Morphological Image Processing for Detecting Objects in Binary Images

Dipanjan Banik (d.banik@stud.uis.no)
Isar Adnan (i.adnan@stud.uis.no)
University of Stavanger, Norway

Summary

In this project, we present the implementation of morphological image processing for detecting objects in binary images. The tasks have been done in python using the OpenCV library. To visualize the output, matplotlib library has been used.

1. Introduction

The binary version of color and grayscale photos, which are the most common, is frequently utilized in processing. Pixels are added or subtracted from images during morphological processing. The structure and shape of the things are studied in order to identify them. Binary convolution and correlation are the core processes in this processing, which are based on logical rather than mathematical calculations. The core operations are dilation and erosion, and the rest of the operations and algorithms are built around them. The minimum and maximum operators are used to extend morphological processing to gray-level images.

2. Theory

Morphological image processing is a method of altering a picture's pixels. The pixels in a grayscale image are recognized by binary values of 0 and 1, and the procedure is carried out utilizing sophisticated image processing algorithms or less mathematically complex activities. Erosion and dilation, as well as opening and closure, are examples. Morphological image processing is used to remove undesirable artifacts from images and increase their quality. It's used to process fingerprints, view images from space telescopes, and analyse medical scans, among other things.

A group of pixels termed object pixels is used to represent an object in an image. Background pixels are white and are shown individually. The erosion process converts pixels related with the object's boundary to background pixels, whereas the dilation action changes the bordering background pixels to ones associated with the object. During erosion, objects shrink, but during dilatation, they increase or even combine.

In morphological image processing, the two techniques can be combined to alter an image by performing erosion and then dilation, which results in opening. To smooth up the image, filaments and isolated pixels can be deleted from the object in this way. The closure operation can be used to filter background pixels, removing holes and pixels that are already known to be out of place. Skeletonization is another morphological image processing approach in which superfluous pixels are deleted to generate single lines. It is frequently utilized in the processing of fingerprints.

To change the image visualization, image processing apps use a few rules or set theory, a mathematical notion that is often more complex than required. The computer only considers the pixels linked with the object when converting a pixel from object to background. It also focuses on the edge regions, examining nearby background pixels before changing the black pixels. If an object pixel is to be modified, it must be surrounded by more than one comparable pixel, as removing pixels at the end of a line can cause the image to distort.

The objective of a morphological image processing tool is to preserve objects entire. If removing a pixel will cause a single item to break up, the program will not do so. Picture processing technology consists of a variety of software applications that enable image modifications to be replayed, changes to be reverted to desired configurations, and analysis of how specific changes influence different sections of an image.

Structuring Element: It is the base structure on which a morphological operation will be performed. A Structuring Element's shape can alter depending on the type of shape we're looking for in a picture. Here are several SE examples in rectangular array format. (Surely, in order to express a SE in programming, we must convert it to a 2D matrix representation.)

There are four morphological operations:

I. Erosion
II. Dilation
III. Opening
IV. Closing

Erosion: The Structure Element travels across the picture in Erosion, and where the image pattern and the Structure Element pattern match exactly, the pixel at the seed point of the Structure Element in the image is changed to 1, and where they don't, to 0. It's worth noting that the erosive process is frequently depicted by the $\ominus$ symbol.

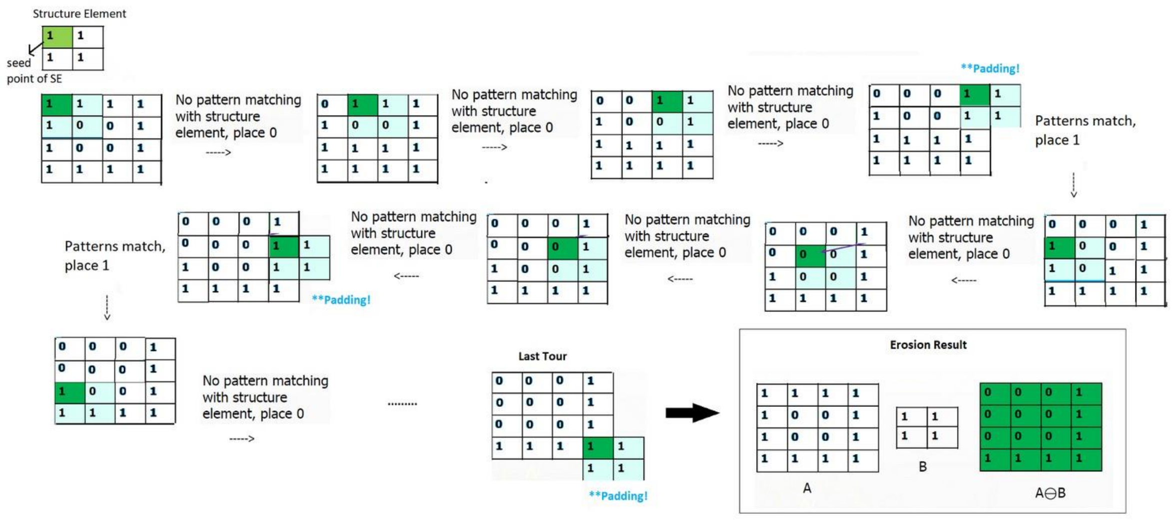


Figure: Erosion [8]

Dilation: The Structure Element passes across the image in Dilation, and if the image pattern and the Structure Element pattern have one pixel in common, 1 is written as the output pixel value, 0 if they don't. It's worth noting that the dilation operation is commonly symbolized by the $\oplus$ symbol.

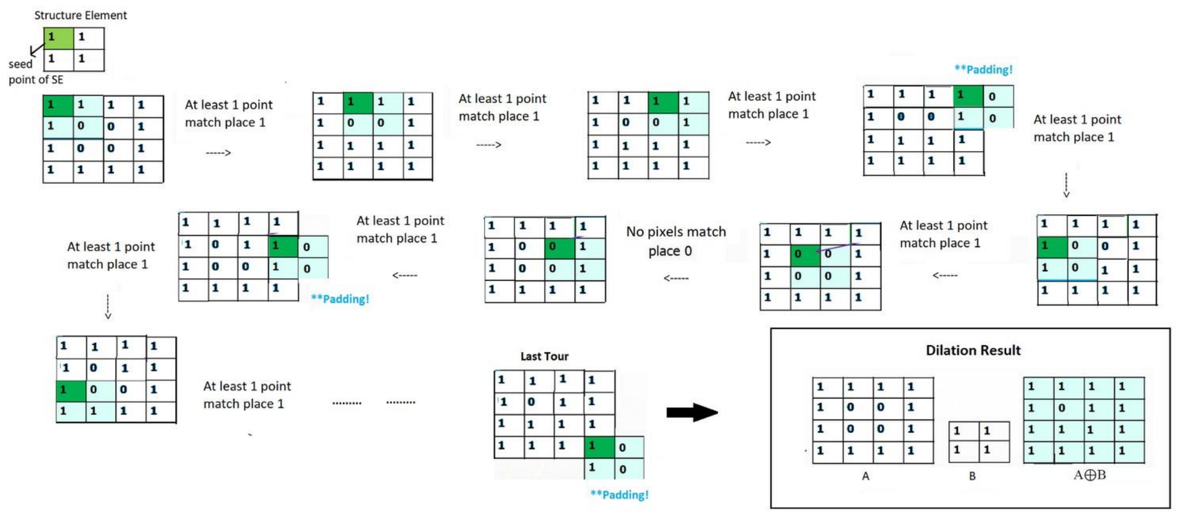


Figure: Dilation [8]

Opening: Opening is a procedure that involves first performing an erosion operation and then doing a dilation operation. It removes the obtained image's tiny protrusions. Opening is used to remove the obtained image's inherent noise.

Closing: Closing is a procedure that involves first performing a dilatation operation and then doing an erosion operation. It removes the little holes from the image that has been obtained. Closing is a technique for blending narrow breaks and smoothing contours.

Morphological Gradient: The procedure that equals the difference between dilation and erosion of a picture is called morphological gradient. The contrast strength in the nearby pixels is represented by each pixel value in the final image. This is used for edge detection, segmentation, and determining an object's outline. It is used to create the image's outline. Internal and external gradients are the two kinds of gradients. Internal boundaries of items that are brighter than their background and external limits of things that are darker than their background are enhanced by the internal gradient. The internal gradient creates a mask of the internal boundaries of the foreground picture objects in binary images.

The Erosion output image is subtracted from the Dilation output image. While external boundary extraction is useful for getting the exterior side pixels of an edge and internal boundary extraction is useful for getting the internal side pixels of an edge, morphological gradient is useful for getting the pixels "on" the edge. When compared to internal or exterior boundary extraction, this results in thicker edges.

Top hat and Black hat Transform: The top-hat and black-hat transforms are operations for extracting minor elements and details from photographs. The difference between the input picture and its opening by some structuring element is characterized as the top-hat transform, whilst the difference between the closing and the input image is defined as the black-hat transform. These transforms are used for a variety of tasks in image processing, including feature extraction, background equalization, picture enhancement, and more.

