🦋

Module 5

Morphology

▼ What are binary images?

images whose pixels can have only two possible colors, for example, black and white images

▼ What are the most common binary image operations?

morphological operations since they change the “shape” of the underlying binary objects

▼ What is mathematical morphology?

- an image processing technique that deals with the shape of objects present in an image
- it is very useful for noise removal, image enhancement, edge detection, and image segmentation
- although morphological operations are normally applied to binary images, there are also versions for grayscale images

▼ Morphological operations require two data inputs. What are they?

1. an image to be eroded or dilated
2. a structuring element (or kernel/filter)

▼ What is the dilation operator?

- This operator gradually enlarges the boundaries of regions of foreground (or white) pixels.

- Therefore, areas of foreground pixels grow in size while holes within those regions become smaller.

▼ What is the erosion operator?

- This operator erodes away the boundaries of regions of foreground pixels.
- Therefore, areas of foreground pixels shrink in size while holes within those areas become larger.

▼ What is the opening operator?

- This operator preserves foreground regions that have a similar shape to the kernel, or that can completely contain the kernel, while eliminating all other regions of foreground pixels.
- The basic **effect is similar to erosion**, however, it is **less destructive of the original boundary shape.**

▼ What is the closing operator?

- This operator preserves background regions that have a similar shape to the kernel, or that can completely contain the kernel, while eliminating all other regions of background pixels.
- The basic **effect is similar to dilation**, however, it is **less destructive of the original boundary shape.**

▼ What is the skeletonization operator?

- This operator reduces foreground regions to a skeletal remnant that largely preserves the extent and connectivity of the original region while throwing away most of the original foreground pixels.

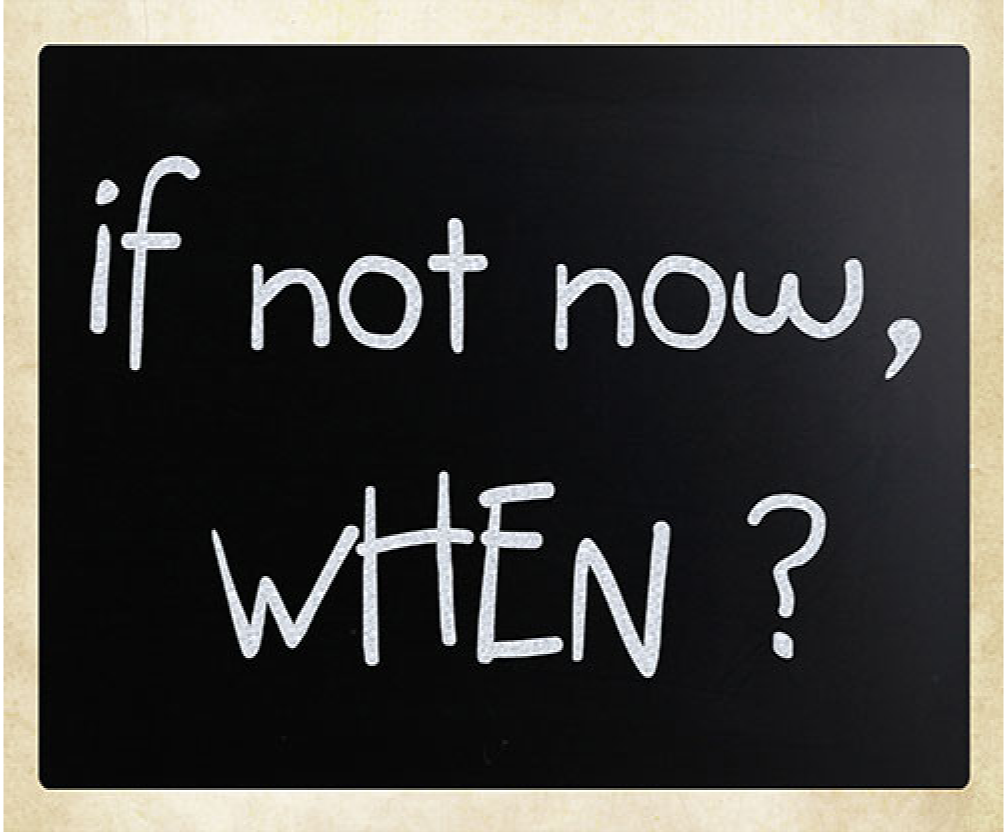
▼ OpenCV example: convert a handwriting image into a binary image

- OpenCV has a function cv2.threshold() into which you will pass in the parameter cv2.THRESH_BINARY to perform this task.

