Basic Color Spaces (OpenCV)

- **RGB (Red, Green, Blue)**: is an additive color space where colors are created by combining red, green, and blue light intensities. **OpenCV uses BGR** (Blue, Green, Red) instead of RGB as its default channel order.
  cv2.COLOR_BGR2RGB to convert from BGR to RGB.

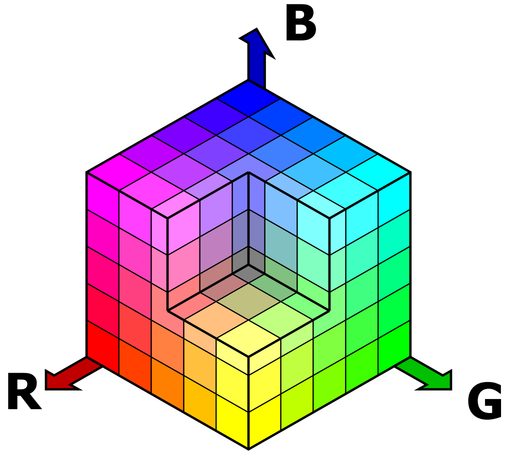
- **Grayscale**: A single-channel representation of an image where each pixel represents intensity (brightness) on a scale from black (0) to white (255). Used when color information is not required.
  cv2.COLOR_BGR2GRAY to convert from BGR to grayscale.

- **HSV (Hue, Saturation, Value)**: HSV separates color information (hue) from intensity (value) and saturation. Hue refers to the type of color (e.g., red or blue), saturation measures the purity of the color, and value represents brightness. Use Case: Ideal for tasks like color-based segmentation because it simplifies the representation of colors.
  cv2.COLOR_BGR2HSV to convert from BGR to HSV.

https://en.wikipedia.org/wiki/Color_model#/media/File:RGBCube_a.svg
https://en.wikipedia.org/wiki/RGB_color_model

Color Space RGB

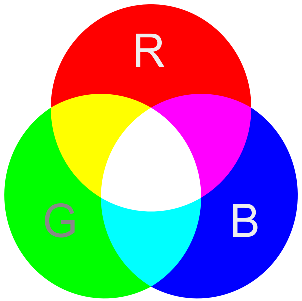
The RGB model works by adding light. When the intensities of red, green, and blue are all at their maximum, the result is white light. When all are at their minimum (zero intensity), the result is black.



Each color channel (R, G, B) typically has a range of values, often from 0 to 255 For example:

(255,0,0)(255,0,0): Pure red
(0,255,0)(0,255,0): Pure green
(0,0,255)(0,0,255): Pure blue
(255,255,0)(255,255,0): Yellow (red + green)

The RGB color space is widely used in electronic displays like computer screens, TVs, and cameras because these devices emit light.

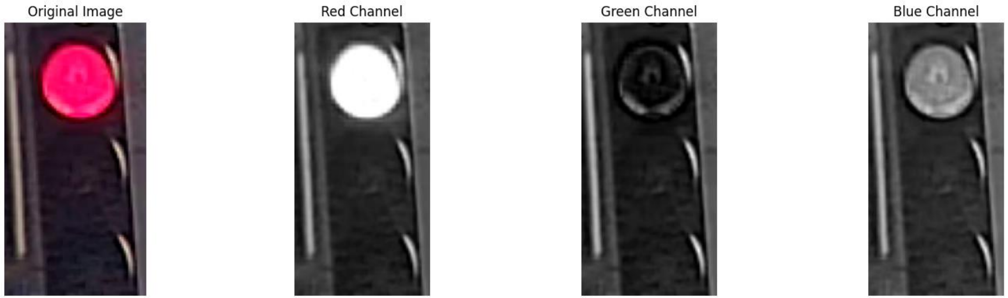


https://en.wikipedia.org/wiki/Color_model#/media/File:RGBCube_a.svg
https://en.wikipedia.org/wiki/RGB_color_model

Color Space RGB

Since colors in the RGB color space are coded using the three channels (in the range *[0, 255]*), it is more difficult to segment an object in the image based on its color.

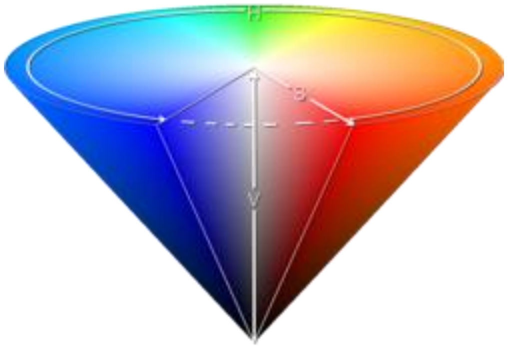
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
image = cv2.imread("red-tlight.png")

# Resize the image
image = cv2.resize(image, (100, 200))

# Obtain shape of the image
h, w, c = image.shape

# Split BRG channels
b, g, r = cv2.split(image)
```

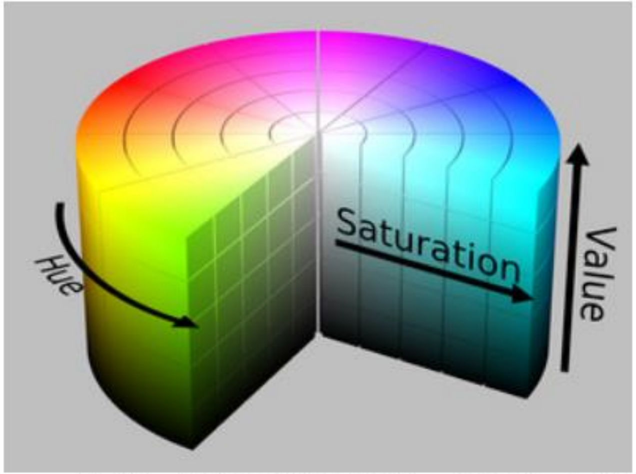


Color Space HSV

HSV colorspace

HSV (hue, saturation, value) colorspace is a model to represent the colorspace similar to the RGB color model. Since the hue channel models the color type, it is very useful in image processing tasks that need to segment objects based on its color. Variation of the saturation goes from unsaturated to represent shades of gray and fully saturated (no white component). Value channel describes the brightness or the intensity of the color. Next image shows the HSV cylinder.





By SharkDderivative work: SharkD [CC BY-SA 3.0 or GFDL], via Wikimedia Commons

https://docs.opencv.org/4.11.0/df/d9d/tutorial_py_colorspaces.html
https://docs.opencv.org/4.11.0/da/d97/tutorial_threshold_inRange.html
https://colorizer.org/

Color Space HSV

How to find HSV values to track?

This is a common question found in stackoverflow.com. It is very simple and you can use the same function, **cv.cvtColor()**. Instead of passing an image, you just pass the BGR values you want. For example, to find the HSV value of Green, try the following commands in a Python terminal:

```
>>> green = np.uint8([[[0,255,0 ]]])
>>> hsv_green = cv.cvtColor(green,cv.COLOR_BGR2HSV)
>>> print( hsv_green )
[[[ 60 255 255]]]
```

Now you take [H-10, 100,100] and [H+10, 255, 255] as the lower bound and upper bound respectively. Apart from this method, you can use any image editing tools like GIMP or any online converters to find these values, but don't forget to adjust the HSV ranges.

Note

For HSV, hue range is [0,179], saturation range is [0,255], and value range is [0,255]. Different software use different scales. So if you are comparing OpenCV values with them, you need to normalize these ranges.

https://docs.opencv.org/4.11.0/df/d9d/tutorial_py_colorspaces.html
https://docs.opencv.org/4.11.0/da/d97/tutorial_threshold_inRange.html
https://colorizer.org/

Thresholding?

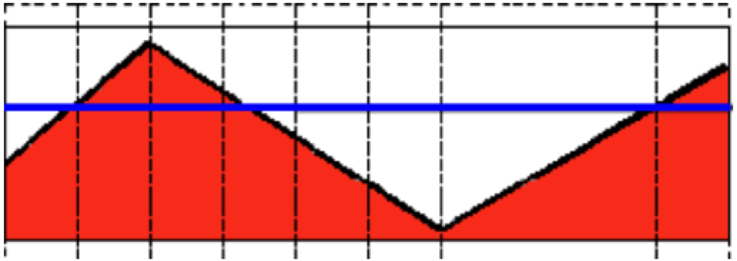
- The simplest segmentation method
- Application example: Separate out regions of an image corresponding to objects which we want to analyze. This separation is based on the variation of intensity between the object pixels and the background pixels.
- To differentiate the pixels we are interested in from the rest (which will eventually be rejected), we perform a comparison of each pixel intensity value with respect to a *threshold* (determined according to the problem to solve).
- Once we have separated properly the important pixels, we can set them with a determined value to identify them (i.e. we can assign them a value of 0 (black), 255 (white) or any value that suits your needs).

https://docs.opencv.org/4.11.0/db/d8e/tutorial_threshold.html

Thresholding

Types of Thresholding

- OpenCV offers the function **cv::threshold** to perform thresholding operations.
- We can effectuate 5 types of Thresholding operations with this function. We will explain them in the following subsections.
- To illustrate how these thresholding processes work, let's consider that we have a source image with pixels with intensity values $src(x, y)$. The plot below depicts this. The horizontal blue line represents the threshold $thresh$ (fixed).

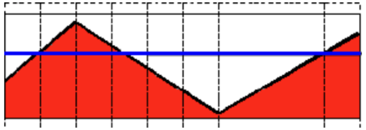


src

https://docs.opencv.org/4.11.0/db/d8e/tutorial_threshold.html

Thresholding

Types of Thresholding



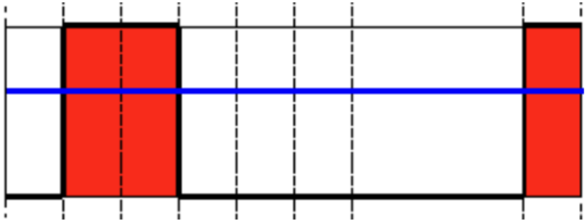
src

Threshold Binary

- This thresholding operation can be expressed as:

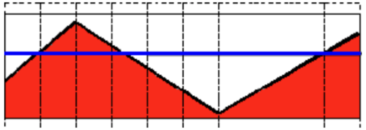
$$\texttt{dst}(x, y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x, y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- So, if the intensity of the pixel $src(x, y)$ is higher than $thresh$, then the new pixel intensity is set to a $MaxVal$. Otherwise, the pixels are set to 0.



https://docs.opencv.org/4.11.0/db/d8e/tutorial_threshold.html

Types of Thresholding



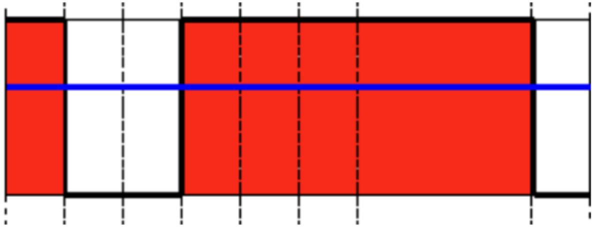
src

Threshold Binary, Inverted

- This thresholding operation can be expressed as:

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

- If the intensity of the pixel $src(x,y)$ is higher than $thresh$, then the new pixel intensity is set to a 0. Otherwise, it is set to $MaxVal$.



https://docs.opencv.org/4.11.0/db/d8e/tutorial_threshold.html

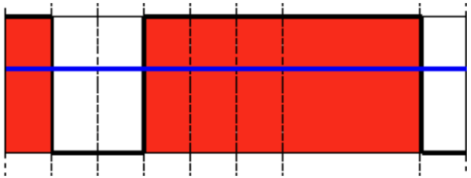
Thresholding + Morph.

Threshold Binary, Inverted

- This thresholding operation can be expressed as:

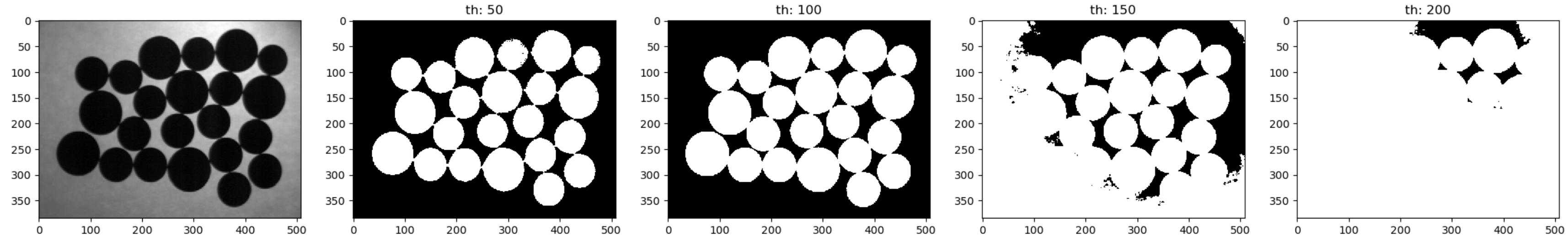
$$\texttt{dst}(x, y) = \begin{cases} 0 & \text{if } \texttt{src}(x, y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

- If the intensity of the pixel $src(x, y)$ is higher than $thresh$, then the new pixel intensity is set to a 0. Otherwise, it is set to $MaxVal$.



th_lst = [50, 100, 150, 200]

cv.threshold(img_input, th_lst[i], 255, cv.THRESH_BINARY_INV)

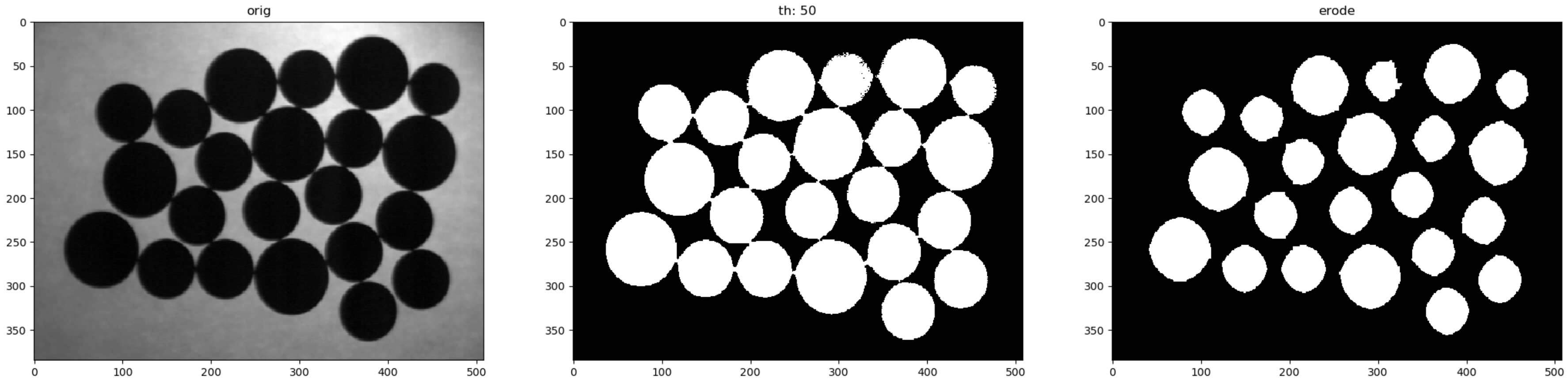


https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html https://docs.opencv.org/4.11.0/db/d8e/tutorial_threshold.html https://docs.opencv.org/4.11.0/d3/db4/tutorial_py_watershed.html

Thresholding + Morph.

