Convolutional Neural Network (CNN)

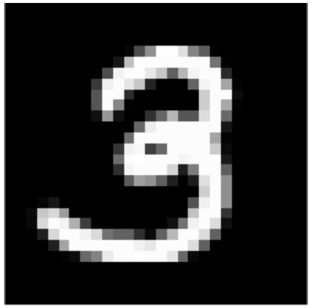
[Video: Deep Learning and Traditional Machine Learning: Choosing the Right Approach]

Why CNN?

If working with simple images, for example MINST data set

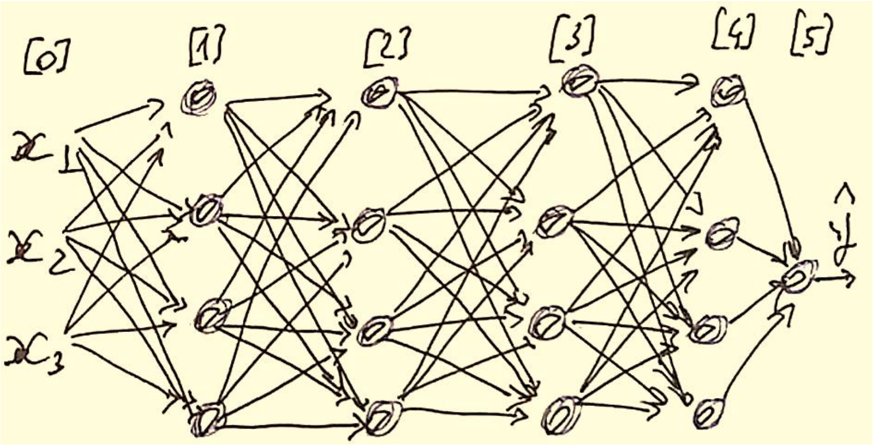
- 28 x 28 x 1 (b&w; 1 channel) = 784 features

http://yann.lecun.com/exdb/mnist/



The size of the input layer in a deep ANN is 784

– can be manageable.



25	43	11	04	70	87	12	31	43	10	05	77	12	06	45	09	29	30	02
56	22	75	03	22	96	45	12	23	03	77	67	81	45	22	04	90	22	21
32	45	41	91	87	62	35	02	00	11	62	25	43	11	04	70	87	12	61
31	43	10	05	77	12	06	45	09	29	30	56	22	75	03	22	96	45	05
12	23	03	77	67	81	45	22	04	90	22	32	45	41	91	87	62	35	44
02	00	11	62	25	43	11	04	70	87	12	31	43	10	05	77	12	06	10
45	09	29	30	56	22	75	03	22	96	45	12	23	03	77	67	81	45	55
22	04	90	22	32	45	41	91	87	62	35	02	00	11	62	25	43	11	80
04	70	87	12	31	43	10	05	77	12	06	45	09	29	30	56	22	75	08
03	22	96	45	12	23	03	77	67	81	45	22	04	90	22	32	45	41	99
91	87	62	35	02	00	11	62	22	01	00	72	65	23	01	00	22	04	30
90	22	32	45	41	91	87	62	35	02	00	11	62	25	43	11	04	70	42
87	12	31	43	10	05	77	12	06	45	09	29	30	56	22	75	03	22	91
96	45	12	23	03	77	67	81	45	22	04	90	22	32	45	41	91	87	40
62	35	02	00	11	62	22	01	00	72	65	23	01	00	56	22	75	03	67
22	96	45	12	23	03	77	67	81	45	22	04	90	22	32	45	41	91	22

What we see | **What computers see**

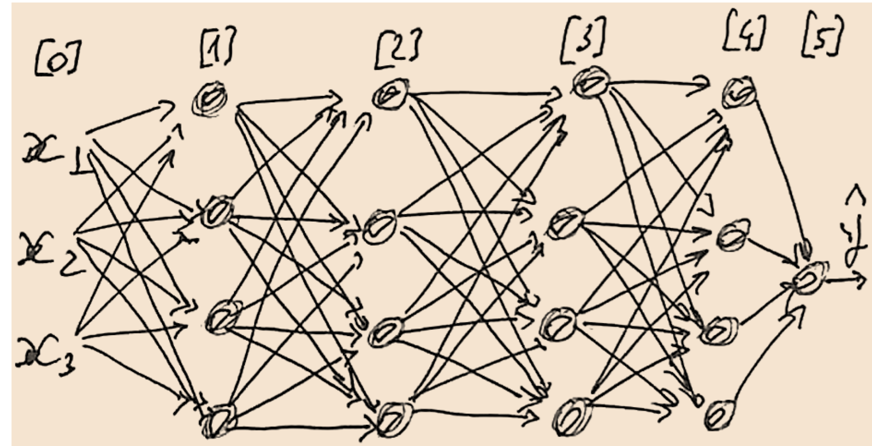
https://medium.com/intelligentmachines/convolutional-neural-network-and-regularization-techniques-with-tensorflow-and-keras-5a09e6e65dc7

Big challenge - dimensionality

If working with "real" images

512 x 512 x 3 (3 channels) = 786,432 features

Is this a lion?
(or a cat, a dog, etc.)



786,432 inputs

1000 hidden units

layer [1]: $W^{[1]}$: (1000; 786,432); **786,432,000 – weights + 1000 biases**

Way too many training parameters, especially in layer [1]

- Very difficult to get enough data to prevent overfitting
- Computation and memory requirements tend to be infeasible

Find a way to use far less parameters for the same problem!

CNN (ConvNet) for deep learning

- **CNN – Convolutional Neural Network**

- A class of deep neural networks, most applied to analyzing **visual imagery.**
- Applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing (NLP), text processing, etc

- Little pre-processing compared to other image classification algorithms.
- The network **learns the filters** that in traditional algorithms were hand-engineered (features extraction).
  - ✓ **This independence from prior knowledge and human effort in feature design and extraction is a major advantage**

CNN (ConvNet) for deep learning

- **CNN – Convolutional Neural Network**

- **CNNs eliminate the need for manual feature extraction**
  - no need to identify features used to classify images
- The CNN works by **extracting features directly from images**.
- The relevant features are not pretrained; they are learned while the network trains on a collection of images.

- The automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification/recognition.

CNN (ConvNet) for deep learning

- The network employs a mathematical operation called **convolution**.
- CNN are simply neural networks that **use convolution in place of general matrix multiplication** in at least one of their layers.
- CNN have learnable parameter like conventional neural network (weights, biases, etc.).

Convolution

- a specialized kind of linear operation
- a mathematical operation on two functions (*f* and *g*) that produces a third function expressing how the shape of one function (*f*) is modified by the other function (*g*).
- defined as the integral of the product of the two functions after one *(g)* is reversed and shifted.

Some intuition

Let’s think about how we recognize a face.

- ✓ We can recognize a face because it present a set of features: eyes, nose, ears, hair, etc.

- ✓ To decide if an object is a face, we do it as if we had some mental **boxes of verification of the features** that we are marking.

- ➢ Sometimes a face may not have an ear (it is covered by hair), but we still classify it with a certain probability as a face due to the presence of the other features.

- ➢ Actually, we can see it as a classifier that predicts a probability that the input image is a face or no face.

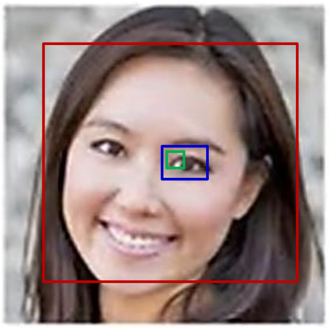
Some intuition

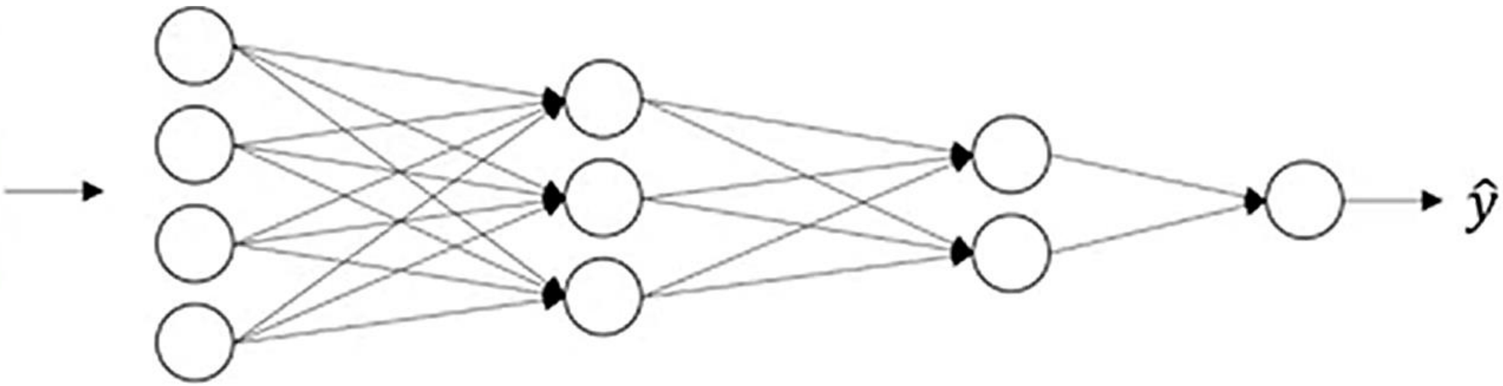
- In reality, we must first know what an eye or a nose is like:
  - ✓ we must previously identify lines, edges, textures or shapes that are like those containing the eyes or noses
    - this is what the layers of a convolutional neuronal network are entrusted to do.
- Identifying these elements is insufficient to say that an object is a face.
- We also must identify how the parts of a face meet each other, relative sizes, etc.; otherwise, the face would not resemble what we are used to.

  - In a convolutional neural network, each layer is learning different levels of abstraction.
  - With networks with many layers, it is possible to identify more complex structures in the input data.

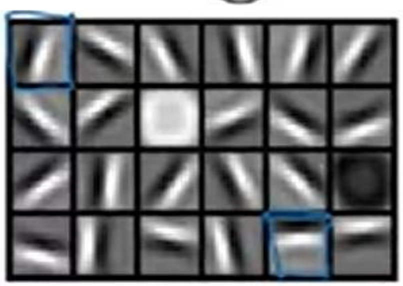
[Jordi TORRES.AI, Convolutional Neural Networks for Beginners using Keras & TensorFlow 2, Apr 22, 2020, https://towardsdatascience.com/convolutional-neural-networks-for-beginners-using-keras-and-tensorflow-2-c578f7b3bf25]

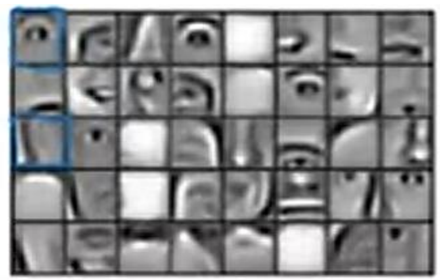
Convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships.
A first convolutional layer can learn basic elements such as edges.
A second convolutional layer can learn patterns composed of basic elements learned in the previous layer.
And so on until it learns very complex patterns.
This allows CNNs to efficiently learn increasingly complex and abstract visual concepts.





Pixels





Simple to complex hieratical representation

Feature detector
Edge detector - where are the edges? (groups pixels to form edges)

Take the detected edges; groups edges together to form part of faces (eye, nose, chin, etc)

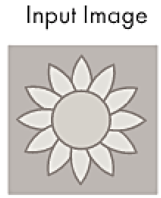
Putting together different parts of the faces to form faces

Simple things — Complex things

The complexity of the detected function increases (**edges** => **parts of faces** => **faces**)

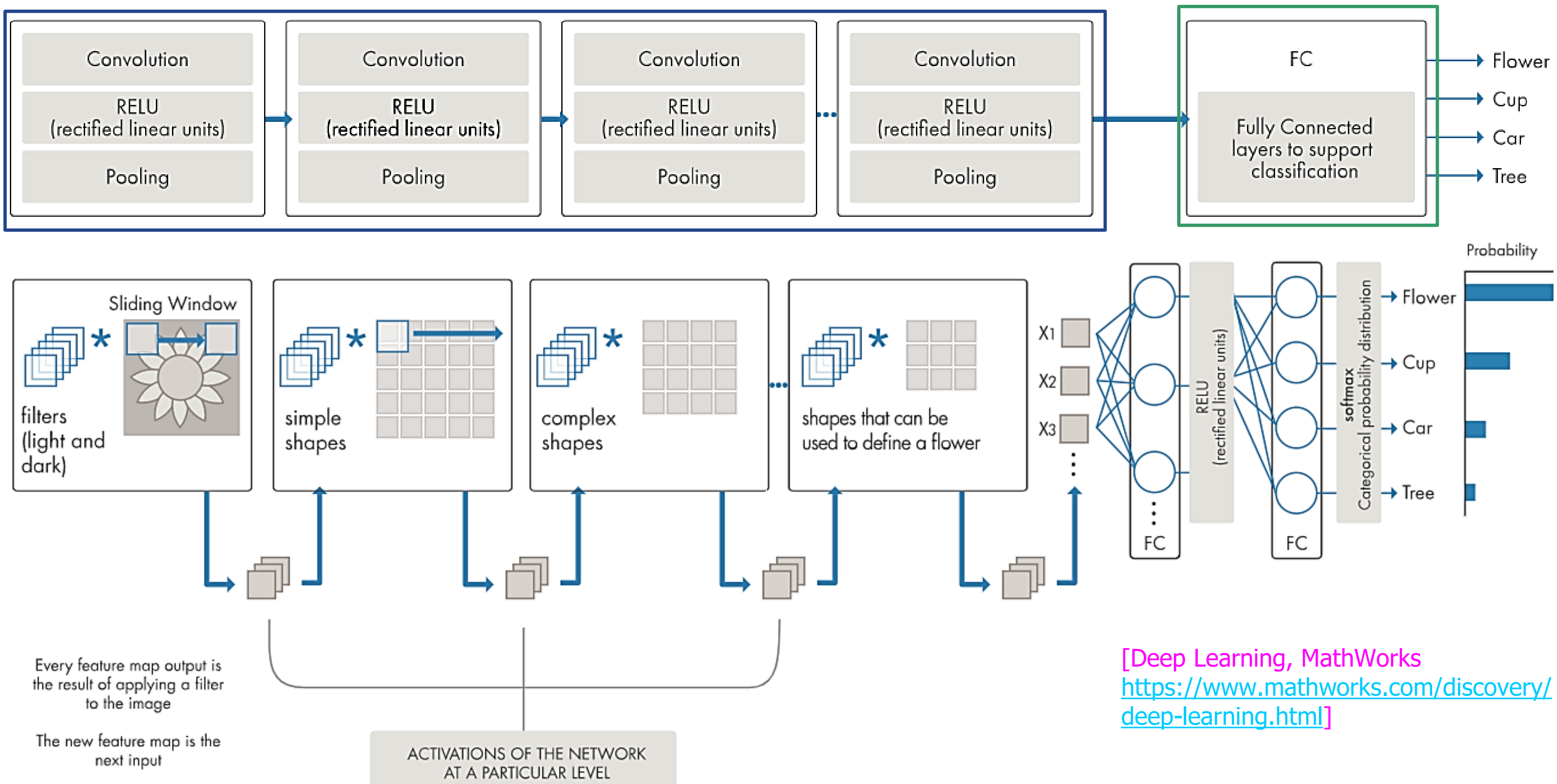
Very small window **Large window**

[Andrew Ng, Why deep representation?, https://www.coursera.org/lecture/neural-networks-deep-learning/why-deep-representations-rz9xJ]



CNN with many convolutional layers (deep)

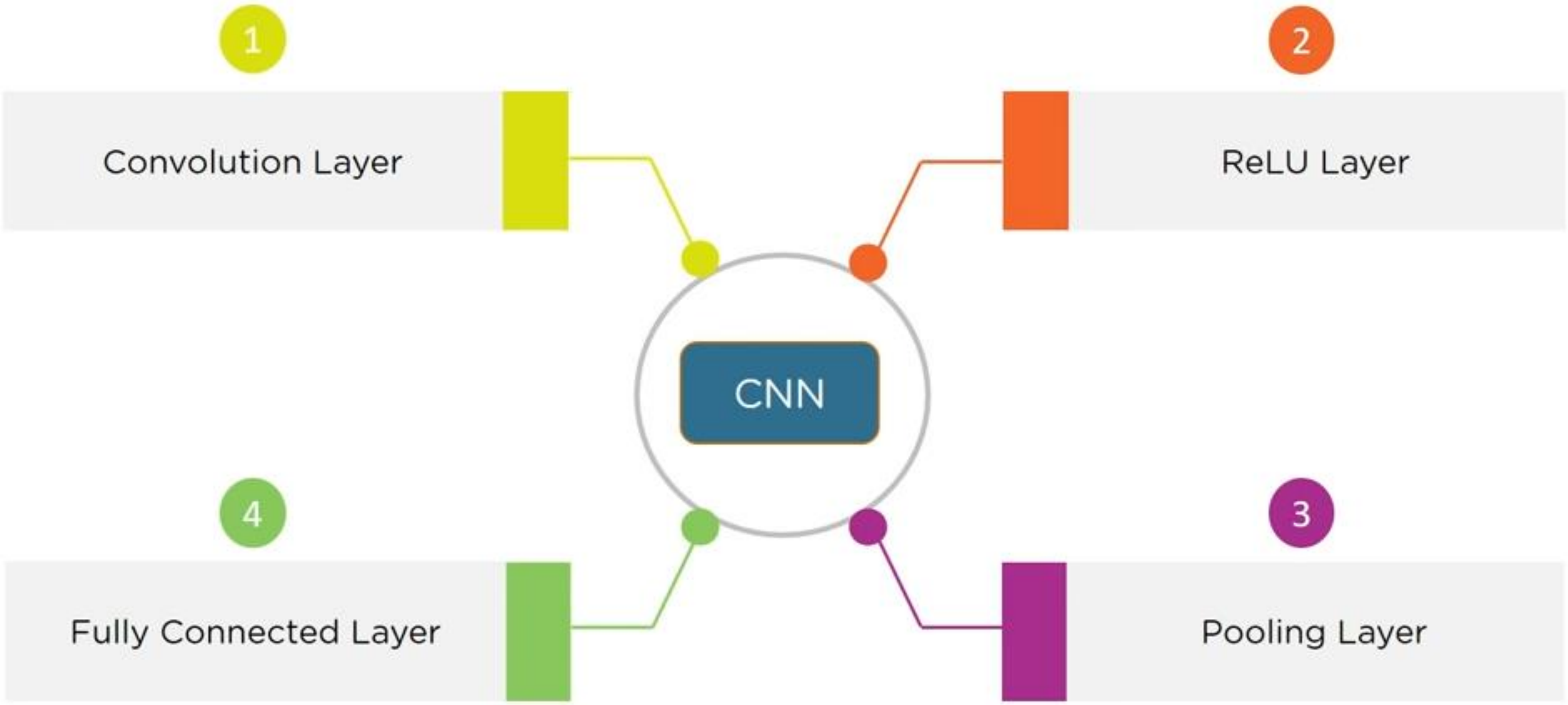
Filters are applied to each training image at different resolutions, and the output of each convolved image serves as the input to the next layer.



