Computational Imaging Tools: Concepts and Python Codes.

May 12, 2017

0.1 Background: Computational Bioimaging.

1. **What is Computational Imaging?**

   Answer: 'Biology is a highly data driven discipline and one of the primary sources of data in Biology is **imaging**.
   There are various sources of image acquisition:

   - Various forms of microscopy (Optical and electron).
   - Magnetic Resonance Imaging.
   - Other mechanisms for image acquisition.

   After acquisition images are processed to generate data. i.e. the process to generate data is Image acquisition $\longrightarrow$ Image processing $\longrightarrow$ Data.
   This can be

   - Sometimes simply finding a suitable image demonstrating some property of interest.
   - And Other times this processing can be 'complex pipeline' designed to extract quantitative data[1].'

2. **What is digital image?**

- 'A natural image captured with a camera, telescope, microscope, or other type of optical instrument displays a continuously varying array of shades and color tones.
- Such images contain a wide spectrum of intensities, ranging from dark to light, and a spectrum of colors that can include just about any imaginable hue and saturation level. Images of this type are referred to as **continuous-tone** because the various tonal shades and hues blend together without disruption to generate a faithful reproduction of the original scene.
- Continuous-tone images are produced by analog optical and electronic devices, which accurately record image data by several methods, such as a sequence of electrical signal fluctuations or changes in the chemical nature of a film emulsion that vary continuously over all dimensions of the image.
- In order for a continuous-tone or analog image to be processed or displayed by a computer, it must first be converted into a computer-readable form or digital format.
- To convert a continuous-tone image into a digital format, the analog image is divided into individual brightness values through two operational processes that are termed *sampling* and *quantization*.
    1. **Sampling:**The sampling process measures the intensity at successive locations in the image and forms a two-dimensional array containing small rectangular blocks of intensity information.
    2. **Quantization:** After sampling in a two-dimensional array, the resulting data is quantized to assign a specific digital brightness value to each sampled data point, ranging from black, through all of the intermediate gray levels, to white. The result is a numerical representation of the intensity, which is commonly referred to as a **picture element or pixel**, for each sampled data point in the array.
- Because images are generally square or rectangular in dimension, each pixel that results from image digitization is represented by a coordinate-pair with specific x and y values arranged in a typical Cartesian coordinate system.
  The x coordinate specifies the horizontal position or column location of the pixel, while the y coordinate indicates the row number or vertical position. By convention, the pixel positioned at coordinates (0,0) is located in the upper left-hand corner of the array.
  Thus, a digital image is composed of a rectangular (or square) pixel array. It has a definite height and a definite width counted in pixels representing a series of intensity values[17].'

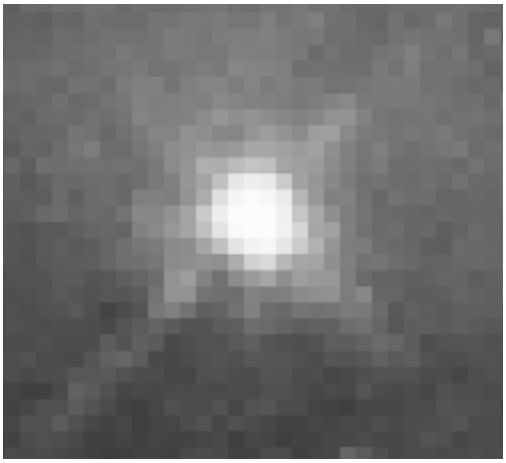


Figure 1: An image — an array or a matrix of pixels arranged in columns and rows[2].

- 'In a (8-bit) greyscale image each picture element has an assigned intensity that ranges from 0 to 255. A greyscale image includes many shades of grey.
  A normal greyscale image has 8 bit colour depth = 2^8 = 256 greyscales.
  A "true colour" image has 24 bit colour depth = 8 x 8 x 8 bits = 256 x 256 x 256 colours = 16 million colours.
  Some greyscale images have more greyscales, for instance 16 bit = 65536 greyscales.

