Convolutional Neural Networks

Peerapon S.

Machine Learning

Data and codes : https://kmutt.me/CPE-ML

Topics

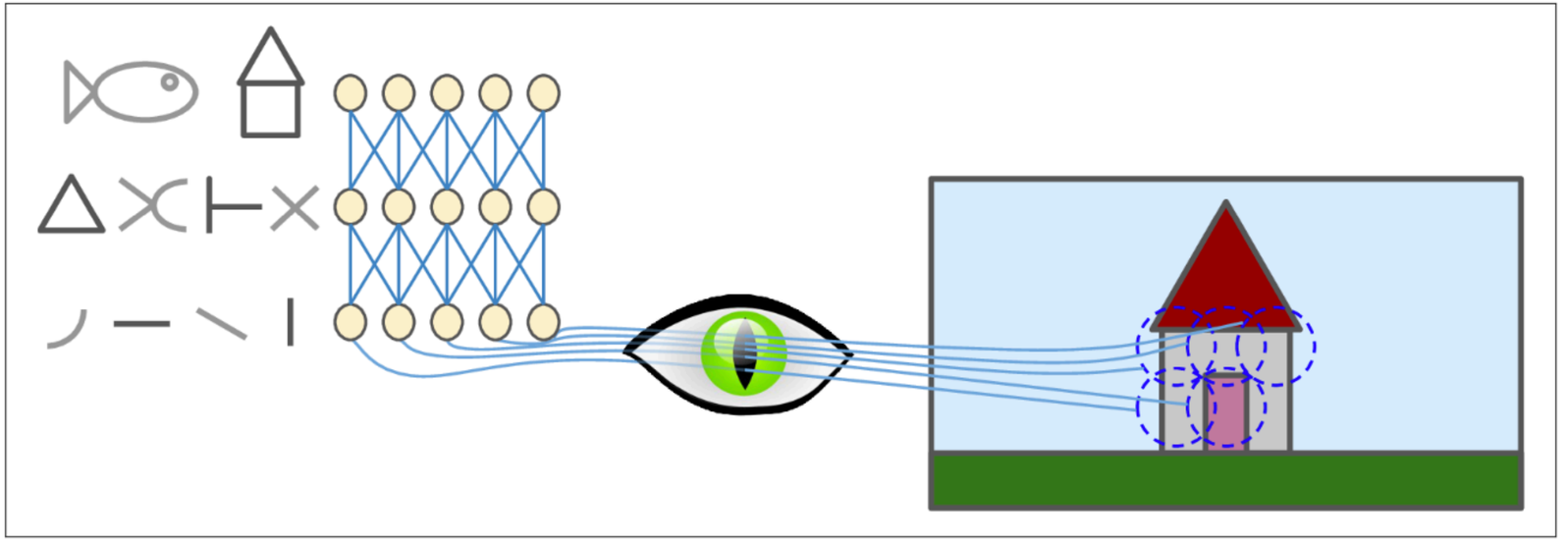
Basic concepts of CNN

CNN layers

Well-known CNN architectures

Transfer learning

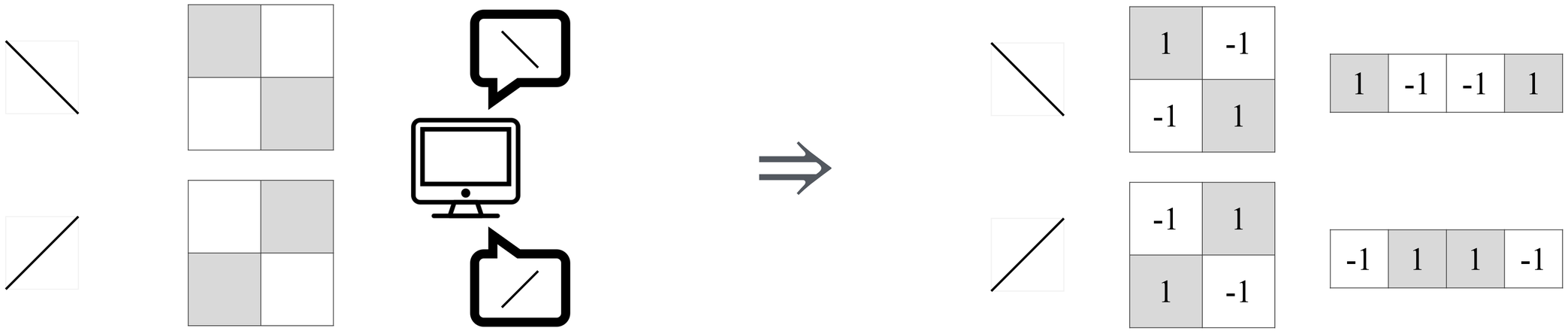
Motivation



Local receptive field of visual cortex

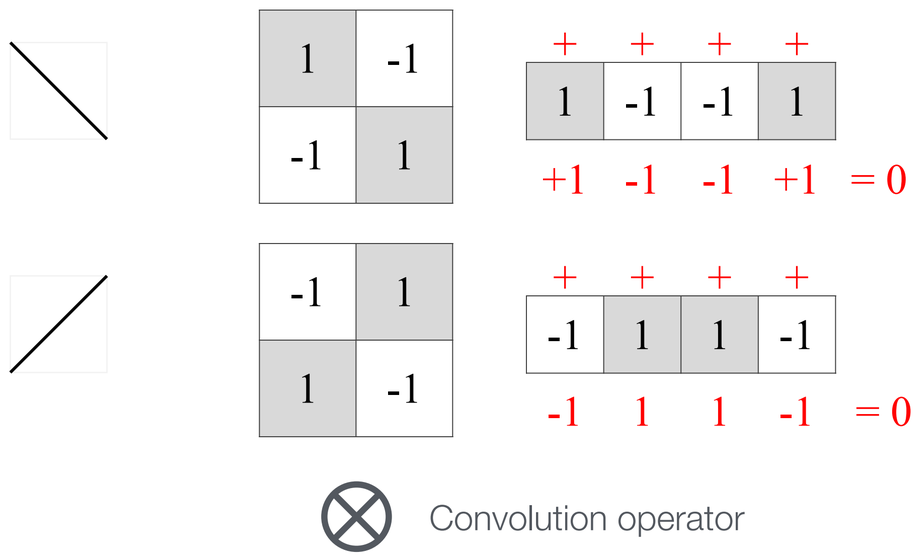
Image Representation

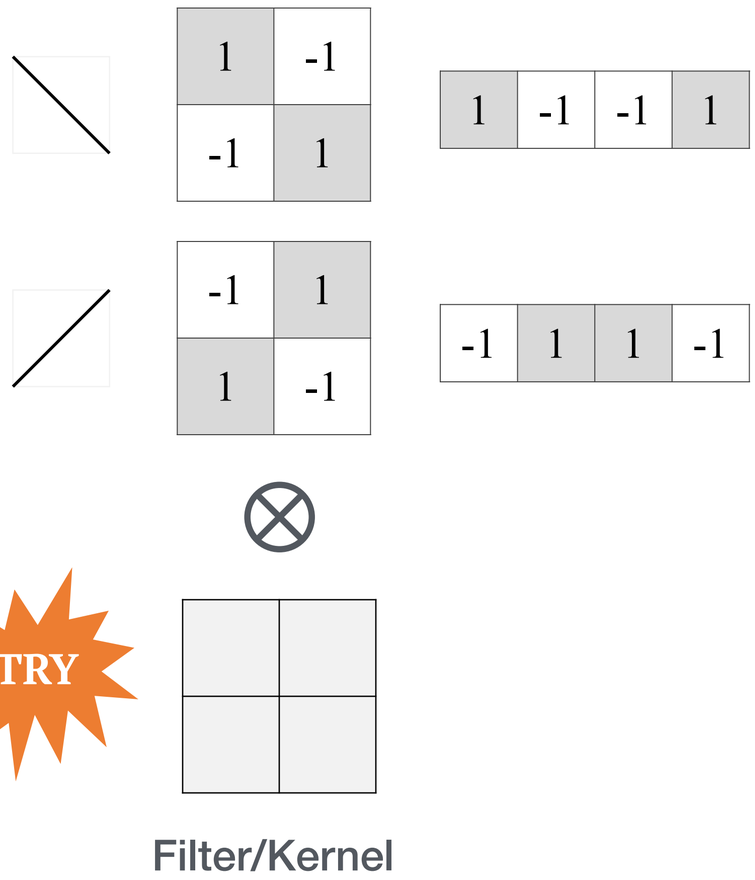
Simple 2x2 Images



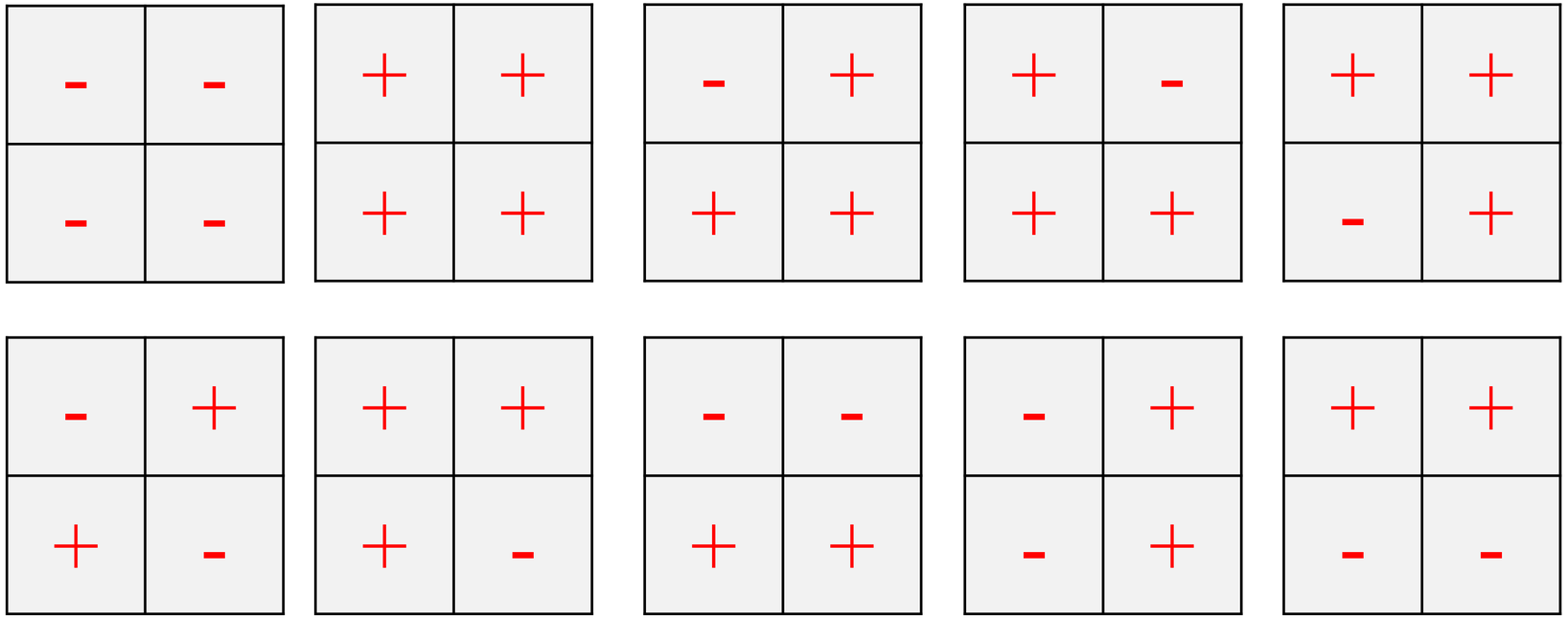
Simple Image Recognition with Filter/Kernel

