Convolutional Neural Networks in TensorFlow

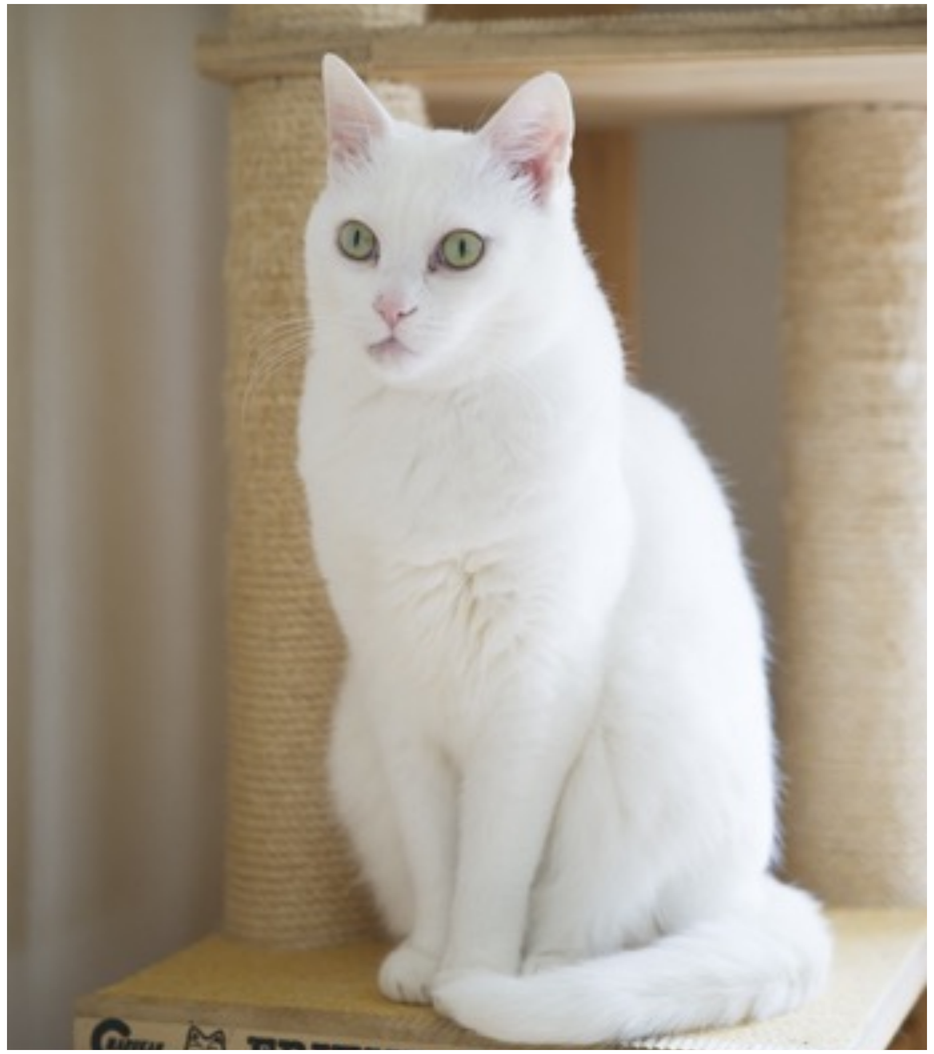
Overview

Convolutional NNs are one kind of NN architecture which work well with 2D data

Modeled on the visual cortex, they are amazing at image classification

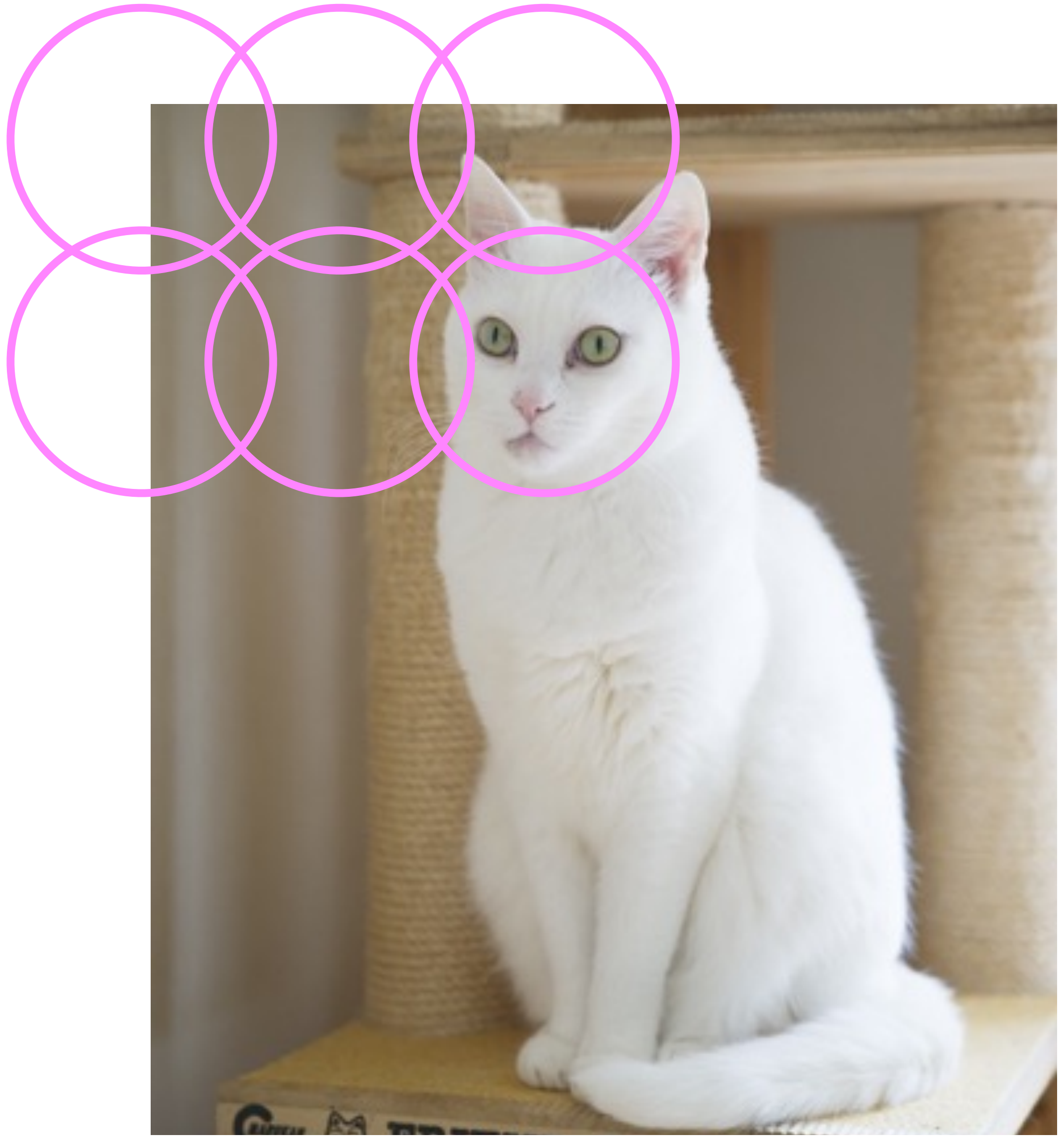
How Do We See?

Viewing an Image



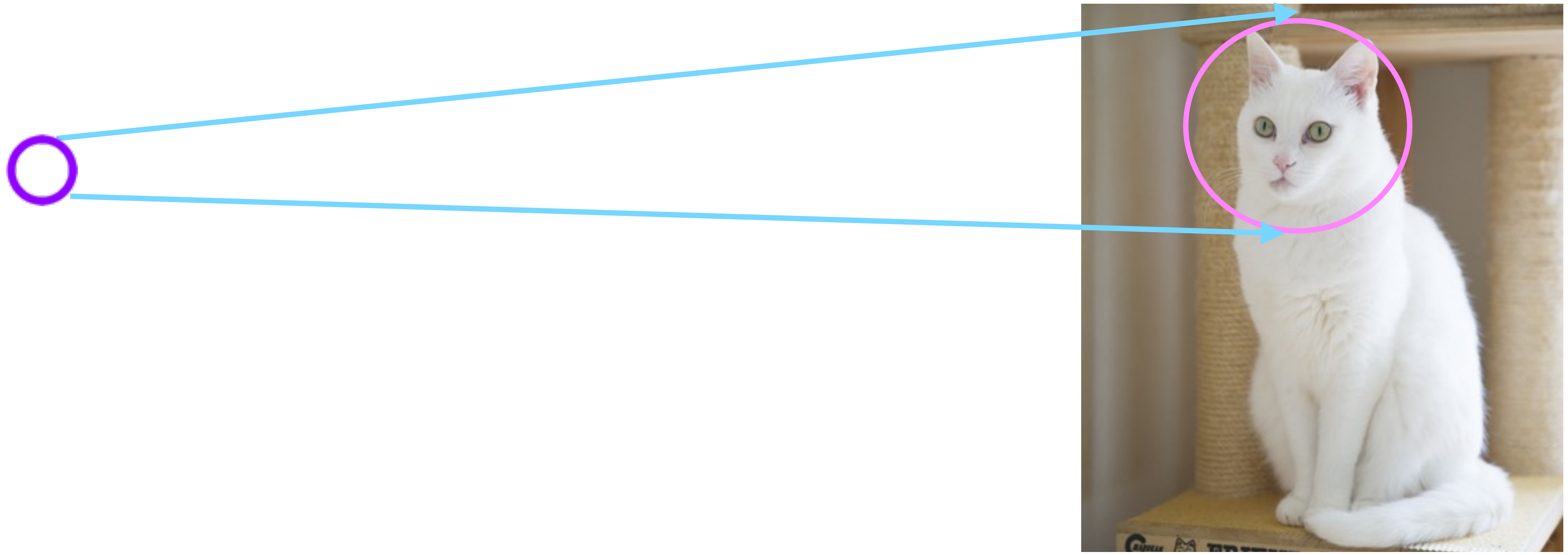
All neurons in the eye don't see the entire image

Viewing an Image



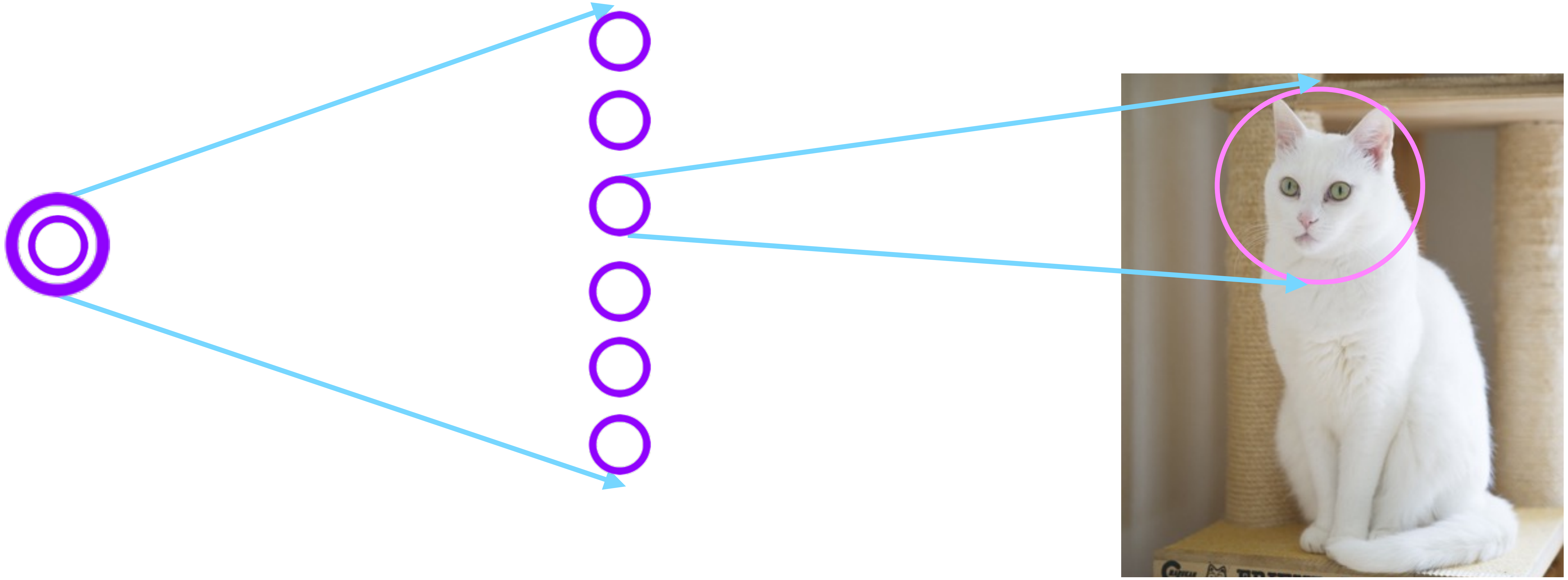
Each neuron has its own local receptive field

Viewing an Image



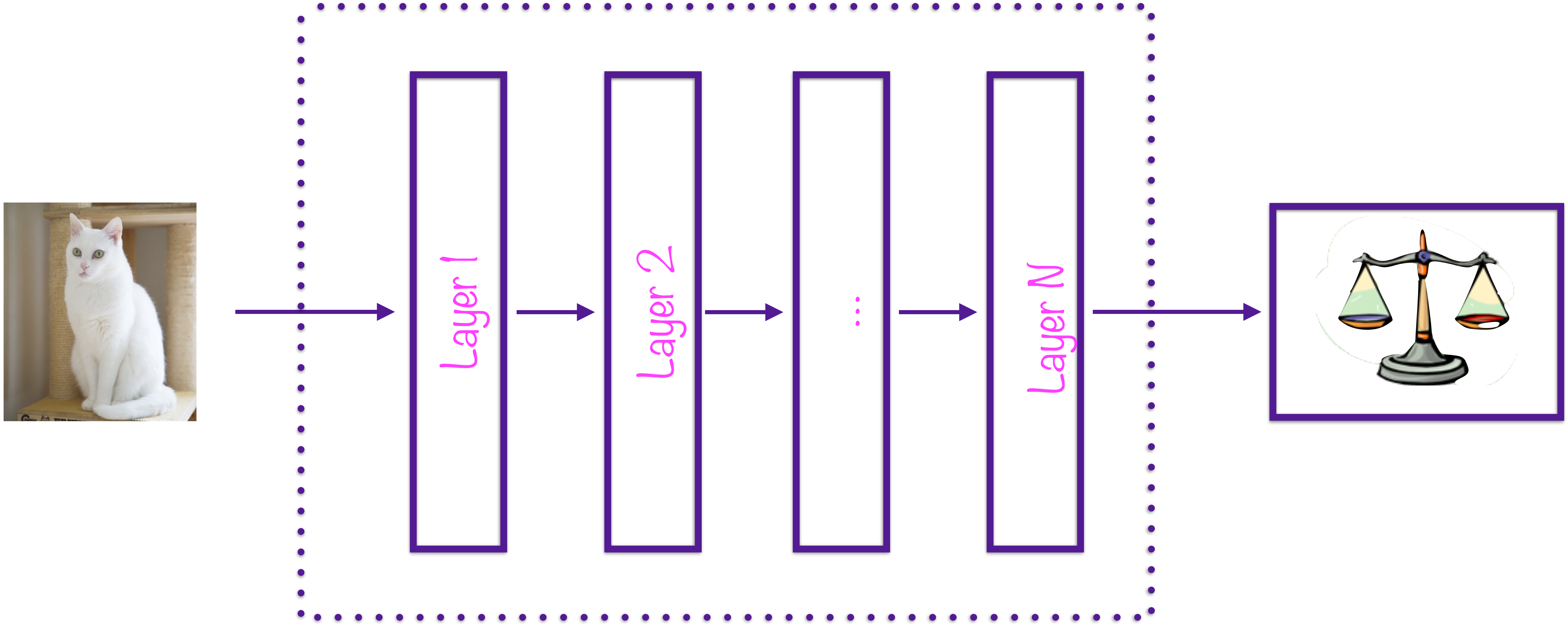
It reacts only to visual stimuli located in its receptive field

Viewing an Image



Some neurons react to more complex patterns that are combinations of lower level patterns

Neural Networks



Sounds like a classic neural network problem

Two Kinds of Layers in CNNs

Convolution

Local receptive field

Pooling

Subsampling of inputs

Convolution

Two Kinds of Layers in CNNs

Convolution

Local receptive field

Pooling

Subsampling of inputs

Convolution

In this context, a sliding window function applied to a matrix

Convolution

In this context, a sliding window function applied to

a matrix

e.g. a matrix of pixels representing

an image

Convolution

In this context, a sliding window function applied to a matrix

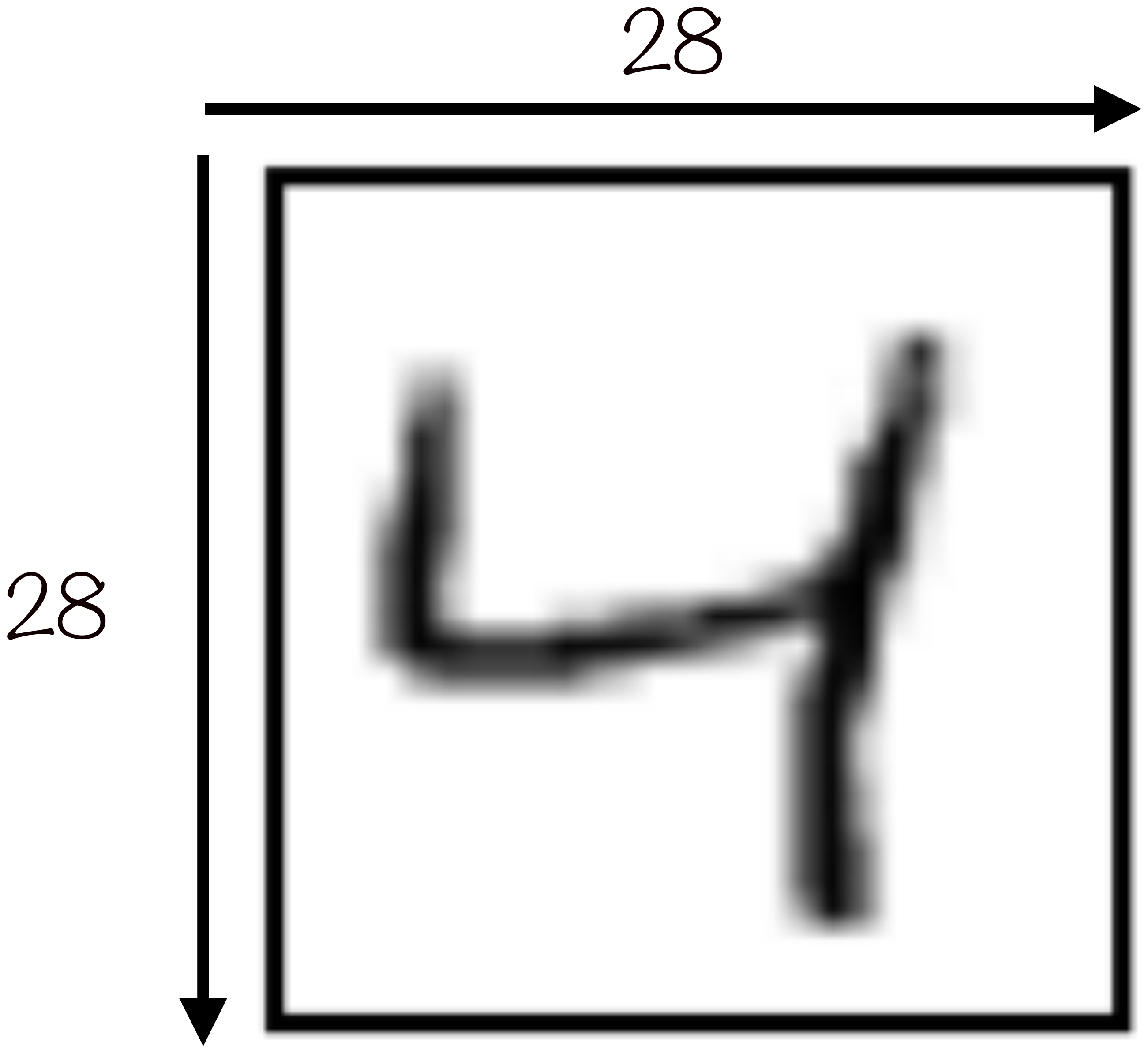
Often called a kernel or filter

Convolution

In this context, a sliding window function applied to a matrix

Kernel is applied element-wise in sliding-window fashion

Representing Images as Matrices



= 784 pixels

Representing Images as Matrices

6

6

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

= 36 pixels

Representing Images

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

Matrix

3

3

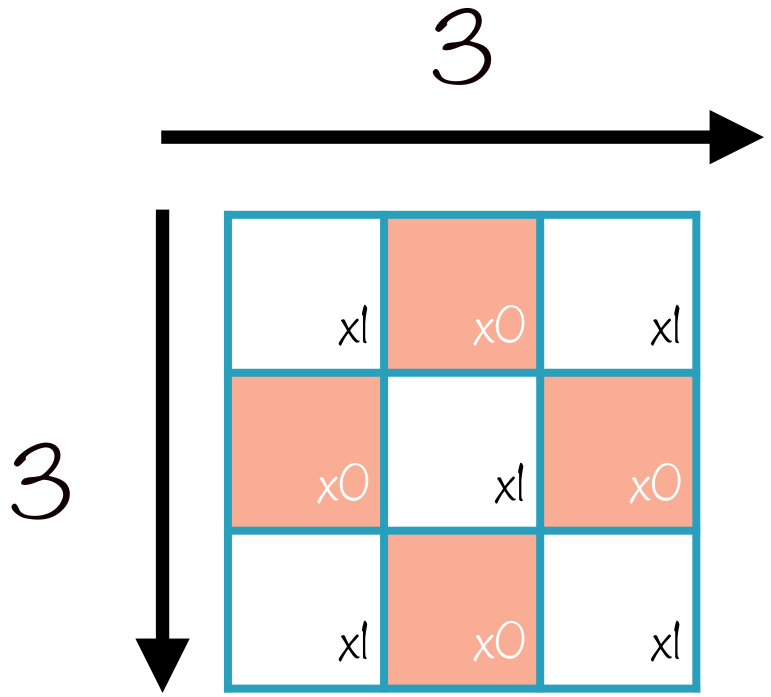
1	0	1
0	1	0
1	0	1

Kernel

Convolution

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

Matrix



Kernel

Convolution

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

Matrix

x1	x0	x1
x0	x1	x0
x1	x0	x1

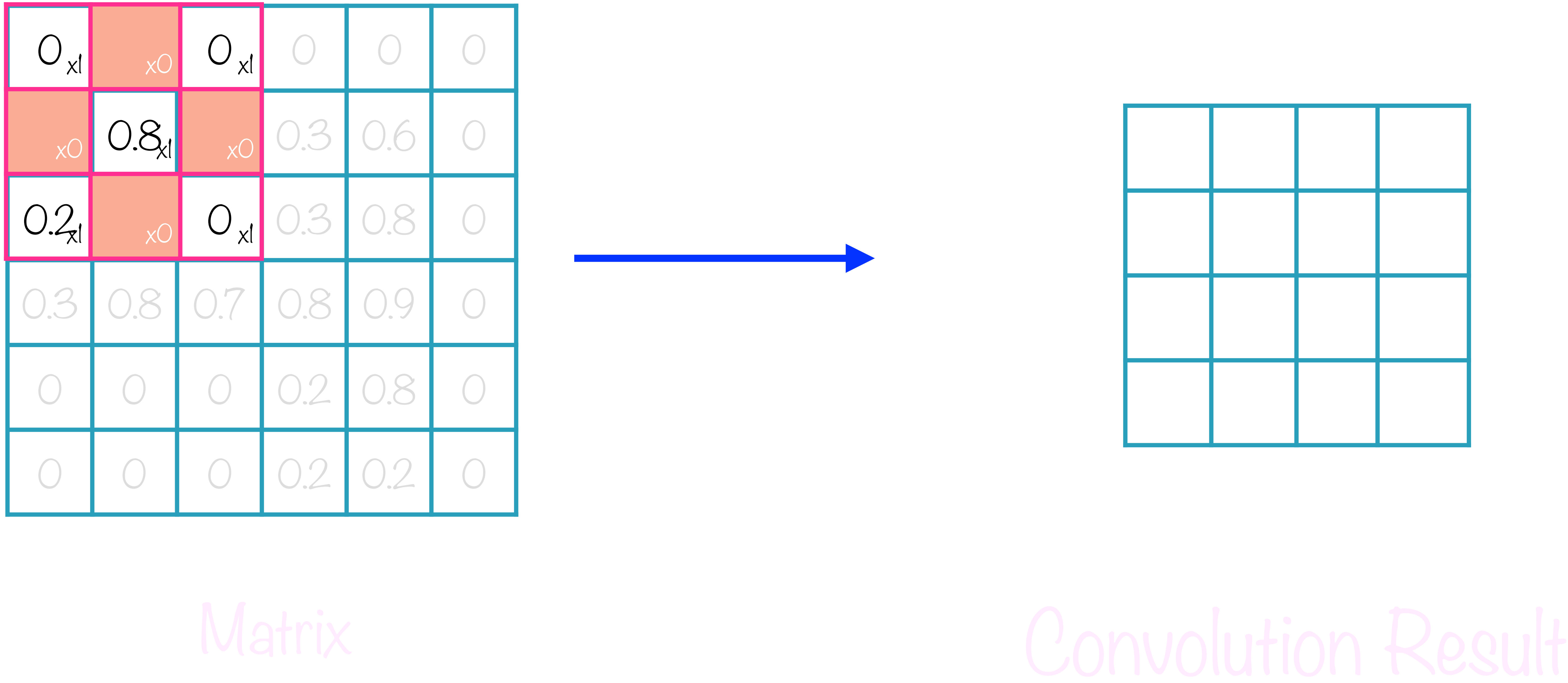
4

4

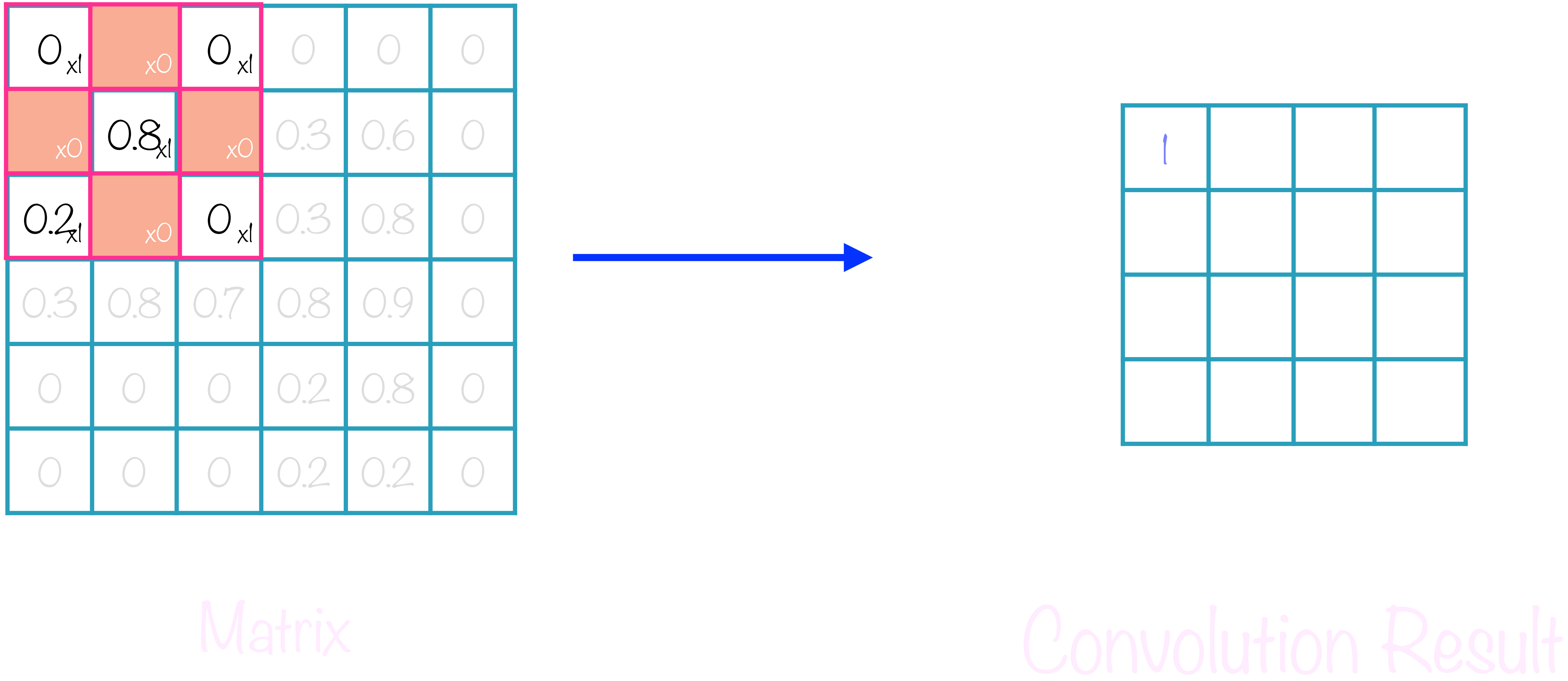
1	1.2	1.1	0.9
1.9	2.7	2.5	1.9
1.0	2.1	2.4	1.4
1.0	1.8	2.0	1.8

