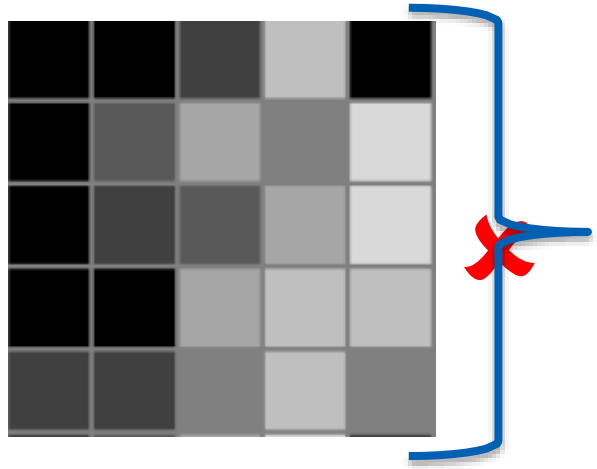
Convolutional Neural Networks

Introduction

- A convolutional neural network (or ConvNet) is a type of feed-forward artificial neural network
- The architecture of a ConvNet is designed to take advantage of the 2D structure of an input image.



VS

A	22	1A	a@a	1	aa
B	33	2B	b@b	2	bb
C	44	3C	c@c	3	cc
D	55	4D	d@d	4	dd
E	66	5E	e@e	5	ee

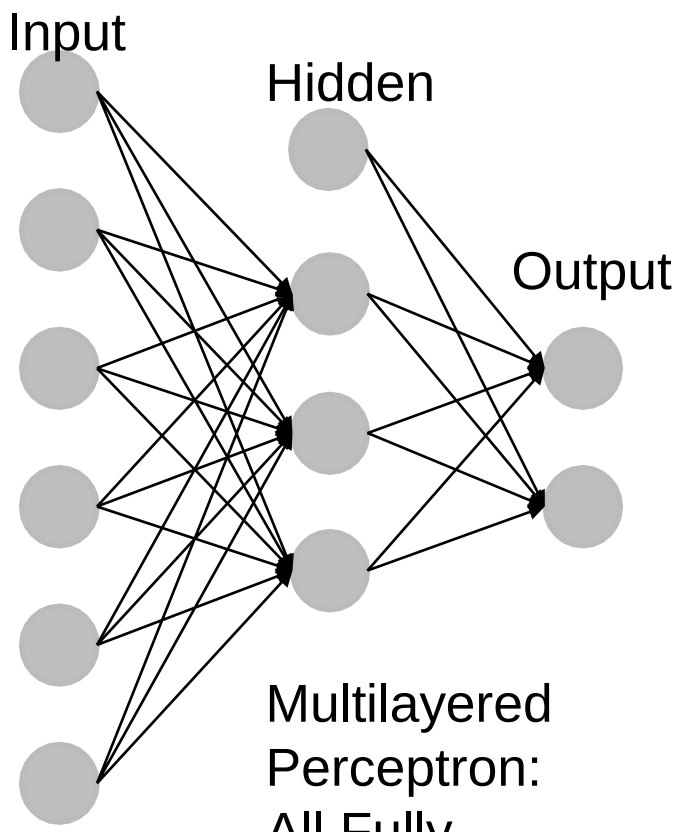
- A ConvNet is comprised of one or more convolutional layers (often with a pooling step) and then followed by one or more fully connected layers as in a standard multilayer neural network.

Motivation behind ConvNets

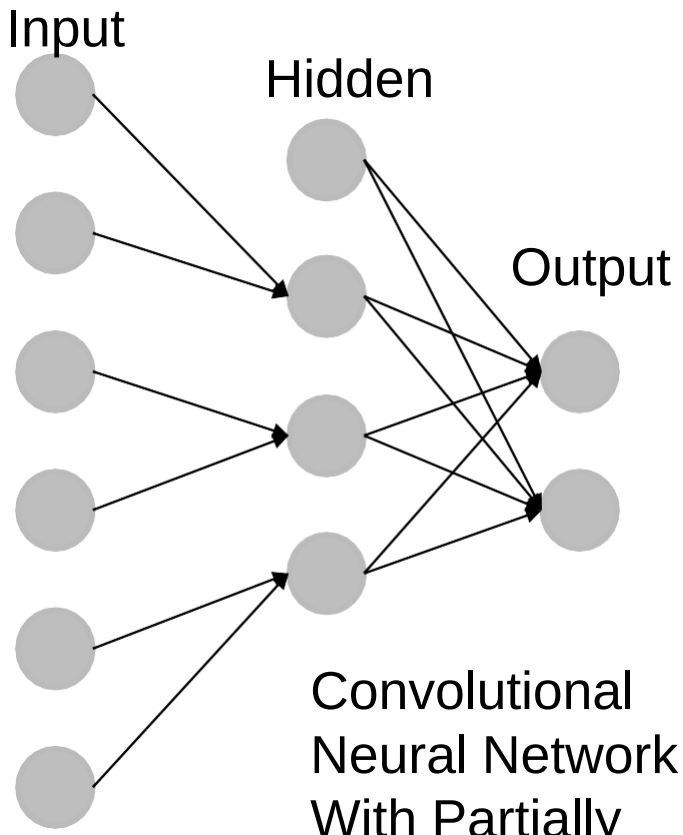
- Consider an image of size 200x200x3 (200 wide, 200 high, 3 color channels)
  - a **single fully-connected neuron** in a first hidden layer of a regular Neural Network would have 200*200*3 = 120,000 weights.
  - Due to the presence of several such neurons, this full connectivity is waste and the huge number of parameters would quickly lead to overfitting

- However, in a ConvNet, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner.
  - the final output layer would have dimensions 1x1xN, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores (for N classes), arranged along the depth dimension
  - Vanishing Gradient Problem in MLP

MLP VS ConvNet



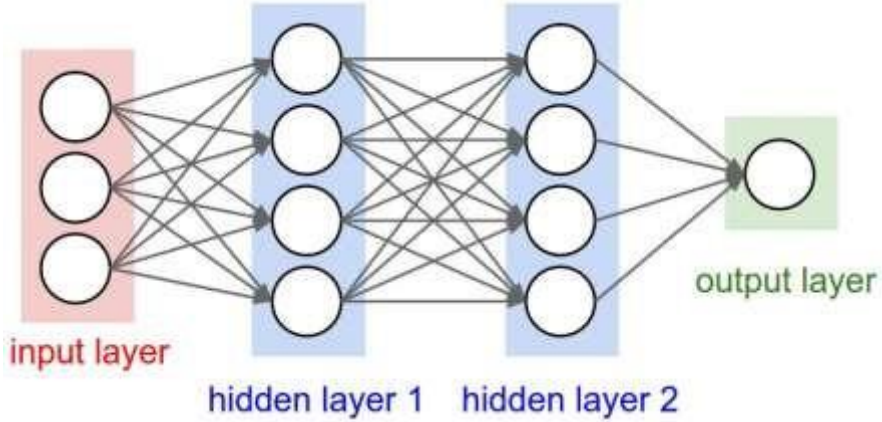
Multilayered Perceptron: All Fully Connected Layers



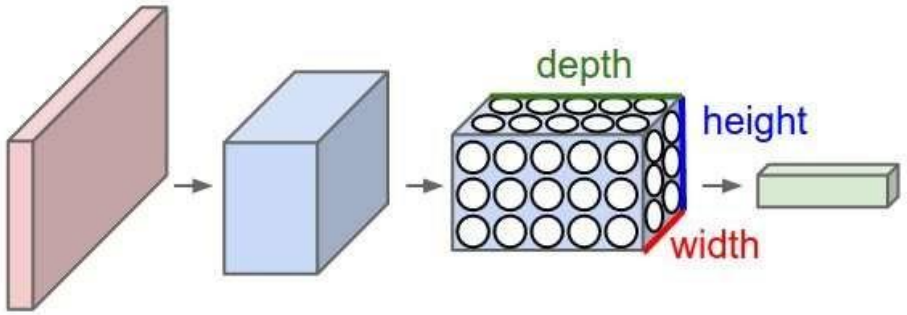
Convolutional Neural Network With Partially Connected Convolution Layer

MLP vs ConvNet

- A regular 3-layer Neural Network.



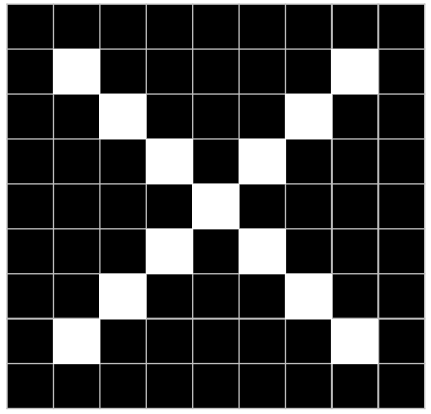
- A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers.



How ConvNet Works

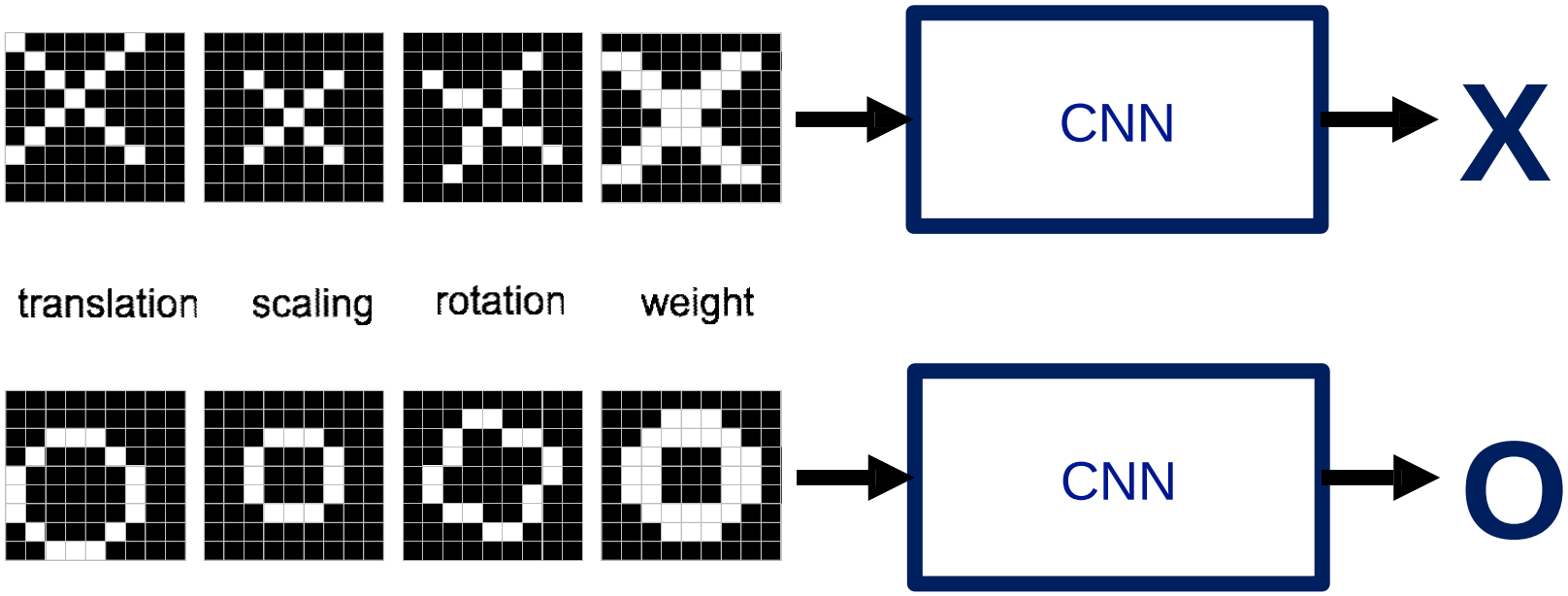
- For example, a ConvNet takes the input as an image which can be classified as ‘X’ or ‘O’

- In a simple case, ‘X’ would look like:



How ConvNet Works

- What about trickier cases?



How ConvNet Works – What Computer Sees

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

How ConvNet Works

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

How ConvNet Works – What Computer Sees

- Since the pattern doesnot match exactly, the computer will not be able to classify this as ‘X’

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

ConvNet Layers (At a Glance)

- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.

- RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged.

- POOL layer will perform a downsampling operation along the spatial dimensions (width, height).

- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1xN], where each of the N numbers correspond to a class score, such as among the N categories.

Recall – What Computer Sees

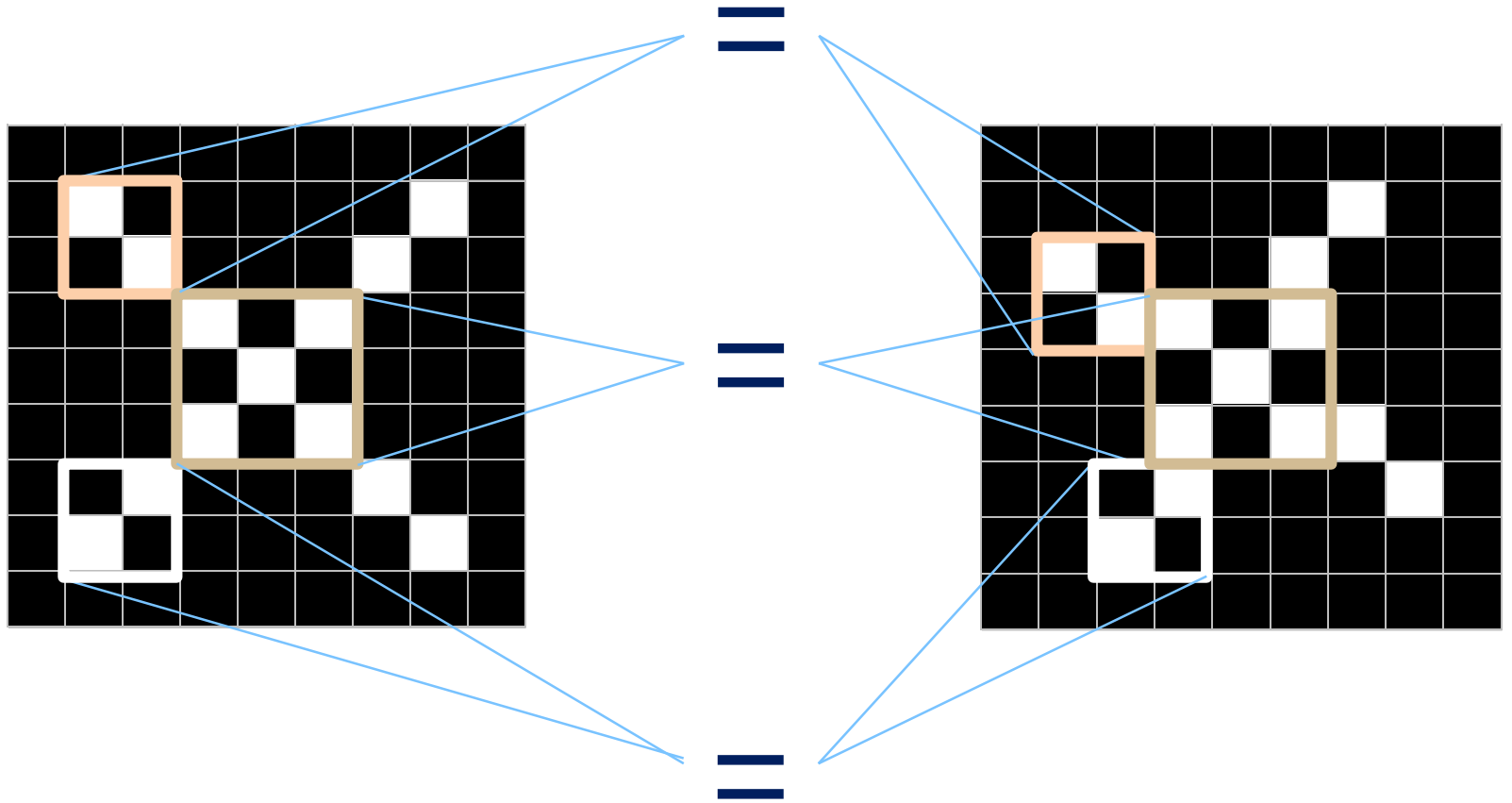
- Since the pattern doesnot match exactly, the computer will not be able to classify this as ‘X’

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	X	-1	-1	-1	-1	X	X	-1
-1	X	X	-1	-1	X	X	-1	-1
-1	-1	X	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	X	-1	-1
-1	-1	X	X	-1	-1	X	X	-1
-1	X	X	-1	-1	-1	-1	X	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

- What got changed?

Convolutional Layer

- Convolution layer will work to identify patterns (features) instead of individual pixels



Convolutional Layer - Filters

- The CONV layer's parameters consist of a set of learnable filters.
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
- During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.