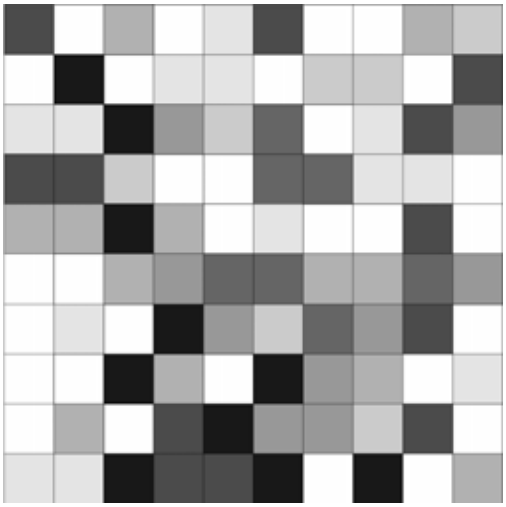


Figure 2: Each pixel has a value from 0 (black) to 255 (white). The possible range of the pixel values depend on the colour depth of the image, here 8 bit = 256 tones or greyscales[2].

In principle three greyscale images can be combined to form an image with 281,474,976,710,656 greyscales[2].'

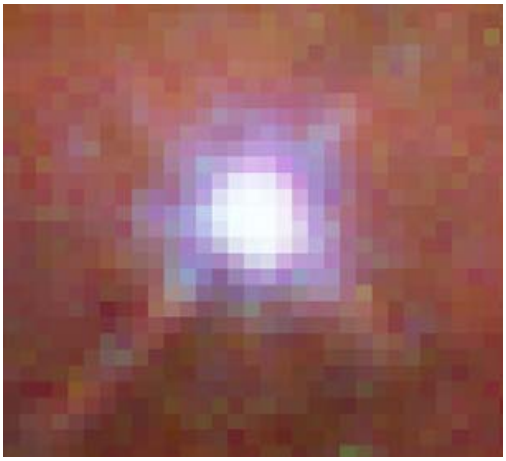


Figure 3: A true-colour image assembled from three greyscale images coloured red, green and blue. Such an image may contain up to 16 million different colours[2].

3. **Basic properties of digital images.**

    1. **Aspect Ratio:**
        - Aspect ratio is the horizontal-to-vertical dimensional ratio of a digital image. It can be calculated by dividing the horizontal width by the vertical height.
        - When a continuous-tone image is sampled and quantized, the pixel dimensions of the resulting digital image acquire the aspect ratio of the original analog image. In this regard, it is important that each individual pixel has a 1:1 aspect ratio (referred to as square pixels) to ensure compatibility with common digital image processing algorithms and to minimize distortion. If the analog image has a 4:3 aspect ratio, more samples must be taken in the horizontal direction than in the vertical direction (4 horizontal samples for each 3 vertical samples).
    2. **Spatial Resolution:**
        - Image resolution, is the quality of a digital image determined by the number of pixels and the range of brightness values available for each pixel utilized in the image. Resolution of the image is regarded as the capability of the digital image to reproduce fine details that were present in the original analog image or scene.
        - **Spatial resolution** is the number of pixels utilized in constructing and rendering a digital image. This quantity is dependent upon how finely the image is sampled during acquisition or digitization. Higher spatial resolution images have a greater number of pixels within the same physical dimensions. Thus, as the number of pixels acquired during sampling and quantization of a digital image increases, the spatial resolution of the image also increases.
    3. **Intensity :**
        - Intensity is the magnitude or quantity of light energy actually reflected from or transmitted through the object being imaged by an analog or digital device[17].'
        - 'Intensity of a pixel is defined as the value of the pixel. For example in an 8 bit grayscale image there are 256 gray levels. Now any pixel in the image can have a value from 0 to 255 and that will be its intensity. In general, higher the intensity the brighter is the pixel[18].'

