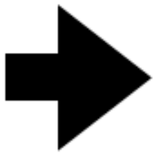
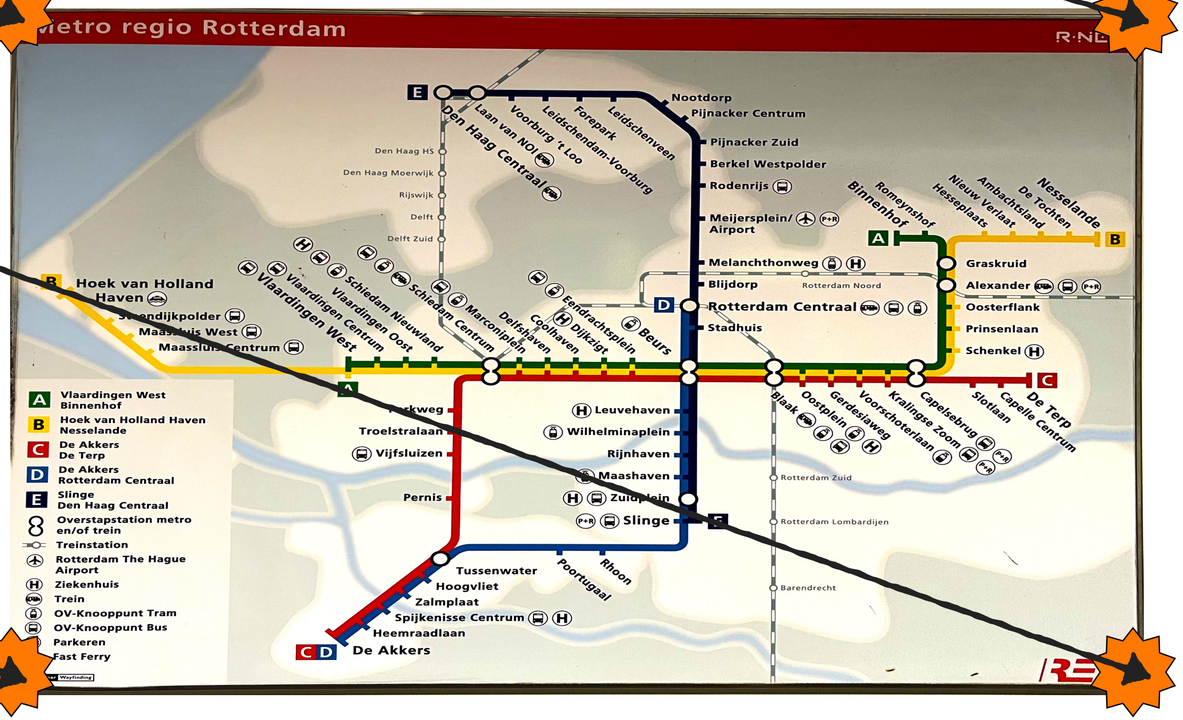


Lecture 11-2: Computer Vision

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

Image Transformation - Perspective Transformation



How To Define a Coordinate?

In this section, we will introduce some basic image transformation operations, including translation, rotation, and scaling. These operations can change the position, orientation, and size of any given iamge. We can define a list of 2D points, $P \in \mathbb{R}^{3\times n}$, where n is the number of points, as below:

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{1}$$

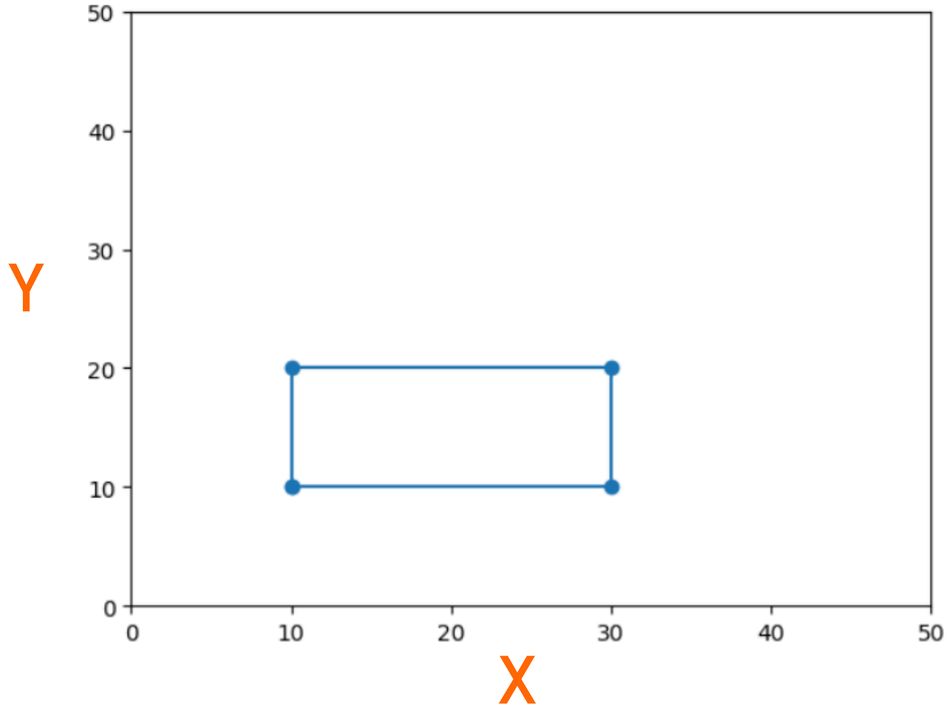
Noted that we need the third element, 1, for linear transformation. This form of vector is also known as "augmented vector".

```
# use lists to describe each vector: x, y, and intercept
x = [10, 30, 30, 10]
y = [10, 10, 20, 20]
i = [1] * 4

# put them in a numpy array
P = np.array([x,
              y,
              i])
```

```
P
 [[10 30 30 10]
 [10 10 20 20]
 [ 1  1  1  1]]
```

How To Define a Coordinate?




```
# use lists to describe each vector: x, y, and intercept
x = [10, 30, 30, 10]
y = [10, 10, 20, 20]
i = [1] * 4

# put them in a numpy array
P = np.array([x,
              y,
              i])
```

```
  P
X [[10 30 30 10]
Y [10 10 20 20]
  [ 1  1  1  1]]
```

Moving an Image

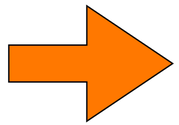
Moving an image can be represented as linear equations:

$$x' = x + t_x$$
$$y' = y + t_y$$
$$1 = 1$$

where t_x and t_y are the translation distances in the x and y directions, respectively. And x' and y' are the new coordinates of the point after translation.