[Deep Learning, MathWorks https://www.mathworks.com/discovery/deep-learning.html]

CNNs learn to detect different features of an image using tens or hundreds of hidden layers.
Every hidden layer **increases the complexity** of the learned image features.
For example, the first hidden layer could learn how to detect edges, and the last learns how to detect more complex shapes specifically catered to the shape of the object we are trying to recognize.

Layers in CNN



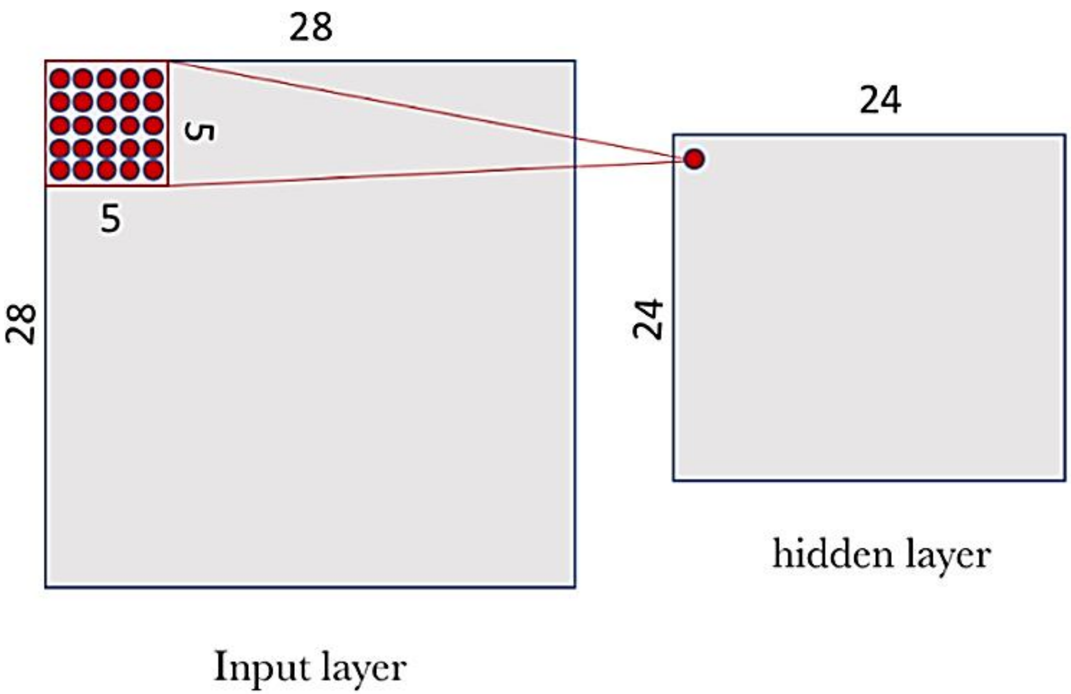
https://www.youtube.com/watch?v=Jy9-aGMB_TE

Convolution

Convolution is the process of adding each element (pixel) of the image to its local neighbors, weighted by a kernel (filter).

The **center** of the filter (kernel) is **aligned** with the **current pixel** and is a square with an odd number (3, 5, 7, etc.) of elements in each dimension.

E.g. kernel size: 5 x 5



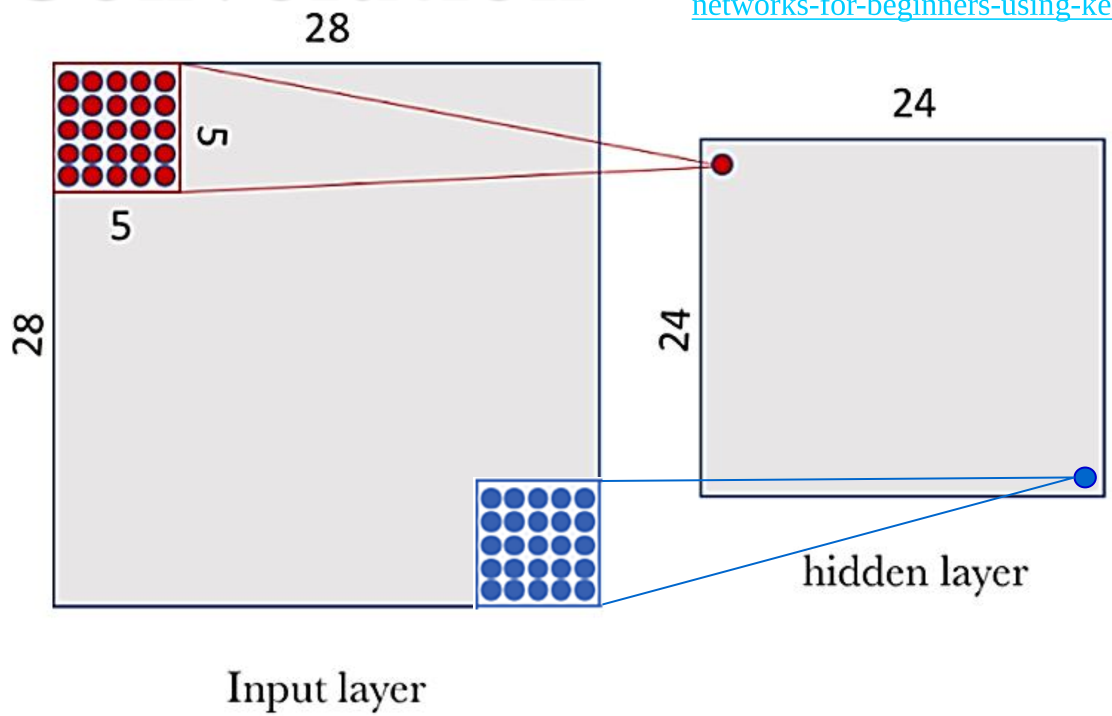
Each neuron in the hidden layer will be connected to a small region of 5×5 neurons (i.e. 25 neurons) of the input layer (28×28).

We can think of a 5×5 size **window that slides** along the entire 28×28 neuron layer of input that contains the image. For each position of the window there is a neuron in the hidden layer that processes this information.

[Jordi TORRES.AI, Convolutional Neural Networks for Beginners using Keras & TensorFlow 2, Apr 22, 2020, https://towardsdatascience.com/convolutional-neural-networks-for-beginners-using-keras-and-tensorflow-2-c578f7b3bf25]

Convolution

[Jordi TORRES.AI, Convolutional Neural Networks for Beginners using Keras & TensorFlow 2, Apr 22, 2020, https://towardsdatascience.com/convolutional-neural-networks-for-beginners-using-keras-and-tensorflow-2-c578f7b3bf25]



We start with the window in the top left corner of the image, and this gives the necessary information for the first neuron of the hidden layer.

Then, we slide the window one position to the right (stride =1) to "connect" the 5×5 neurons of the input layer included in this window with the second neuron of the hidden layer.

And so, successively, we go through the entire space of the input layer, from left to right and top to bottom.

For convolution:
25 weights in a *W* matrix (kernel)
1 bias values

In total 26 parameters.

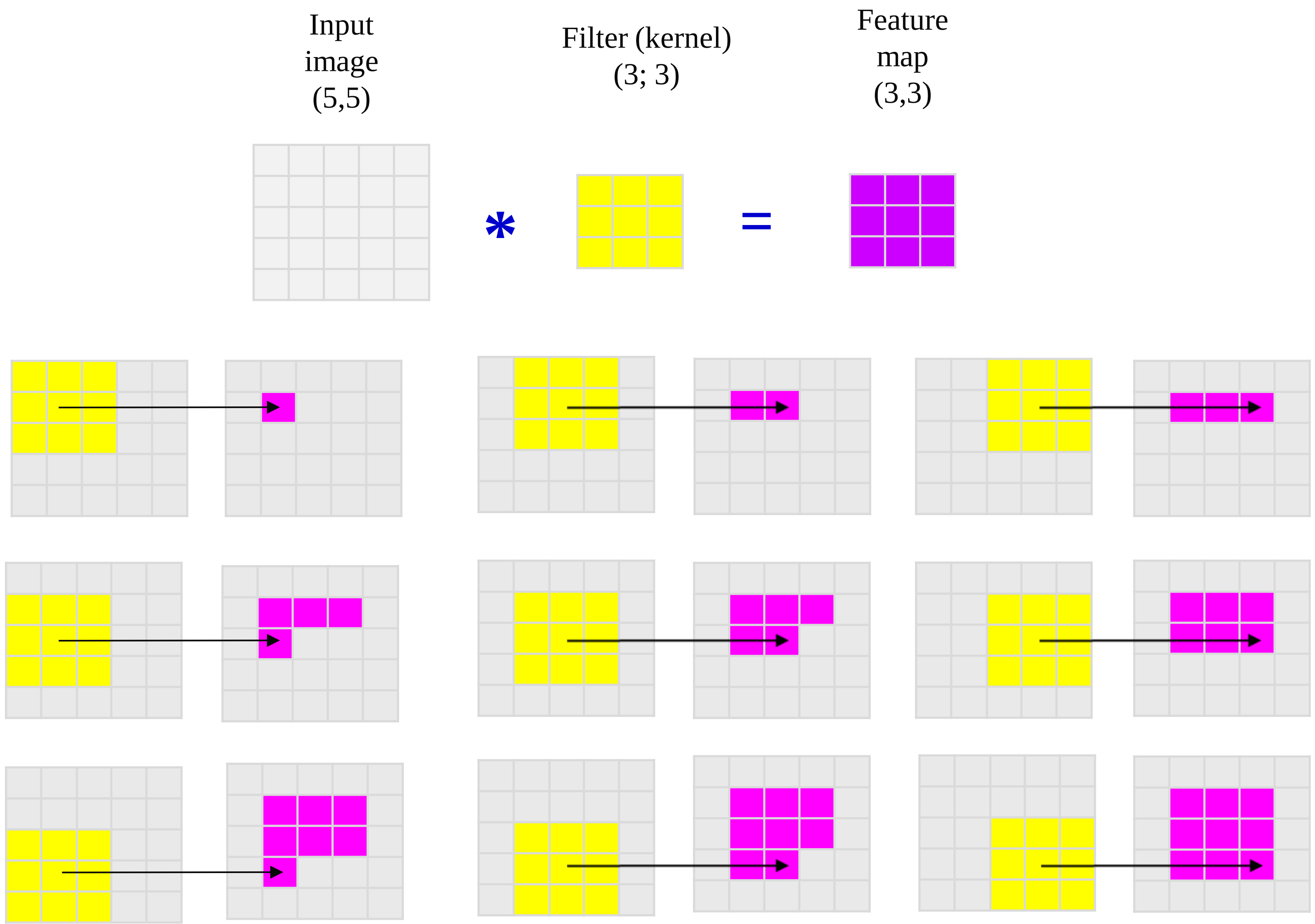
For a conventional ANN (not fully connected)
14,400 = (24x24) x (5x5) weights in a *W* matrix
576 = 24 x 24 bias values

In total 14,976 parameters.

Each neuron in layer *l* is connected only with 5x5 neurons in the *l*-1 layer (as is the case for convolution)

Drastically reduces the number of parameters

Illustration for input size: (5,5); filter (kernel) size: (3,3), stride = (1,1)



Convolution – edge (feature) detection in CNN

- To detect edges in complicated images, we may want vertical, horizontal, different degree edges, or even more complex ones
  - It makes almost impossible for a researcher to figure out the most appropriate filter (that 25 numeric values in a 5 x 5 filter)
- What about **learning the filter as parameters (using backpropagation):**

w_{11}	w_{12}	w_{13}	w_{14}	w_{15}
w_{21}	w_{22}	w_{23}	w_{24}	w_{25}
w_{31}	w_{32}	w_{33}	w_{34}	w_{35}
w_{41}	w_{42}	w_{43}	w_{44}	w_{45}
w_{51}	w_{52}	w_{53}	w_{54}	w_{55}

- The filter can learn from data to detect (extract) interesting low-level feature from the input image
- Very powerful idea in computer vision

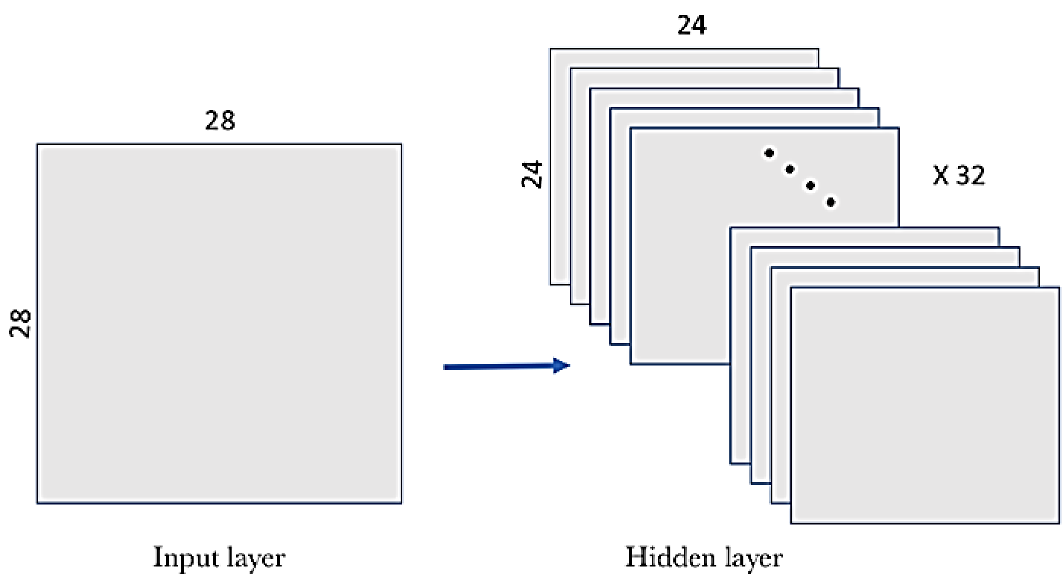
Convolution layers use different filters to be able to identify different aspects in an image: edges, corners, body parts (eyes, ear, paw, fur, etc.)
The **filters** (the weights and biases) **are learned** during the training process

One filter defined by one matrix ***W*** and one bias ***b*** only allows detecting a specific characteristic **(one characteristic)** in an image.

To perform image recognition, it is necessary to use **several filters** at the same time, to extract **several characteristics** in the same convolutional layer.

A complete convolutional layer in a convolutional neuronal network includes several filters.

E.g.: using 32 filters (one filter for each characteristic), we can extract 32 different characteristics at once, for the same input layer



[Jordi TORRES.AI, Convolutional Neural Networks for Beginners using Keras & TensorFlow 2, Apr 22, 2020, https://towardsdatascience.com/convolutional-neural-networks-for-beginners-using-keras-and-tensorflow-2-c578f7b3bf25]

Image filtering is useful for many applications, including smoothing, sharpening, removing noise, and edge detection.

A filter is defined by a kernel, which is a small array applied to each pixel and its neighbors within an image.

The center of the kernel is aligned with the current pixel

- a square with an odd number (3, 5, 7, etc.) of elements in each dimension.

A **high pass filter** is the basis for most sharpening methods.
An image is sharpened when contrast is enhanced between adjoining areas with little variation in brightness or darkness.

A high pass filter tends to retain the high frequency information within an image while reducing the low frequency information.

The kernel of the high pass filter is designed to increase the brightness of the center pixel relative to neighboring pixels.