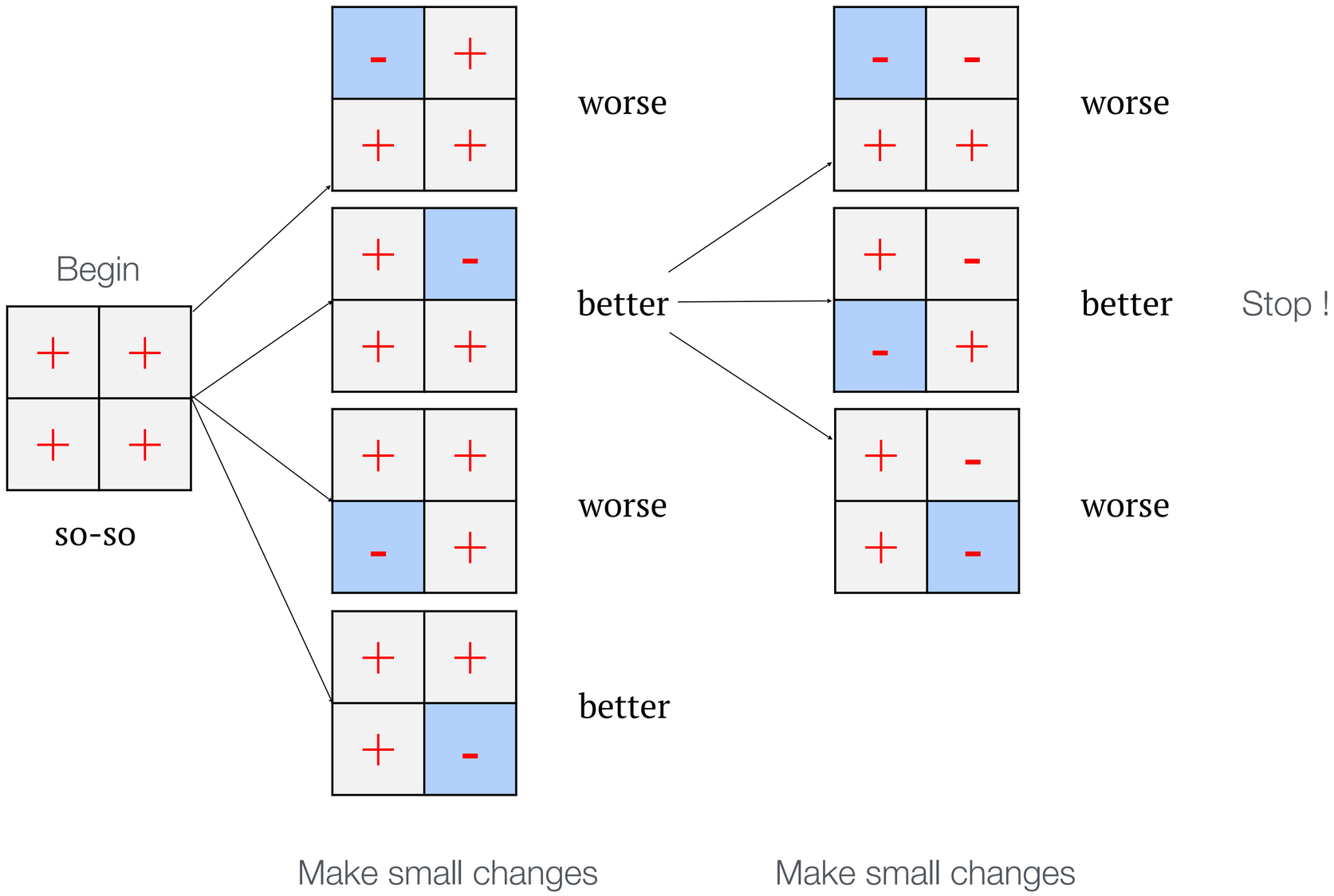
What math operation (***filter/kernel***) can be used to distinguish the two images ?

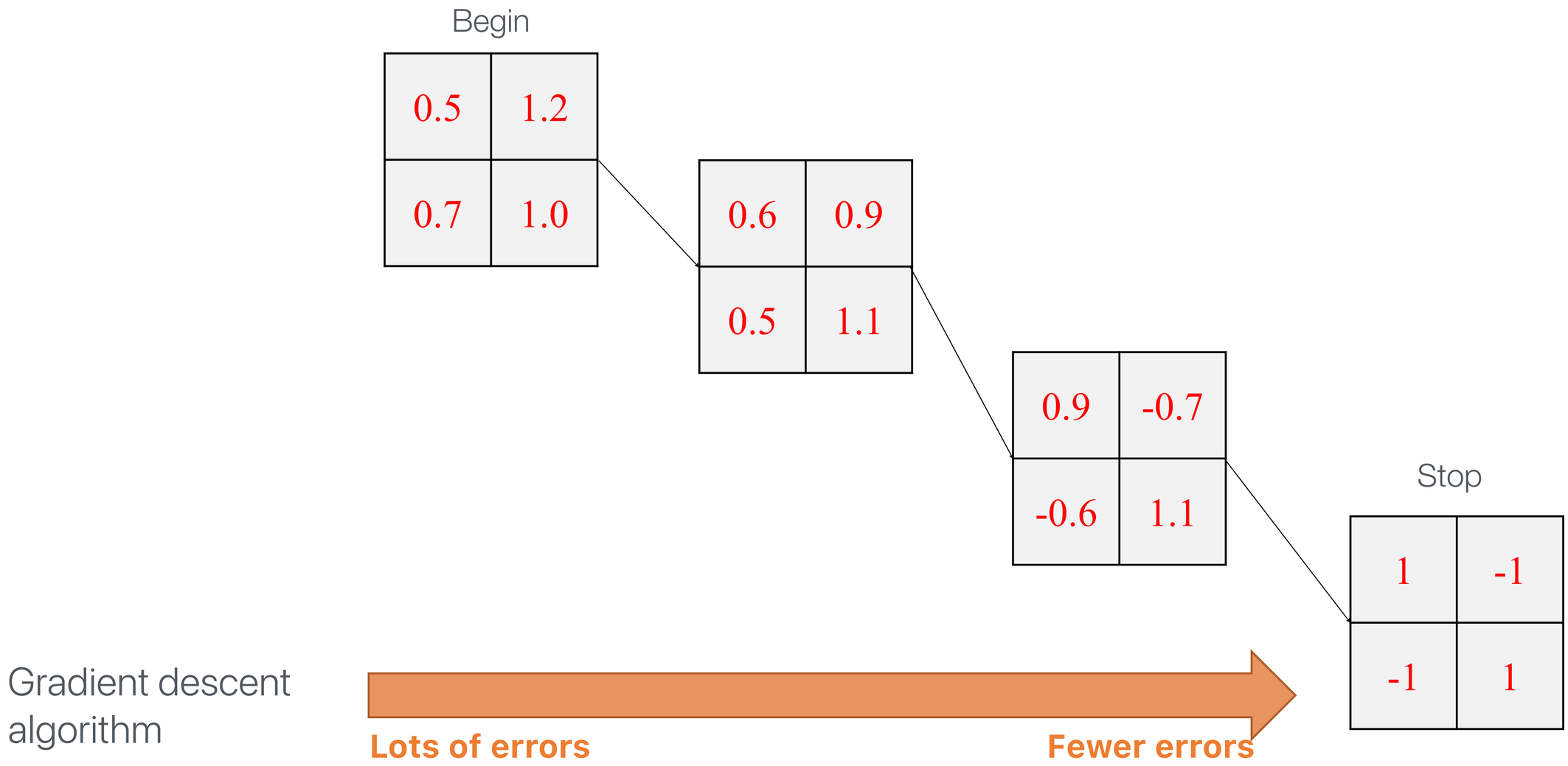




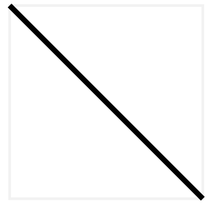
How can a computer find the filters/kernels ?



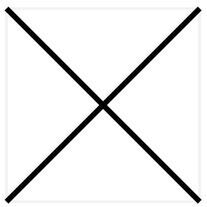

Begin
so-so
worse
better
worse
better
worse
better
Stop !
Make small changes
Make small changes


Begin
0.5
1.2
0.7
1.0
0.6
0.9
0.5
1.1
0.9
-0.7
-0.6
1.1
Stop
1
-1
-1
1
Gradient descent algorithm
Lots of errors
Fewer errors

