Building an Image Classification Model

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Convolutional NNs are a deep learning technique easily implemented in PyTorch

ResNet is a famous CNN architecture

Transfer learning is a great way to re-use pre-trained models

PyTorch offers great support for transfer learning

Image classification using transfer learning and ResNet

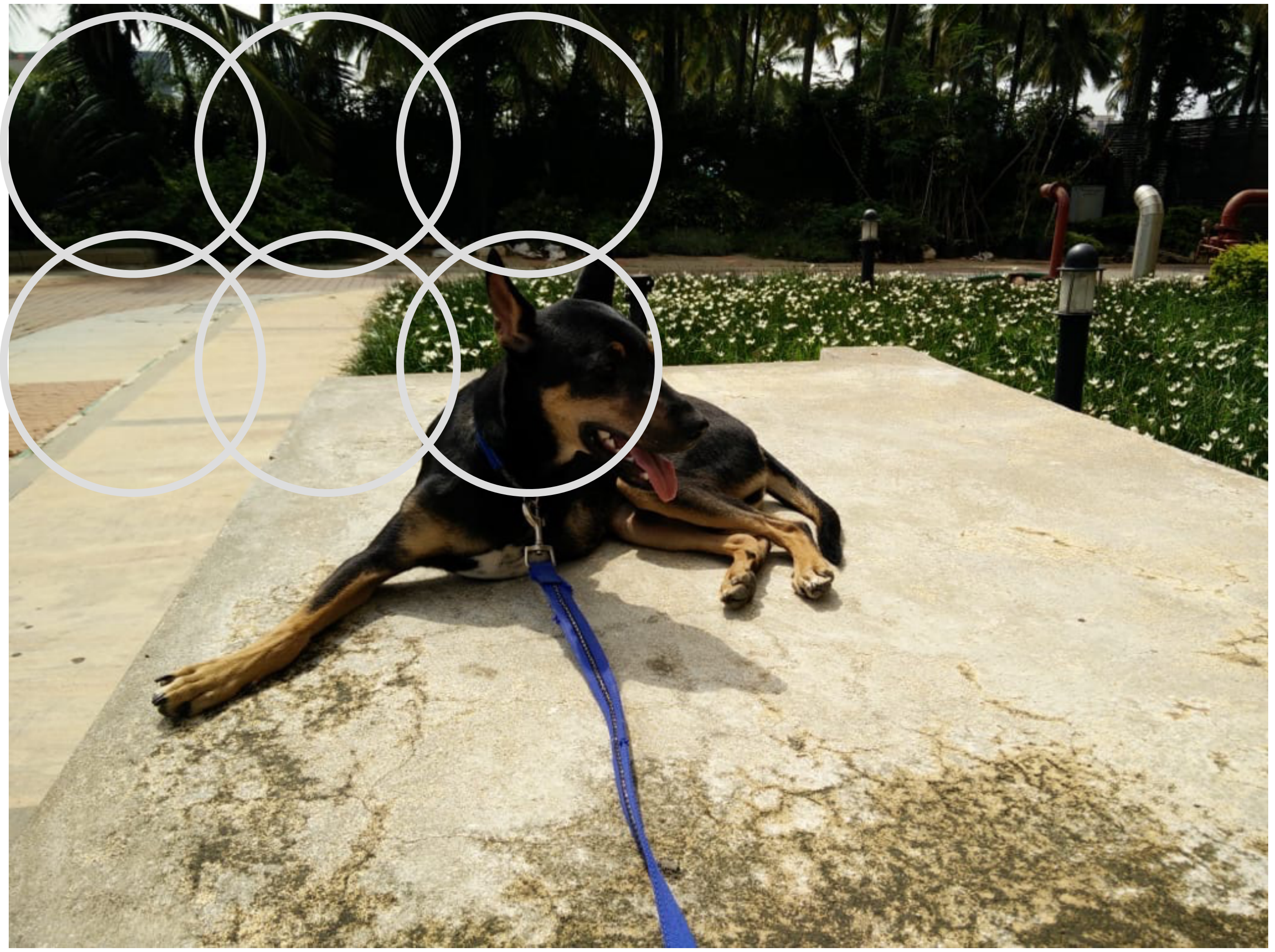
How Do We See?

"Sometimes the mind can see what is invisible to the eye"

Viewing an Image

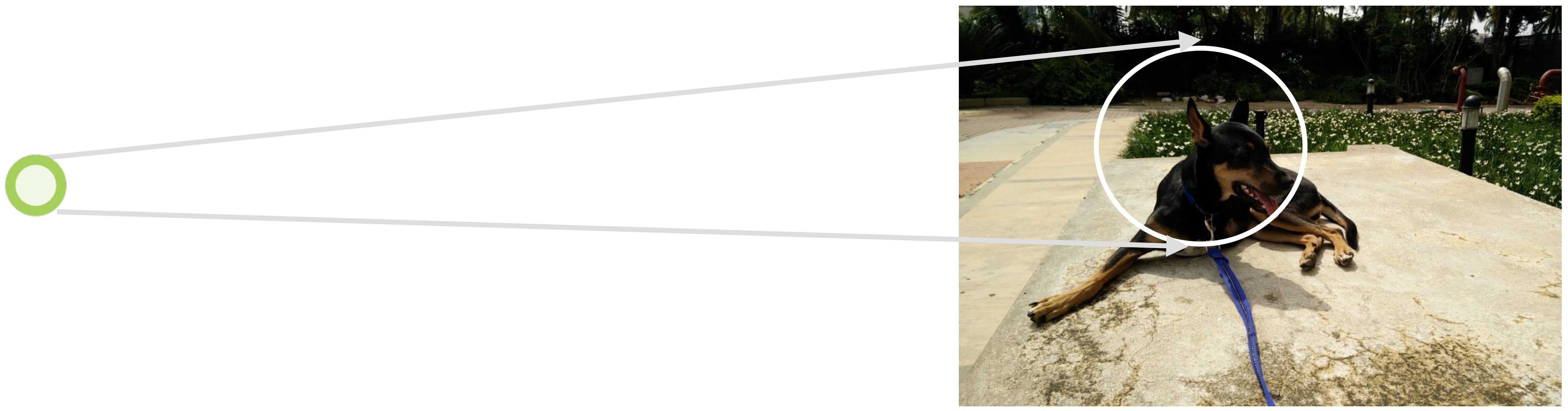
All neurons in the eye don't see the entire image

Viewing an Image



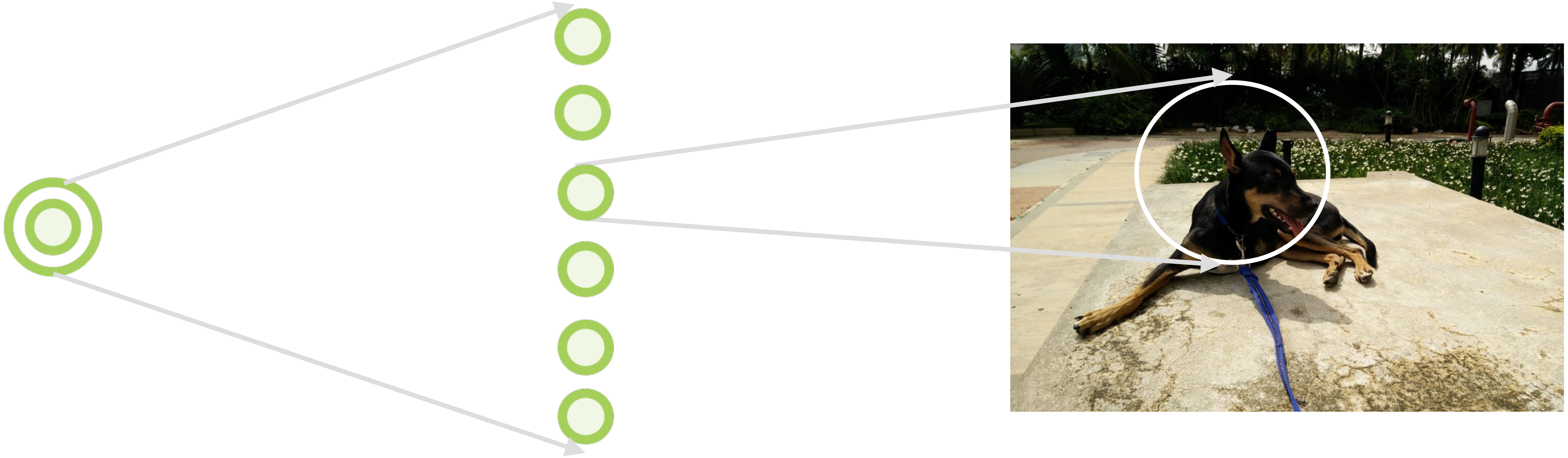
Each neuron has its own local receptive field

Viewing an Image



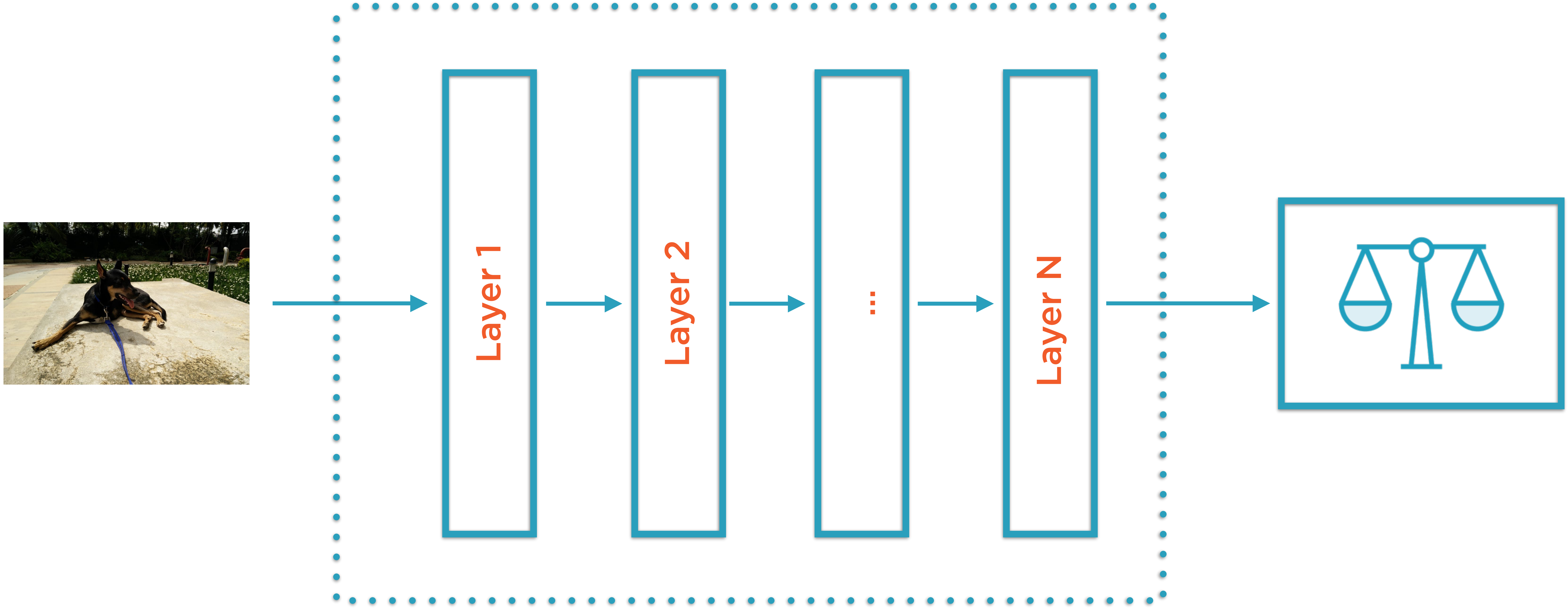
It reacts only to visual stimuli located in its receptive field

