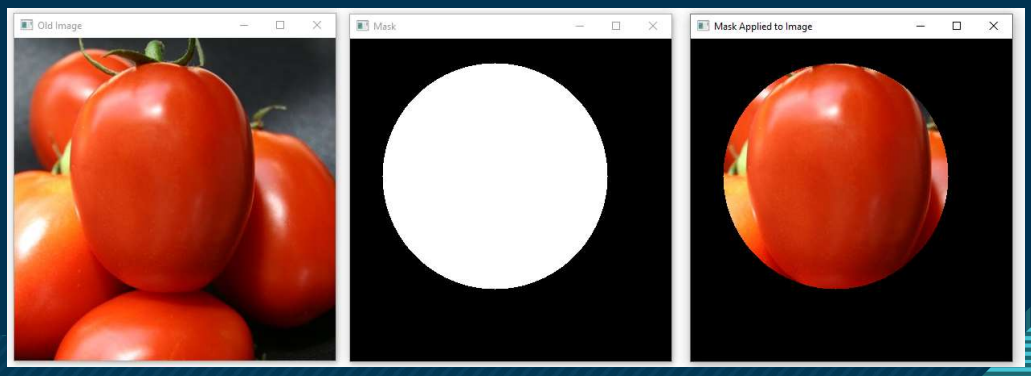
2.3. Basic Image Processing

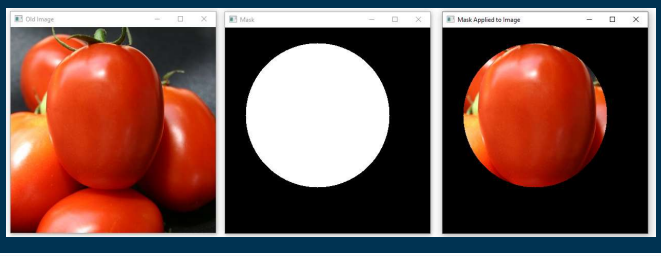
Masking: is a technique that allows us to focus only on the part of the image we are interested in.



1

2.3. Basic Image Processing

Masking:



```
import cv2
img = cv2.imread("C:/Users/quang/OneDrive/Pictures/Screenshots/new_tomato.jpg")
cv2.imshow("Old Image", img)
mask = np.zeros(img.shape[:2], dtype="uint8")
cv2.circle(mask, (180, 170), 140, 255, -1)
masked = cv2.bitwise_and(img, img, mask=mask)
cv2.imshow("Mask", mask)
cv2.imshow("After masking", masked)
cv2.waitKey(0)
```

2

2.3. Basic Image Processing

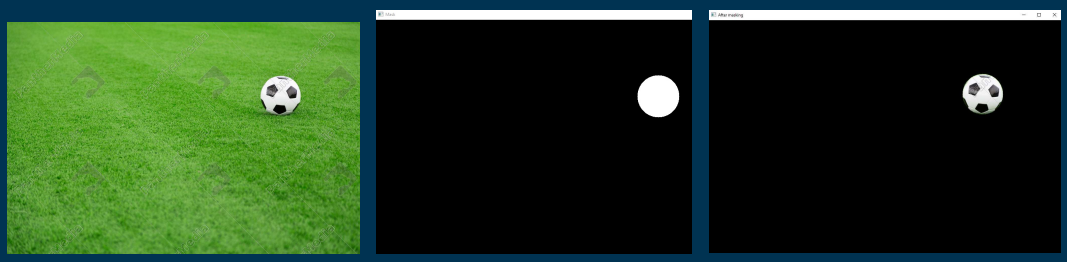
Exercise: Extract the ball using the "Masking" technique.

3

3

2.3. Basic Image Processing

Exercise: Extract the ball using the "Masking" technique.



4

4

2.3. Basic Image Processing

Exercise: Create an additional similar ball at a different position in the image.