External Boundary Extraction: To acquire external edges, we subtract the original input picture from the image's Dilation.

Internal Boundary Extraction: Internal edges are obtained by subtracting the picture's Erosion from the original input image.

3. Implementation

Image components that are important in the representation and description of region layout are extracted using morphological techniques. Morphological operations are a set of basic tasks that depend on the structure of the image. It is usually done with binary images and it requires two data sources: one is the input image, and the other is the structural component, commonly known as the kernel. There are also some sets of the task we will discuss in the later part of this section.

3.1 Image selection

We have used a set of images mostly from google image. Since the concept behind this work is based on morphological image processing, we have shown two or more examples in each part of the result analysis.

3.2 Code structure

The entire programming was done in one main file for ease of interpretation and flexibility in modifying the parameters. We have separated the functions of morphological operators for a better understanding of each output segment. We have used ***python*** as our base interpreter and coding, testing and result analysis has been done in Jupyter Notebook. Additionally, we have used the *OpenCV* library for implementing the main functional task. *NumPy* library is used for data manipulation, numerical operations and generating kernel and *matplotlib* library is used for plotting and showing the results.

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

3.2.1 Converting input image into a binary image

Morphological operations are simple transformations applied to binary or grayscale images. More specifically, we apply morphological operations to shapes and structures inside of images [1]. Binary images consist of pixels that can have exactly two colors, usually black and white. This types image may contain numerous imperfections. Noises and textures can be distorted when a simple threshold produces the binary regions. Morphological Operations in Image Processing pursues the goals of removing these imperfections by accounting for the form and structure of the image. For creating a binary image, it needs two data sources, one is the input image, the second one is called structuring component. First, we have imported the image using *cv2.imread* function and then converted the image to a grayscale image because of simplicity. With the smaller data set, it enables us to do more complex operations in a shorter time.

After loading the grayscale image, we generated the histogram to find the threshold value. With the help of NumPy histogram (*np.histogram*) formula, we seek to convert a grayscale image to a binary image, where the pixels are either 0 or 255 [3]. If the pixel value is smaller than the threshold, it is set to 0; otherwise, it is set to a maximum value [2]. For this project, we have applied two types of thresholding methods. One is the simple threshold (*cv2.threshold*), and another is the adaptive threshold (*cv2.adaptiveThreshold*). Most of the coding was done using simple thresholding but we have also shown the comparison result.

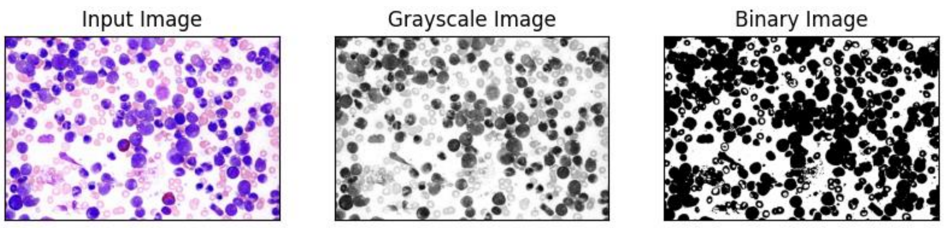


Figure: Conversion of binary image

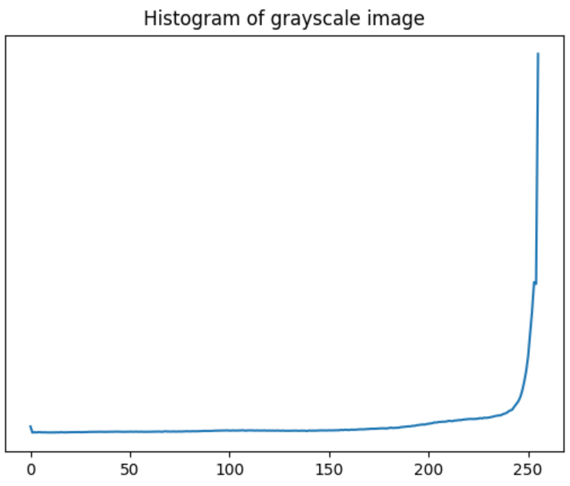


Figure: Histogram of grayscale image

From the histogram, we can see that the threshold value lies close to 230. In the cv2.threshold function, we had manually input our grayscale image, threshold value and a formula (THRESH_BINARY). The cv2.threshold function returns a tuple of 2 values. Then we can save our binary image for further processing. In our experiment, any pixel value greater than 230 is set to 255 and any value less than 230 is set to 0. But the threshold value is different for each example.

```
(thresh, binaryImage) = cv2.threshold(grayScaleImage, 230, 255,
cv2.THRESH_BINARY)
```

Figure: Thresholding image using OpenCV

3.2.2 Applying algorithm

After the conversion of a binary image, we then proceed to select the structuring element. The structuring element is a small binary image which is consists of a small matrix pixel. The value of the structure element is either zero (0) or one (1). The structured element is also referred to as the kernel. The kernel is a matrix that identifies the pixel in the image being processed and defines the neighbourhood used in the processing of each pixel is used to probe an image in these Morphological techniques. It is placed in all conceivable positions in the input image and compared to the pixels in the surrounding area. Each pixel of a structuring element is associated with the equivalent pixel of the

```
kernel = np.ones((2, 2), np.uint8)
```

Figure: Generating kernel using NumPy

neighborhood under the structuring element when it is inserted in a binary image. If each of the structuring element's pixels is set to one, the corresponding image pixel is also one, the structuring element is said to fit the image. A structuring element is said to hit, or intersect, an image if at least one of its pixels set to 1 corresponds to a matching image pixel [5]. Zero-valued pixels of the structuring element are ignored [5].

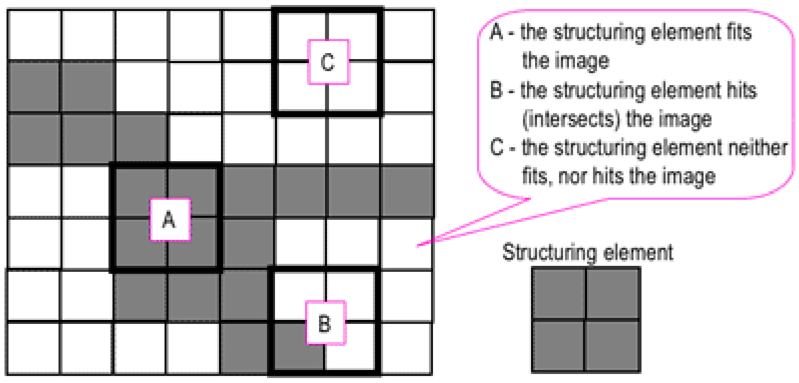

A - the structuring element fits the image
B - the structuring element hits (intersects) the image
C - the structuring element neither fits, nor hits the image
Structuring element
A
B
C

Figure: Probing of an image with a structuring element [4]

When we have selected the kernel, we then proceed to the main part of the morphological image operation. In our project, we have done some basic fundamental operations: erosion, Dilation, Opening, closing, top hat, bottom hat, and morphological gradient. Along with these experiments, we have also shown two types of edge detection process and noise reduction process form an image with the help of composite result of opening and closing operation. Now we will discuss the code structure.

Erosion: Making a call to the *cv2.erode* function performs the actual erosion. This function accepts three arguments, two of which are mandatory and one of which is optional. The first argument is the image that we want to erode. The second argument takes the structuring element. As we have already defined above that using NumPy we have generated the structuring element. The number of iterations the erosion will be conducted is the final argument.

```
erosionExample = cv2.erode(binaryImage, kernel, iterations=1)
```

Figure: OpenCV erosion function

Dilation: Dilation is the total opposite of erosion. We apply dilations using the *cv2.dilate* function. The coding structure of dilation is the same as erosion. *cv2.dialate* also takes three input parameters.

```
dilationExample = cv2. dilate (binaryImage, kernel, iterations=1)
```

Figure: OpenCV dilation function

Opening: An opening is a type of operator that is strongly related to dilation and erosion. It's essentially a two-step procedure that starts with erosion and ends with dilatation. Making a call to the *cv2.morphologyEx* function performs the actual opening action. In some ways, the *cv2.morphologyEx* function is abstract: it lets us feed in any morphological operation we choose, followed by our kernel. For opening we will choose *cv2.MORPH_OPEN*. The image on which we wish to apply the morphological process is the first necessary argument of *cv2.morphologyEx*. The second argument is to consider the type of morphological operation. The last required argument is the kernel that we are using.

```
openingExample = cv2.morphologyEx(binaryImage, cv2.MORPH_OPEN, kernel)
```

Figure: OpenCV opening function

Closing: The closing operation is an inverse of the opening. This method also takes the same input as the opening but for closing, we will choose *cv2.MORPH_CLOSE* as out input parameter.

```
closingExample = cv2.morphologyEx(binaryImage, cv2.MORPH_CLOSE, kernel)
```

Figure: OpenCV close function

Top Hat: A top hat morphological operation is the difference between the original input image and the opening. A top hat morphological operation is the difference between the original input image and the opening. This will also take the same input parameter but we simply change the type of operator to *cv2.MORPH_TOPHAT*.

```
tophatExample = cv2.morphologyEx(binaryImage, cv2.MORPH_TOPHAT, kernel)
```

Figure: OpenCV tophat function

Black Hat: The black-hat operation is used to do the opposite, enhance dark objects of interest in a bright background [6]. This will also take the same input parameter but we simply change the type of operator to *cv2.MORPH_BLACKHAT*.

```
blackhatExample = cv2.morphologyEx(binaryImage, cv2.MORPH_BLACKHAT, kernel)
```

Figure: OpenCV blackhat function

Morphological gradient: We make use of the operation *cv2.MORPH_GRADIENT* in morphologyEx() function to perform morphological gradient operation on a given image.

```
morphgradientExample = cv2.morphologyEx(binaryImage, cv2.MORPH_GRADIENT,
kernel)
```

Figure: OpenCV morphological gradient function

3.2.3 Plotting results

We have used Python’s *matplotlib* library for plotting the data in jupyter notebook. All the results and analysis will be discussed in the late part of this section

4. Result analysis

4.1 Erosion

Erosion Erodes the boundaries of the foreground object in Morphological processes. They create an output image by modifying an input image with a structural element. We have used a 2x2 kernel for generate the output image.

