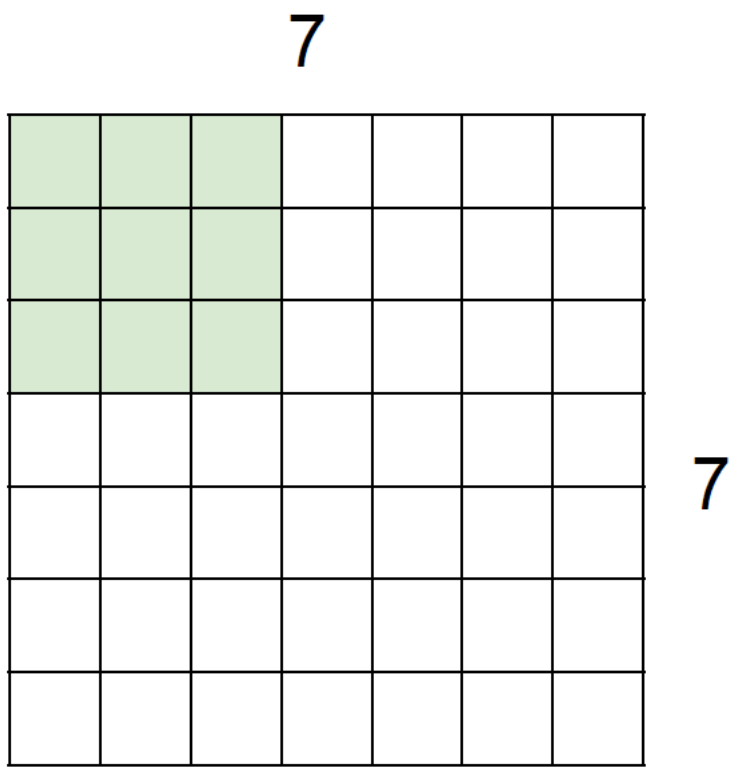
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



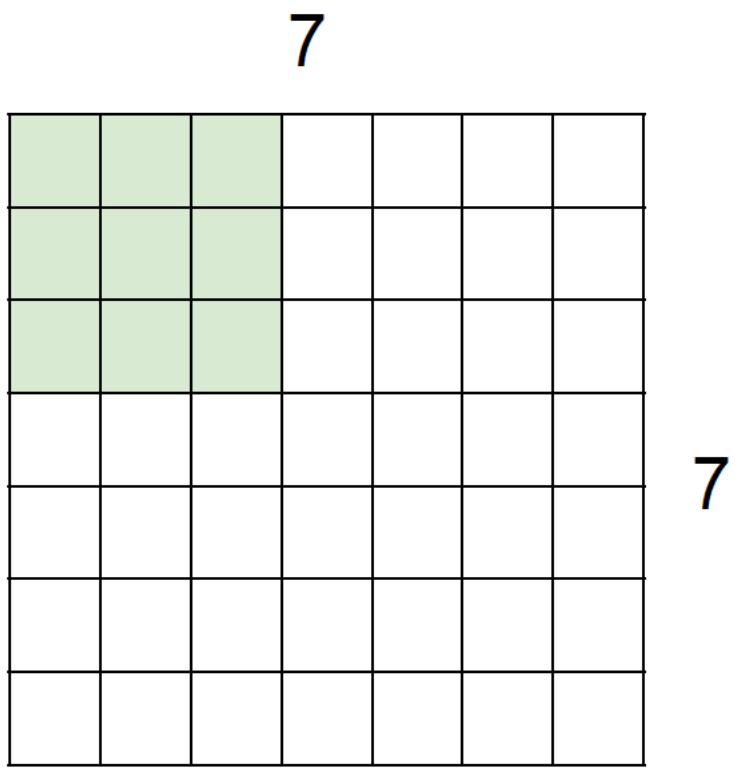
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



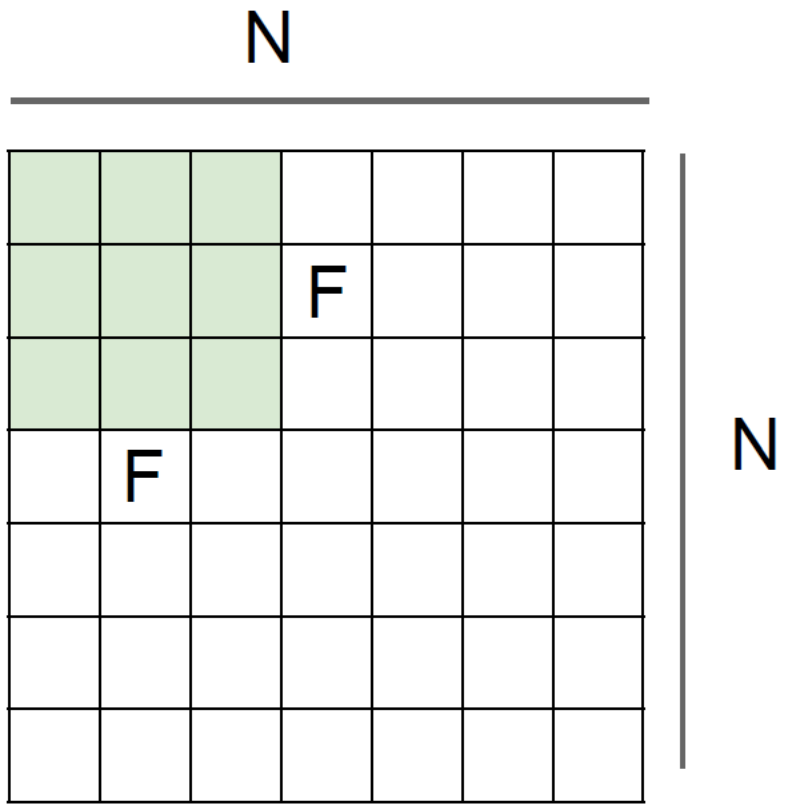
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

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.

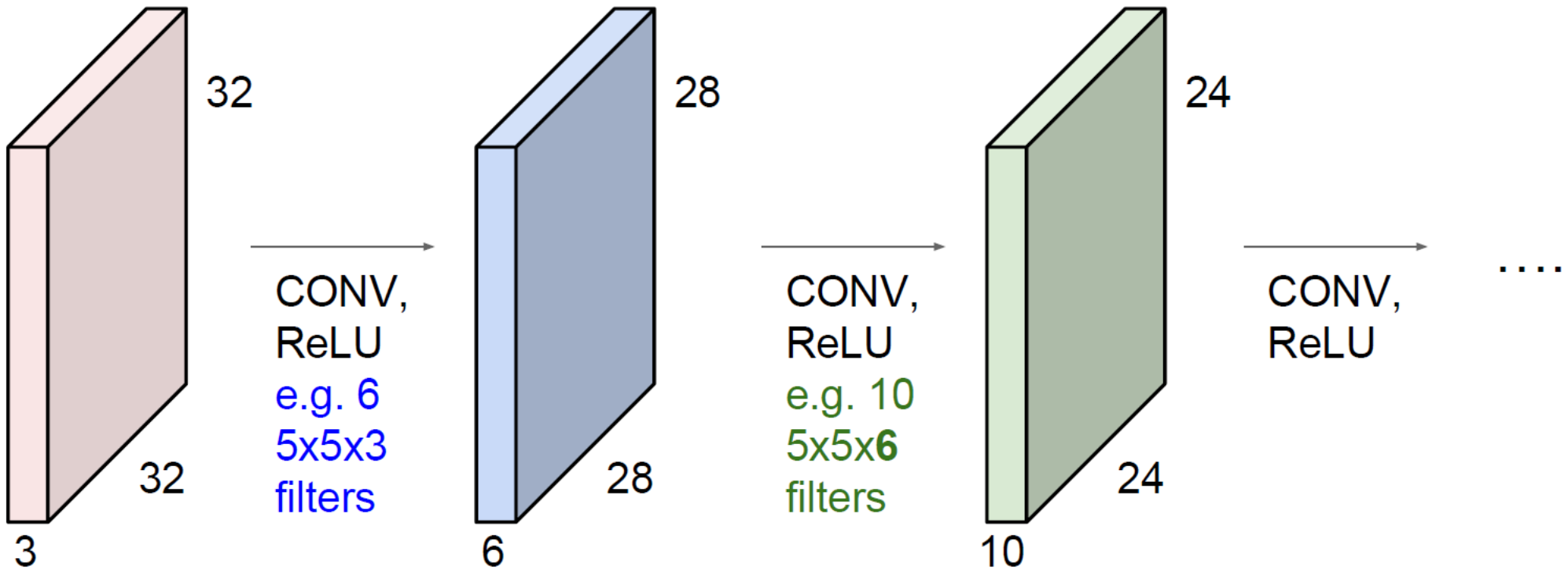


Output size:
(N - F) / stride + 1

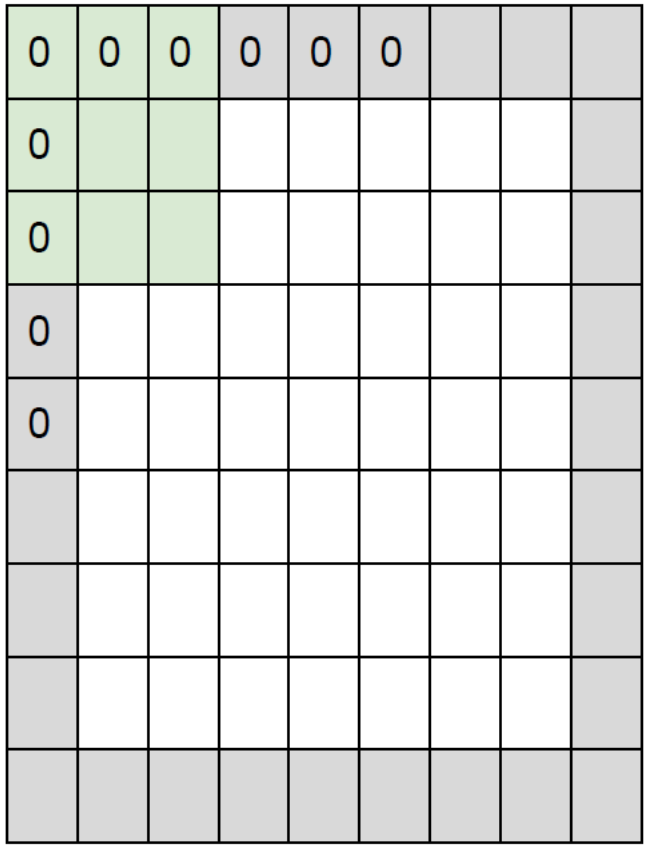
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

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.



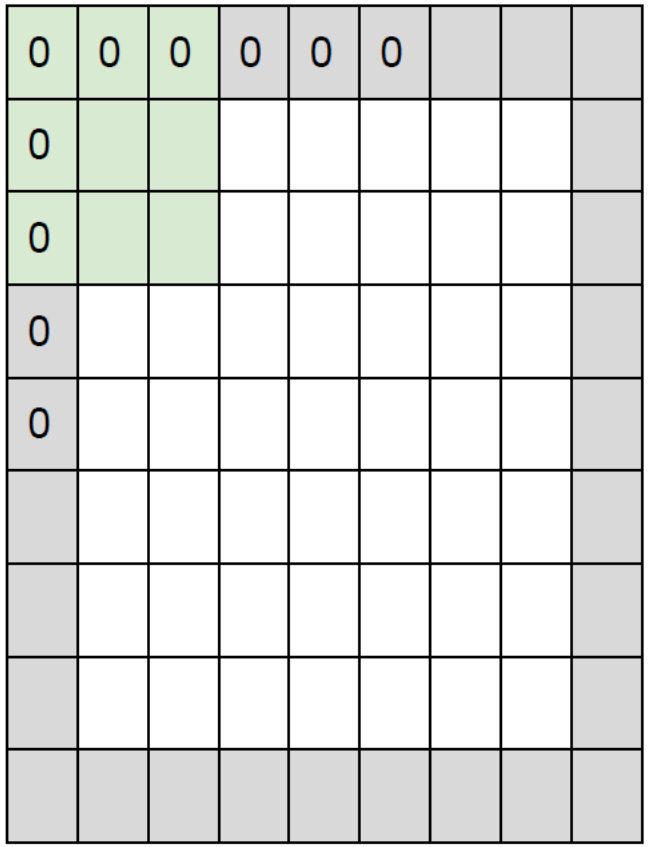
How to keep spatial size: zero paddings



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)

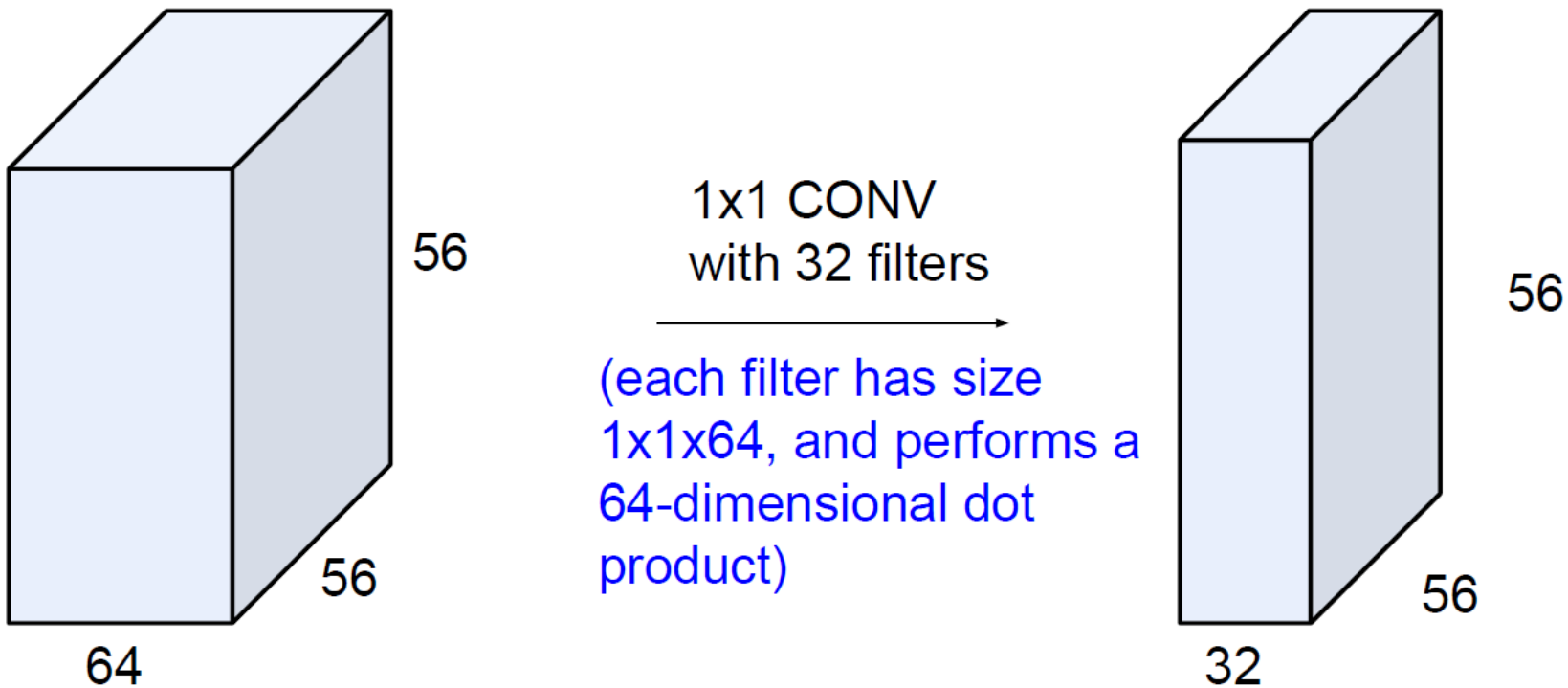
- Input a volume of size $\boldsymbol{W_1 \times H_1 \times D_1}$, convoluted with
  1. Number of filters $\boldsymbol{K}$
  2. Kernel of size $\boldsymbol{F}$
  3. Stride $\boldsymbol{S}$
  4. Number of zero-padding $\boldsymbol{P}$
- Output a volume of size $\boldsymbol{W_2 \times H_2 \times D_2}$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

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.