1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

-1	-1	1
-1	1	-1
1	-1	-1

Vertical edge detection

$3\times1+1\times1+2\times1+0\times0+5\times0+7\times0+1\times-1+8\times-1+2\times-1=-5$

3 (1)	0 (0)	1 (-1)	2	7	4
1 (1)	5 (0)	8 (-1)	9	3	1
2 (1)	7 (0)	2 (-1)	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6×6

"convolution"

*

1	0	-1
1	0	-1
1	0	-1

3×3 filter

=

-5			

4×4

Vertical edge detection

$3\times1+1\times1+2\times1+0\times0+5\times0+7\times0+1\times-1+8\times-1+2\times-1=-5$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6×6

"convolution"

*

1	0	-1
1	0	-1
1	0	-1

3×3 filter

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

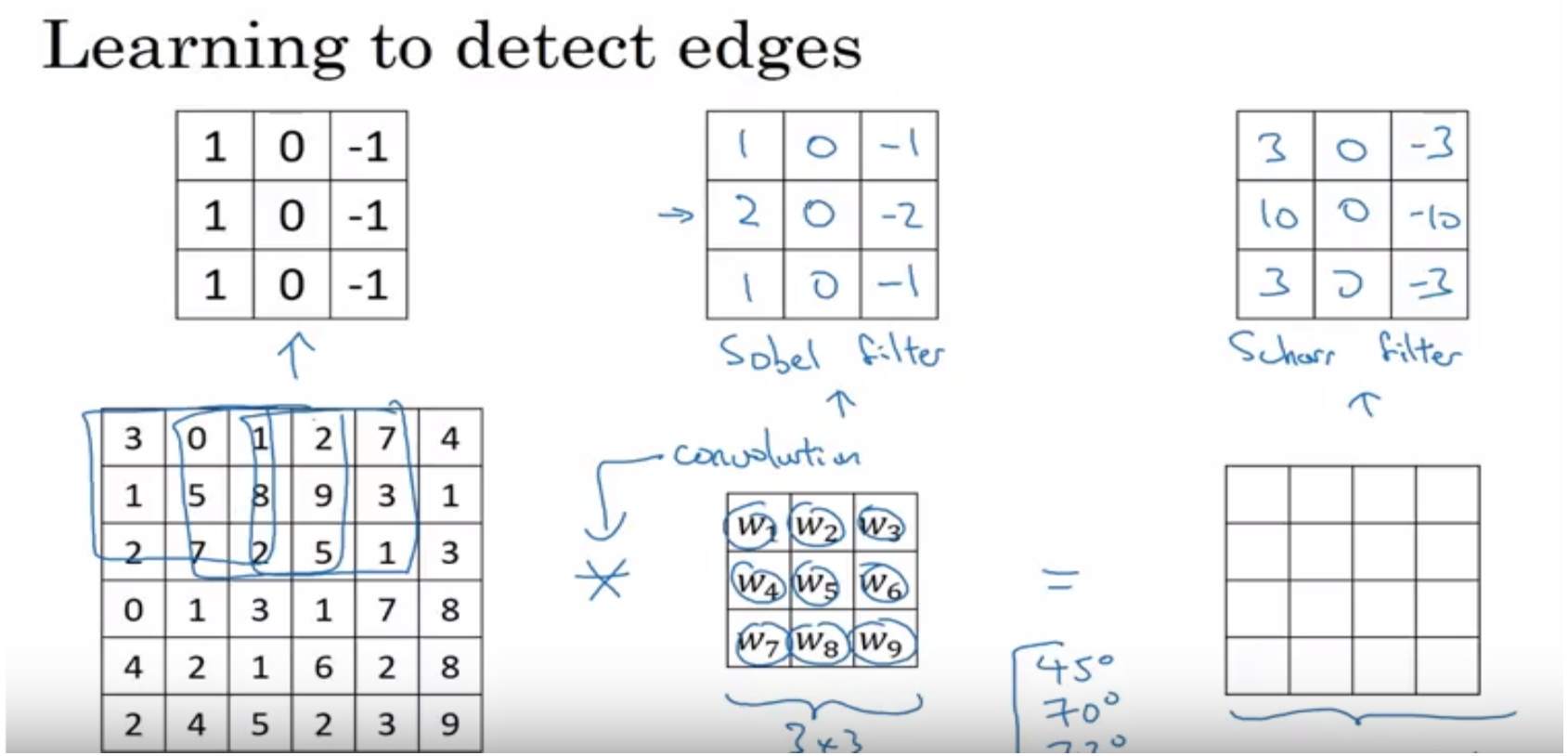
4×4

python: conv-forward

tensorflow: tf.nn.conv2d

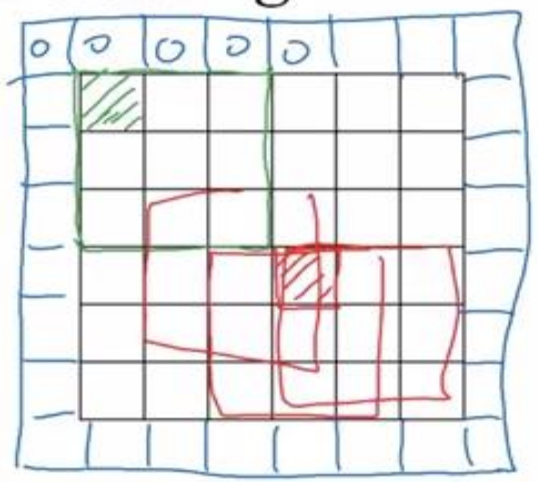
keras: Conv2D

Learning to detect edges



Padding

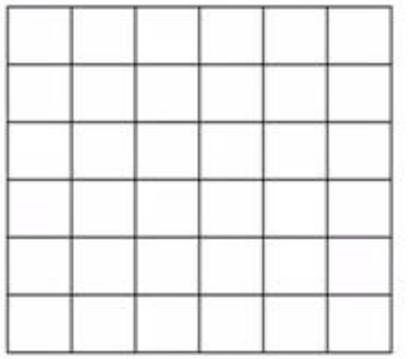
- shrinky output
- throw away info from edge



*

3×3
f×f

=



6×6

6×6 → 8×8
n×n

p = padding

n-f+1 × n-f+1
6-3+1=4

~~4×4~~

n+2p-f+1 × n+2p-f+1
6+2-3+1 × ___ = 6×6

Valid and Same convolutions

↳ no padding

"Valid": $n \times n \quad * \quad f \times f \quad \rightarrow \quad n-f+1 \times n-f+1$

$6 \times 6 \quad * \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

"Same": Pad so that output size is the same as the input size.

$n+2p-f+1 \times n+2p-f+1$

$p+2p-f+1 = p \Rightarrow \boxed{p = \frac{f-1}{2}}$

$3 \times 3 \quad p = \frac{3-1}{2} = 1$

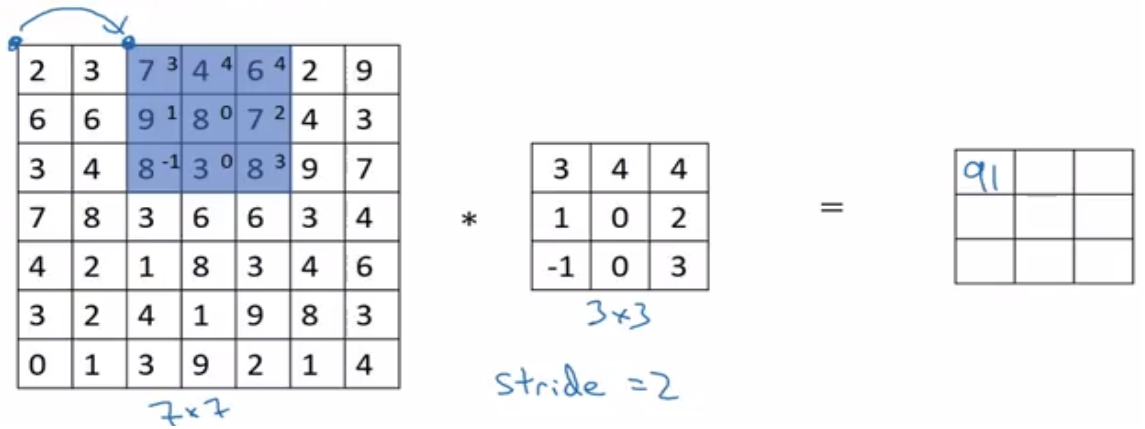
$5 \times 5 \quad p = 2$

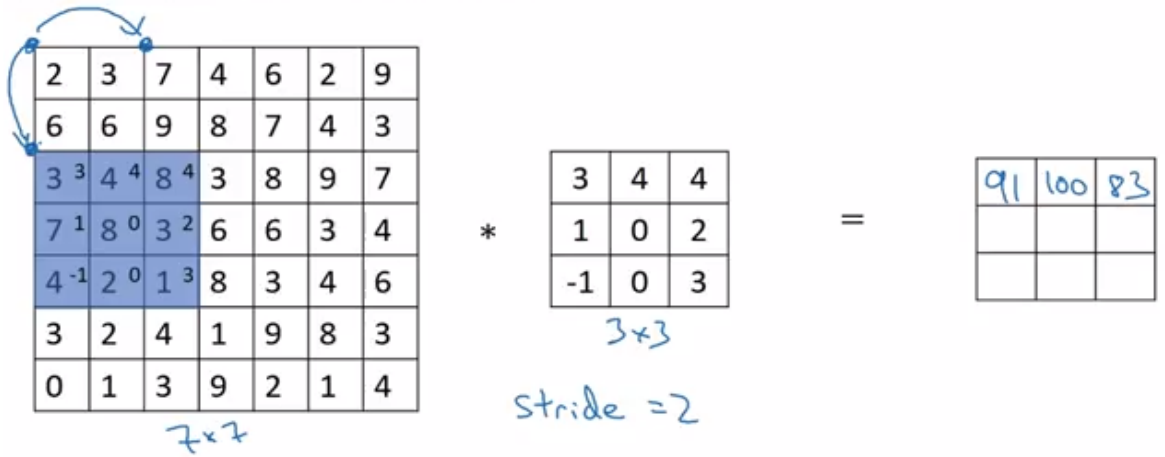
f = 5

f is usually odd

1×1
3×3
5×5
7×7

Strided convolution





$n \times n$ * $f \times f$

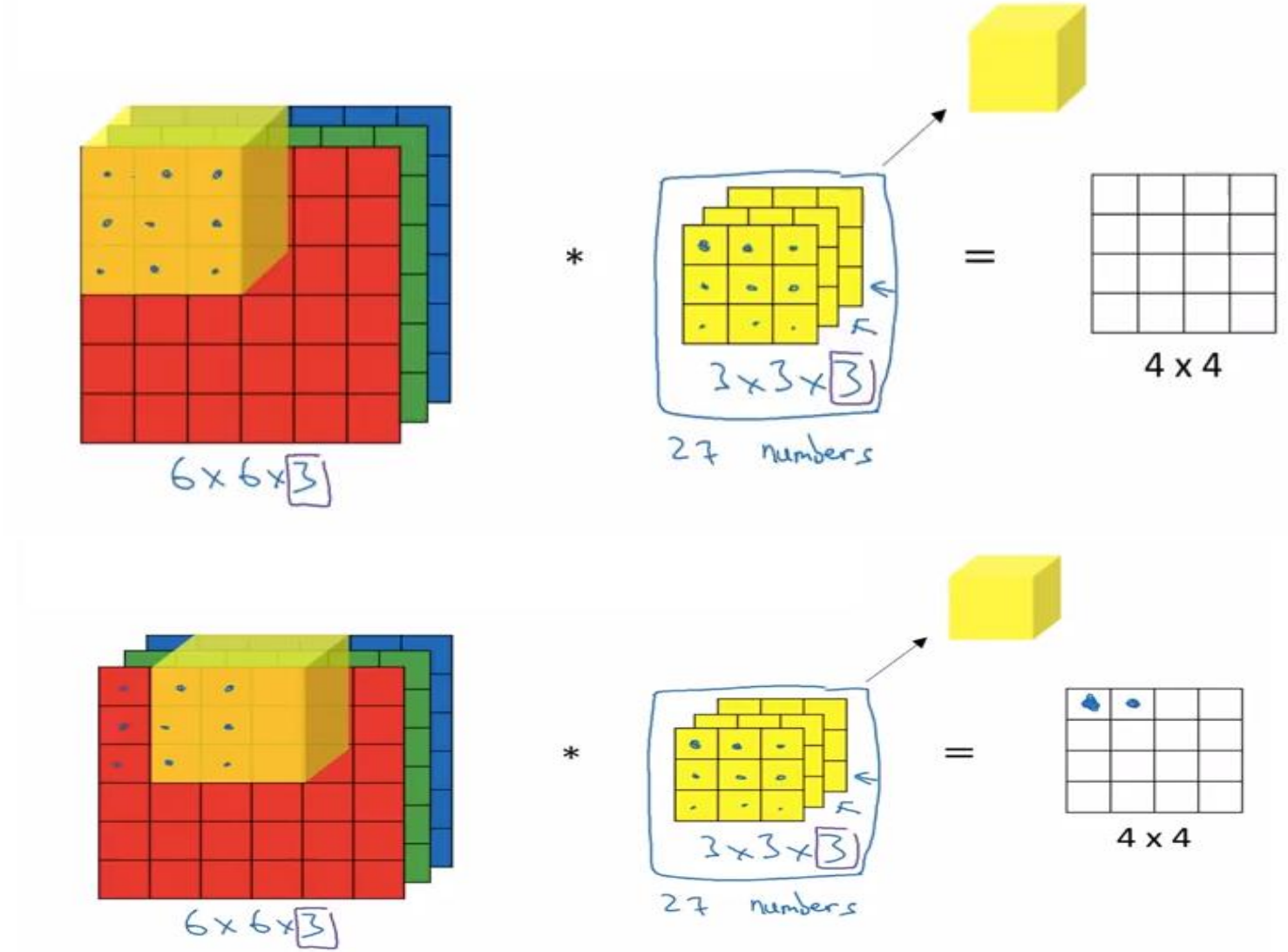
padding p stride s

$s = 2$

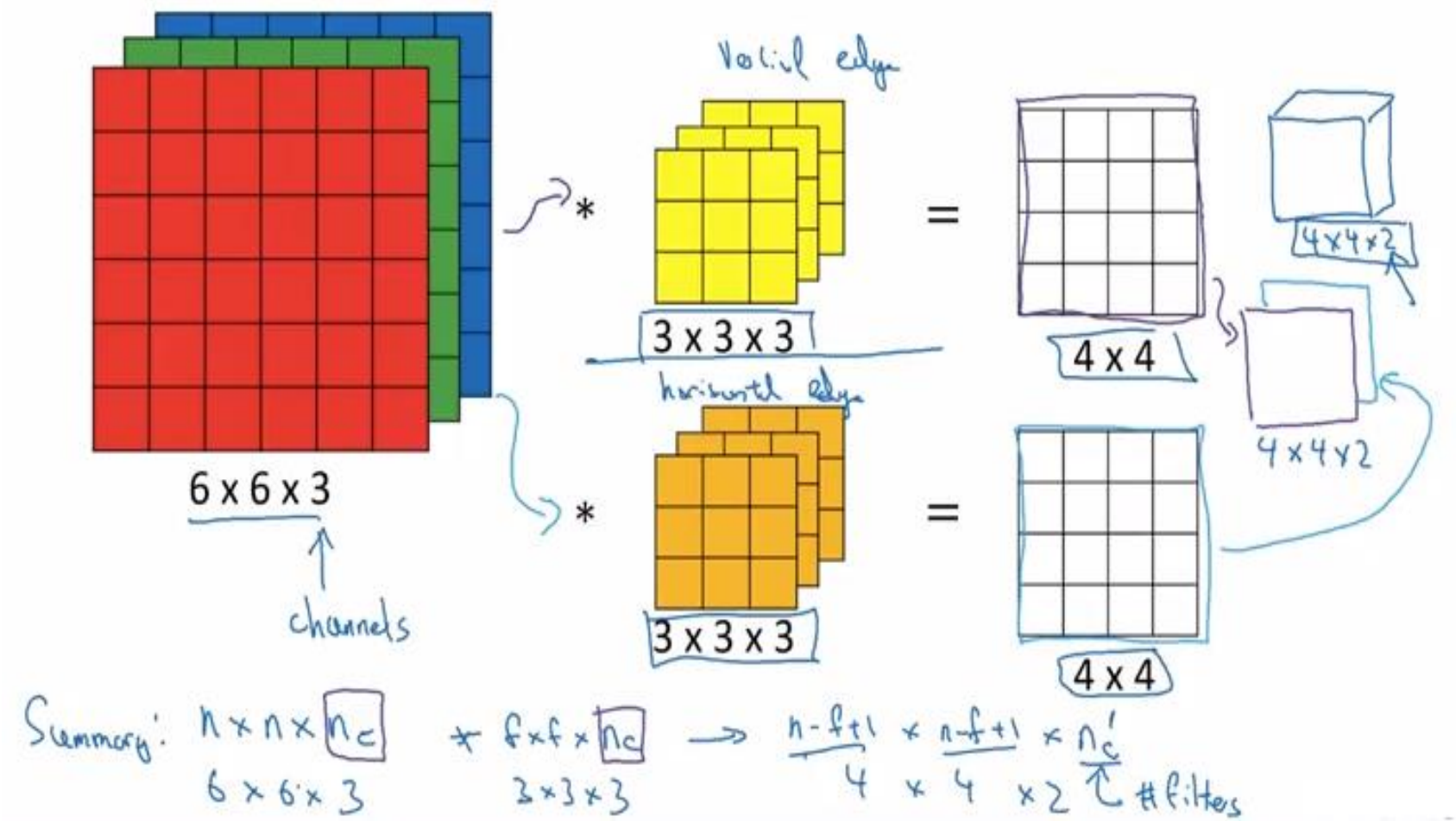
$$\frac{n+2p-f}{s}+1 \times \frac{n+2p-f}{s}+1$$

