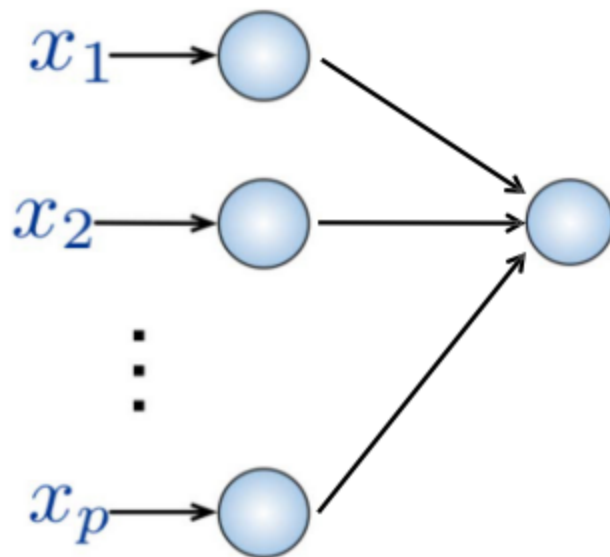


Convolutional Neural Network

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values

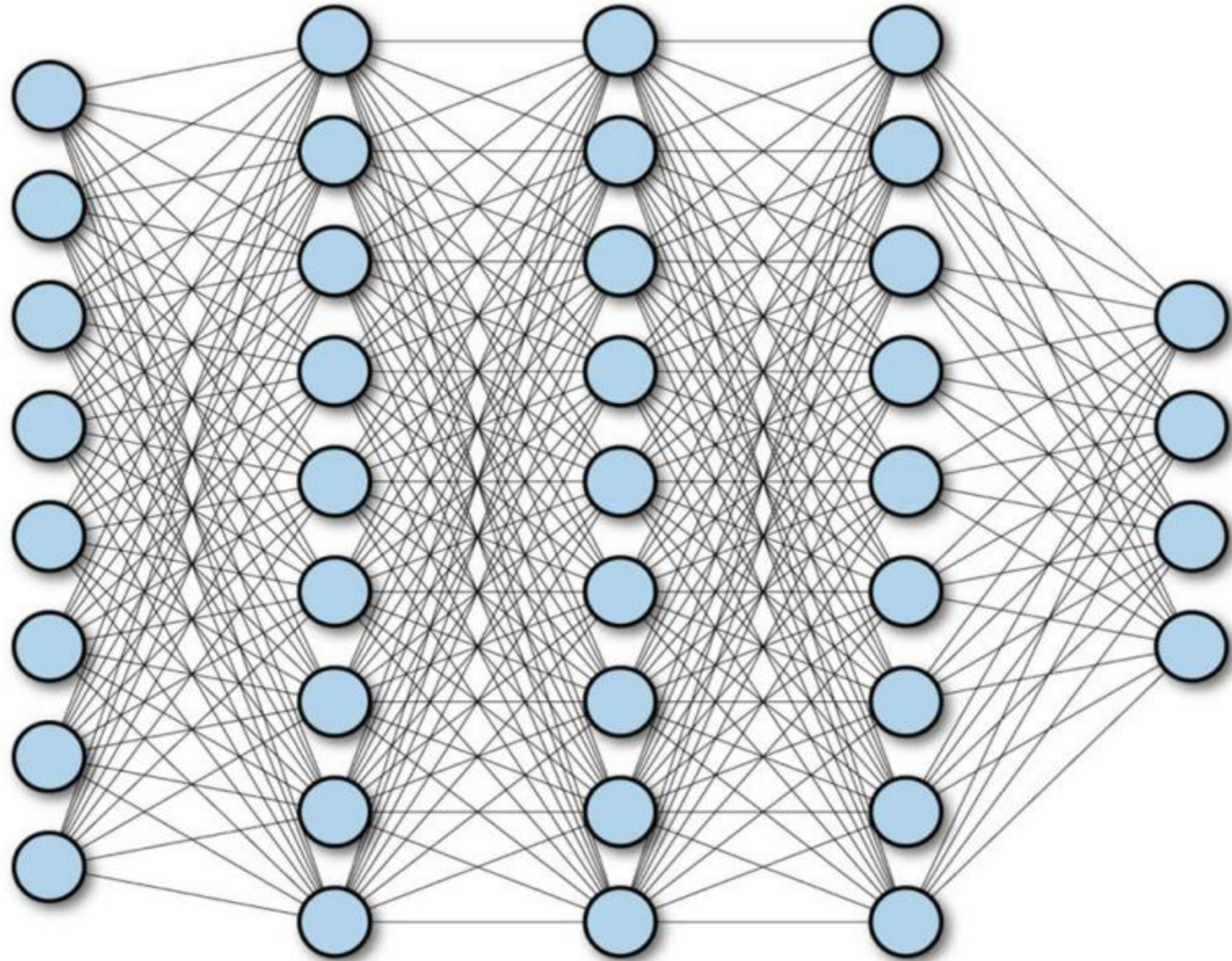


Fully Connected:

- Each neuron in hidden layer connected to all neurons in input layer
- No spatial information
- Many, many parameters