Convolution Result

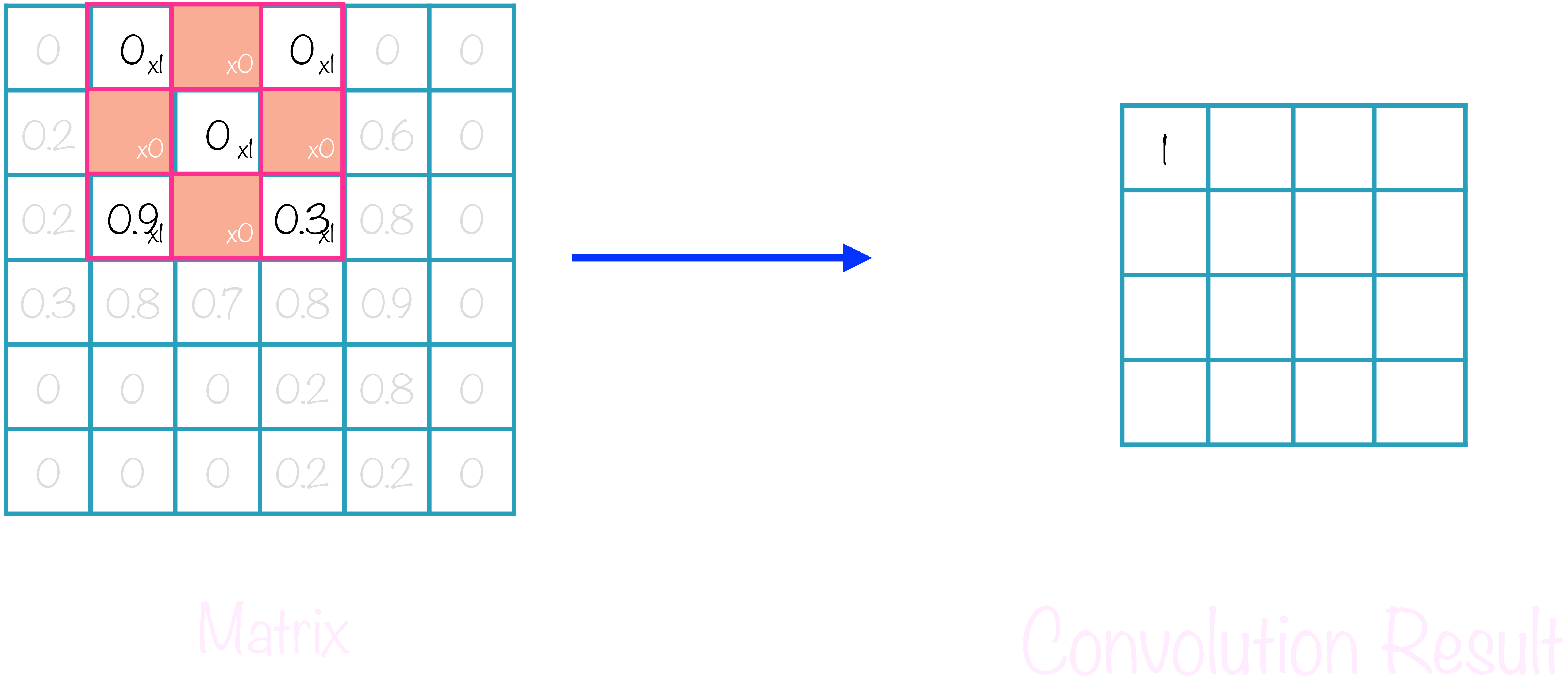
Convolution



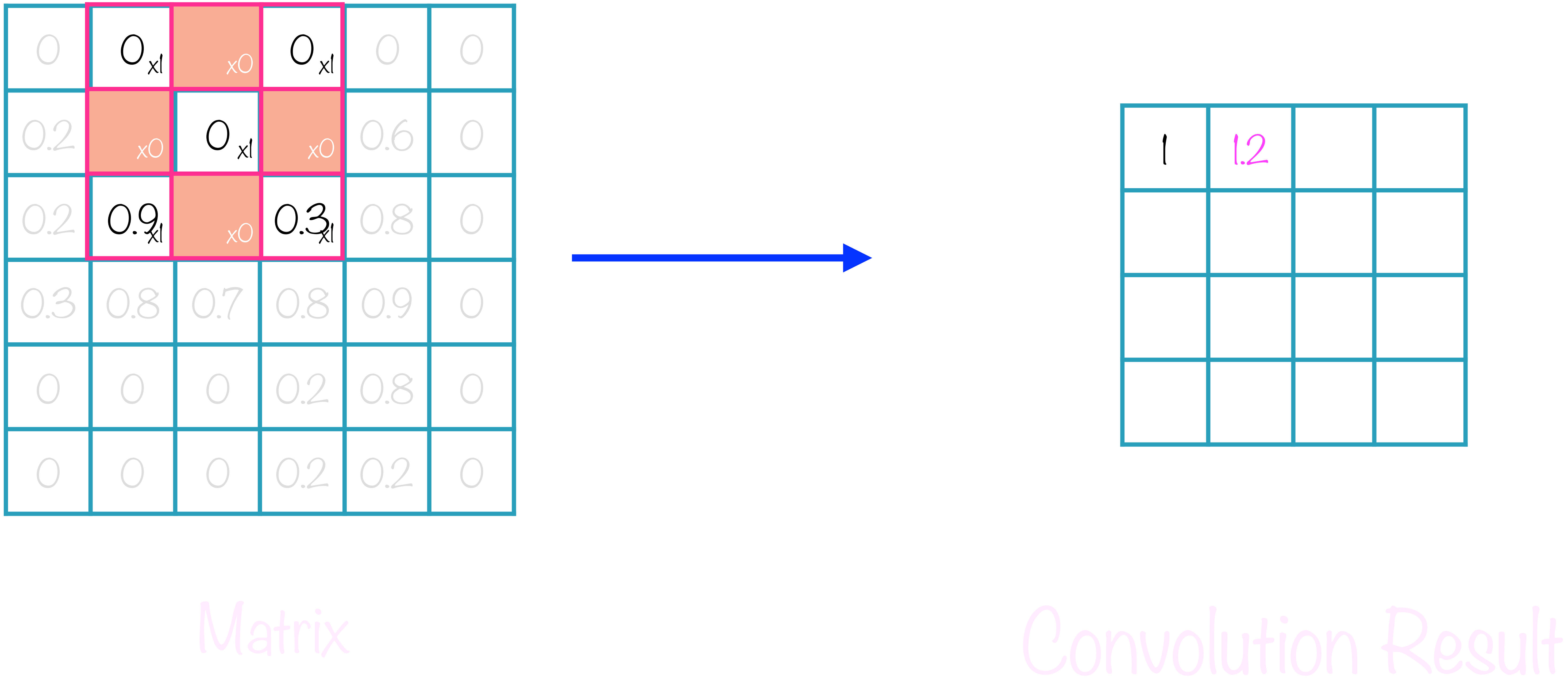
Convolution



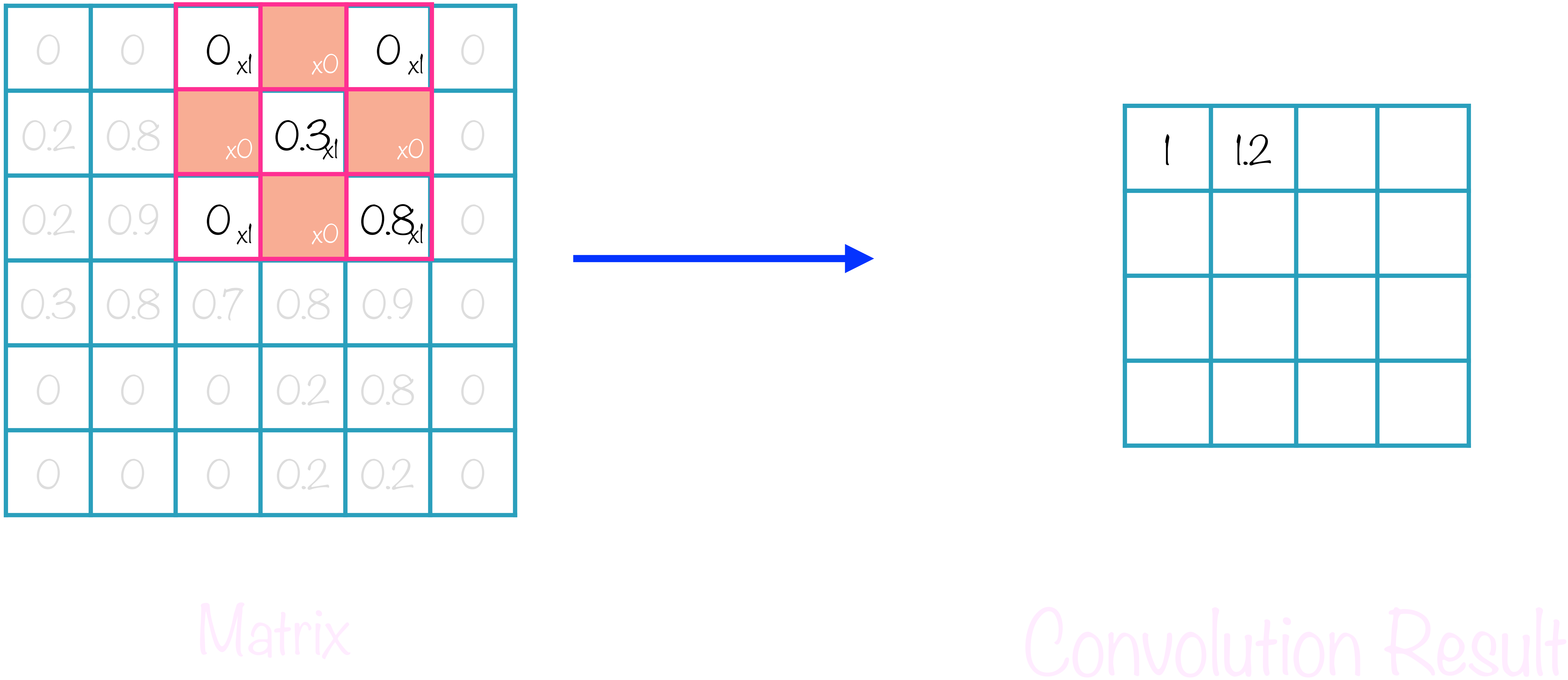
Convolution



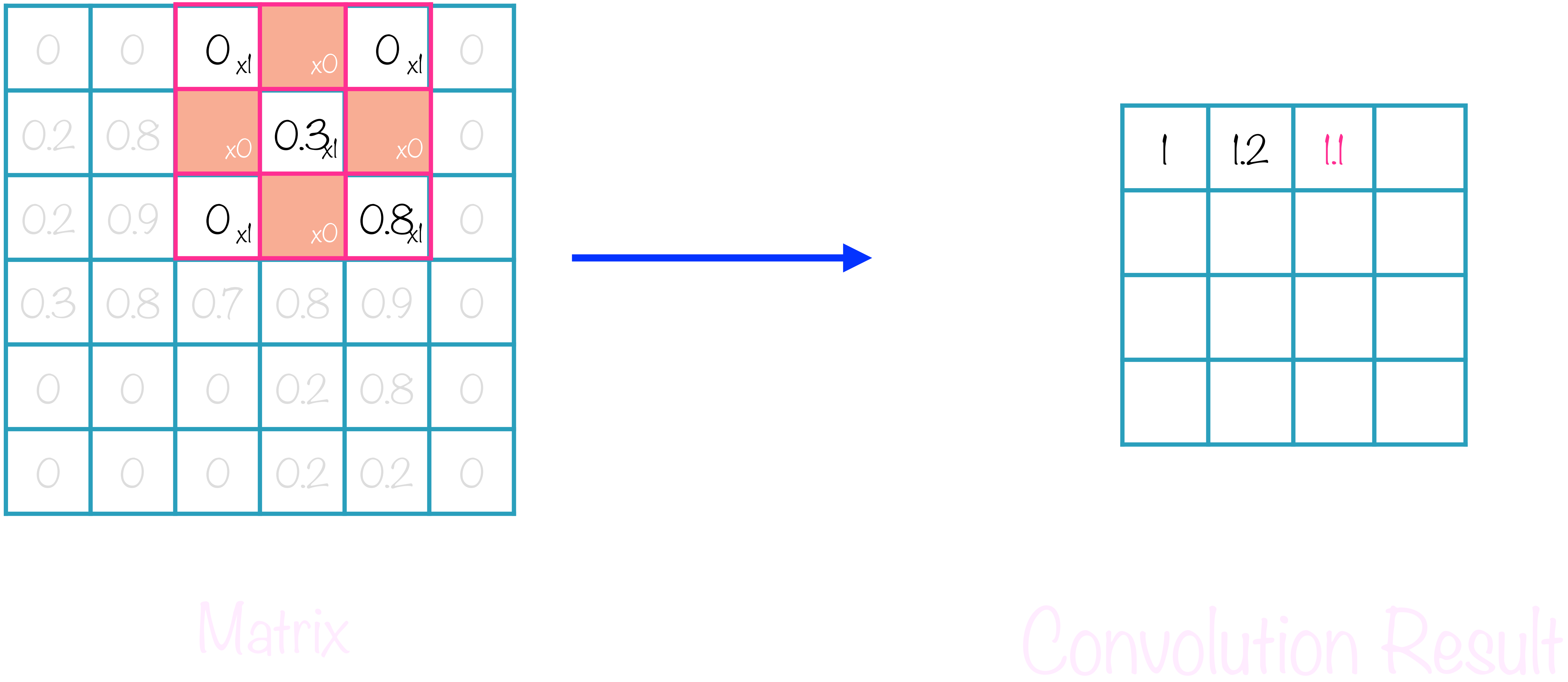
Convolution



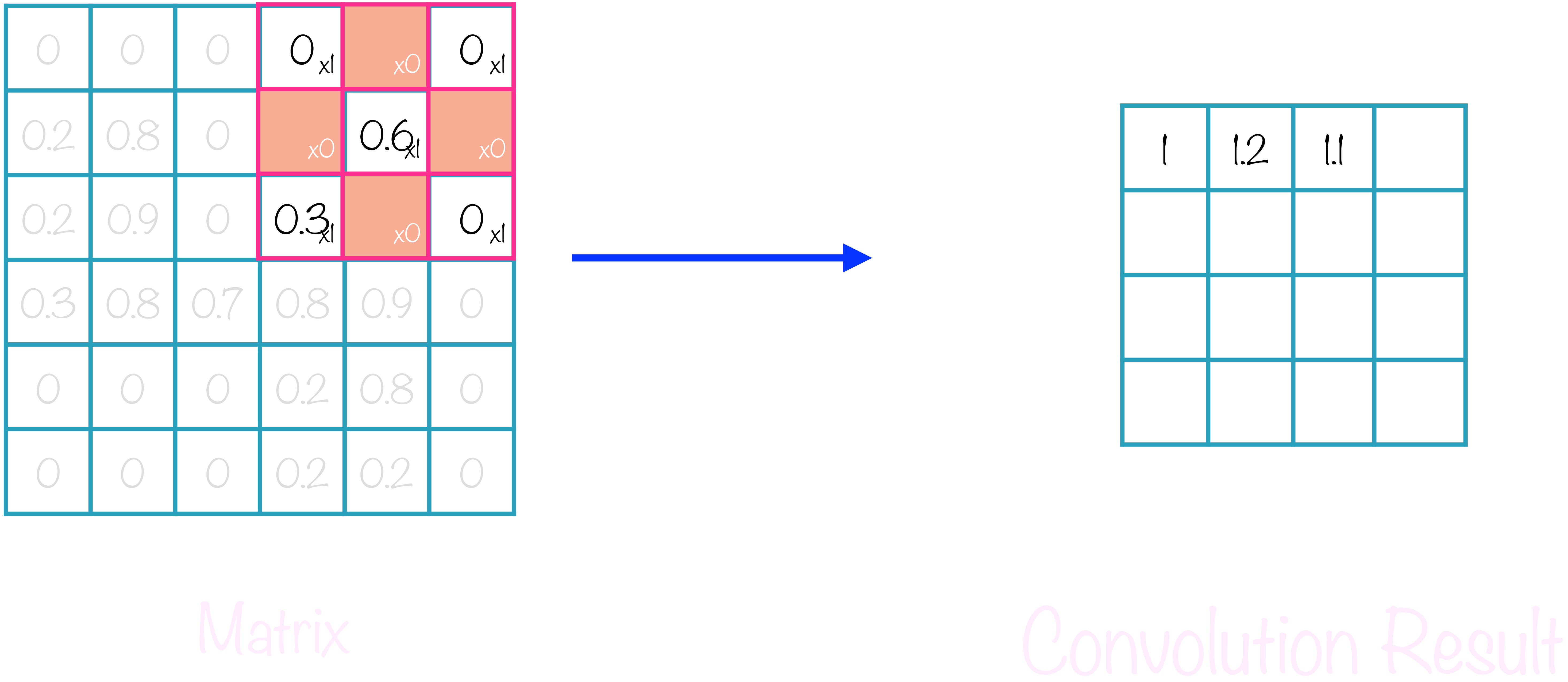
Convolution



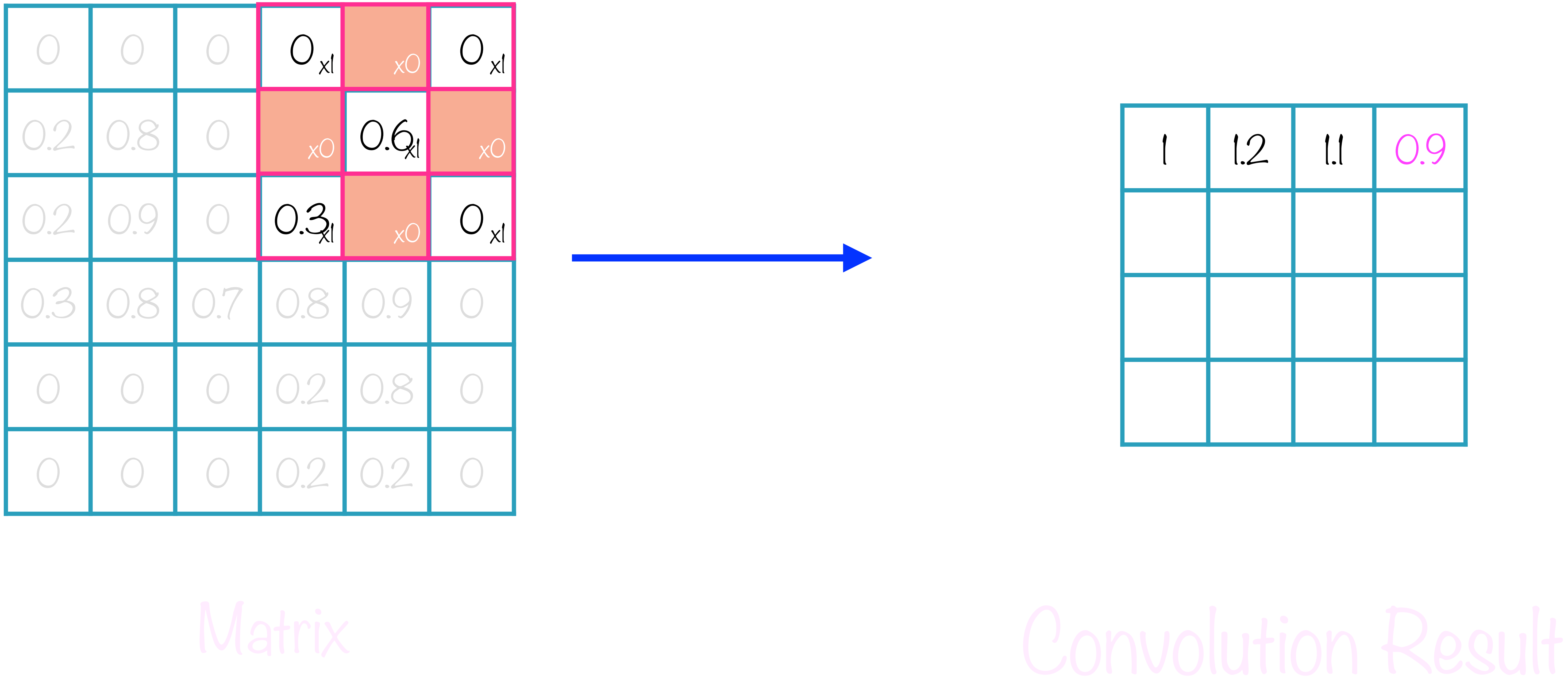
Convolution



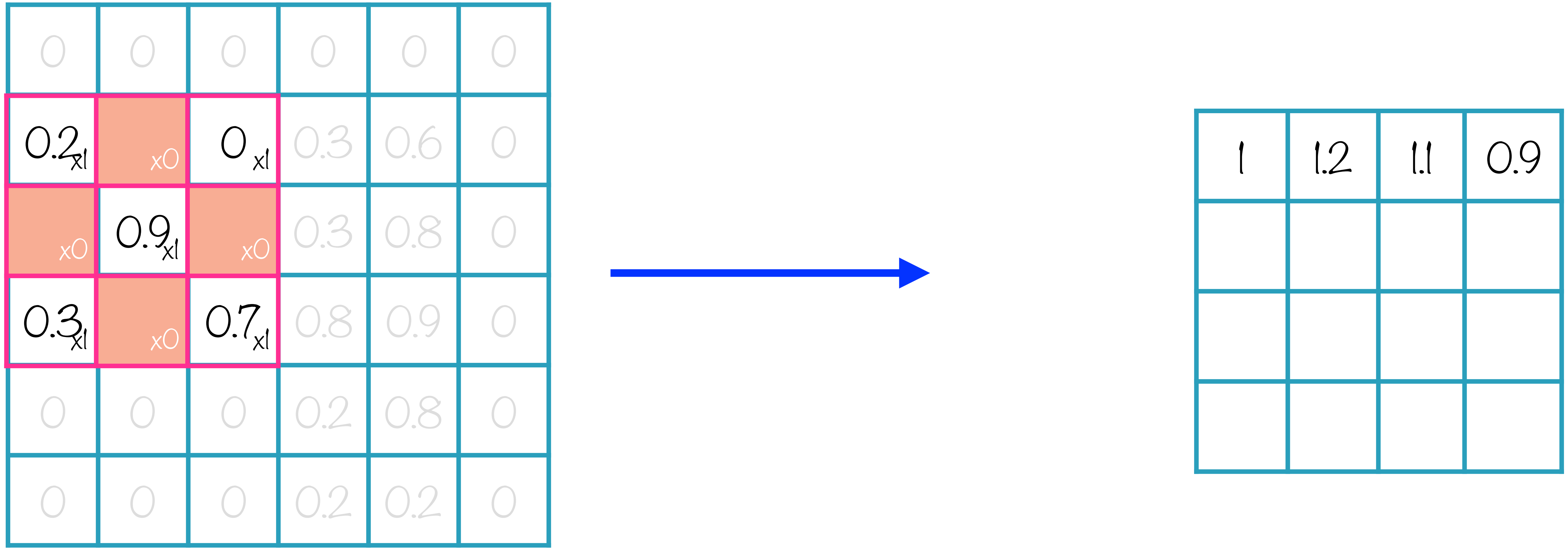
Convolution



Convolution



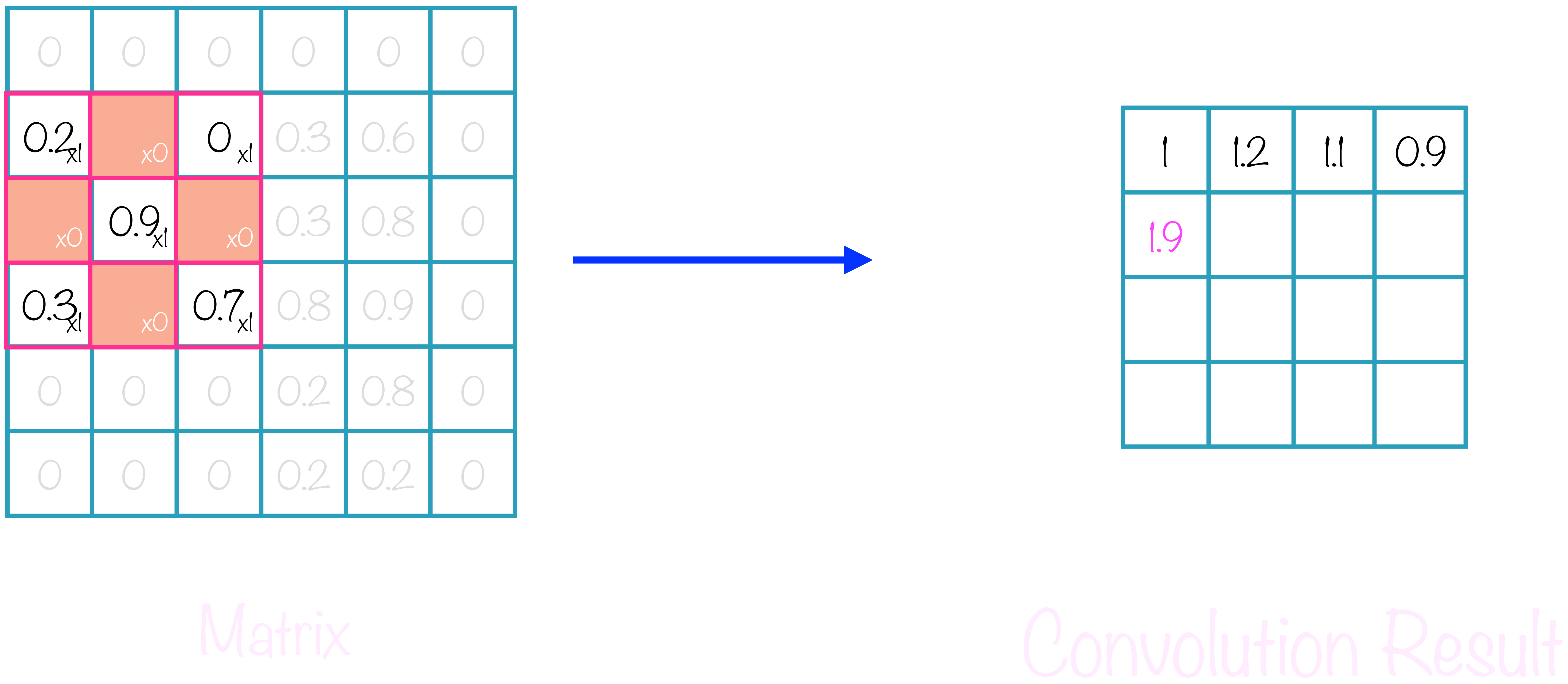
Convolution



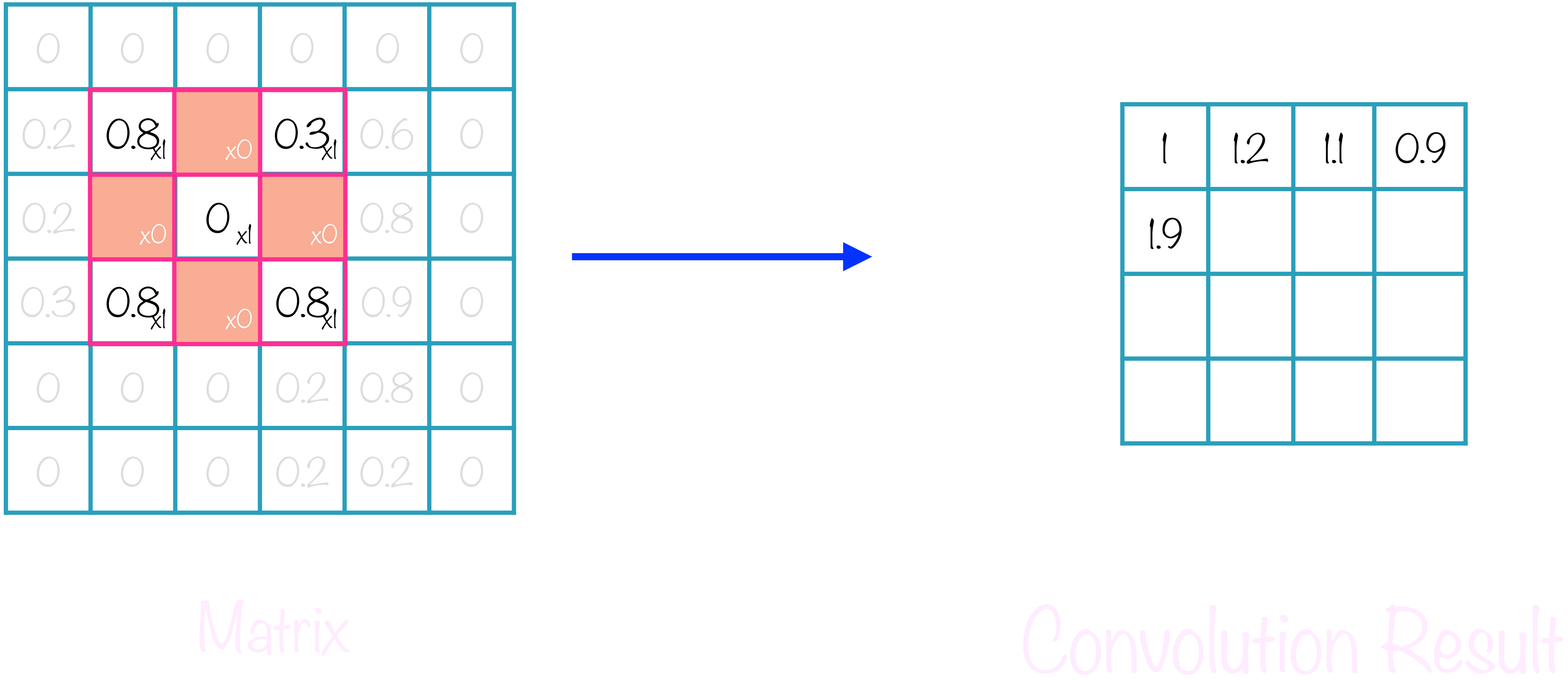
Matrix

Convolution Result

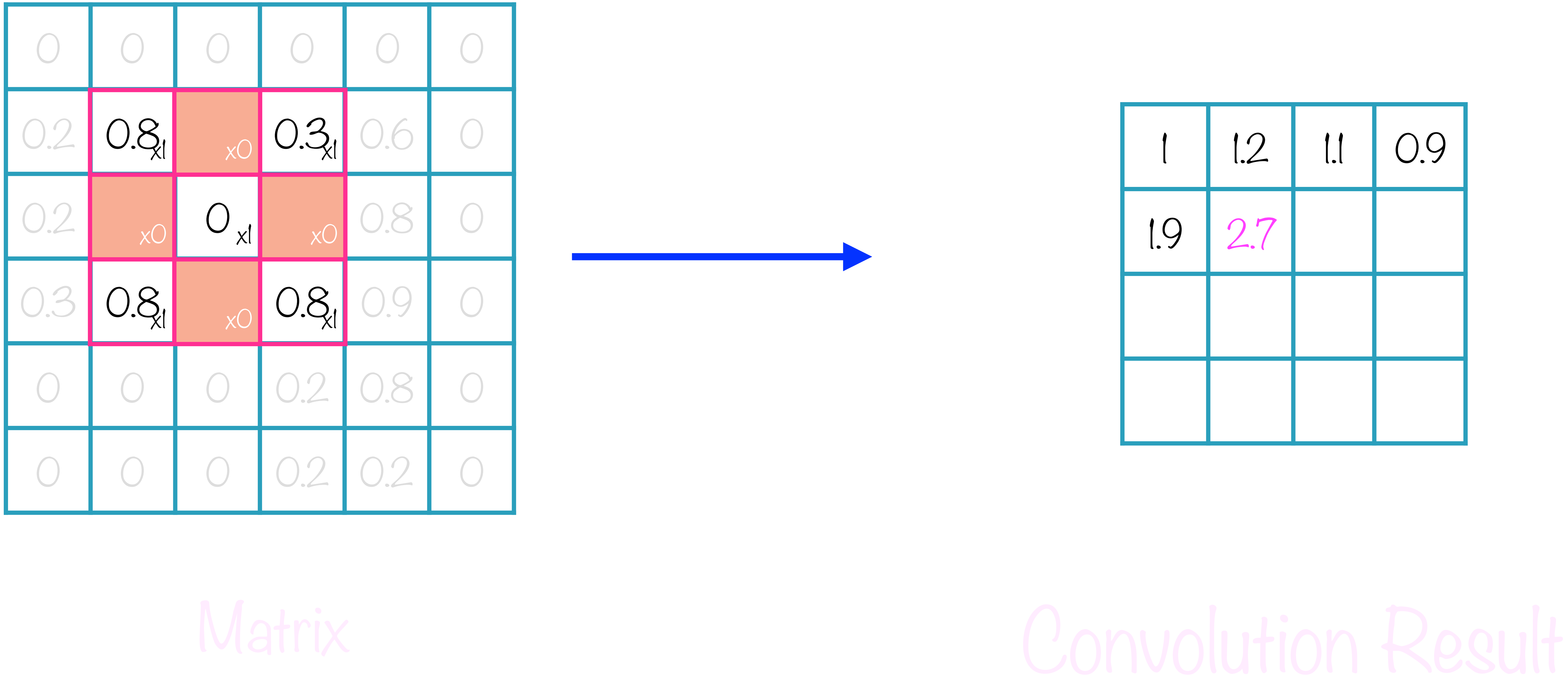
Convolution



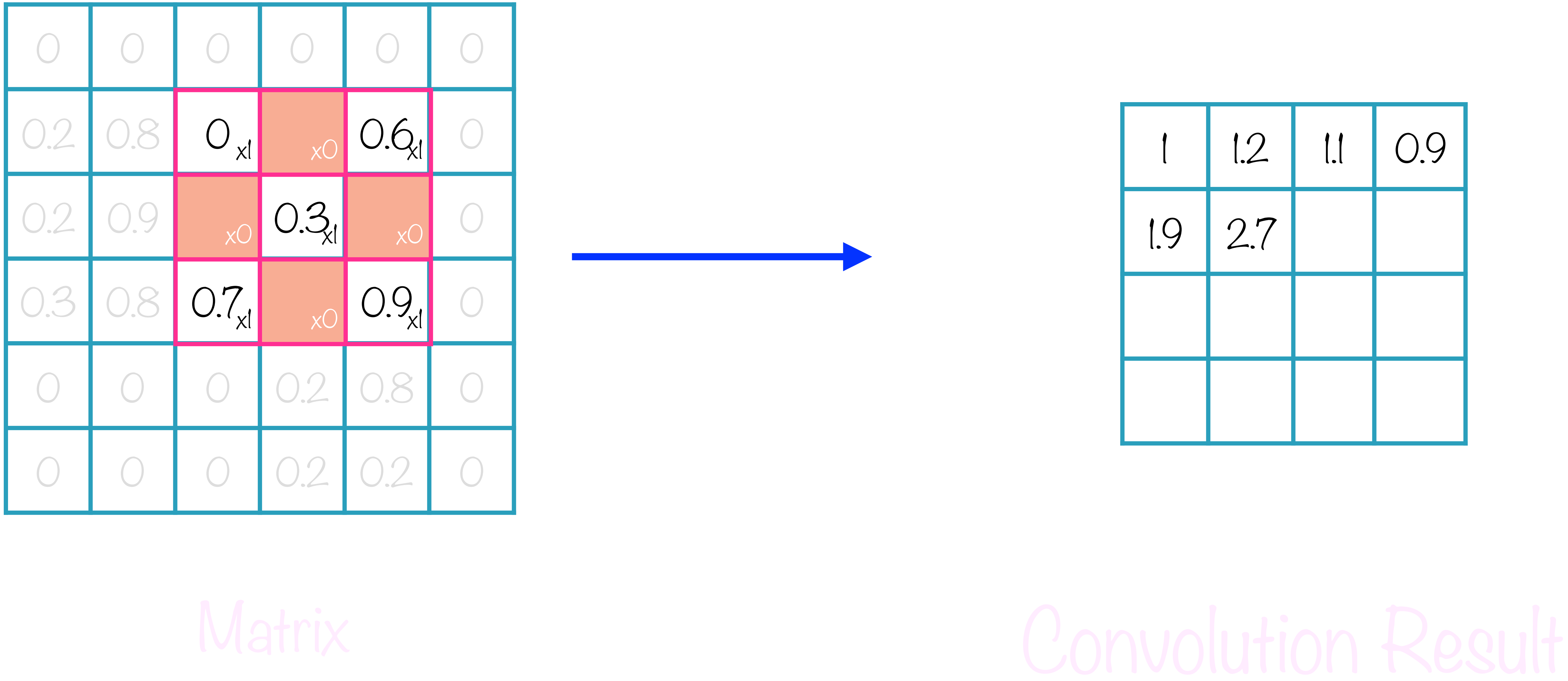
Convolution



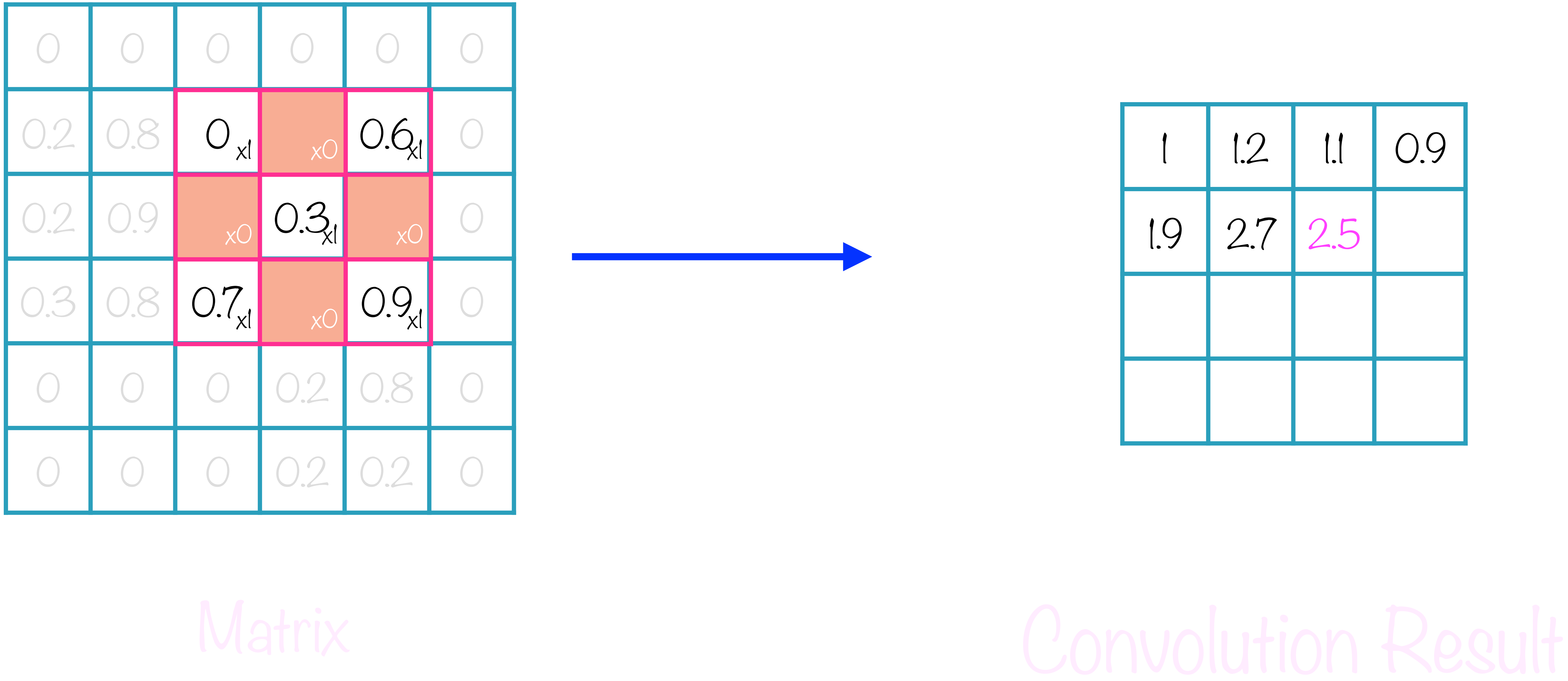
Convolution



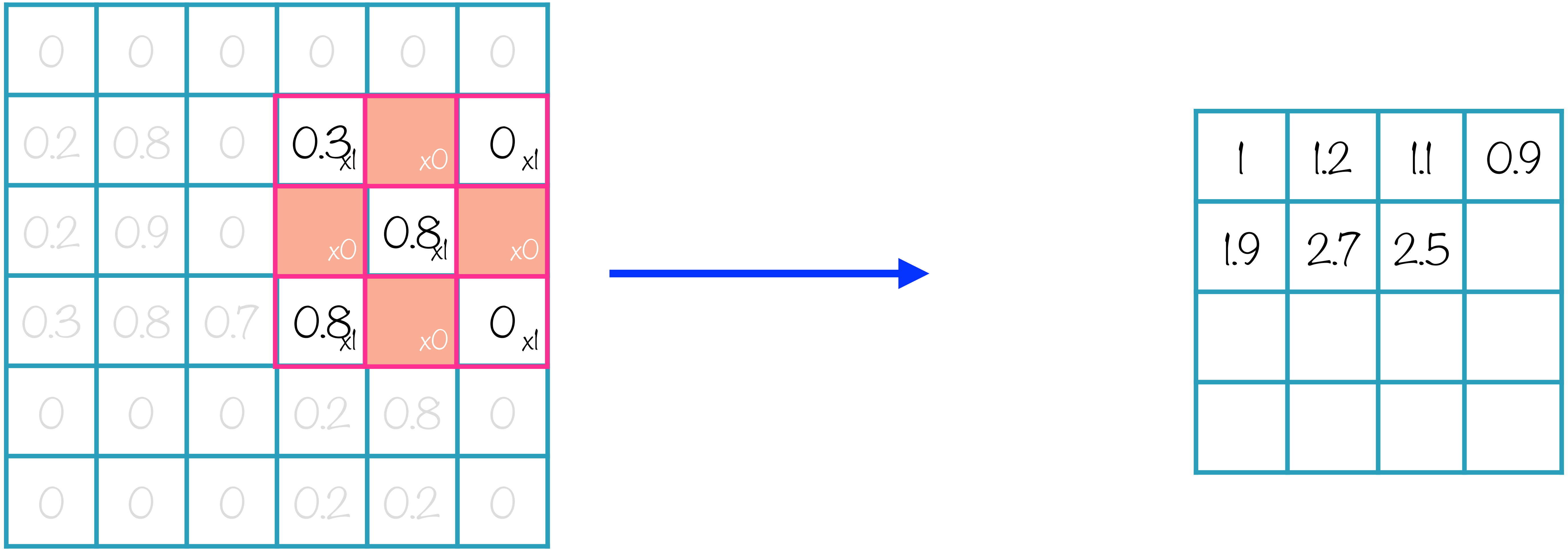
Convolution



Convolution



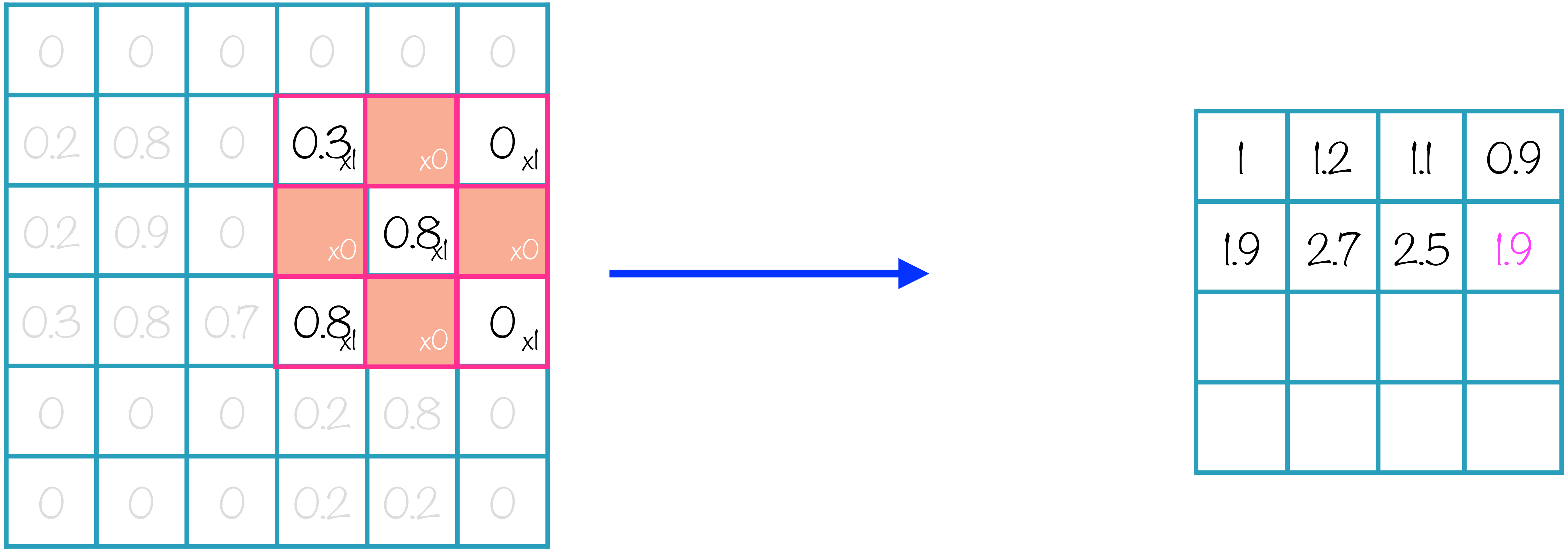
Convolution



Matrix

Convolution Result

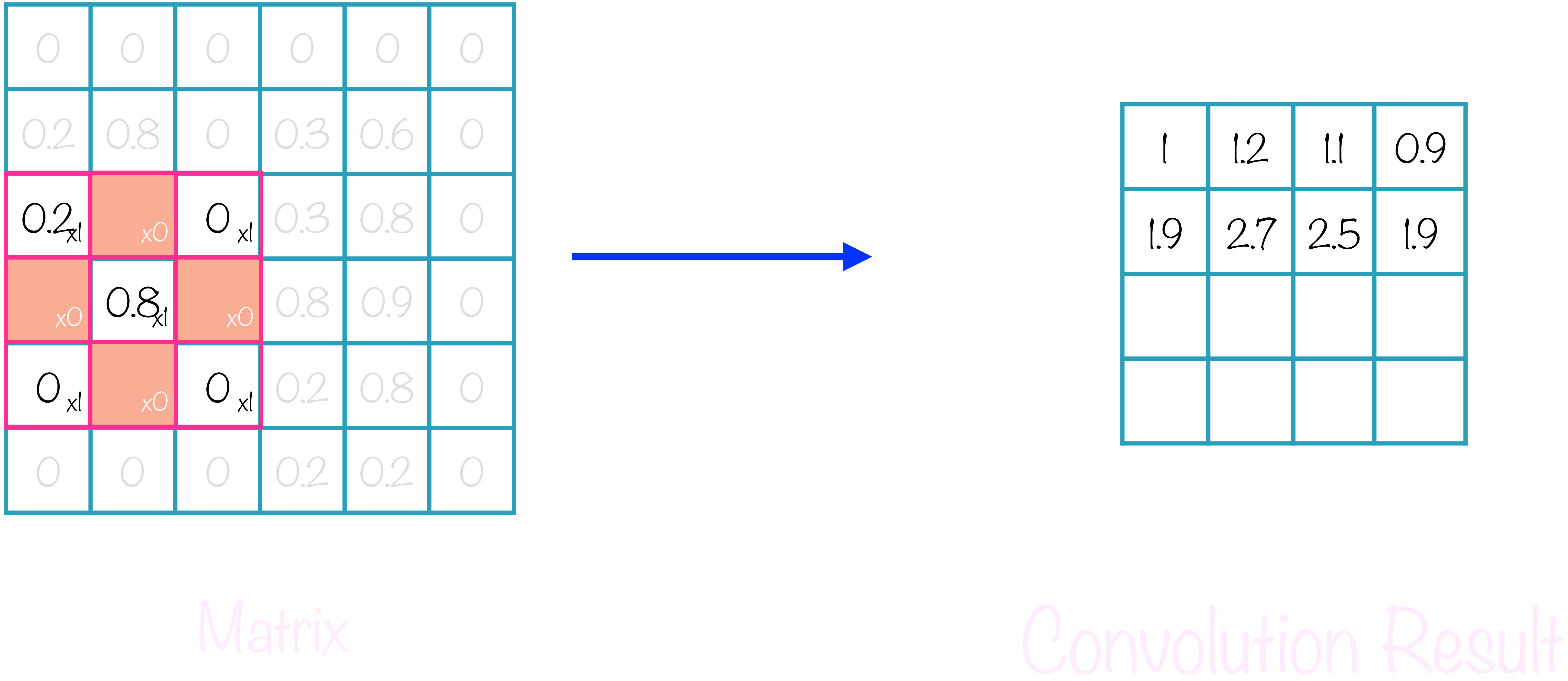
Convolution



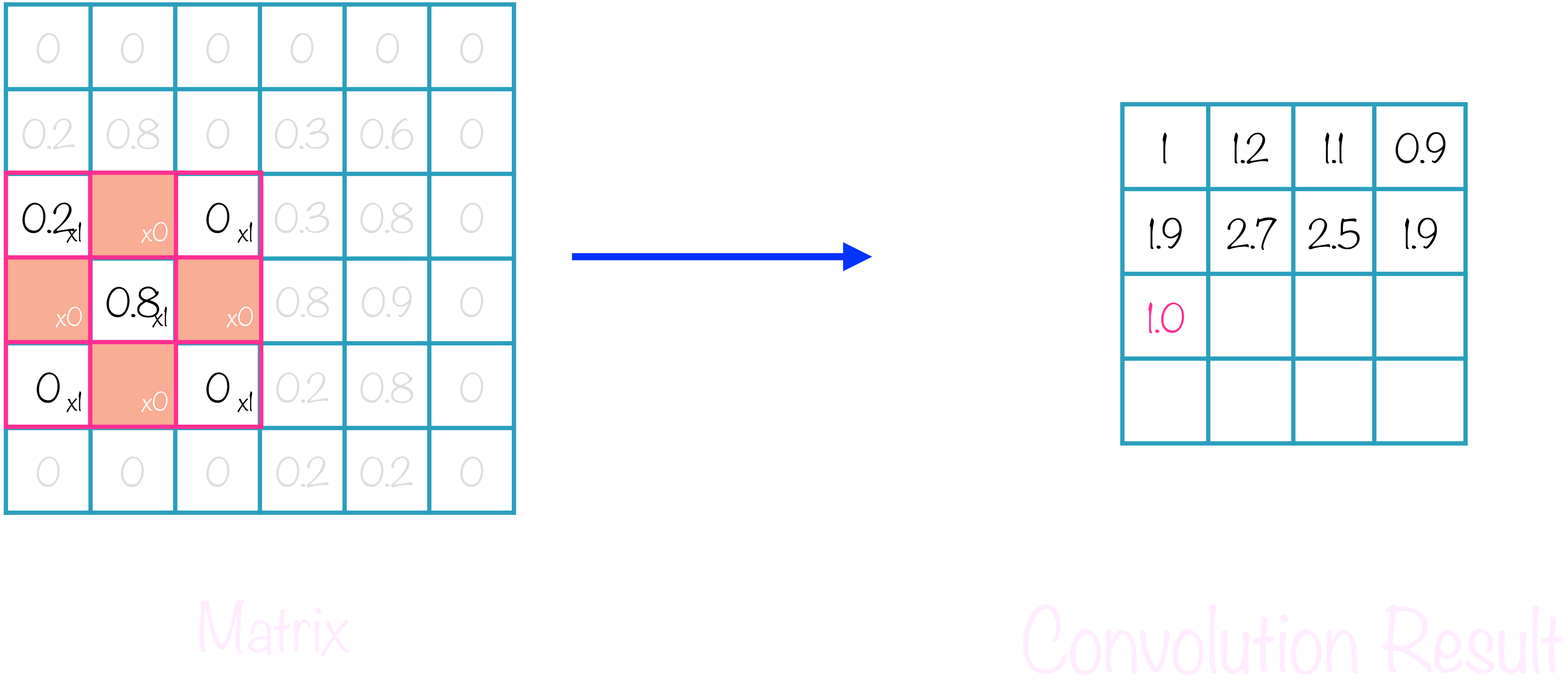
Matrix

Convolution Result

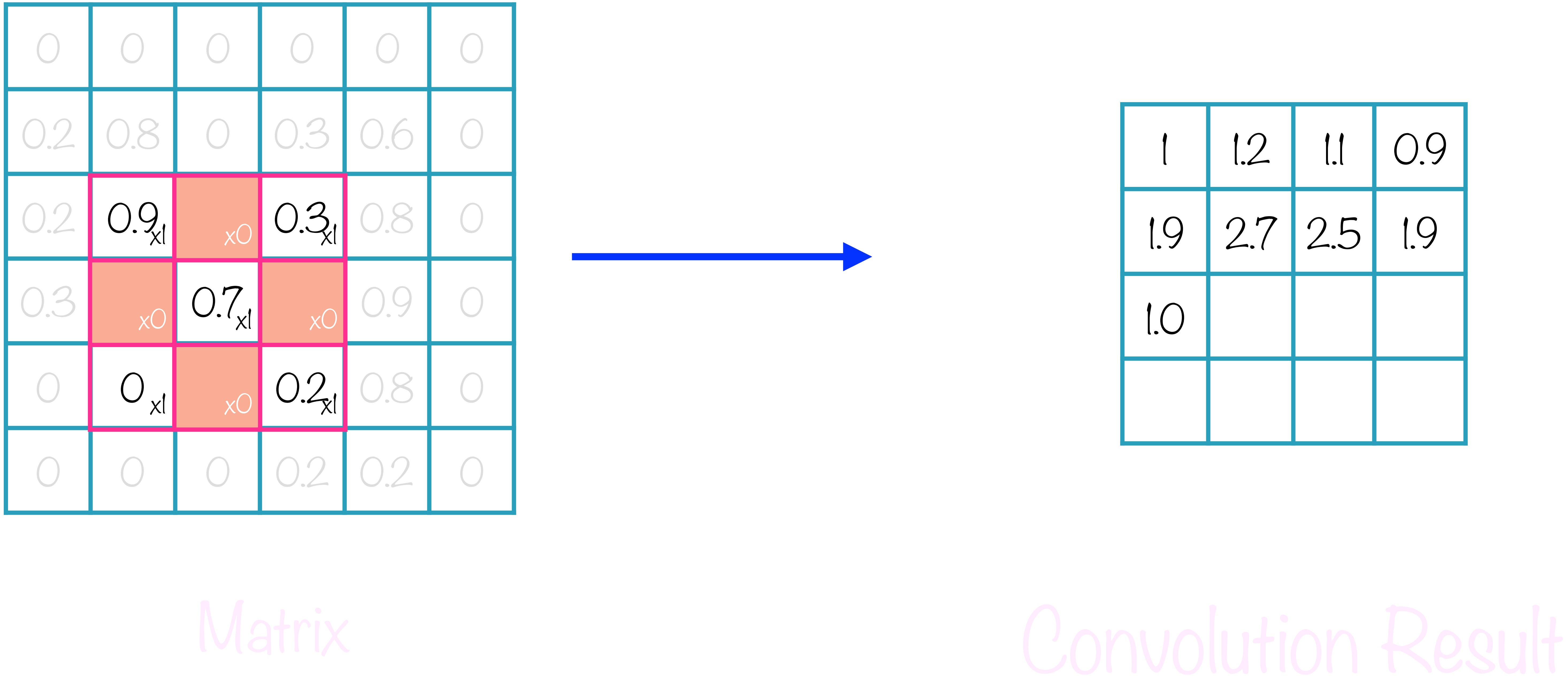
Convolution



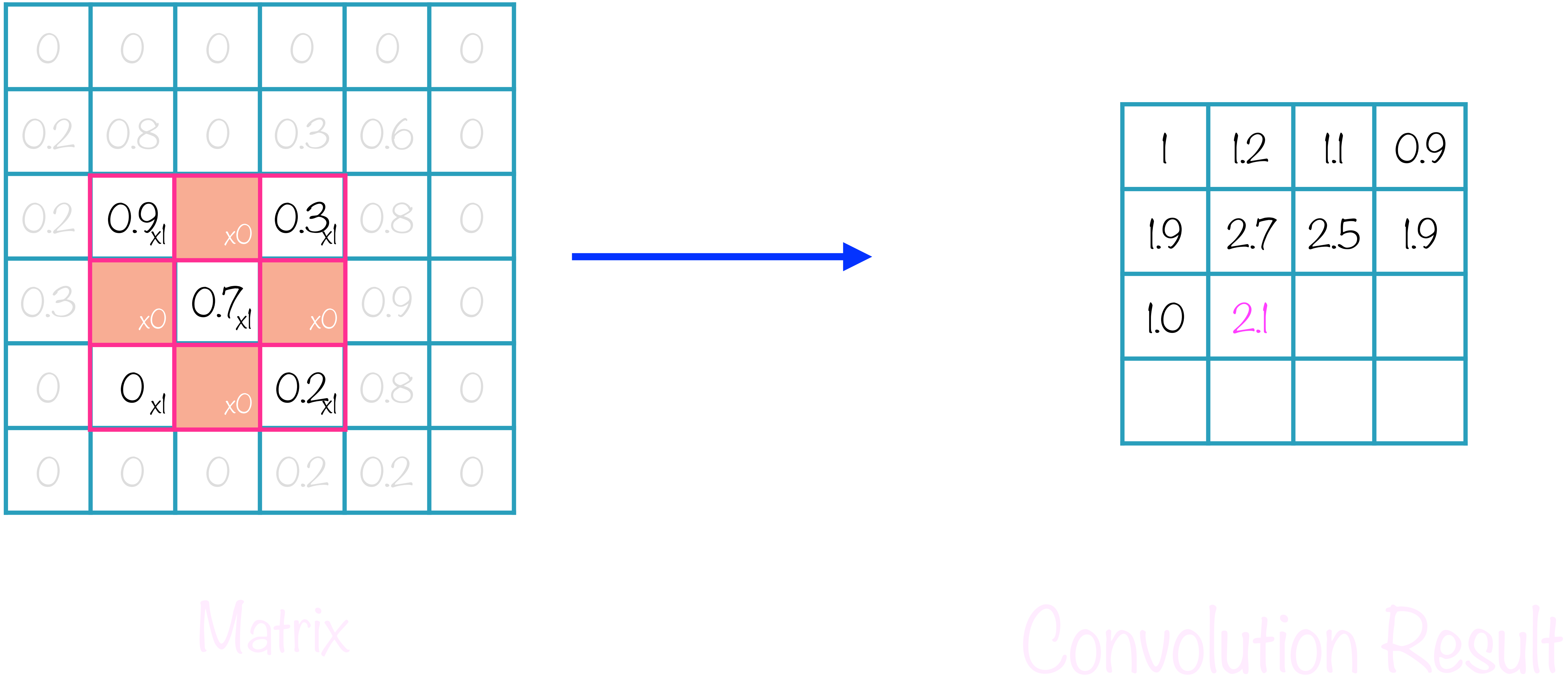
Convolution



Convolution



Convolution



Convolution

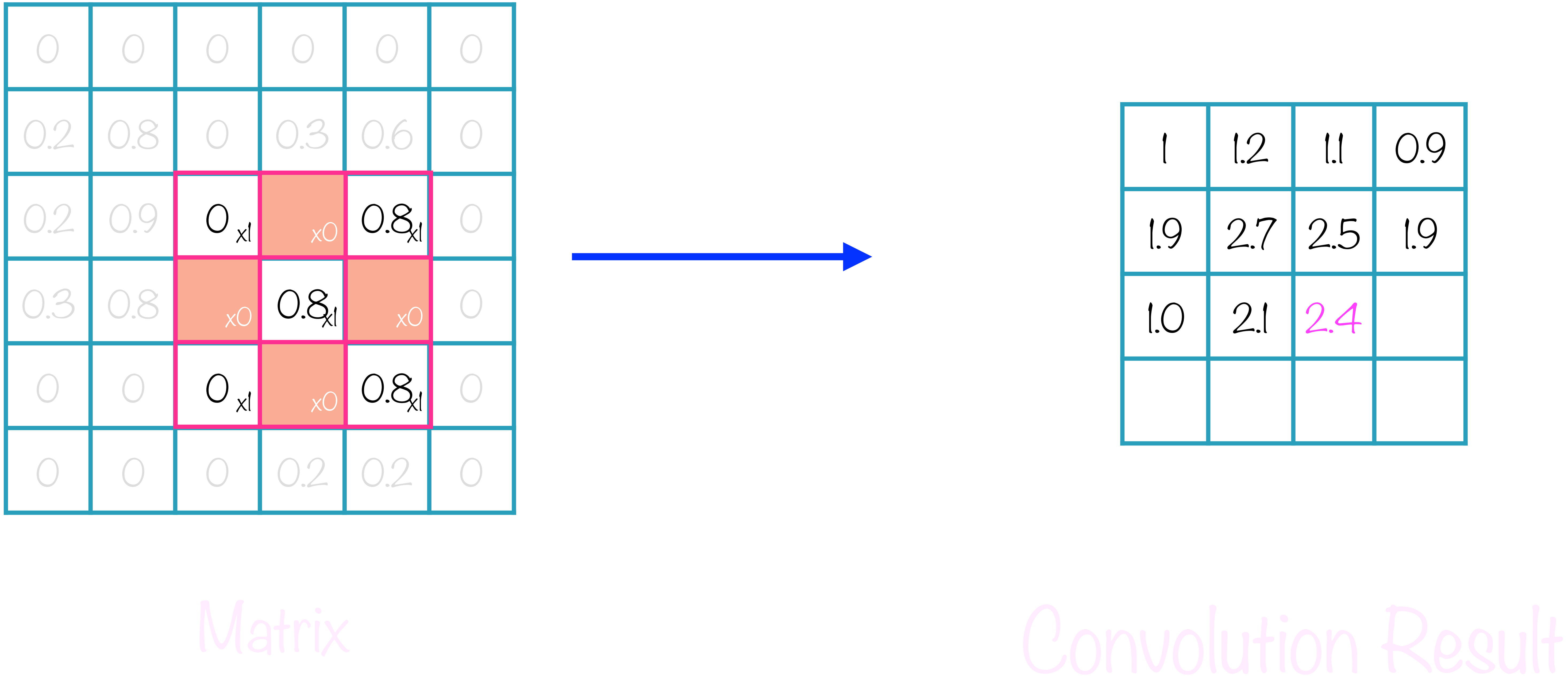