From examples 1 and 2, we can see that it removes the white noises and makes the text more visible to us. In erosion, bright areas of the image (the background of the image) get thinner, whereas in the dark zones, the writings get bigger.

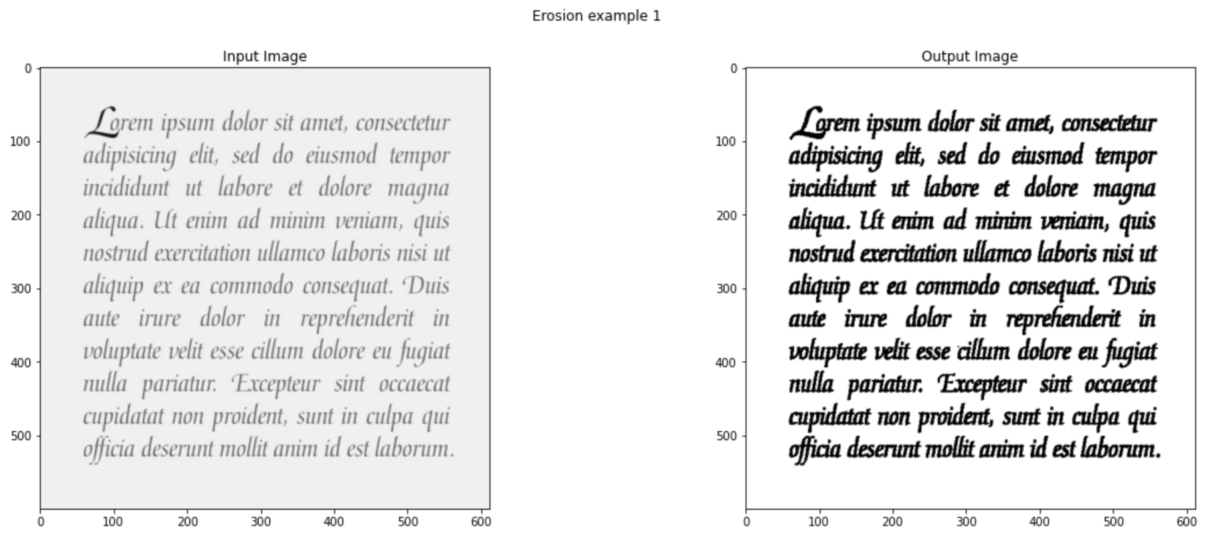


Figure: Erosion example 1

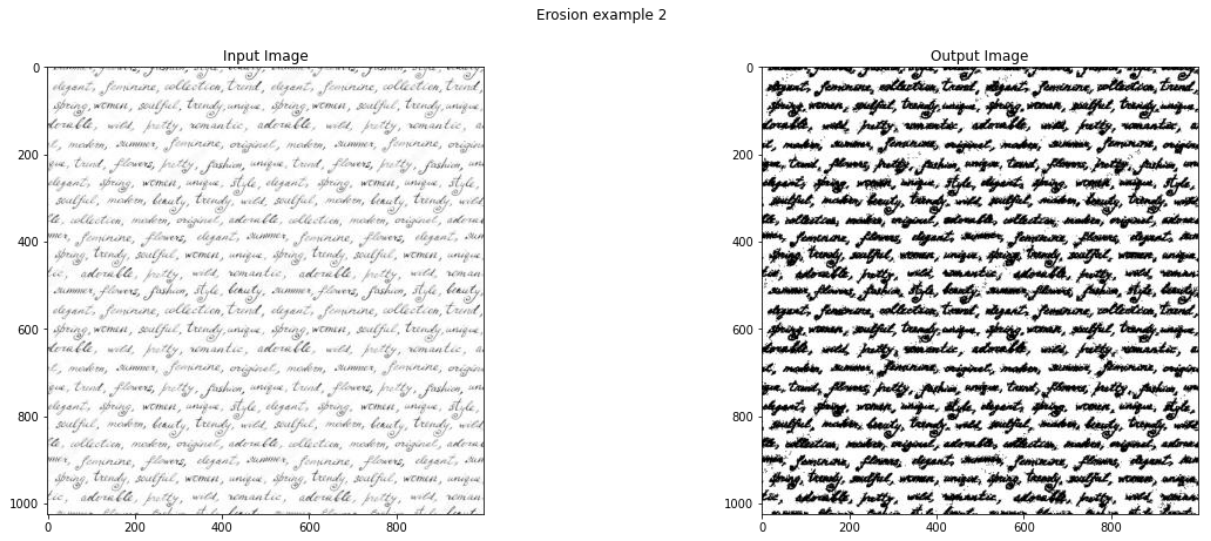


Figure: Erosion example 2

4.2 Erosion using adaptive thresholding

We have also compared our results using the adaptive thresholding method(*cv2.adaptiveThreshold()*). Here we have used the adaptive method *cv2.ADAPTIVE_THRESH_GAUSSIAN_C* and results are showing better than the non-adaptive thresholding method.

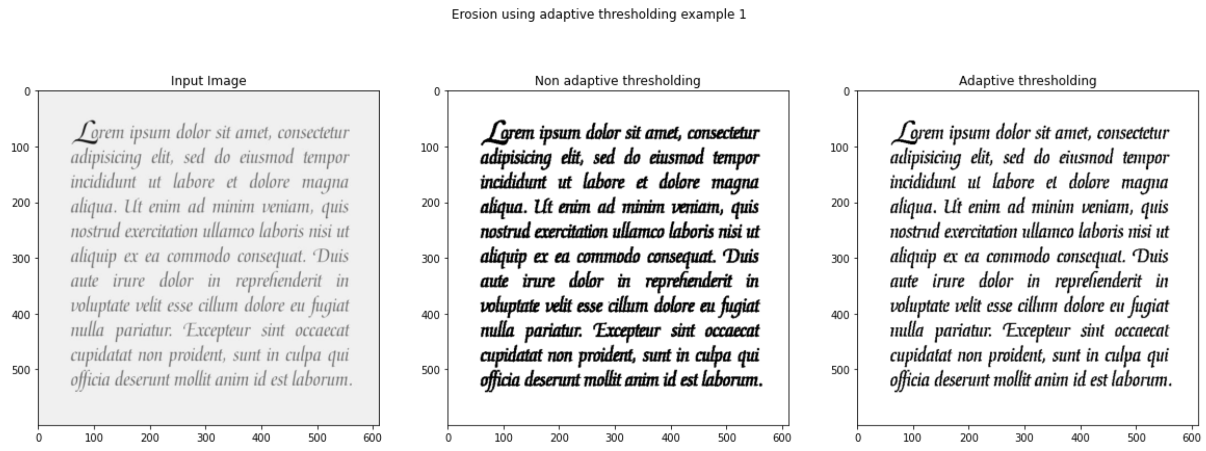


Figure: Erosion using adaptive thresholding

4.3 Dilation

Dilation is the opposite of erosion. Pixels are added to the object boundaries when dilation is used. The dilatation makes the object in white more oversized.