Key idea: Use spatial structure in input to inform architecture of the network

Fully Connected Neural Network



High Level Feature Detection

Let's identify key features in each image category



Nose, Eyes, Mouth



Wheels, License Plate,
Headlights



Door, Windows, Steps

Producing Feature Maps

Weighted Sum of pixels

Filter (Kernel): Weight array/matrix

Feature Map: output generated by applying a filter



Original



Sharpen



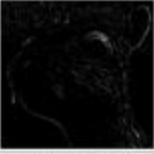






Edge Detect

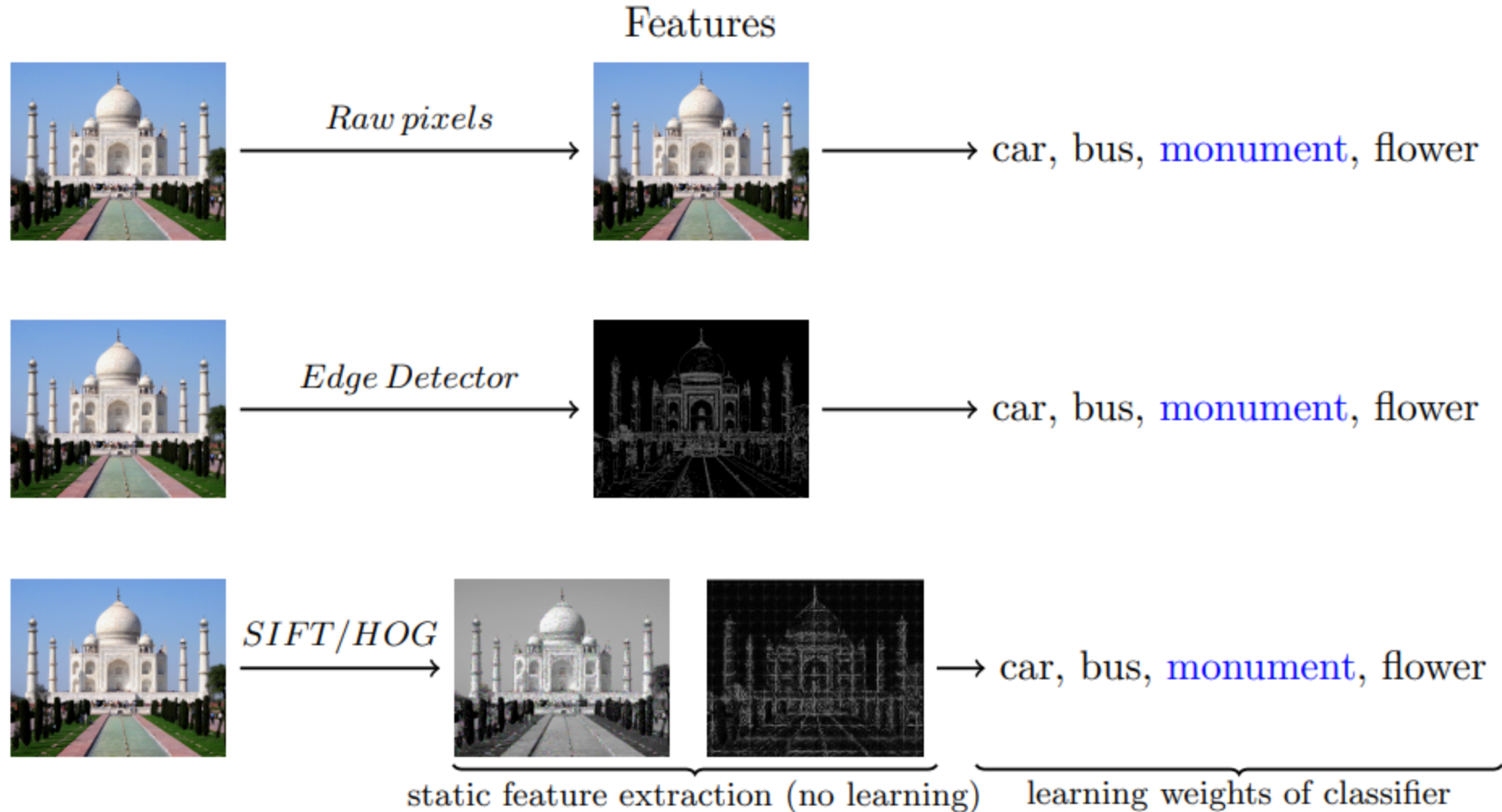


“Strong” Edge
Detect

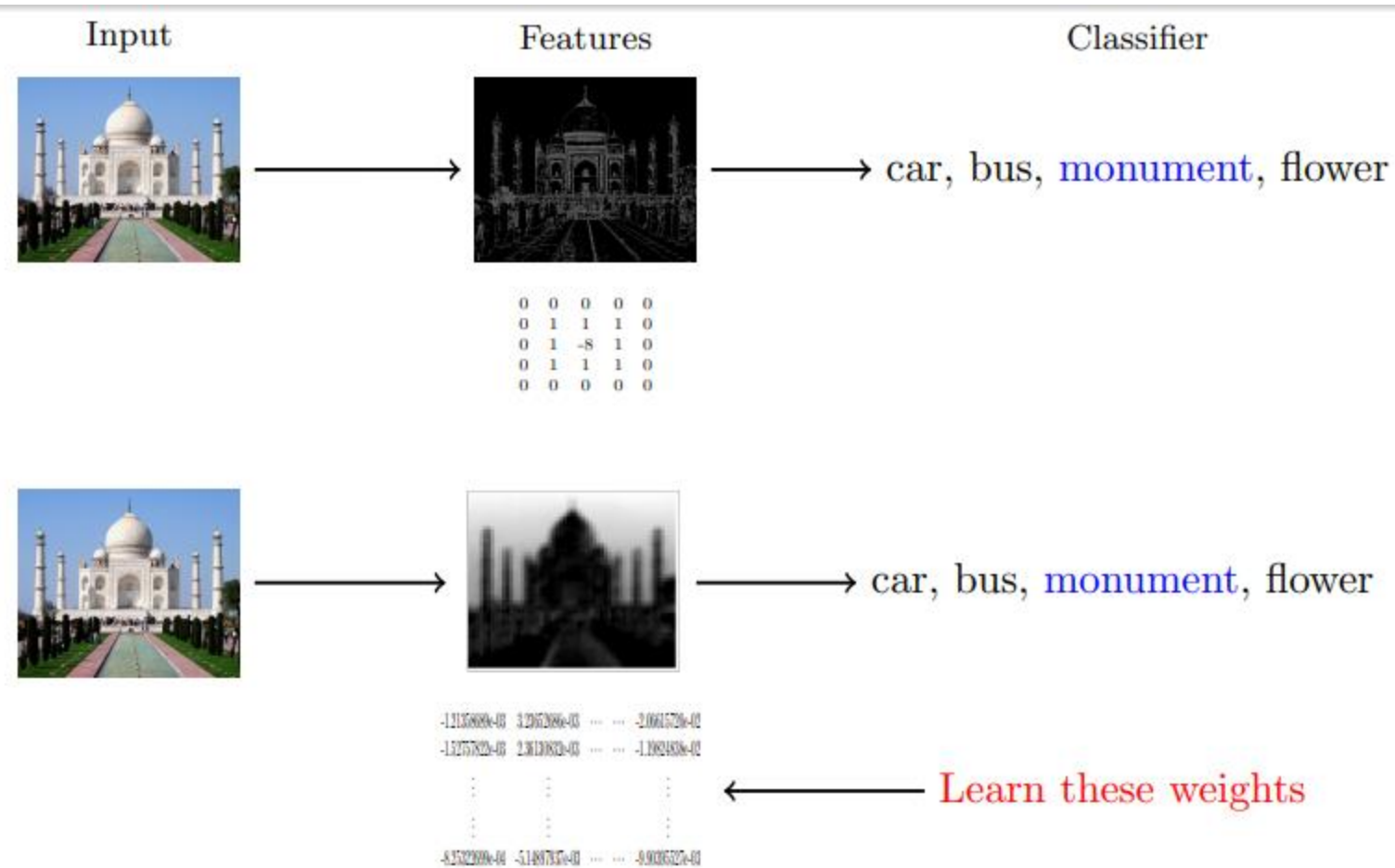
Simple Kernels / Filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Static Feature Extraction



Learn Filters



- **Initial Filters:**
- Apply to input image and find feature maps
- **Loss Calculation:**
- **Backpropagation:** The loss is propagated back through the network. The network adjusts the filter values by computing the gradients of the loss with respect to the filter weights.
- **Update Filters:**

- Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

Convolution operation is element wise multiply and add

1	0	1
0	1	0
1	0	1

Filter / Kernel

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

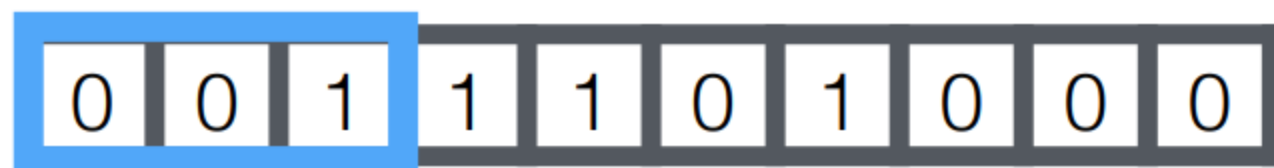
4		

Convolved
Feature

The filter is **slid** over the input image. At each position, the filter overlaps with a section of the image, and an element-wise multiplication is performed between the values in the filter and the corresponding values in the image.