Updated points, P’ | Transition matrix, T | Original points, P

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



$$\hat{P} = TP$$

Moving an Image - Example

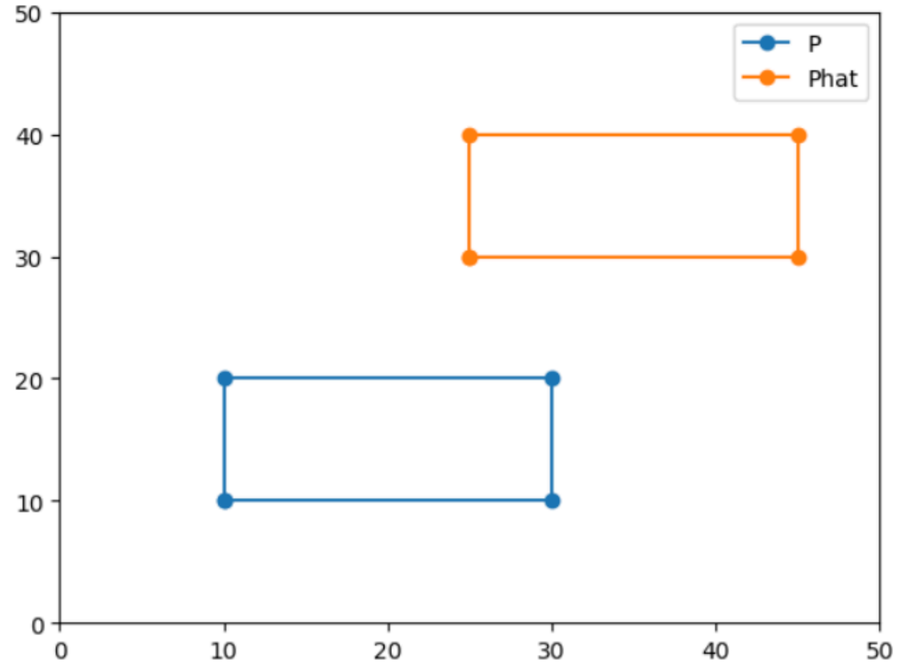
Here we can set $t_x = 15$ and $t_y = 20$ to move the image 15 pixels to the right and 20 pixels down. The translation matrix is:

$$\hat{P} = TP \qquad T = \begin{bmatrix} 1 & 0 & 15 \\ 0 & 1 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

```
tx = 15
ty = 20
T = np.array([[1, 0, tx],
              [0, 1, ty],
              [0, 0, 1]])
Phat = T @ P
```

```
P:
 [[10 30 30 10]
 [10 10 20 20]
 [ 1  1  1  1]]
Phat:
 [[25 45 45 25]
 [30 30 40 40]
 [ 1  1  1  1]]

<matplotlib.legend.Legend at 0×2f805ddf0>
```



Rotating an Image

If we futher define other elements in the transformation matrix, we are able to rotate an image by any specific angle. We can re-define the transformation as:

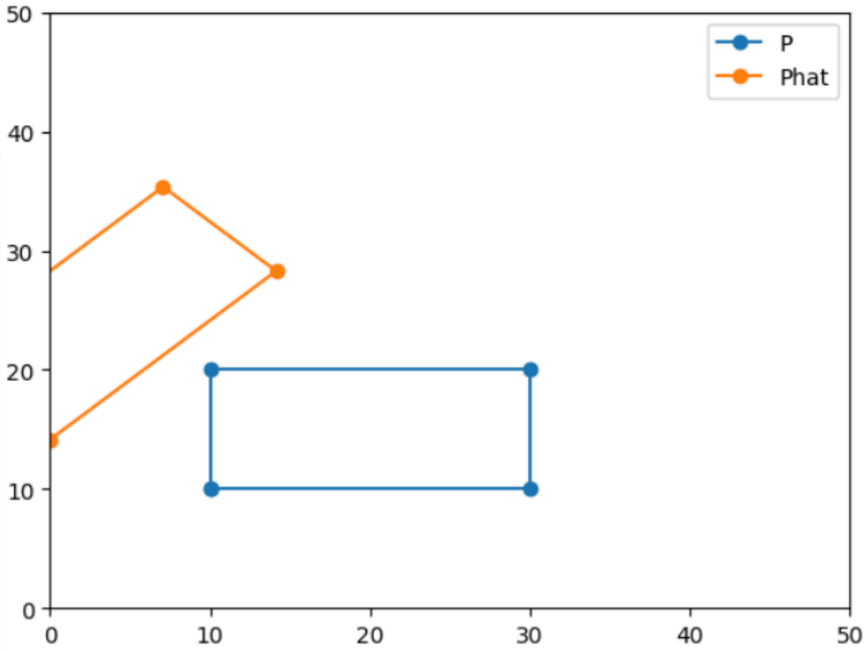
$$\hat{P} = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} P$$

where θ is the rotation angle.

We can rotate the image by 45 degrees to demonstrate the image rotation.

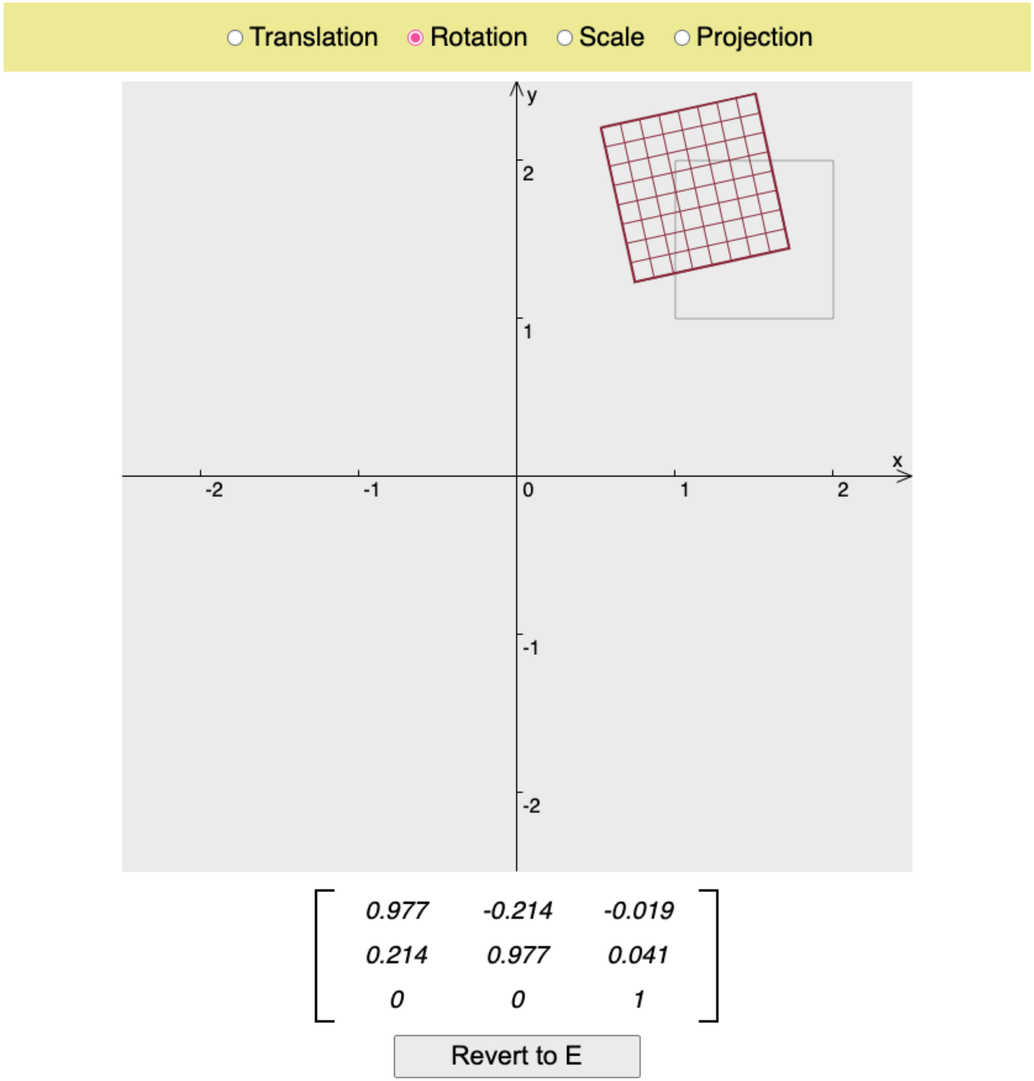
```
angle = 45
tx = 0
ty = 0
radian = angle * np.pi / 180 # 360 degrees = 2 * pi * radians
T = np.array([[np.cos(radian), -np.sin(radian), tx],
              [np.sin(radian), np.cos(radian), ty],
              [0, 0, 1]])

Phat = T @ P
```