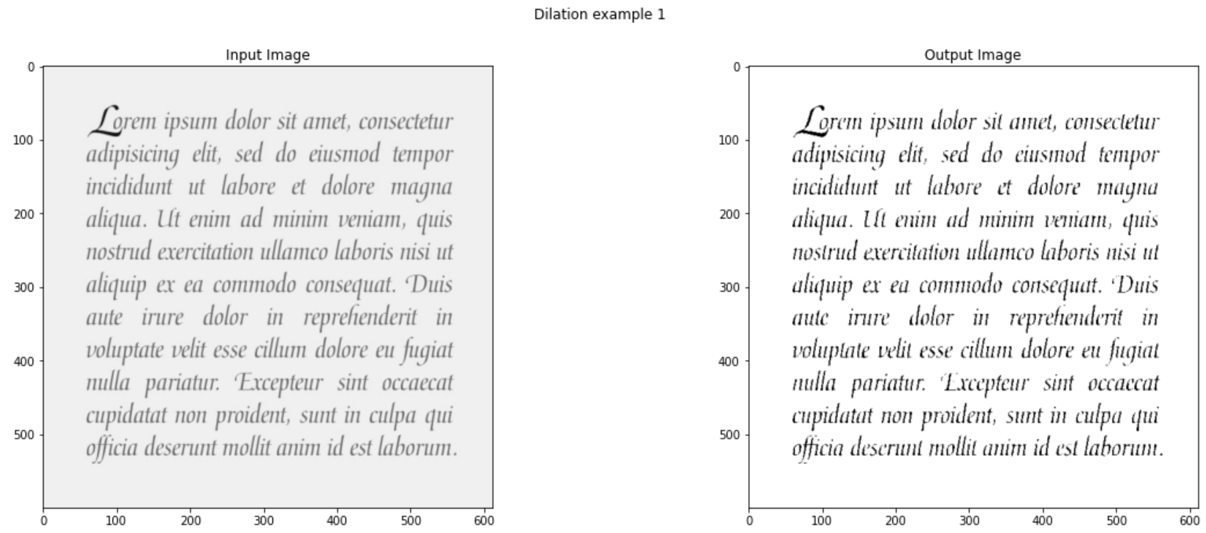


Figure: Dilation example 1

From examples 1 and 2, we can see that it The dilatation makes the object in white bigger, the size of the foreground object increases.

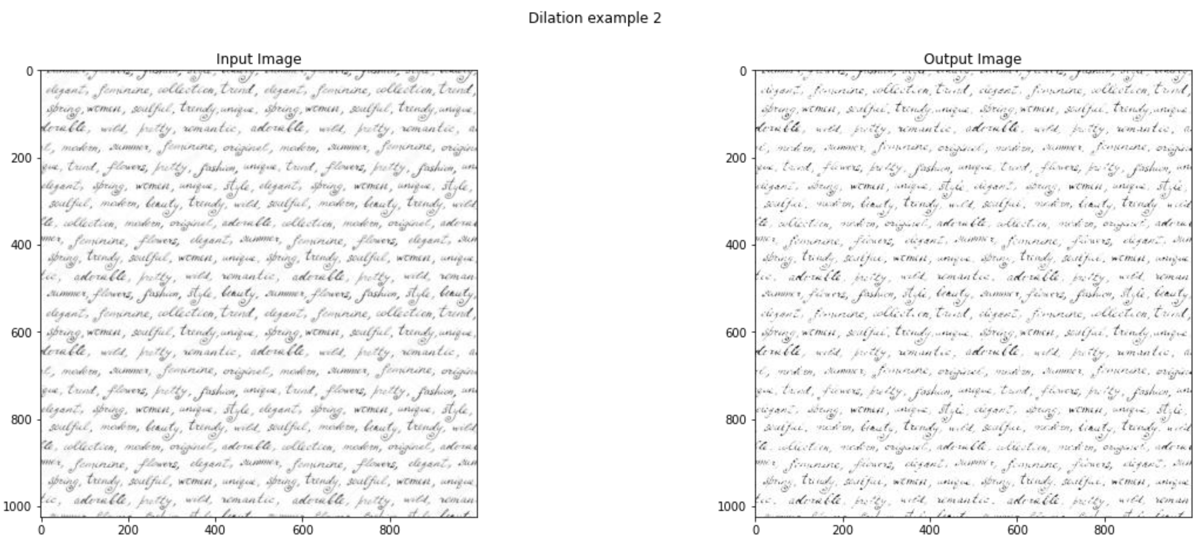
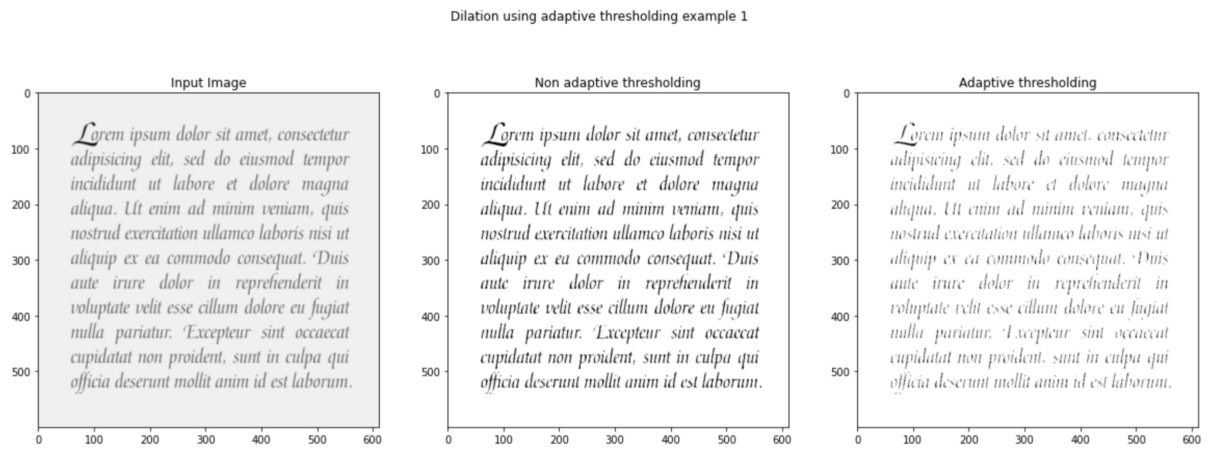


Figure: Dilation example 2

4.4 Dilation using adaptive thresholding

We have used *cv2.ADAPTIVE_THRESH_MEAN_C* for showing the comparison of adaptive and non-adaptive thresholding. We can see that the white background is more extensive than the adaptive thresholding method.


Dilation using adaptive thresholding example 1
Input Image
Non adaptive thresholding
Adaptive thresholding
0
100
200
300
400
500
600

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. 'Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. 'Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figure: Erosion using adaptive thresholding

4.5 Opening

Opening helps remove the white noise (salt noise) from the image. From examples 1 and 2, we can see that white noise has been removed by opening. In example 2 the image consists of both white (salt) and black (pepper) noise, but opening algorithm only removes the white noises.