0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0 x1	x0	0.8 x1	0
0.3	0.8	x0	0.8 x1	x0	0
0	0	0 x1	x0	0.8 x1	0
0	0	0	0.2	0.2	0

Matrix

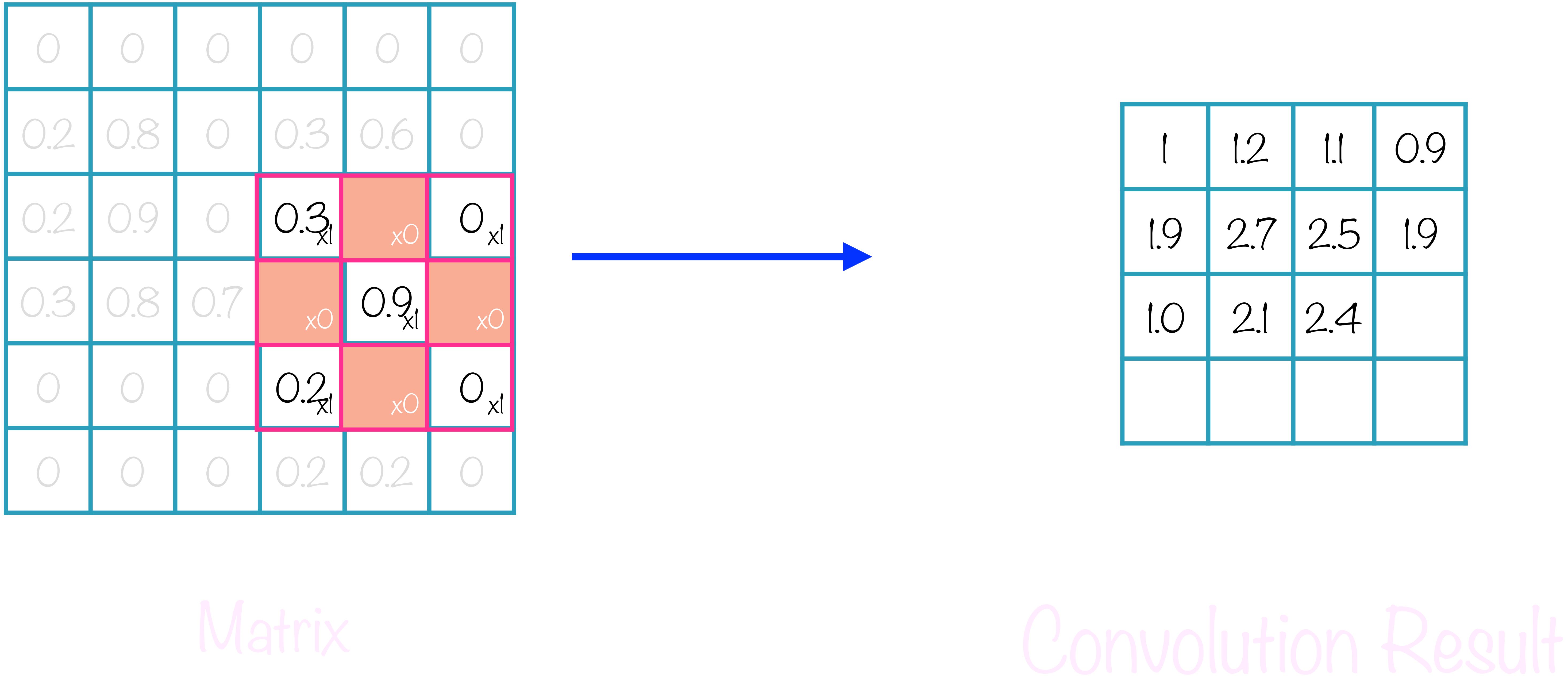
1	1.2	1.1	0.9
1.9	2.7	2.5	1.9
1.0	2.1		

Convolution Result

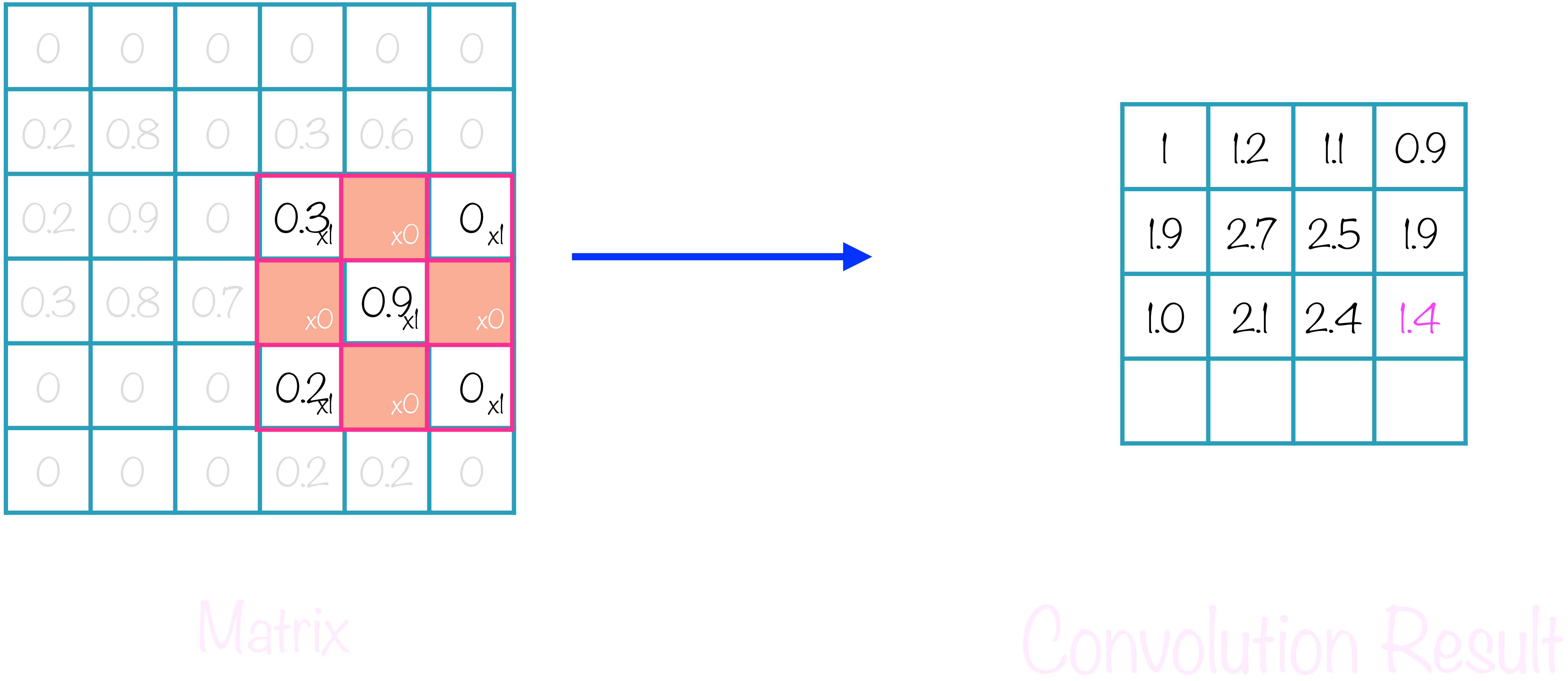
Convolution



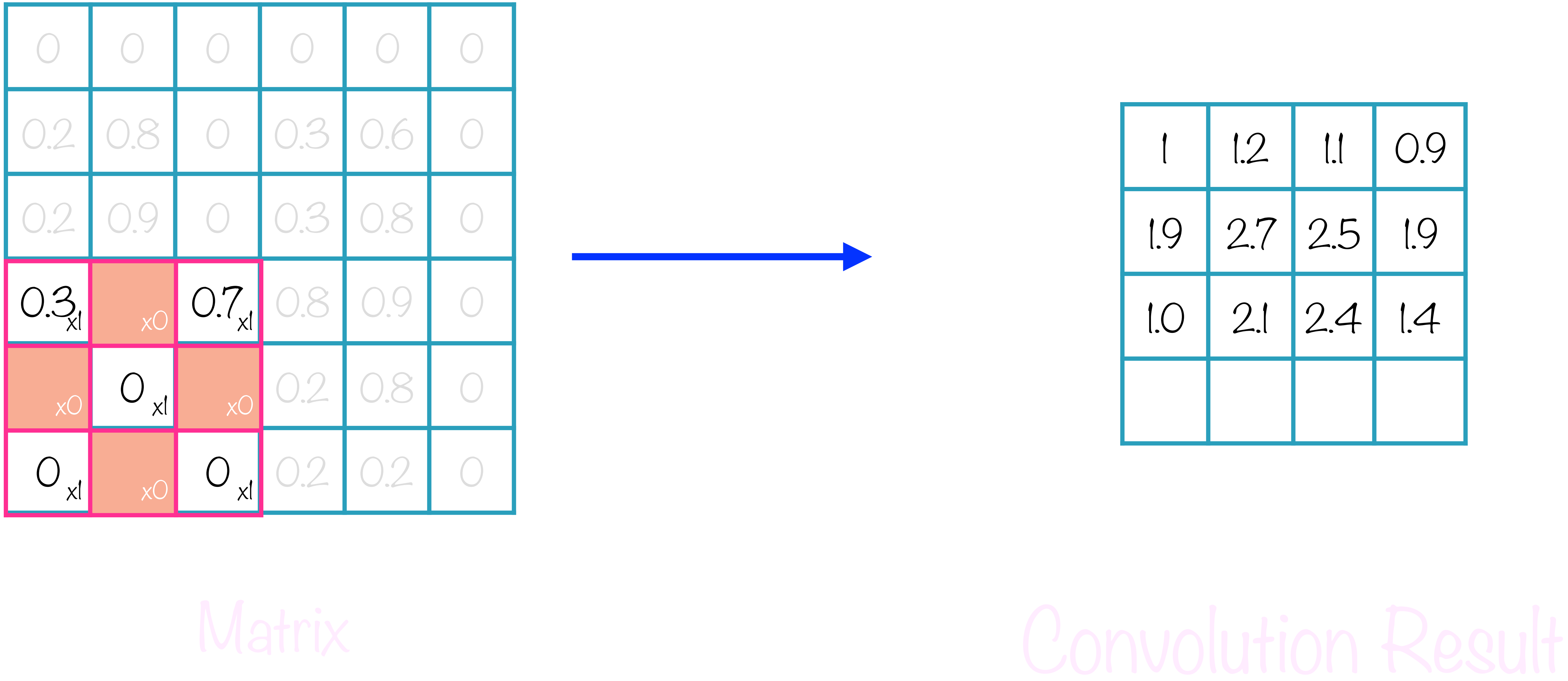
Convolution



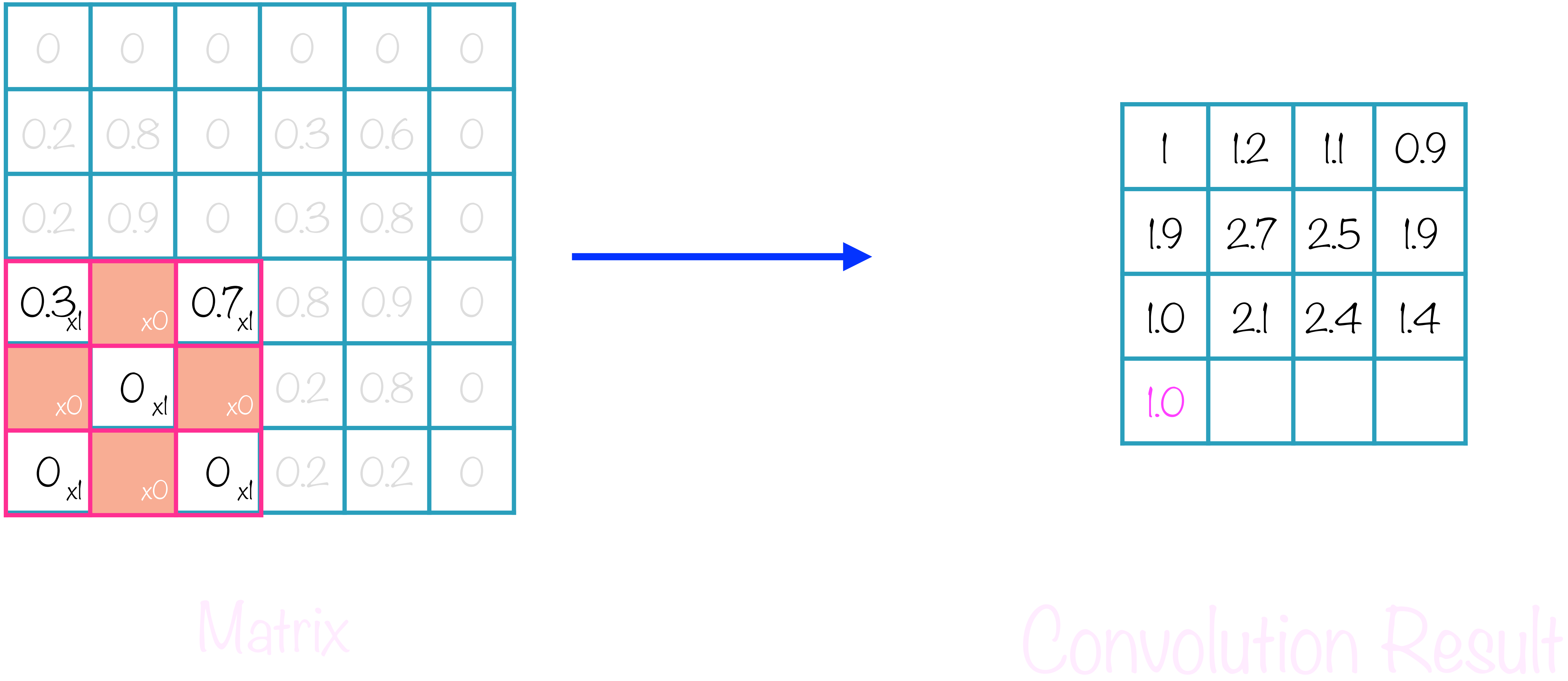
Convolution



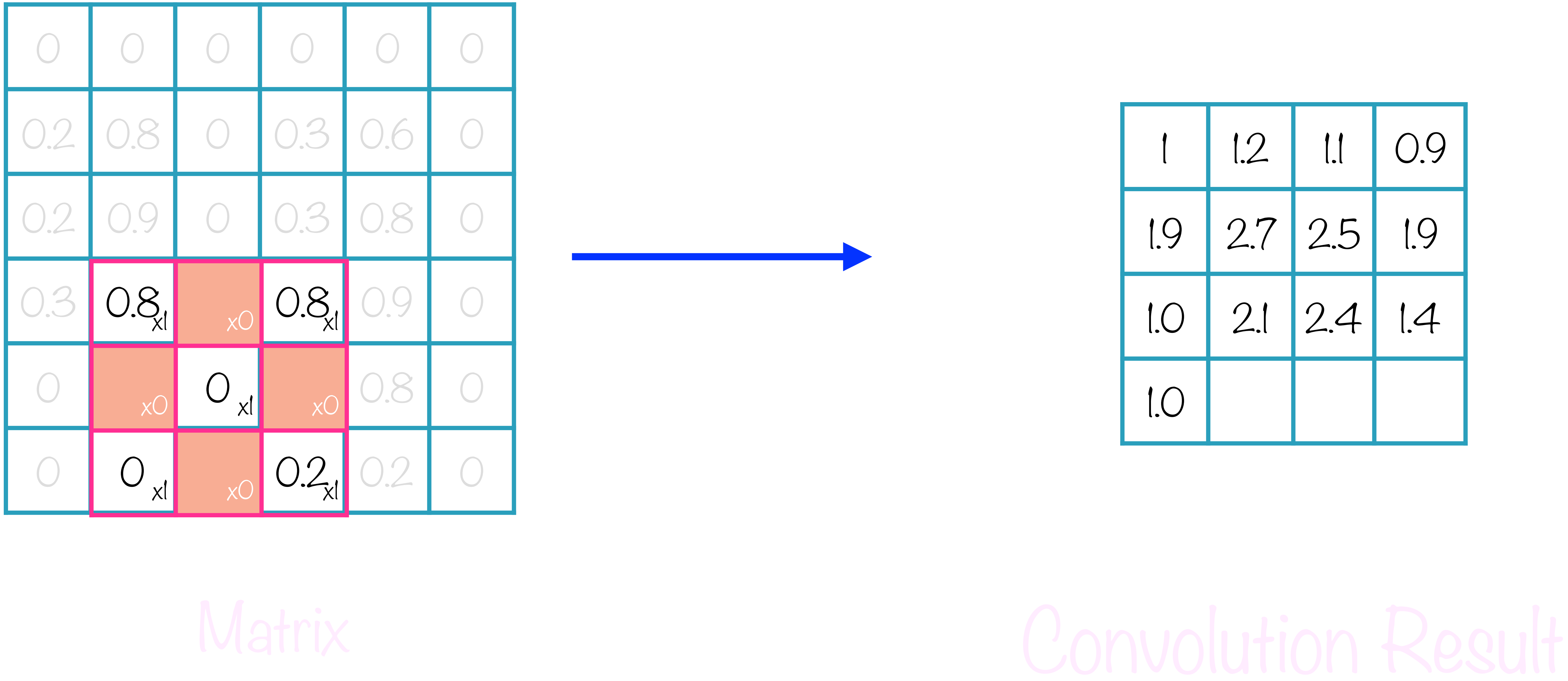
Convolution



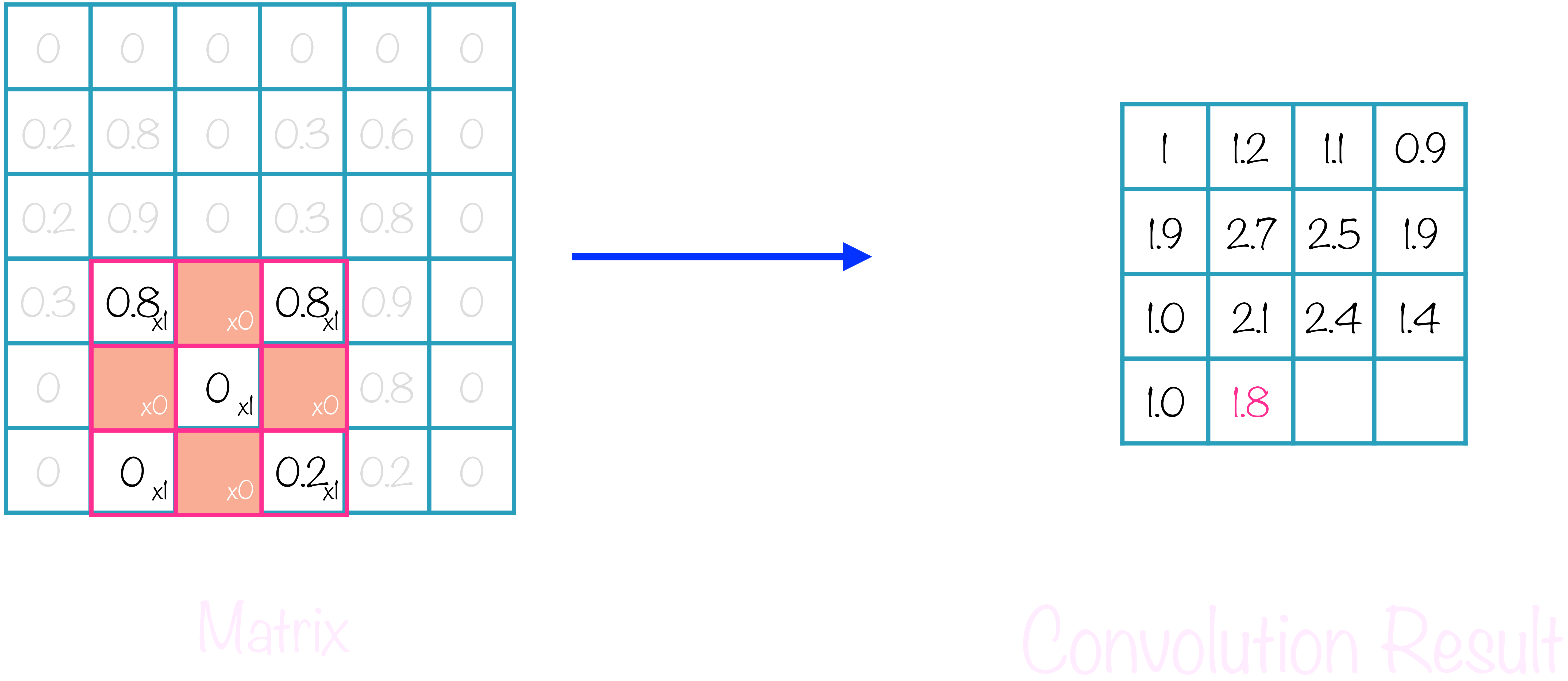
Convolution



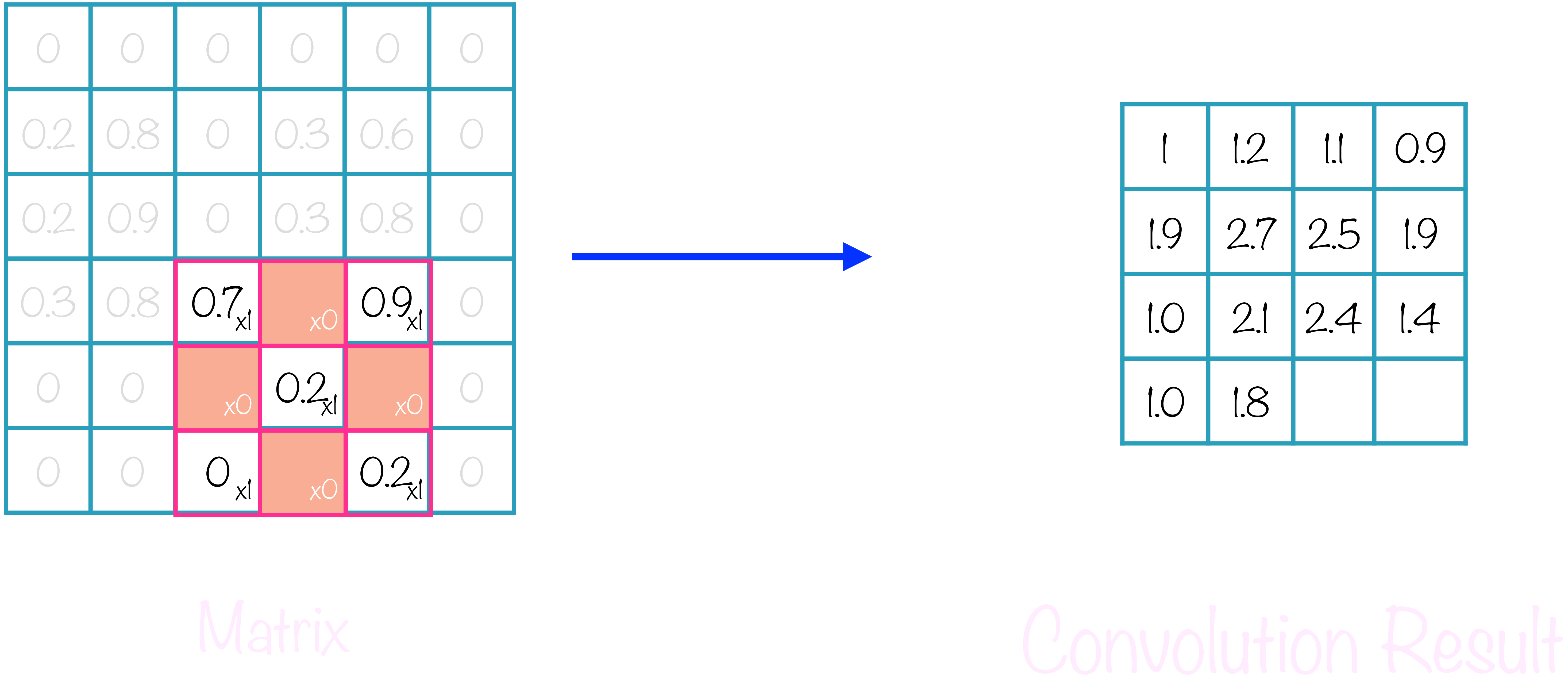
Convolution



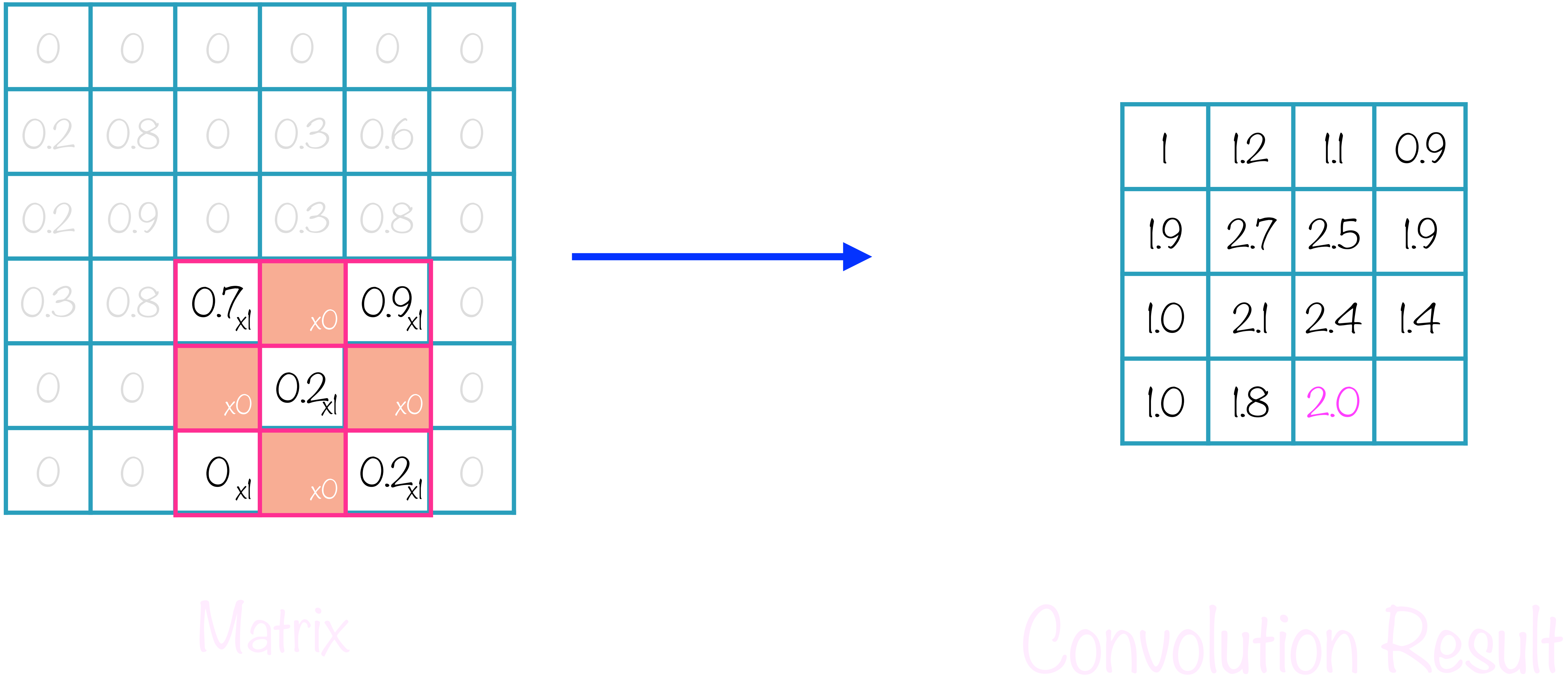
Convolution



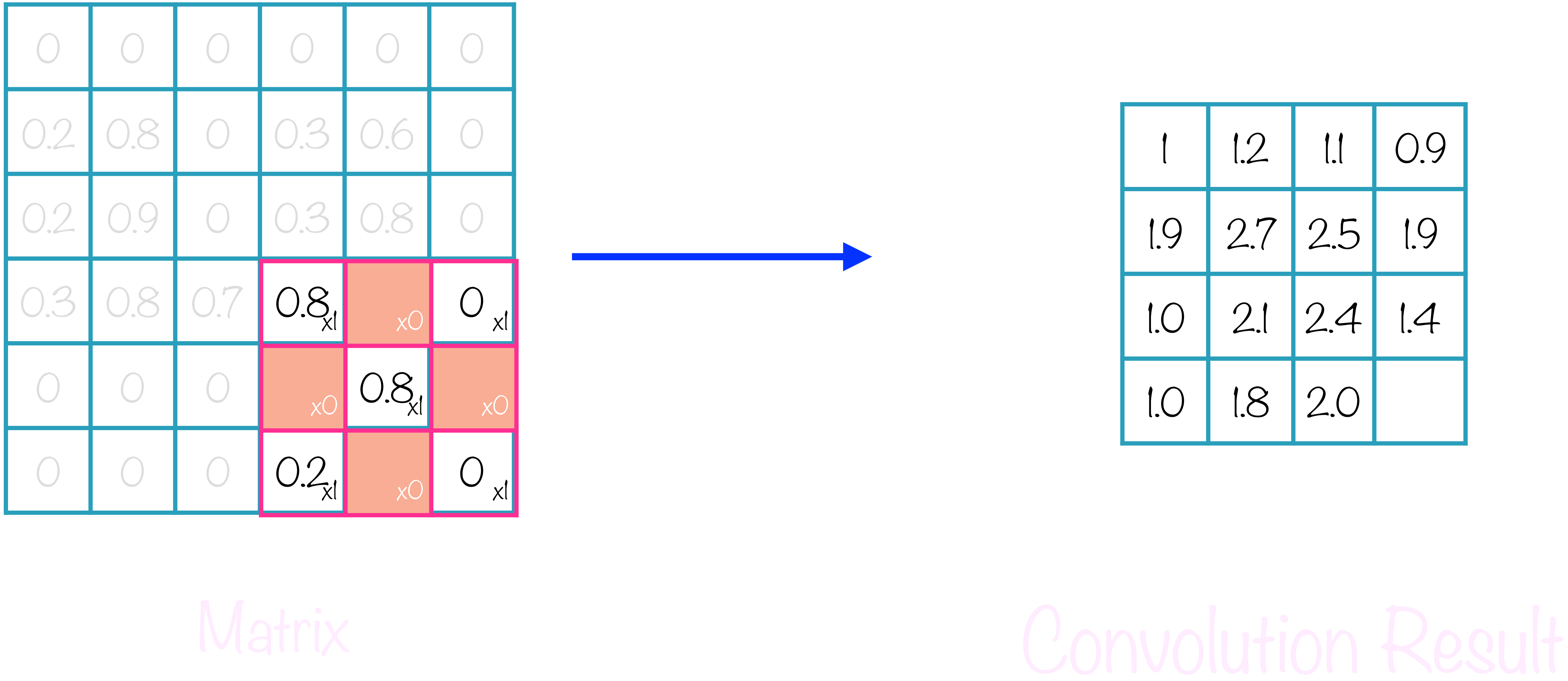
Convolution



Convolution



Convolution



Convolution

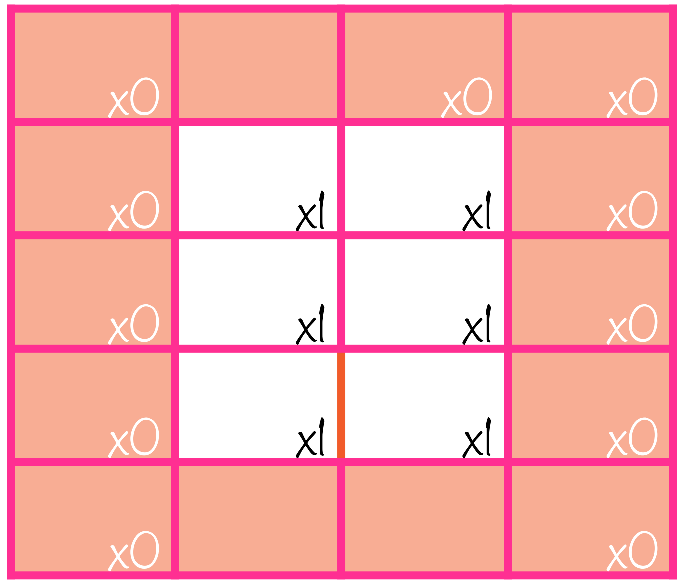
0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8 x1	x0	0 x1
0	0	0	x0	0.8 x1	x0
0	0	0	0.2 x1	x0	0 x1

Matrix

1	1.2	1.1	0.9
1.9	2.7	2.5	1.9
1.0	2.1	2.4	1.4
1.0	1.8	2.0	1.8

Convolution Result

Choice of Kernel Function



Averaging neighbouring pixels ~ Blurring

Subtracting neighbouring pixels ~ Edge detection

Positive middle, negative neighbours ~ Sharpen

Negative corners, zero elsewhere ~ Edge enhance

More complex patterns ~ Emboss

...