The kernel array usually contains a single positive value at its center, which is surrounded by negative values.

high pass filters example:

$$\begin{bmatrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 8/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{bmatrix}$$

https://northstar-www.dartmouth.edu/doc/idl/html_6.2/Filtering_an_Imagehvr.html

Convolution / Cross correlation

Image matrix (function) *f*

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 2 & 0 \end{matrix}$$

Filter matrix (function) *g*

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

For **convolution**, the initial filter matrix *(g)* is initially flipped vertically and horizontally

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 2 & 0 \end{matrix} \ast \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} = \mathbf{26}$$

$$\begin{matrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{matrix} \qquad \begin{matrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{matrix}$$

9*1+8*0+7*0+
6*0+5*2+4*0+
3*1+2*2+1*0 = 26

For **cross-correlation**, the initial filter matrix *(g)* is used as it is

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 2 & 0 \end{matrix} \ast \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} = \mathbf{34}$$

1*1+2*0+3*0+
4*0+5*2+6*0+
7*1+8*2+9*0 = 34

By convention in machine learning /deep learning we will **use the term convolution for cross-correlation.**

Convolution – vertical edge detection

Original image (8, 8)		Filter (kernel) (3, 3)		Output image (6, 6)

$$
\begin{matrix}
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100\\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100\\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100\\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100\\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100\\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100\\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100\\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100
\end{matrix}
\; * \;
\begin{matrix}
-1 & 0 & 1\\
-1 & 0 & 1\\
-1 & 0 & 1
\end{matrix}
\; = \;
\begin{matrix}
0 & 0 & 270 & 270 & 0 & 0\\
0 & 0 & 270 & 270 & 0 & 0\\
0 & 0 & 270 & 270 & 0 & 0\\
0 & 0 & 270 & 270 & 0 & 0\\
0 & 0 & 270 & 270 & 0 & 0\\
0 & 0 & 270 & 270 & 0 & 0
\end{matrix}
$$

Dimensions

(n, n) $\quad (f, f)$ $\quad (n - f + 1, n - f + 1)$

$(8, 8)$ $\quad (3, 3)$ $\quad (8 - 3 + 1, 8 - 3 + 1)$

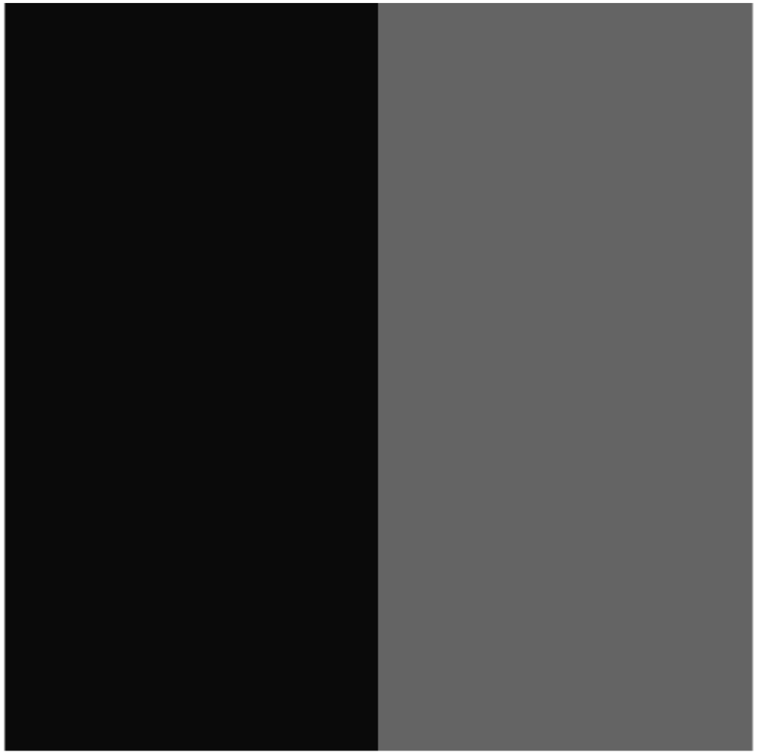
$(6, 6)$

At each edge of the original image, $\left(\frac{f}{2} - 0.5\right)$ pixels are lost.

In total $2 \cdot \left(\frac{f}{2} - 0.5\right)$, meaning $\boldsymbol{f - 1}$ **pixels are lost**, on each dimension.

Convolution – vertical edge detection

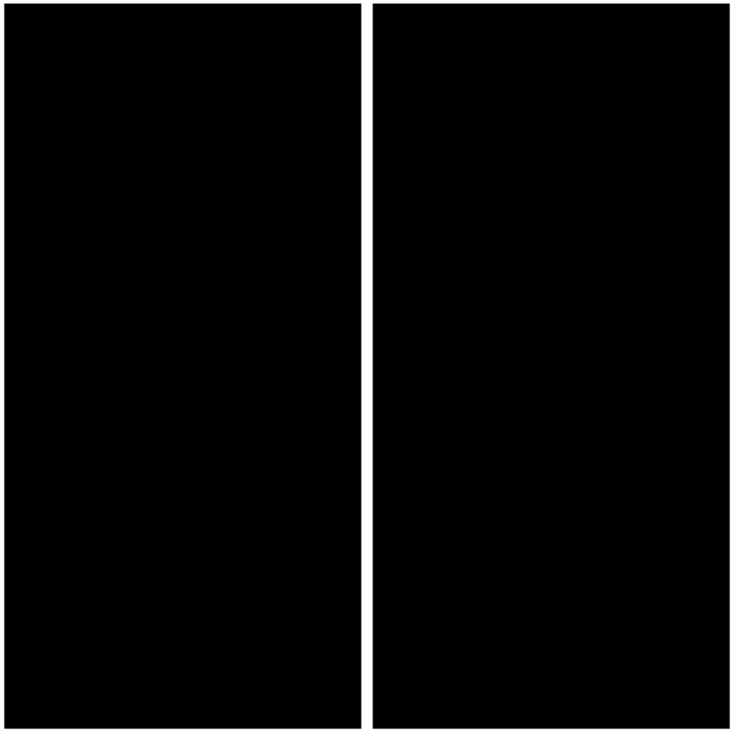
Original image
(128, 128)



Filter (kernel)
(3, 3)

$$* \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} =$$

Output image
(126, 126)



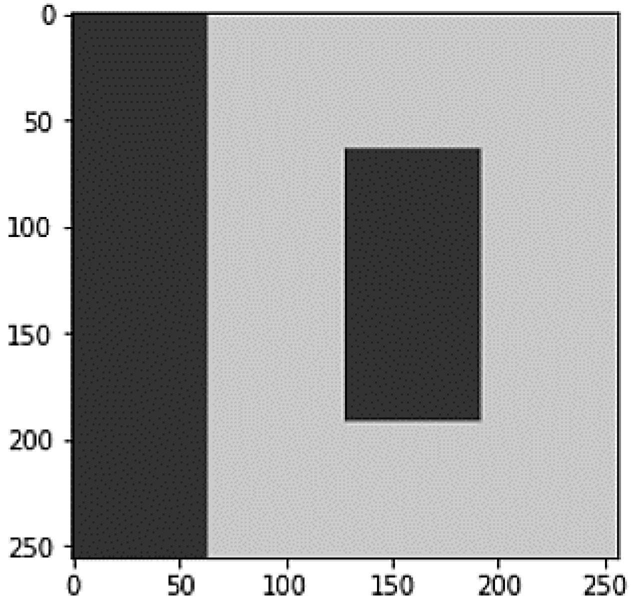
Dark to light transition

Convolution – vertical edge detection

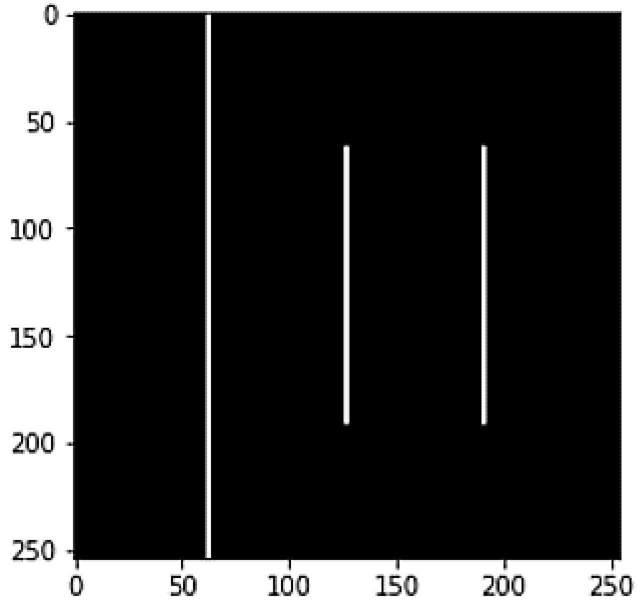
Original image
(256, 256)

Filter (kernel)
(3, 3)

Output image
(254, 254)



$$* \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} =$$



dark to light; light to dark transitions
on the horizontal direction

Convolution – vertical edge detection

$$
\begin{bmatrix}
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100
\end{bmatrix}
*
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1
\end{bmatrix}
=
\begin{bmatrix}
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0
\end{bmatrix}
$$

Sobel

$$
\begin{bmatrix}
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100
\end{bmatrix}
*
\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix}
=
\begin{bmatrix}
0 & 0 & 360 & 360 & 0 & 0 \\
0 & 0 & 360 & 360 & 0 & 0 \\
0 & 0 & 360 & 360 & 0 & 0 \\
0 & 0 & 360 & 360 & 0 & 0 \\
0 & 0 & 360 & 360 & 0 & 0 \\
0 & 0 & 360 & 360 & 0 & 0
\end{bmatrix}
$$

Scharr

$$
\begin{bmatrix}
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100
\end{bmatrix}
*
\begin{bmatrix}
-3 & 0 & 3 \\
-10 & 0 & 10 \\
-3 & 0 & 3
\end{bmatrix}
=
\begin{bmatrix}
0 & 0 & 1440 & 1440 & 0 & 0 \\
0 & 0 & 1440 & 1440 & 0 & 0 \\
0 & 0 & 1440 & 1440 & 0 & 0 \\
0 & 0 & 1440 & 1440 & 0 & 0 \\
0 & 0 & 1440 & 1440 & 0 & 0 \\
0 & 0 & 1440 & 1440 & 0 & 0
\end{bmatrix}
$$

Convolution – horizontal edge detection

Original image
(8, 8)

10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100

*

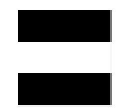
Filter (kernel)
(3, 3)

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

=

Output image
(6, 6)
no padding

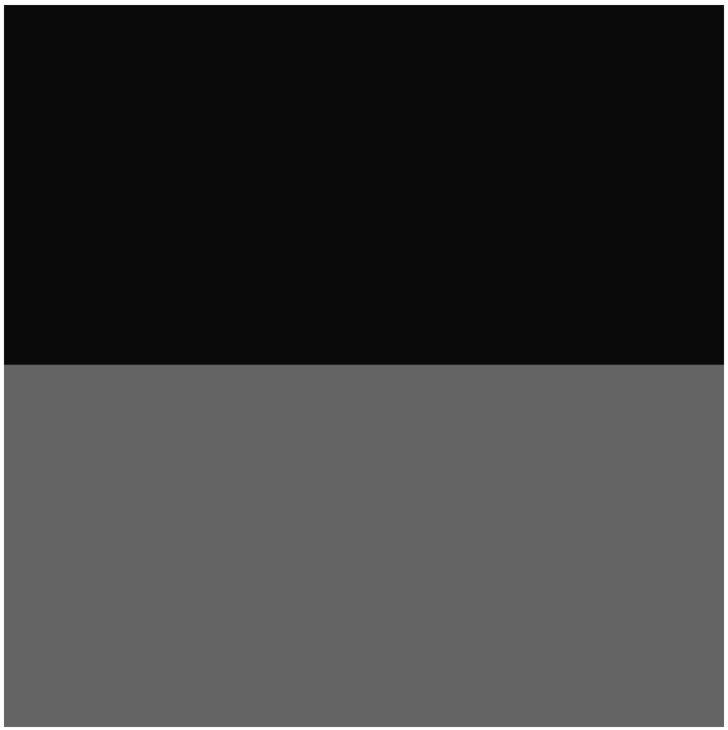
0	0	0	0	0	0
0	0	0	0	0	0
270	270	270	270	270	270
270	270	270	270	270	270
0	0	0	0	0	0
0	0	0	0	0	0



dark to light; light to dark transitions
on the vertical direction

Convolution – horizontal edge detection

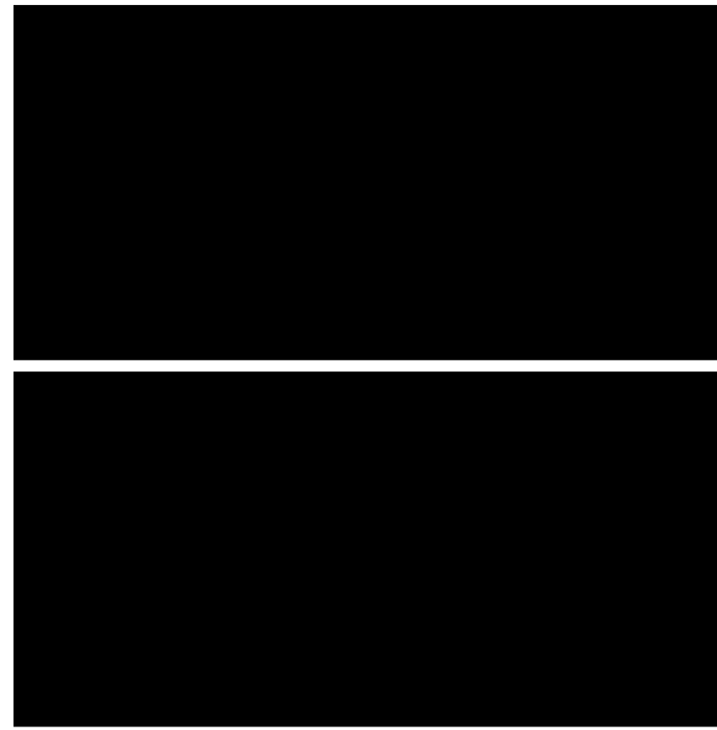
Original image
(128, 128)



Filter (kernel)
(3, 3)

$$* \begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix} =$$

Output image
(126, 126)
no padding

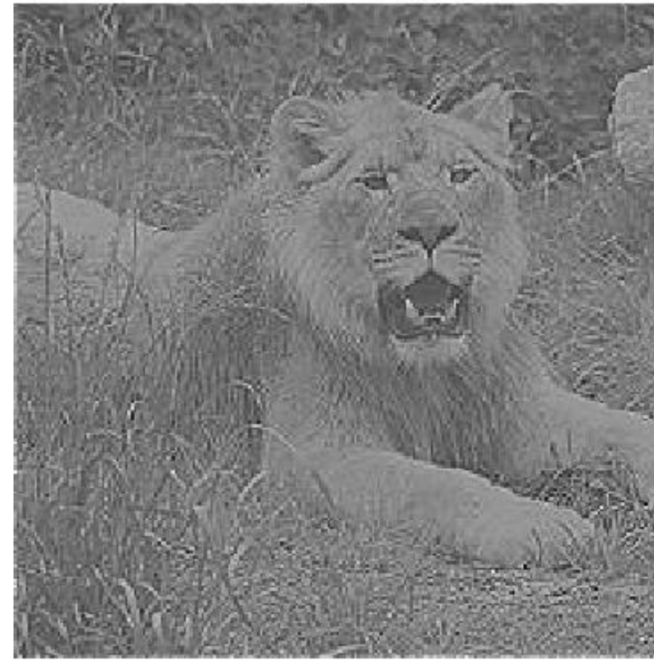


Dark to light transition

Sharpening filter

Original gray image

Sharpen image



stride = 1

$*$

0	-1	0
-1	5	-1
0	-1	0

$=$

Original gray image - histogram equalization

Sharpen image - histogram equalization

Adjust the contrast of the image by applying Histogram Equalization

Edge detection filter

Original gray image

stride = 1

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

$*$ ... $=$

Edge detection image

Original gray image - histogram equalization

Edge detection image - histogram equalization

Original gray image

Edge detection filter

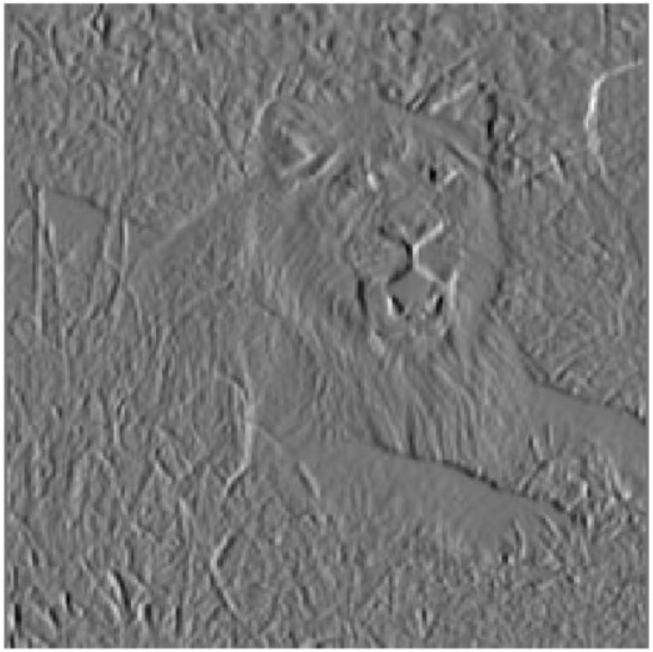
stride = 1

*

-1	0	+1
-2	0	+2
-1	0	+1

=

Edge detection image



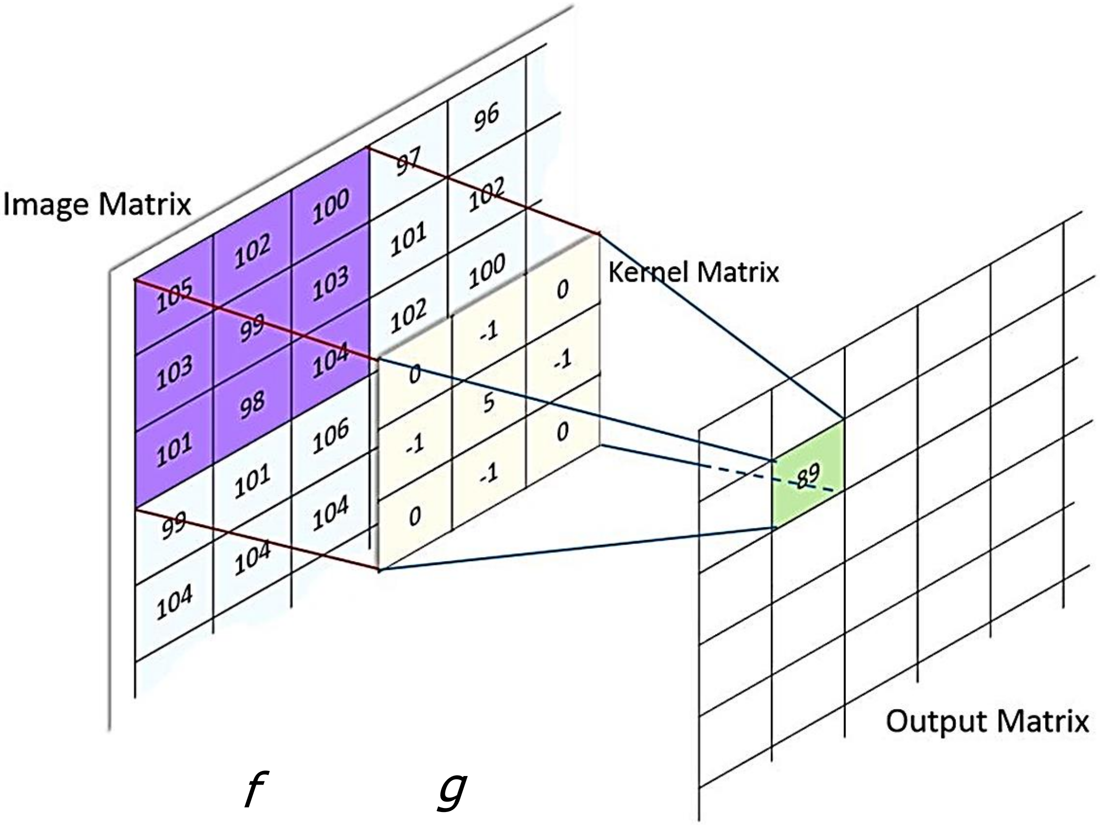
Original gray image - histogram equalization

Sobel; horizontal changes (vertical edges)

Edge detection image - histogram equalization

Convolution

http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html



Sharpening filter (kernel)

Sharpening an image increases the contrast between bright and dark regions to bring out features.

