1D/2D Convolutional Neural Networks

Liang Liang

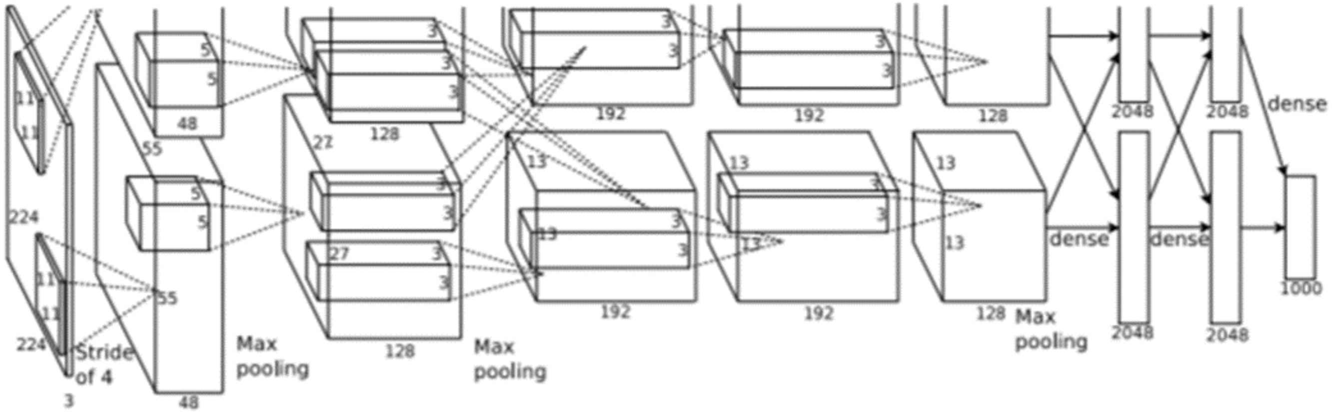
Cross-correlation / Convolution:
a simple & powerful method to process 1D/2D/3D/N-D signals

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

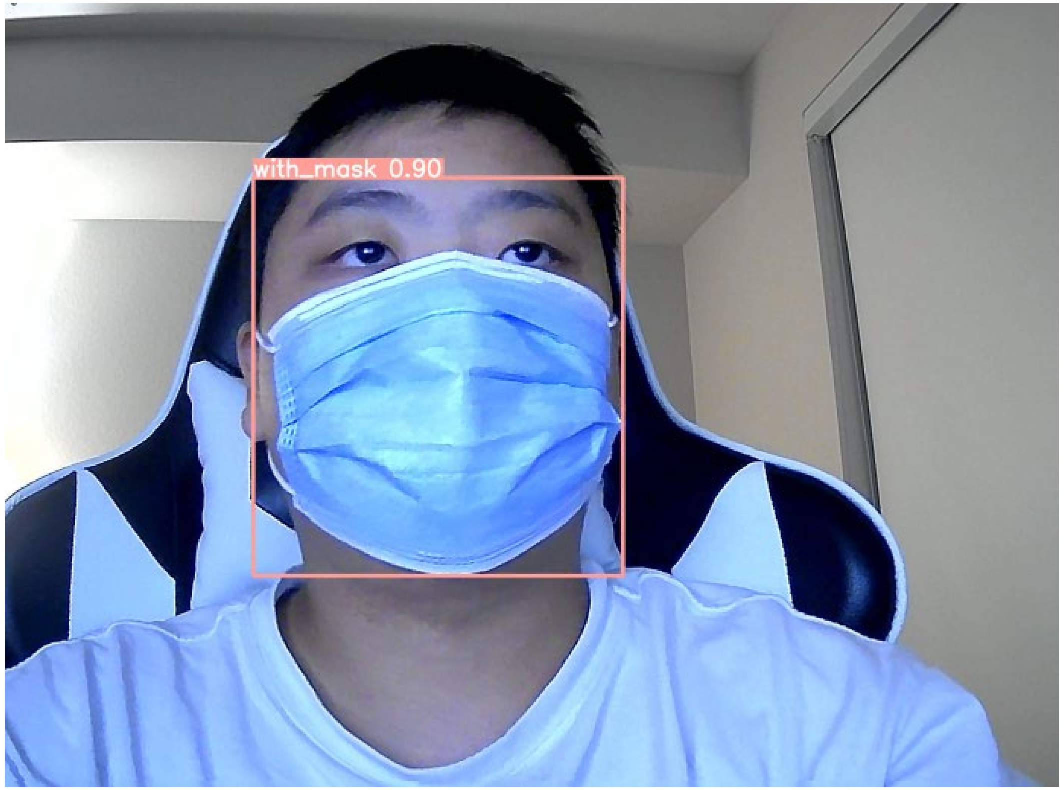
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

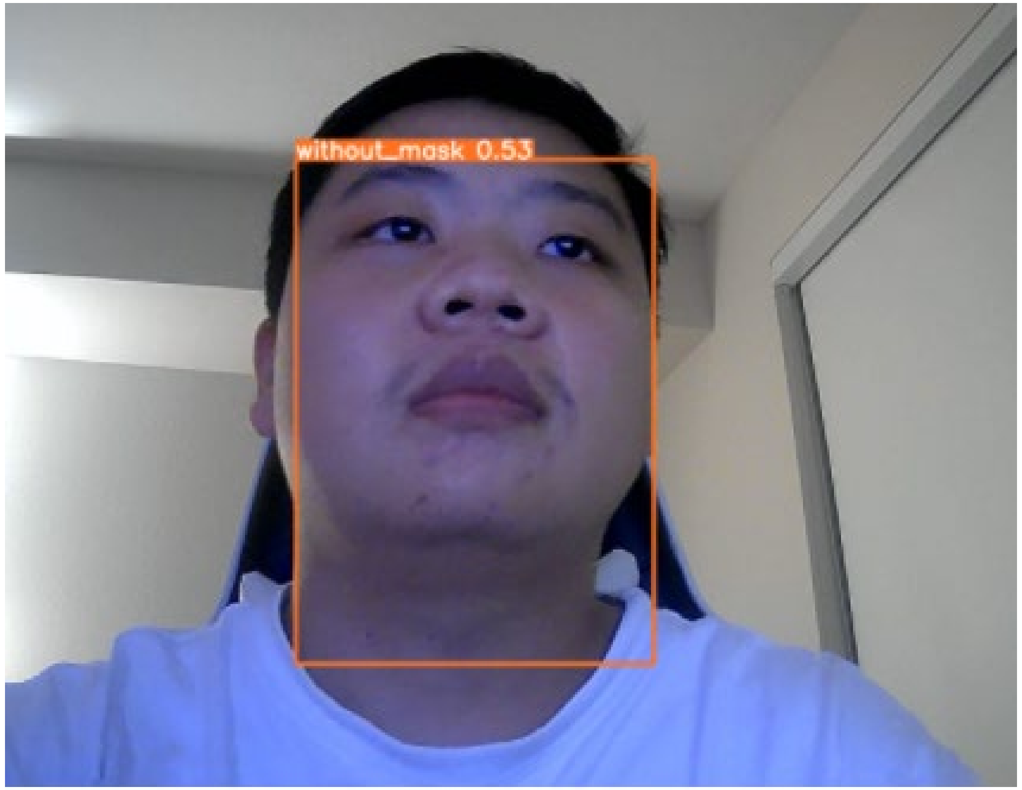


A probability distribution

In the field of machine learning, convolution is cross-correlation

Mask detection using a CNN





Student: Zipei Chen, CSC411

1D signal cross-correlation

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

kernel

w_0	w_1	w_2

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain $\boldsymbol{y_0}$

Padding a

Input Signal

a	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

w_0	w_1	w_2

$$y_0 = \boldsymbol{a} \times w_0 + x_0 \times w_1 + x_1 \times w_2$$

a = 0 or $\boldsymbol{x_0}$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain $\boldsymbol{y_1}$

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

w_0	w_1	w_2

$$\boldsymbol{y_1 = x_0 \times w_0 + x_1 \times w_1 + x_2 \times w_2}$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain $\boldsymbol{y_2}$

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

	w_0	w_1	w_2						

$$\boldsymbol{y_2 = x_1 \times w_0 + x_2 \times w_1 + x_3 \times w_2}$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain $\boldsymbol{y_3}$

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

w_0	w_1	w_2

$$\boldsymbol{y_3 = x_2 \times w_0 + x_3 \times w_1 + x_4 \times w_2}$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain y_4

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

w_0	w_1	w_2

$$y_4 = x_3 \times w_0 + x_4 \times w_1 + x_5 \times w_2$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain y_5

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

w_0	w_1	w_2

$$y_5 = x_4 \times w_0 + x_5 \times w_1 + x_6 \times w_2$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain y_6

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

w_0	w_1	w_2

$$y_6 = x_5 \times w_0 + x_6 \times w_1 + x_7 \times w_2$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain $\boldsymbol{y_7}$

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

w_0	w_1	w_2

$$\boldsymbol{y_7 = x_6 \times w_0 + x_7 \times w_1 + x_8 \times w_2}$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain y_8

Input Signal

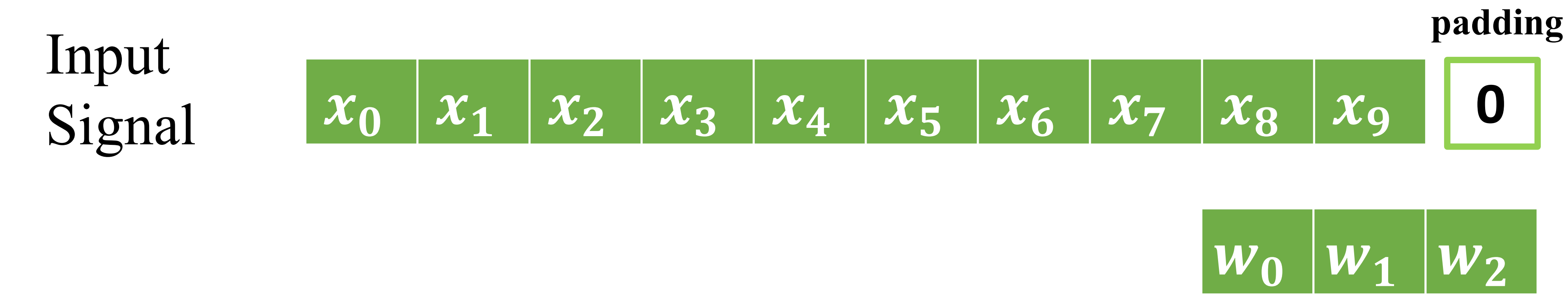
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
							w_0	w_1	w_2

$$
\begin{aligned}
y_8 = x_7 \times w_0 &+ \\
x_8 \times w_1 &+ \\
x_9 \times w_2 &
\end{aligned}
$$