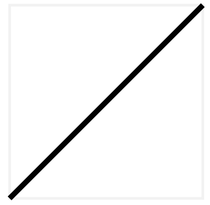
More Complex Examples



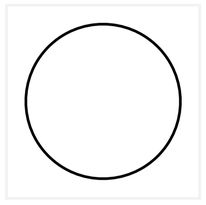
1	-1	-1
-1	1	-1
-1	-1	1



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



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

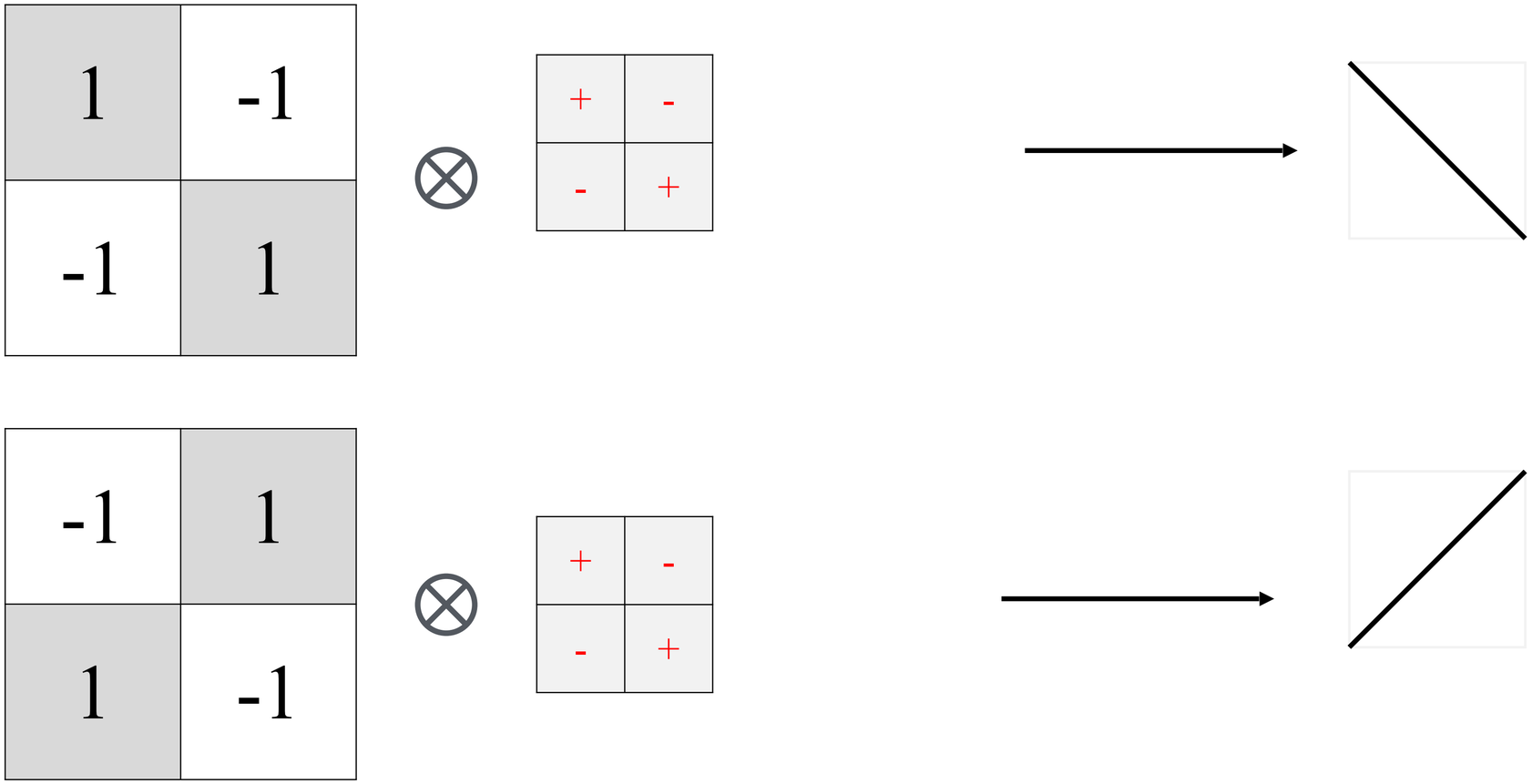


-1	1	-1
1	-1	1
-1	1	-1

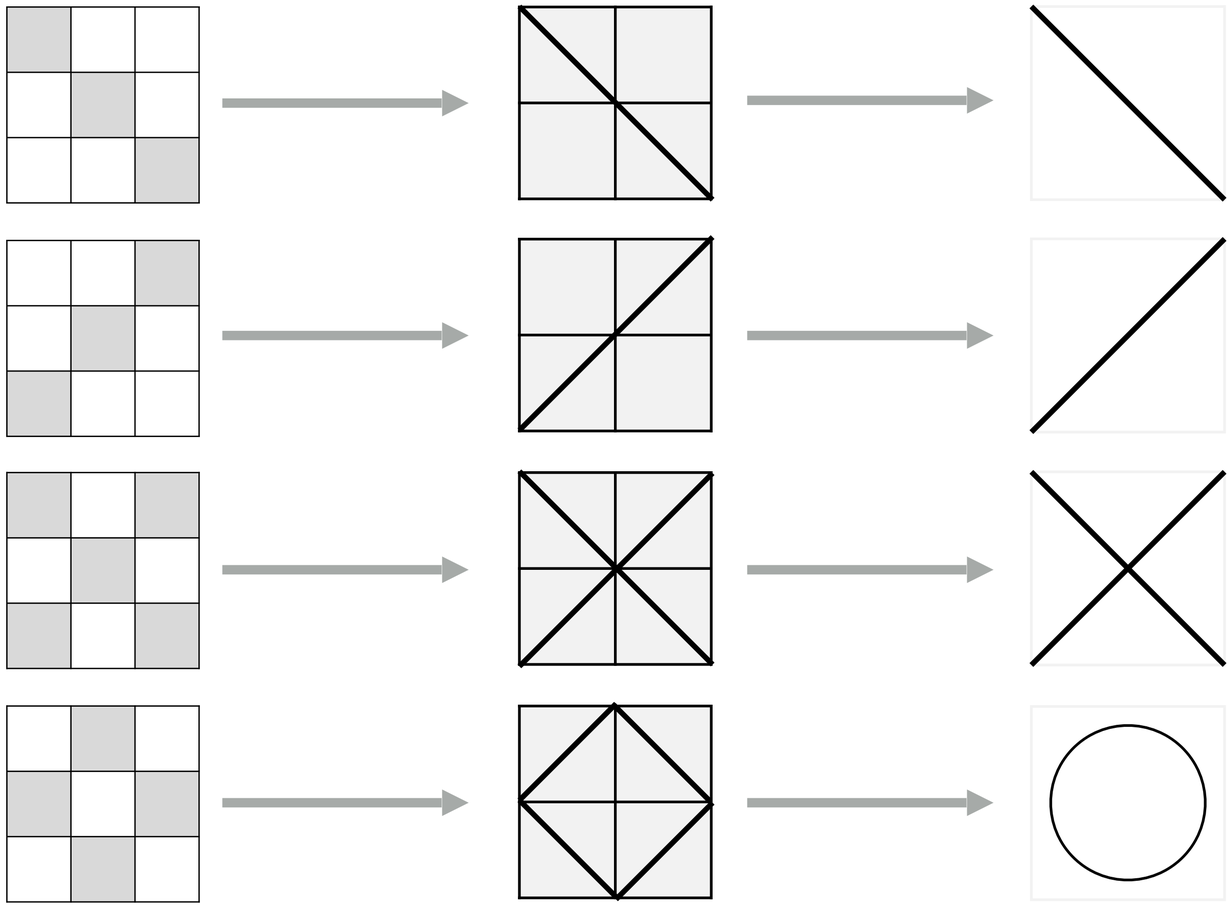
How about this filter ?

