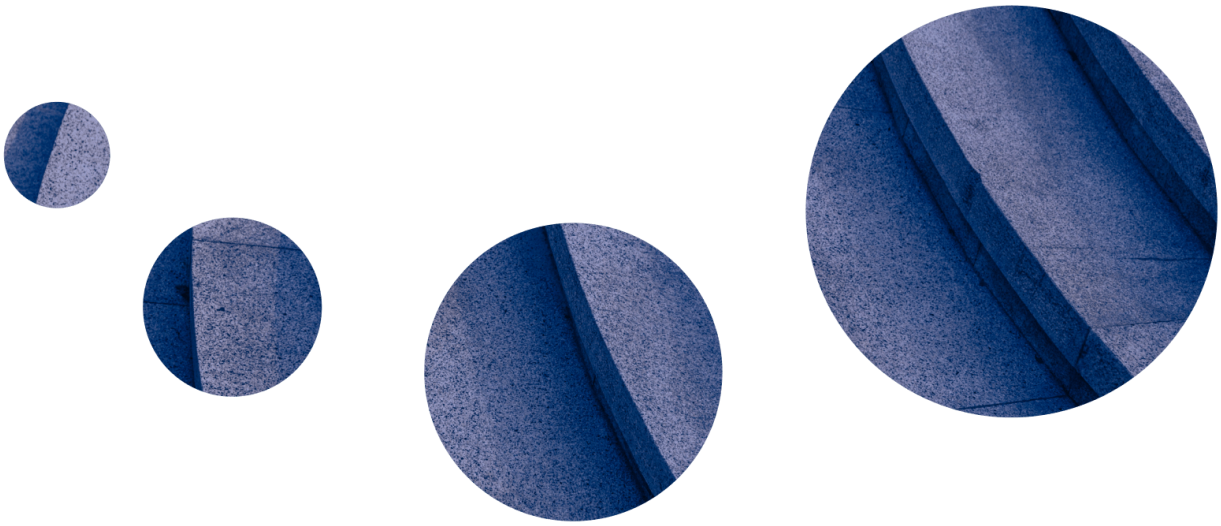


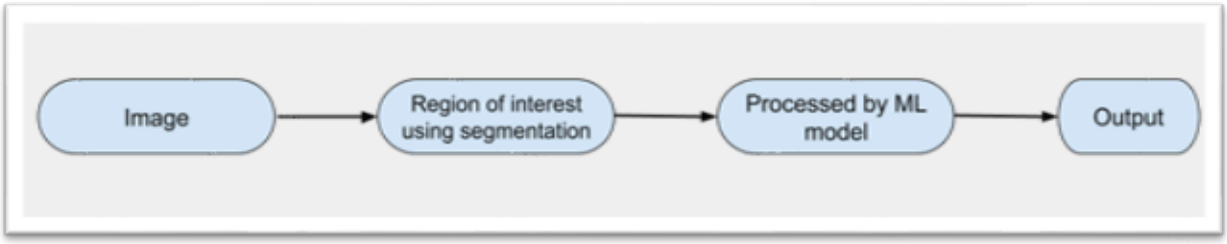
Image Segmentation

Prof. Amitava Chatterjee

By
Souvik Mandal (001810801135)
Prashant Giri (001810801161)
Saurabh Shit (001810801165)
Mrinmoy Barman()
Soumyadeep Basak(001810801132)

Image Segmentation:

Image segmentation is a method in which a digital image is broken down into various subgroups called Image segments which helps in reducing the complexity of the image to make further processing or analysis of the image simpler. Segmentation in easy words is assigning labels to pixels. All picture elements or pixels belonging to the same category have a common label assigned to them. For example: Let's take a problem where the picture has to be provided as input for object detection. Rather than processing the whole image, the detector can be inputted with a region selected by a segmentation algorithm. This will prevent the detector from processing the whole image thereby reducing inference time.



Approaches in Image Segmentation

1. **Similarity approach:** This approach is based on detecting similarity between image pixels to form a segment, based on a threshold. ML algorithms like clustering are based on this type of approach to segment an image.
2. **Discontinuity approach:** This approach relies on the discontinuity of pixel intensity values of the image. Line, Point, and Edge Detection techniques use this type of approach for obtaining intermediate segmentation results which can be later processed to obtain the final segmented image.