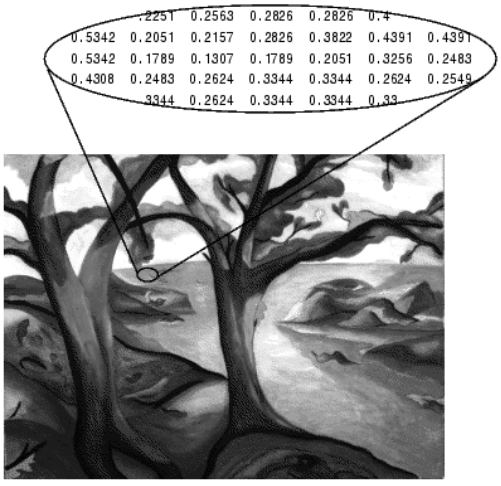


Figure 4: Pixel Values in an Intensity Image Define Gray Levels[1].

4. **Image brightness:**
    - 'The brightness of a digital image is a measure of relative intensity values across the pixel array after the image has been acquired with a digital camera or digitized by an analog-to-digital converter.
    - In terms of digital image processing, brightness is more properly described as the measured intensity of all the pixels comprising an ensemble that constitutes the digital image after it has been captured, digitized, and displayed.
    - Pixel brightness is an important factor in digital images, because (other than color) it is the only variable that can be utilized by processing techniques to quantitatively adjust the image.

5. **Bit depth:**
    - The bit depth refers to the binary range of possible grayscale values utilized by the analog-to-digital converter to translate analog image information into discrete digital values capable of being read and analyzed by a computer.
    - The bit depth of the analog-to-digital converter determines the size of the gray scale increments, with higher bit depths corresponding to a greater range of useful image information available from the camera.

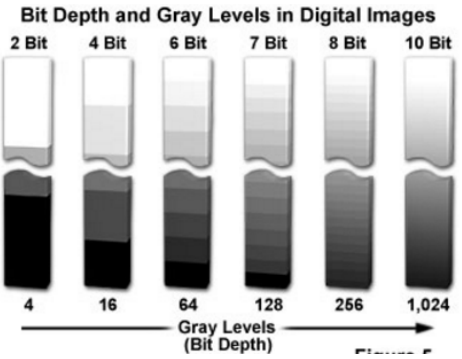


Figure 5: Bit depth and Gray levels in digital images[18].

    - The accuracy of the digital value is directly proportional to the bit depth of the digitizing device. If two bits are utilized, the image can only be represented by four brightness values or levels. Likewise, if three or four bits are processed, the corresponding images have eight and 16 brightness levels, respectively.
    - A higher number of gray levels corresponds to greater bit depth and the ability to accurately represent a greater signal dynamic range(The maximum signal level with respect to noise that the CCD sensor can transfer for image display).

6. **Contrast:**Contrast is the difference in luminance or colour that makes an object (or its representation in an image or display) distinguishable. In visual perception of the real world, contrast is determined by the difference in the color and brightness of the object and other objects within the same field of view(Ref reqd).

7. **Texture:**An image texture is a set of metrics calculated in image processing designed to quantify the perceived texture of an image. Image texture gives us information about the

spatial arrangement of color or intensities in an image or selected region of an image(Ref reqd).

Figure 6: Texture[2].

8. **Image Histogram:**
   - The intensity or brightness of the pixels comprising a digital image can be graphically depicted in a grayscale histogram, which maps the number of pixels at each gray level present in the image.
   - The histogram provides a convenient representation of a digital image by indicating the relative population of pixels at each brightness level and the overall intensity distribution of the image in general.
   - Statistics derived from the histogram can be employed to compare contrast and intensity between images, or the histogram can be altered by image processing algorithms to produce corresponding changes in the image. In addition, the number of pixels in the histogram can be employed to ascertain area measurements of specific image details, or to evaluate and compare the performance of a video camera or digitizer.
   - One of the most popular and practical uses for a digital image histogram is adjustment of contrast. The grayscale histogram also reveals the extent of available gray-level range that is being utilized by a digital image.
9. **Digital image storage :**
   - Images are digitized in sequential order by raster scanning or array readout of either an analog or optical image by the digitizing device (CCD, scanner, etc.), which transfers data to the computer in a serial string of pixel brightness values. The image is then displayed by incremental counting of pixels, according to the established image vertical and horizontal dimensions, which are usually recorded in the image file header.
   - The characteristics of digital images can be expressed in several manners. For example,

(a) By specifying number of pixels in a given length dimension (such as pixels per inch).