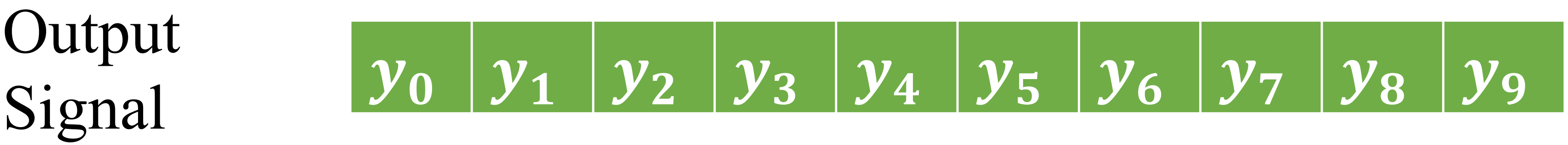
Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

obtain y_9



$$\begin{aligned} y_9 = \; & x_8 \times w_0 \\ & + x_9 \times w_1 \\ & + 0 \times w_2 \end{aligned}$$



Moving Average: cross-correlation with a special kernel

Input Signal

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9

kernel

w_0	w_1	w_2

average $\boldsymbol{w_0} = \boldsymbol{w_1} = \boldsymbol{w_2} = 1/3$

$$\begin{aligned} \boldsymbol{y_1} &= \boldsymbol{x_0} \times \boldsymbol{w_0} + \boldsymbol{x_1} \times \boldsymbol{w_1} + \boldsymbol{x_2} \times \boldsymbol{w_2} \\ &= (\boldsymbol{x_0} + \boldsymbol{x_1} + \boldsymbol{x_2})/3 \end{aligned}$$

Processed Signal

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9

2D Convolution with Padding

image **A**
(1 channel)

A[0,0]	A[0,1]	A[0,2]	A[0,3]	A[0,4]
A[1,0]	A[1,1]	A[1,2]	A[1,3]	A[1,4]
A[2,0]	A[2,1]	A[2,2]	A[2,3]	A[2,4]
A[3,0]	A[3,1]	A[3,2]	A[3,3]	A[3,4]
A[4,0]	A[4,1]	A[4,2]	A[4,3]	A[4,4]

$\otimes$

1 kernel **W**

W[0,0]	W[0,1]	W[0,2]
W[1,0]	W[1,1]	W[1,2]
W[2,0]	W[2,1]	W[2,2]

$\Rightarrow$

image **B**
"1 Feature Map"
(1 channel)

B[0,0]	B[0,1]	B[0,2]	B[0,3]	B[0,4]
B[1,0]	B[1,1]	B[1,2]	B[1,3]	B[1,4]
B[2,0]	B[2,1]	B[2,2]	B[2,3]	B[2,4]
B[3,0]	B[3,1]	B[3,2]	B[3,3]	B[3,4]
B[4,0]	B[4,1]	B[4,2]	B[4,3]	B[4,4]

image A

A[0,0] W[0,0]	A[0,1] W[0,1]	A[0,2] W[0,2]	A[0,3]	A[0,4]
A[1,0] W[1,0]	A[1,1] W[1,1]	A[1,2] W[1,2]	A[1,3]	A[1,4]
A[2,0] W[2,0]	A[2,1] W[2,1]	A[2,2] W[2,2]	A[2,3]	A[2,4]
A[3,0]	A[3,1]	A[3,2]	A[3,3]	A[3,4]
A[4,0]	A[4,1]	A[4,2]	A[4,3]	A[4,4]

- Multiply the two numbers at each pixel location
- Take the sum of products

image B

	B[1,1]			

$$
\begin{aligned}
B[1,1] = {} & W[0,0]\times A[0,0]+W[0,1]\times A[0,1]+W[0,2]\times A[0,2] \\
& +W[1,0]\times A[1,0]+W[1,1]\times A[1,1]+W[1,2]\times A[1,2] \\
& +W[2,0]\times A[2,0]+W[2,1]\times A[2,1]+W[2,2]\times A[2,2]
\end{aligned}
$$

image **A** (input)

A[0,0]	A[0,1] W[0,0]	A[0,2] W[0,1]	A[0,3] W[0,2]	A[0,4]
A[1,0]	A[1,1] W[1,0]	A[1,2] W[1,1]	A[1,3] W[1,2]	A[1,4]
A[2,0]	A[2,1] W[2,0]	A[2,2] W[2,1]	A[2,3] W[2,2]	A[2,4]
A[3,0]	A[3,1]	A[3,2]	A[3,3]	A[3,4]
A[4,0]	A[4,1]	A[4,2]	A[4,3]	A[4,4]

- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	**B[1,2]**		

$$\begin{aligned} B[1,2] = \ & W[0,0]\times A[0,1]+W[0,1]\times A[0,2]+W[0,2]\times A[0,3] \\ & +W[1,0]\times A[1,1]+W[1,1]\times A[1,2]+W[1,2]\times A[1,3] \\ & +W[2,0]\times A[2,1]+W[2,1]\times A[2,2]+W[2,2]\times A[2,3] \end{aligned}$$

image **A** (input)

A[0,0]	**A[0,1]**	**A[0,2]** W[0,0]	**A[0,3]** W[0,1]	**A[0,4]** W[0,2]
A[1,0]	**A[1,1]**	**A[1,2]** W[1,0]	**A[1,3]** W[1,1]	**A[1,4]** W[1,2]
A[2,0]	**A[2,1]**	**A[2,2]** W[2,0]	**A[2,3]** W[2,1]	**A[2,4]** W[2,2]
A[3,0]	**A[3,1]**	**A[3,2]**	**A[3,3]**	**A[3,4]**
A[4,0]	**A[4,1]**	**A[4,2]**	**A[4,3]**	**A[4,4]**

- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	B[1,2]	**B[1,3]**	

$$
\begin{aligned}
B[1,3] = \; & W[0,0]\times A[0,2]+W[0,1]\times A[0,3]+W[0,2]\times A[0,4] \\
& +W[1,0]\times A[1,2]+W[1,1]\times A[1,3]+W[1,2]\times A[1,4] \\
& +W[2,0]\times A[2,2]+W[2,1]\times A[2,3]+W[2,2]\times A[2,4]
\end{aligned}
$$

image **A** (input)

A[0,0]	**A[0,1]**	**A[0,2]**	**A[0,3]**	**A[0,4]**
A[1,0] **W[0,0]**	**A[1,1]** **W[0,1]**	**A[1,2]** **W[0,2]**	**A[1,3]**	**A[1,4]**
A[2,0] **W[1,0]**	**A[2,1]** **W[1,1]**	**A[2,2]** **W[1,2]**	**A[2,3]**	**A[2,4]**
A[3,0] **W[2,0]**	**A[3,1]** **W[2,1]**	**A[3,2]** **W[2,2]**	**A[3,3]**	**A[3,4]**
A[4,0]	**A[4,1]**	**A[4,2]**	**A[4,3]**	**A[4,4]**

- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	B[1,2]	B[1,3]	
	B[2,1]			

$$\begin{aligned} B[2,1] = \; & W[0,0]\times A[1,0] + W[0,1]\times A[1,1] + W[0,2]\times A[1,2] \\ & + W[1,0]\times A[2,0] + W[1,1]\times A[2,1] + W[1,2]\times A[2,2] \\ & + W[2,0]\times A[3,0] + W[2,1]\times A[3,1] + W[2,2]\times A[3,2] \end{aligned}$$

image **A** (input)

A[0,0]	**A[0,1]**	**A[0,2]**	**A[0,3]**	**A[0,4]**
A[1,0]	**A[1,1]** W[0,0]	**A[1,2]** W[0,1]	**A[1,3]** W[0,2]	**A[1,4]**
A[2,0]	**A[2,1]** W[1,0]	**A[2,2]** W[1,1]	**A[2,3]** W[1,2]	**A[2,4]**
A[3,0]	**A[3,1]** W[2,0]	**A[3,2]** W[2,1]	**A[3,3]** W[2,2]	**A[3,4]**
A[4,0]	**A[4,1]**	**A[4,2]**	**A[4,3]**	**A[4,4]**

- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	B[1,2]	B[1,3]	
	B[2,1]	**B[2,2]**		

$$
\begin{aligned}
B[2,2] = {} & W[0,0]\times A[1,1]+W[0,1]\times A[1,2]+W[0,2]\times A[1,3] \\
& +W[1,0]\times A[2,1]+W[1,1]\times A[2,2]+W[1,2]\times A[2,3] \\
& +W[2,0]\times A[3,1]+W[2,1]\times A[3,2]+W[2,2]\times A[3,3]
\end{aligned}
$$

image **A** (input)

A[0,0]	**A[0,1]**	**A[0,2]**	**A[0,3]**	**A[0,4]**
A[1,0]	**A[1,1]**	**A[1,2]** W[0,0]	**A[1,3]** W[0,1]	**A[1,4]** W[0,2]
A[2,0]	**A[2,1]**	**A[2,2]** W[1,0]	**A[2,3]** W[1,1]	**A[2,4]** W[1,2]
A[3,0]	**A[3,1]**	**A[3,2]** W[2,0]	**A[3,3]** W[2,1]	**A[3,4]** W[2,2]
A[4,0]	**A[4,1]**	**A[4,2]**	**A[4,3]**	**A[4,4]**

- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	B[1,2]	B[1,3]	
	B[2,1]	B[2,2]	**B[2,3]**	

$$
\begin{aligned}
B[2,3] = \; & W[0,0]\times A[1,2]+W[0,1]\times A[1,3]+W[0,2]\times A[1,4] \\
& +W[1,0]\times A[2,2]+W[1,1]\times A[2,3]+W[1,2]\times A[2,4] \\
& +W[2,0]\times A[3,2]+W[2,1]\times A[3,3]+W[2,2]\times A[3,4]
\end{aligned}
$$

image **A** (input)

A[0,0]	A[0,1]	A[0,2]	A[0,3]	A[0,4]
A[1,0]	A[1,1]	A[1,2]	A[1,3]	A[1,4]
A[2,0] W[0,0]	A[2,1] W[0,1]	A[2,2] W[0,2]	A[2,3]	A[2,4]
A[3,0] W[1,0]	A[3,1] W[1,1]	A[3,2] W[1,2]	A[3,3]	A[3,4]
A[4,0] W[2,0]	A[4,1] W[2,1]	A[4,2] W[2,2]	A[4,3]	A[4,4]

- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	B[1,2]	B[1,3]	
	B[2,1]	B[2,2]	B[2,3]	
	B[3,1]			

$$
\begin{aligned}
B[3,1] = \; & W[0,0]\times A[2,0]+W[0,1]\times A[2,1]+W[0,2]\times A[2,2] \\
& +W[1,0]\times A[3,0]+W[1,1]\times A[3,1]+W[1,2]\times A[3,2] \\
& +W[2,0]\times A[4,0]+W[2,1]\times A[4,1]+W[2,2]\times A[4,2]
\end{aligned}
$$

image **A** (input)