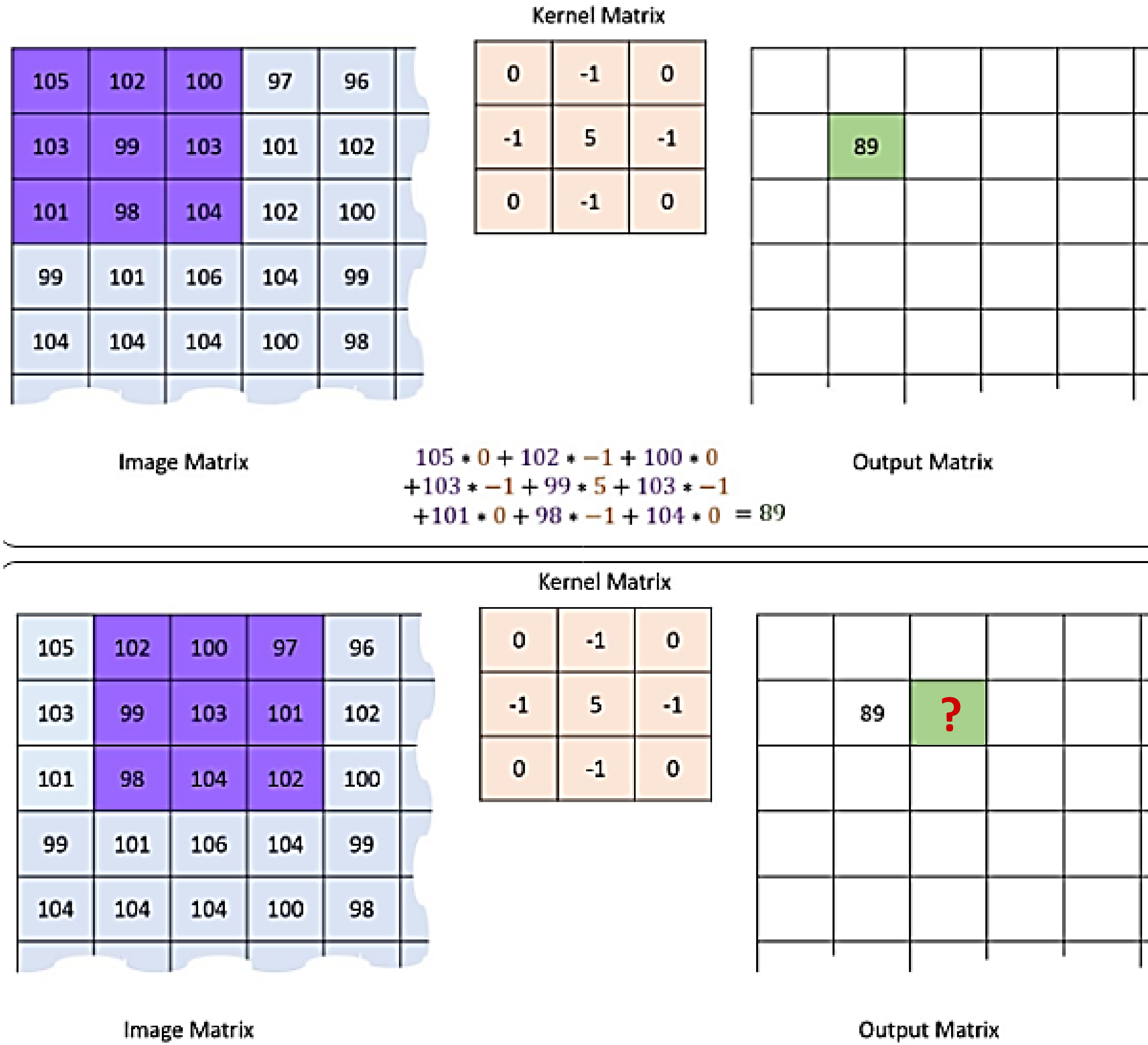
The sharpening process is basically the application of a **high pass filter** to an image.

$f * g$:

105*0 + 102*(-1) + 100*0 + 103*(-1) + 99*5 + 103*(-1) + 101*0 + 98*(-1) + 104*0 = **89**

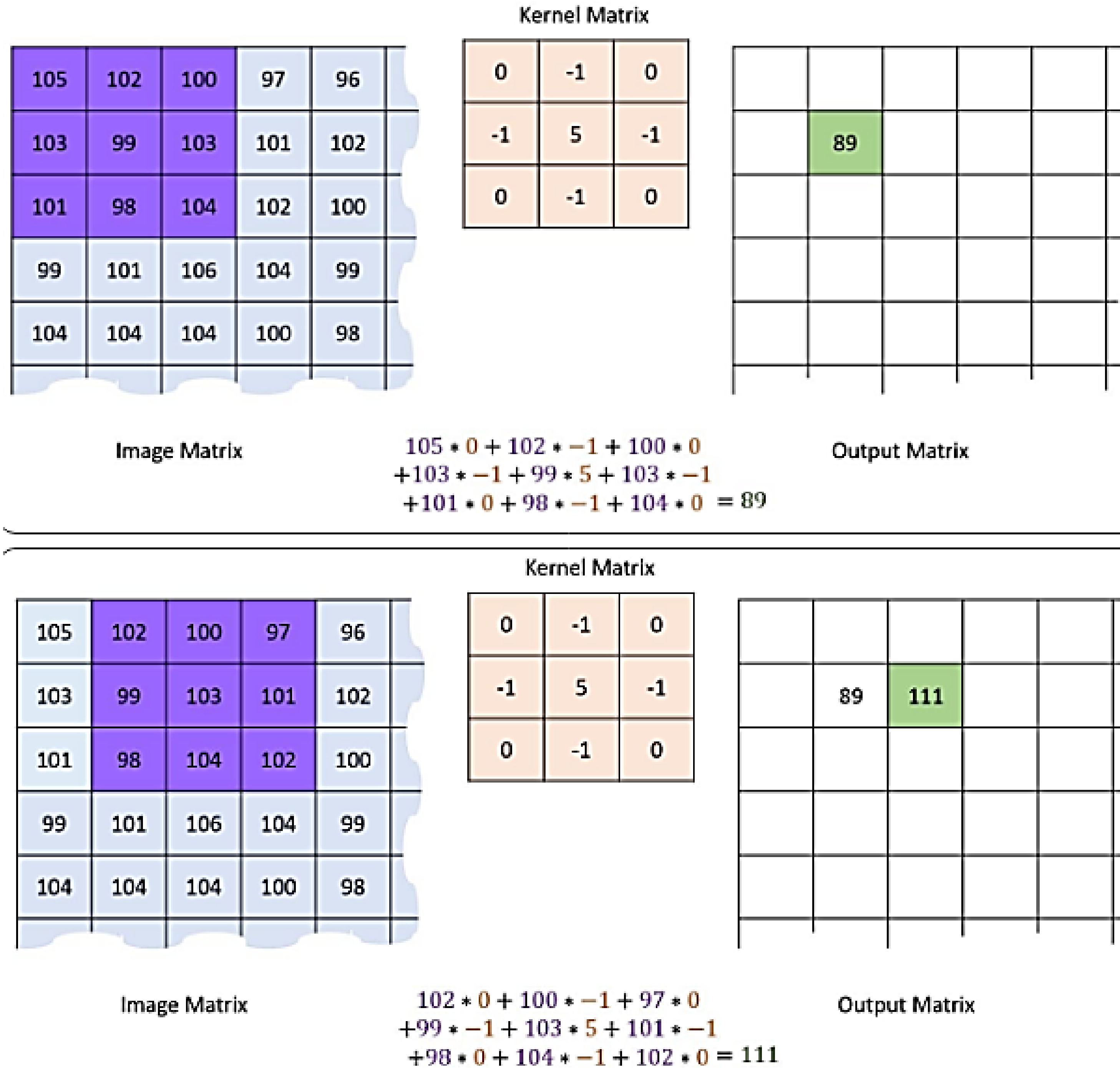
Element-wise multiplication and addition

Convolution



http://machinelearningguru.com/computer_vision/basics/convolution/image_convolution_1.html

Convolution



http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html

Convolution

Pixels on the border of image matrix?

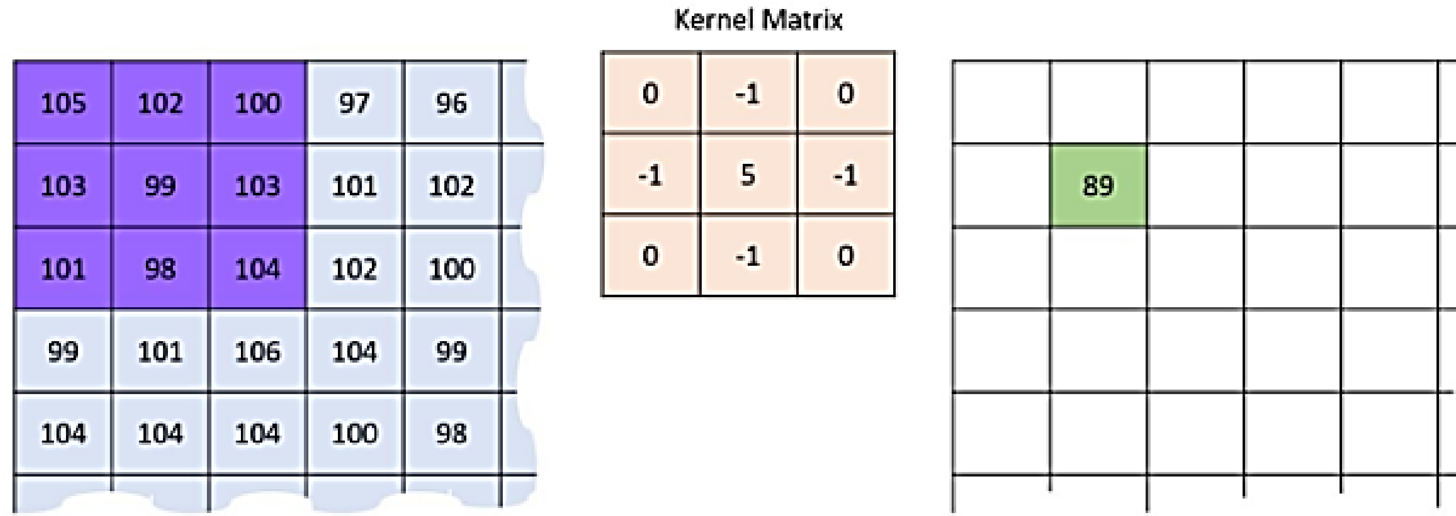


Image Matrix

$$105*0+102*-1+100*0$$
$$+103*-1+99*5+103*-1$$
$$+101*0+98*-1+104*0 \;=89$$

Output Matrix

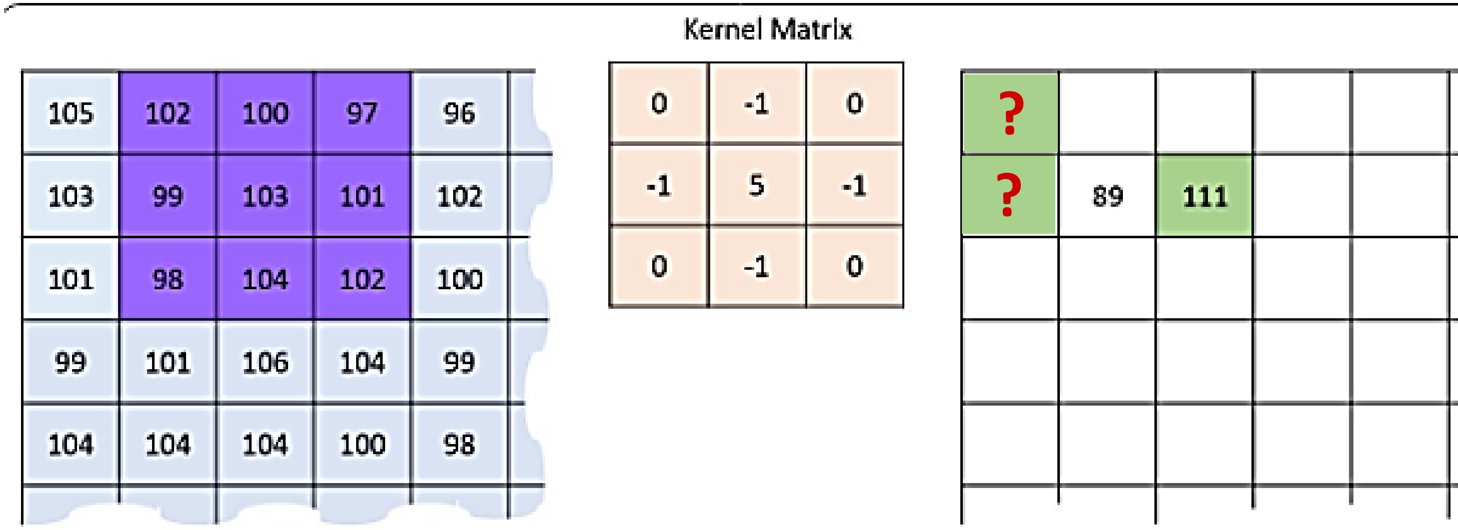


Image Matrix

$$102*0+100*-1+97*0$$
$$+99*-1+103*5+101*-1$$
$$+98*0+104*-1+102*0=111$$

Output Matrix

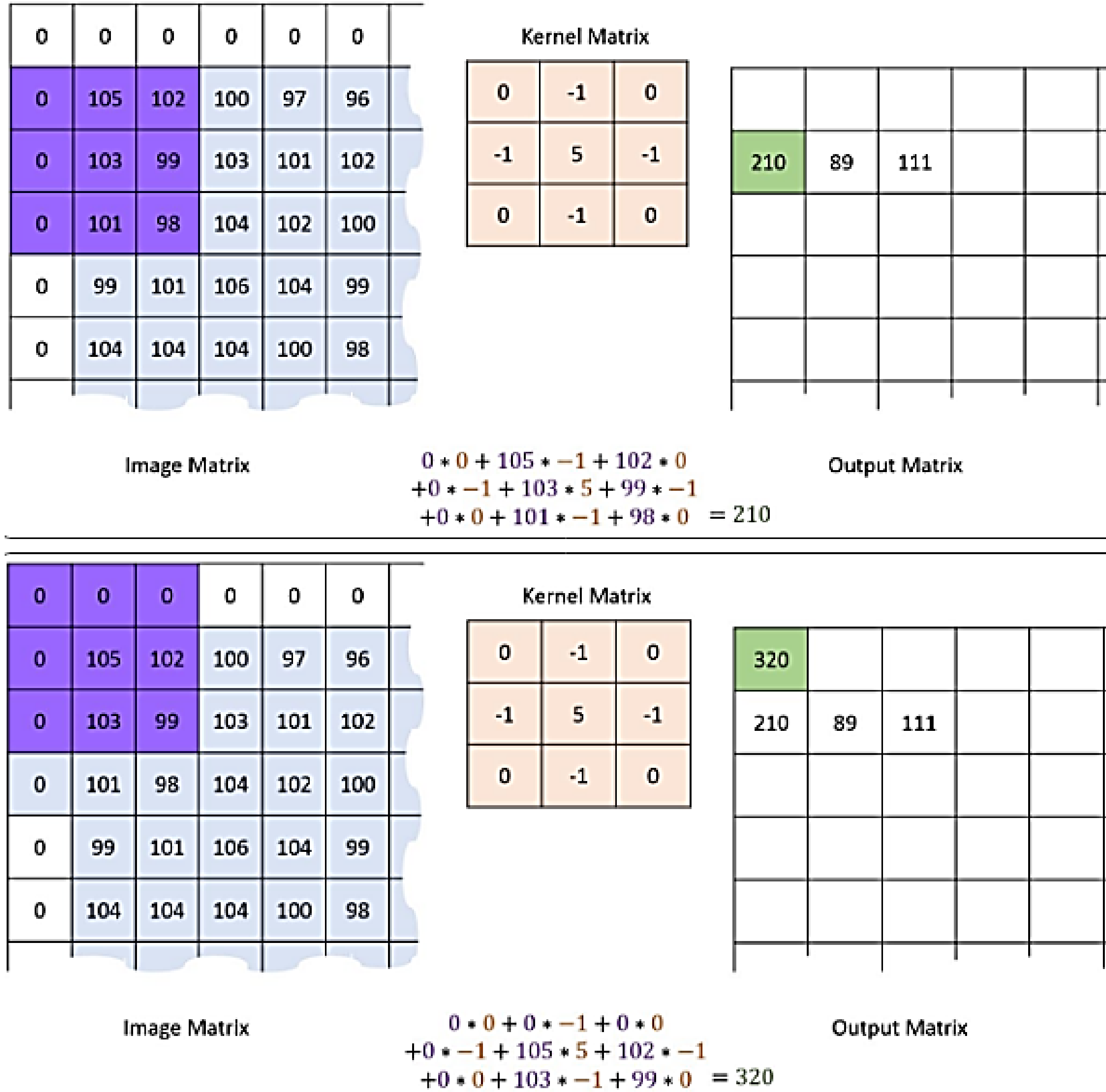
http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html

Convolution Padding

The process of adding zeros to the input matrix symmetrically to **maintain the dimension** of output as in input.