5

5

2.3. Basic Image Processing

Exercise: Create an additional similar ball at a different position in the image (the quality of the copied ball is better)

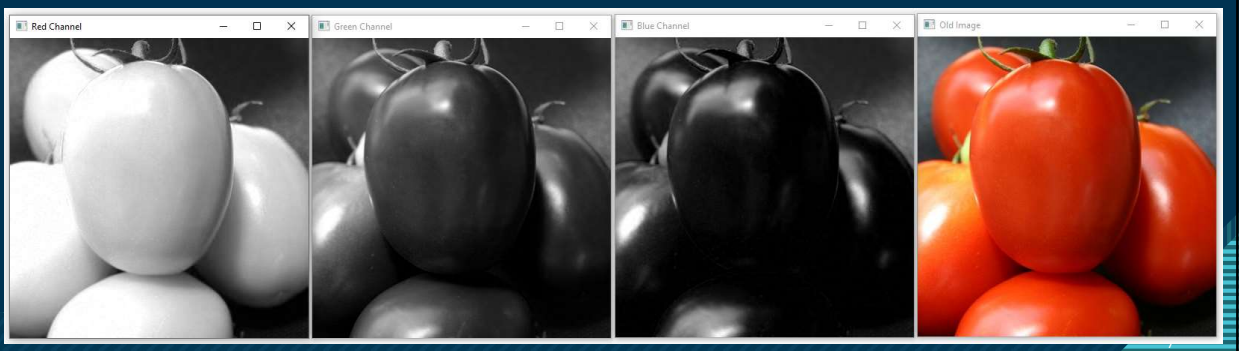
6

6

2.3. Basic Image Processing

A color image is composed of 3 channels: Red, Green, and Blue.

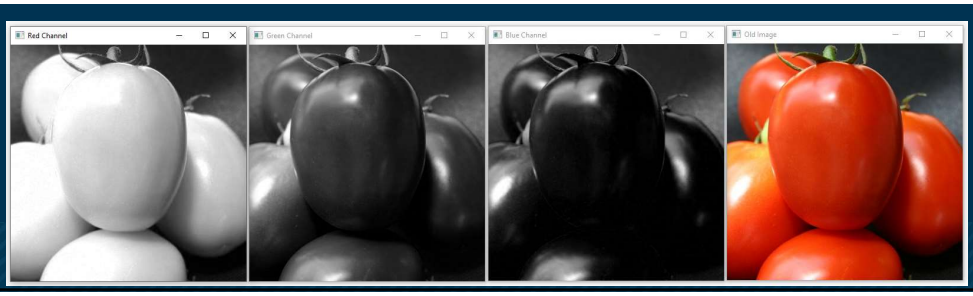
- Splitting channels: cv2.split()
- Merging channels: cv2.merge()



7

2.3. Basic Image Processing

```
import cv2
img = cv2.imread("C:/Users/quang/OneDrive/Pictures/Screenshots/new_tomato.jpg")
cv2.imshow("Old Image", img)
(B, G, R) = cv2.split(img)
cv2.imshow("Red Channel", R)
cv2.imshow("Green Channel", G)
cv2.imshow("Blue Channel", B)
cv2.waitKey(0)
```

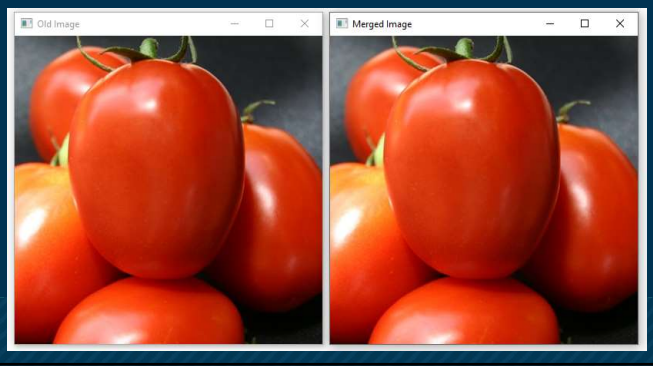


8

2.3. Basic Image Processing

Merging channels: cv2.merge()

```
merged_img = cv2.merge([B, G, R])
cv2.imshow("Merged Image", merged_img)
cv2.waitKey(0)
```



9

9

2.3. Basic Image Processing

Exercise: Split each color channel (Blue, Green, Red) from a color image and then increase the brightness of the Blue channel and decrease the brightness of the Red channel. Finally, merge the channels and display the resulting image.