A[0,0]	**A[0,1]**	**A[0,2]**	**A[0,3]**	**A[0,4]**
A[1,0]	**A[1,1]**	**A[1,2]**	**A[1,3]**	**A[1,4]**
A[2,0]	**A[2,1]** W[0,0]	**A[2,2]** W[0,1]	**A[2,3]** W[0,2]	**A[2,4]**
A[3,0]	**A[3,1]** W[1,0]	**A[3,2]** W[1,1]	**A[3,3]** W[1,2]	**A[3,4]**
A[4,0]	**A[4,1]** W[2,0]	**A[4,2]** W[2,1]	**A[4,3]** W[2,2]	**A[4,4]**

- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	B[1,2]	B[1,3]	
	B[2,1]	B[2,2]	B[2,3]	
	B[3,1]	**B[3,2]**		

$$
\begin{aligned}
B[3,2] = \; & W[0,0]\times A[2,1]+W[0,1]\times A[2,2]+W[0,2]\times A[2,3] \\
& +W[1,0]\times A[3,1]+W[1,1]\times A[3,2]+W[1,2]\times A[3,3] \\
& +W[2,0]\times A[4,1]+W[2,1]\times A[4,2]+W[2,2]\times A[4,3]
\end{aligned}
$$

image **A** (input)

A[0,0]	**A[0,1]**	**A[0,2]**	**A[0,3]**	**A[0,4]**
A[1,0]	**A[1,1]**	**A[1,2]**	**A[1,3]**	**A[1,4]**
A[2,0]	**A[2,1]**	**A[2,2]** **W[0,0]**	**A[2,3]** **W[0,1]**	**A[2,4]** **W[0,2]**
A[3,0]	**A[3,1]**	**A[3,2]** **W[1,0]**	**A[3,3]** **W[1,1]**	**A[3,4]** **W[1,2]**
A[4,0]	**A[4,1]**	**A[4,2]** **W[2,0]**	**A[4,3]** **W[2,1]**	**A[4,4]** **W[2,2]**

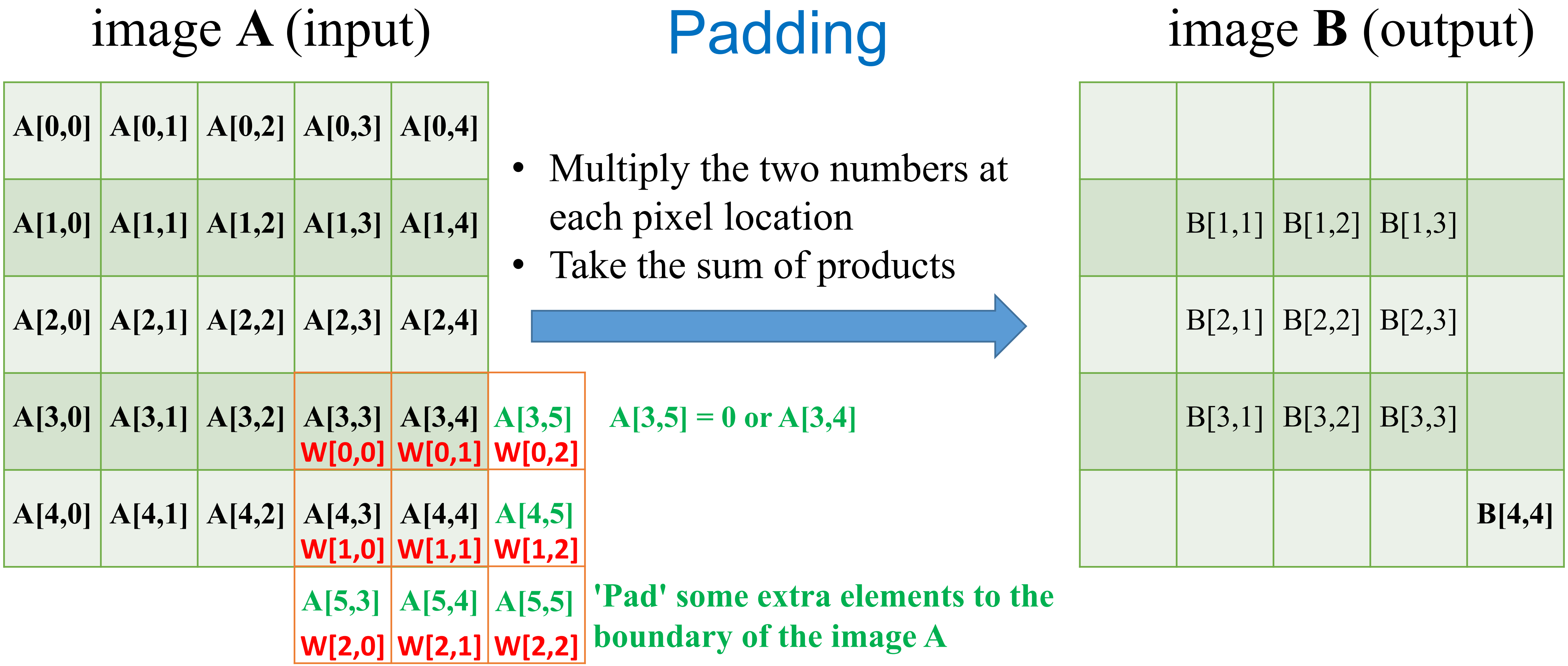
- Multiply the two numbers at each pixel location
- Take the sum of products

image **B** (output)

	B[1,1]	B[1,2]	B[1,3]	
	B[2,1]	B[2,2]	B[2,3]	
	B[3,1]	B[3,2]	**B[3,3]**	

$$
\begin{aligned}
B[3,3] = \; & W[0,0]\times A[2,2]+W[0,1]\times A[2,3]+W[0,2]\times A[2,4] \\
& +W[1,0]\times A[3,2]+W[1,1]\times A[3,3]+W[1,2]\times A[3,4] \\
& +W[2,0]\times A[4,2]+W[2,1]\times A[4,3]+W[2,2]\times A[4,4]
\end{aligned}
$$

Padding



$$
\begin{aligned}
B[4,4] = {} & W[0,0]\times A[3,3]+W[0,1]\times A[3,4]+W[0,2]\times A[3,5] \\
& +W[1,0]\times A[4,3]+W[1,1]\times A[4,4]+W[1,2]\times A[4,5] \\
& +W[2,0]\times A[5,3]+W[2,1]\times A[5,4]+W[2,2]\times A[5,5]
\end{aligned}
$$

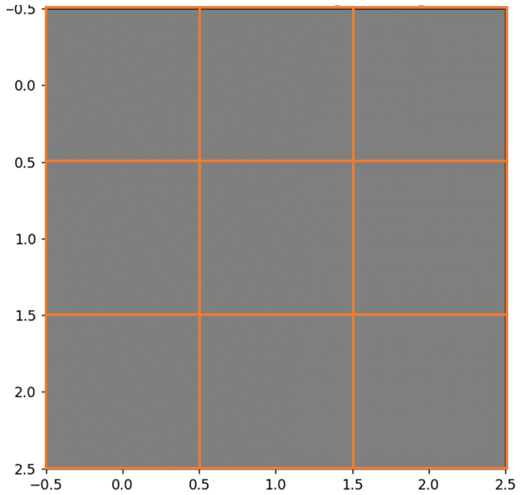
Run 2D_Image_Processing_Convolution.ipynb

