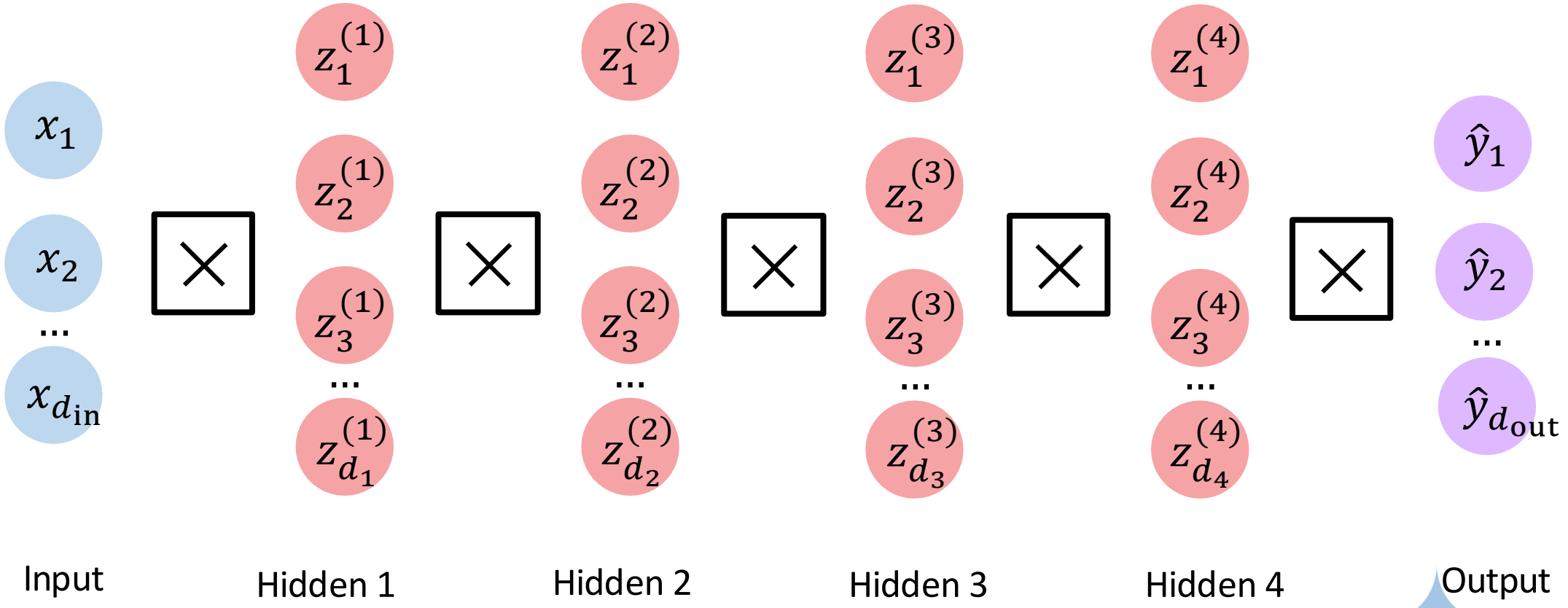
Convolutional Neural Networks

Qiang Sun

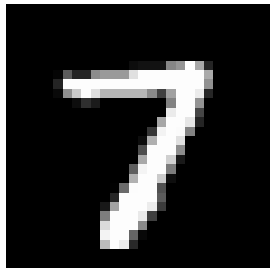
University of Toronto

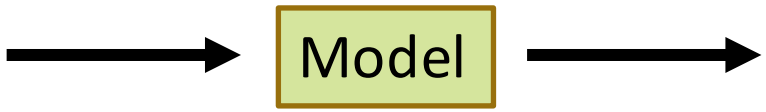
Fully-Connected (Dense) Deep Neural Network



Each neuron in one layer is connected to all neurons in the next layer

Deep Learning for Computer Vision

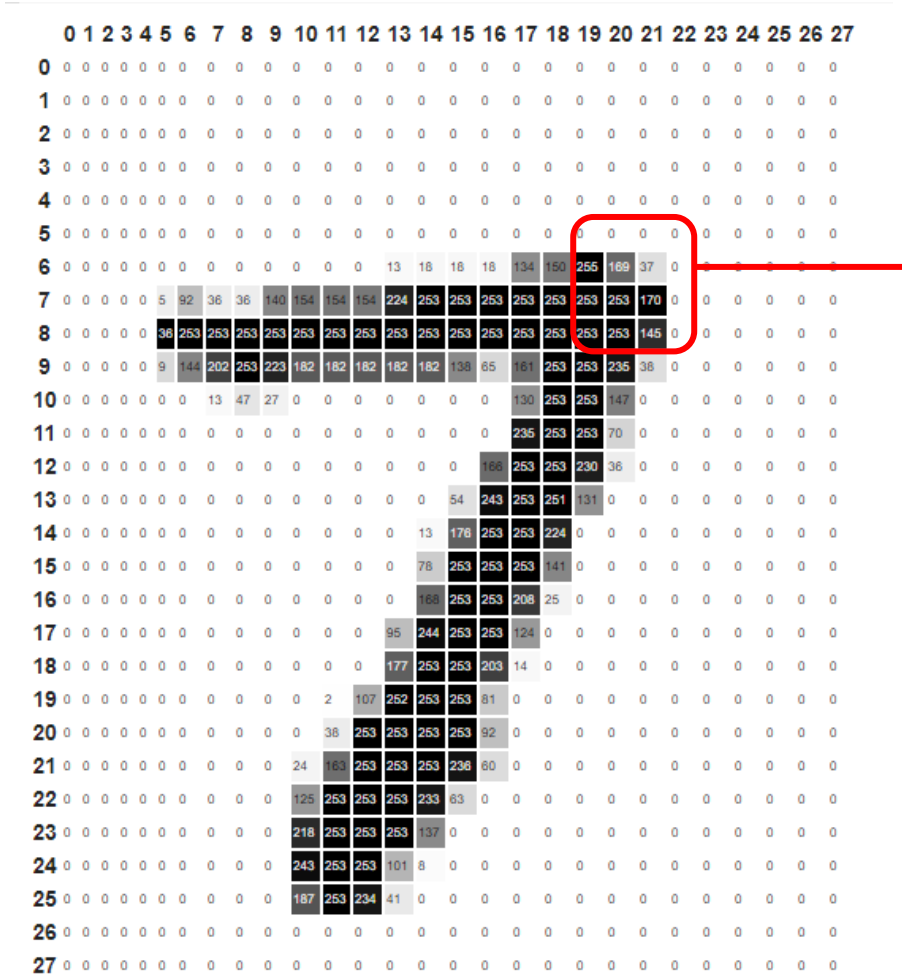


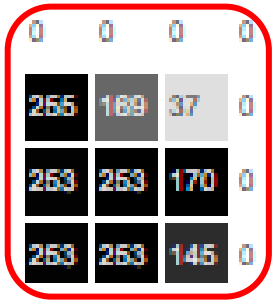


Predict –
The class of digit “7”

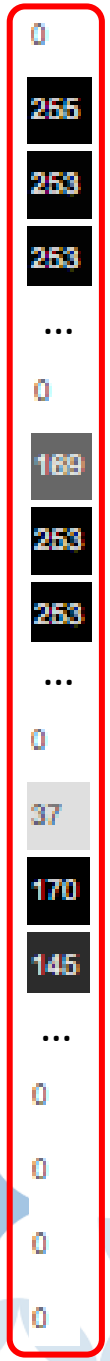
- Input (feature) space:
  - 28 x 28 matrix of numbers [0,255] for a grayscale image
    - (If it is an RGB image then 28 x 28 x 3)
- Applying a fully-connected neural network
  - Reshape to a 784 dimension vector as an input
  - Features need to be extracted before fed into the model
  - 3 layers, each has 784 neurons: ~ 2 million weights to learn

Deep Learning for Computer Vision



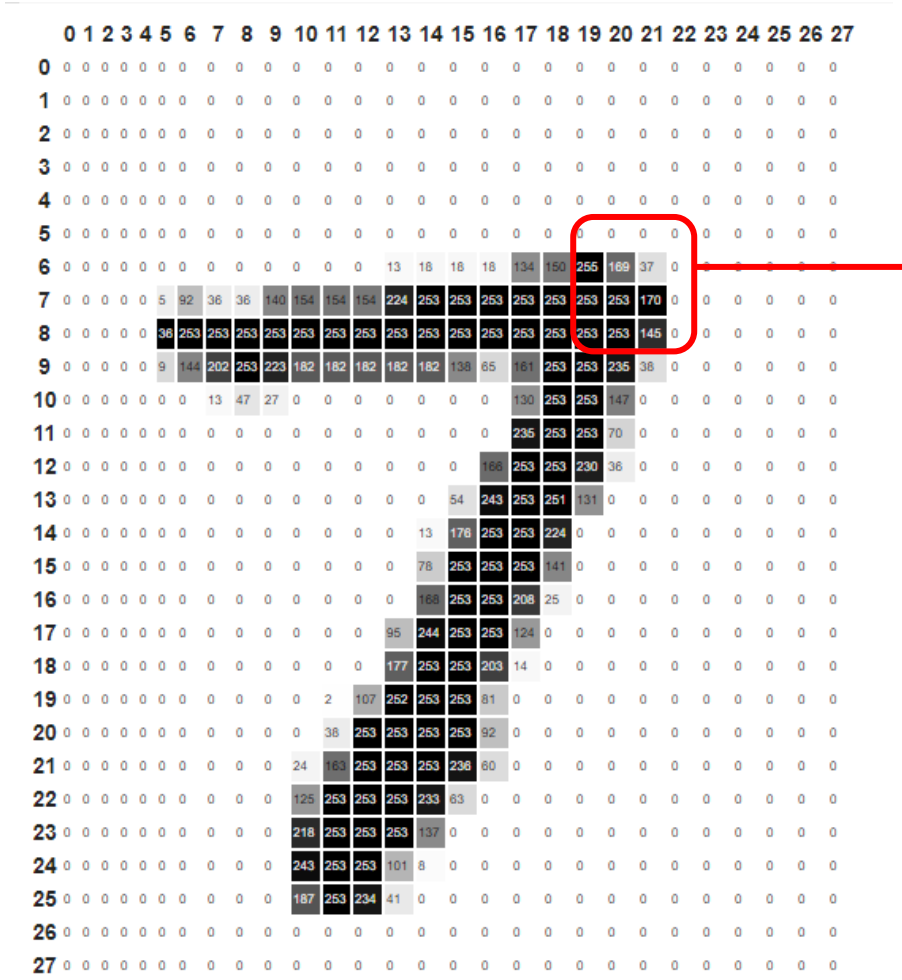


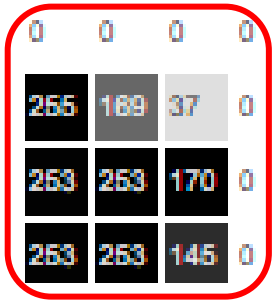
What computers see:



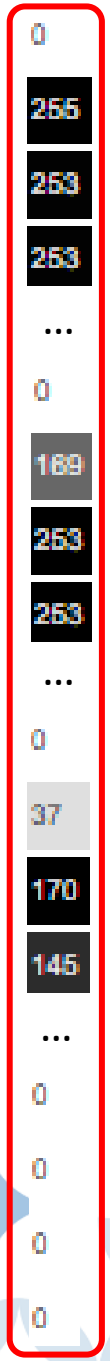
- Input: 784 dimension vector
- Weights: 2 million, too large

Deep Learning for Computer Vision



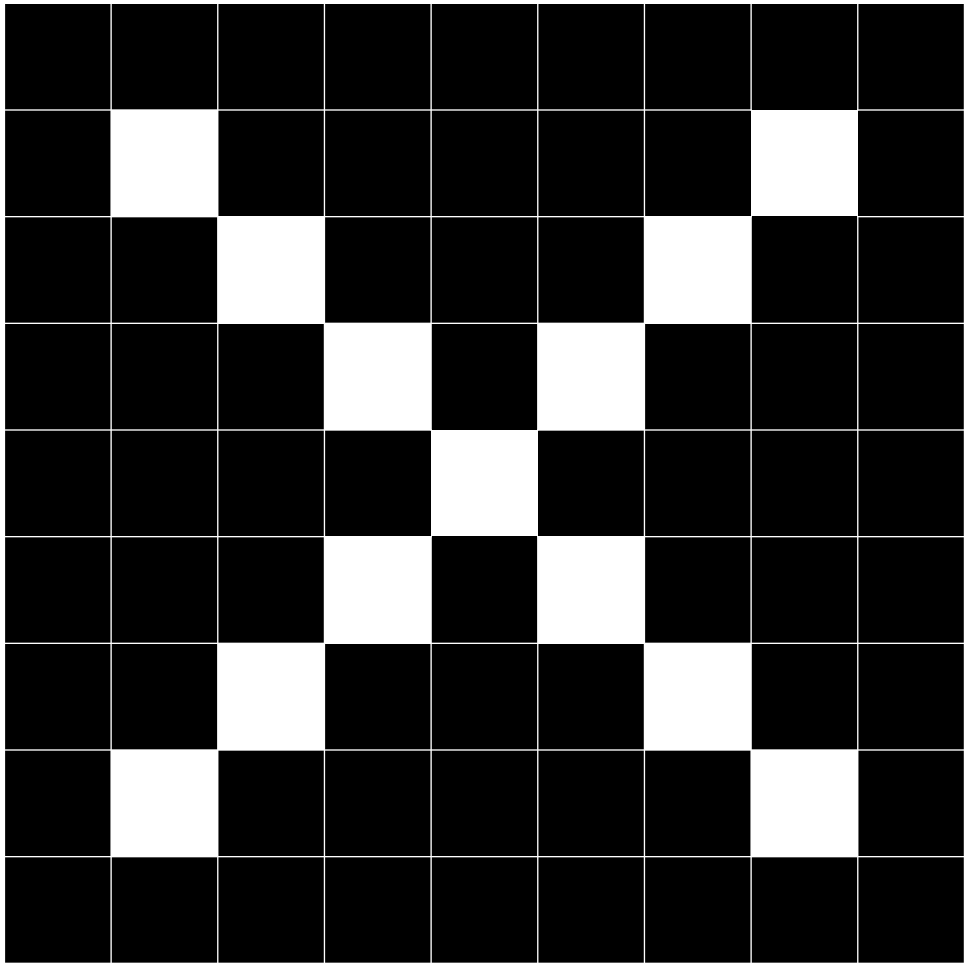


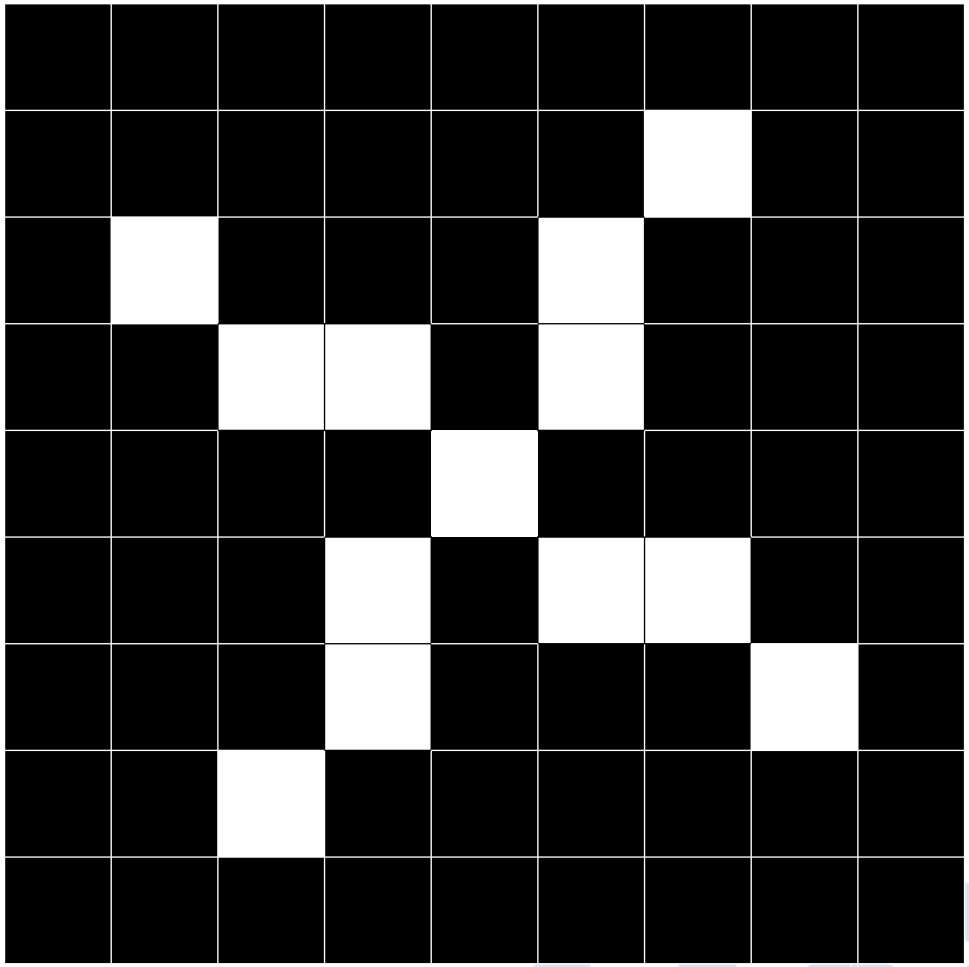
What computers see:



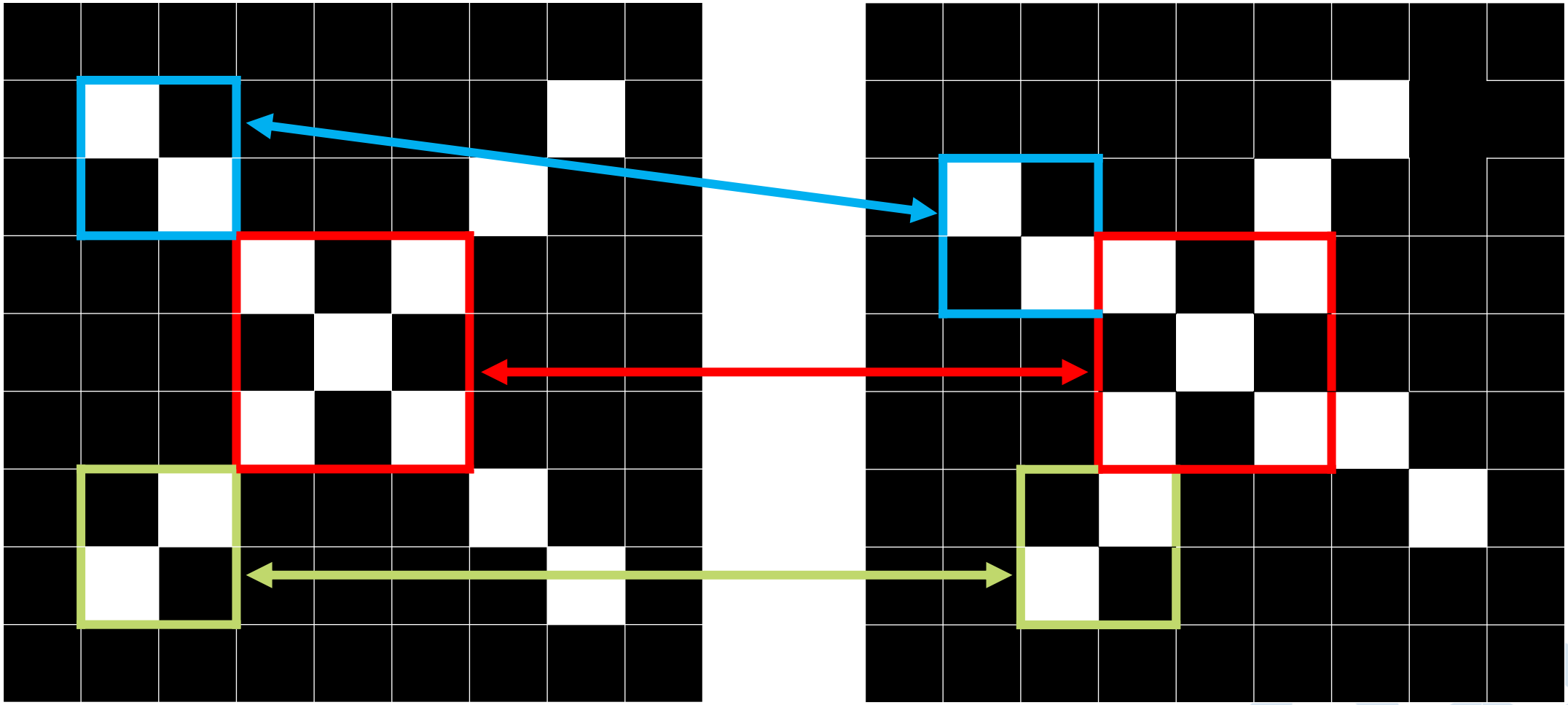
- Input: 784 dimension vector
- Weights: 2 million, too large
- More importantly:
  - spatial information not being used