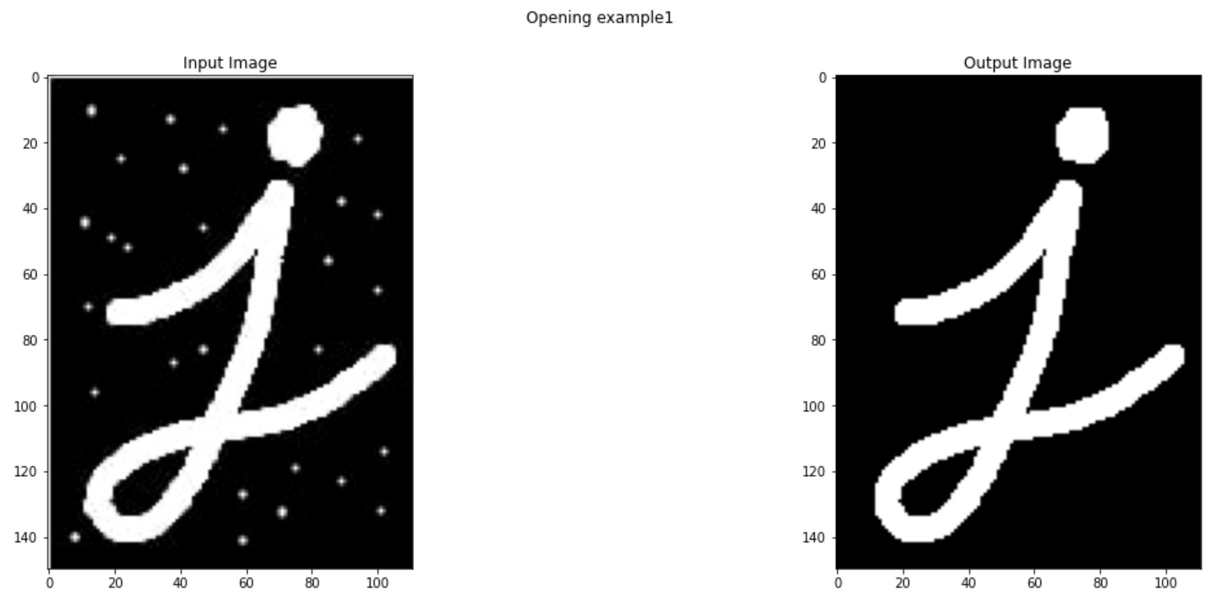


Figure: Opening example 1

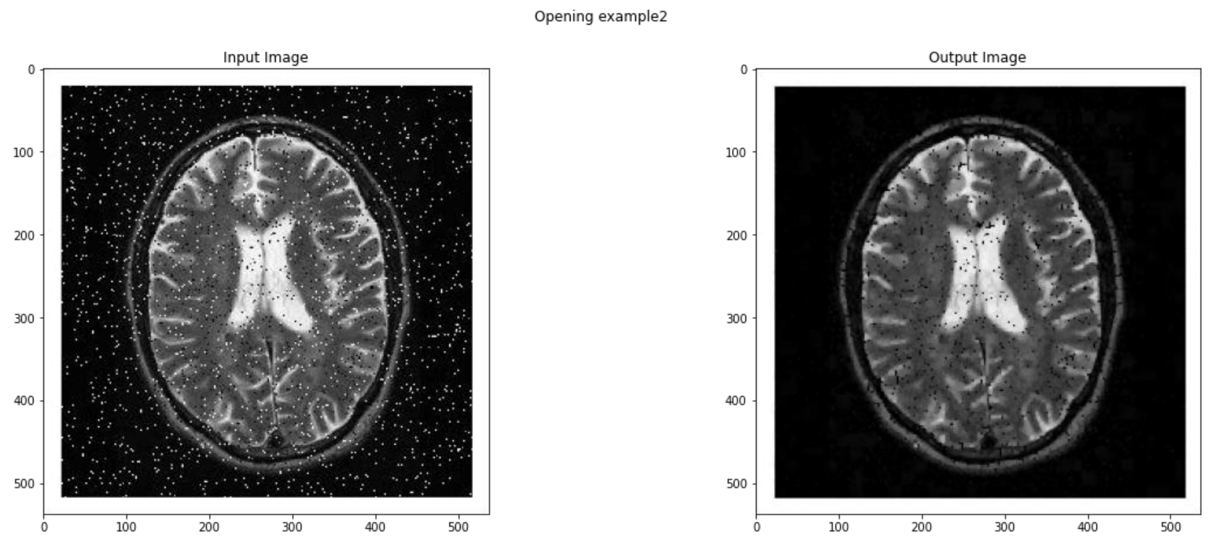


Figure: Opening example 2

4.6 Closing

Closing is reverse of opening. Closing removes black noises from the image. From examples 1 and 2, we can see that black noise has been removed by closing. In example 2 only the black noise has been removed and white noise is still present in the image.

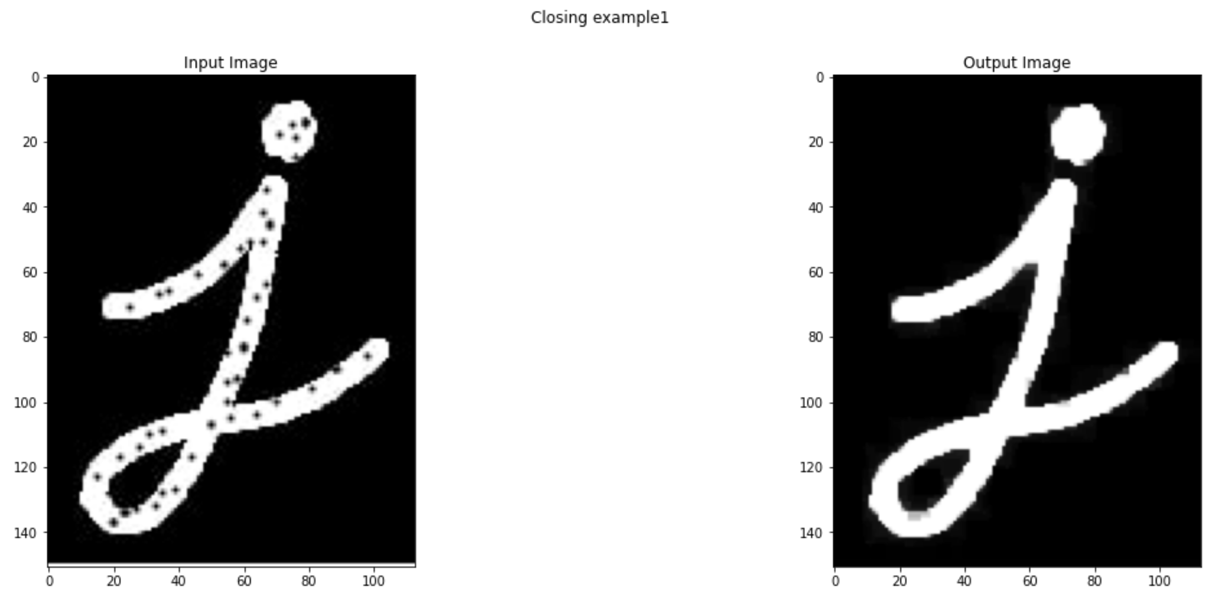


Figure: Closing example 1

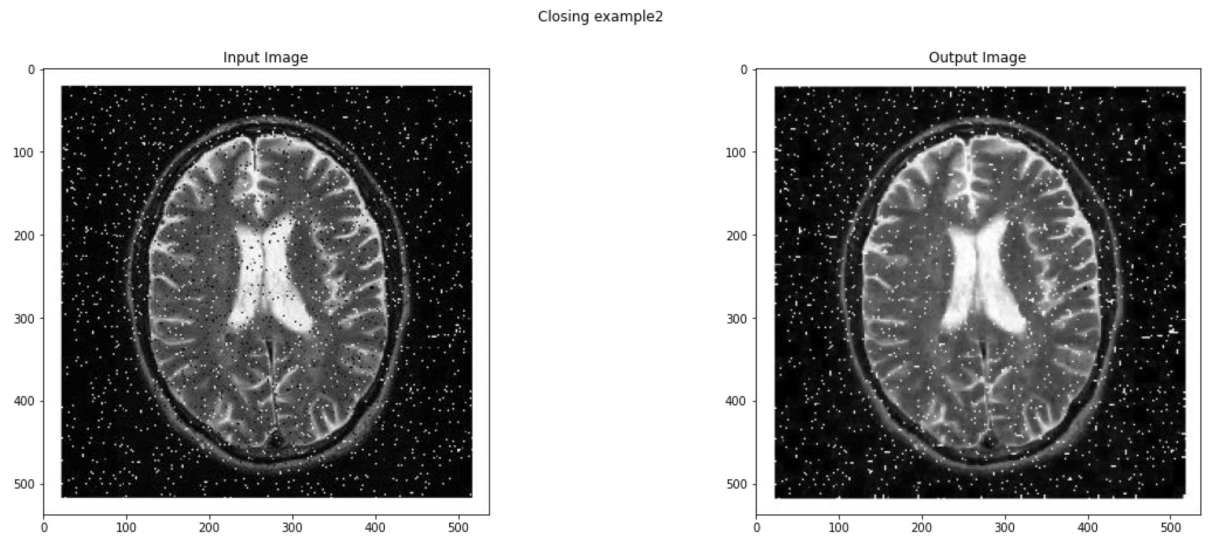


Figure: Closing example 2

4.7 Combining opening and closing

If we combine the opening and closing for the same image, we can reduce white and black noise. The final image will be much better with this composition.