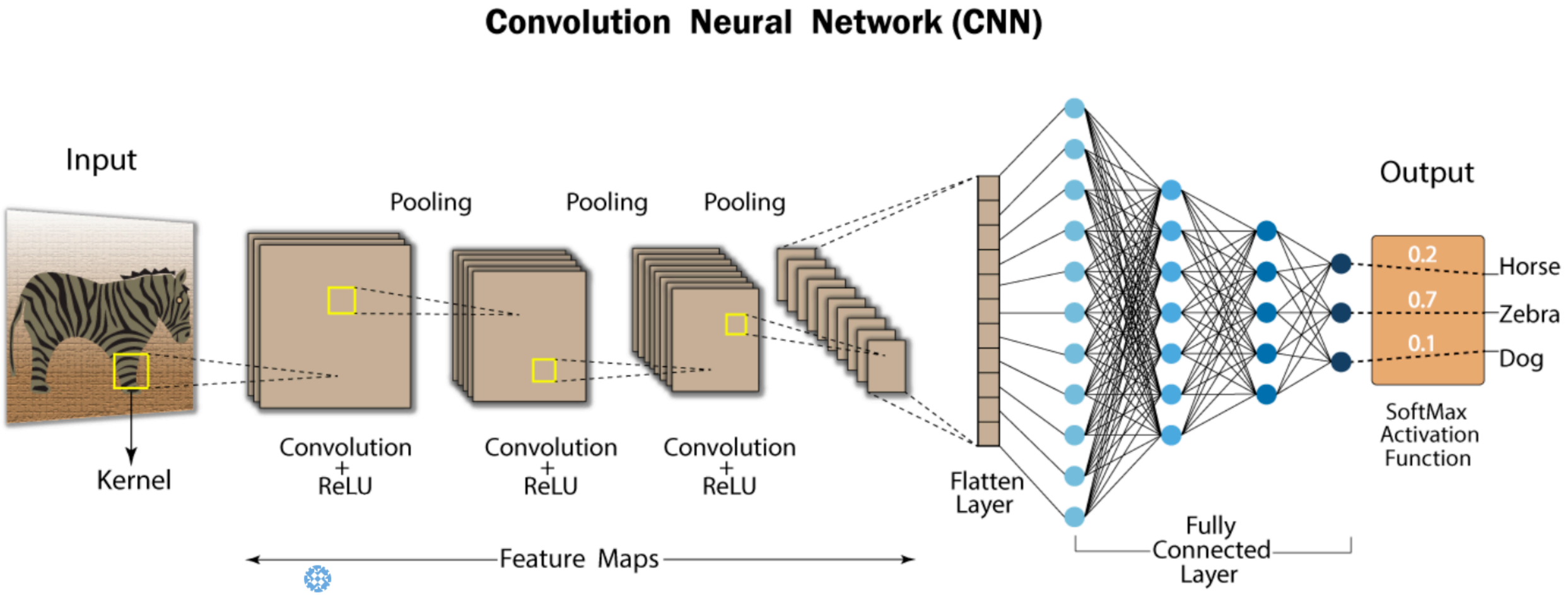
Interactive Website

https://wordsandbuttons.online/interactive_guide_to_homogeneous_coordinates.html



Convolution Operation

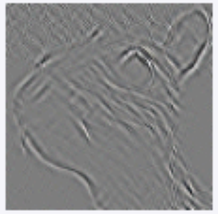
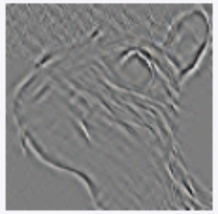
The convolution operation is an essential way to extract features from an image



https://developersbreach.com/convolution-neural-network-deep-learning/

Convolution Operation

The convolution operation is an essential way to extract features from an image

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
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 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Convolution Operation

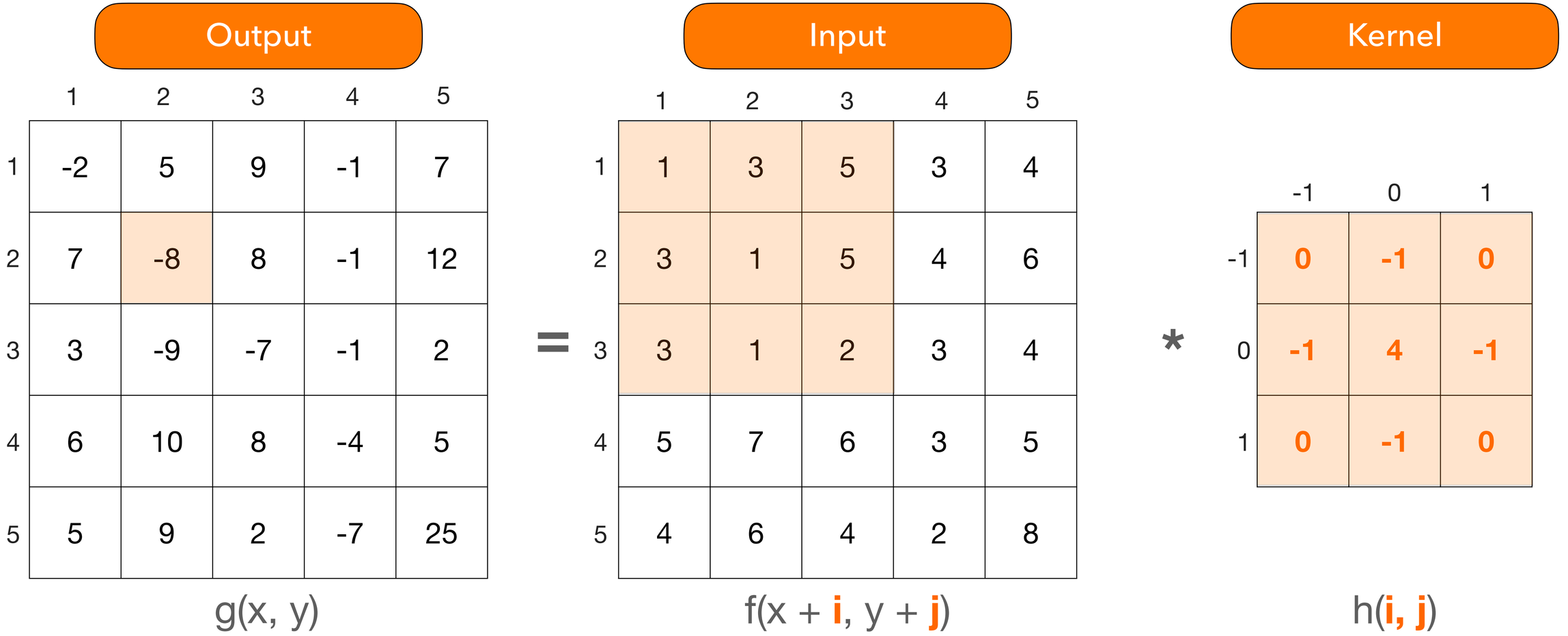
Formulation

Image filtering is a common operation in image processing. It is used to enhance the image by removing noise, or to extract features from the image. The mathematical formulation of image filtering is called convolution. We can use the following equation to describe the convolution operation:

$$g(x,y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(x+i, y+j)h(i,j)$$

Output | Input | Kernel

Convolution Operation



	1	2	3	4	5
1	-2	5	9	-1	7
2	7	-8	8	-1	12
3	3	-9	-7	-1	2
4	6	10	8	-4	5
5	5	9	2	-7	25