Choice of Kernel Function

http://aishack.in/tutorials/image-convolution-examples/

Blur

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

A simple blur done with convolutions

Line Detection

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

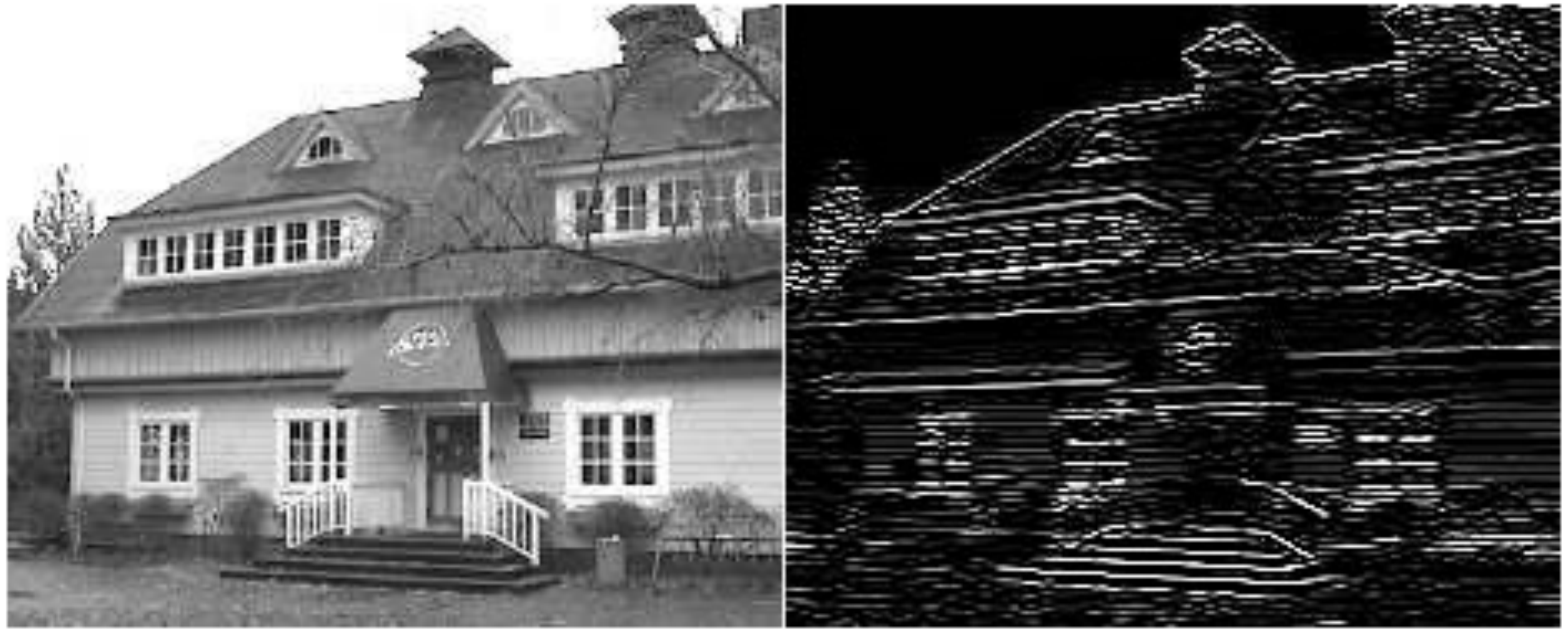
2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

Horizontal Lines

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines



Edge Detection

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

Zero-padding, Stride Size

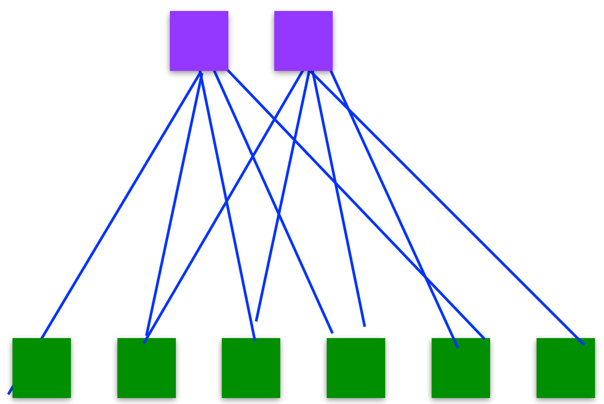
Narrow vs. Wide Convolution

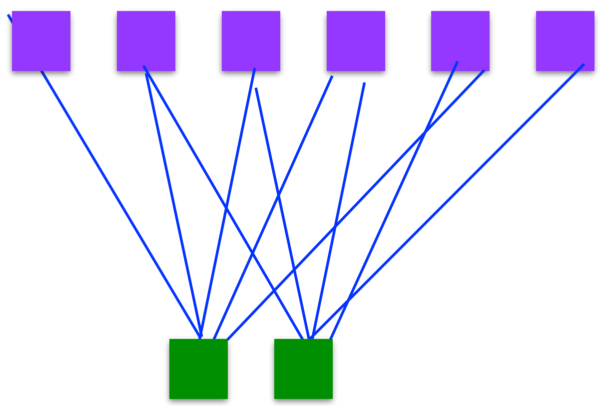
Input matrix i.e. image

Narrow vs. Wide Convolution

Convolution result

Narrow vs. Wide Convolution





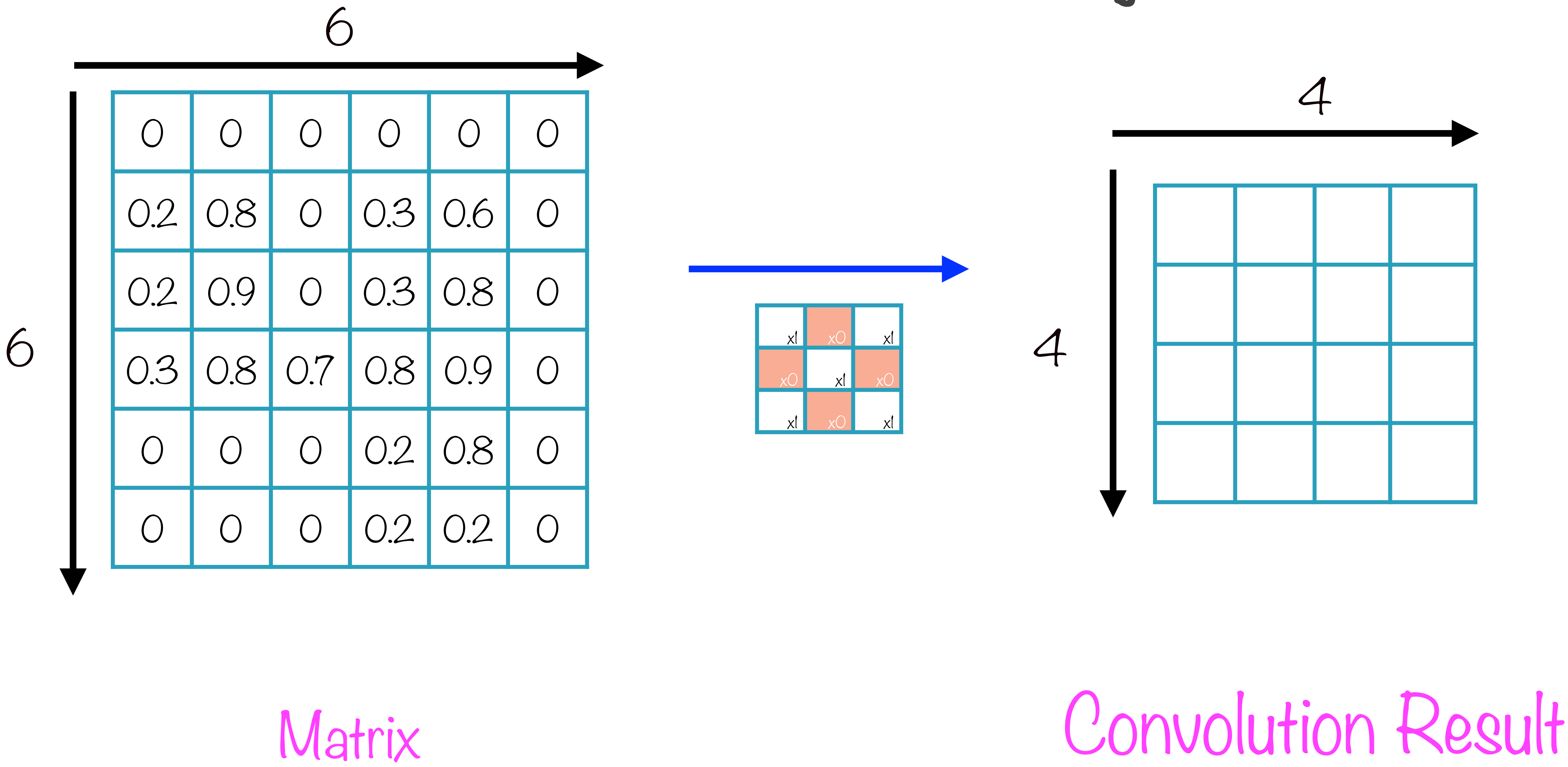
Narrow Convolution

Little zero padding; output narrower than input

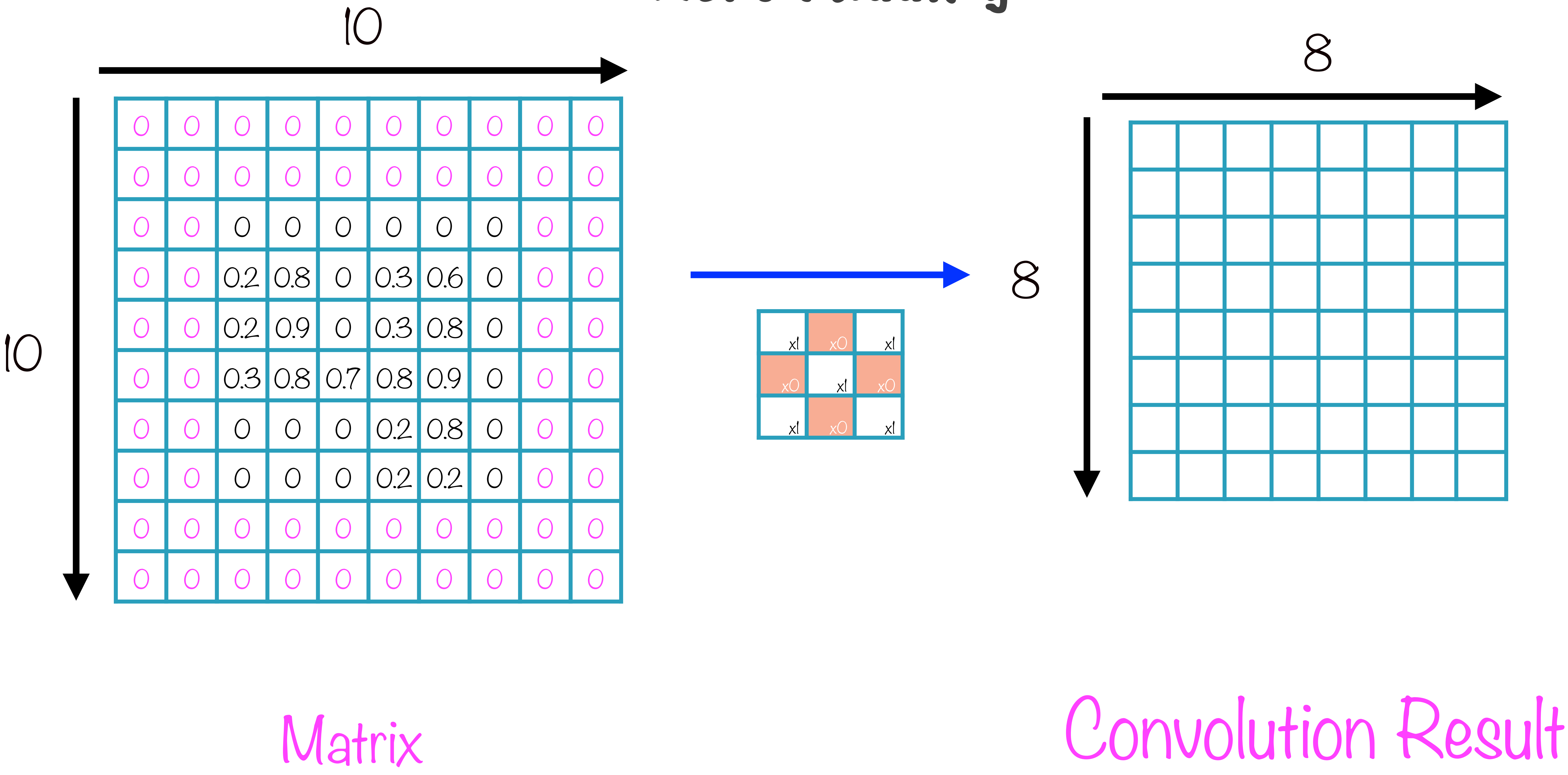
Wide Convolution

Lots of zero padding; output wider than input

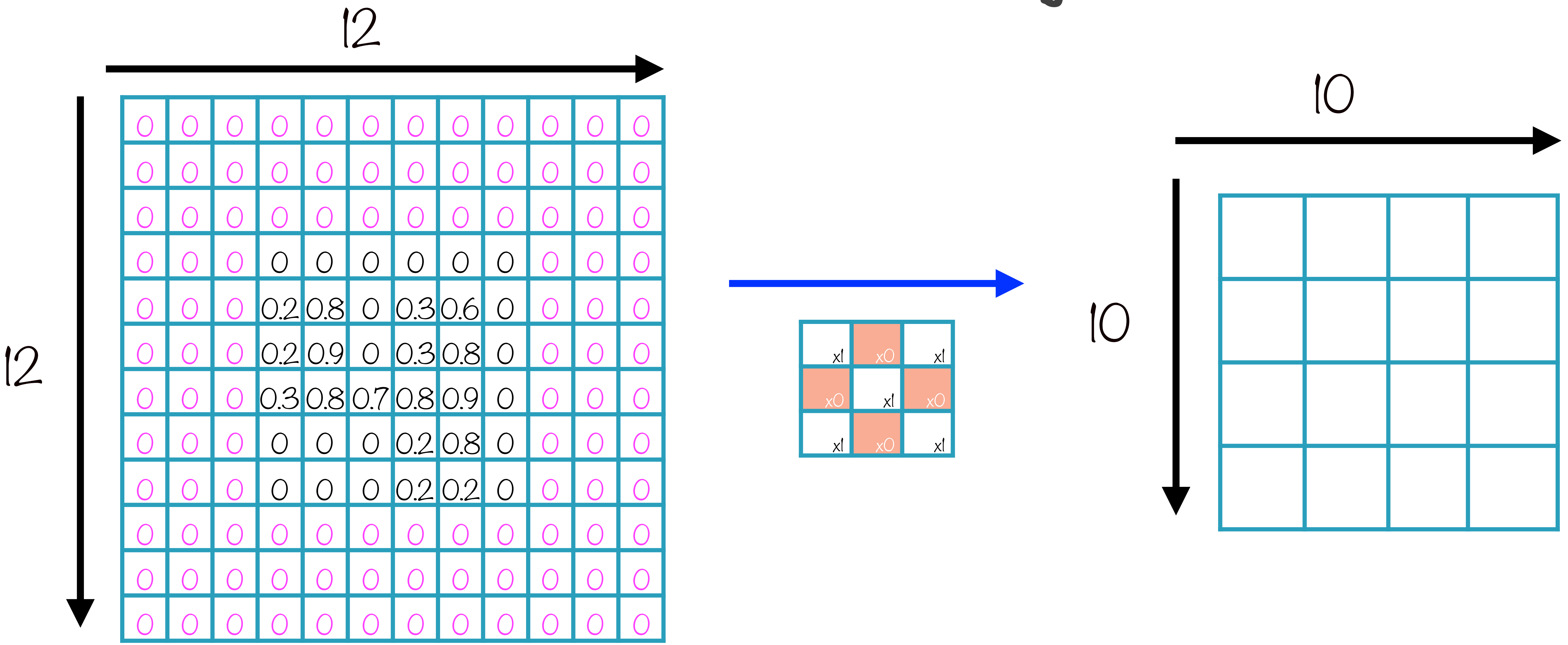
Without Zero Padding



Zero Padding



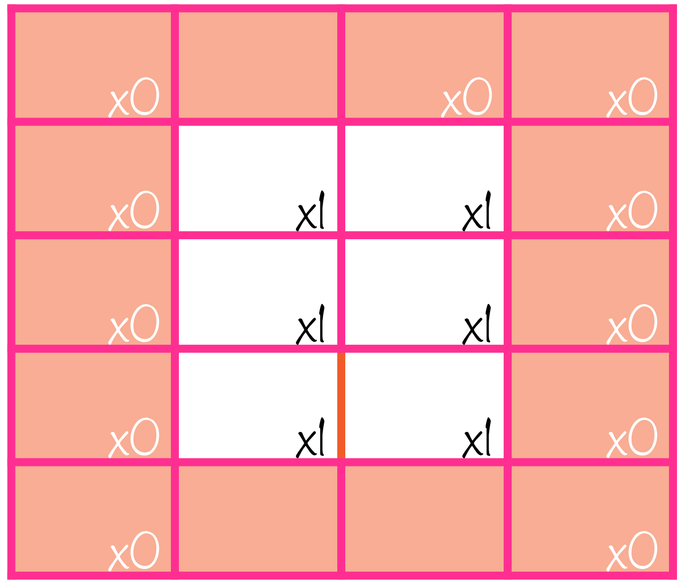
Zero Padding



Matrix

Convolution Result

Zero Padding

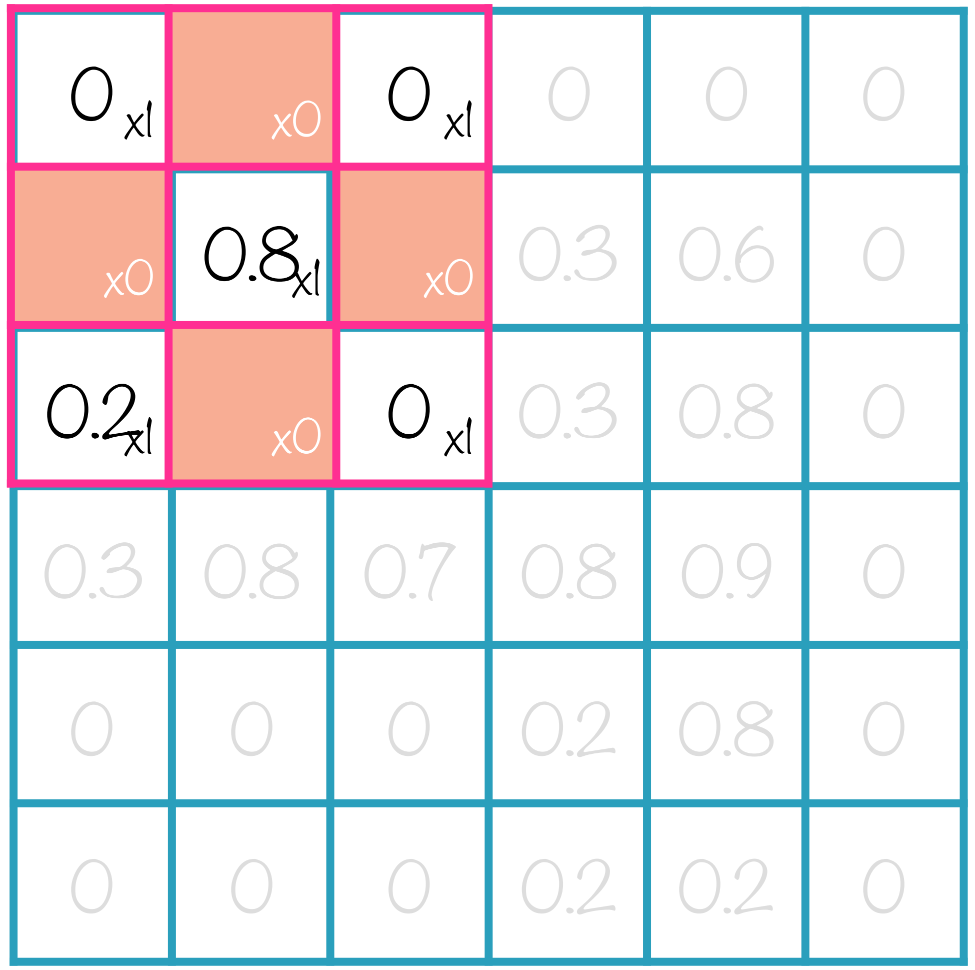


With zero-padding, every element of matrix will be passed into filter

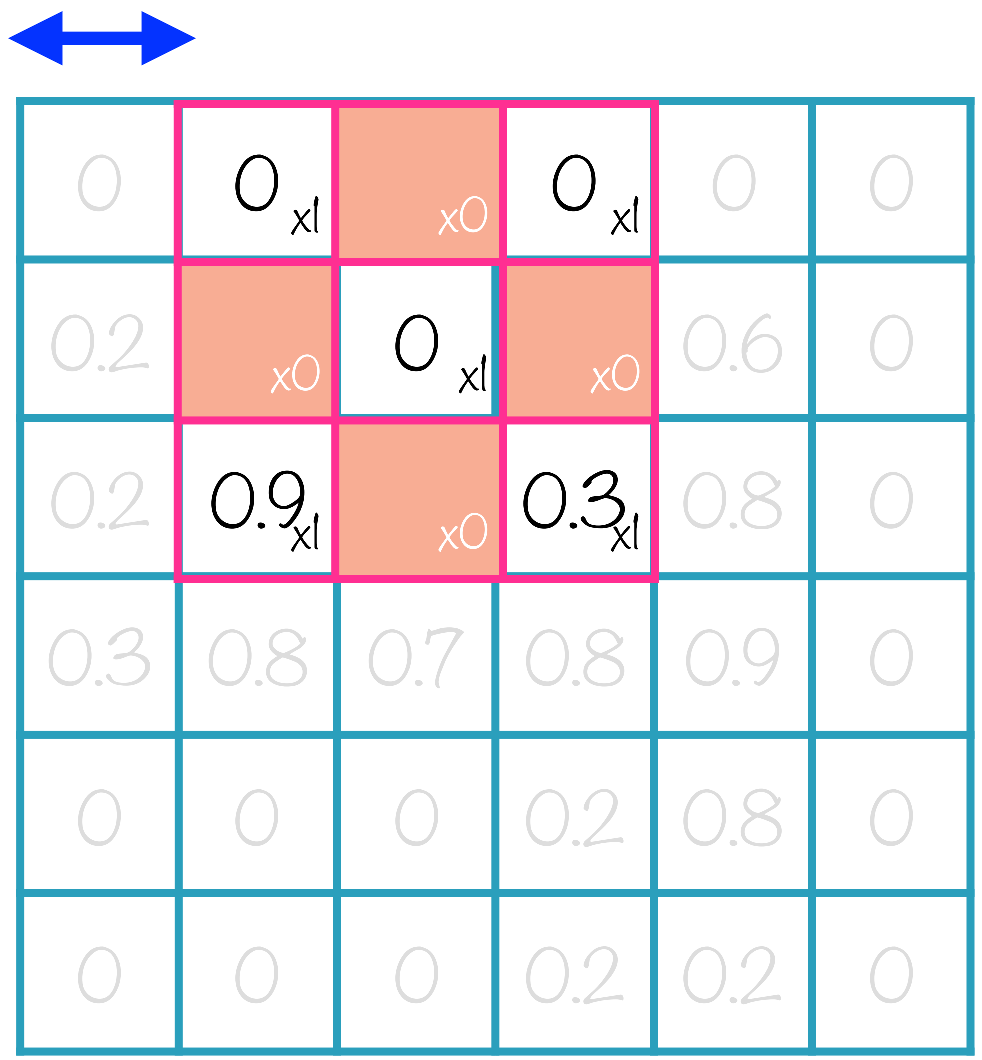
Can decide number of zero columns to pad with

Use to get output larger than input

Stride Size

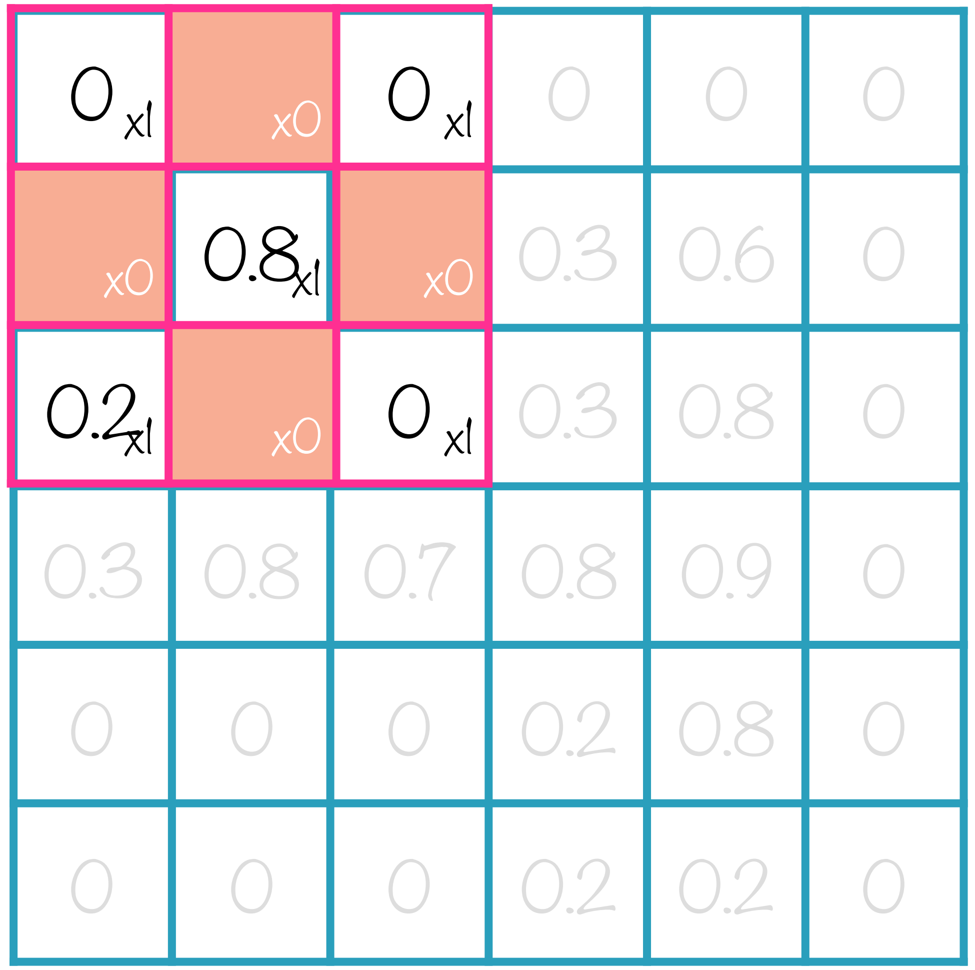

0 x1
x0
0 x1
0
0
0
x0
0.8 x1
x0
0.3
0.6
0
0.2 x1
x0
0 x1
0.3
0.8
0
0.3
0.8
0.7
0.8
0.9
0
0
0
0
0.2
0.8
0
0
0
0
0.2
0.2
0

Stride Size



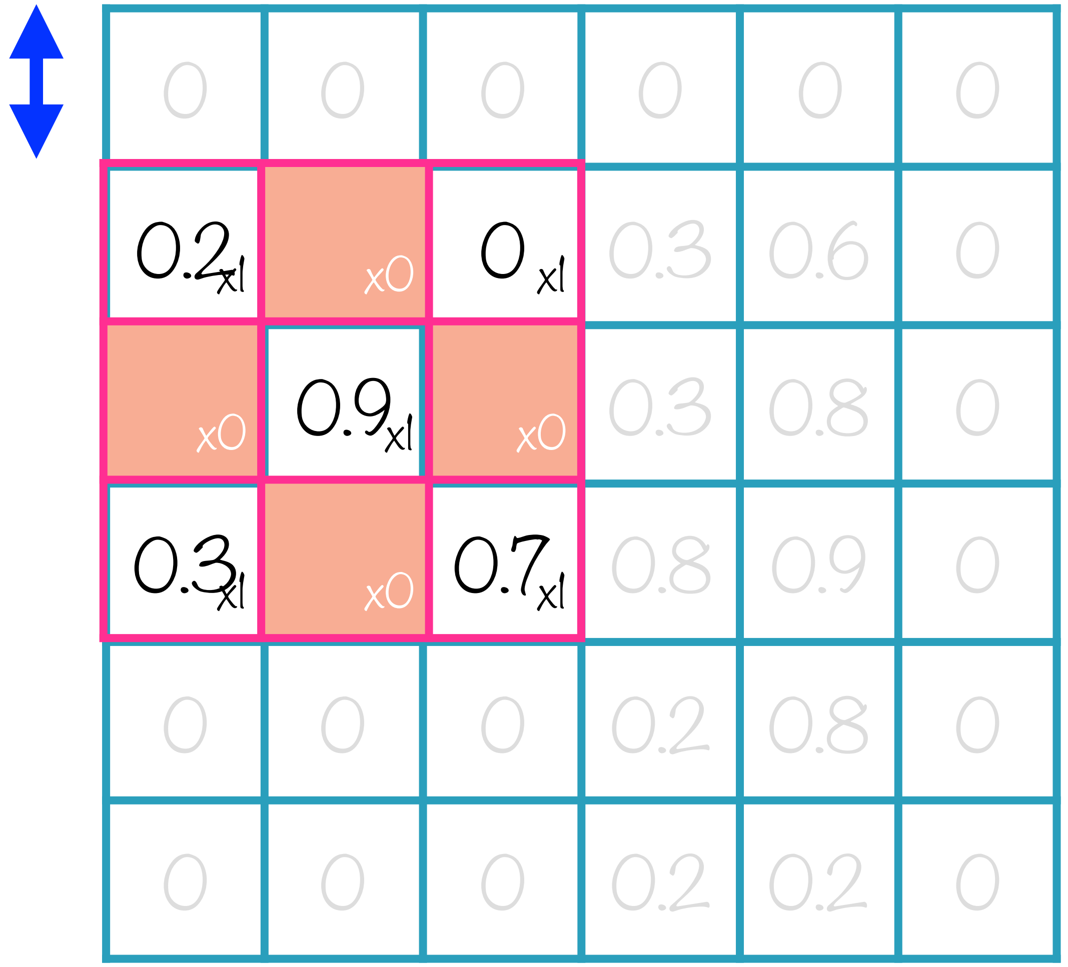
Horizontal stride of 1

Stride Size



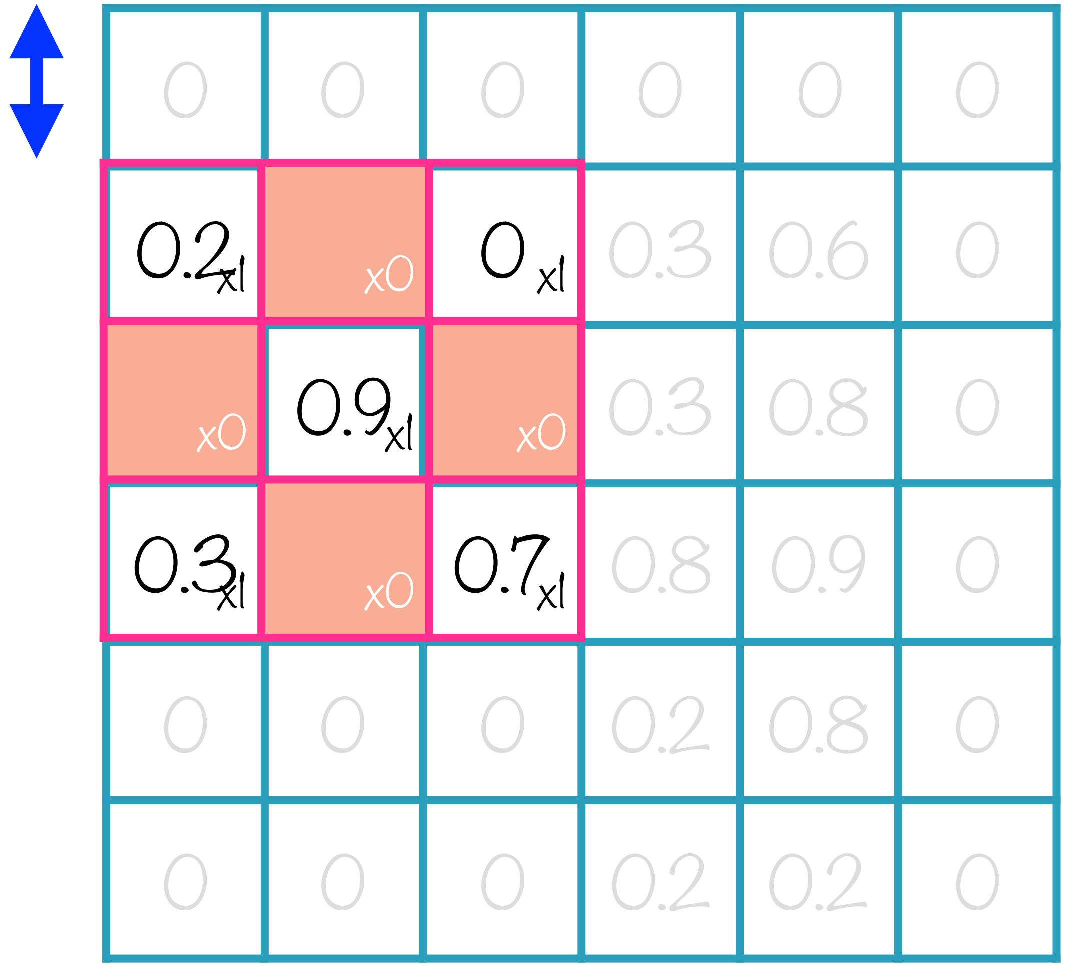
0 x1
x0
0 x1
0
0
0
x0
0.8 x1
x0
0.3
0.6
0
0.2 x1
x0
0 x1
0.3
0.8
0
0.3
0.8
0.7
0.8
0.9
0
0
0
0
0.2
0.8
0
0
0
0
0.2
0.2
0

Stride Size



Vertical stride of 1

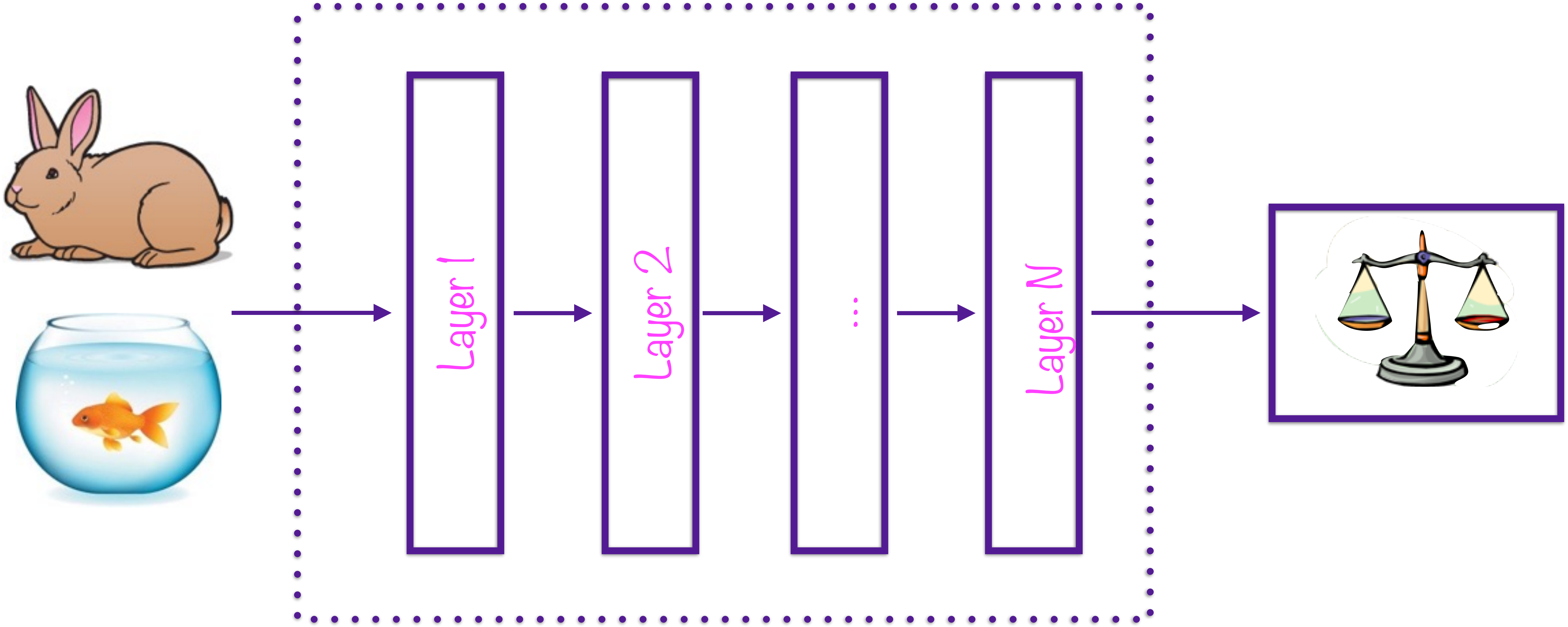
Stride Size



Stride size is an important hyper parameter in CNNs

Convolutional Neural Networks

Neural Networks for Image Classification

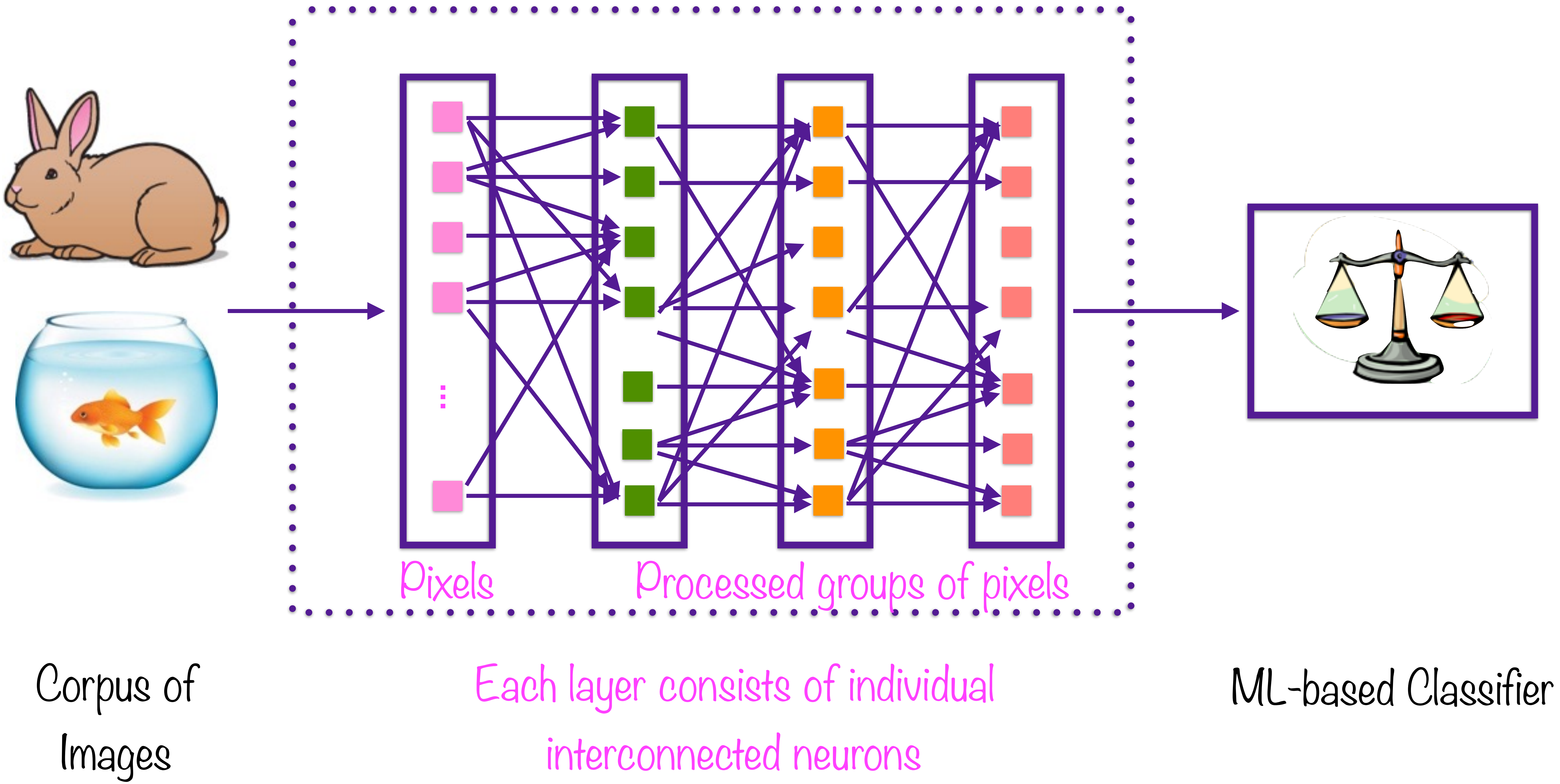


Corpus of Images

Layers in a neural network

ML-based Classifier

Neural Networks for Image Classification



Parameter Explosion

Consider a 100 x 100 pixel image (10,000 pixels)

If first layer = 10,000 neurons

Interconnections ~ O(10,000 * 10,000)

100 million parameters to train neural network!