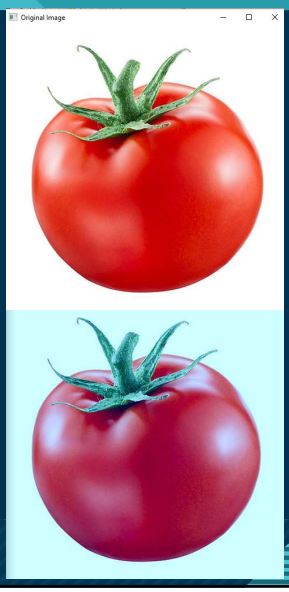
10

10

2.3. Basic Image Processing

Solution:

```
import cv2
import numpy as np
# Đọc ảnh màu
img = cv2.imread('C:/Users/quang/OneDrive/Desktop/tomato.jpg')
# Tách kênh màu
b, g, r = cv2.split(img)
# Tăng độ sáng của kênh Blue
b = cv2.add(b, 50)
# Giảm độ sáng của kênh Red
r = cv2.subtract(r, 50)
# Ghép các kênh lại với nhau
merged_img = cv2.merge([b, g, r])
# Hiển thị hình ảnh kết quả
cv2.imshow('Original Image', img)
cv2.imshow('Modified Image', merged_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

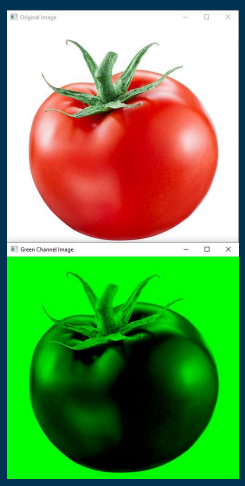


11

11

2.3. Basic Image Processing

Exercise 2: Split each color channel from a color image and then keep only the Green channel. Combine the other channels with a value of 0 to create a grayscale image containing only information from the Green channel.



12

12

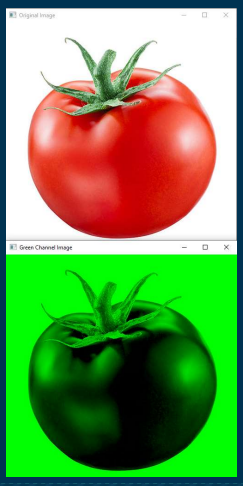
2.3. Basic Image Processing

Solution:

```
import cv2
import numpy as np

# Đọc ảnh màu
img = cv2.imread('C:/Users/quang/OneDrive/Desktop/tomato.jpg')
# Tách kênh màu
b, g, r = cv2.split(img)
# Tạo ảnh grayscale chỉ giữ lại kênh màu xanh lá
zero_channel = np.zeros_like(b)
green_img = cv2.merge([zero_channel, g, zero_channel])

# Hiển thị hình ảnh kết quả
cv2.imshow('Original Image', img)
cv2.imshow('Green Channel Image', green_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