Convolutional Layer: 1D example

A 1D image:



A filter:



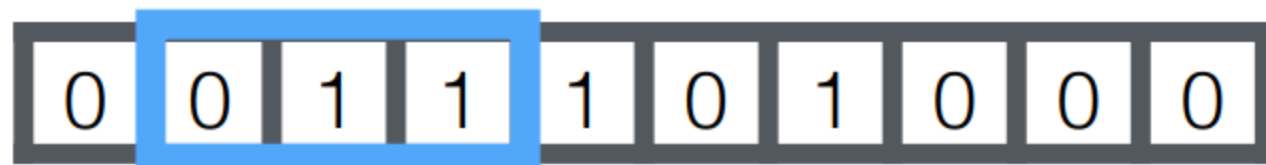
$$0 * -1 + 0 * 1 + 1 * -1 = -1$$

After
convolution*:



Convolutional Layer: 1D example

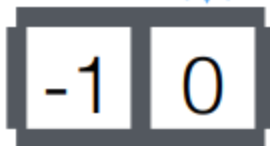
A 1D image:



A filter:

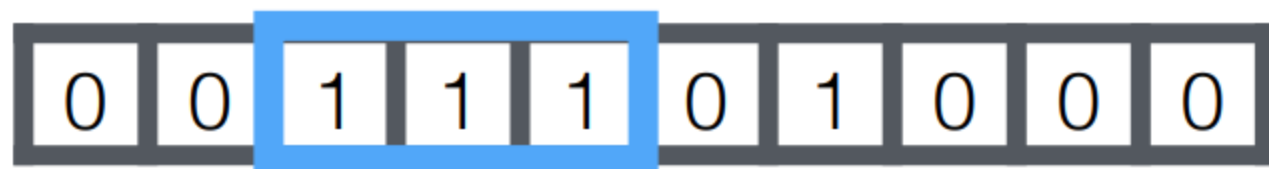


After
convolution*:



Convolutional Layer: 1D example

A 1D image:



A filter:

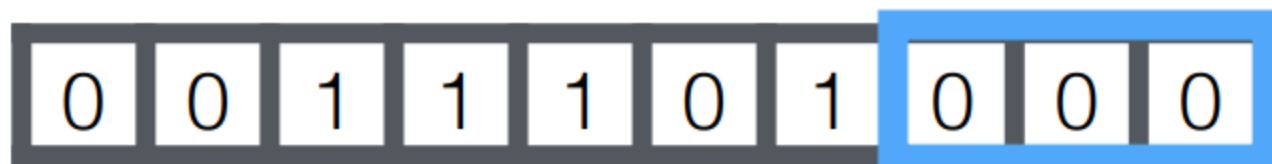


After
convolution*:



Convolutional Layer: 1D example

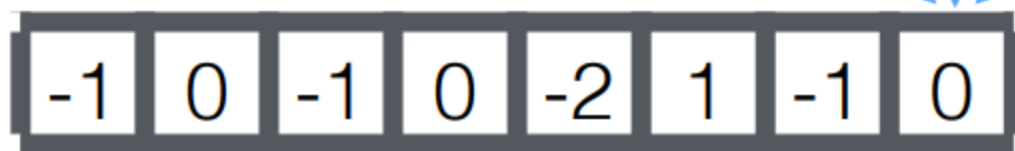
A 1D image:



A filter:



After
convolution*:



Convolutional Layer: 2D example

A 2D
image:

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

A filter:

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

Convolutional Layer: 2D example

A 2D
image:

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

A filter:

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

$$\begin{aligned} & -1 + 0 + -1 \\ & + -1 + 0 + -1 \\ & + -1 + -1 + -1 \\ & = -7 \end{aligned}$$

After
convolution:

-7

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

Convolutional Layer: 2D example

A 2D
image:

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

A filter:

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

7	-2
---	----

After
convolution:

Convolutional Layer: 2D example

A 2D image:

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

A filter:

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

When a filter (kernel) is applied to an image, it **reduces the spatial dimensions of the output feature map.**

3x3 filter to a 5x5 image reduces the output size to 3x3, which can lead to loss of information, especially around the edges.

$$\text{Output Size} = (n - f + 1) \times (n - f + 1)$$

After convolution:

-7	-2	-4
-5	-2	-5
-7	2	-5

Convolutional Layer: 2D example

A 2D
image:

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

A filter:

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

After
convolution:

-7	-2	-4
-5	-2	-5
-7	-2	-5

Convolutional Layer: 2D example

A 2D
image:

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

A filter:

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

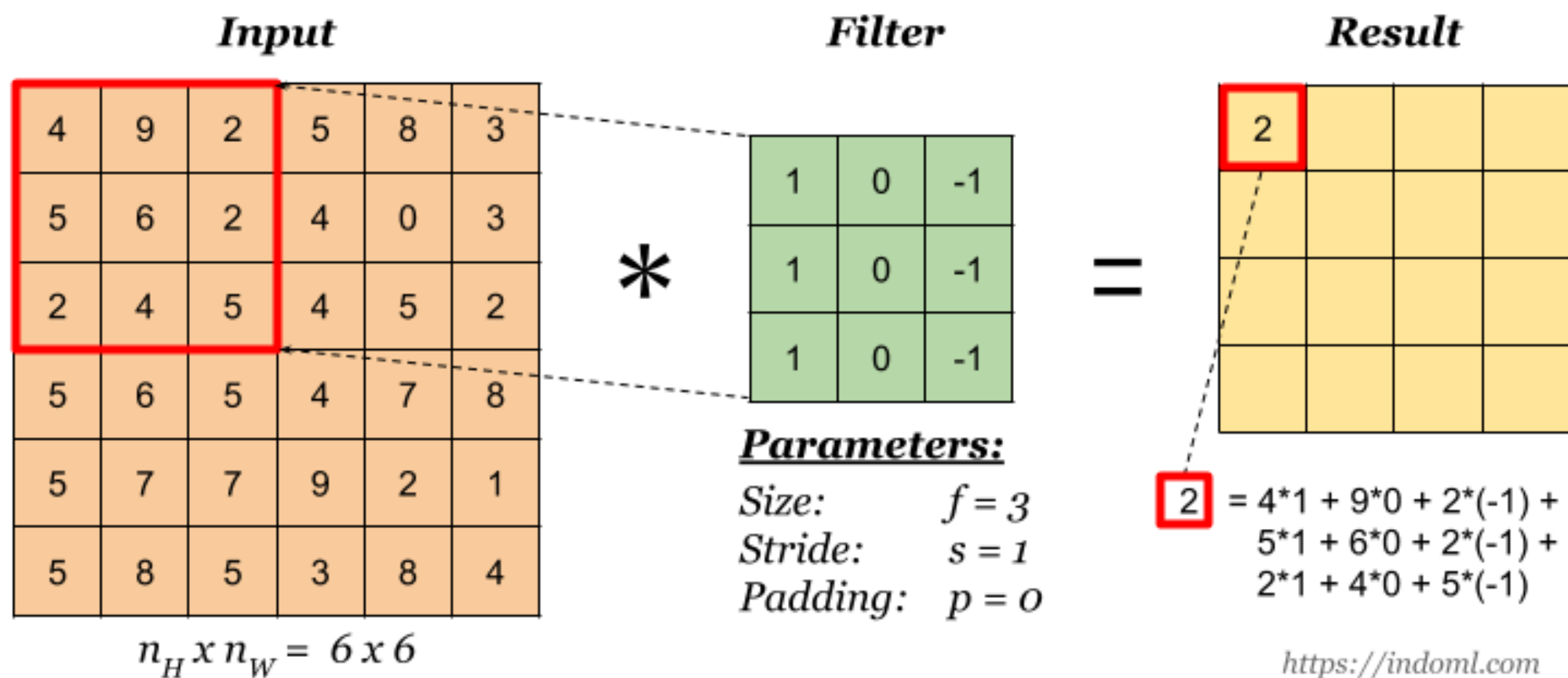
maintain the spatial
dimensions of the input
and output.