Image Segmentation Techniques

1. Threshold Based Segmentation
2. Region-Based Segmentation
3. Edge-Based Segmentation

4. Clustering Based Segmentation
5. Artificial Neural Network Based Segmentation

In this article, we will cover Threshold Based and Region-based Segmentation. Other segmentation techniques will be discussed in later parts.

Threshold Based Segmentation

Image thresholding segmentation is a simple form of image segmentation. It is a way to create a binary or multi-color image based on setting a threshold value on the pixel intensity of the original image. In this thresholding process, we will consider the intensity histogram of all the pixels in the image. Then we will set a threshold to divide the image into sections. For example, considering image pixels ranging from 0 to 255, we set a threshold of 60. So all the pixels with values less than or equal to 60 will be provided with a value of 0(black) and all the pixels with a value greater than 60 will be provided with a value of 255(white).

Considering an image with a background and an object, we can divide an image into regions based on the intensity of the object and the background. But this threshold has to be perfectly set to segment an image into an object and a background.

Simple Thresholding:

Here, the matter is straight-forward. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. The function **cv.threshold** is used to apply the thresholding. The first argument is the source image, which **should be a grayscale image**. The second argument is the threshold value which is used to classify the pixel values. The third argument is the maximum value which is assigned to pixel values exceeding the threshold. OpenCV provides different types of thresholding which is given by the fourth parameter of the function. Basic thresholding as described above is done by using the type **cv.THRESH_BINARY**. All simple thresholding types are:

- **cv.THRESH_BINARY**
- **cv.THRESH_BINARY_INV**
- **cv.THRESH_TRUNC**
- **cv.THRESH_TOZERO**
- **cv.THRESH_TOZERO_INV**

See the documentation of the types for the differences.
The method returns two outputs. The first is the threshold that was used and the second output is the **thresholded image**.

Adaptive Thresholding:
In the previous section, we used one global value as a threshold. But this might not be good in all cases, e.g. if an image has different lighting conditions in different areas. In that case, adaptive thresholding can help. Here, the algorithm determines the threshold for a pixel based on a small region around it. So we get different thresholds for different regions of the same image which gives better results for images with varying illumination.

In addition to the parameters described above, the method **cv.adaptiveThreshold** takes three input parameters:
The **adaptiveMethod** decides how the threshold value is calculated:

- **cv.ADAPTIVE_THRESH_MEAN_C**: The threshold value is the mean of the neighbourhood area minus the constant **C**.
- **cv.ADAPTIVE_THRESH_GAUSSIAN_C**: The threshold value is a gaussian-weighted sum of the neighbourhood values minus the constant **C**.

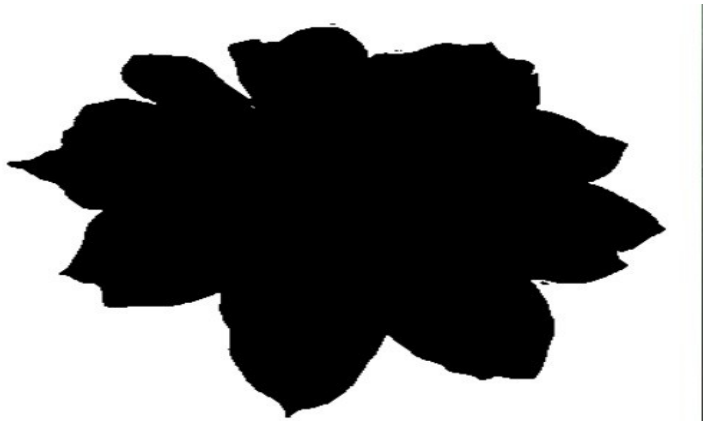
The **blockSize** determines the size of the neighbourhood area and **C** is a constant that is subtracted from the mean or weighted sum of the neighbourhood pixels.