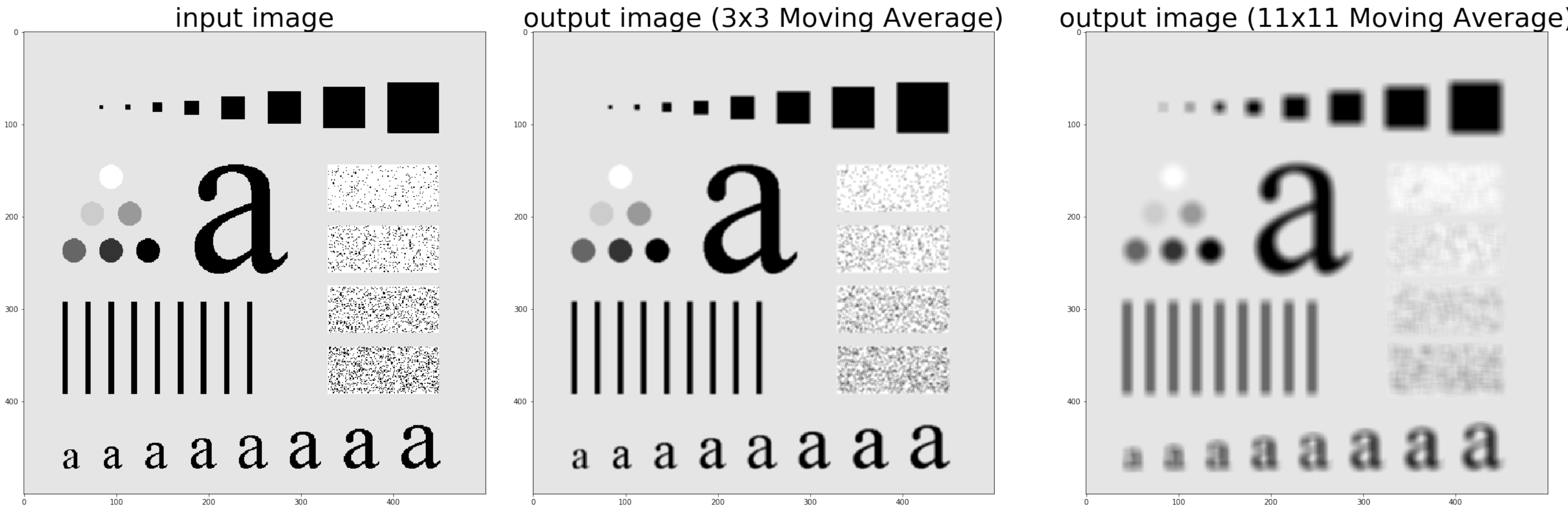
2D Moving Average Kernel

2D moving average kernel

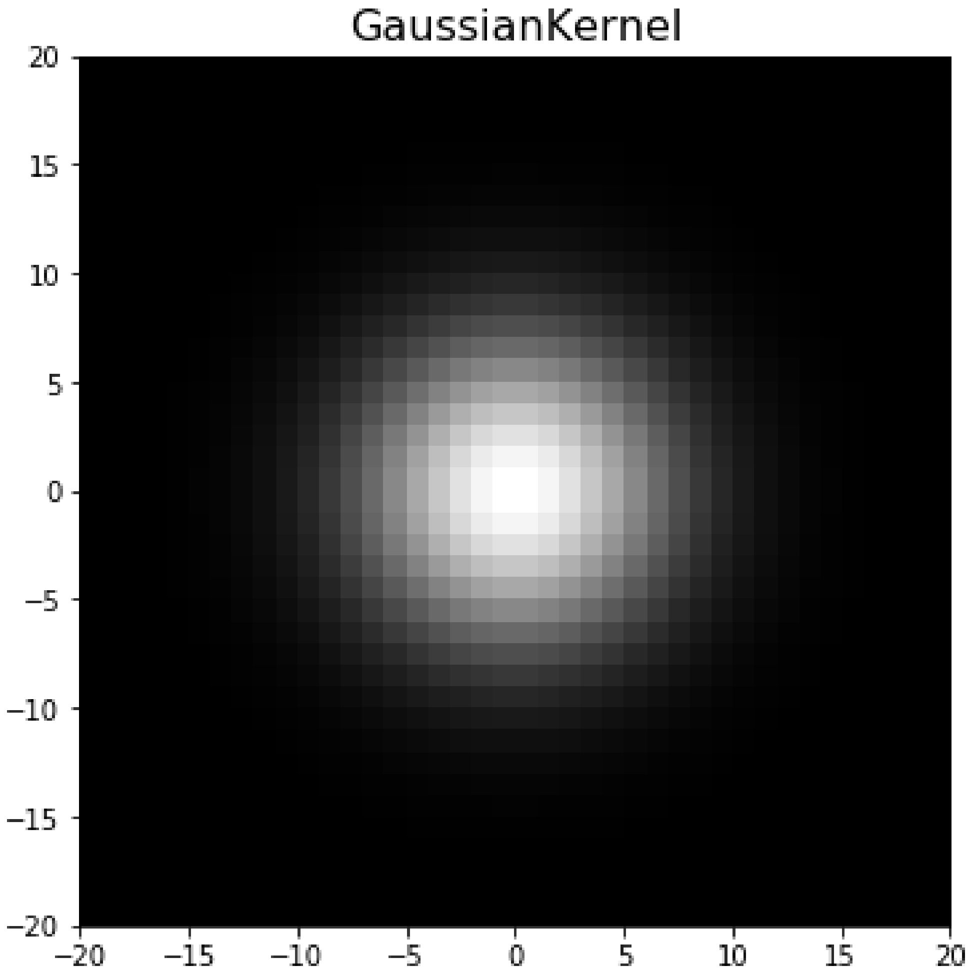
1/9	**1/9**	**1/9**
1/9	**1/9**	**1/9**
1/9	**1/9**	**1/9**

a 2D kernel is a small 2D image




input image
output image (3x3 Moving Average)
output image (11x11 Moving Average)
0
100
200
300
400

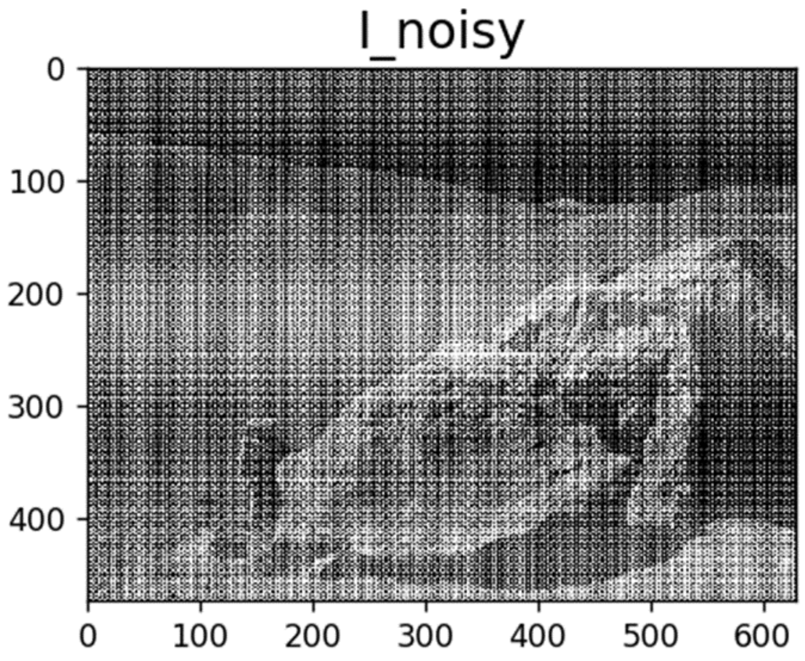
2D Gaussian Kernel for Image Denoising



The shape of the Kernel is (40, 40)

$$g[i,j] = e^{\frac{-(i^2+j^2)}{2\sigma^2}}$$

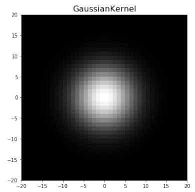
2D 'Convolution' for image processing

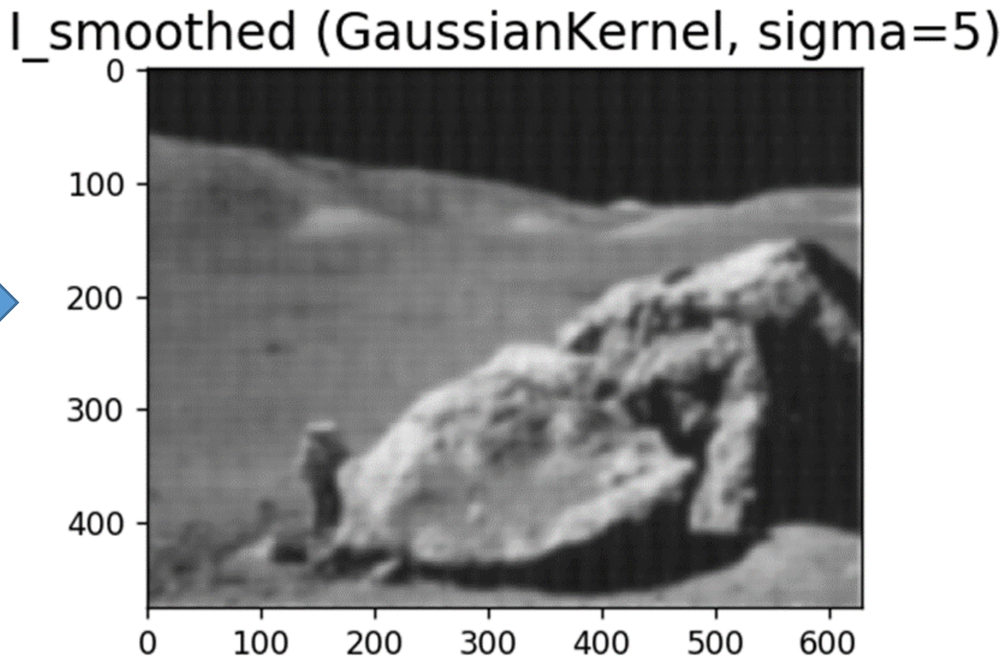


shape=474x630

⊗

2D Gaussian kernel





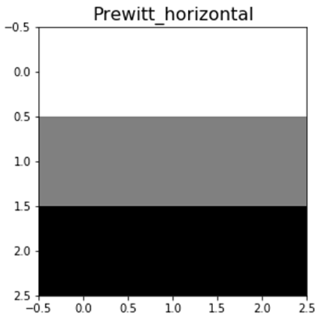
Convolution with a 2D Gaussian kernel to remove noises from the image

2D 'Convolution' for image processing

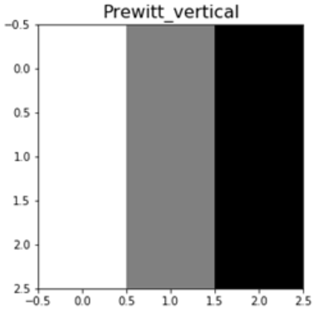
Convolution with 2D Prewitt kernels to detect object edges

```
Prewitt_horizontal = np.array([[1, 1, 1],
                               [0, 0, 0],
                               [-1,-1,-1]]
```

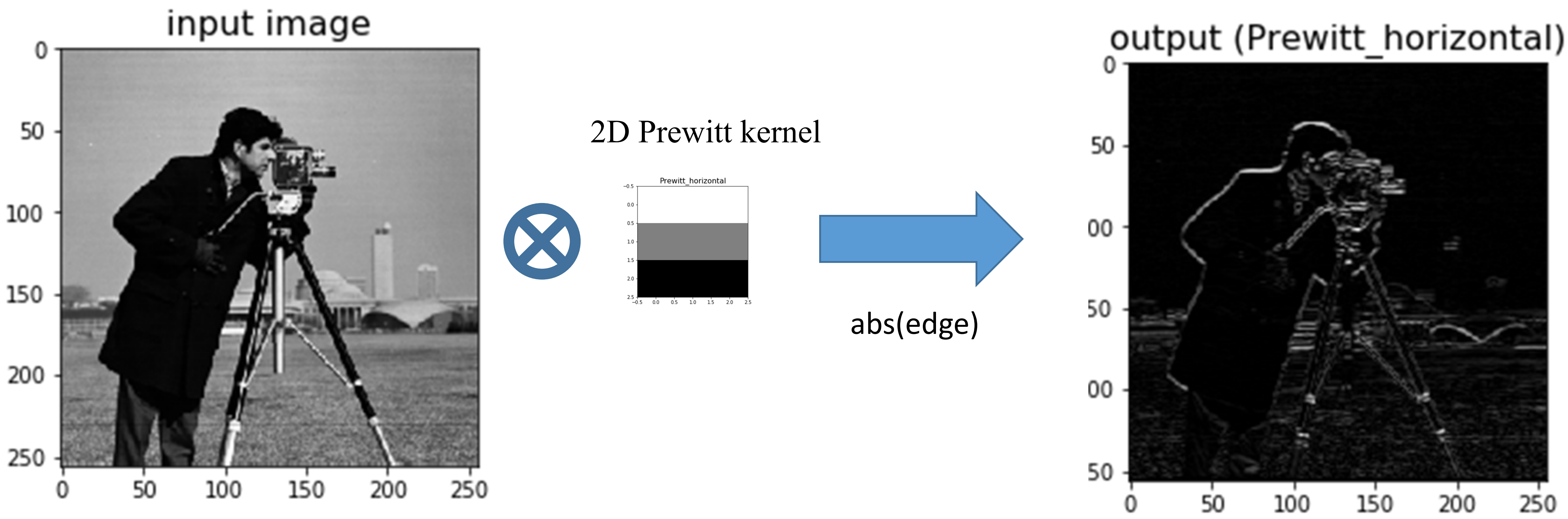
2D Prewitt kernel



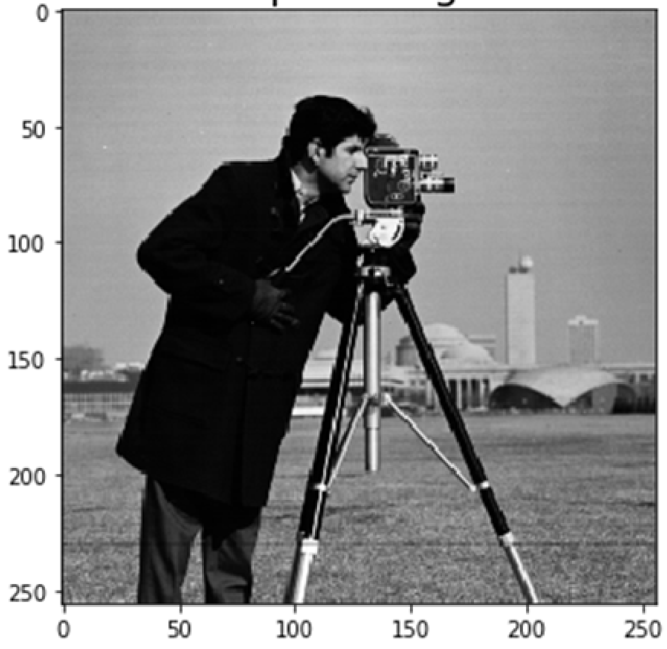

```
Prewitt_vertical = np.array([[1, 0, -1],
                             [1, 0, -1],
                             [1, 0, -1]],
```