```
img_input = cv.imread('mon1.png', 0)
kernel = cv.getStructuringElement(cv.MORPH_RECT, (11, 11))
ret, img_th = cv.threshold(img_input, 50, 255, cv.THRESH_BINARY_INV)
img_erode = cv.erode(img_th, kernel, iterations = 1)
```

Morphological Operation



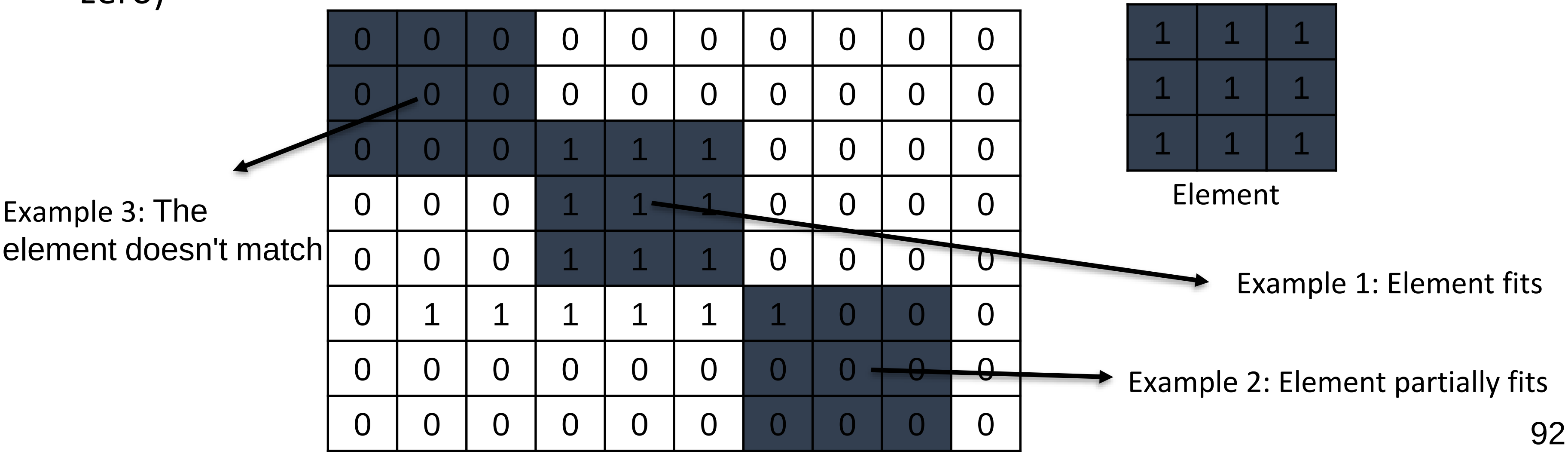
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html https://docs.opencv.org/4.11.0/db/d8e/tutorial_threshold.html https://docs.opencv.org/4.11.0/d3/db4/tutorial_py_watershed.html

Morphological Operation

- Simple operations that are very often used on binary images
- To perform morphological operations, we need input image and kernel (structuring element)
- The morphological operations (most common) that we will discuss deeper are:
  - Erosion
  - Dilation
  - Opening
  - Closing
  - Gradient

Morphological Operation

- The main principle is that the kernel is moved through to the input image
- The corresponding pixels inside the kernel vs. input image are examined
- Based on the operation (erosion, dilation), the pixels are for example eroded (made to zero)



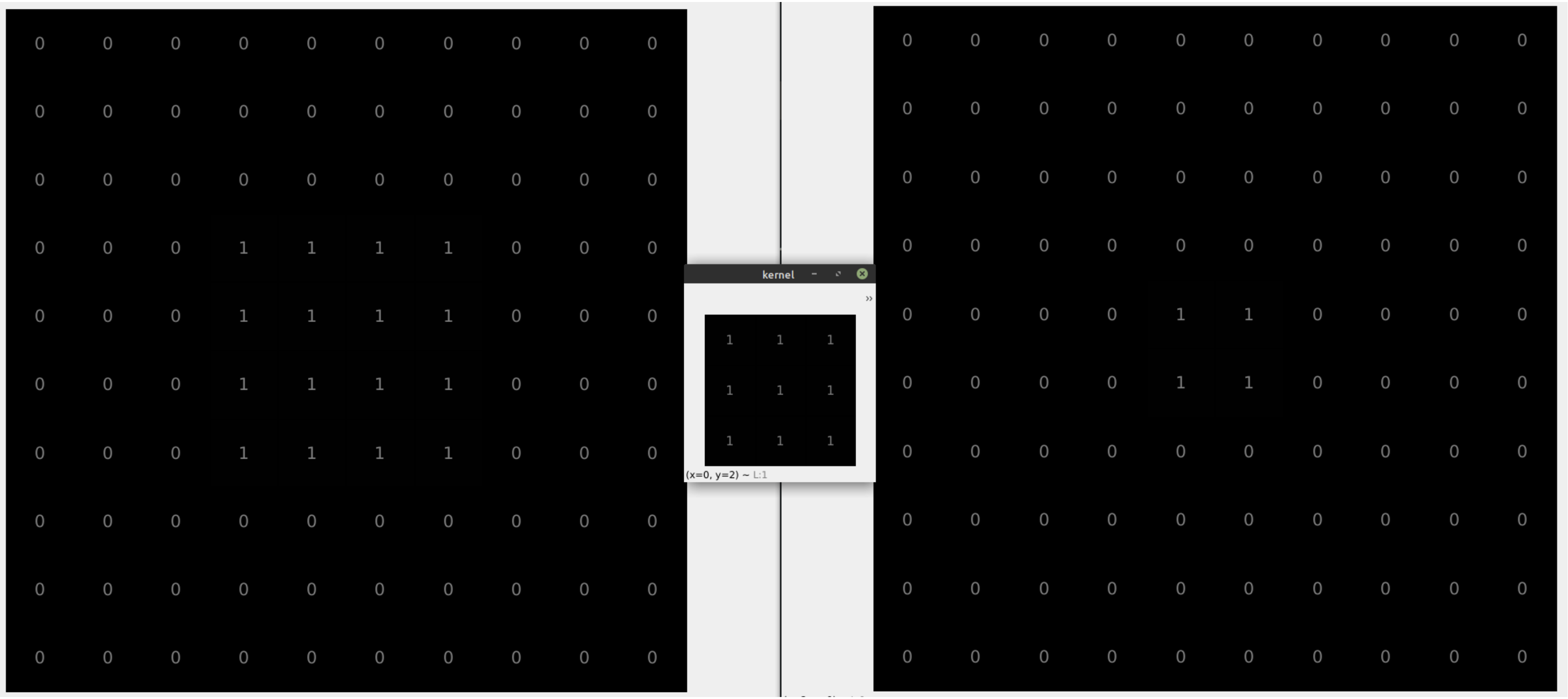
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

1	1	1
1	1	1
1	1	1

Element

Erosion

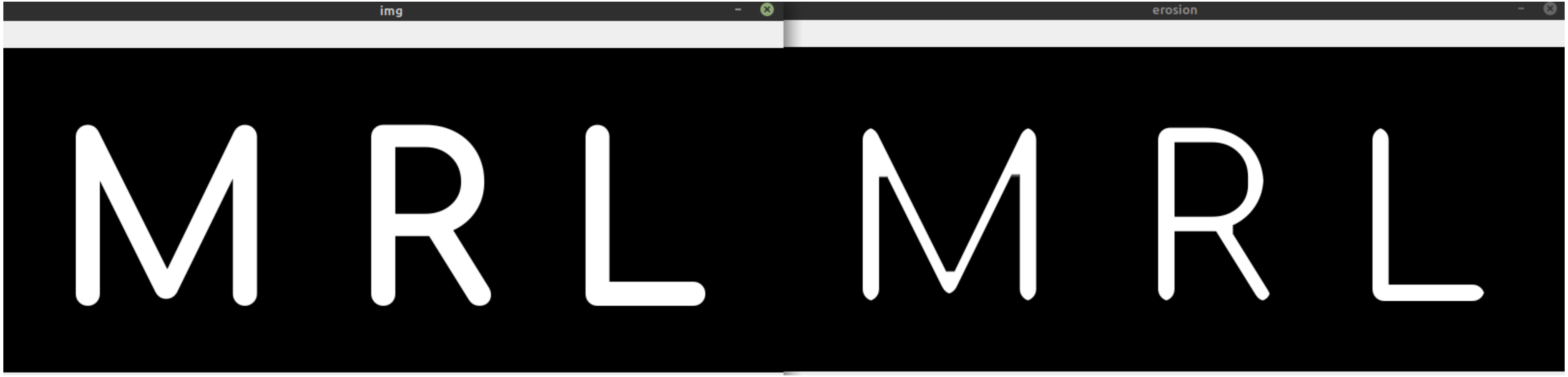