Spatial Structure : Feature of X

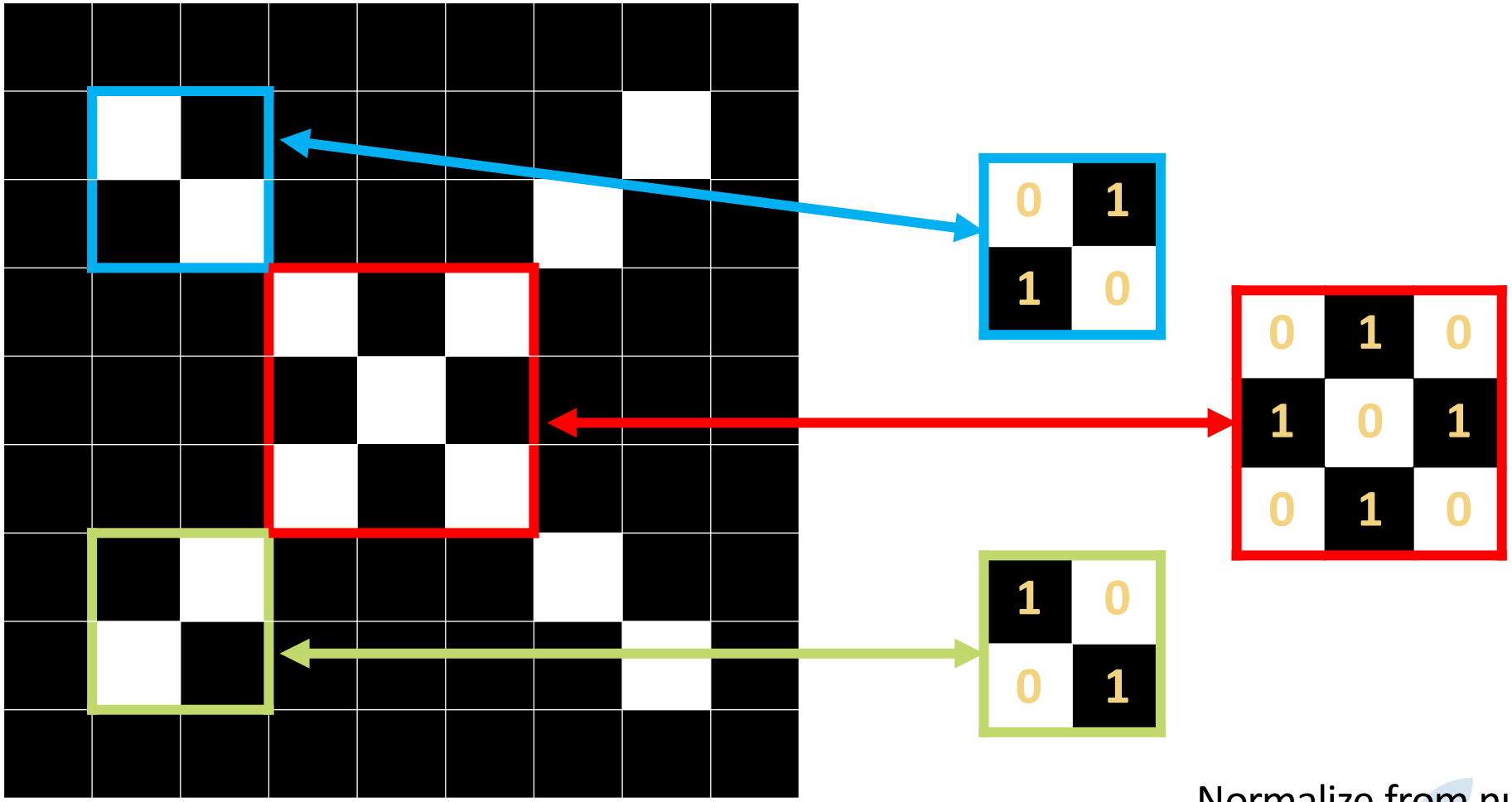




Spatial Structure : Feature of X



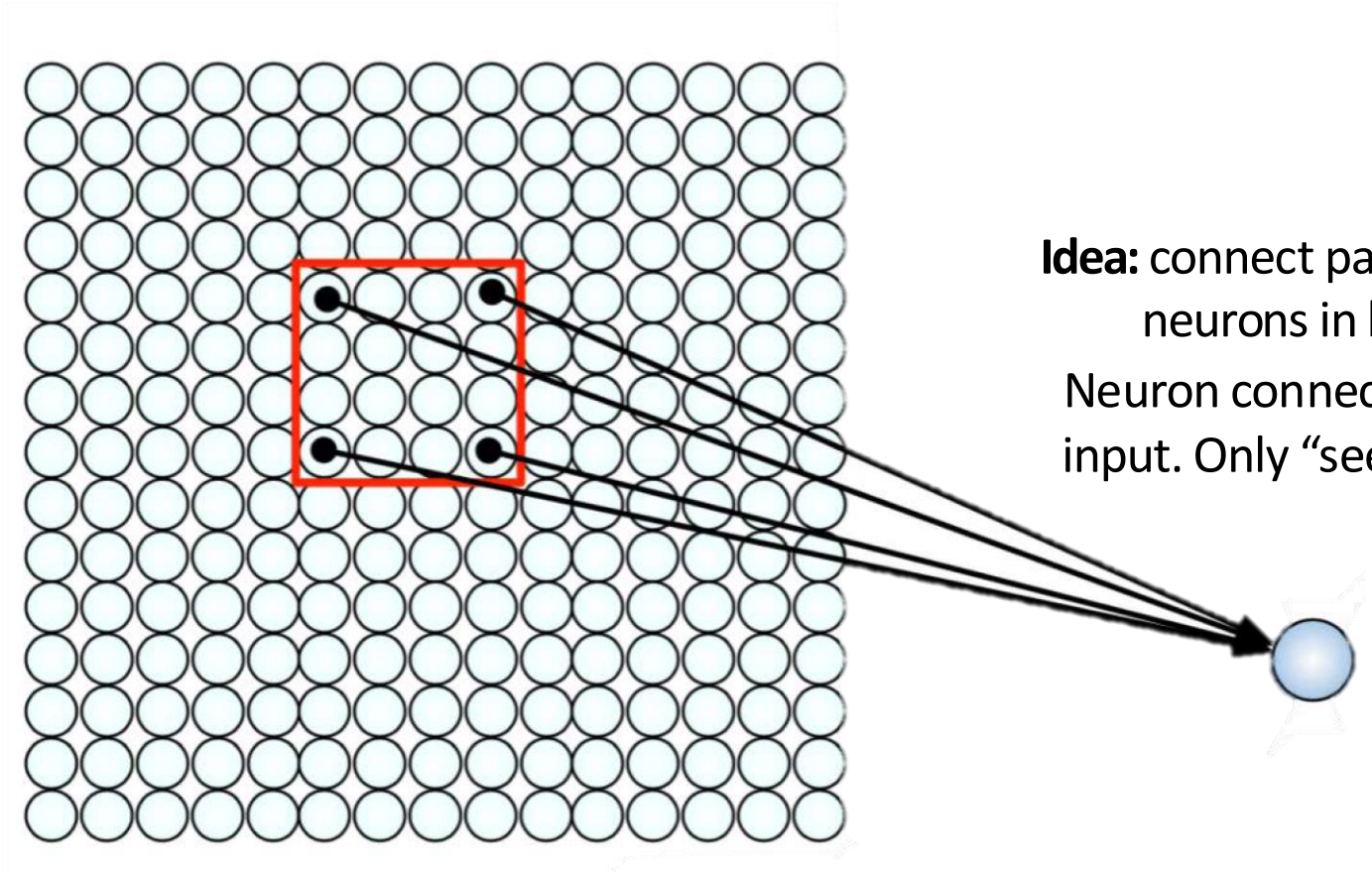
Spatial Structure : Feature of X



Normalize from numbers [0,255] to [0,1]

Use Spatial Structure

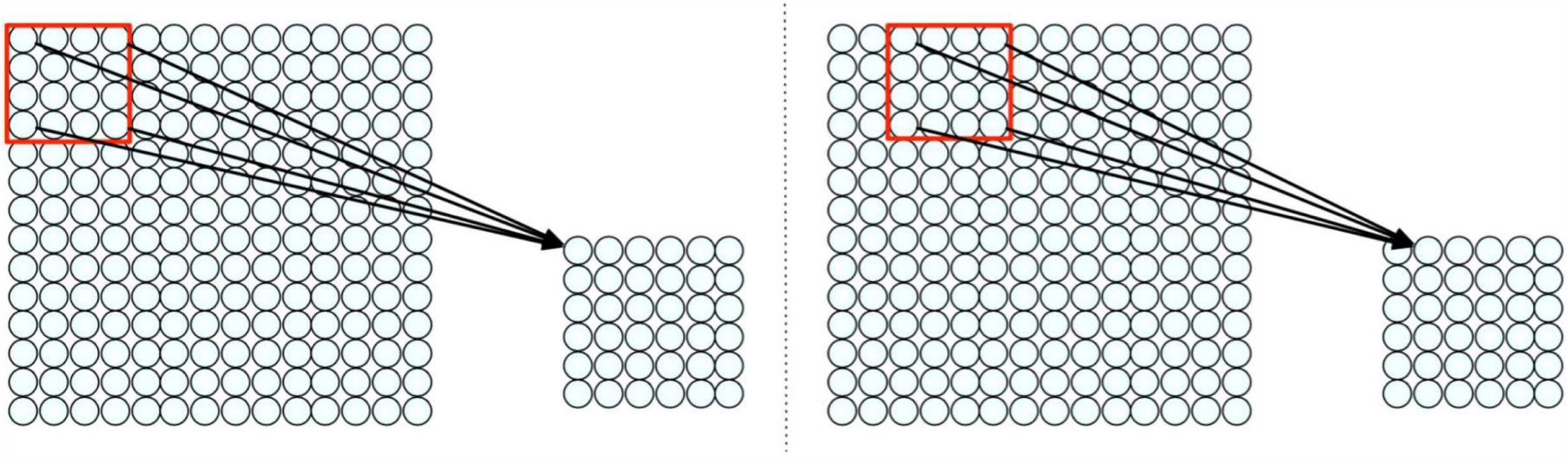
How can we use spatial structure in the input to inform the architecture of the network?



Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only “sees” these values.

Using Spatial Structure



Use a sliding window to define connections.

Convolution

Convolution: Toy Example

1	1	1	1
0	0	1	0
0	1	0	0
1	0	0	0

Image

4x4

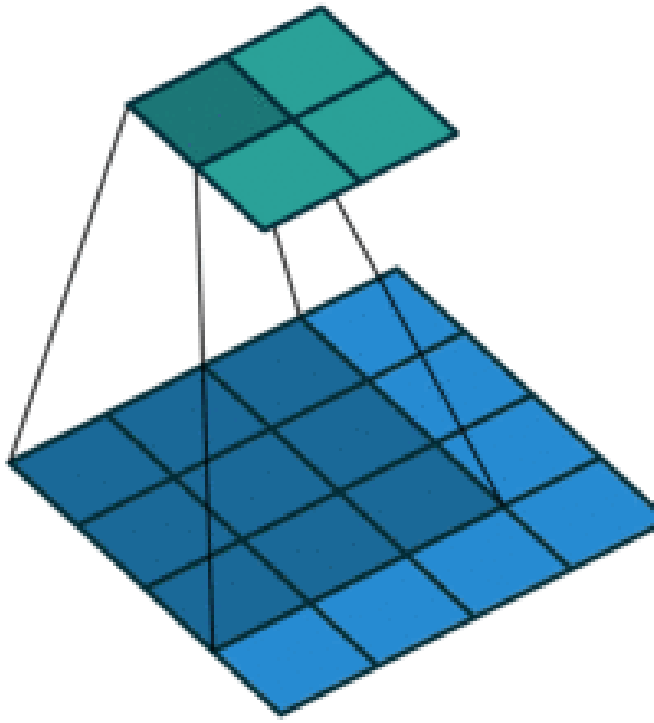
⊗

1	0	1
0	1	0
1	0	1

Filter (Kernel)

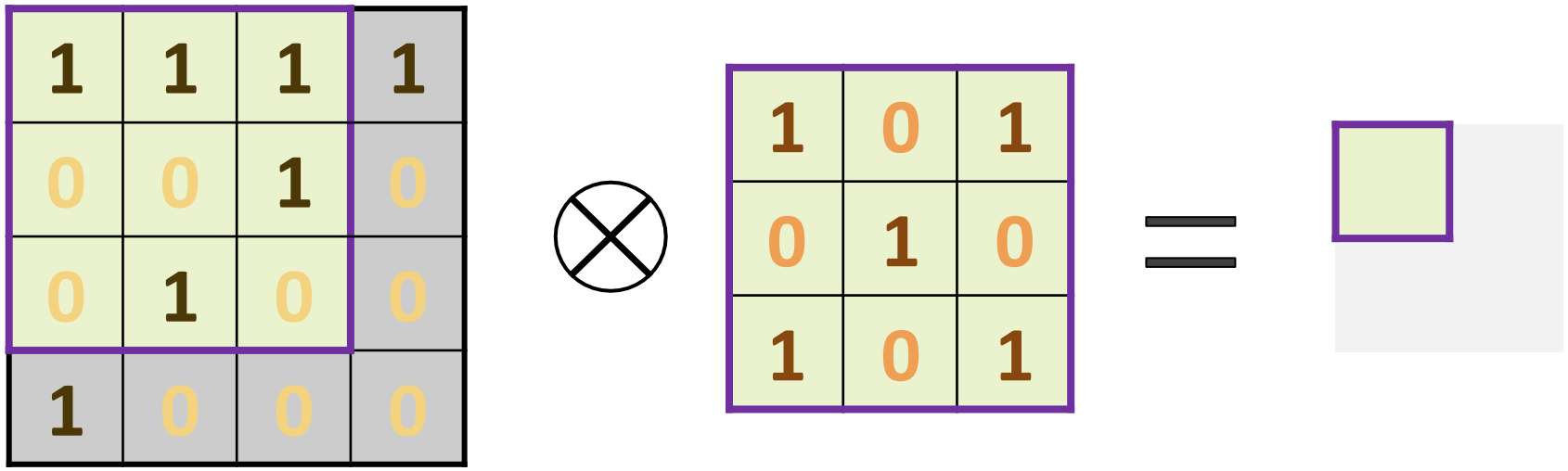
3x3

=



Convolution: Toy Example

Patch 1



Image

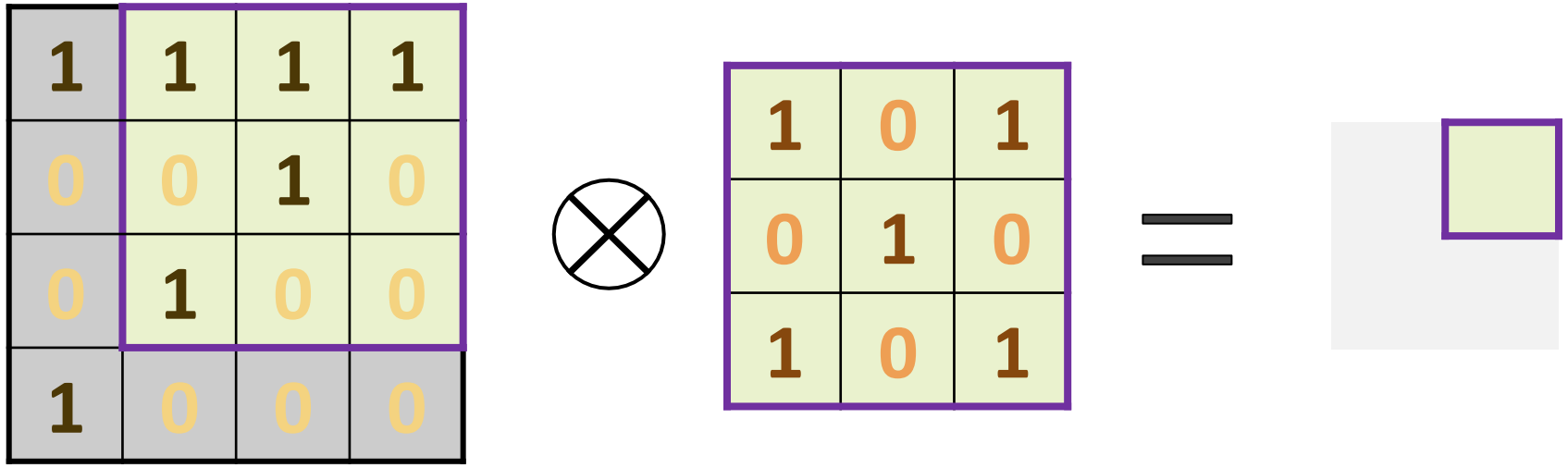
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

Patch 2



Image

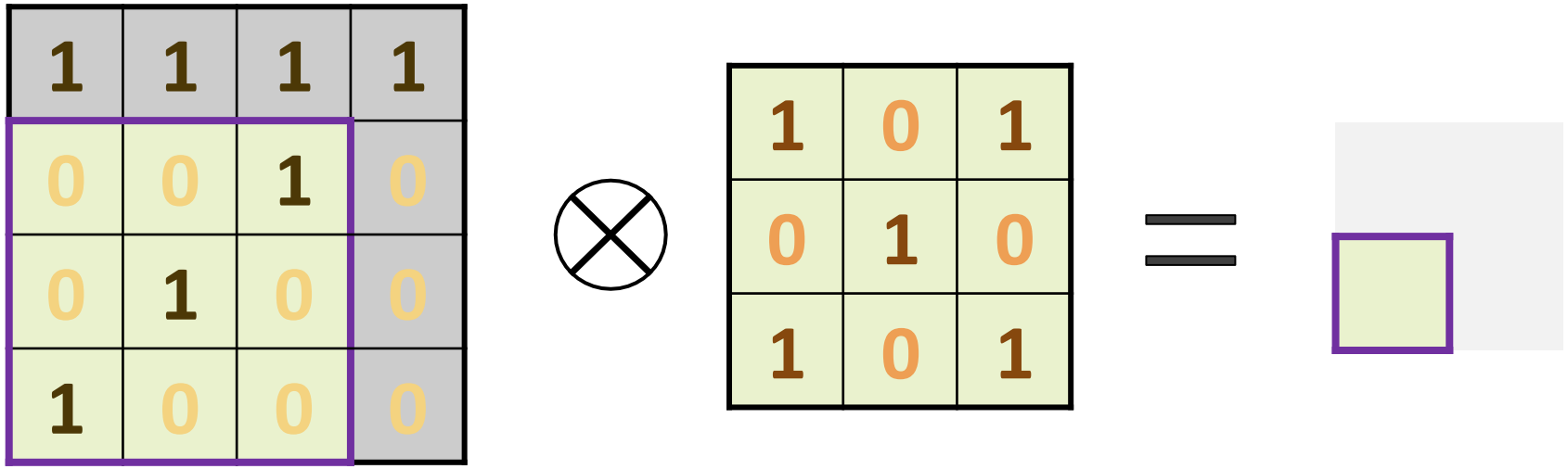
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

Patch 3



Image

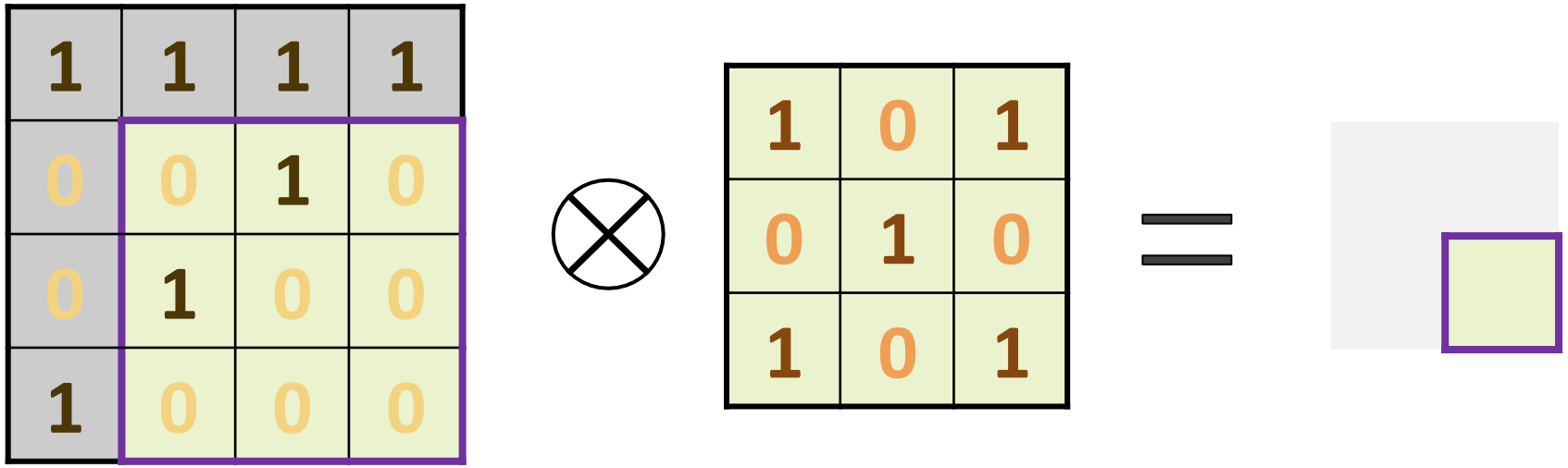
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

Patch 4



Image

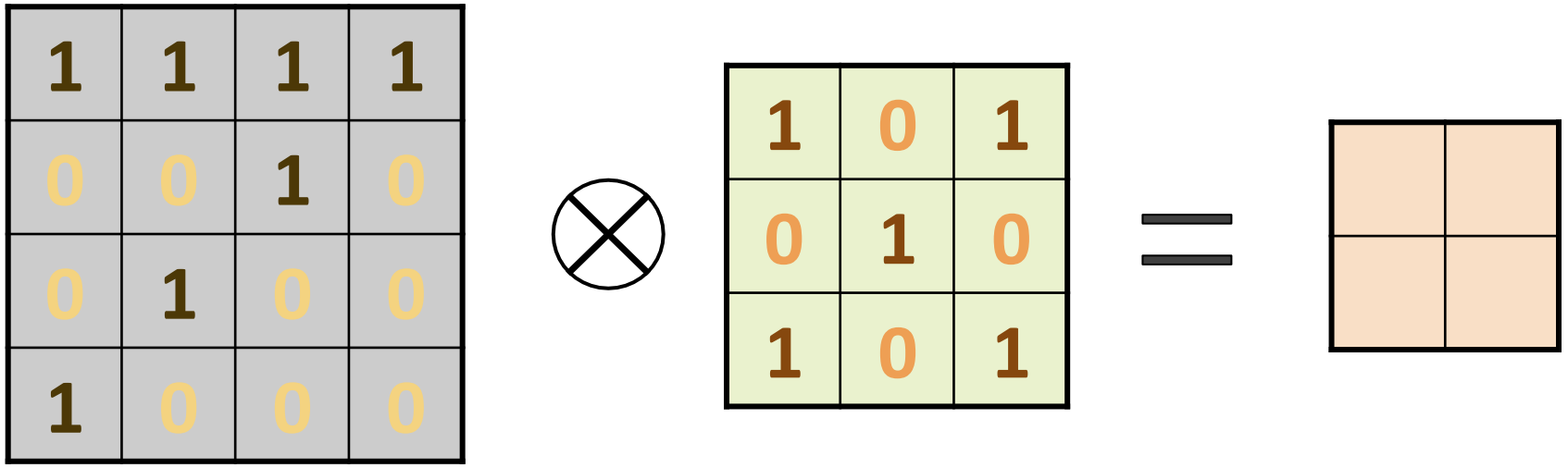
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

How many 3 x 3 patches in total if we have a 4 x 4 image?



Image

4x4

Filter (Kernel)

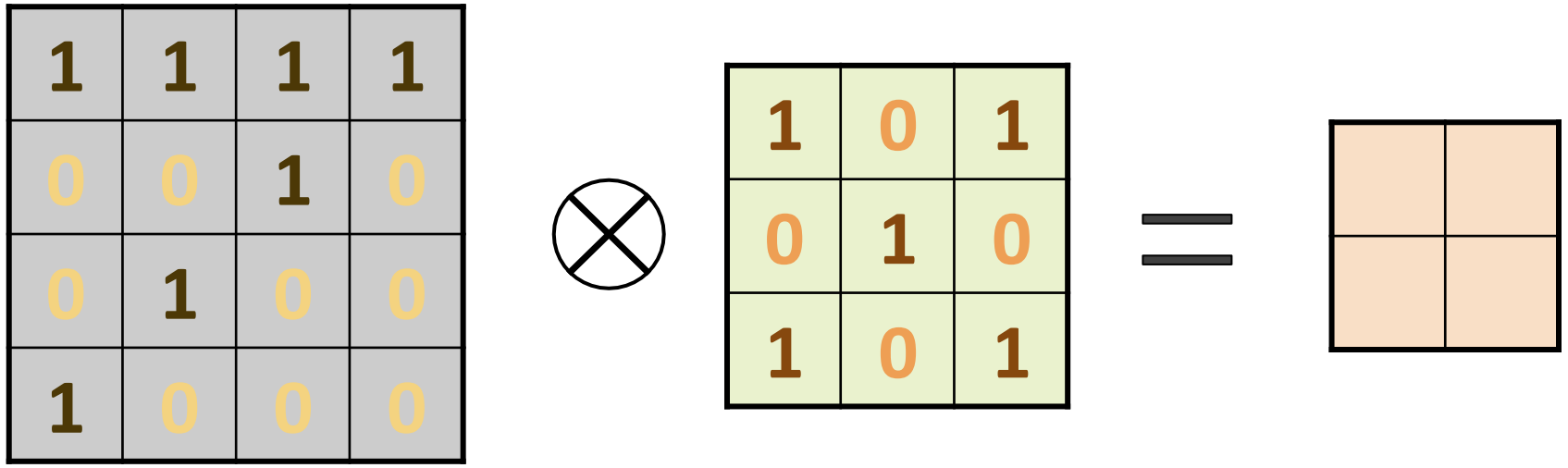
3x3

2 x 2 patches

(4-3+1) x (4-3+1)

Convolution: Toy Example

How many h x h patches in total if we have a R x R image?