$$\frac{7+0-3}{2}+1 = \frac{4}{2}+1 = 3$$

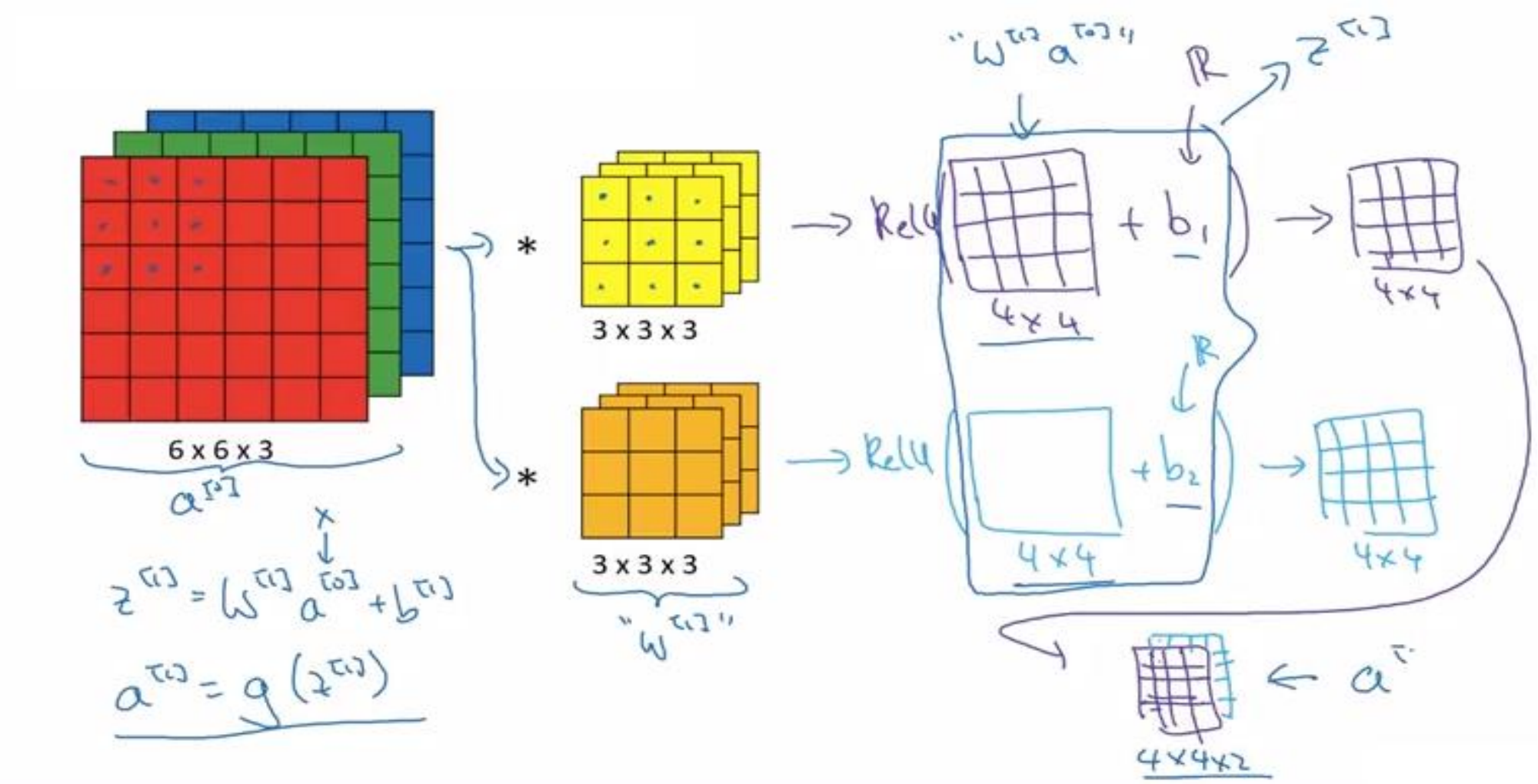
Convolutions on RGB image



Multiple Filters



Example of a layer



Convolutional Layer – Filters – Computation Example

Input Size (W): 9
Filter Size (F): 3 X 3
Stride (S): 1
Filters: 1
Padding (P): 0

9 X 9

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

⊗

1	-1	-1
-1	1	-1
-1	-1	1

=

7 X 7

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Feature Map Size = 1+ (W – F + 2P)/S
= 1+ (9 – 3 + 2 X 0)/1 = 7

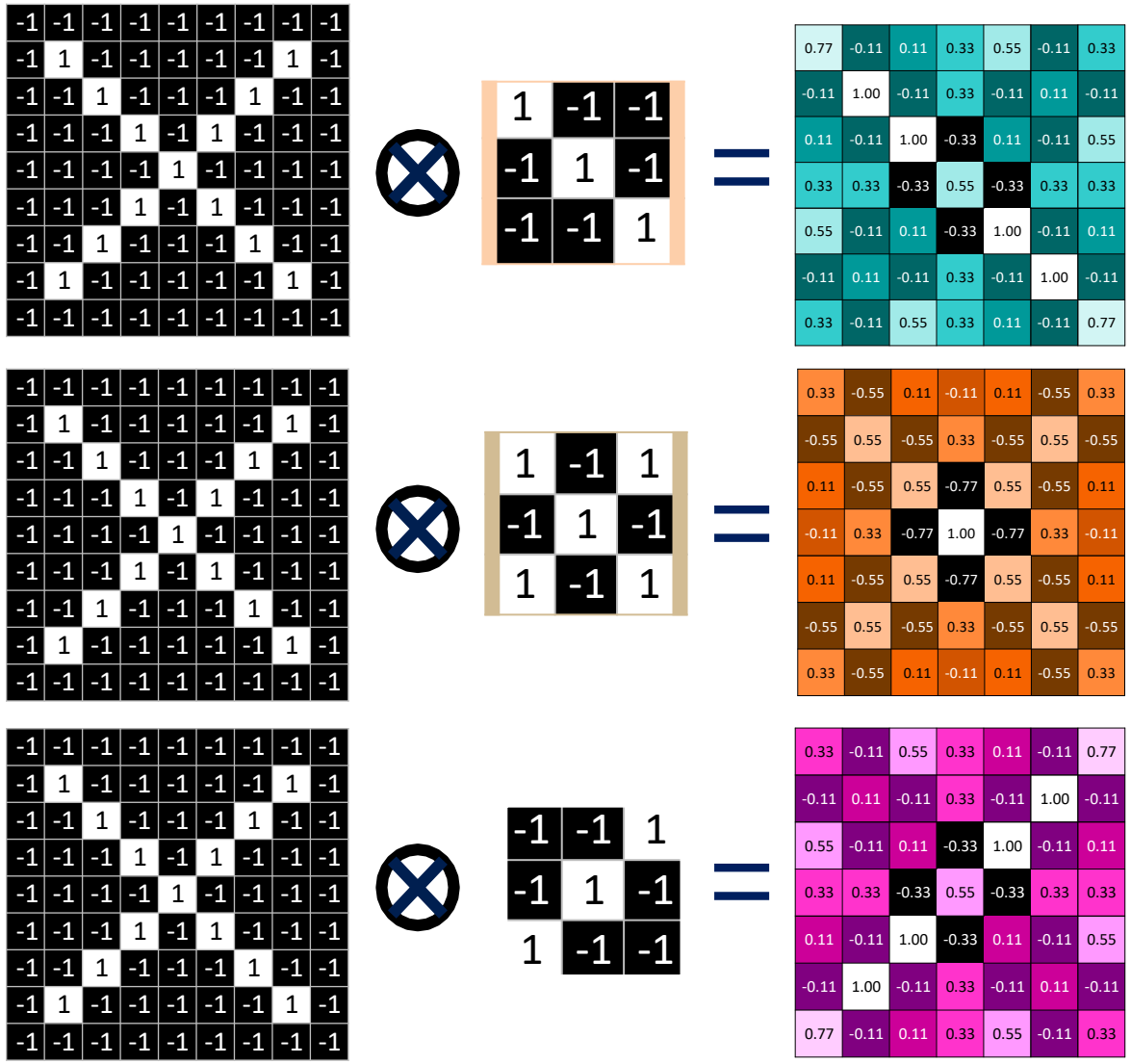
Convolutional Layer – Filters – Output Feature Map

- Output Feature Map of One complete convolution:
  - Filters: 3
  - Filter Size: 3 X 3
  - Stride: 1

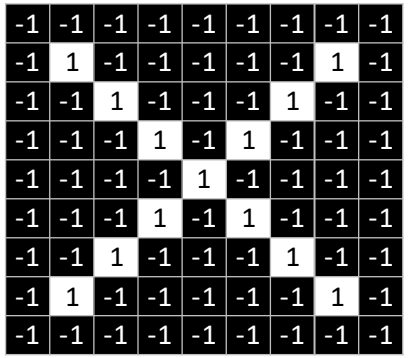
- Conclusion:
  - Input Image:

    9 X 9
  - Output of Convolution:

    7 X 7 X 3



Convolutional Layer – Output



-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

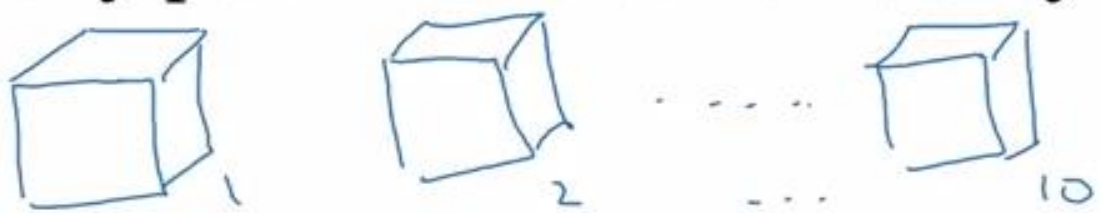
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

Number of parameters in one layer

If you have 10 filters that are 3 x 3 x 3 in one layer of a neural network, how many parameters does that layer have?



3×3×3

27 parameters.

+bias

→ 28 parameters.

280 parameters.

Summary of notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$ ← #filters in layer l.

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ ←

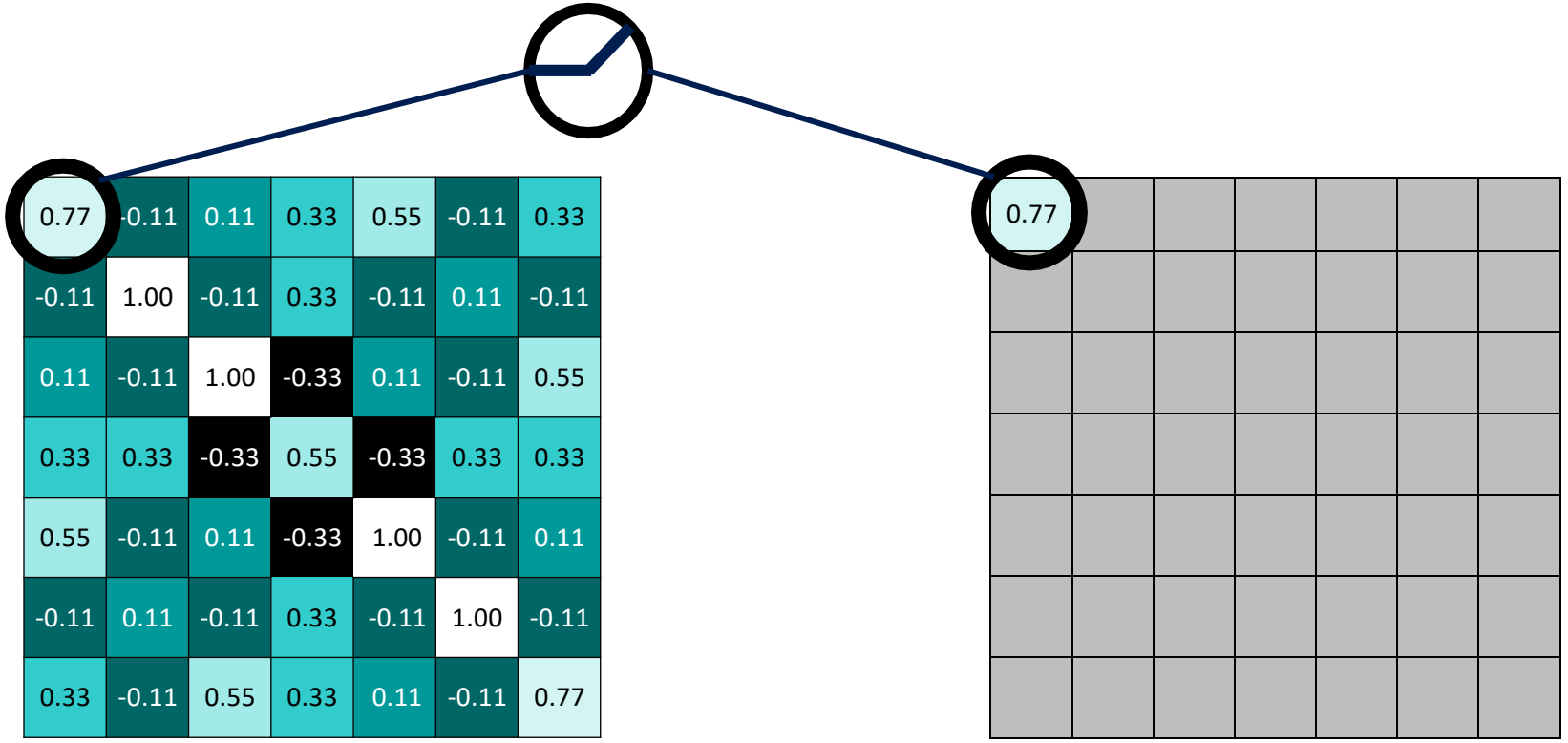
Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ ←

$$n_{H,W}^{[l]} = \left\lfloor \frac{n_{H,W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

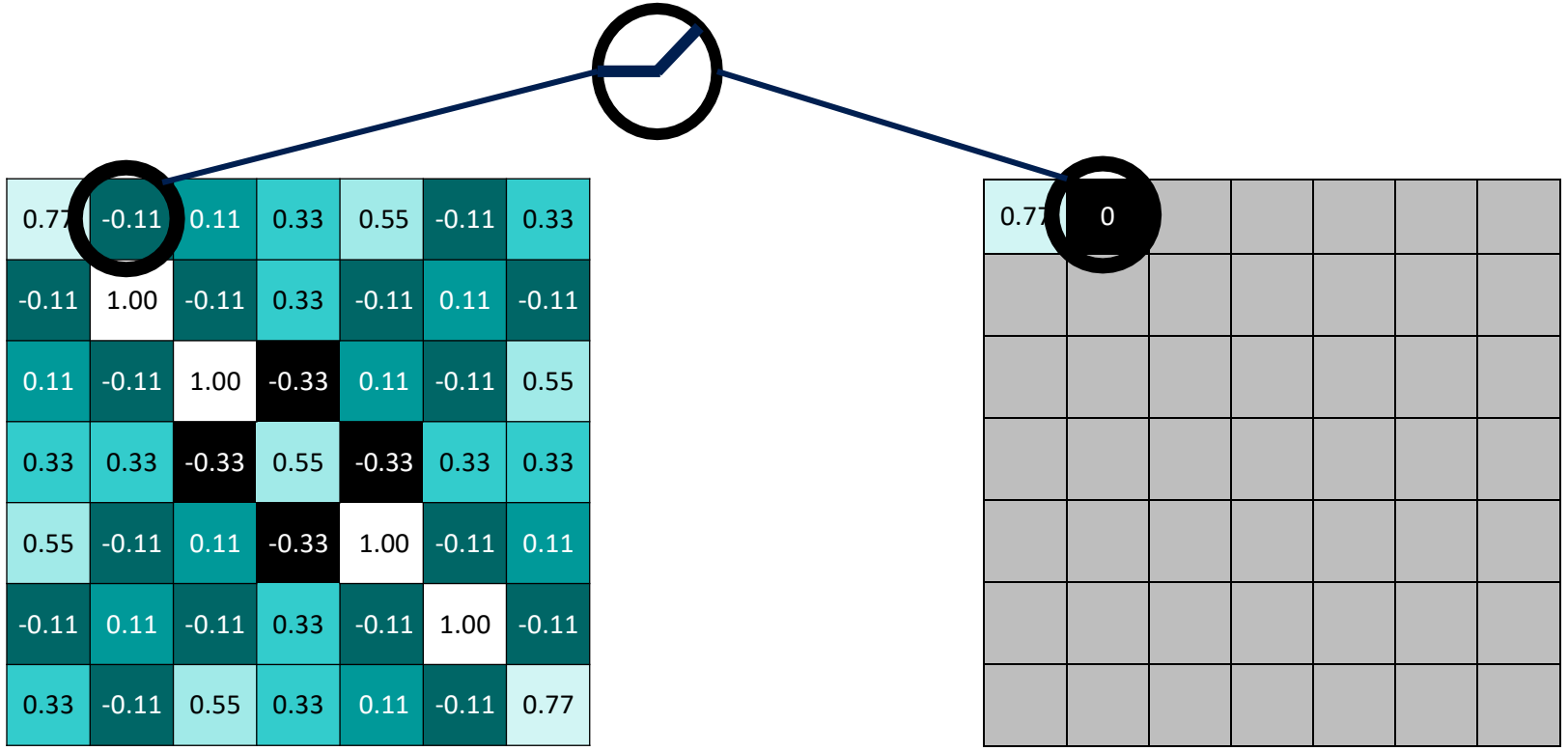
$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$n_c \times n_H \times n_W$

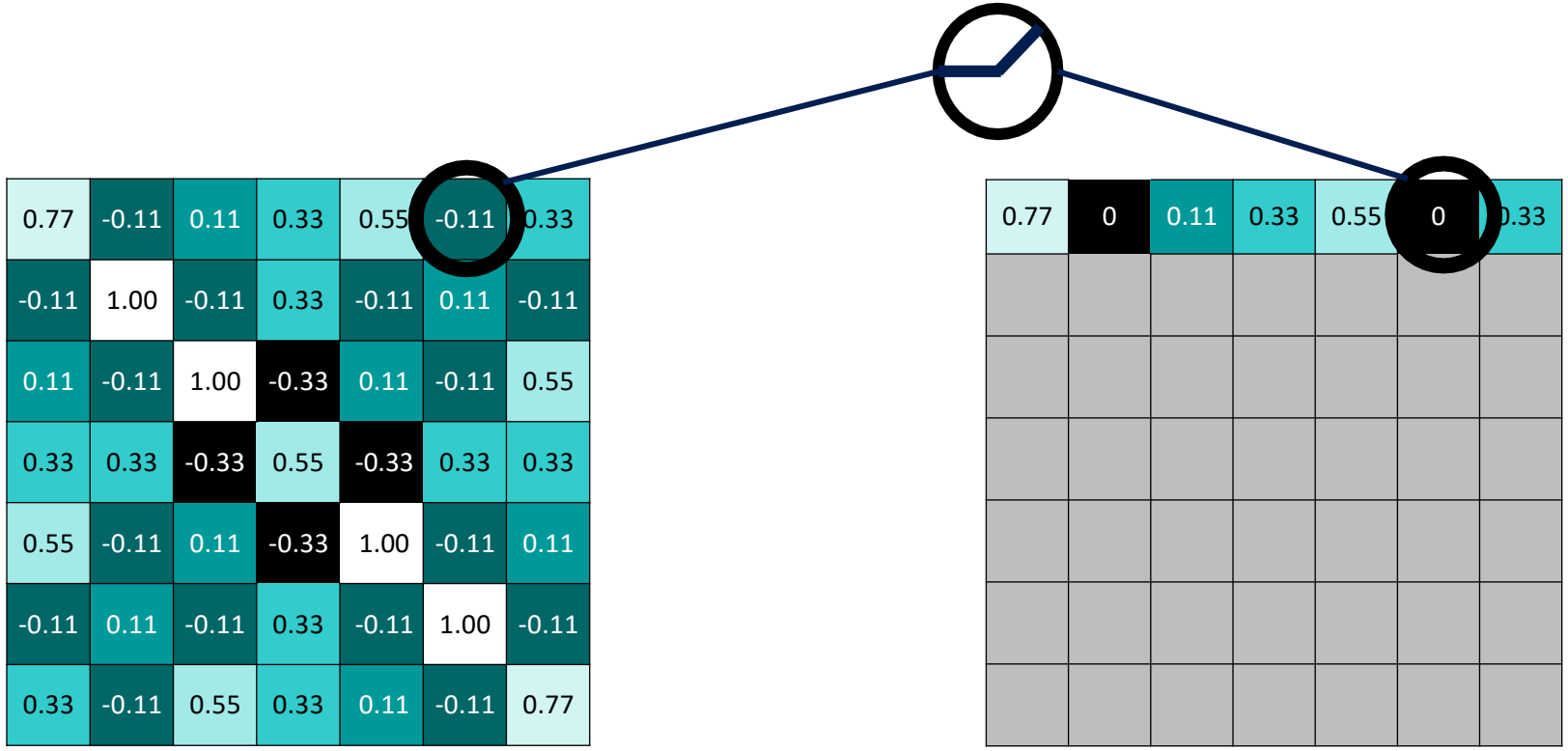
Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)