Output Size = $n \times n$

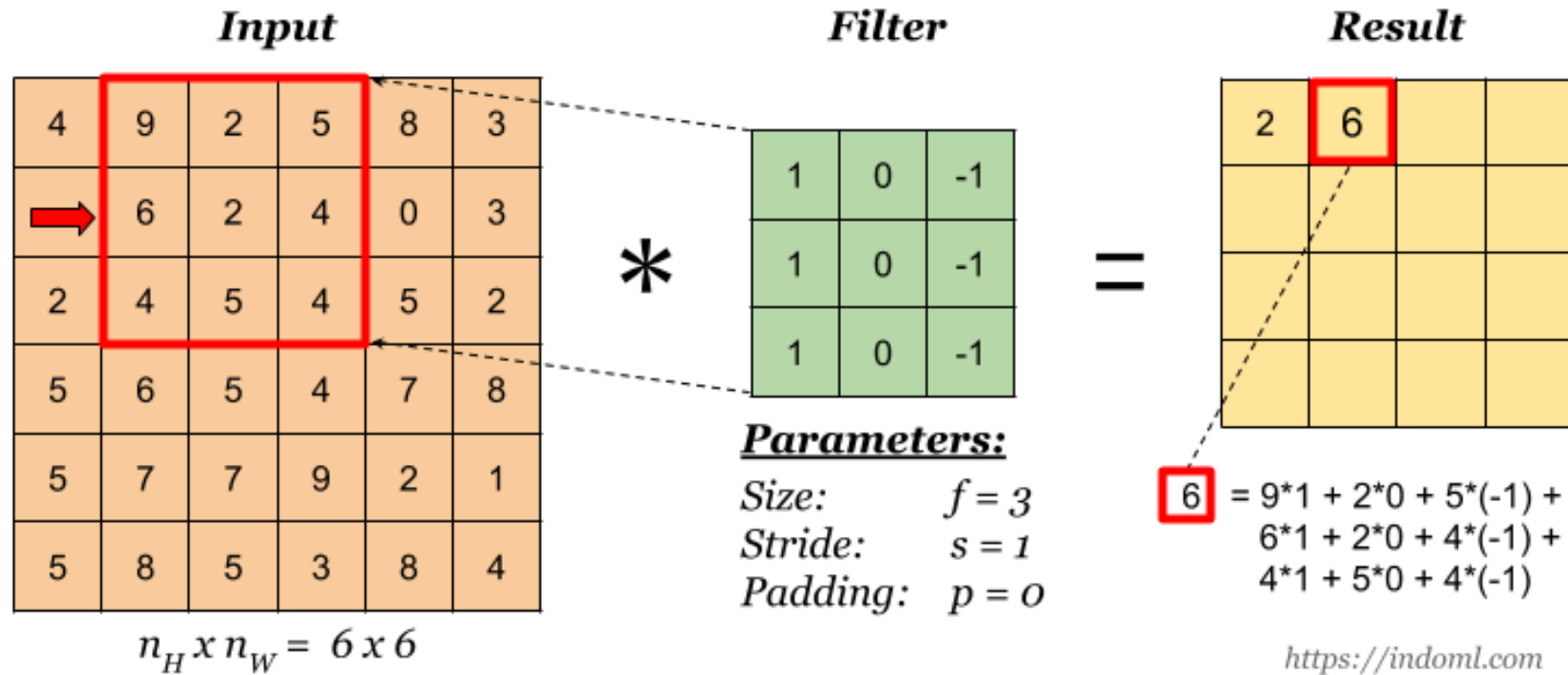
After
convolution:

0	-4	0	-3	-1
-2	-7	-2	-4	1
-2	-5	-2	-5	-2
-2	-7	-2	-5	0
0	-4	0	-4	0

Step 1: overlay the filter to the input, perform element wise multiplication, and add the result.



Step 2: move the overlay right one position (or according to the **stride** setting), and do the same calculation above to get the next result. And so on.

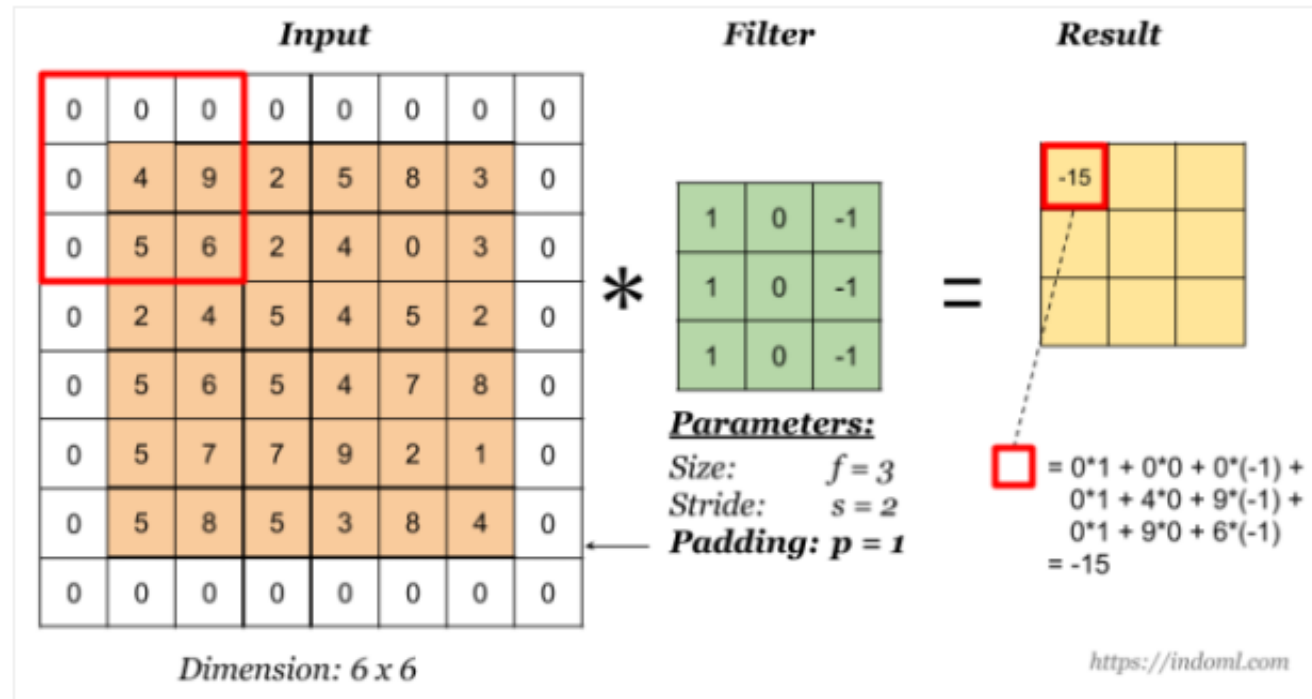


The total number of multiplications to calculate the result above is $(4 \times 4) \times (3 \times 3) = 144$.