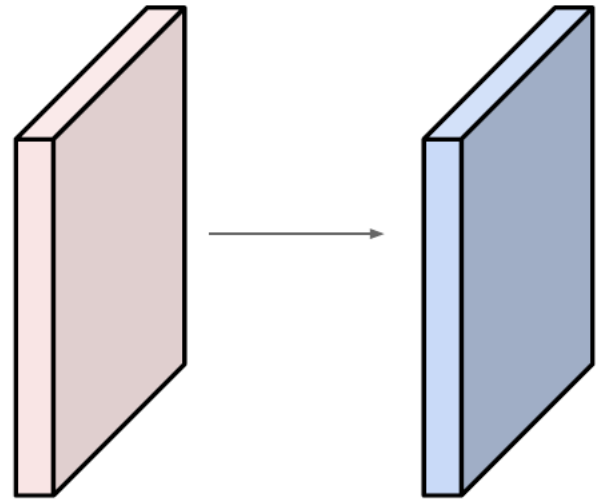
- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. 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.html

Magic formula (number of parameters)

- Input a volume of size $\boldsymbol{W_1 \times H_1 \times D_1}$, 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. $\boldsymbol{F \times F \times D_1}$ per filter
  2. $\boldsymbol{F \times F \times D_1 \times K + K}$ (total number of parameters: weights + basis)

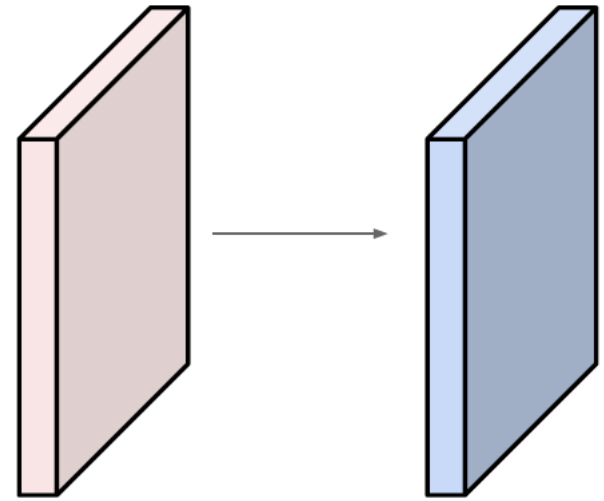
Examples time:



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

Output volume size: ?

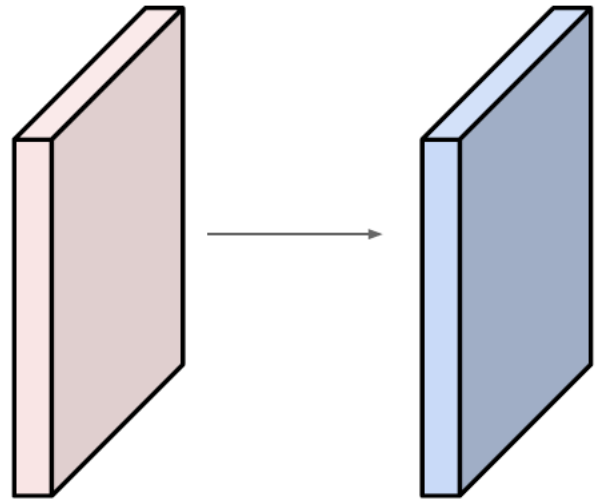
Examples time:



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

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

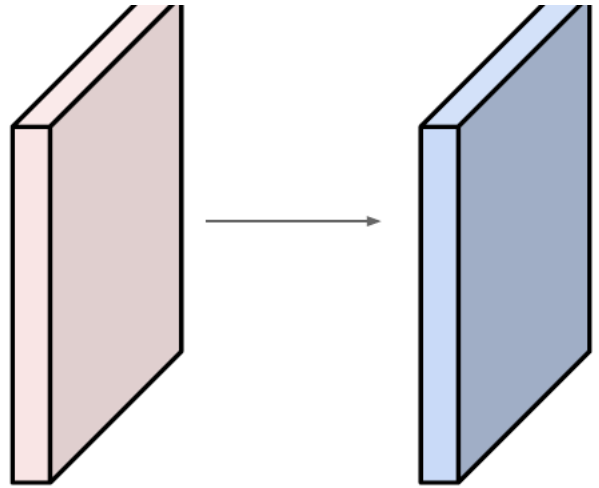
Examples time:



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

Number of parameters in this layer?

Examples time:

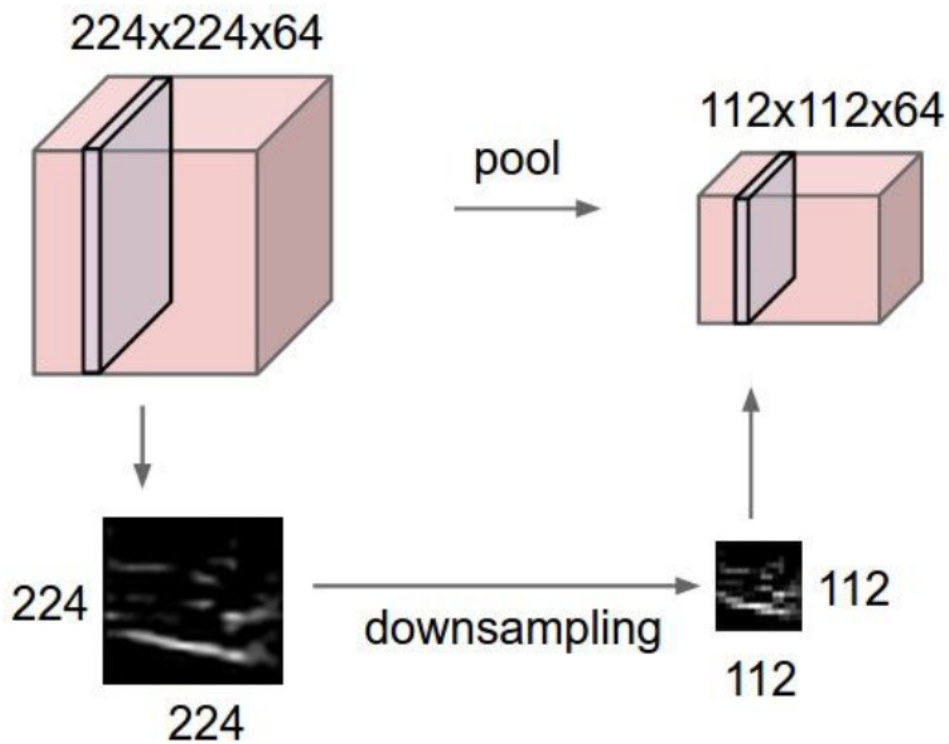


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

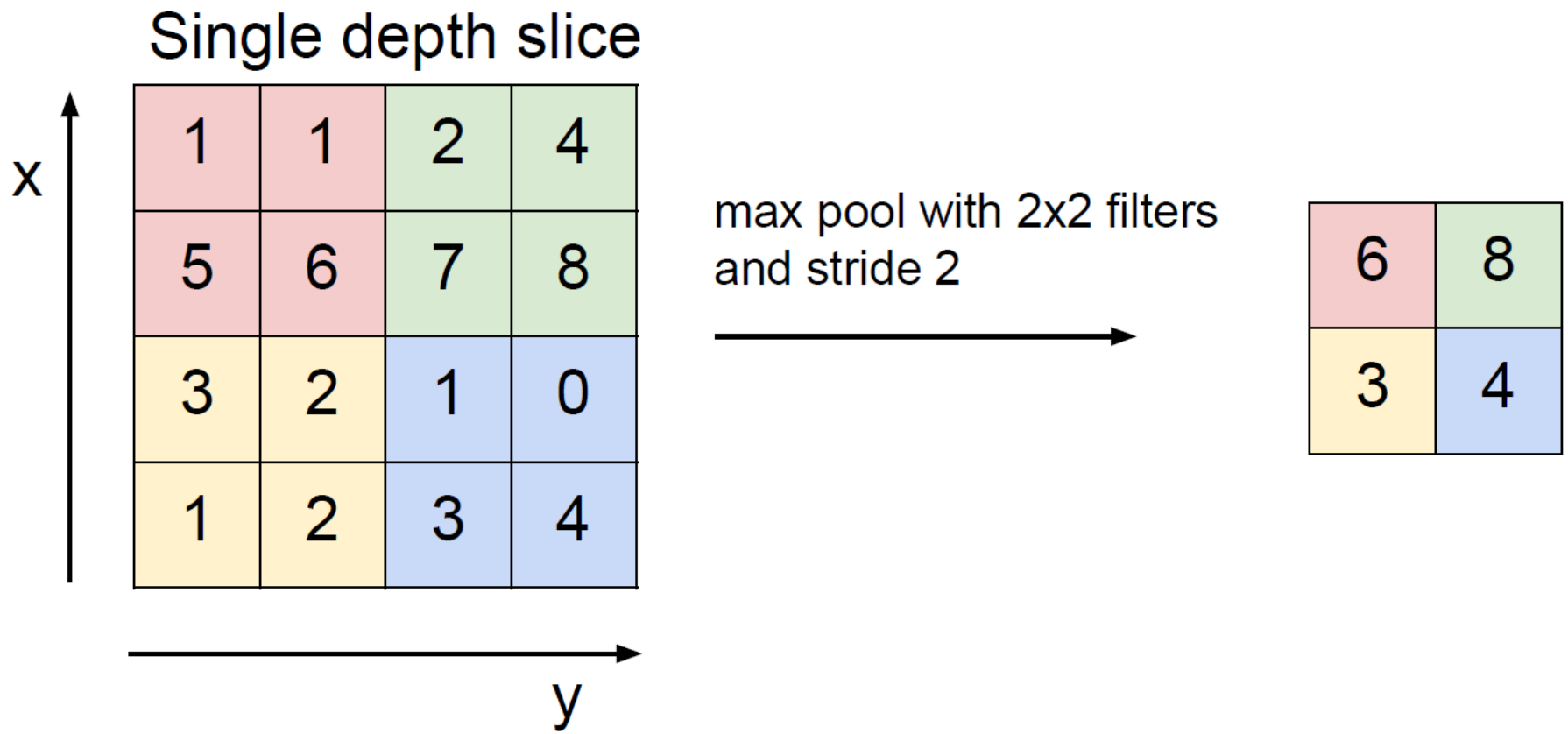
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

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)

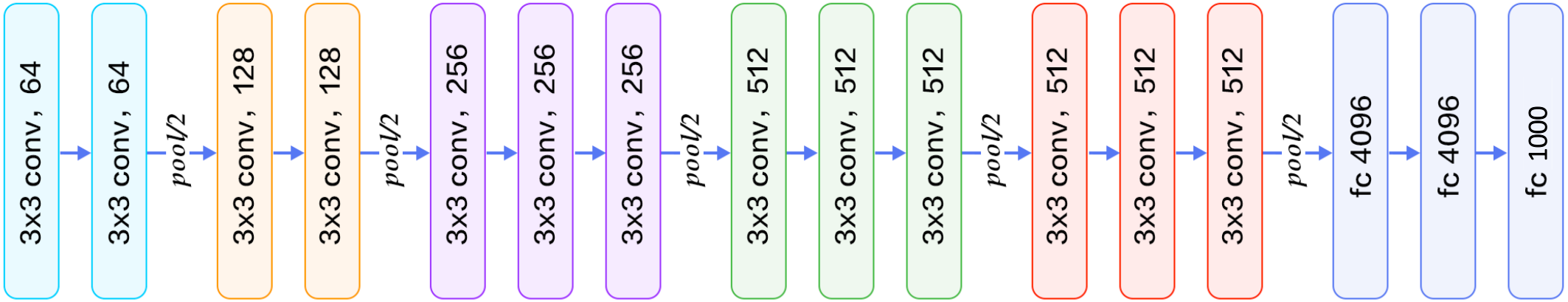
- Input a volume of size $\boldsymbol{W_1 \times H_1 \times D_1}$, convoluted with
  1. Kernel of size $\boldsymbol{F}$
  2. Stride $\boldsymbol{S}$
- Output a volume of size $\boldsymbol{W_2 \times H_2 \times D_2}$
  1. $\boldsymbol{W_2 = \frac{W_1 - F}{S} + 1}$
  2. $\boldsymbol{H_2 = \frac{H_1 - F}{S}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

Common settings:

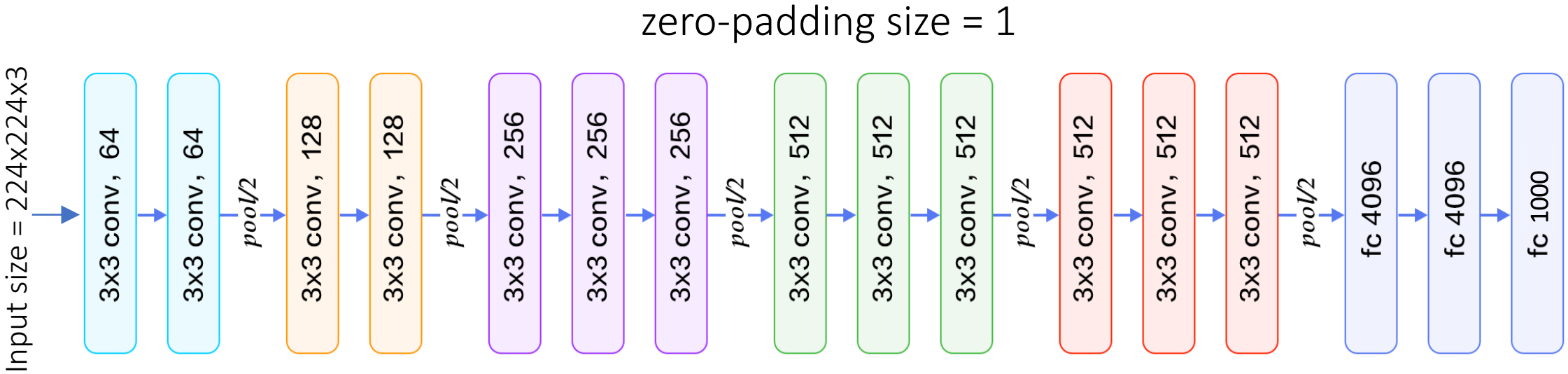
F = 2, S = 2

Case study: VGG16 for ImageNet classification

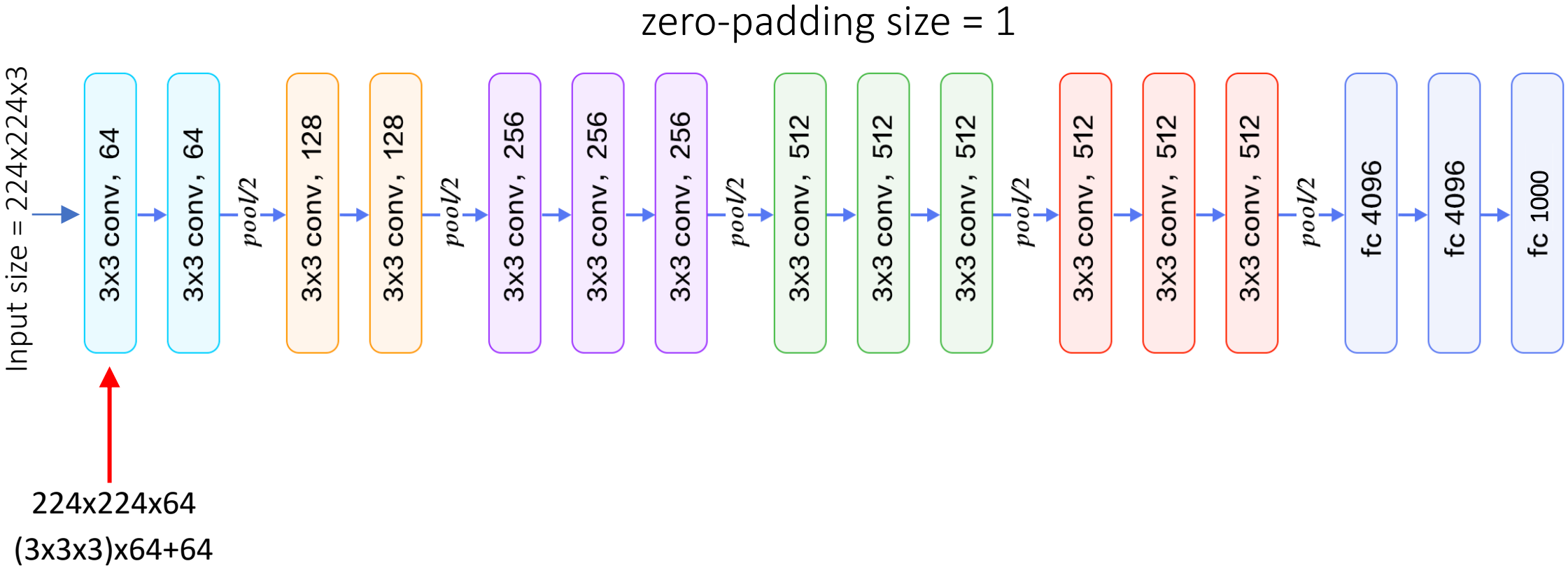
zero-padding size = 1



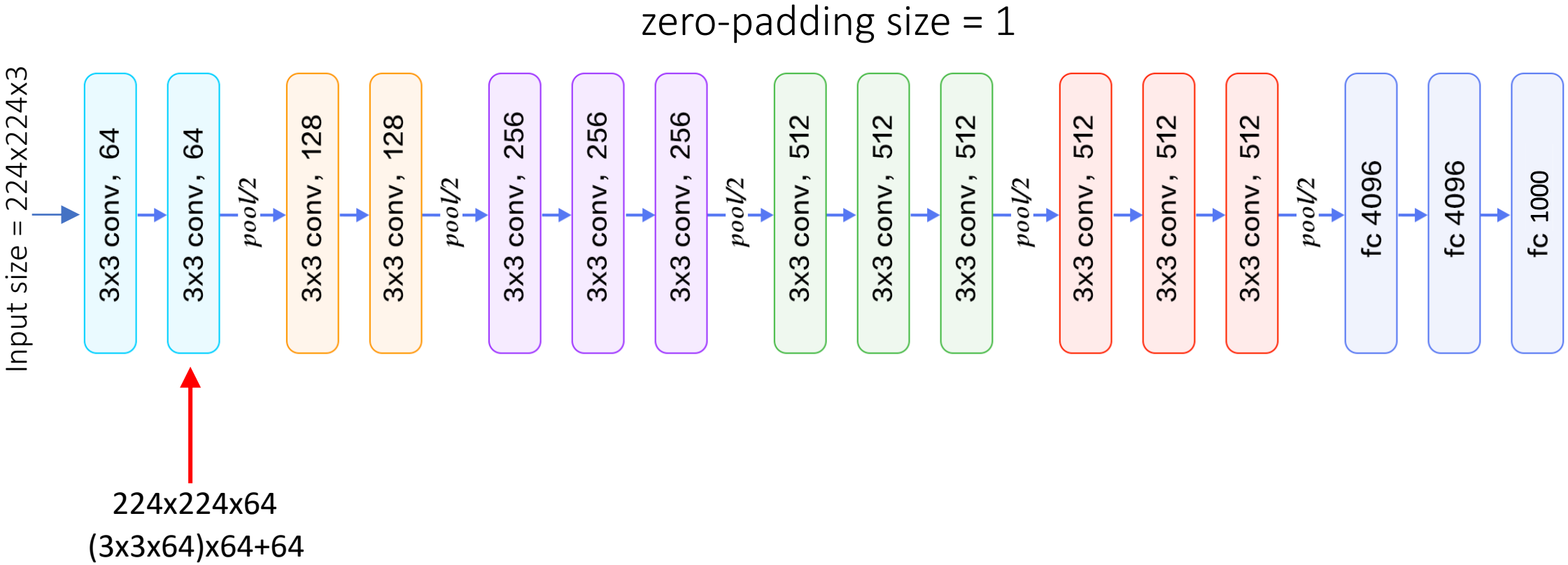
Case study: VGG16 for ImageNet classification



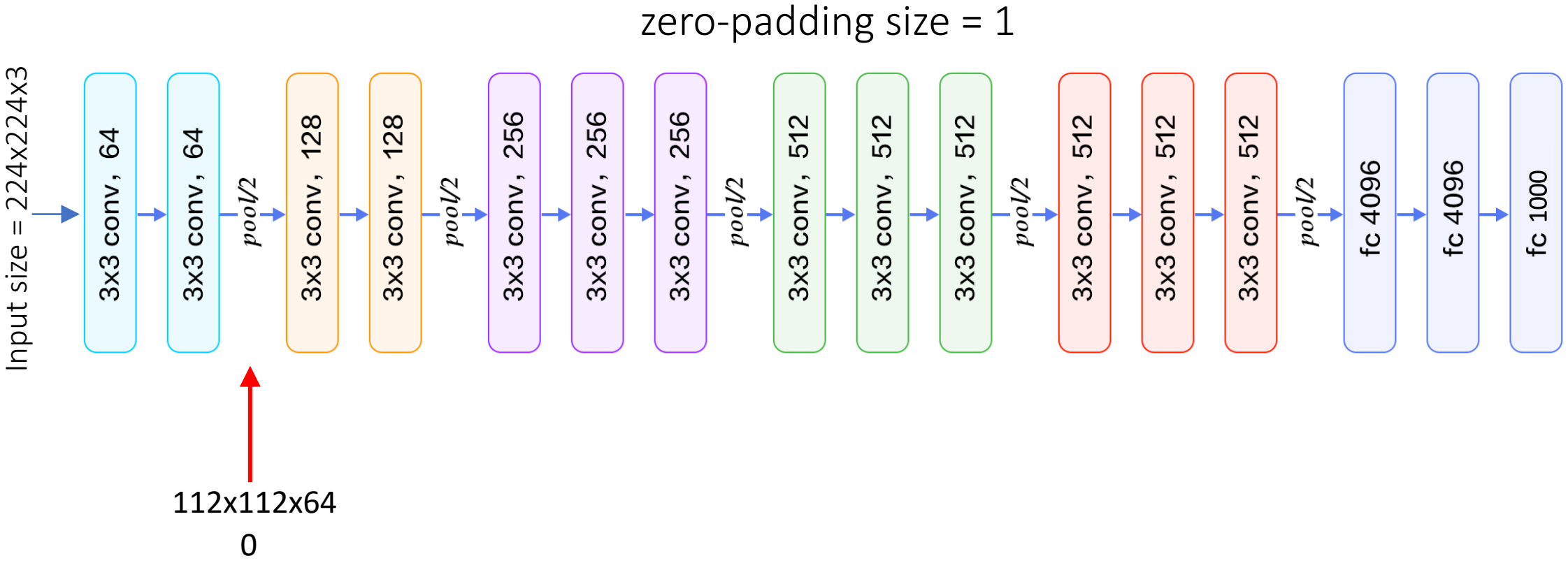
Case study: VGG16 for ImageNet classification



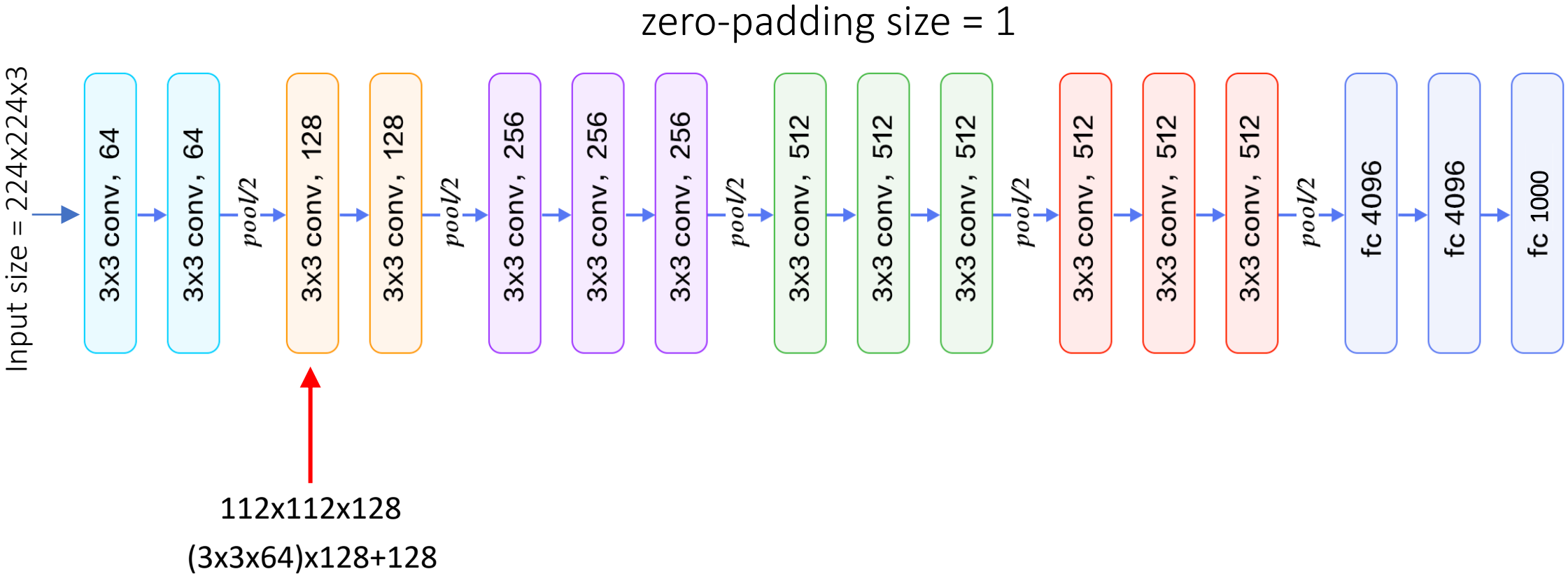
Case study: VGG16 for ImageNet classification



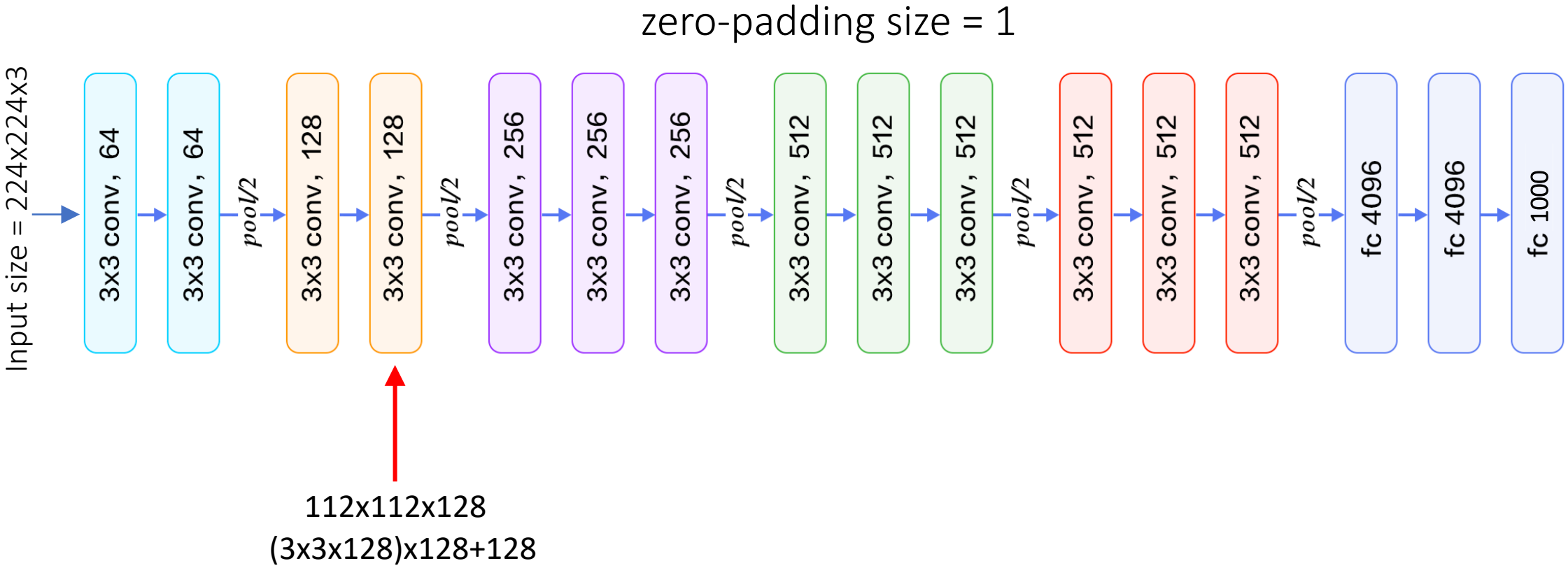
Case study: VGG16 for ImageNet classification



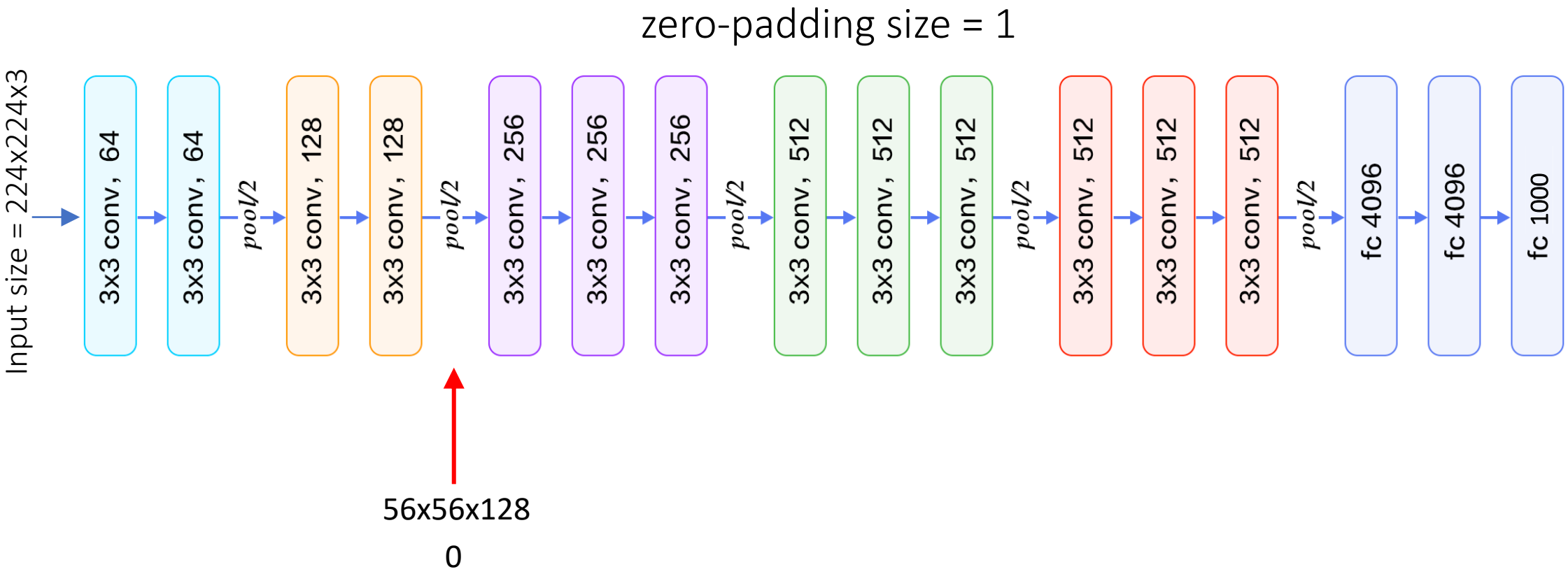
Case study: VGG16 for ImageNet classification



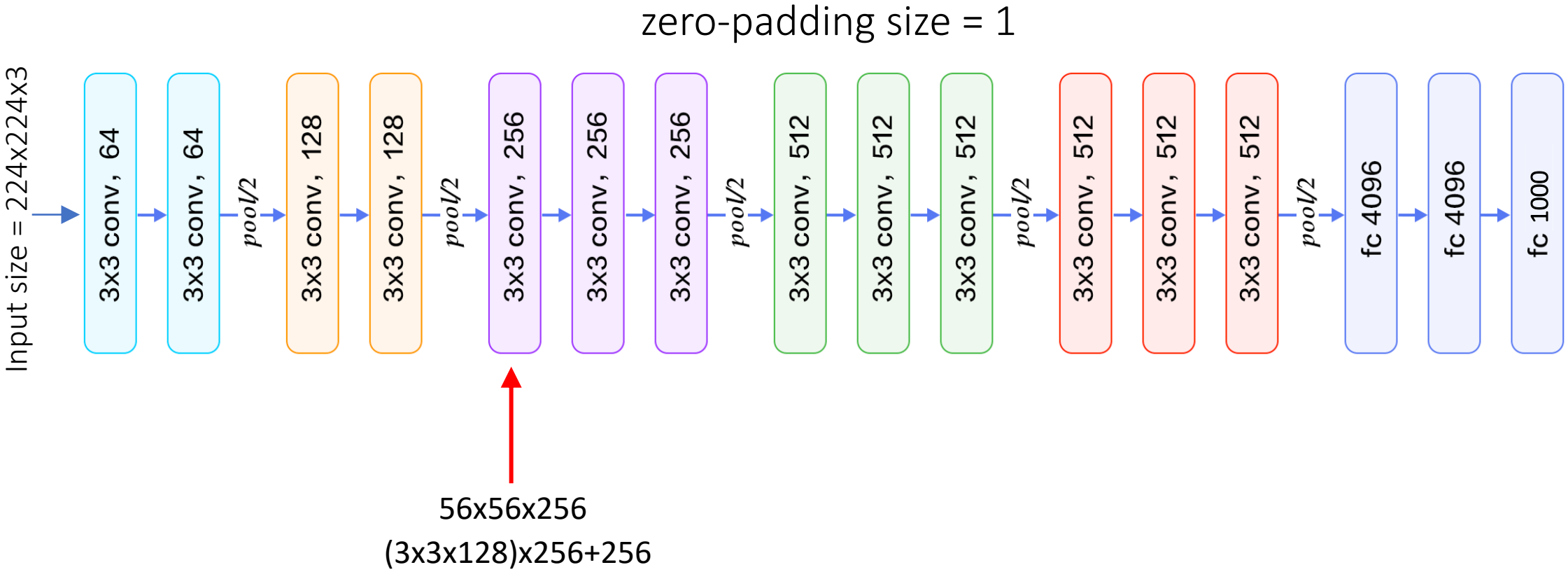
Case study: VGG16 for ImageNet classification