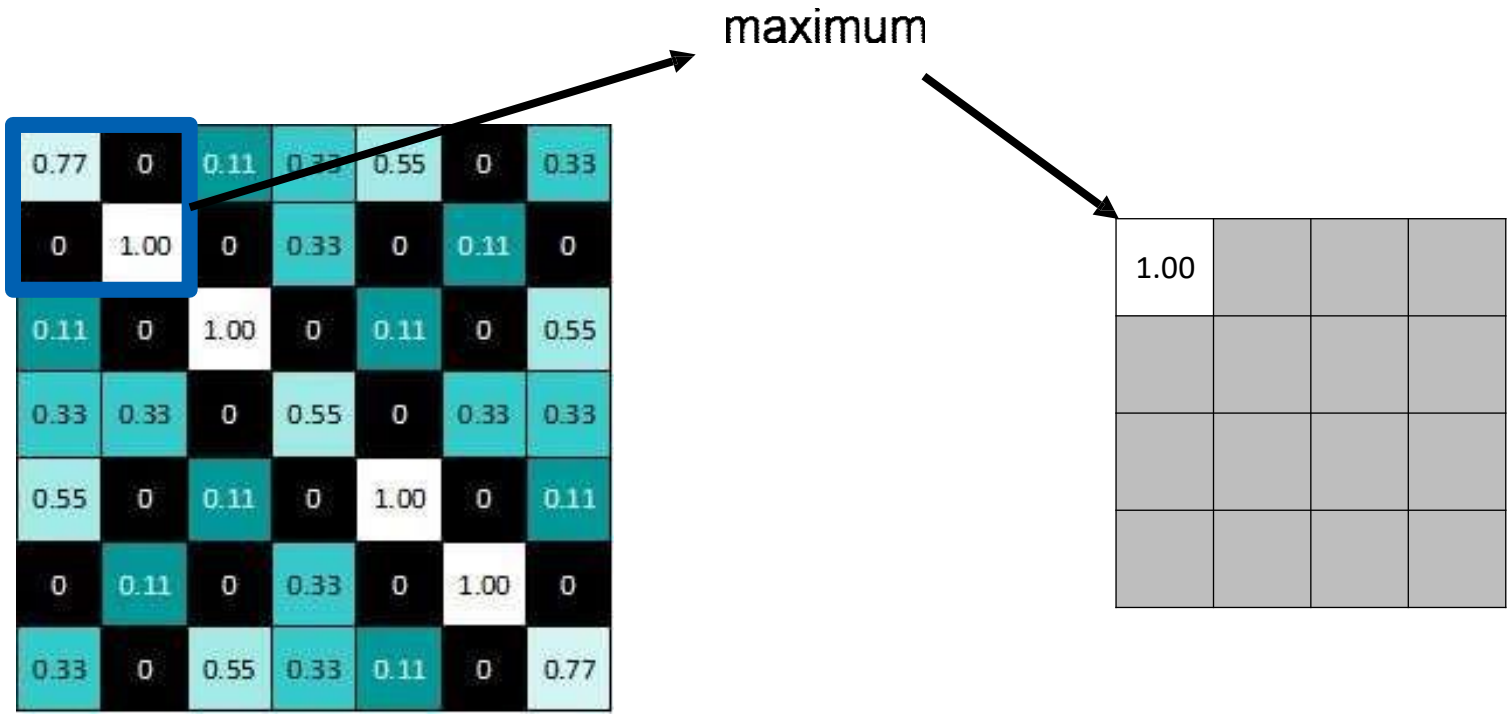


Pooling Layer

- The pooling layers down-sample the previous layers feature map.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network
- The pooling layer often uses the Max operation to perform the downsampling process