Padding

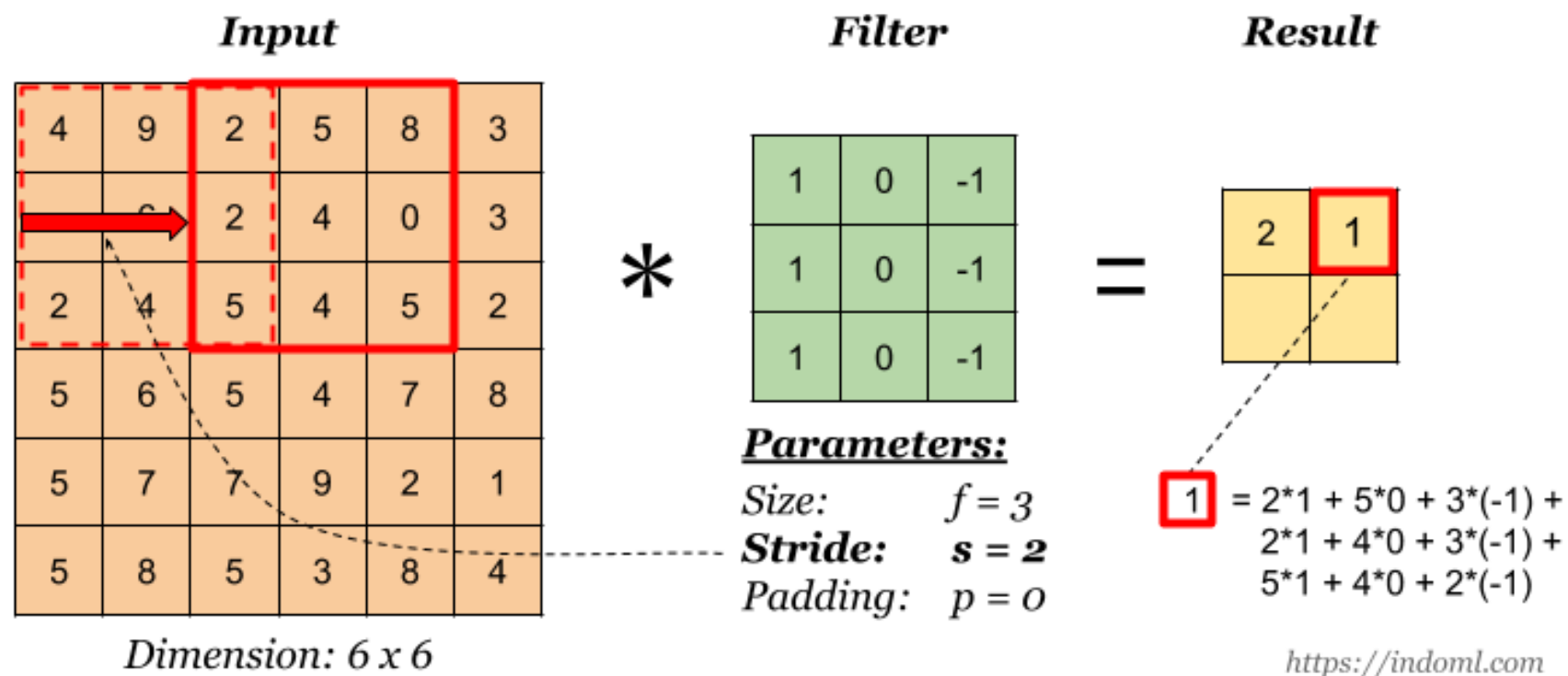
Padding has the following benefits:

1. It allows us to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.
2. It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.



Stride

Stride governs how many cells the filter is moved in the input to calculate the next cell in the result.



The total number of multiplications to calculate the result above is $(2 \times 2) \times (3 \times 3) = 36$.