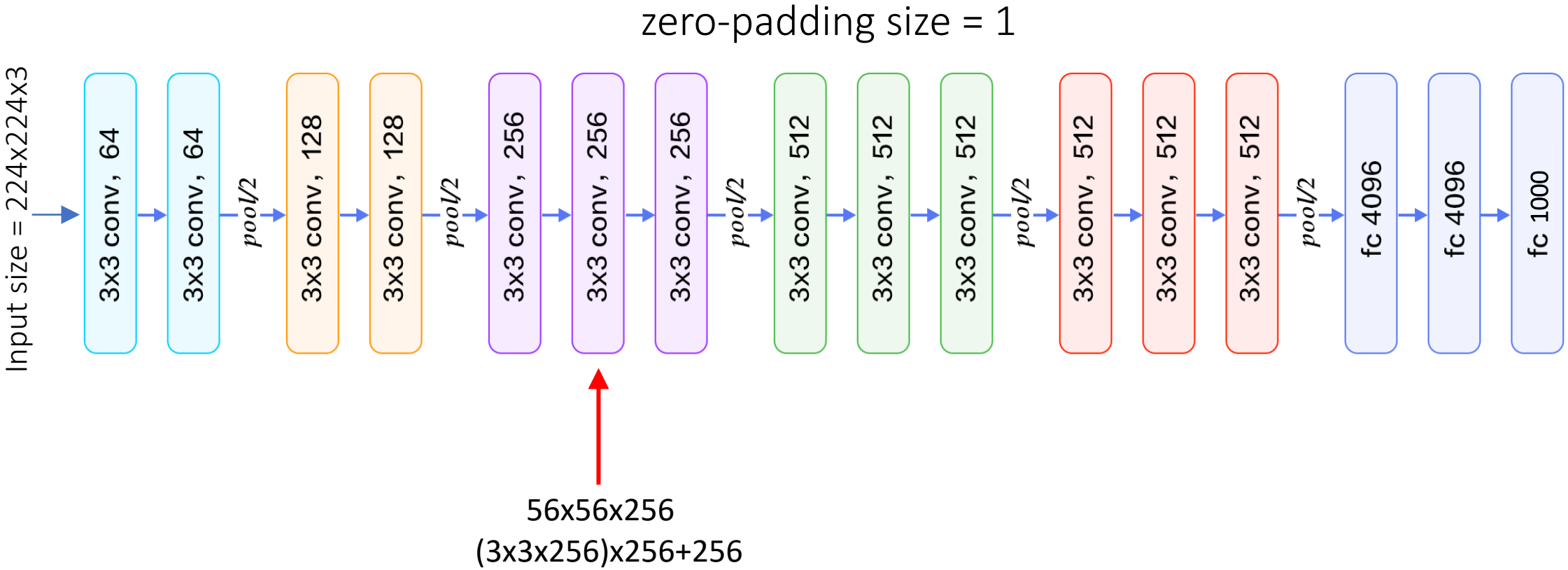
Case study: VGG16 for ImageNet classification



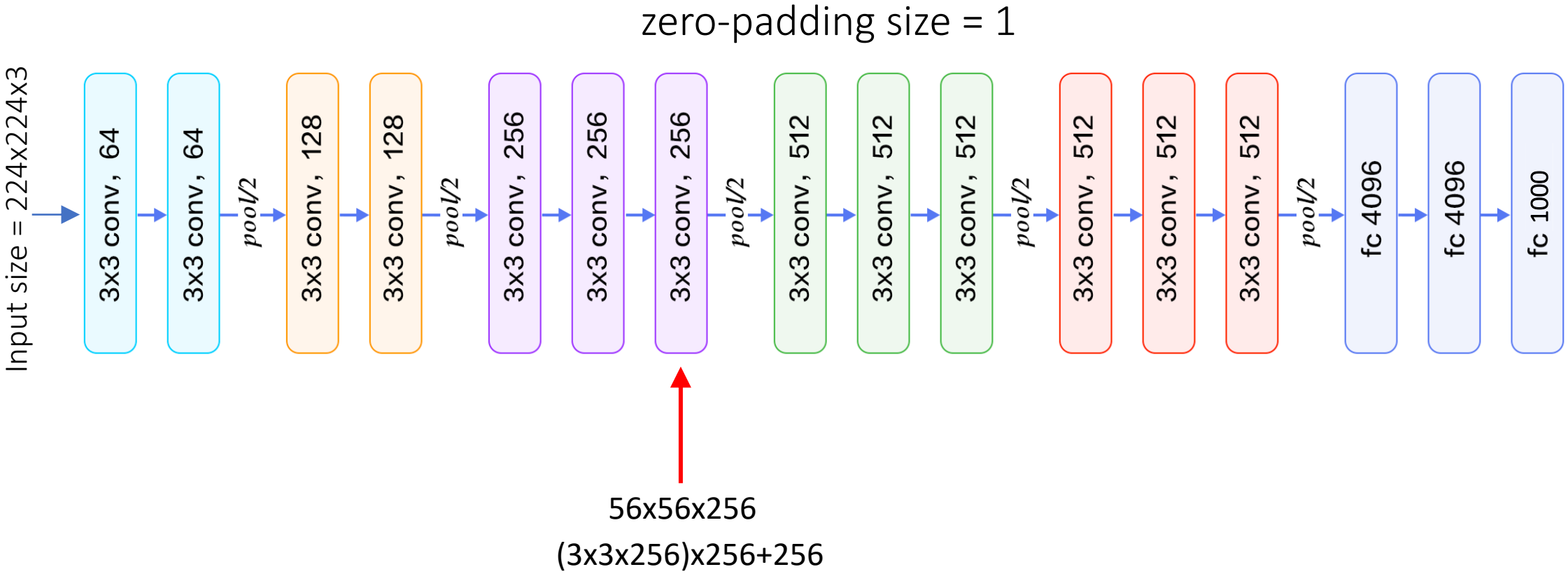
Case study: VGG16 for ImageNet classification



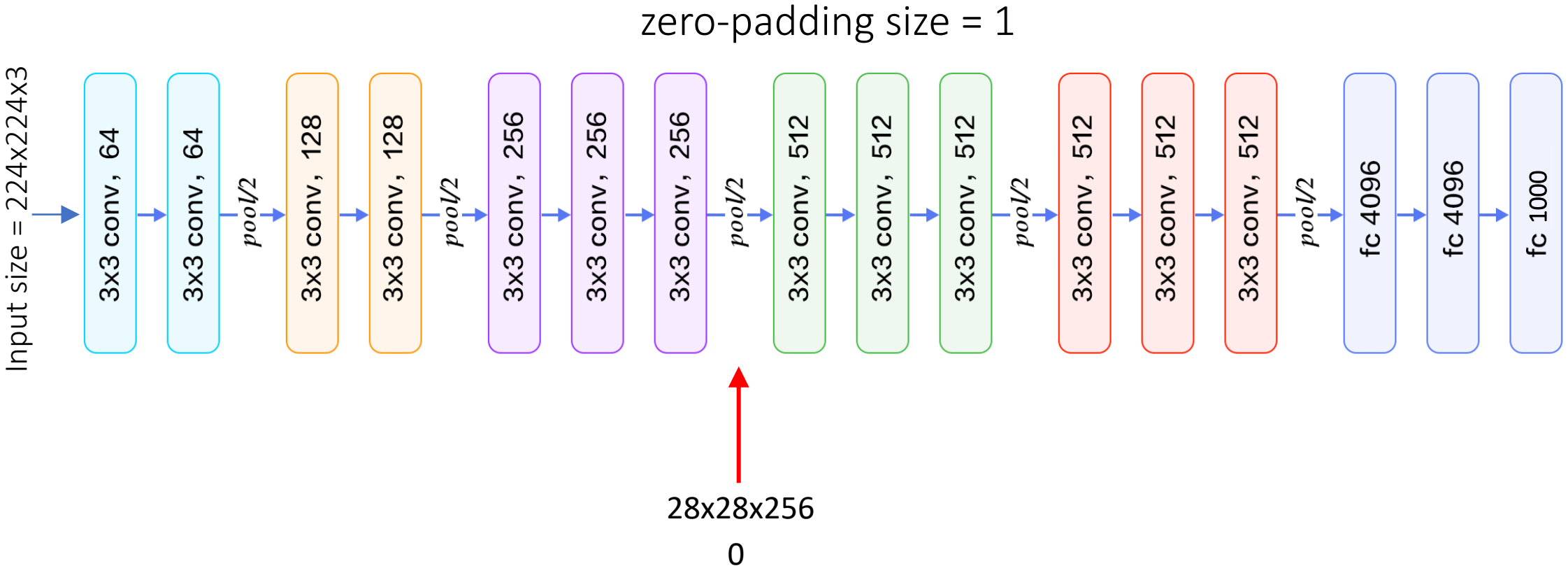
Case study: VGG16 for ImageNet classification



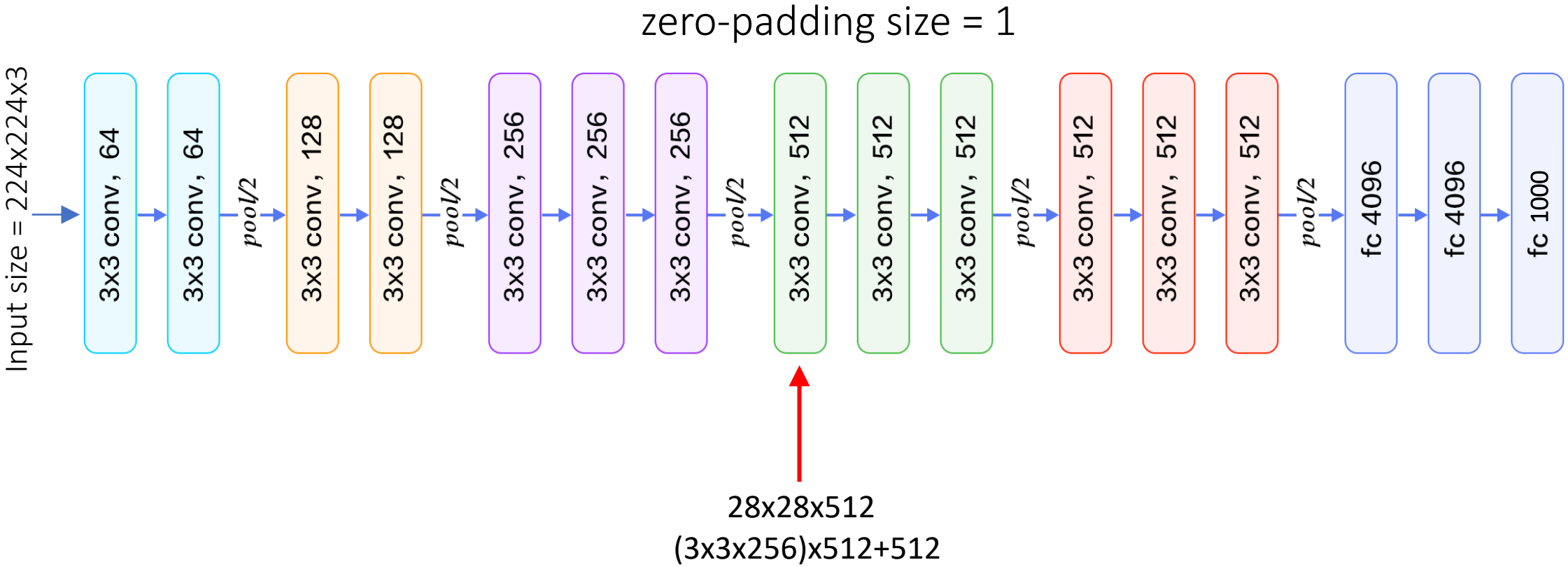
Case study: VGG16 for ImageNet classification



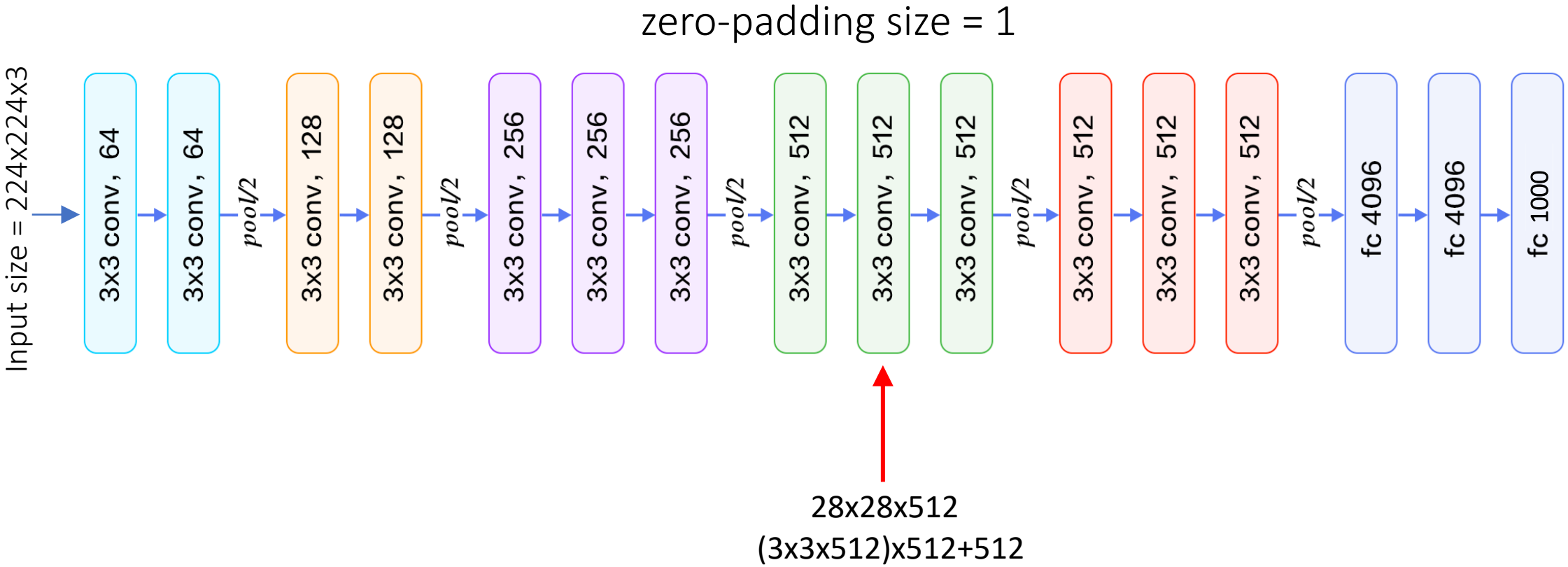
Case study: VGG16 for ImageNet classification



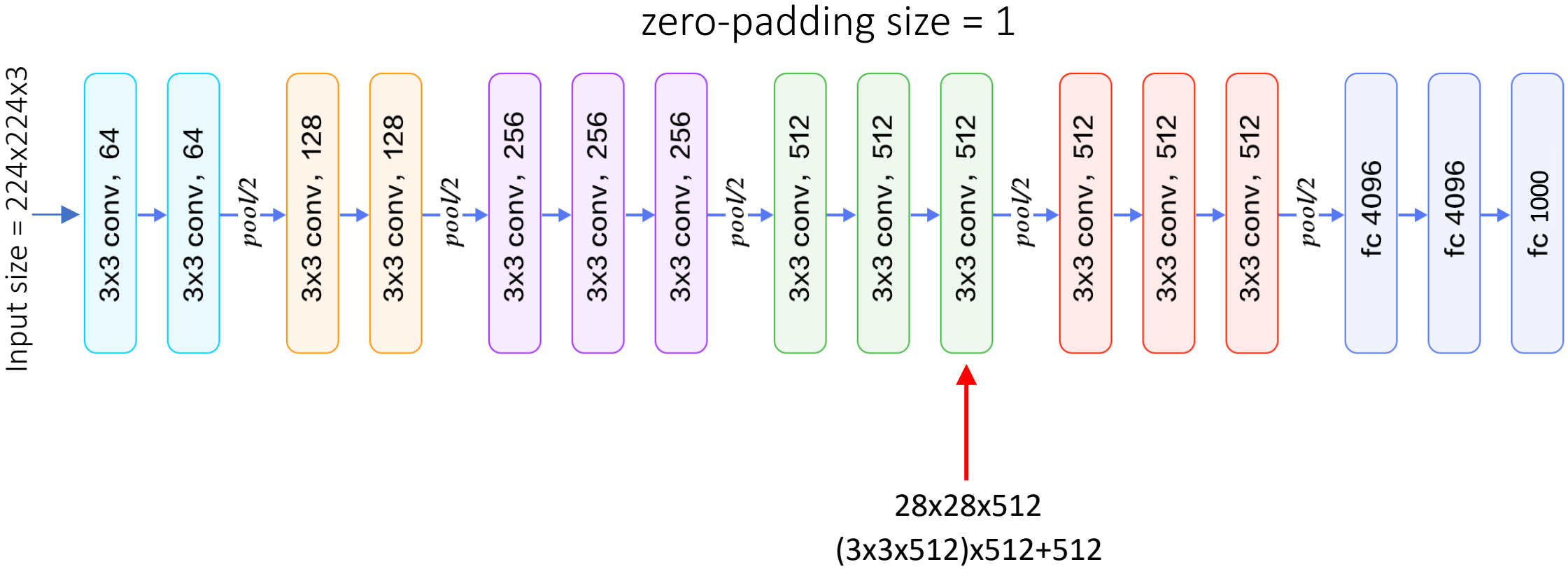
Case study: VGG16 for ImageNet classification