(b) The pixel array size (for example, 640 x 480) can be used to describe the image.

(c) The total number of image pixels or the computer storage file size that indicates the image dimensions.

- **File sizes**, in bytes, can be determined by *multiplying the pixel dimensions by the bit depth and dividing that number by 8*, the number of bits per byte. For example, a 640 x 480 (pixel) image having 8-bit resolution would translate into 302 kilobytes of computer memory (see Table 1 below). Likewise a high-resolution 1280 x 1024 true color image with 24-bit depth requires over 3.8 megabytes of storage space[17].'

Digital File Format Memory Requirements

Pixel Dimensions	Grayscale (8-Bit)	Bitmap (24-Bit)	JPEG (24-Bit)	TIFF (24-Bit)
16 x 16	2k	2k	2k	2k
64 x 64	6k	13k	5k	13k
128 x 128	18k	49k	12k	49k
256 x 256	66k	193k	22k	193k
320 x 240	77k	226k	24k	226k
512 x 512	258k	769k	52k	770k
640 x 480	302k	901k	56k	902k
800 x 600	470k	1,407k	75k	1,408k
1024 x 768	770k	2,305k	104k	2,306k
1280 x 1024	1,282k	3,841k	147k	3,842k
1600 x 1200	1,877k	5,626k	161k	5,627k
2250 x 1800	3,960k	11,869k	276k	11,867k
3200 x 2560	8,002k	24,001k	458k	24,002k
3840 x 3072	11,522k	34,561k	611k	34,562k

Figure 7: Digital File formats memory requirements[17].

- 'There are two general groups of 'images': vector graphics (or line art) and bitmaps (pixel-based or 'images'). Some of the most common file formats are:

(a) GIF — an 8-bit (256 colour), non-destructively compressed bitmap format. Mostly used for web. Has several sub-standards one of which is the animated GIF.

(b) JPEG — a very efficient (i.e. much information per byte) destructively compressed 24 bit (16 million colours) bitmap format. Widely used, especially for web and Internet (bandwidth-limited).

(c) TIFF — the standard 24 bit publication bitmap format. Compresses non-destructively with, for instance, Lempel-Ziv-Welch (LZW) compression.

(d) PS — Postscript, a standard vector format. Has numerous sub-standards and can be difficult to transport across platforms and operating systems.

(e) PSD – a dedicated Photoshop format that keeps all the information in an image including all the layers[2].'

0.2 Computational Imaging Tools : Image processing techniques using Python.

Computational Imaging Tools Tasks:

1. Make class 'ImageMaker' in python : This includes different functions to create various shapes using numpy arrays. Also, a function 'my_imshow' to plot the numpy array images. etc.
2. Create LPF, HPF, BPF, and Weiner filter.
3. Drift correction or Image registration.
4. Thresholding, Binarization and Morphological Tools.
5. Segmentation (shape based, size based, edge based, texture)

4. **Image processing using Python:**

- Python is a clear and concise language with good support for input/output, numerics, images and plotting.Python has various libraries to support image processing[12]. Most useful libraries for this tutorial are:

    1. **Numpy:** Numpy is python's one of core scientific modules. It provides a high-performance multidimensional array object (for representing vectors, matrices, images and much more), linear algebra functions and tools for working with these arrays. The array object are used to do important operations such as matrix multiplication, transposition, solving equation systems, vector multiplication, and normalization, which are needed to do things like aligning images, warping images, modeling variations, classifying images, grouping images etc[12].

    2. **scikit-image:** It is a Python package containing collection of algorithms dedicated to image processing. It uses NumPy arrays as image objects[13]. This library includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection in images[14].