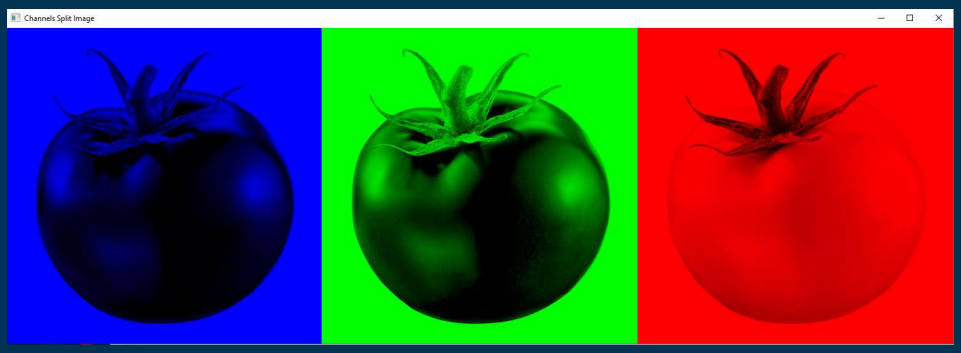


13

13

2.3. Basic Image Processing

Exercise 3: Split each color channel from a color image and display them side by side in a single image.



14

14

2.3. Basic Image Processing

Solution:

```
import cv2
import numpy as np

# Đọc ảnh màu
img = cv2.imread('C:/Users/quang/OneDrive/Desktop/tomato.jpg')
# Tách kênh màu
b, g, r = cv2.split(img)
# Tạo các hình ảnh chứa từng kênh màu riêng lẻ
blue_img = cv2.merge([b, np.zeros_like(b), np.zeros_like(b)])
green_img = cv2.merge([np.zeros_like(g), g, np.zeros_like(g)])
red_img = cv2.merge([np.zeros_like(r), np.zeros_like(r), r])
# Ghép các hình ảnh lại với nhau
combined_img = np.hstack((blue_img, green_img, red_img))
# Hiển thị hình ảnh kết quả
cv2.imshow('Original Image', img)
cv2.imshow('Channels Split Image', combined_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

15

15

2.3. Basic Image Processing - Challenge

Challenge 1: Advanced Image Augmentation Pipeline

- **Description:** Create an advanced image augmentation pipeline for data augmentation in machine learning.
- **Techniques:** Random translation, rotation, resizing, flipping, and cropping.
- **Challenge:**
  - Randomly apply a series of transformations to an input image.
  - Allow users to define the probability of each transformation.
  - Save and visualize the augmented images along with a grid of transformations applied.

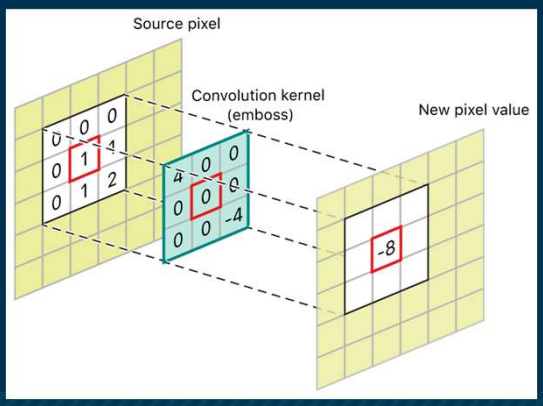
Note: random.randint(); random.uniform()

16

16

2.3. Basic Image Processing

Kernels: are small-sized matrices used for blurring, edge detection, sharpening, and other image processing functions.



Basically, the kernel sits on top of the large image and slides from left to right and from top to bottom, applying an operation at each (x, y) coordinate in the original image. We can blur and sharpen them, similar to how we edit images in Photoshop.