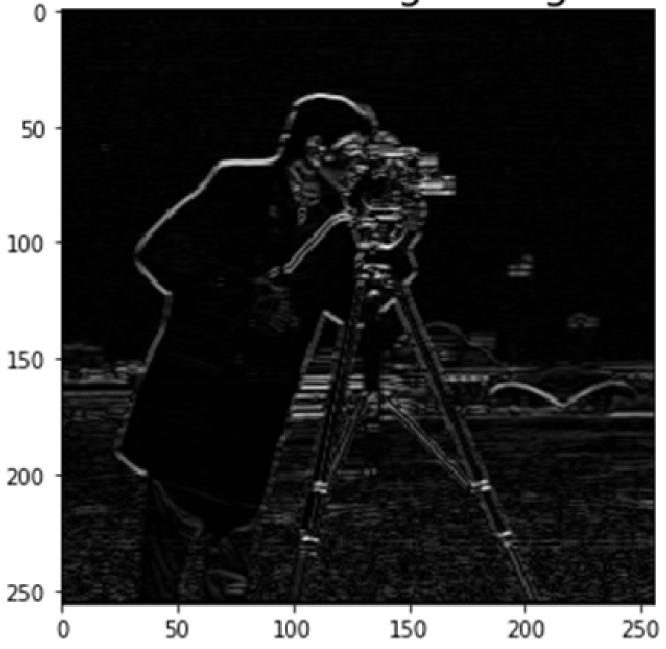
2D 'Convolution' for image processing

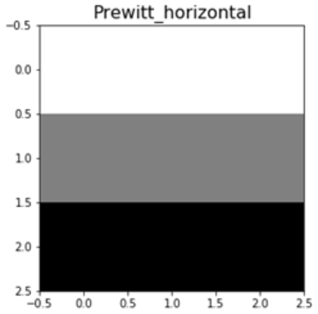


Convolution with a 2D Prewitt kernel to detect object edges input image



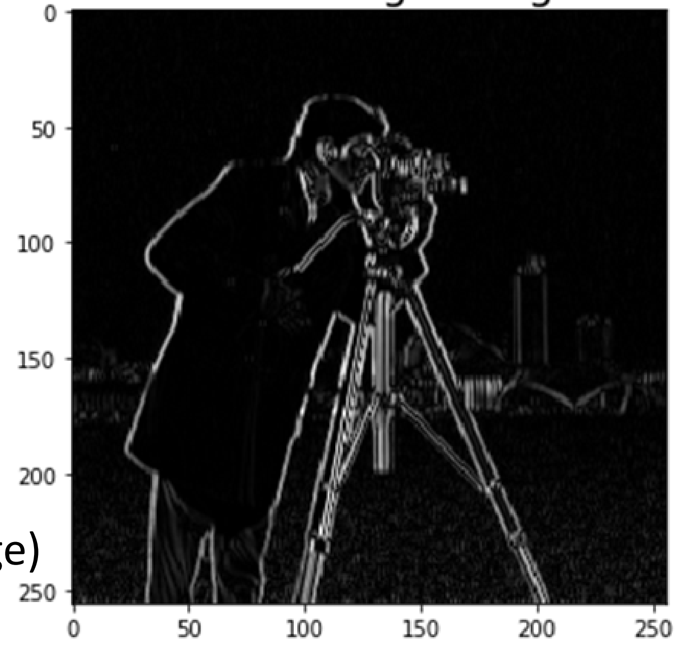
horizontal edge image

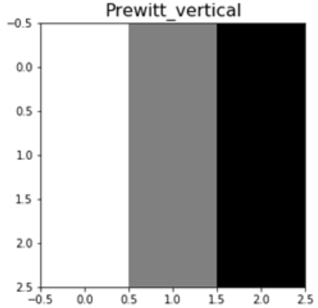




abs(edge)

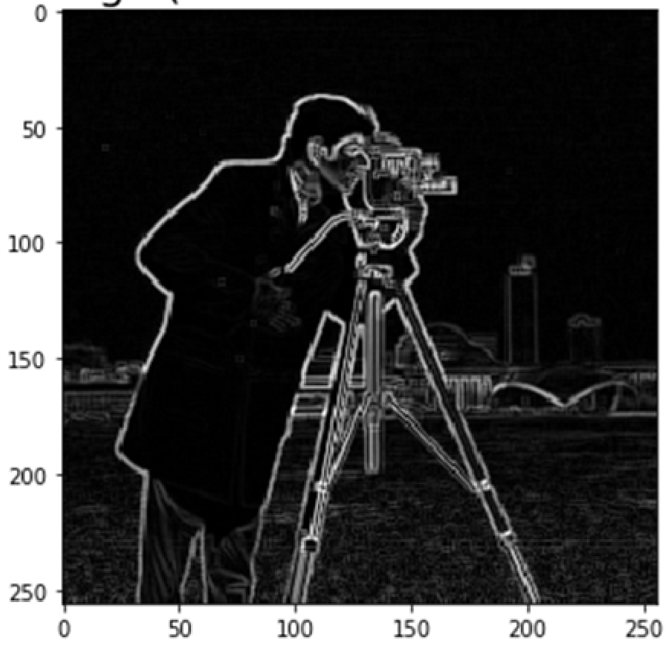
vertical edge image



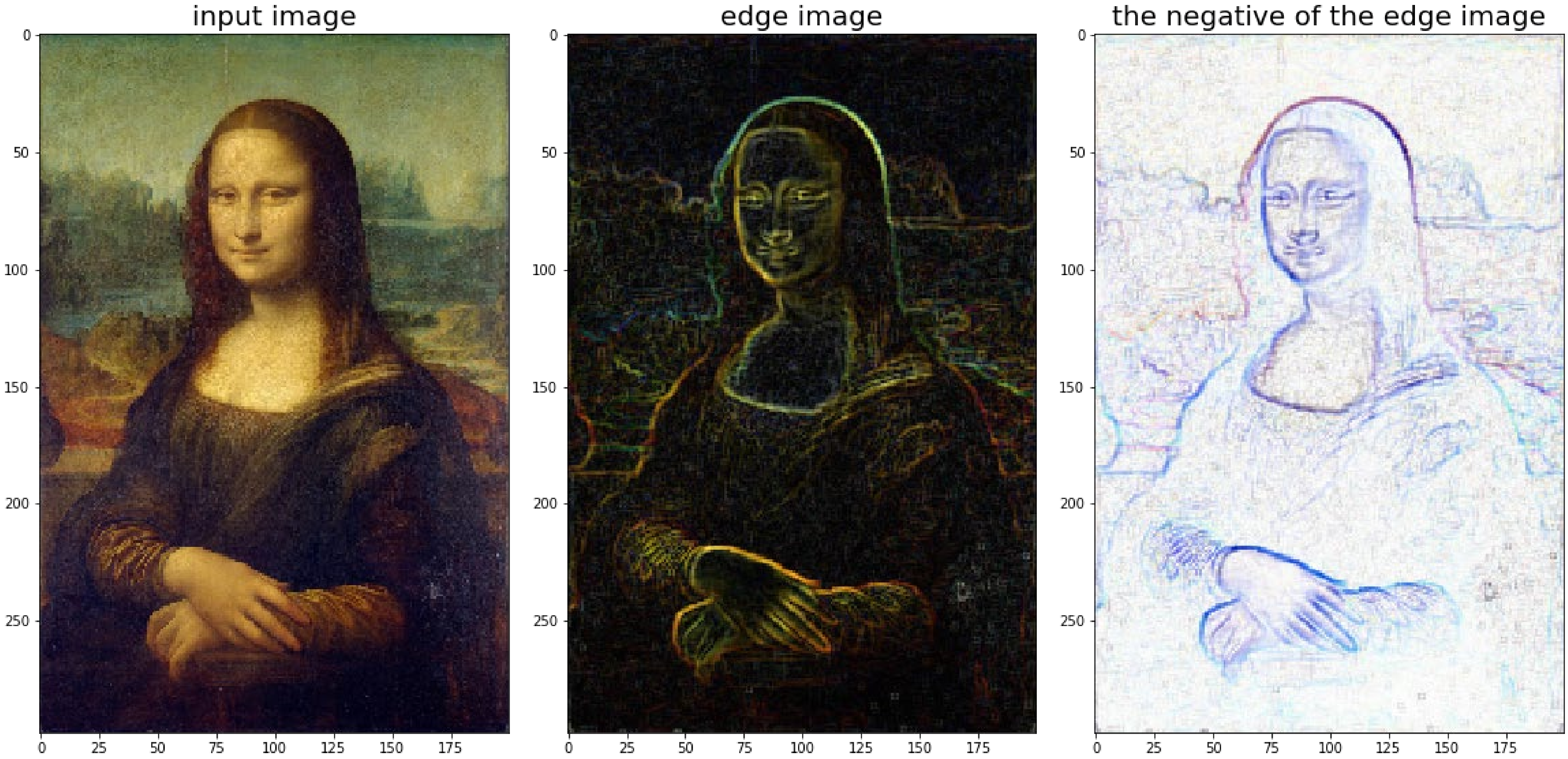


abs(edge)