=

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

*

	-1	0	1
-1	0	-1	0
0	-1	4	-1
1	0	-1	0

g(2, 2) =

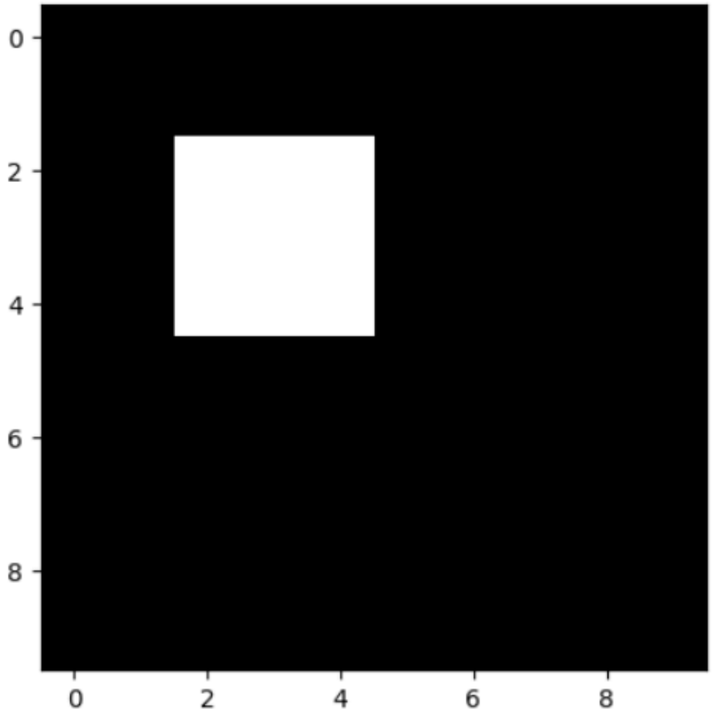
f(2 + (**-1**), 2 + (**-1**)) * h(**-1**, **-1**) + f(2 + (**-1**), 2 + (**0**)) * h(**-1**, **0**) + f(2 + (**-1**), 2 + (**1**)) * h(**-1**, **1**) +
f(2 + (**0**), 2 + (**-1**)) * h(**0**, **-1**) + f(2 + (**0**), 2 + (**0**)) * h(**0**, **0**) + f(2 + (**0**), 2 + (**1**)) * h(**0**, **1**) +
f(2 + (**1**), 2 + (**-1**)) * h(**1**, **-1**) + f(2 + (**1**), 2 + (**0**)) * h(**1**, **0**) + f(2 + (**1**), 2 + (**1**)) * h(**1**, **1**)

Convolution Operation - Input Matrix

Input matrix

Let's generate an example 2D matrix to demonstrate the convolution operation.

$$f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$




```python
import numpy as np

# generate a 10×10 matrix filled with zeros
input = np.zeros((10, 10))
# set part of the matrix to 1
input[2:5, 2:5] = 1
# visualization
plt.imshow(input, cmap="gray")
```

Convolution Operation - Kernel

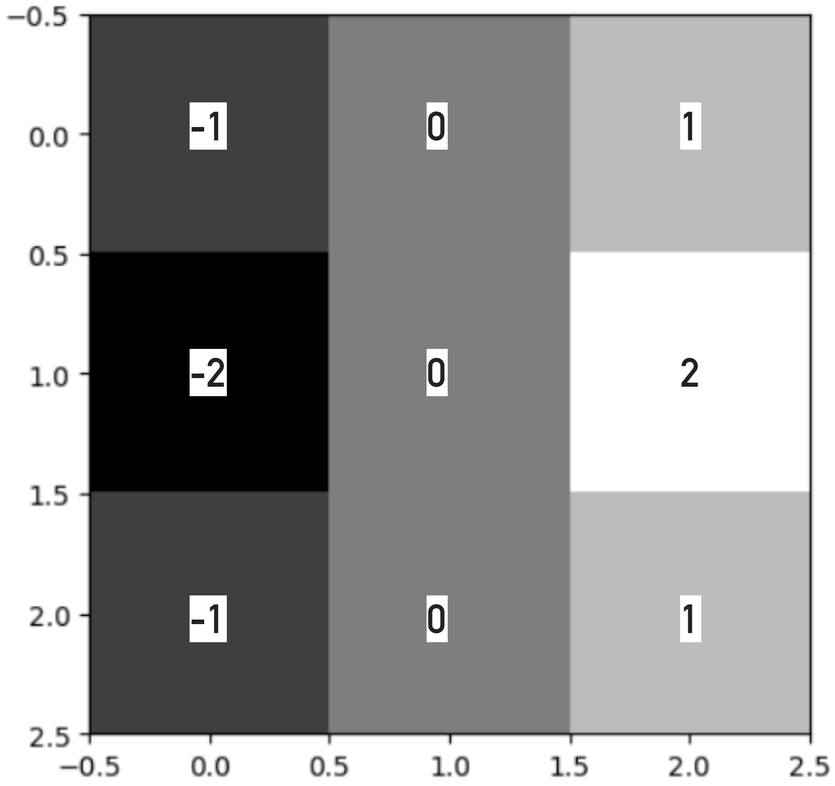
Kernel

We can define a sobel filtering kernl as below:

$$h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

```
kernel_x = np.array([[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]])
plt.imshow(kernel_x, cmap="gray")
```

✓ 0.0s



Convolution Operation - Output

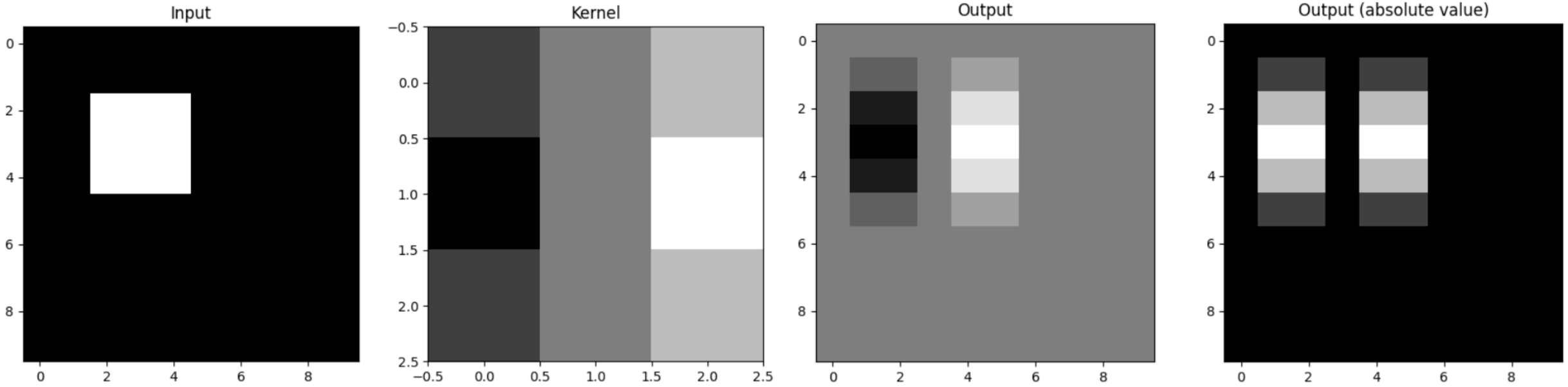
Convolution

With the defined input matrix and the kernel, we can calculate the convolution operation as we defined above. We can use `scipy.signal.convolve2d` to calculate the convolution operation.

```
from scipy.signal import convolve2d

# convolution
output = convolve2d(input, kernel_x, mode="same")
```

The **output** shows an x-direction gradient of the **input** matrix. We want to take the **absolute value** of the **output** to get the magnitude of the gradient.