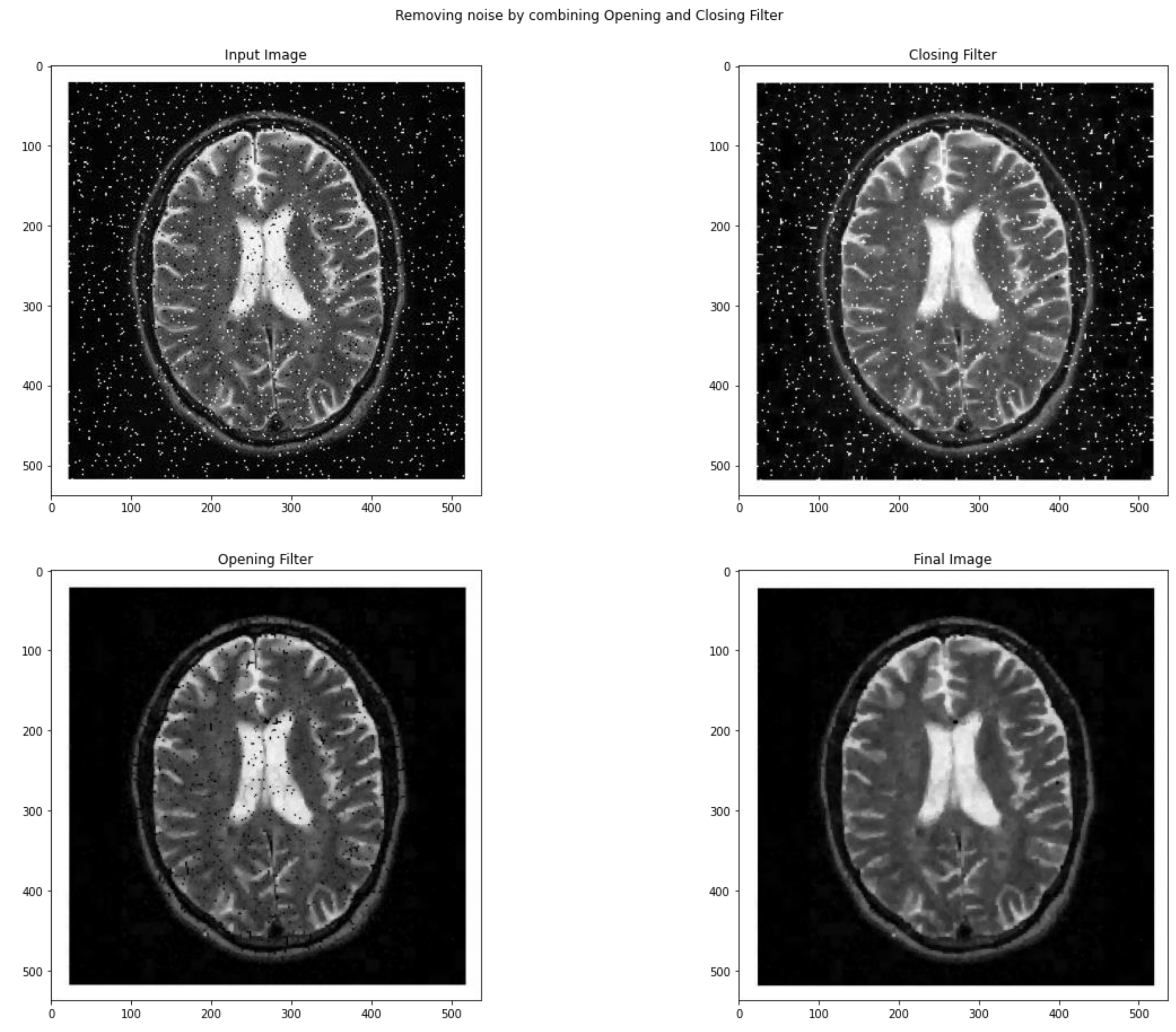


Figure: Composition of opening and closing

4.8 Top Hat

In a dark background, the top-hat filter is used to emphasize bright pixels. Our experiment shows that The Top-Hat procedure is used to enhance and remove very small details. As a result, when the inputs are present as light pixels on a dark background, it is useful for observing intricate details.

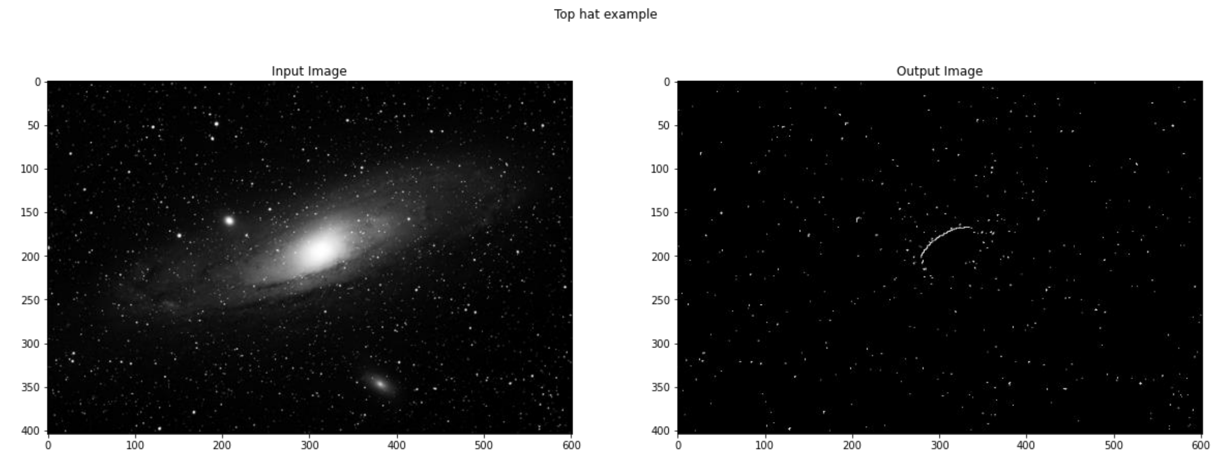


Figure: Top hat example

4.9 Black hat

In a bright background, the BlackHat transform is used to enhance dark objects of interest.

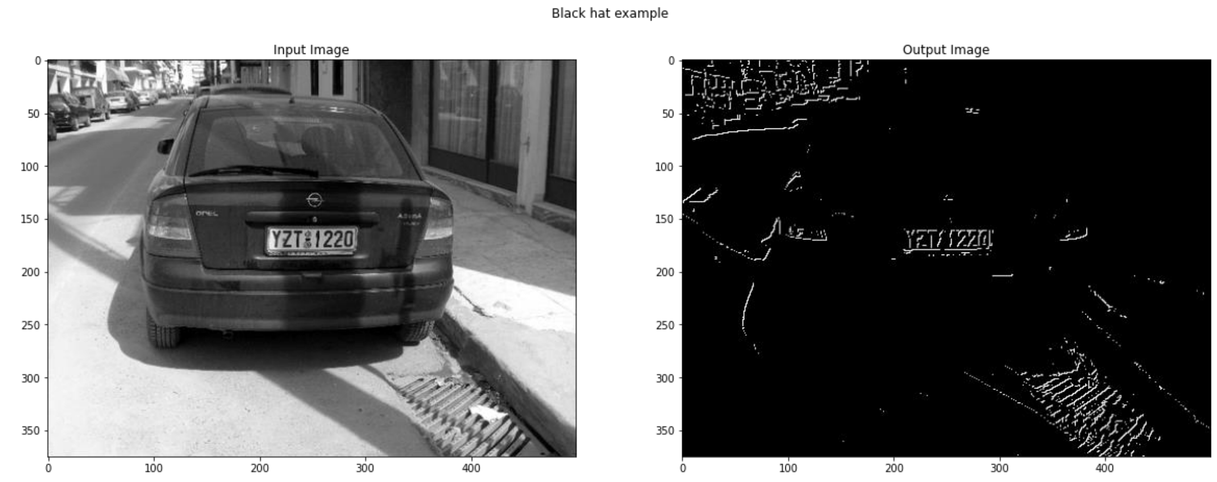


Figure: Black hat example

4.10 Morphological Gradient

The difference between dilatation and erosion is referred to as a morphological gradient. It can be used to determine the outline of a certain object in a picture. In our experiment, we have successfully extracted a vehicle numberplate outline using morphological gradient.

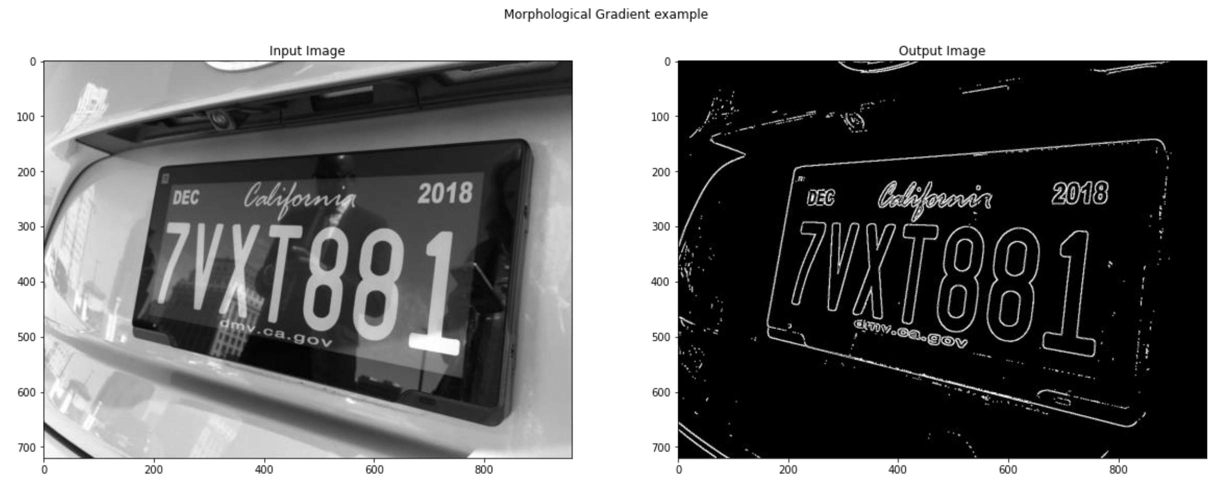


Figure: Morphological gradient example

4.11 Edge Detection with Morphology [8]

Edges are one of the most significant characteristics of images. We have done two types of edge detecting processes using Morphological Operations.

4.11.1 External Boundary Extraction

To get the external edges, we take the image's Dilation and subtract the original input image.