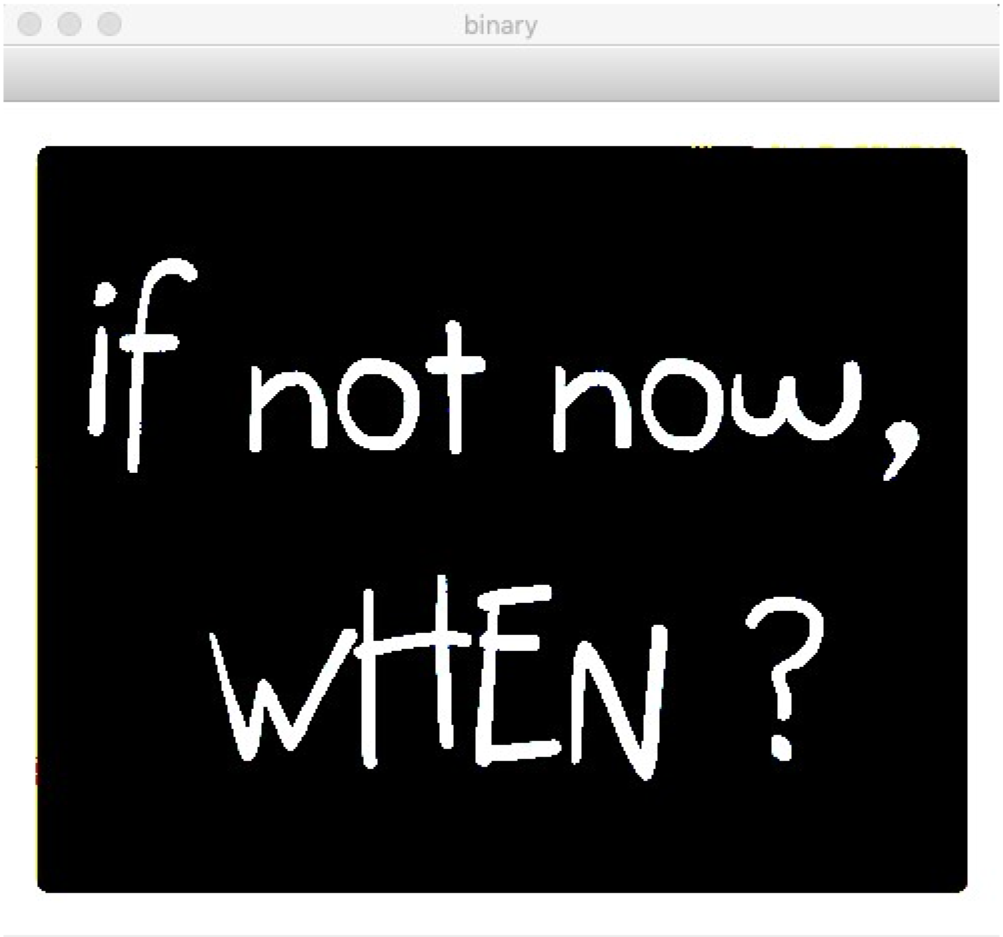
```
import cv2

img = cv2.imread('handwriting.jpg')
```

```
(thresh, binary_img) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)


cv2.imshow('binary', binary_img)


cv2.waitKey(0)

cv2.destroyAllWindows()
```

Original Image:



Binary Image:



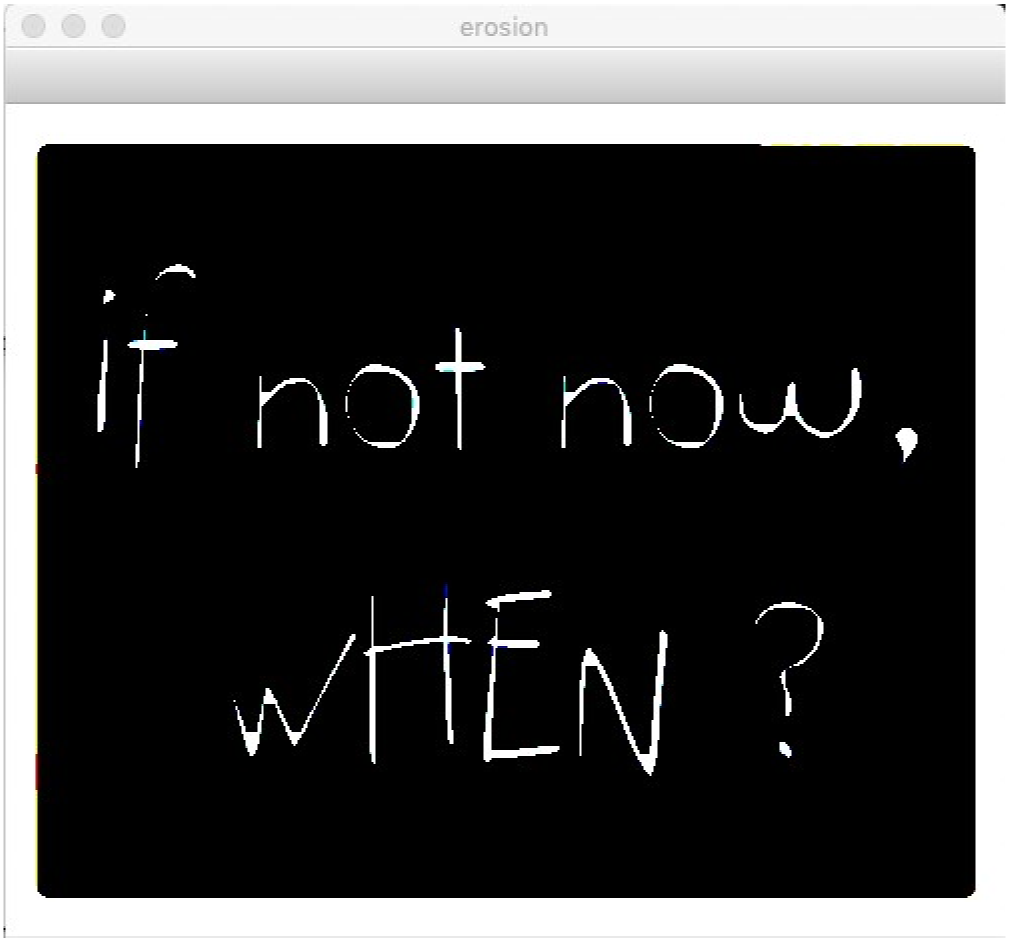
▼ OpenCV example: erosion on a binary image

- OpenCV has a function **cv2.erode()** to perform erosion on an image.
- The following code uses a 5x5 kernel filled with all ones as the structuring element.
- Additionally, you pass in the parameter, **iterations = 1** to perform only one erosion (or iteration) at once.

```
import cv2

import numpy as np


img = cv2.imread('handwriting.jpg')
```

```
(thresh, binary_img) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

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


erosion = cv2.erode(binary_img, kernel, iterations = 1)


cv2.imshow('erosion', erosion)


cv2.waitKey(0)

cv2.destroyAllWindows()
```

Erosion:

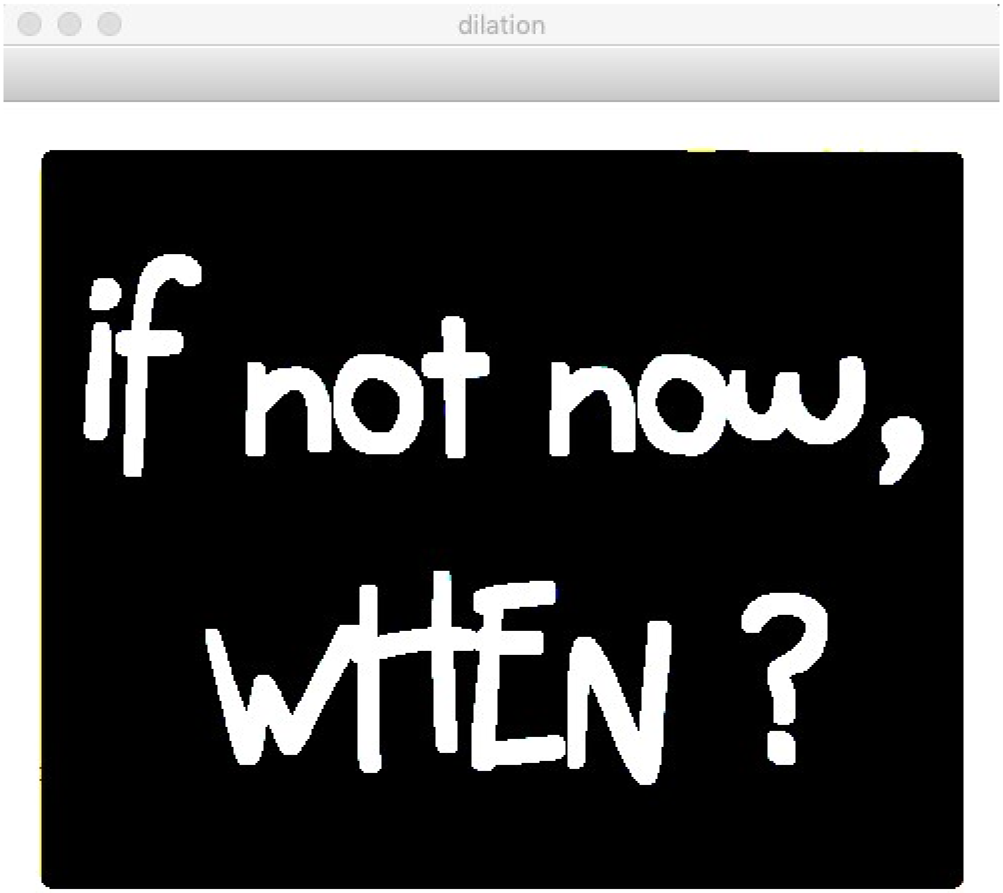


▼ OpenCV example: dilation on a binary image

- OpenCV has a function **cv2.dilate()** to perform dilation on an image.
- The following code uses a 5x5 kernel filled with all ones as the structuring element.
- Additionally, you pass in the parameter, **iterations = 1** to perform only one dilation (or iteration) at once.

```
import cv2
import numpy as np

img = cv2.imread('handwriting.jpg')

(thresh, binary_img) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((5,5), np.uint8)

dilation = cv2.dilate(binary_img, kernel, iterations = 1)

cv2.imshow('dilation', dilation)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Dilation



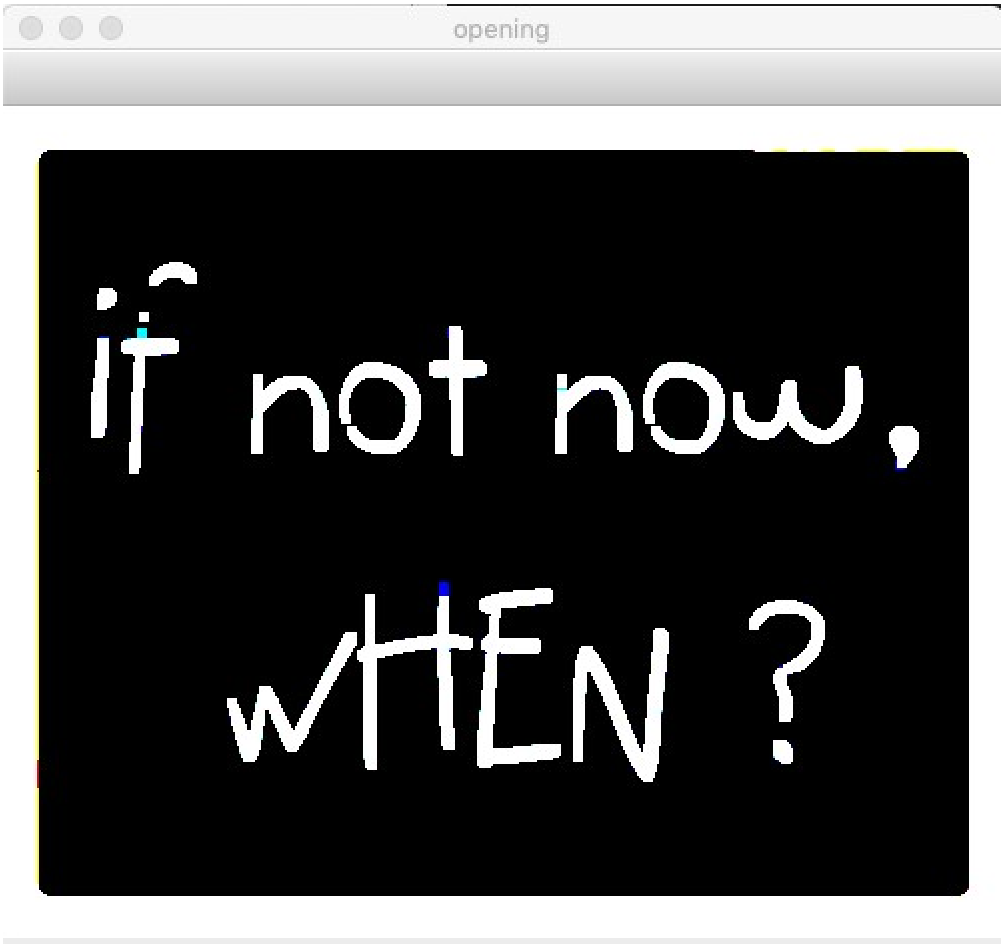
▼ OpenCV example: opening on a binary image

- OpenCV has a function **cv2.morphologyEX()** to perform advanced morphological operations on an image.
- The following code uses a 5x5 kernel filled with all ones as the structuring element.
- Additionally, you pass in the function **cv2.MORPH_OPEN** to specify that you are applying an opening operation.