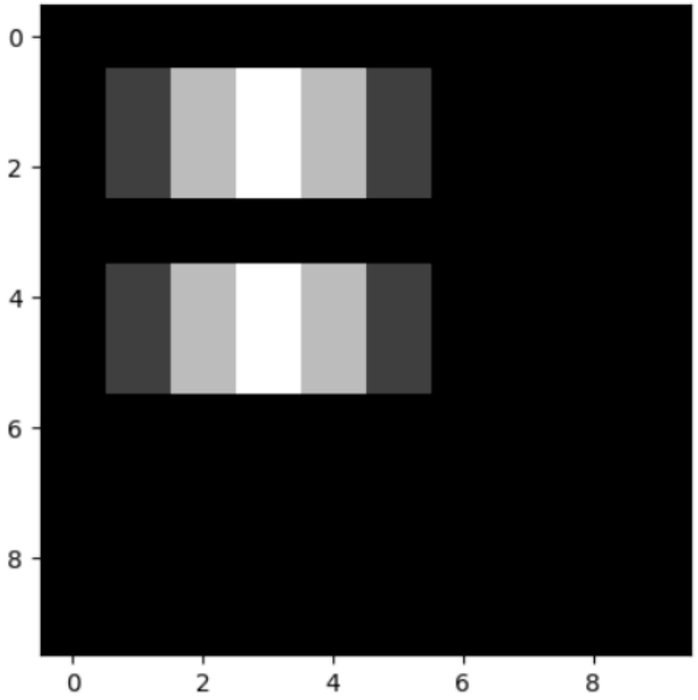
Convolution Operation - Anther Kernel

Y-direction

```
kernel_y = np.array([[-1, -2, -1],
                     [0, 0, 0],
                     [1, 2, 1]])
# convolution
output = convolve2d(input, kernel_y, mode="same")
plt.imshow(abs(output), cmap="gray")
```

✓ 0.1s

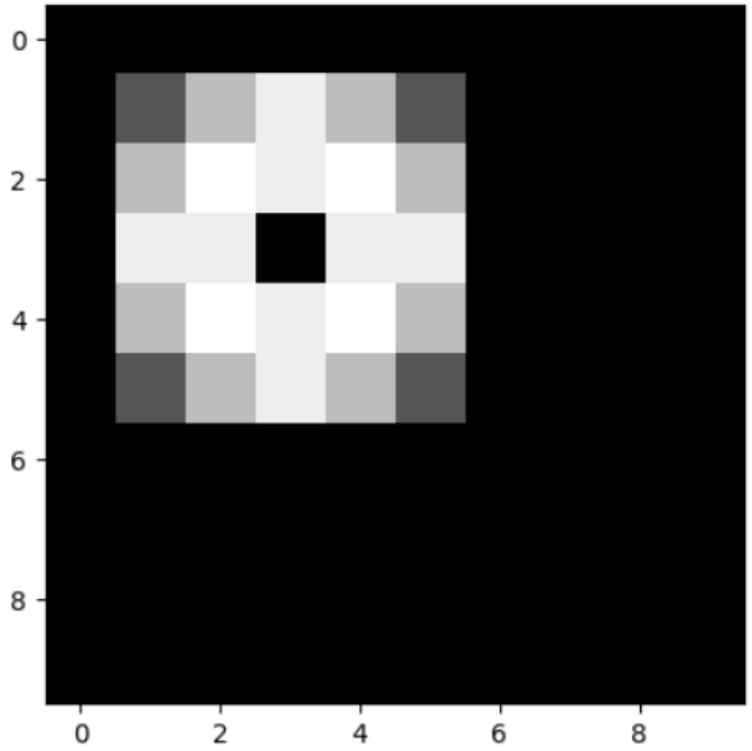
```
<matplotlib.image.AxesImage at 0×2ed5c7430>
```



Both X- and Y-direction

Real + imaginary numbers

```
kernel = np.array([ [-1-1j, 0-2j, 1-1j],
                    [-2+0j, 0+0j, 2+0j],
                    [-1+1j, 0+2j, 1+1j]])
output = convolve2d(input, kernel, mode="same")
```



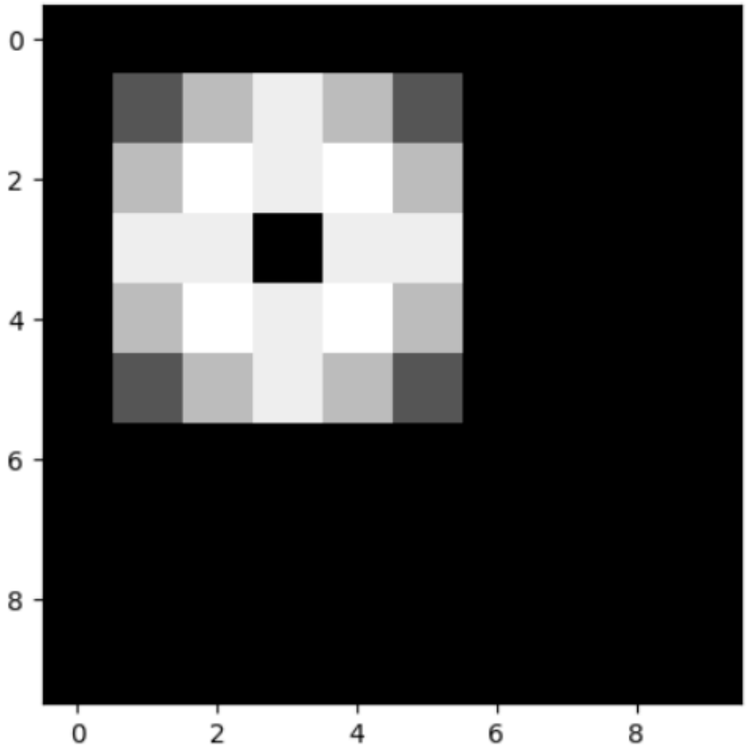
Binarization

Once we obtain the filtered image by an convolution operation, we can set a threshold to binarize the image.

```
img_conv2d = abs(output)
print("quantile 0.5: ", np.quantile(img_conv2d[img_conv2d > 0], 0.5))
print("quantile 0.9: ", np.quantile(img_conv2d[img_conv2d > 0], 0.9))
```