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

Convolution

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

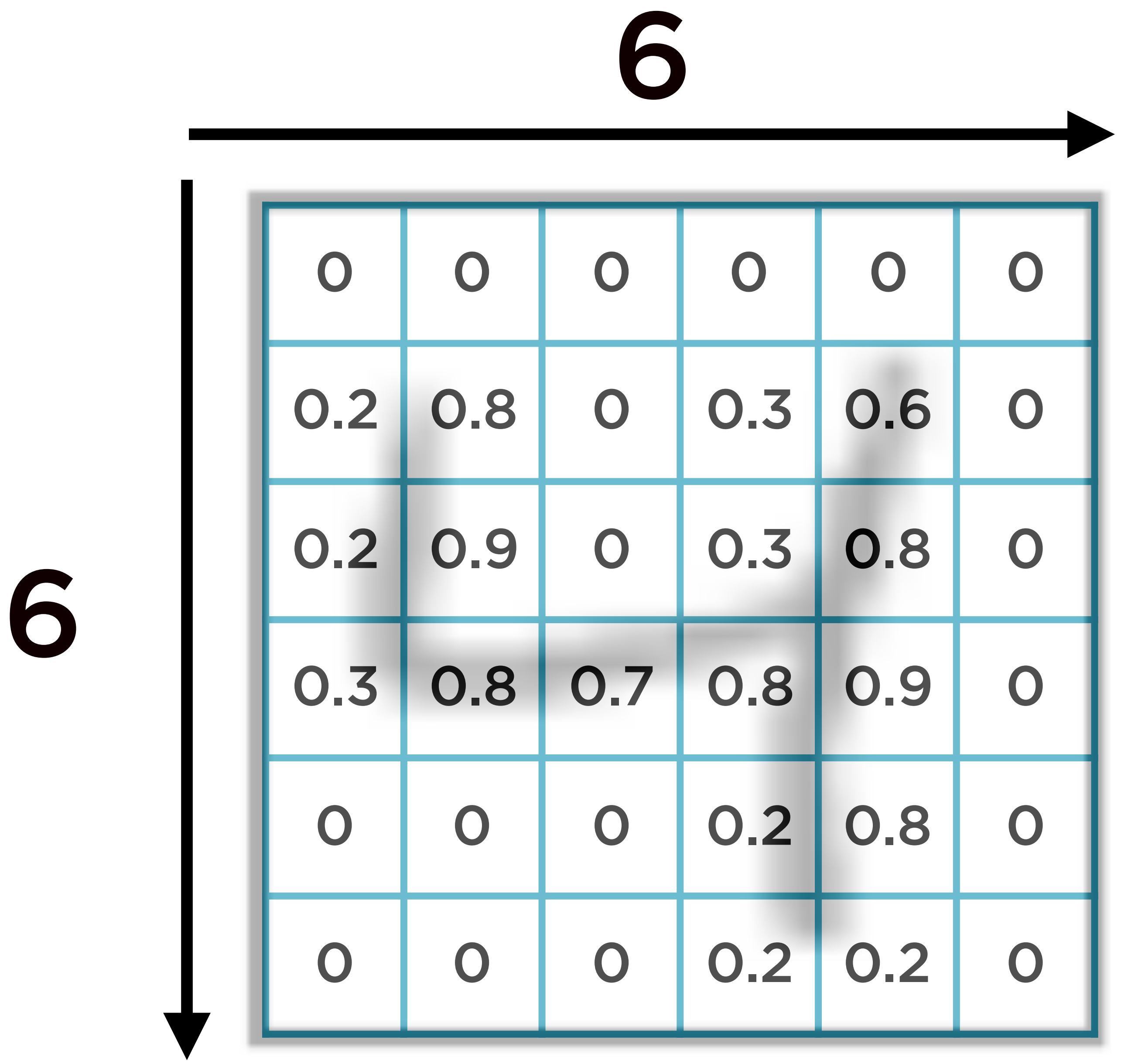
Representing Images as Matrices

28

28

= 784 pixels

Representing Images as Matrices



= 36 pixels

Representing Images

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

3 × 3

1	0	1
0	1	0
1	0	1

Kernel

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

3

3

x1	x0	x1
x0	x1	x0
x1	x0	x1

Kernel

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

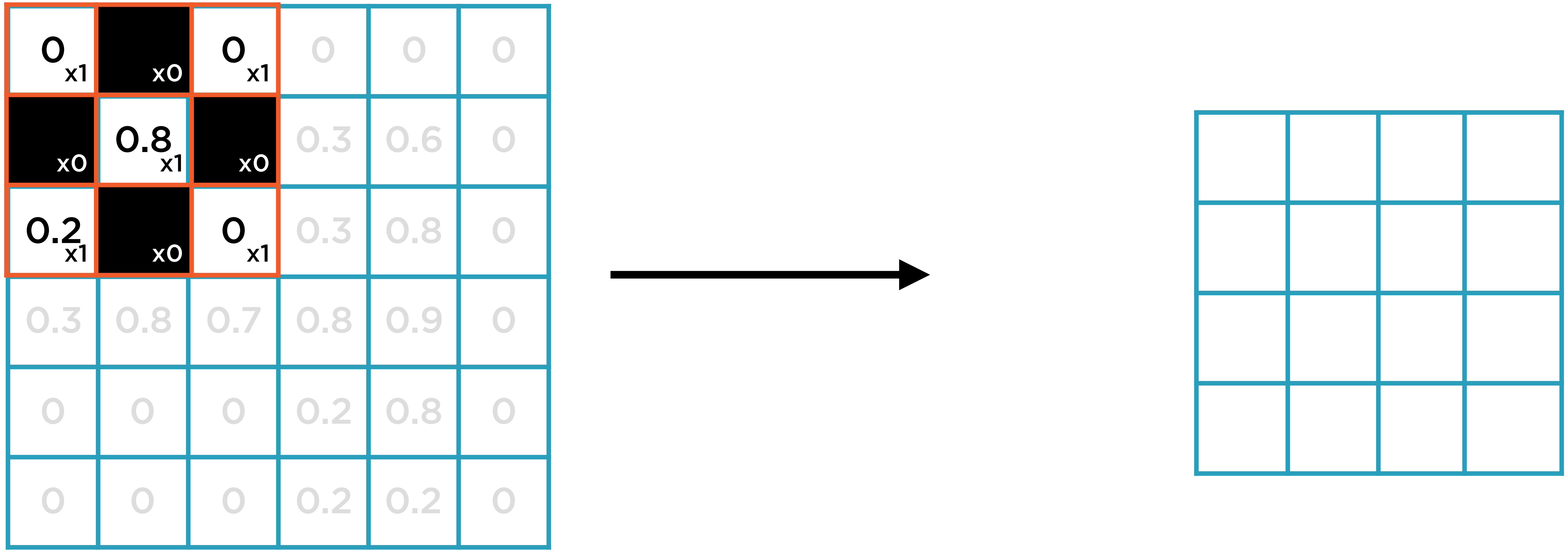
4

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

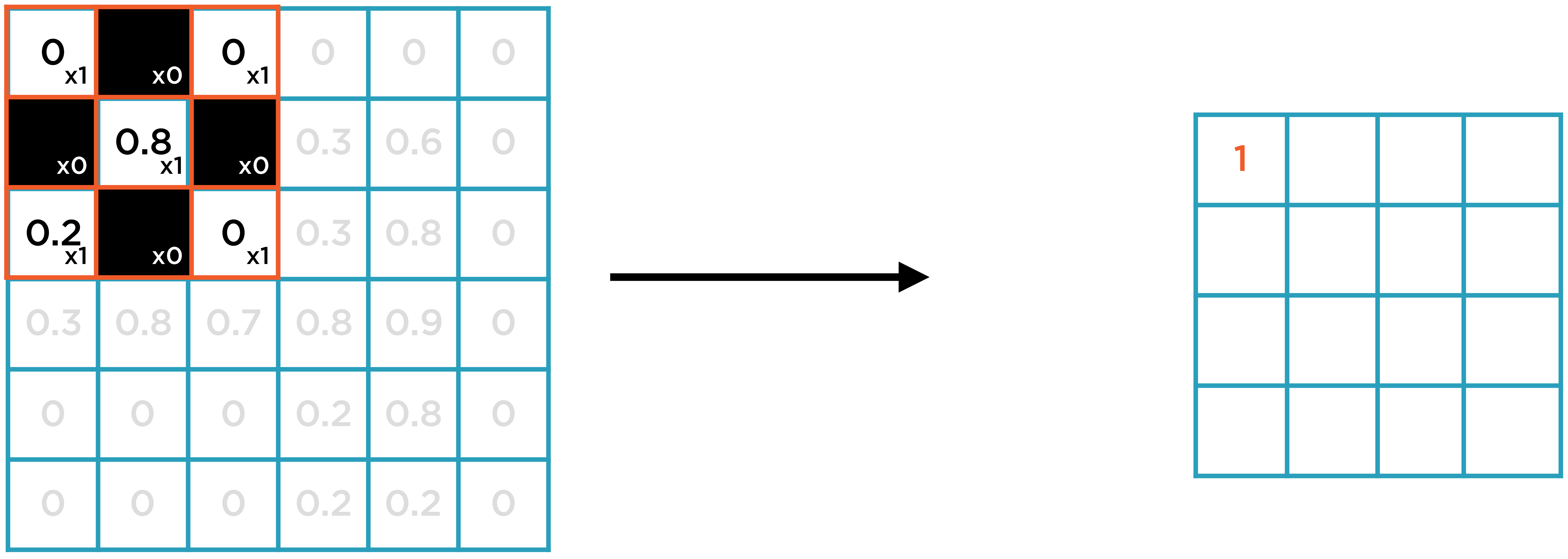
Convolution



Matrix

Convolution Result

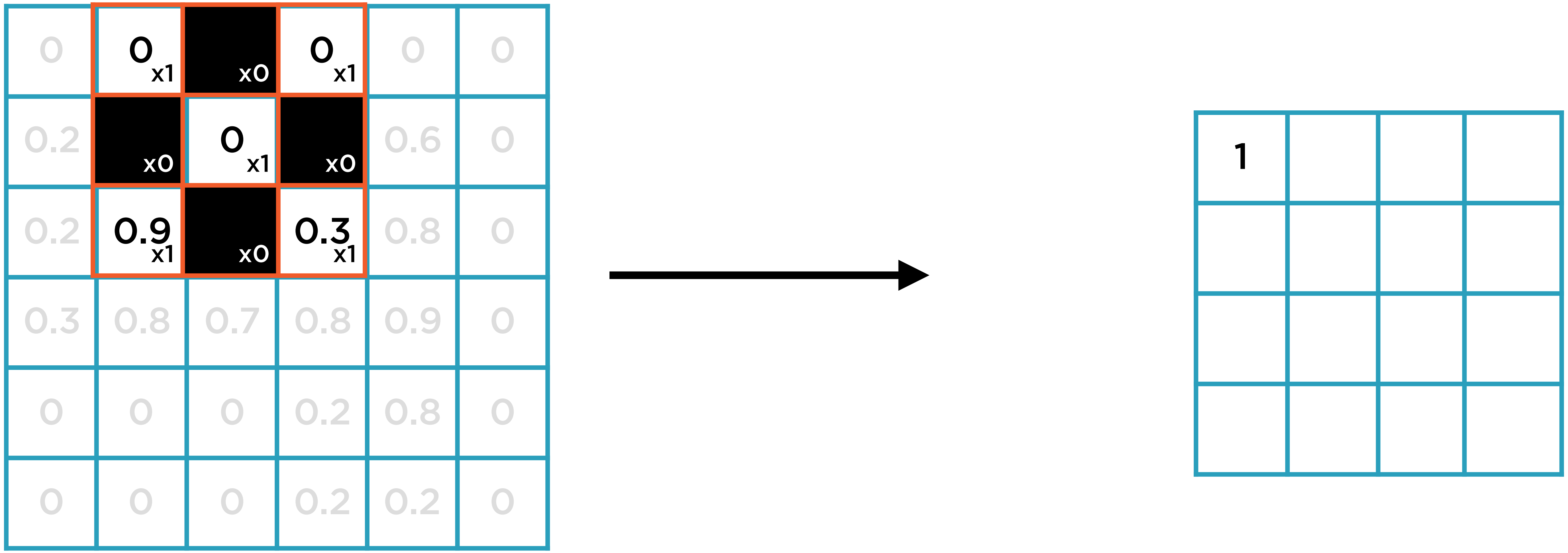
Convolution



Matrix

Convolution Result

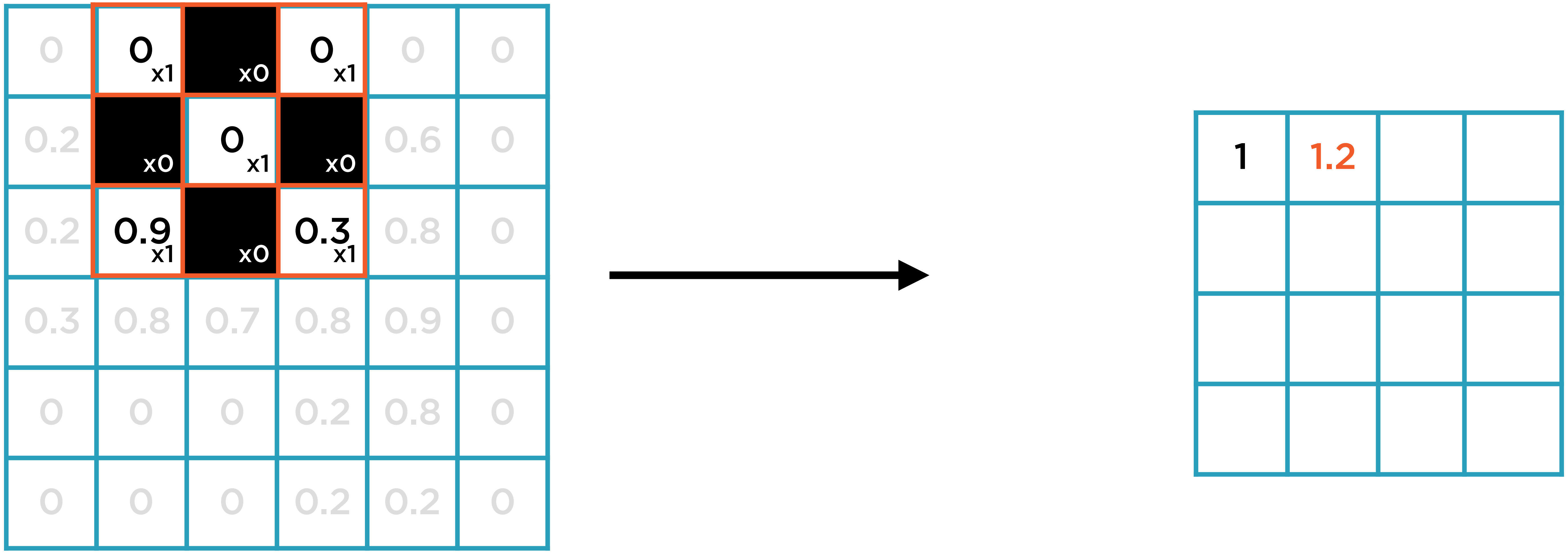
Convolution



Matrix

Convolution Result

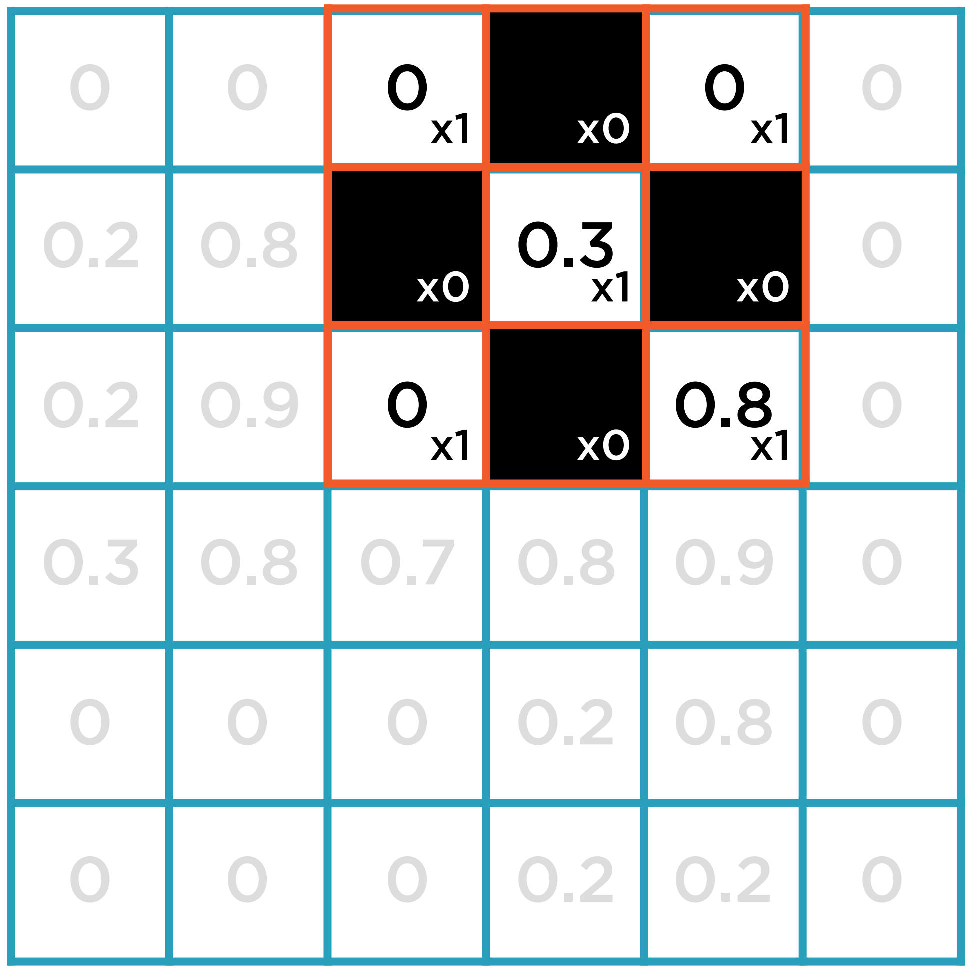
Convolution



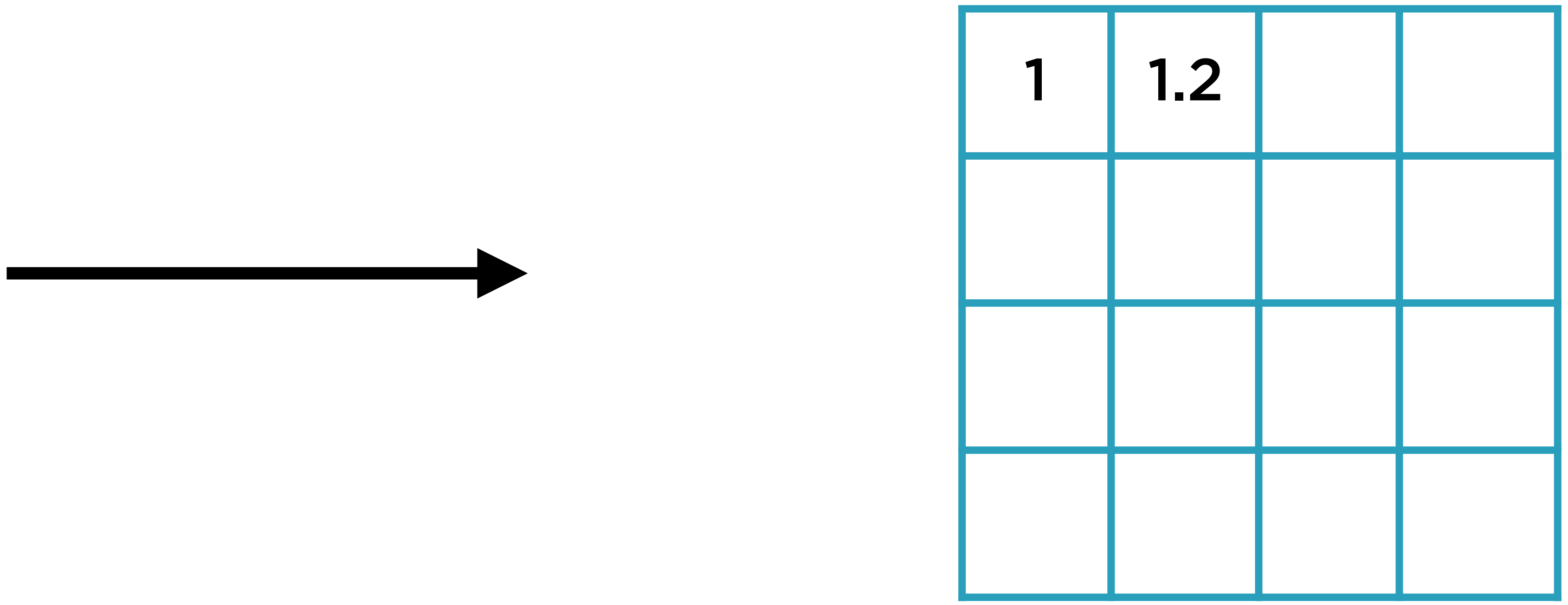
Matrix

Convolution Result

Convolution

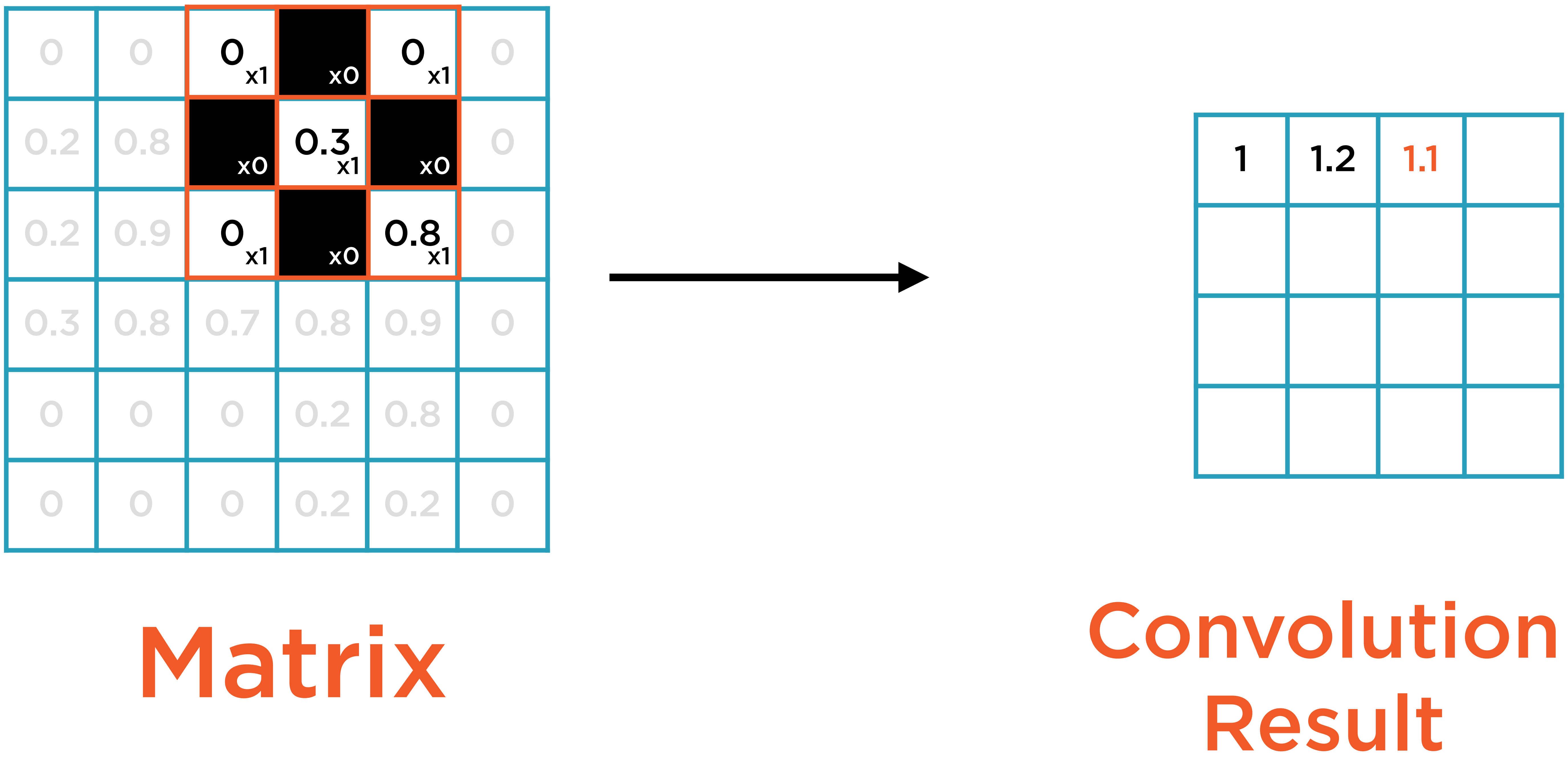


Matrix

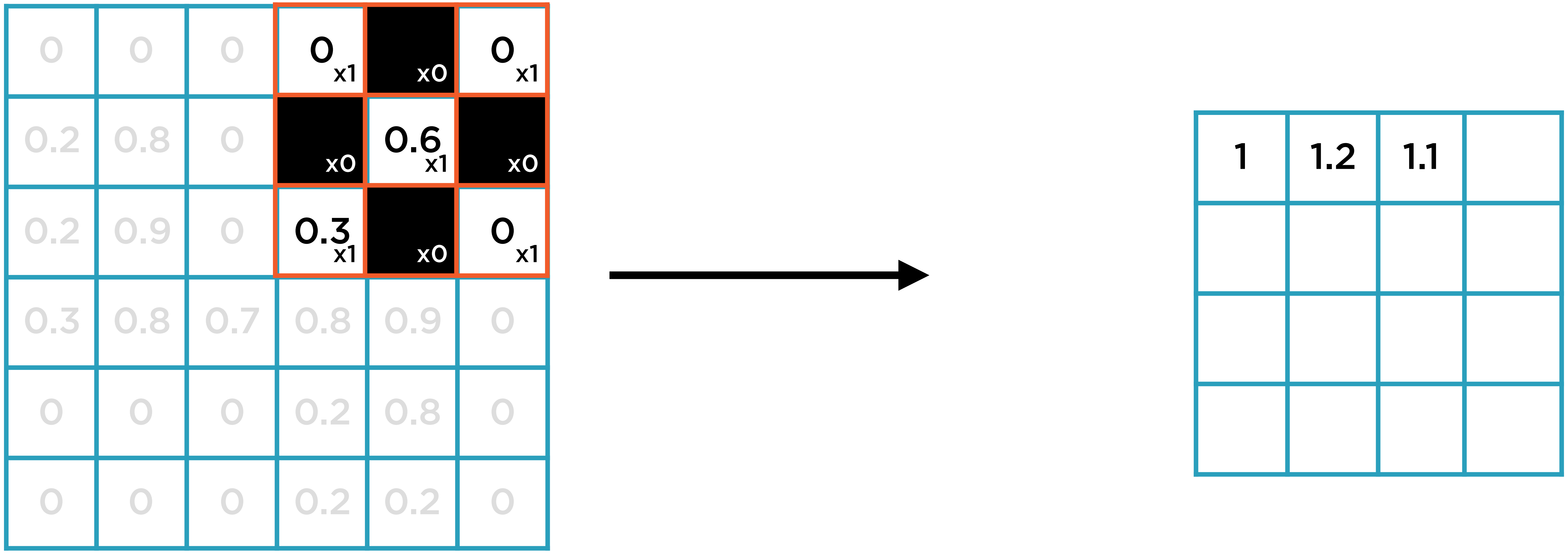


Convolution Result

Convolution



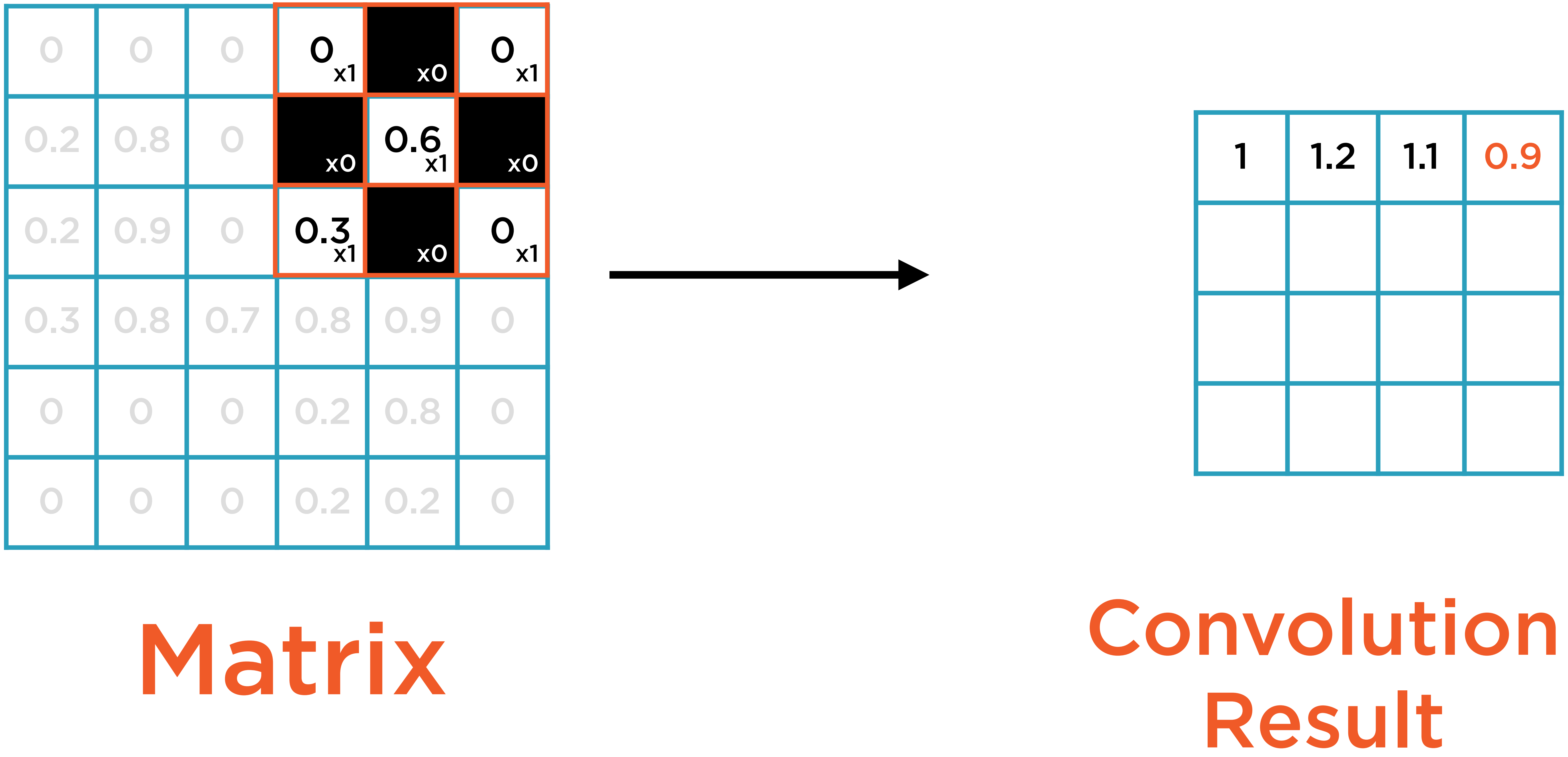
Convolution



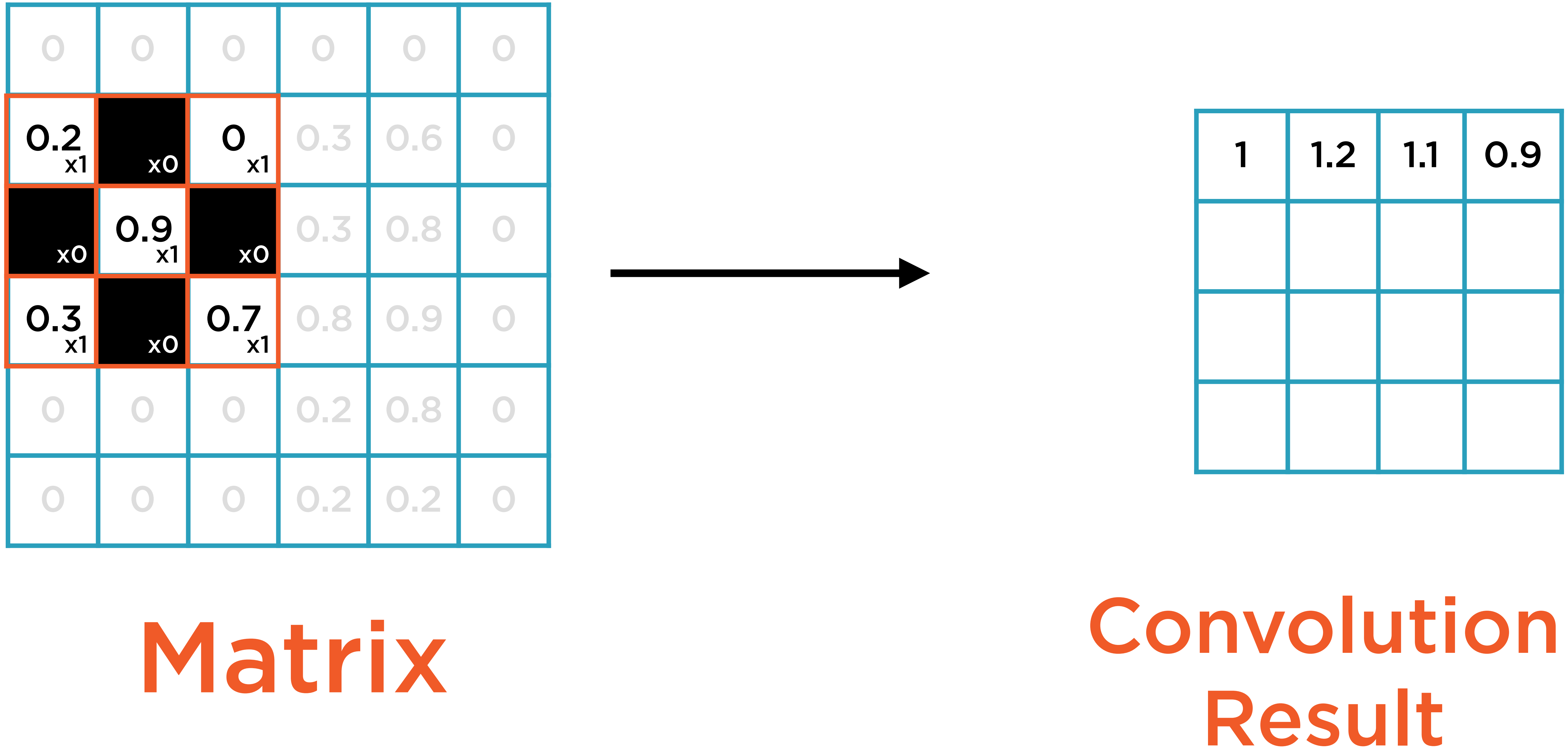
Matrix

Convolution Result

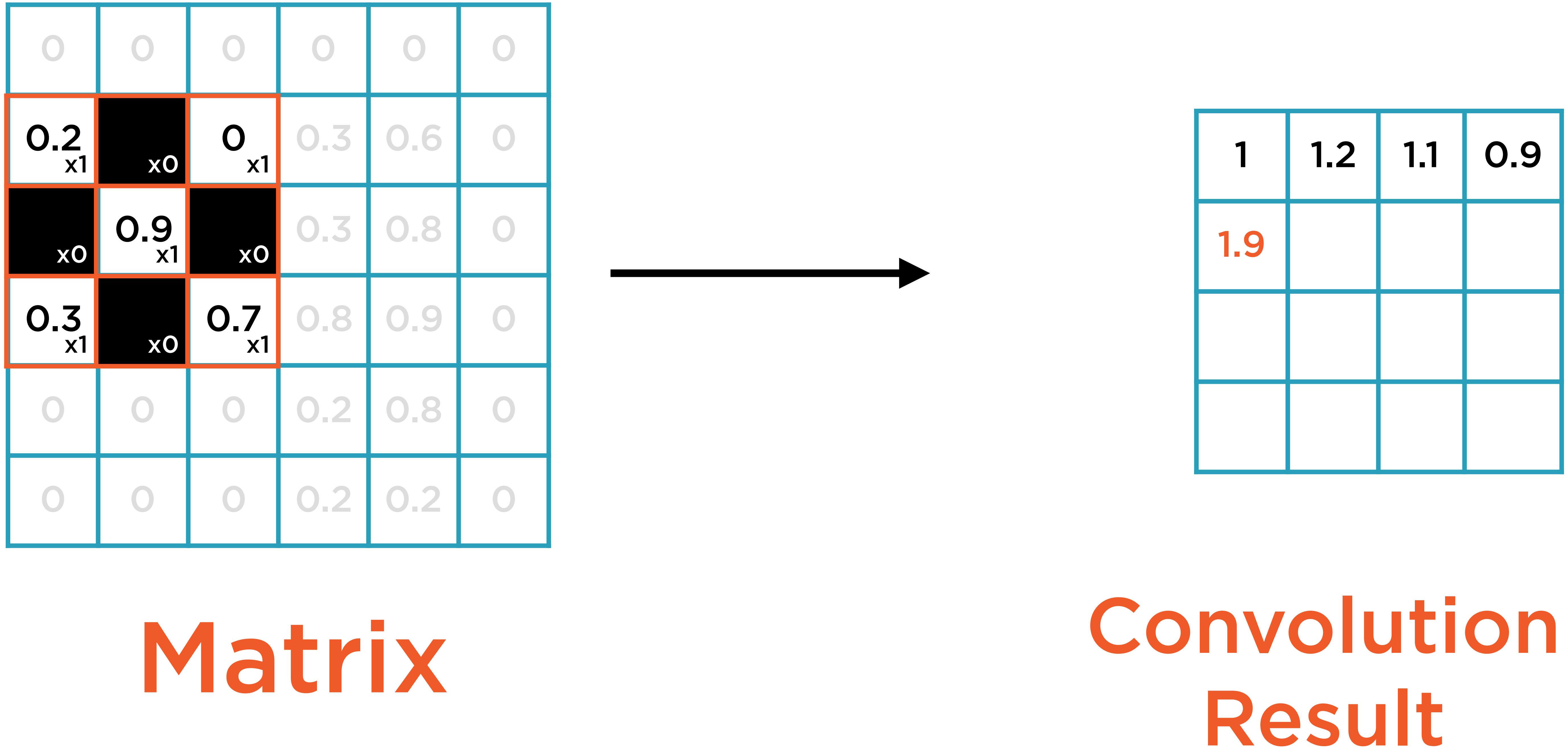
Convolution



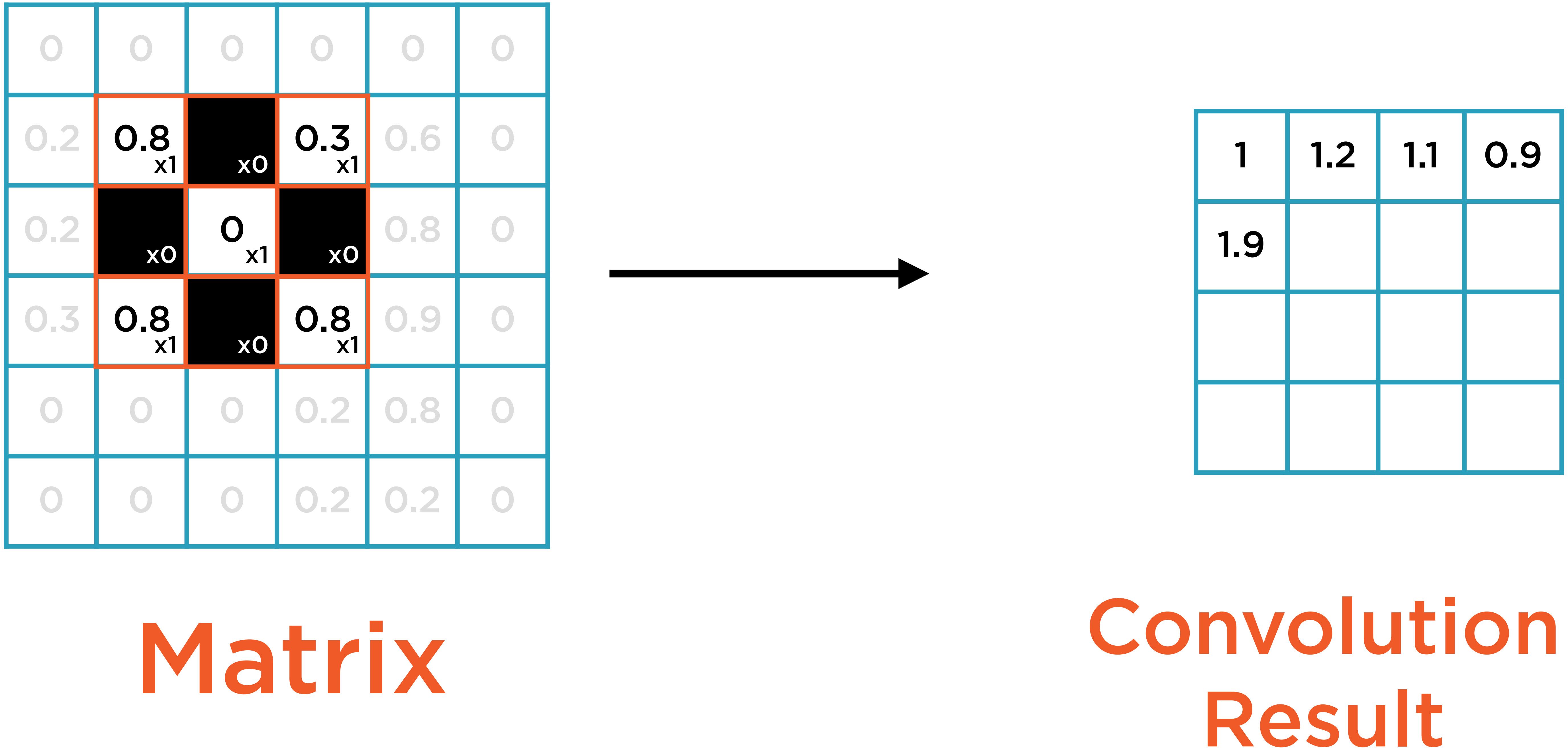
Convolution



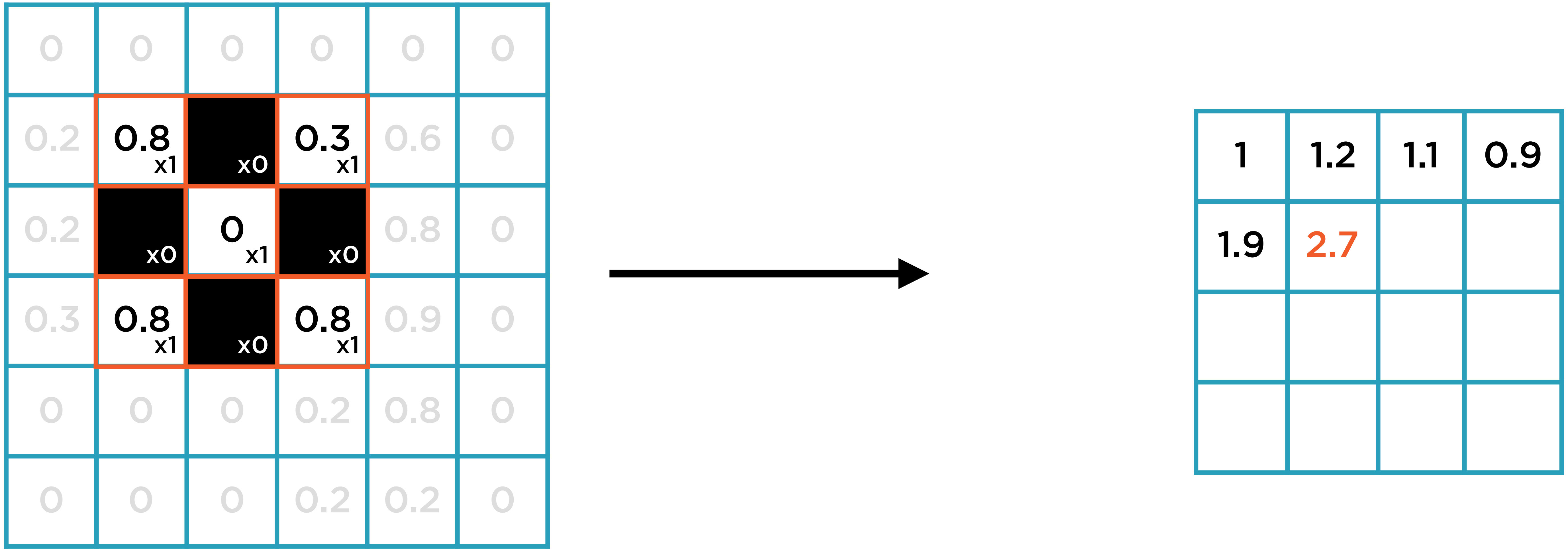
Convolution



Convolution



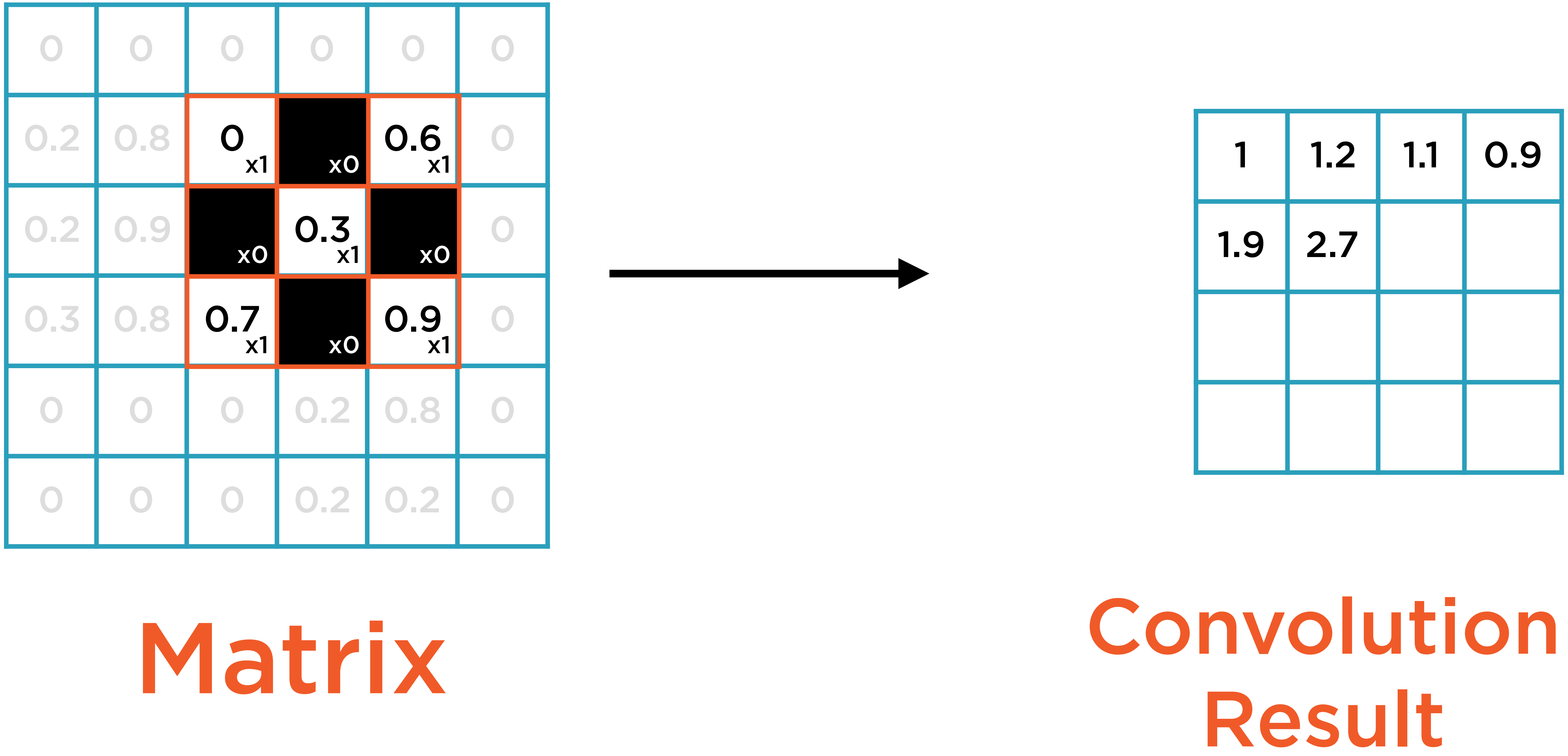
Convolution



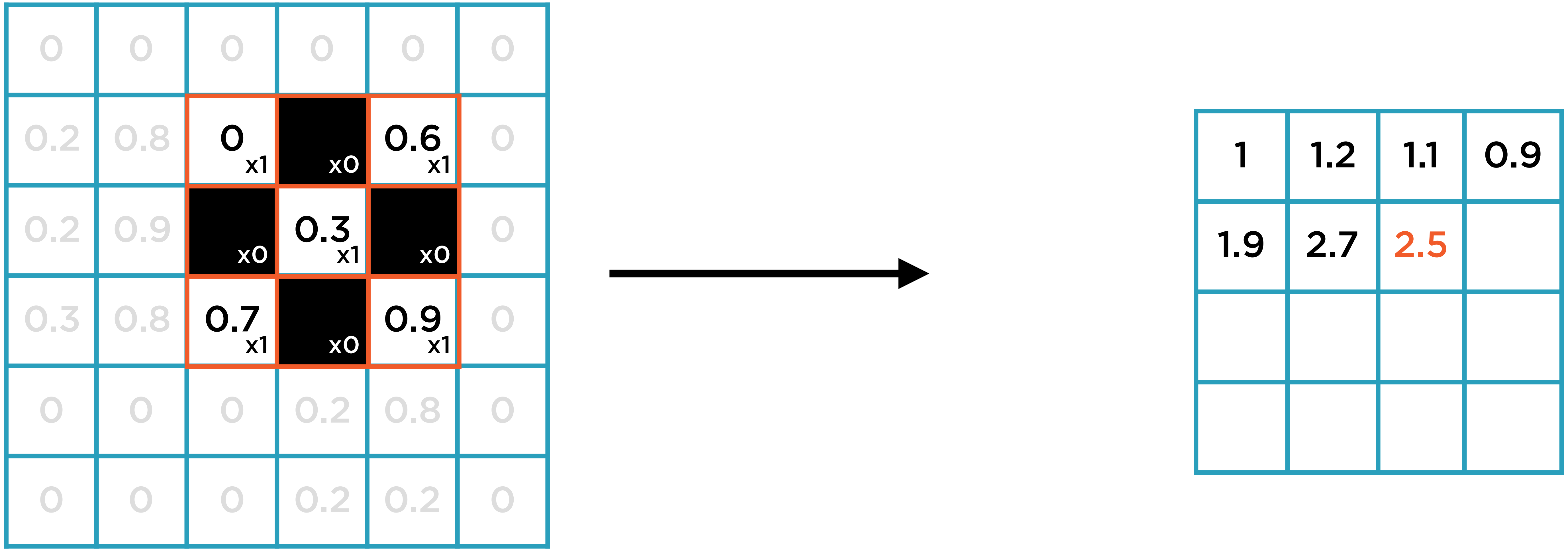
Matrix

Convolution Result

Convolution



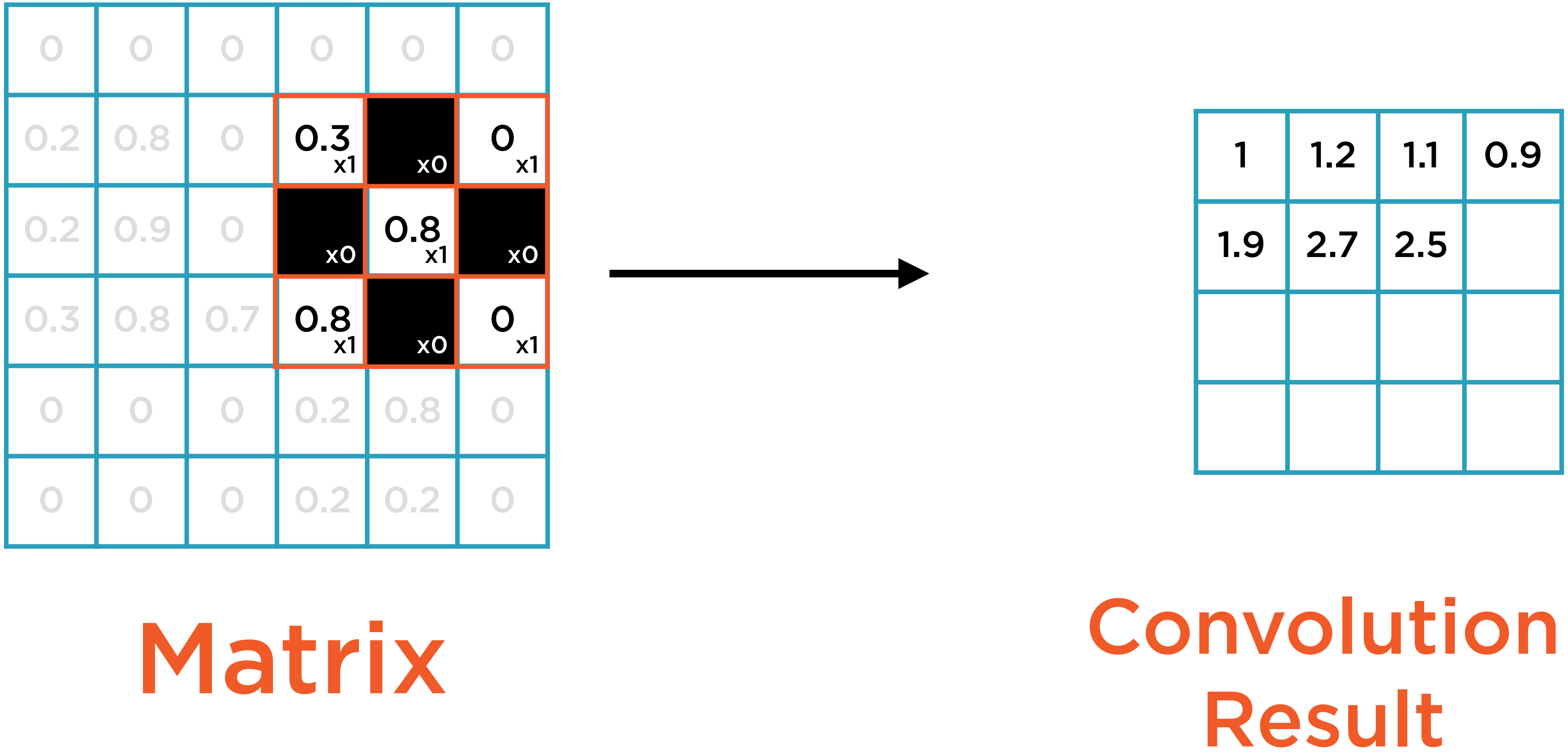
Convolution



Matrix

Convolution Result

Convolution



Convolution

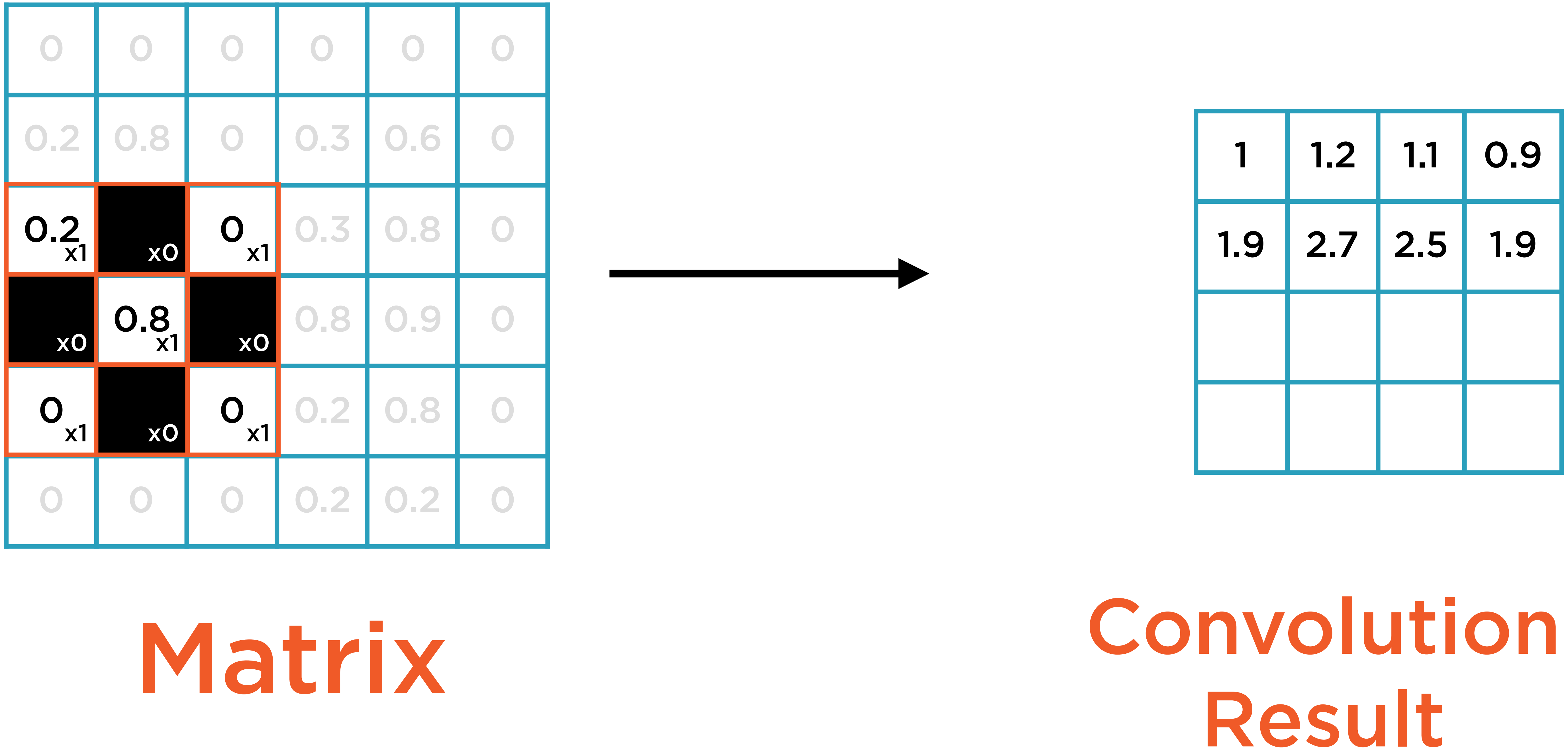
0	0	0	0	0	0
0.2	0.8	0	0.3 x1	x0	0 x1
0.2	0.9	0	x0	0.8 x1	x0
0.3	0.8	0.7	0.8 x1	x0	0 x1
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