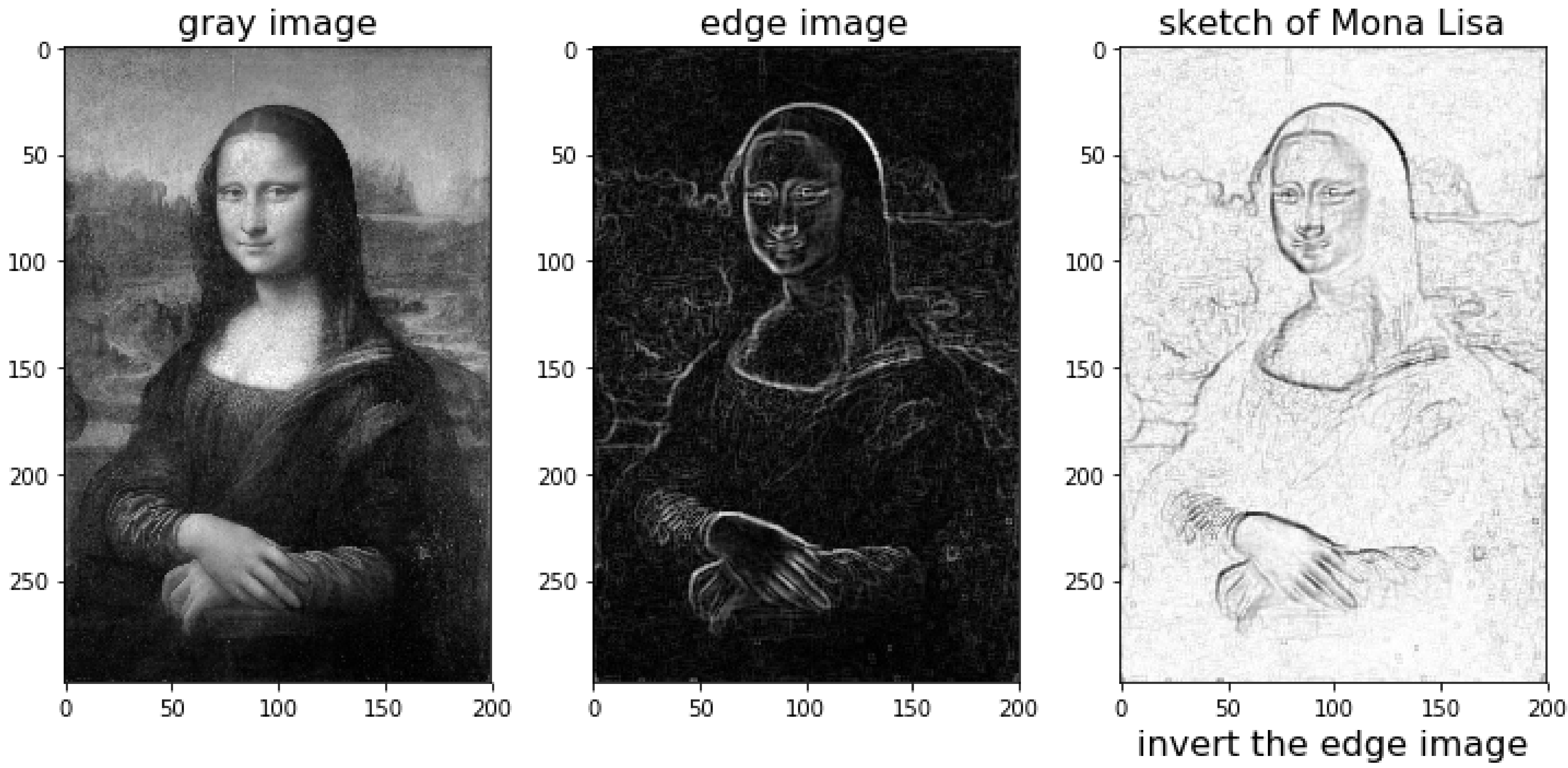
edge image (vertical and horizontal edges)



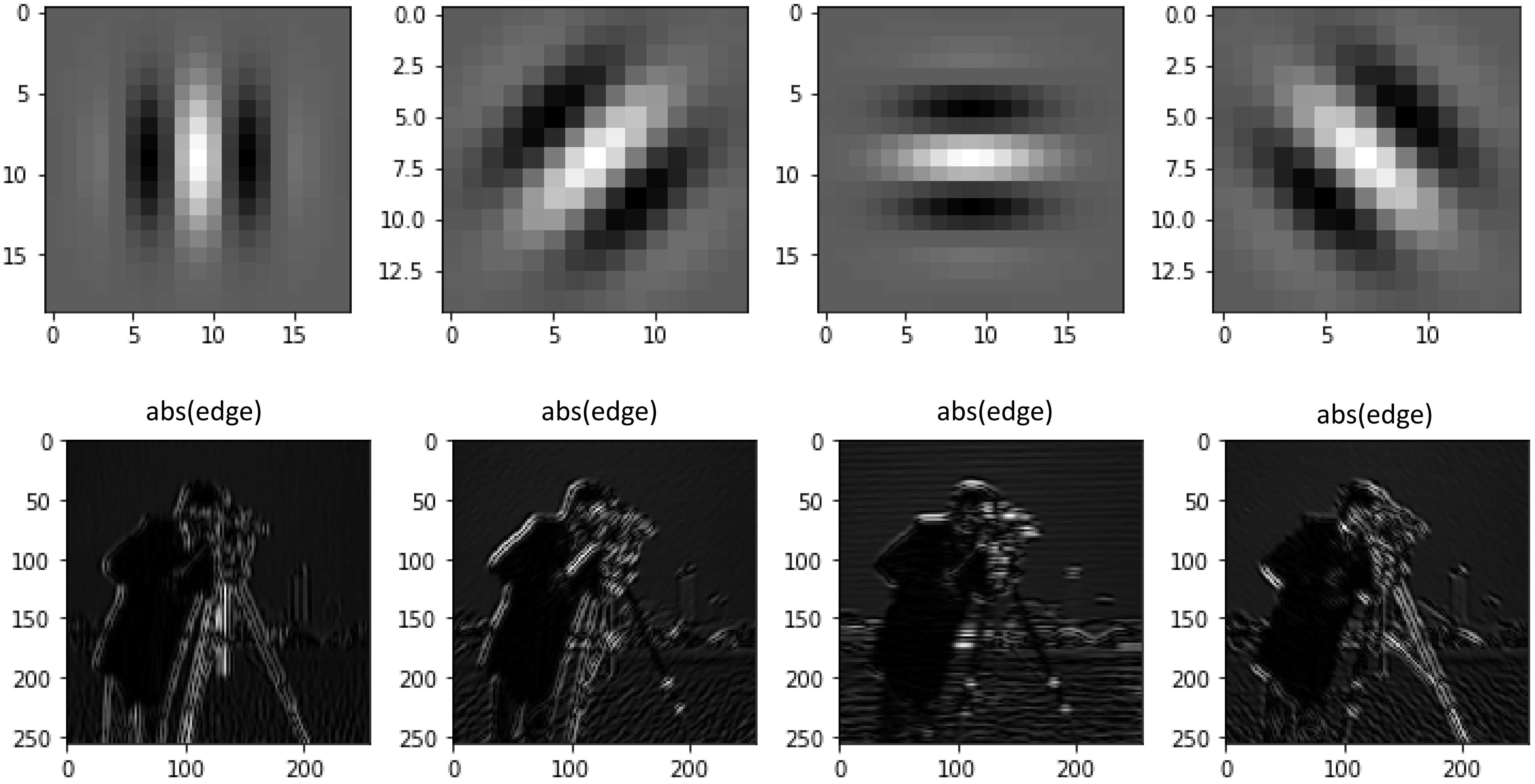
horizontal edge image + vertical edge image



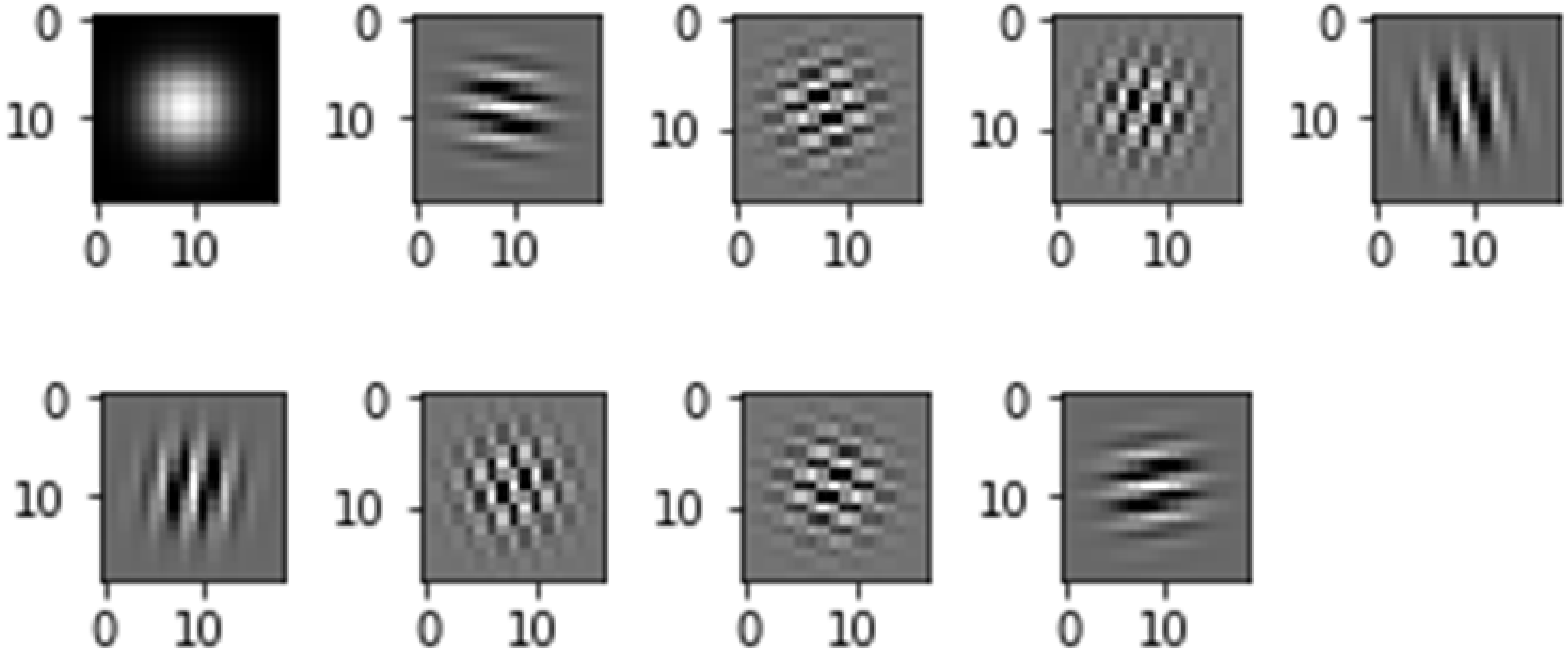
invert the edge image


gray image
edge image
sketch of Mona Lisa
0
50
100
150
200
250
0
50
100
150
200
invert the edge image