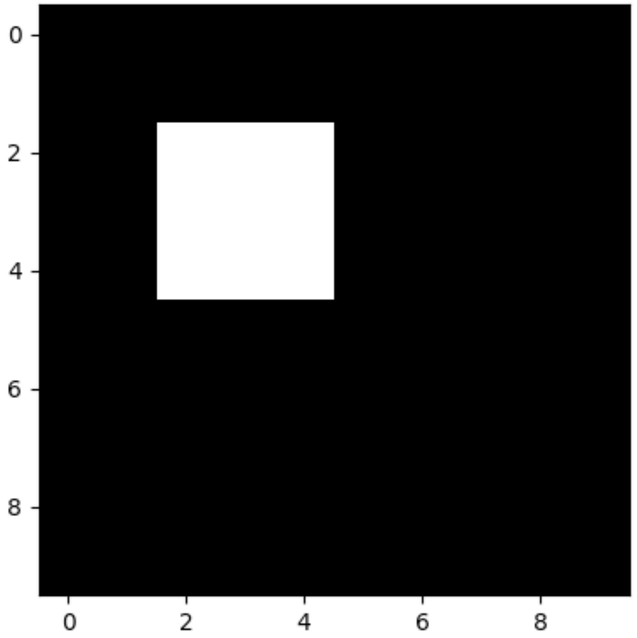
✓ 0.0s MagicPython

```
quantile 0.5:  3.581138830084 19
quantile 0.9:  4.242640687119285
```

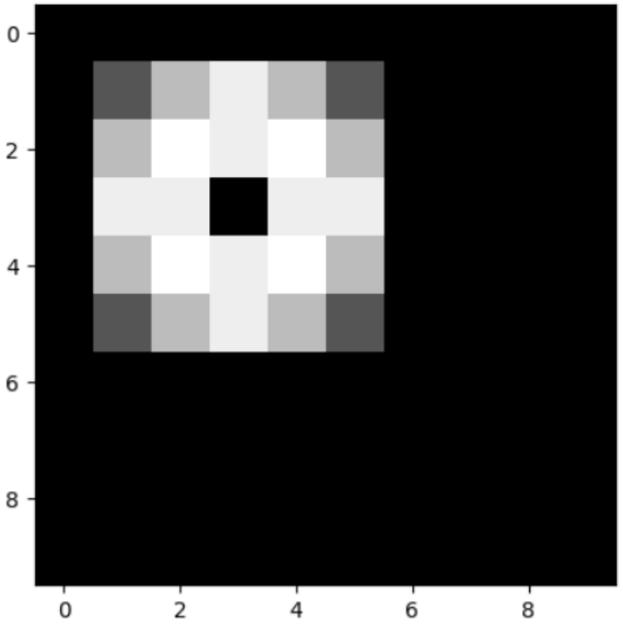


Binarization

Input

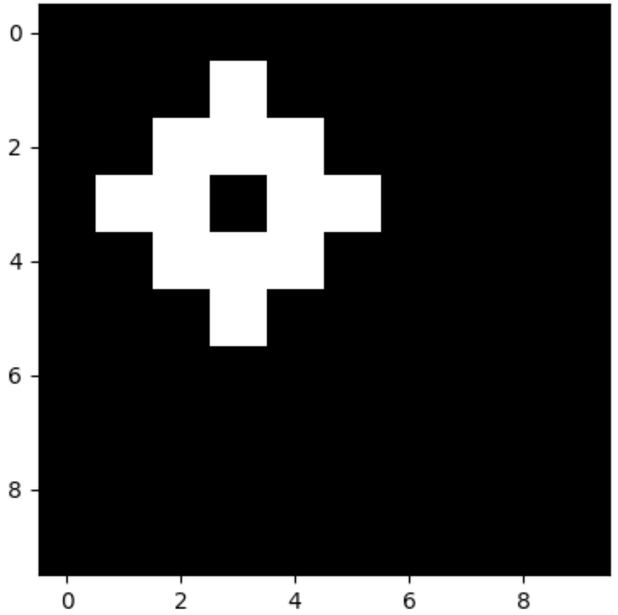


Output

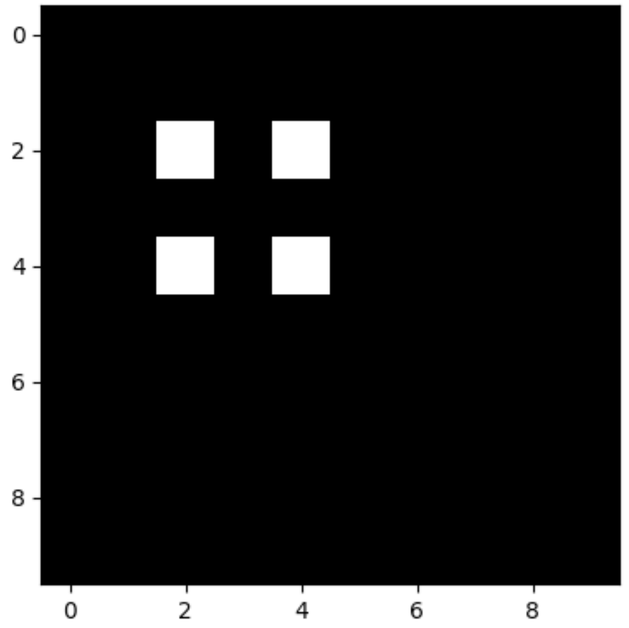


Binarization

The 50th percentile



The 90th percentile



Convolution Operation - Real Image

Real image

Input

```
with PIL.Image.open("cow.jpg") as f:
    image = np.array(f)[:,:,0] # only use the first channel
plt.imshow(image, cmap="gray")
```

✓ 0.3s

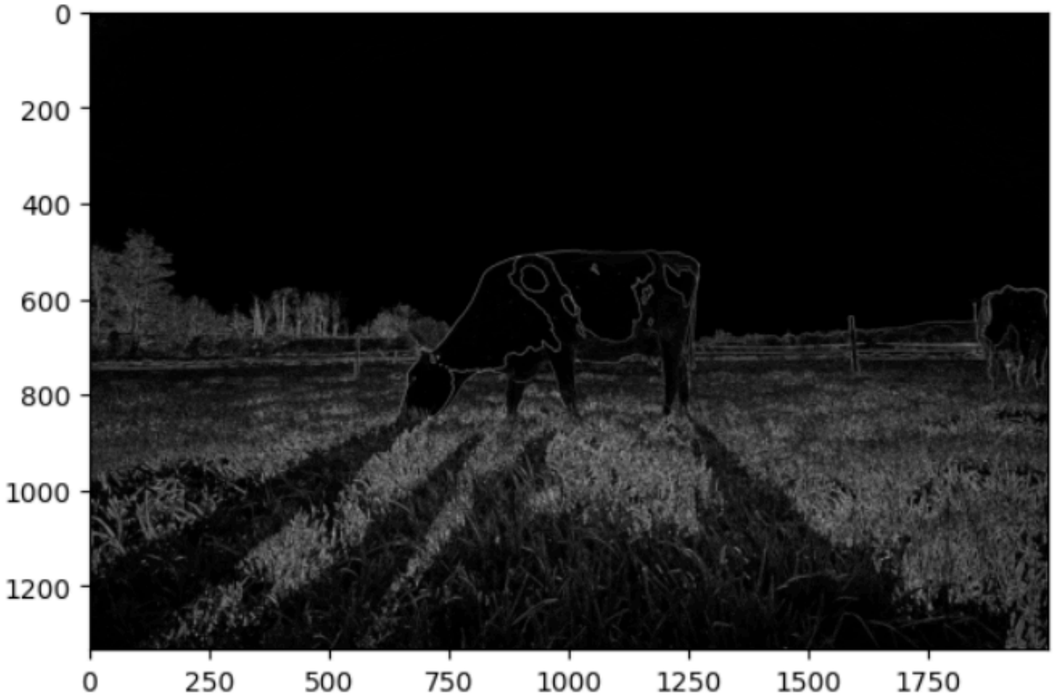
<matplotlib.image.AxesImage at 0x13fc8ae80>