+ 1	- -1	+ 1
- -1	+ 1	- -1
+ 1	- -1	+ 1

What we have so far

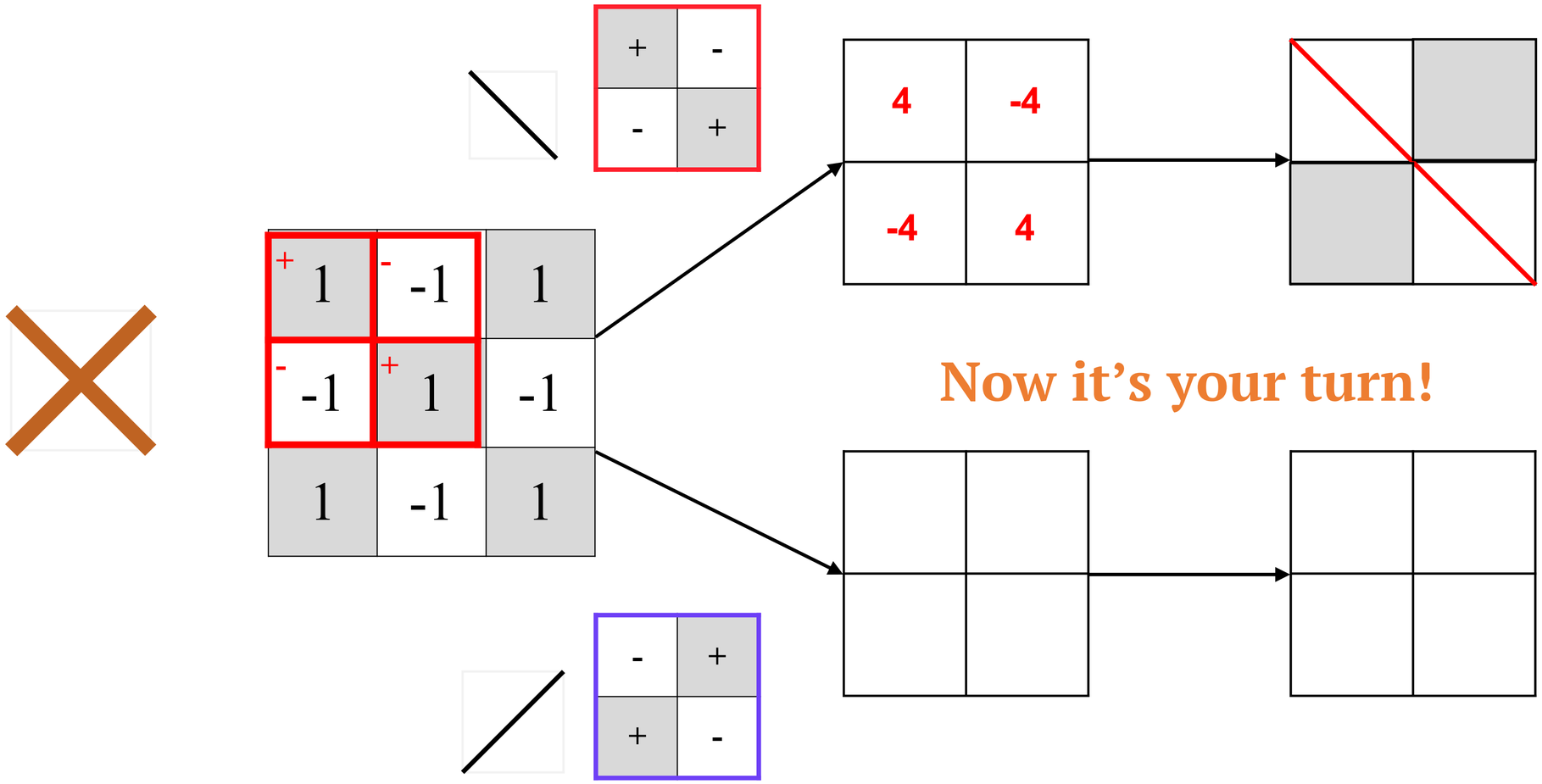


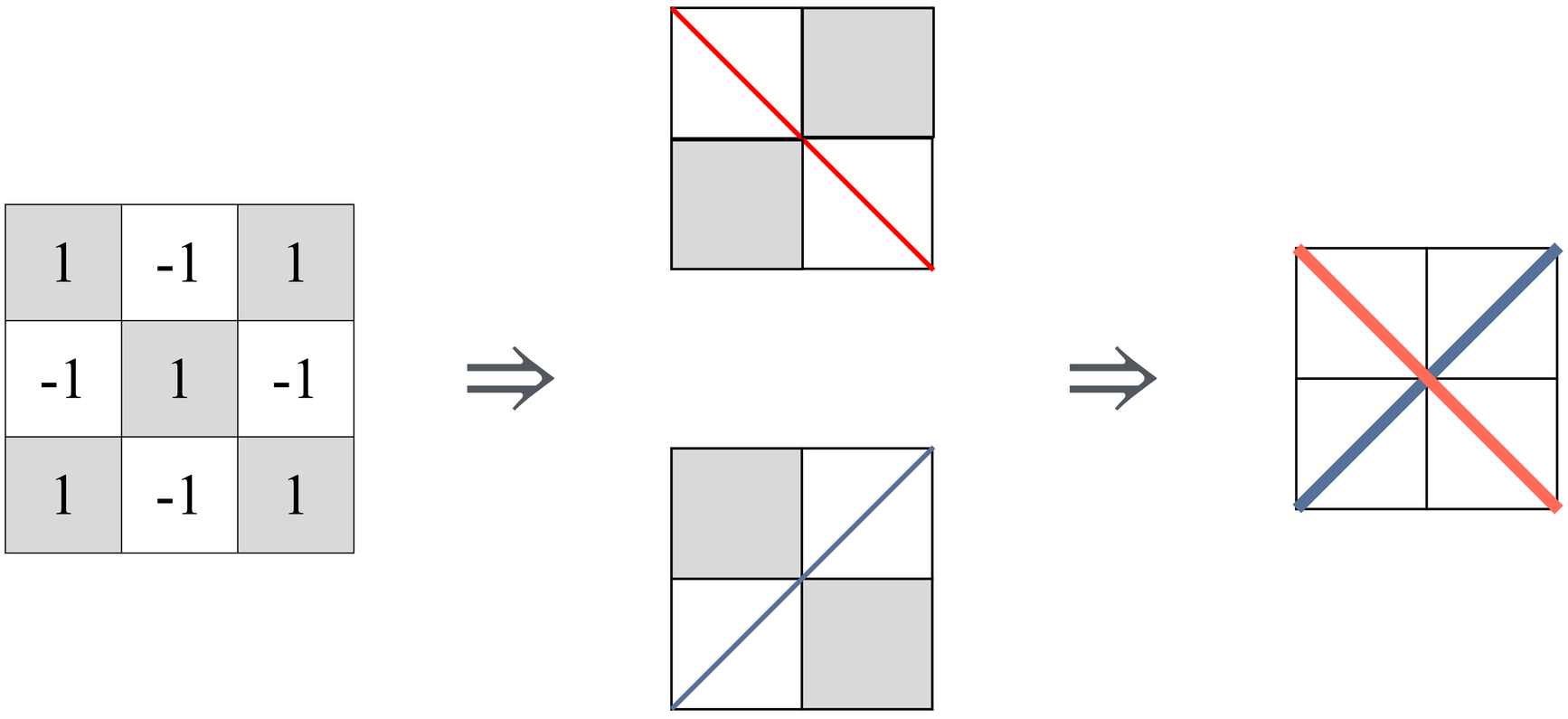
Basic Concepts of CNN



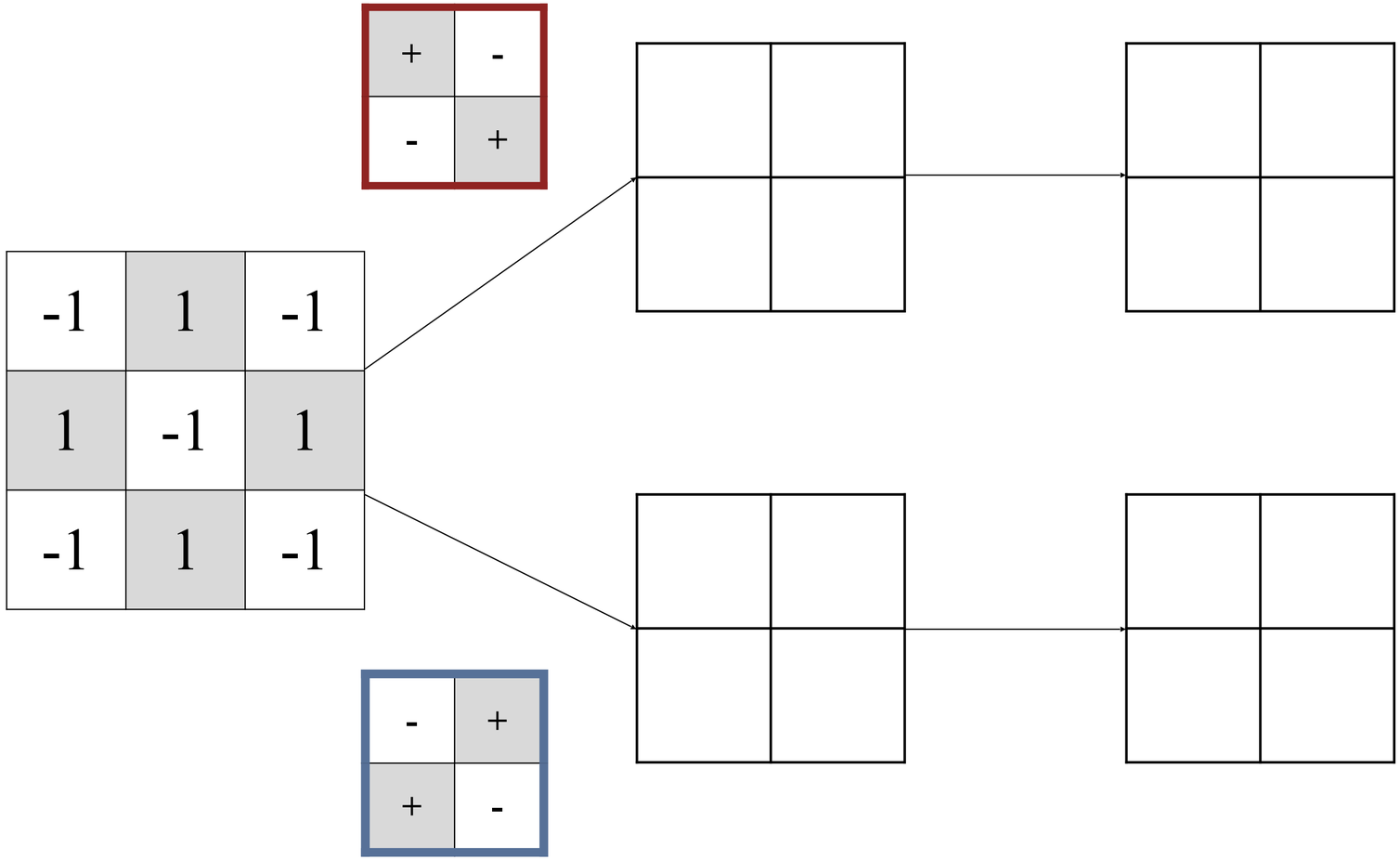
Convolutional Layer
Pooling Layer

Fully Connected Layer

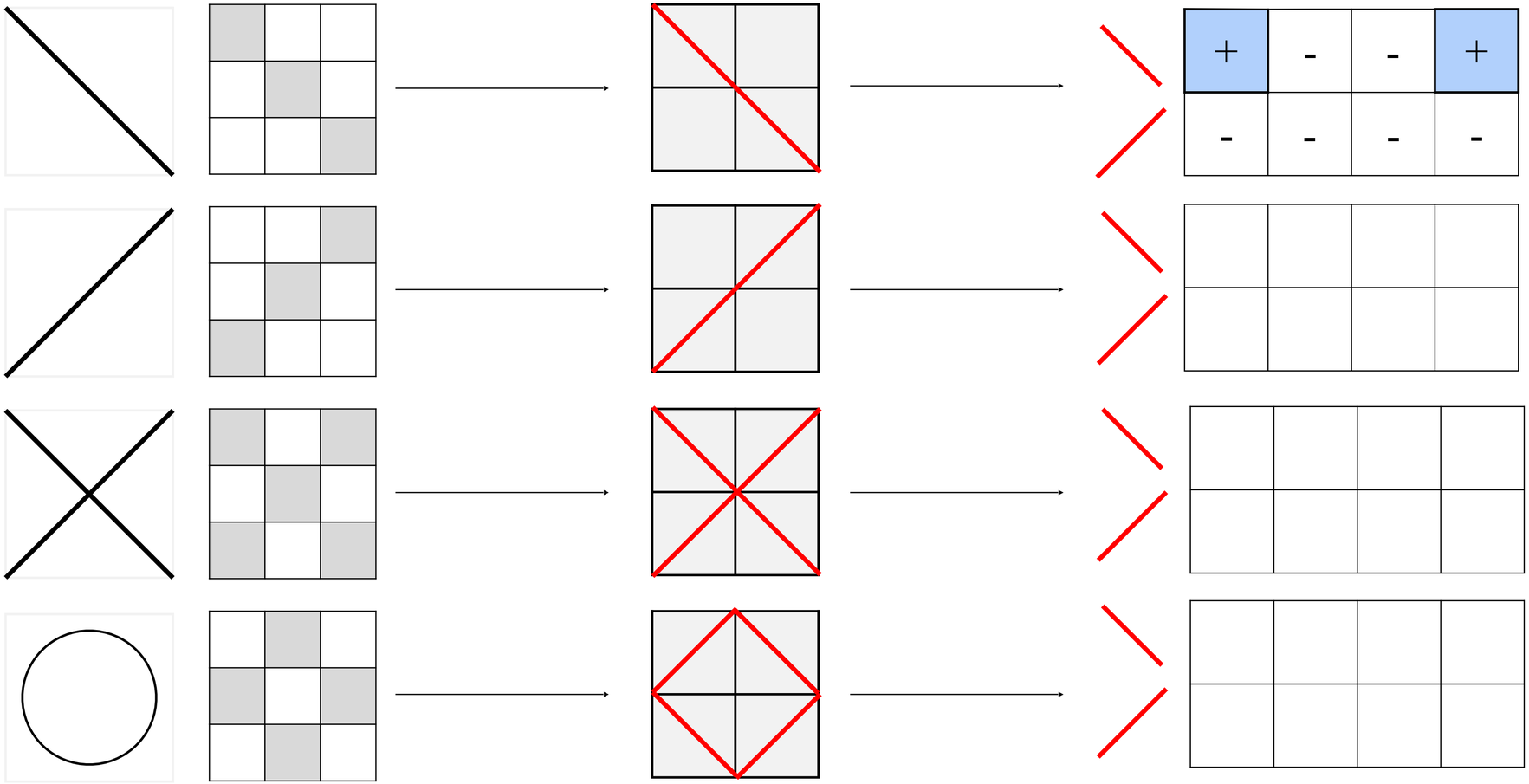

+ -
- +
4 -4
-4 4
+ 1 - -1 1
- -1 + 1 -1
1 -1 1
- +
+ -
Now it’s your turn!


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

What about Oval Shape ?