Summary of convolutions

$n \times n$ image $f \times f$ filter

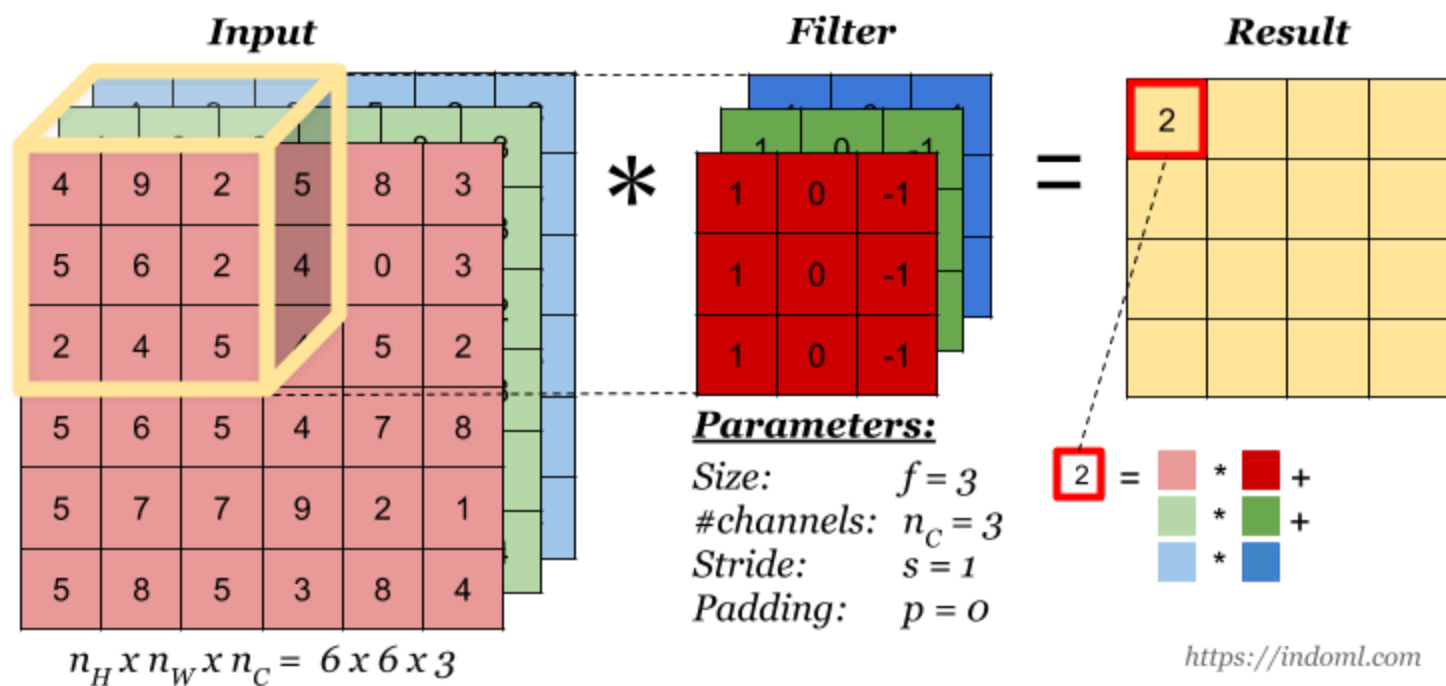
padding p stride s

Output Size:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

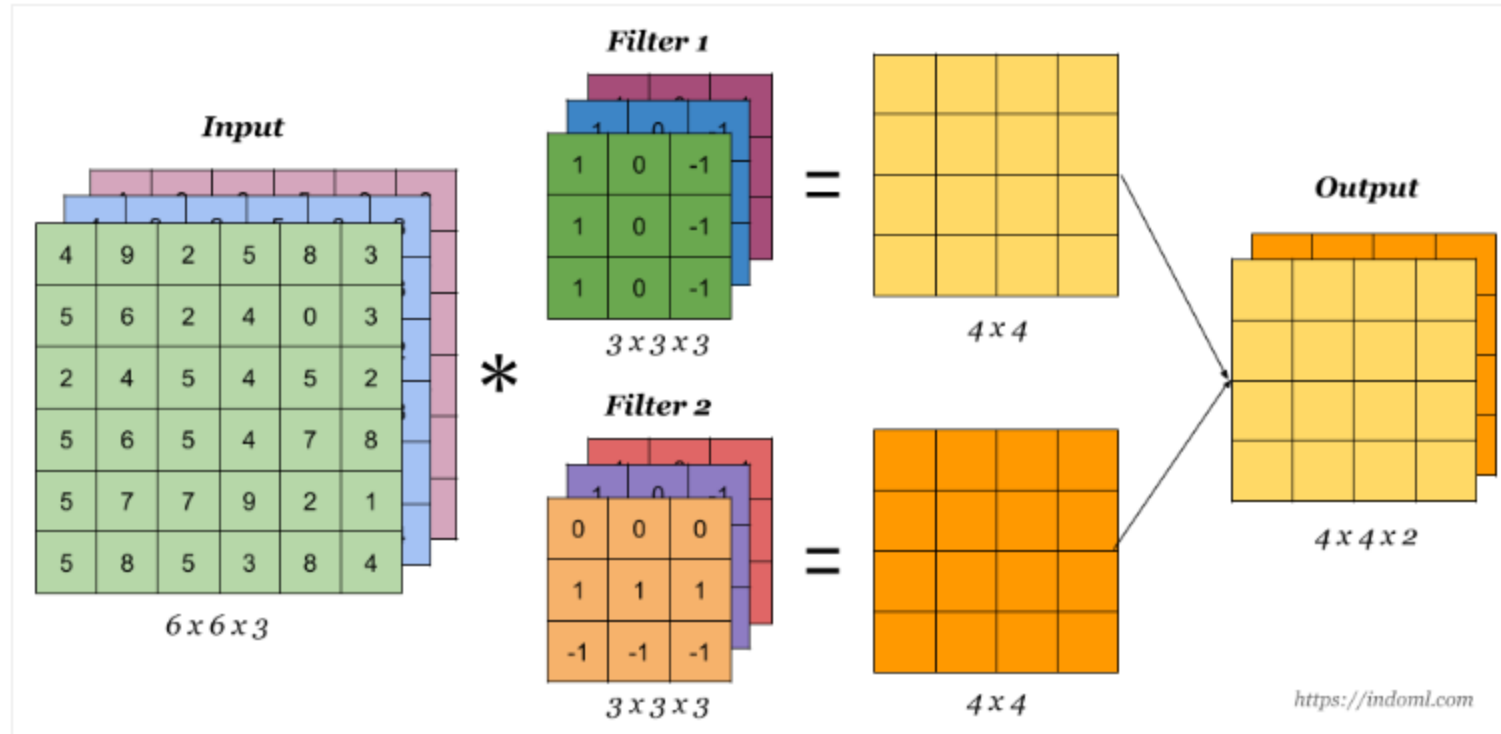
Convolution Operation on Volume

When the input has more than one channels (e.g. an RGB image), the filter should have matching number of channels. To calculate one output cell, perform convolution on each matching channel, then add the result together.



Convolution Operation with Multiple Filters

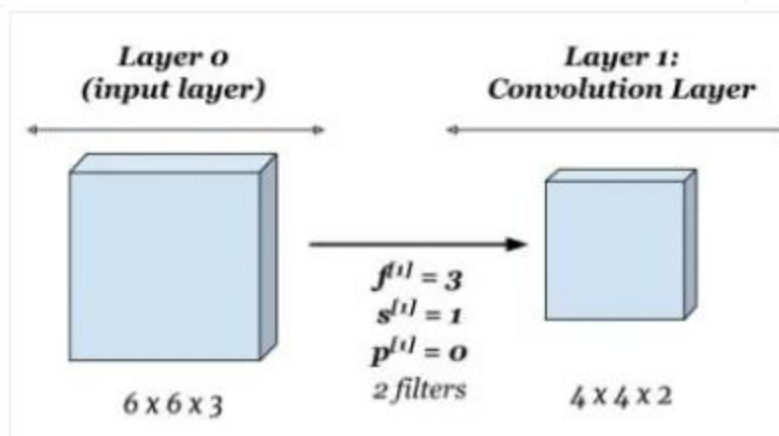
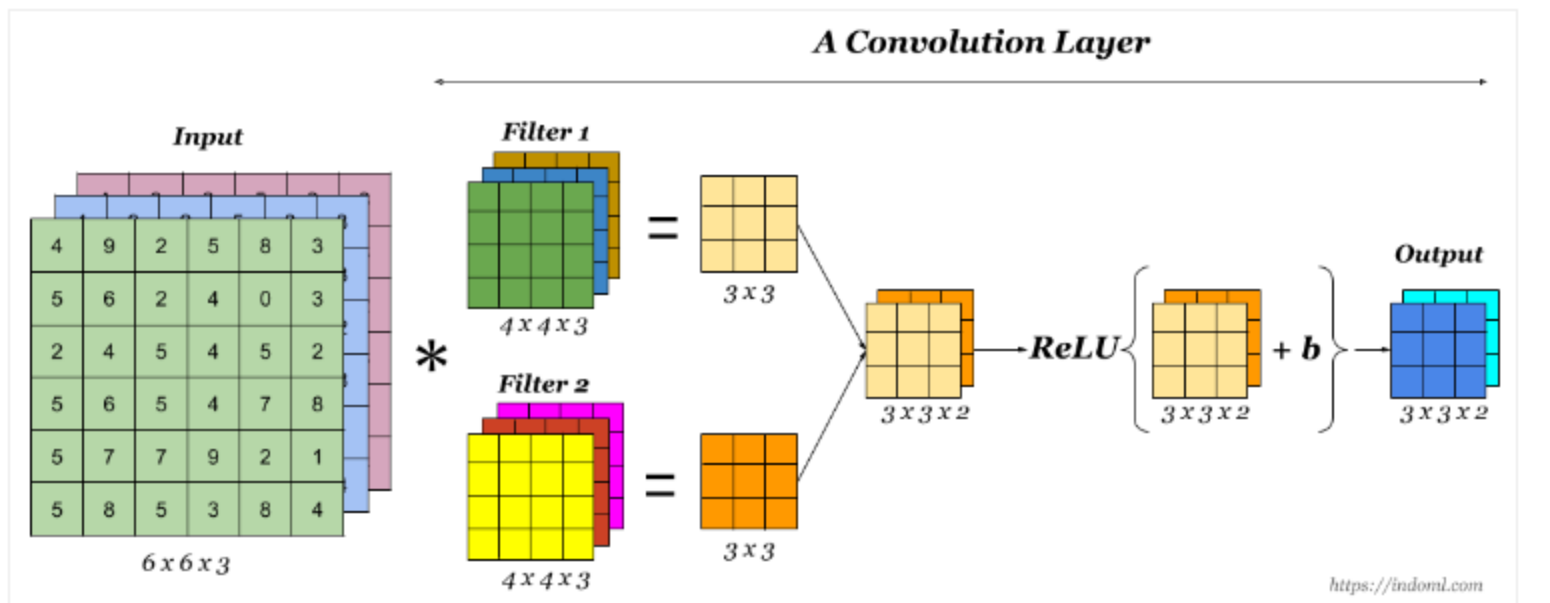
Multiple filters can be used in a convolution layer to detect multiple features. The output of the layer then will have the same number of channels as the number of filters in the layer.