(1 pixel padding here, all around)

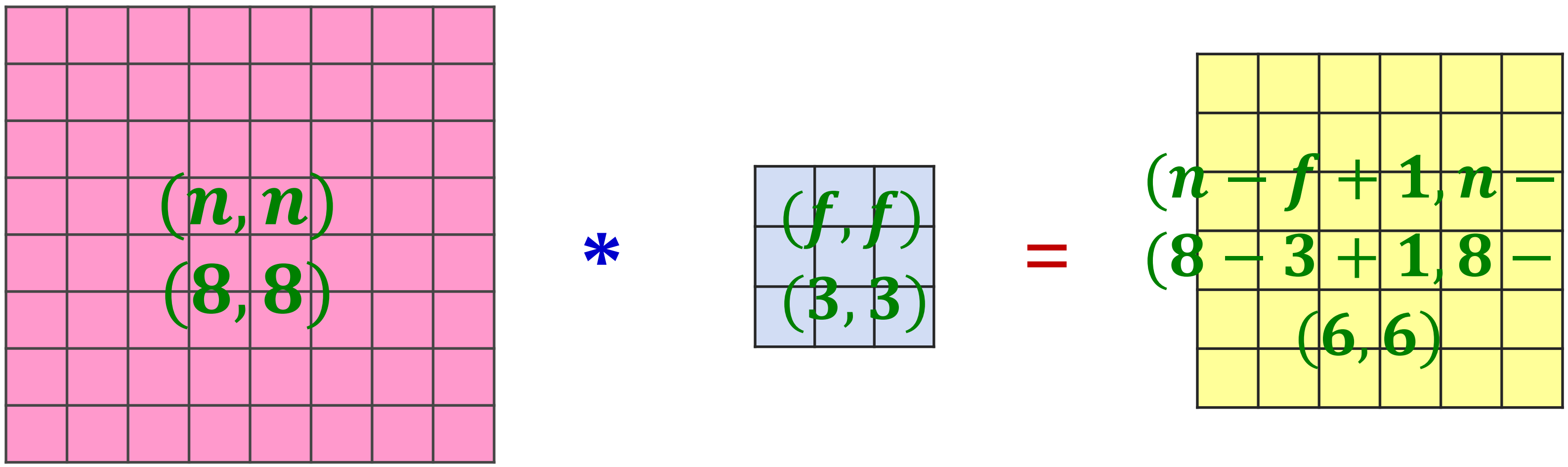
Padding depends on the dimension of the filter.



http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html

Padding – padded convolution

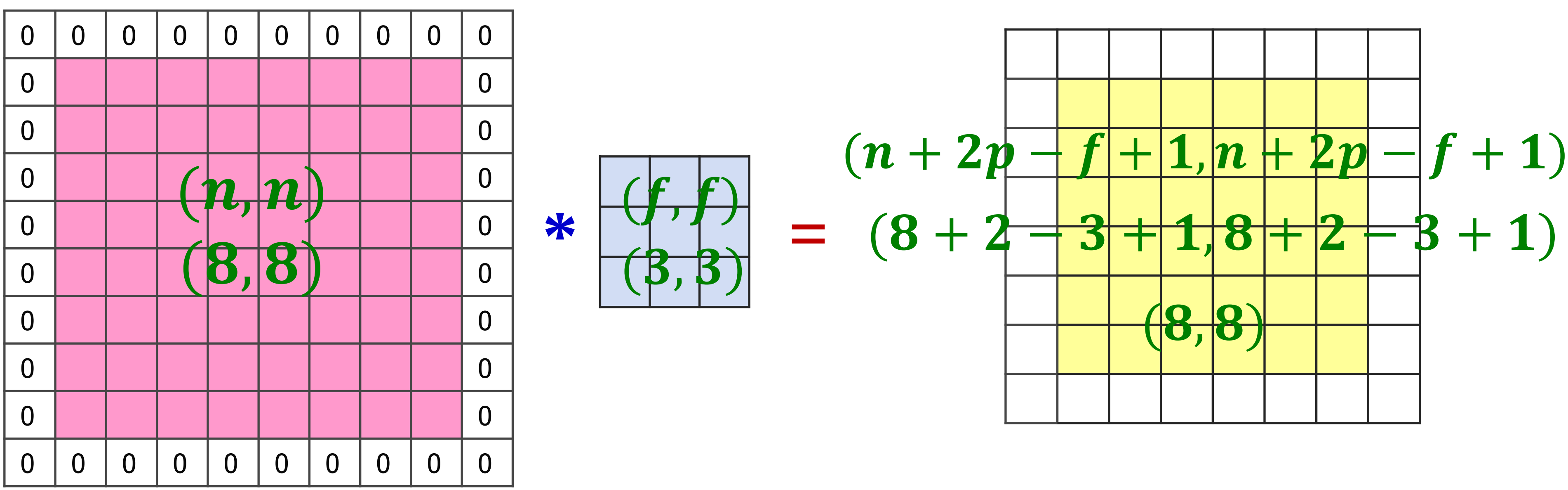
$p = 0$ **Valid** convolution – no padding, the image is shrinking



$$(n, n) * (f, f) = (n - f + 1, n - f + 1)$$

$$(8, 8) * (3, 3) = (8 - 3 + 1, 8 - 3 + 1) = (6, 6)$$

$p = 1$ **Same** convolution – padding, the size is the same



$$(n, n) * (f, f) = (n + 2p - f + 1, n + 2p - f + 1)$$

$$(8, 8) * (3, 3) = (8 + 2 - 3 + 1, 8 + 2 - 3 + 1) = (8, 8)$$

Padding – padded convolution

Same convolution

- padding,
- the size of the feature map is the same with the size of the input image.

Compute the necessary padding size, p =?

$$\boldsymbol{n + 2p - f + 1 = n}$$

$$\boldsymbol{p = \frac{f - 1}{2}}$$

Odd number for the filter size (3, 5, 7) is recommended.

There is a center of the filter, so one can talk about the position of the filter.

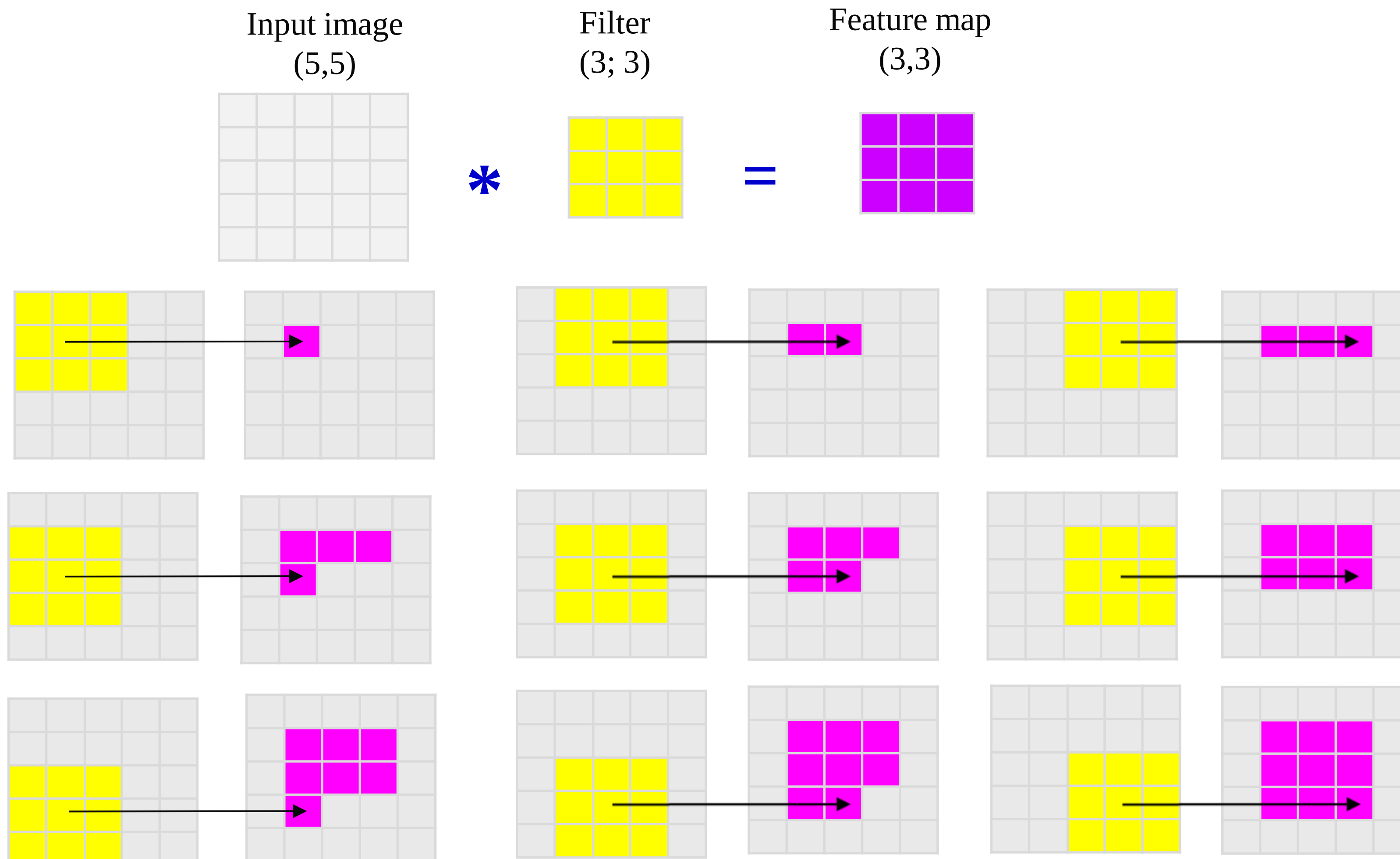
Convolution Stride

Stride denotes how many steps we are moving in each steps in convolution.

Usually, it is $s = 1$

stride = amount you move the window each time you slide

Illustration for input size: (5,5); filter size: (3,3), stride = (1,1)



Size of the output image (feature map)

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

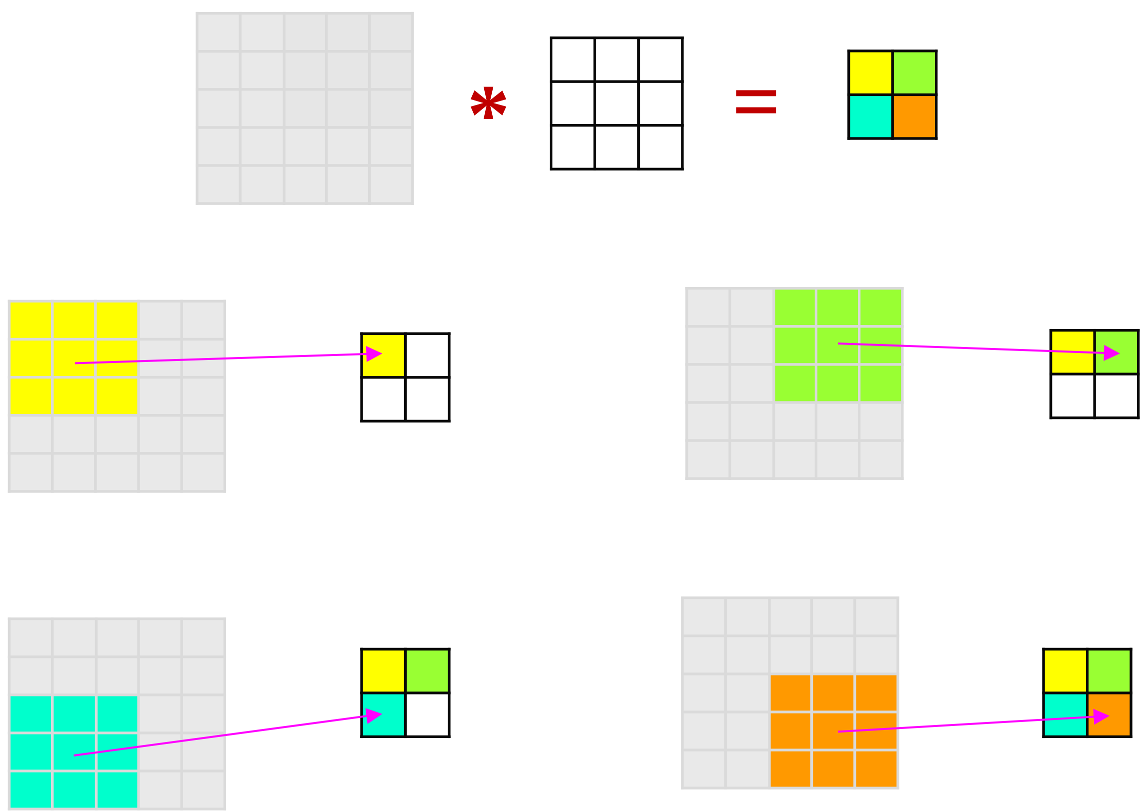
rounded down (floor)

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

$$\left(\left\lfloor \frac{5+2\cdot 0-3}{2} + 1 \right\rfloor ; \left\lfloor \frac{5+2\cdot 0-3}{2} + 1 \right\rfloor \right)$$

$$(2; 2)$$

Input:	$(n; n)$,	$(5; 5)$
Filter:	$(f; f)$	$(3; 3)$
Padding:		$p = 0$
Stride:		$s = 2$
Output:		$(2; 2)$

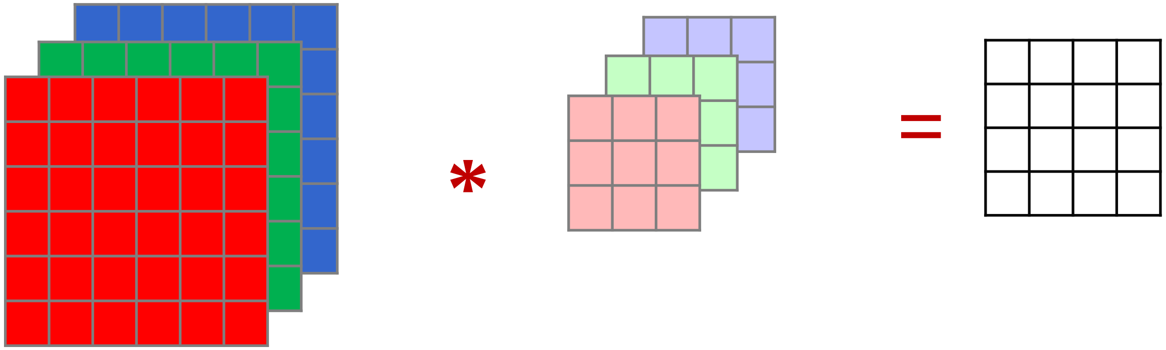


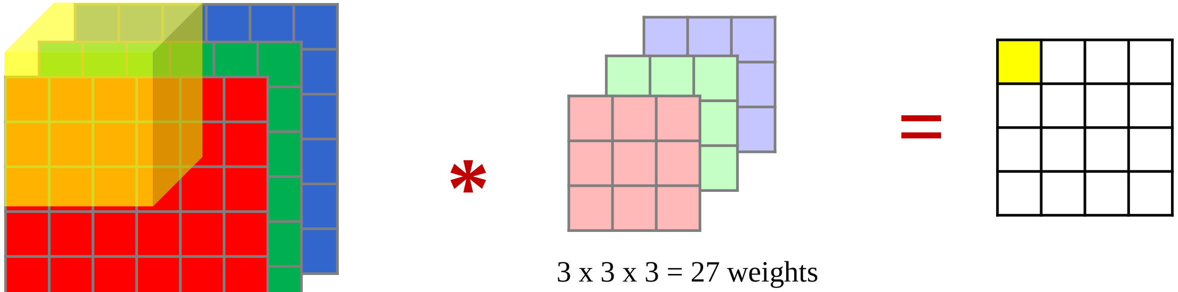
Convolution over volume

Convolution over a RGB input image

$n = 6$
$f = 3$
$p = 0$
$s = 1$

3 input channels





Each convolution over volume produces one 2D output

(6, 6, **3**) (3, 3, **3**) (4, 4)

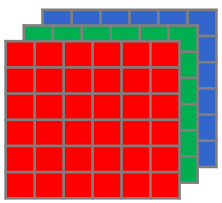
(height, width, channels)

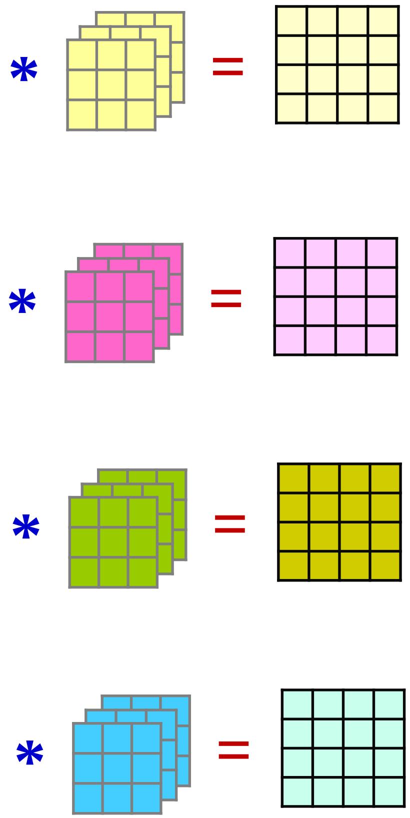
Convolution over volume - multichannel

We can extract multiple features (using multiple filters) in one step

1 3D input image; **4 3D filters**; **4 2D output images**
3 input channels; **4 output channels**

3 input channels
6x6x3 = 108 pixels





Vertical edges on green channel

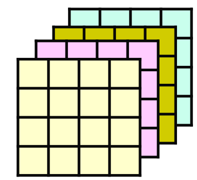
Horizontal edges on all channels

73 degrees edges

…………

4 output channels

output (4 channels)
4x4x4 = 64 numbers



In case of 128 3D filters;
128 output channels
4x4x128 = 2048 values

Illustration for multichannel convolution

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	1	1	1	0	0
0	2	1	1	2	2	0
0	2	1	1	0	0	0
0	1	2	0	2	2	0
0	0	2	0	1	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	0	1	1	2	0
0	1	2	1	1	0	0
0	0	0	2	0	2	0
0	2	0	0	0	2	0
0	0	2	2	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	2	0	2	1	0
0	0	0	2	1	2	0
0	0	0	1	0	2	0
0	1	2	2	0	2	0
0	2	0	1	1	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