Parameter Explosion

Dense, fully connected neural networks can't cope

Convolutional neural networks to the rescue

CNNs Introduced

Eye perceives visual stimulus in 2D visual field

Eye sends 2D image to visual cortex

Visual cortex adds depth perception

Individual neurons in cortex focus on small field

"Local receptive field"

CNNs Introduced

CNNs perform spectacularly well at many tasks

Particularly at image recognition

Dramatically fewer parameters than DNN with similar performance

Inspirations for CNNs

Two Dimensions

Data comes in expressed in 2D

Local Receptive Fields

Neurons focus on narrow portions

CNN Layers

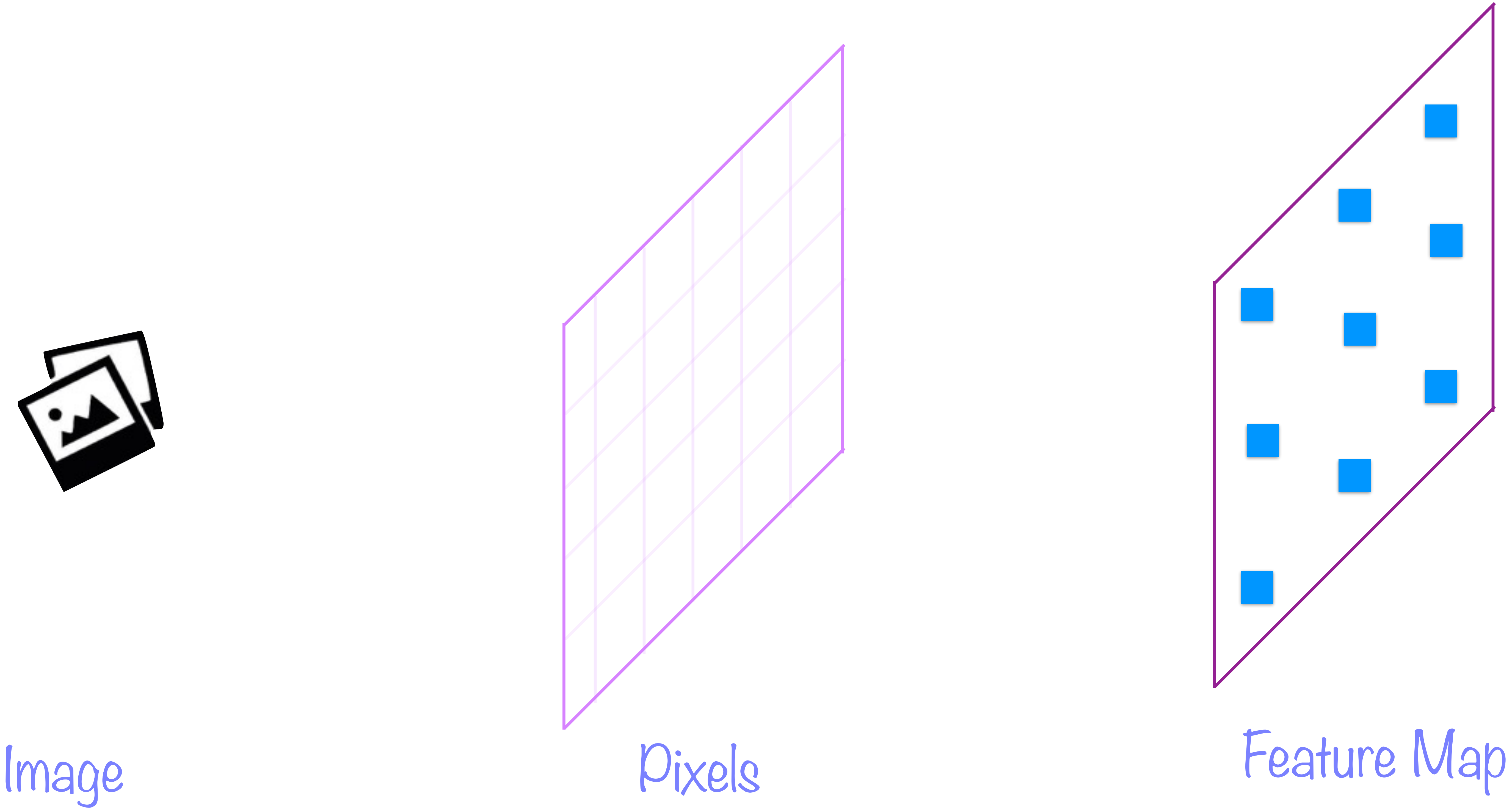
Convolution layers - zoom in on specific bits of input

Successive layers aggregate inputs into higher level features

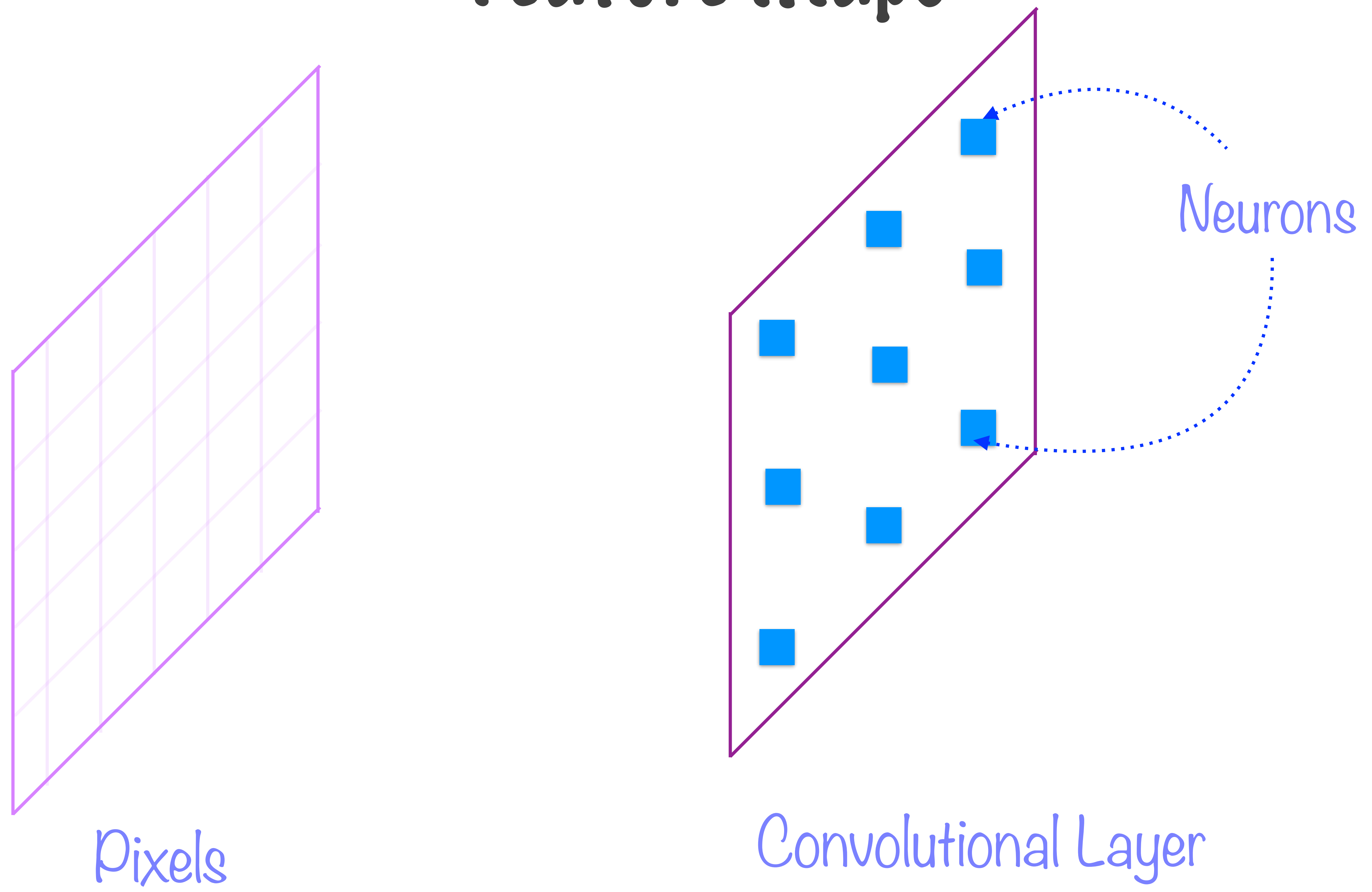
Pixels >> Lines >> Contours/Edges >> Object

Convolutional Layers

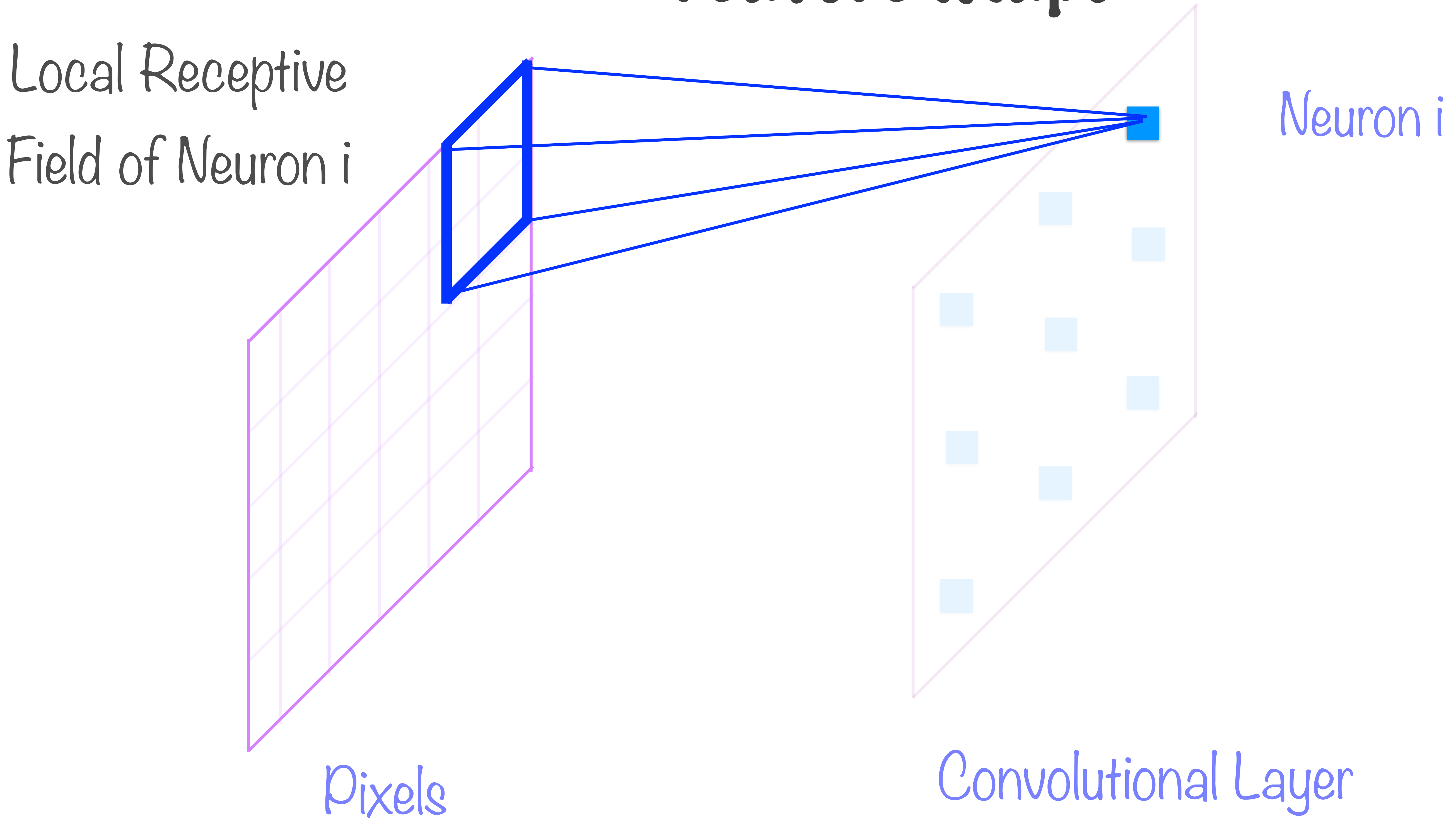
Feature Maps



Feature Maps

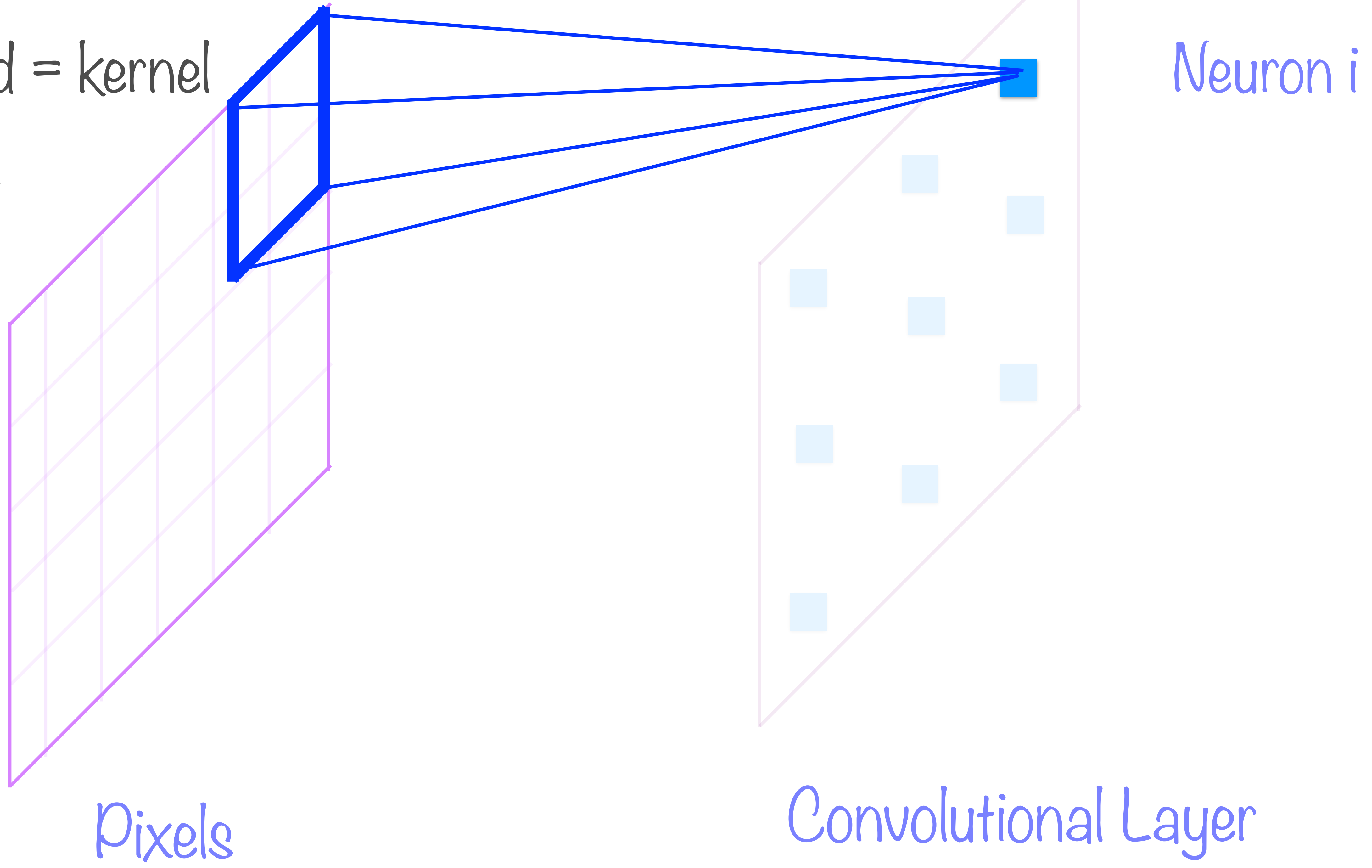


Feature Maps

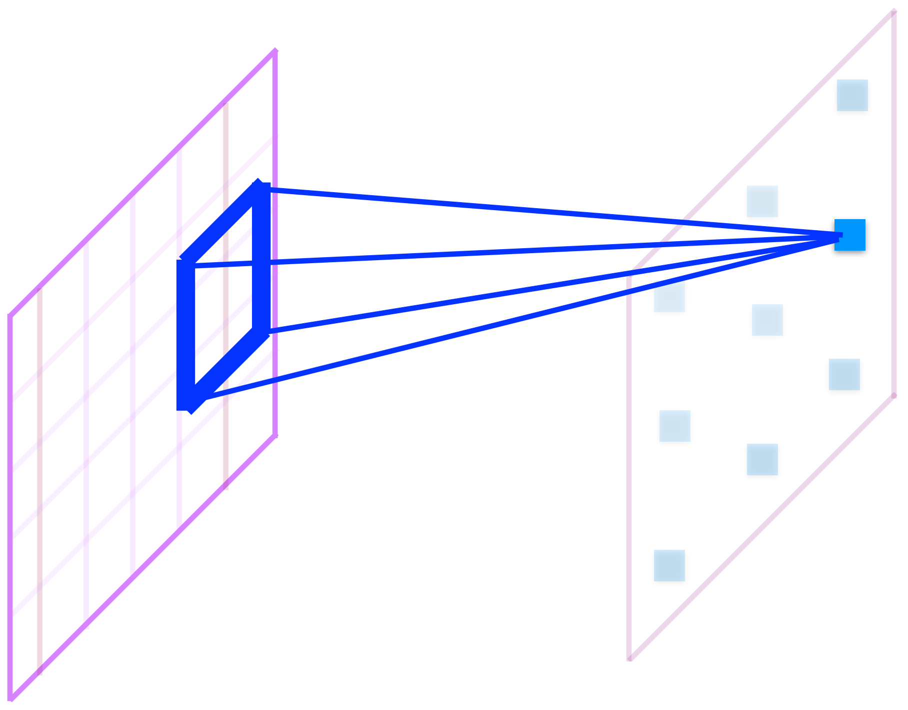


Feature Maps

Number of neurons in receptive field = kernel size



Kernel Size

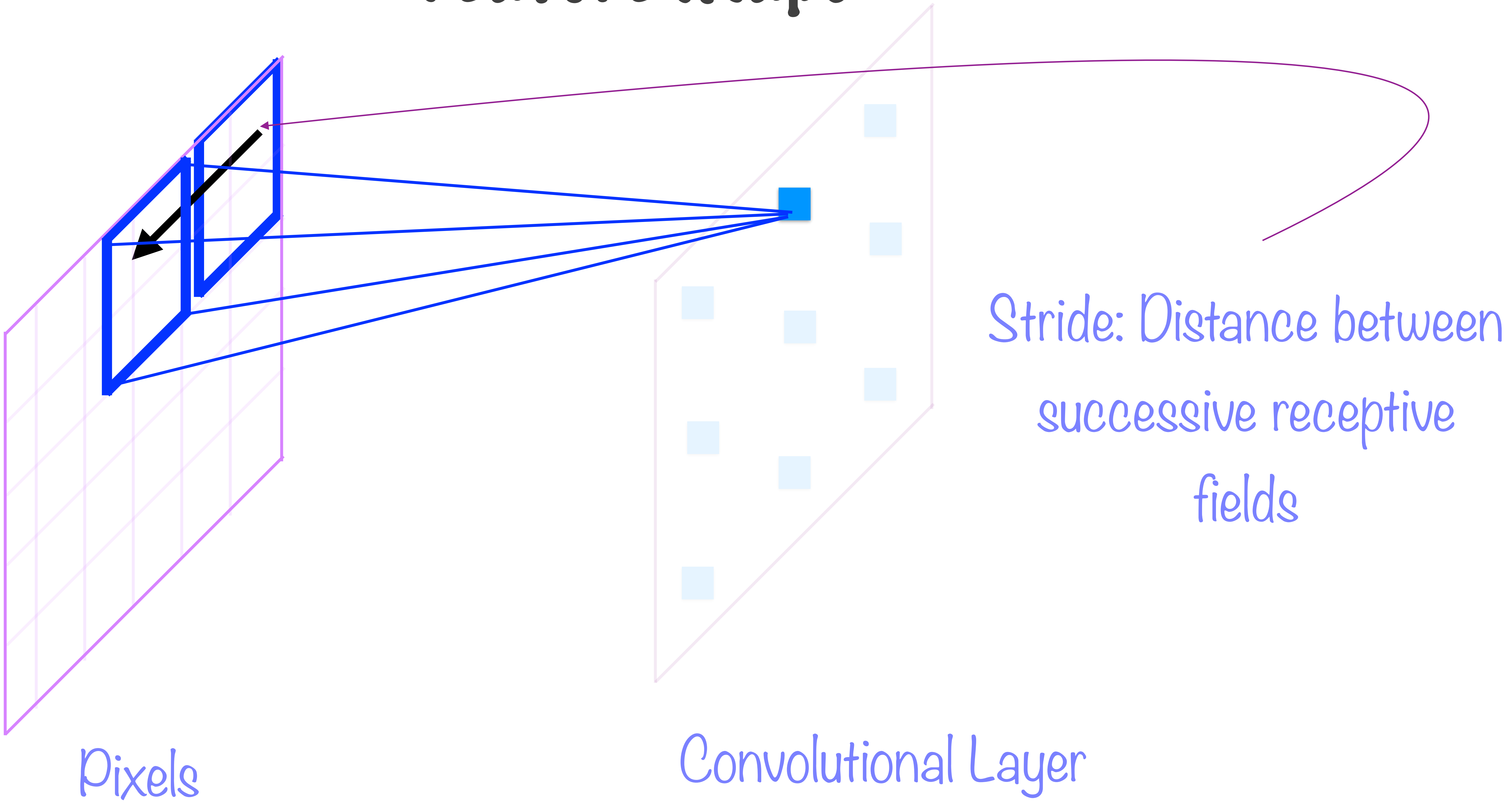


The convolutional kernel size is usually expressed in terms of width and height of receptive area

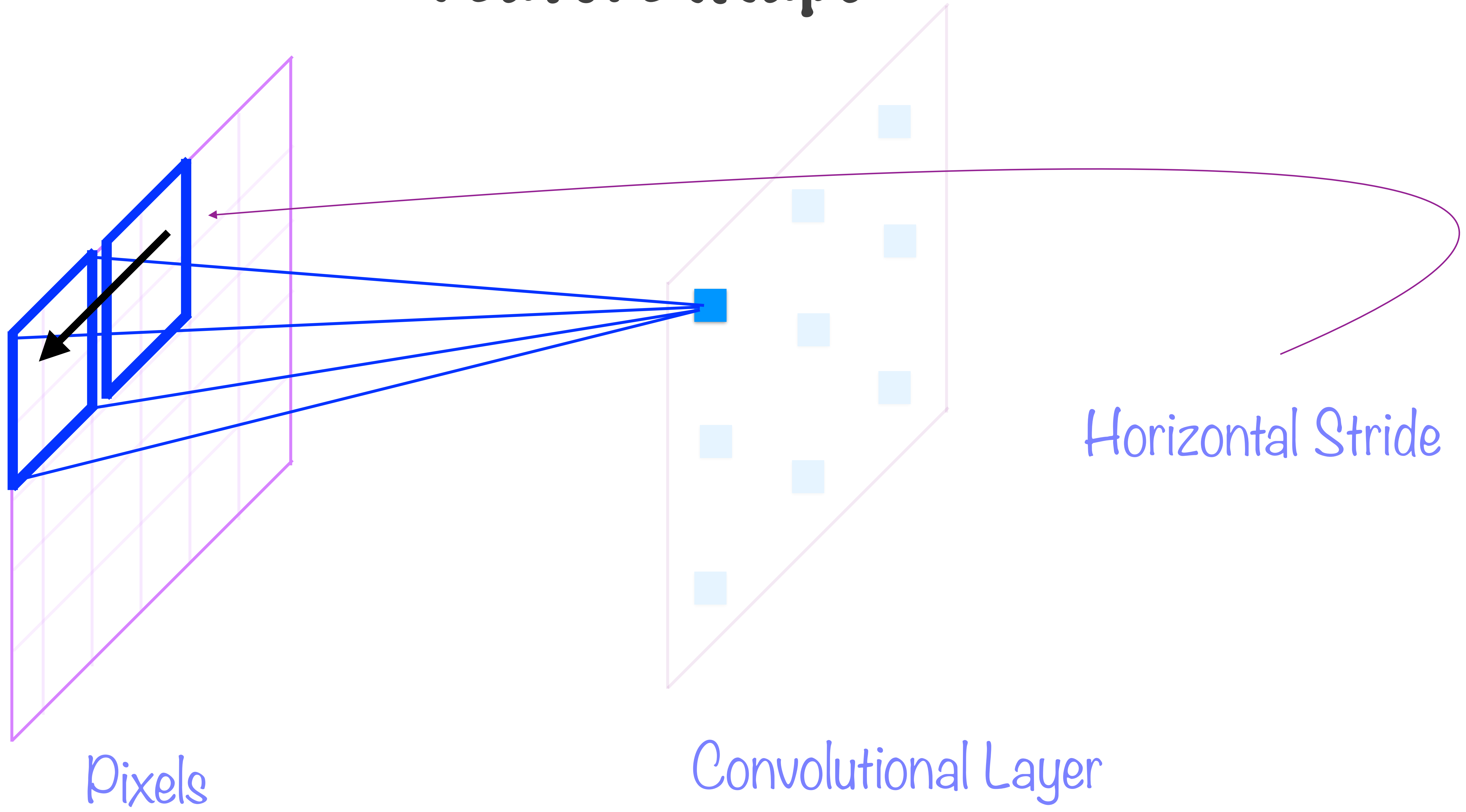
Use small convolutional kernels, more efficient