17

17

2.3. Basic Image Processing

Kernel application method:

- ✓ Select the coordinates (x, y) on the original image.
- ✓ Place the center of the kernel on that coordinate.
- ✓ Perform the convolution.
- ✓ Place the result from the previous step into the selected pixel of the original image.

131	162	232	84	91	207
104	-1	0	+1	237	109
243	-2	0	+2	135	26
185	-1	0	+1	61	225
157	124	25	14	102	108
5	155	176	218	232	249

Note: size of kernel and convolution method

18

18

2.3. Basic Image Processing

Kernels: consists of matrices MxN (M and N: odd numbers???)

For example: 3x3; 5x5

131	162	232	84	91	207
104	-1	0	+1	237	109
243	-2	0	+2	135	26
185	-1	0	+1	61	225
157	124	25	14	102	108
5	155	176	218	232	249

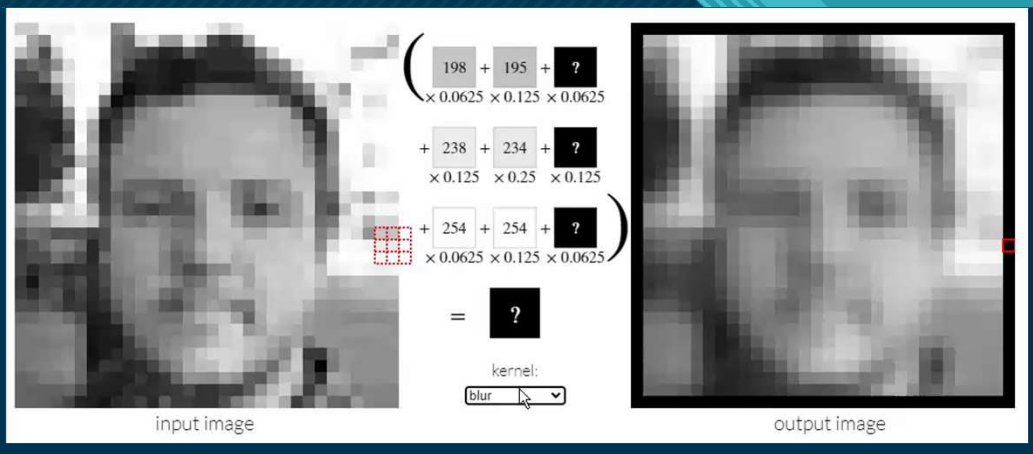
$$O_{i,j} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \star \begin{bmatrix} 93 & 139 & 101 \\ 26 & 252 & 196 \\ 135 & 230 & 18 \end{bmatrix} = \begin{bmatrix} -1 \times 93 & 0 \times 139 & 1 \times 101 \\ -2 \times 26 & 0 \times 252 & 2 \times 196 \\ -1 \times 135 & 0 \times 230 & 1 \times 18 \end{bmatrix}$$

$$O_{i,j} = \sum \begin{bmatrix} -93 & 0 & 101 \\ -52 & 0 & 392 \\ -135 & 0 & 18 \end{bmatrix} = 231$$

19

19

2.3. Basic Image Processing



20

20

2.3. Basic Image Processing

Calculate Convolution:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \star \begin{bmatrix} 202 & 119 & 154 \\ 106 & 119 & 11 \\ 186 & 48 & 250 \end{bmatrix} = ?$$

21

21

2.4. Morphological

Morphological image processing techniques:

- Erosion - Dilation
- Opening - Closing
- Morphological gradient
- Black hat - White hat

Applying morphological operations alters the shape and structure within an image. We can use morphological operations to increase or decrease the size of objects in the image. We can also use morphological operations to narrow the distance between objects or to expand them.