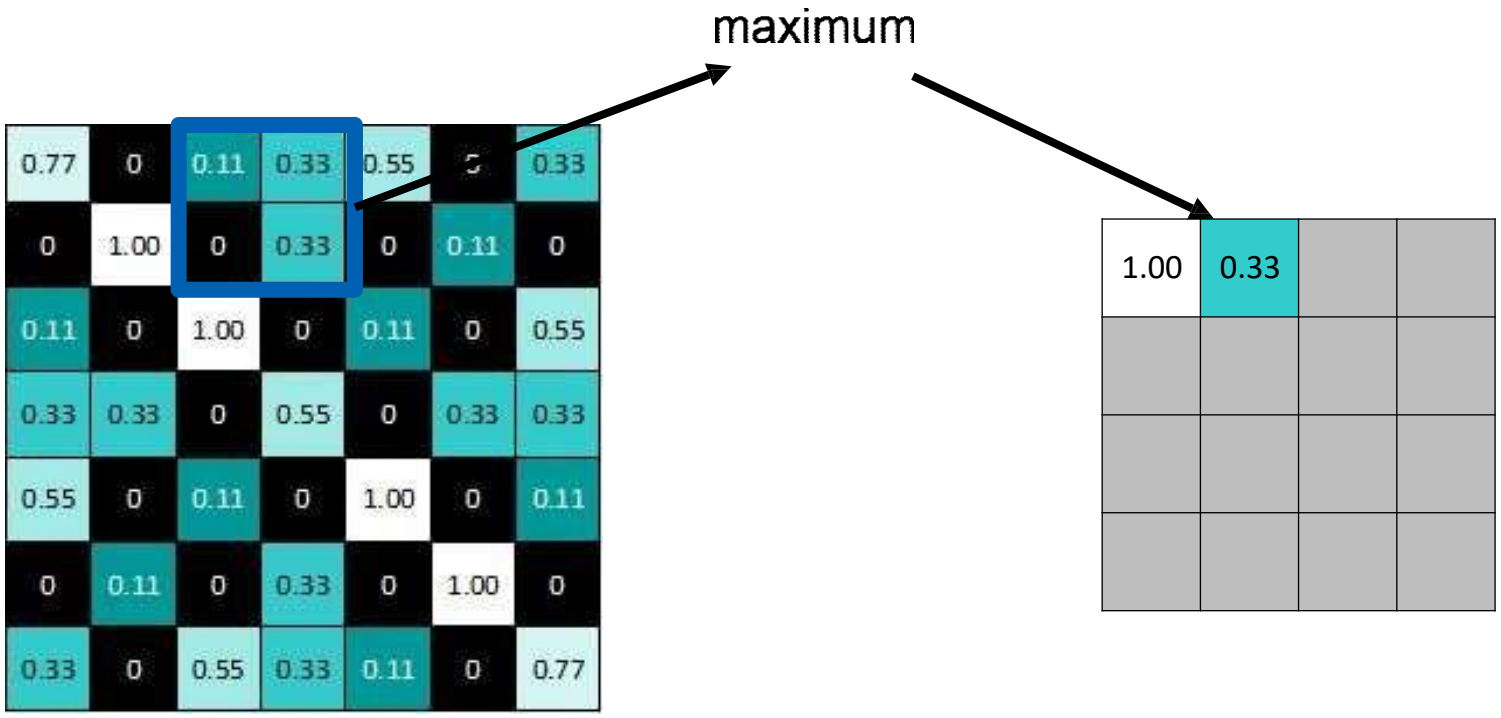
Pooling

- Pooling Filter Size = 2 X 2, Stride = 2



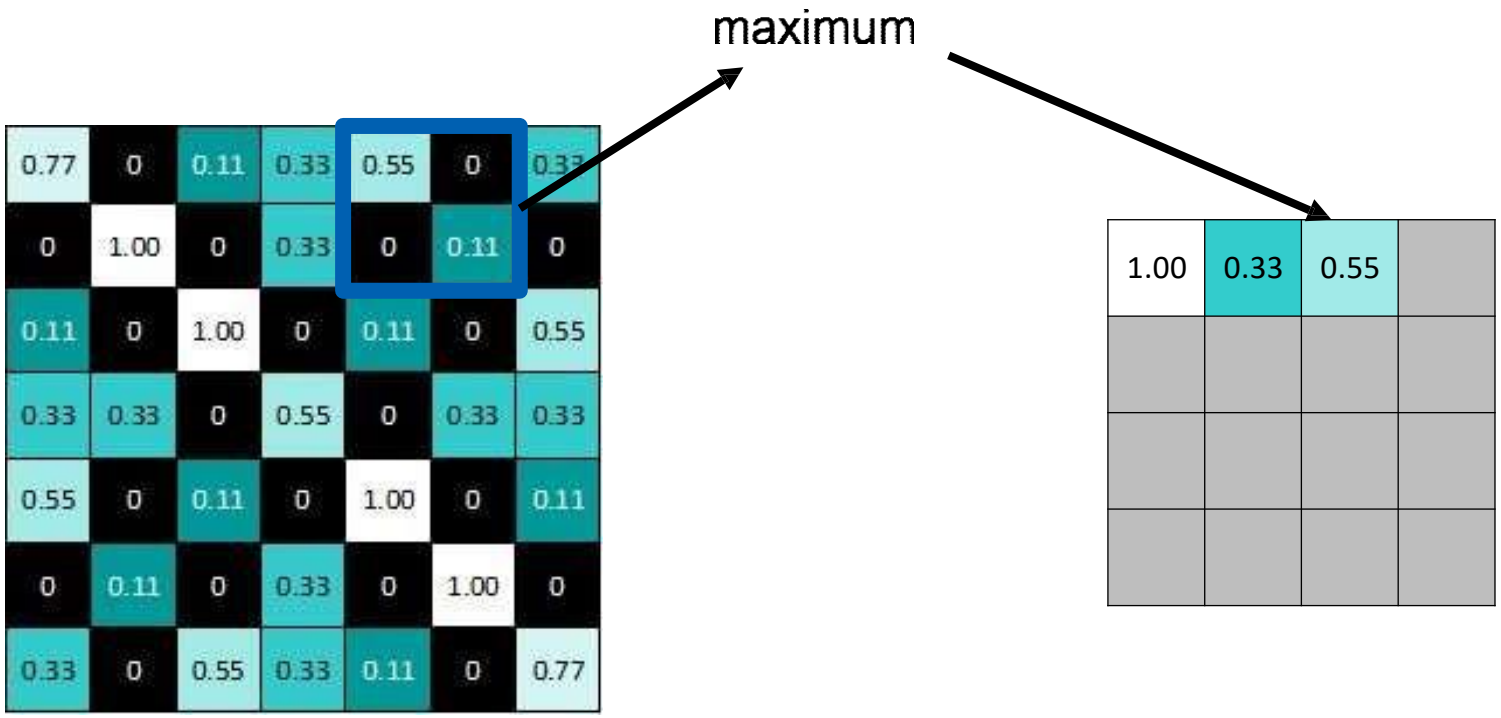
Pooling

- Pooling Filter Size = 2 X 2, Stride = 2



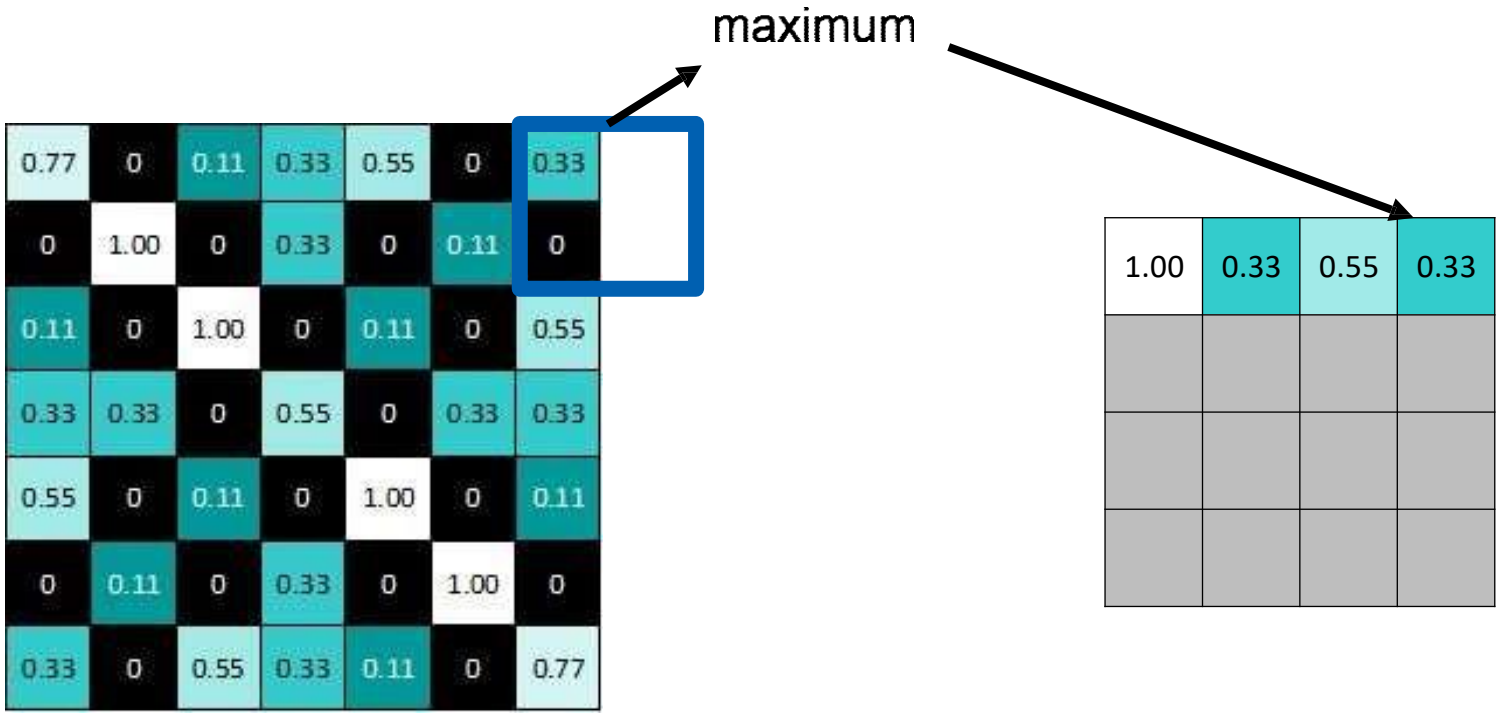
Pooling

- Pooling Filter Size = 2 X 2, Stride = 2



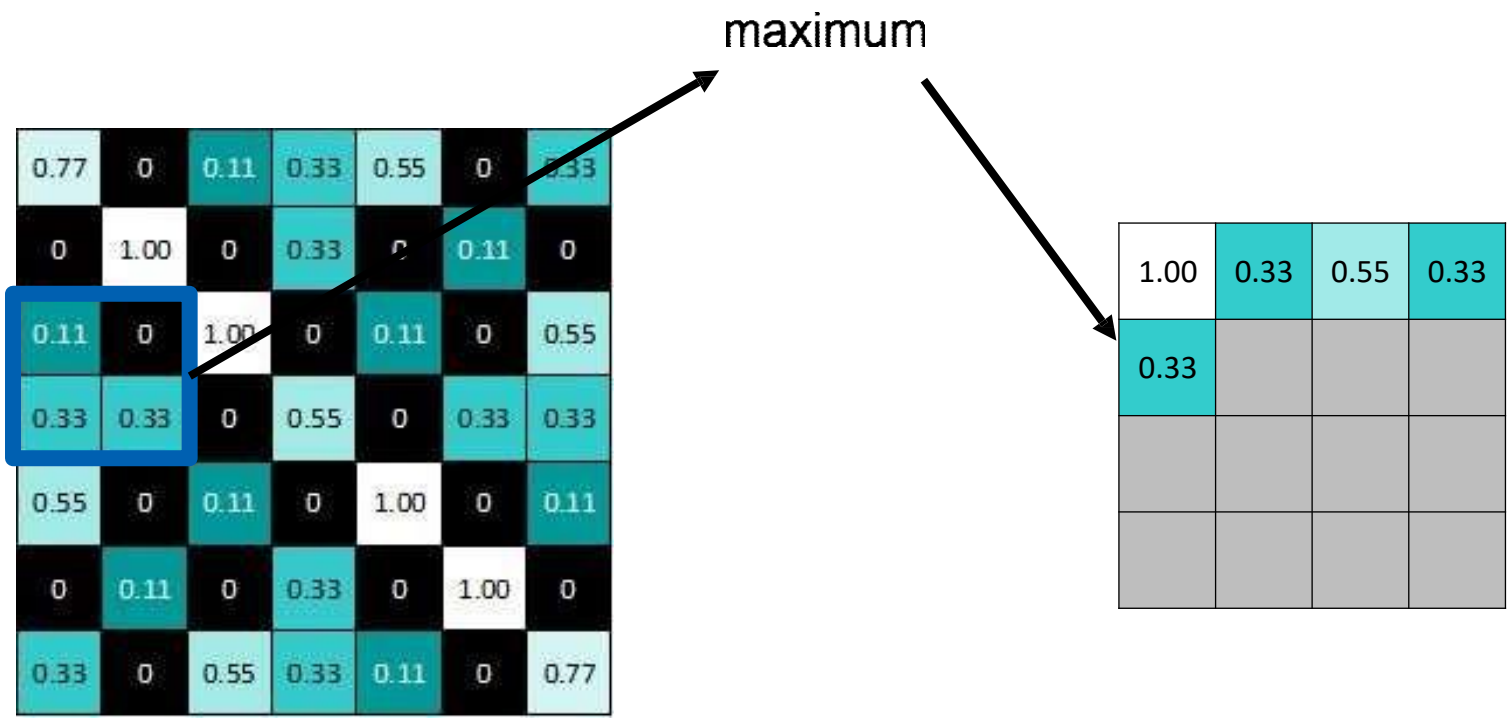
Pooling

- Pooling Filter Size = 2 X 2, Stride = 2



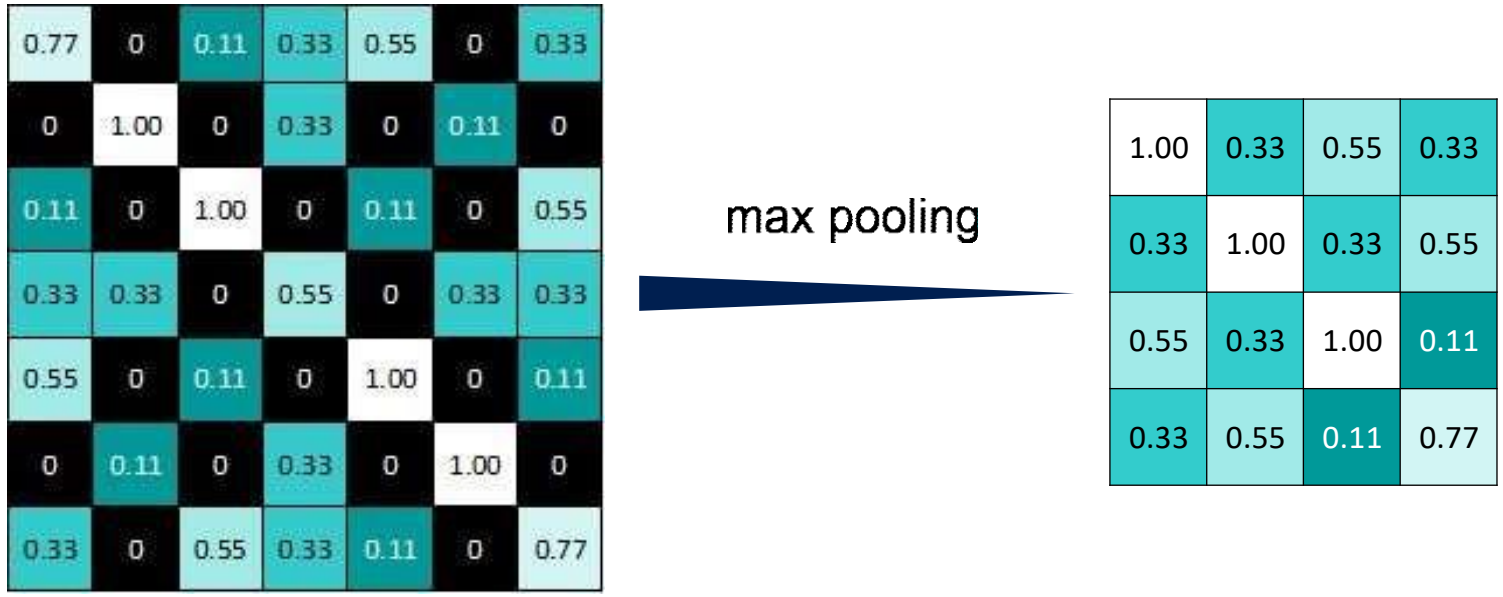
Pooling

- Pooling Filter Size = 2 X 2, Stride = 2

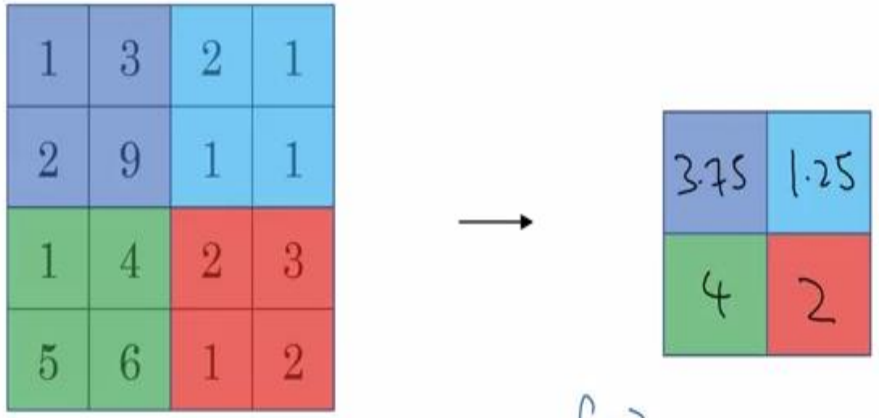


Pooling

- Pooling Filter Size = 2 X 2, Stride = 2



Pooling Layer : Average Pooling



$f=2$
$s=2$

Hyperparameters:

- f : filter size — $f=2, s=2$ / $f=3, s=2$
- s : stride
- Max or average pooling

$n_H \times n_w \times n_c$

$\downarrow$

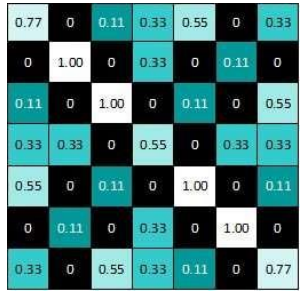
$\left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_w - f}{s} + 1 \right\rfloor$

$\times n_c$

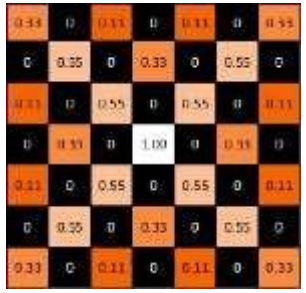
No parameters to learn!

Pooling

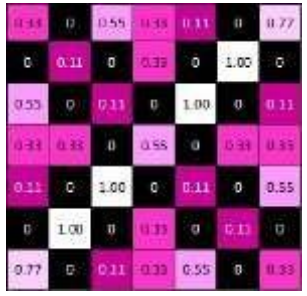
A stack of 3 larger images becomes a stack of smaller images.



1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77



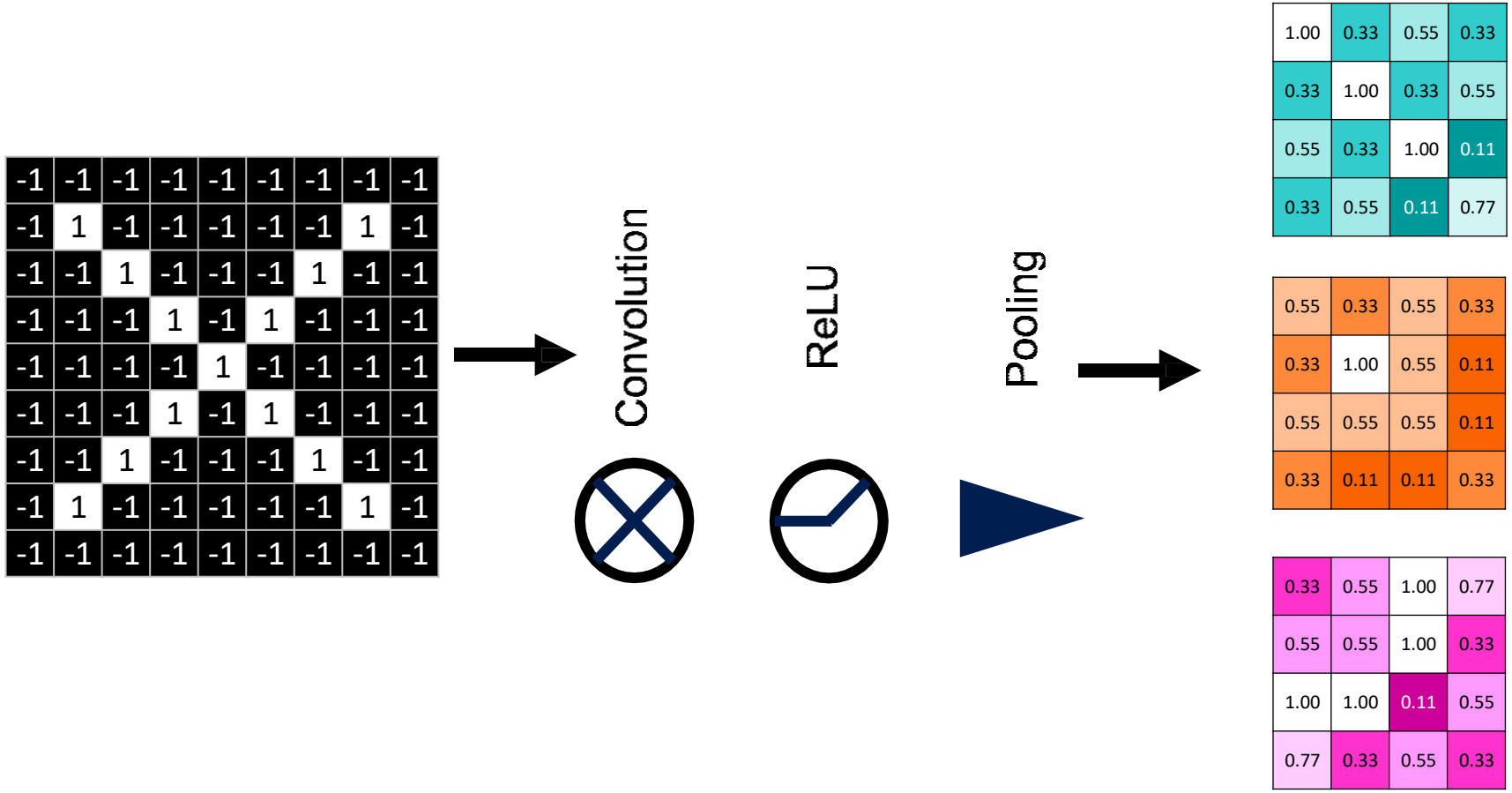
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33



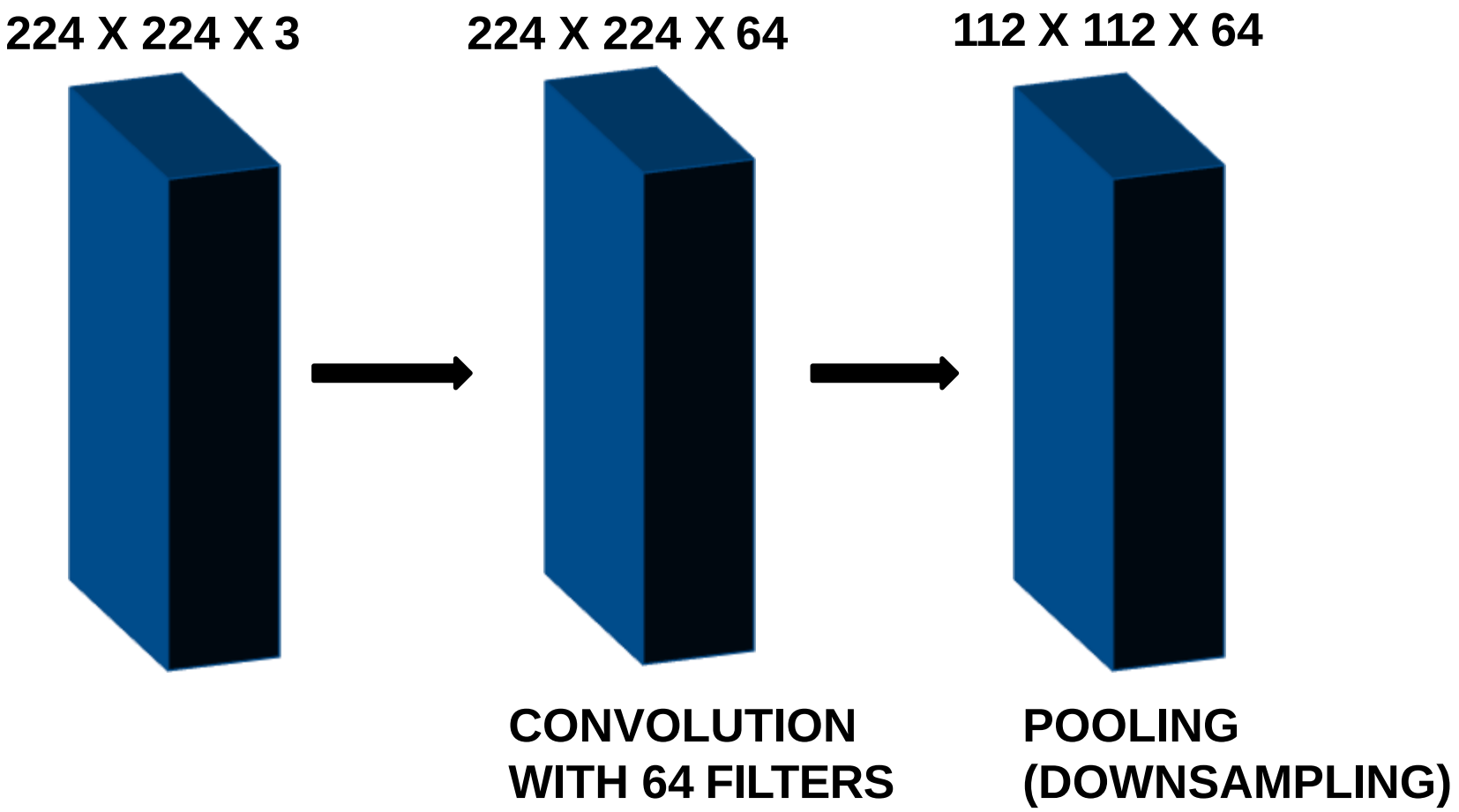
0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

Layers get stacked

The output of one becomes the input of the next.

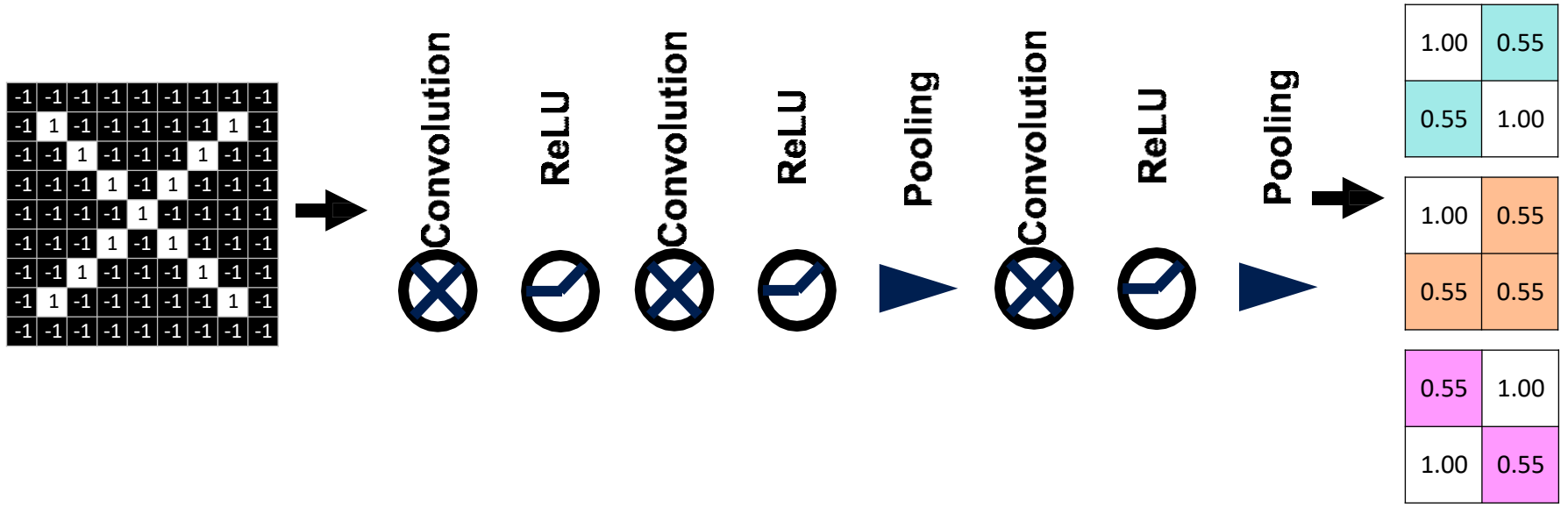


Layers Get Stacked - Example



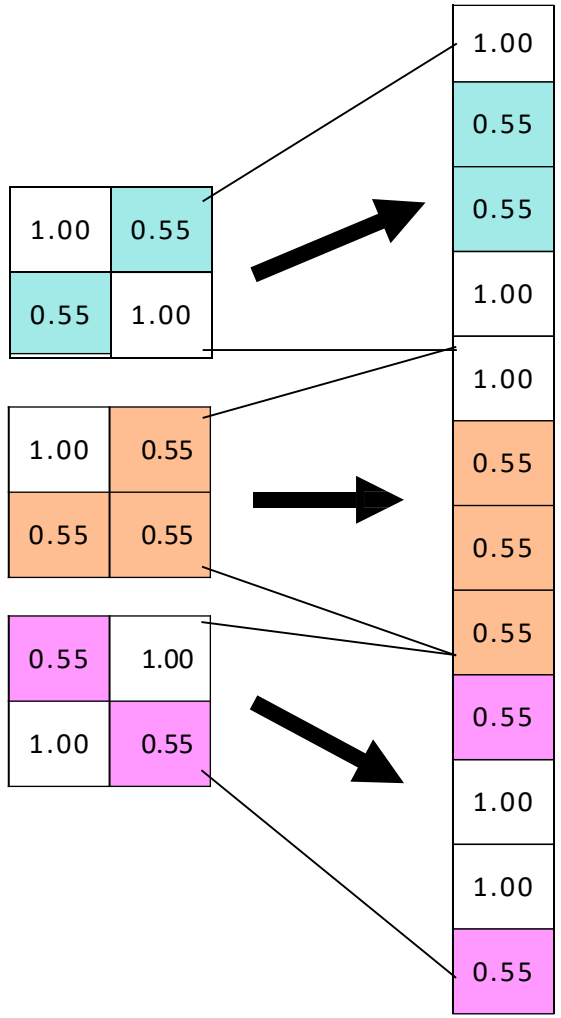
Deep stacking

Layers can be repeated several (or many) times.