Matrix

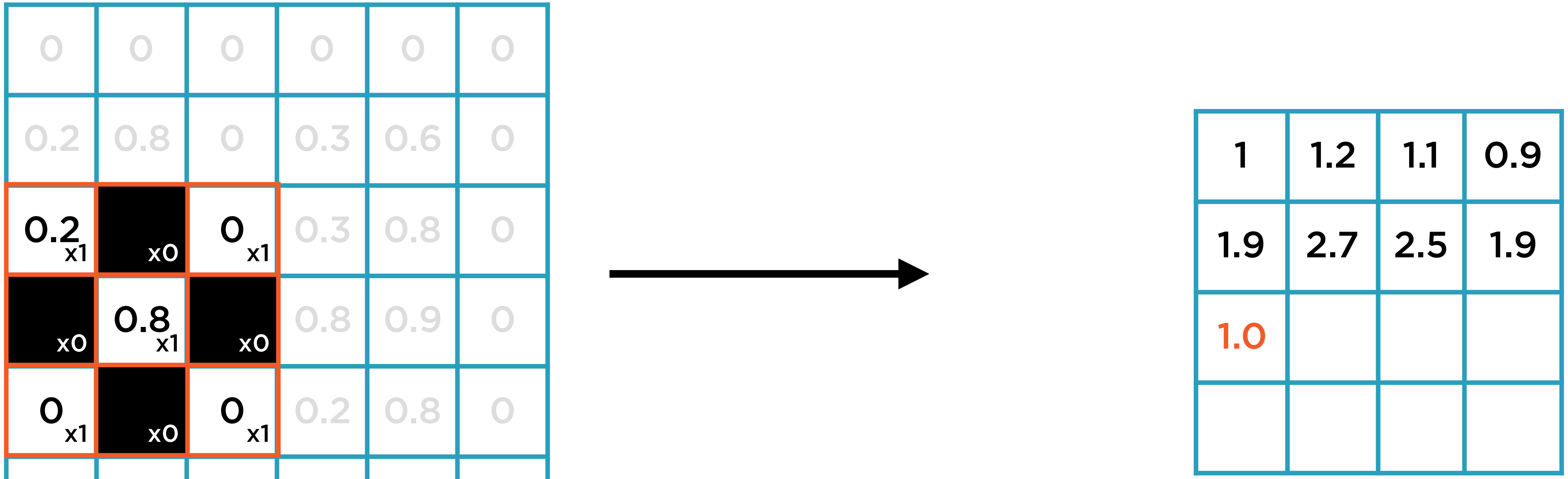
1	1.2	1.1	0.9
1.9	2.7	2.5	1.9

Convolution Result

Convolution



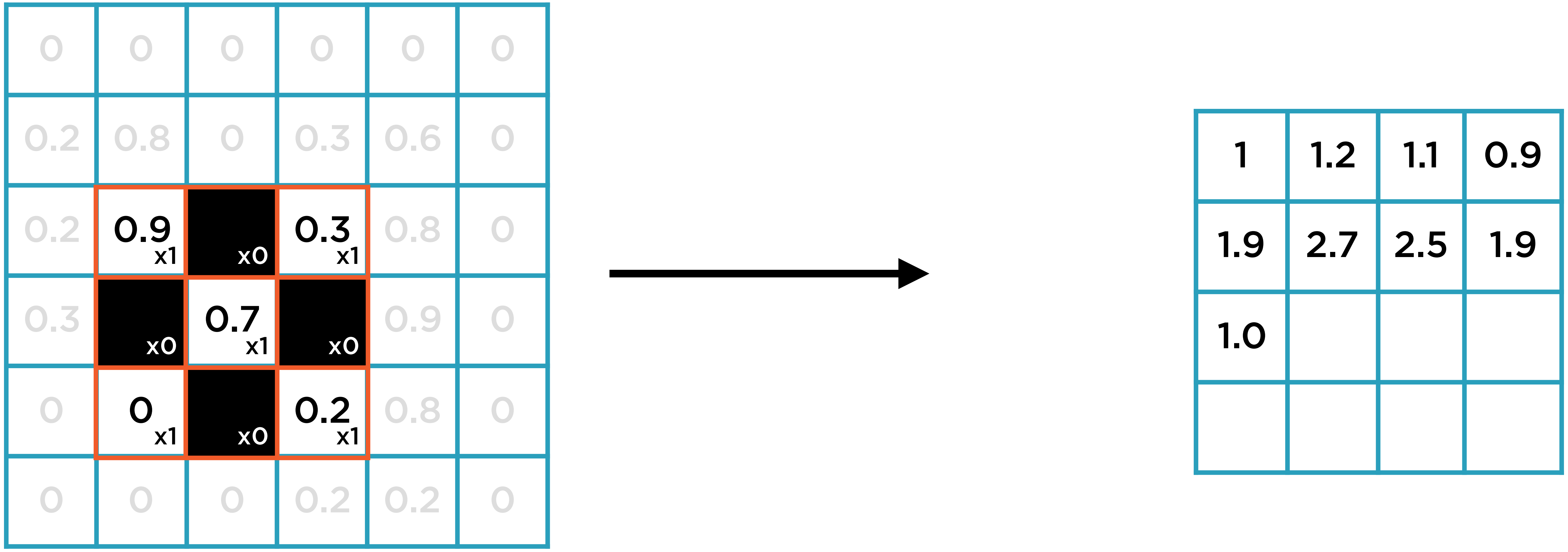
Convolution



Matrix

Convolution Result

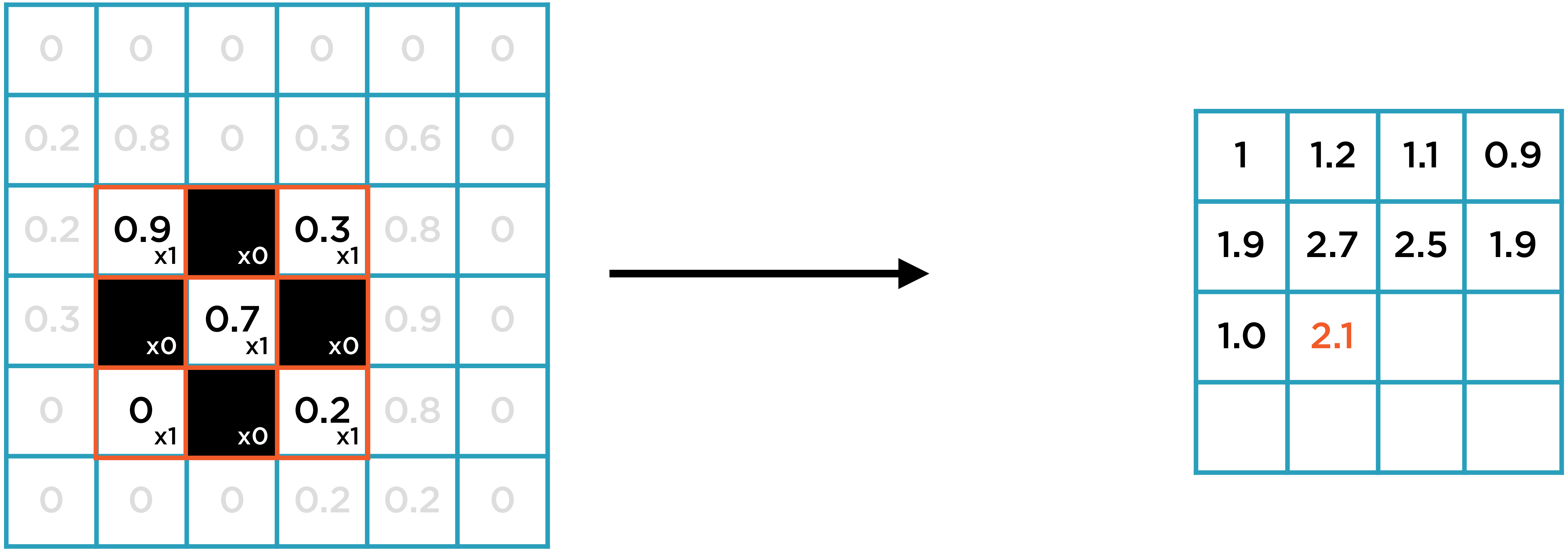
Convolution



Matrix

Convolution Result

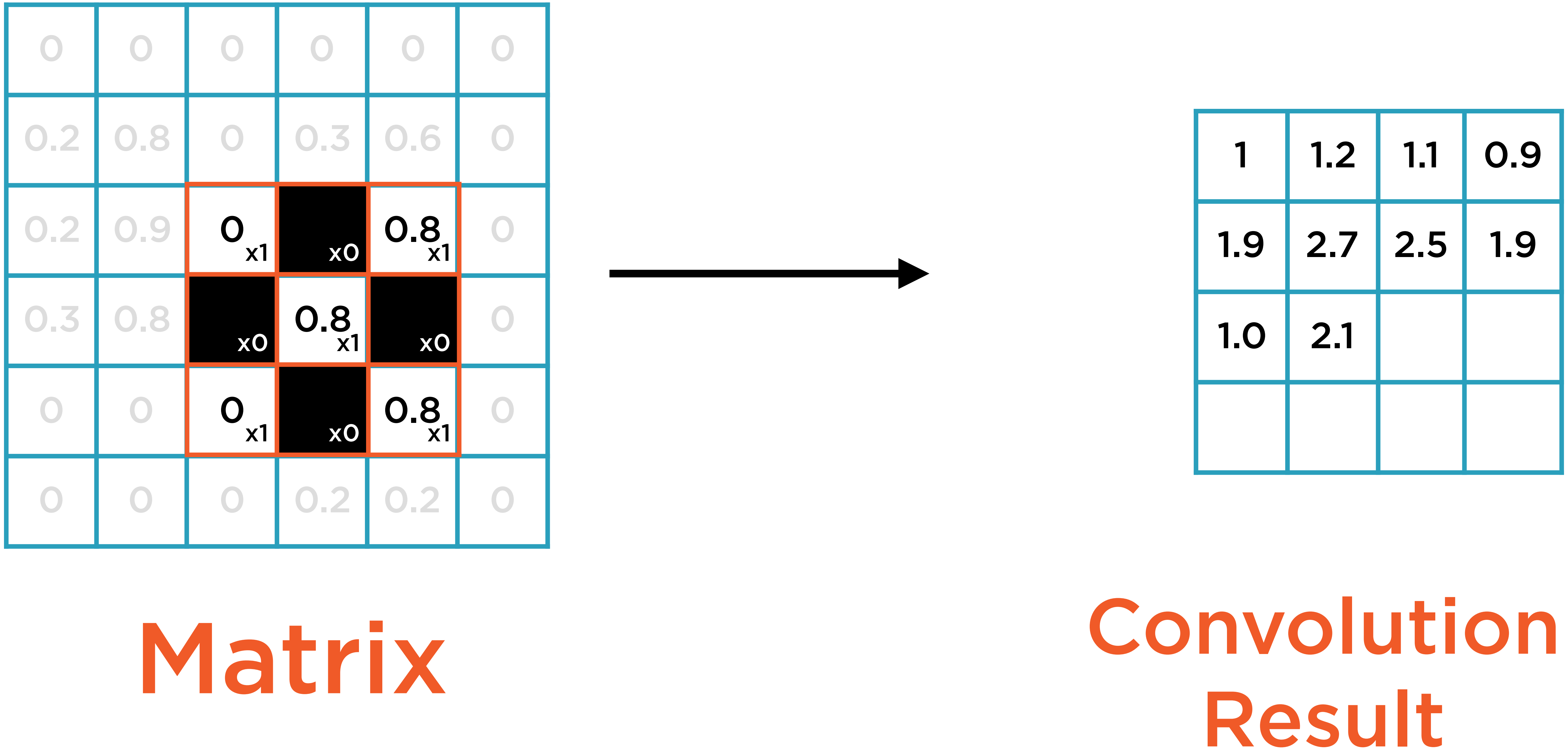
Convolution



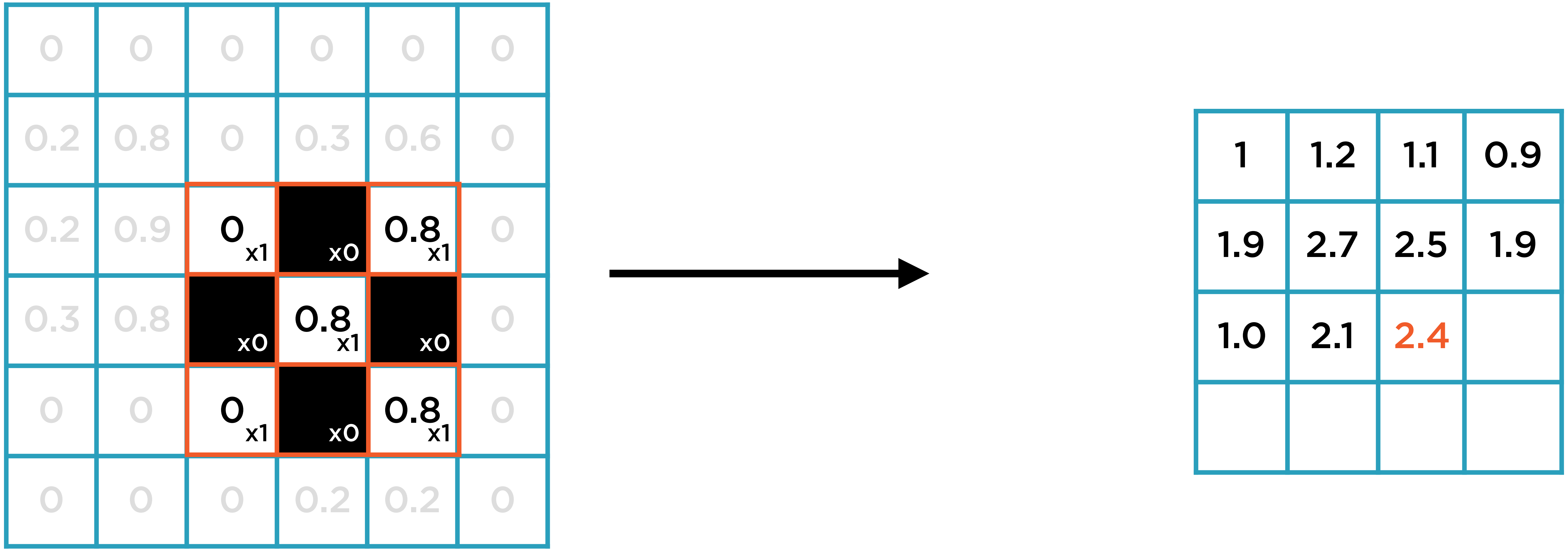
Matrix

Convolution Result

Convolution



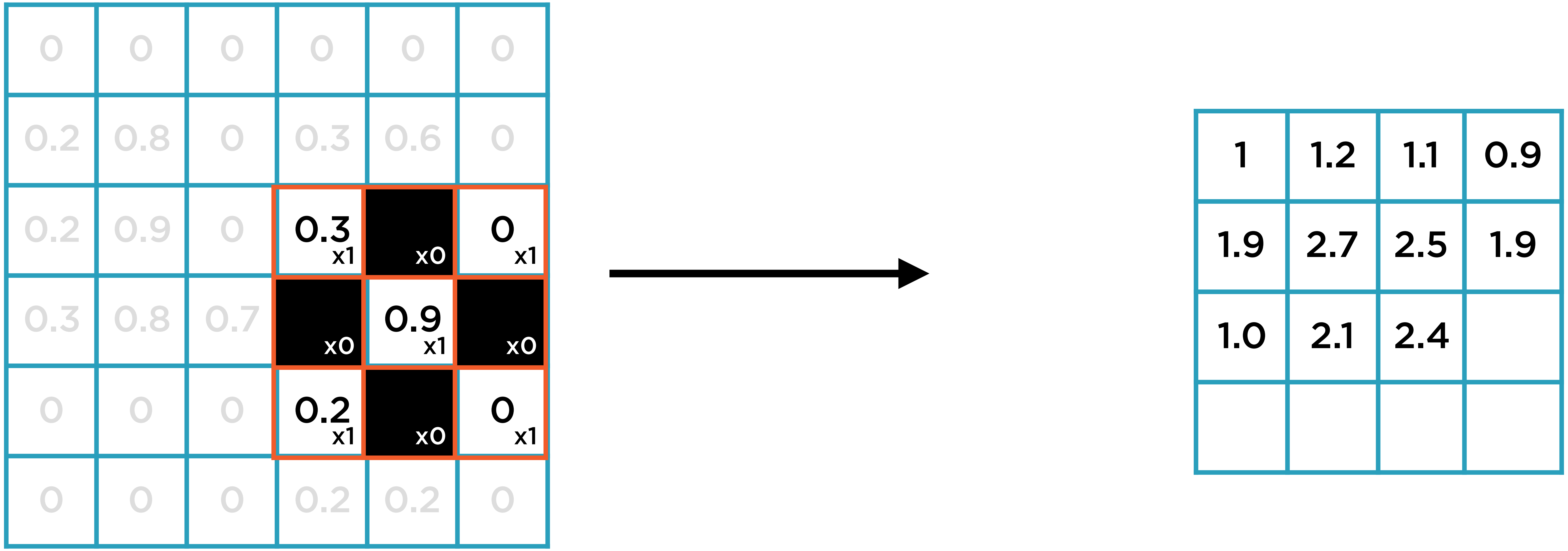
Convolution



Matrix

Convolution Result

Convolution



Matrix

Convolution Result

Convolution

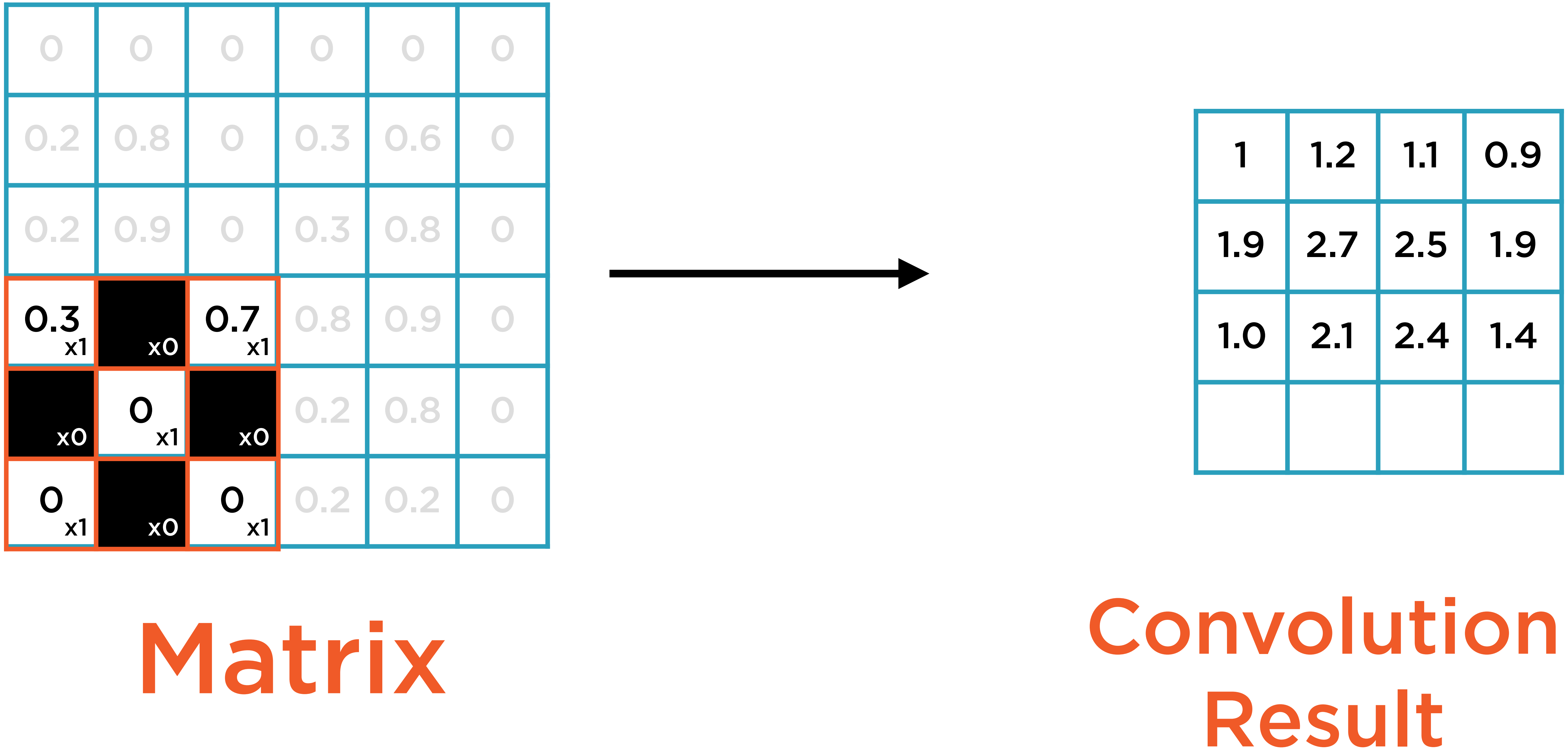
0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3 x1	x0	0 x1
0.3	0.8	0.7	x0	0.9 x1	x0
0	0	0	0.2 x1	x0	0 x1
0	0	0	0.2	0.2	0

Matrix

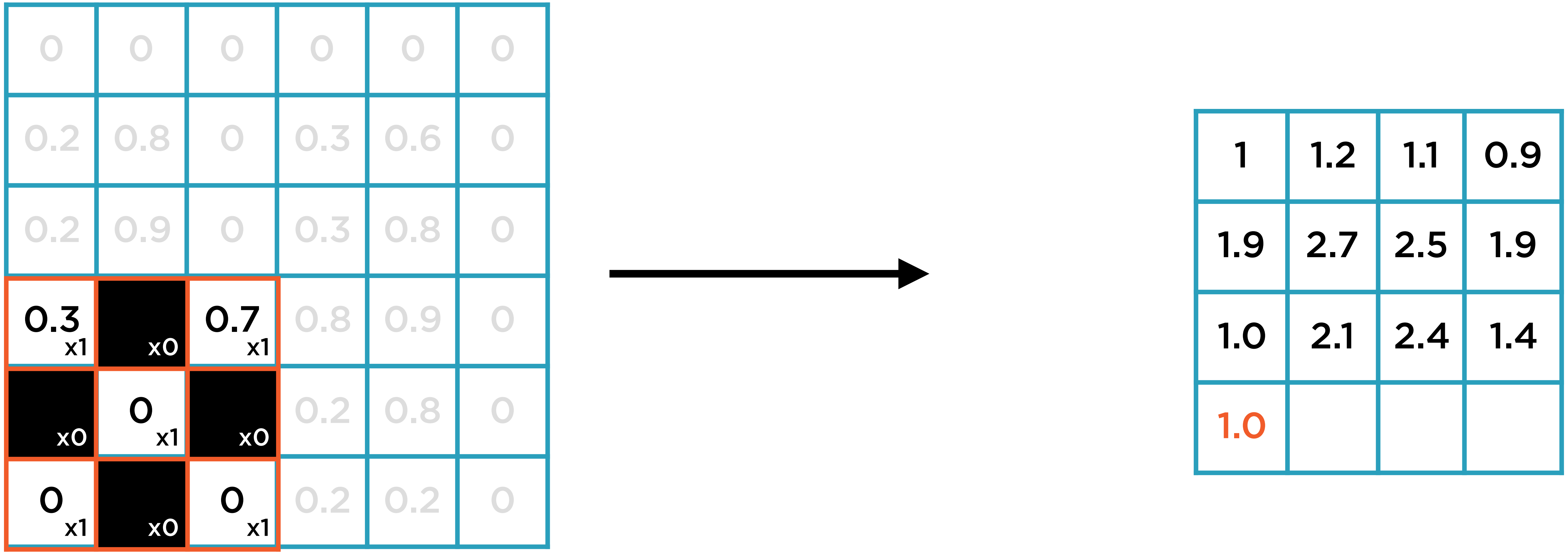
1	1.2	1.1	0.9
1.9	2.7	2.5	1.9
1.0	2.1	2.4	1.4

Convolution Result

Convolution



Convolution



Matrix

Convolution Result

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 x1	x0	0.8 x1	0.9	0
0	x0	0 x1	x0	0.8	0
0	0 x1	x0	0.2 x1	0.2	0

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			

Convolution Result

Convolution

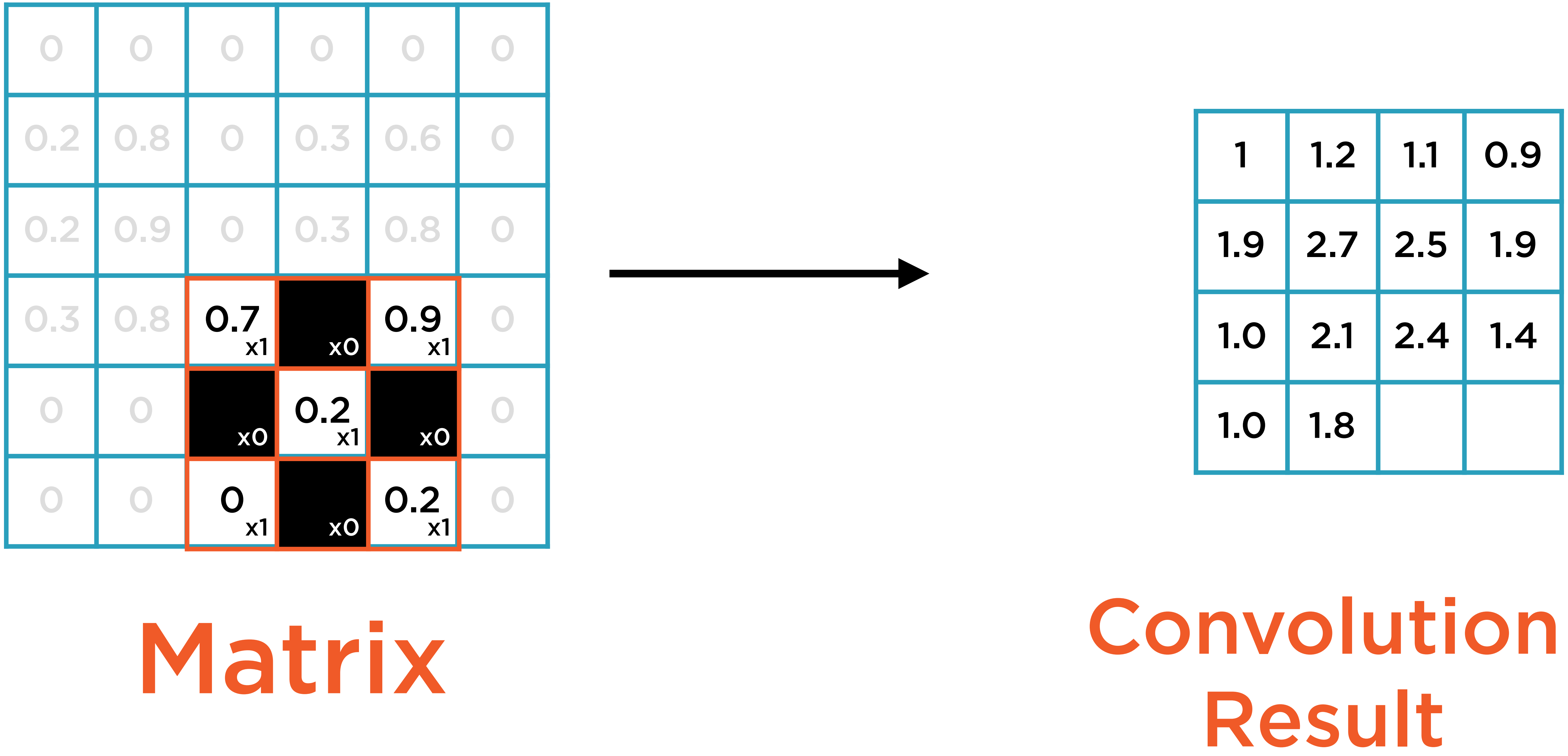
0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8 x1	x0	0.8 x1	0.9	0
0	x0	0 x1	x0	0.8	0
0	0 x1	x0	0.2 x1	0.2	0

Matrix

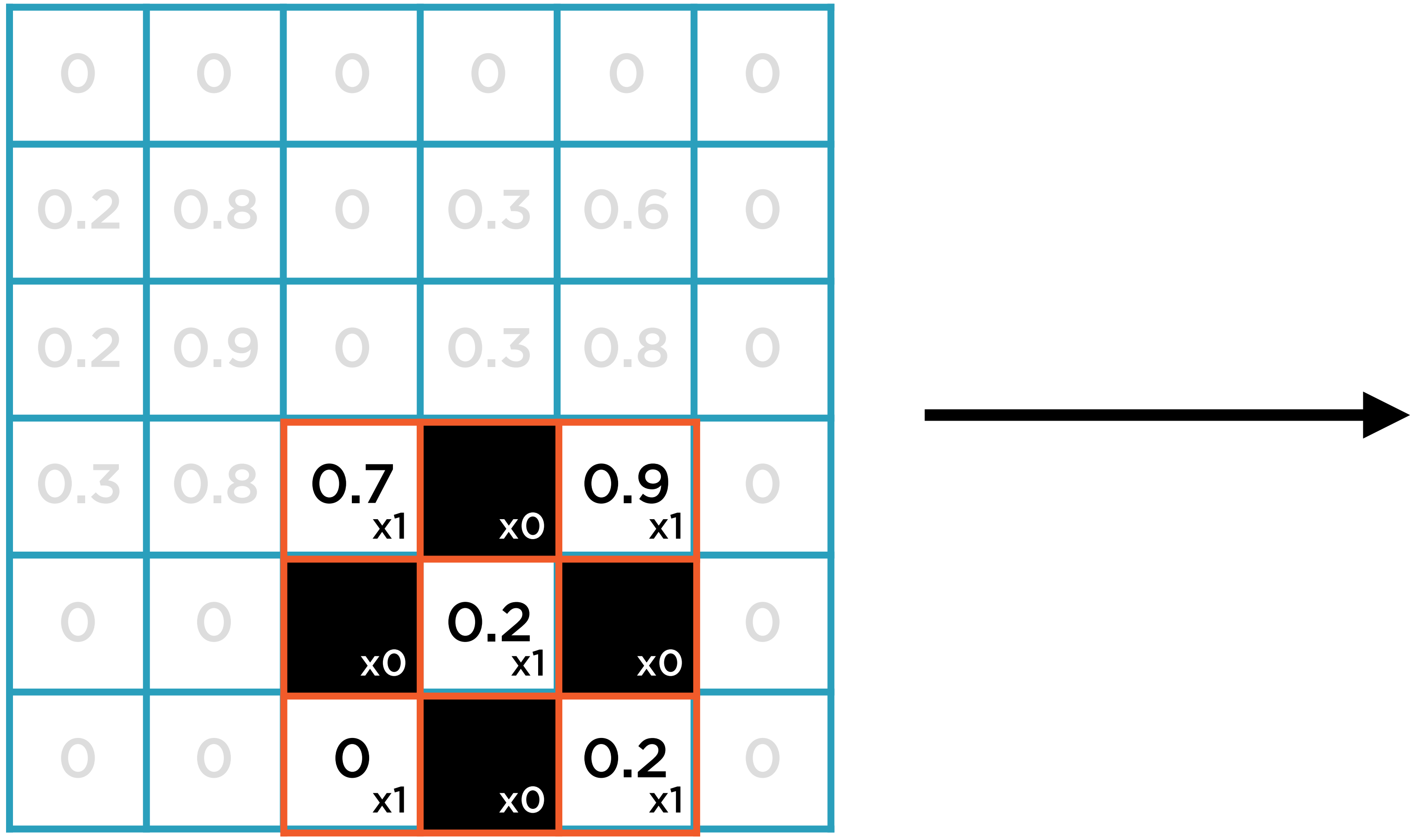
1	1.2	1.1	0.9
1.9	2.7	2.5	1.9
1.0	2.1	2.4	1.4
1.0	1.8		

Convolution Result

Convolution



Convolution



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	

Convolution Result

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

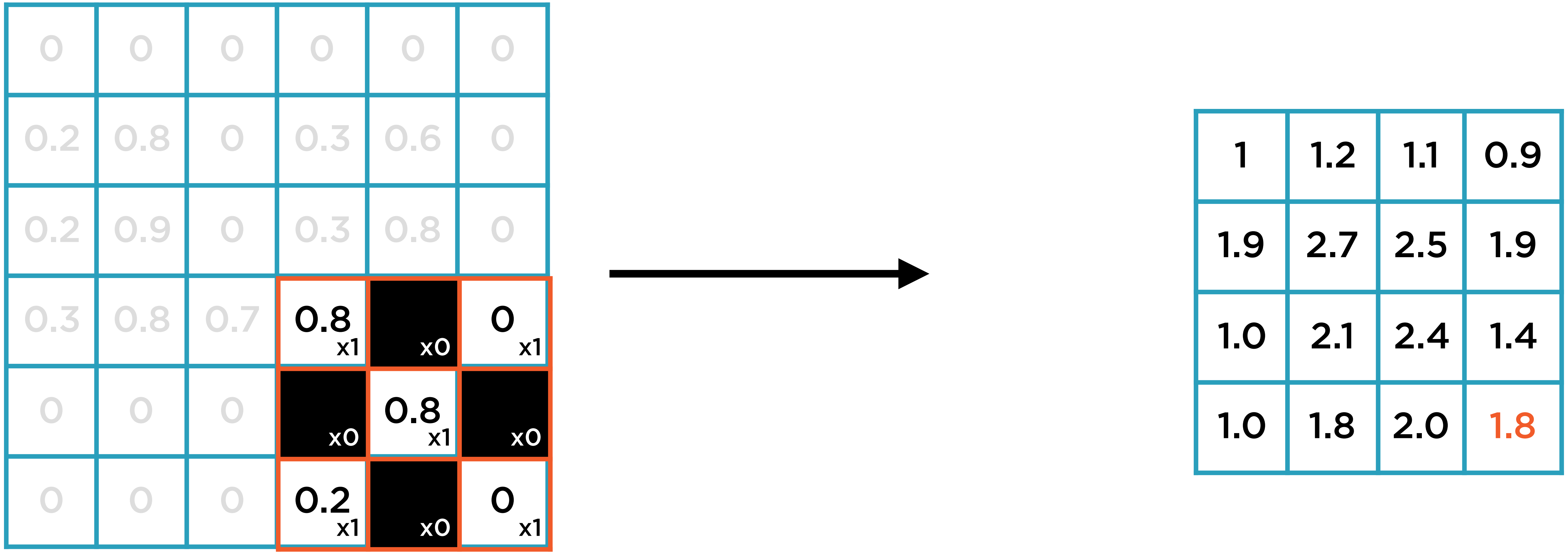
Matrix

→

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	

Convolution Result

Convolution



Matrix

Convolution Result

Convolutional Layers

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

Inspirations for CNNs

Two Dimensions

Data comes in expressed in 2D

Local Receptive Fields

Neurons focus on narrow portions

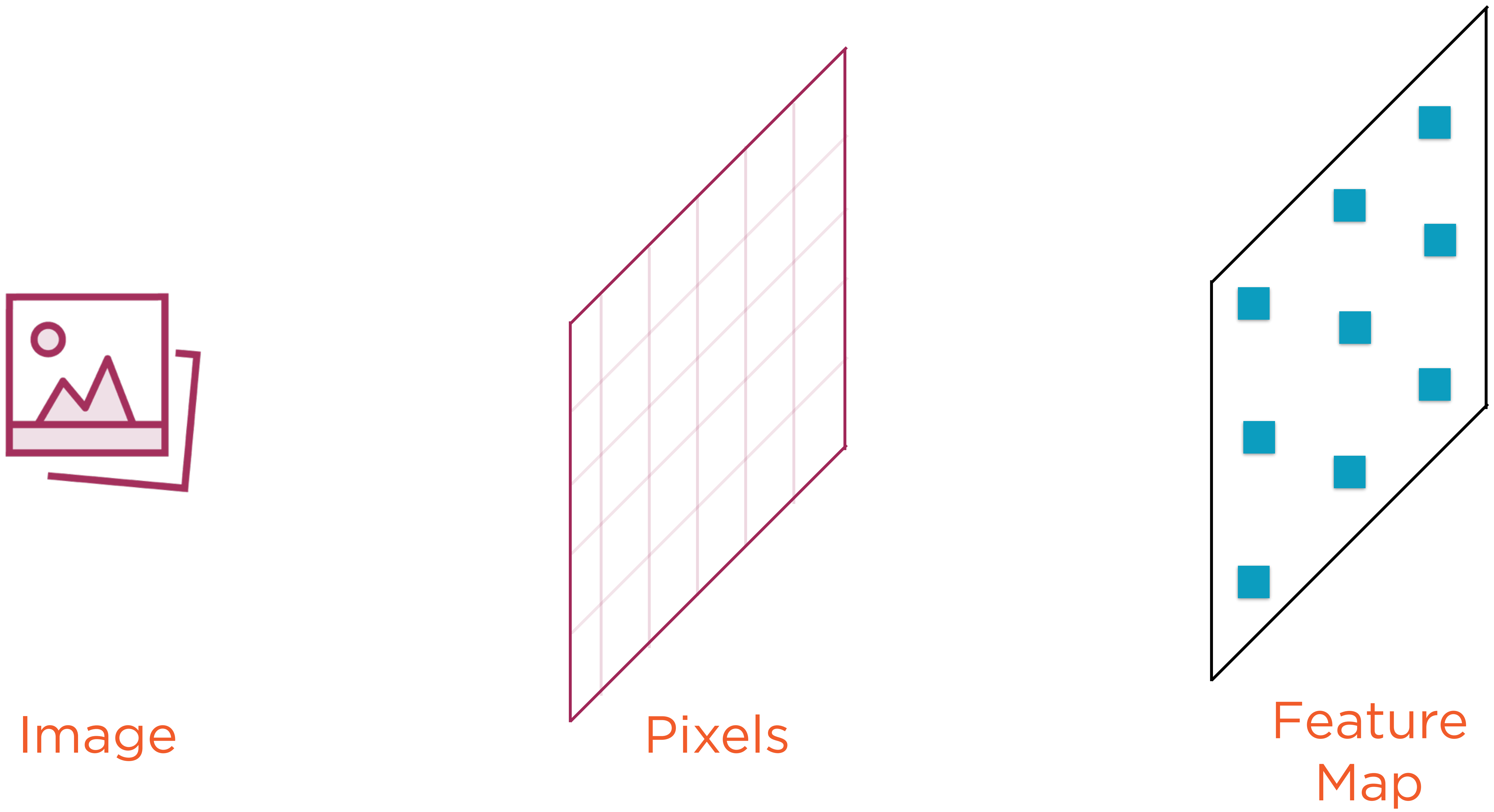
CNN Layers

Convolution layers - zoom in on specific bits of input

Successive layers aggregate inputs into higher level features

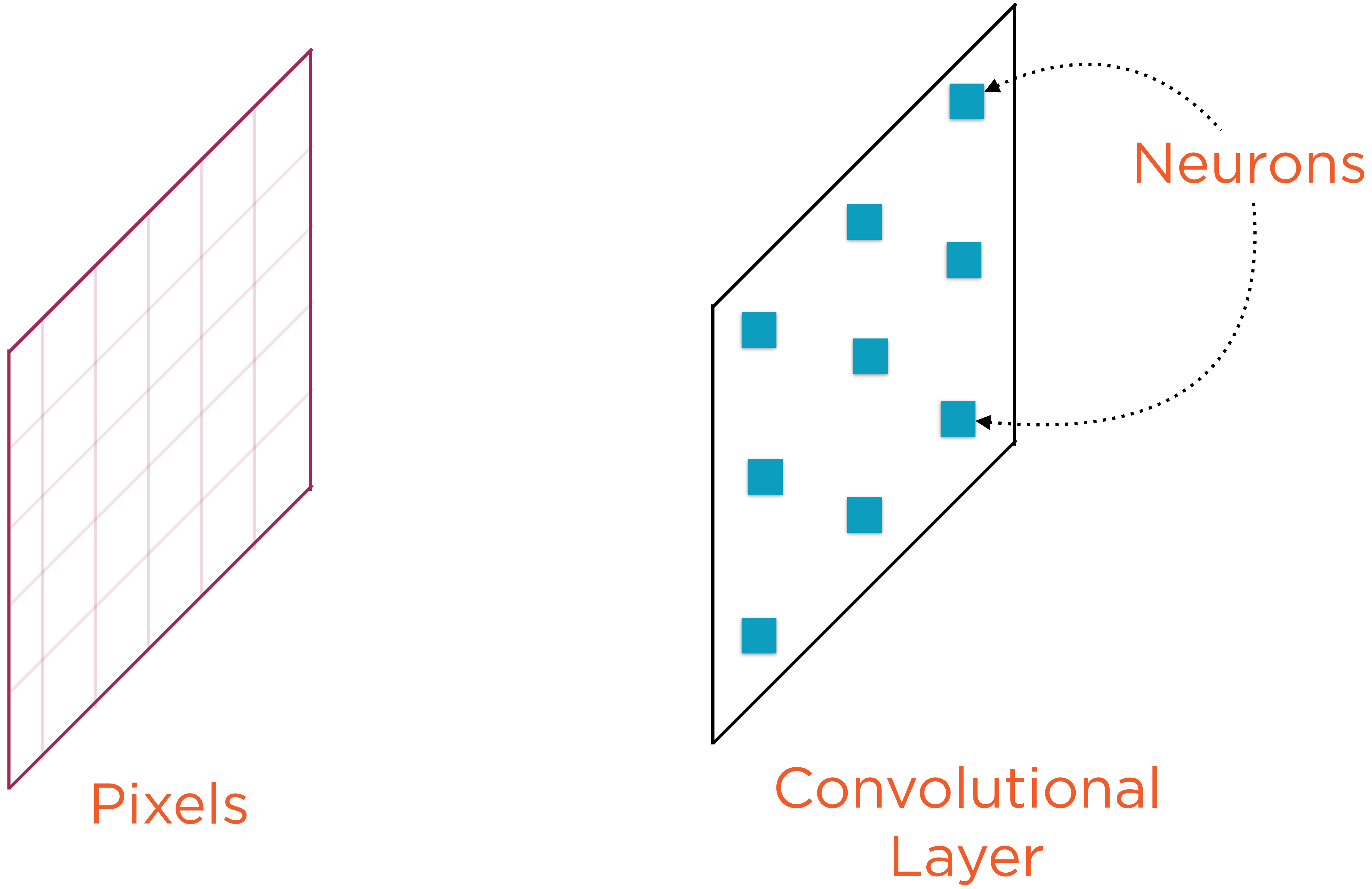
Pixels >> Lines >> Contours/Edges >> Object