Two output feature maps

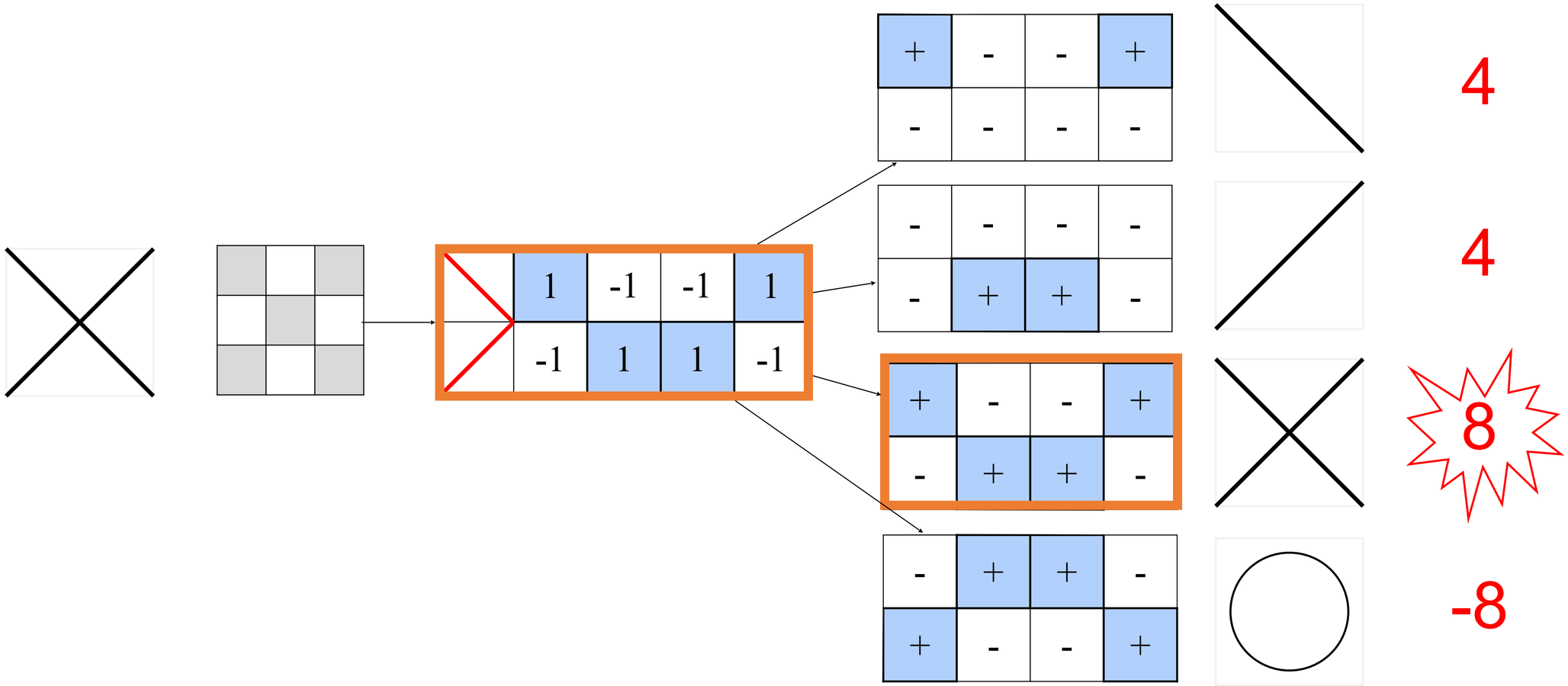


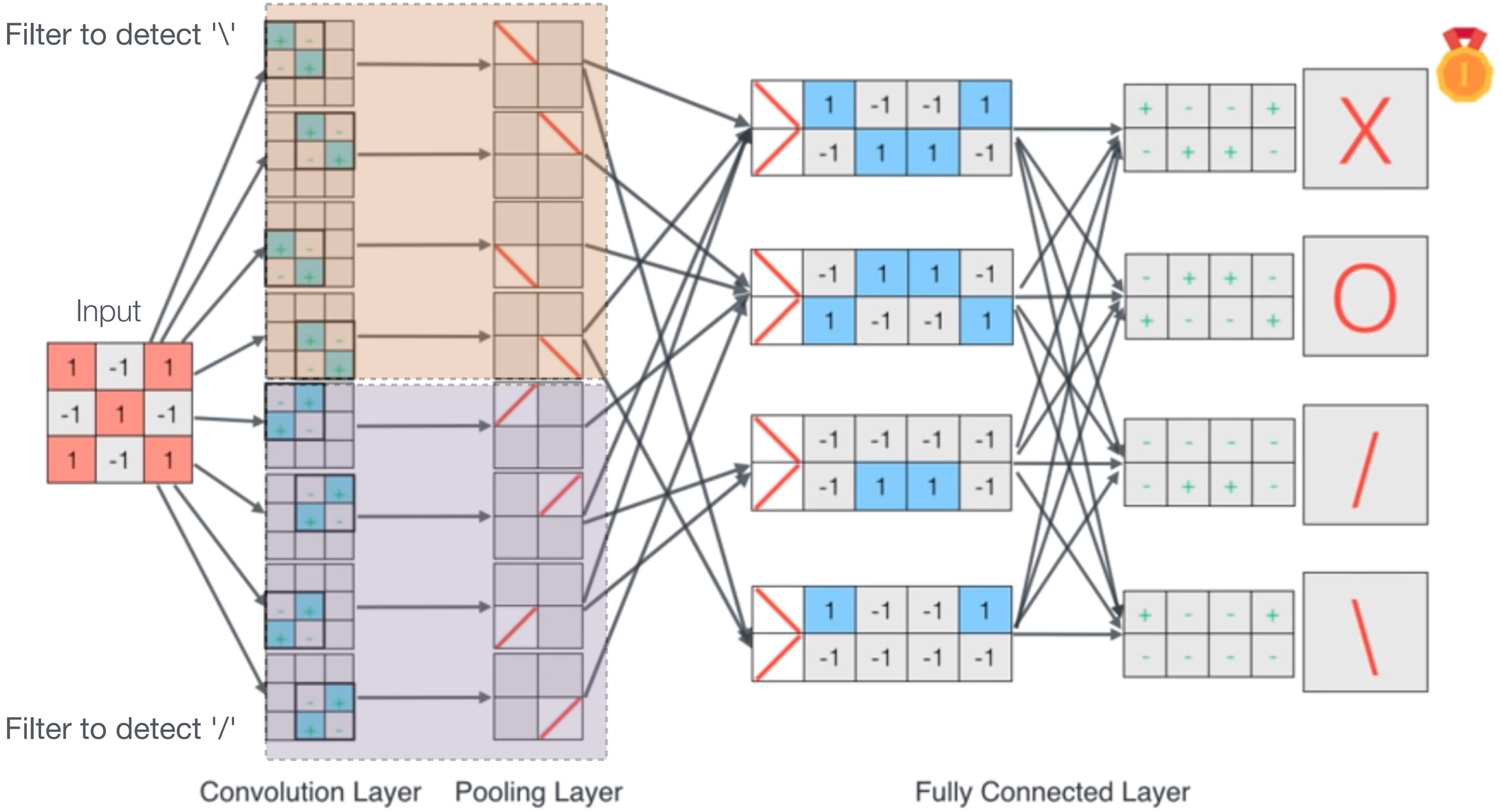
To Fully-connected layer

Fully Connected Layer

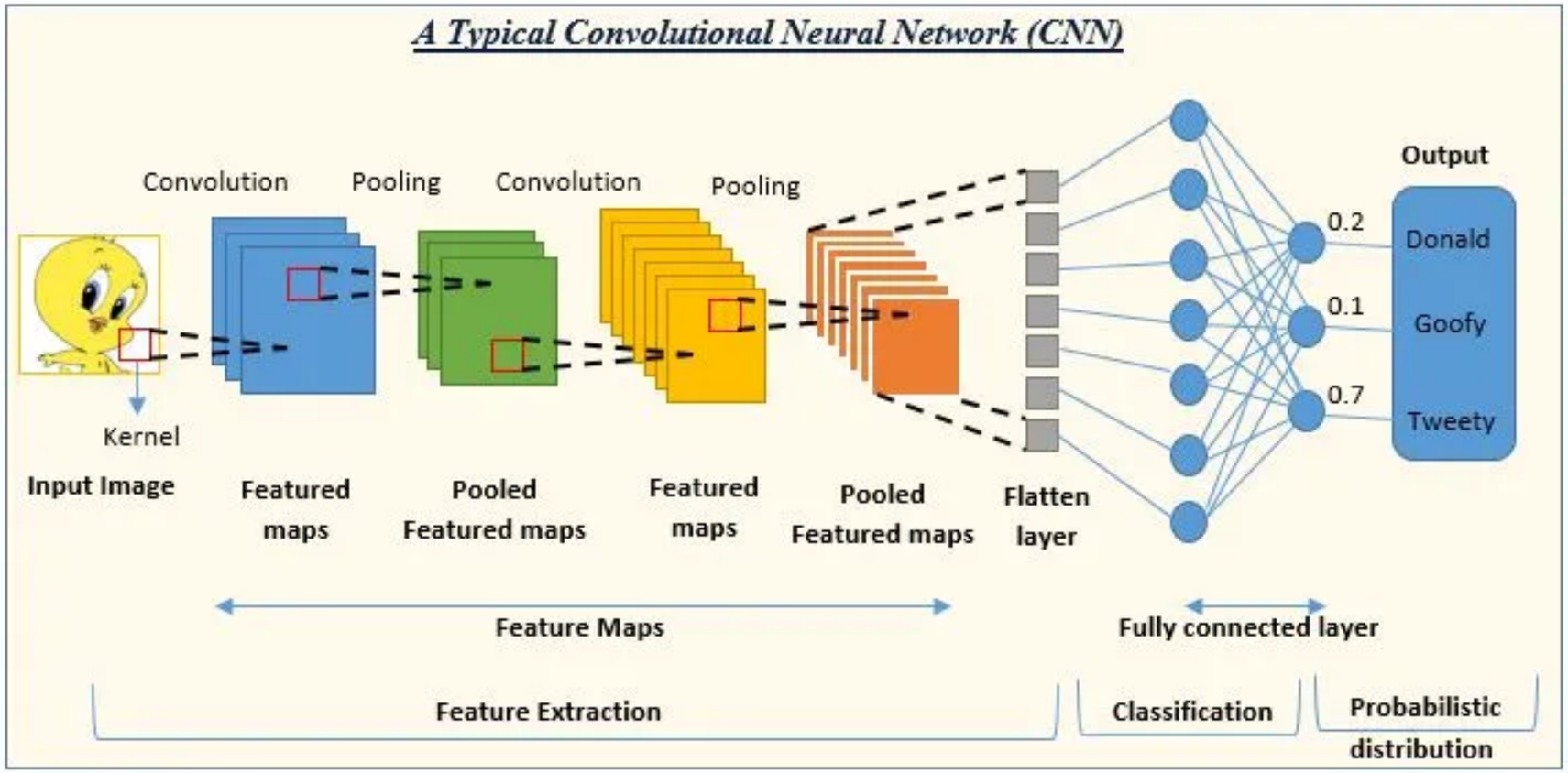
Output nodes = # Targets

Node outputs from FC layer

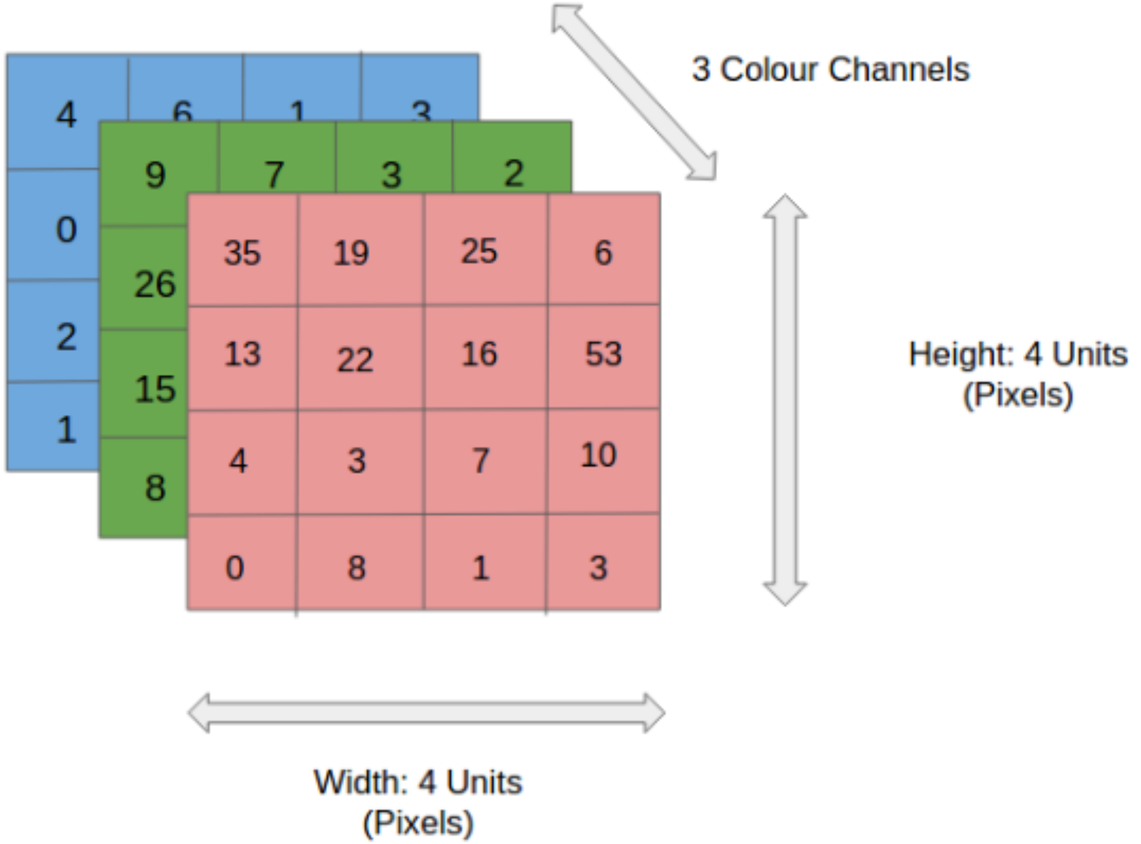



Filter to detect '\'
Input
Filter to detect '/'
Convolution Layer
Pooling Layer
Fully Connected Layer
X
O
/
\

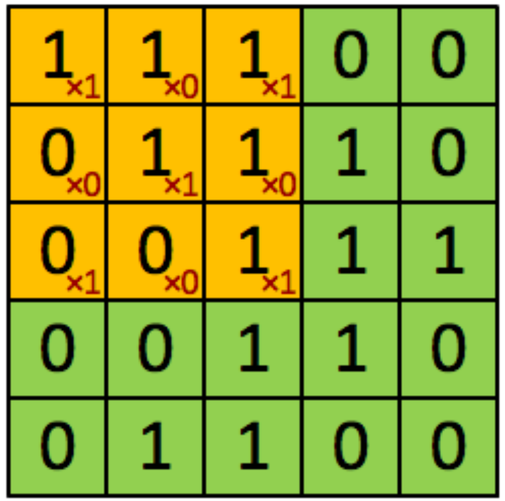
Basic CNN Architecture



Input Image

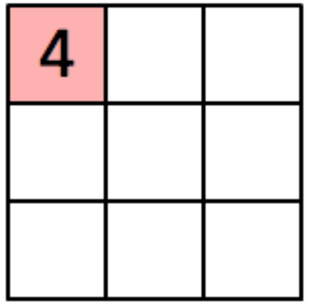

3 Colour Channels
4
6
1
3
9
7
3
2
0
26
35
19
25
6
2
15
13
22
16
53
1
8
4
3
7
10
0
8
1
3
Height: 4 Units
(Pixels)
Width: 4 Units
(Pixels)

Convolutional Layer

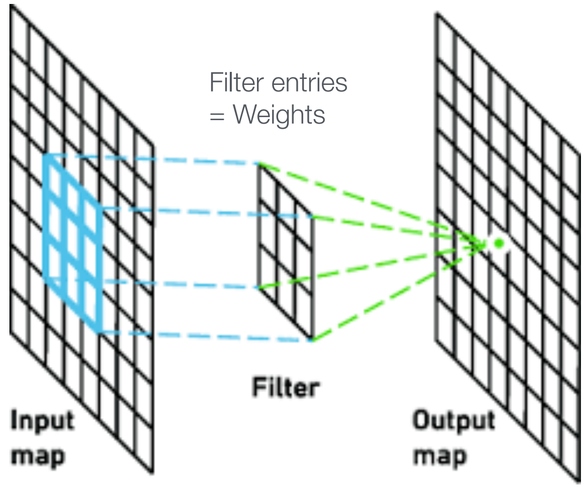


Image

(Single channel)



Convolved Feature



For each filter in the 1st convolutional layer,
- Convolute the filter (+ activation function) to each input channel
- Combine (element-wise) the output feature maps from all channels.