Run
2D_Image_Processing_Convolution_Gabor.ipynb


abs(edge)
abs(edge)
abs(edge)
abs(edge)

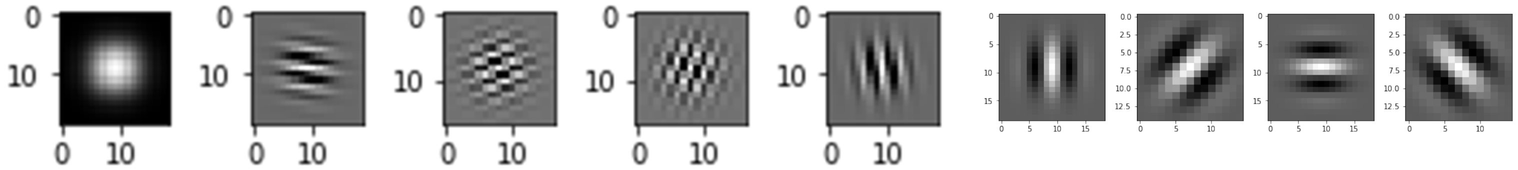
Gabor Kernels for Image Feature (texture) Detection



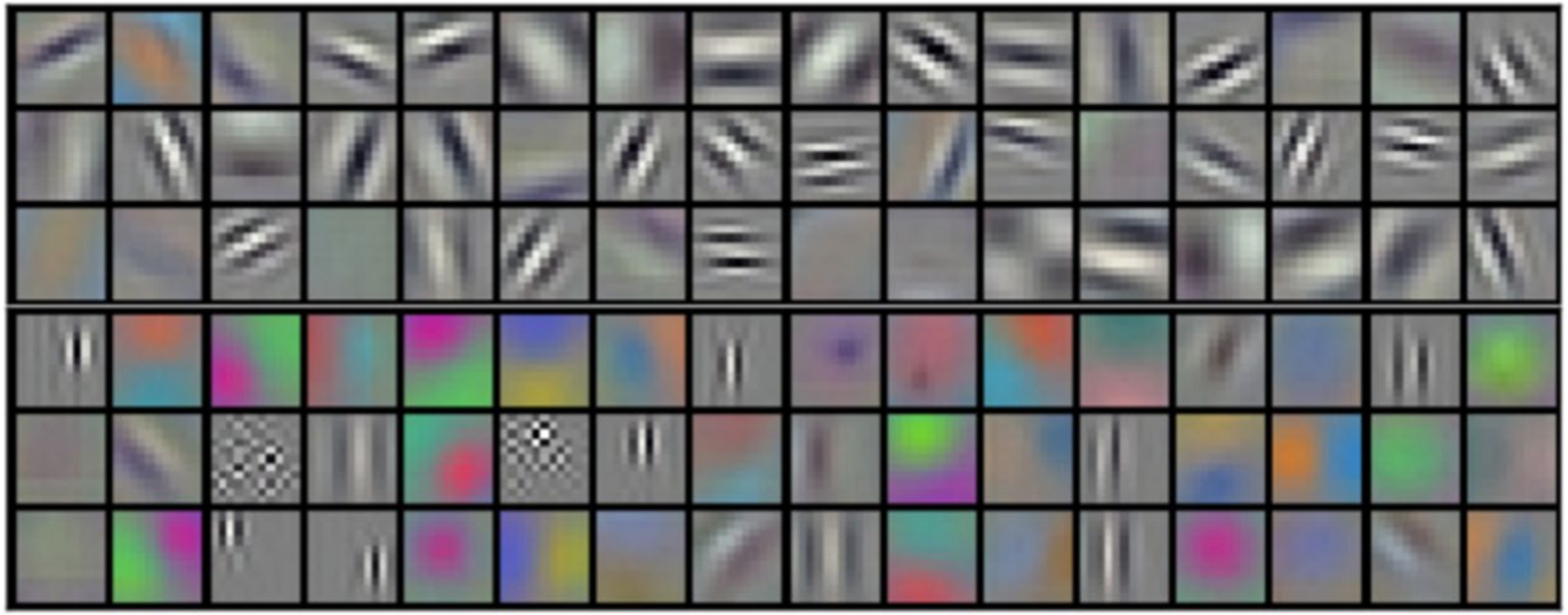
A Gabor kernel is a Gaussian function multiplied by a sin/cos wave
https://web.archive.org/web/20180127125930/http://mplab.ucsd.edu/tutorials/gabor.pdf

Gabor Kernels vs Kernels Learned from Data

Gabor kernels are defined by math equations.

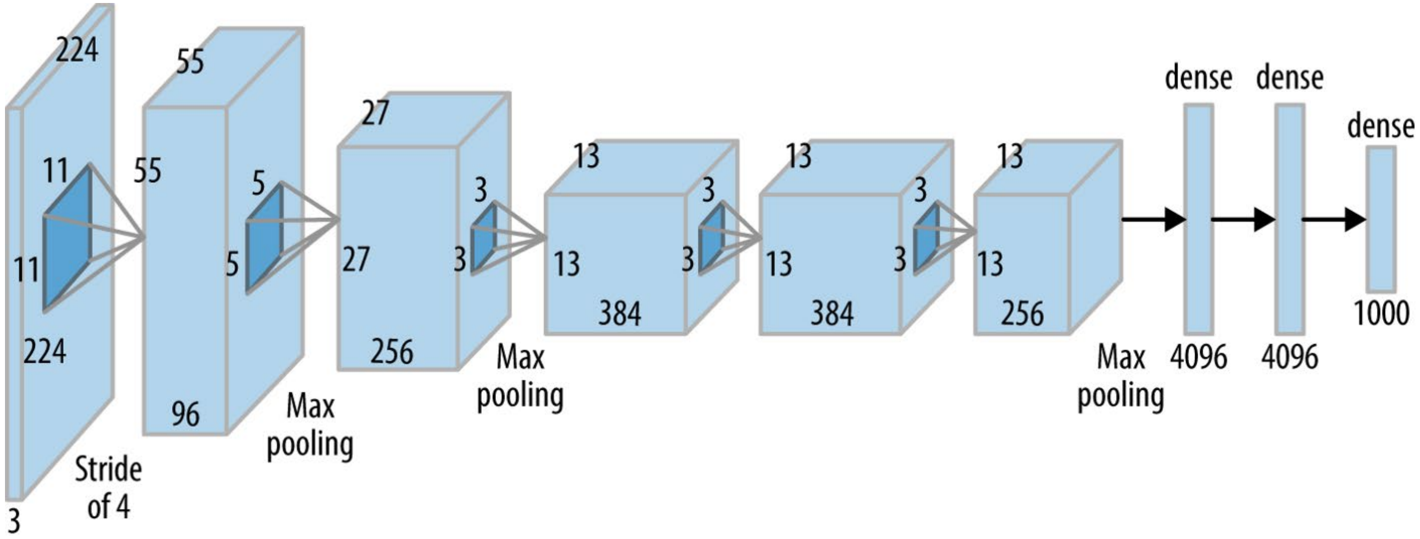


In deep learning, convolution kernels are learned from data



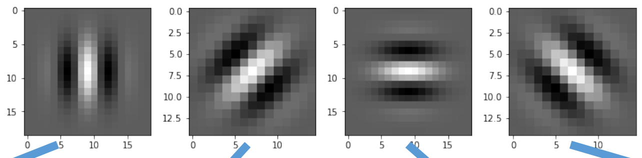
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

The Alexnet (2D CNN) for image classification

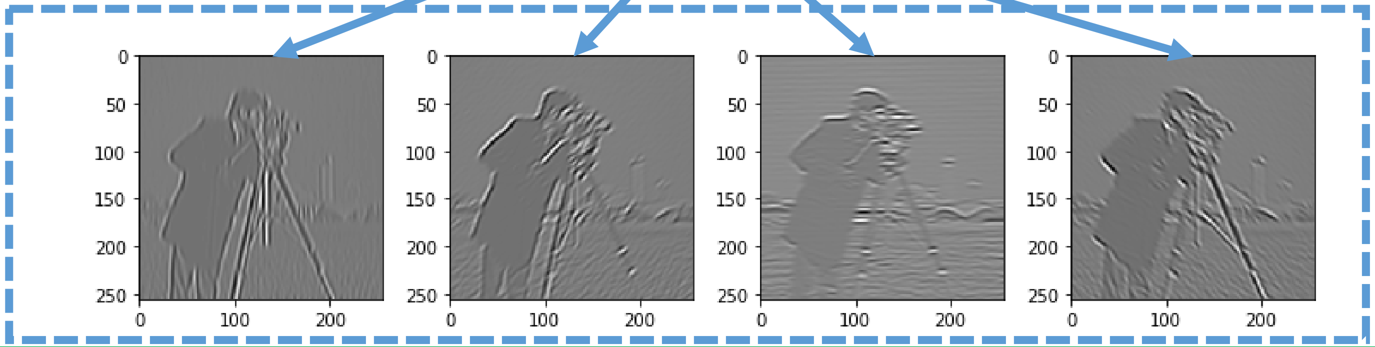


output: “people”

Explanation of the first layer of a convolutional neural network

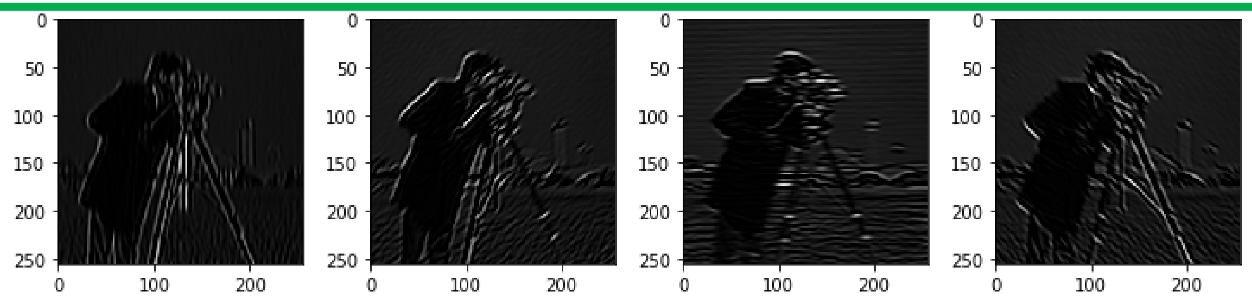


Let's assume the first layer only has four kernels



each edge image has positive pixels and negative pixels

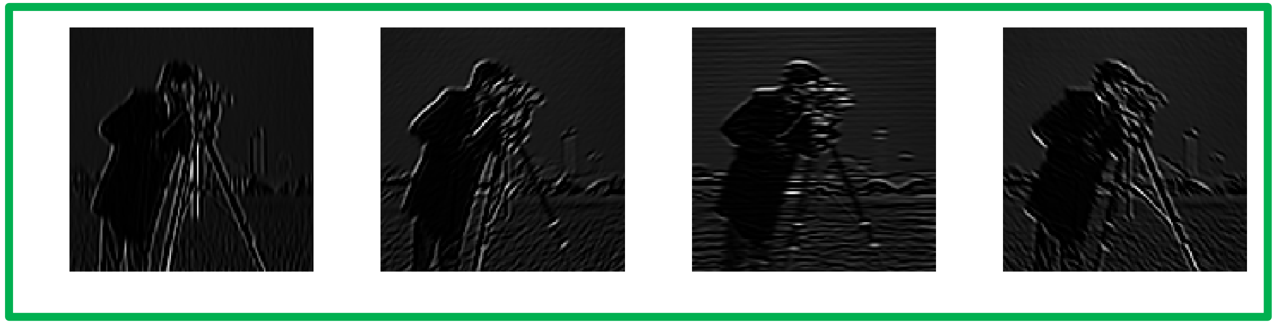
Modify the value of every pixel using a nonlinear activation function:
Relu: f(p) = p if p >0; f(p) = 0 if p <=0
If the value of a pixel is less than 0, then set the pixel value to zero