Image

R x R

Filter (Kernel)

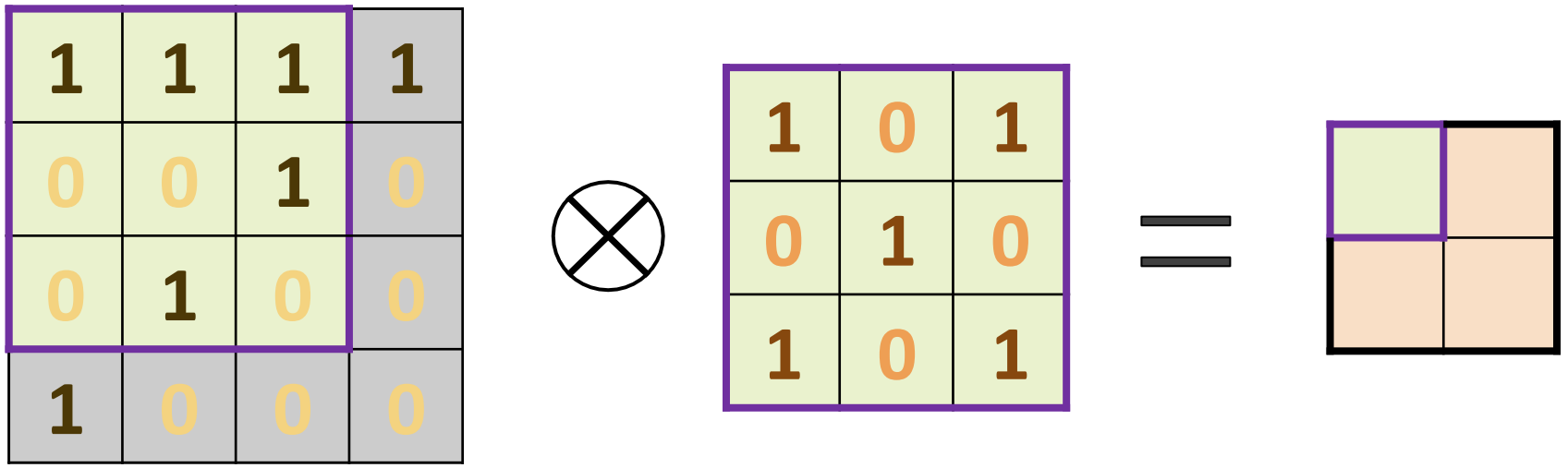
h x h

? patches

(R-h+1) x (R-h+1)

Convolution: Toy Example

- Calculate convolution for Patch 1: Element 1



Image

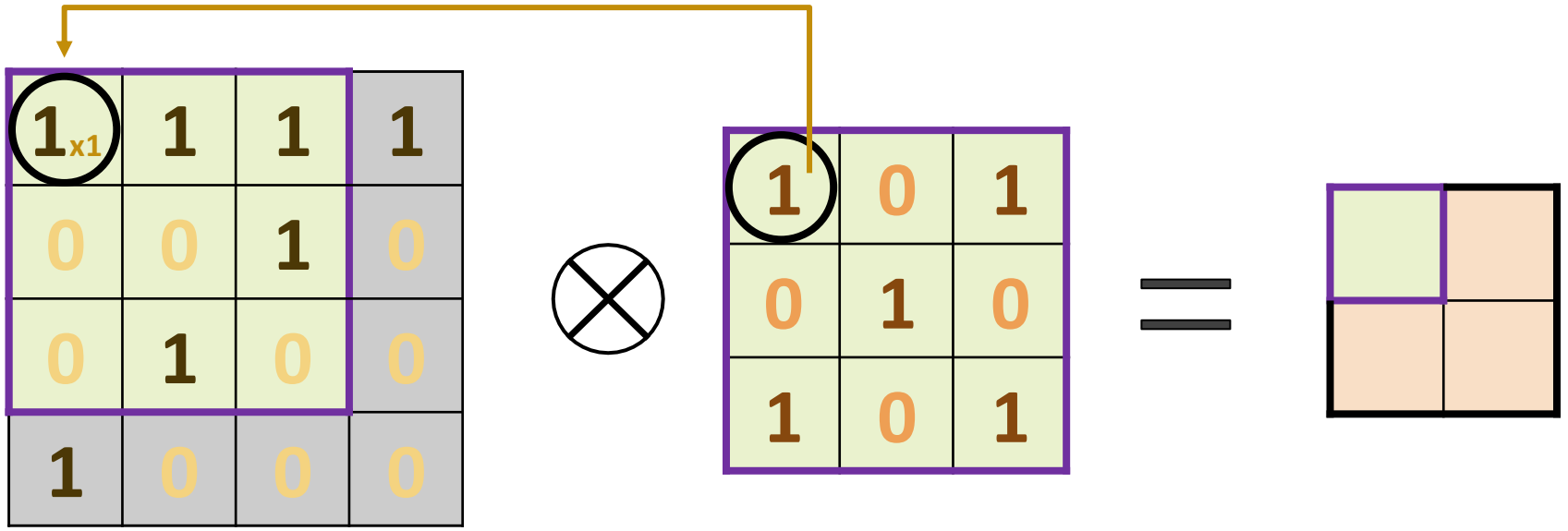
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

- Calculate convolution for Patch 1: Element 1



Image

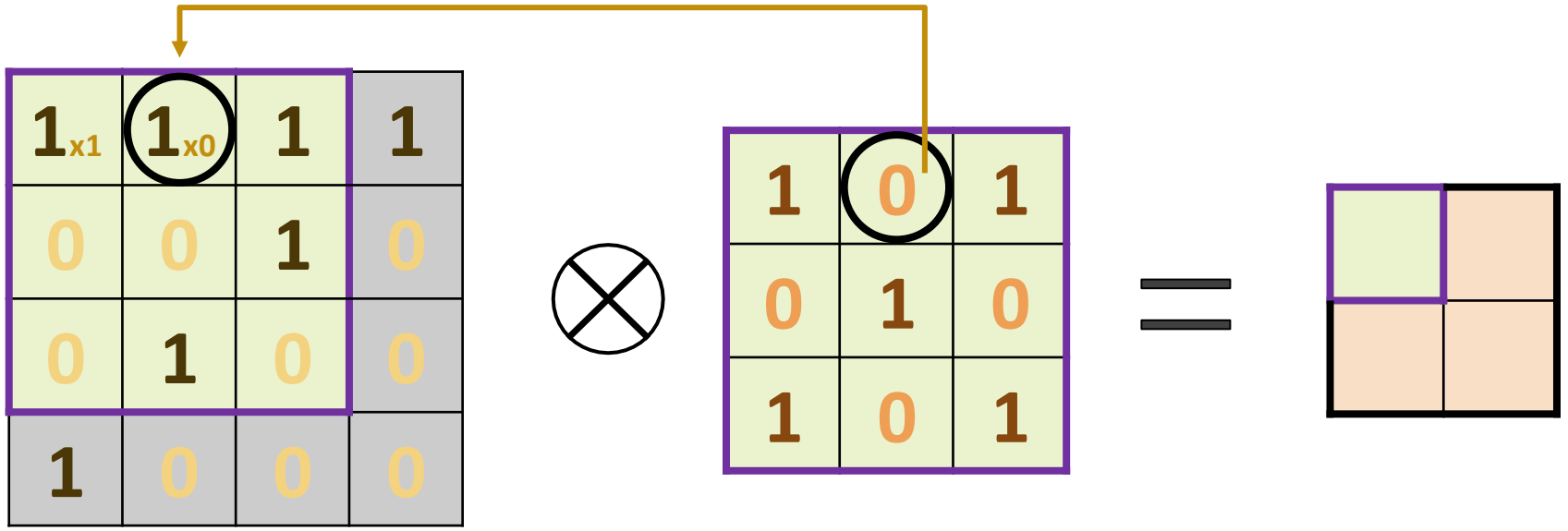
4x4

Filter (Kernel)

3x3

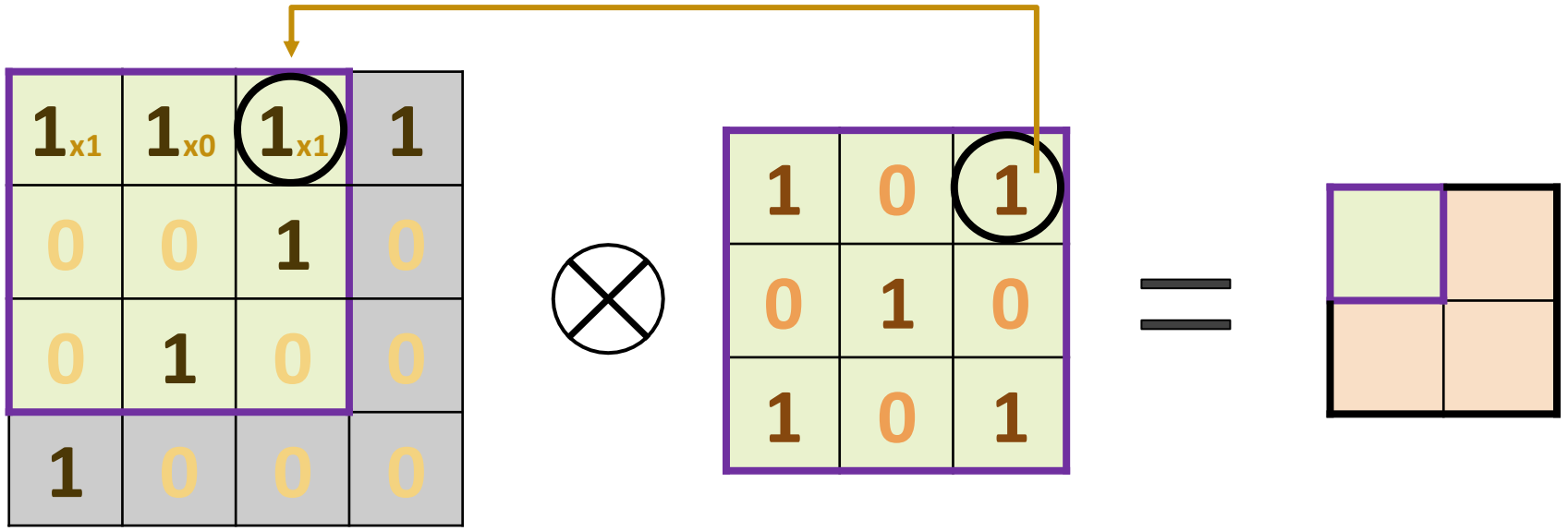
Convolution: Toy Example

- Calculate convolution for Patch 1: Element 2



Convolution: Toy Example

- Calculate convolution for Patch 1: Element 3



Image

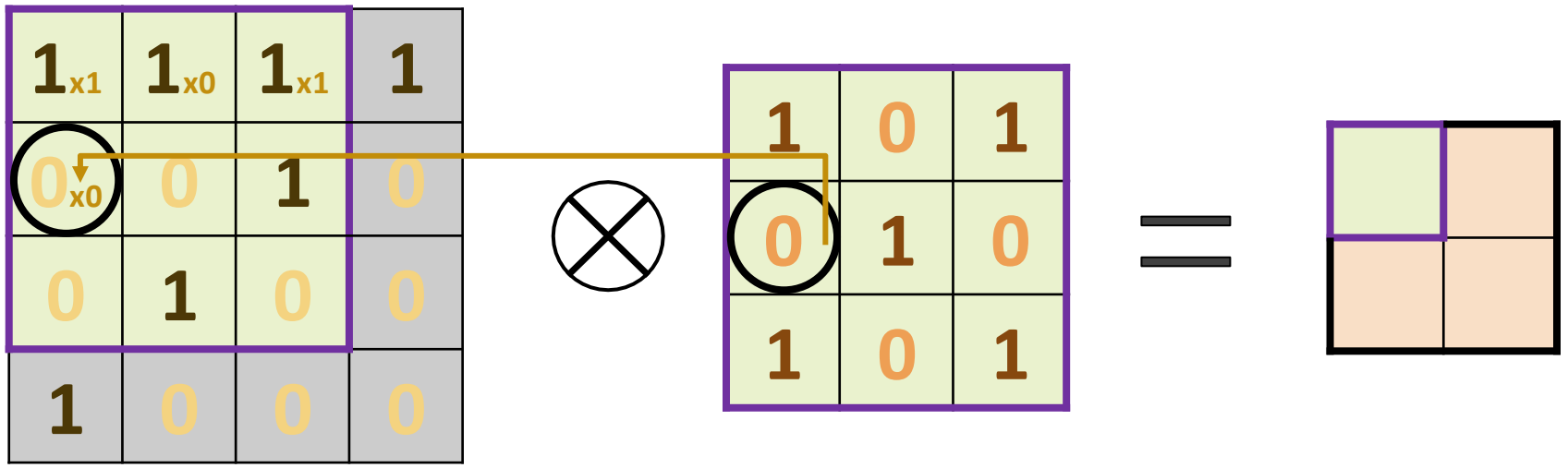
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

- Calculate convolution for Patch 1: Element 4



Image

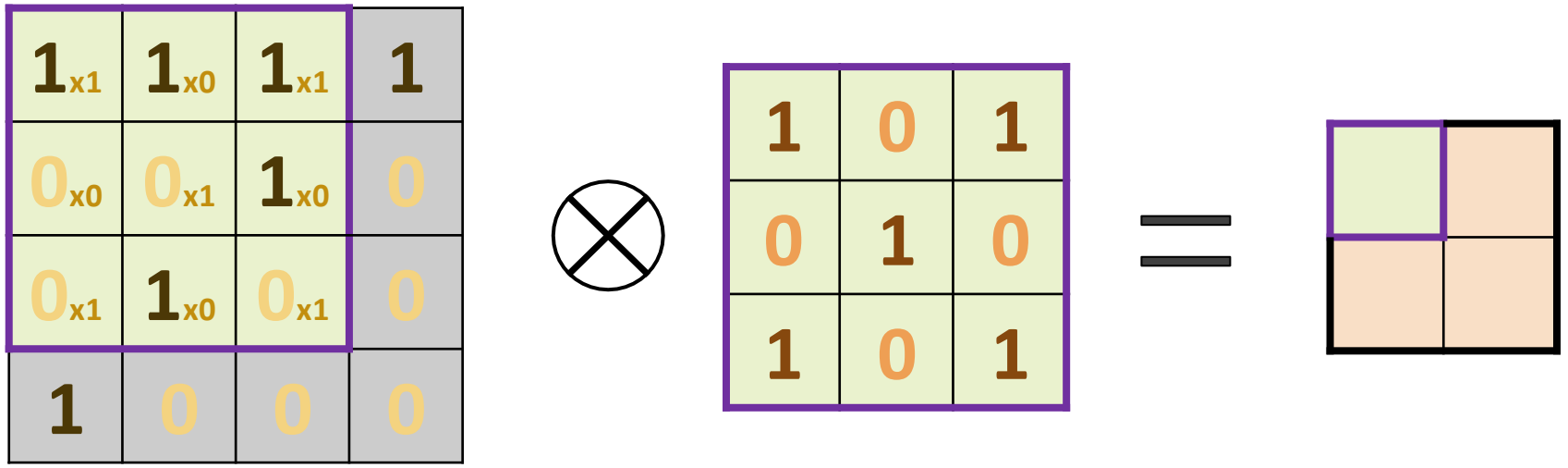
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

- Calculate convolution for Patch 1: All elements



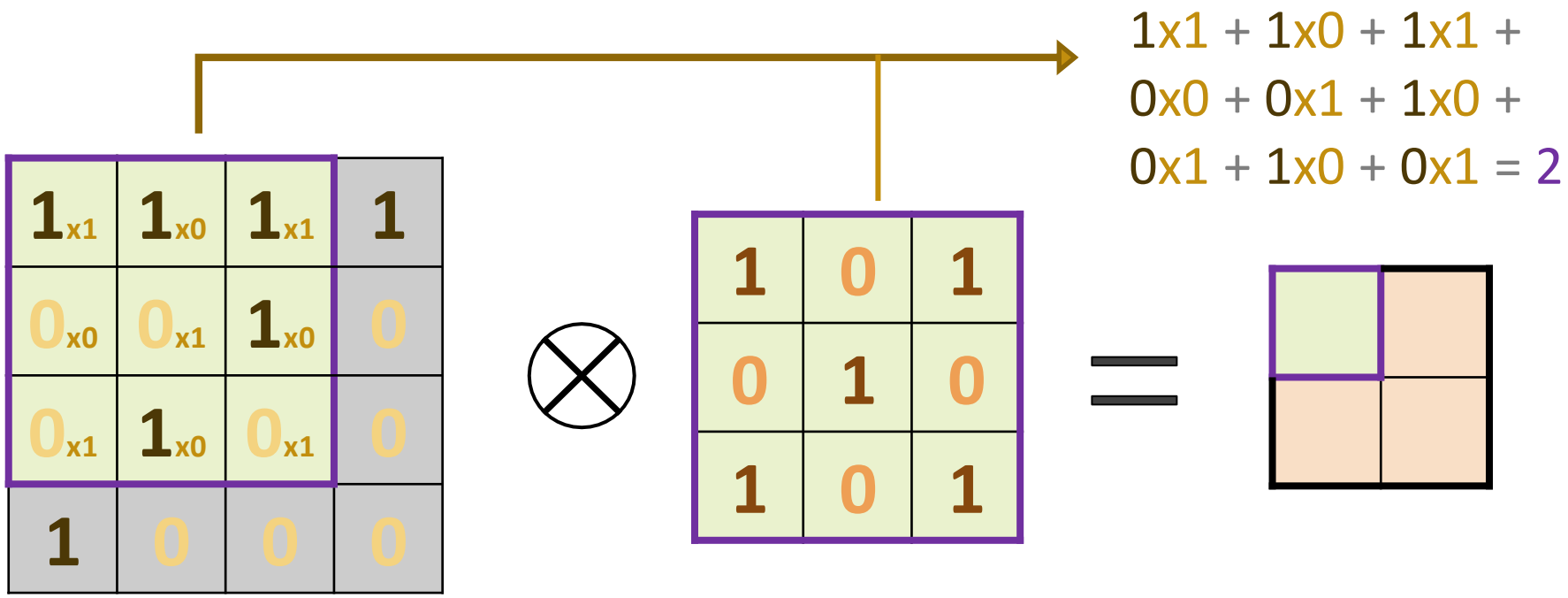
Image

4x4

Filter (Kernel)

3x3

Convolution: Toy Example



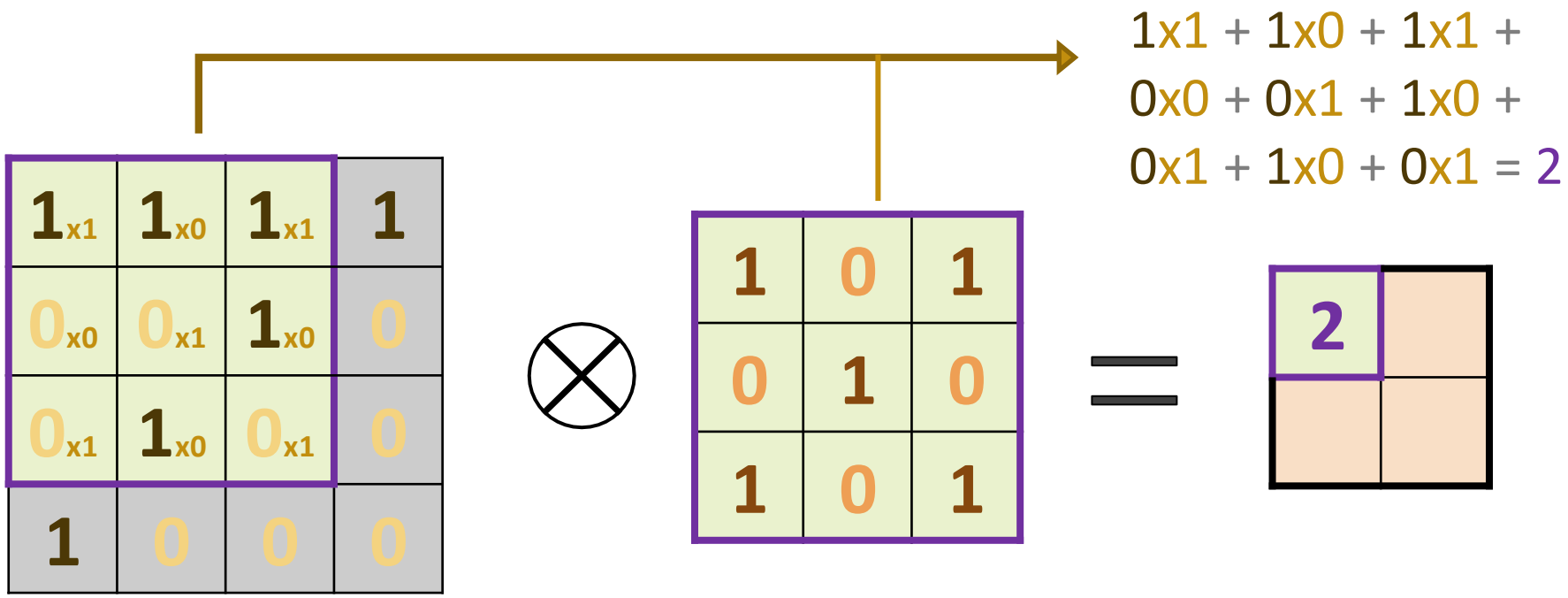
Image

4x4

Filter (Kernel)

3x3

Convolution: Toy Example



Image

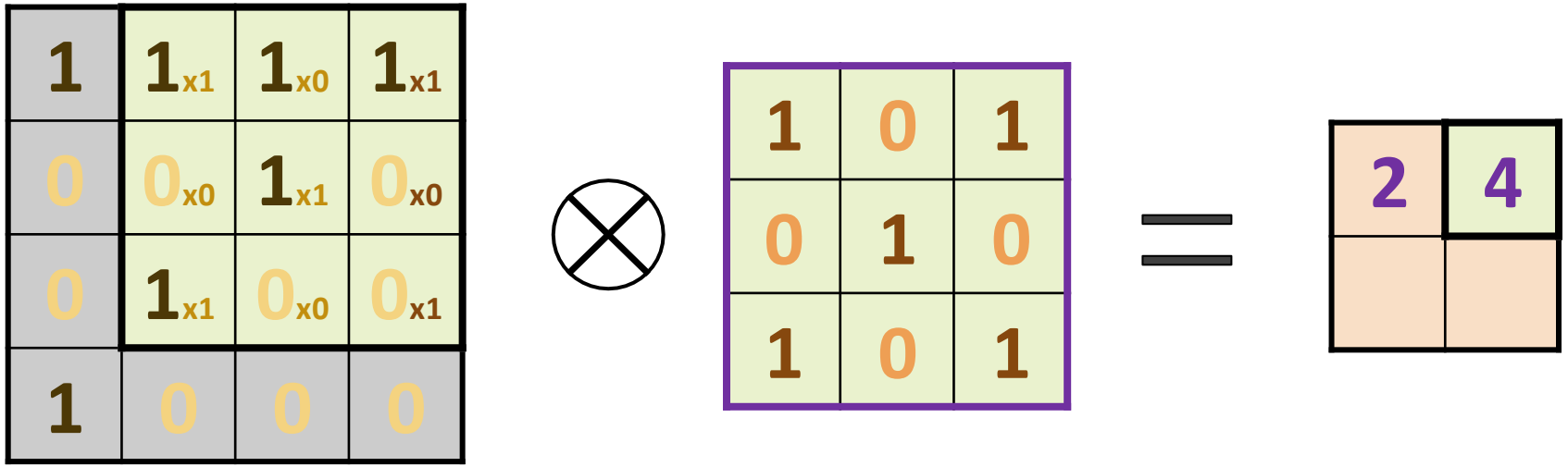
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

Patch 2



Image

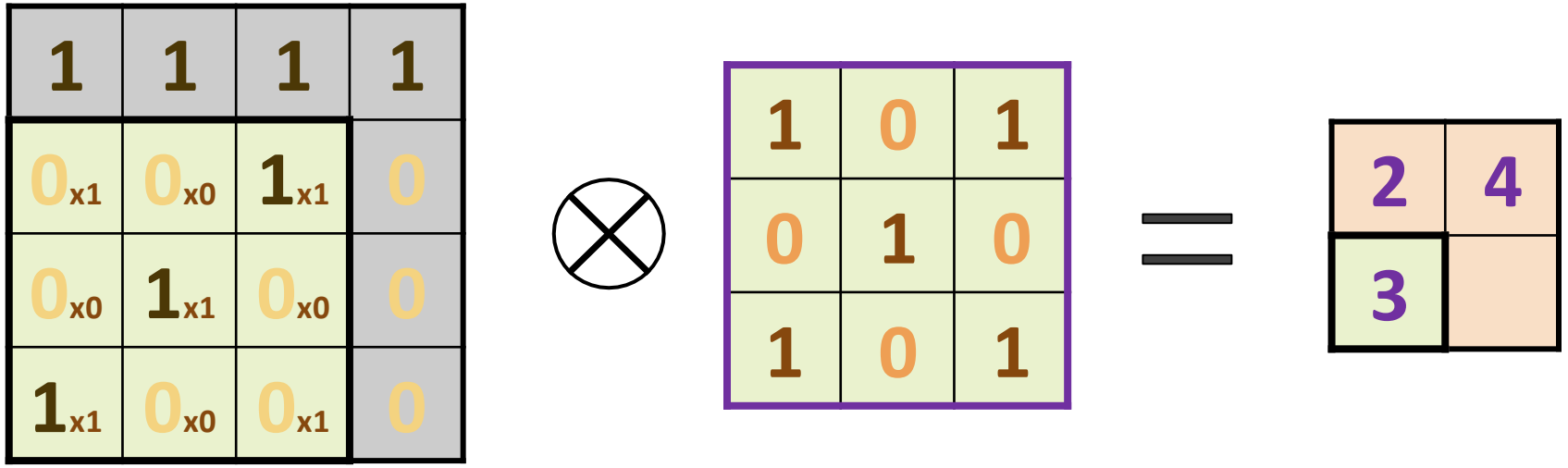
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