Stacking 2 3x3 kernels is preferable to 1 9x9 kernel

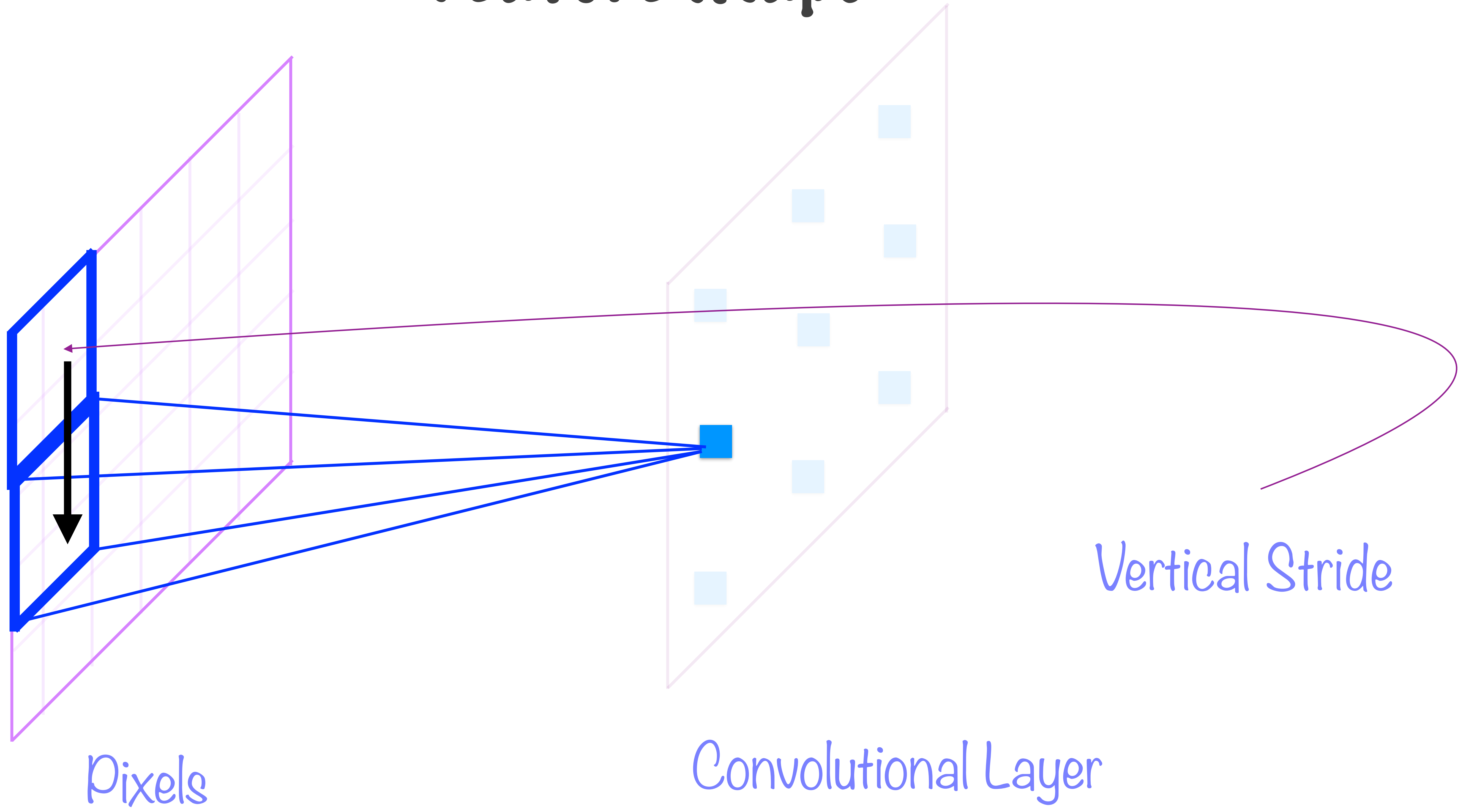
Feature Maps



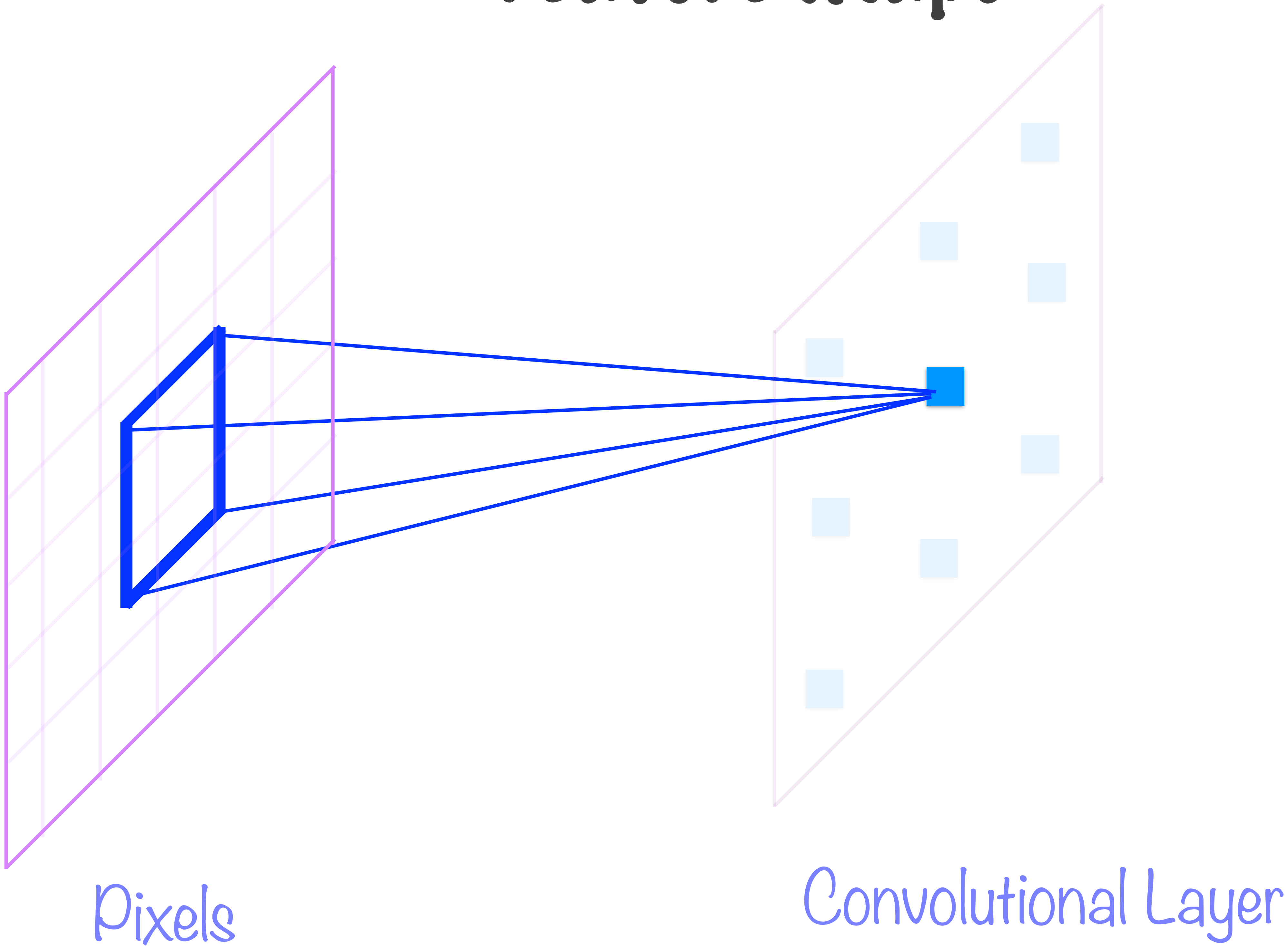
Feature Maps



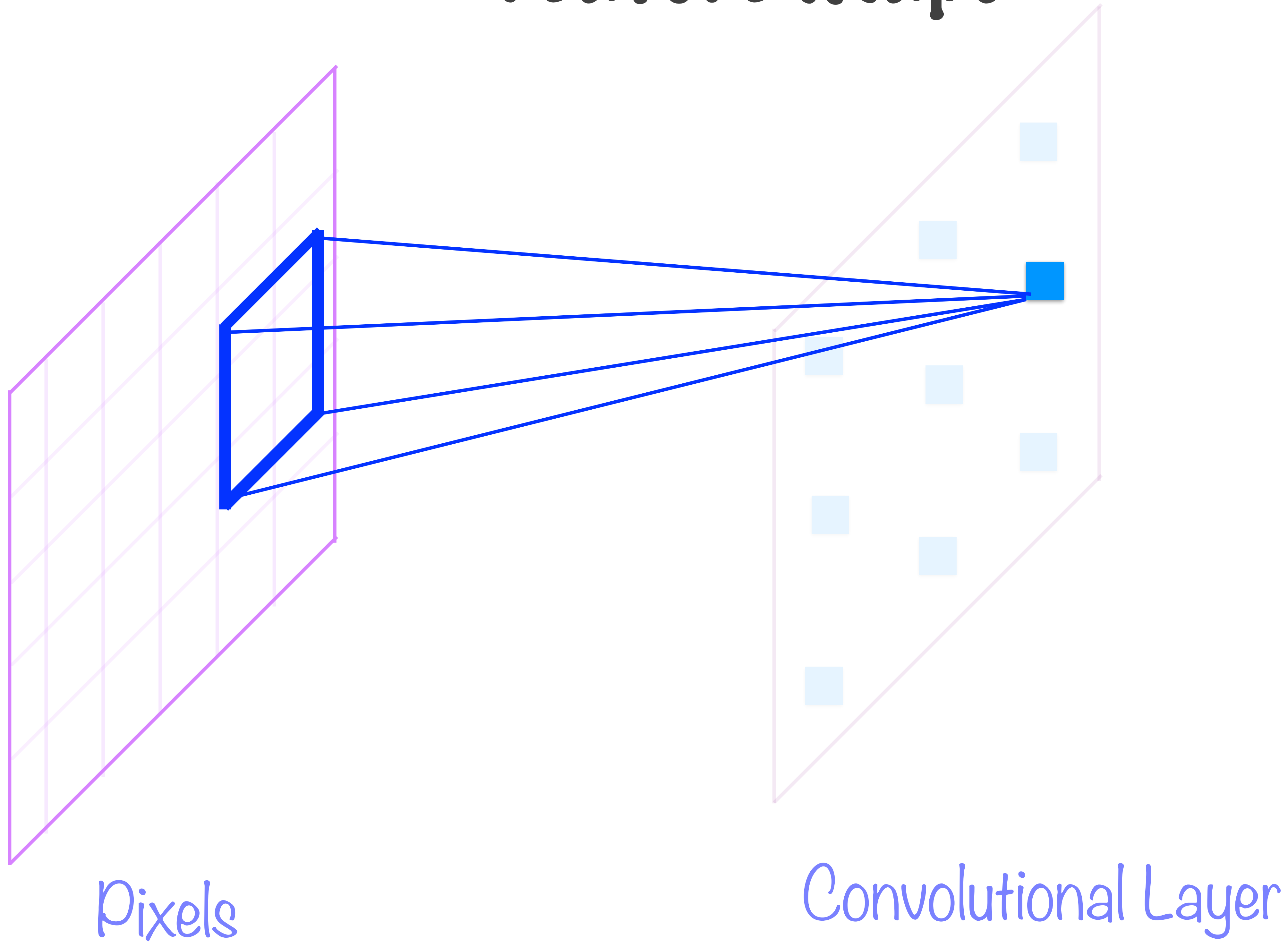
Feature Maps



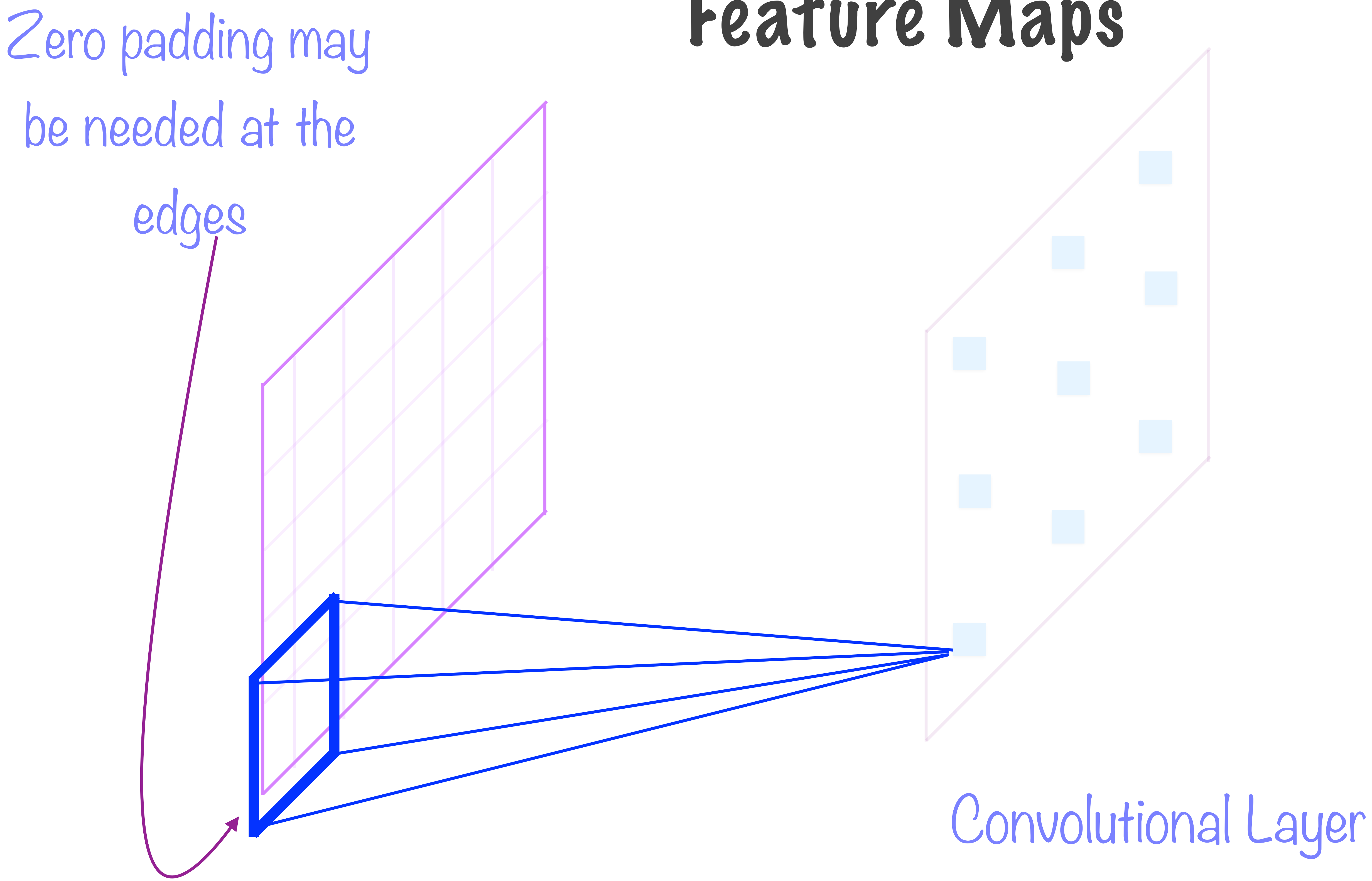
Feature Maps



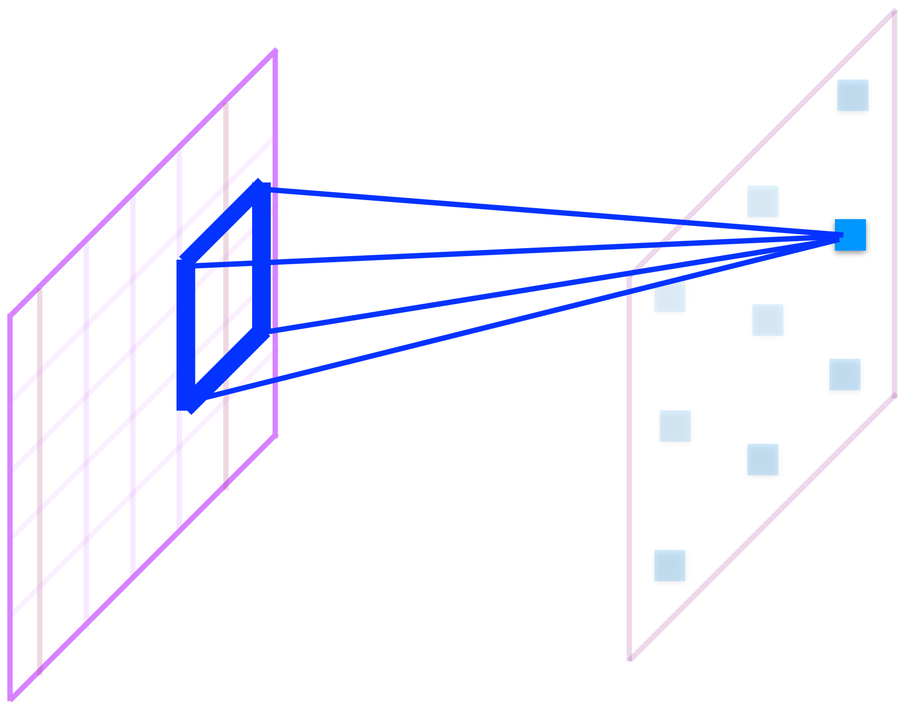
Feature Maps



Feature Maps



Feature Maps

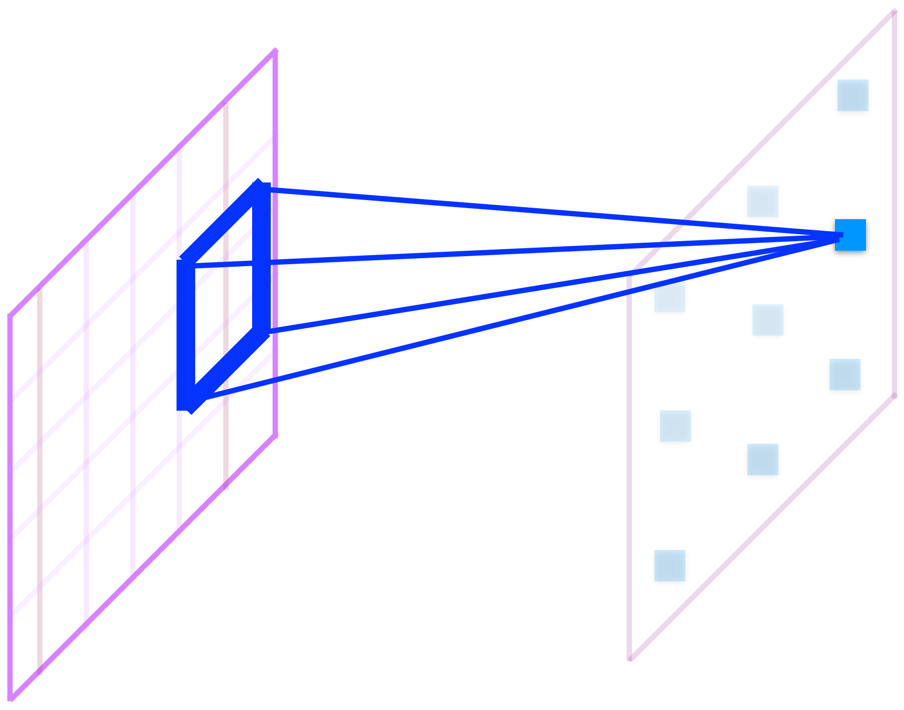


All neurons in a feature map have the same weights and biases

Two big advantages over DNNs

- Dramatically fewer parameters to train
- CNN can recognise feature patterns independent of location

Feature Maps

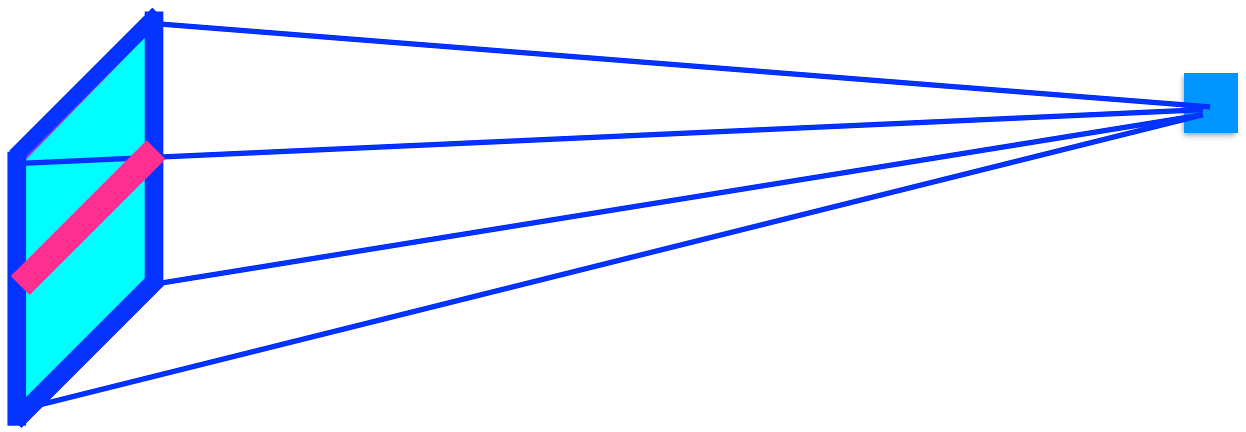


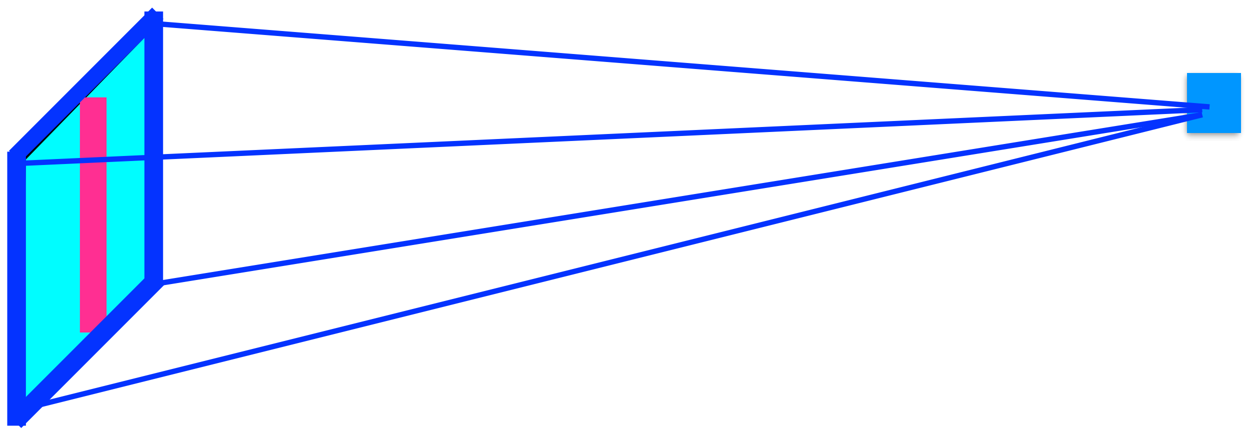
The parameters of all neurons in a feature map are collectively called the filter

Why filter?

Because weights highlight (filter) specific patterns from the input pixels

Filters





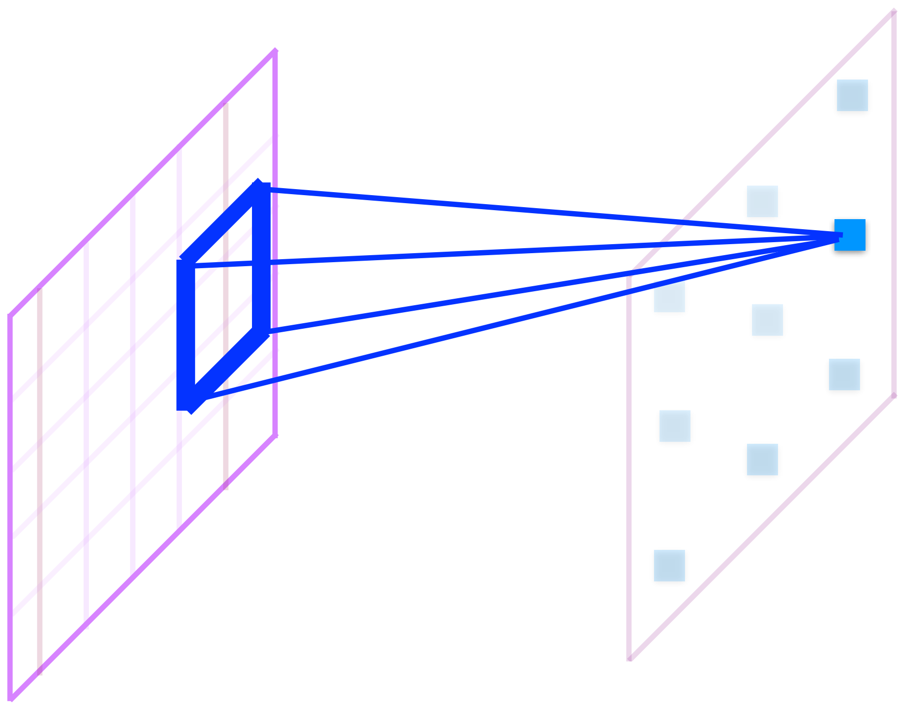
Horizontal Filter

Neuron will detect horizontal lines in input

Vertical Filter

Neuron will detect vertical lines in input

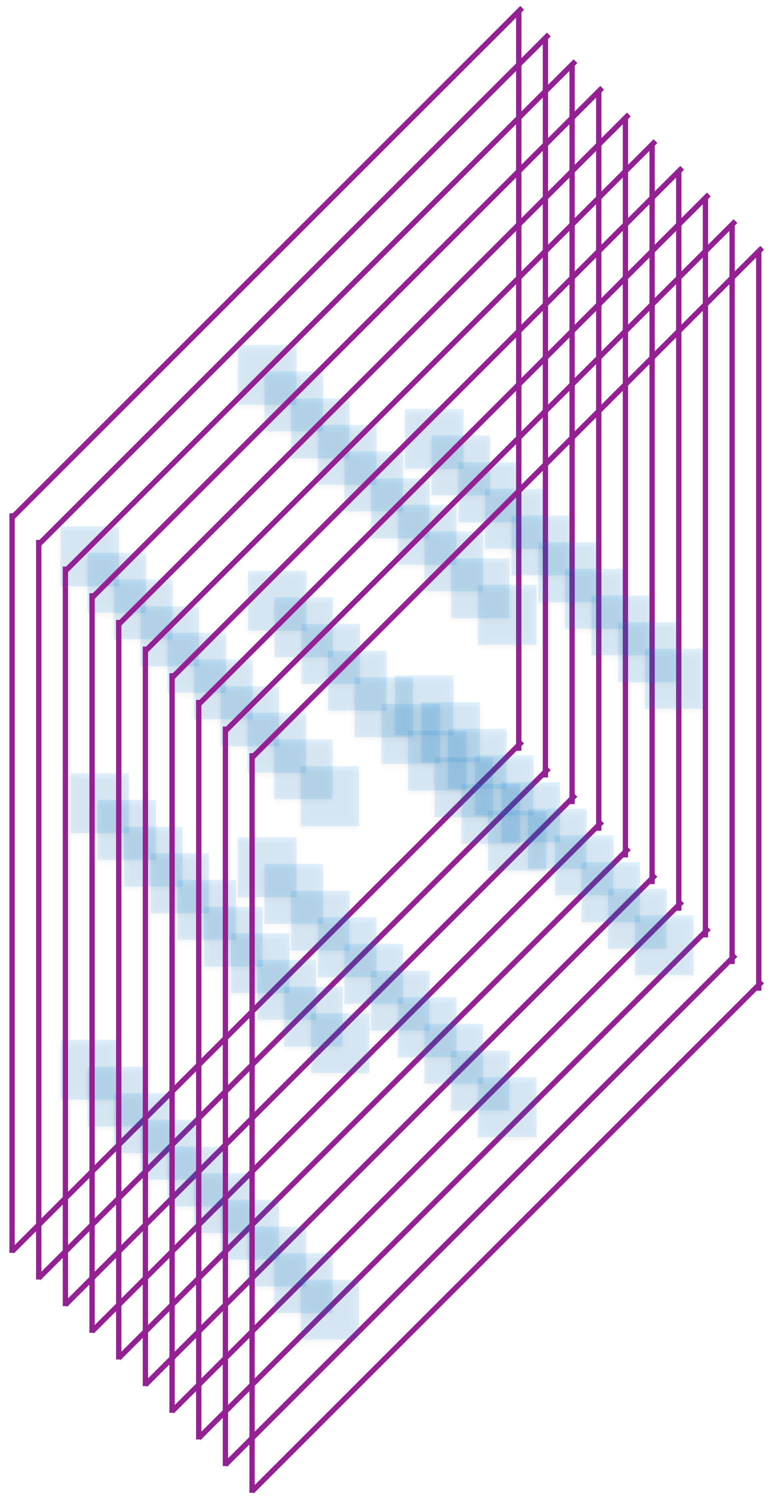
Feature Maps



Notice also that neurons are not connected to all pixels

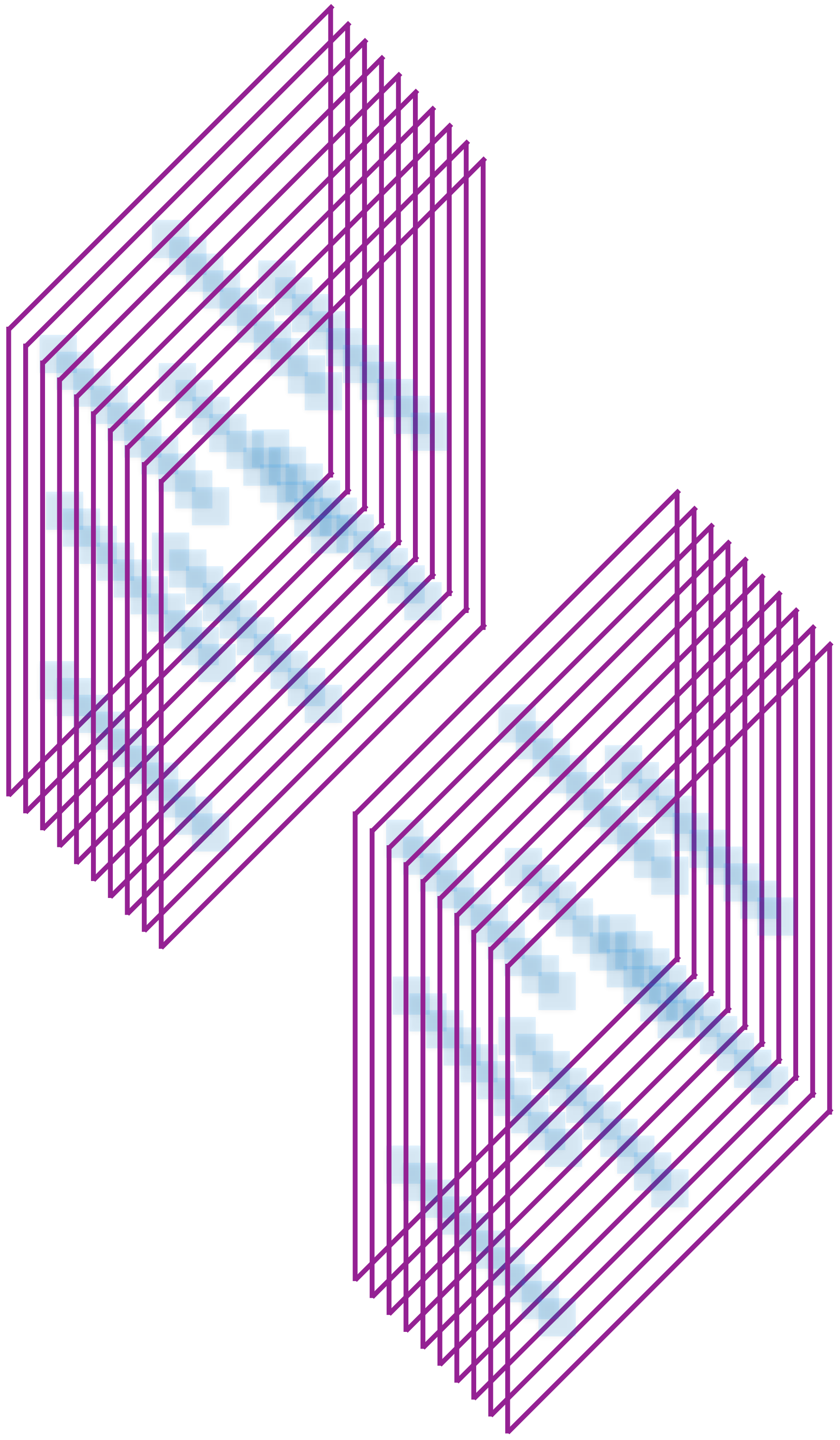
CNNs are sparse neural networks

Convolutional Layer



Each convolutional layer consists of several feature maps of equal sizes

The different feature maps have different parameters



Convolutional Layer

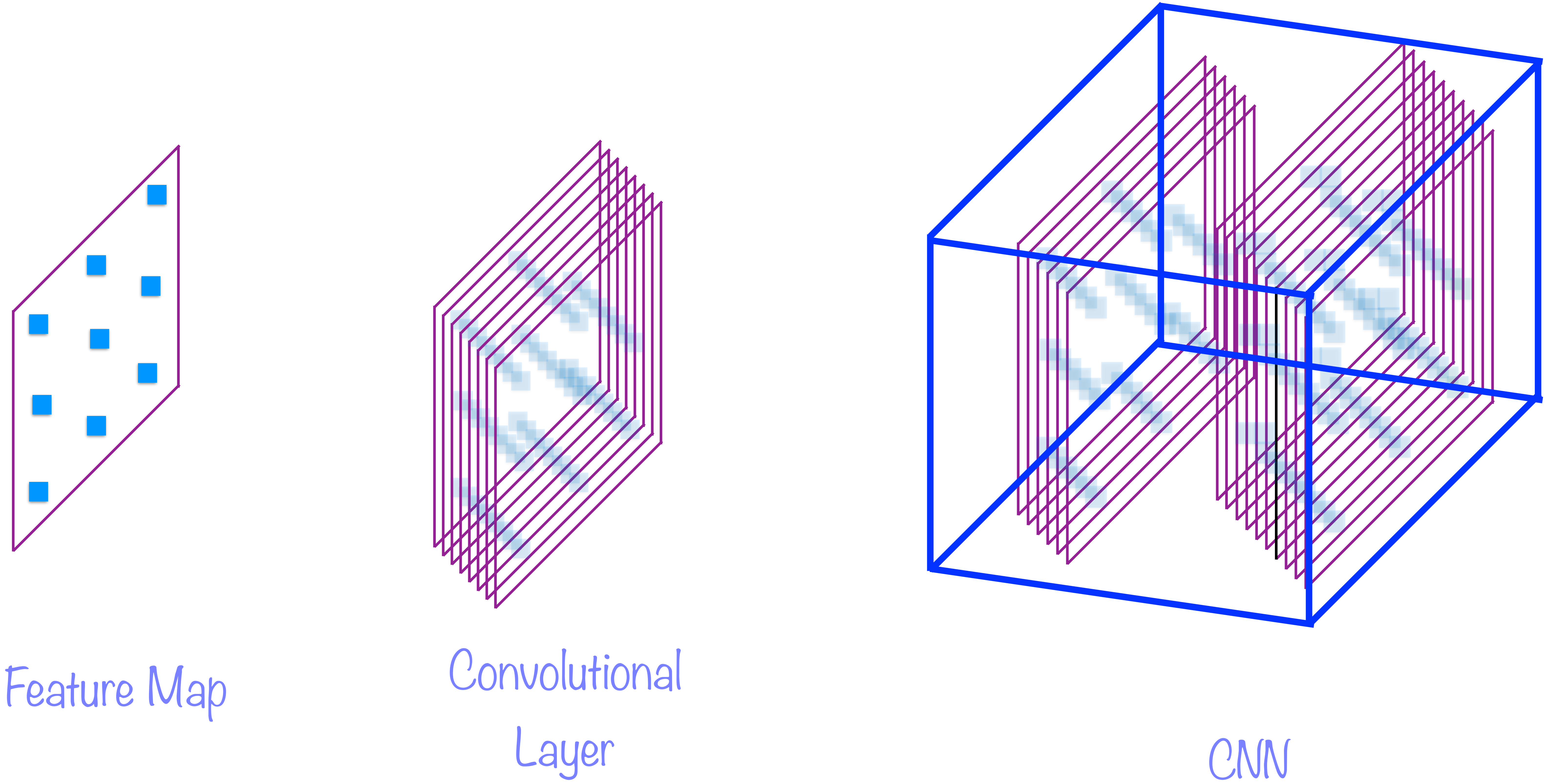
Each neuron's receptive field includes the feature maps of all previous layers

This is how aggregated features are picked up

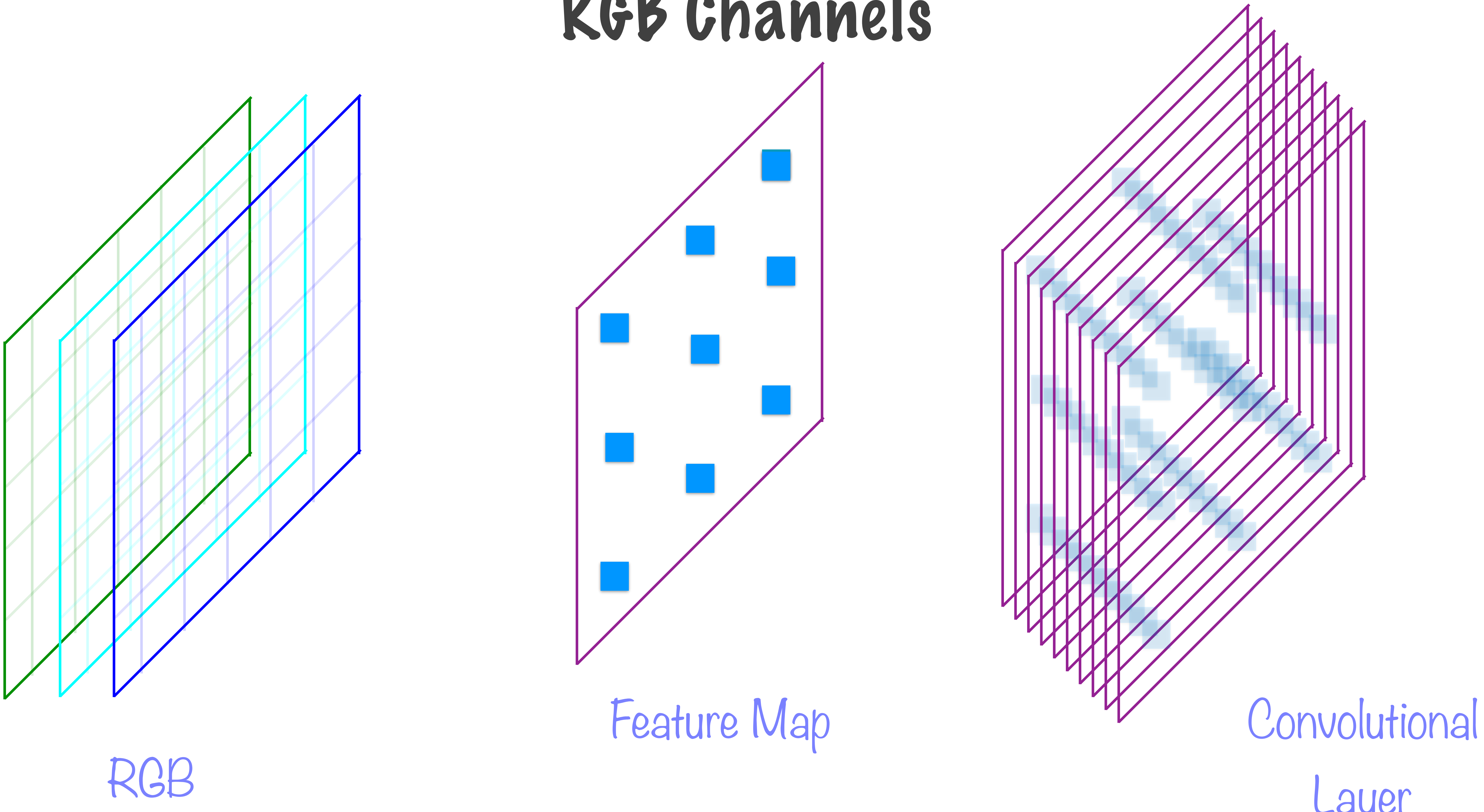
The CNN as a whole consists of multiple convolutional (and pooling) layers

More on pooling layers in a bit

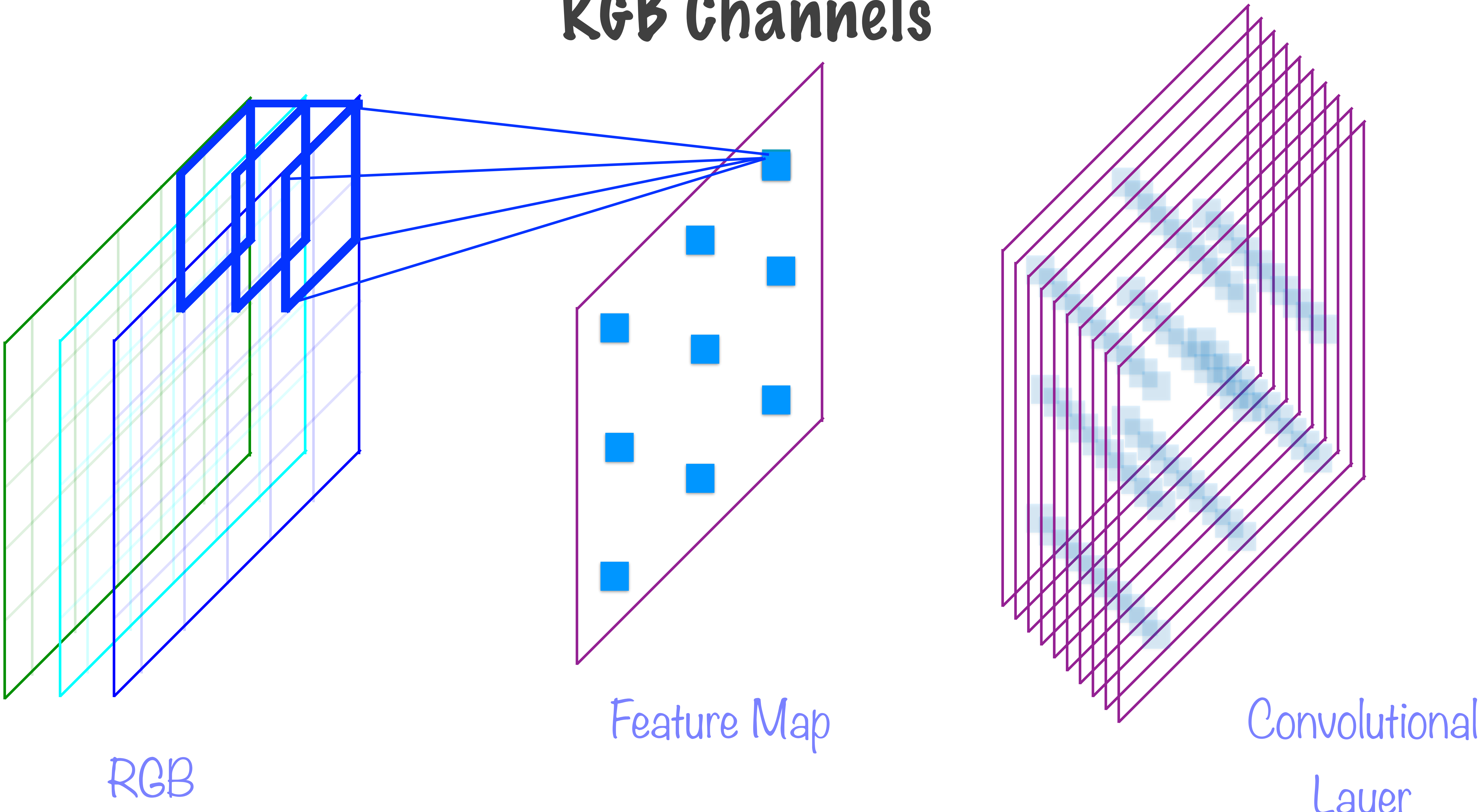
CNNs



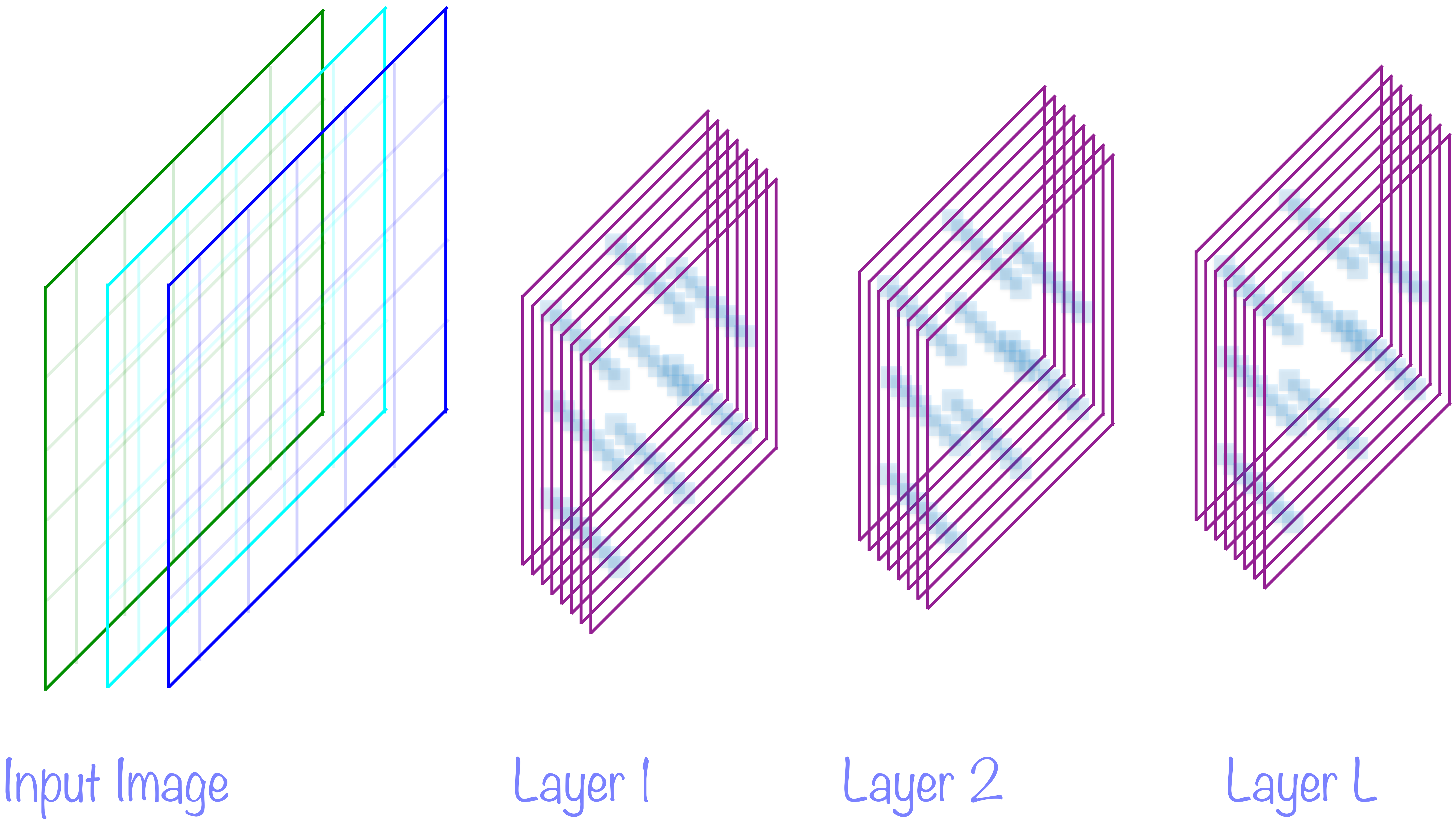
RGB Channels



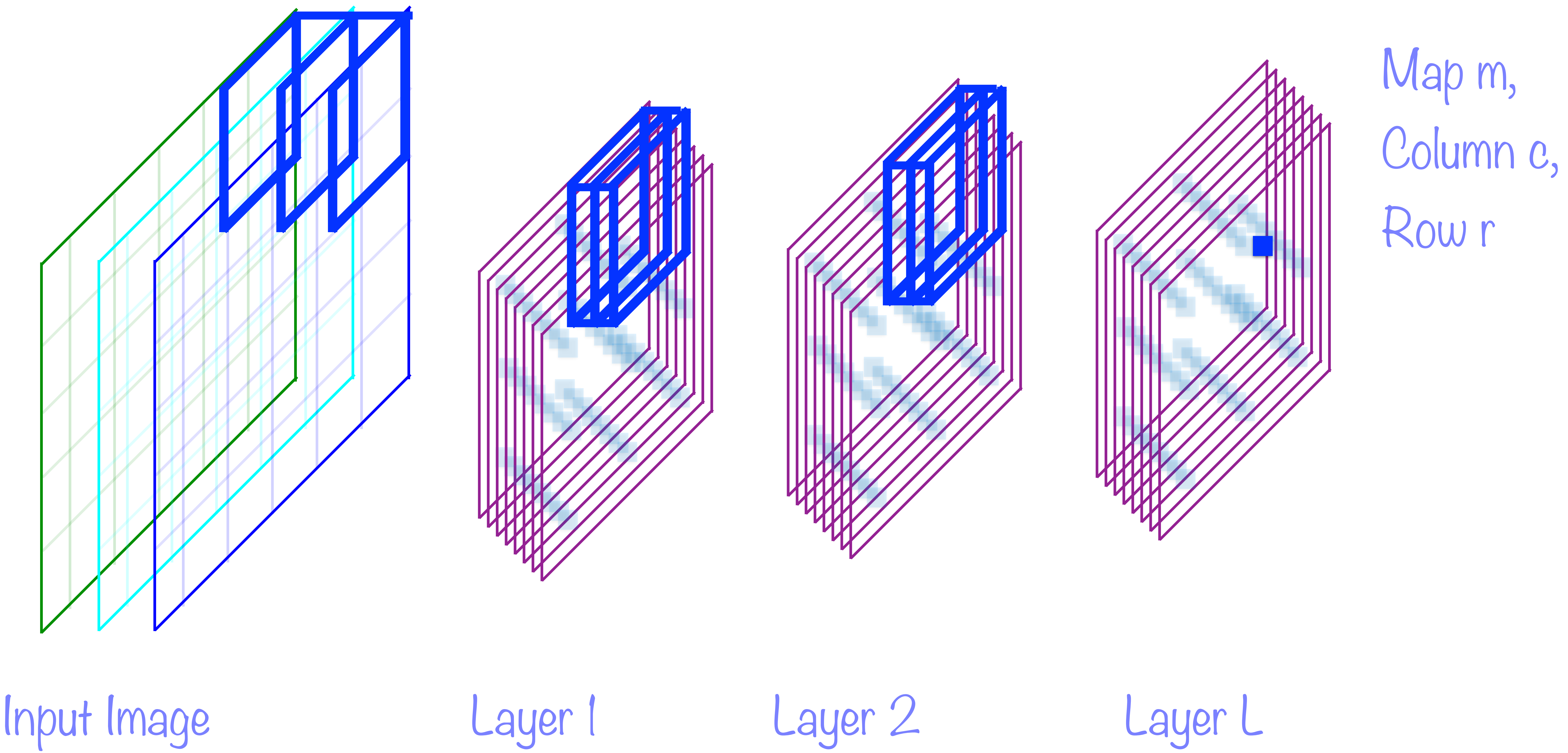
RGB Channels



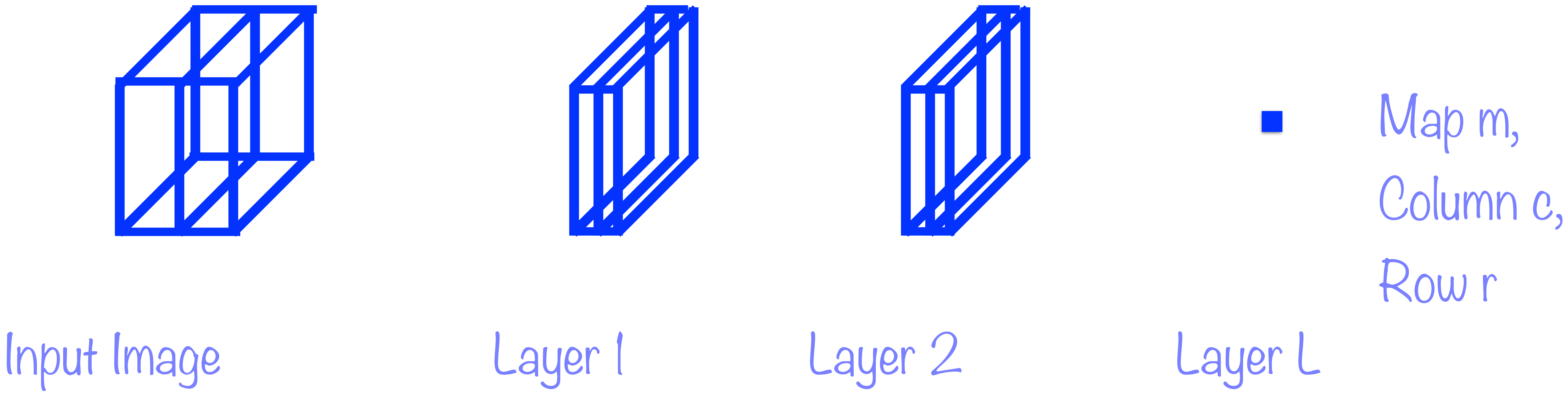
Output of a Convolution Layer Neuron



Output of a Convolution Layer Neuron



Output of a Convolution Layer Neuron



Neuron output depends on corresponding* neurons from each preceding layer
(*corresponding: same receptive field and feature maps, different layers)