    3. **Matplotlib:** Matplotlib is a graphics library which is used for displaying frames from videos and images[12].

    4. **PIL(Python Imaging library):** PIL is one of core libraries for image manipulation. This python library supports opening, manipulating and saving the images in many file formats. It supports various image manipulations like filtering, enhancing, masking, handling transparency, additions and the like[14].

5. **Image processing in ImageJ:**

- 'ImageJ is a public domain Java image processing and analysis program developed on Mac OS X using its built in editor and Java compiler, BBEdit editor and the Ant build tool. It runs, either as an online applet or as a downloadable application, on any computer with a Java 1.5 or later virtual machine.
- It can display, edit, analyze, process, save and print 8–bit, 16–bit and 32–bit images.
- It can read many image formats including TIFF, GIF, JPEG, BMP, DICOM, FITS and 'raw'. It supports 'stacks' (and hyperstacks), a series of images that share a single window.
- It is multithreaded, so time-consuming operations such as image file reading can be performed in parallel with other operations.
- It can
  1. Calculate area and pixel value statistics of user-defined selections.
  2. Measure distances and angles.
  3. Create density histograms and line profile plots.
  4. It supports standard image processing functions such as contrast manipulation, sharpening, smoothing, edge detection and median filtering.
  5. It does geometric transformations such as scaling, rotation and flips.
- ImageJ was designed with an open architecture that provides extensibility via Java plugins. Custom acquisition, analysis and processing plugins can be developed using ImageJ's built in editor and Java compiler. User-written plugins make it possible to solve almost any image processing or analysis problem[15].'

6. Task 1 : Using python numpy arrays create an image with black background and following shapes in the foreground-circle, rectangle, square and triangle.
   Hint:

   1. Import required libraries.
   2. Define a function for each shape.
   3. For each function, define an array of random numbers for background.
   4. Define 'kernel' specifying a particular shape.
   5. Take Fourier transform of kernel and background array.
   6. Perform convolution on fourier transform of both arrays.
   7. Take inverse fourier transform of the convolution of fourier transform of both arrays.
   8. Define functions for other mentioned shapes with 'kernel' specifying that particular shape.
   9. plot this image. (You can define a separate function to plot/ show images.)
   10. Plot an image with mixed shapes on black background.

   Expected output:

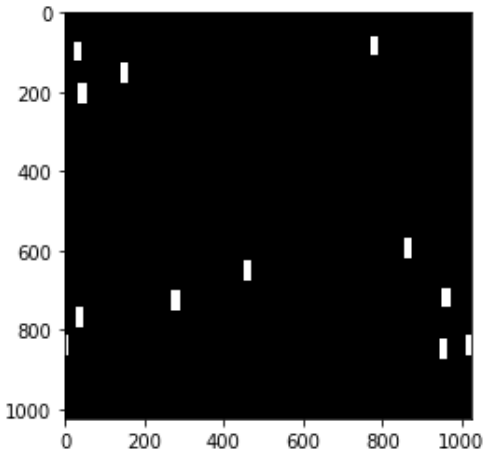


Figure 8: Numpy array based image with rectangle shapes with black background.

Answer key :**Python code: Image_maker.py (numpy arrays to make different shapes.)**[9].

Task 2 : Write a python class to create different geometric shapes using numpy array.

Hint: To learn about python class and examples,
read https://docs.python.org/2/tutorial/classes.html and
https://www.learnpython.org/en/Classes_and_Objects.

Answer key: **Python code: Class_img_maker.py**[9].

7. **What is the purpose of filtering in image processing?**
   Answer: Filtering is a technique for modifying or enhancing an image. For example, to emphasize certain features or remove unwanted features. Filtering in image processing is implemented with **Low pass filter**, **High pass filter**, **Band pass filter** and **Wiener filter**.
   Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

   - **Smoothing:** Smoothing is often used to reduce noise within an image or to produce a less pixelated image. Most smoothing methods are based on low pass filters. Smoothing is also usually based on a single value representing the image such as the average value the median value of the image(Ref reqd).