Patch 3



Image

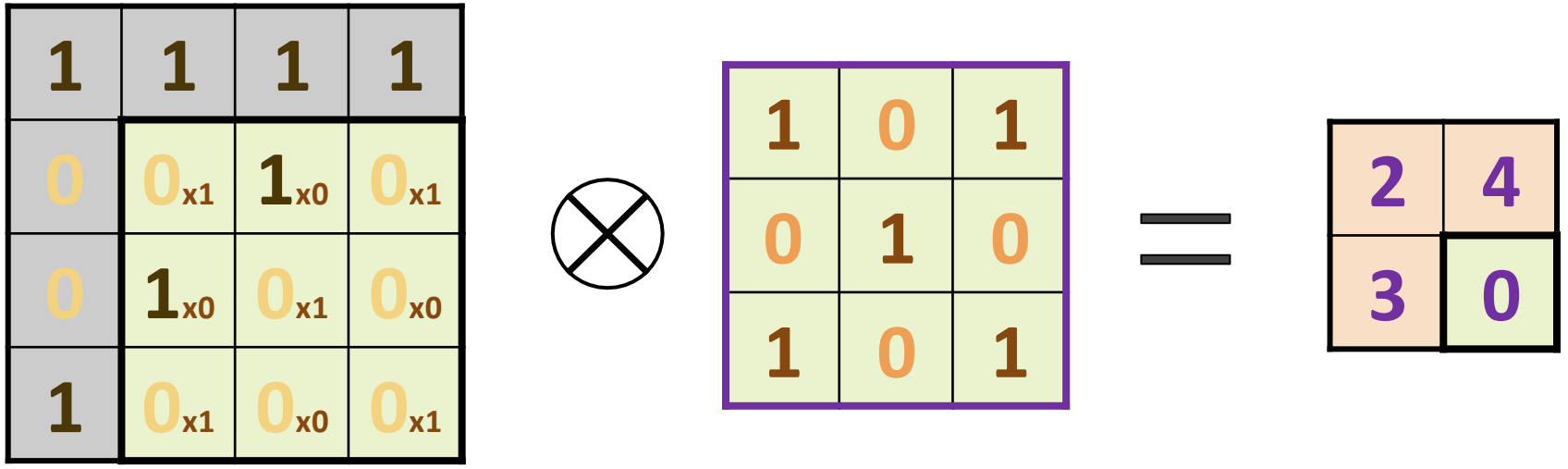
4x4

Filter (Kernel)

3x3

Convolution: Toy Example

Patch 4



Image

4x4

Filter (Kernel)

3x3

Producing Feature Maps

⊗

0	0	0
0	1	0
0	0	0

=

Filtered (no change)

Producing Feature Maps: Shift

$\otimes$

0	0	0
1	0	0
0	0	0

$=$

Shifted right by

one pixel

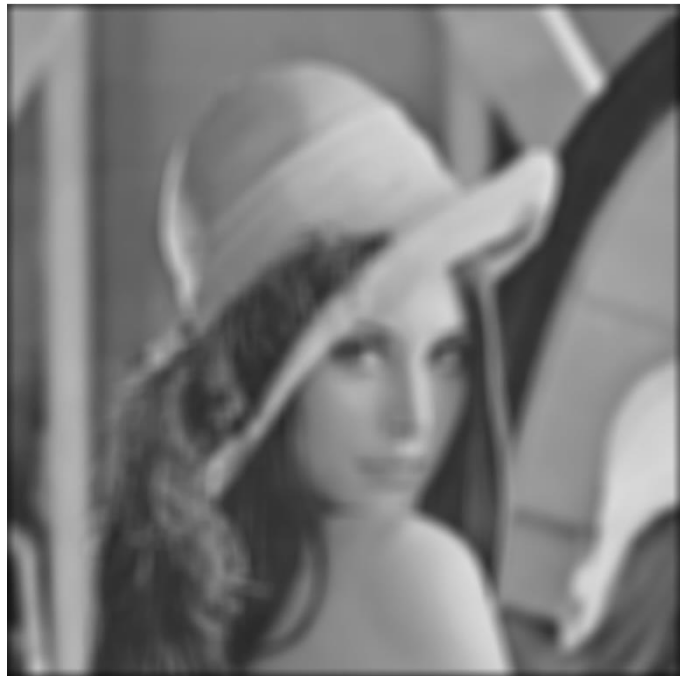
Producing Feature Maps: Blur

$\otimes$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$=$



Blurred

Producing Feature Maps: Sharpen

$\otimes$

$\frac{1}{5}$

0	-1	0
-1	5	-1
0	-1	0

=

Sharpen

Producing Feature Maps: Edge Detection

$\otimes$

$\frac{1}{9}$

-1	-1	-1
-1	8	-1
-1	-1	-1

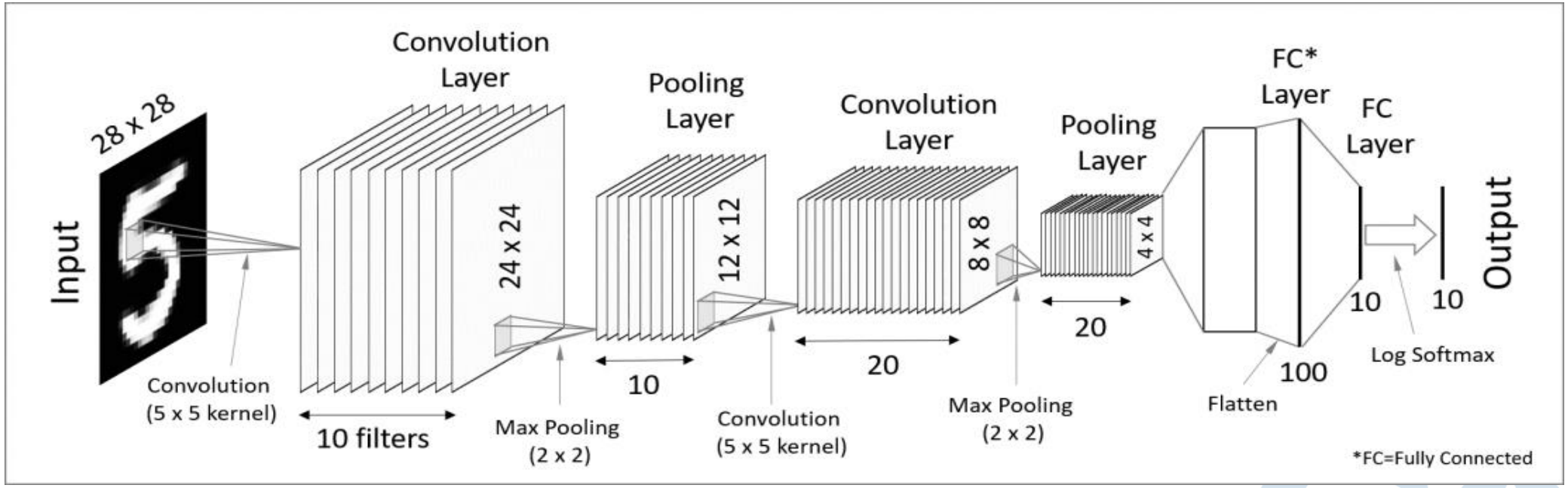
$=$

Edge Detection

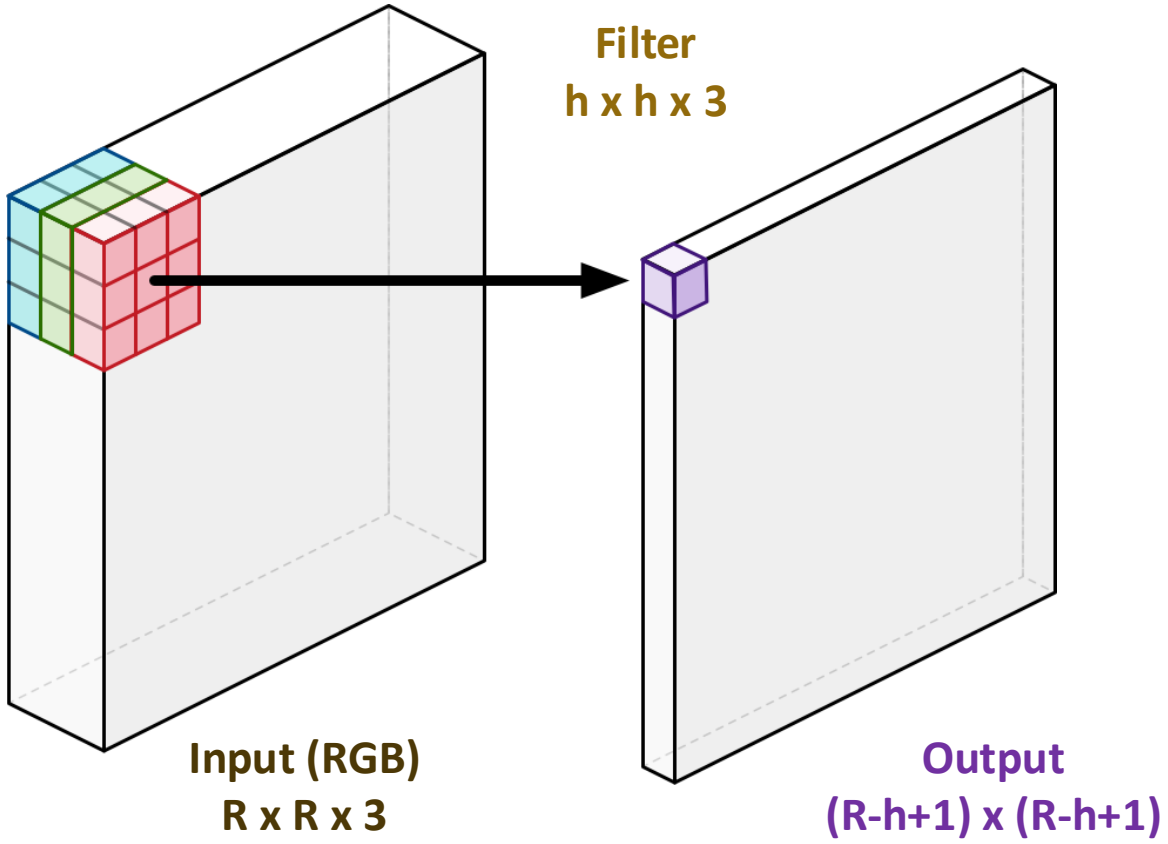
(Details)

Multiple filters (in one layer)

- MNIST: $R = 28$, we use $d = 10$ filters of 5×5 ($h = 5$)
  - Input dimension: $28 \times 28 = 784$; Feature Map size 24×24 : $R - h + 1 = 24$
  - Total weights to learn $5 \times 5 \times 10 + 10 = 260$, including 10 bias
  - Hidden layer 1 dimension (for neurons in hidden layer 1): $10 \times 24 \times 24 = 5760$



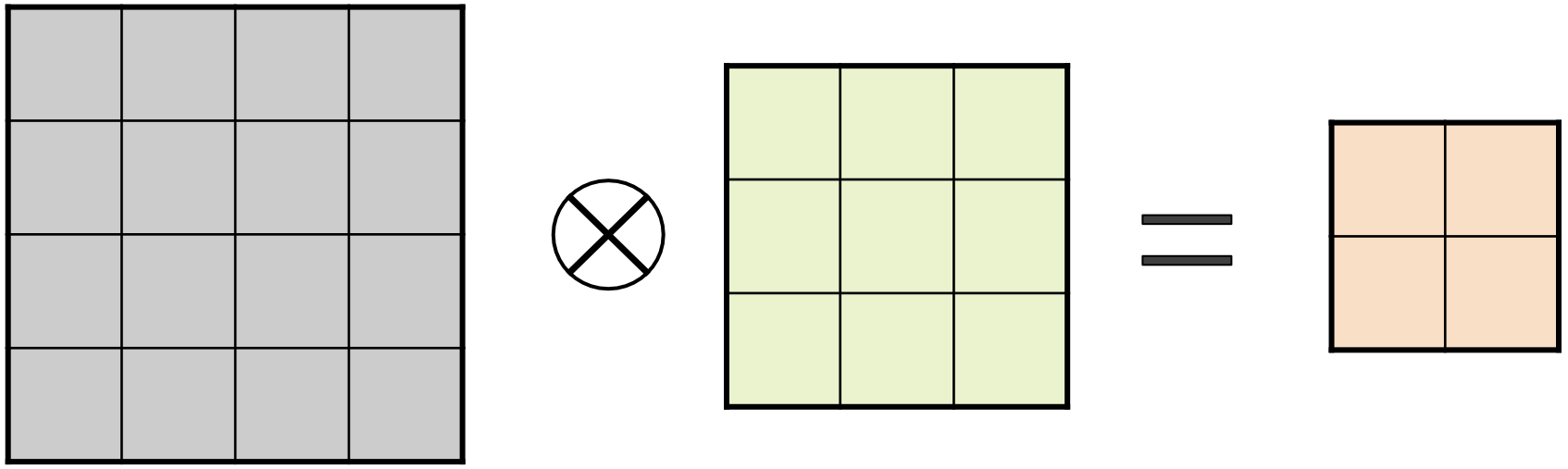
Convolution for color images



- What if we have a color image ?
- Input $R \times R \times 3$ (RGB) is a 3-way matrix
  - Tensor
- We use an $h \times h \times 3$ filter (tensor).
  - But when we write code, we still write h x h, we don't need to write x 3
- Output: $(R - h + 1)^2$ matrix
  - Not a tensor

Convolution: Zero Padding

Output is 2 x 2 when we have a 4 x 4 input passing through a 3 x 3 filter



Input

4 x 4

Filter (Kernel)

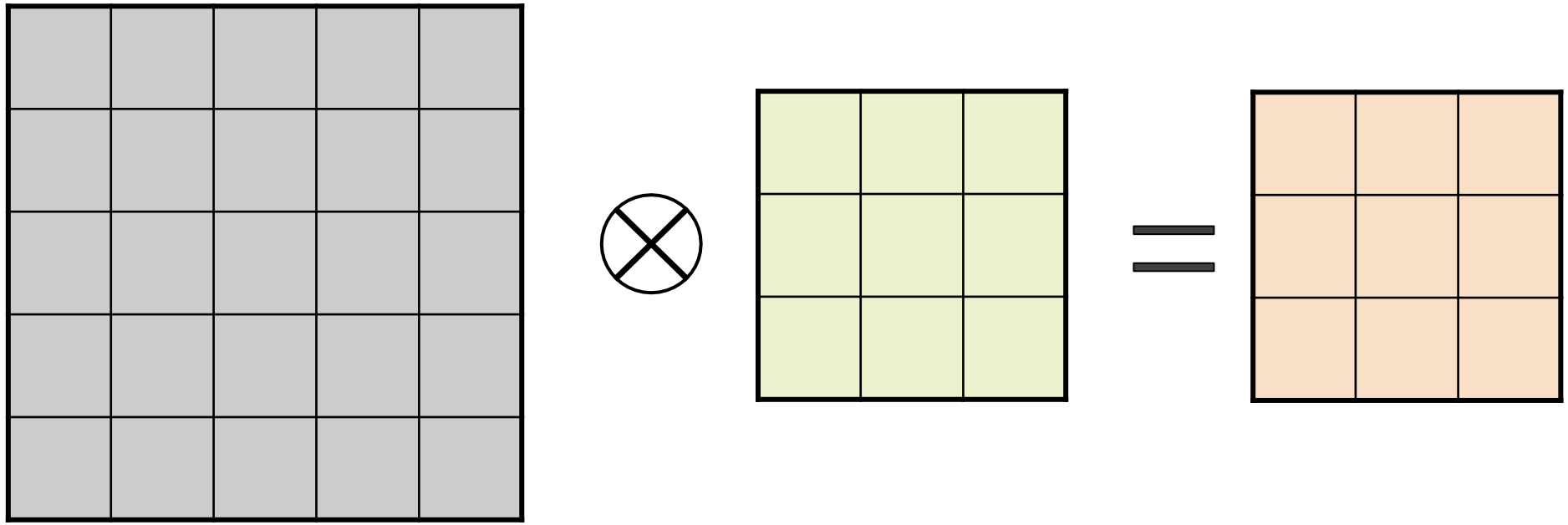
3 x 3

Output (of convolution):

2 x 2

Convolution: Zero Padding

Output is 3 x 3 when we have a 5 x 5 input passing through a 3 x 3 filter



Input

5 x 5

Filter (Kernel)

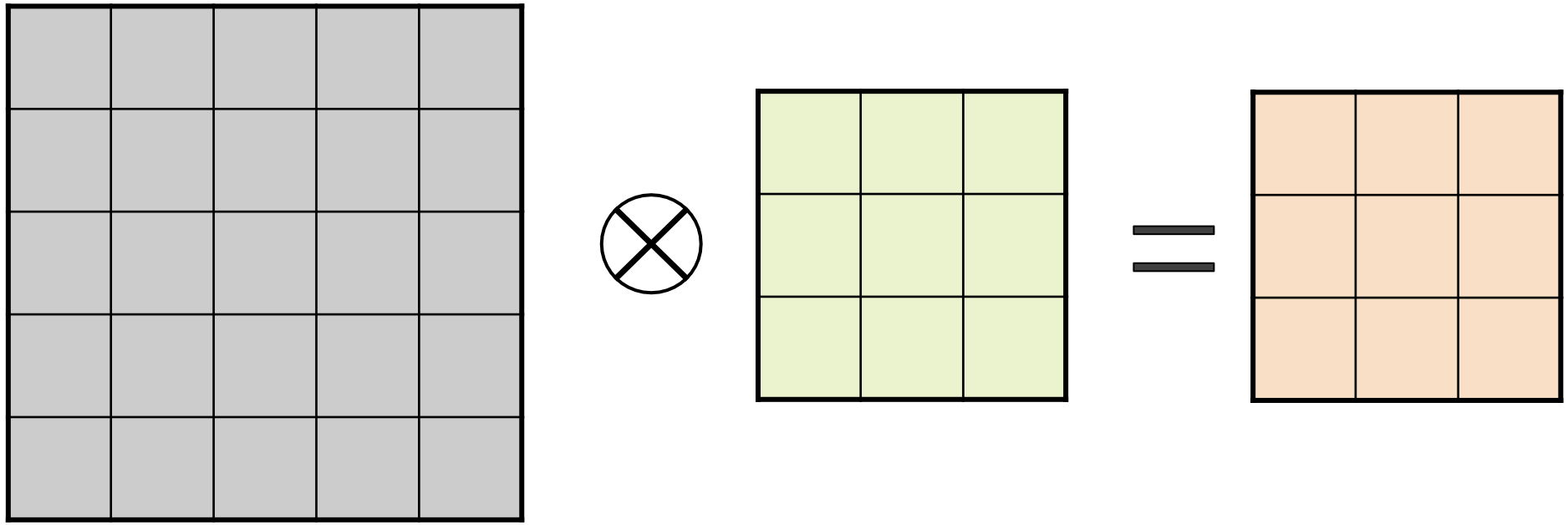
3 x 3

Output:

3 x 3

Convolution: Zero Padding

The output is smaller than the input



Input

R x R

Filter (Kernel)

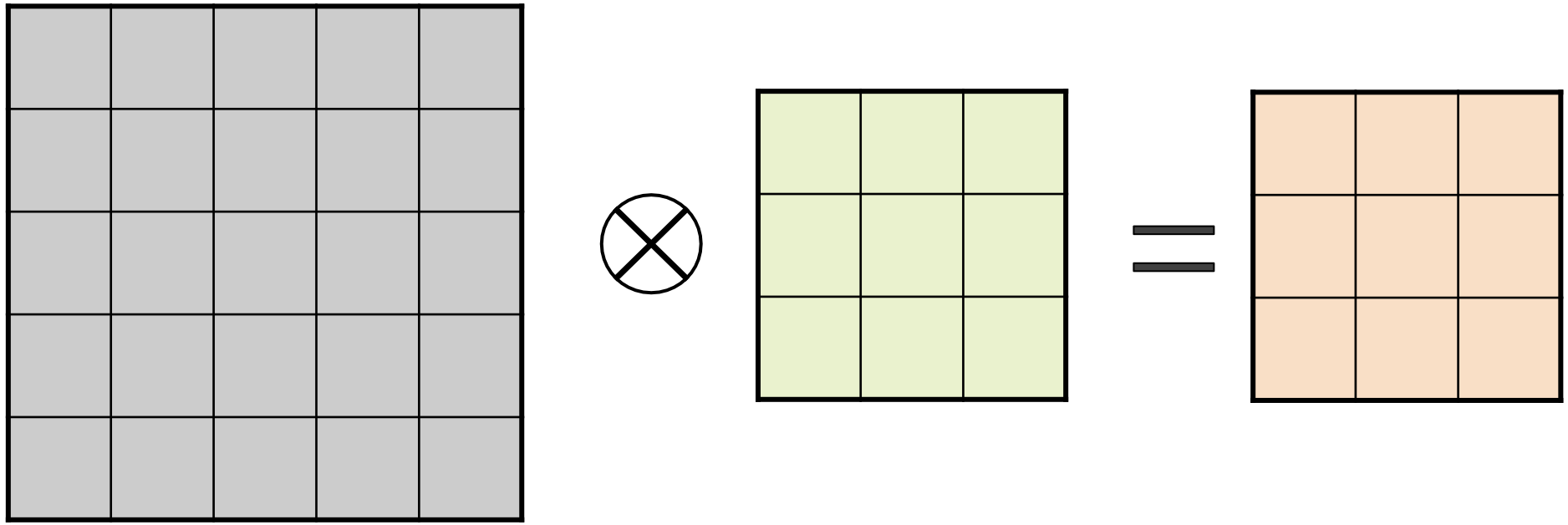
3 x 3

Output:

(R-2) x (R-2)

Convolution: Zero Padding

The output is smaller than the input



Input

R x R

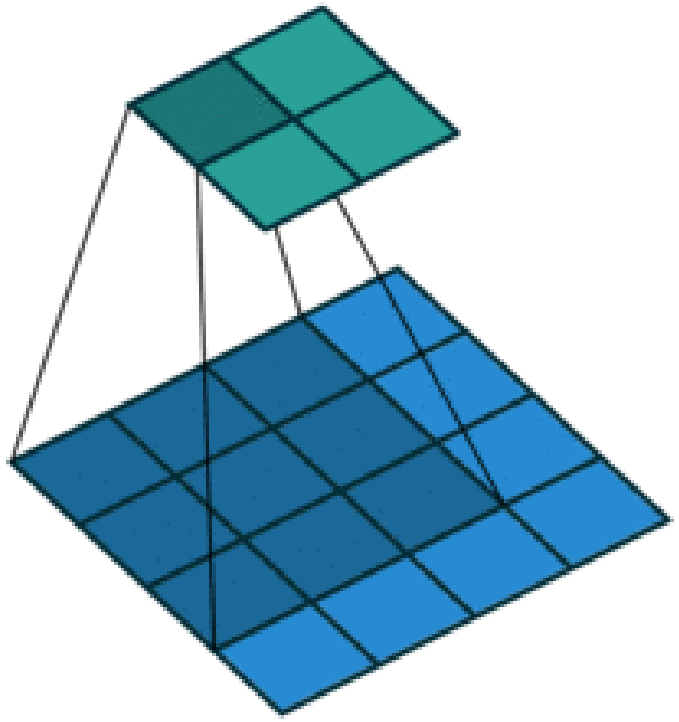
Filter (Kernel)

3 x 3

Output:

(R-2) x (R-2)

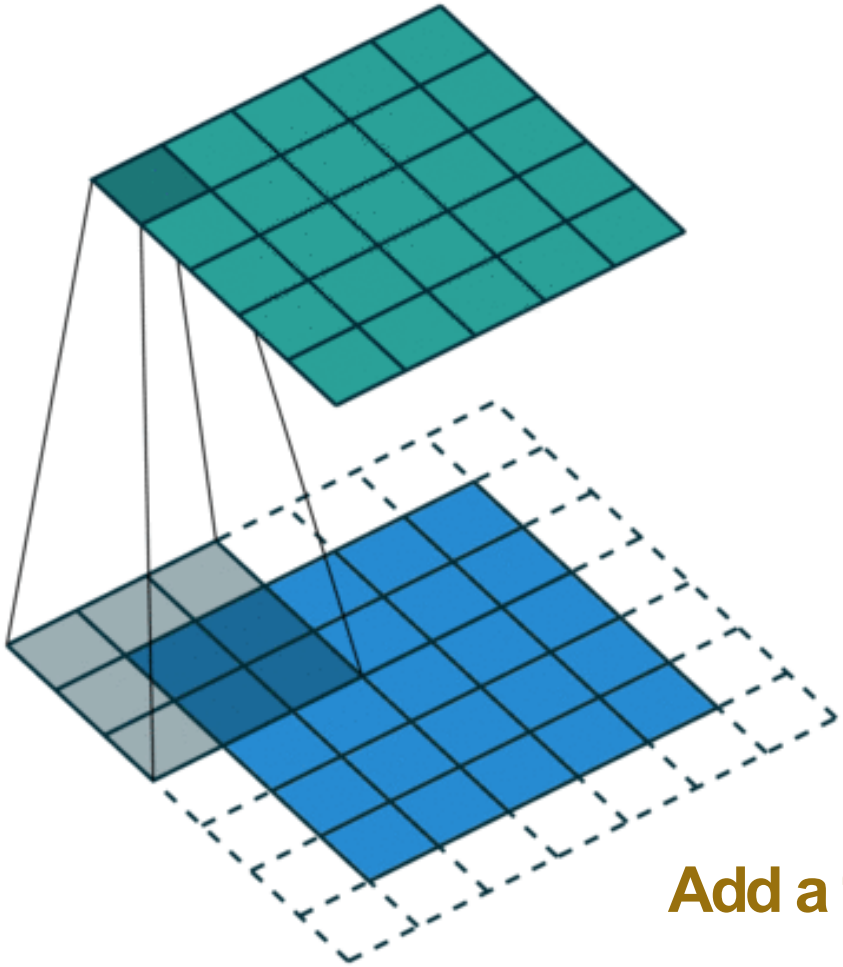
Zero Padding



https://theano-pymc.readthedocs.io/en/latest/tutorial/conv_arithmetic.html

- Output is smaller than Input
- An $R \times R$ input passing through an 3×3 filter:
  An $(R-2) \times (R-2)$ output
- An $R \times R$ input passing through an $h \times h$ filter:
  An $(R-h+1) \times (R-h+1)$ output

Zero Padding

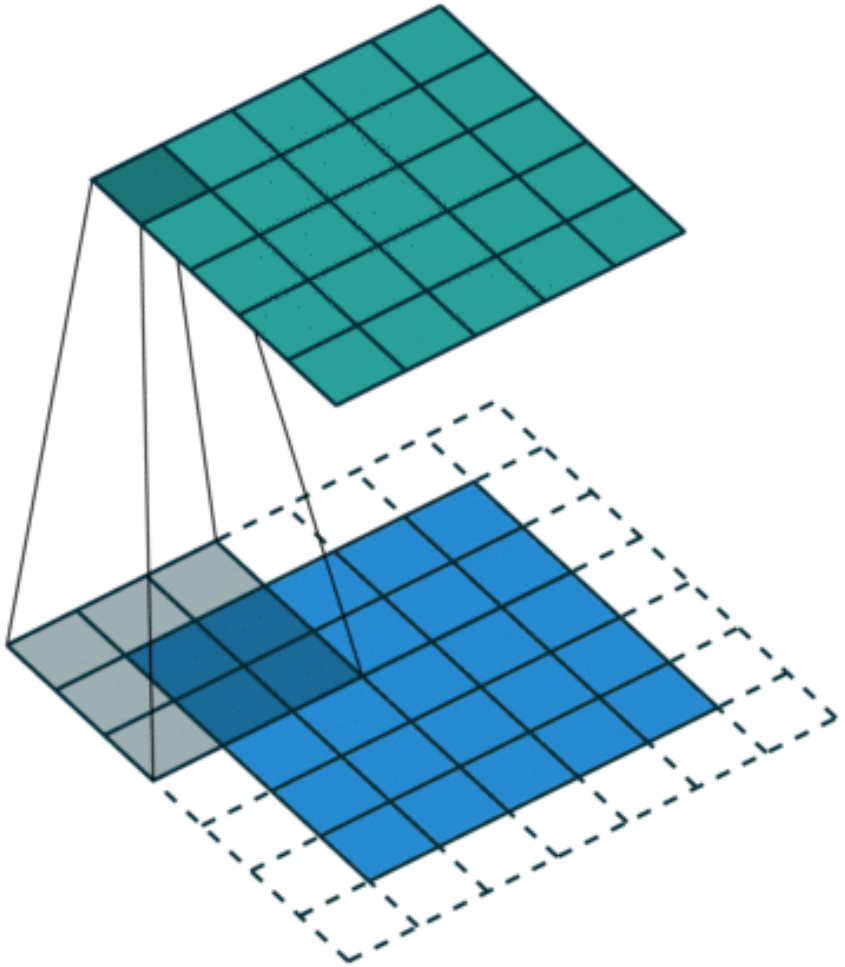


0	0	0	0	0	0	0	0
0							0
0							0
0							0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

Add a "boarder" of all-zeros.