A pixel in the original image will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). AND operator.



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Erosion

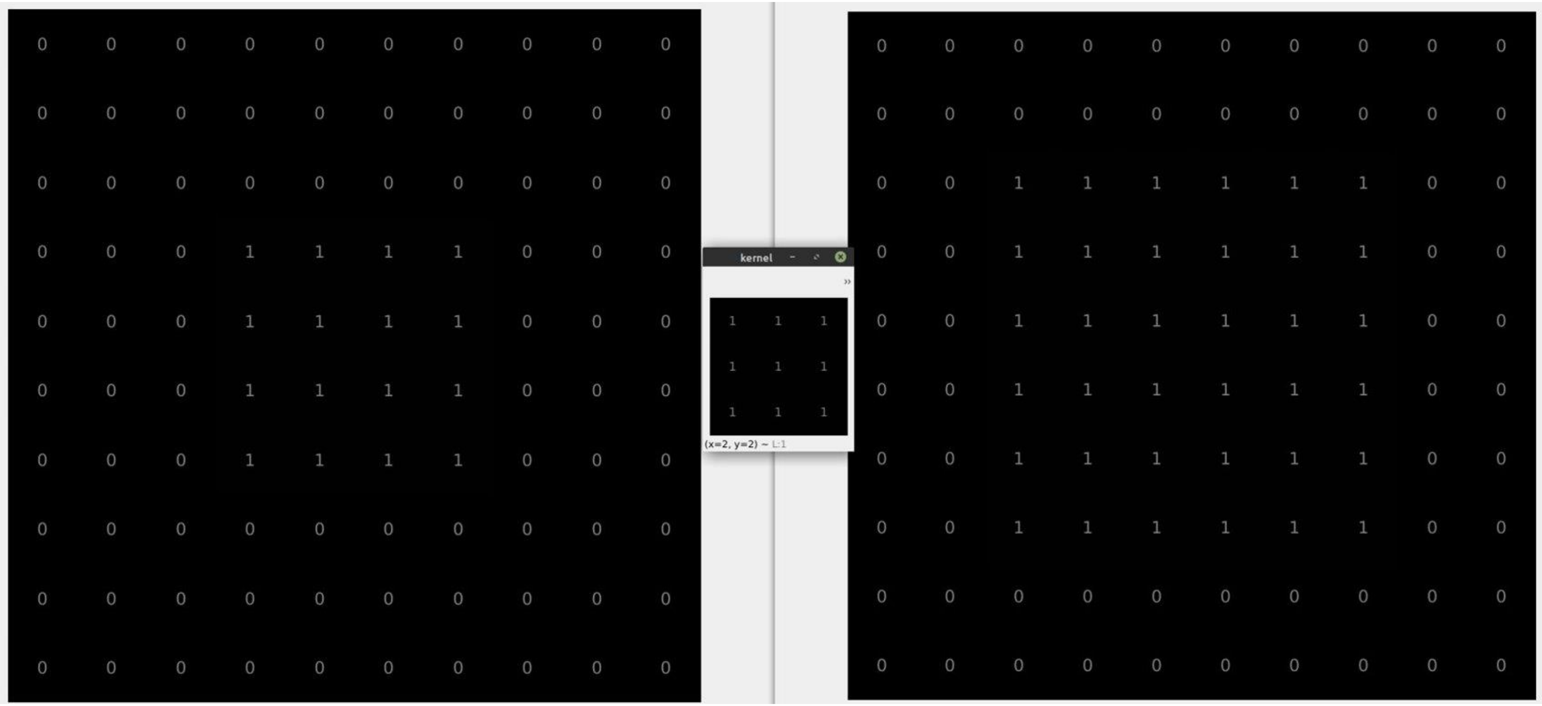
A pixel in the original image will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Dilation

Opposite of erosion: a pixel element is '1' if at least one pixel under the kernel is '1'.



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

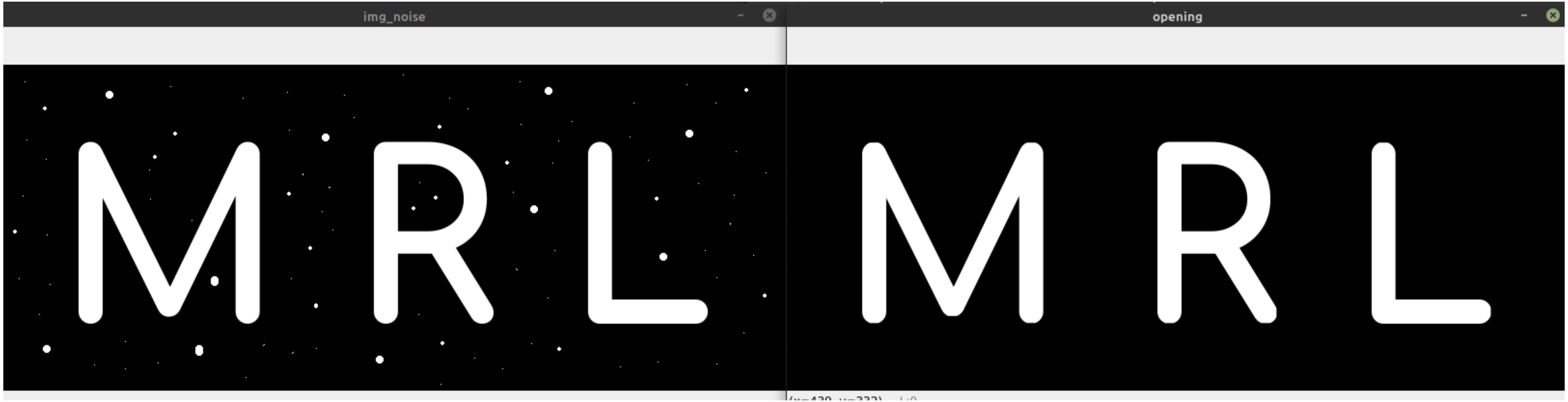
Dilation

Opposite of erosion: a pixel element is '1' if at least one pixel under the kernel is '1'.

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Opening

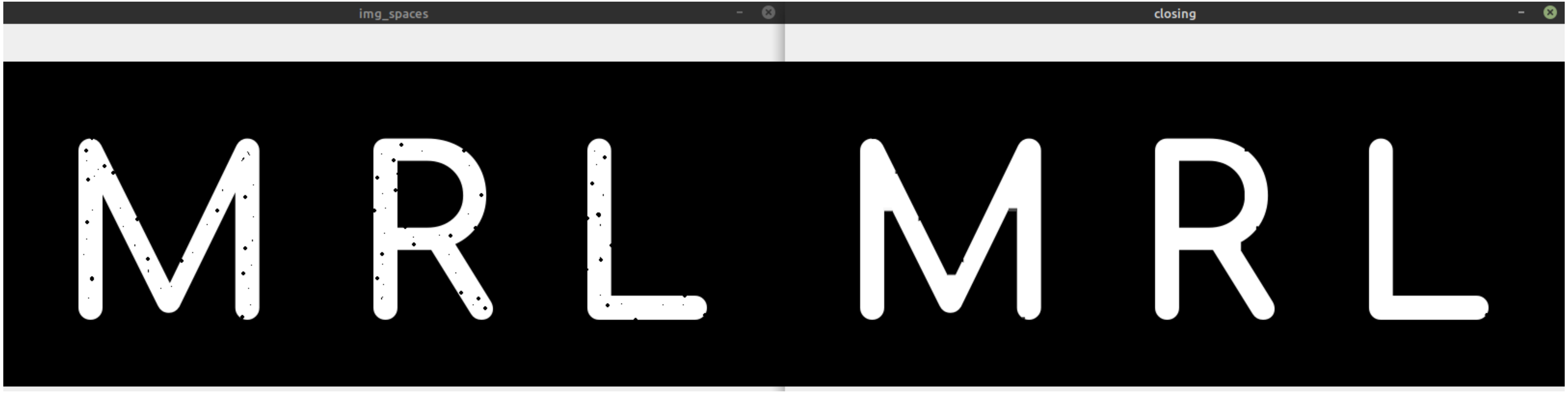
Opening: **erosion** followed by **dilation**



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Closing

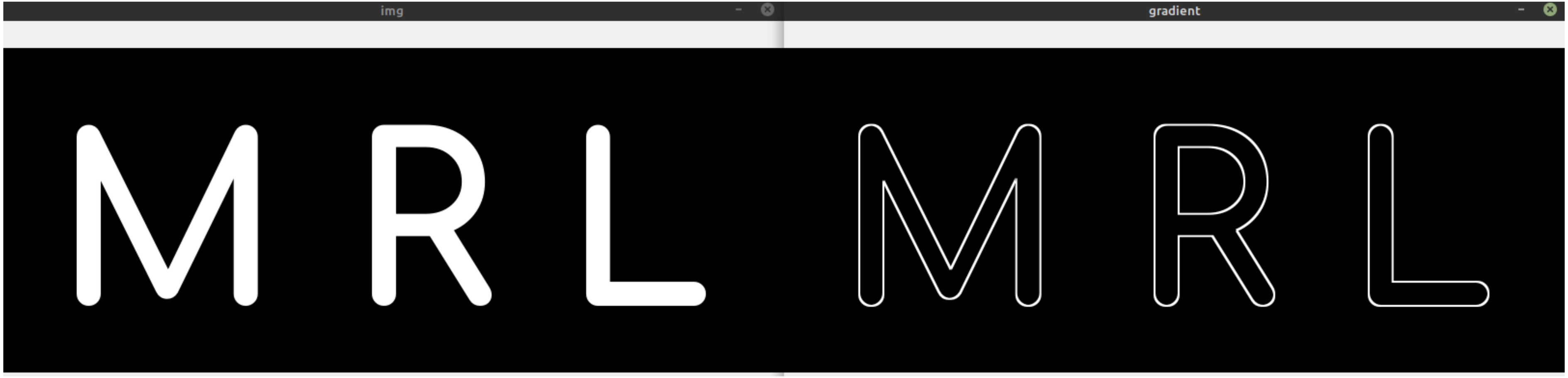
Opening: **dilation** followed by **erosion**



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Morphological Gradient

Difference between **dilation** and **erosion** of an image



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Structuring Elements

kernel = np.ones((5,5), np.uint8)

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

kernel = cv.getStructuringElement(cv.MORPH_CROSS, (5,5))