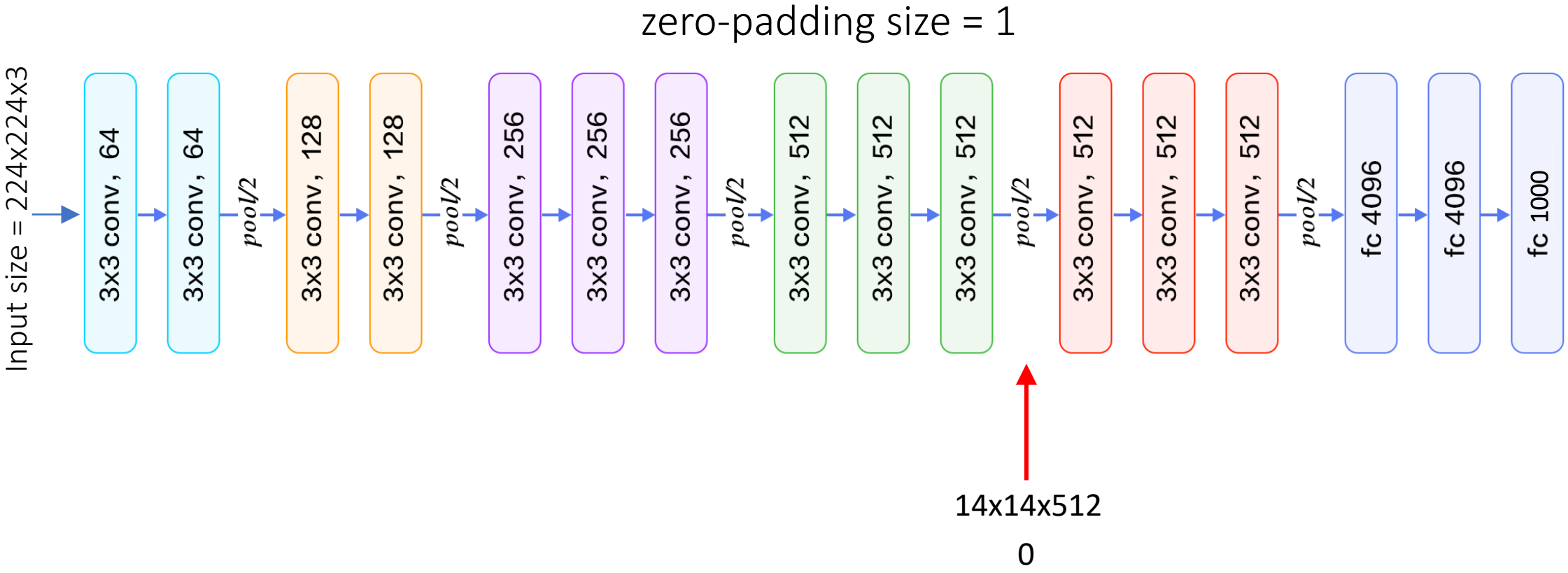
Case study: VGG16 for ImageNet classification



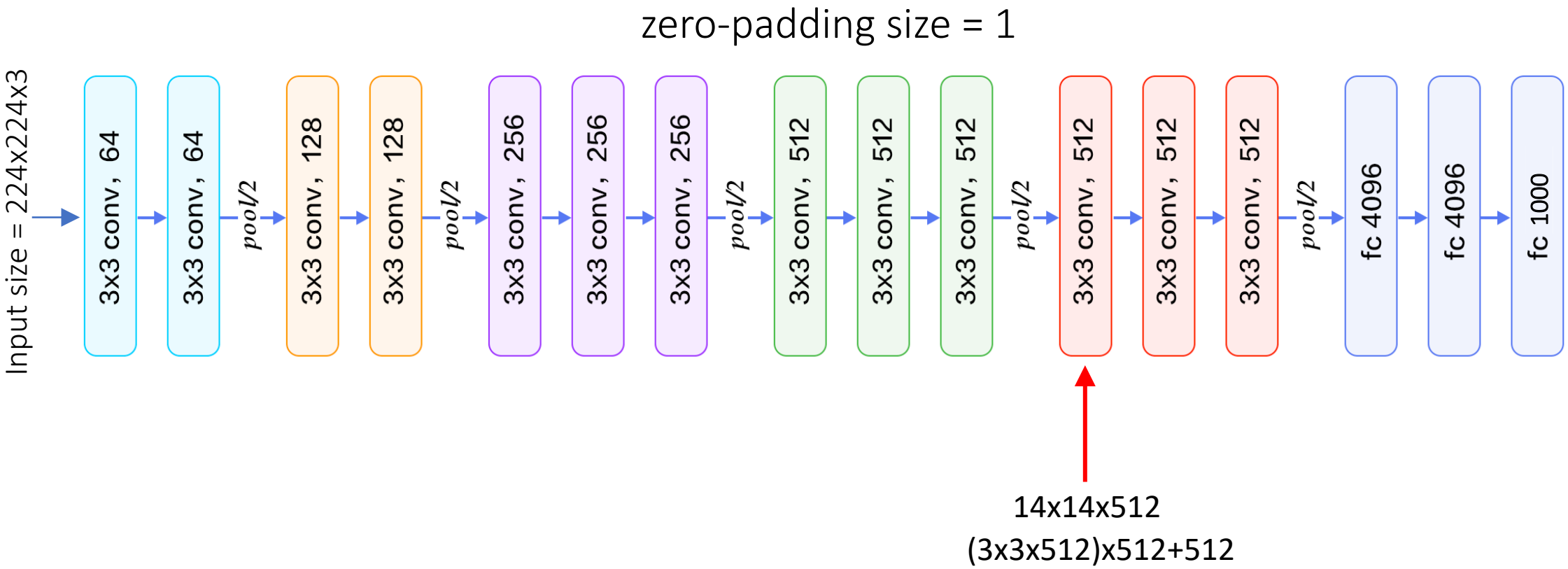
Case study: VGG16 for ImageNet classification



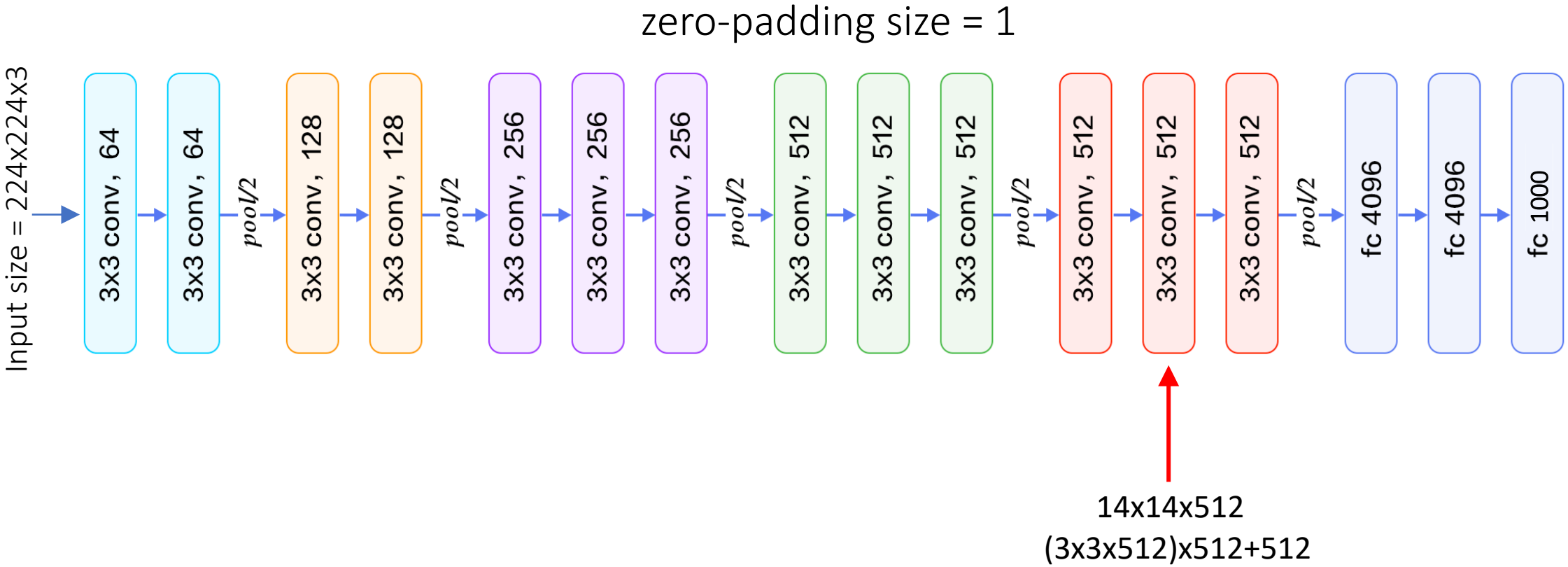
Case study: VGG16 for ImageNet classification



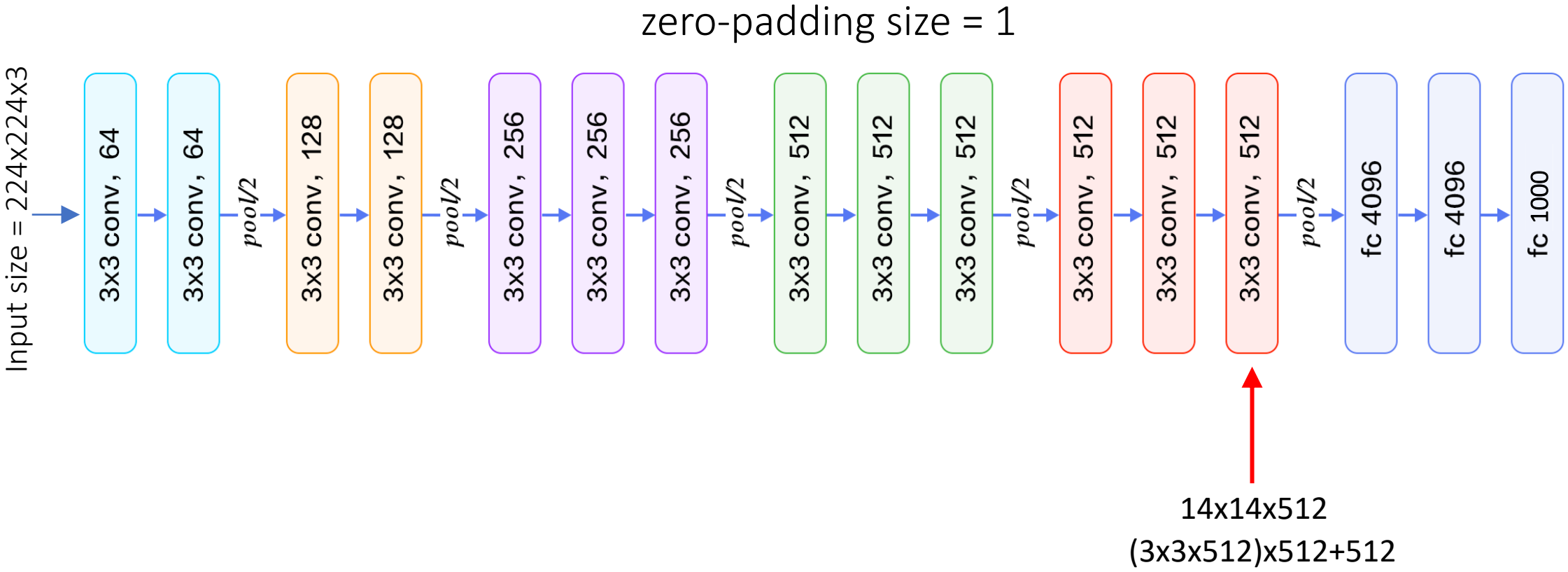
Case study: VGG16 for ImageNet classification



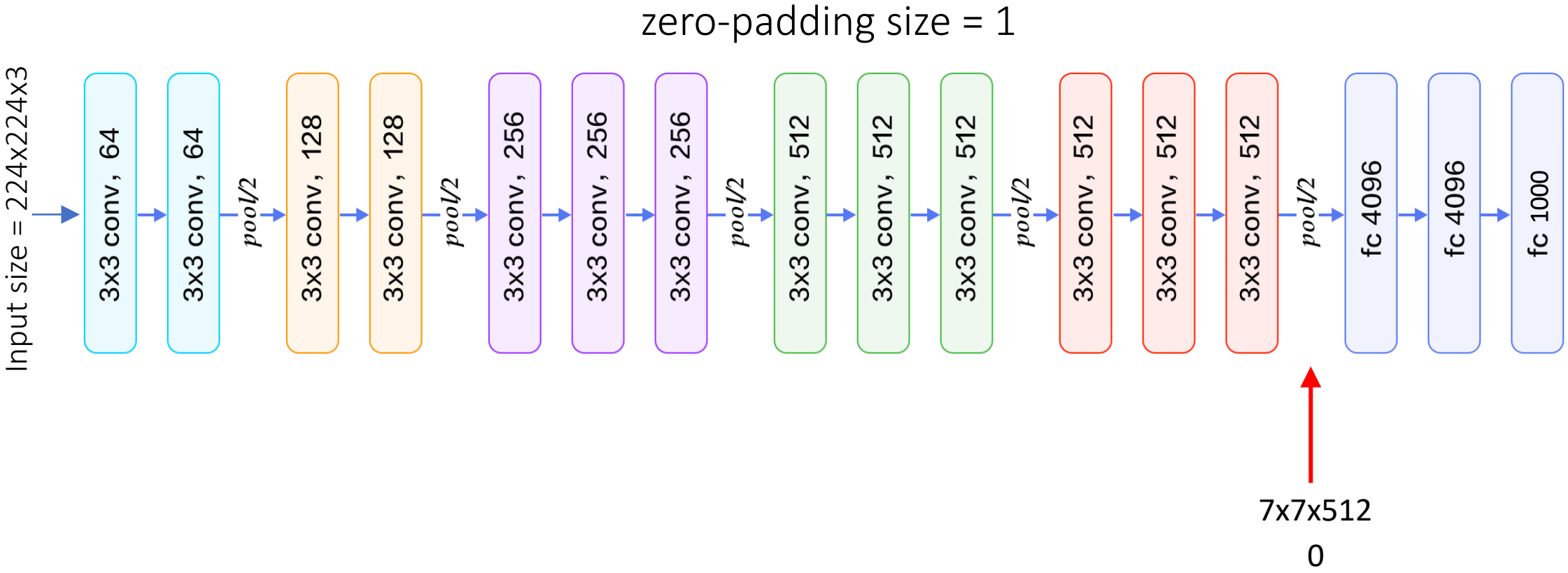
Case study: VGG16 for ImageNet classification