Output

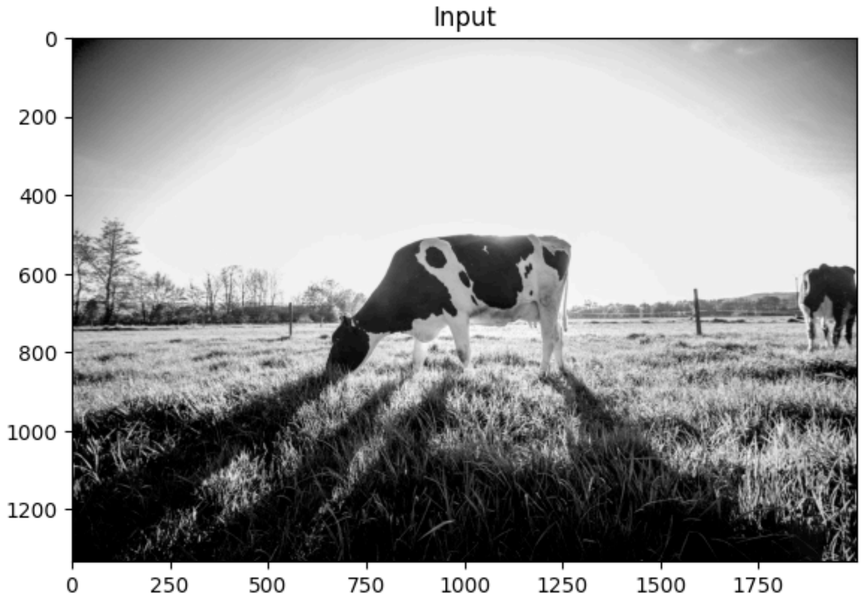
```
img_conv = convolve2d(image, kernel, mode="same")
img_conv = abs(img_conv)
plt.imshow(img_conv, cmap="gray")
```