The total number of multiplications to calculate the result is $(4 \times 4 \times 2) \times (3 \times 3 \times 3) = 864$.

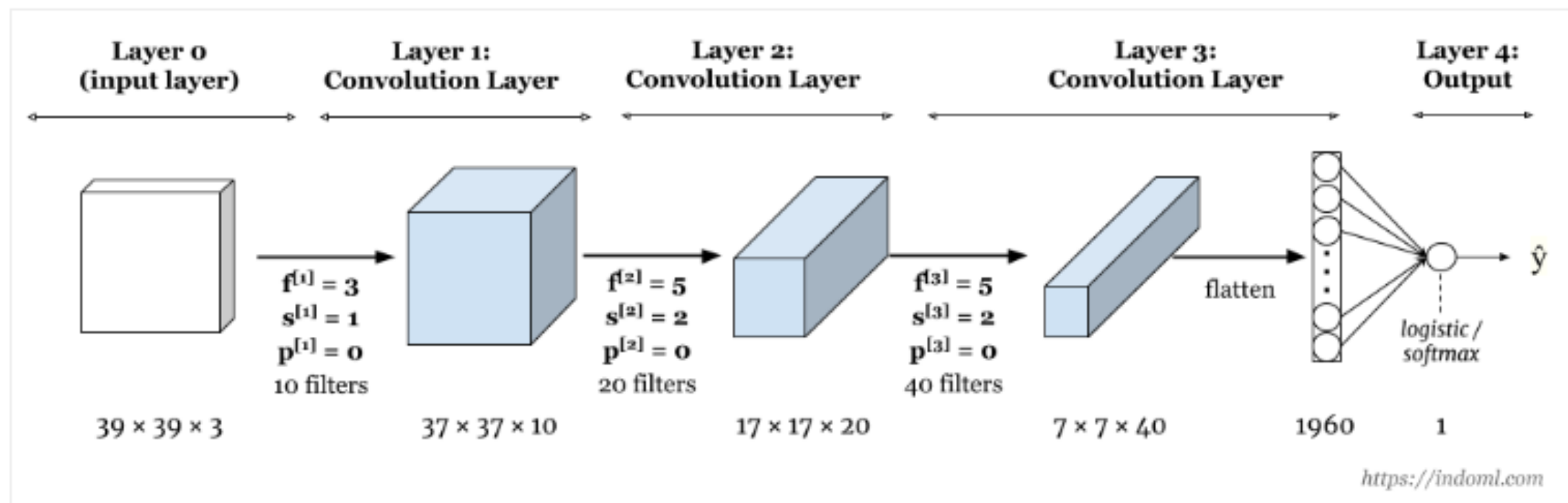
One Convolution Layer

Finally to make up a convolution layer, a bias ($\in \mathbb{R}$) is added and an activation function such as **ReLU** or **tanh** is applied.



Sample Complete Network

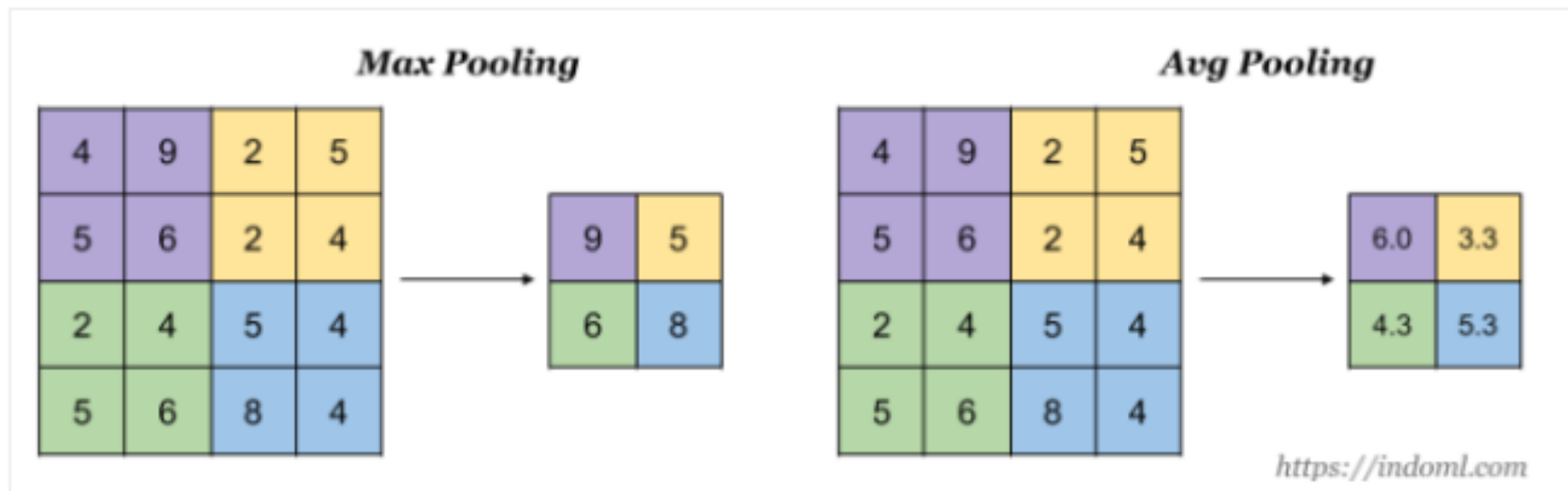
This is a sample network with three convolution layers. At the end of the network, the output of the convolution layer is flattened and is connected to a logistic regression or a softmax output layer.

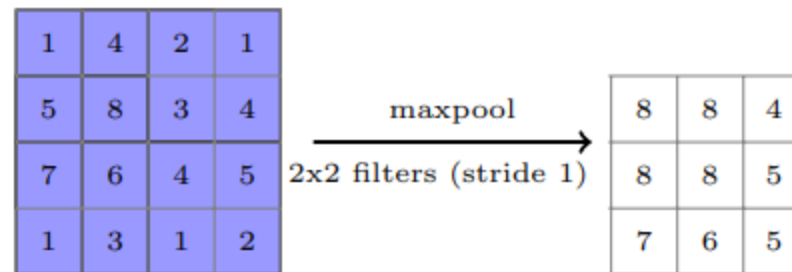
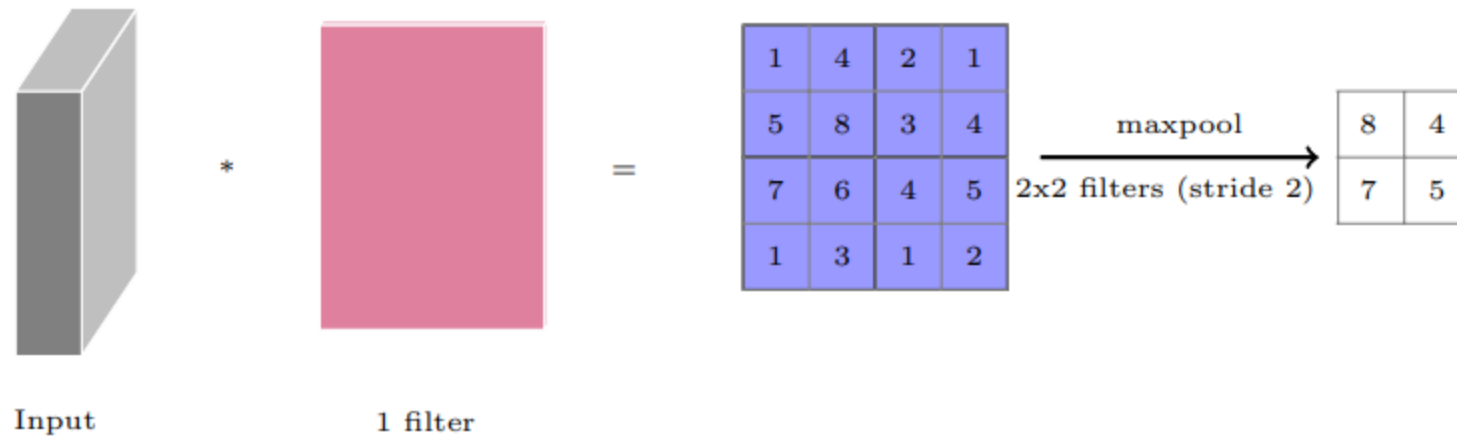