Feature Maps

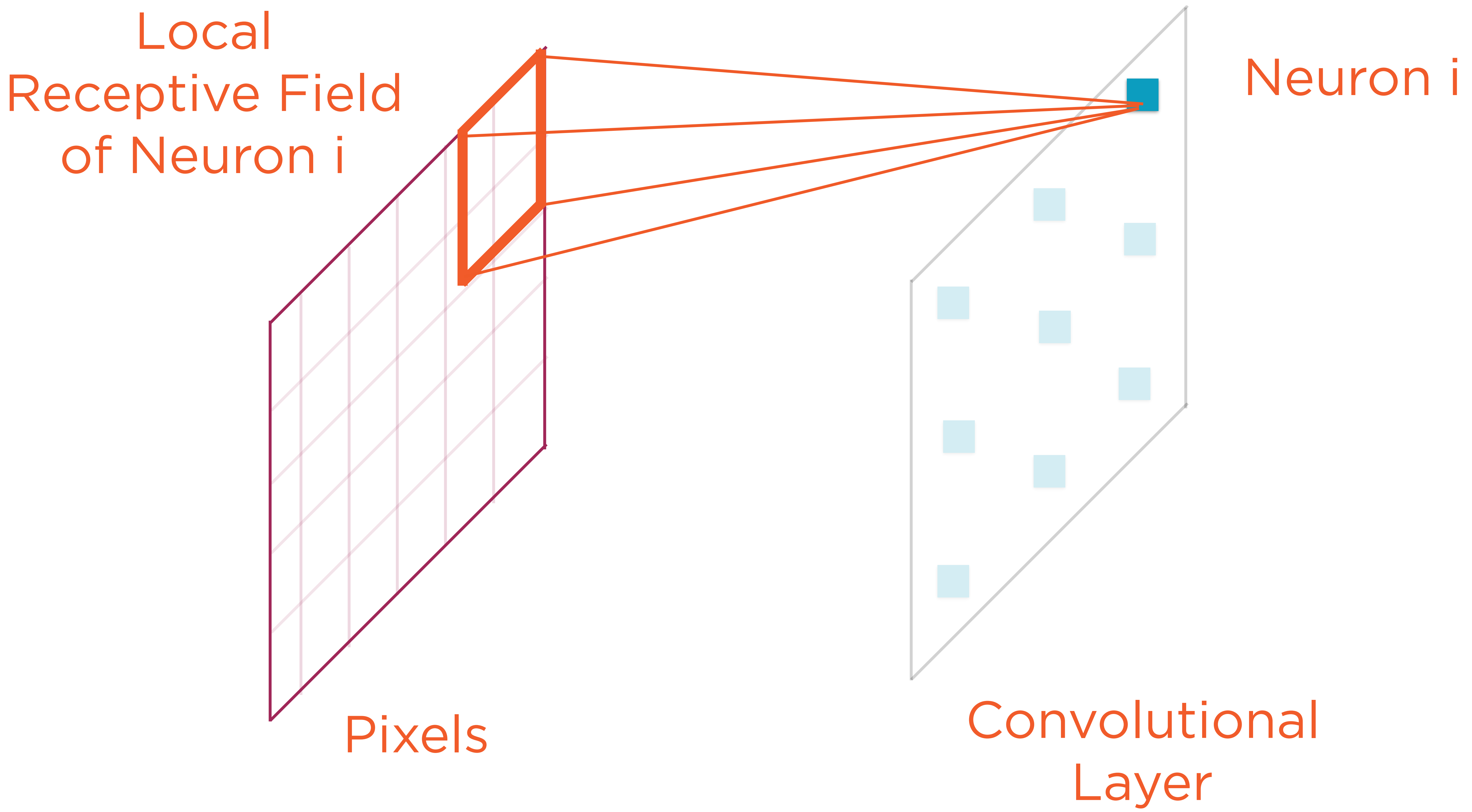


Feature maps are convolutional layers generated by applying a convolutional kernel to the input