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

w0[:,:,1]

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

w0[:,:,2]

1	0	0
-1	1	0
-1	0	0

Bias b0 (1x1x1)

b0[:,:,0]

1

filter 0

Filter W1 (3x3x3)

w1[:,:,0]

1	0	-1
1	1	0
-1	0	1

w1[:,:,1]

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

w1[:,:,2]

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

Bias b1 (1x1x1)

b1[:,:,0]

0

filter 1

Output Volume (3x3x2)

o[:,:,0]

-2	2	1
-7	0	3
2	4	3

output channel 0

o[:,:,1]

0	0	-4
4	2	1
0	1	0

output channel 1

3 input channels; 2 3D filters, 2 output channels

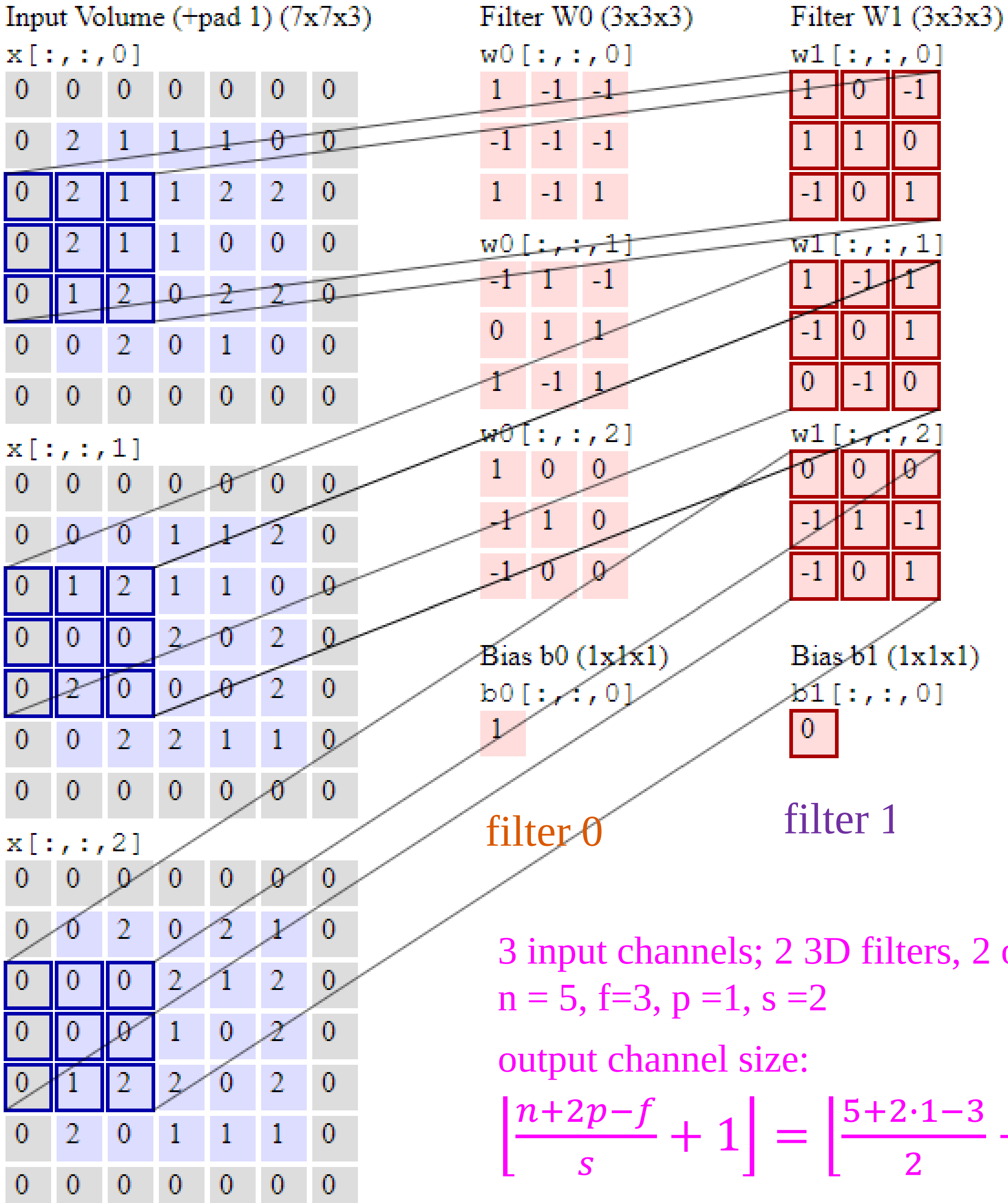
$n = 5$, $f=3$, $p=1$, $s=2$

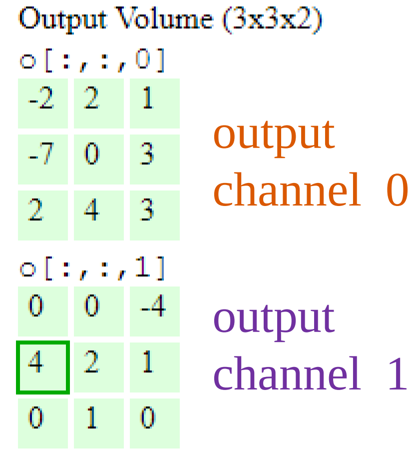
output channel size:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor = \left\lfloor \frac{5+2\cdot1-3}{2} + 1 \right\rfloor = 3$$

http://cs231n.github.io/convolutional-networks/

Illustration for multichannel convolution





3 input channels; 2 3D filters, 2 output channels

$n = 5$, $f=3$, $p=1$, $s=2$

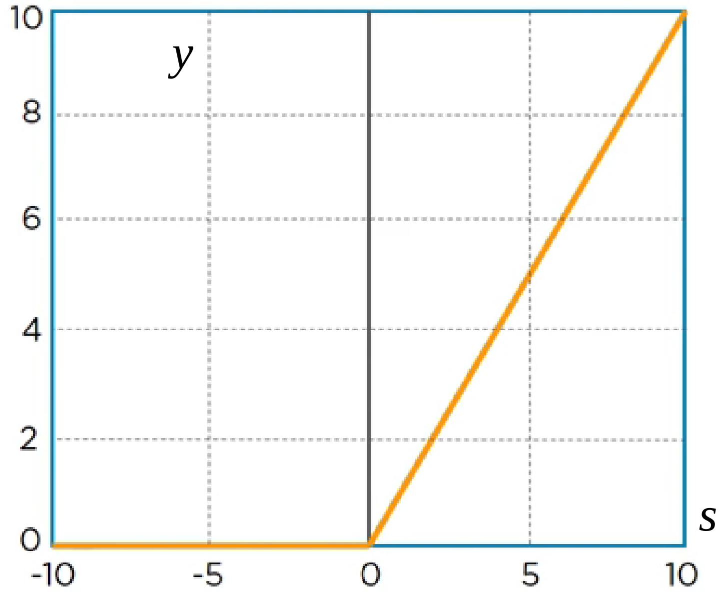
output channel size:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor = \left\lfloor \frac{5+2\cdot 1-3}{2} + 1 \right\rfloor = 3$$

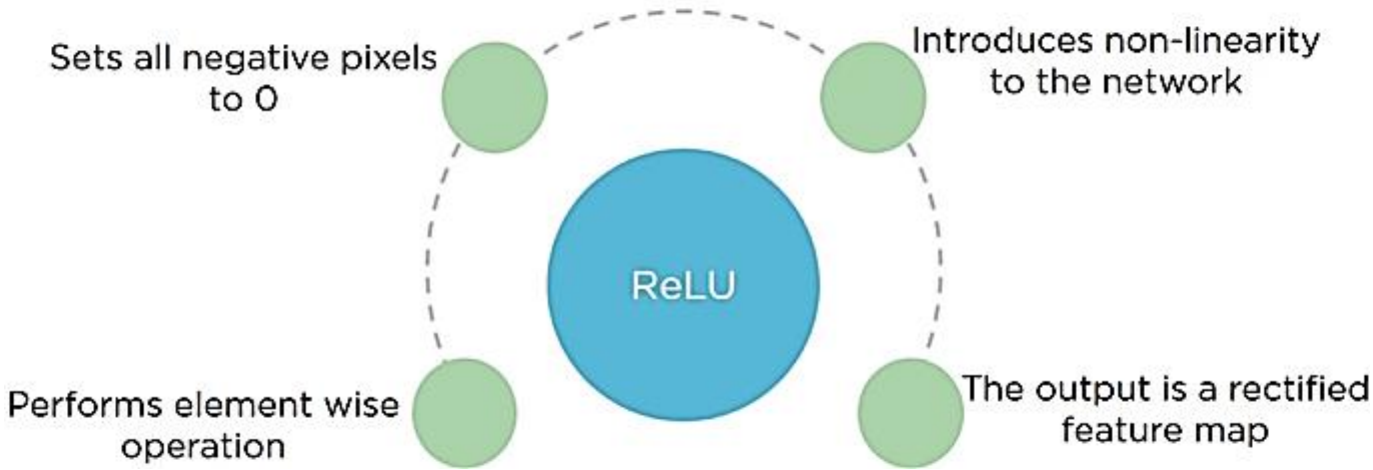
http://cs231n.github.io/convolutional-networks/

ReLU layer

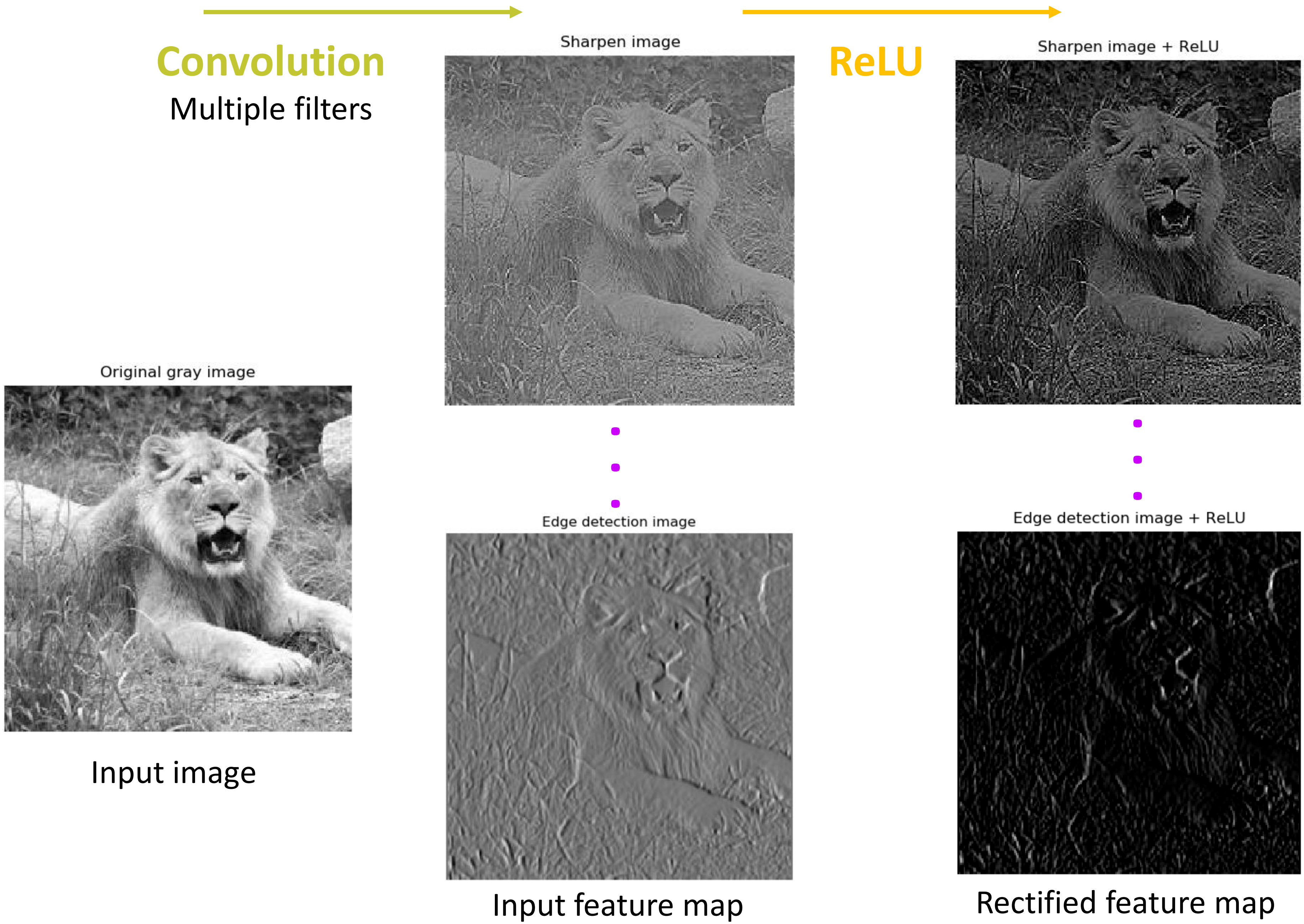
Once the feature map is extracted by the convolution, the next operation is to apply the ReLU activation function



$$y = \max(0, s)$$



https://www.youtube.com/watch?v=Jy9-aGMB_TE



Input image is scanned in multiple convolution and ReLU layers

Original gray image

First 5 columns and rows of the input image matrix:

```
[[33.6861 21.462  22.3216 32.8946 29.1734]
 [62.8323 38.6861 21.462  15.7388 12.1618]
 [53.7622 55.191  50.8323 39.2553 27.7504]
 [67.3392 79.3392 74.3372 57.8323 47.2553]
 [85.2028 79.8441 50.8421 29.3314 34.8323]]
```

Sharpen image

First 5 columns and rows of the image sharpen matrix:

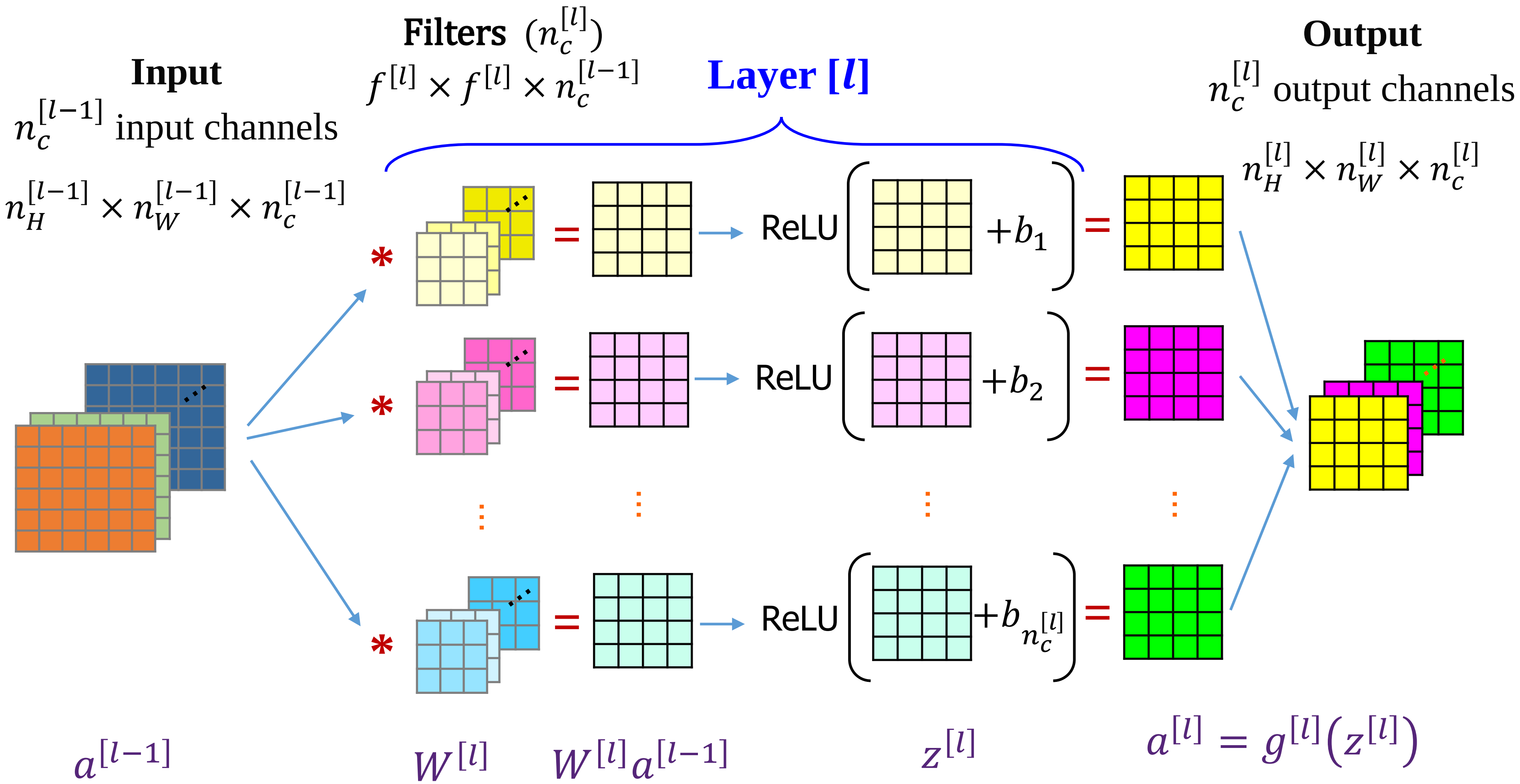
```
[[ 84.1362  12.3258  36.2923  95.4401  78.5729]
 [188.1617  33.9352 -19.8438 -29.4384 -28.3799]
 [ 80.8206  54.1131  63.916   44.8381  15.6216]
 [121.9746 116.2651 133.6276  98.9823  78.1111]
 [196.6922  98.347    8.774  -31.7601  18.4111]]
```

Sharpen image + ReLU

First 5 columns and rows of the image sharpen+ReLU matrix:

```
[[ 84.1362  12.3258  36.2923  95.4401  78.5729]
 [188.1617  33.9352   0.       0.       0.    ]
 [ 80.8206  54.1131  63.916   44.8381  15.6216]
 [121.9746 116.2651 133.6276  98.9823  78.1111]
 [196.6922  98.347    8.774    0.      18.4111]]
```