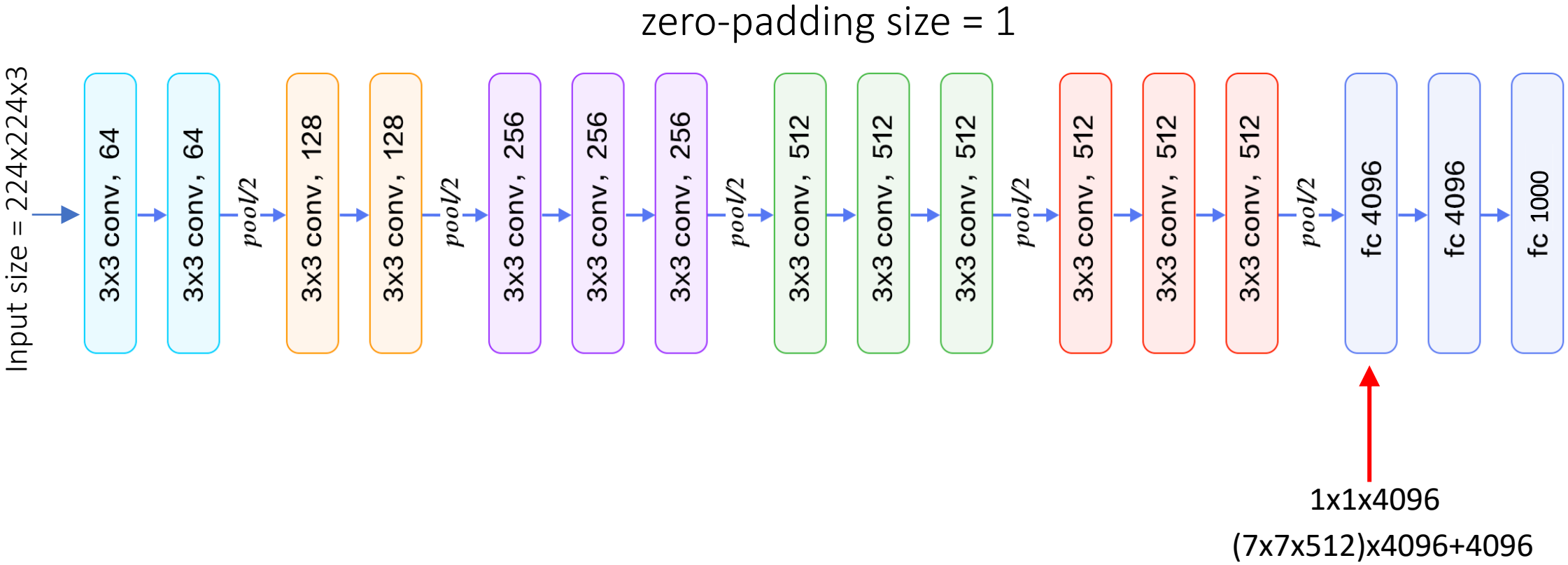
Case study: VGG16 for ImageNet classification



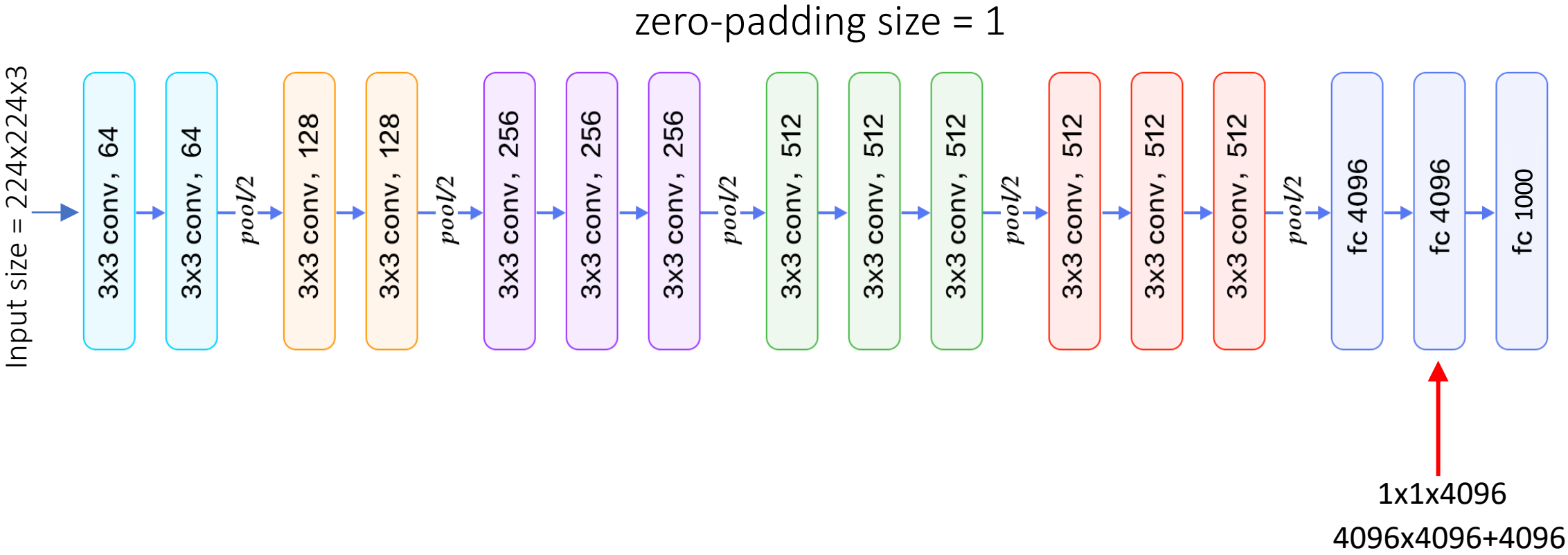
Case study: VGG16 for ImageNet classification



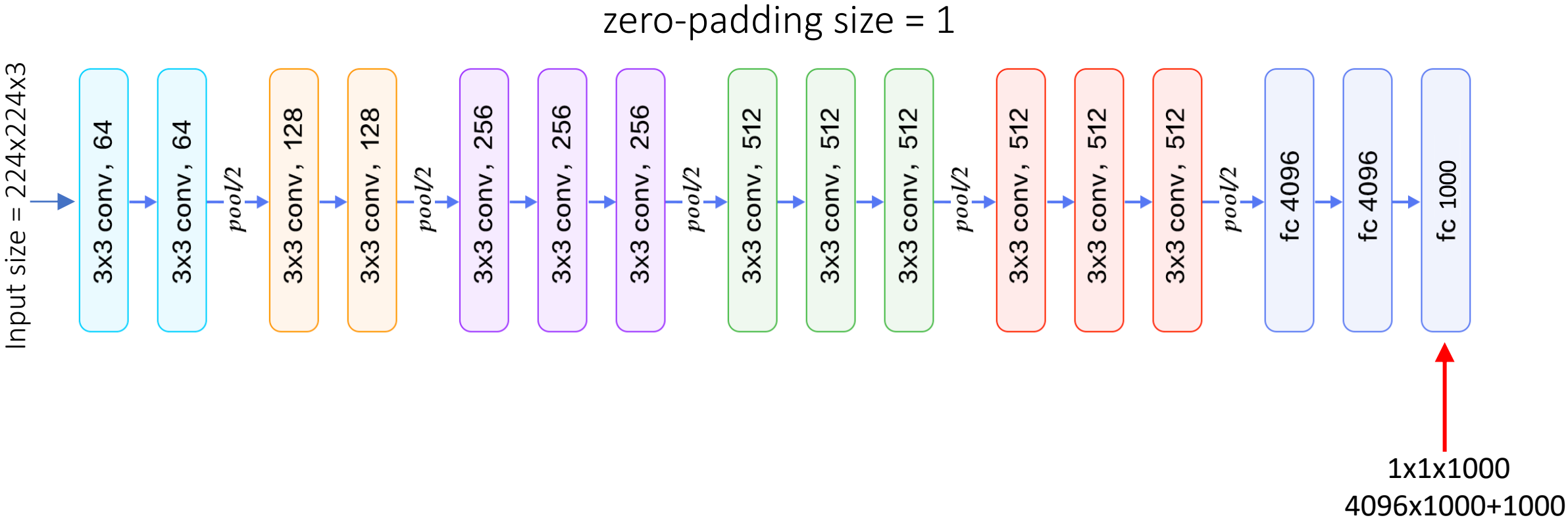
Case study: VGG16 for ImageNet classification



Case study: VGG16 for ImageNet classification



Case study: VGG16 for ImageNet classification



Total parameters: 138M

CNN: Continue

So far: Image Classification

This image is CC0 public domain

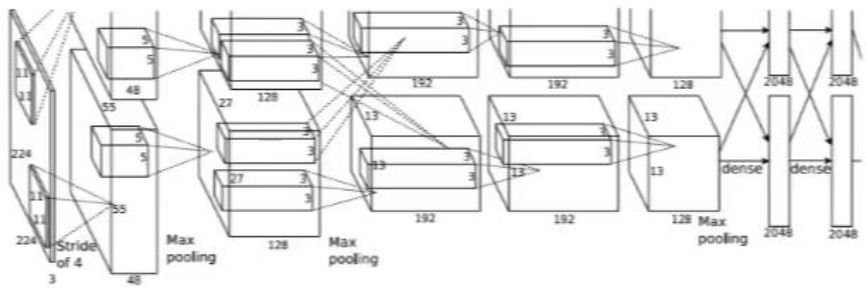


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Computer Vision Tasks

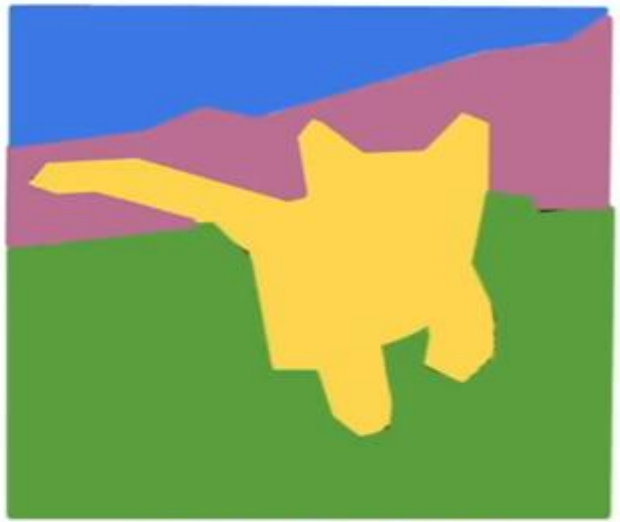
Classification + Localization

CAT

Object Detection

DOG, DOG, CAT

Semantic Segmentation

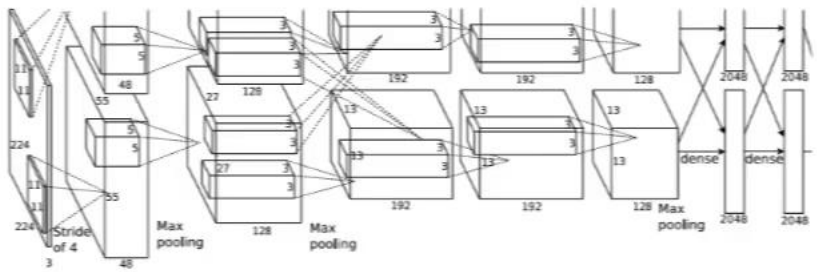


GRASS, CAT, TREE, SKY

Many others

Classification + Localization

This image is CC0 public domain



Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Vector: 4096

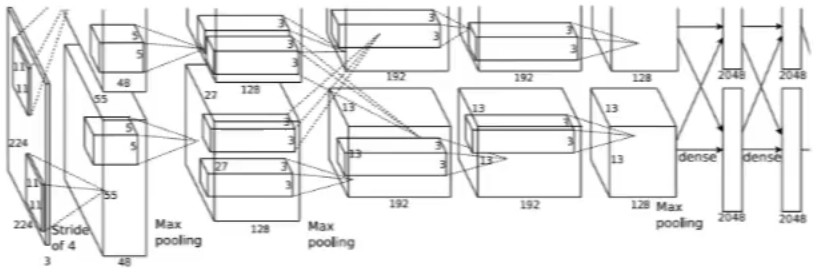
Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

Treat localization as a regression problem!

Classification + Localization

This image is CC0 public domain



Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

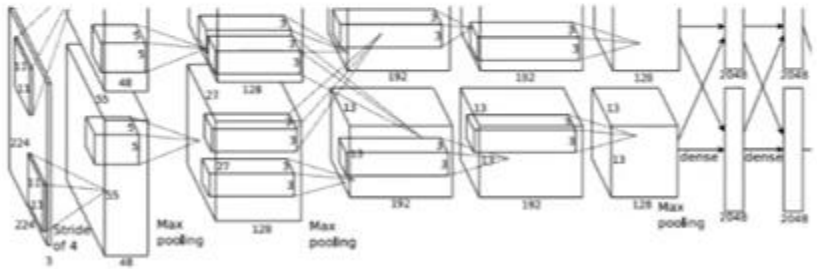
L2 Loss

Correct box: (x', y', w', h')

Treat localization as a regression problem!

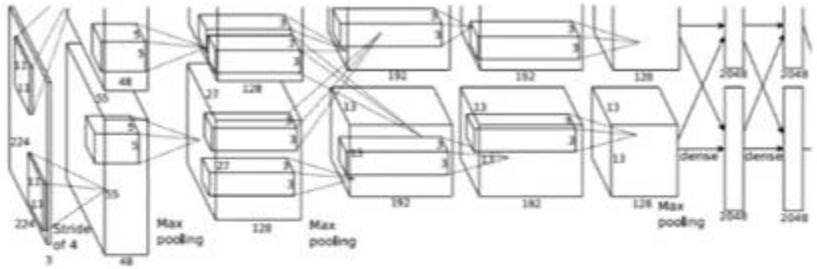
Object Detection as Regression

Each image needs a different number of outputs!



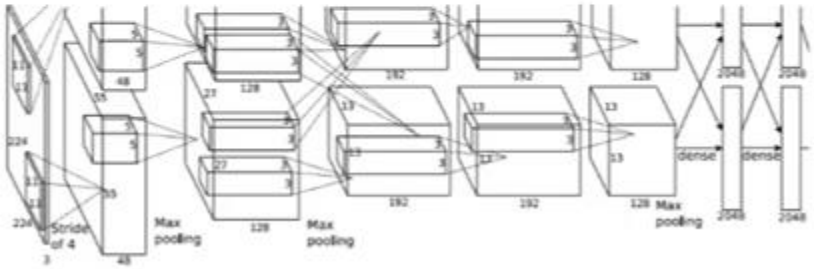
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

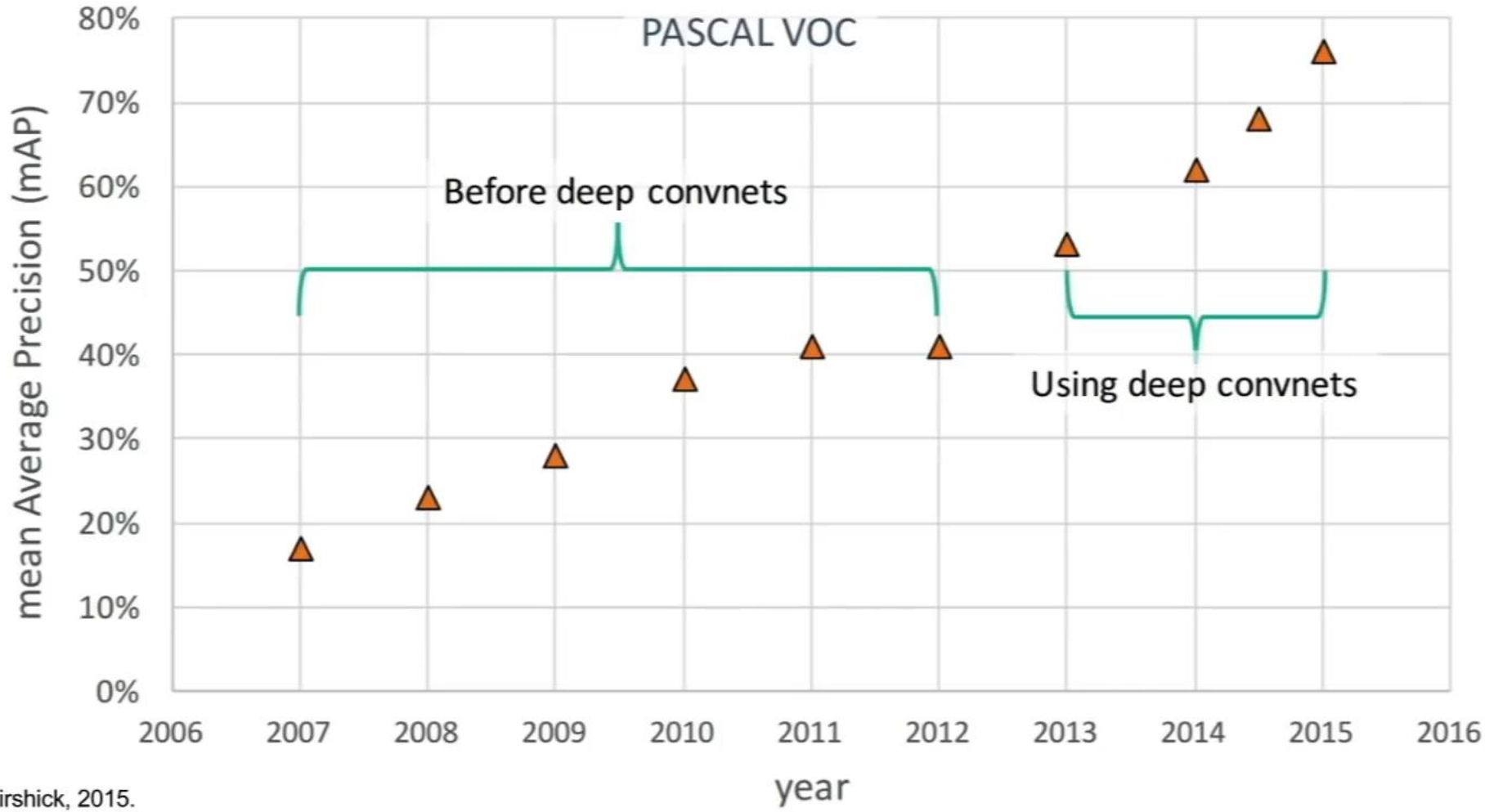
12 numbers



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

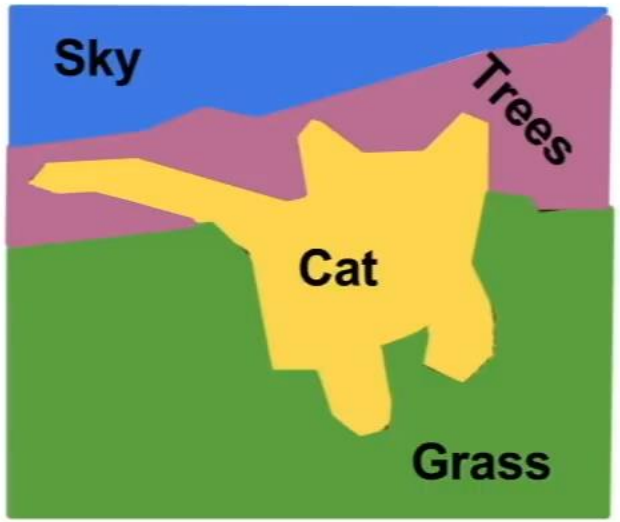
Many numbers!