Code of threshold based segmentation:

```python
import cv2 as cv
import numpy as np
from skimage.metrics import structural_similarity as compare_ssim

img = cv.imread("C:\\Users\\jayan\\OneDrive\\Desktop\\project\\test_noise.png",  0)
_, th1 = cv.threshold(img, 120, 200, cv.THRESH_BINARY)
_, th2 = cv.threshold(img, 120, 200, cv.THRESH_BINARY_INV)
_, th3 = cv.threshold(img, 120, 200, cv.THRESH_TRUNC)
_, th4 = cv.threshold(img, 120, 200, cv.THRESH_TOZERO)


print("No of Pixels: ", img.size)

print("--------------------img and th1--------------------")

Y = np.square(np.subtract(img, th1)).mean()
print("MSE:", Y)

(score, diff) = compare_ssim(img, th1, full=True)
diff = (diff * 255).astype("uint8")
print("SSIM: {}".format(score))

print("--------------------th1 and th2--------------------")

Y = np.square(np.subtract(th1, th2)).mean()
print("MSE:", Y)

(score, diff) = compare_ssim(th1, th2, full=True)
diff = (diff * 255).astype("uint8")
print("SSIM: {}".format(score))

print("--------------------th2 and th3--------------------")

Y1 = np.square(np.subtract(th2, th3)).mean()
print("MSE:", Y1)

(score1, diff1) = compare_ssim(th2, th3, full=True)
diff1 = (diff1 * 255).astype("uint8")
print("SSIM: {}".format(score1))

cv.imshow("Image", img)
cv.imshow("th1", th1)
cv.imshow("th2", th2)
cv.imshow("th3", th3)
cv.imshow("th4", th4)

cv.waitKey(0)
```

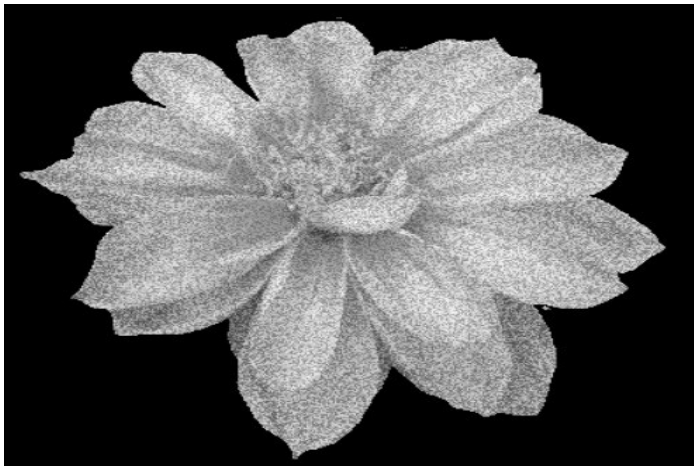
Original Image:

Result Image after Threshold based segmentation:



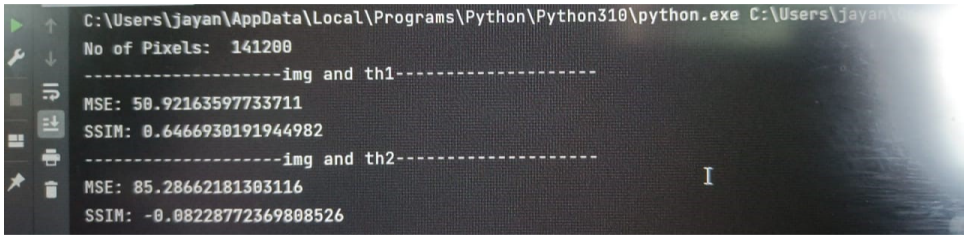
With Noise

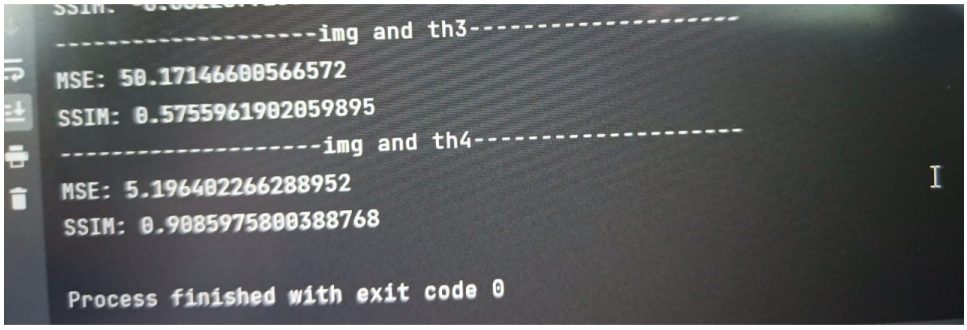
Without Noise



With Noise Original

Result of MSE and SSIM:





Img and th1		Img and th2		Img and th3		Img and th4	
MSE	SSIM	MSE	SSIM	MSE	SSIM	MSE	SSIM
50.921	0.647	85.286	-0.8228	50.171	0.5755	5.196	0.9085