```
[[0 0 1 0 0]
 [0 0 1 0 0]
 [1 1 1 1 1]
 [0 0 1 0 0]
 [0 0 1 0 0]]
```

kernel = cv.getStructuringElement(cv.MORPH_RECT, (5,5))

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Structuring Elements

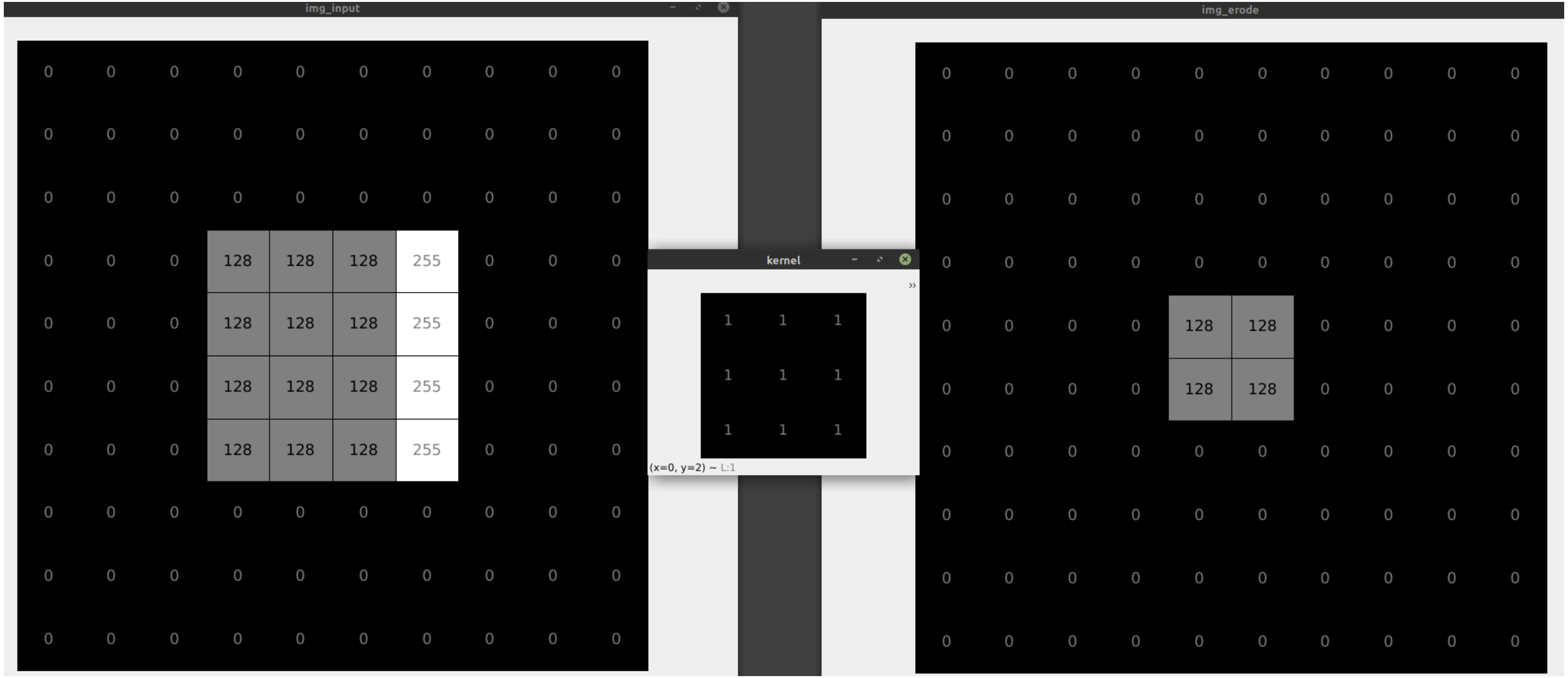
kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (15,15))