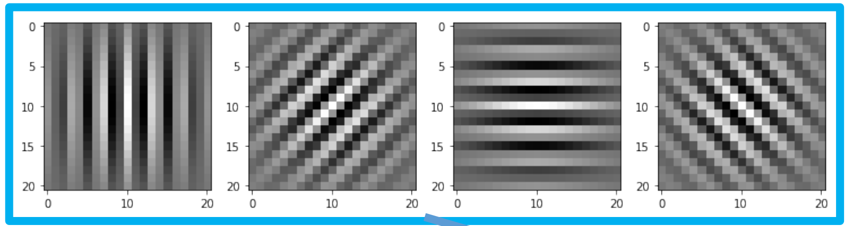
store the four images in an array (tensor), each of the output images is called a feature map.

An Explanation of the Operations in the second layer of a convolutional neural network (CNN)

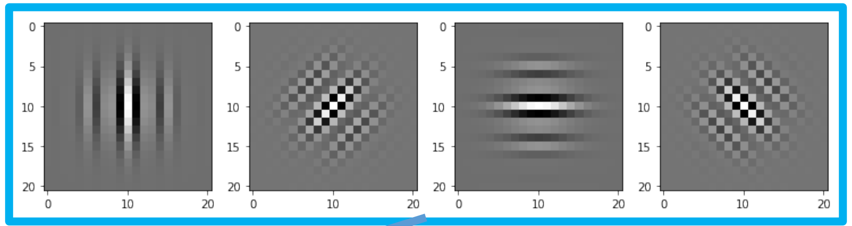
a 4-channel image (input to the second layer)



Let's assume the second layer has two kernels (each one has 4 channels)



W1

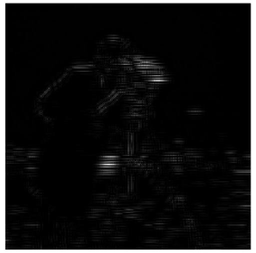


W2

the physical sizes of kernels in the second layer are usually larger than the sizes of kernels in the first layer

B1

B2



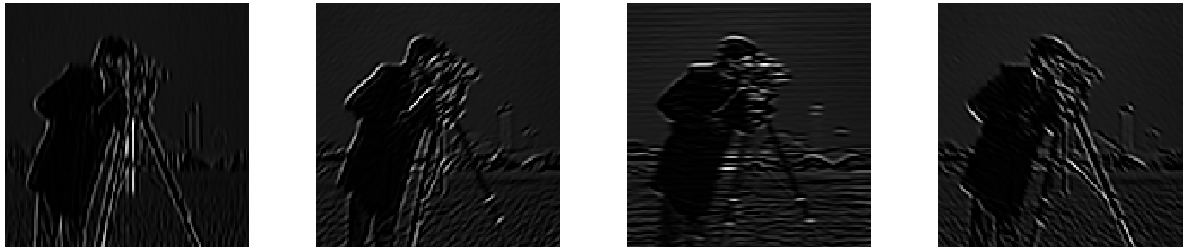
Attention:
the kernels of the second layer may not look like Gabor kernels in many machine learning applications.

N kernels are needed to generate *N* feature maps

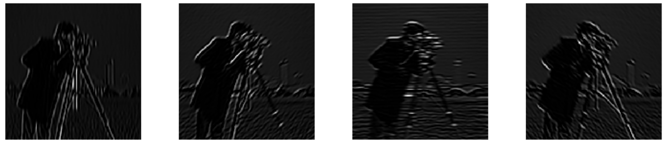
Input Image (an array / tensor)

The First Layer of a CNN

4 kernels



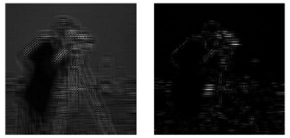
4 Feature Maps => an array (tensor)



Downsized Feature Maps

The Second Layer of a CNN

2 kernels



2 Feature Maps => an array (tensor)

2D_Image_Processing_Convolution_Gabor_and_CNN.ipynb

LeNet-5

PROC. OF THE IEEE, NOVEMBER 1998

1998

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

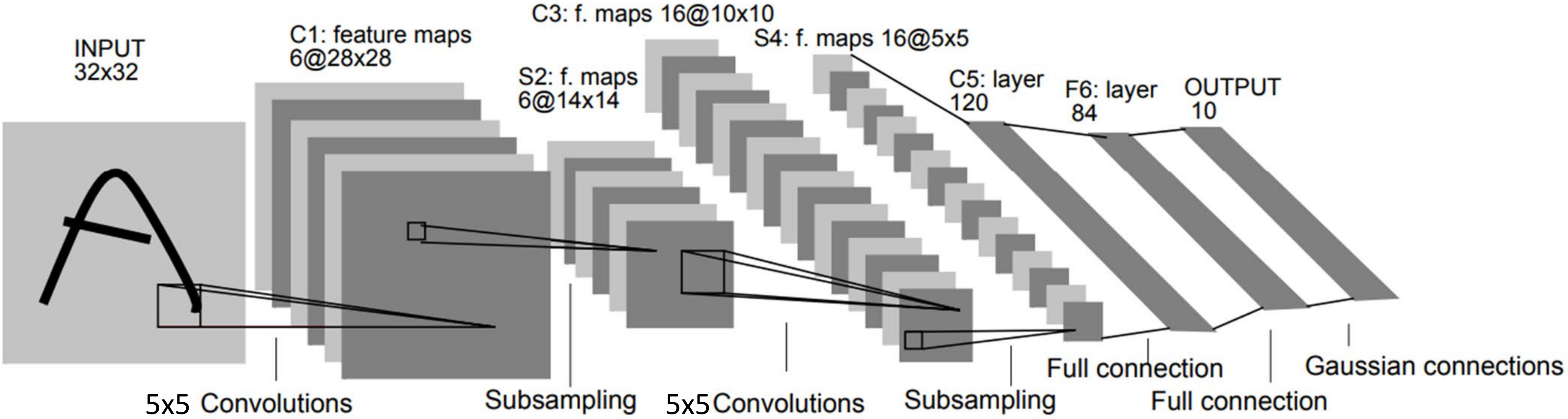


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

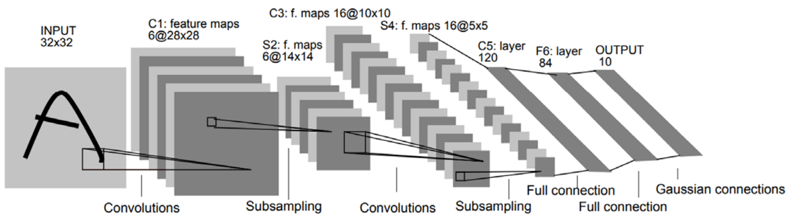


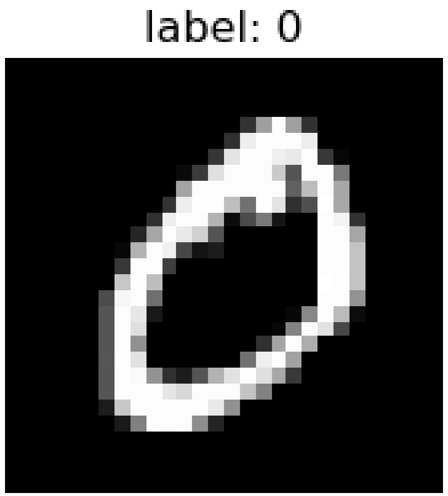
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Keras

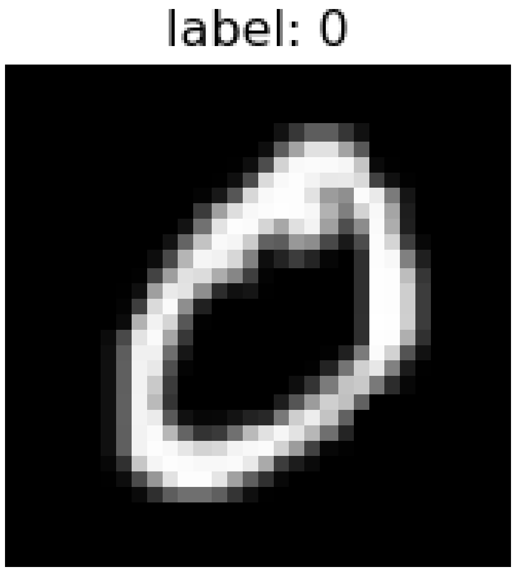
```
model = Sequential()
model.add(Conv2D(filters=6, kernel_size=(5,5), strides=(1,1), padding='valid', activation = 'relu', input_shape=(32,32,1)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=16, kernel_size=(5,5), strides=(1,1), padding='valid', activation = 'relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=120, kernel_size=(5,5), strides=(1,1), padding='valid', activation = 'relu'))
model.add(Flatten())
model.add(Dense(units=84, activation='relu'))
model.add(Dense(units=10,  activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer=SGD(lr=0.01, momentum=0.9), metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 1, 1, 120)         48120
_________________________________________________________________
flatten (Flatten)            (None, 120)               0
_________________________________________________________________
dense (Dense)                (None, 84)                10164
_________________________________________________________________
dense_1 (Dense)              (None, 10)                850
=================================================================
```

LeNet-5: input size is 32x32



resize to 32x32



28x28 in MNIST dataset

```
from skimage.transform import resize
```

LeNet5_Keras.ipynb

https://adamharley.com/nn_vis/