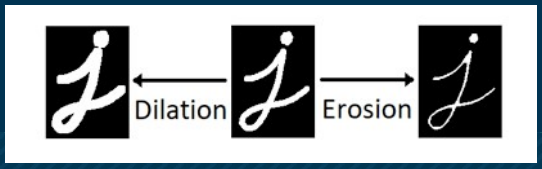
22

22

2.4. Morphological

Morphological image processing techniques:

- **Erosion:** This operation shrinks the white regions in a binary image and expands the black regions. Erosion is often used to remove small noise and to separate closely spaced objects.
- **Dilation:** This operation expands the white regions in a binary image and shrinks the black regions. Dilation is commonly used to fill small gaps, connect broken regions, and enhance details in the image.



23

23

2.4. Morphological

Morphological image processing techniques:

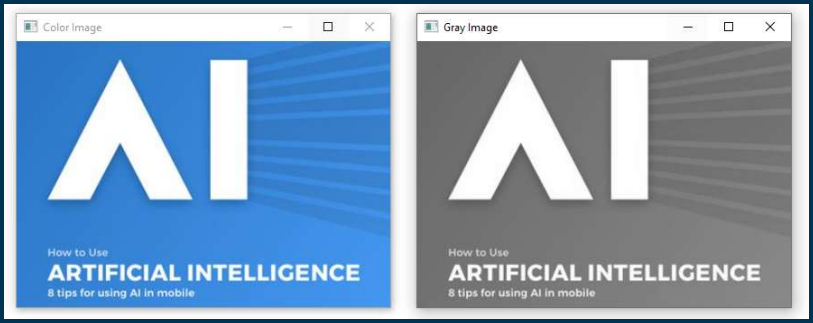
- **Erosion:** This operation shrinks the white regions in a binary image and expands the black regions. Erosion is often used to remove small noise and to separate closely spaced objects.
- **Dilation:** This operation expands the white regions in a binary image and shrinks the black regions. Dilation is commonly used to fill small gaps, connect broken regions, and enhance details in the image.

24

24

2.4. Morphological

- Converting a Color Image to Grayscale: cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)



Note: Morphological operations are simple transformations applied to binary or grayscale images.

25

25

2.4. Morphological

Converting a color image to grayscale is the process of changing an image from RGB (or other color spaces) to a single-color space. You can use one of the following methods:

1. Average Method

$$\text{Gray} = \frac{R+G+B}{3}$$

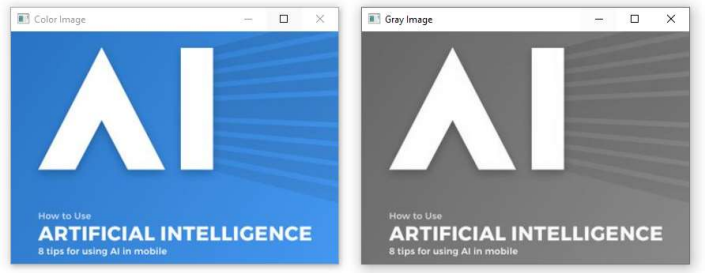
2. Weighted Method: Calculate the grayscale value using weighted sums for each color channel, as the human eye is more sensitive to some colors than others. A commonly used formula is:

$$\text{Gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

26

26

2.4. Morphological



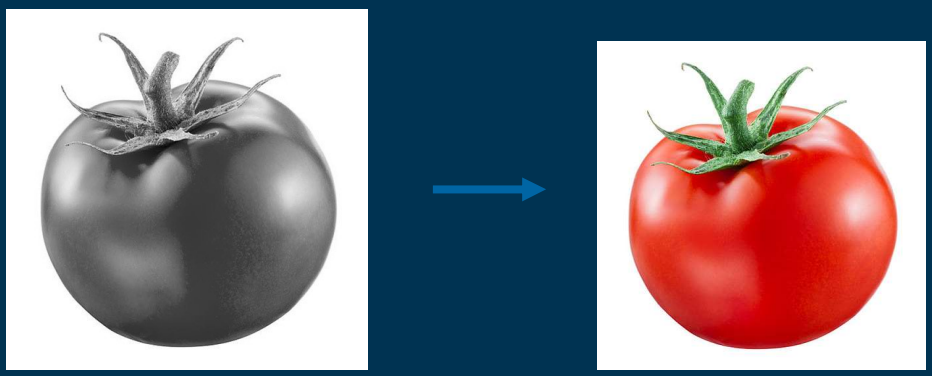