```
import cv2

import numpy as np


img = cv2.imread('handwriting.jpg')
```

```
(thresh, binary_img) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

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


opening = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel)


cv2.imshow('opening', opening)


cv2.waitKey(0)

cv2.destroyAllWindows()
```

Opening

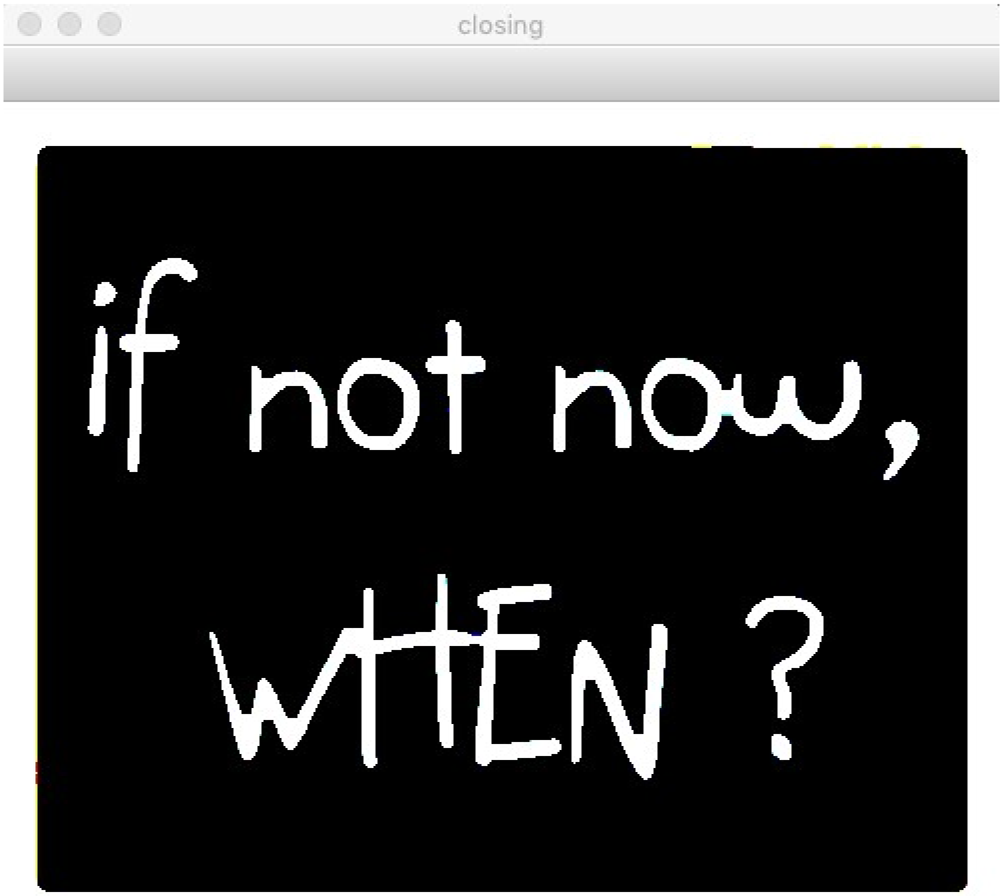


▼ OpenCV example: closing on a binary image

- OpenCV has a function **cv2.morphologyEX()** to perform advanced morphological operations on an image.
- The following code uses a 5x5 kernel filled with all ones as the structuring element.
- Additionally, you pass in the function **cv2.MORPH_CLOSE** to specify that you are applying a closing operation.

```
import cv2
import numpy as np

img = cv2.imread('handwriting.jpg')

(thresh, binary_img) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((5,5), np.uint8)

closing = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, kernel)

cv2.imshow('closing', closing)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Closing:



if not now,
WHEN ?