Region-Based Segmentation

In this segmentation, we grow regions by recursively including the neighboring pixels that are similar and connected to the seed pixel. We use similarity measures such as

differences in gray levels for regions with homogeneous gray levels. We use connectivity to prevent connecting different parts of the image.
There are two variants of region-based segmentation:

- **Top-down approach**
  - First, we need to define the predefined seed pixel. Either we can define all pixels as seed pixels or randomly chosen pixels. Grow regions until all pixels in the image belongs to the region.
- **Bottom-Up approach**
  - Select seed only from objects of interest. Grow regions only if the similarity criterion is fulfilled.
- **Similarity Measures:**
  - Similarity measures can be of different types: For the grayscale image the similarity measure can be the different textures and other spatial properties, intensity difference within a region or the distance b/w mean value of the region.
- **Region merging techniques:**
  - In the region merging technique, we try to combine the regions that contain the single object and separate it from the background.. There are many regions merging techniques such as Watershed algorithm, Split and merge algorithm, etc**.**
- **Pros:**
  - Since it performs simple threshold calculation, it is faster to perform.
  - Region-based segmentation works better when the object and background have high contrast.
- **Limitations:**
  - It did not produce many accurate segmentation results when there are no significant differences b/w pixel values of the object and the background.

Implementation:

- In this implementation, we will be performing edge and region-based segmentation. We will be using scikit image module for that and an image from its dataset provided.

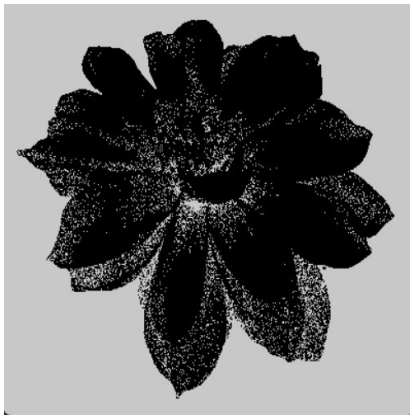
Code on Region based image segmentation:

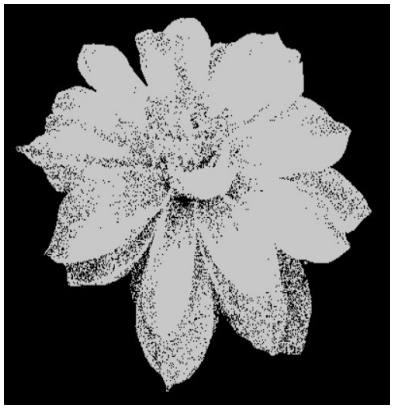
```python
import numpy as np
import cv2
from skimage.metrics import structural_similarity as compare_ssim


class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def getX(self):
        return self.x

    def getY(self):
        return self.y


def getGrayDiff(img, currentPoint, tmpPoint):
    return abs(int(img[currentPoint.x, currentPoint.y]) - int(img[tmpPoint.x, tmpPoint.y]))


def selectConnects(p):
    if p != 0:
        connects = [Point(-1, -1), Point(0, -1), Point(1, -1), Point(1, 0), Point(1, 1),
                    Point(0, 1), Point(-1, 1), Point(-1, 0)]
    else:
        connects = [Point(0, -1), Point(1, 0), Point(0, 1), Point(-1, 0)]
    return connects


def regionGrow(img, seeds, thresh, p=1):
    height, weight = img.shape
    seedMark = np.zeros(img.shape)
    seedList = []
    for seed in seeds:
        seedList.append(seed)
    label = 1
    connects = selectConnects(p)
    while (len(seedList) > 0):
        currentPoint = seedList.pop(0)

        seedMark[currentPoint.x, currentPoint.y] = label
        for i in range(8):
            tmpX = currentPoint.x + connects[i].x
            tmpY = currentPoint.y + connects[i].y
            if tmpX < 0 or tmpY < 0 or tmpX >= height or tmpY >= weight:
                continue
```