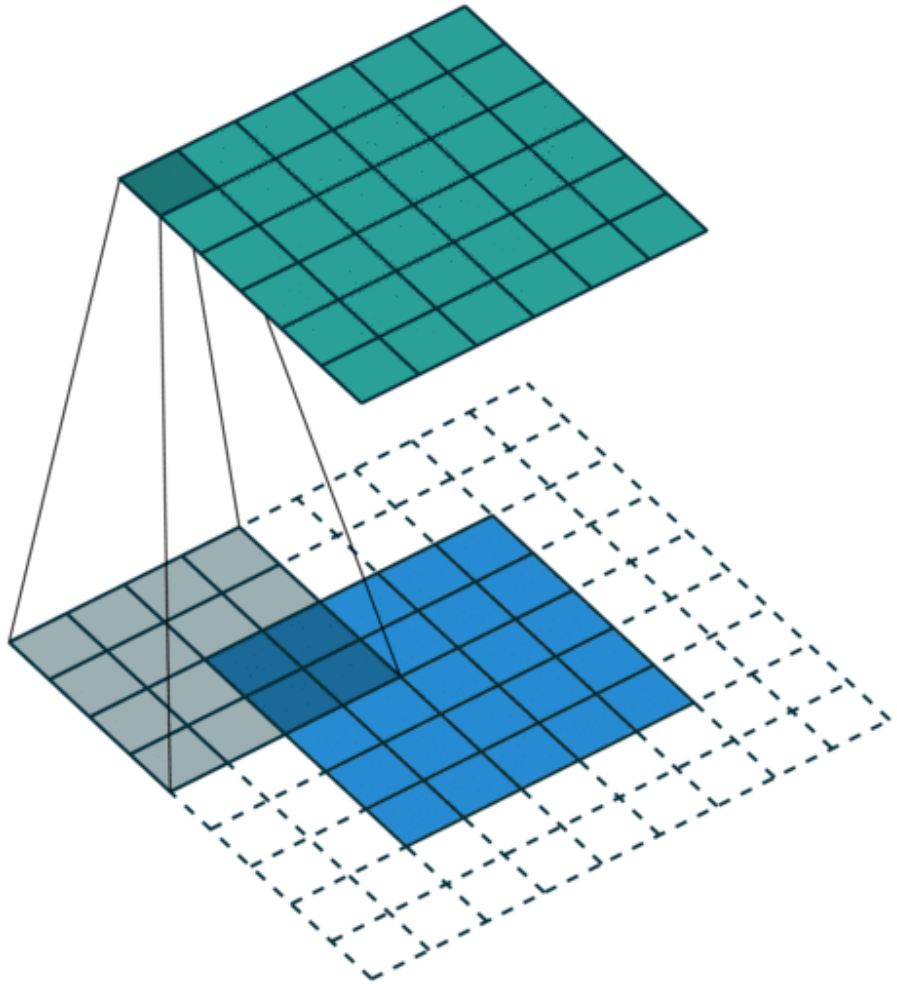
Zero-padding added to image

Zero Padding



- Add a “boarder” of all-zeros.
- Increase the input shape:
  - From $R \times R$ to $(R+2) \times (R+2)$
- Passing through a 3×3 filter
  - Output is having the same size as Input
  - Called same zero padding
- An $R \times R$ input passing through an $h \times h$ filter with zero padding:
  An $(R-h+3) \times (R-h+3)$ output

Multiple Zero Paddings



- You can add multiple zero paddings

  A 5×5 input passing through a 4×4 filter with 2 zero padding: 6×6 output
  - Can be larger than input

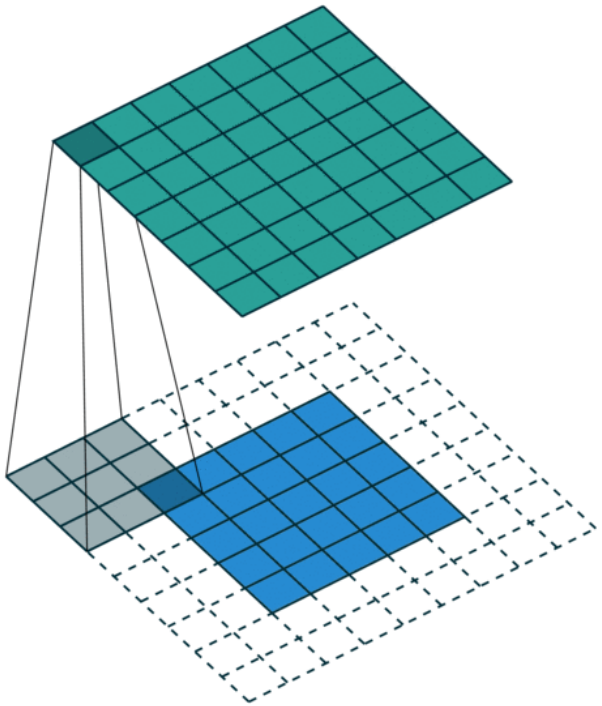
  An $R \times R$ input passing through an $h \times h$ filter with p zero padding:
  An $(R - h + 2p + 1)^2$ output

Typical Zero paddings

- An $R \times R$ input passing through an $h \times h$ filter with p zero padding gets an $(R - h + 2p + 1) \times (R - h + 2p + 1)$ output

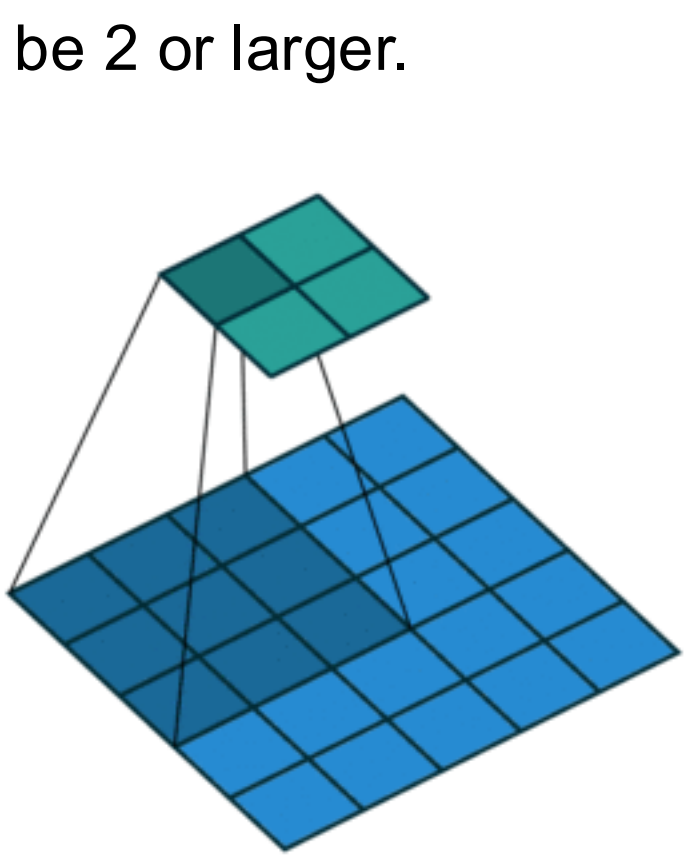
- Half (same) paddings for odd size of filter
  - $R \times R$ input, $(2p + 1) \times (2p + 1)$ filter, p zero padding:
  - $R \times R$ output: $R - (2p + 1) + 2p + 1 = R$

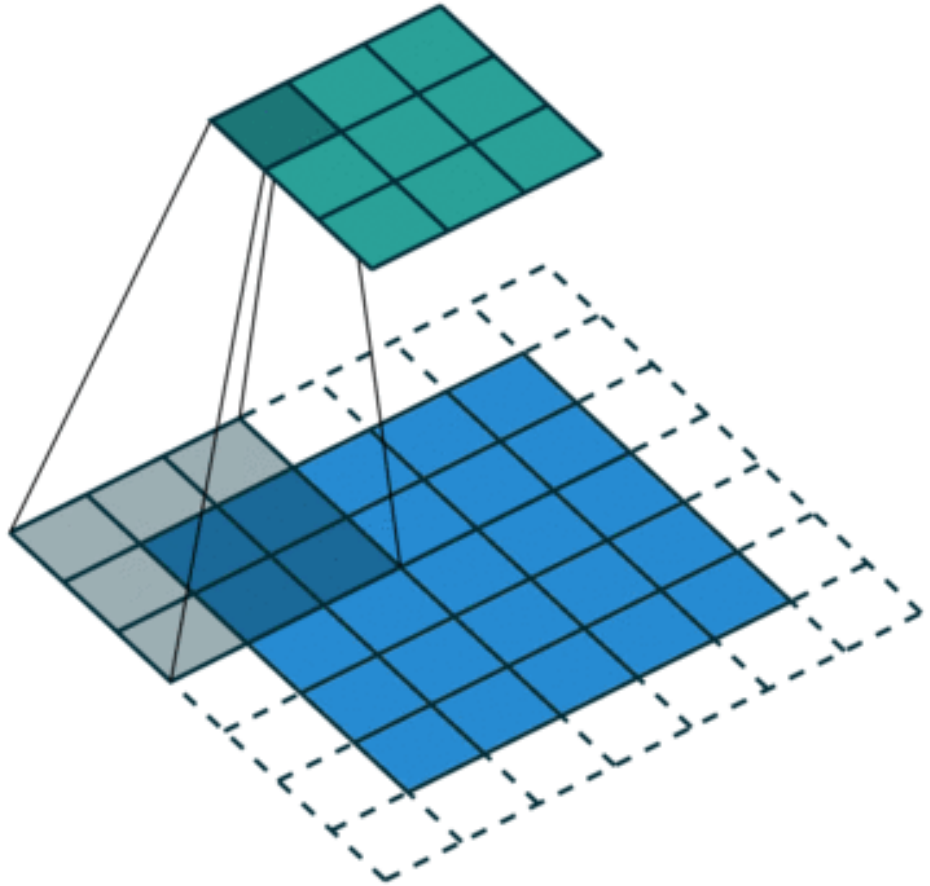
- Full paddings
  - $R \times R$ input, $(p + 1) \times (p + 1)$ filter, p zero padding:
  - $(R + p) \times (R + p)$ output: $R - (p + 1) + 2p + 1 = R + p$



Strides

- In the previous examples, the filter moves 1 step each time (Stride 1).
- The stride can be 2 or larger.





Paddings and Strides

- Input size $R \times R$, filter (kernel) size $h \times h$, zero padding p, stride s
- Output (of the convolution) size $\left(\left\lfloor \frac{R-h+2p}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{R-h+2p}{s} \right\rfloor + 1\right)$
- Paddings increase the output size
- strides decrease the output size
- Use them according to your tasks

Multiple filters (in one layer)

- If we have d filters in one layer, each is with size $h \times h$
  - The layer dimension for weights is $h \times h \times d$
  - Given an $R \times R$ input, we receive d feature maps of size $(R - h + 1)^2$

- MNIST: $R = 28$, we use $d = 10$ filters of 5×5 ($h = 5$)
  - We have 10 feature maps of 24×24
  - Input dimension: $28 \times 28 = 784$
  - Hidden layer 1 dimension (for neurons in hidden layer 1): $10 \times 24 \times 24 = 5760$
  - Layer dimension too large
    - Cannot increase h (specified to extract *local* features)
    - Pooling (subsampling) – similar as using a stride larger than 1

Pooling

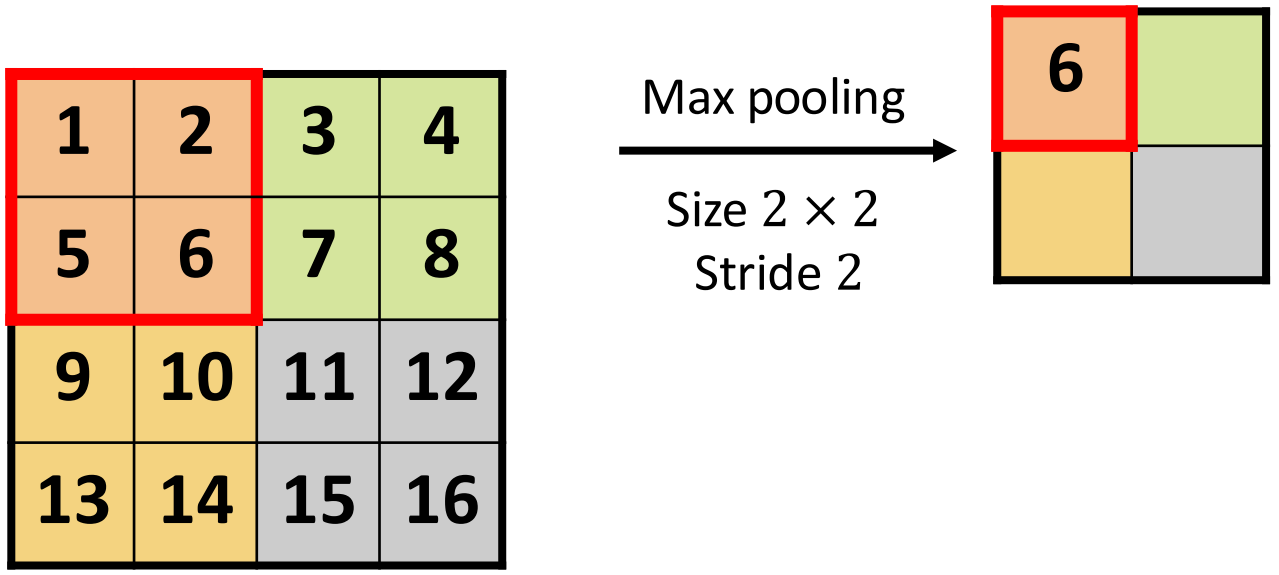
How else can we downsample and preserve spatial invariance?

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- Dividing a $k \times k$ feature map into $(k/2) \times (k/2)$ patches
- Striding a 2×2 (pooling) filter across the feature map
- Take Max value of each 2×2 patch
- Stride 2 (generally equals to the pool size)

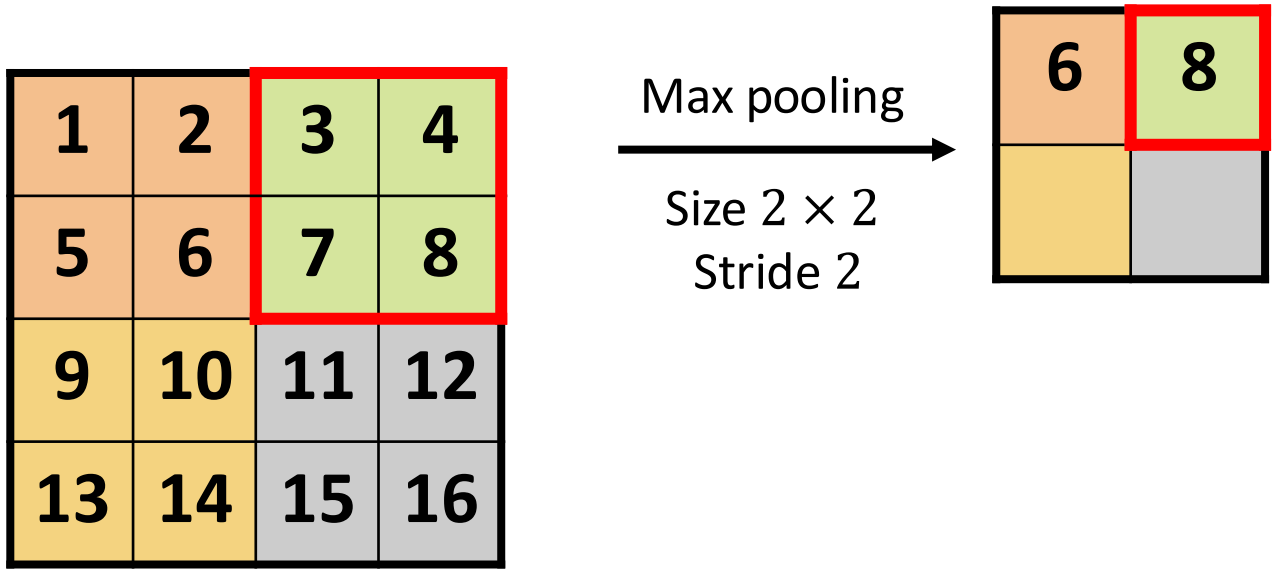
Pooling

How else can we downsample and preserve spatial invariance?



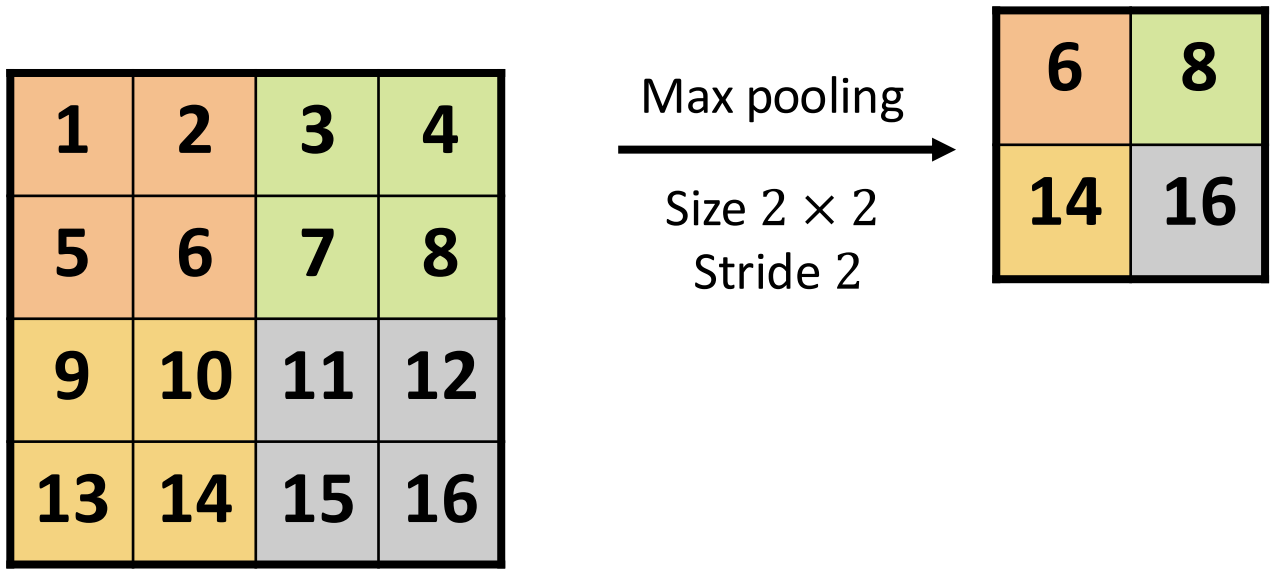
Pooling

How else can we downsample and preserve spatial invariance?



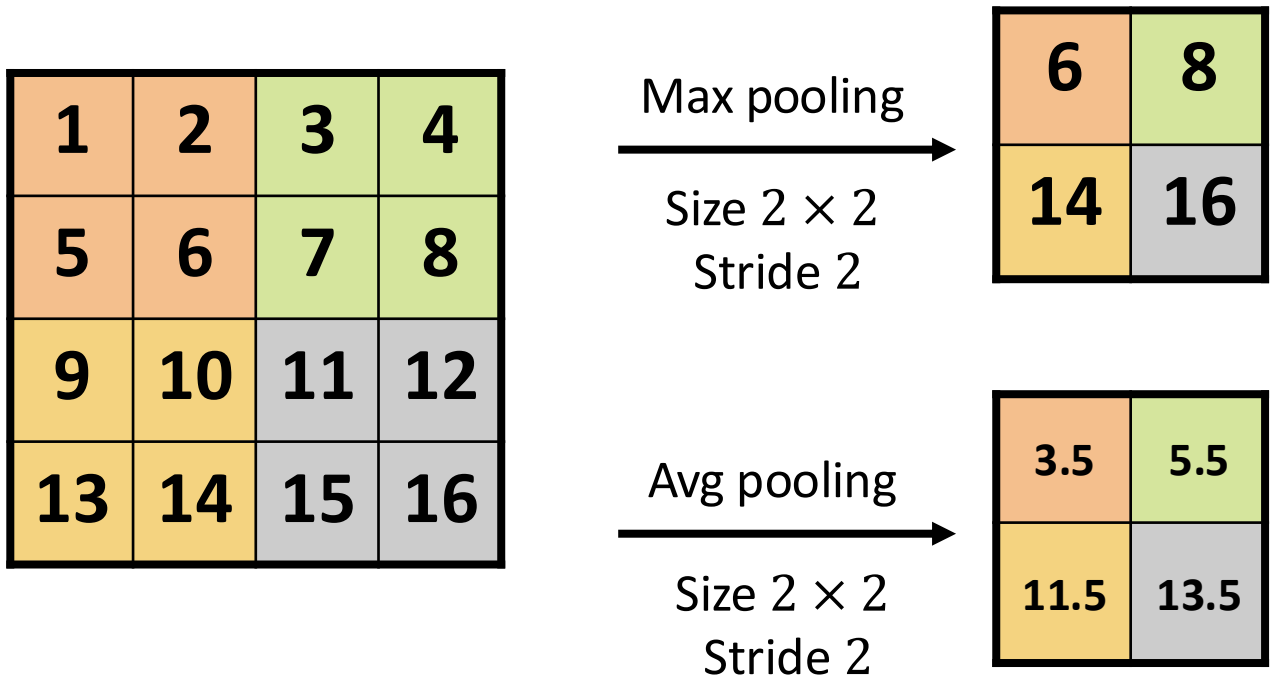
Pooling

How else can we downsample and preserve spatial invariance?