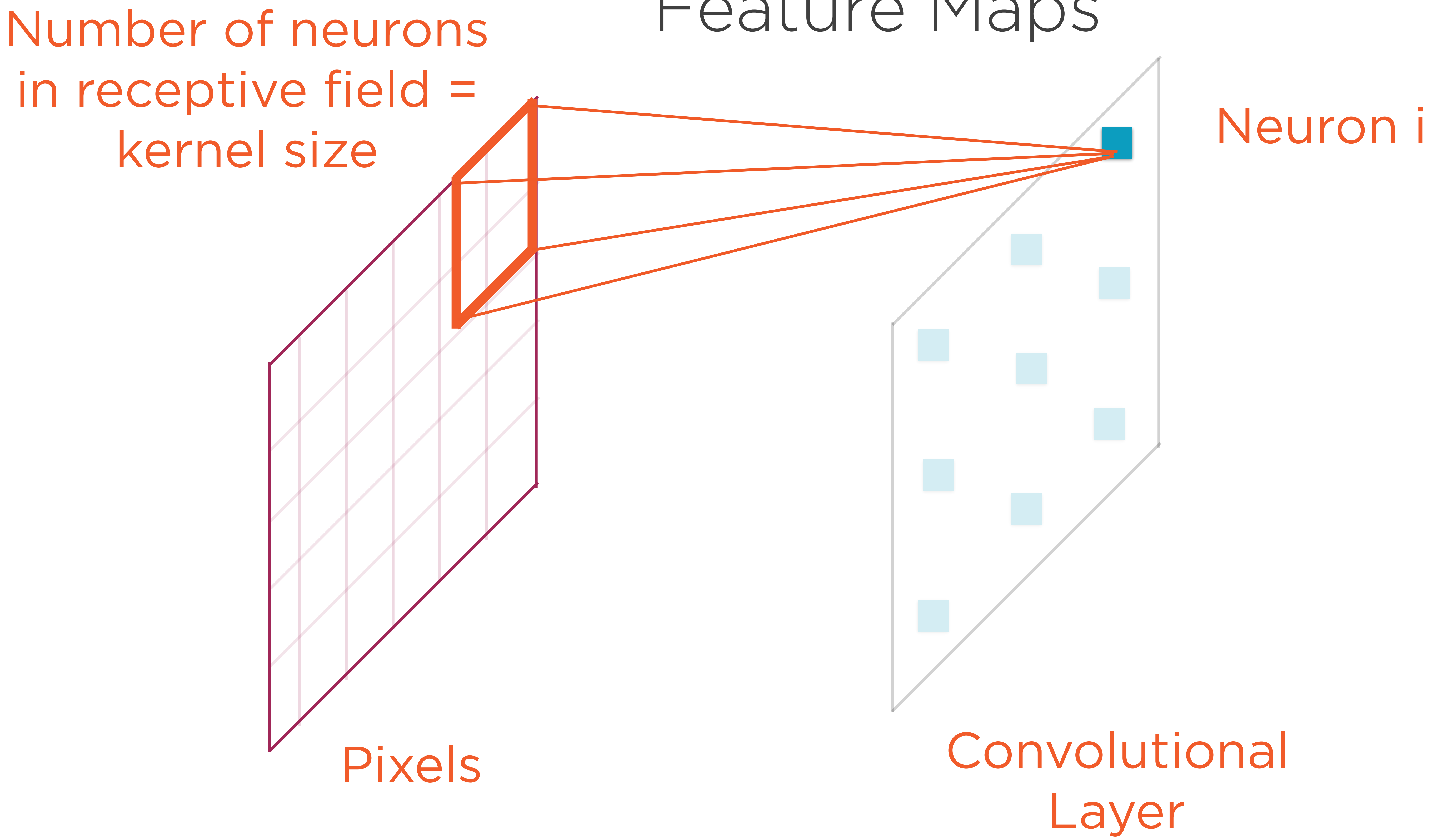
Feature Maps



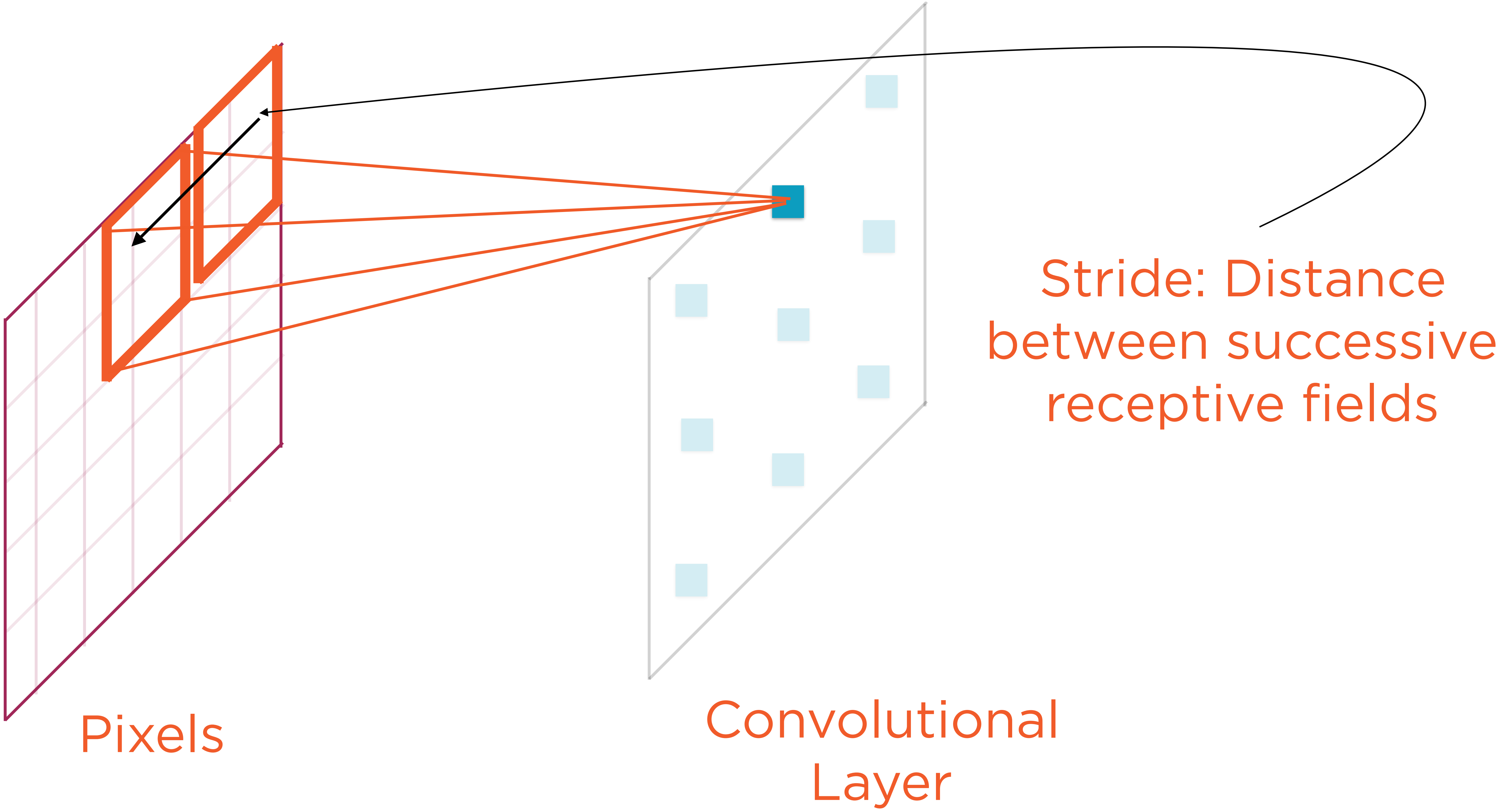
Feature Maps



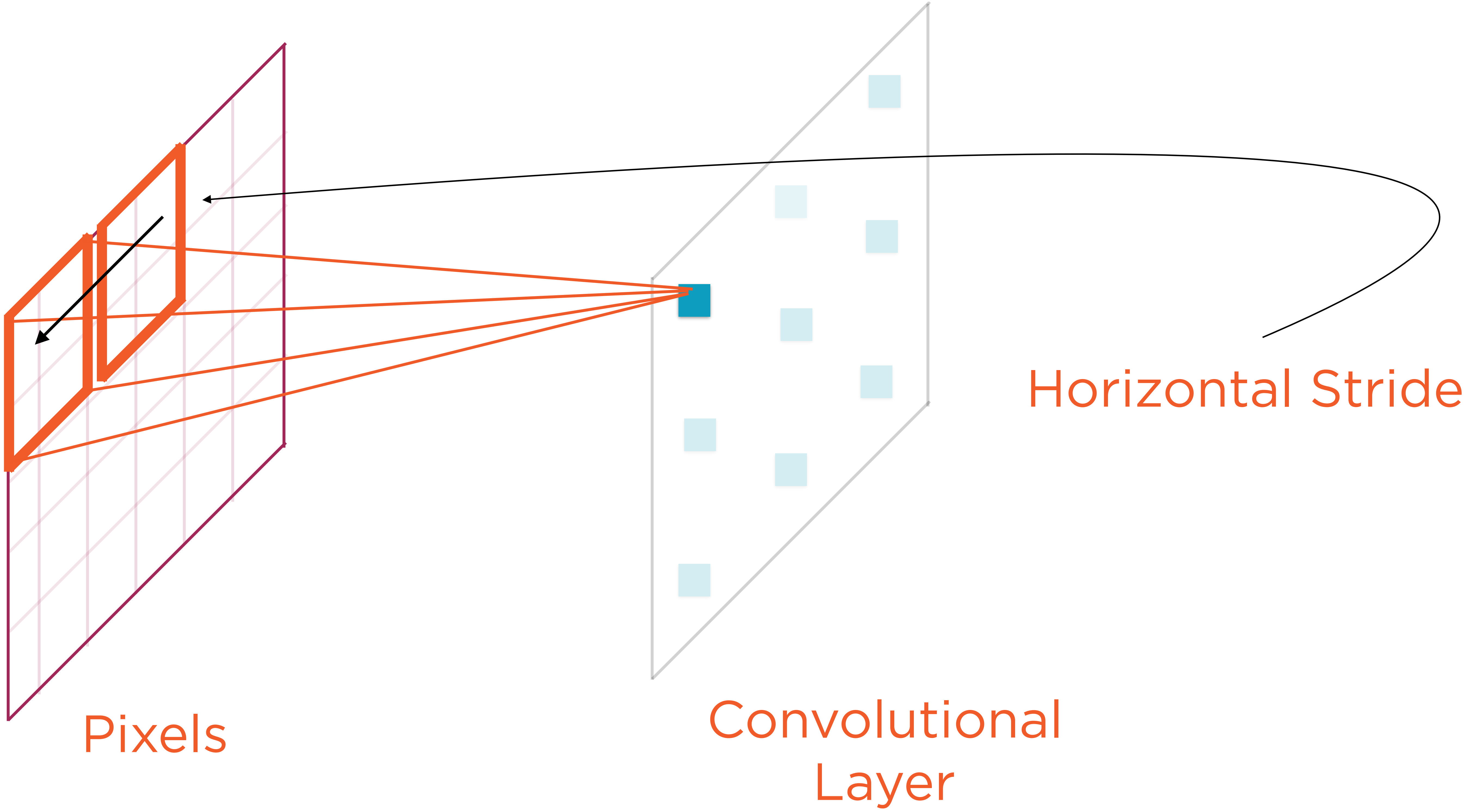
Feature Maps



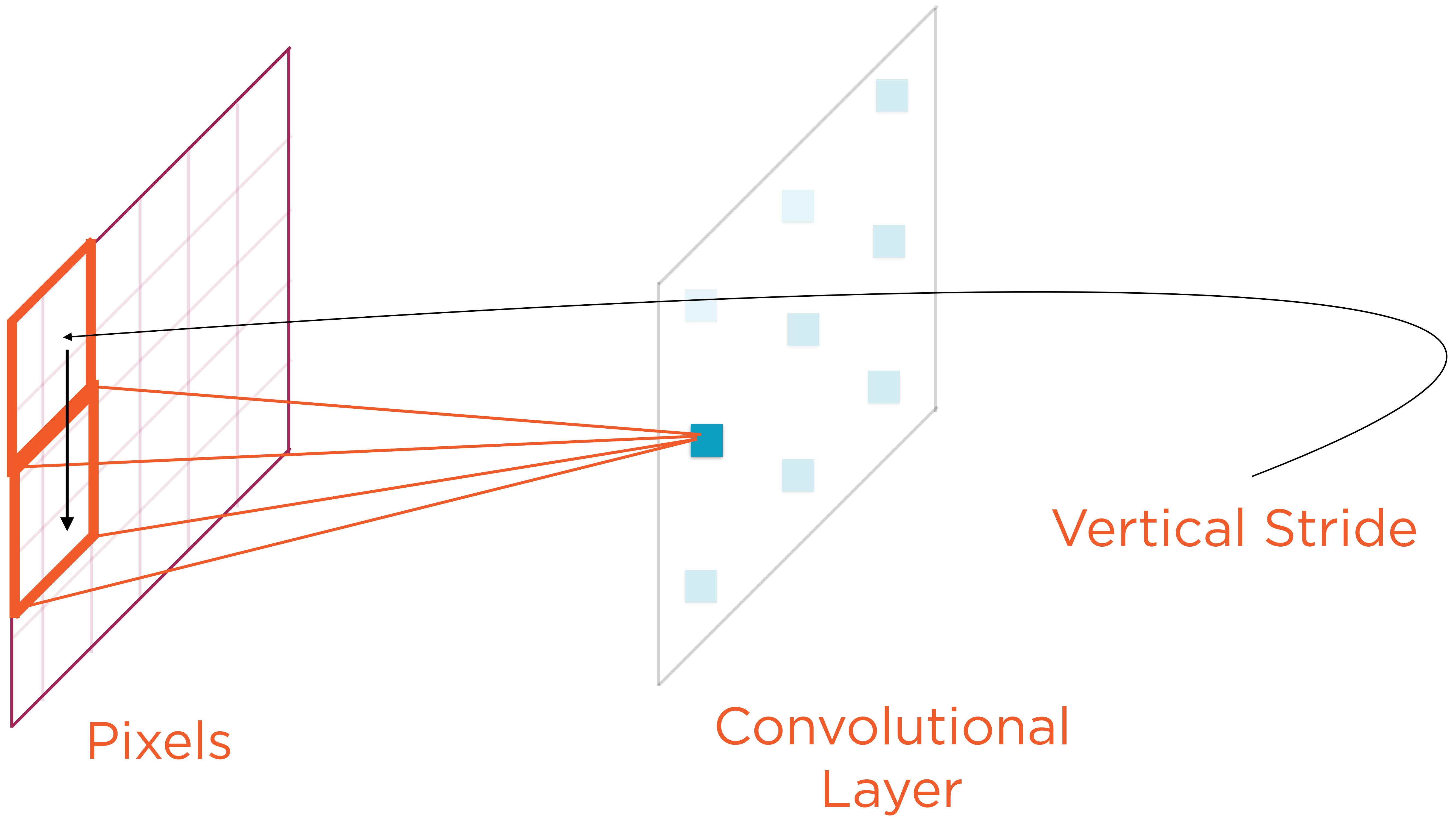
Feature Maps



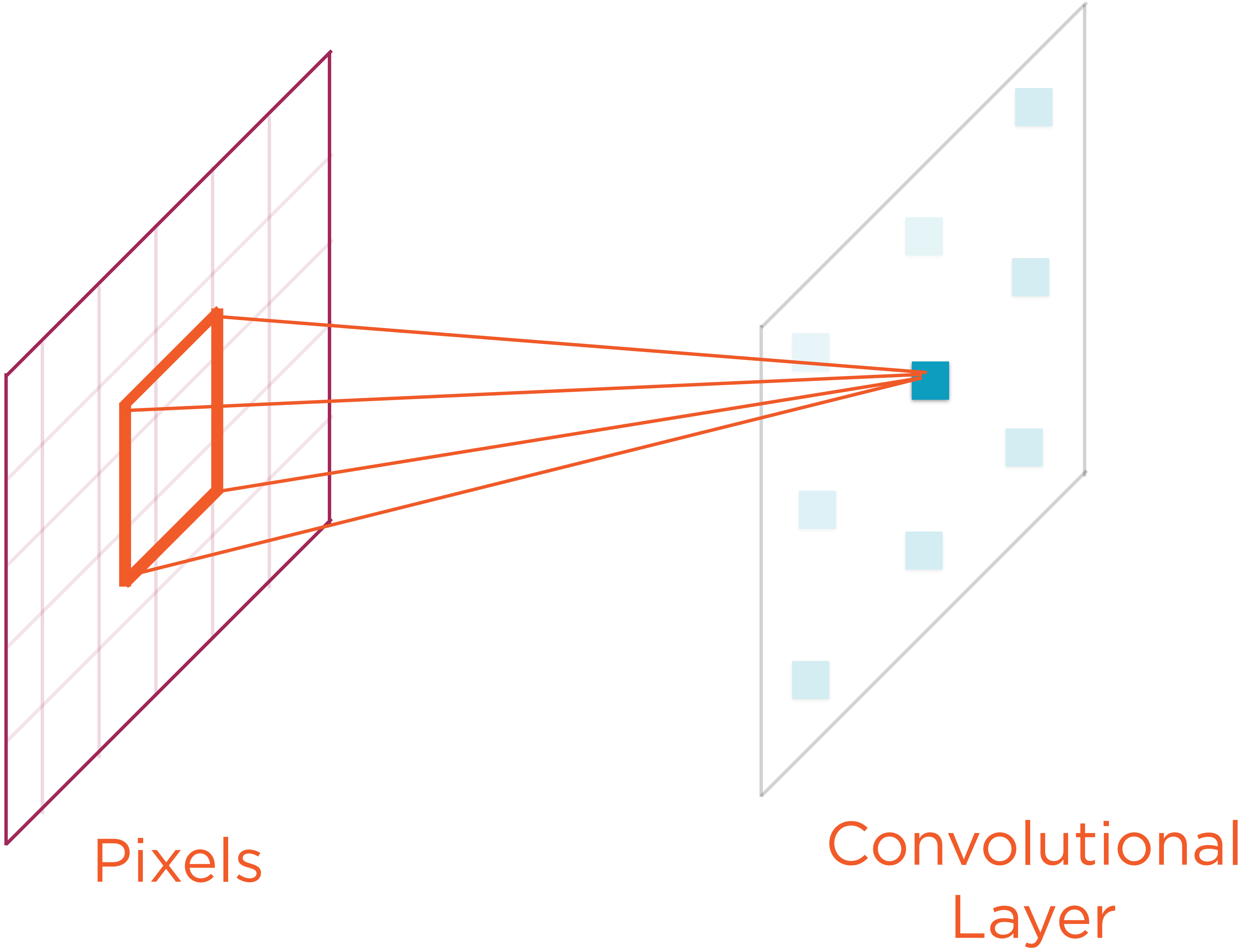
Feature Maps



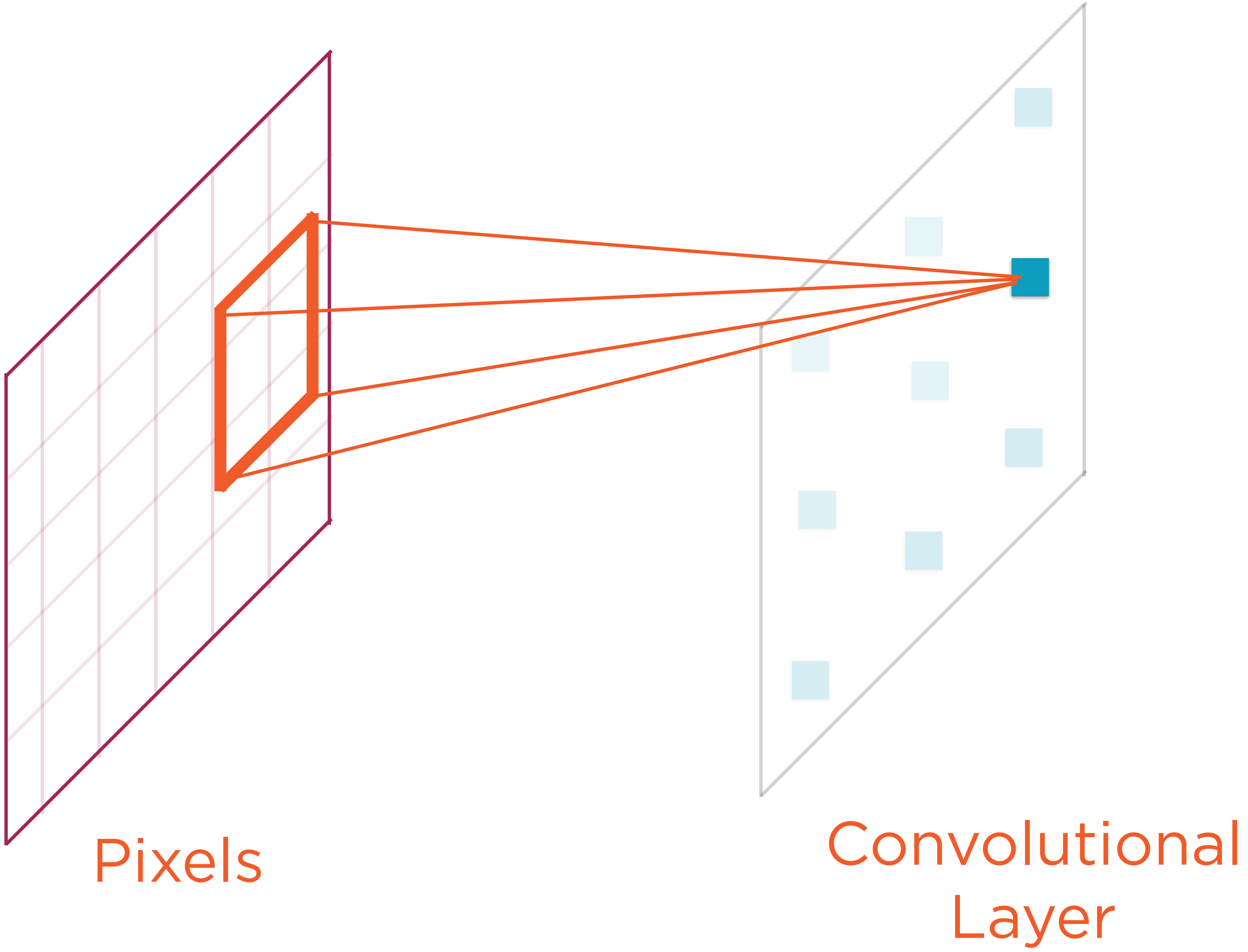
Feature Maps



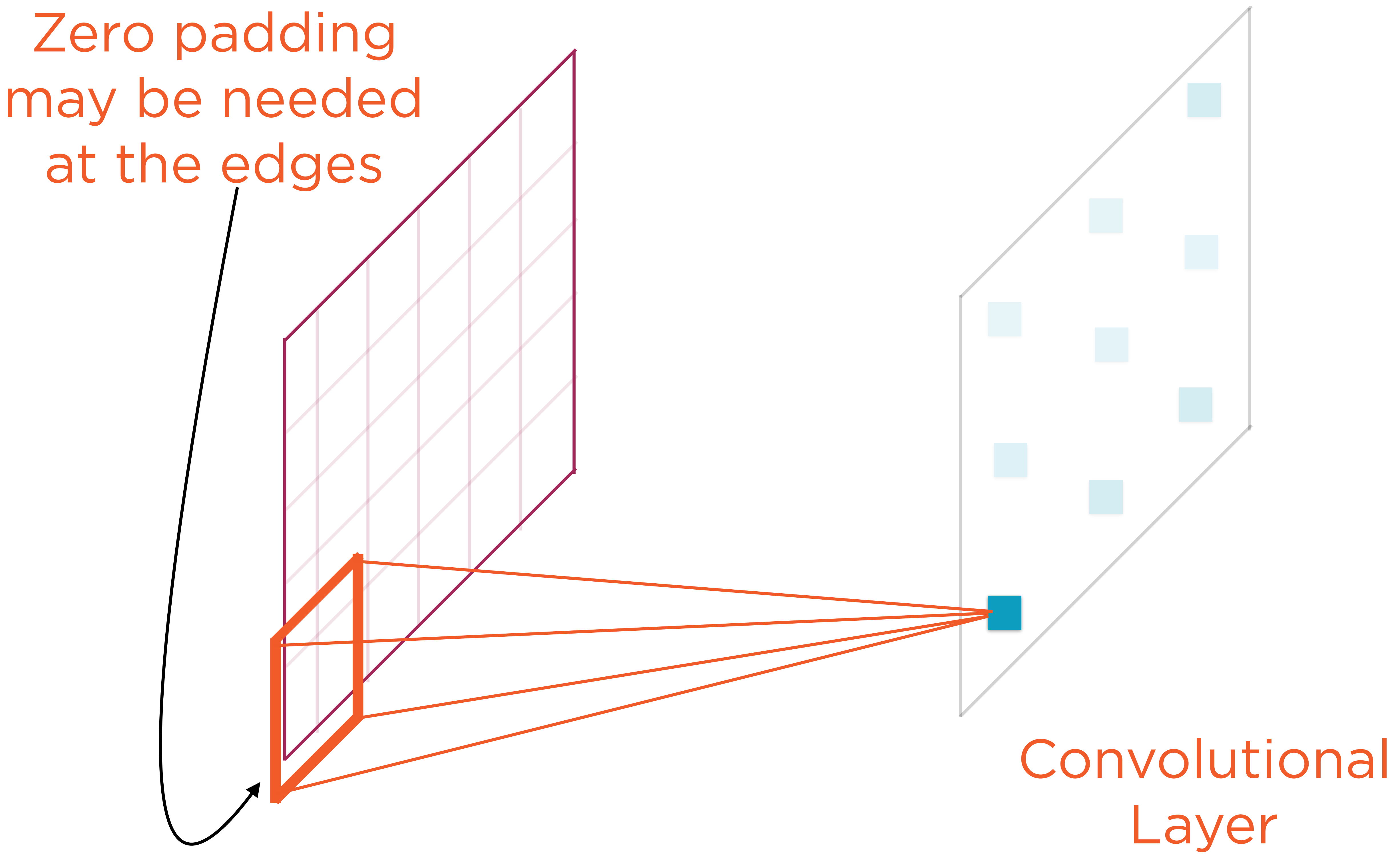
Feature Maps



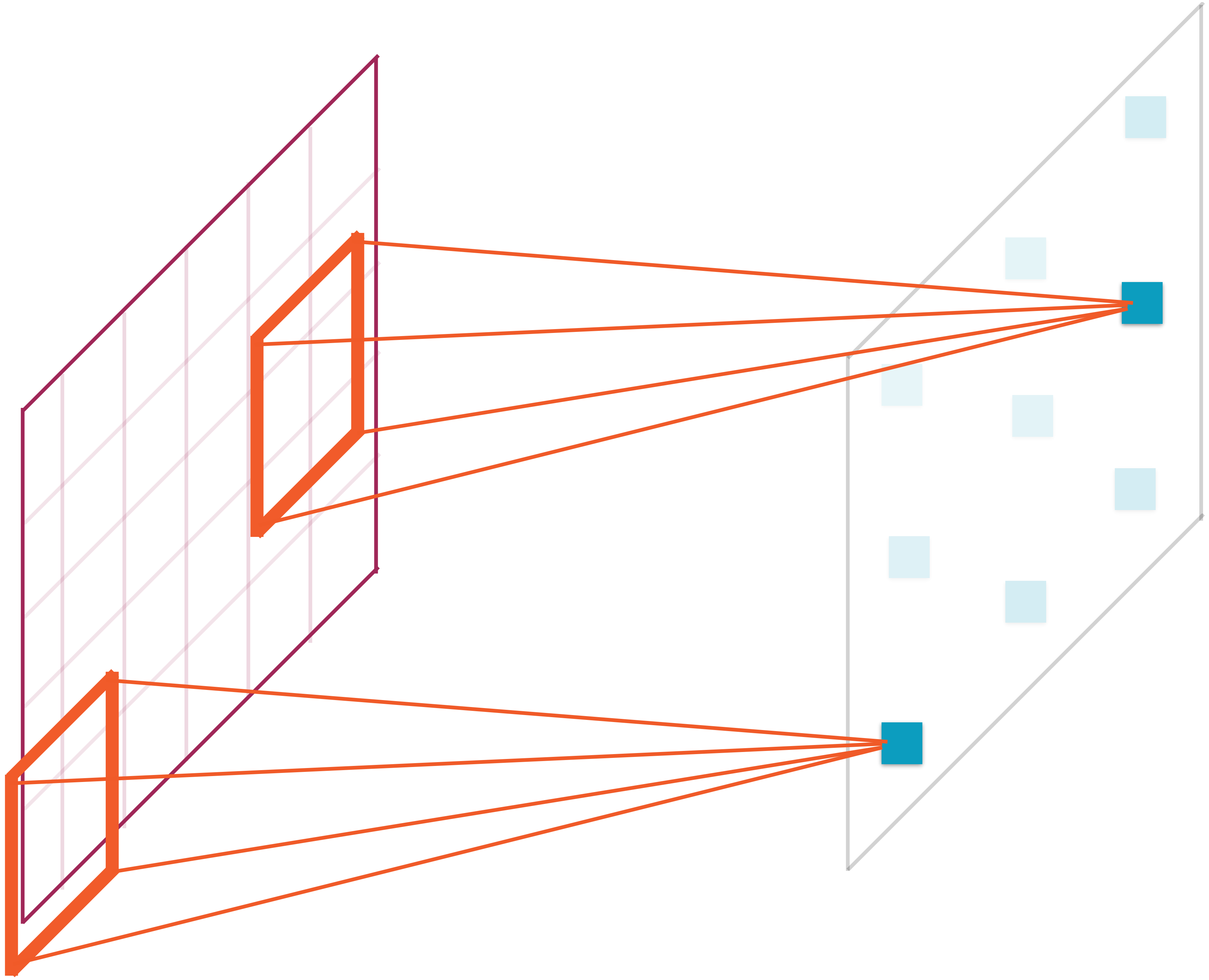
Feature Maps



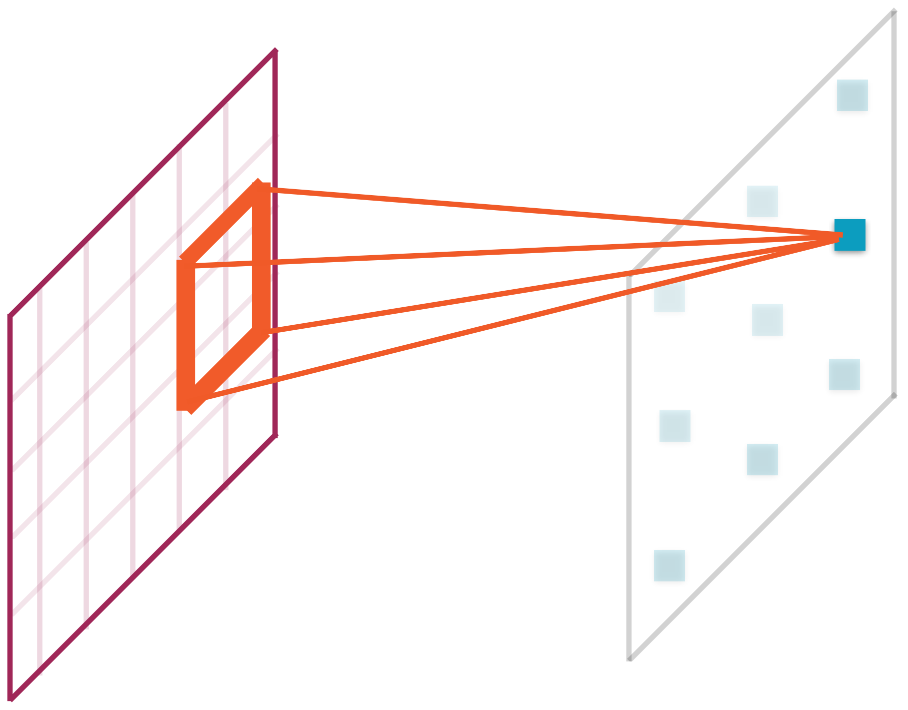
Feature Maps



Feature Maps



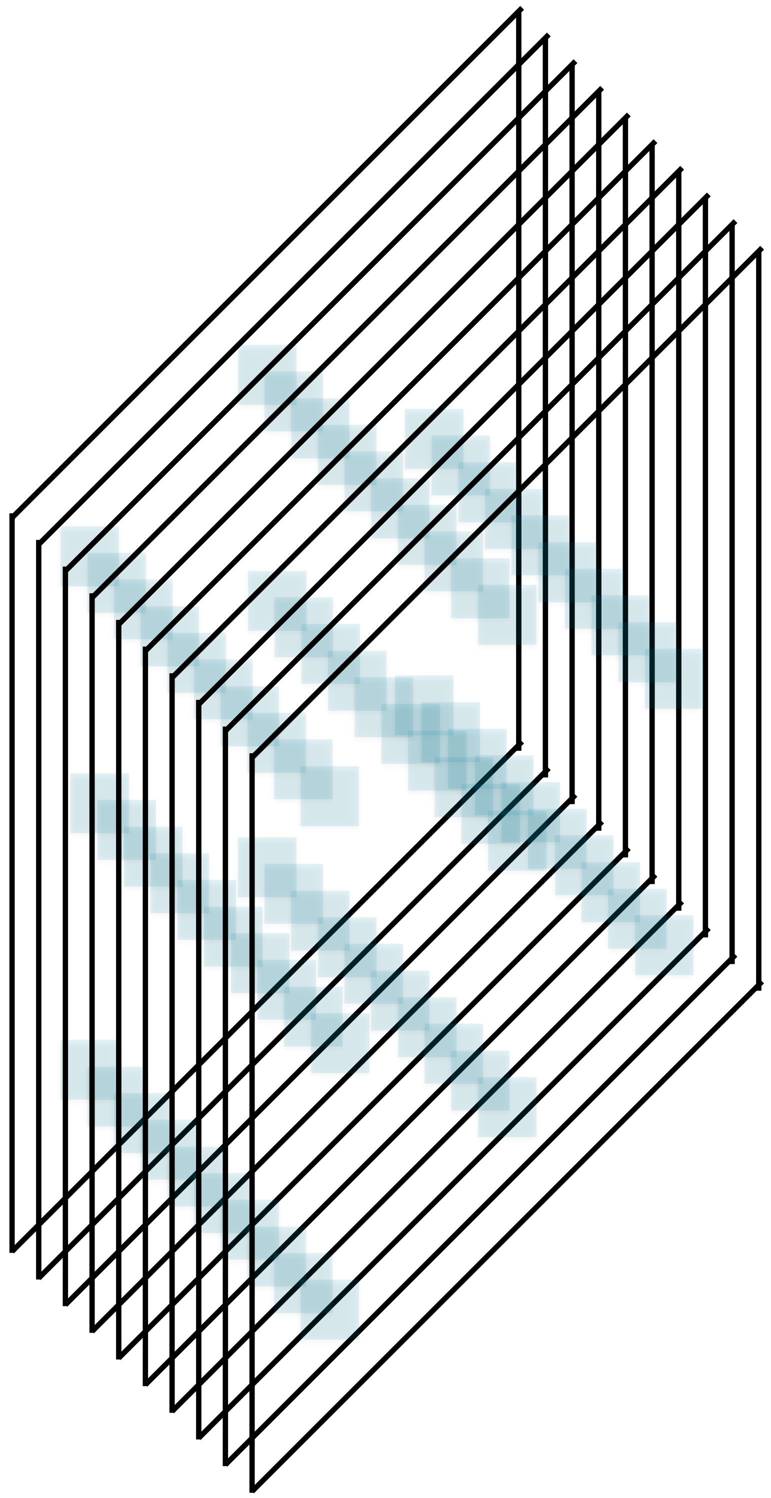
Sparse, not Dense



Feature Maps

Notice also that neurons are not connected to all pixels

CNNs are sparse neural networks

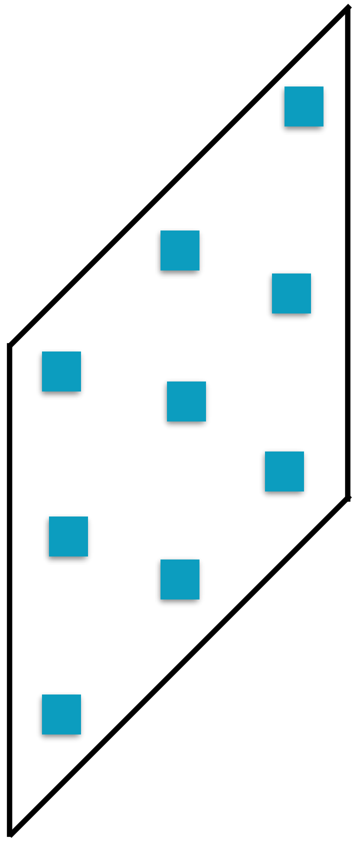


Convolutional Layer

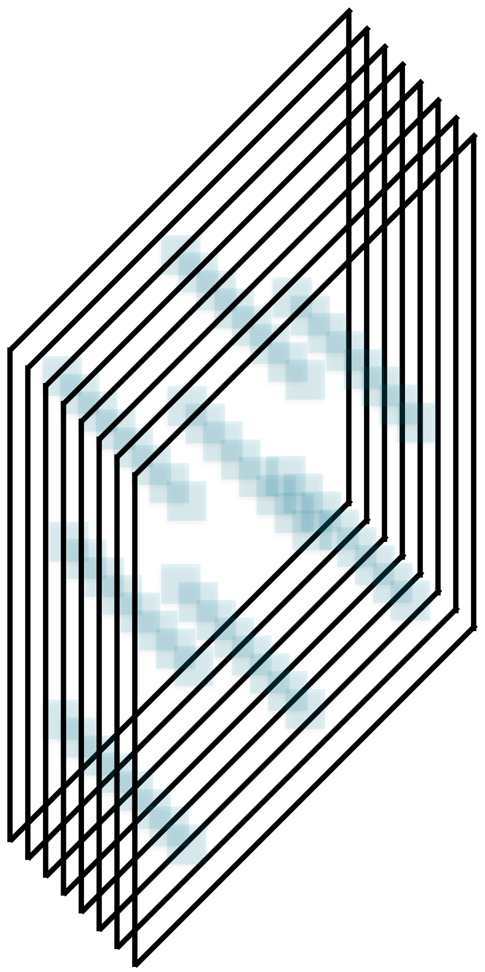
Each convolutional layer consists of several feature maps of equal sizes

The different feature maps have different parameters

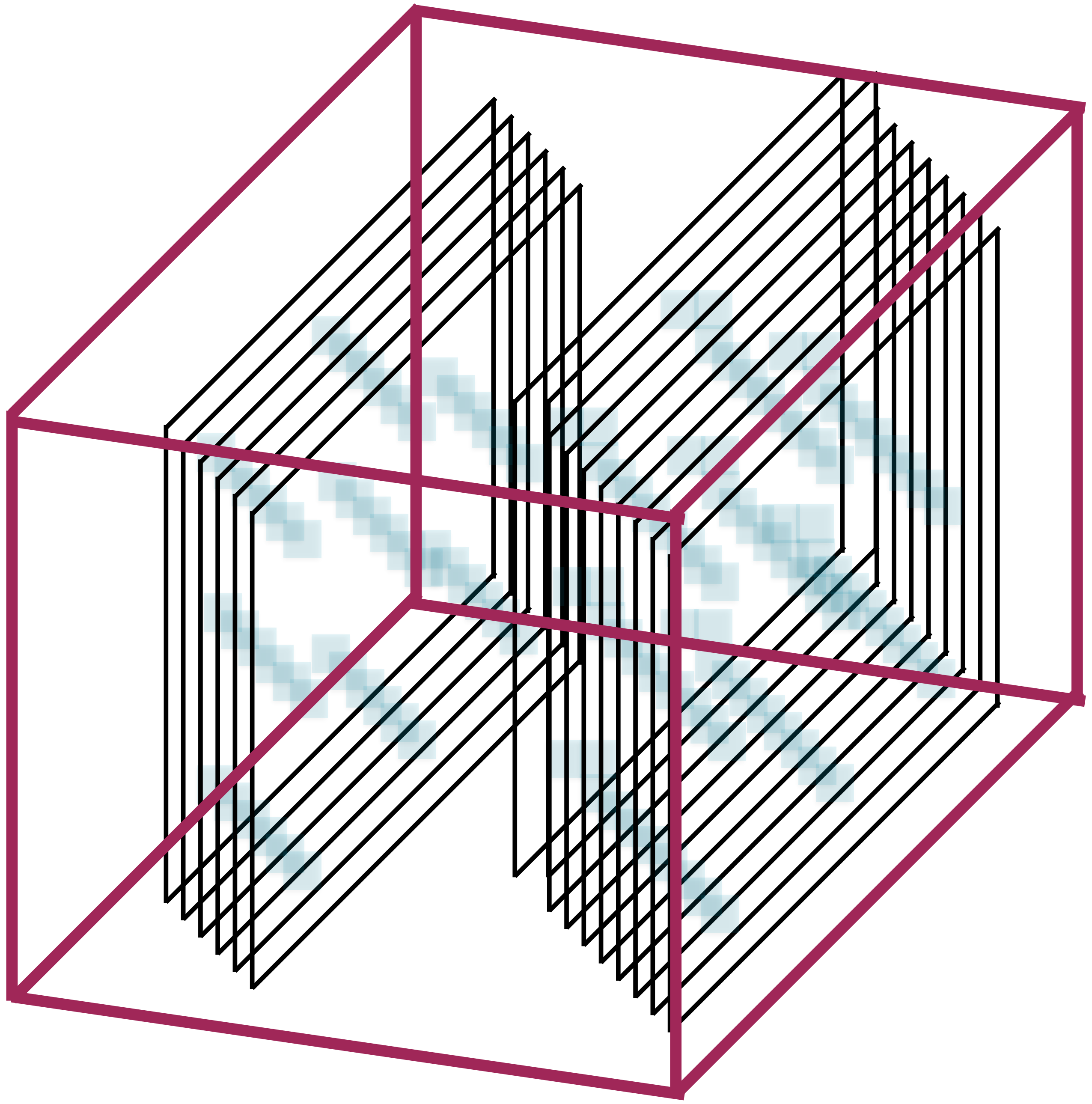
CNNs



Feature Map



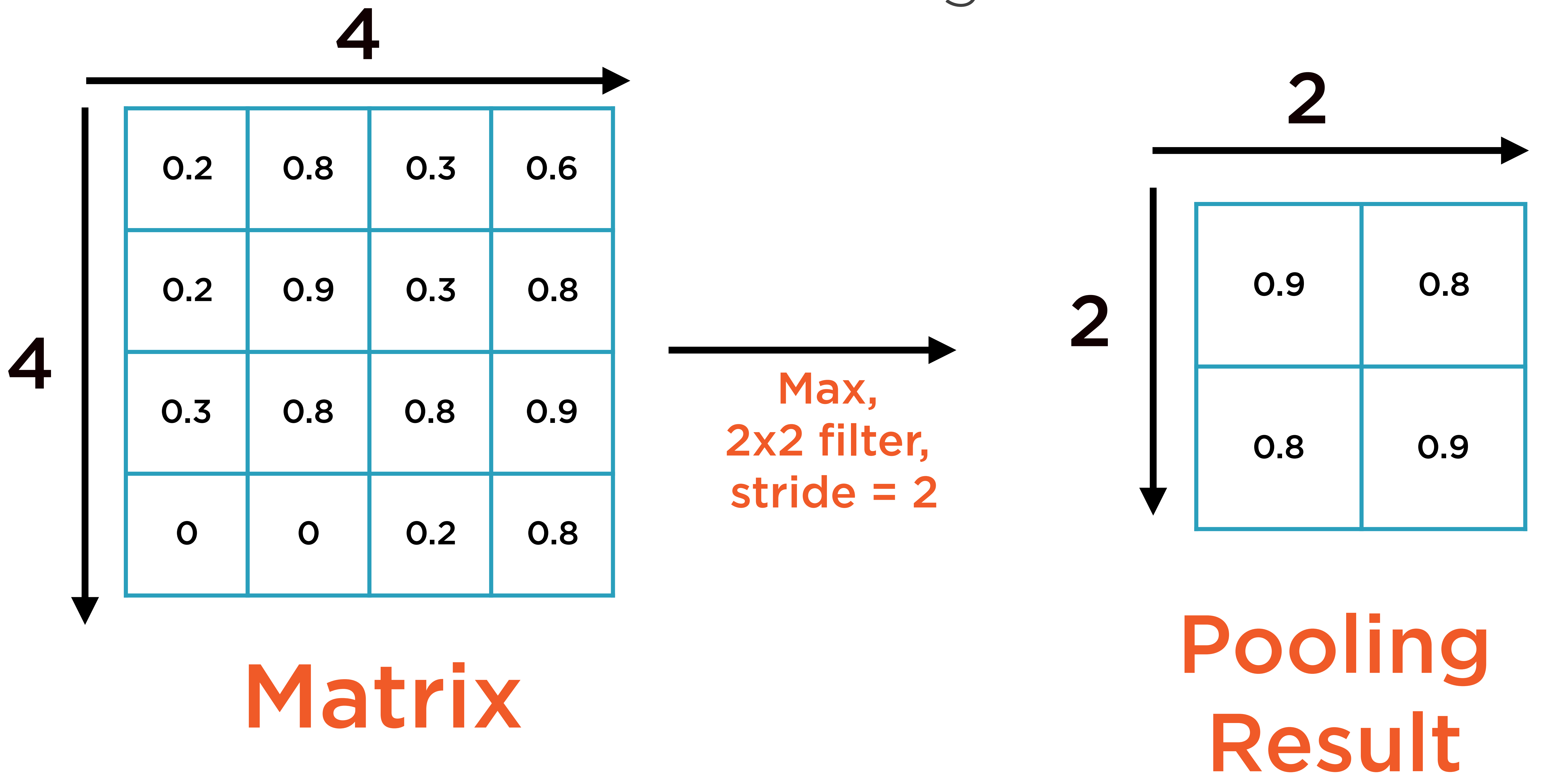
Convolutional Layer



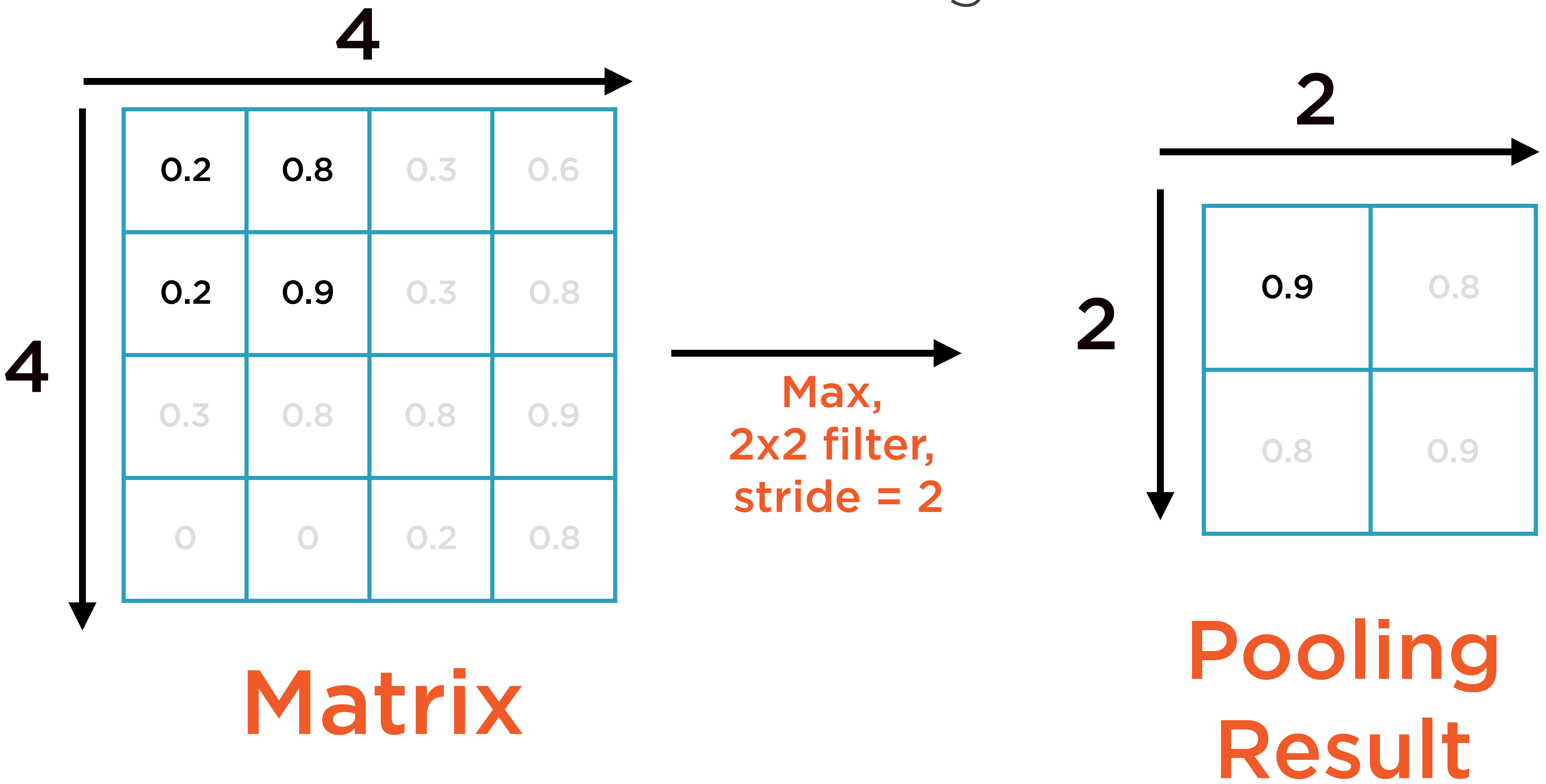
CNN

Pooling Layers

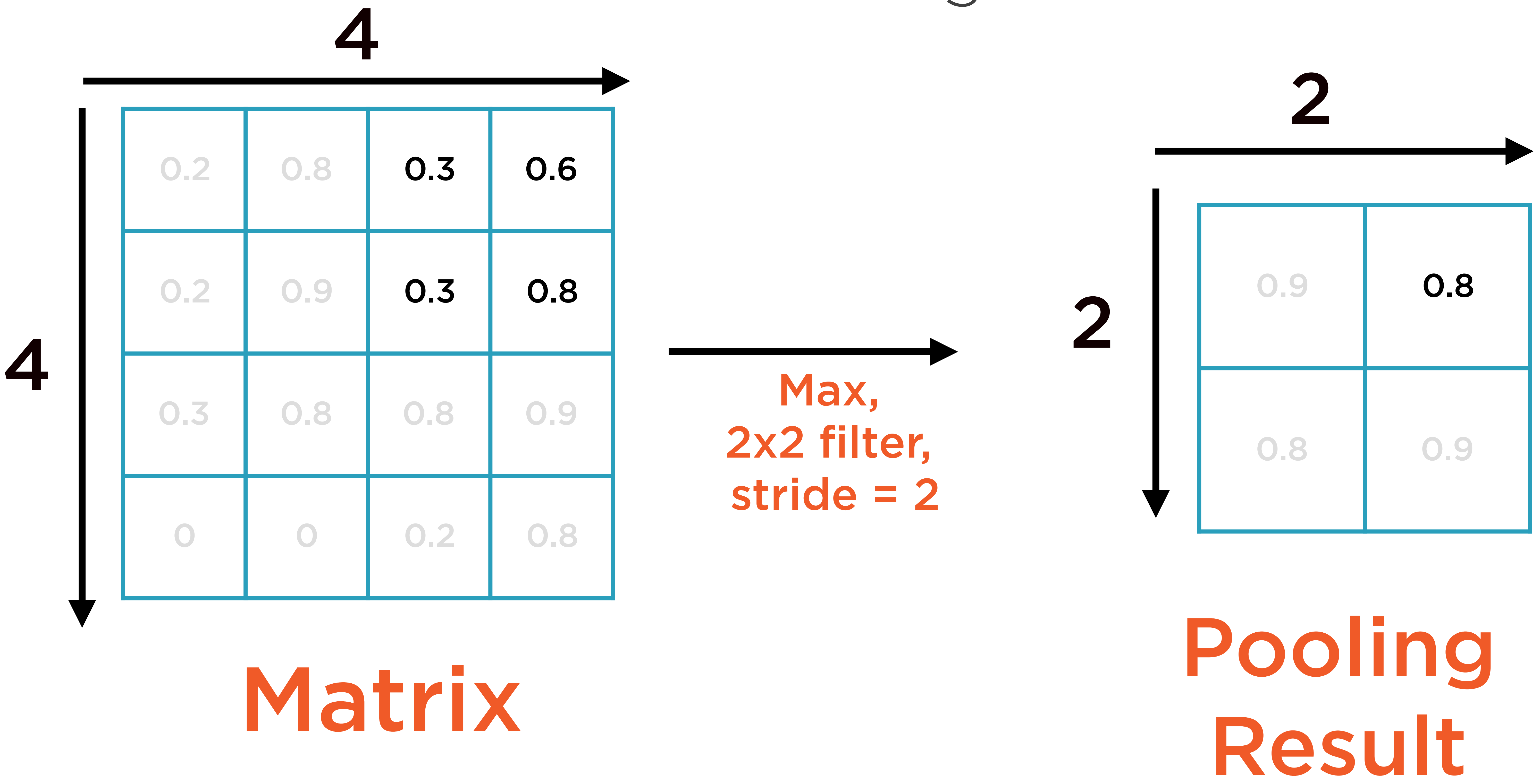
Pooling


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
Matrix
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Pooling
Result

Pooling

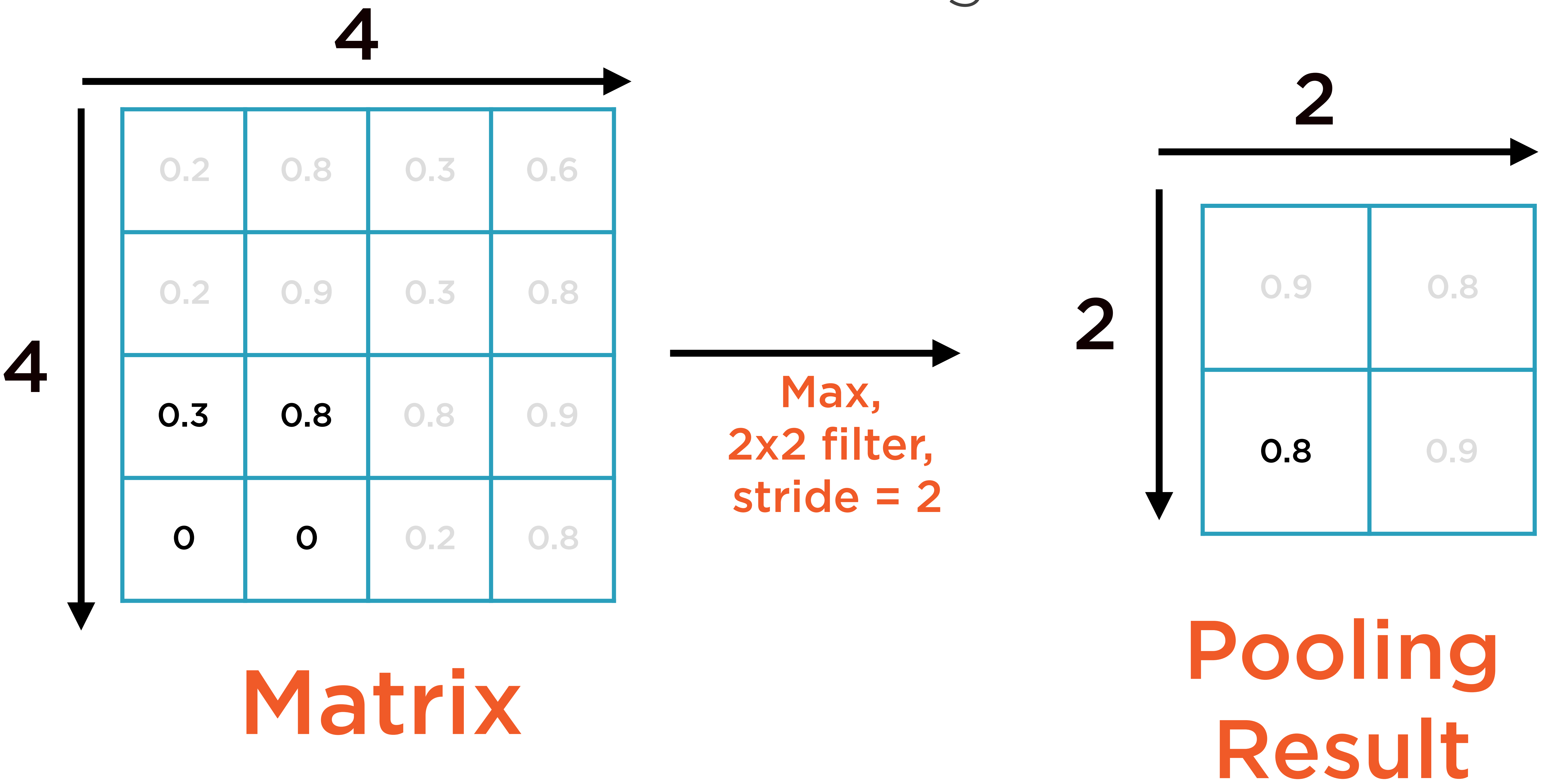

4
4
0.2 0.8 0.3 0.6
0.2 0.9 0.3 0.8
0.3 0.8 0.8 0.9
0 0 0.2 0.8
Matrix
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Pooling
Result

Pooling

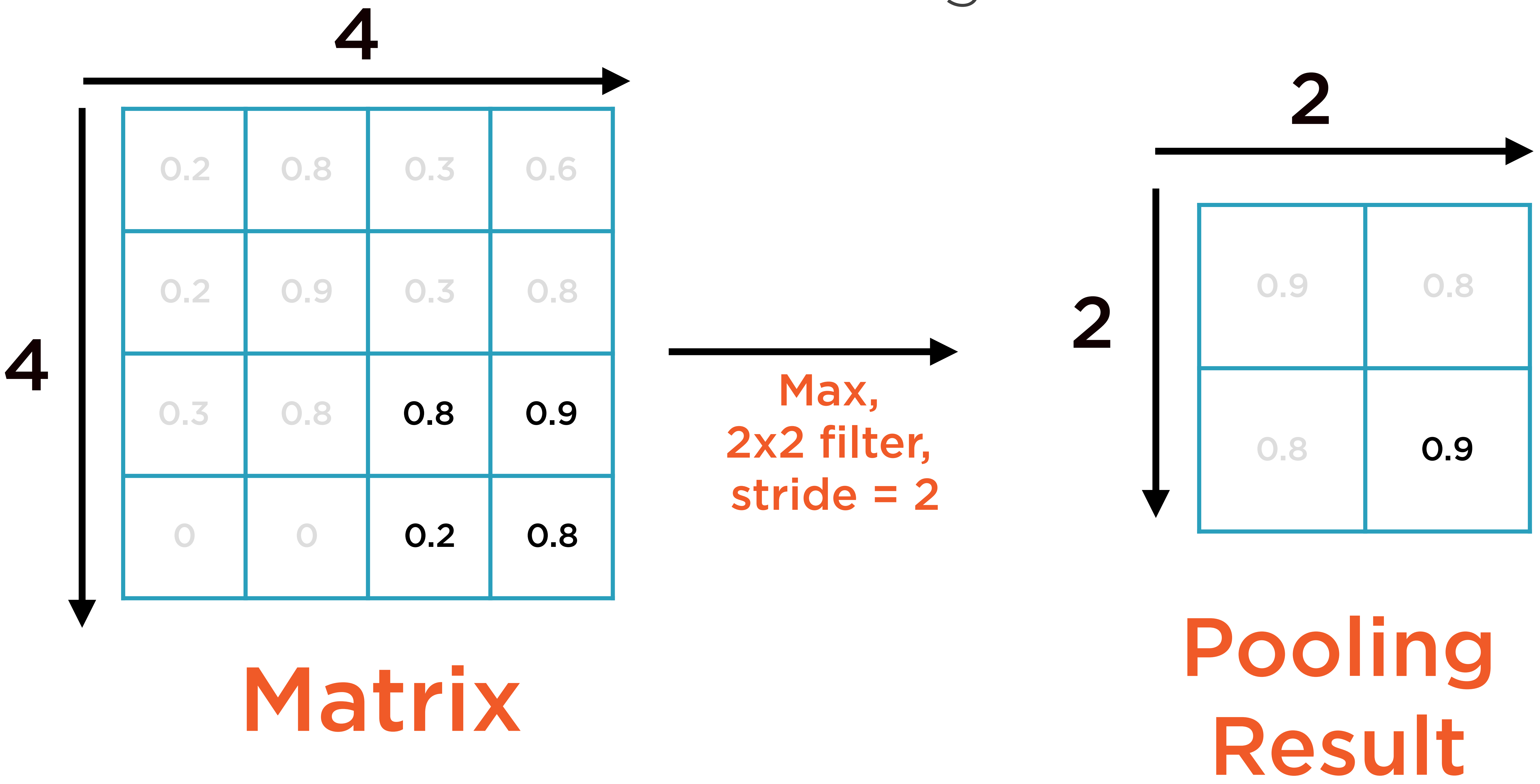


4
4
0.2 0.8 0.3 0.6
0.2 0.9 0.3 0.8
0.3 0.8 0.8 0.9
0 0 0.2 0.8
Matrix
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Pooling
Result