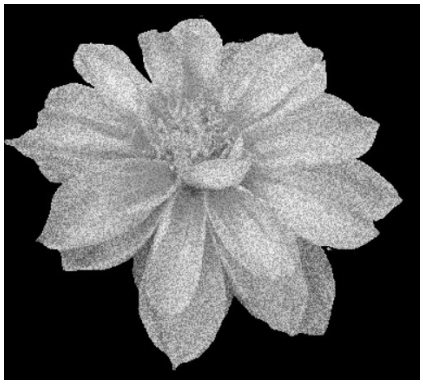
```python
    while (len(seedList) > 0):
        currentPoint = seedList.pop(0)

        seedMark[currentPoint.x, currentPoint.y] = label
        for i in range(8):
            tmpX = currentPoint.x + connects[i].x
            tmpY = currentPoint.y + connects[i].y
            if tmpX < 0 or tmpY < 0 or tmpX >= height or tmpY >= weight:
                continue
            grayDiff = getGrayDiff(img, currentPoint, Point(tmpX, tmpY))
            if grayDiff < thresh and seedMark[tmpX, tmpY] == 0:
                seedMark[tmpX, tmpY] = label
                seedList.append(Point(tmpX, tmpY))
    return seedMark


img = cv2.imread("C:\\Users\\jayan\\OneDrive\\Desktop\\project\\test.png", 0)
cv2.imshow("Without Noise original", img)
seeds = [Point(10, 10), Point(82, 150), Point(20, 300)]
binaryImg = regionGrow(img, seeds, 10)
cv2.imshow("Without Noise", binaryImg)

img1 = cv2.imread("C:\\Users\\jayan\\OneDrive\\Desktop\\project\\test_noise.png", 0)
cv2.imshow("With Noise original", img1)
binaryImg1 = regionGrow(img1, seeds, 10)
cv2.imshow("With Noise", binaryImg1)

print("--------------------without noise image and region growing--------------------")

Y1 = np.square(np.subtract(img, binaryImg)).mean()
print("MSE:", Y1)

(score1, diff1) = compare_ssim(img, binaryImg, full=True)
diff1 = (diff1 * 255).astype("uint8")
print("SSIM: {}".format(score1))

print("--------------------with noise image and region growing--------------------")

Y2 = np.square(np.subtract(img1, binaryImg1)).mean()
print("MSE:", Y2)

(score2, diff2) = compare_ssim(img1, binaryImg1, full=True)
diff2 = (diff2 * 255).astype("uint8")
print("SSIM: {}".format(score2))

cv2.waitKey(0)
```

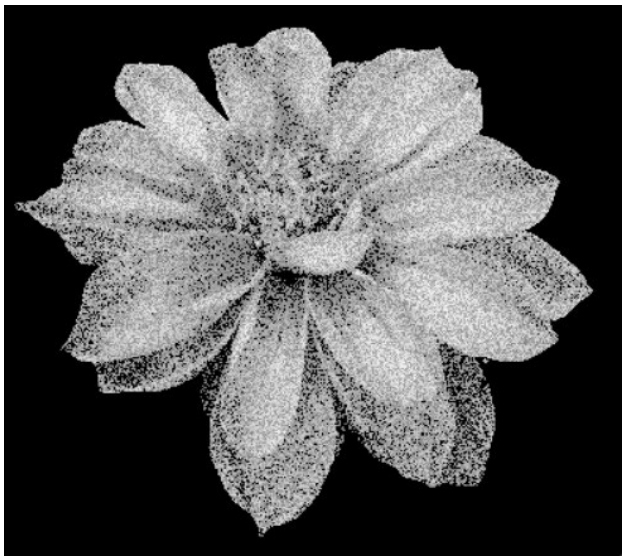
Original Image:

After Region based segmentation:

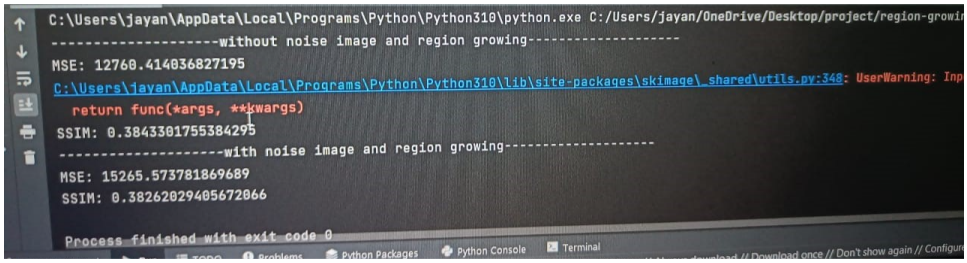








Result of MSE and SSIM:



Without noise image and region growing		With noise image and region growing	
MSE	SSIM	MSE	SSIM
12768.4140	0.3843301	15265.5737	0.38262029