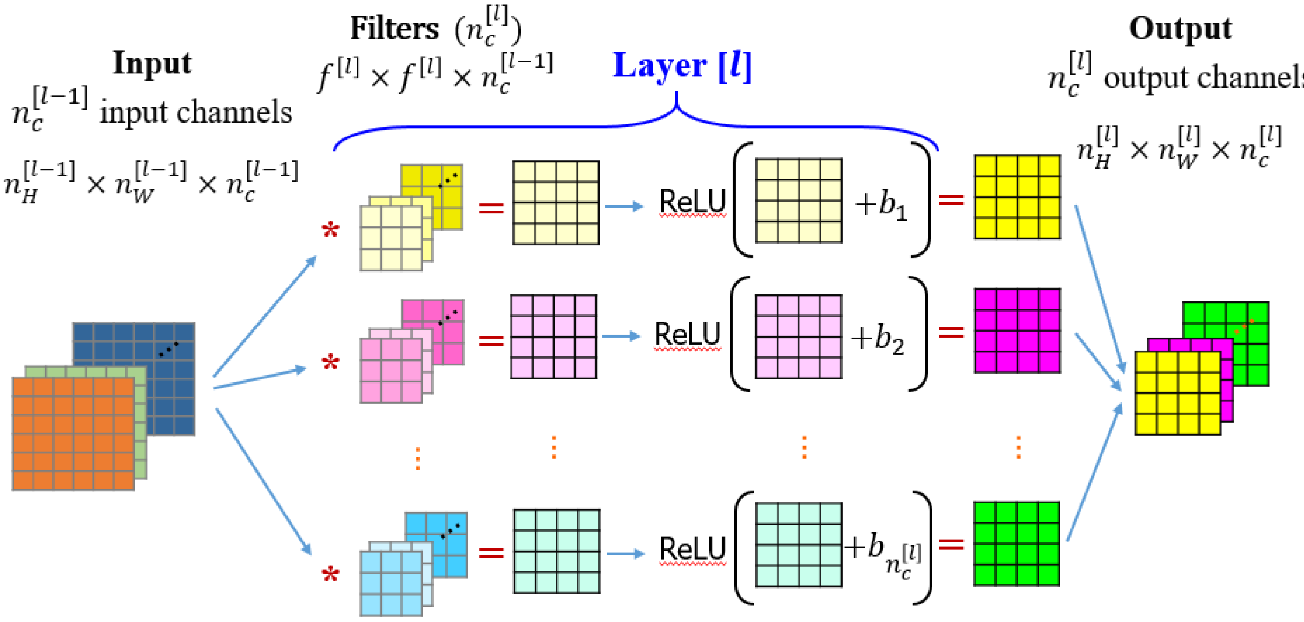
Convolution + ReLU – one layer operation



Forward propagation

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

CNN – one layer.
Parameters
Hyperparameters



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

b: $n_c^{[l]}$ one scalar for each filter

$f^{[l]} = 3$ filter size

$n_c^{[l-1]} = 3$ number of input channel

$n_c^{[l]} = 16$ number of filters (output channel)

$3 \times 3 \times 3 \times 16 + 16 = 448$ parameters in layer $[l]$

No matter how big the input (image) is,
the number of parameters is the same.

Hyperparameters

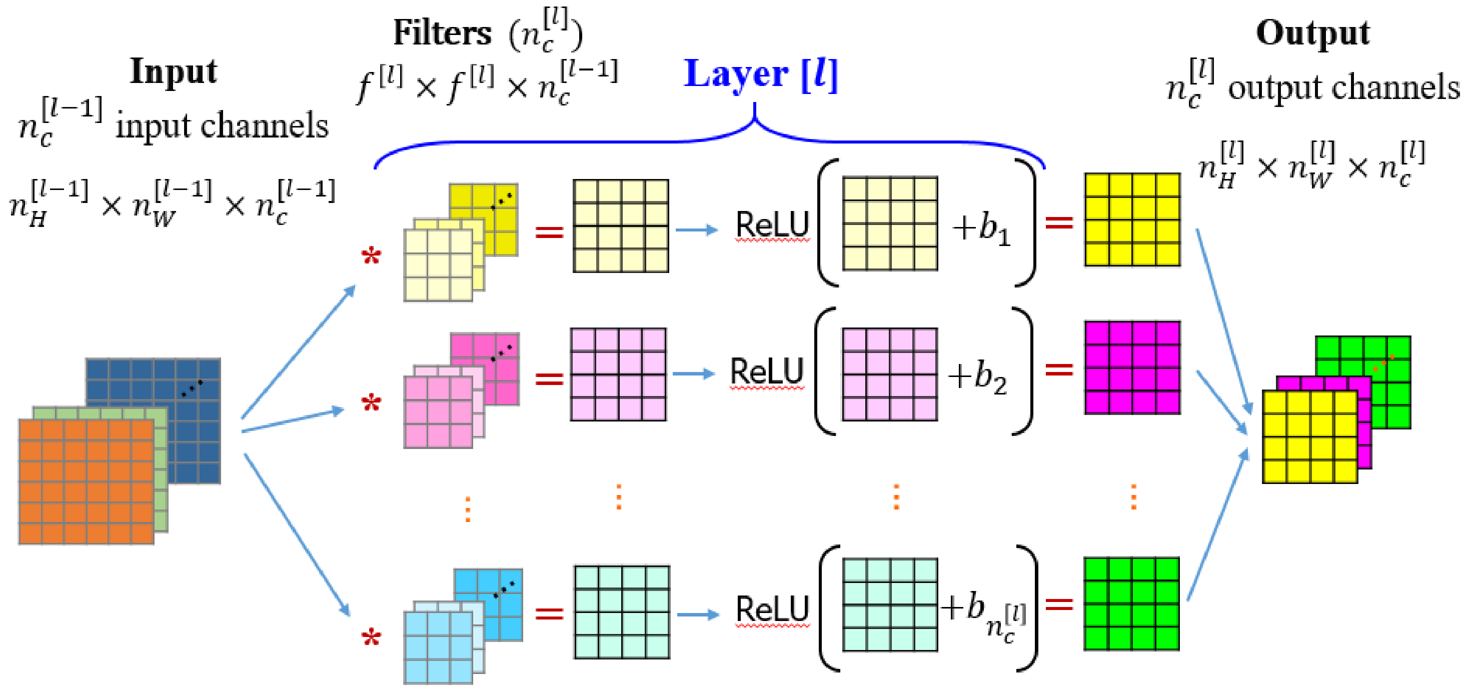
$n_c^{[l]}$ number of filters (output channels)

$f^{[l]}$ filter size

$s^{[l]}$ stride

$p^{[l]}$ padding

Convolution + ReLU



$p^{[l]}$ padding

$s^{[l]}$ stride

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \qquad n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

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

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

Vectorized for m examples (batch)

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

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

Simple Convolutional Neural Network (ConvNet)

CONV [1]

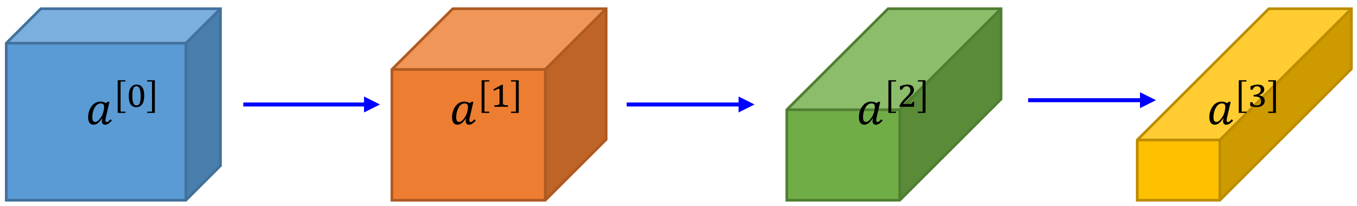
$f^{[1]} = 3$
$s^{[1]} = 1$
$p^{[1]} = 0$
$n_c^{[1]} = 8$

CONV [2]

$f^{[2]} = 5$
$s^{[2]} = 2$
$p^{[2]} = 0$
$n_c^{[2]} = 16$

CONV [3]

$f^{[3]} = 7$
$s^{[3]} = 2$
$p^{[2]} = 0$
$n_c^{[2]} = 32$



$64 \times 64 \times 3$ $\quad$ $62 \times 62 \times 8$ $\quad$ $29 \times 29 \times 16$ $\quad$ $12 \times 12 \times 32$

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \qquad n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

- **image size stays almost the same in the beginning, then slightly decreases**
- **number of channels (filters) increases**

Pooling layer

Once the feature map is rectified by the ReLU activation function, the next operation is to **down-sampling** the images to **reduce the dimensionality** through a **pooling layer**

0	0	14	82
149	32	0	0
28	53	64	44
39	120	133	99

Rectified feature map

$4 \times 4 = 16$

max pooling →

2 x 2 filter
stride 2

149	82
120	133

Pooled feature map

$2 \times 2 = 4$

Max pooling – how valuable is a feature in the area of the filter

- best (maximum feature value)

Dimensionality reduction (given by the filter size and stride):

4 to 1;   4 times

Dimensionality reduction for 2x2 filter and stride 1?

Pooling layer

0	0	25	14	82
149	32	31	0	0
111	200	20	135	10
28	53	20	64	44
39	120	210	13	99

$n_H \times n_W \times n_c$

Rectified feature map

max pooling

3 x 3 filter; $f = 3$

stride 1; $s = 1$

200	200	135
200	200	135
210	210	210

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

Pooled feature map

Apply on each channel independently

No parameters to learn

Hyperparameters: f, s

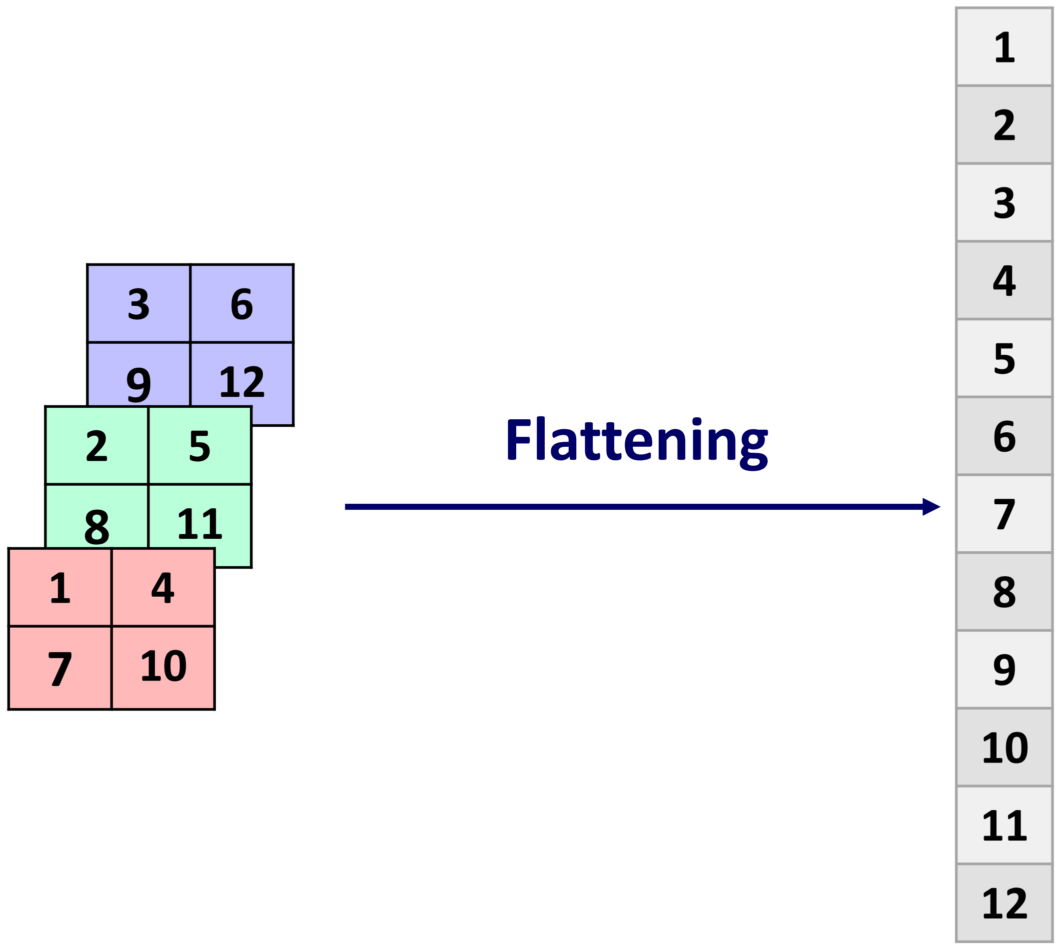
Average pooling is (very) rarely used

Flattening

Once the dimensionality of the data volume is reduced to a manageable size, we must connect further with the fully connected (FC) layer.

We need to convert the pooled feature map to a column vector using the **flattening** operation

Pooled feature map



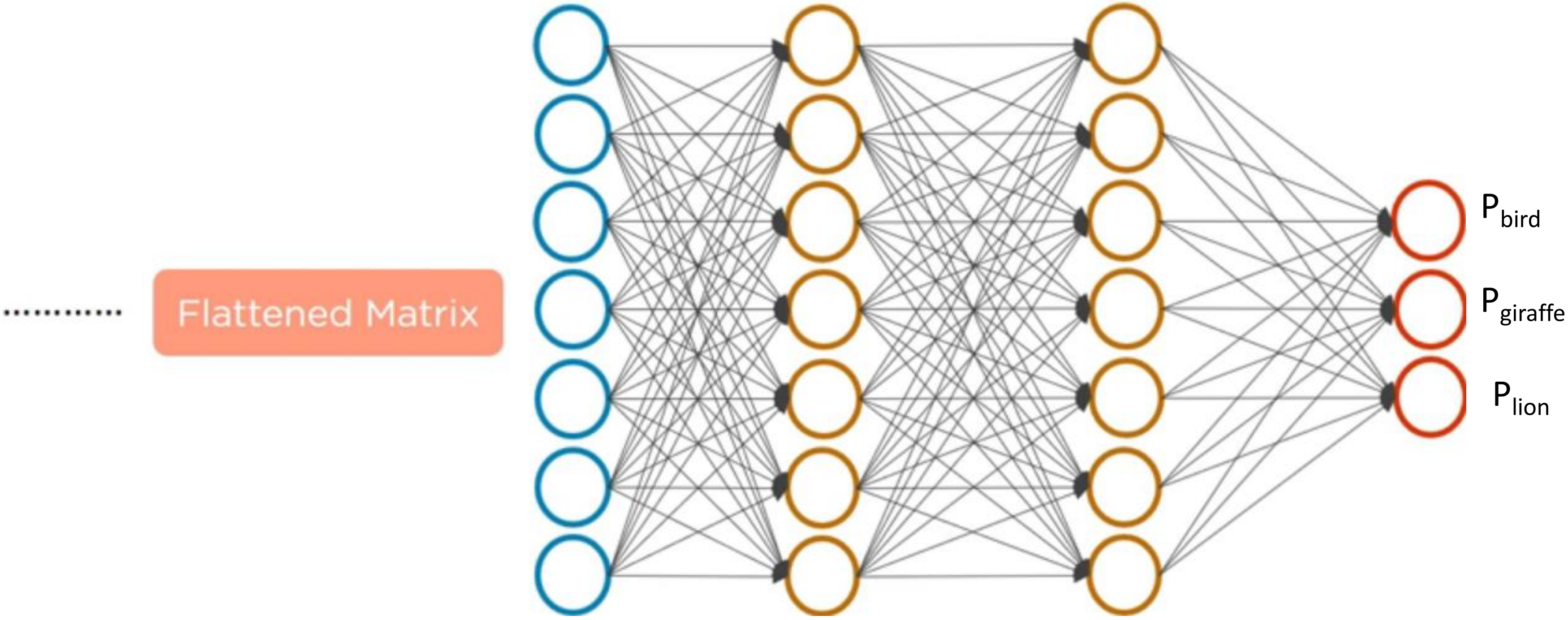

```
[[[1  2  3]
 [4  5  6]]

[[7  8  9]
[10 11 12]]]
```

```
[1  2  3  4  5  6  7   8  9  10  11 12]
```

Fully connected layer

The vector (flattened 2D array) from the pooling layer is fed to the fully connected layer (conventional feed-forward ANN) to classify the image



https://www.youtube.com/watch?v=Jy9-aGMB_TE

Fully connected layer

Softmax activation function (used in CNN)

In deep learning, the term logits layer is popularly used for the last neuron layer of neural network for classification task which produces raw prediction values as real numbers ranging from [-infinity, +infinity]. Before activation take place.
Softmax acts as an activation function, and it turns logits (numeric output of the last linear layer of a multi-class classification neural network) into probabilities by taking the exponents of each output and then normalizing each number by the sum of those exponents.