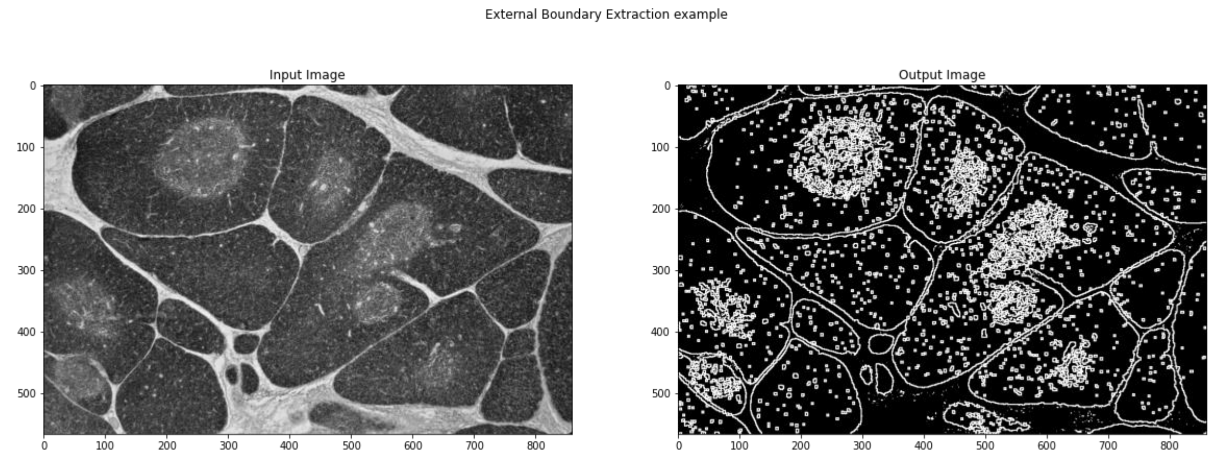


Figure: External boundary extraction example

4.11.2 Internal Boundary Extraction

To get the external edges, we take the image's erosion and subtract the original input image.

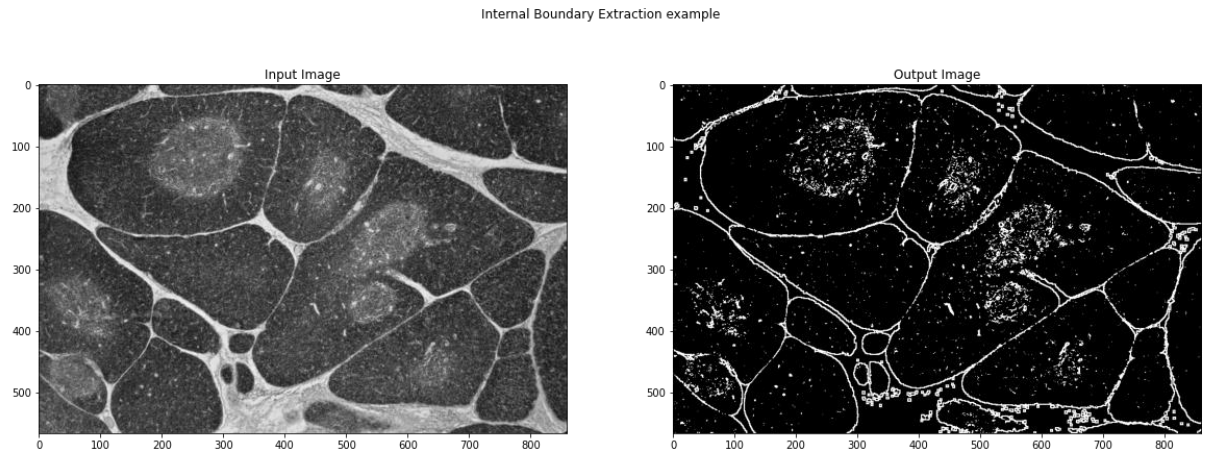


Figure: Internal boundary extraction example

5. Conclusion

In this work, we have presented morphological image processing to detect object from binary images. Here we have used structuring element after the conversion of a binary image. Moreover, morphological operations like erosion, dilation, opening, closing, top hat, bottom hat and morphological gradient have been done. Furthermore, to generate the output, we have performed 2x2 kernel on the images.

References

[1] "OpenCV Morphological Operations," *PyImageSearch*, Apr. 28, 2021. https://www.pyimagesearch.com/2021/04/28/opencv-morphological-operations/ (accessed Nov. 11, 2021).
[2] "OpenCV: Image Thresholding." https://docs.opencv.org/4.5.4/d7/d4d/tutorial_py_thresholding.html (accessed Nov. 11, 2021).
[3] "OpenCV Thresholding (cv2.threshold)," *PyImageSearch*, Apr. 28, 2021. https://www.pyimagesearch.com/2021/04/28/opencv-thresholding-cv2-threshold/ (accessed Nov. 11, 2021).
[4] N. Joram, "Morphological Operations in Image Processing," *Medium*, Jan. 01, 2020. https://himnickson.medium.com/morphological-operations-in-image-processing-cb8045b98fcc (accessed Nov. 11, 2021).
[5] "Morphological Image Processing." https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm (accessed Nov. 11, 2021).
[6] "Top Hat and Black Hat Transform using Python-OpenCV," *GeeksforGeeks*, Jun. 04, 2020. https://www.geeksforgeeks.org/top-hat-and-black-hat-transform-using-python-opencv/ (accessed Nov. 11, 2021).
[7] "OpenCV Morphological Operations," *PyImageSearch*, Apr. 28, 2021. https://www.pyimagesearch.com/2021/04/28/opencv-morphological-operations/ (accessed Nov. 11, 2021).
[8] Y. Ç. Aktaş, "A Comprehensive Guide to Image Processing: Part 3," *Medium*, Sep. 02, 2021. https://towardsdatascience.com/image-processing-part-3-dbf103622909 (accessed Nov. 11, 2021).
[9] Aktaş, Yağmur Çiğdem. "Image Processing Part 2." Medium, 2 Sept. 2021, https://towardsdatascience.com/image-processing-part-2-1fb84931364a.
[10] Sundararajan, D. Digital Image Processing. Springer Singapore, 2017. DOI.org (Crossref), https://doi.org/10.1007/978-981-10-6113-4.

Appendix

The entire code is written in jupyter notebook. Inputs and outputs have a better observation using a jupyter notebook file.

CODE LISTING

```
#%% md

# Erosion
The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object
(Always try to keep foreground in white).

#%%

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

```
#%%

# reading image from disk
img = cv2.imread('./images/lorem_ipsum.jpg', 0)

# generating histogram of loaded image
hist, bins = np.histogram(img.ravel(),256,[0,256])

# plotting histogram
plt.figure()
plt.title("Histogram of erosion example 1")
plt.plot(bins[0:-1], hist)
plt.show()

# converting image into binary image
(thresh, blackAndWhiteImage) = cv2.threshold(img, 230, 255, cv2.THRESH_BINARY)

# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
erosionExample1 = cv2.erode(blackAndWhiteImage, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Erosion example 1")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(erosionExample1, cmap = 'gray')
plt.show()

#%%

# reading image from disk
img = cv2.imread('./images/handwrite.jpg', 0)

# generating histogram of loaded image
hist, bins = np.histogram(img.ravel(),256,[0,256])

# plotting histogram
plt.figure()
plt.title("Histogram of erosion example 2")
plt.plot(bins[0:-1], hist)
plt.show()

# converting image into binary image
(thresh, blackAndWhiteImage) = cv2.threshold(img, 240, 255, cv2.THRESH_BINARY)

# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
erosionExample2 = cv2.erode(blackAndWhiteImage, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Erosion example 2")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(erosionExample2, cmap = 'gray')
plt.show()

#%% md

### Erosion using adaptive thresholding

#%%

# reading image from disk
img = cv2.imread('./images/lorem_ipsum.jpg', 0)

# converting image into binary image using non adaptive algorithm
(thresh, blackAndWhiteImage) = cv2.threshold(img, 230, 255, cv2.THRESH_BINARY)

# converting image into binary image
mask = cv2.adaptiveThreshold(img, maxValue = 255, adaptiveMethod =
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, thresholdType = cv2.THRESH_BINARY, blockSize = 11, C =
25)

# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
nonAdaptiveErosion1 = cv2.erode(blackAndWhiteImage, kernel, iterations=1)
adaptiveErosion1 = cv2.erode(mask, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Erosion using adaptive thresholding example 1")
plt.subplot(131)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(132)
plt.title("Non adaptive thresholding")
plt.imshow(nonAdaptiveErosion1, cmap = 'gray')
plt.subplot(133)
plt.title("Adaptive thresholding")
plt.imshow(adaptiveErosion1, cmap = 'gray')
plt.show()

#%%

# reading image from disk
img = cv2.imread('./images/handwrite.jpg', 0)

# converting image into binary image using non adaptive algorithm
(thresh, blackAndWhiteImage) = cv2.threshold(img, 220, 255, cv2.THRESH_BINARY)

# converting image into binary image
mask = cv2.adaptiveThreshold(img, maxValue = 255, adaptiveMethod =
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, thresholdType = cv2.THRESH_BINARY, blockSize = 11, C =
15)
```

```
# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
nonAdaptiveErosion1 = cv2.erode(blackAndWhiteImage, kernel, iterations=1)
adaptiveErosion1 = cv2.erode(mask, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Erosion using adaptive thresholding example 2")
plt.subplot(131)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(132)
plt.title("Non adaptive thresholding")
plt.imshow(nonAdaptiveErosion1, cmap = 'gray')
plt.subplot(133)
plt.title("Adaptive thresholding")
plt.imshow(adaptiveErosion1, cmap = 'gray')
plt.show()

#%% md

# Dilation
It is just opposite of erosion. Here, a pixel element is ‘1’ if atleast one pixel under the kernel is ‘1’. So it
increases the white region in the image or size of foreground object increases.