Strides

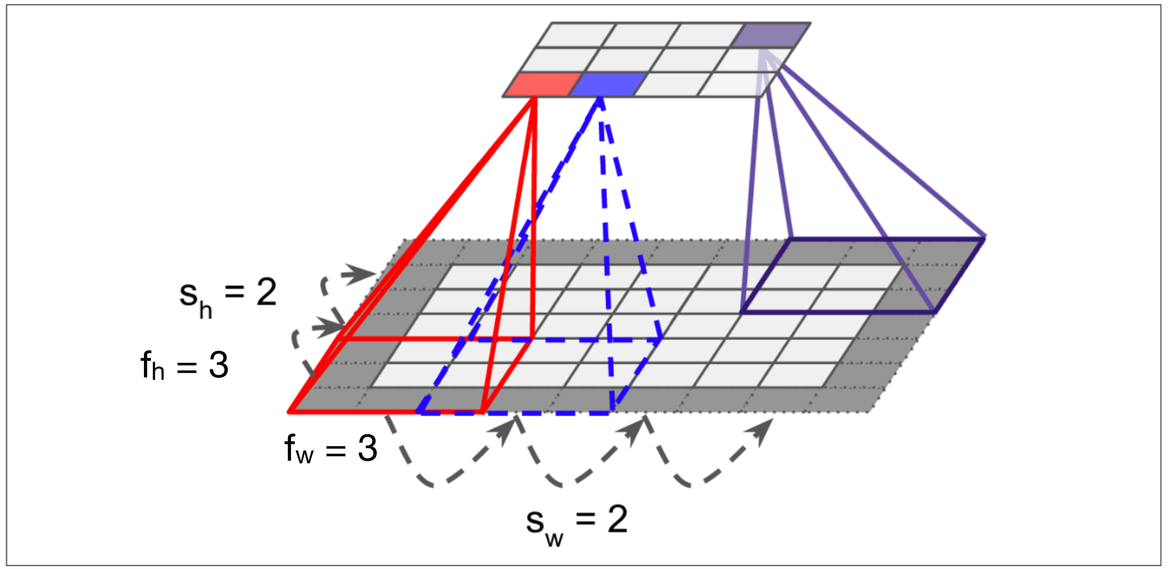
The amount of horizontal/vertical shifts from one receptive field to the next.

Let s_h and s_w be the vertical and horizontal strides.

Let f_h and f_w be the vertical and horizontal filter dimensions.

Neuron (i, j) in the upper layer is connected to the output of the neurons in the previous layer located in rows $i \cdot s_h$ to $i \cdot s_h + f_h - 1$, columns $j \cdot s_w$ to $j \cdot s_w + f_w - 1$.

$s_h = s_w$ and $f_h = f_w$ are commonly used.

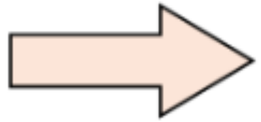


Stride of 2 pixels

(Source: Raghav Prabhu)

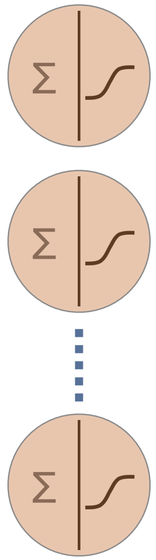
1	2	3	4	5	6	7
11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	34	35	36	37
41	42	43	44	45	46	47
51	52	53	54	55	56	57
61	62	63	64	65	66	67
71	72	73	74	75	76	77

Convolve with 3x3 filters filled with ones



108	126	
288	306	

1	1	1
1	1	1
1	1	1

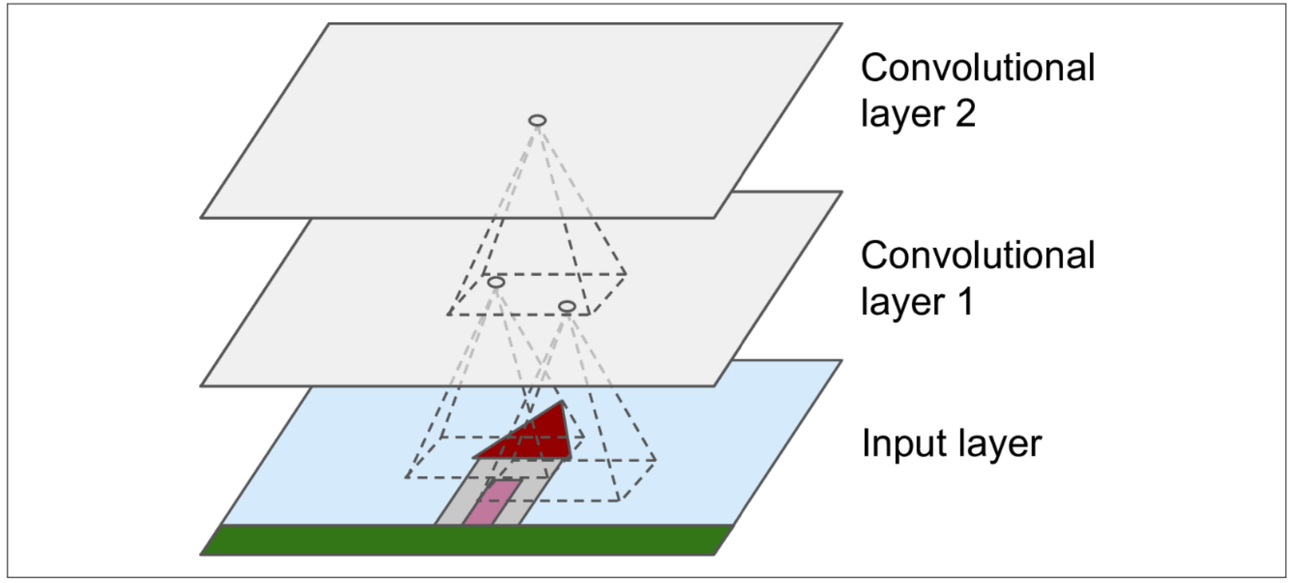


Same activation function (+ bias unit) as an option

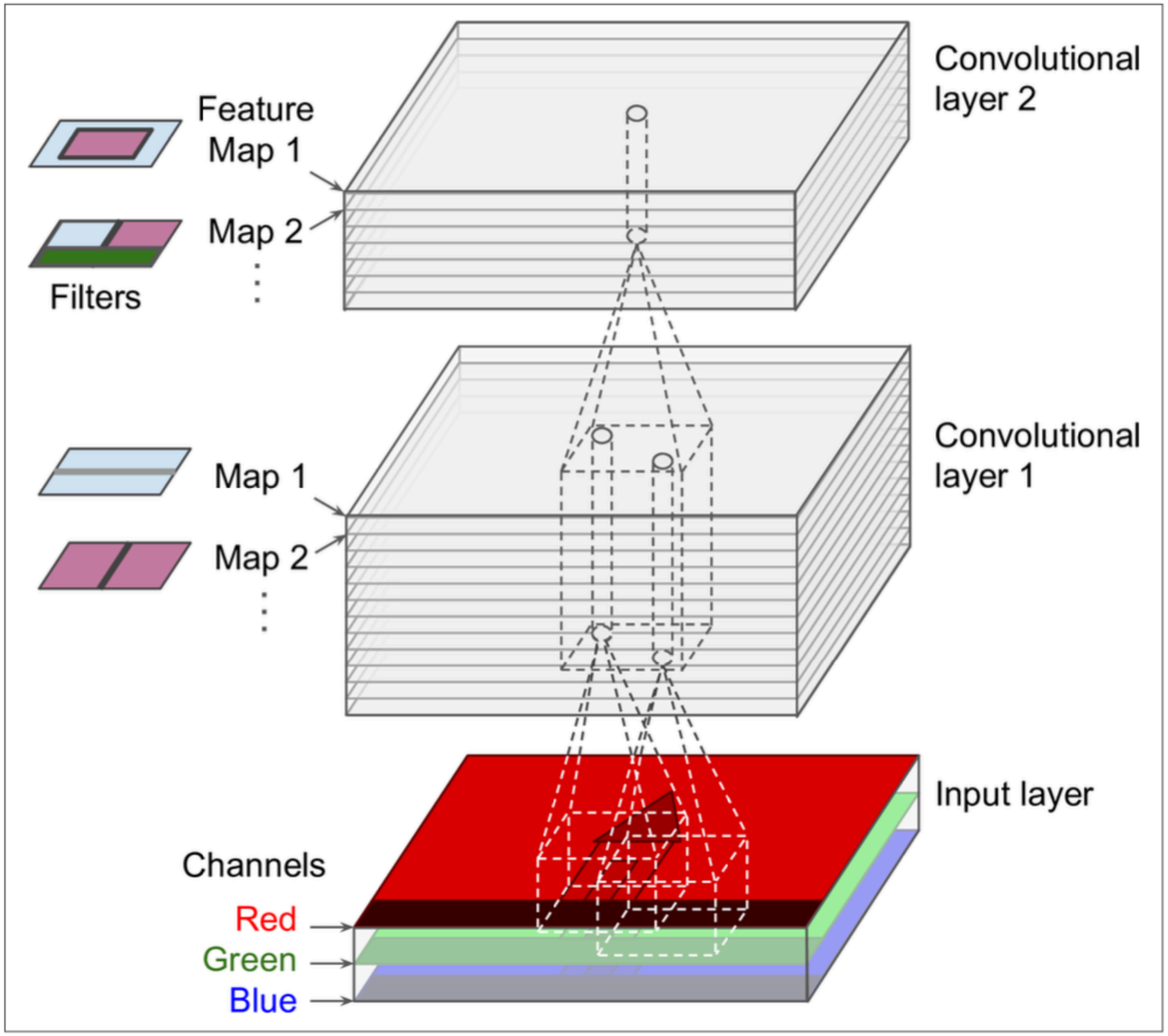
One or more convolutional layers can be added.

- Neurons in the 1^{st} convolutional layer connect to pixels in their receptive fields.
- Neurons in the 2^{nd} convolutional layer connect to neurons located within a small rectangle in the 1^{st} layer.

Allows the network to concentrate on small low-level features in the 1^{st} hidden layer, then assemble them into larger higher-level features in the next hidden layer, and so on.