```
[[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 1 1 1 1 1 1 1 1 1 1 1 0 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
 [0 0 1 1 1 1 1 1 1 1 1 1 1 0 0]
 [0 0 0 1 1 1 1 1 1 1 1 1 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]]
```


https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Grayscale/Color

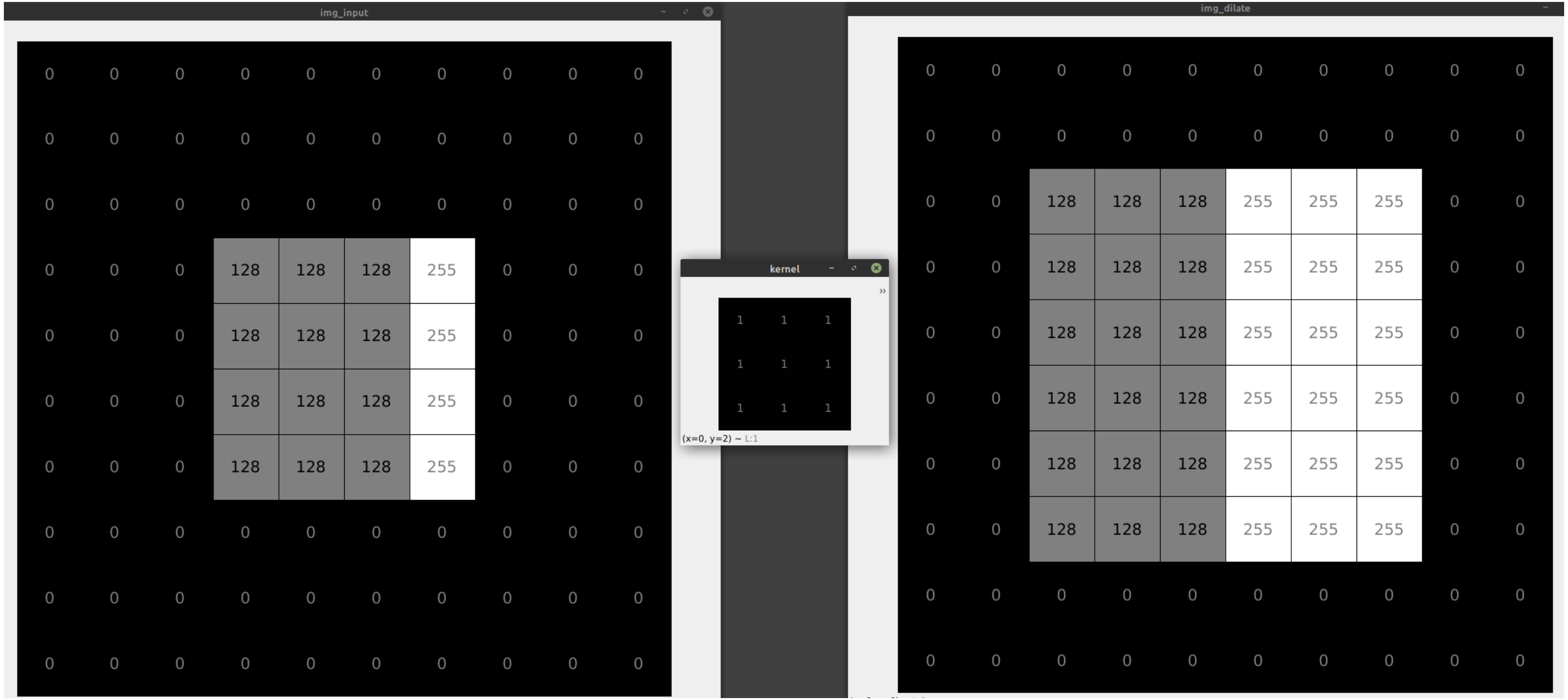
Idea of binary morphology can be extended to gray/color images with the use of max (Dilation) and min (Erosion) operation



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Grayscale/Color

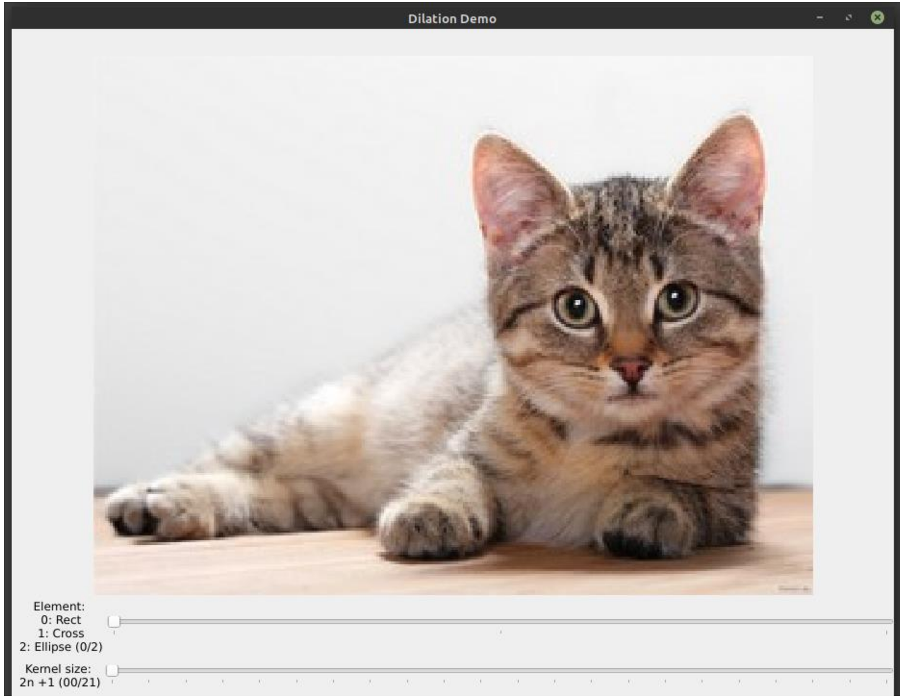
Idea of binary morphology can be extended to gray/color images with the use of max (Dilation) and min (Erosion) operation

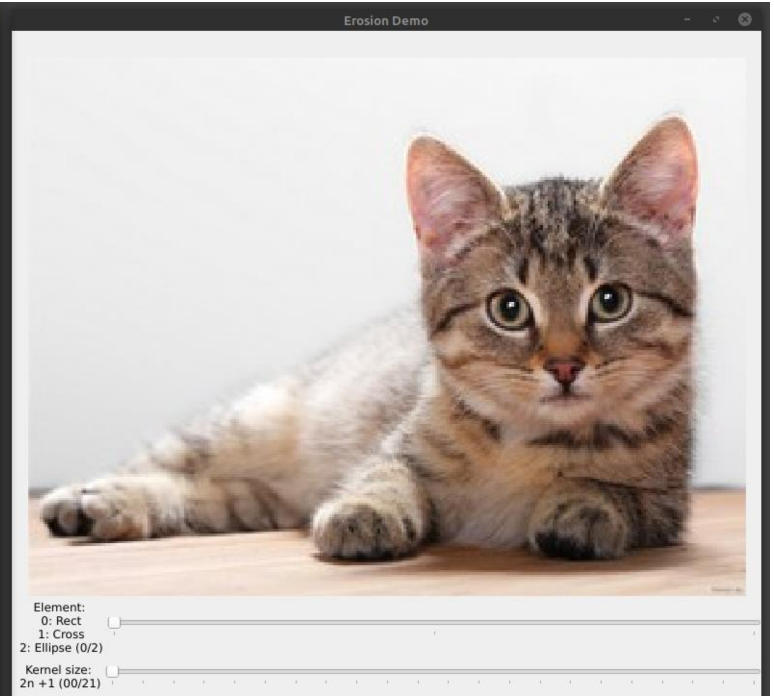


https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Grayscale/Color