Pooling Layers

Two Kinds of Layers in CNNs

Convolutional

Local receptive field

Pooling

Subsampling of inputs

Convolution

6

6

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

Matrix

x1	x0	x1
x0	x1	x0
x1	x0	x1

4

4

1	1.2	1.1	0.9
1.9	2.7	2.5	1.9
1.0	2.1	2.4	1.4
1.0	1.8	2.0	1.8

Convolution Result

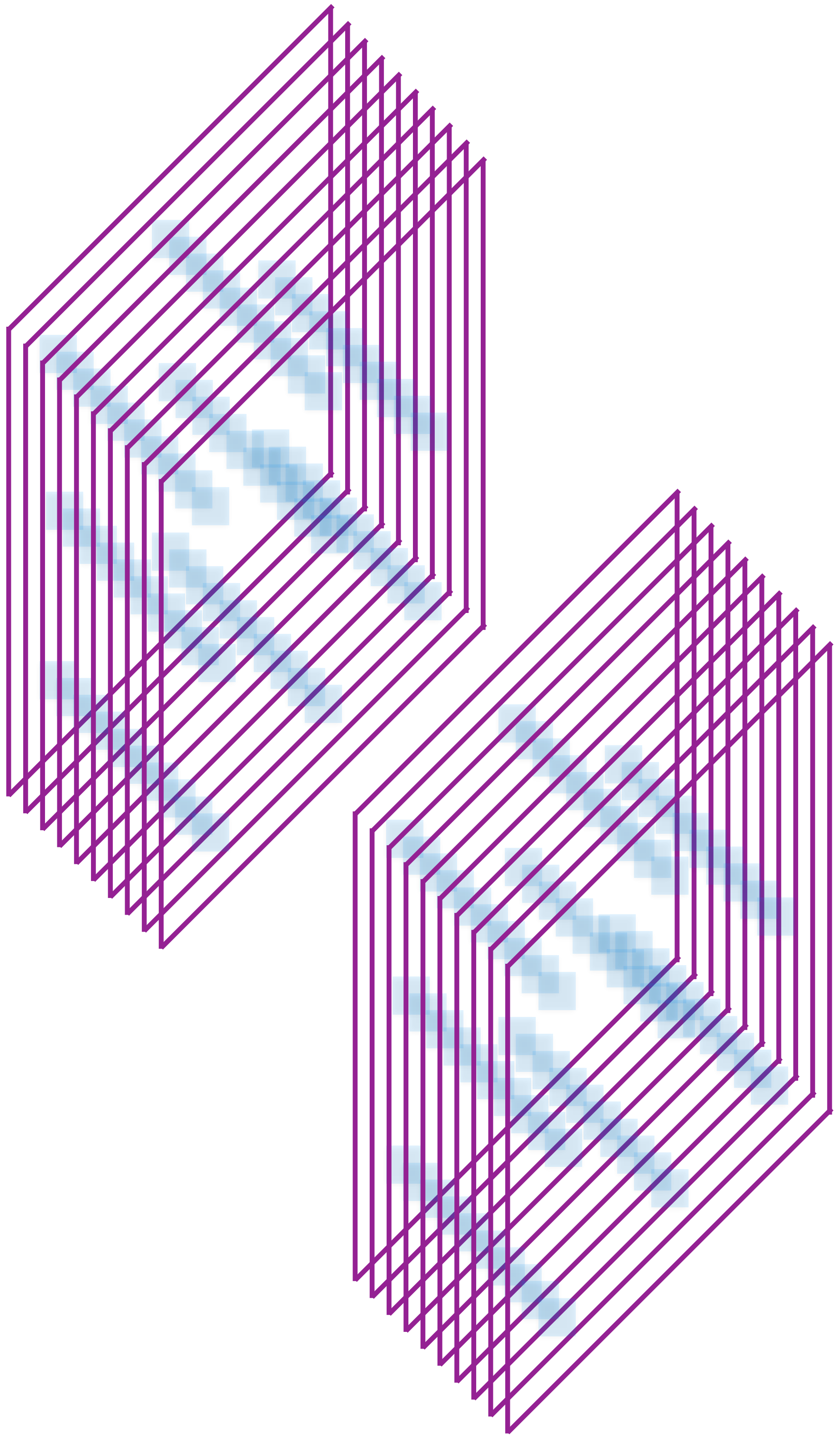
Two Kinds of Layers in CNNs

Convolutional

Local receptive field

Pooling

Subsampling of inputs



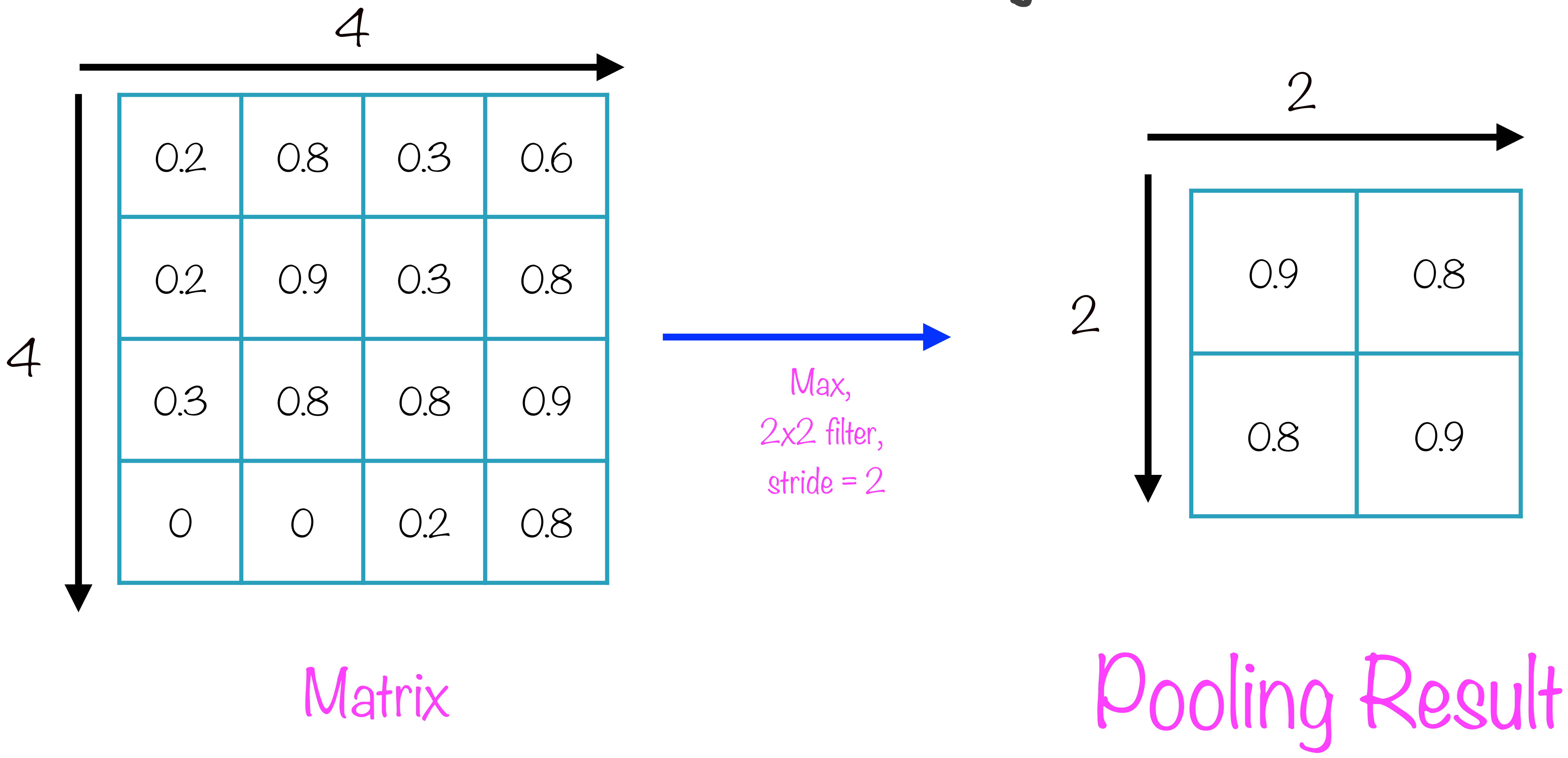
Pooling Layers

Neurons in a pooling layer have no weights or biases

A pooling neuron simply applies some aggregation function to all inputs

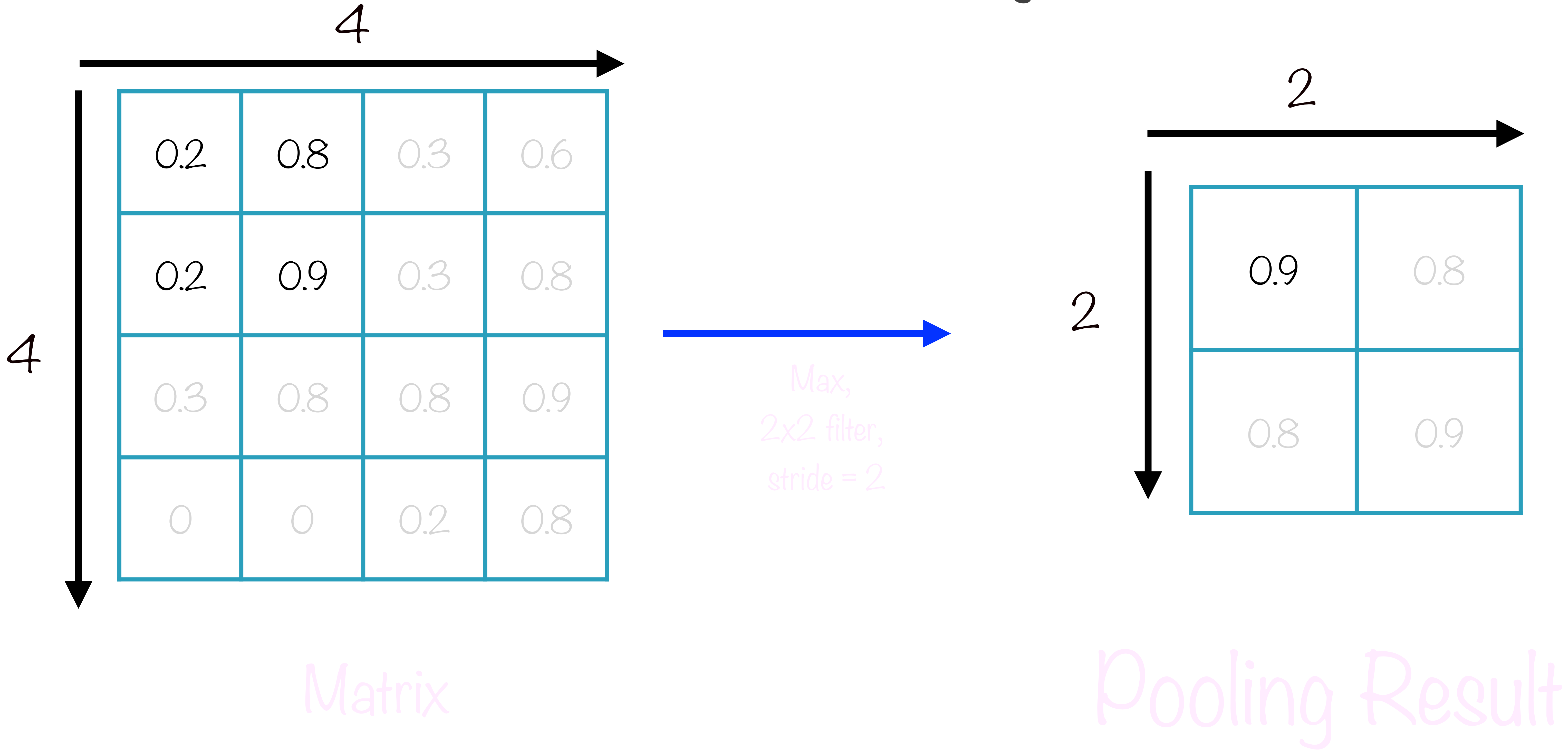
Max, sum, average...

Max Pooling



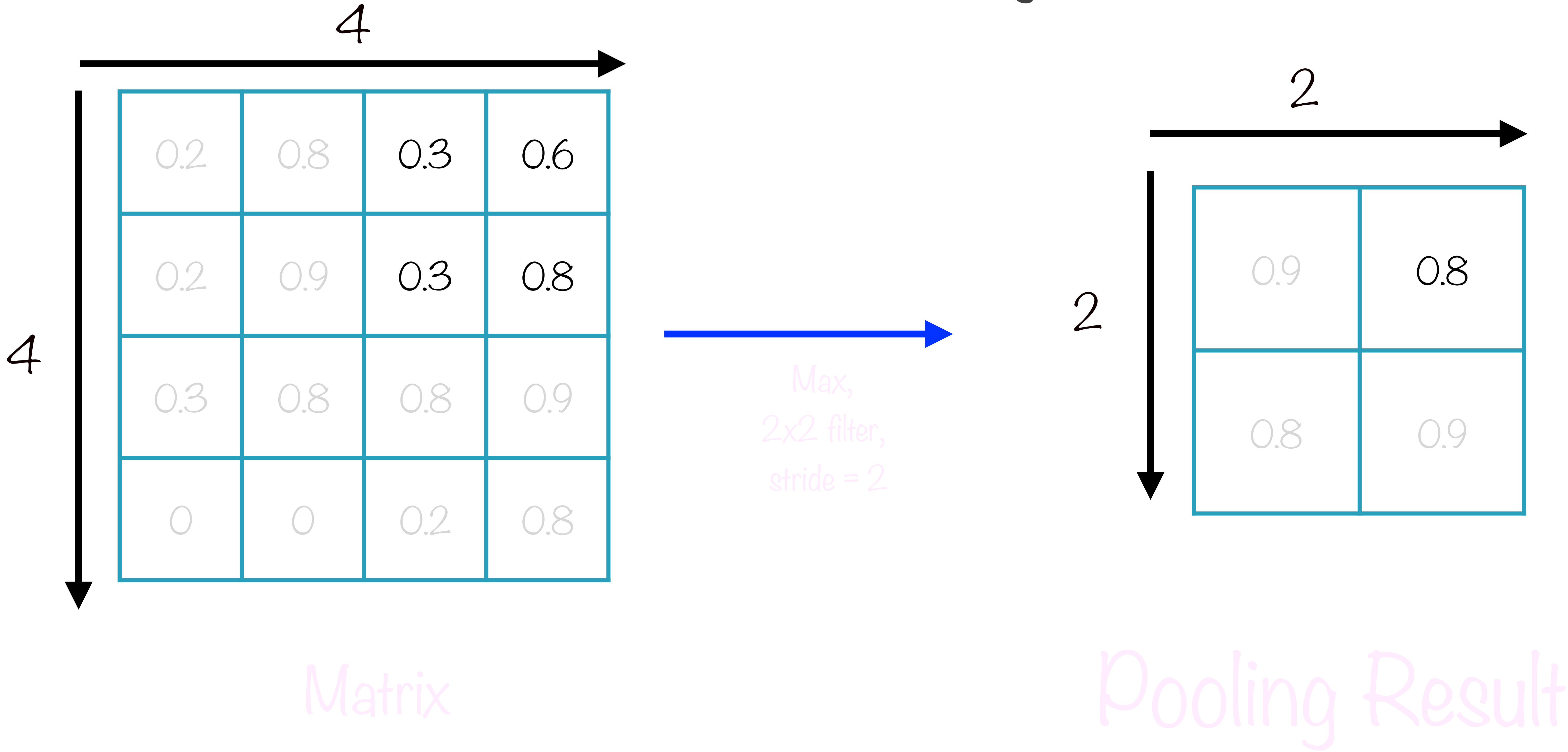
4
4
0.2 0.8 0.3 0.6
0.2 0.9 0.3 0.8
0.3 0.8 0.8 0.9
0 0 0.2 0.8
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Matrix
Pooling Result

Max Pooling

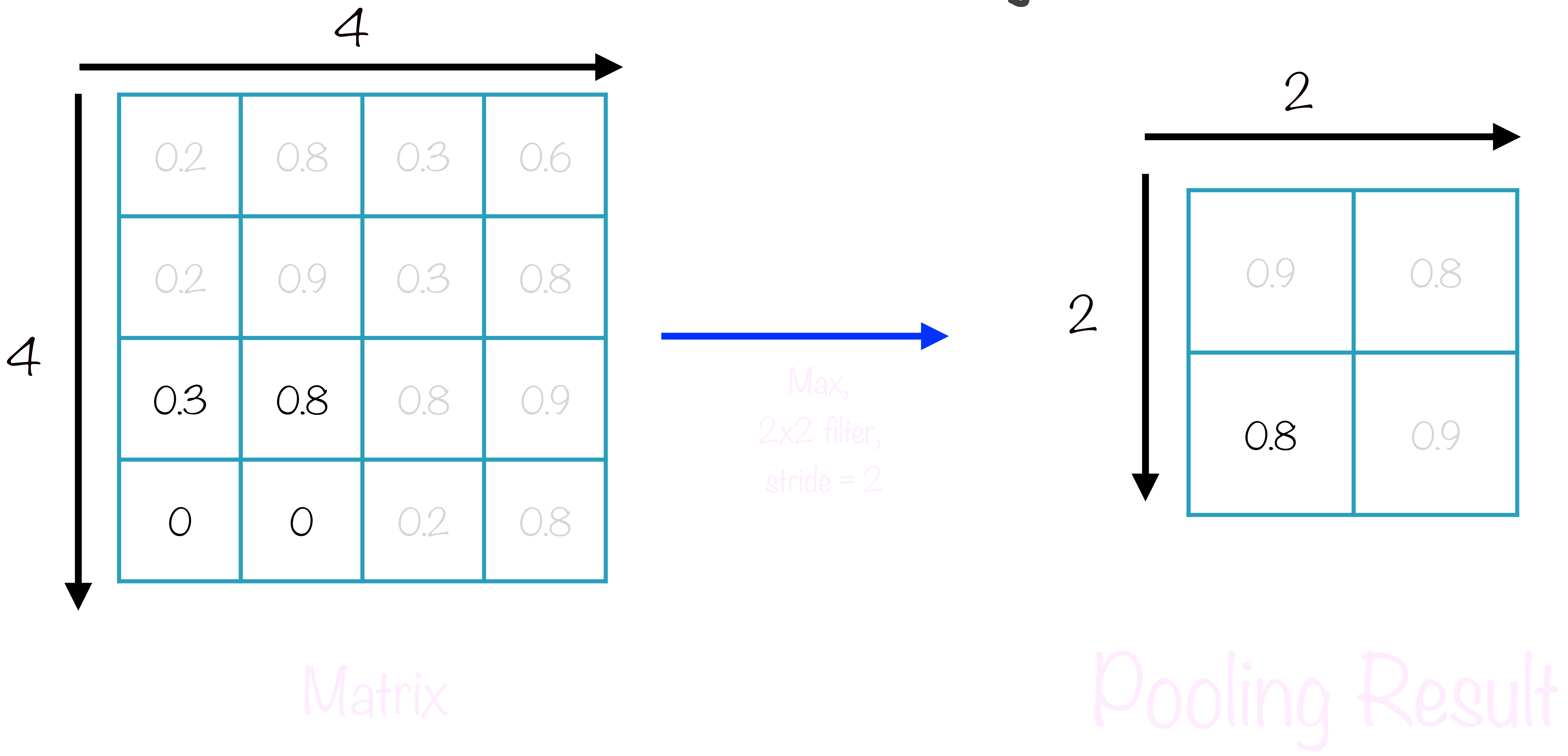


4
4
0.2
0.8
0.3
0.6
0.2
0.9
0.3
0.8
0.3
0.8
0.8
0.9
0
0
0.2
0.8
Max,
2x2 filter,
stride = 2
2
2
0.9
0.8
0.8
0.9
Matrix
Pooling Result

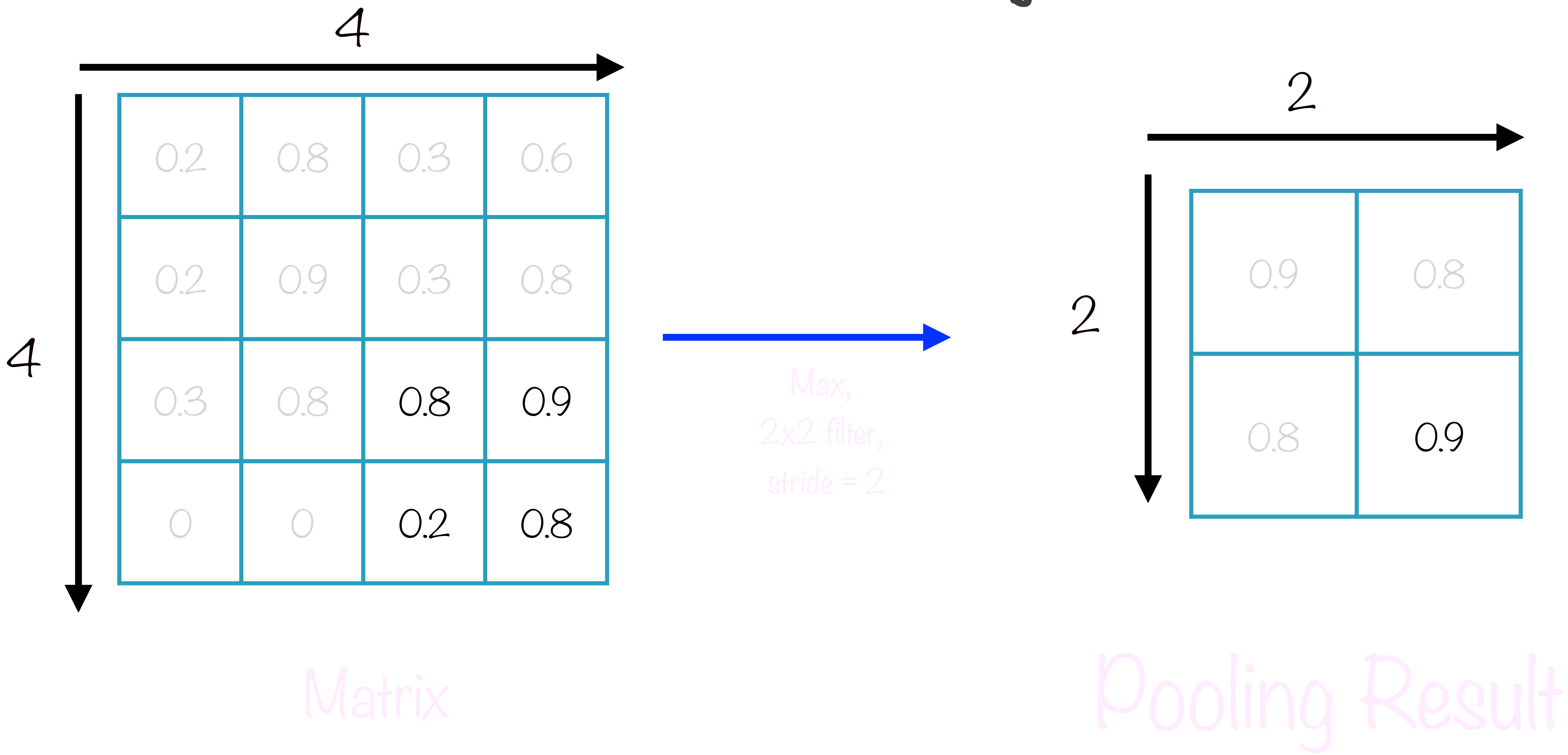
Max Pooling

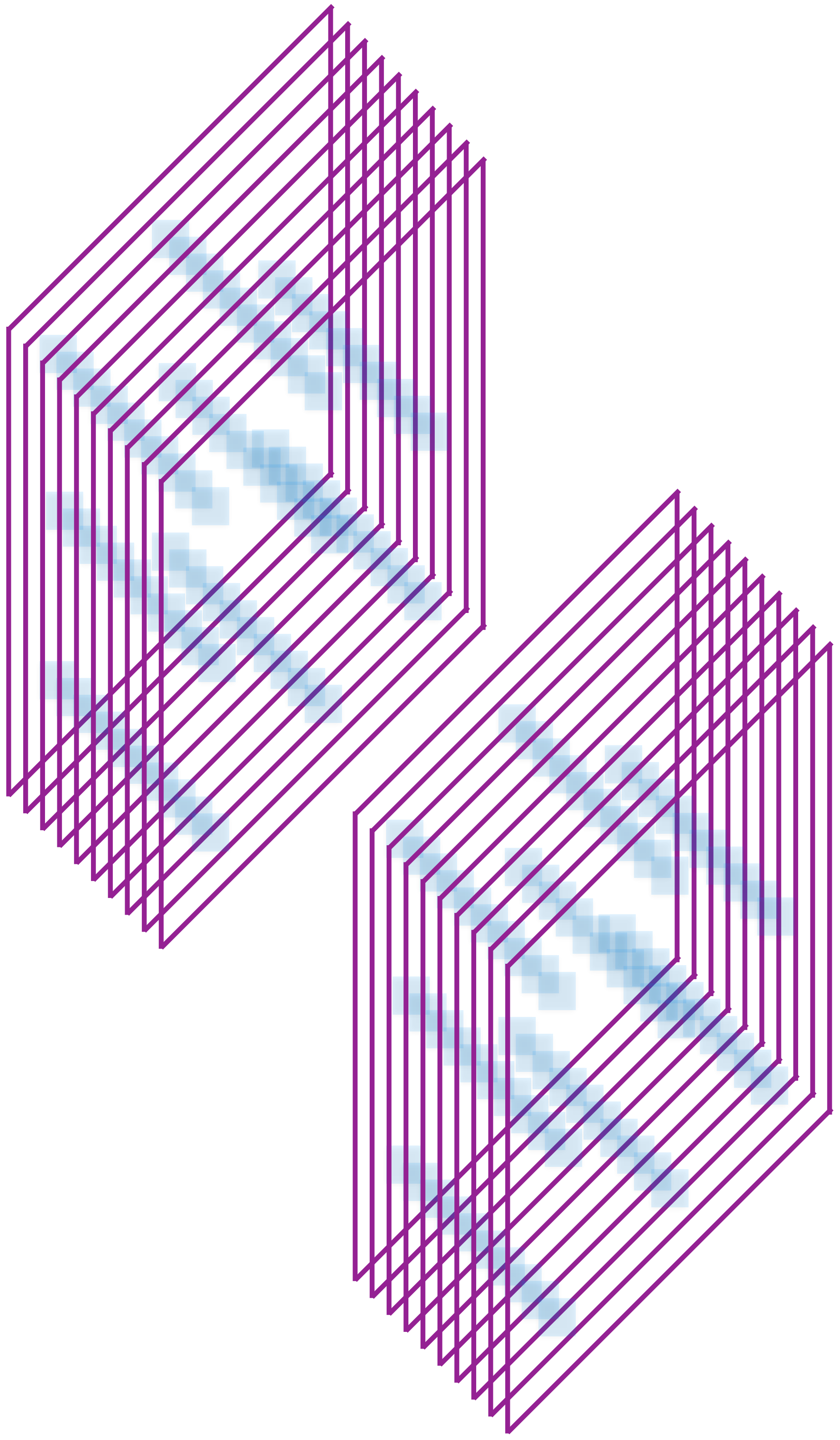

4
4
0.2 0.8 0.3 0.6
0.2 0.9 0.3 0.8
0.3 0.8 0.8 0.9
0 0 0.2 0.8
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Matrix
Pooling Result

Max Pooling


4
4
0.2 0.8 0.3 0.6
0.2 0.9 0.3 0.8
0.3 0.8 0.8 0.9
0 0 0.2 0.8
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Matrix
Pooling Result

Max Pooling


4
4
0.2 0.8 0.3 0.6
0.2 0.9 0.3 0.8
0.3 0.8 0.8 0.9
0 0 0.2 0.8
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Matrix
Pooling Result

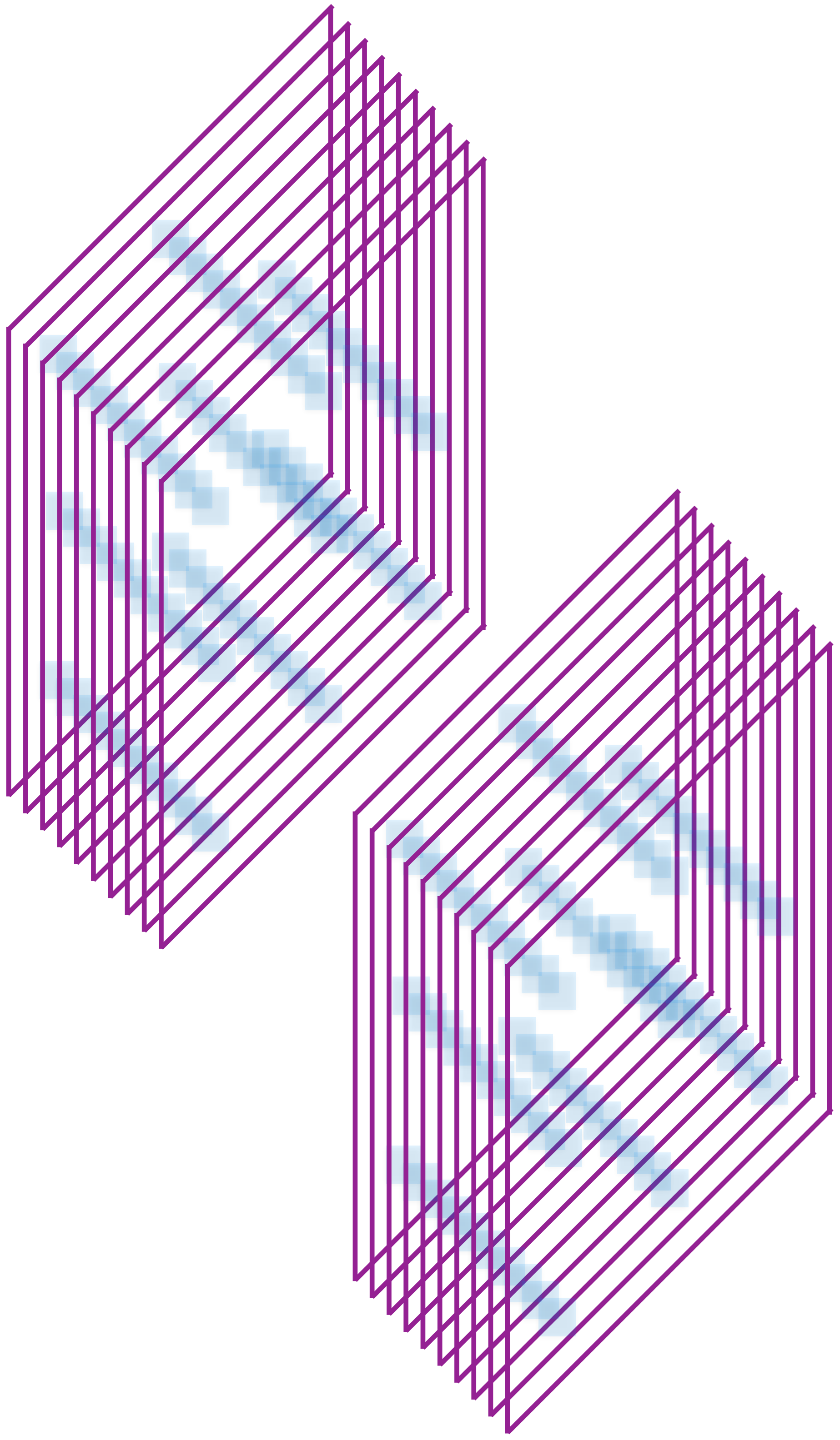


Pooling Layers

Why use them?