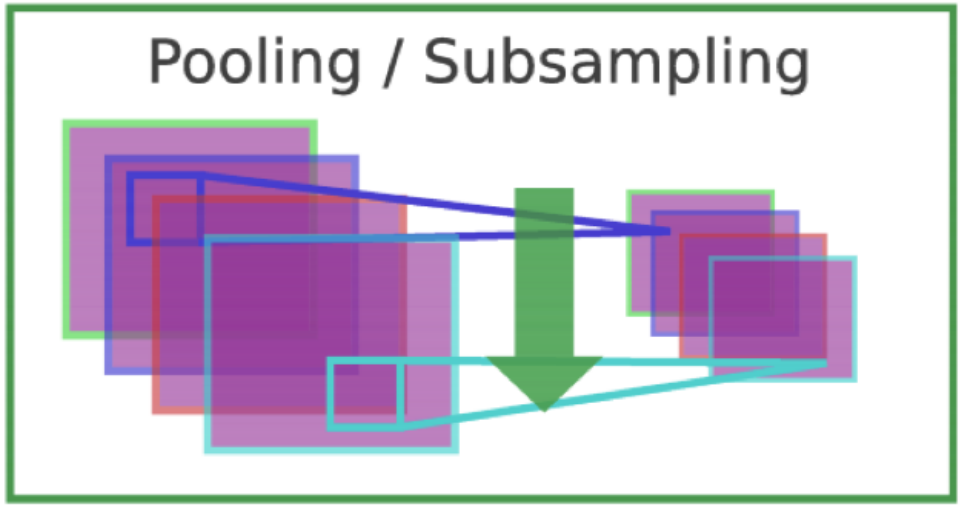
Pooling

How else can we downsample and preserve spatial invariance?

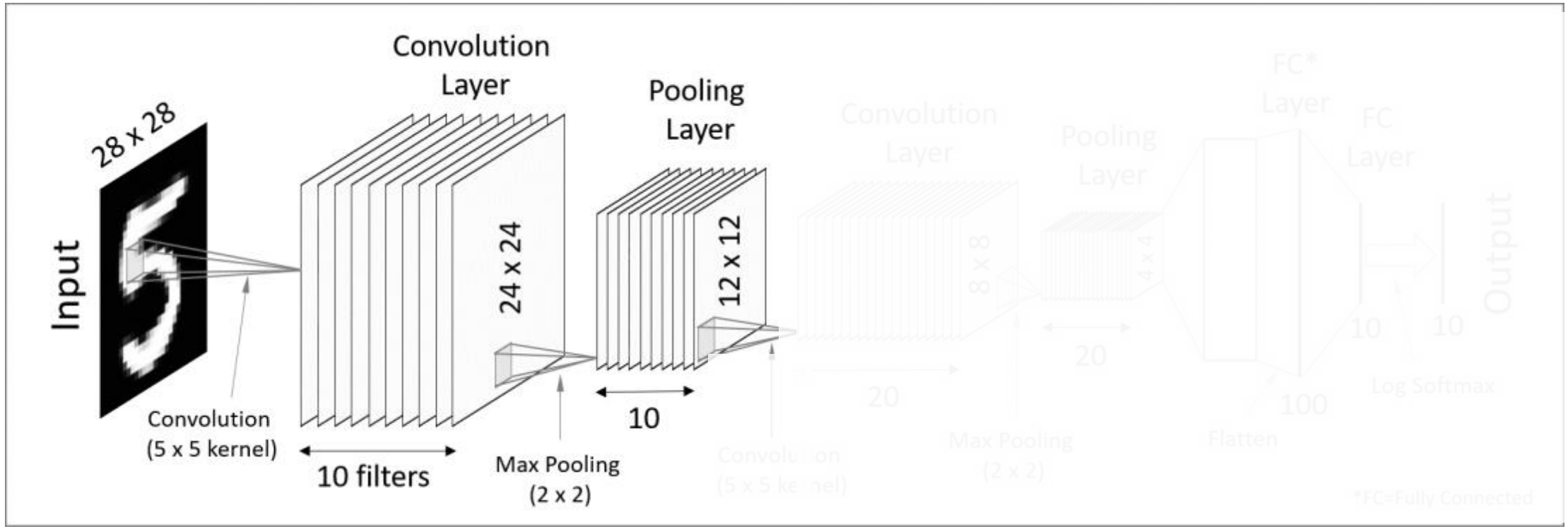


Stride in Pooling vs. in Convolution

- A pooling is very similar to a stride in the convolution layer
  - Max pooling generates additional non-linearity due to the max function
  - A stride in the convolution layer reduces computation
  - Their performance generally comparable
    - Striving for Simplicity: The All Convolutional Net (Springenberg et al, 2015)

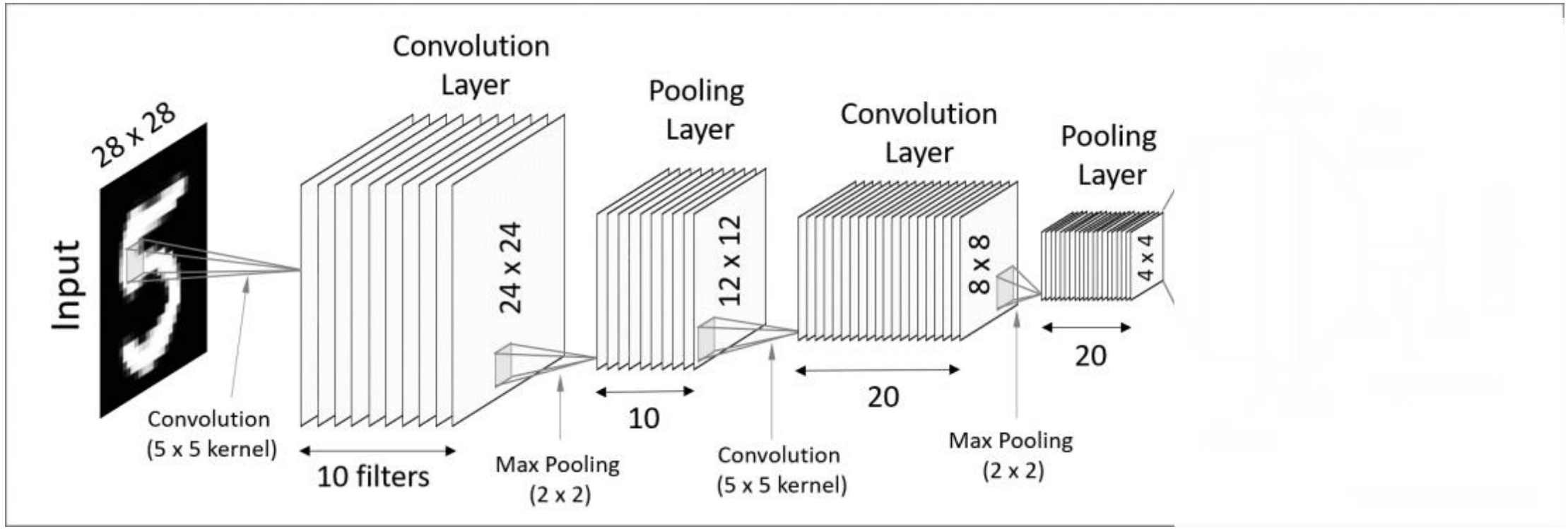


CNN: Feature Learning - Classification



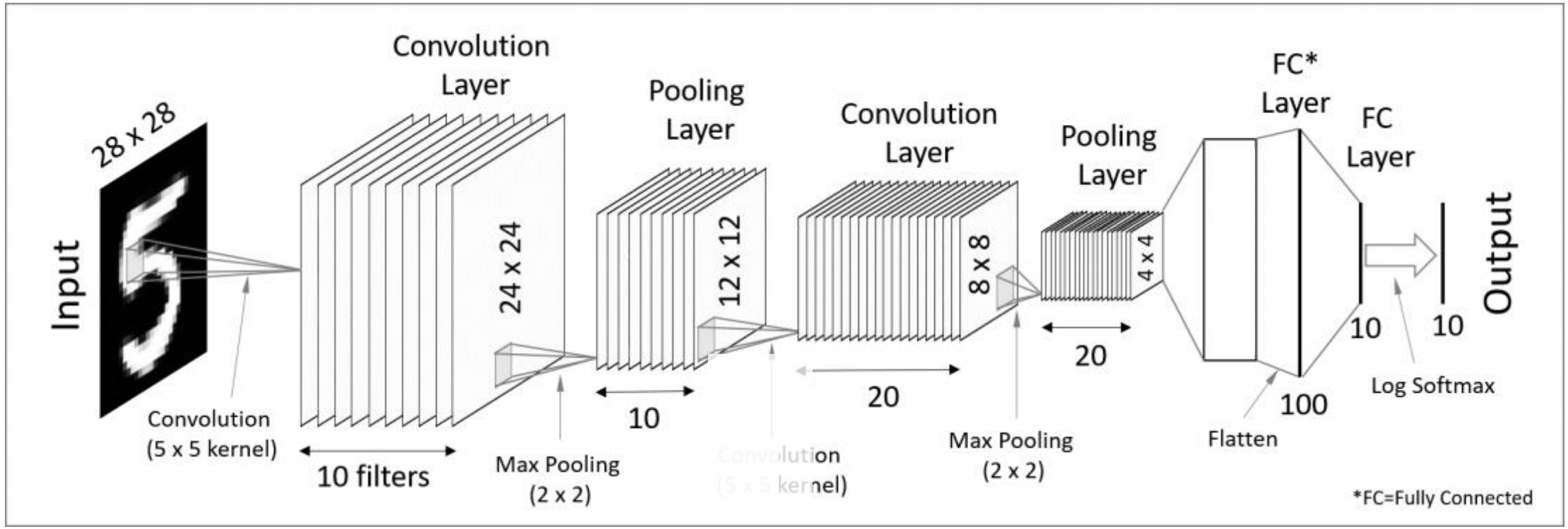
1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNN: Feature Learning - Classification



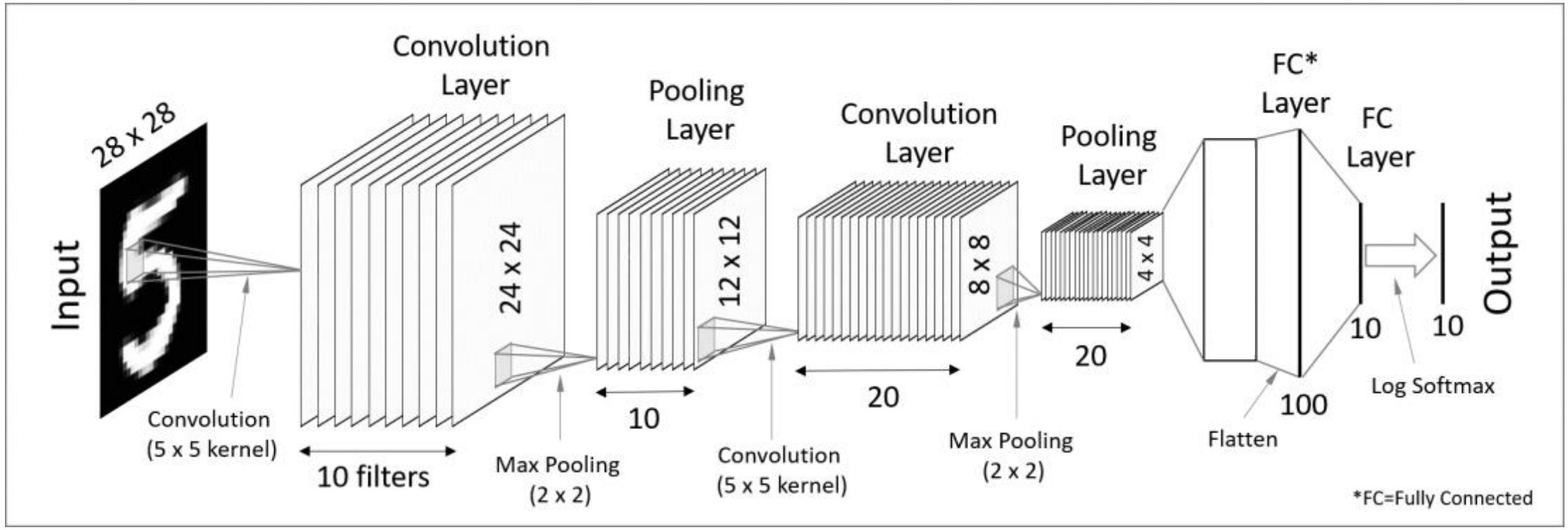
4. Stack one more pair of Convolution and Pooling Layer
   - The input of this convolution layer is 12 x 12 x 10
   - When we say 5 x 5 kernel in this layer, the actual filter size is 5 x 5 x 10, for each filter
5. Output 20 of 4x4 feature maps containing high-level features of data

CNN: Feature Learning - Classification



7. Flatten the 4 × 4 × 20 feature map to a 320 x 1 vector
8. Stack FC layers using these features (flattened) as an input for classifying images
9. Express output as probability of image belonging to a class using $\text{softmax}(s) = \frac{\exp(s_k)}{\sum \exp(s_i)}$

CNN: Feature Learning - Classification



28x28x1	24x24x10	12x12x10	8x8x20	4x4x20	320	100	10
Input Layer	Conv 1	MaxPool 1	Conv 2	MaxPool 2	Flatten	Dense	Output

# Weights:	(5x5+1)x10	(5x5x10+1)x20	(320+1)x100	(100+1)x100

Example: ImageNet Challenge

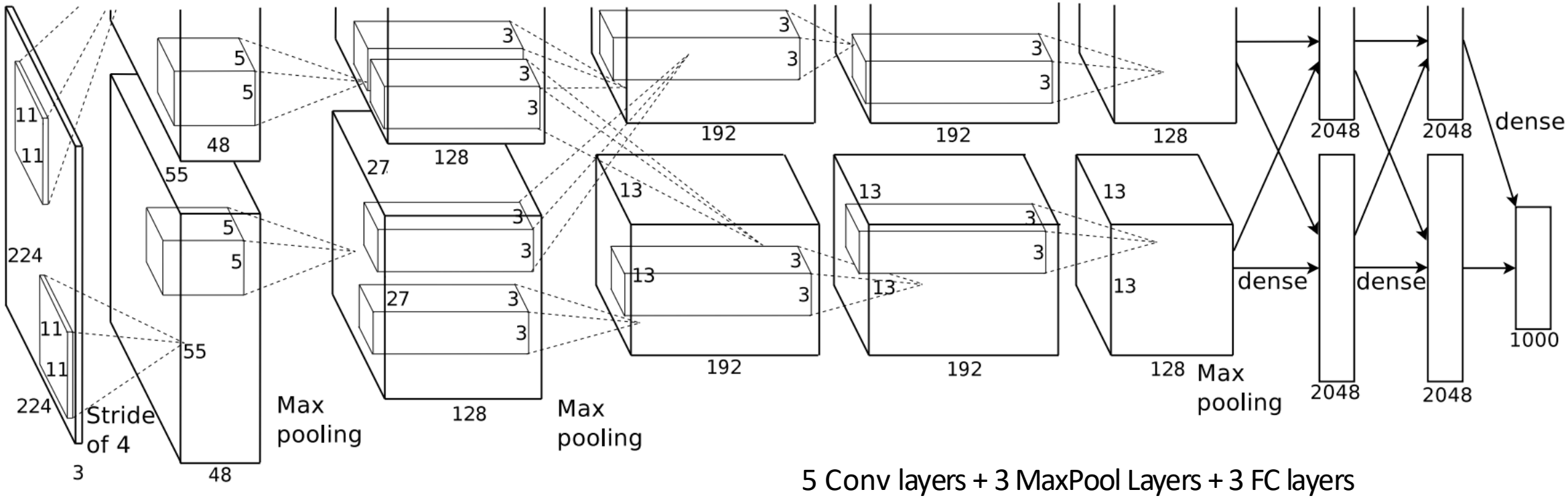
Dataset of over 14 million images across 21,841 categories, 300 G
https://gluon-cv.mxnet.io/build/examples_datasets/imagenet.html (2009)

ImageNet Challenge 2012 Winner:

AlexNet

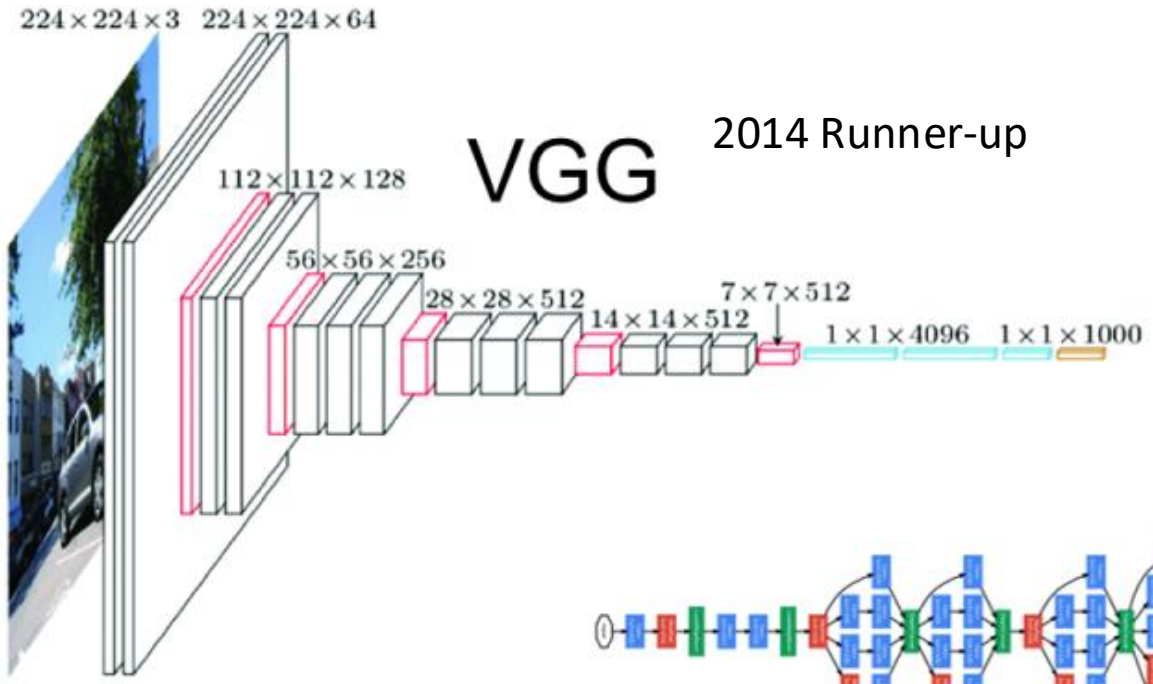
ImageNet Large Scale Visual Recognition Challenge

- https://image-net.org/challenges/LSVRC/, Winner 2012: Alex Krizhevsky, et al, 2012