Idea of binary morphology can be extended to gray/color images with the use of max (dilation) and min (erosion) operation. Dilation and erosion change the brightness.
dilation - increased overall brightnes vs. erosion - dims the image

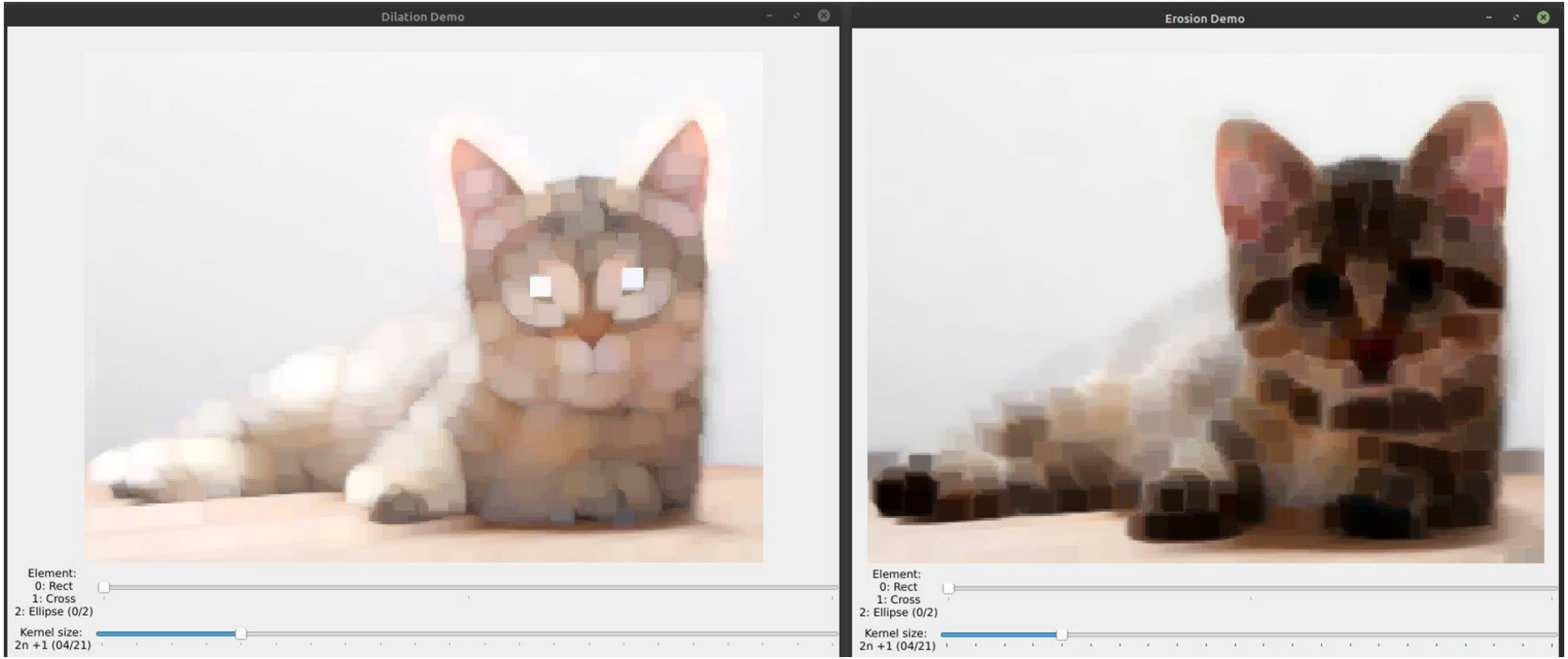




https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Grayscale/Color

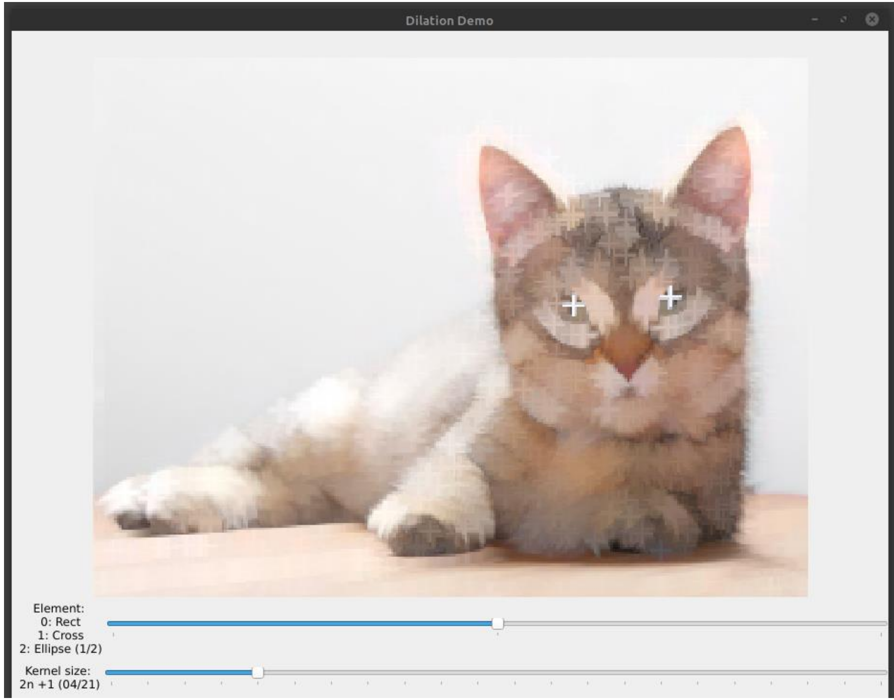
Idea of binary morphology can be extended to gray/color images with the use of max (dilation) and min (erosion) operation. Dilation and erosion change the brightness.
dilation - increased overall brightnes vs. erosion - dims the image

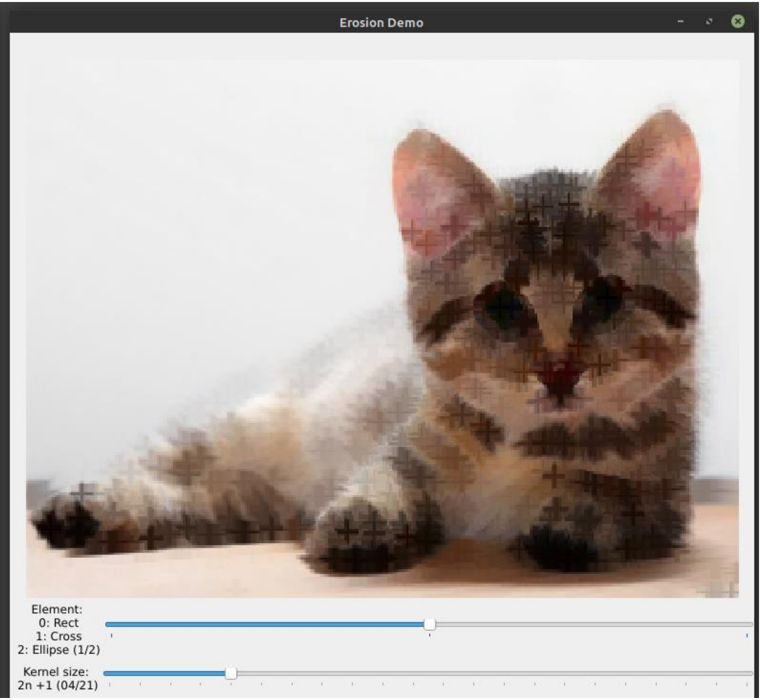


https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Grayscale/Color

Idea of binary morphology can be extended to gray/color images with the use of max (dilation) and min (erosion) operation. Dilation and erosion change the brightness.