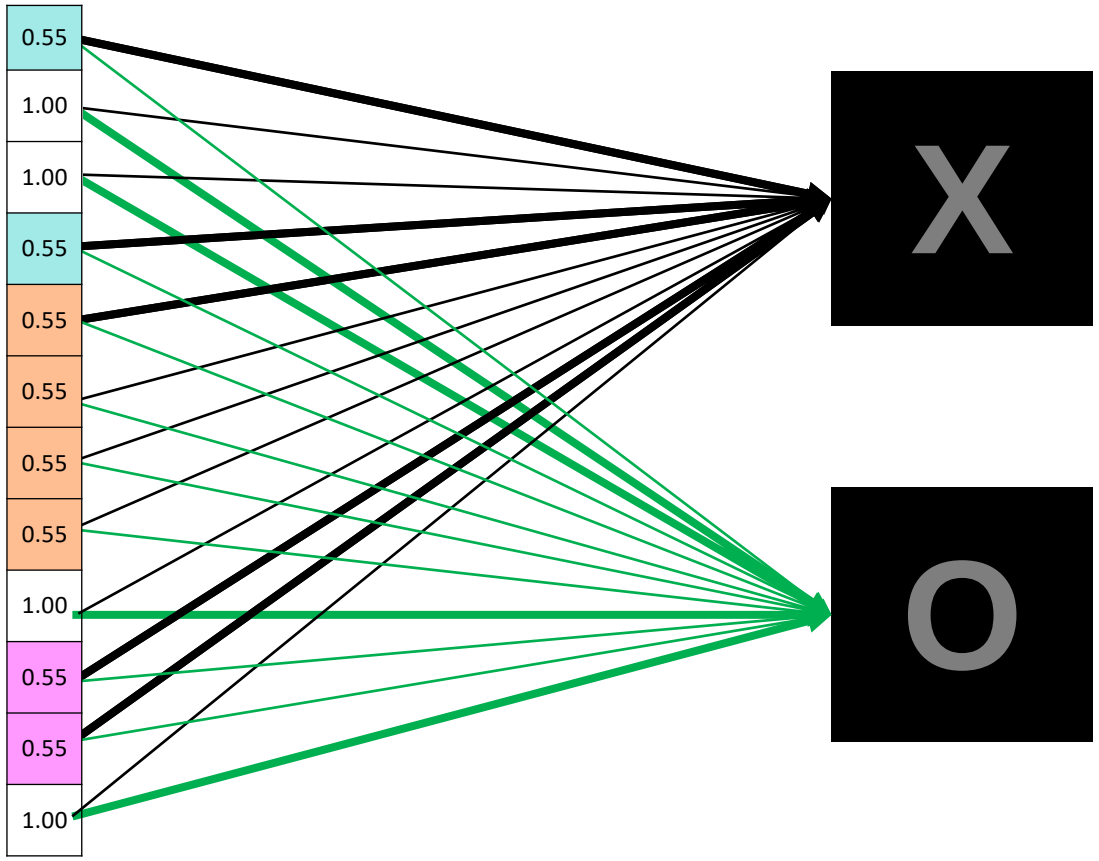
Fully connected layer

- Fully connected layers are the normal flat feed-forward neural network layers.
- These layers may have a non-linear activation function or a softmax activation in order to predict classes.
- To compute our output, we simply re-arrange the output matrices as a 1-D array.

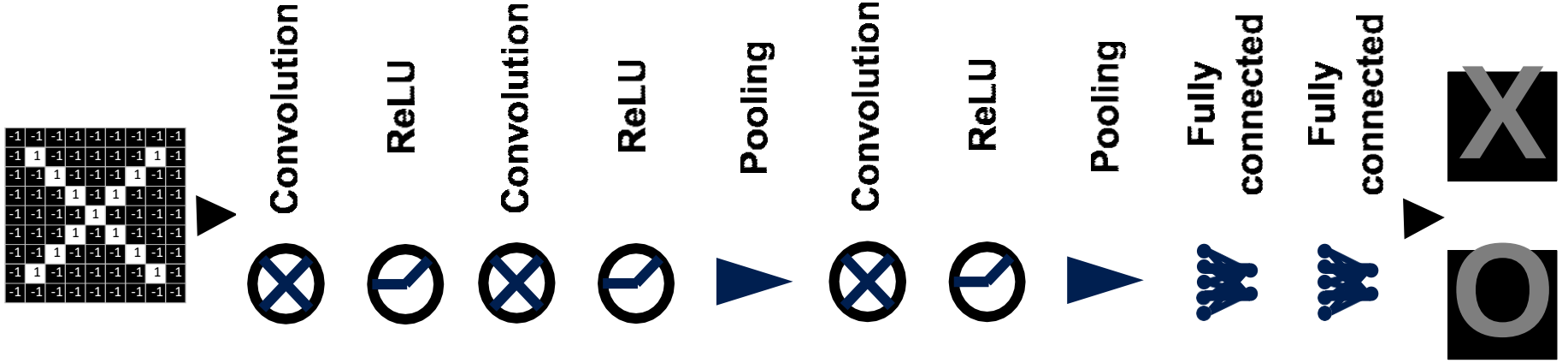


Fully connected layer

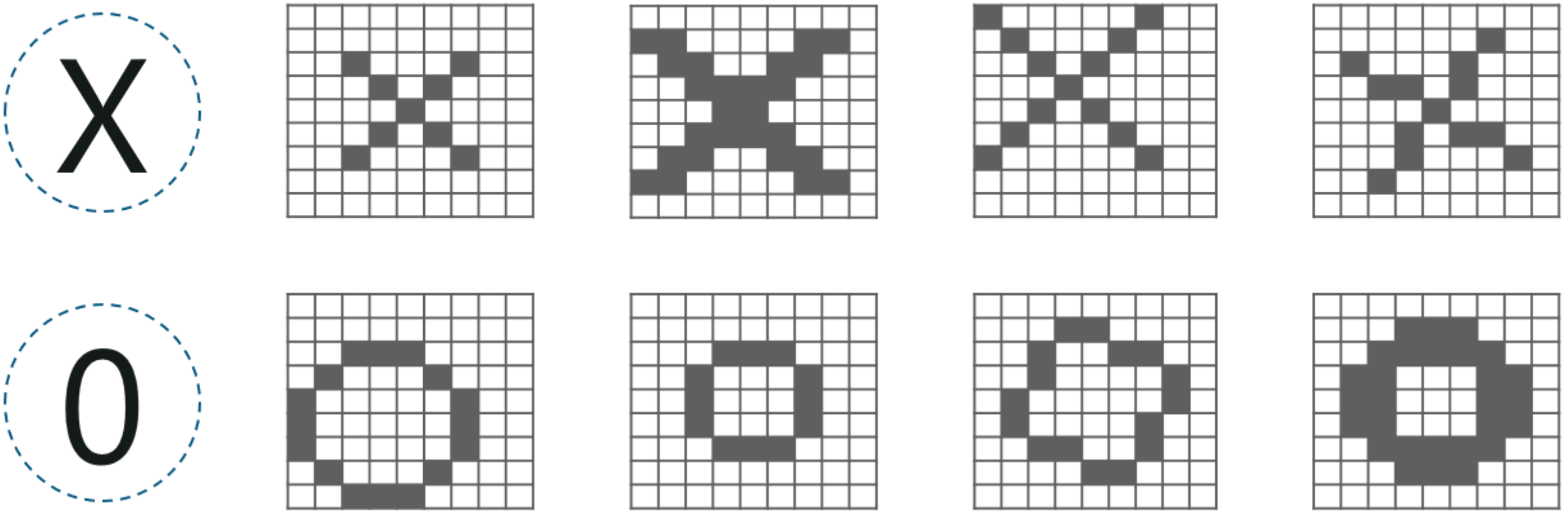
- A summation of product of inputs and weights at each output node determines the final prediction

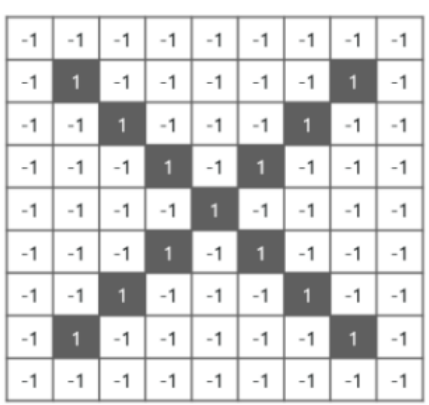


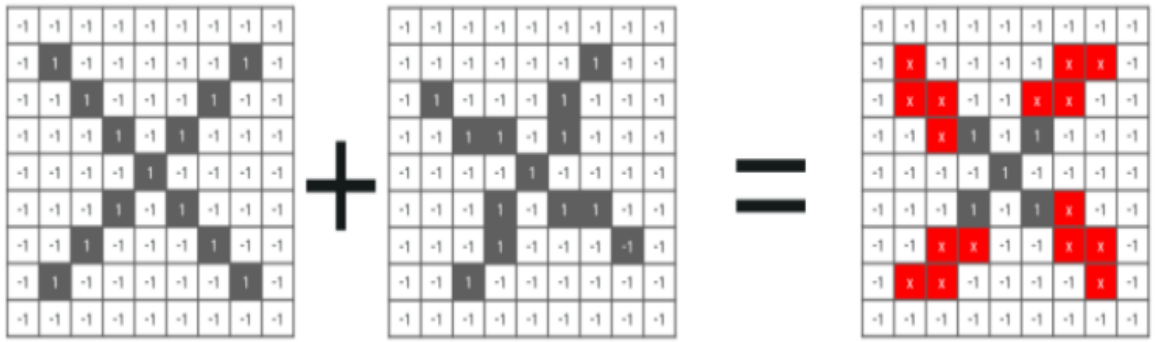
Putting it all together

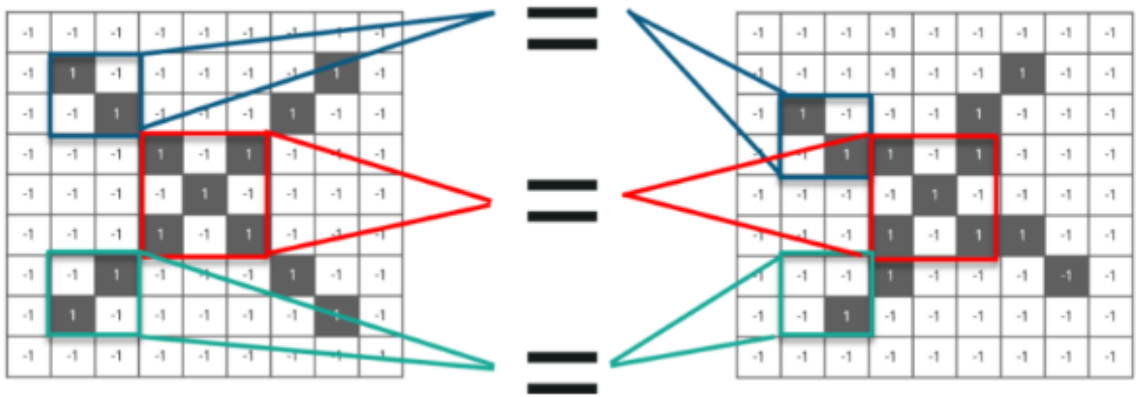


Example of CNN:







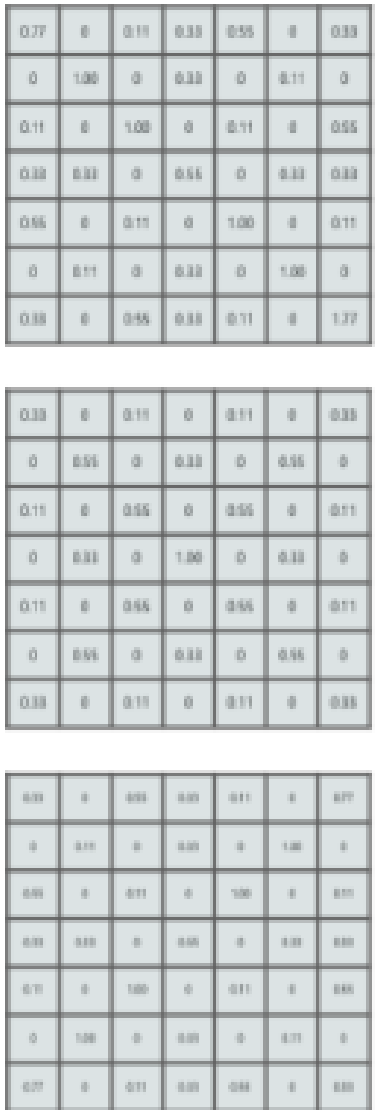


ReLu Function

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.0	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.0	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

Pooling Layer

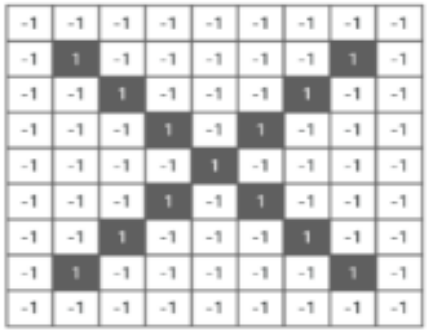


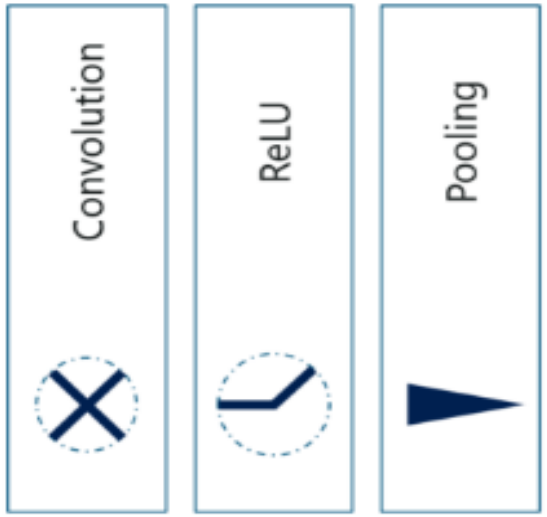
1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