5 Conv layers + 3 MaxPool Layers + 3 FC layers

ImageNet Challenge 2014 and 2015

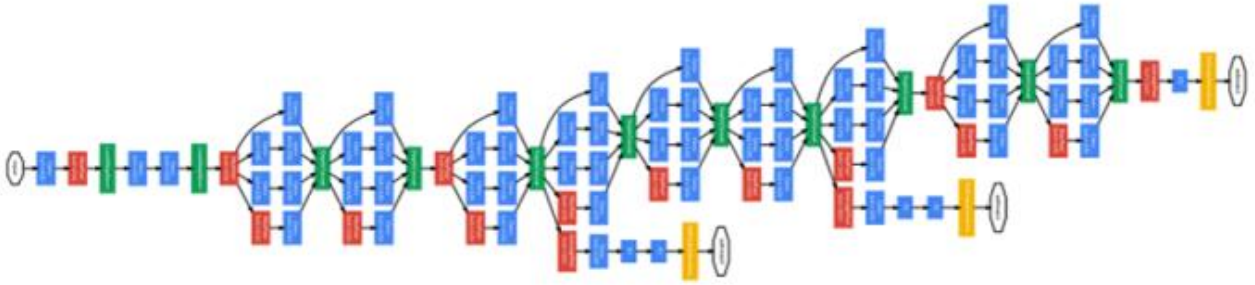


VGG

2014 Runner-up

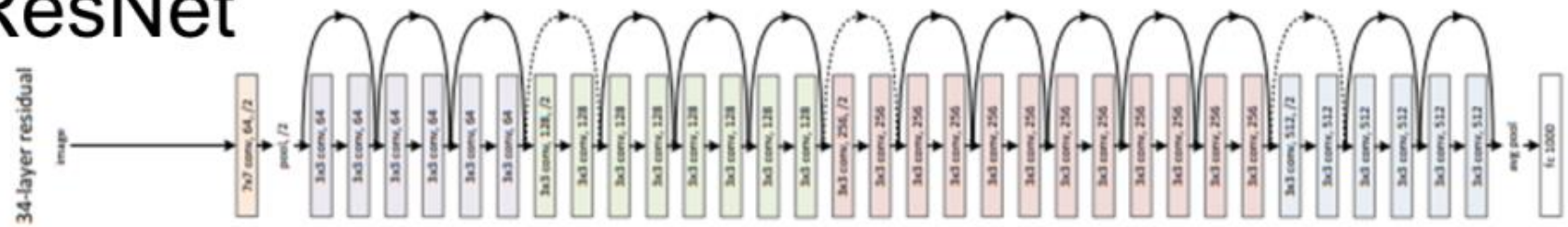
2014 Winner

GoogLeNet



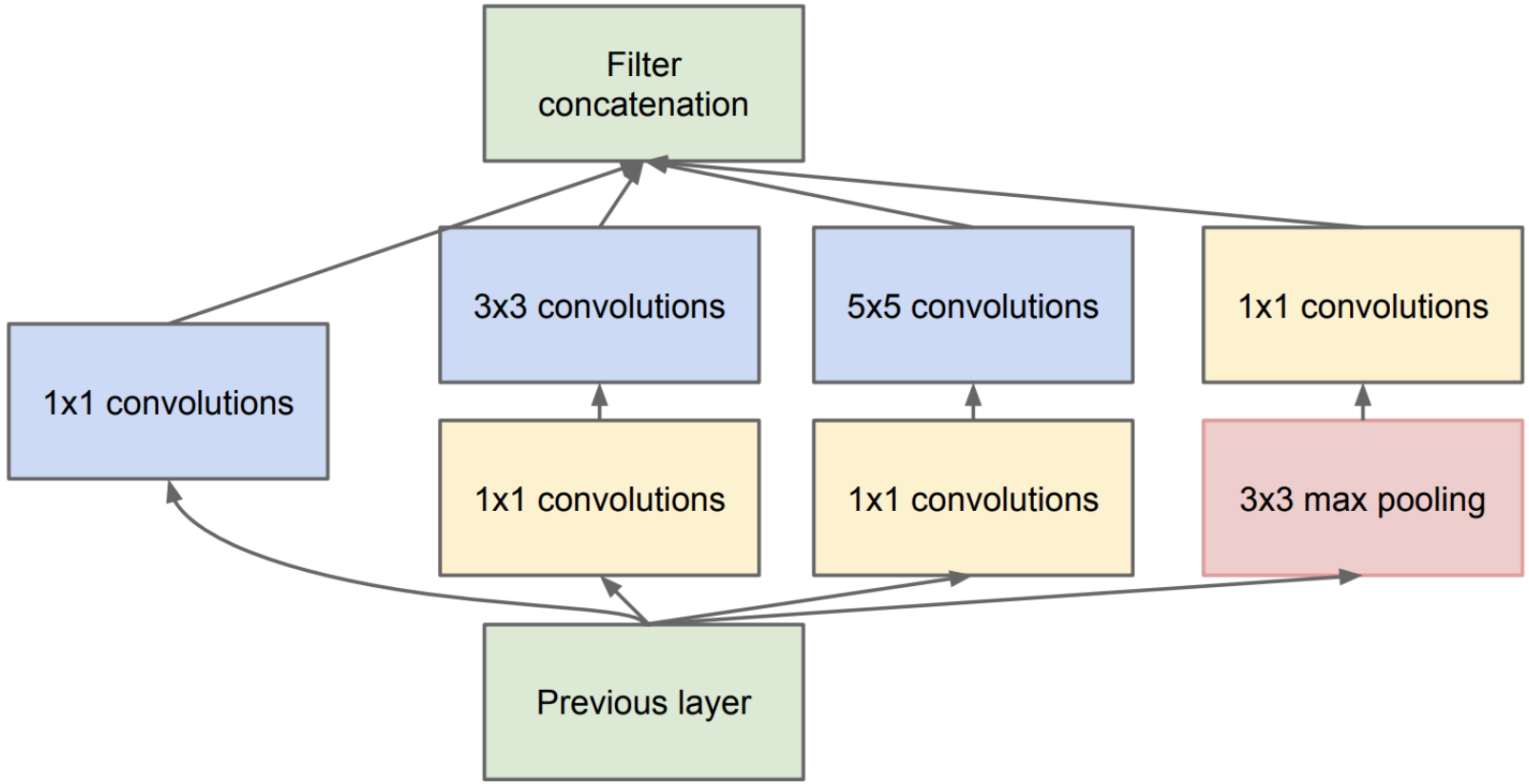
ResNet

2015 Winner

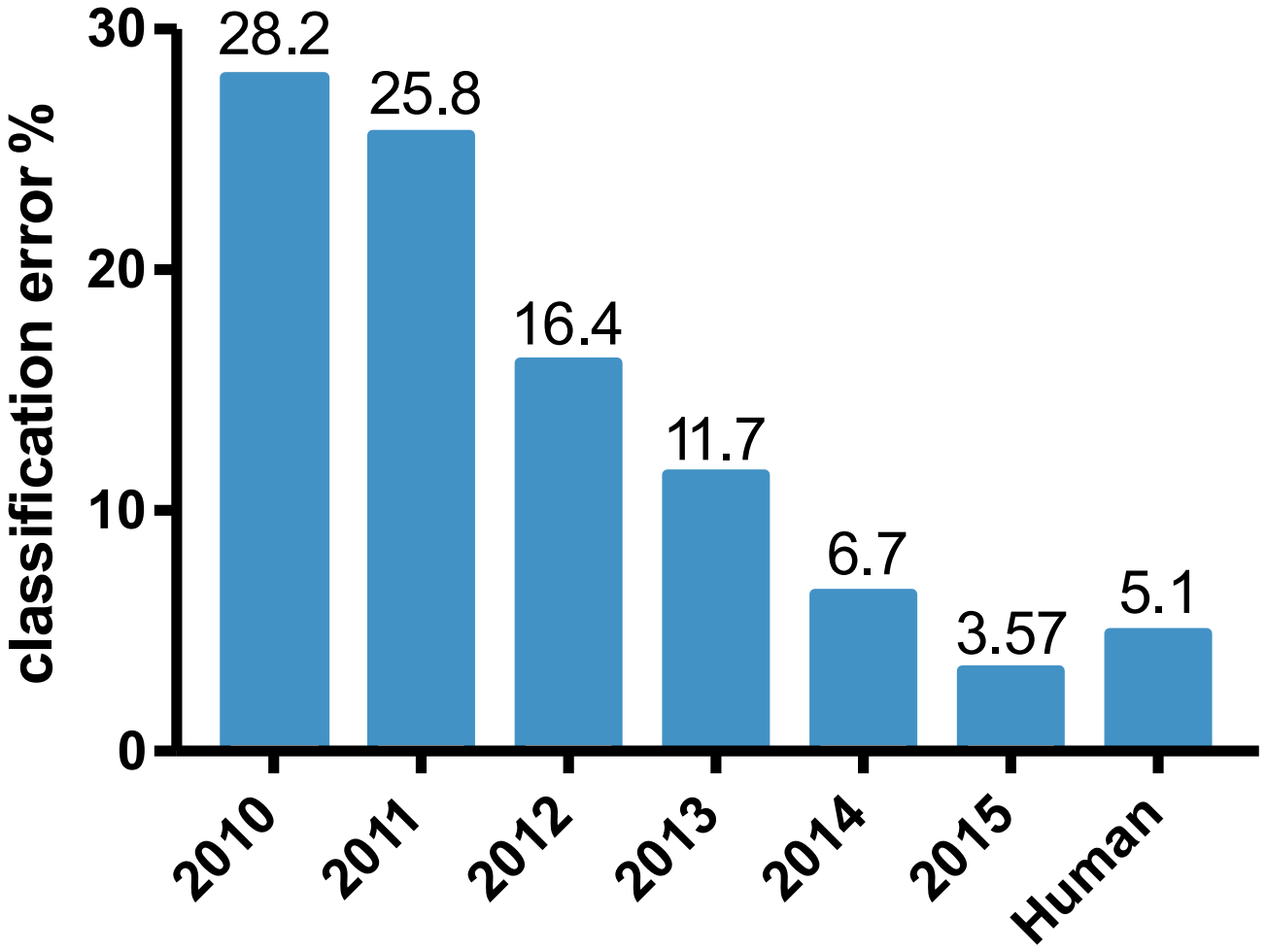


ImageNet Challenge 2014 Winner: GoogLeNet

Inception



ImageNet Challenge: ClassificationTask



2012: AlexNet. First CNN towin.