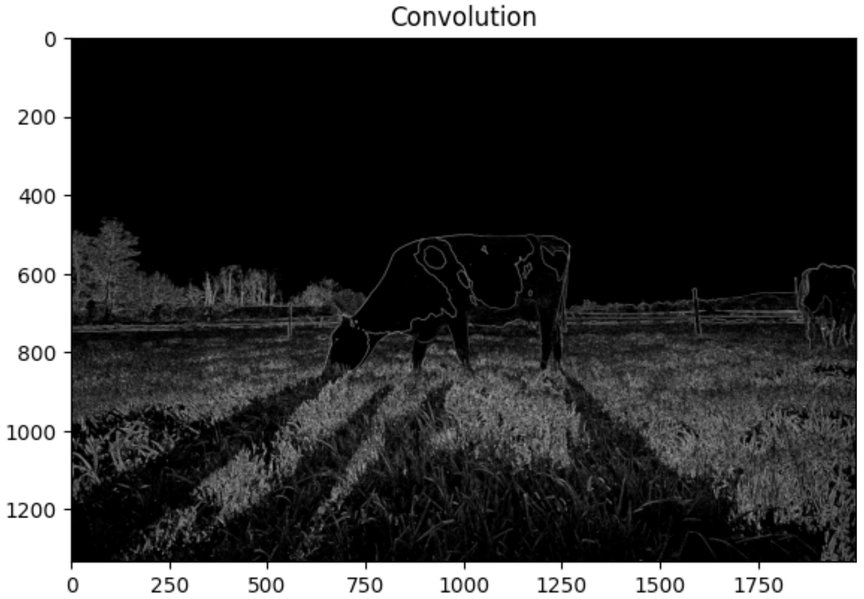
✓ 0.3s

<matplotlib.image.AxesImage at 0x13fc101f0>

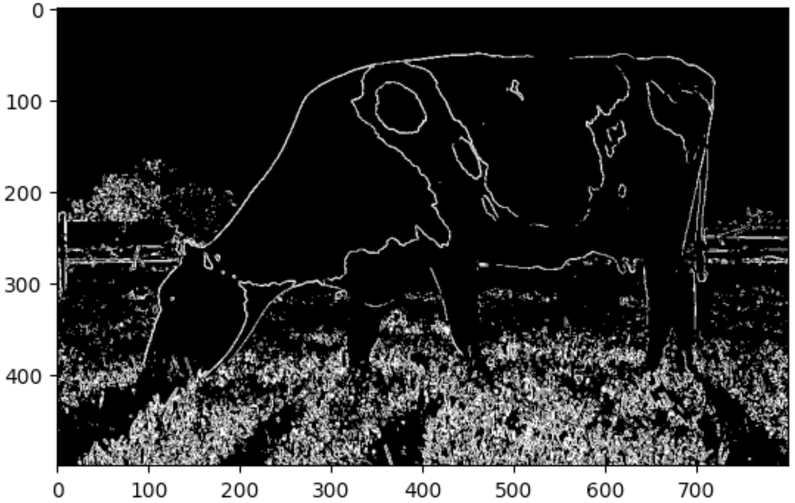


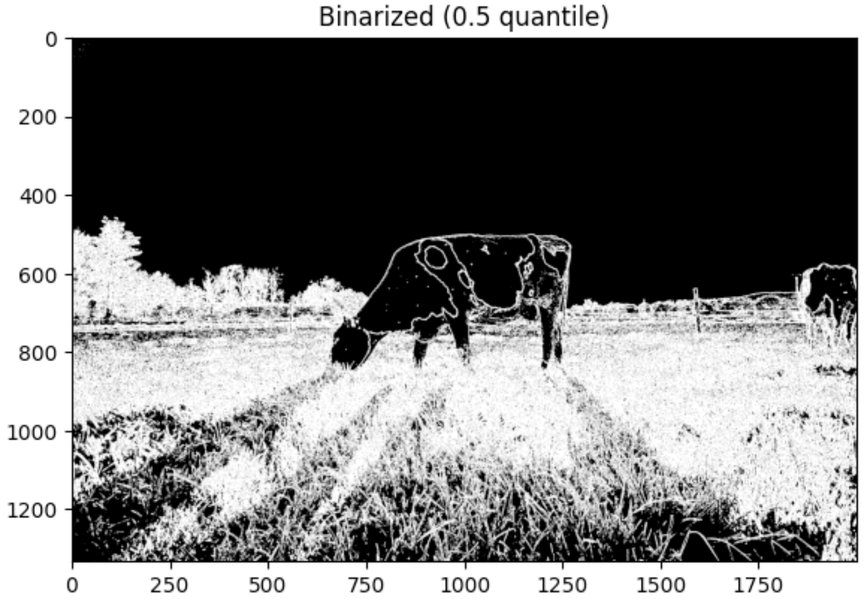
Convolution Operation - Real Image

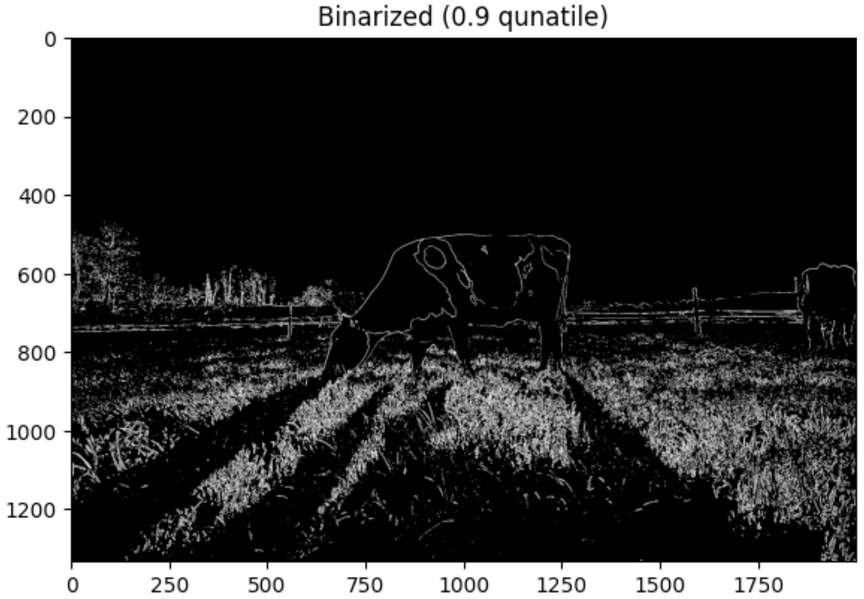




0.9 quantile
(Edge detection)

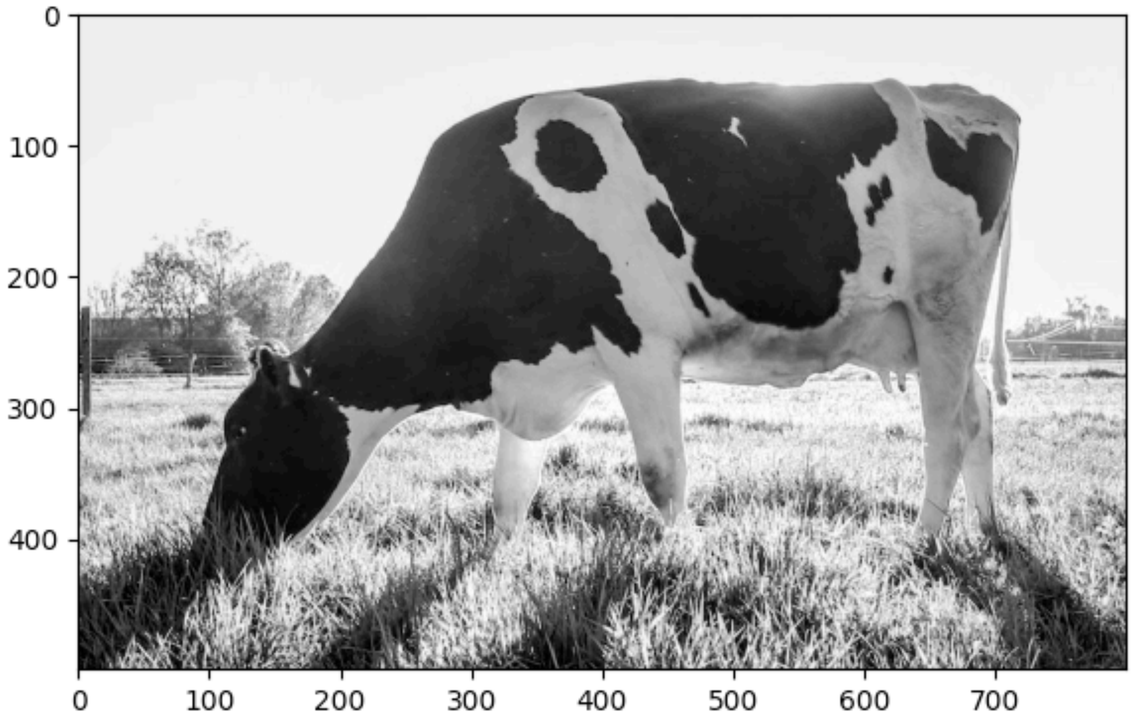


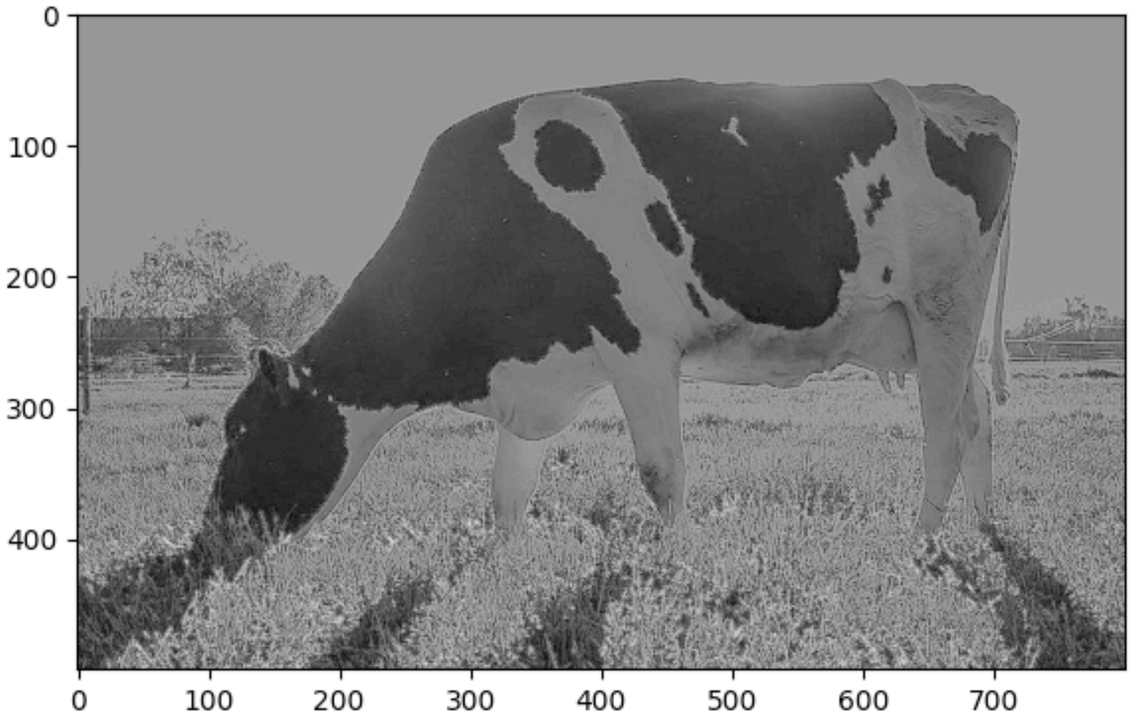




Other Kernels - Sharpen


```
k_shp = np.array((
    [0, -1, 0],
    [-1, 5, -1],
    [0, -1, 0]),
    dtype='int')
imgc = convolve2d(image, k_shp, mode="same")
imgc = abs(imgc)
```





Other Kernels - Gaussian Blur


```
k_gauss = np.array((
    [1, 4, 1],
    [4, 9, 4],
    [1, 4, 1]),
    dtype='int') / 29

imgc = convolve2d(image, k_gauss, mode="same")
for _ in range(50):
    imgc = convolve2d(imgc, k_gauss, mode="same")
```