Stacking Up The Layers

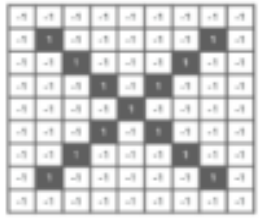




1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33



1	0.55
0.55	1.00

1	0.55
0.55	0.55

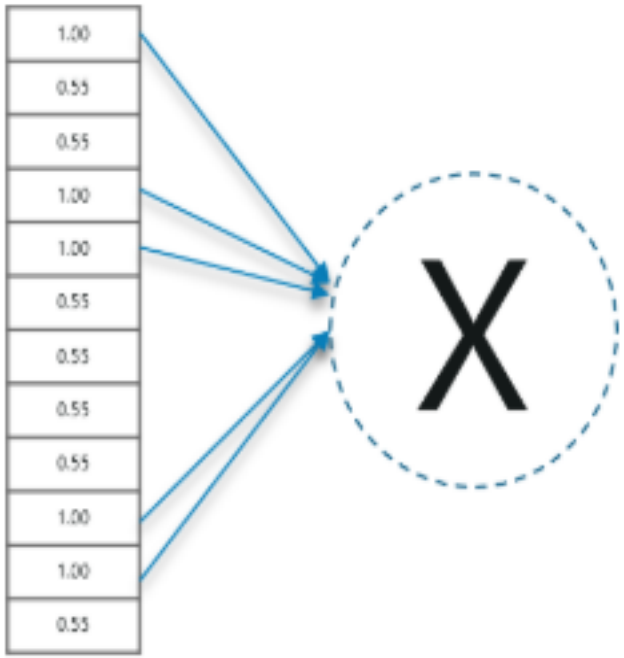
0.55	1.00
1.00	0.55

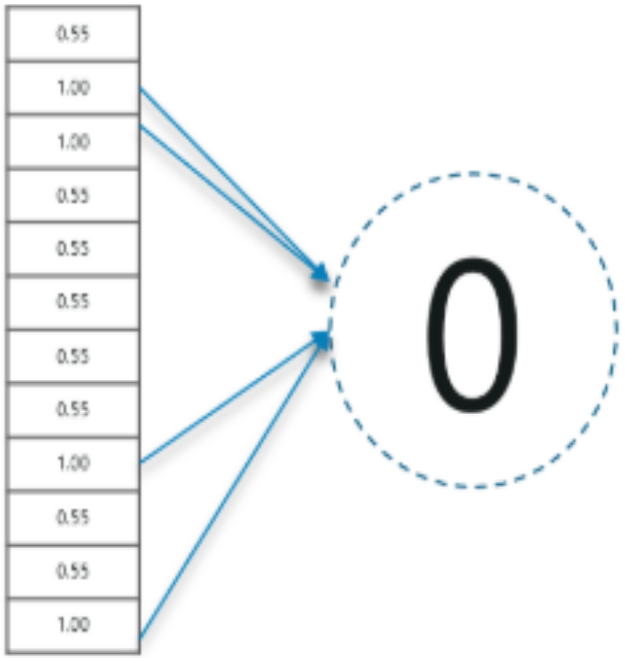
1	0.55
0.55	1.00

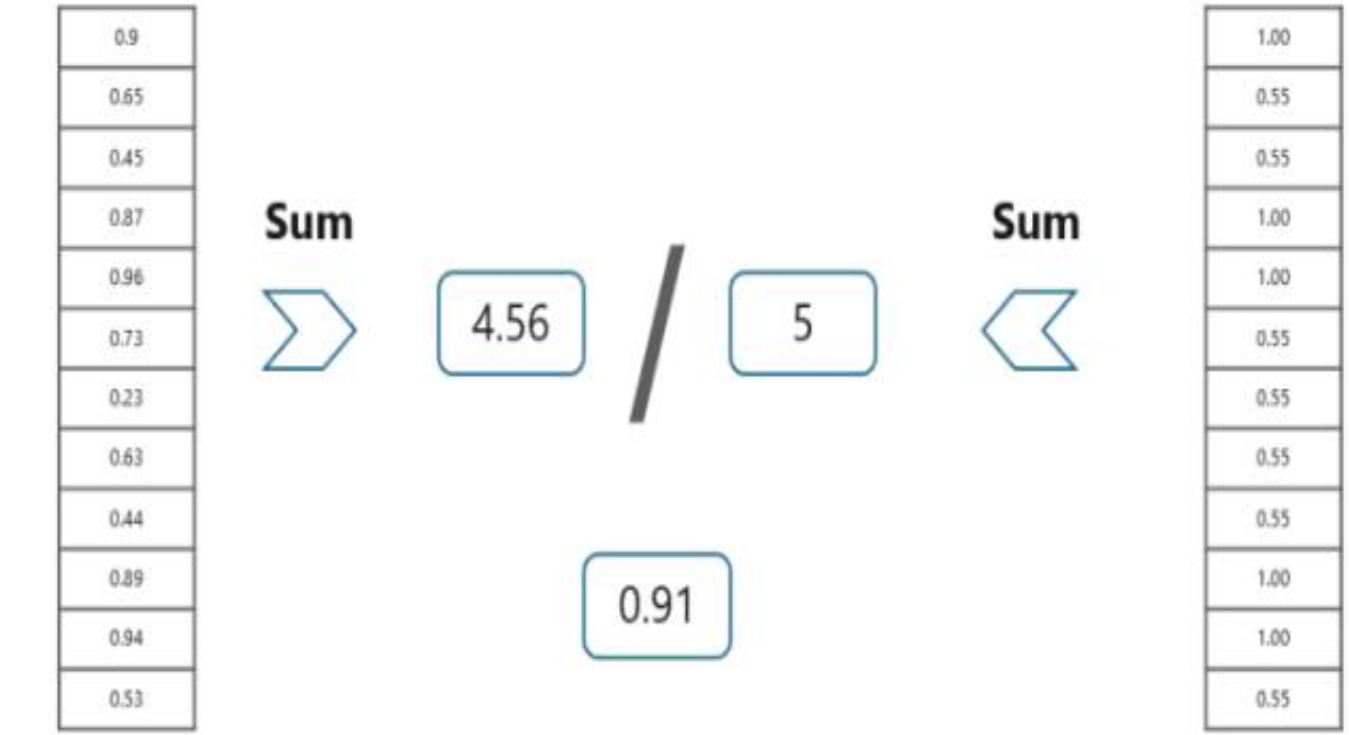
1	0.55
0.55	0.55

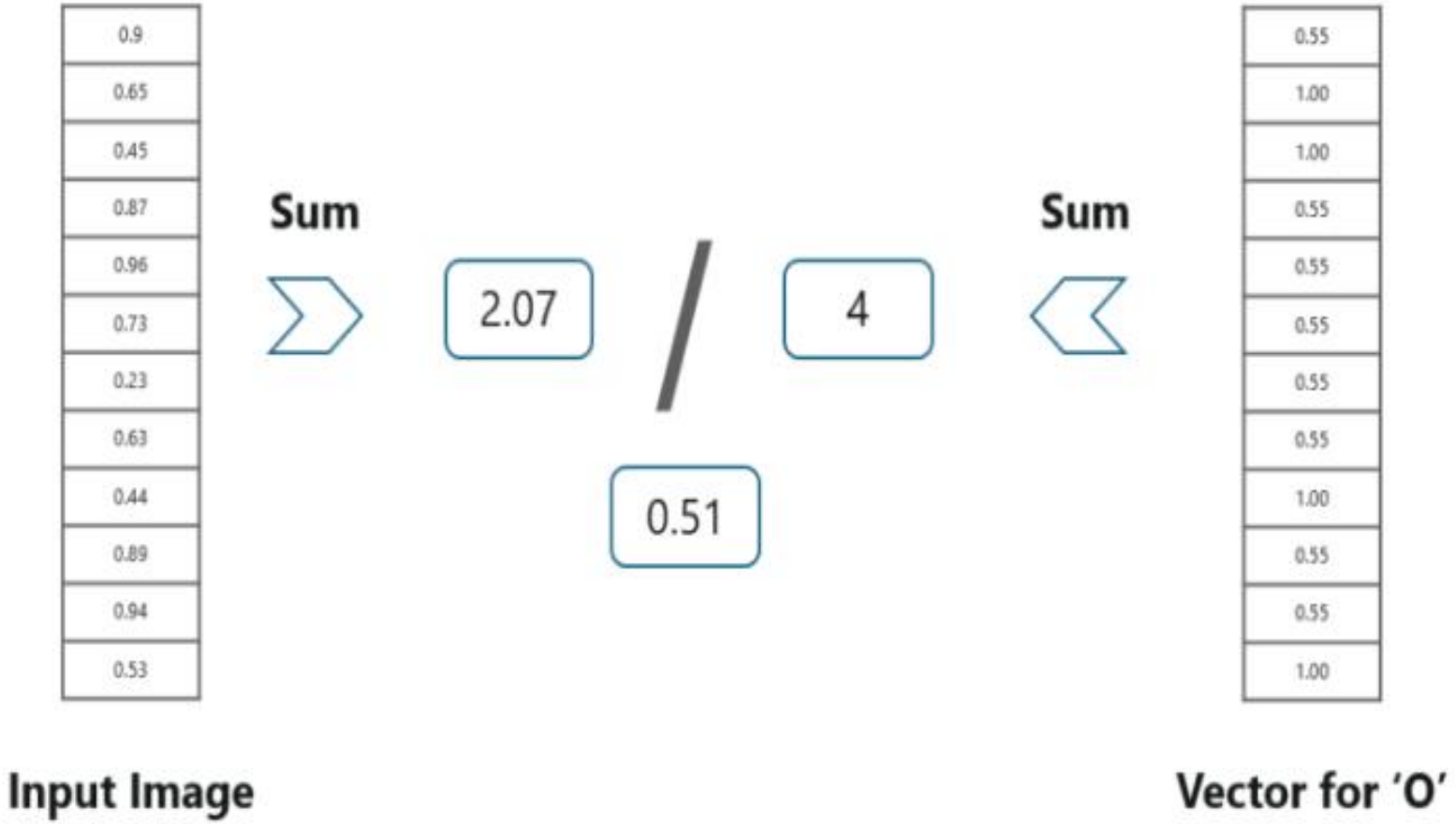
0.55	1.00
1.00	0.55

1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55

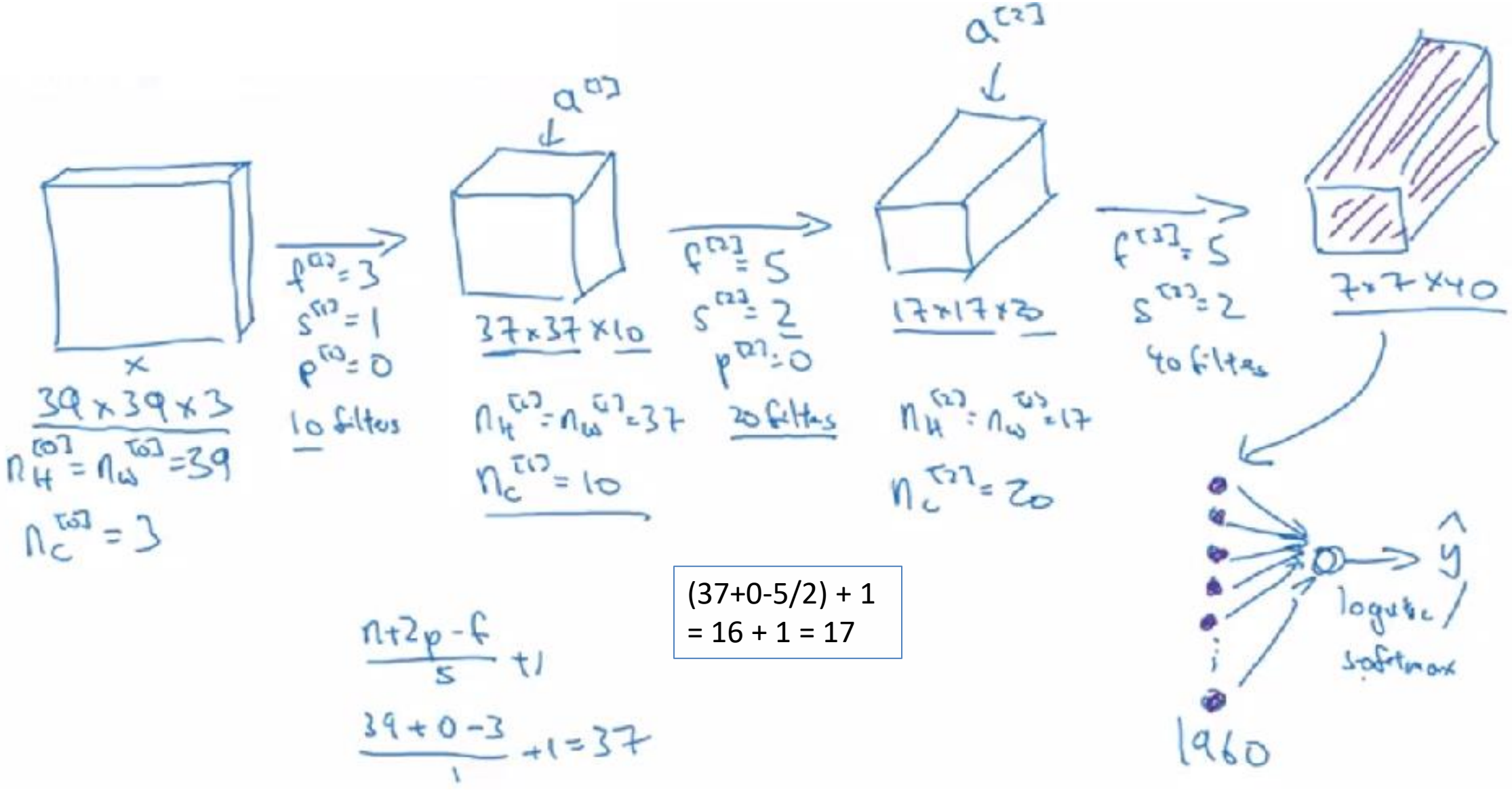





0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53
Sum
4.56
/
5
Sum
0.91
1.00
0.55
0.55
1.00
1.00
0.55
0.55
0.55
0.55
1.00
1.00
0.55
Input Image
Vector for 'X'


0.9
0.65
0.45
0.87
0.96
0.73
0.23
0.63
0.44
0.89
0.94
0.53
Sum
2.07
/
4
0.51
Sum
0.55
1.00
1.00
0.55
0.55
0.55
0.55
0.55
1.00
0.55
0.55
1.00
Input Image
Vector for 'O'

Example of ConvNet



$$\frac{n+2p-f}{s} + 1$$

$$\frac{39+0-3}{1} + 1 = 37$$

(37+0-5/2) + 1
= 16 + 1 = 17

Different CNN Architectures

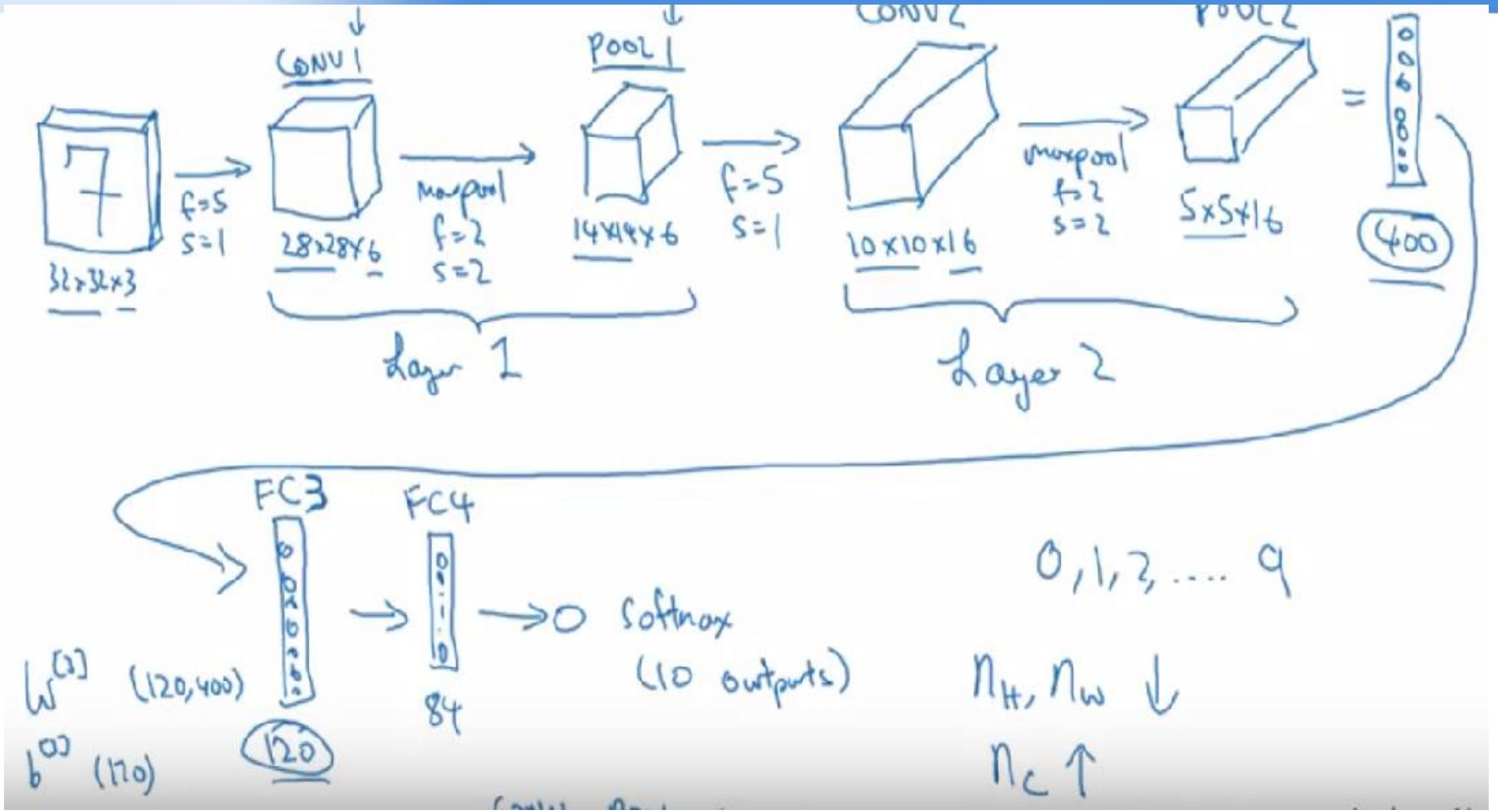
LeNet

Alexnet

VGG-19

ResNet

LeNet5


CONV 1
POOL 1
CONV 2
POOL 2
7
f=5
s=1
32×32×3
28×28×6
maxpool
f=2
s=2
14×14×6
f=5
s=1
10×10×16
maxpool
f=2
s=2
5×5×16
400
Layer 1
Layer 2
FC3
FC4
120
84
Softmax
(10 outputs)
0, 1, 2 9
$W^{[3]}$ (120,400)
$b^{[3]}$ (120)
$n_H, n_W \downarrow$
$n_C \uparrow$

Sample Exercises:

CNN architecture for image classification: The input images are RGB images with dimensions 128x128 pixels.