5 Conv layers + 3 FC layers

2013: ZFNet

8 layers with more filters

2014: VGG16

13 Conv layers + 3 FC layers

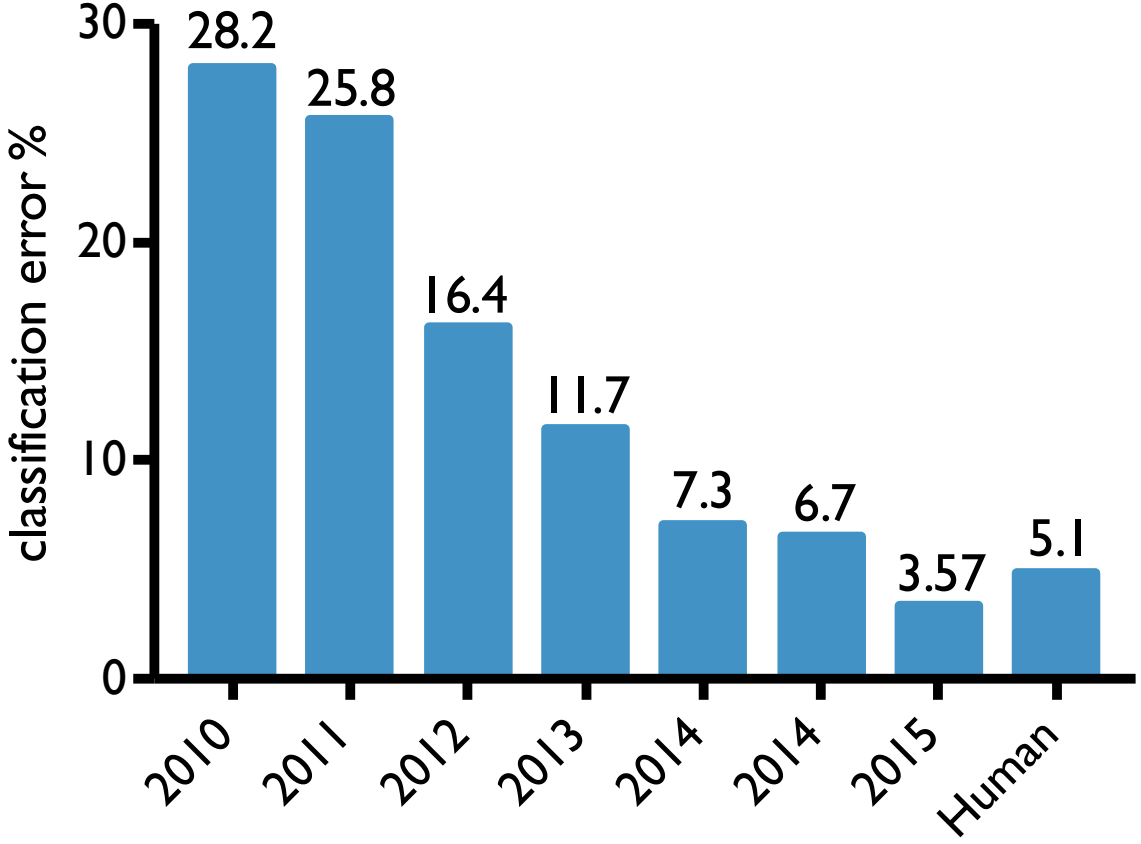
2014: GoogLeNet / Inception

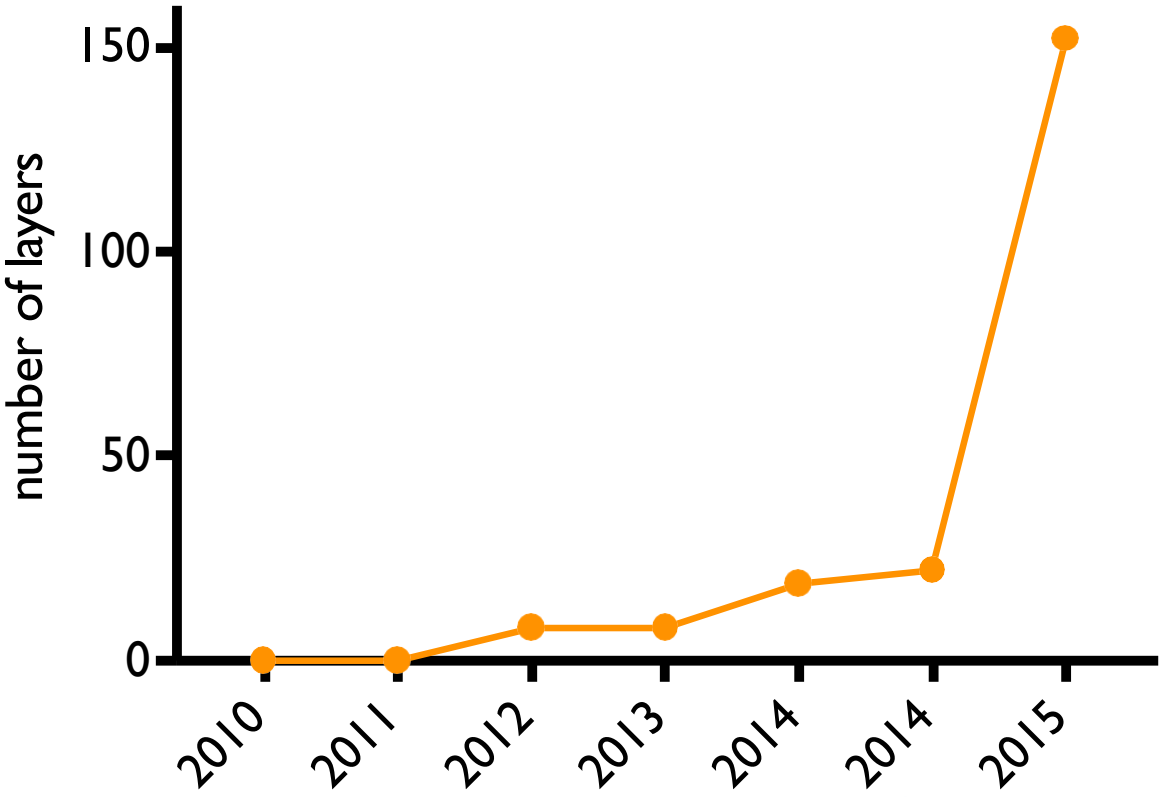
21 Conv layers + 1 FC layer

2015: ResNet

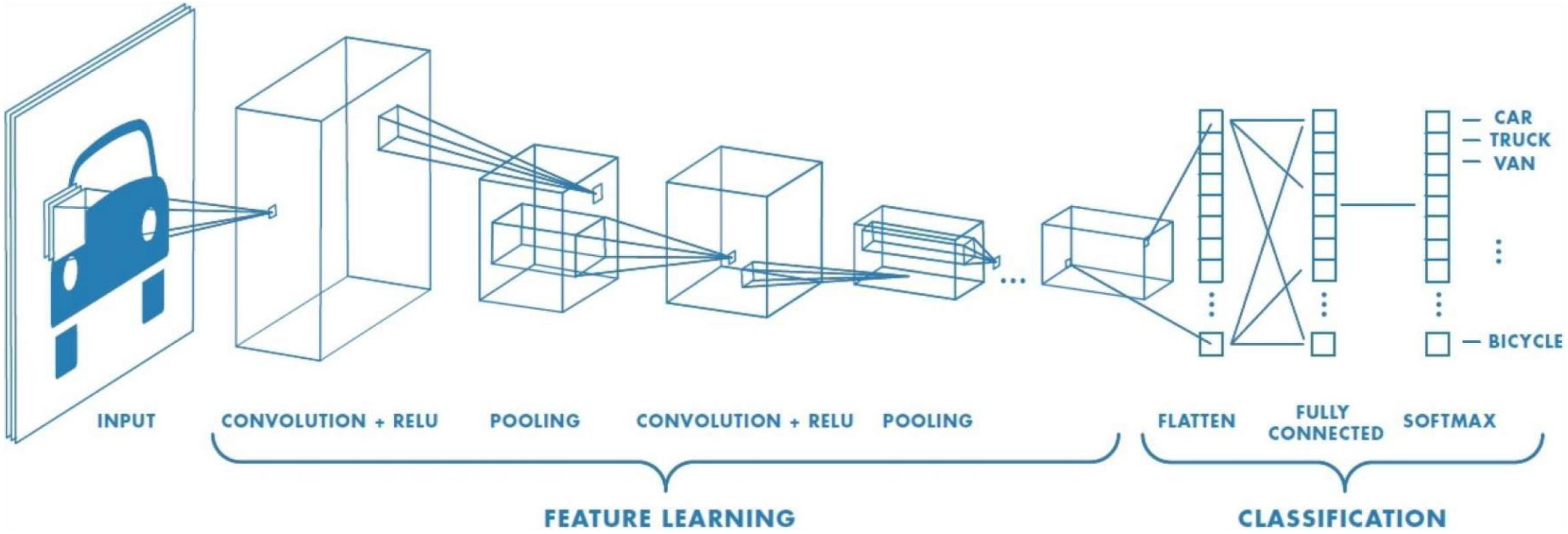
151 Conv layers + 1 FC layer

ImageNet Challenge: ClassificationTask

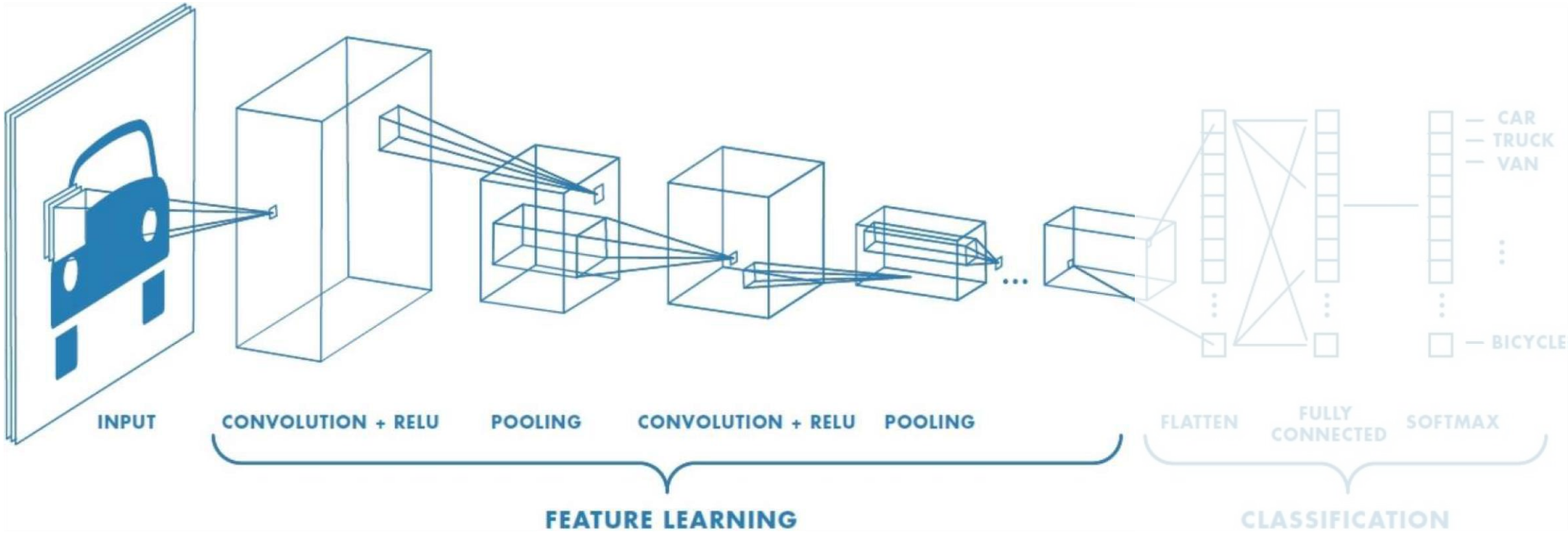




Feature Learning in Deep CNNs

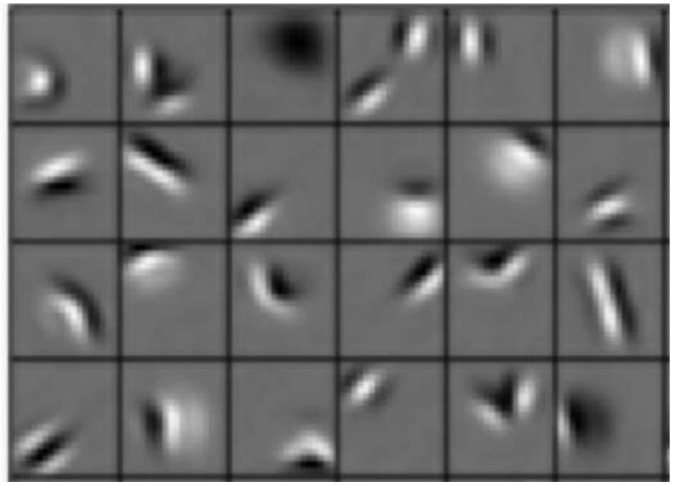


Feature Learning in Deep CNNs



Feature Learning in Deep CNNs

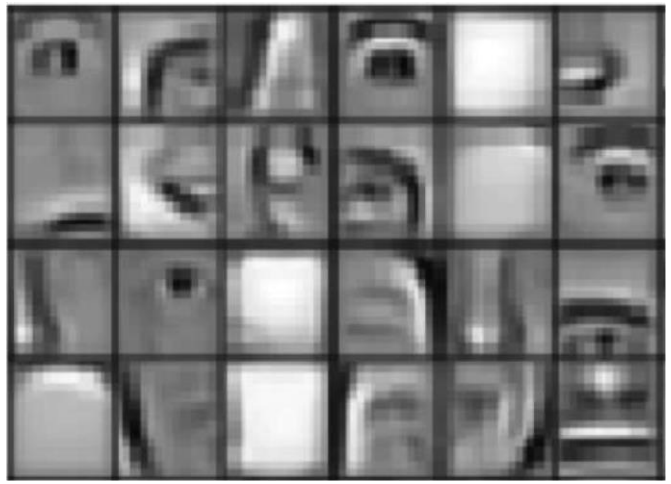
Low level features



Edges, dark spots

Conv Layer 1

Mid level features



Eyes, ears, nose

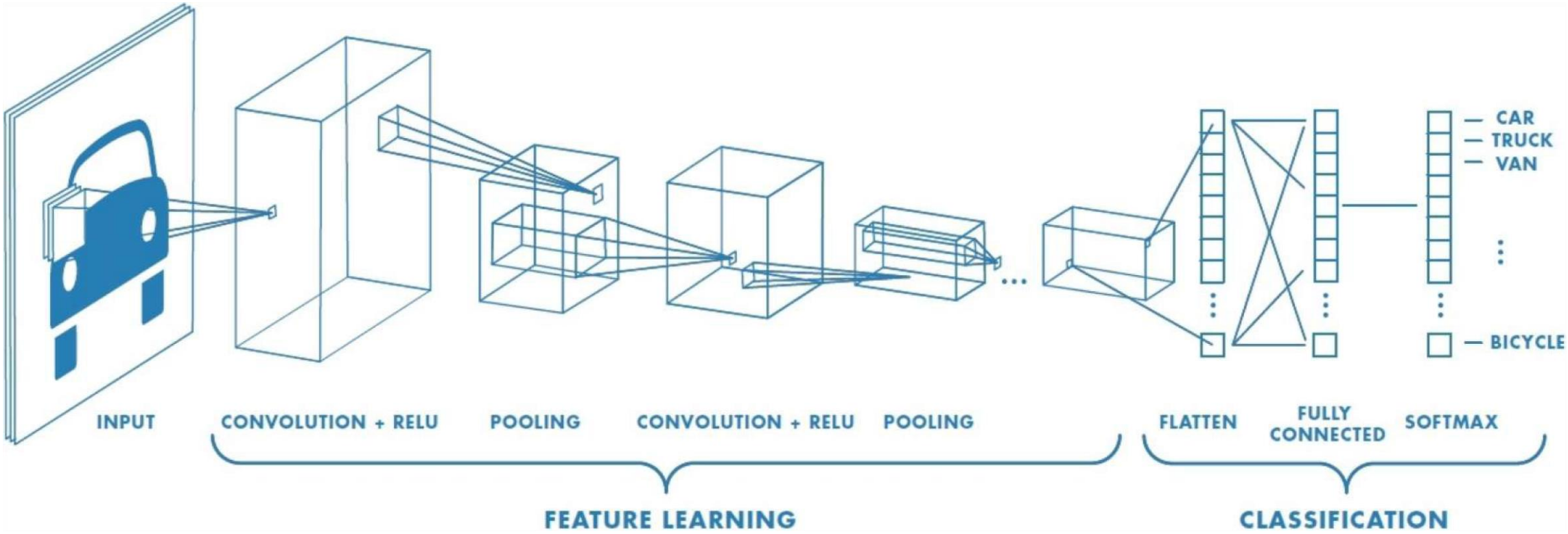
Conv Layer 2

High level features

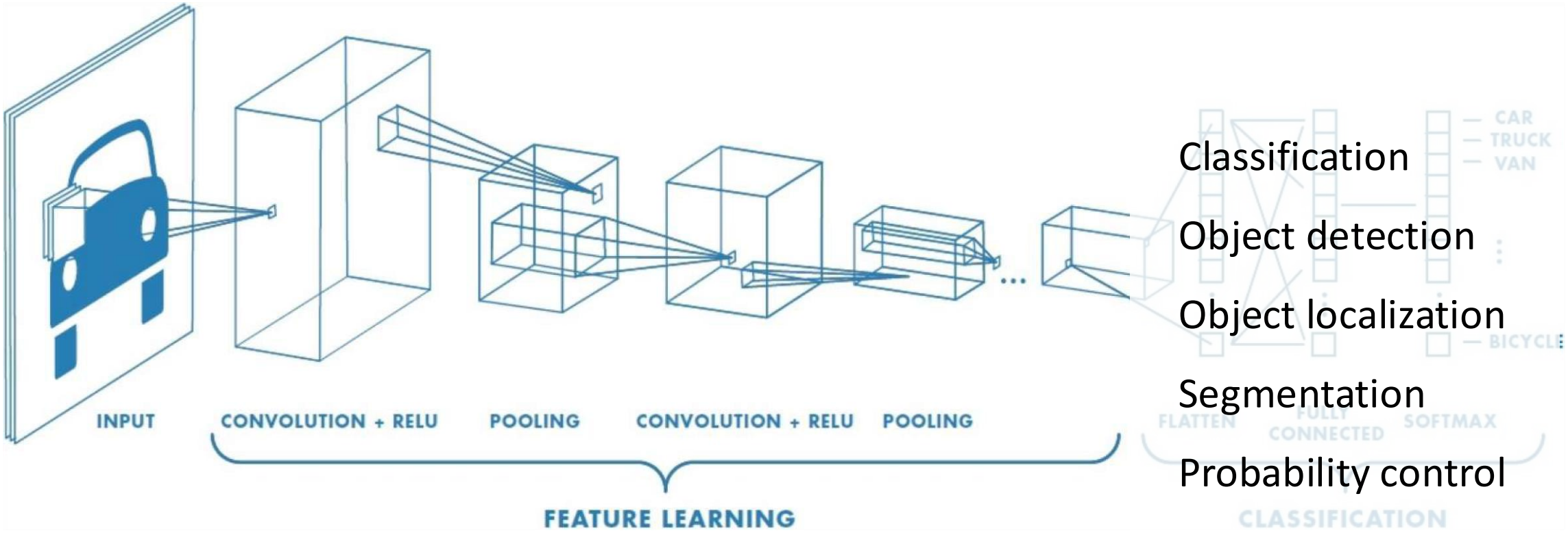
Facial structure

Conv Layer 3

An architecture of many applications



An architecture of many applications



Classification

Object detection

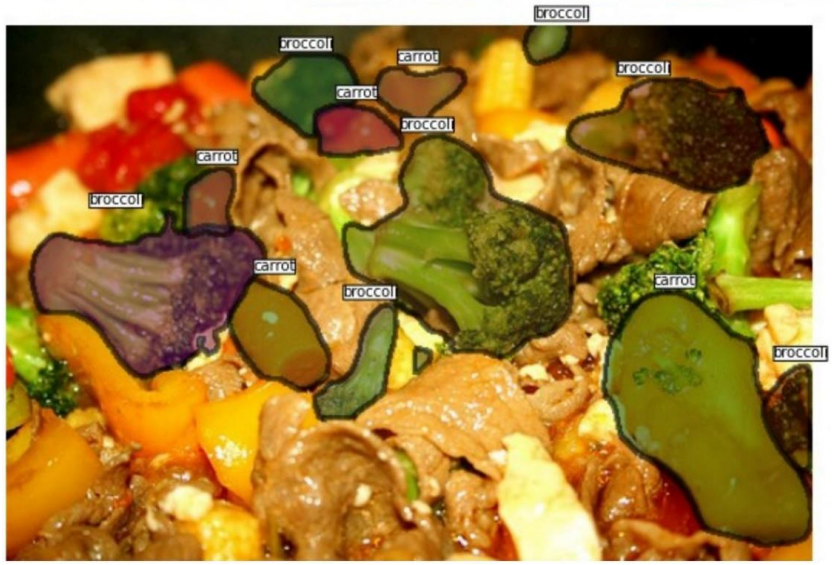
Object localization

Segmentation

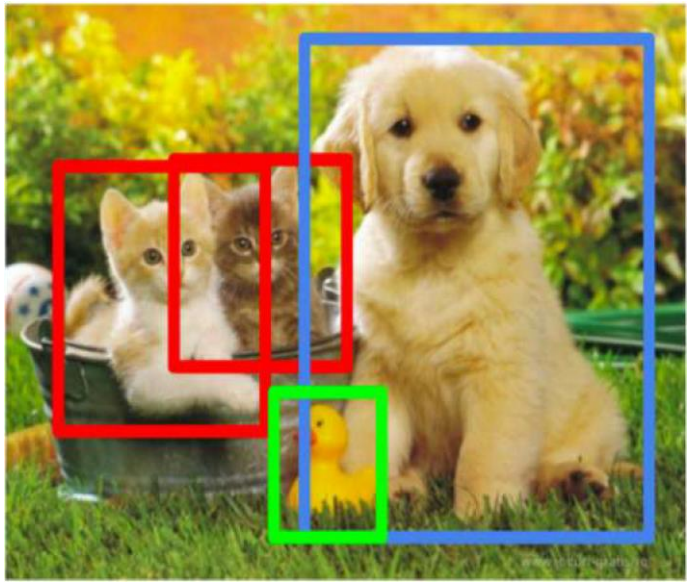
Probability control

CNN: Beyond Classification

Semantic Segmentation



Object Detection



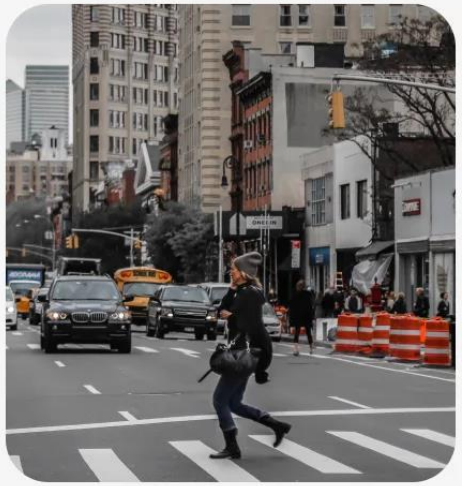
CAT, DOG, DUCK

Image Captioning

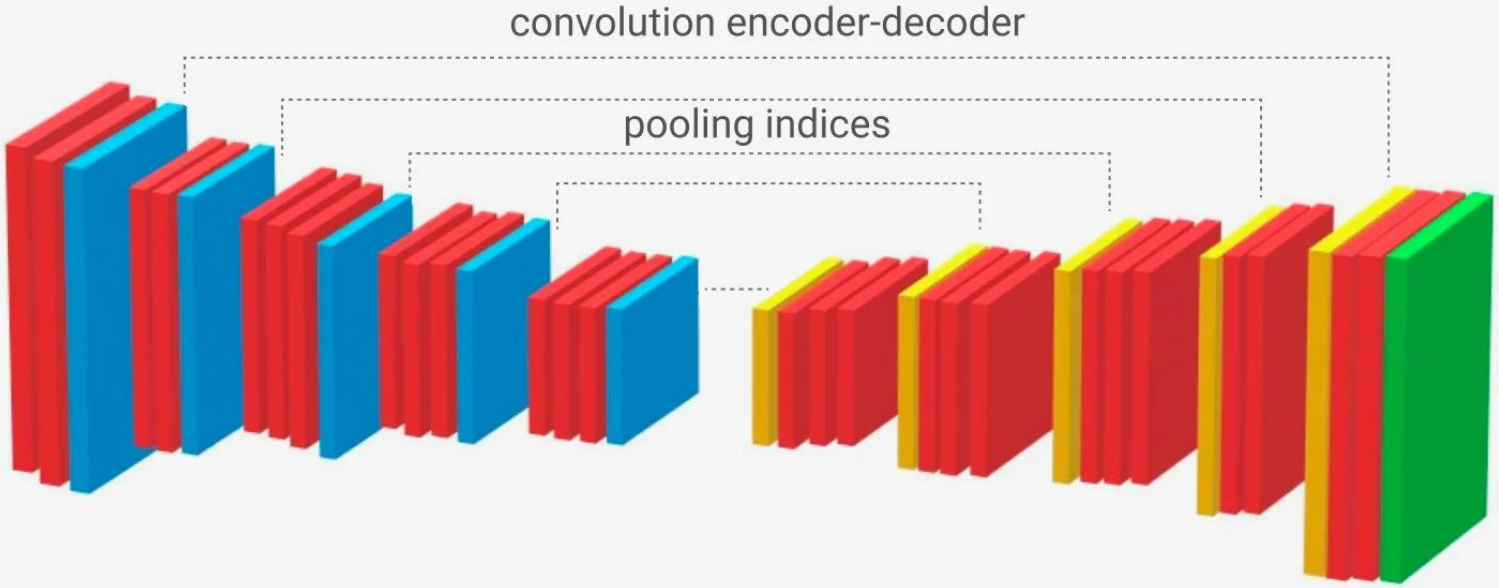
The cat is in the grass.

A CNN ARCHITECTURE FOR IMAGE SEGMENTATION.

Input



RGB Image



Output

Segmentation

Conv + Batch Normalisation + ReLU

Polling

Upsampling

Softmax

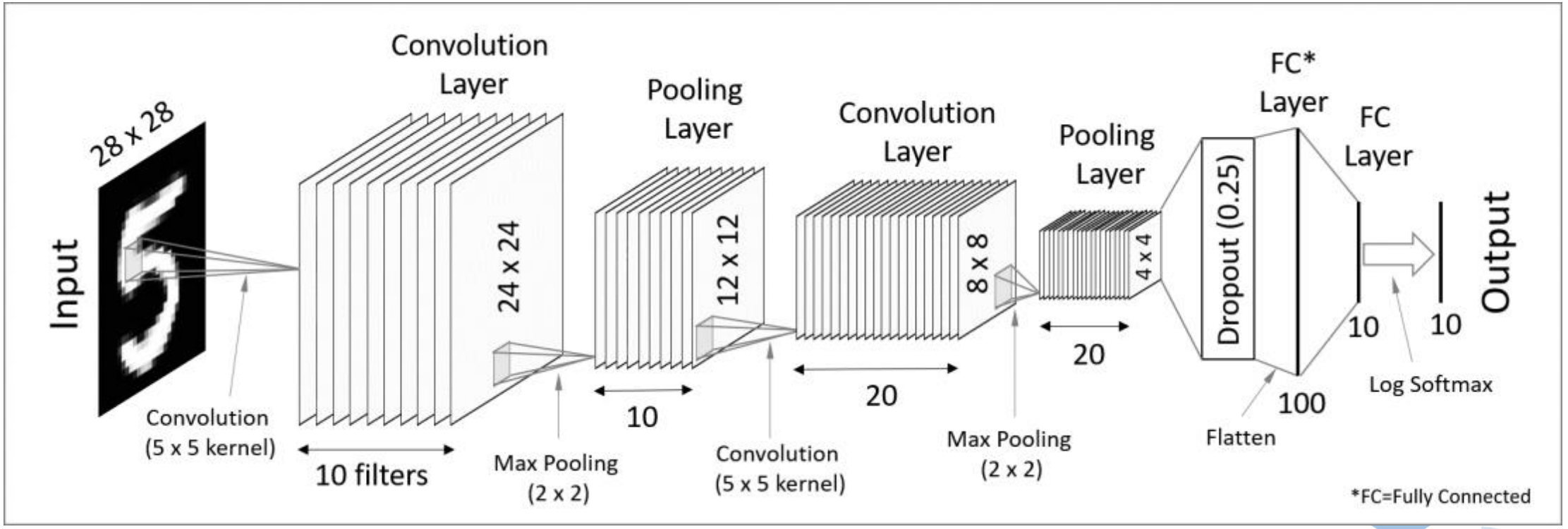
https://mindy-support.com/news-post/complete-guide-to-semantic-segmentation/

Typical Tricks in CNN

- Regularization - Dropout
- Data augmentation
- Pretraining
- Ensemble methods
- Batch Normalization Layer

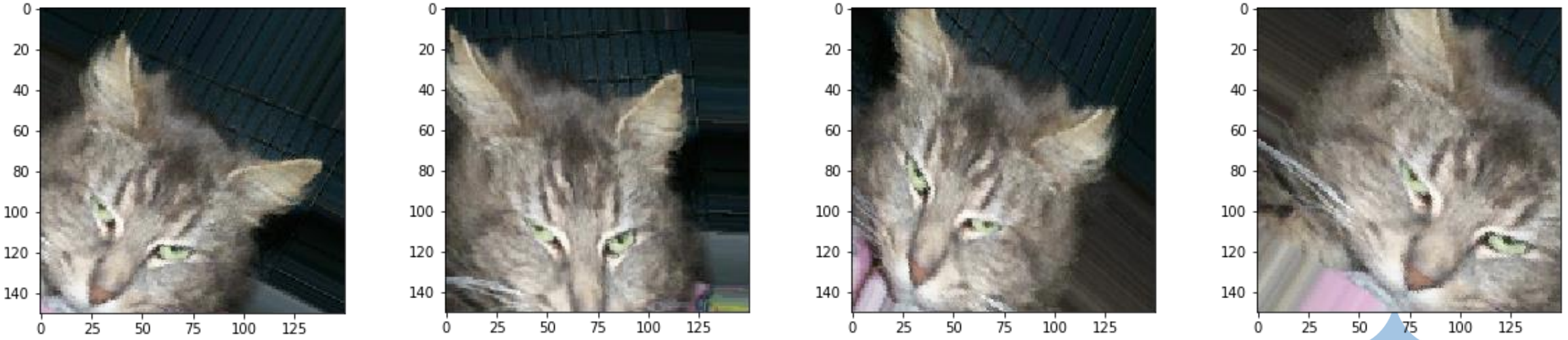
Dropout Regularization

- Use before the dense layer
  - model.add(layers.Dropout(0.25))



Data augmentation

- Generating more training samples from existing training data
  - E.g., flip, rotation, crop, shift, add random noise
- Gives you more data for free

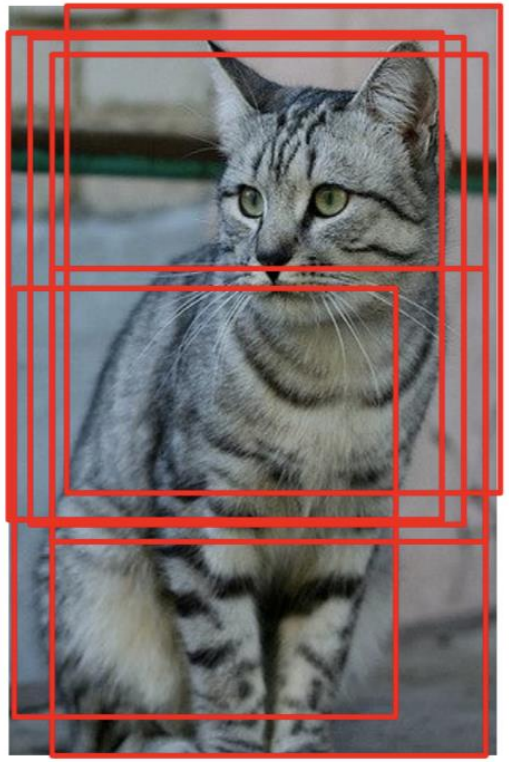


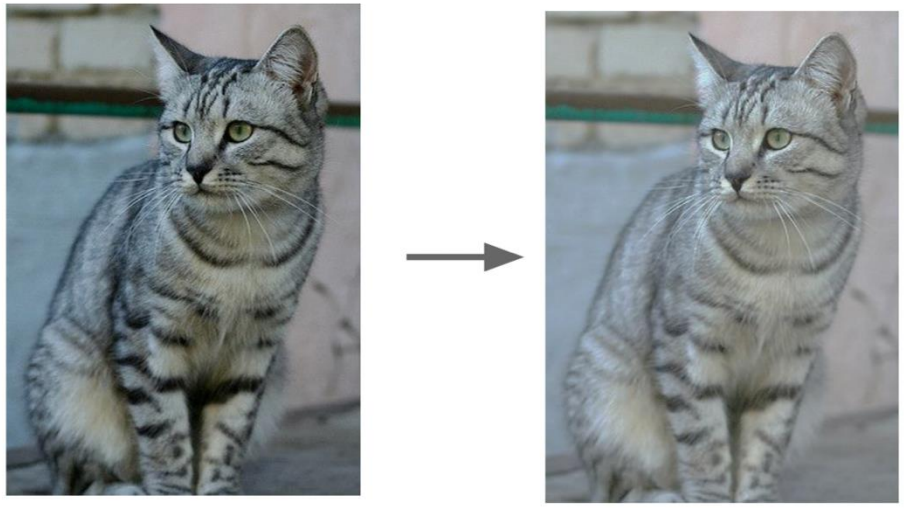
Data augmentation

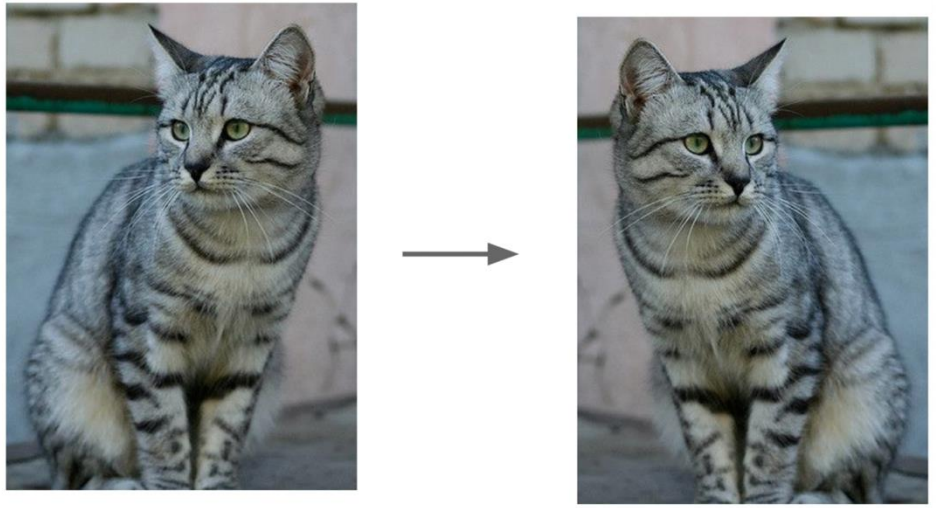
randomize contrast and brightness

Random crops and scales

Horizontal Flips







ImageDataGenerator in Keras

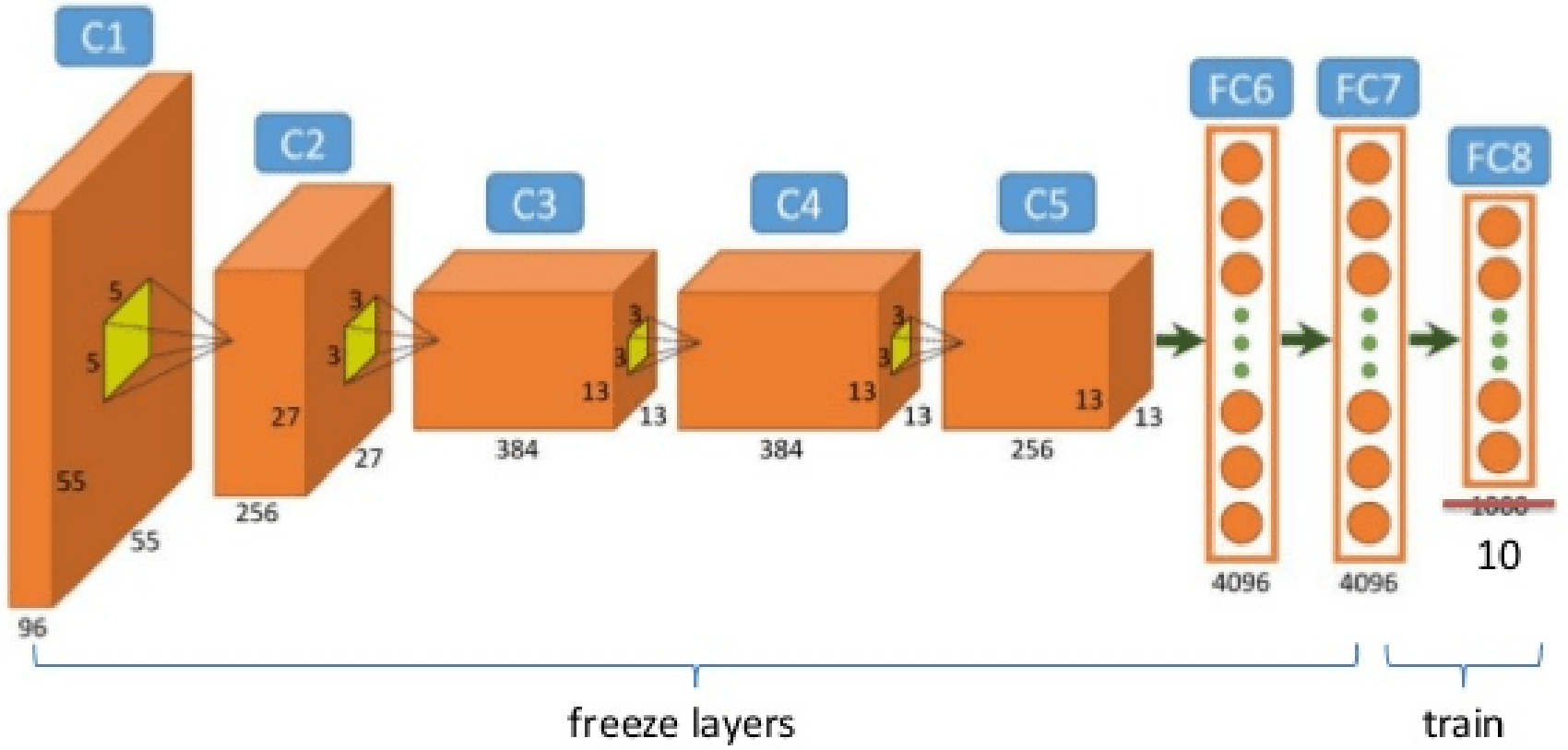
Pretraining

- Transfer learning with multiple tasks
  - Pre-train a deep network on task A (e.g. ImageNet, 1M images and 1K classes)
  - Remove the top layers (FC layers) and keep the base layers (Conv. Layers)
  - *Freeze* the base layers; *Train* the top layers for your new task
  - Your new task may be just classifying two classes, but with limited data

  - Optional: **Fine-tune** the top Conv Layers.

- Greedy layer-wise pretraining
  - develop deep NN whilst only ever training shallow NN
  - iteratively deepen a model

Pretraining



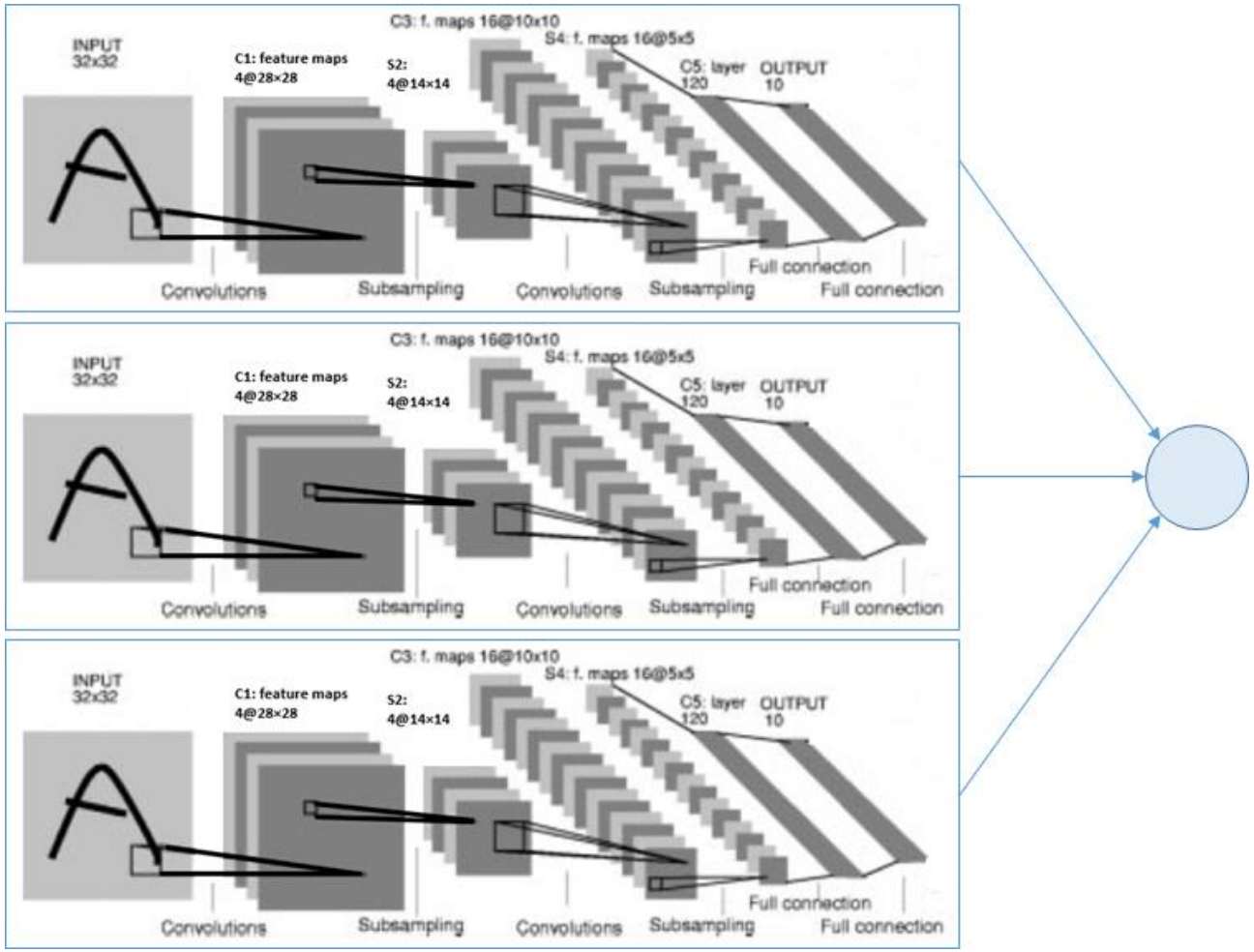
Standing on the shoulder of giants. https://keras.io/api/applications/

Ensemble methods

- Deep neural networks are unstable and random
  - Sensitive to hyper-parameters
  - Random initialization
  - Stochastic gradient

- Ensemble method reduces variance
  - Fit several networks with different network structures and learn with different random initializations and different optimization algorithms
  - Use bagging to ensemble the methods (majority voting)

Ensemble methods: Boosted LeNet-4



Batch Normalization Layer

- Feature standardization for the hidden layers to achieve faster convergence
- Step 1: find batch means and variances to standardize the output of the Convolution
- Step 2: apply a shifting and a scaling trainable parameter before feed into the Activation
- Add 2 non-trainable parameters: *mean*, *variance* + 2 trainable paras: *shifting* and *scaling*
- Split the Convolution layer: Put the BN layer after Convolution and before Activation.
  - model.add(layers.Conv2D(filters = 10, kernel_size = (5, 5), input_shape = (28, 28, 1)))
  - *model.add(layers.BatchNormalization())*
  - model.add(layers.Activation('relu'))
  - model.add(layers.MaxPooling2D((2, 2)))

Questions?