#%%

# reading image from disk
img = cv2.imread('./images/lorem_ipsum.jpg', 0)

# generating histogram of loaded image
hist, bins = np.histogram(img.ravel(),256,[0,256])

# plotting histogram
plt.figure()
plt.title("Histogram of Dilation example 1")
plt.plot(bins[0:-1], hist)
plt.show()

# converting image into binary image
(thresh, blackAndWhiteImage) = cv2.threshold(img, 230, 255, cv2.THRESH_BINARY)

# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
dilationExample1 = cv2.dilate(blackAndWhiteImage, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Dilation example 1")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(dilationExample1, cmap = 'gray')
plt.show()
```

```
#%%

# reading image from disk
img = cv2.imread('./images/handwrite.jpg', 0)

# generating histogram of loaded image
hist, bins = np.histogram(img.ravel(),256,[0,256])

# plotting histogram
plt.figure()
plt.title("Histogram of Dilation example 2")
plt.plot(bins[0:-1], hist)
plt.show()

# converting image into binary image
(thresh, blackAndWhiteImage) = cv2.threshold(img, 220, 255, cv2.THRESH_BINARY)

# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
dilationExample2 = cv2.dilate(blackAndWhiteImage, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Dilation example 2")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(dilationExample2, cmap = 'gray')
plt.show()

#%%

# reading image from disk
img = cv2.imread('./images/lorem_ipsum.jpg', 0)

# converting image into binary image using non adaptive algorithm
(thresh, blackAndWhiteImage) = cv2.threshold(img, 230, 255, cv2.THRESH_BINARY)

# converting image into binary image
mask = cv2.adaptiveThreshold(img, maxValue = 255, adaptiveMethod =
cv2.ADAPTIVE_THRESH_MEAN_C, thresholdType = cv2.THRESH_BINARY, blockSize = 11, C = 25)

# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
nonAdaptiveDilation1 = cv2.dilate(blackAndWhiteImage, kernel, iterations=1)
adaptiveDilation1 = cv2.dilate(mask, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Dilation using adaptive thresholding example 1")
plt.subplot(131)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(132)
plt.title("Non adaptive thresholding")
plt.imshow(nonAdaptiveDilation1, cmap = 'gray')
plt.subplot(133)
plt.title("Adaptive thresholding")
plt.imshow(adaptiveDilation1, cmap = 'gray')
plt.show()

#%%

# reading image from disk
img = cv2.imread('./images/handwrite.jpg', 0)

# converting image into binary image using non adaptive algorithm
(thresh, blackAndWhiteImage) = cv2.threshold(img, 220, 255, cv2.THRESH_BINARY)

# converting image into binary image
mask = cv2.adaptiveThreshold(img, maxValue = 255, adaptiveMethod =
cv2.ADAPTIVE_THRESH_MEAN_C, thresholdType = cv2.THRESH_BINARY, blockSize = 11, C = 25)

# selecting kernel
kernel = np.ones((2, 2), np.uint8)

# generating erosion image
nonAdaptiveDilation2 = cv2.dilate(blackAndWhiteImage, kernel, iterations=1)
adaptiveDilation2 = cv2.dilate(mask, kernel, iterations=1)

# plotting output
plt.figure(figsize=(20,7))
plt.suptitle("Dilation using adaptive thresholding example 2")
plt.subplot(131)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(132)
plt.title("Non adaptive thresholding")
plt.imshow(nonAdaptiveDilation2, cmap = 'gray')
plt.subplot(133)
plt.title("Adaptive thresholding")
plt.imshow(adaptiveDilation2, cmap = 'gray')
plt.show()

#%% md

## Opening
It is useful in removing noise

#%%

img = cv2.imread('./images/noise_opening.jpg', 0)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((5, 5), np.uint8)
openingExample1 = cv2.morphologyEx(blackAndWhiteImage, cv2.MORPH_OPEN, kernel)

plt.figure(figsize=(20,7))
plt.suptitle("Opening example1")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(openingExample1, cmap = 'gray')
plt.show()

img = cv2.imread('./images/noisy.jpg', 0)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((4, 4), np.uint8)
# mask = cv2.adaptiveThreshold(img, maxValue = 255, adaptiveMethod =
cv2.ADAPTIVE_THRESH_MEAN_C, thresholdType = cv2.THRESH_BINARY, blockSize = 11, C = 5)
openingExample2 = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

plt.figure(figsize=(20,7))
plt.suptitle("Opening example2")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(openingExample2, cmap = 'gray')
plt.show()

#%% md

## Closing
As the name suggests, a closing is used to close holes inside of objects or for connecting components together.

#%%

img = cv2.imread('./images/noise_closing.jpg', 0)
kernel = np.ones((4, 4), np.uint8)
closingExample1 = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

plt.figure(figsize=(20,7))
plt.suptitle("Closing example1")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(closingExample1, cmap = 'gray')
plt.show()

img = cv2.imread('./images/noisy.jpg', 0)
kernel = np.ones((4, 4), np.uint8)
closingExample2 = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

plt.figure(figsize=(20,7))
plt.suptitle("Closing example2")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(closingExample2, cmap = 'gray')
plt.show()

#%% md

## Combining Opening and Closing
```

```
#%%

img = cv2.imread('./images/noisy.jpg', 0)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((4, 4), np.uint8)
combineOpenCloseExample1 = cv2.morphologyEx(openingExample2, cv2.MORPH_CLOSE, kernel)

plt.figure(figsize=(20,7))
plt.suptitle("Removing noise by combining Opening and Closing Filter")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Closing Filter")
plt.imshow(closingExample2, cmap = 'gray')

plt.figure(figsize=(20,7))
plt.subplot(121)
plt.title("Opening Filter")
plt.imshow(openingExample2, cmap = 'gray')
plt.subplot(122)
plt.title("Final Image")
plt.imshow(combineOpenCloseExample1, cmap = 'gray')
plt.show()


#%% md

## Top Hat
A top hat (also known as a white hat) morphological operation is the difference between the original
(grayscale/single channel) input image and the opening.

#%%

img = cv2.imread('./images/galaxy.jpg', 0)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 160, 255, cv2.THRESH_BINARY)
kernel = np.ones((2, 2), np.uint8)
tophatExample1 = cv2.morphologyEx(blackAndWhiteImage, cv2.MORPH_TOPHAT, kernel)

plt.figure(figsize=(20,7))
plt.suptitle("Top hat example")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(tophatExample1, cmap = 'gray')
plt.show()

#%% md

## Black Hat
t is the difference between the closing of the input image and input image.

#%%

img = cv2.imread('./images/carbackside.jpg', 0)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 160, 255, cv2.THRESH_BINARY)
kernel = np.ones((2, 2), np.uint8)
blackhatExample1 = cv2.morphologyEx(blackAndWhiteImage, cv2.MORPH_BLACKHAT, kernel)
```

```
plt.figure(figsize=(20,7))
plt.suptitle("Black hat example")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(blackhatExample1, cmap = 'gray')
plt.show()

#%% md

## Morphological Gradient
It is the difference between dilation and erosion of an image.

#%%

img = cv2.imread('./images/gradientimage.jpg', 0)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 160, 255, cv2.THRESH_BINARY)
kernel = np.ones((2, 2), np.uint8)
mask = cv2.adaptiveThreshold(img, maxValue = 255, adaptiveMethod =
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, thresholdType = cv2.THRESH_BINARY, blockSize = 5, C = 10)
gradientExample1 = cv2.morphologyEx(mask, cv2.MORPH_GRADIENT, kernel)

plt.figure(figsize=(20,7))
plt.suptitle("Morphological Gradient example")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(gradientExample1, cmap = 'gray')
plt.show()

#%% md

# Edge Detection with Morphology

* ## External Boundary Extraction
We take the Dilation of the image and substract the original input image to obtain external edges.

#%%

img = cv2.imread('./images/leukemia_G.jpg', 0)
# img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((5, 5), np.uint8)
dilate = cv2.dilate(blackAndWhiteImage,kernel,iterations = 1)
externalBoundaryExtraction = np.subtract(dilate, blackAndWhiteImage)

plt.figure(figsize=(20,7))
plt.suptitle("External Boundary Extraction example")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(externalBoundaryExtraction, cmap = 'gray')
plt.show()
```

```
#%% md

* ## Internal Boundary Extraction
We take the Erosion of the image and substract it from the original input image to obtain internal edges.

#%%

img = cv2.imread('./images/leukemia_G.jpg', 0)
# img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
(thresh, blackAndWhiteImage) = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
kernel = np.ones((5, 5), np.uint8)
erode = cv2.erode(blackAndWhiteImage,kernel,iterations = 1)
internalBoundaryExtraction = np.subtract(erode, blackAndWhiteImage)

plt.figure(figsize=(20,7))
plt.suptitle("Internal Boundary Extraction example")
plt.subplot(121)
plt.title("Input Image")
plt.imshow(img, cmap = 'gray')
plt.subplot(122)
plt.title("Output Image")
plt.imshow(internalBoundaryExtraction, cmap = 'gray')
plt.show()

#%%
```