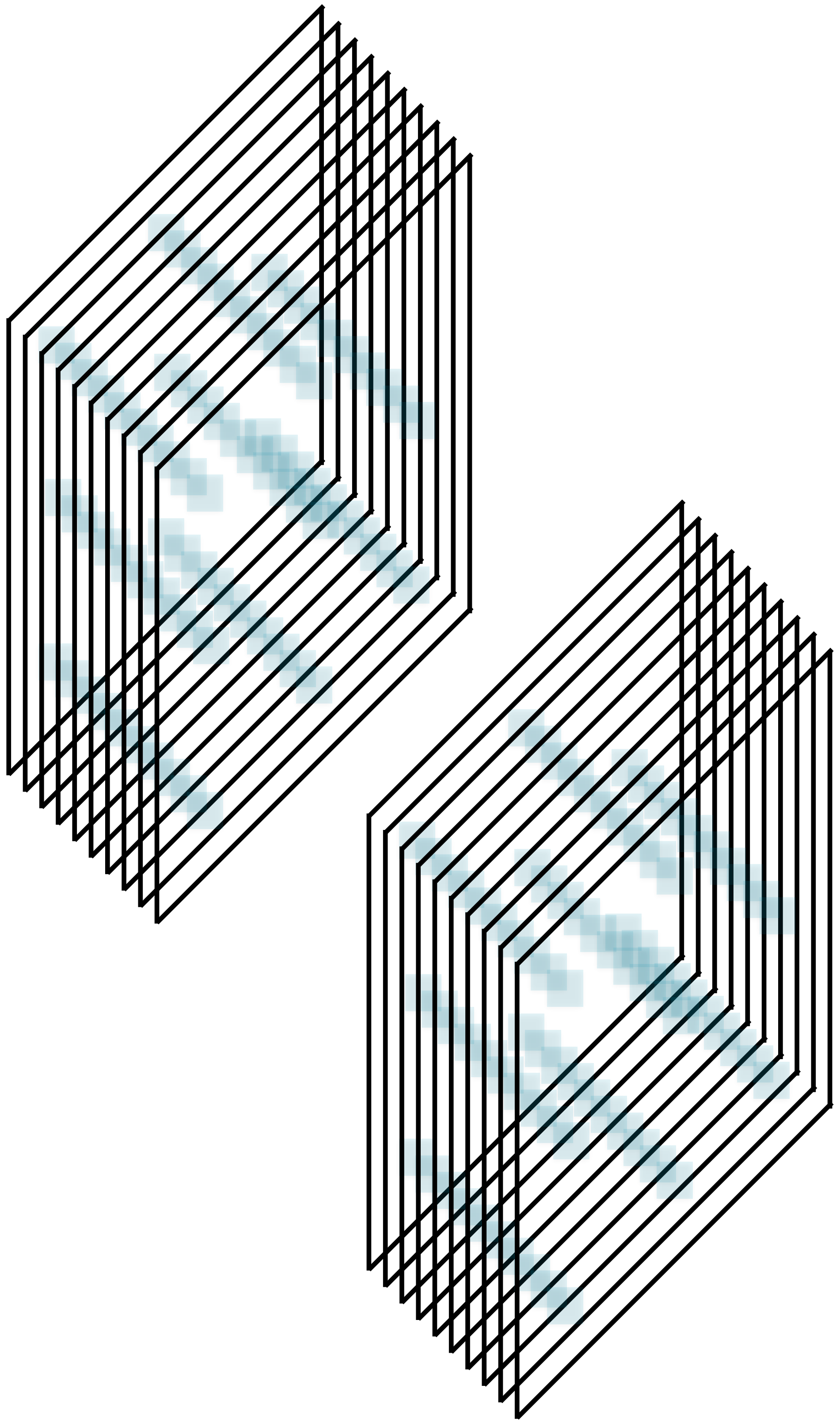
Pooling


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
Matrix
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Pooling
Result

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
Matrix
Max,
2x2 filter,
stride = 2
2
2
0.9 0.8
0.8 0.9
Pooling
Result

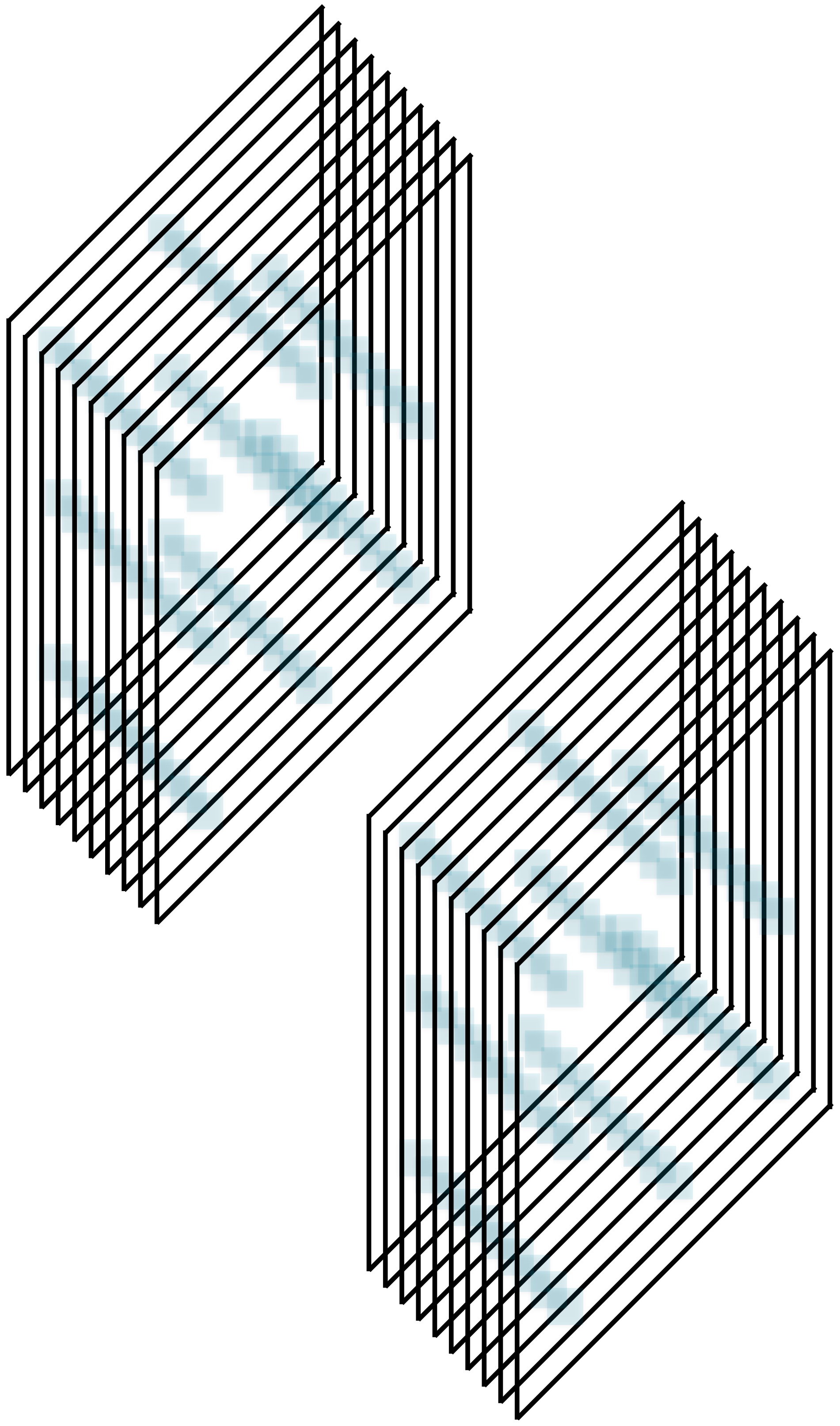


Pooling Layers

Neurons in a pooling layer have no weights or biases

A pooling neuron simply applies some aggregation function to all inputs

Max, sum, average...



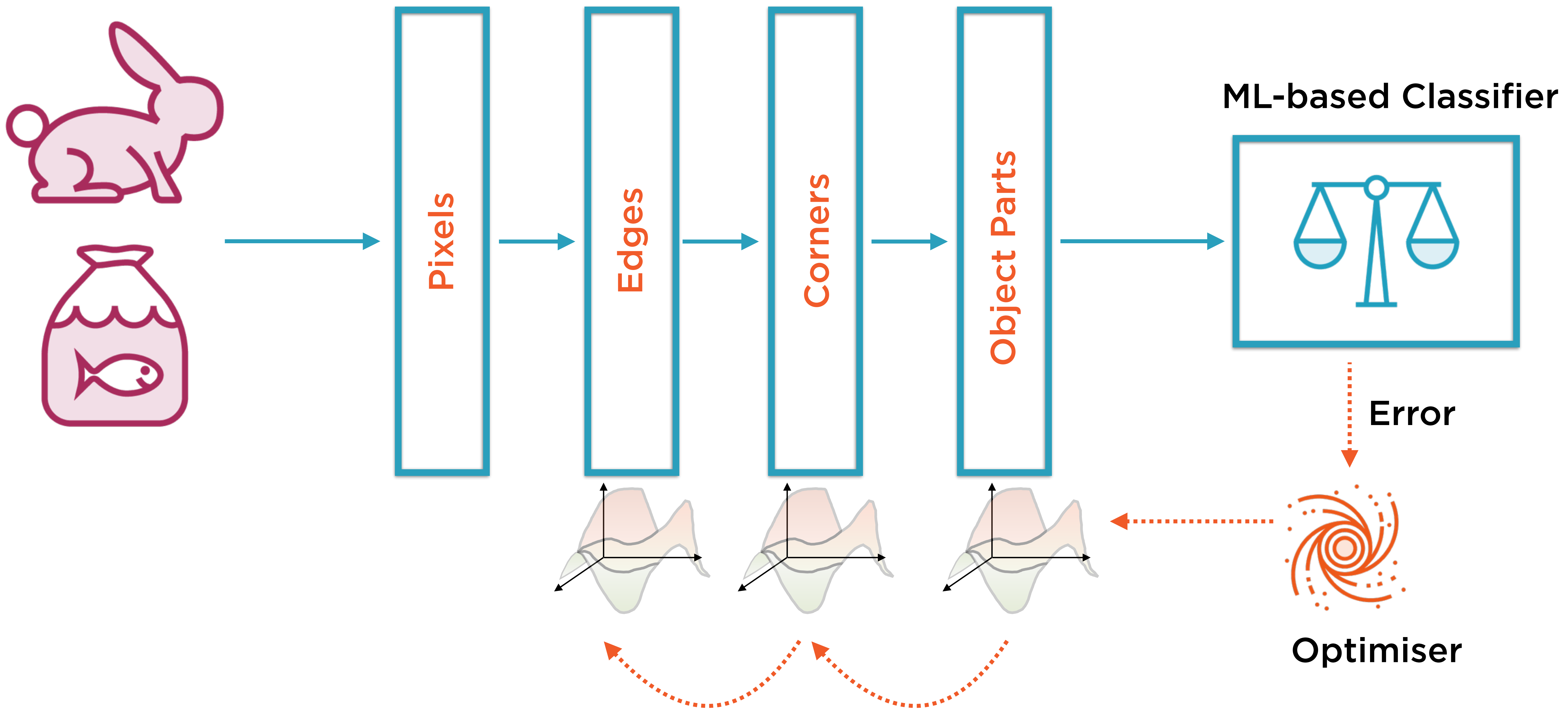
Pooling Layers

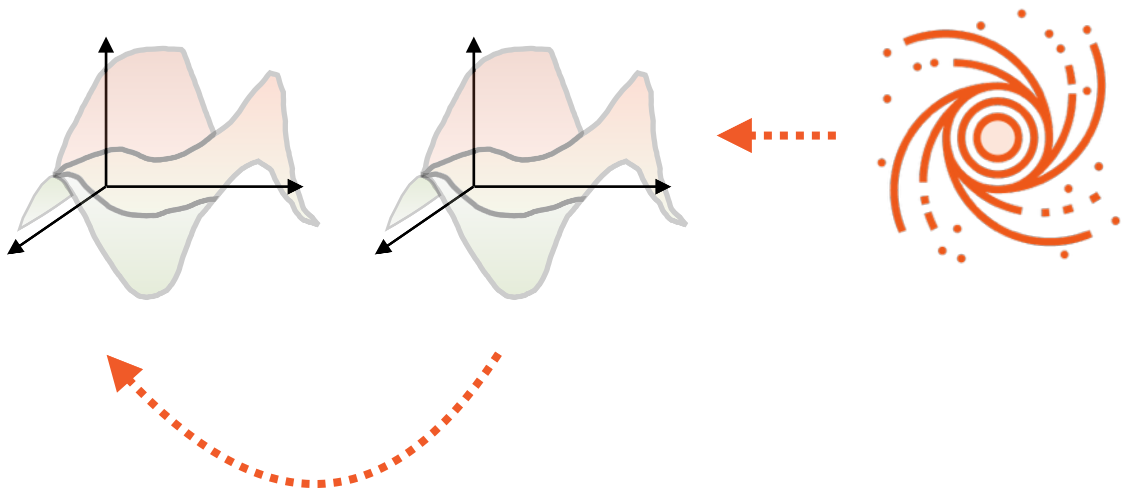
Why use them?

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

Batch Normalization

Training via Back Propagation





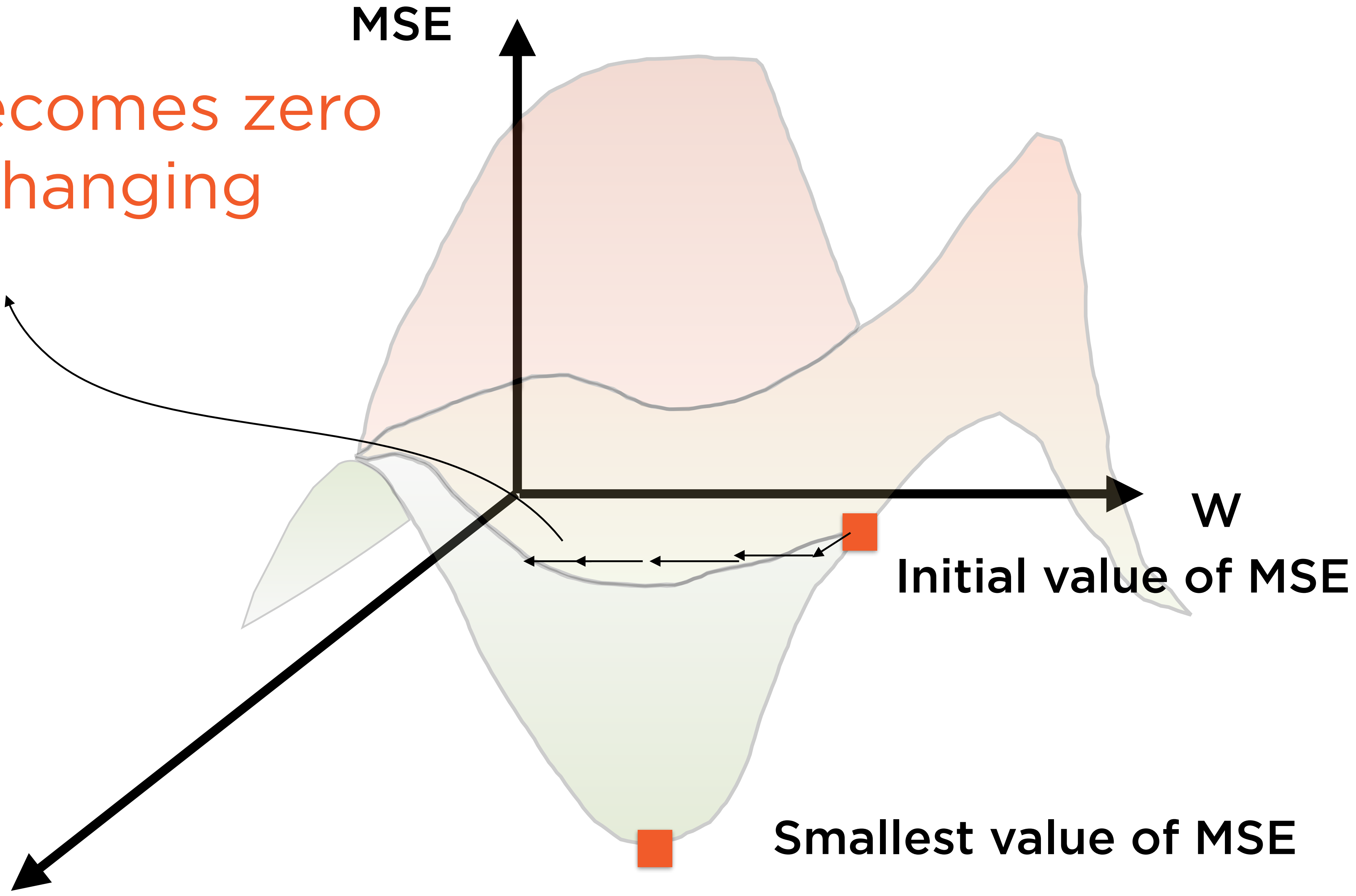
Vanishing and Exploding Gradients

Back propagation fails if

- gradients are vanishing
- gradients are exploding

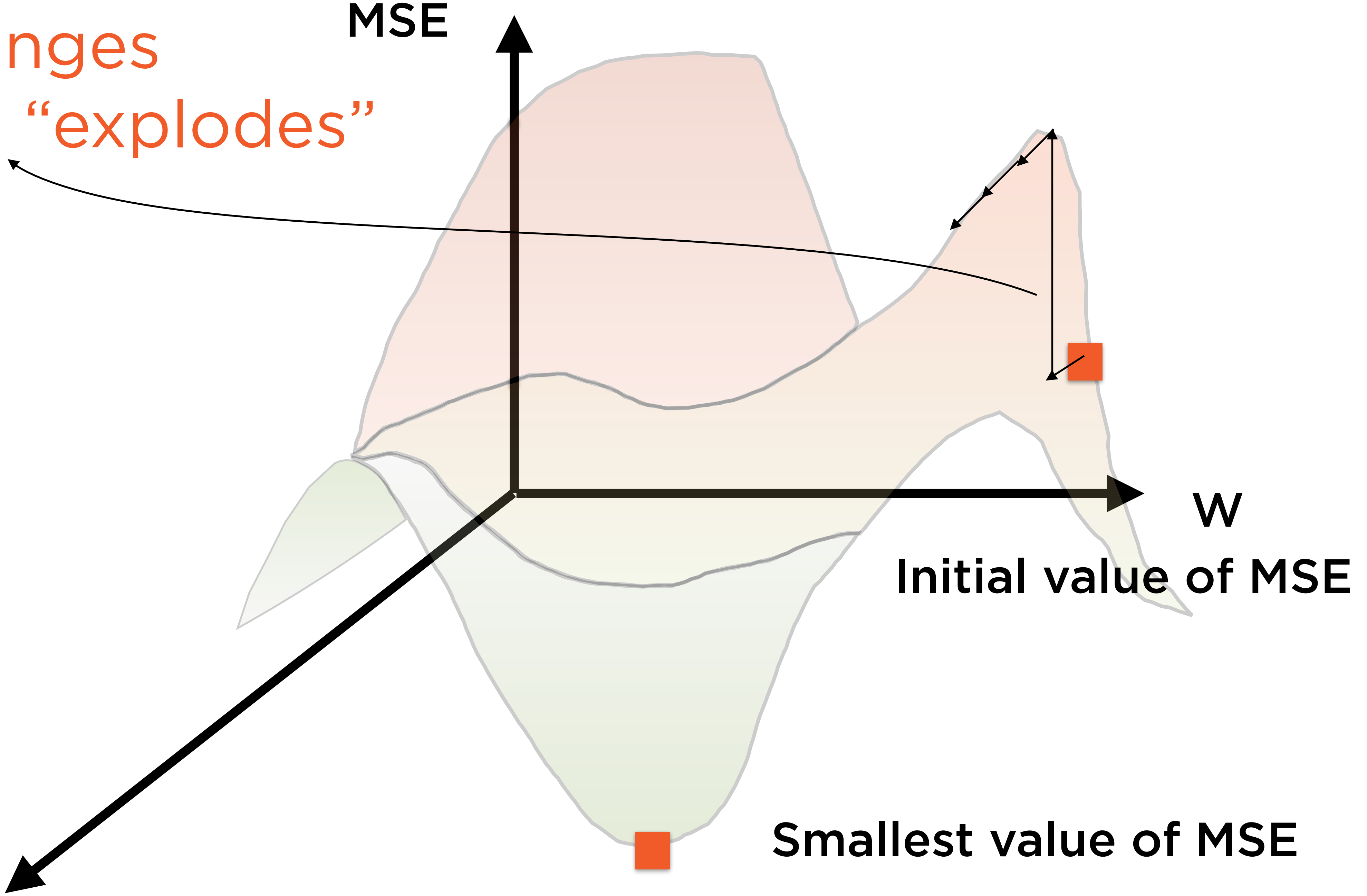
“Vanishing Gradient Problem”

Gradient becomes zero and stops changing



“Exploding Gradient Problem”

Gradient changes abruptly and “explodes”

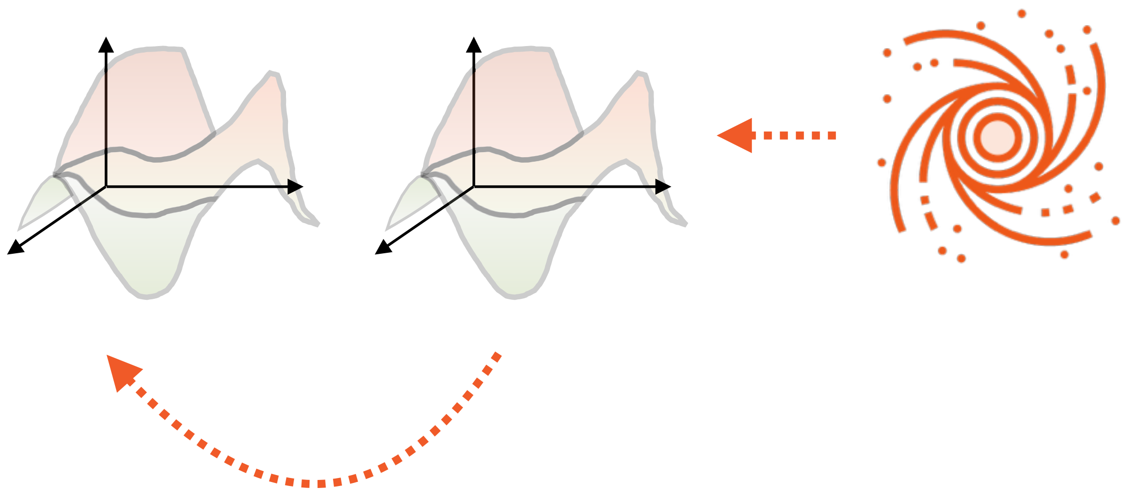


Coping with Vanishing/Exploding Gradients

Proper initialisation	Non-saturating activation function
Batch normalization	Gradient clipping

Coping with Vanishing/Exploding Gradients

Proper initialisation	Non-saturating activation function
Batch normalization	Gradient clipping

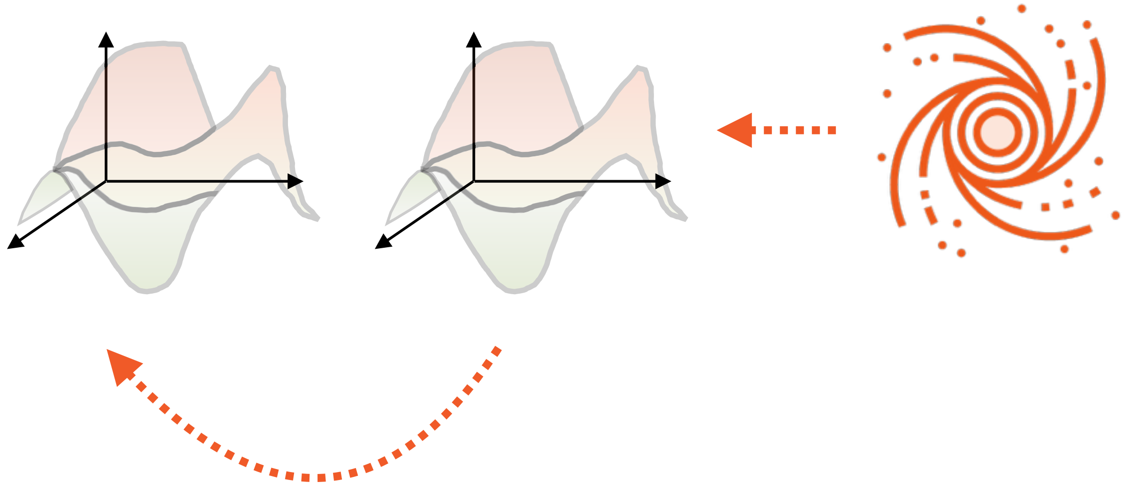


Batch Normalization

Just before applying activation function

First, “normalize” inputs

Second, “scale and shift” inputs



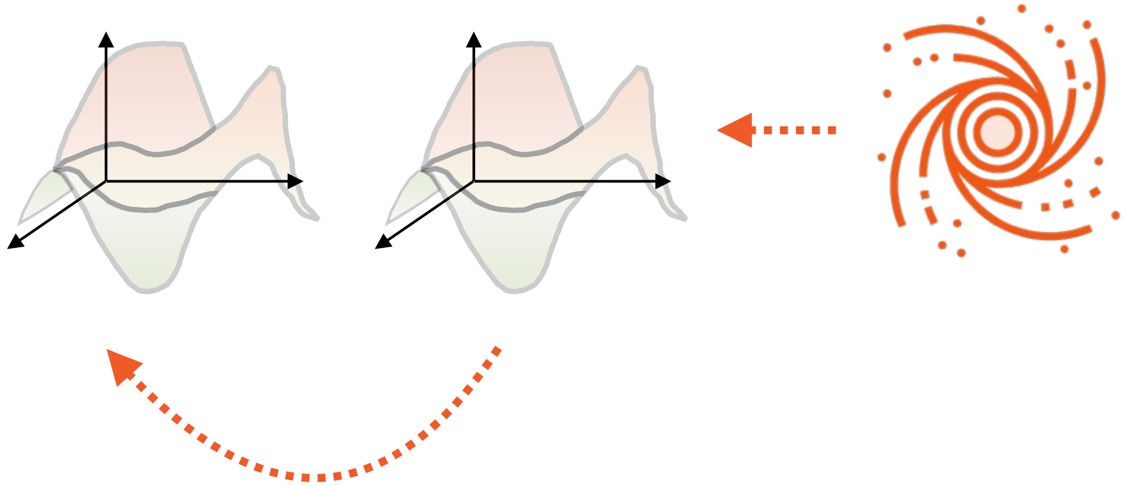
Batch Normalization

“Normalize” inputs

- subtract mean
- divide by standard deviation

“Scale and shift” inputs

- scale = multiply by constant
- shift = add constant



Batch Normalization

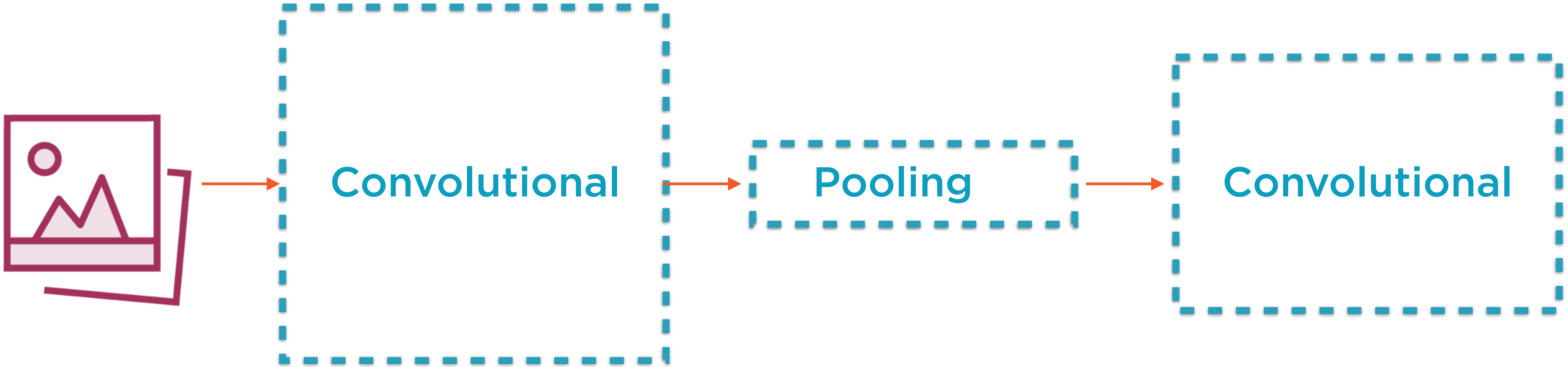
Supported in PyTorch

Many other benefits

- allows much larger learn rate
- reduces overfitting
- speeds convergence of training

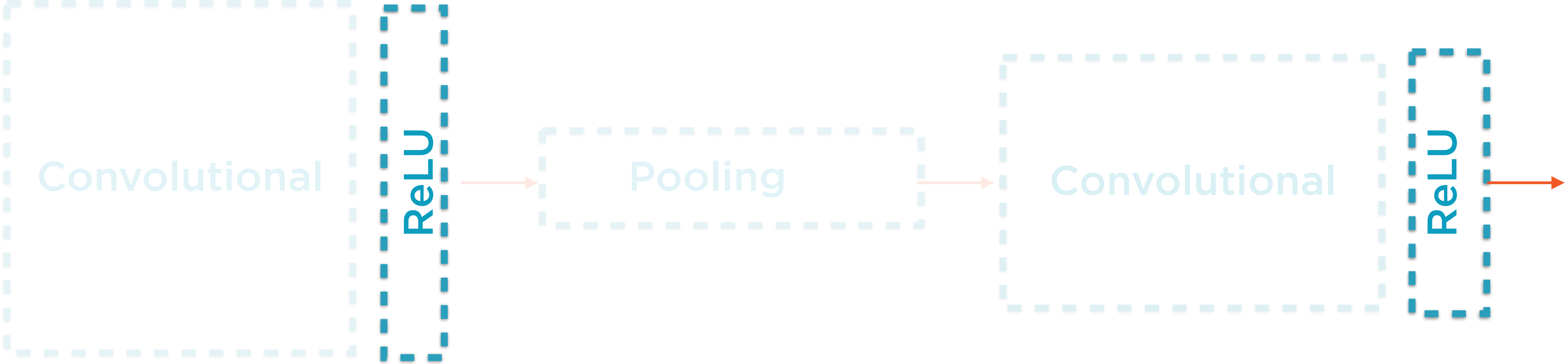
CNN Architectures

Typical CNN Architecture



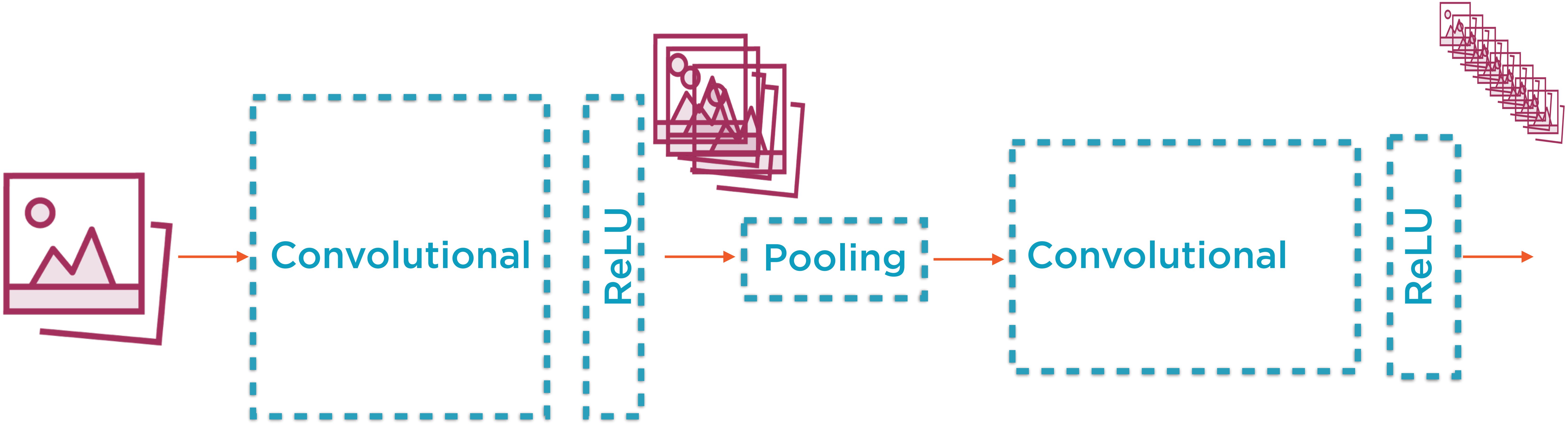
Alternating groups of convolutional and pooling layers

Typical CNN Architecture



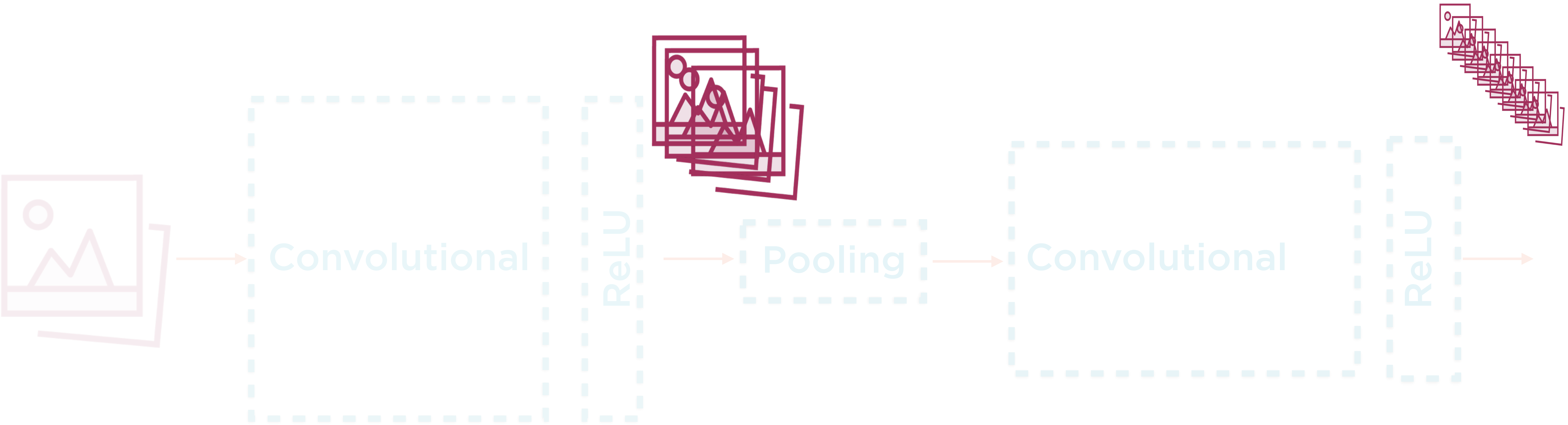
Each group of convolutional layers usually followed by a ReLU layer

Typical CNN Architecture



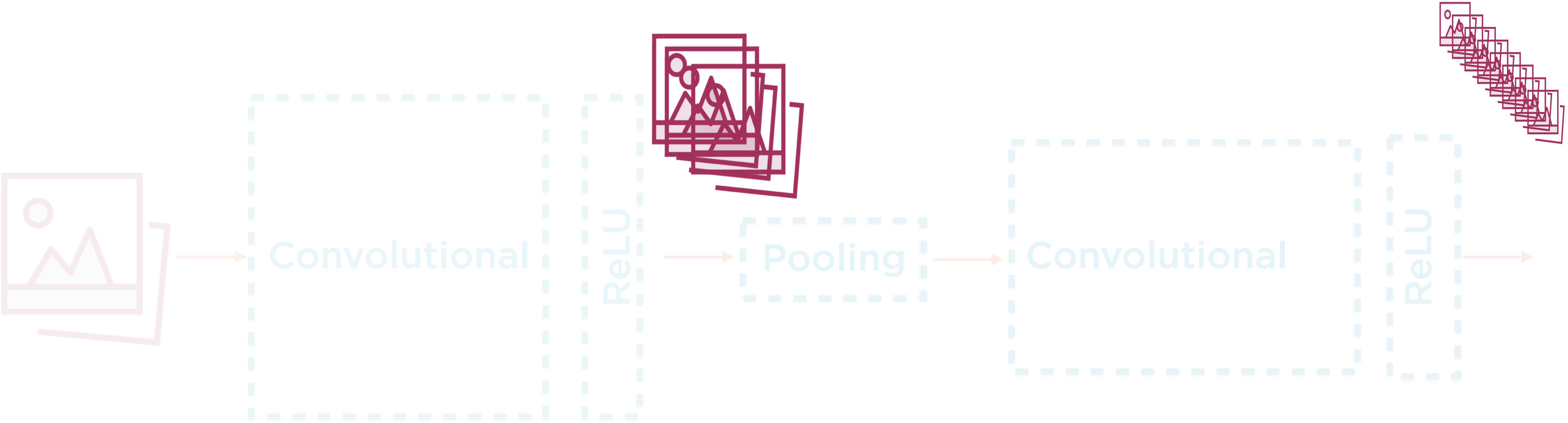
The outputs of each layer are also images