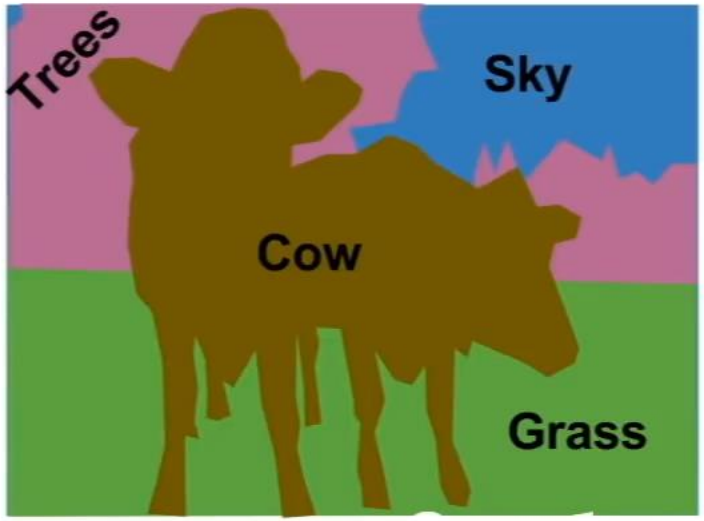
Object Detection: Impact of Deep Learning



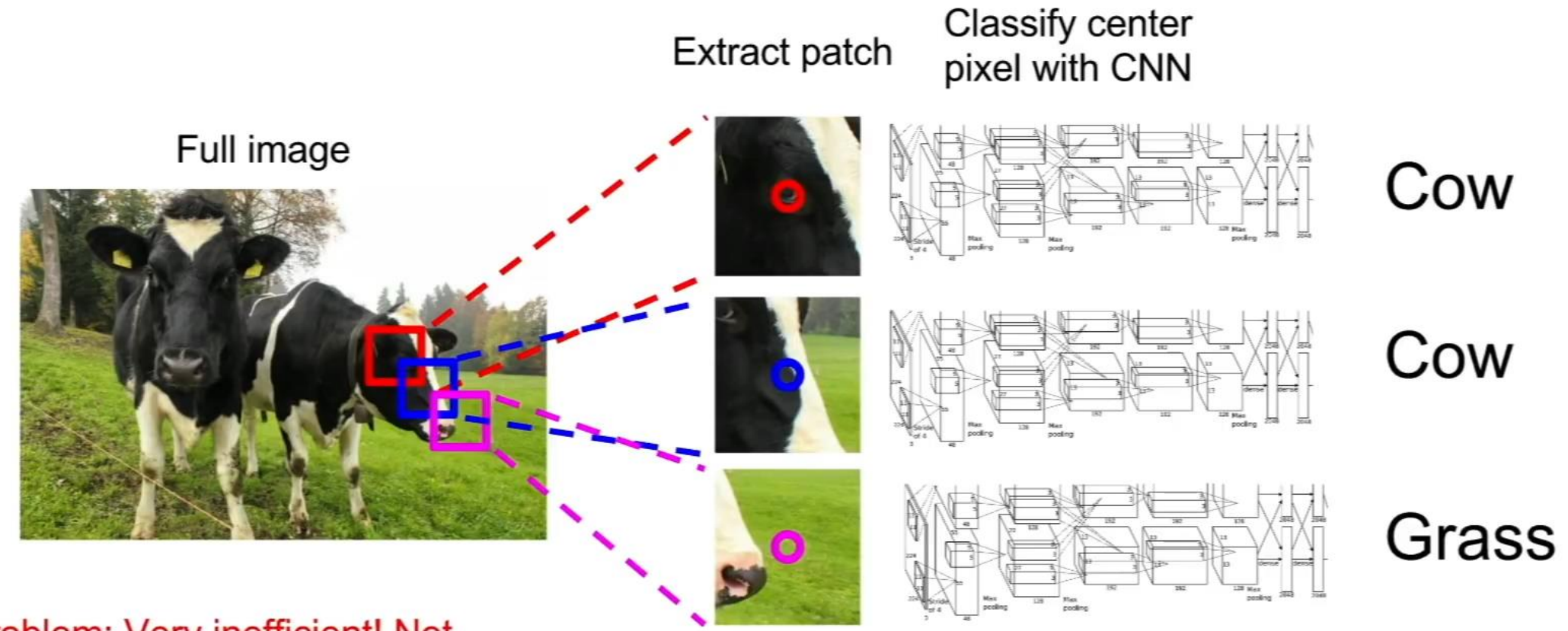
Semantic Segmentation

Label each pixel in the image with a category label





Semantic Segmentation Idea: Sliding Window



Problem: Very inefficient! Not reusing shared features between overlapping patches

Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

Input:
3 x H x W

Conv → Conv → Conv

Convolutions:
D x H x W

Conv

Scores:
C x H x W

argmax

Predictions:

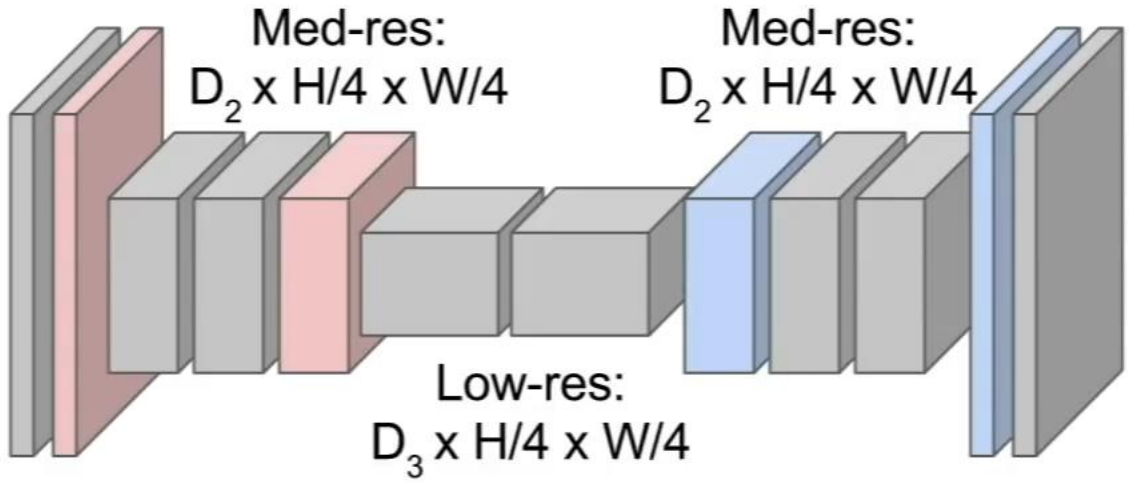
Ground truth

Cross Entropy loss

Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

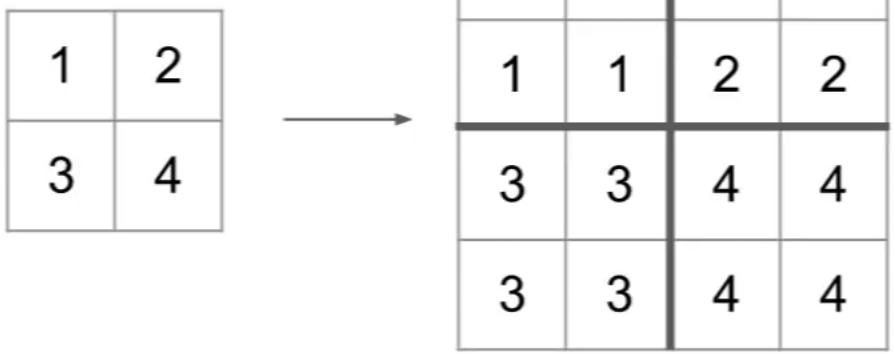
High-res:
D_1 x H/2 x W/2

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

In-Network upsampling: “Unpooling”

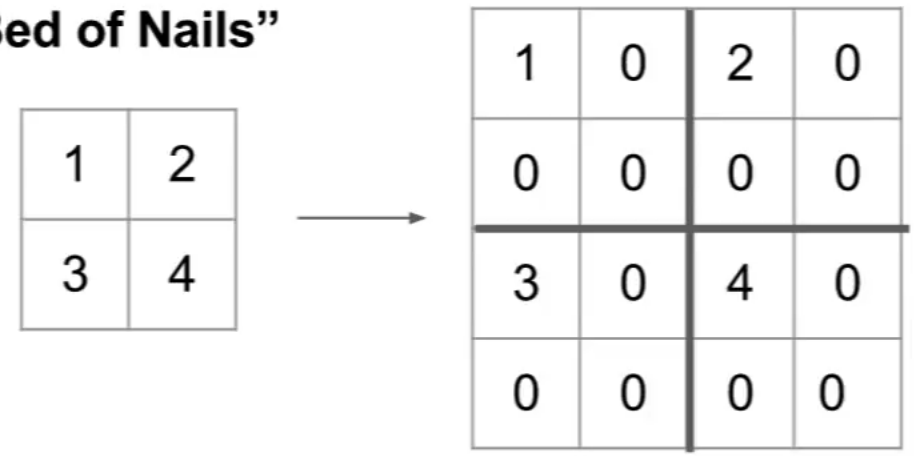
Nearest Neighbor



Input: 2 x 2

Output: 4 x 4

“Bed of Nails”

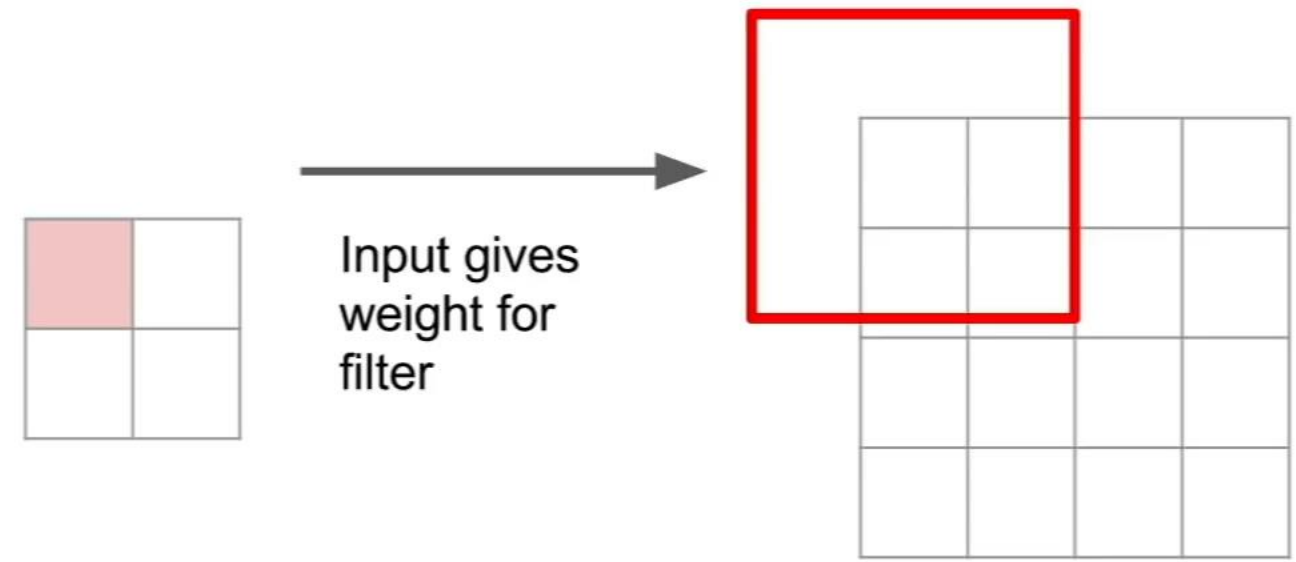


Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Transpose Convolution

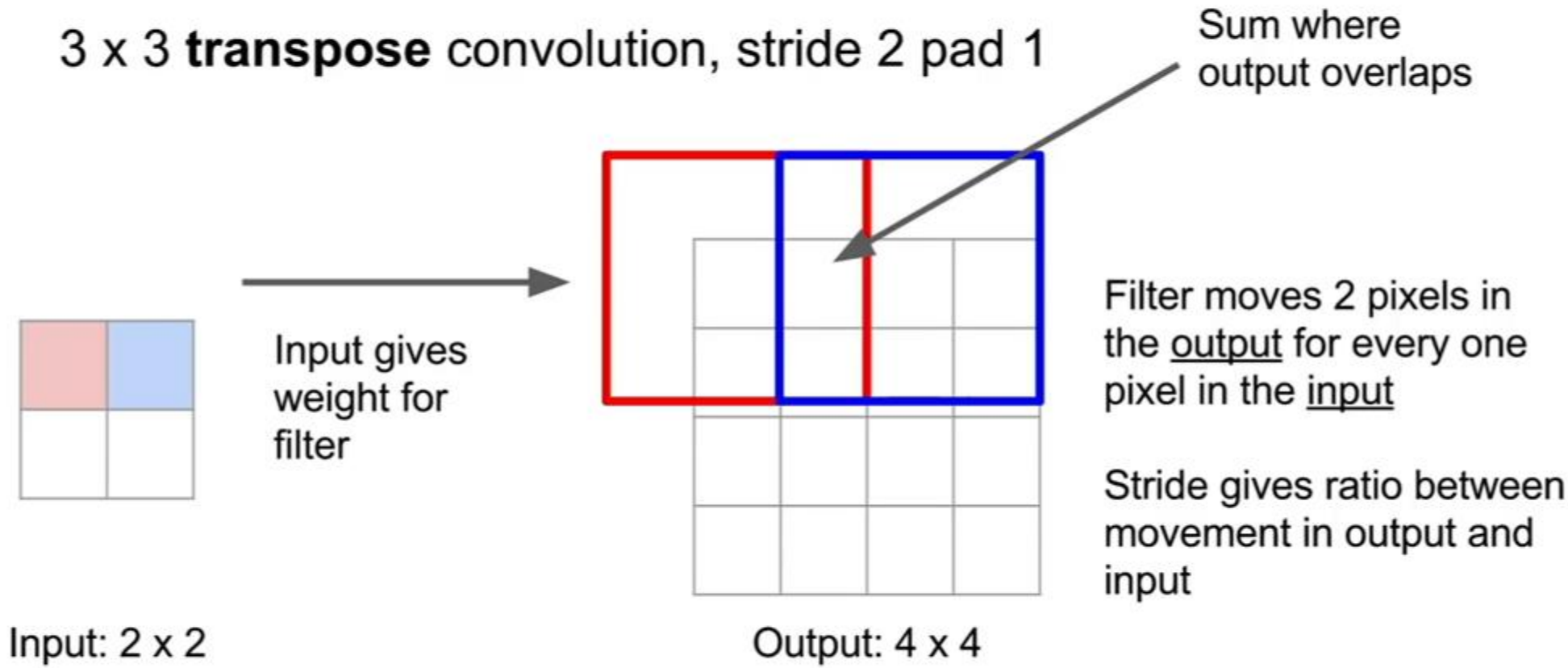
3 x 3 **transpose** convolution, stride 2 pad 1