Design a CNN architecture with the following components:

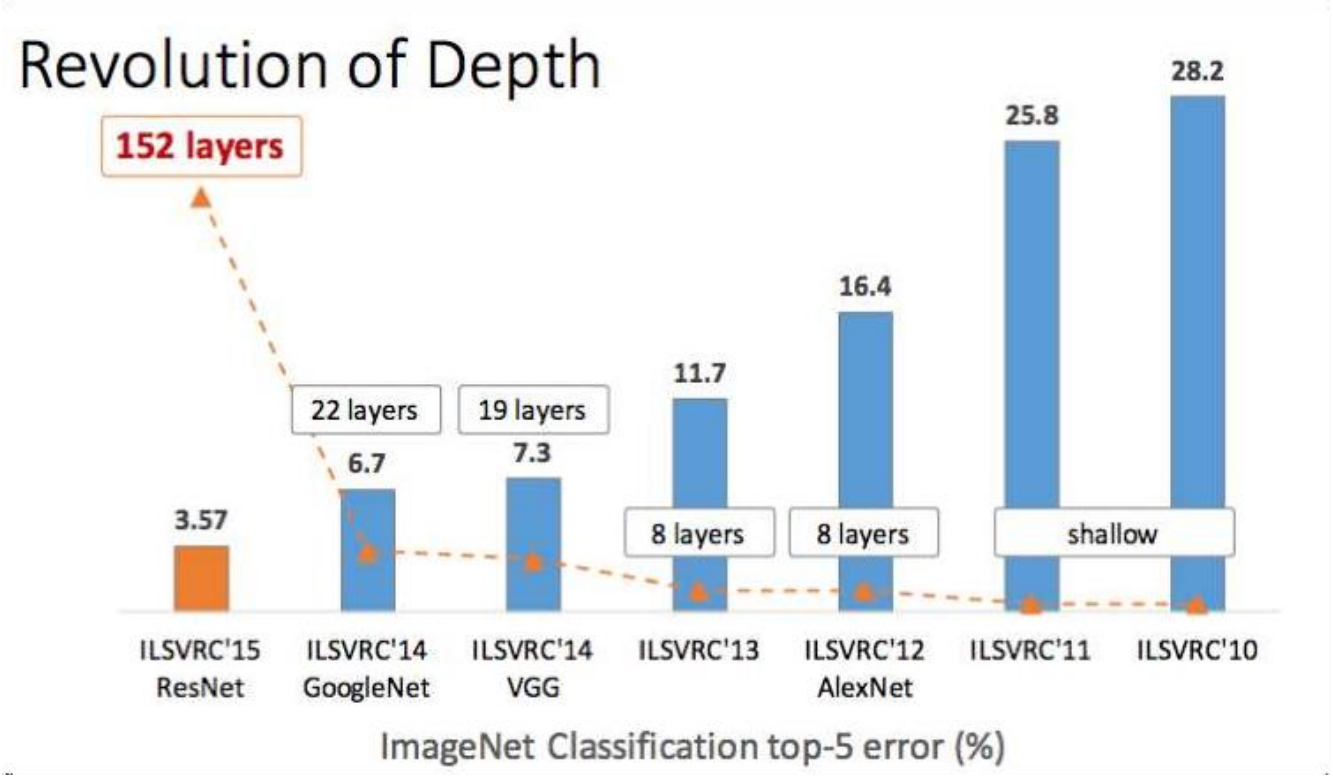
- Two convolutional layers with 3x3 filters, ReLU activation, and 32 filters each.
- Max pooling (2x2) after each convolutional layer.
- A fully connected layer with 128 neurons and ReLU activation.
- Output layer with 10 neurons and softmax activation for multiclass classification.
- Calculate the total number of parameters in the convolutional layers, the fully connected layer, and the entire network. Draw architecture and show your calculations step by step.

Hyperparameters (knobs)

- Convolution
  - Filter Size
  - Number of Filters
  - Padding
  - Stride
- Pooling
  - Window Size
  - Stride
- Fully Connected
  - Number of neurons

Case Studies

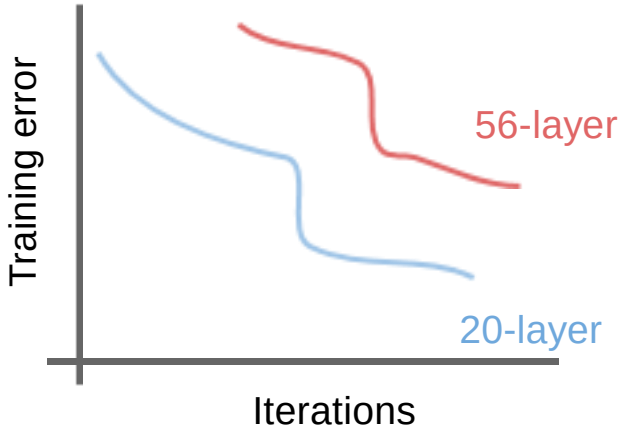
- LeNet – 1998
- AlexNet – 2012
- ZFNet – 2013
- VGG – 2014
- GoogLeNet – 2014
- ResNet – 2015

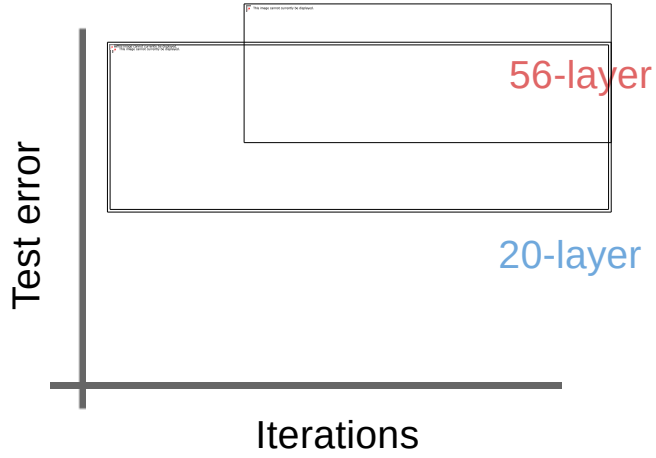


The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Deep vs Shallow Networks

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?





56-layer model performs worse on both training and test error

-> The deeper model performs worse, but it’s not caused by overfitting!

Deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

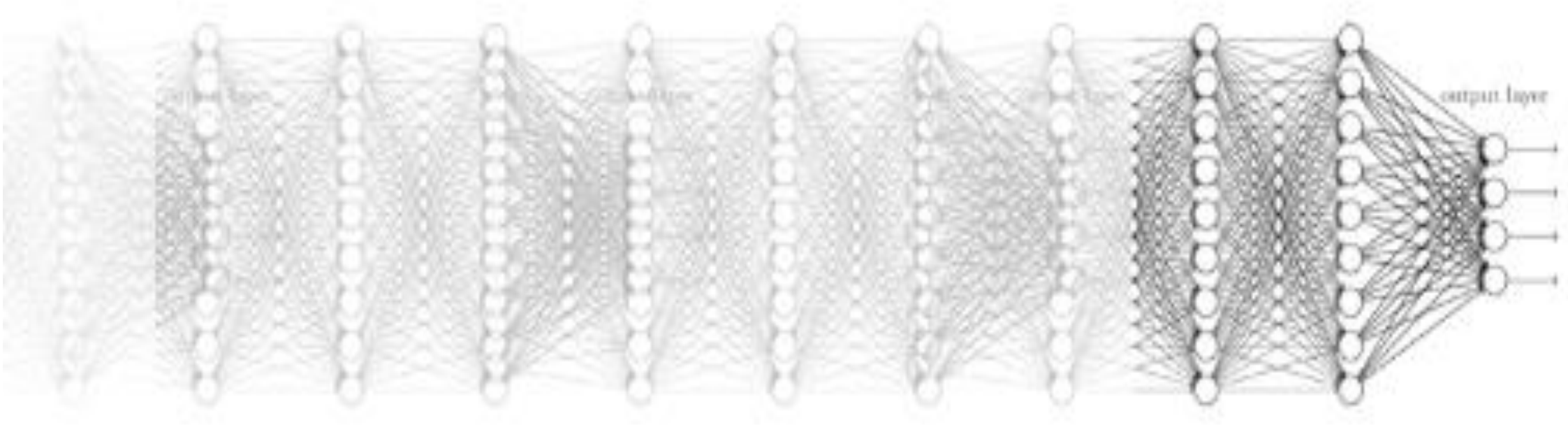
A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Challenges

- Deeper Neural Networks start to degrade in performance.
- Vanish/Exploding Gradient – May lead for extremely complex parameters initializations to make it work. Still might suffer from Vanish/Exploding even for the best parameters.
- Long training times – Due to too many training parameters.

Partial Solutions for Vanish/Exploding gradients

- Batch Normalization – To rescale the weights over some batch.
- Smart Initialization of weights – Like for example Xavier initialization.
- Train portions of the network individually.



Related Prior Work - Highway networks

- Adding features from previous time steps has been used in various tasks
- Most notable of these are Highway networks proposed by Srivastava et al.
- Highway networks feature residual connections Residual networks have the form

$$Y = f(x) + x$$

- Highway networks have the form

$$Y = f(x)\,\mathrm{sigmoid}(Wx + b) + x(1 - \mathrm{sigmoid}(Wx + b))$$

Highway networks – cont.

Highway networks enabled information flow from the past but due to the gating function

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

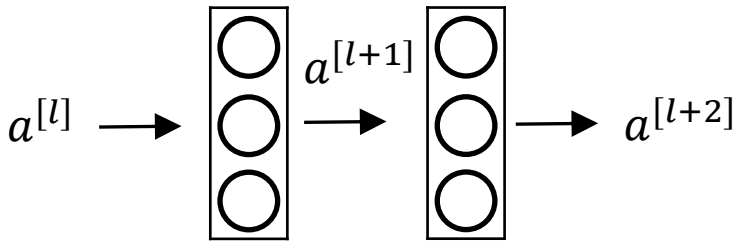
$$T(\mathbf{x}) = \sigma(\mathbf{W_T}^T \mathbf{x} + \mathbf{b_T}).$$

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$

ResNet

- A specialized network introduced by Microsoft.
- Connects inputs of layers into farther part of that network to allow “shortcuts”.
- Simple idea – great improvements with both performance and train time.

Plain Network



$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$

"linear"

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

"output"

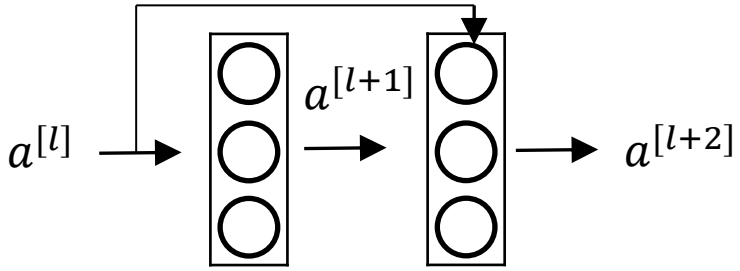
$$a^{[l+1]} = g(z^{[l+1]})$$

"relu"

$$a^{[l+2]} = g\left(z^{[l+2]}\right)$$

"relu on output"

Residual Blocks



$$a^{[l+1]} = g(z^{[l+1]})$$

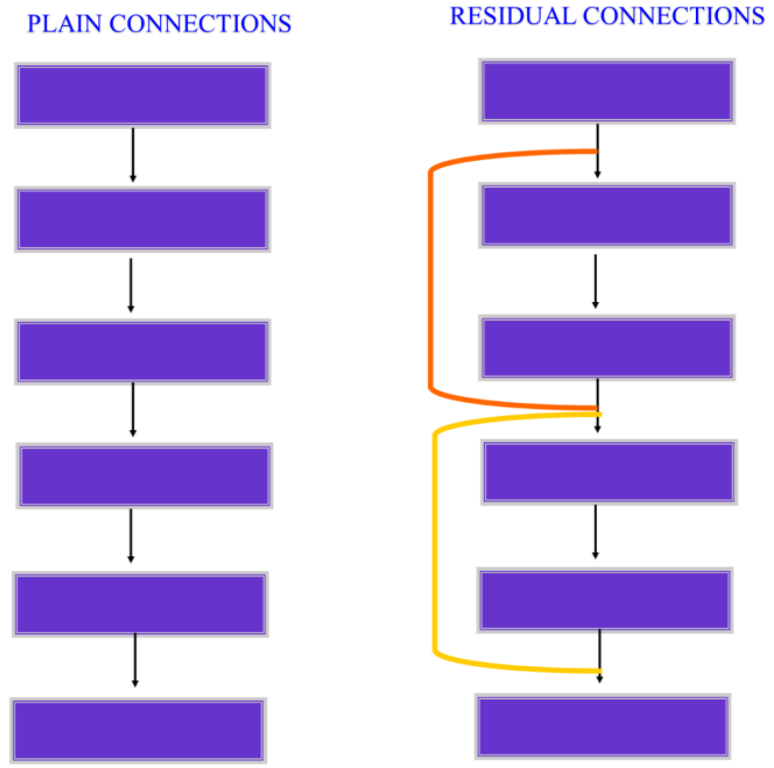
"relu"

$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$

"linear"

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

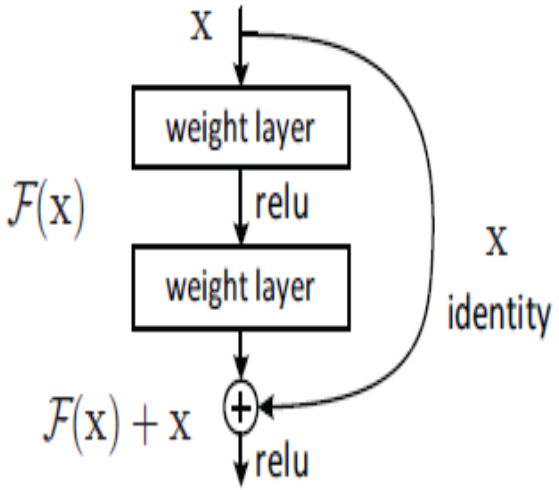
"output"



$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

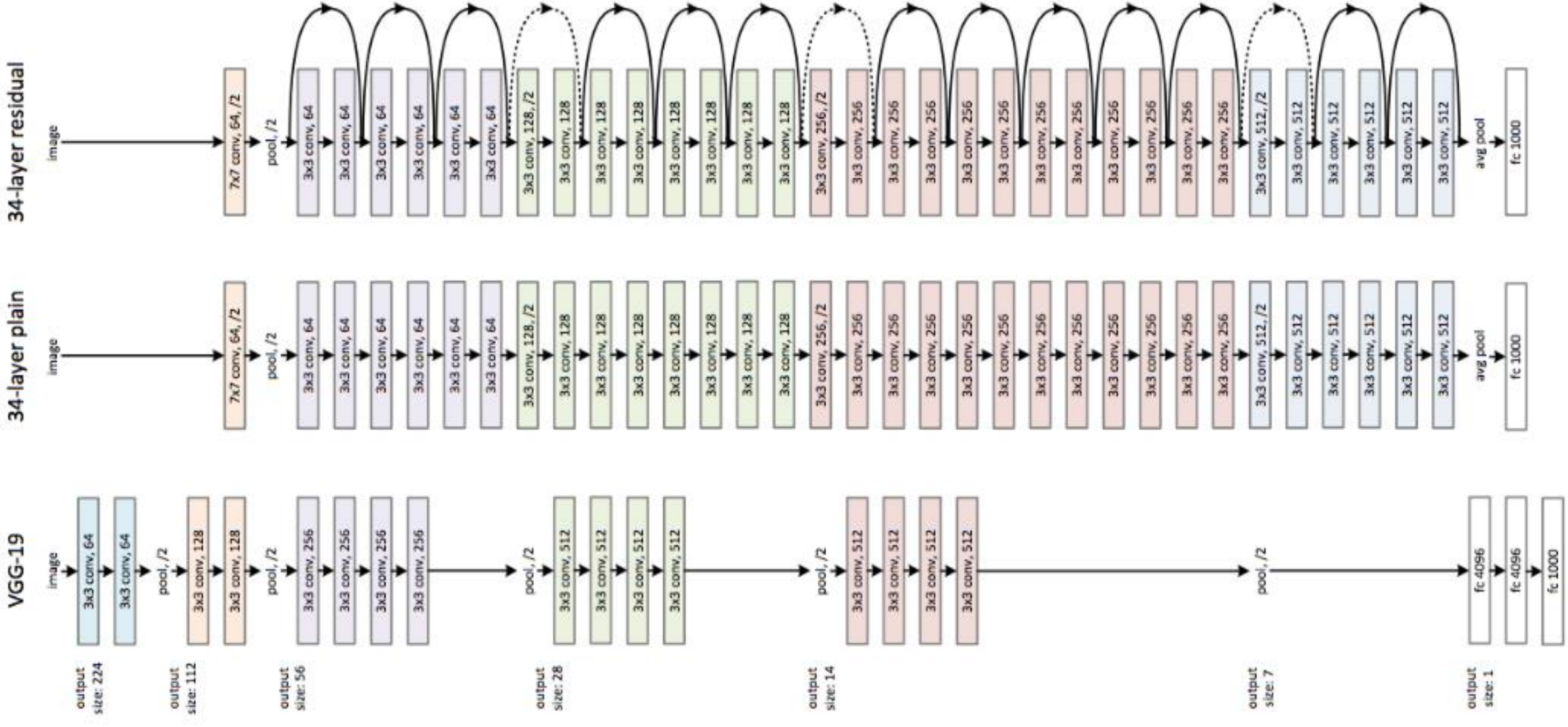
"relu on output **plus input**"

Skip Connections “shortcuts”



- Such connections are referred as skipped connections or shortcuts. In general similar models could skip over several layers.
- They refer to residual part of the network as a unit with input and output.
- Such residual part receives the input as an amplifier to its output – The dimensions usually are the same.
- Another option is to use a projection to the output space.
- Either way – no additional training parameters are used.

Residual Blocks (skip connections)



Deeper Bottleneck Architecture (Cont.)

- Addresses high training time of very deep networks.
- Keeps the time complexity same as the two layered convolution
- Allows us to increase the number of layers
- Allows the model to converge much faster.
- 152-layer ResNet has 11.3 billion FLOPS while VGG-16/19 nets has 15.3/19.6 billion FLOPS.

Why Do ResNets Work Well?

- Having a “regular” network that is very deep might actually hurt performance because of the vanishing and exploding gradients
- In most cases, ResNets will simply stop improving rather then decrease in performance
- $a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right) = g(w^{[l+1]}a^{[l+1]} + b^{[l]} + a^{[l]})$
- If the layer is not “useful”, L2 regularization will bring it’s parameters very close to zero, resulting in $a^{[l+2]} = g\left(a^{[l]}\right) = a^{[l]}$ (when using ReLU)

Why Do ResNets Work Well? (Cont)

- In theory ResNet is still identical to plain networks, but in practice due to the above the convergence is much faster.
- No additional training parameters introduced.
- No addition complexity introduced.

Training ResNet in practice

- Batch Normalization after every CONV layer.
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus.
- Mini-batch size 256.
- Weight decay of 1e-5.
- No dropout used.

Loss Function

- For measuring the loss of the model a combination of cross-entropy and softmax were used.
- The output of the cross-entropy was normalized using softmax function.

$$H(p, q) = -\sum_{x} p(x) \log q(x).$$

$$\sigma : \mathbb{R}^K \to \left\{ z \in \mathbb{R}^K \middle| z_i > 0, \sum_{i=1}^{K} z_i = 1 \right\}$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Reduce Learning Time with Random Layer Drops

- Dropping layers during training, and using the full network in testing.
- Residual block are used as network's building block.
- During training, input flows through both the shortcut and the weights.
- Training: Each layer has a "survival probability" and is randomly dropped.
- Testing: all blocks are kept active.
- Re-calibrated according to its survival probability during training.

Thank you!!