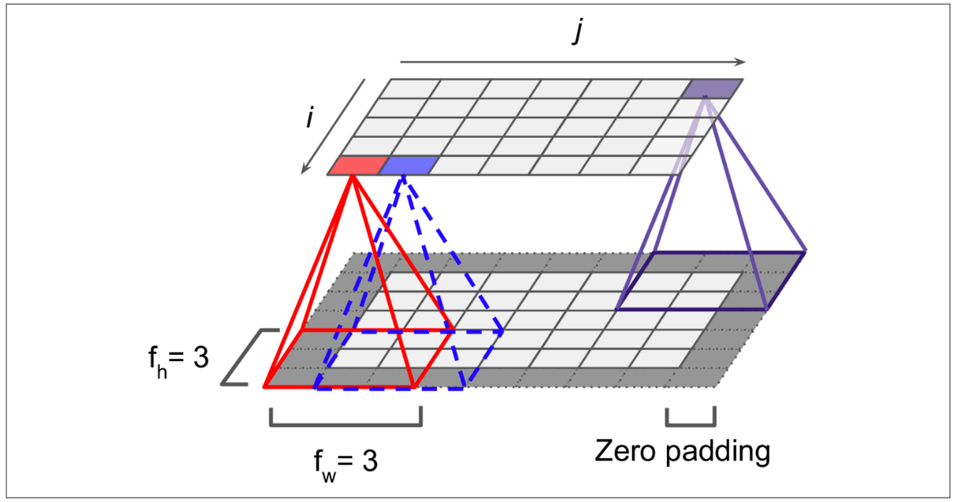
Geron. Figure 14-2

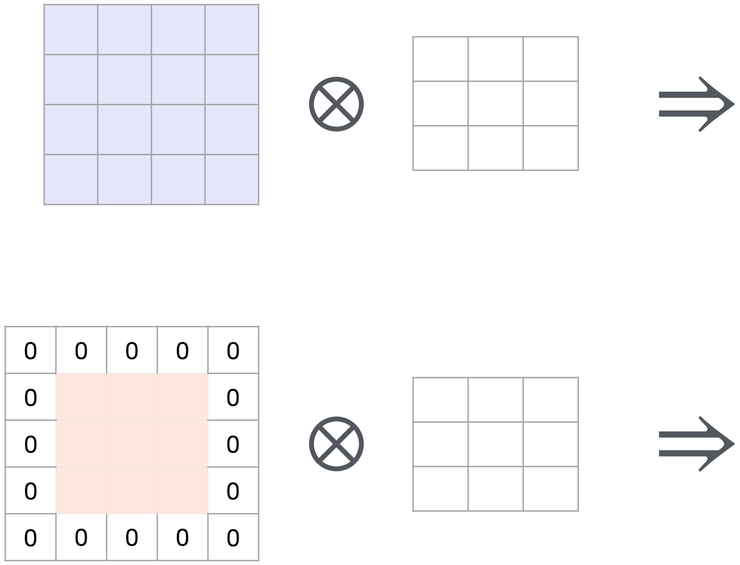

Convolutional layer 2
Feature Map 1
Map 2
Filters
Convolutional layer 1
Map 1
Map 2
Input layer
Channels
Red
Green
Blue

Border Effect and Padding

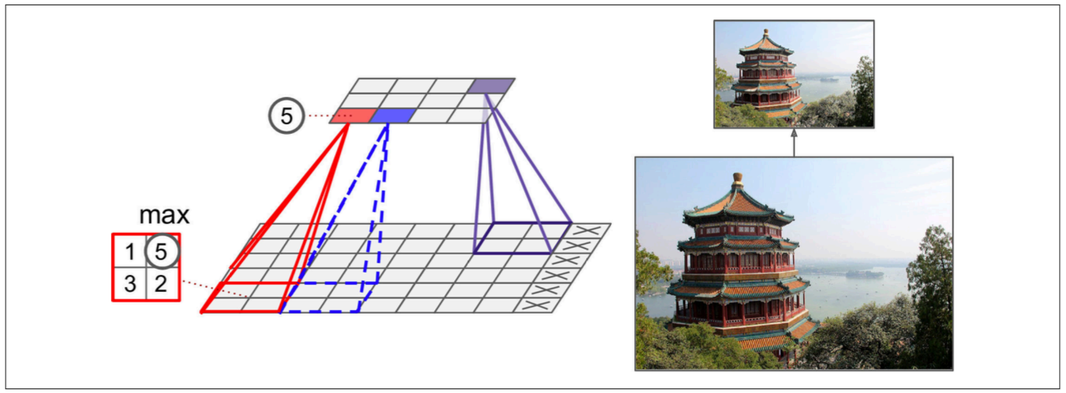
The sizes of output feature maps keep decreasing through multiple convolutional layers.

In order for an output to have the same height and width as that of the previous layer, it is common to add zeros around the inputs, called ***zero padding***.





Pooling Layer

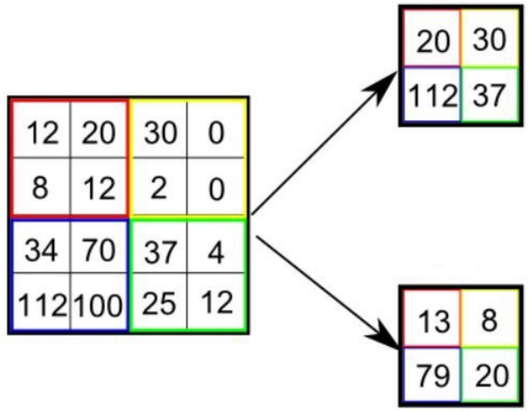


Added between CNN layers

- Max pooling
- Average pooling
- Sum pooling

Why ?

- Reduce the dimensionality, # parameters, and computations
- Shorten the training time
- Control overfitting



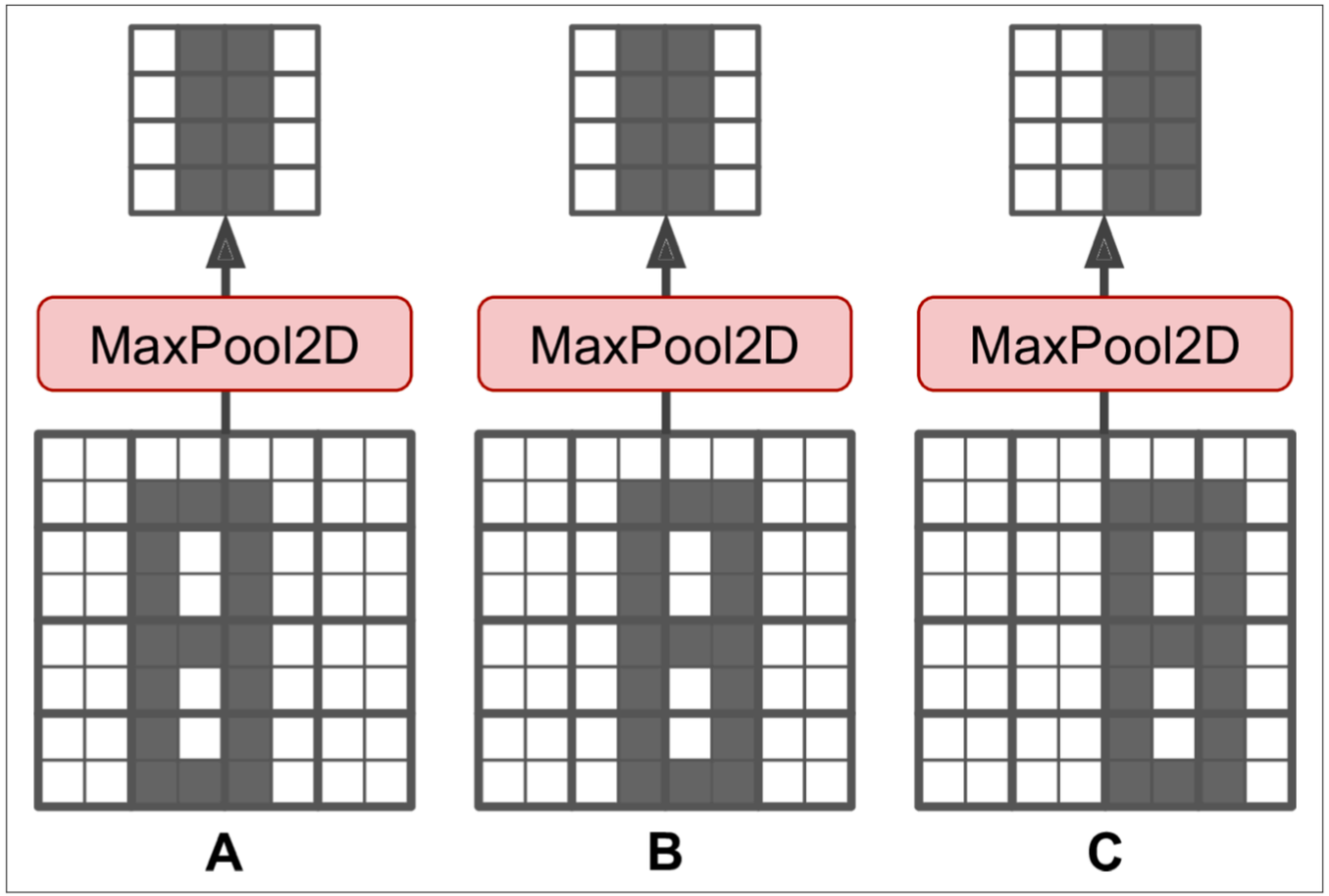
2×2 max pooling kernel
with stride 2 (no padding by default)

2×2 average pooling kernel
with stride 2 (no padding by default)

ReLu layer can be applied after the pooling layer to eliminate negatives.

2×2 max pooling kernel with stride 2

Feature maps



Well–known CNN Architectures

Classical architecture:

- LeNet-5 (1998)

Three winners of the ILSVRC challenge:

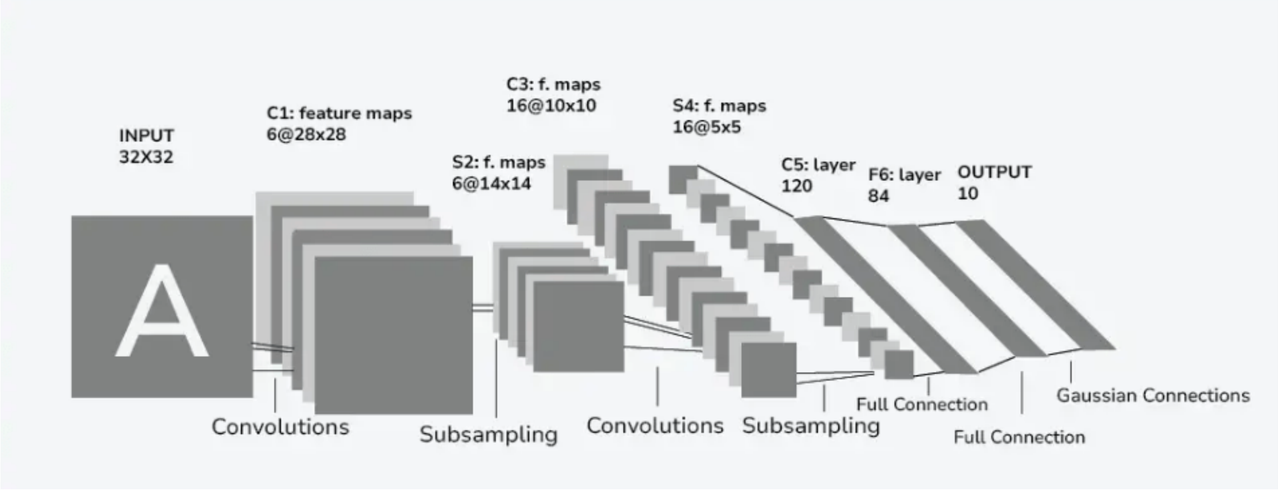
- AlexNet (2012)
- InceptionNet/GoogLeNet (2014)
- ResNet (2015)

Other architectures - VGGNet, Xception, SENet,

LeNet-5 Architecture

Early CNN primarily designed to recognize handwritten and machine-printed characters.

Set the foundation for subsequent CNN development.