Pooling Layer

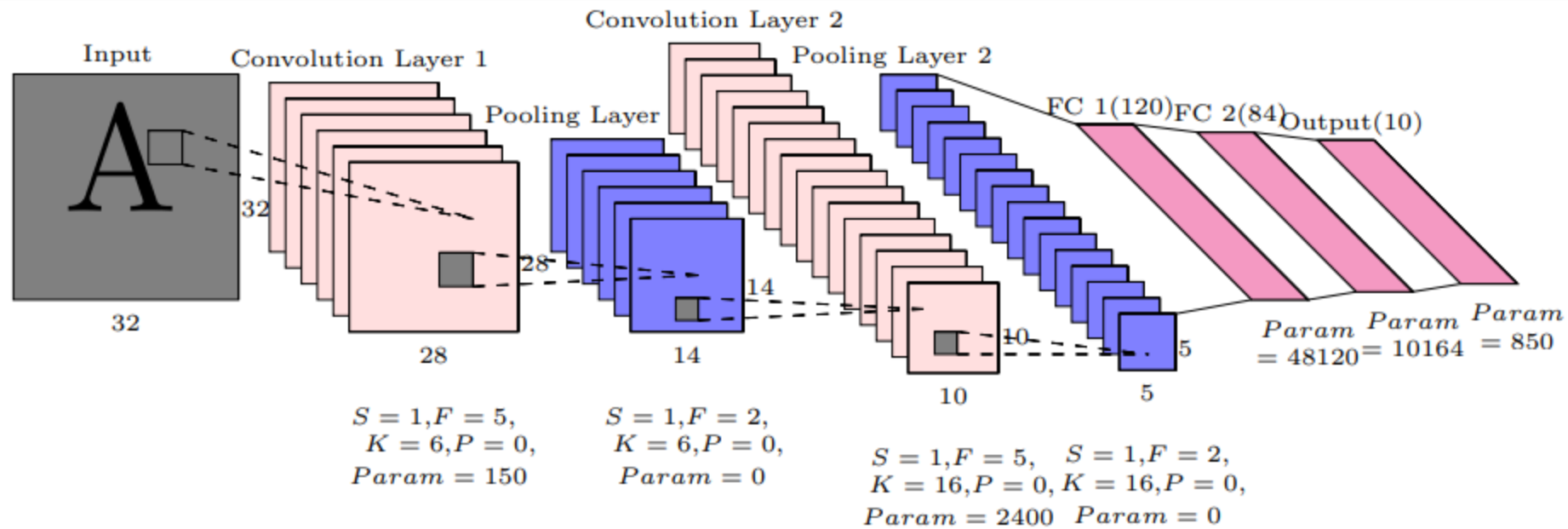
Pooling layer is used to reduce the size of the representations and to speed up calculations, as well as to make some of the features it detects a bit more robust.

Sample types of pooling are **max pooling** and **avg pooling**, but these days max pooling is more common.





- Instead of max pooling we can also do average pooling



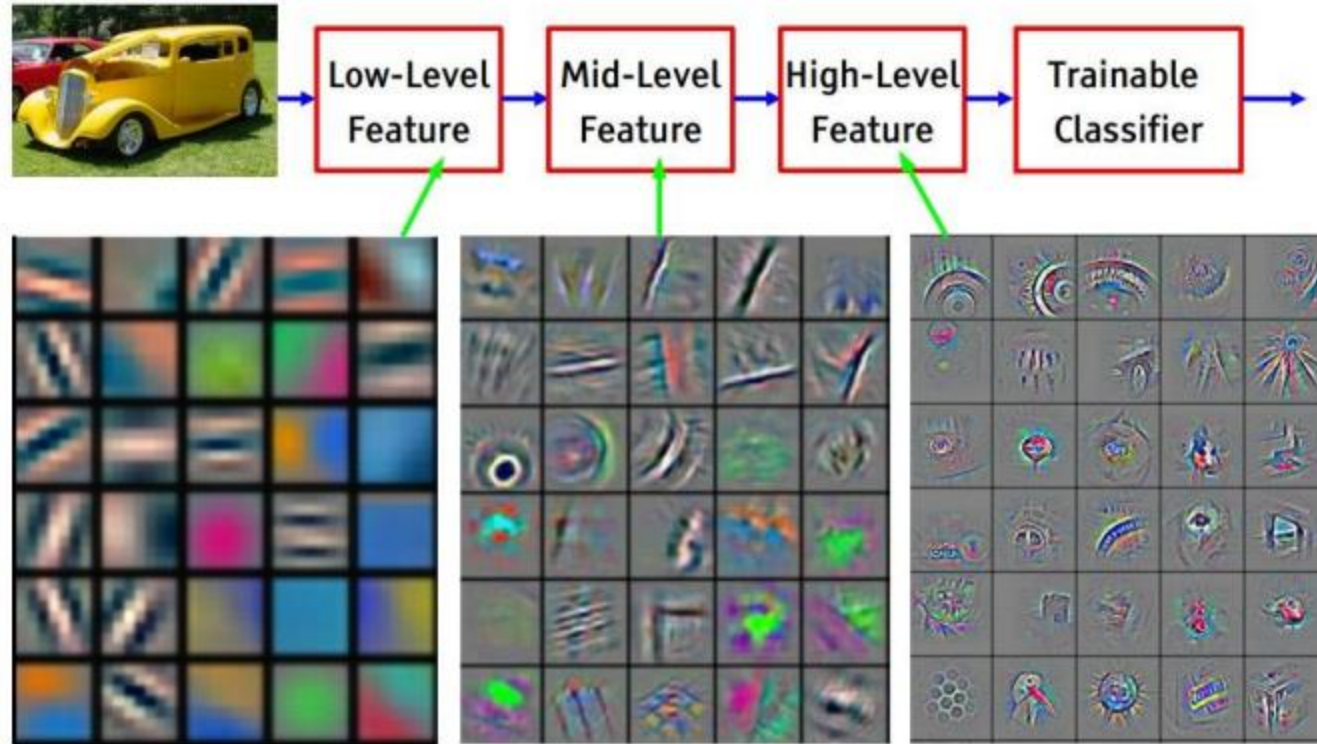
https://openlearninglibrary.mit.edu/assets/courseware/v1/cda92ed2c6672271916e8cb8974af568/asset-v1:MITx+6.036+1T2019+type@asset+block/notes_conv_nets_slides.pdf

<https://github.com/ashishpatel26/Andrew-NG-Notes/blob/master/andrewng-p-4-convolutional-neural-network.md#foundations-of-cnns>

<https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>

Preview

*[From recent Yann
LeCun slides]*



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]