Typical CNN Architecture



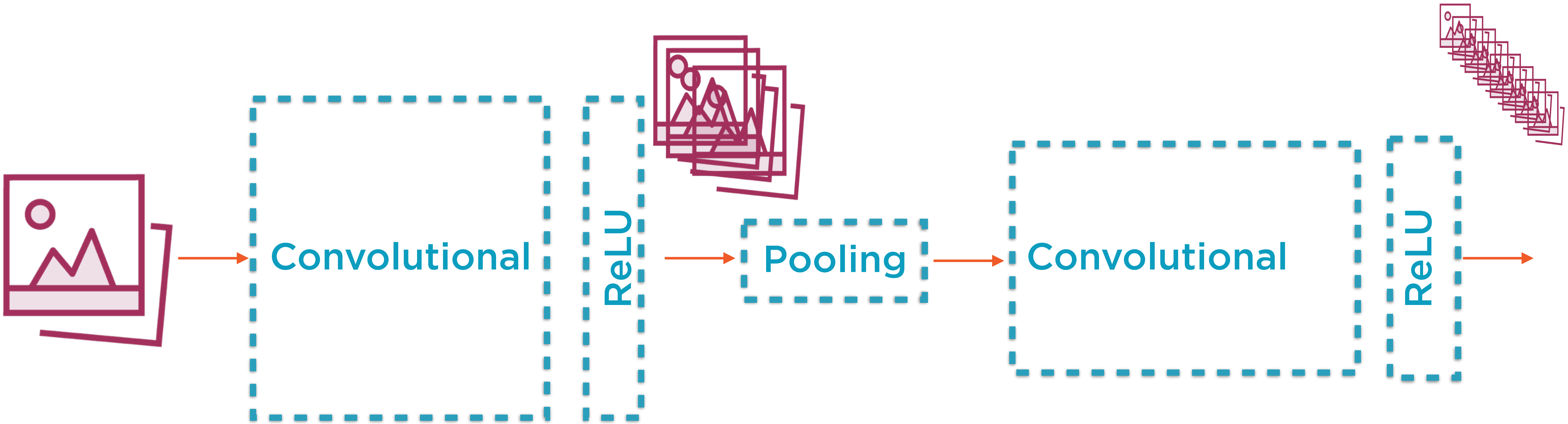
However successive outputs are smaller and smaller (due to pooling layers)

Typical CNN Architecture



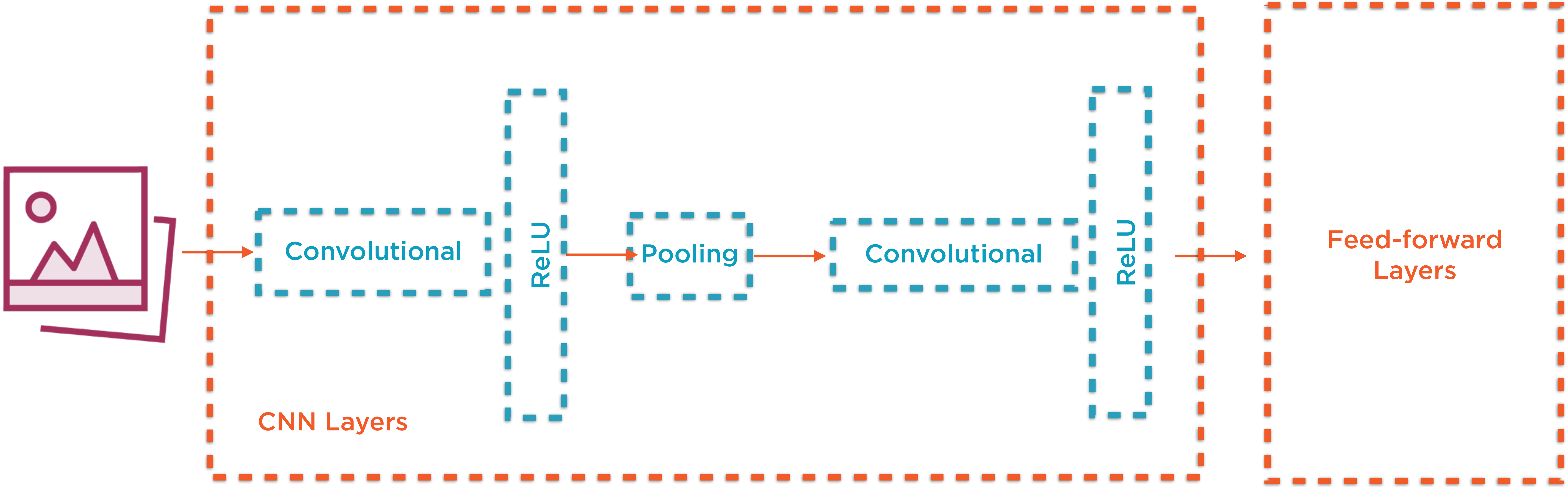
As well as deeper and deeper (due to feature maps in the convolutional layers)

Typical CNN Architecture



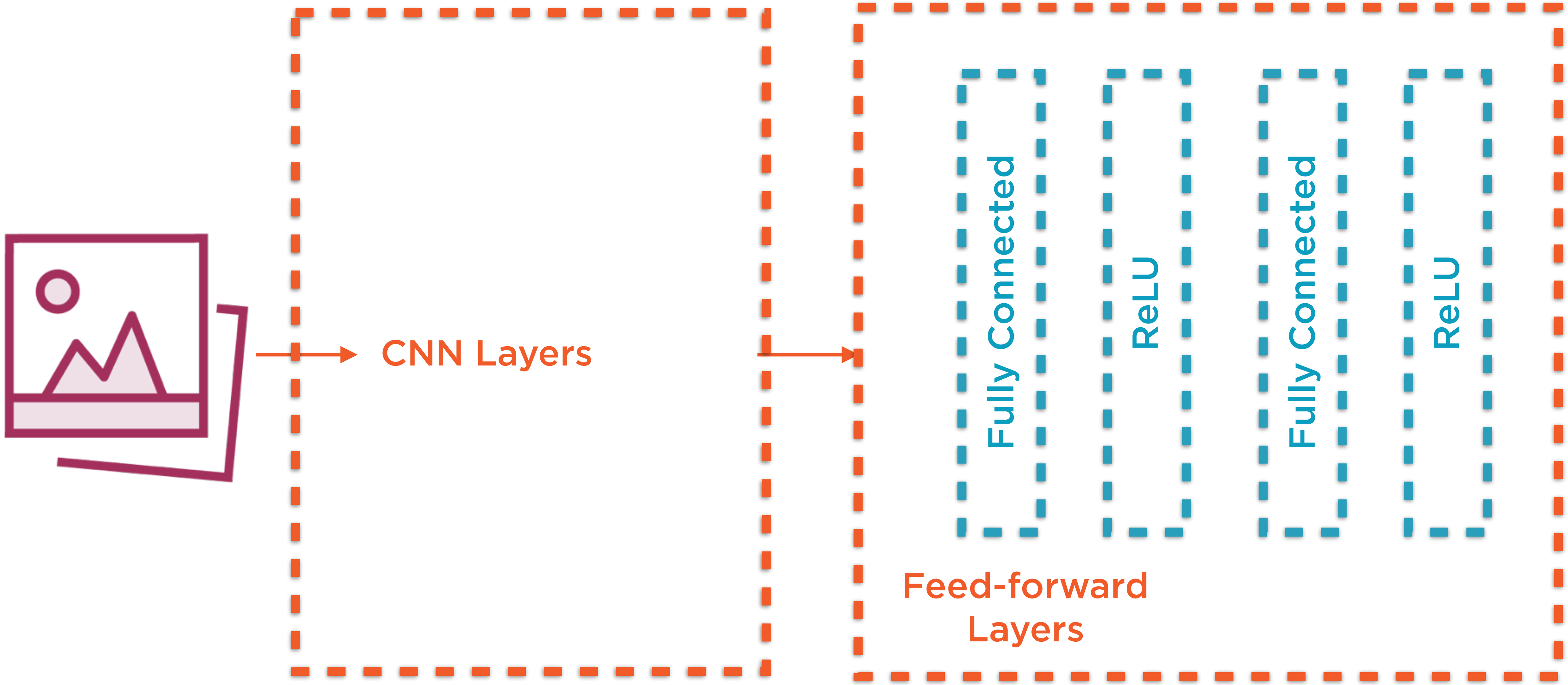
This entire set of layers is then fed into a regular, feed-forward NN

Typical CNN Architecture



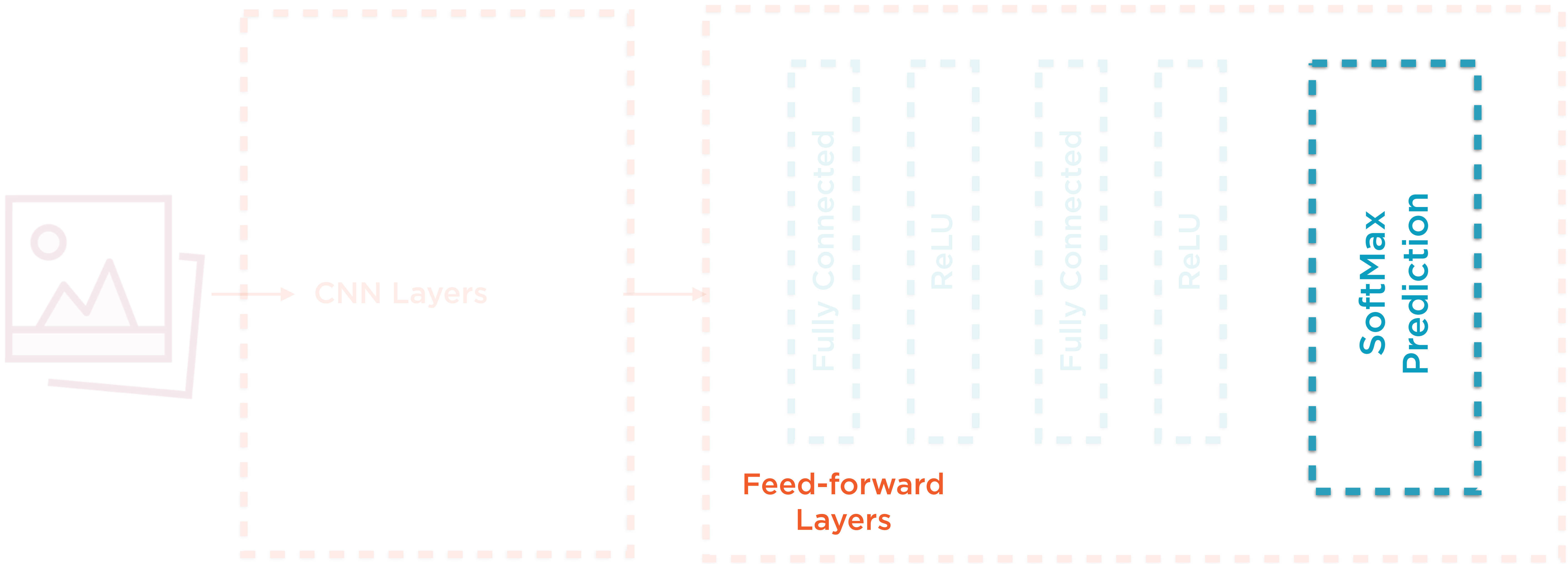
This entire set of layers is then fed into a regular, feed-forward NN

Typical CNN Architecture



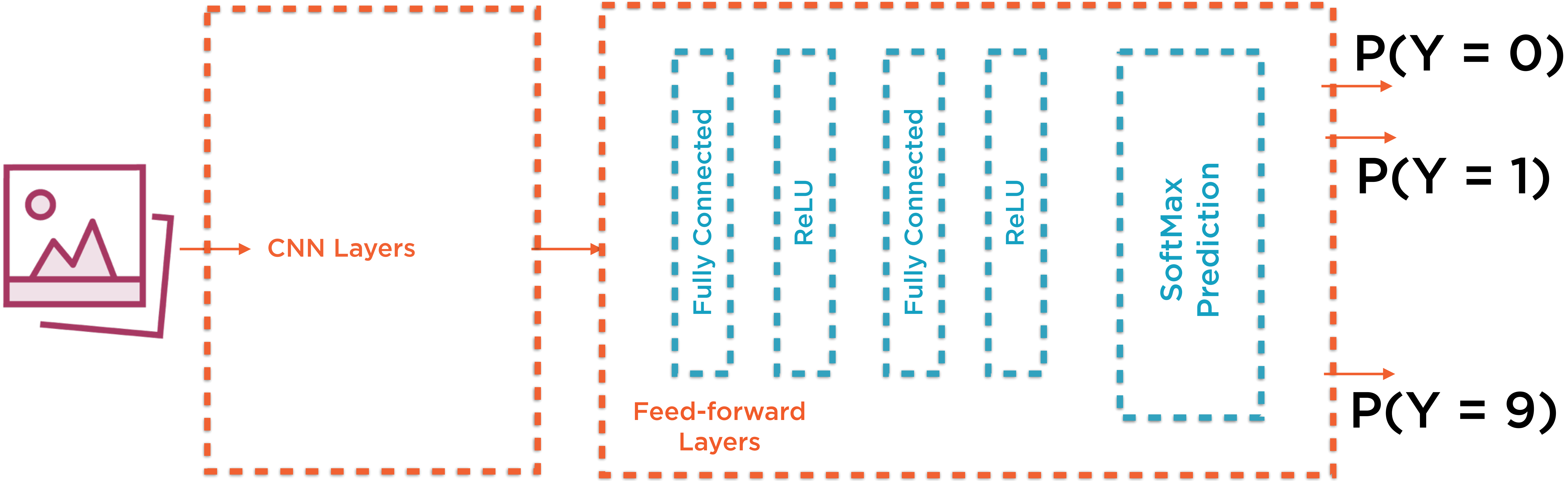
This feed-forward has a few fully connected layers with ReLU activation

Typical CNN Architecture



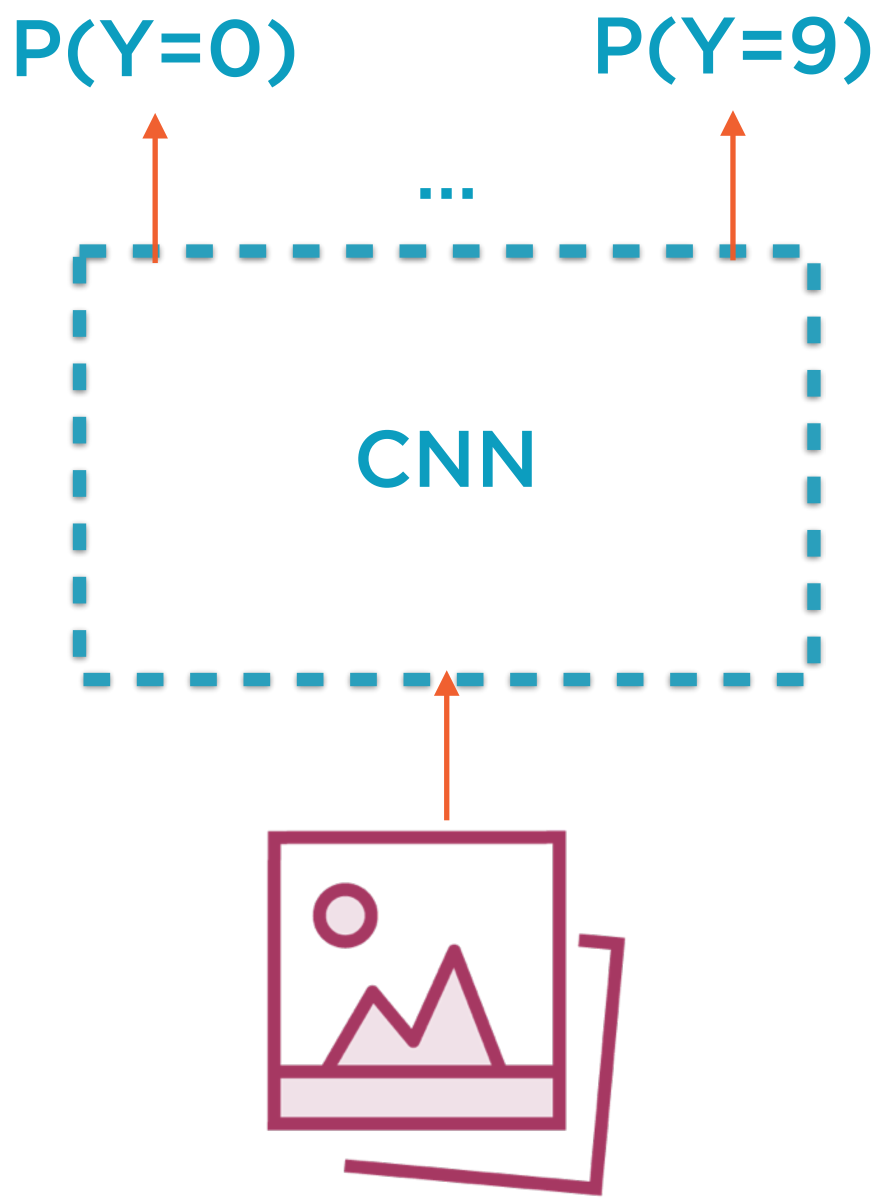
Finally a SoftMax prediction layer

Typical CNN Architecture



This is the output layer, emitting probabilities

Typical CNN Architectures



Input is an image

Outputs are probabilities

Demo

Building a CNN to classify images from the CIFAR-10 dataset

Transfer Learning

Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

Avoid designing NN architecture from scratch

Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

Also saves on time and effort of re-training from scratch

Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

Only makes sense for common, widely studied use-cases...

Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

...in which basic problem structure stays same, but details vary

Transfer Learning

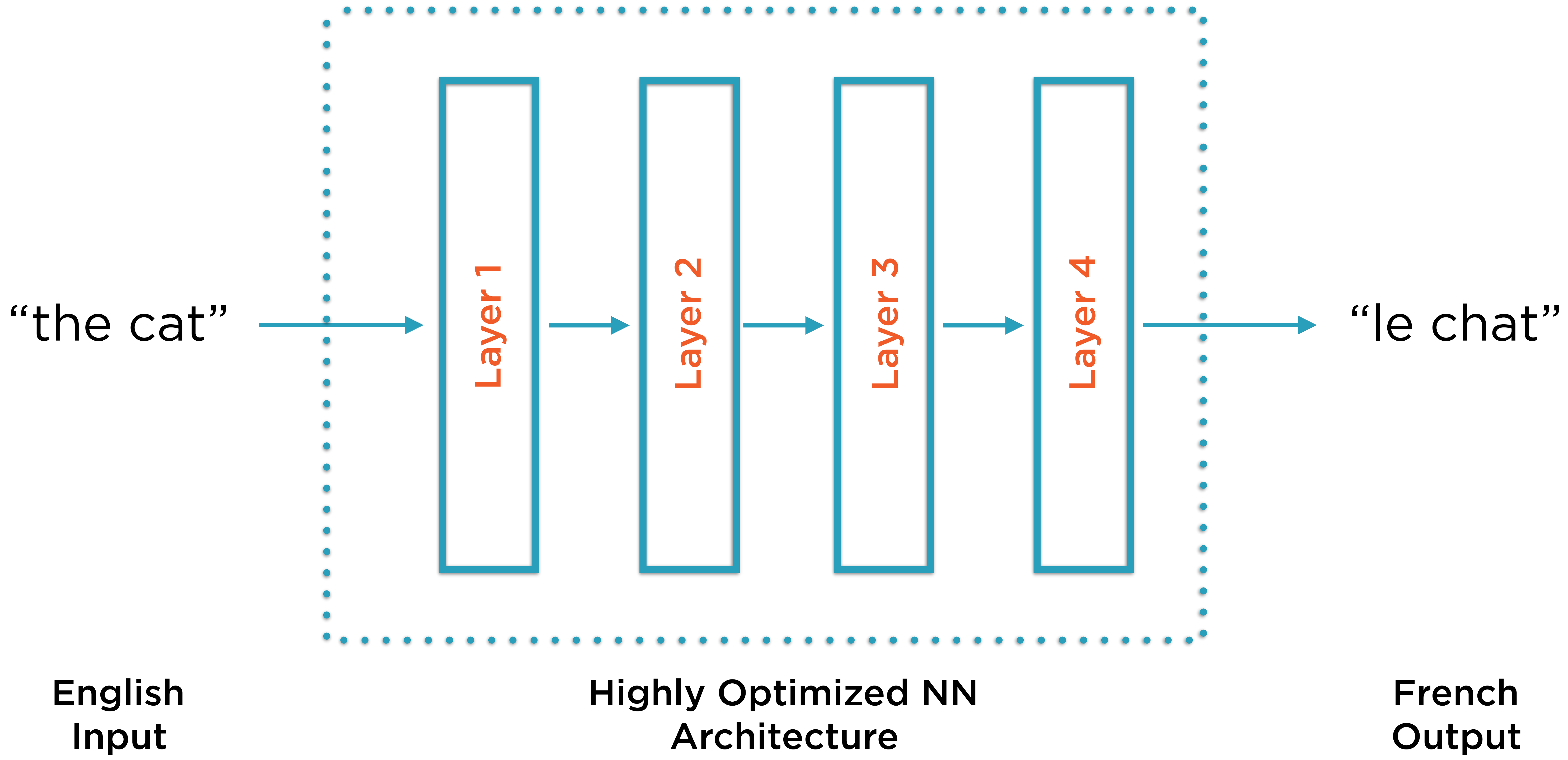
The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

Image recognition, language translation are classic examples

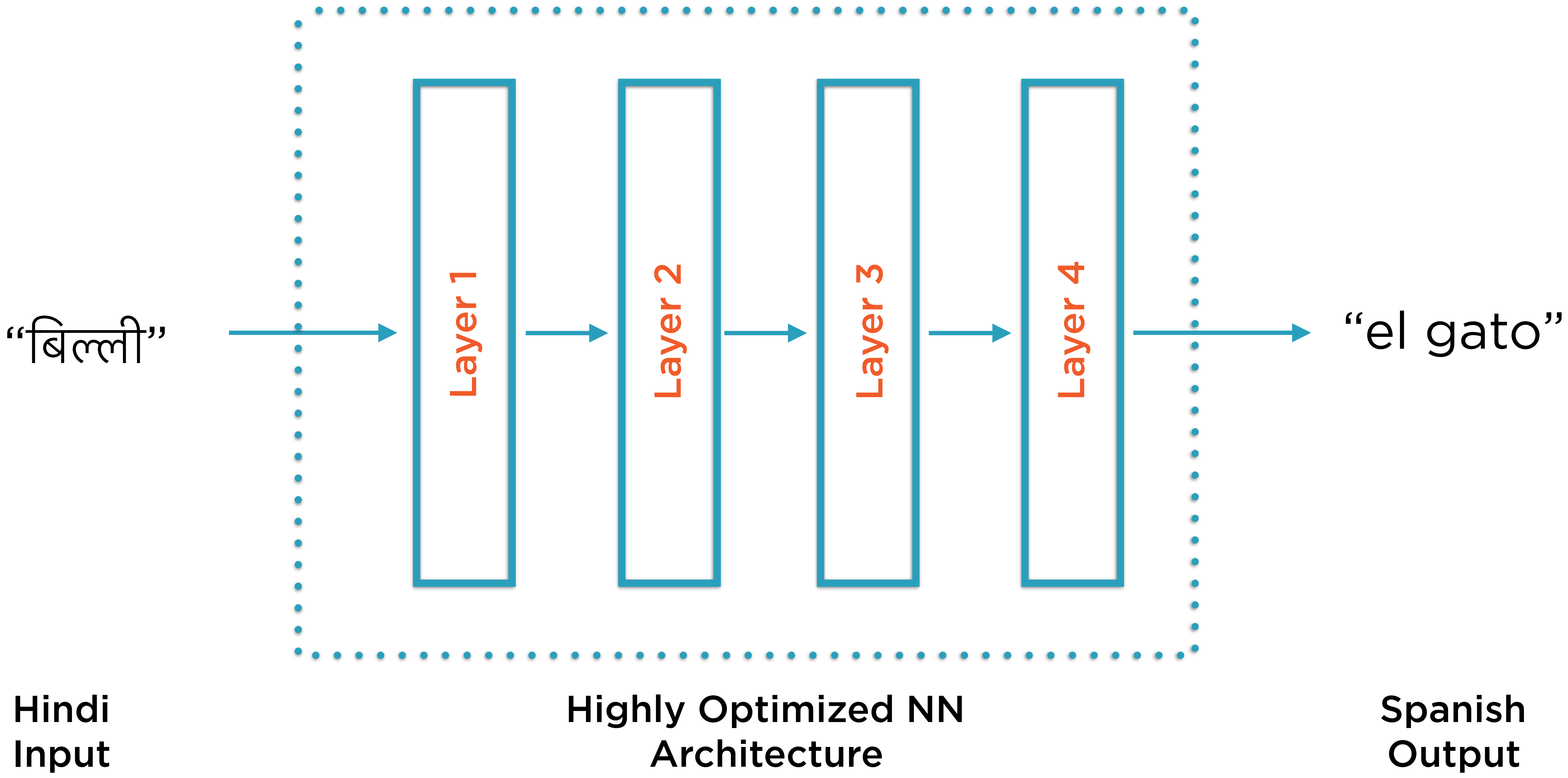
Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

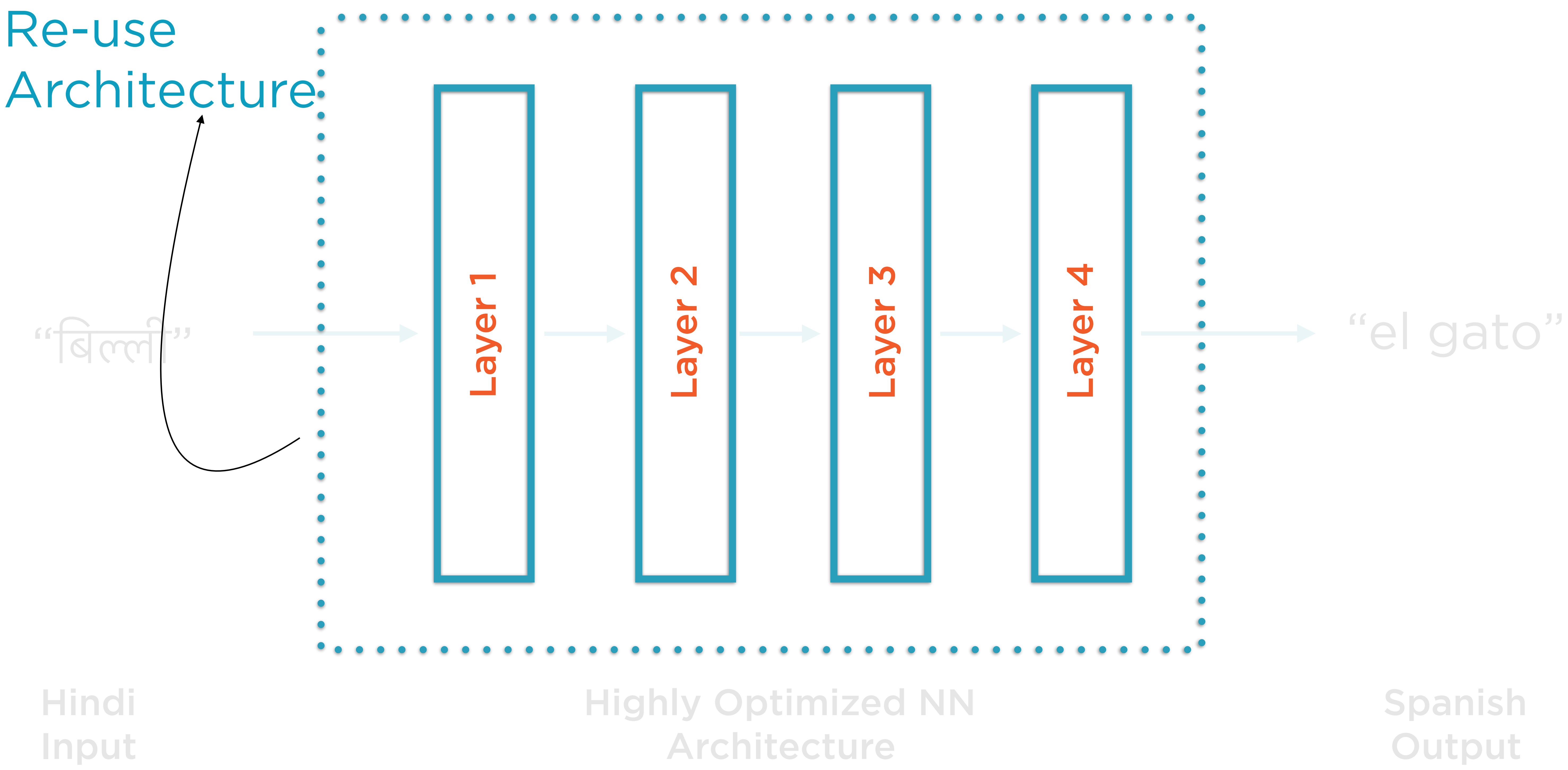
Original Model: English to French



Transfer Learning: Hindi to Spanish



Transfer Learning: Hindi to Spanish



Lower layers mostly perform feature extraction

Transfer Learning

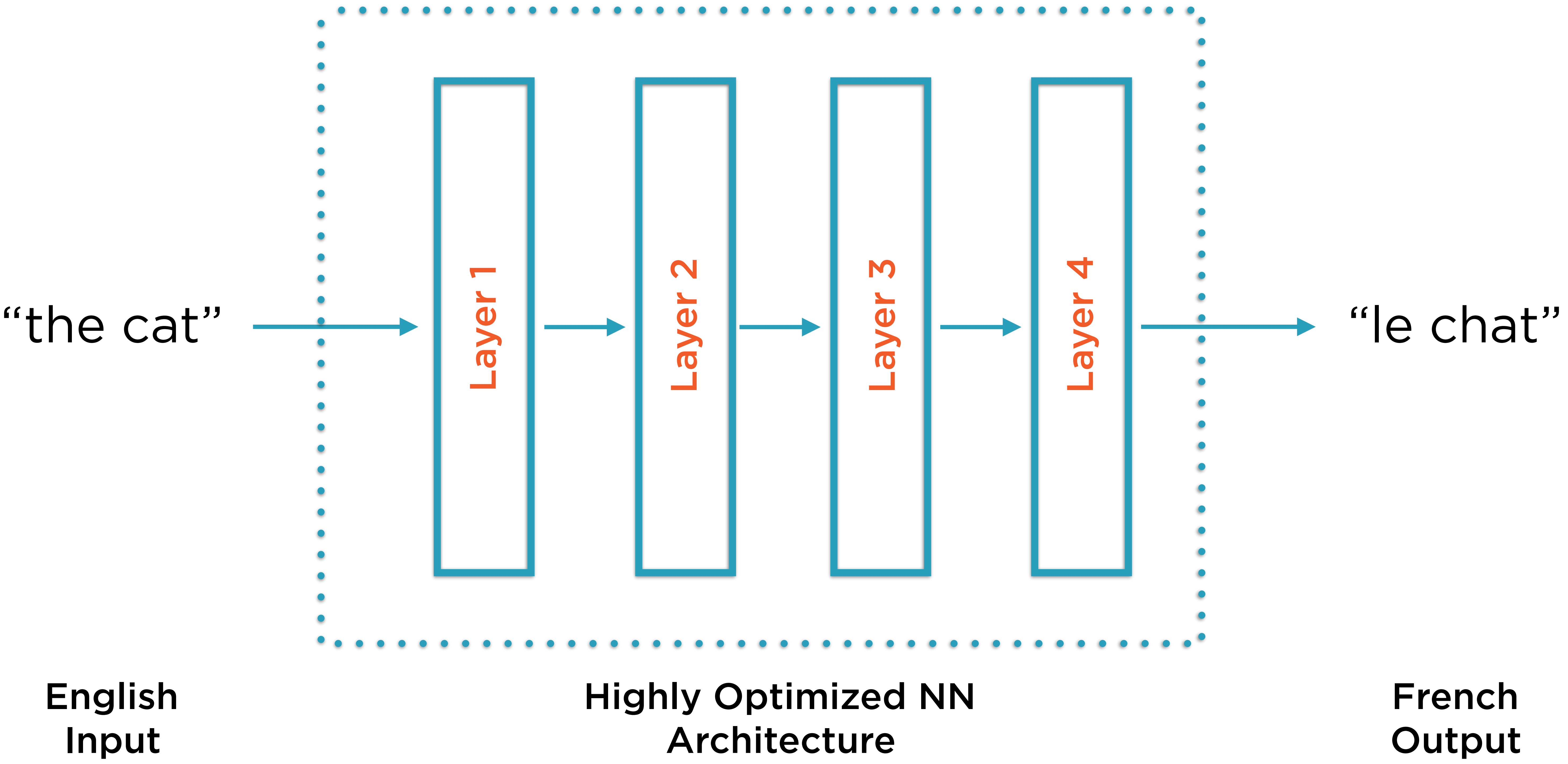
The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

Re-use as-is without even changing parameter weights

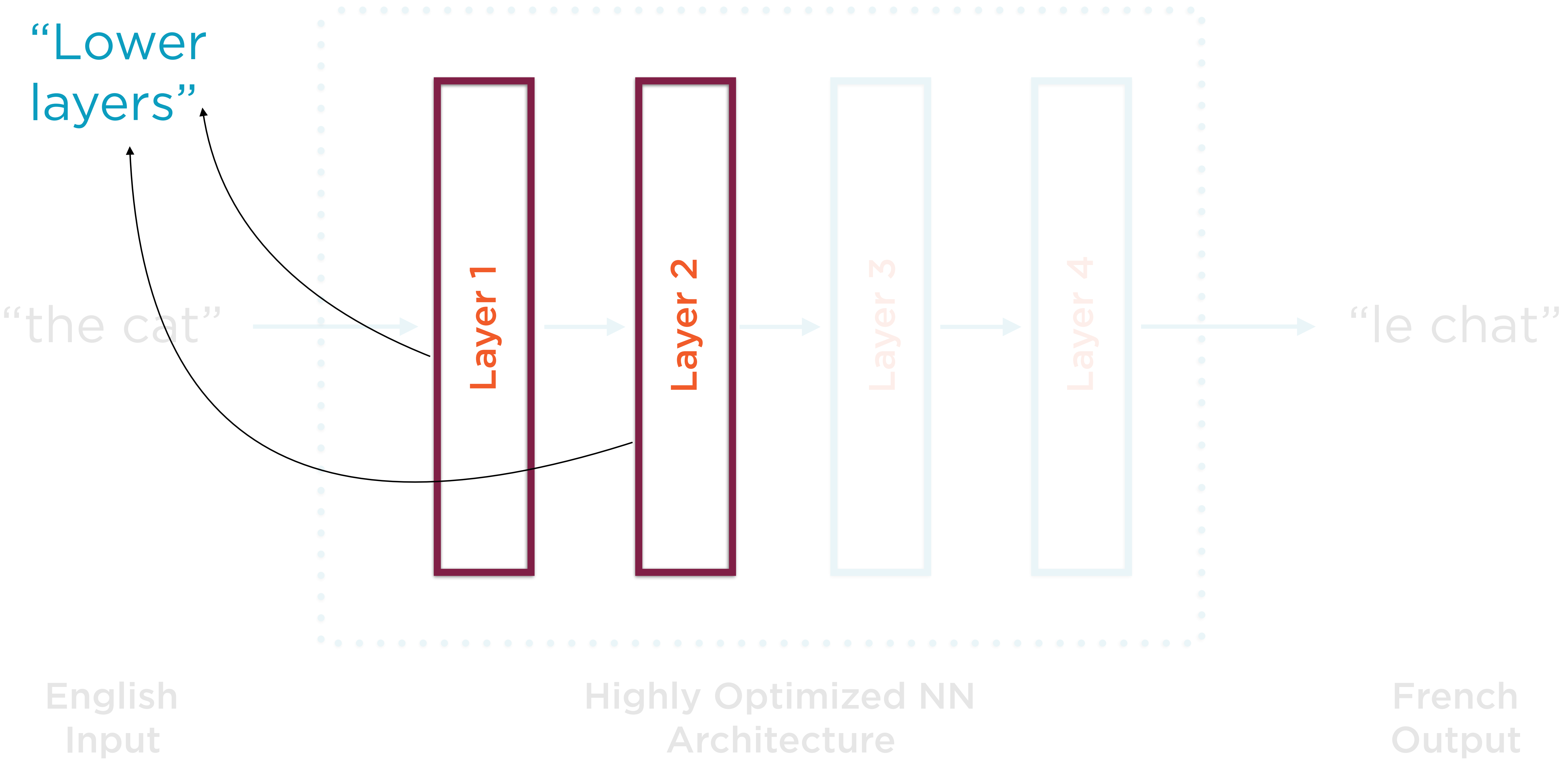
Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