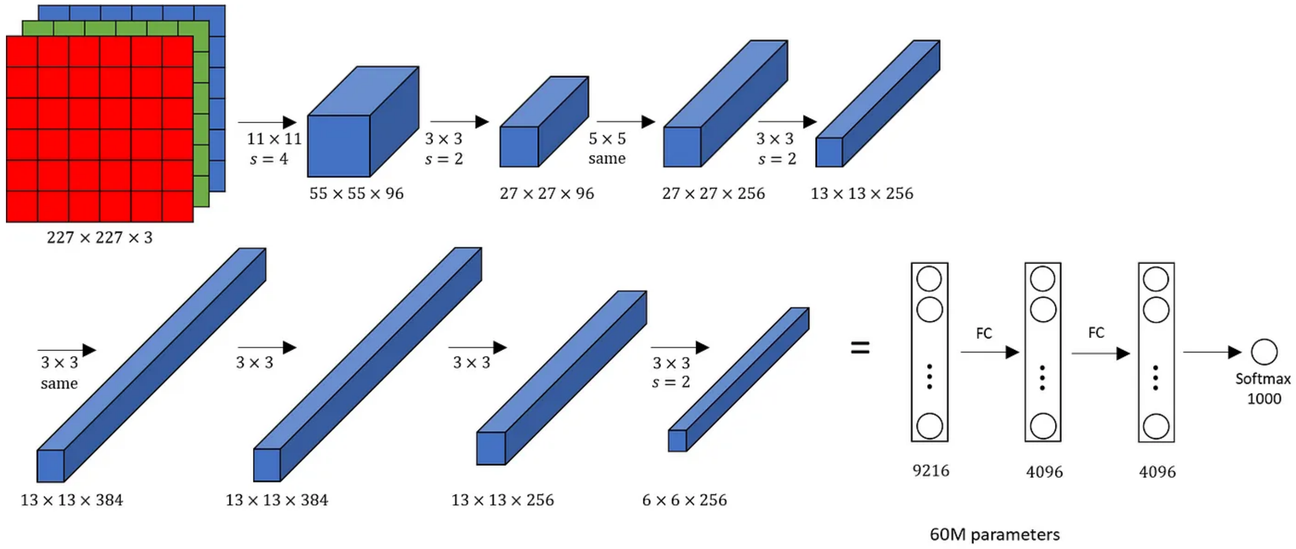


Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg Pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg Pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

AlexNet

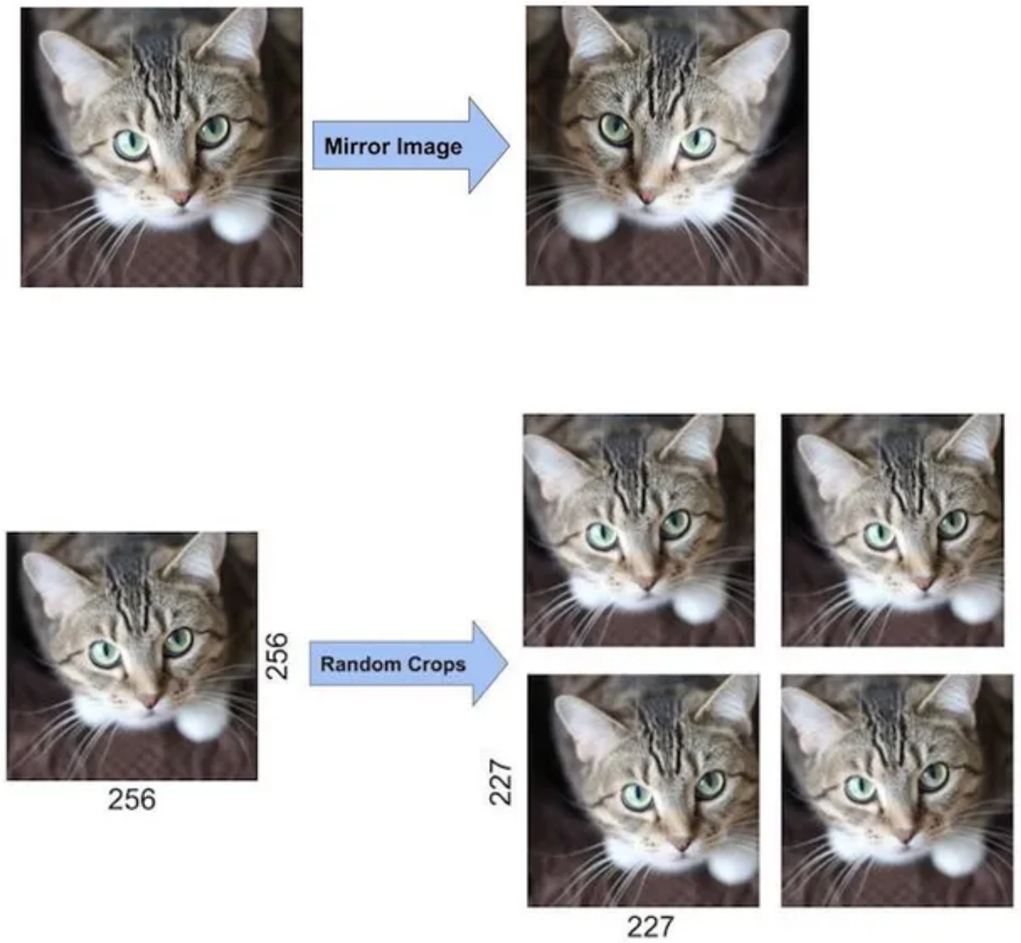
Marked significant advancement in CNNs for improved accuracy.

- Larger filter sizes
- Leveraging GPUs for faster training
- Data augmentation

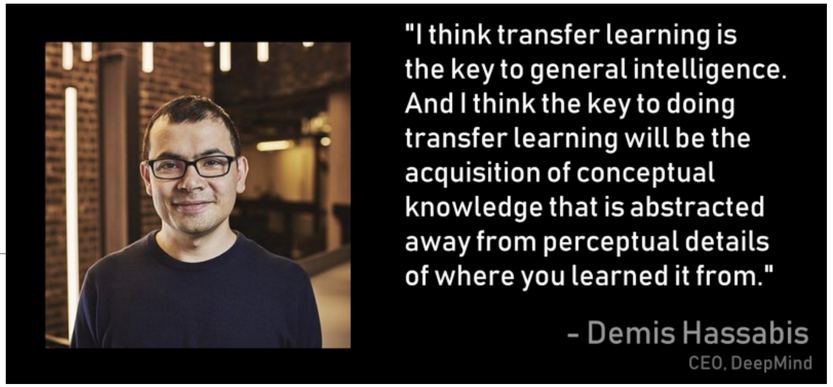


Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

Data Augmentation



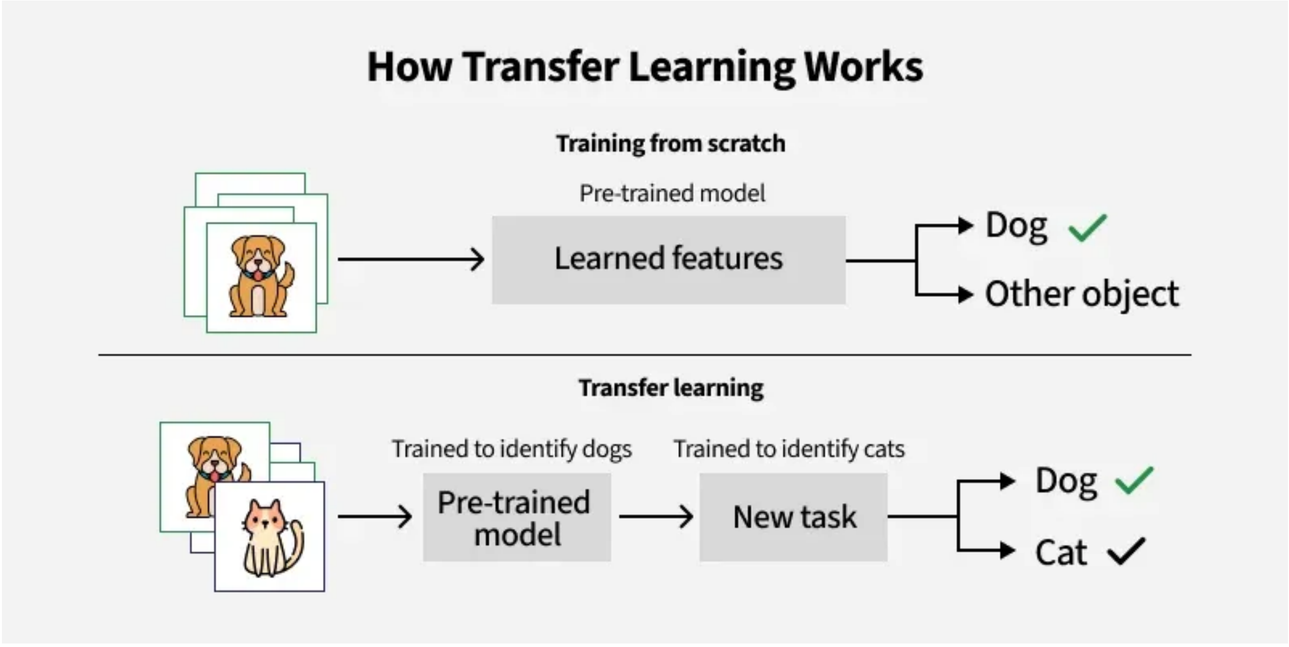
Transfer Learning

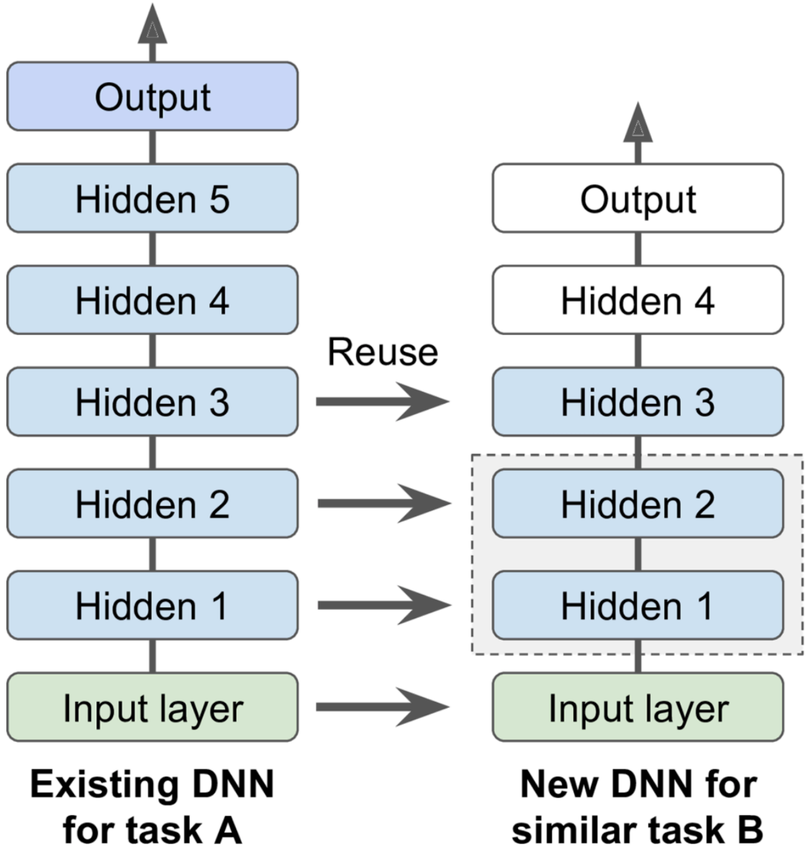


Model trained on one task is re-purposed on a second related task.

- Train a simple classifier to predict whether an image contains a backpack.
- Use the knowledge gained during its training to recognize other objects like sunglasses.

Mostly used in computer vision and natural language processing tasks





Trainable weights

Fixed weights

Layers to reuse ∝ Task similarity

Layers to freeze ∝ 1/Data availability

Base (Pre-trained) model

Fine-tuned model

1. Frozen Layers: These layers from a pre-trained model remain unchanged during fine-tuning. They retain general features learned from the original task, extracting universal patterns from input data.
2. Trainable Layers: These layers are adjusted during fine-tuning to learn task-specific features from the new dataset, allowing the model to meet the new task's unique requirements.

Which Layers to Freeze or Train ?

Small, Similar Dataset: For smaller datasets that resemble the original dataset, you freeze most layers and only fine-tune the last one or two upper layers to prevent overfitting.

Large, Similar Dataset: With large, similar datasets, you can unfreeze more upper layers, allowing the model to adapt while retaining learned features from the base model.

Small, Different Dataset: For smaller, dissimilar datasets, fine-tuning layers closer to the input layer helps the model learn task-specific features from scratch.

Large, Different Dataset: In this case, fine-tuning the entire model helps the model adapt to the new task while leveraging the broad knowledge from the pre-trained model.