- greatly reduce memory usage during training
- mitigate overfitting (via subsampling)
- make NN recognise features independent of location (location invariance)

Pooling Layers



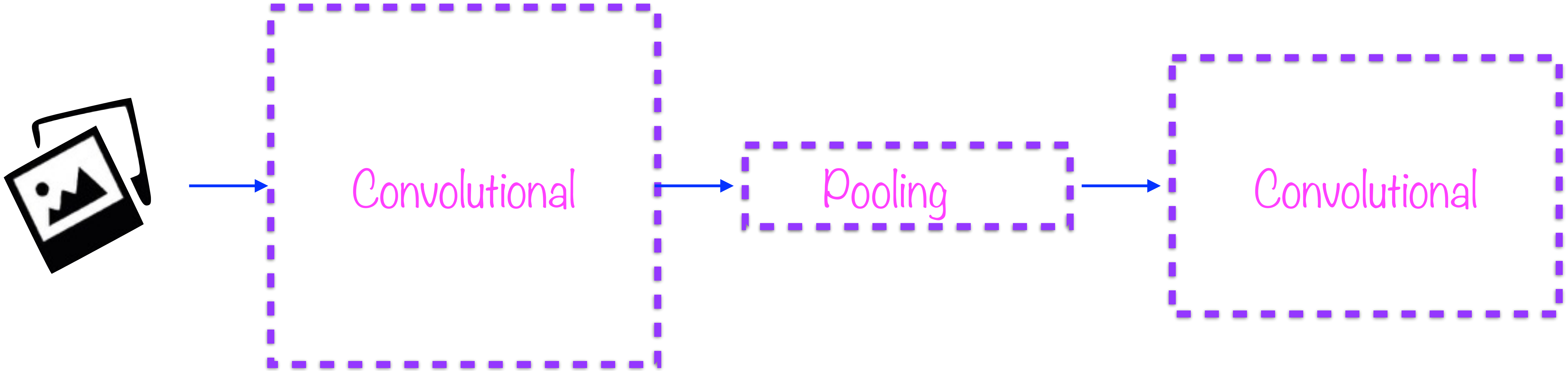
Pooling layers typically act on each channel independently

So, usually, output area < input area but

Output depth = Input depth

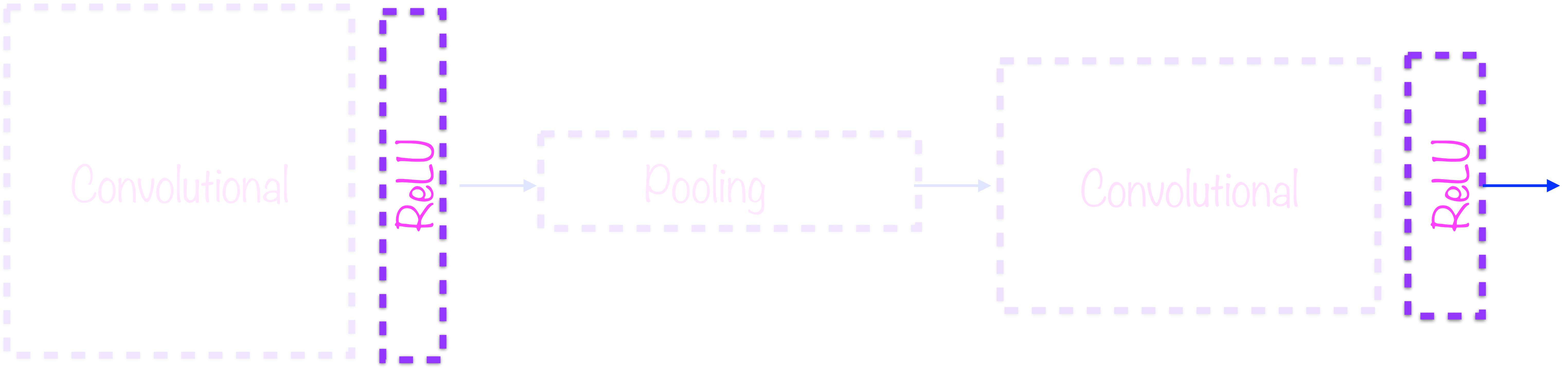
CNNs for Classification

Typical CNN Architecture



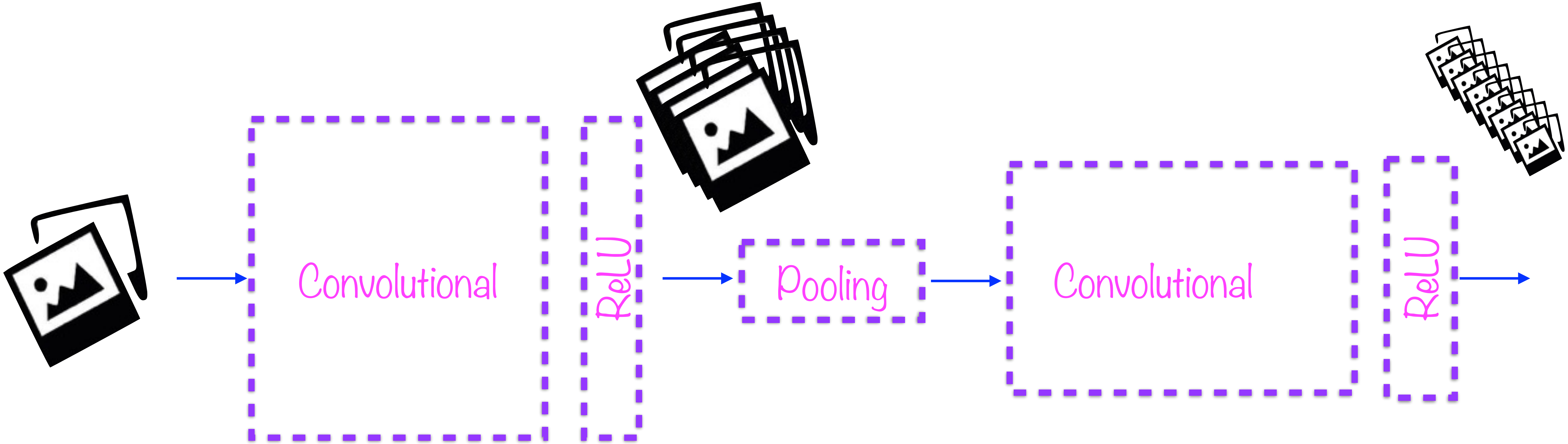
Alternating groups of convolutional and pooling layers

Typical CNN Architecture



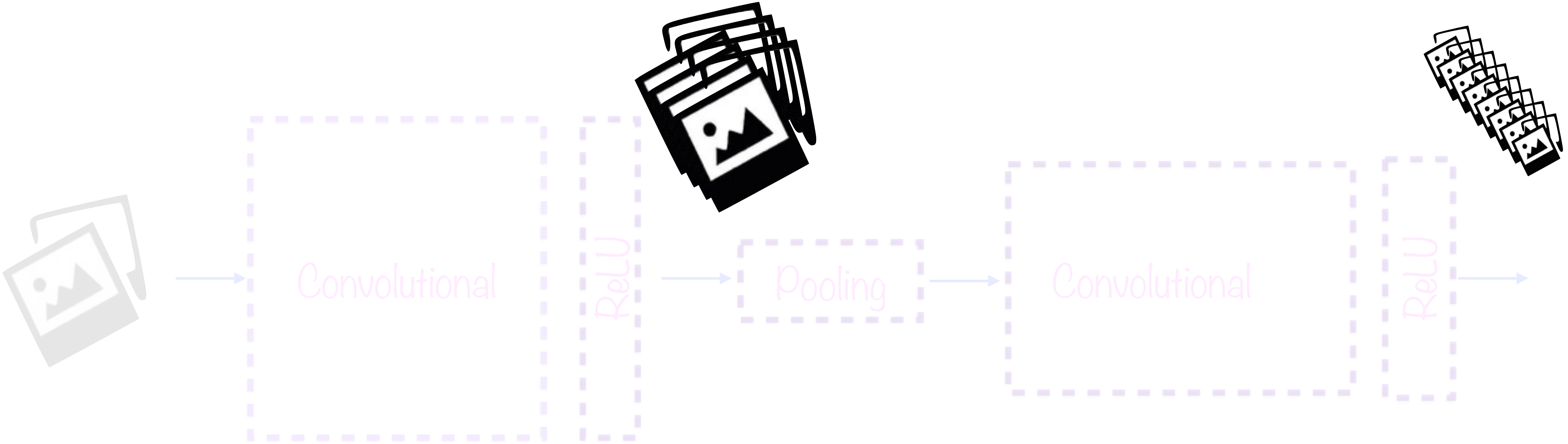
Each group of convolutional layers usually followed by a ReLU layer

Typical CNN Architecture



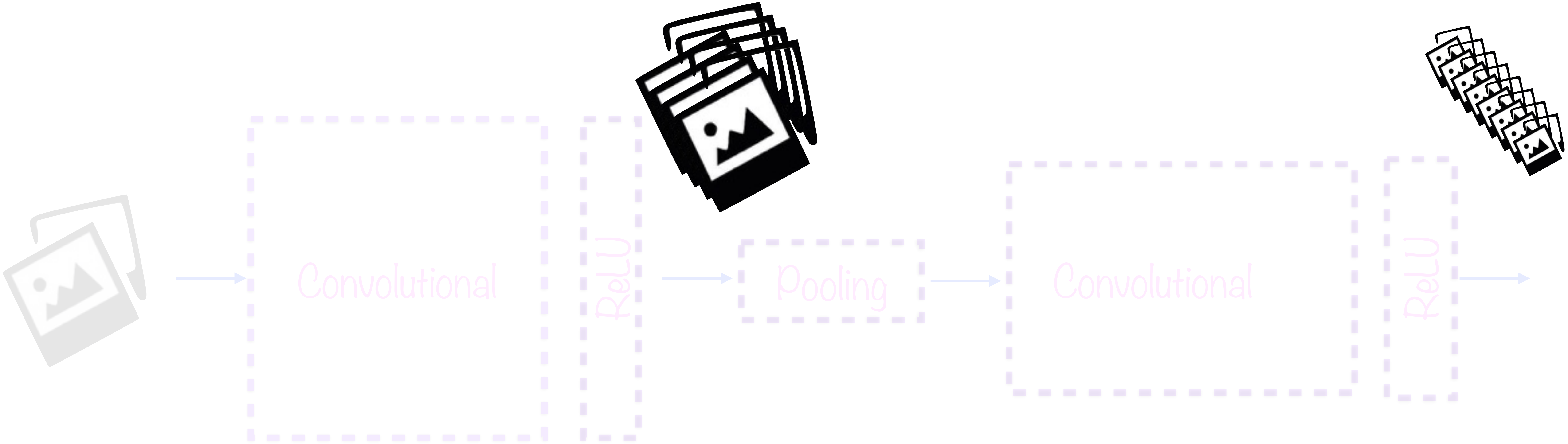
The output of each layer is also an image

Typical CNN Architecture



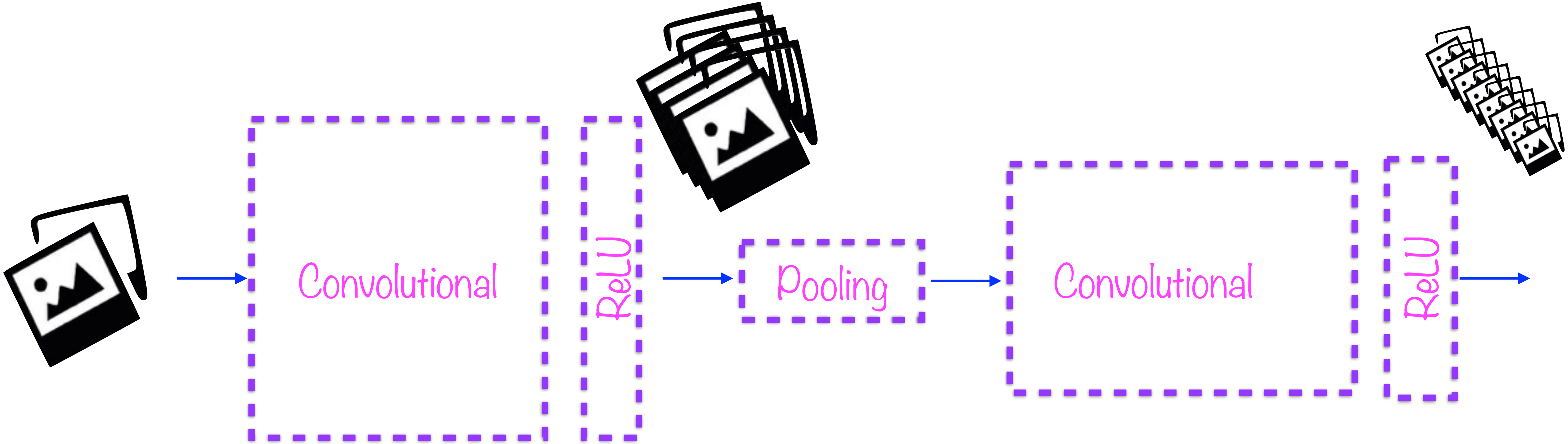
However successive outputs are smaller and smaller (due to pooling layers)

Typical CNN Architecture



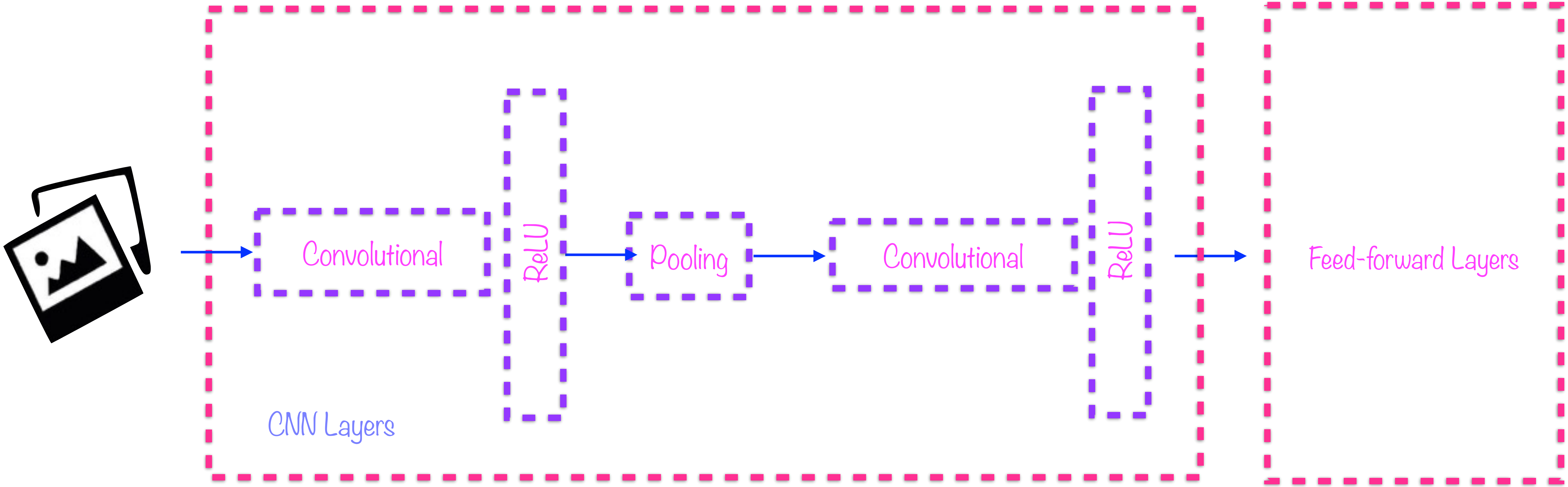
As well as deeper and deeper (due to feature maps in the convolutional layers)

Typical CNN Architecture



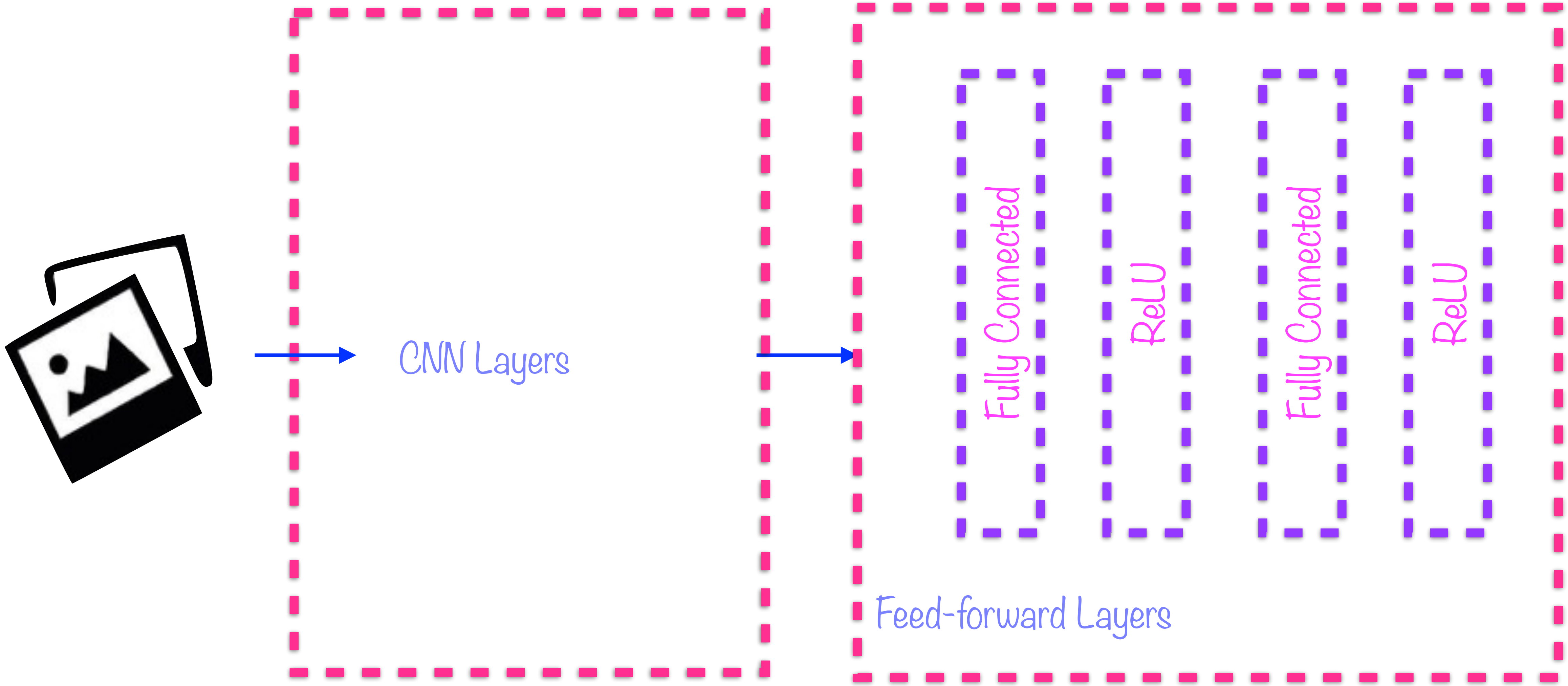
This entire set of layers is then fed into a regular, feed-forward NN

Typical CNN Architecture



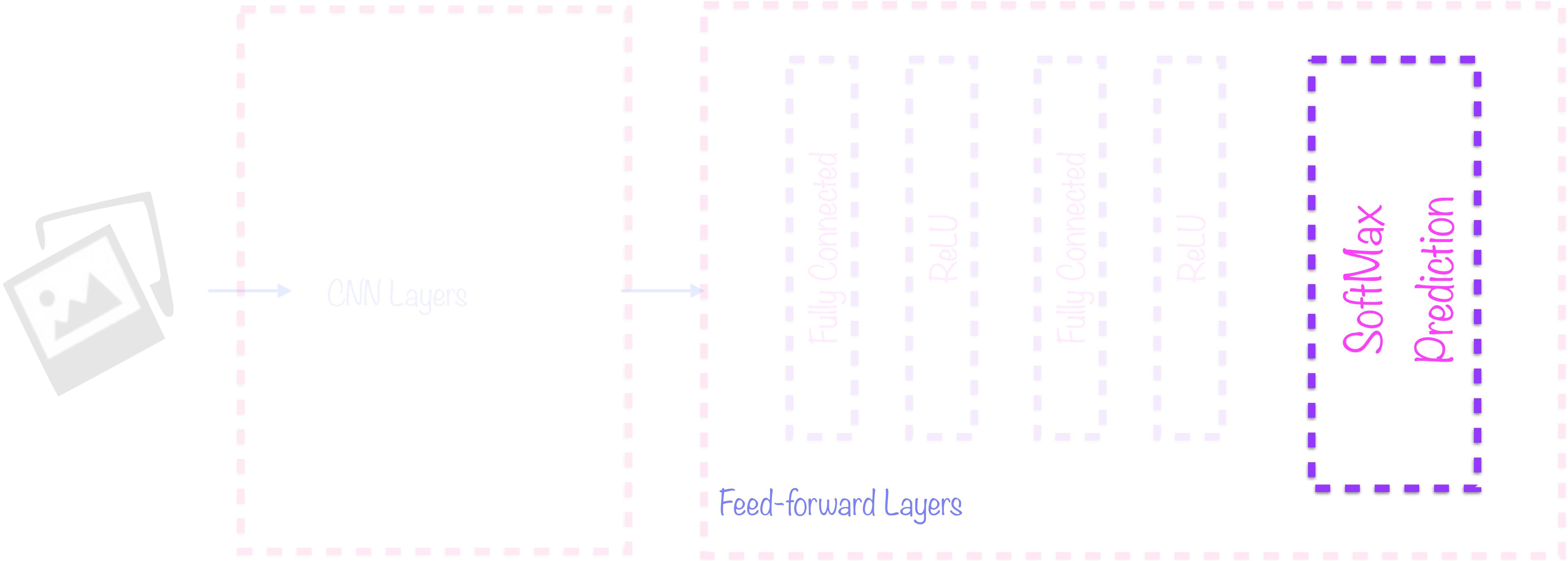
This entire set of layers is then fed into a regular, feed-forward NN

Typical CNN Architecture



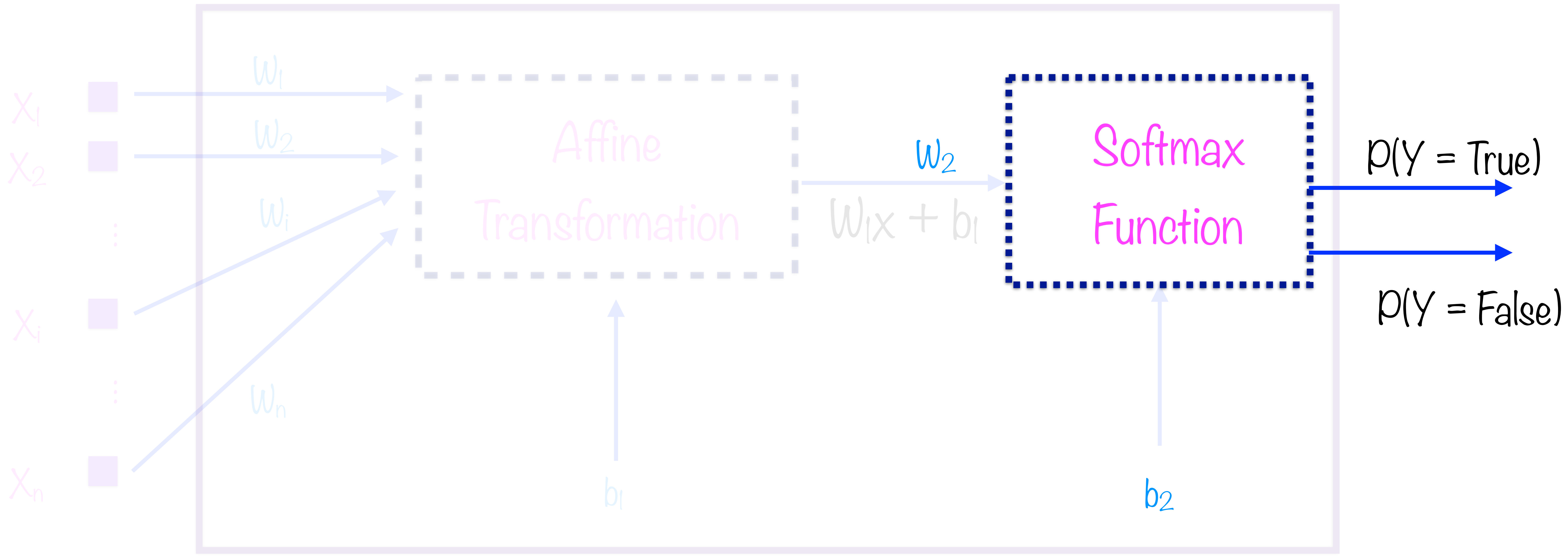
This feed-forward has a few fully connected layers with ReLU activation

Typical CNN Architecture

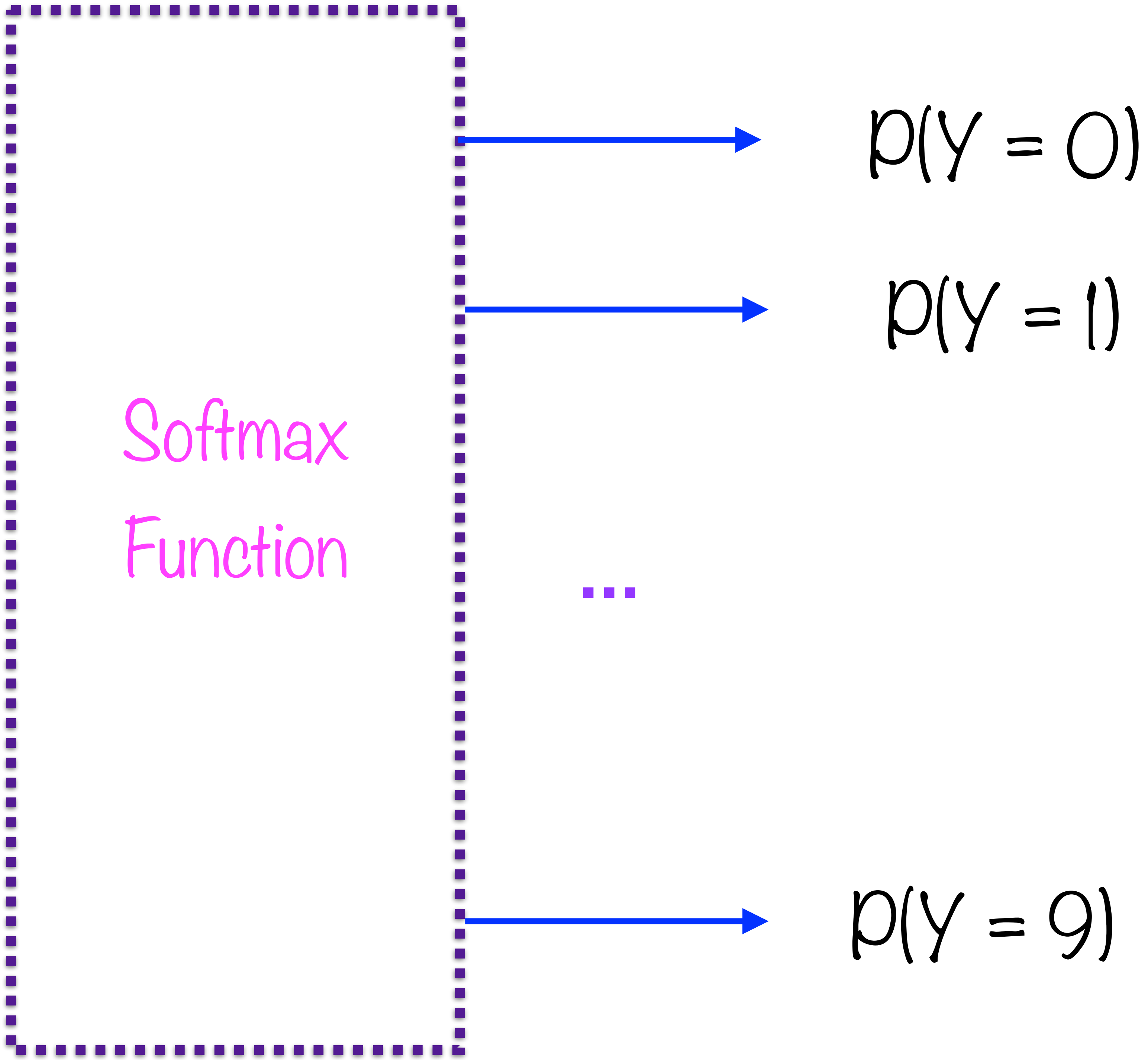


Finally a SoftMax prediction layer

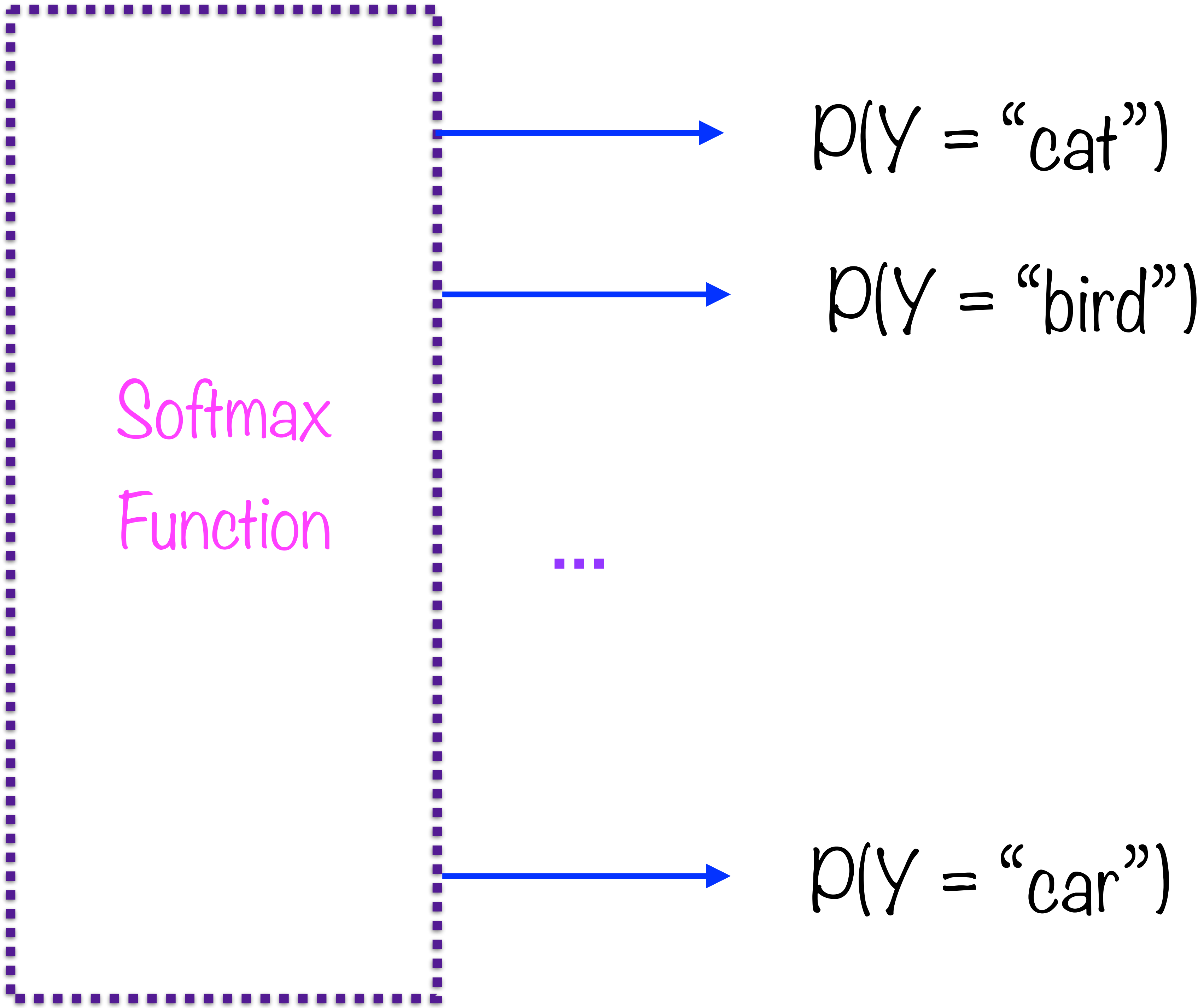
Logistic Regression with One Neuron



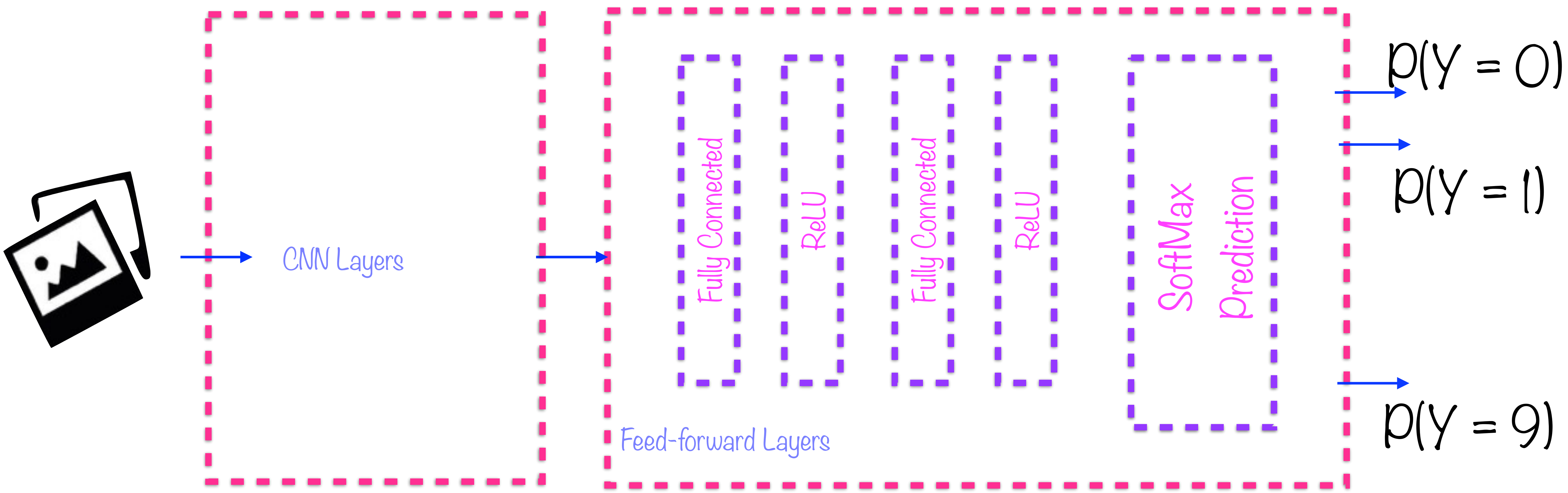
SoftMax for Digit Classification



SoftMax for Image Classification

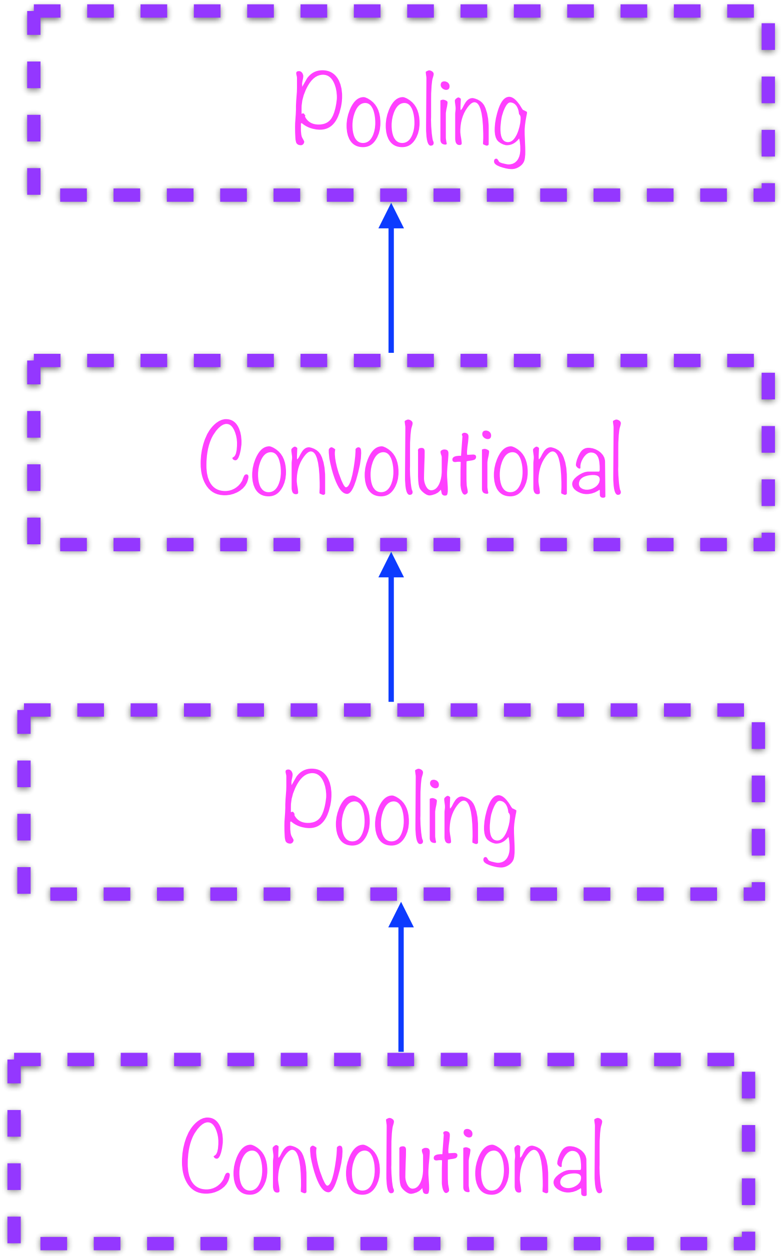


Typical CNN Architecture



This is the output layer, emitting probabilities

Typical CNN Architectures



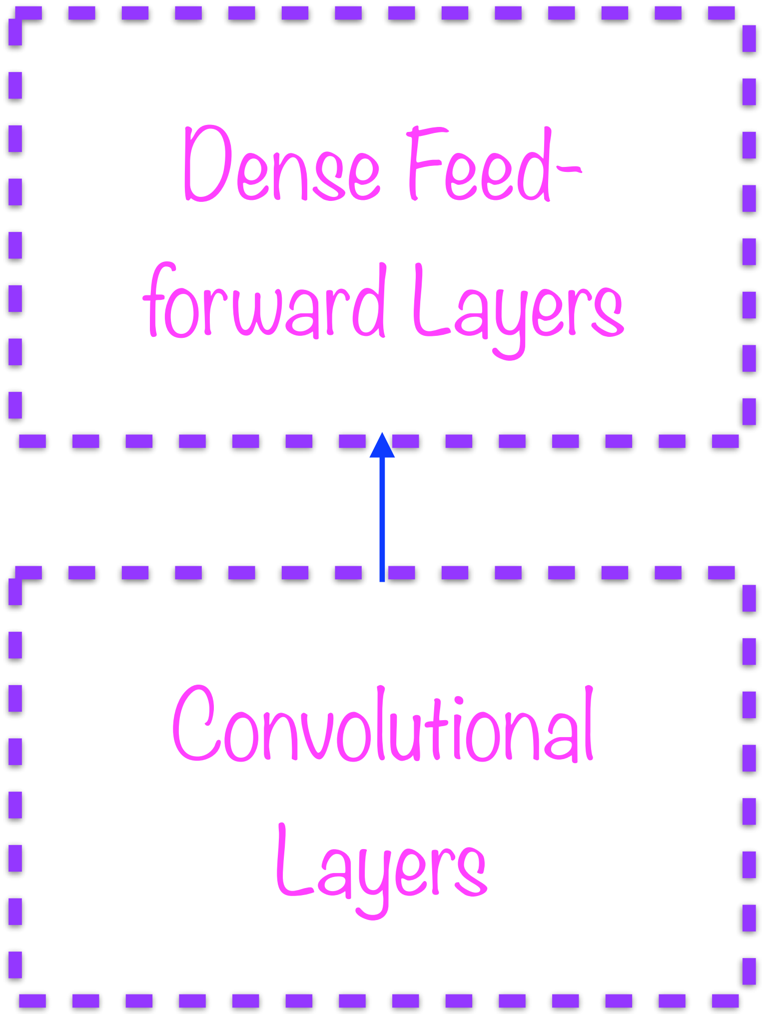
Alternating groups of convolutional and pooling layers

Each group of convolutional layers usually followed by a ReLU layer

Image gets smaller and smaller (due to pooling)

Also deeper and deeper (due to convolution)

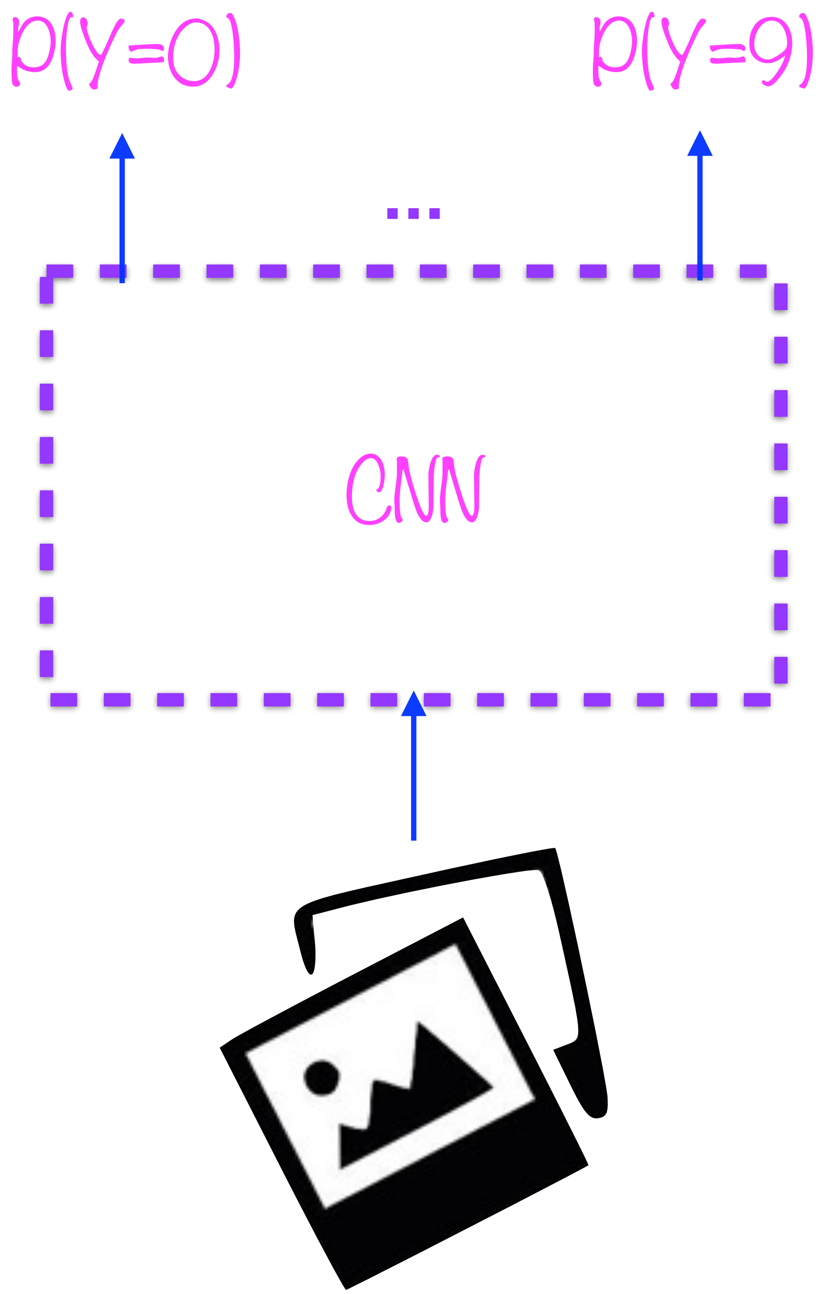
Typical CNN Architectures



At output end of CNN, regular feedforward NN stacked on

- Few fully connected layers
- Input into these are small images
- ReLU activations
- Finally, a Softmax prediction layer

Typical CNN Architectures



Input is an image

Outputs are probabilities