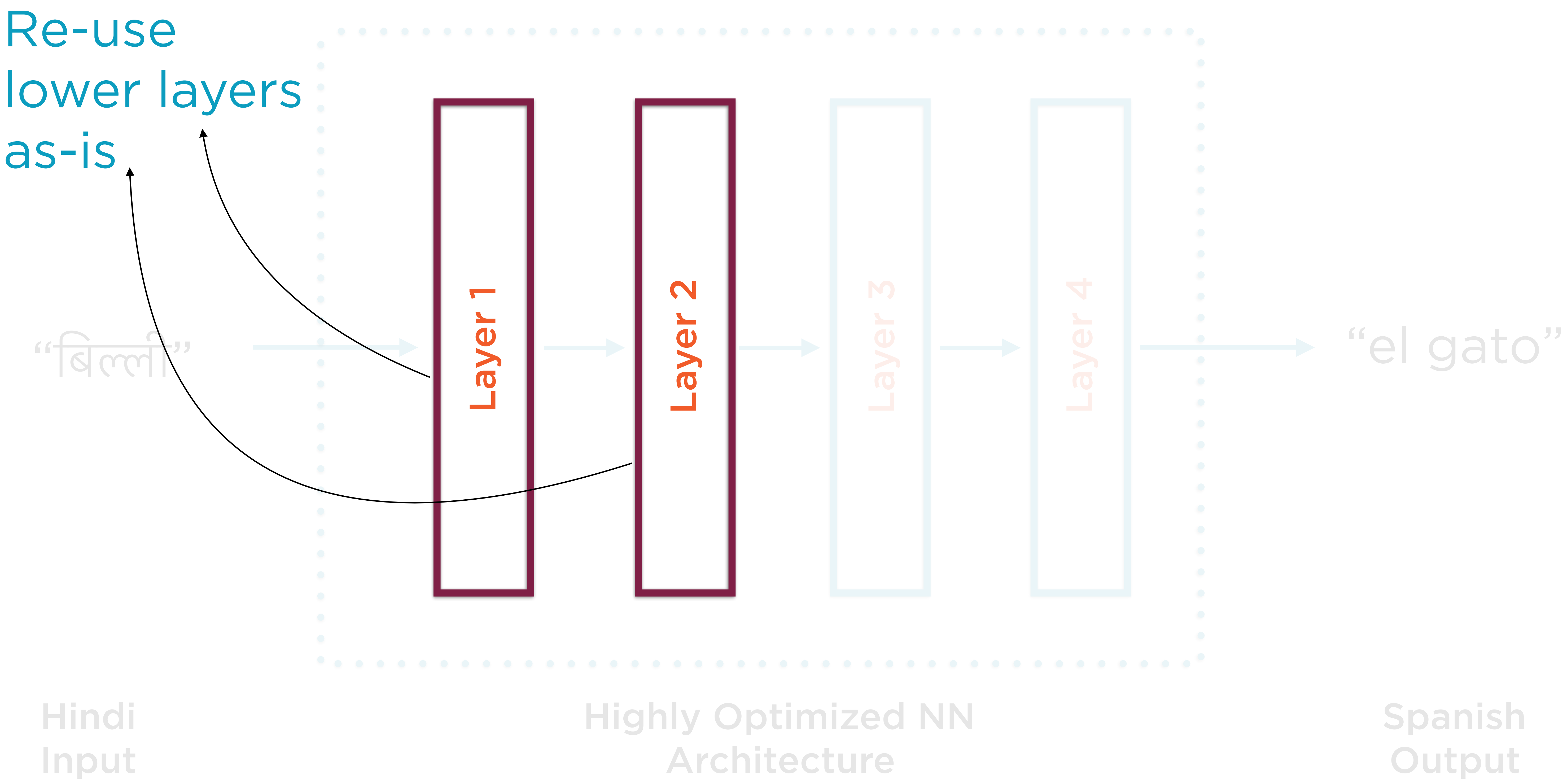
Original Model: English to French



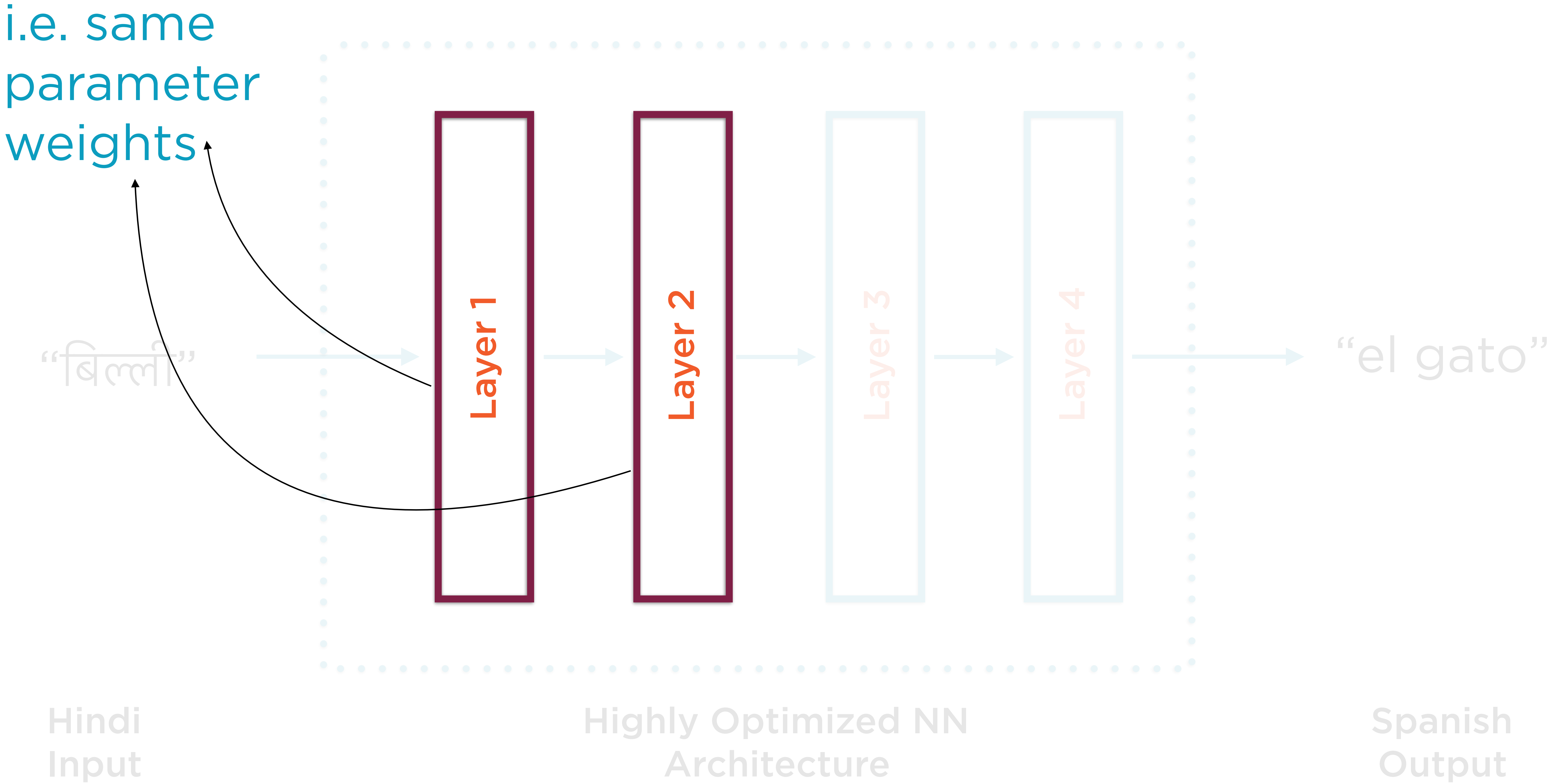
Original Model: English to French



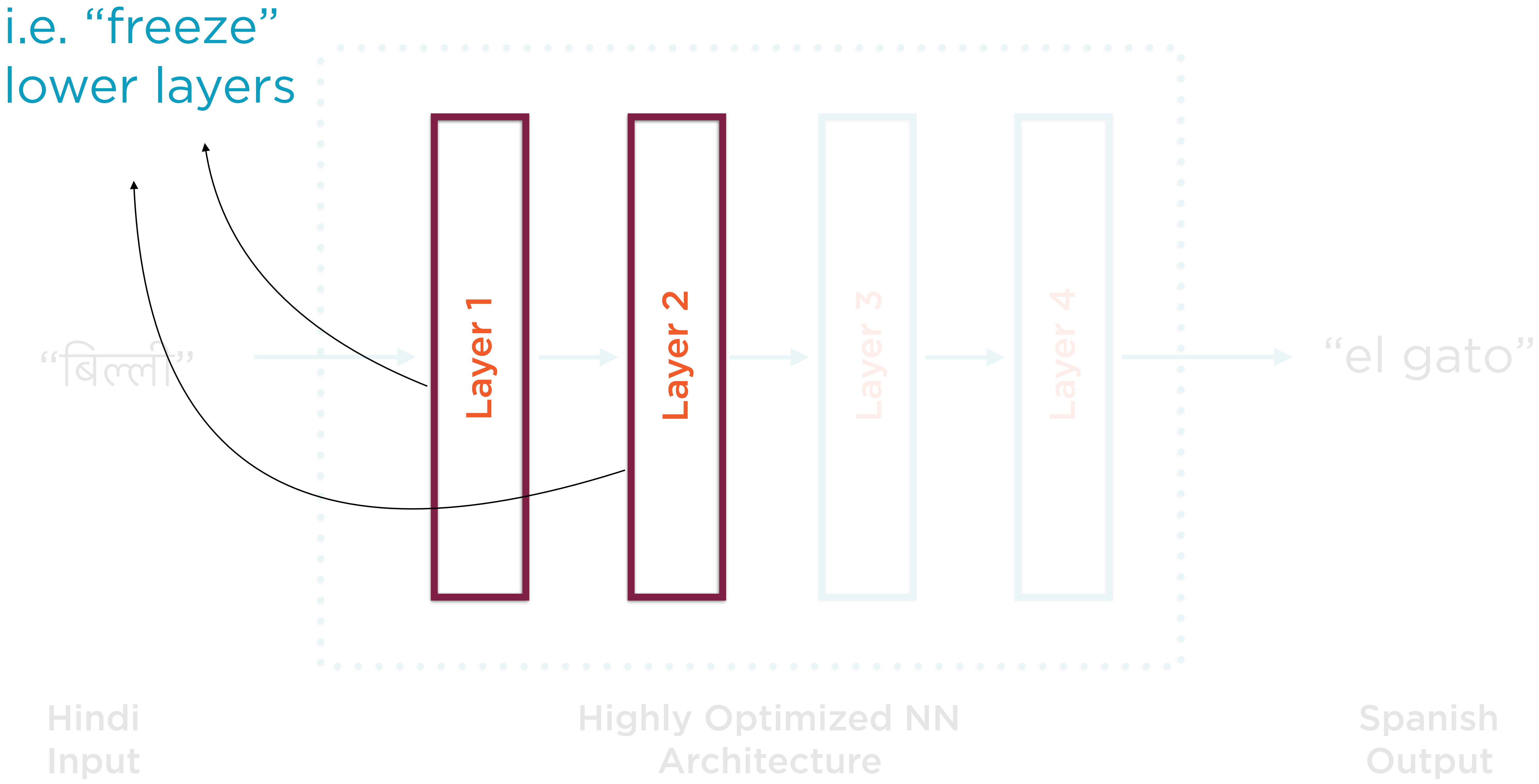
Transfer Learning: Hindi to Spanish



Transfer Learning: Hindi to Spanish



Transfer Learning: Hindi to Spanish



Re-use as-is without even changing parameter weights

Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

Transfer learning for “fixed feature extraction”

Transfer Learning

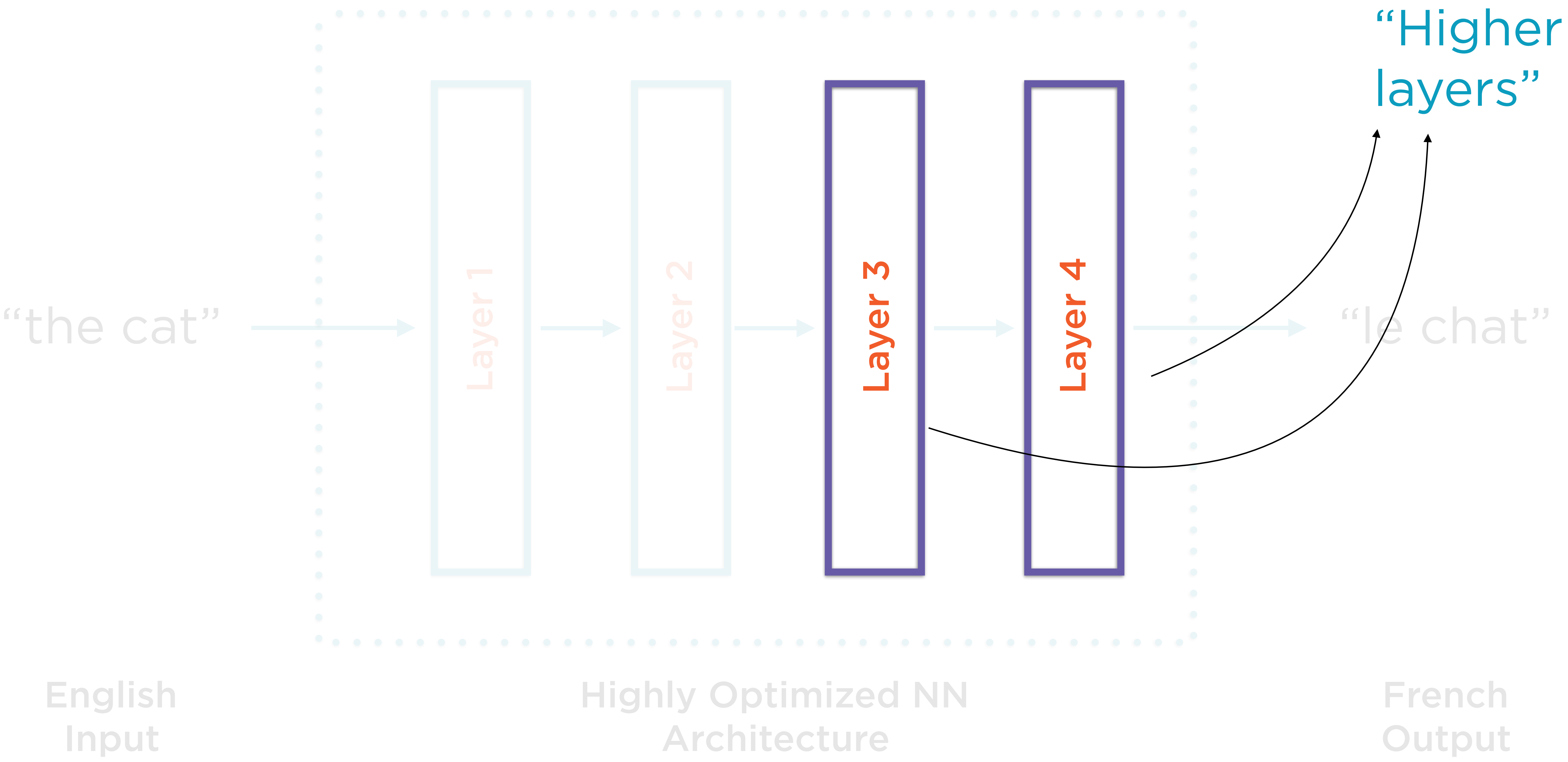
The practice of re-using a trained neural network that solves a problem similar to yours, **freezing the lower layers** and only re-training the higher layers

Can’t avoid this - higher layers are more “high-level”

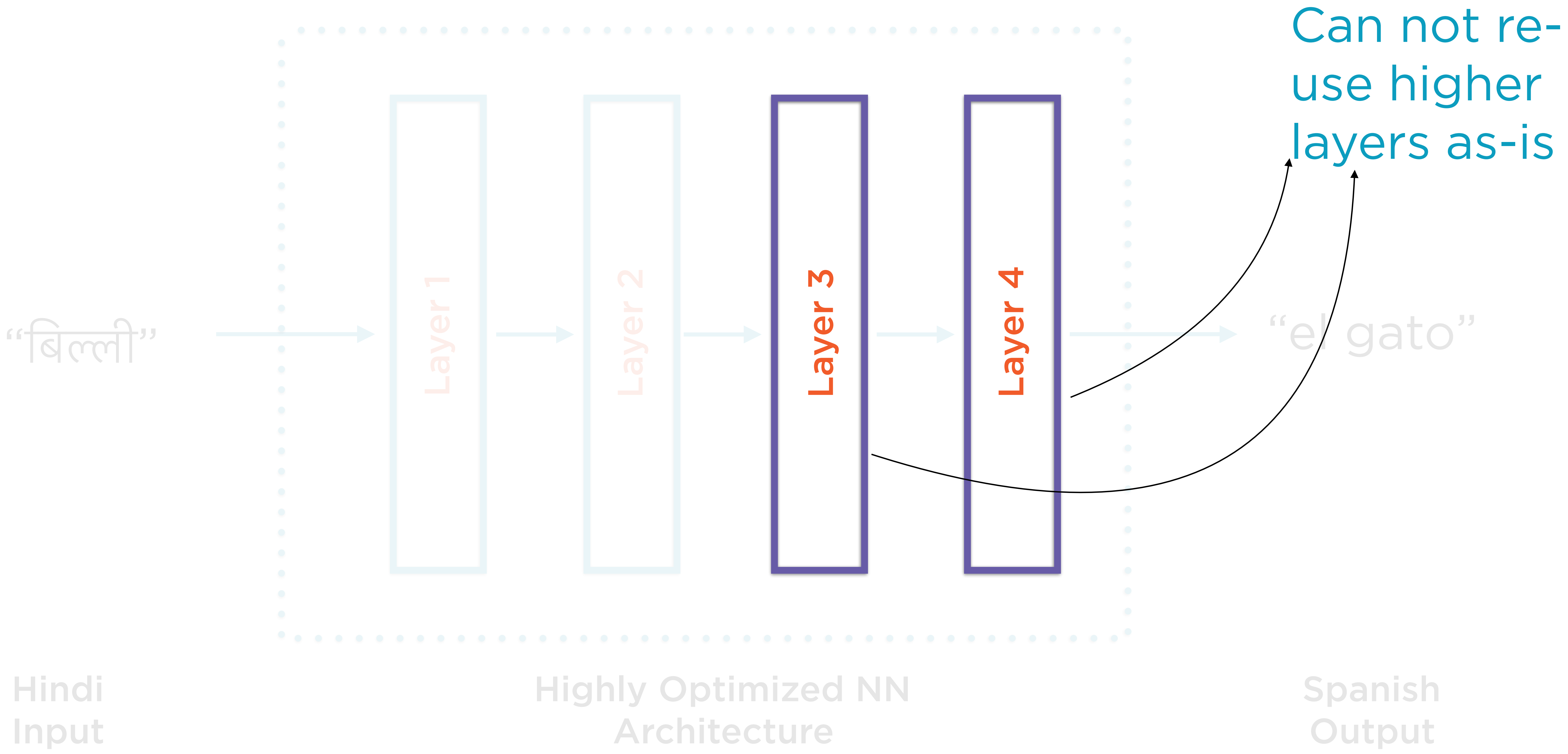
Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

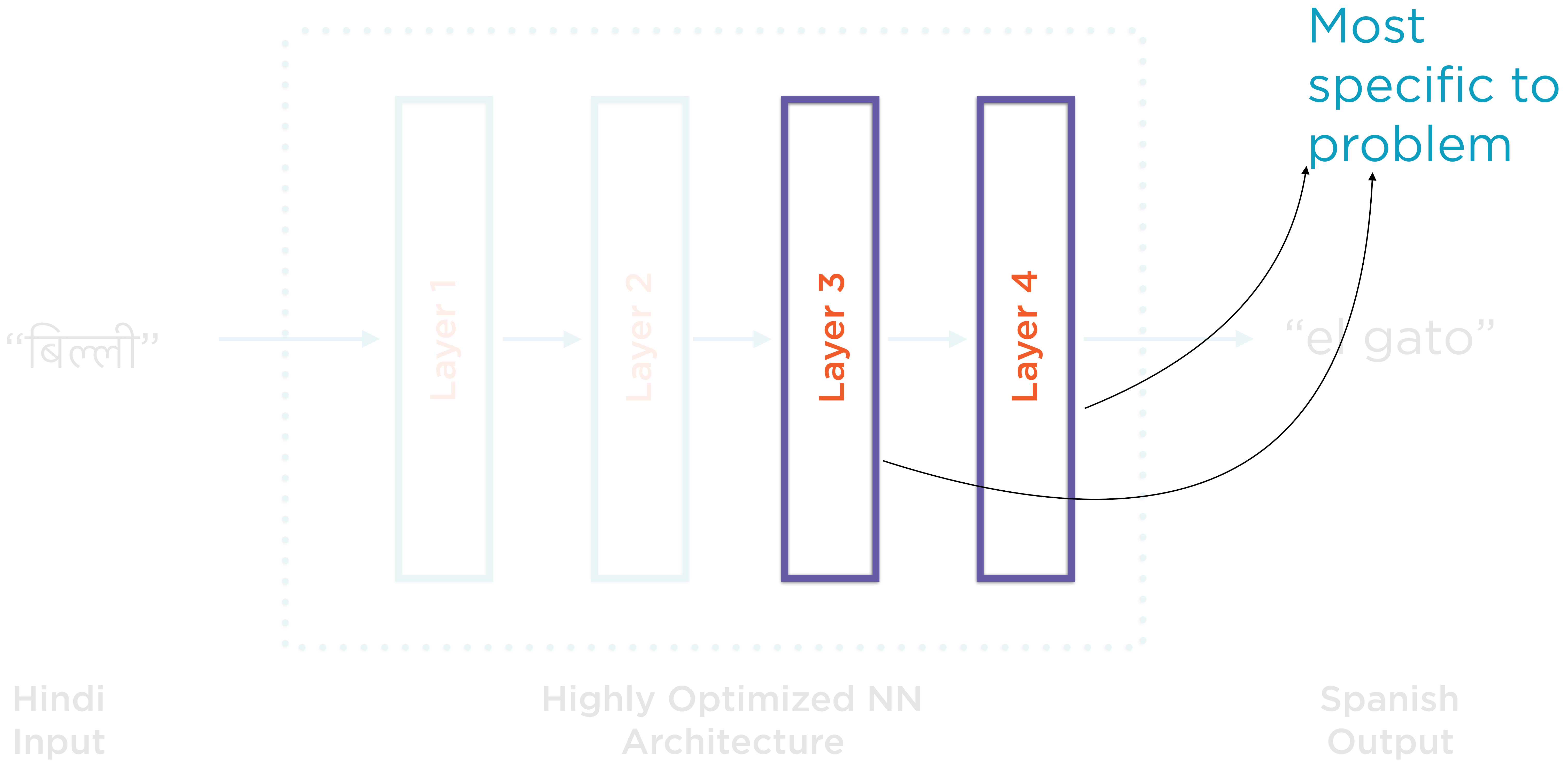
Original Model: English to French



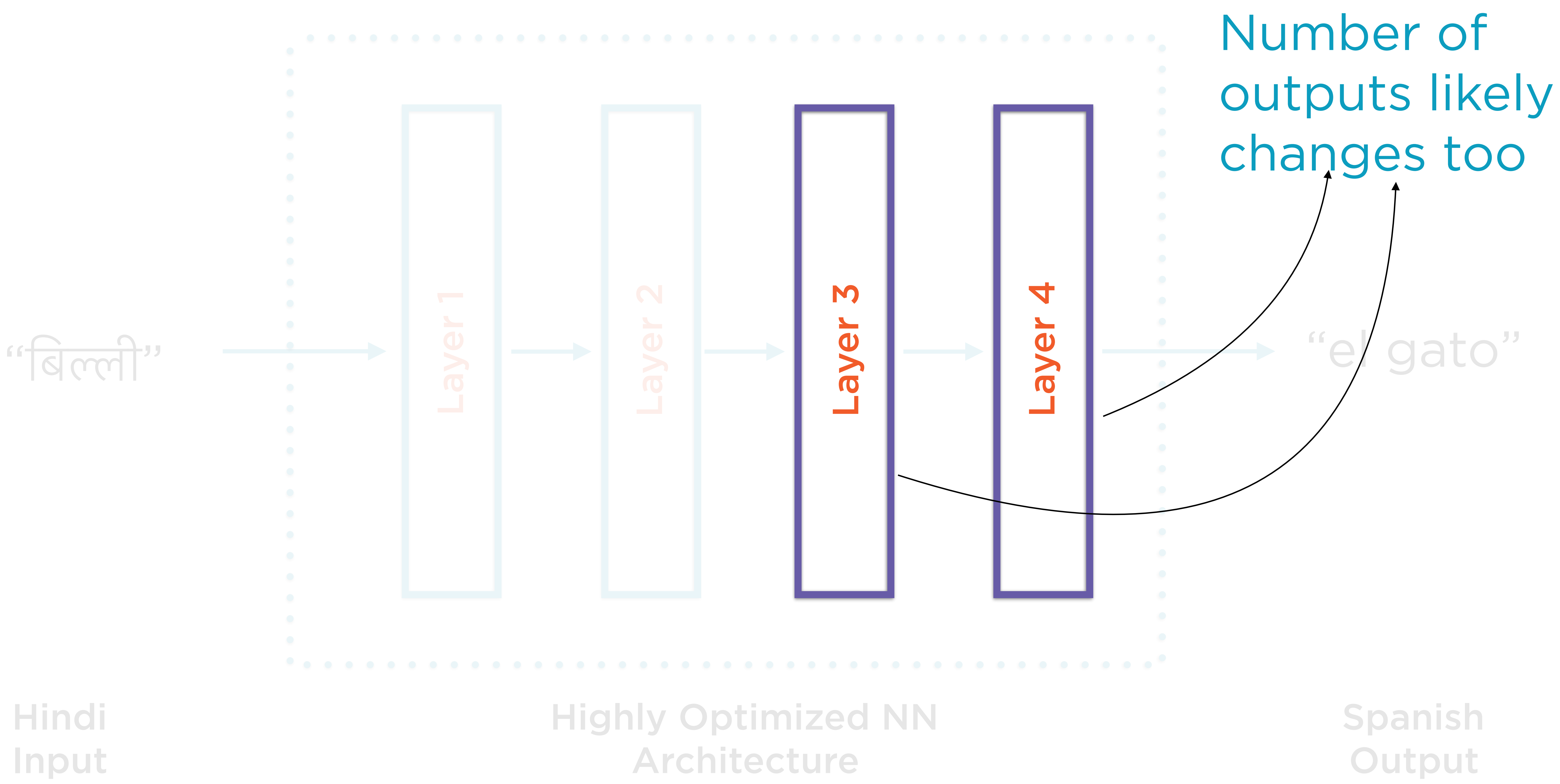
Original Model: English to French



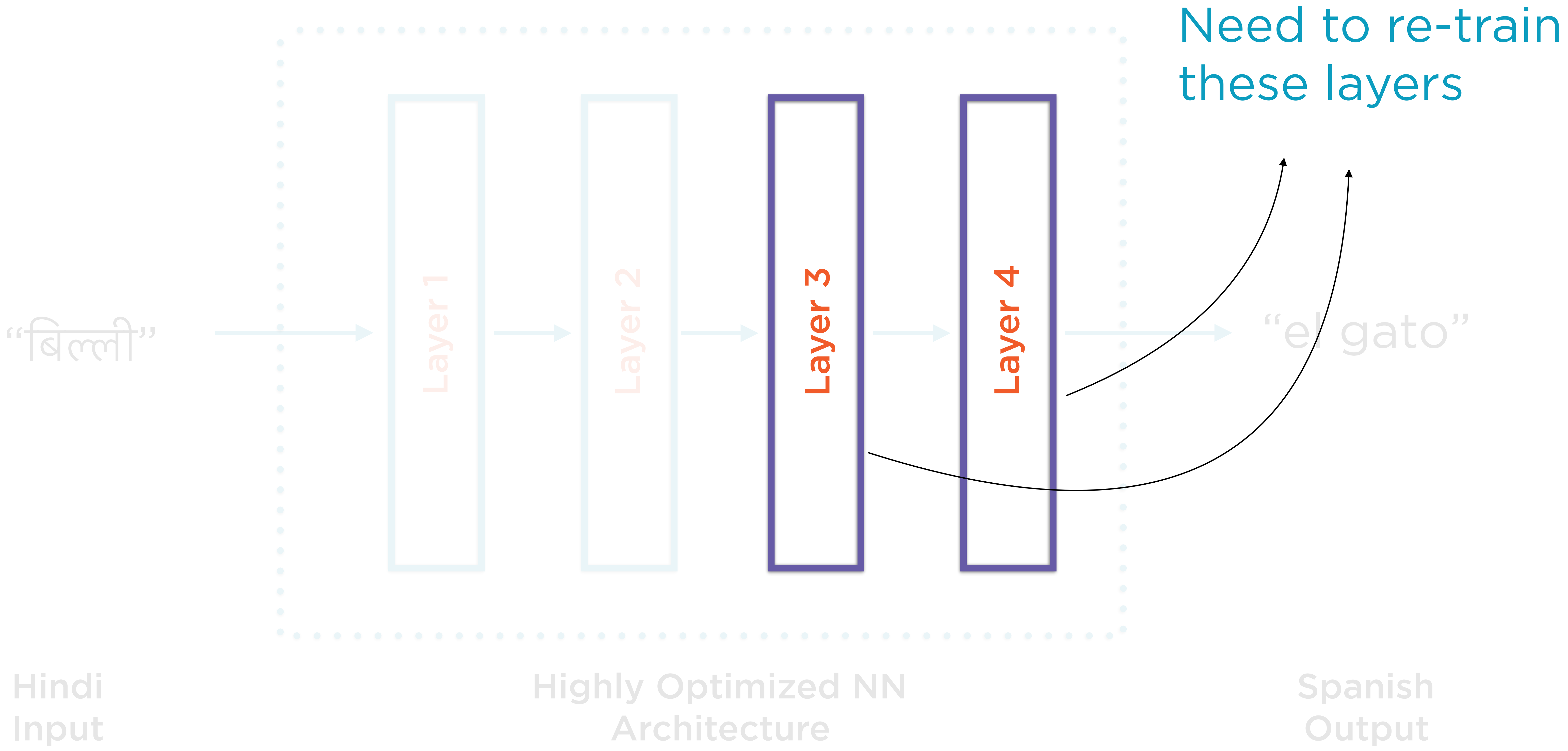
Transfer Learning: Hindi to Spanish



Transfer Learning: Hindi to Spanish



Transfer Learning: Hindi to Spanish



Transfer Learning

The practice of re-using a trained neural network that solves a problem similar to yours, freezing the lower layers and only re-training the higher layers

Benefits of Transfer Learning

“Ride on the shoulders of giants”

- NN architecture
- Choice of initialization
- Activation functions
- Number and density of layers

Benefits of Transfer Learning

“Do more with less”

Make do with less training data

- English to French: Lots of training data
- Hindi to Spanish: Little or no training data

Benefits of Transfer Learning

“Faster, cheaper”

Training process is far faster, easier

- Smaller training data
- Only higher layers to train
- In a cloud-enabled world, less time => less money

Transfer Learning in PyTorch

Support for several famous NN architectures

torchvision.models

- Alexnet
- VGG
- ResNet
- Inception
- ...

Transfer Learning in PyTorch

Support for several famous NN architectures

torchvision.models

- Alexnet
- VGG
- ResNet
- Inception
- ...

Demo

Use the ResNet-18 pre-trained model to classify flowers

ResNet

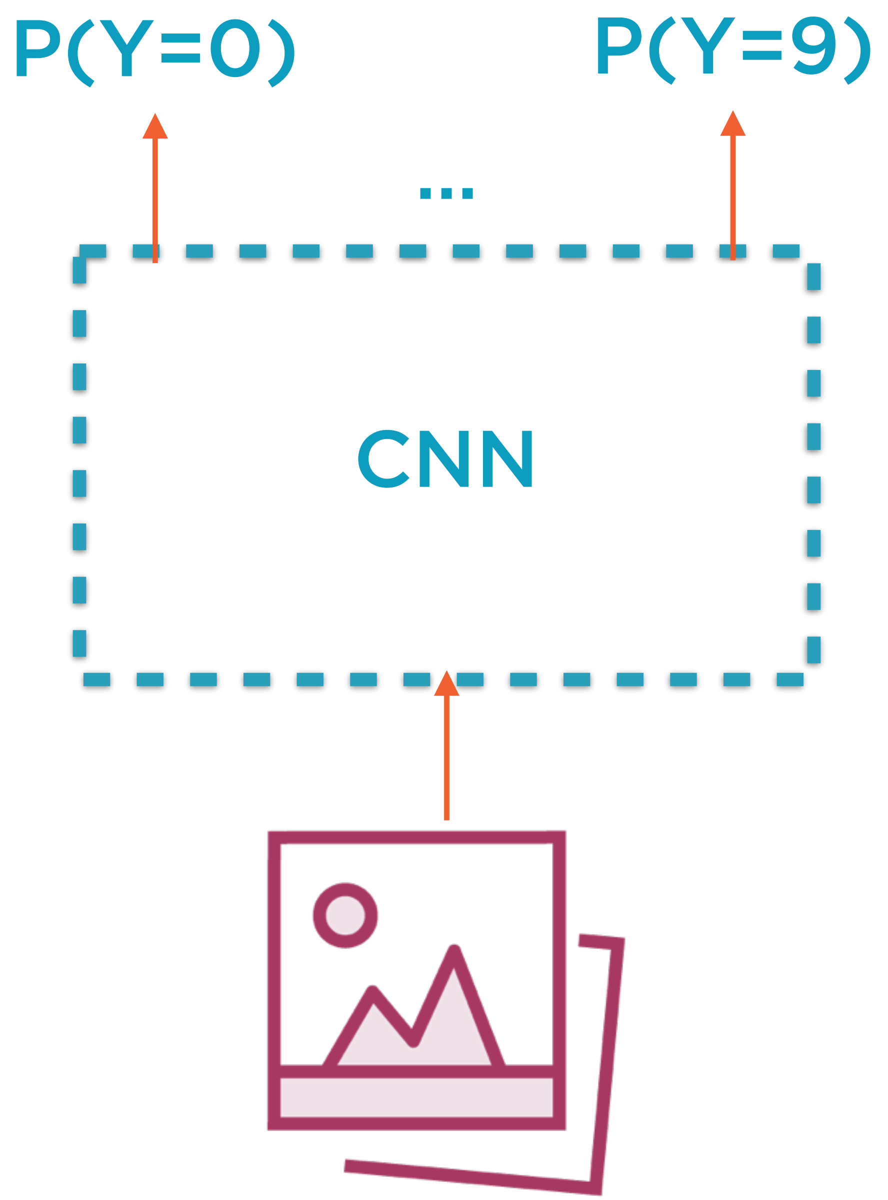
Famous CNN architecture

Won famous challenge in 2015

Extremely deep

“Skip connections” aka shortcut connections

Shares many features with typical CNN architectures



ResNet

Big innovation - “skip connections”

Connect output of lower layers to far-ahead higher layers

Model is forced to focus on what is not learnt by intermediate layers

“Residual Learning”

Summary

Convolutional NNs are a deep learning technique easily implemented in PyTorch

ResNet is a famous CNN architecture

Transfer learning is a great way to re-use pre-trained models

PyTorch offers great support for transfer learning

Provides a ResNet model

Image classification using transfer learning and ResNet