Logits scores (s)	Softmax	Output probabilities	Classes
5.0	$f(s_i) = \frac{e^{s_i}}{\sum_j e^{s_j}}$	0.2641	Car
6.0		0.7179	Truck
1.0		0.0048	Motorcycle
2.0		0.0131	Bus
		Sum = 1	

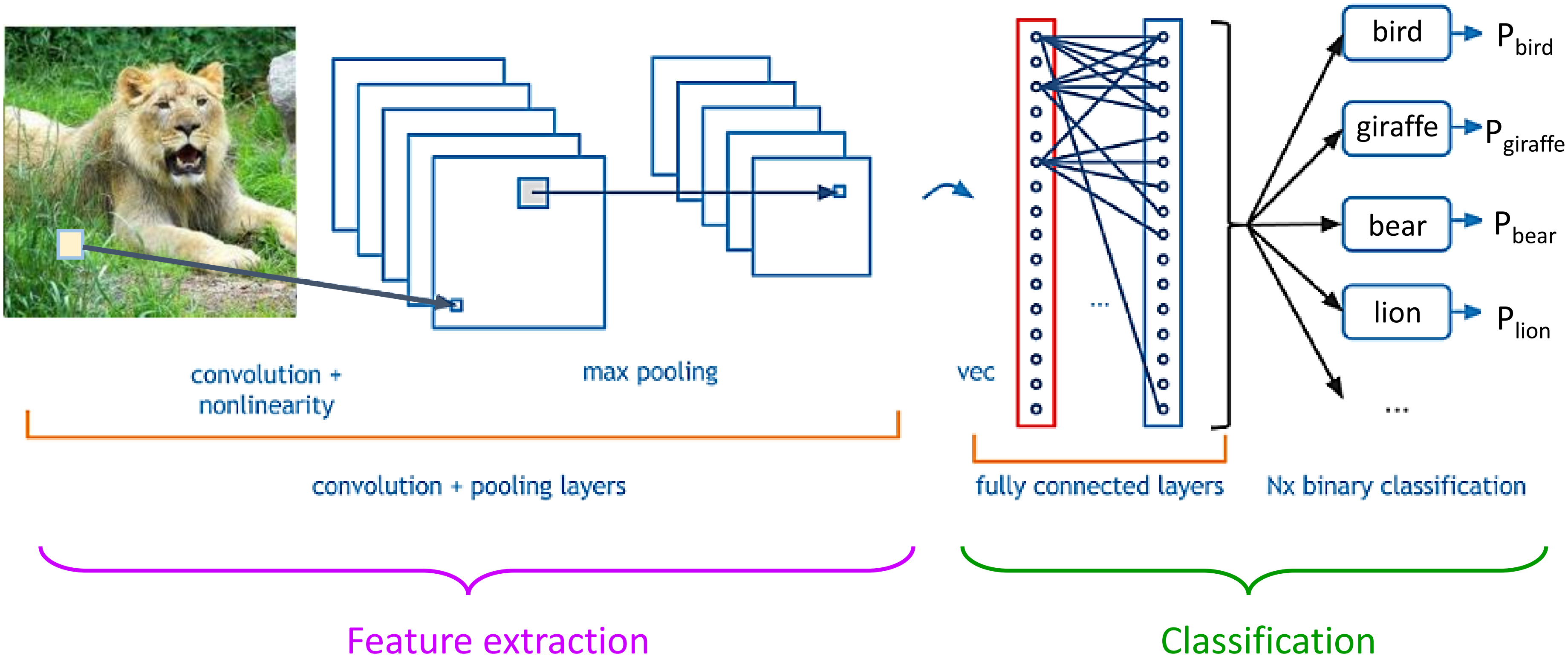
So, the entire output vector adds up to one — all probabilities should add up to one.

Cross entropy loss is usually the loss function for such a multi-class classification problem.

Softmax is frequently appended to the **last layer of a multi-class image classification network** such as those in **CNN** (Alexnet, VGG16, etc.) used in ImageNet competitions.

[Understand the Softmax Function in Minutes, January 2018, https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d]

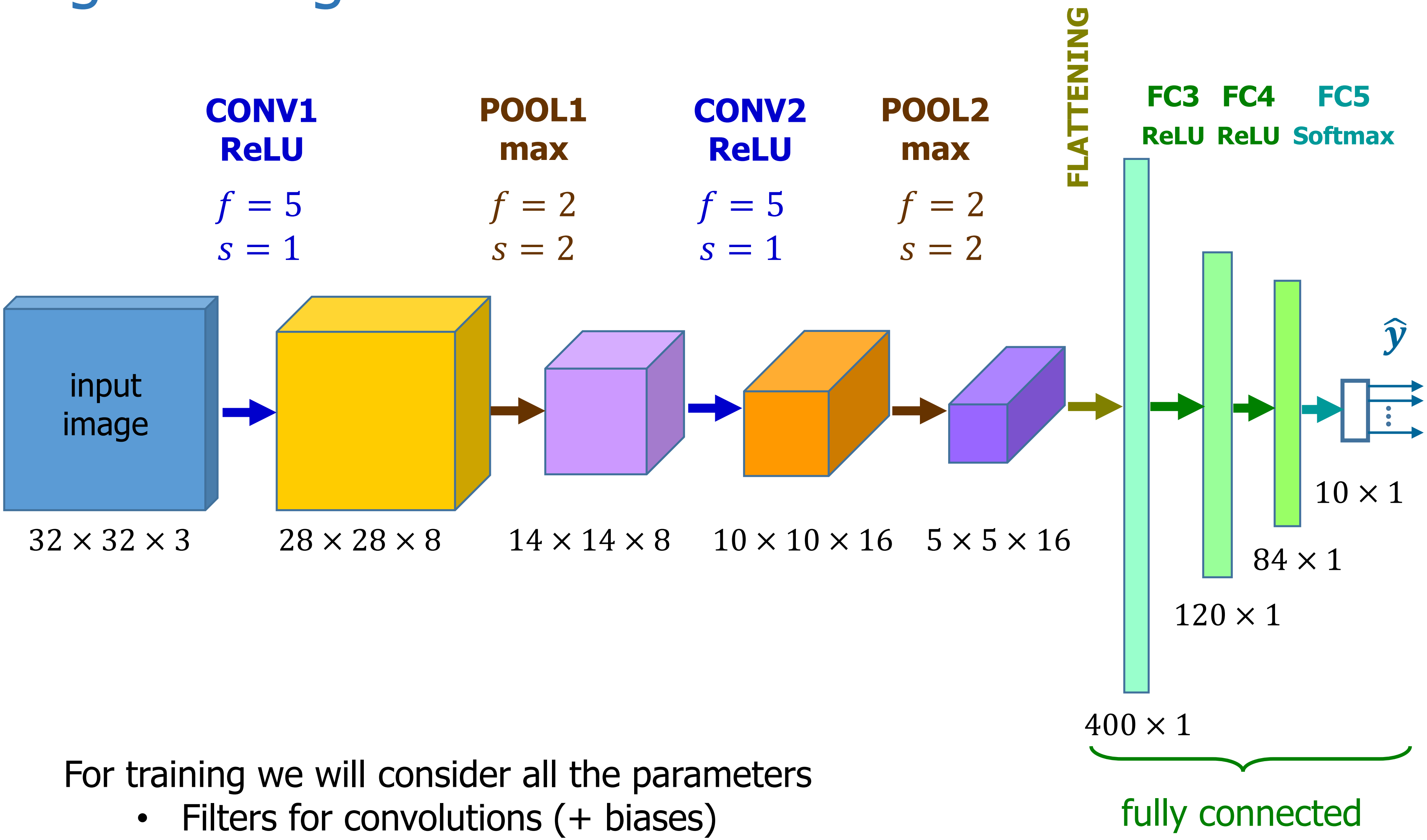
CNN – big picture



Adaptation after: https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

Full CNN - illustration

Digits recognition



For training we will consider all the parameters

- Filters for convolutions (+ biases)
- Weights and biases for FC

	Activation shape	Activation size	# parameters
Input image	(32, 32, 3)	3,072	-
CONV1 (f=5, s=1, n_c=8)	(28, 28, 8)	6,272	5x5x3x8+8 608
POOL1 (f=2, s=2)	(14, 14, 8)	1,568	-
CONV2 (f=5, s=1, n_c=16)	(10, 10, 16)	1,600	5x5x8x16+16 3,216
POOL2 (f=2, s=2)	(5, 5, 16)	400	-
FC3	(120, 1)	120	120x400+120 48,120
FC4	(84, 1)	84	84x120+84 10,164
FC5 Softmax	(10, 1)	10	10x84+10 850
			62,958

Even if the activation size in smaller in the FC layers, here the number of learning parameters is larger.

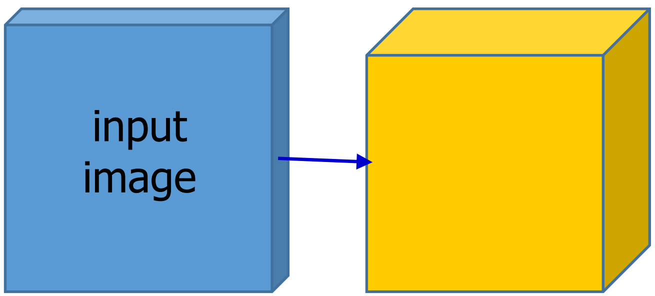
Convolution vs FC

The advantage of a convolution layer over a FC layer is the number of parameters

- Parameters sharing
- Sparsity of the connections

conv

$f = 5$
$s = 1$



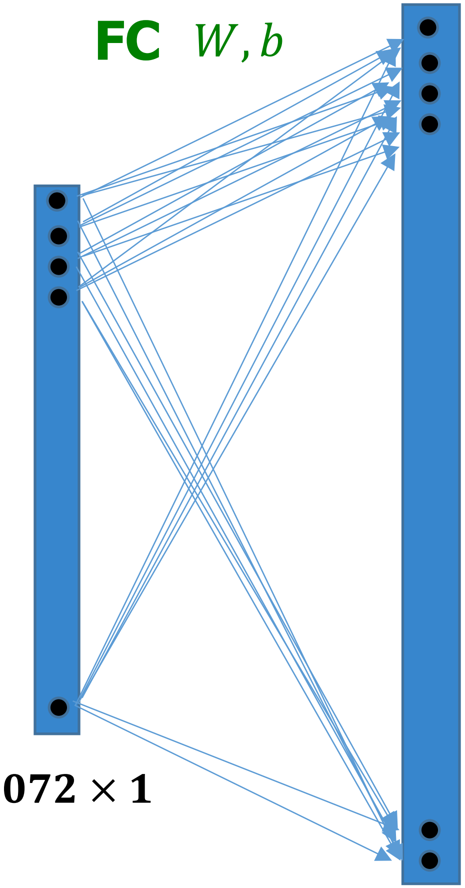
$32 \times 32 \times 3$ $\quad$ $28 \times 28 \times 8$

$\mathbf{3,072}$ $\quad$ $\mathbf{6,272}$

$5 \times 5 \times 3 \times 8 + 8$

$\mathbf{608}$

FC W, b



$\mathbf{3,072 \times 1}$

$\mathbf{6,272 \times 1}$

$6{,}272 \times 3{,}272 + 6{,}272$

$\mathbf{20,528,256 \approx 20,5M}$

Training parameters

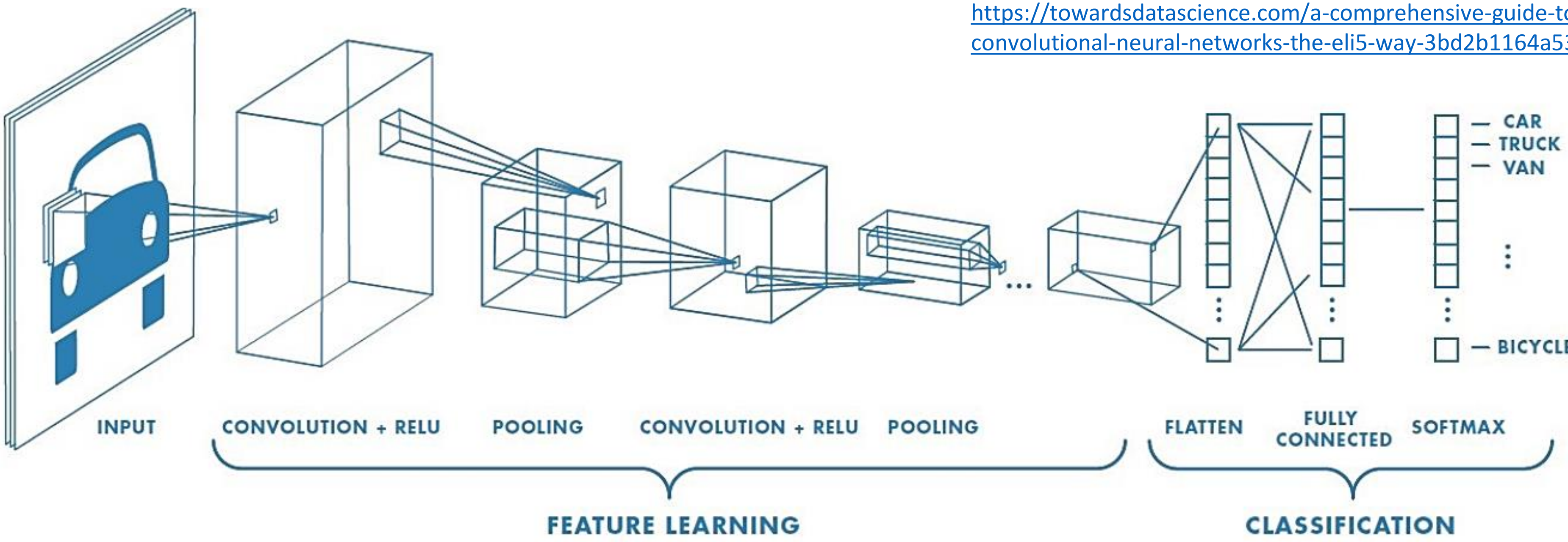
$$\mathbf{608 \ll 20.5\ M}$$

- **Parameter sharing**: a filter that detects a certain feature (e.g., vertical edges), useful in one part of an image most probably is useful in another part of that image.
    - Highly decreases the training parameters number

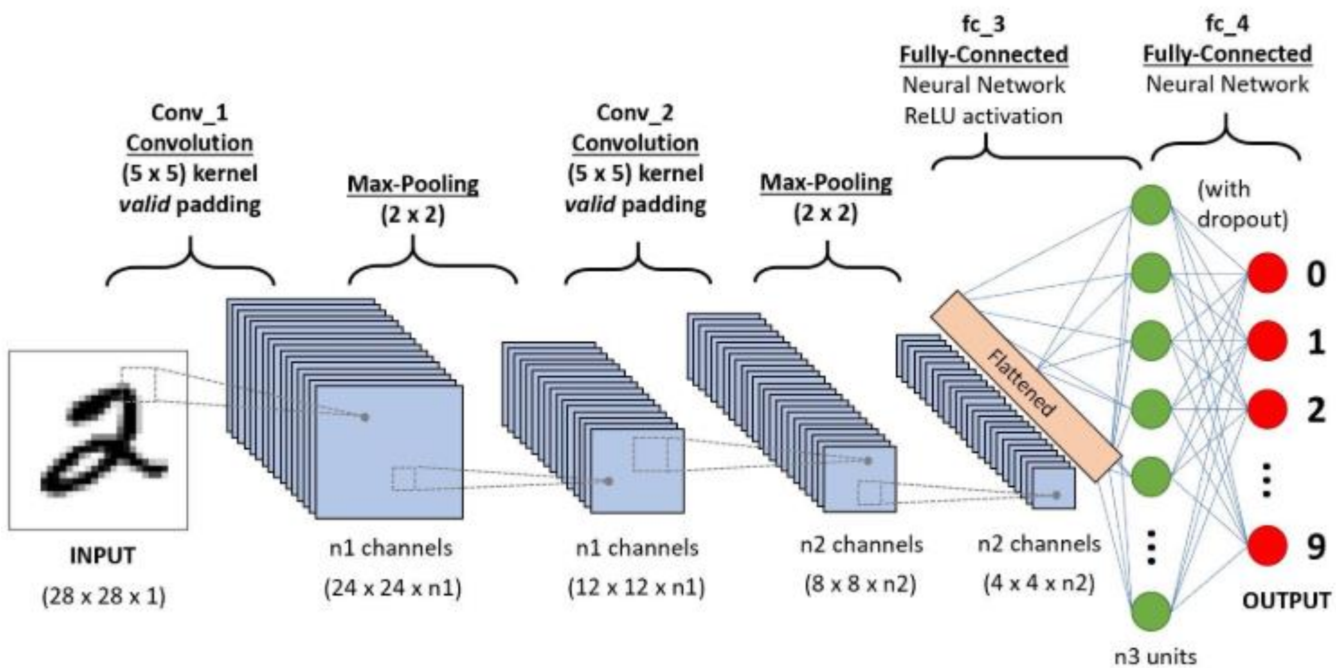
$$
\begin{matrix}
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100 \\
10 & 10 & 10 & 10 & 100 & 100 & 100 & 100
\end{matrix}
\;*\;
\begin{matrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1
\end{matrix}
\;=\;
\begin{matrix}
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0 \\
0 & 0 & 270 & 270 & 0 & 0
\end{matrix}
$$

- **Sparsity of connections**: in each layer, each output value depends only on a small number of inputs – no full connection
    - Highly decreases the training parameters number

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53



Block diagram of some CNNs



A CNN sequence to classify handwritten digits

Exercise

Consider some layers in a Convolutional Neural Network: Convolution layer, ReLU layer, Pooling layer, and FC (Fully Connected) layer.

a) **0.5p** For the Convolution layer there are 3 input channels and 1 output channel. The input channels and the corresponding convolution filters are presented in the next tables:

For the convolution:

- the bias is $b = 7$.
- use no padding ($p = 0$)
- use stride ($s = 1$)

Determine the shape (dimension) of the feature map after convolution.

For the feature map, compute the values corresponding to:

- 4th column, 4th row
- 3rd column, 2nd row
- 2nd column, 4th row

Input channels

1	2	0	2	1	2
3	2	6	0	3	0
0	0	0	1	0	1
1	3	4	2	2	0
0	2	2	0	3	1
5	3	6	1	2	0

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

1	0	2	3	0	1
0	4	5	2	3	2
2	0	6	0	0	2
0	2	4	1	2	4
2	6	2	0	2	3
2	0	3	0	0	0

Convolution filters

0	0	1
3	4	0
0	0	-1

-1	-2	0
0	2	0
2	-1	0

0	-2	0
-1	1	0
0	-1	0

b) 0.5p Suppose that the feature map after the convolution, presented to the input of the ReLU layer is the next one.

35	20	20	3
1	-1	21	-3
21	7	15	22
-2	22	22	29

Plot the ReLU activation function.
What is the rectified feature map, to the output of the ReLU layer?

The rectified feature map is then presented to the input of the Pooling layer:

- Max pooling
- Filter shape: 2 x 2
- Stride $s = 1$

Determine the pooled feature map to the output of the Pooling layer.

c) 0.5p The pooled feature map (after flattening) is connected with a FC layers with 32 neurons.

Determine the total number of training parameters involved in all layers (Convolution + ReLU, Pooling, and FC).

d) 0.5p For a convolution layer, the shape of the input volume is 256 x 256 x 3 (*height* x *width* x *channels*). The convolution uses 7 x 7 filters, stride s=1, 8 output channels.

What is the necessary padding, p, for the "same" convolution (the feature map preserves the size of the input image)?

Determine the number of training parameters