```
import cv2
img = cv2.imread("C:/Users/quang/OneDrive/Pictures/Screenshots/logo.png")
cv2.imshow("Color Image", img)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("Gray Image", gray_img)
cv2.waitKey(0)
```

27

27

2.4. Morphological

Convert grayscale to color image: cv2.applyColorMap(gray_img, cv2.COLORMAP_JET)

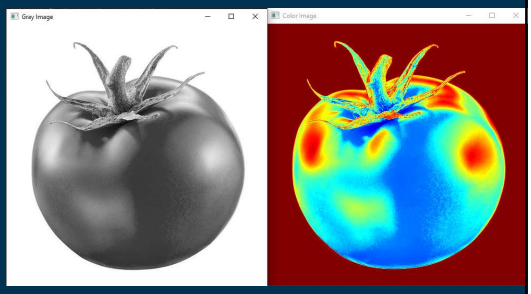


28

28

2.4. Morphological

```
import cv2
import numpy as np
# Đọc ảnh xám
gray_img =
cv2.imread('C:/Users/quang/OneDrive/Desktop/tomato_gray.jpg',
cv2.IMREAD_GRAYSCALE)
# Áp dụng colormap để chuyển ảnh xám sang ảnh màu
color_img = cv2.applyColorMap(gray_img, cv2.COLORMAP_JET)
# Hiển thị ảnh xám và ảnh màu
cv2.imshow('Gray Image', gray_img)
cv2.imshow('Color Image', color_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



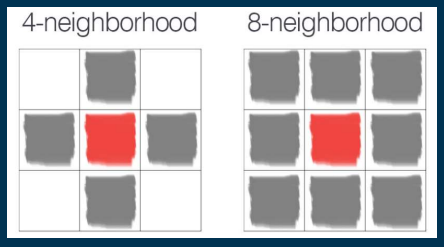
To restore a grayscale image to a color image, you need to add color information to the color channels of the image. However, this is not a straightforward task because you need to know the original color information of each pixel, which is often not feasible if you only have the grayscale image. However, you can add pseudo-color to the grayscale image using OpenCV's color mapping techniques.

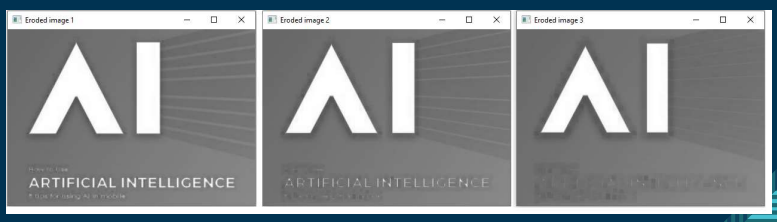
29

29

2.4. Morphological

erosion is a morphological operation used to reduce the size of the white regions (foreground) in a binary image and to expand the black regions (background). It is typically used to remove small noise points or to separate connected objects.



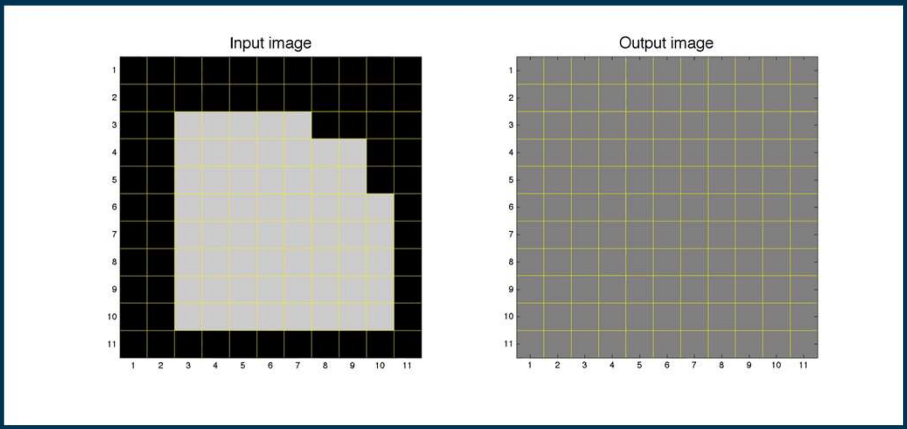


30

30

2.4. Morphological

erosion



31

31

2.4. Morphological

erosion


```
for i in range(0, 3):
    eroded_img = cv2.erode(gray_img.copy(), None, iterations=i + 1)
    cv2.imshow("Eroded image {} ".format(i + 1), eroded_img)
    cv2.waitKey(0)
```

32

32

2.4. Morphological

Example: remove the gray spots from the image

33