dilation - increased overall brightnes vs. erosion - dims the image

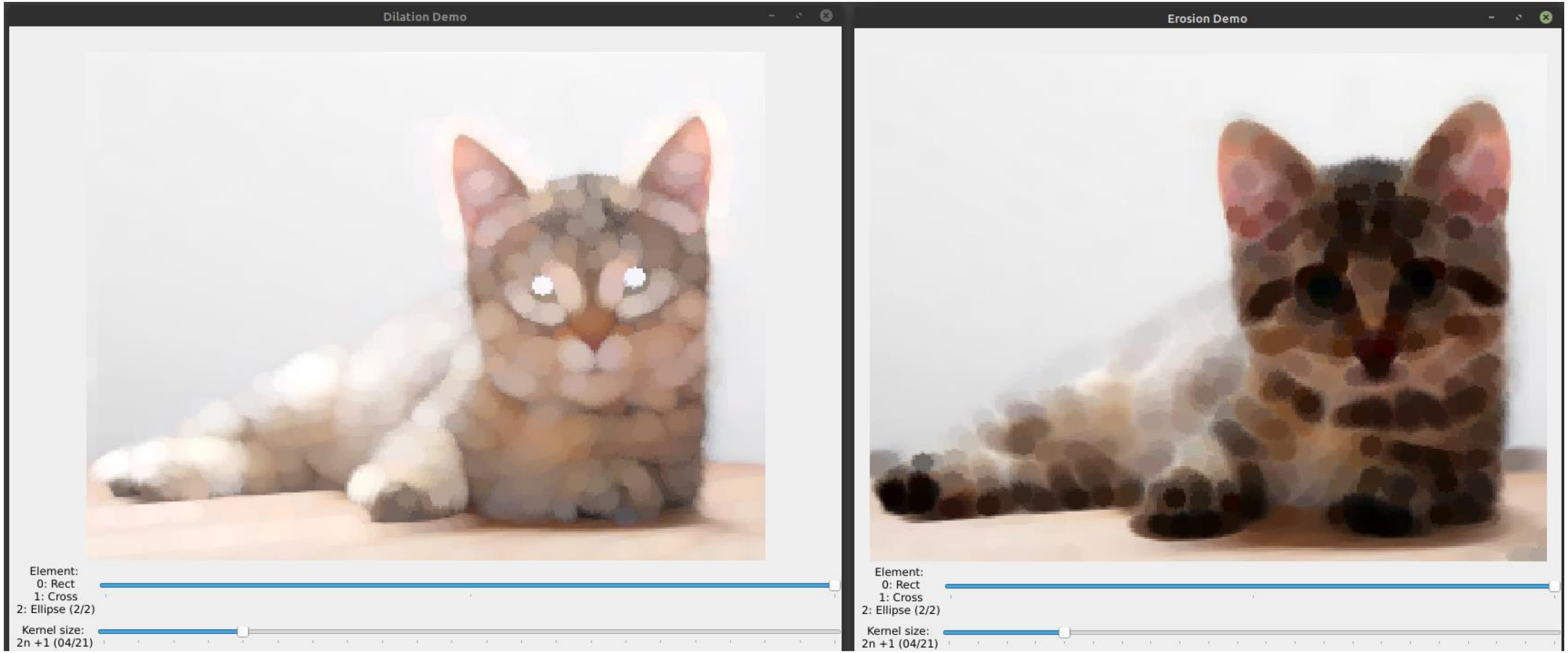




https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Grayscale/Color

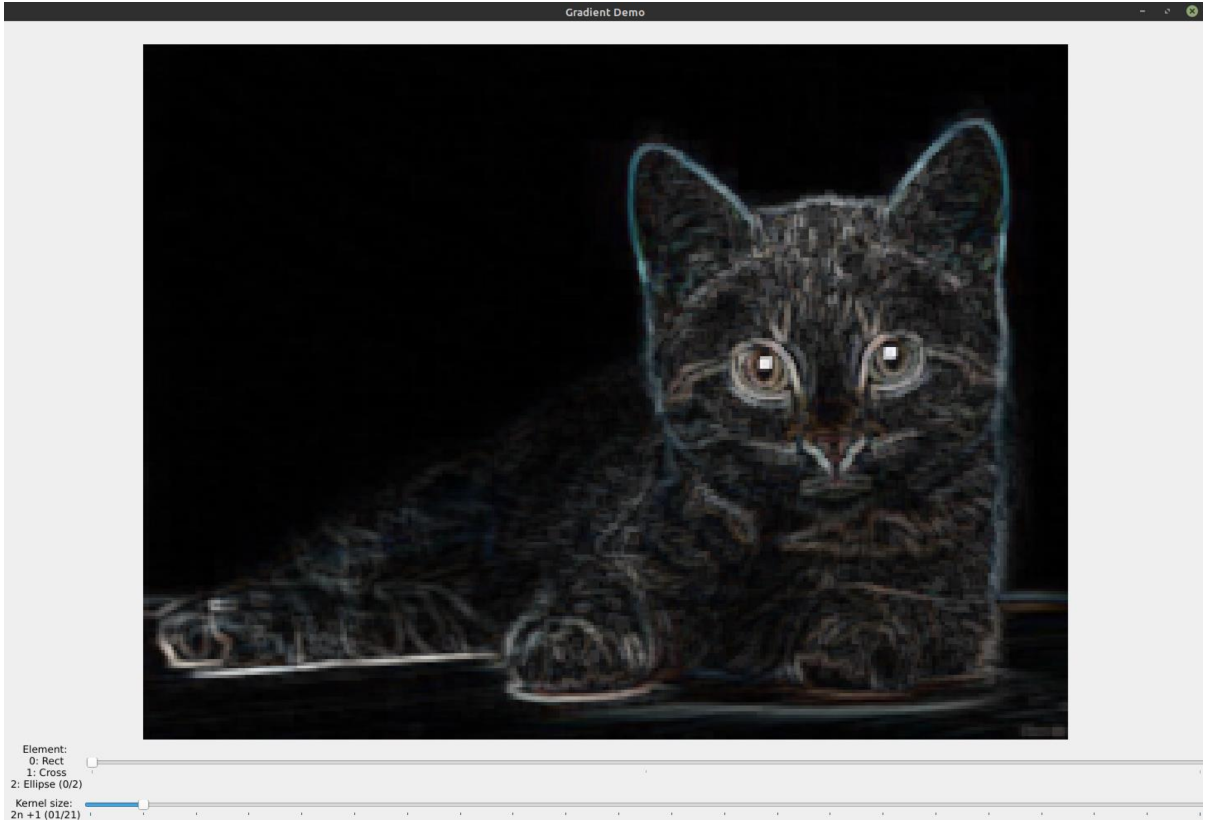
Idea of binary morphology can be extended to gray/color images with the use of max (dilation) and min (erosion) operation. Dilation and erosion change the brightness.
dilation - increased overall brightnes vs. erosion - dims the image



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html

Grayscale/Color

Analogous way: a morphological gradient is the difference between a dilation and an erosion



https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html https://docs.opencv.org/4.7.0/db/df6/tutorial_erosion_dilatation.html