Input: 2 x 2

Output: 4 x 4

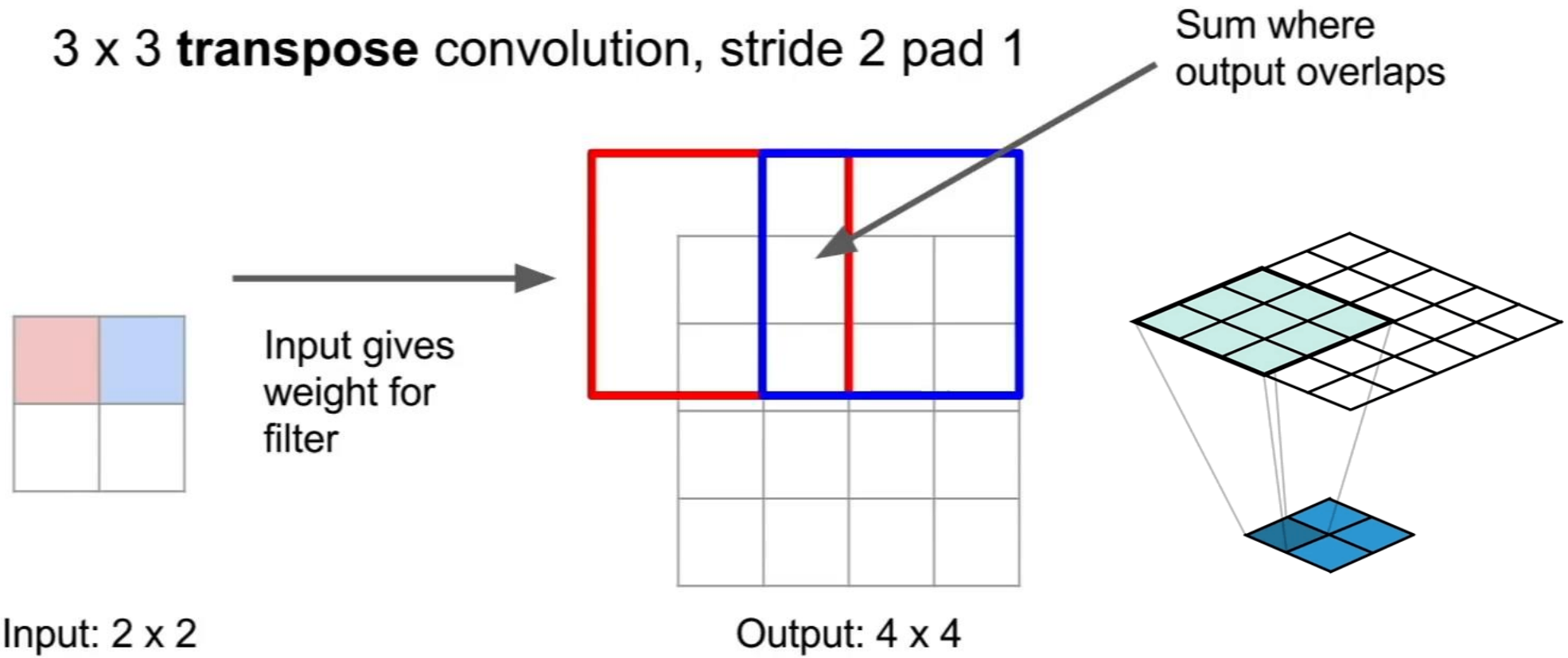
Learnable Upsampling: Transpose Convolution



Learnable Upsampling: Transpose Convolution

Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

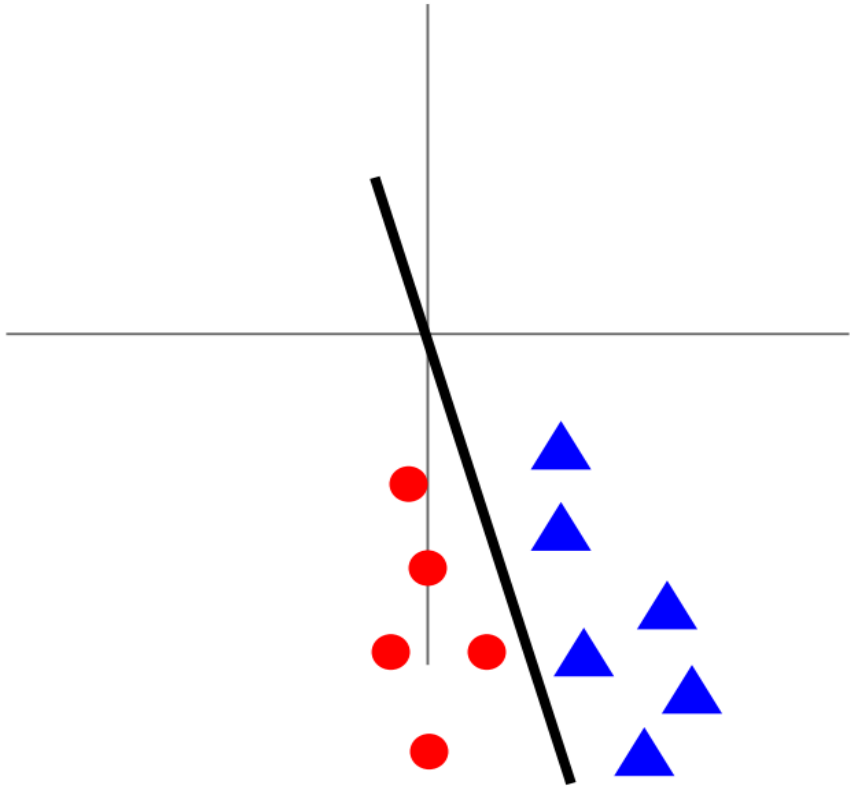
3 x 3 **transpose** convolution, stride 2 pad 1



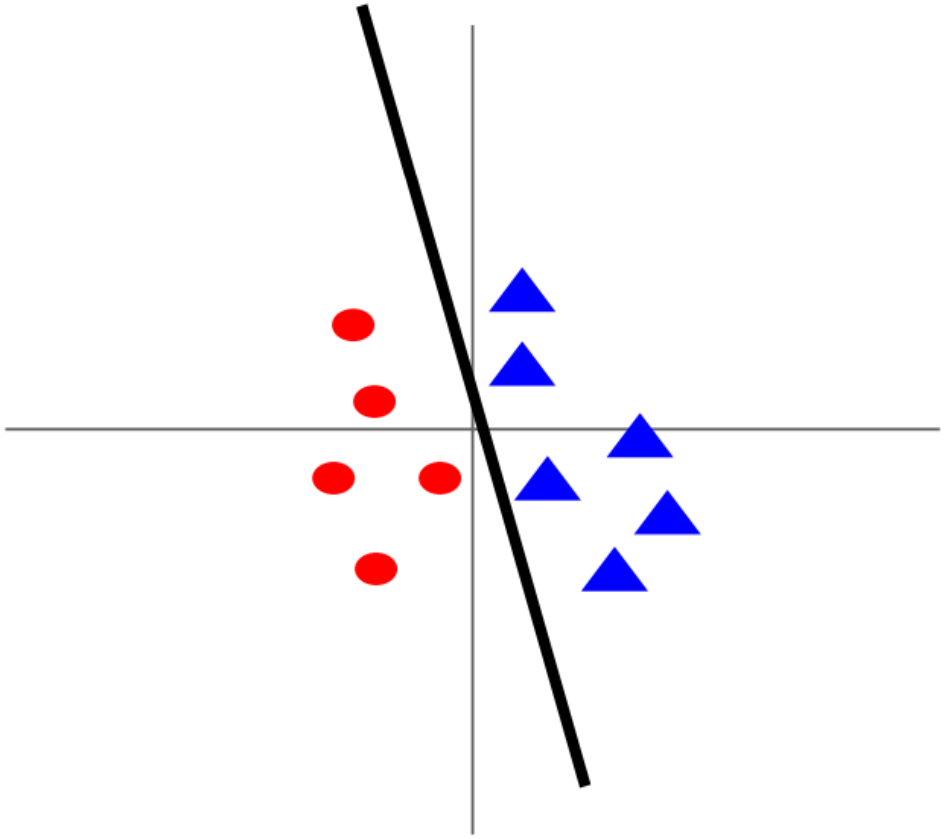
Magic formula (output size after upsampling)

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

Data preprocessing

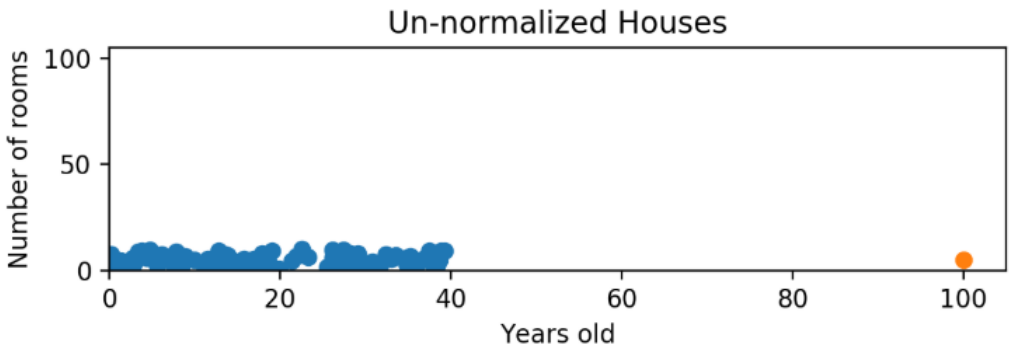


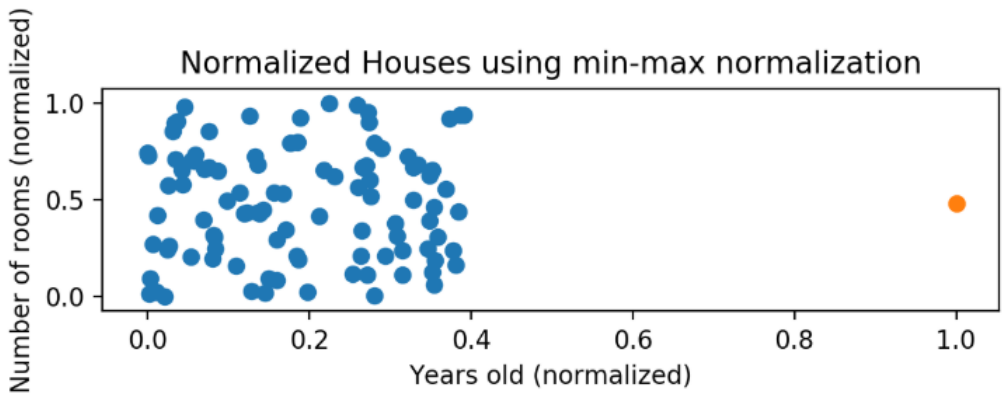
Before normalization, loss is sensitive to changes in model parameters, hard to optimize



After normalization, loss is less sensitive to changes in model parameters, easier to optimize

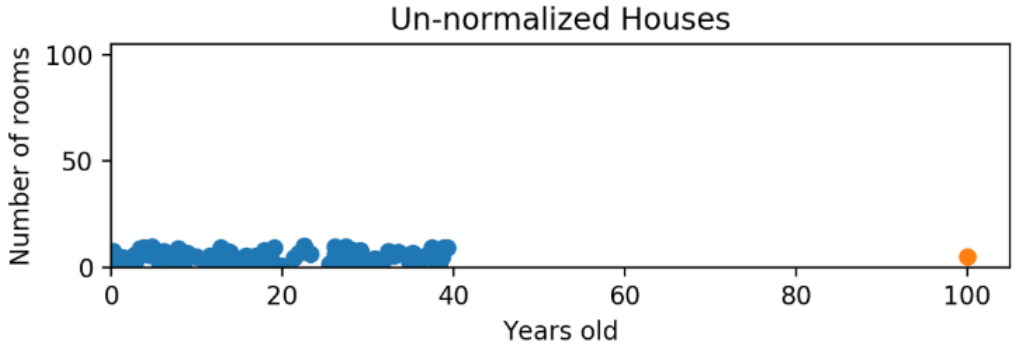
$$\text{Min-Max} = \frac{value - min}{max - min}$$

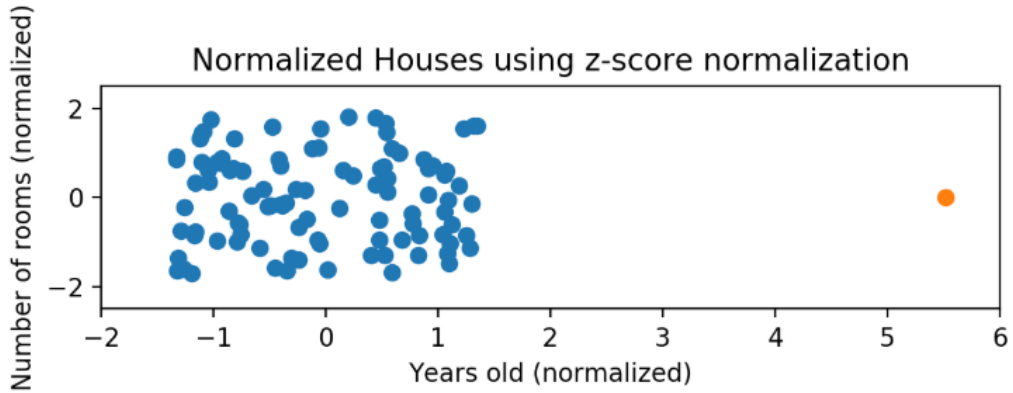




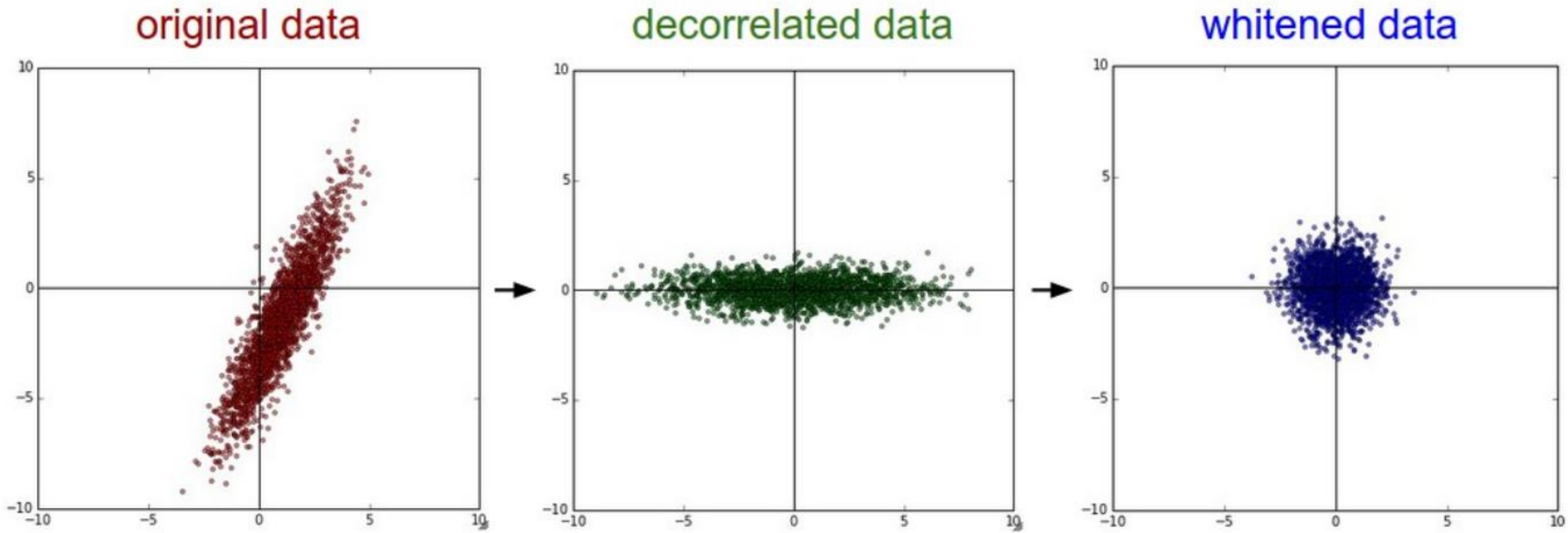
Min-max normalization: Guarantees all features will have the exact same scale but does not handle outliers well

$$\text{Z-score} = \frac{value - mean}{std}$$





Z-score normalization: Handles outliers, but does not produce normalized data with the *exact* same scale.



PCA / Whitening. **Left**: Original toy, 2-dimensional input data. **Middle**: After performing PCA. The data is centered at zero and then rotated into the eigenbasis of the data covariance matrix. This decorrelates the data (the covariance matrix becomes diagonal). **Right**: Each dimension is additionally scaled by the eigenvalues, transforming the data covariance matrix into the identity matrix. Geometrically, this corresponds to stretching and squeezing the data into an isotropic gaussian blob.

Less used in convolutional neural network

Terminologies

1. Batch
2. Epoch
3. Stochastic gradient descent
4. Neuron
5. Activation/feature map
6. Activation function
7. Channels/depth
8. Respective field
9. Hidden layer (including many activation maps)
10. Parameter
11. Hyperparameter
12. Network architecture
13. Inference (deployment)
14. Forward pass
15. Backward propagation

References for these slides

- http://cs231n.stanford.edu/
- https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8
- http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html
- https://pytorch.org/docs/stable/generated/torch.transpose.html
- https://arxiv.org/pdf/1409.1556.pdf