   - **Sharpening:** Image sharpening is a powerful tool for emphasizing texture and drawing viewer focus. Digital camera sensors and lenses always blur an image to some degree, and this requires correction.

   - **Edge detection:** This technique is used to find the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction.

   1. **Low pass filter(LPF):** Low pass filter is used for smoothing noise. LPF reduces noise but accentuates blurring. It filters out high frequency noise.

   2. **High pass filter(HPF):** High pass filter is used for image sharpening and edge detection. It is just the opposite of low pass filter. It allows high frequency components of the image to pass through and filters out low frequency noise(Ref reqd). If there is no change intensity then nothing happens. But if one pixel is brighter than its immediate neighbours then it gets boosted.
      HPF reduces blurring but accentuates noise. i.e. though HPF can often improve an image by sharpening the details, overdoing it can actually degrade the image quality significantly(Ref reqd).

   3. **Band pass filter(BPF):** Band pass filtering is a tradeoff between Blurring vs. Noise. It boosts certain midrange frequencies and partially corrects for blurring, but does not boost very high (most noise corrupted) frequencies.

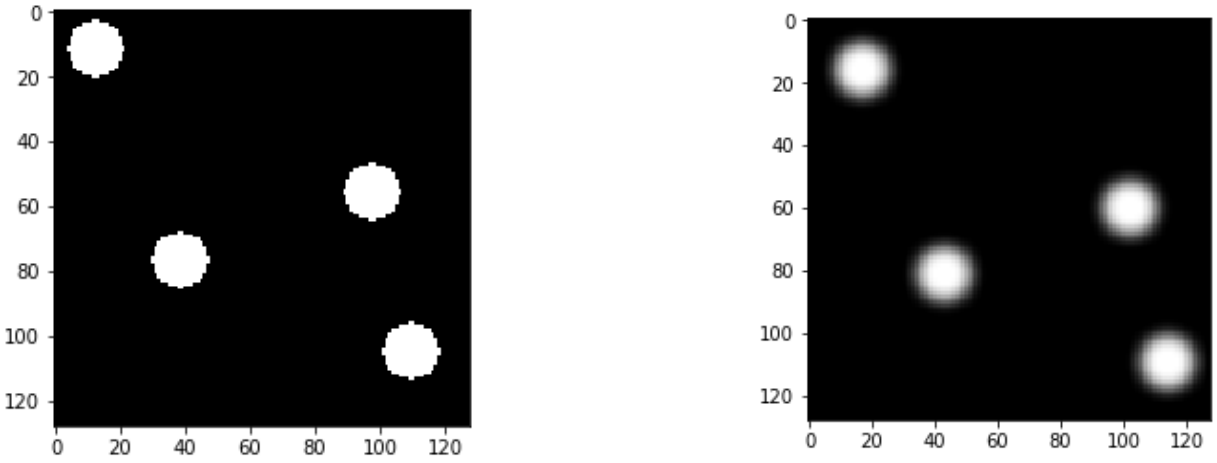
   4. **Wiener filter:** Wiener filtering is the most important technique for removal of blur in images due to linear motion or unfocussed optics[7]. The Wiener filter is Mean Square Error(MSE)- optimal stationary linear filter for images degraded by additive noise and blurring[8].

   These filters are in the form of matrices and they are called as **'Kernels'** in image processing.

8. Task 3: Write a python code to make a low pass filter using appropriate kernel. Apply this low pass filter to one of the geometric shape images created in Task 1.

   Hint: Low pass filter can be implemented using convolution function.
   Output image should be blurred as shown below.

   Expected Result:



(a) Original Image.

(b) Low pass filtered image.

Figure 9: Result of low pass filtered image.

Answer key: **Python code: LPF.py (Low pass filter.)**

Task 4: Write a python code to create a 'Class low pass filter' using appropriate kernel. Apply this low pass filter to one of the geometric shape images created in Task 1.

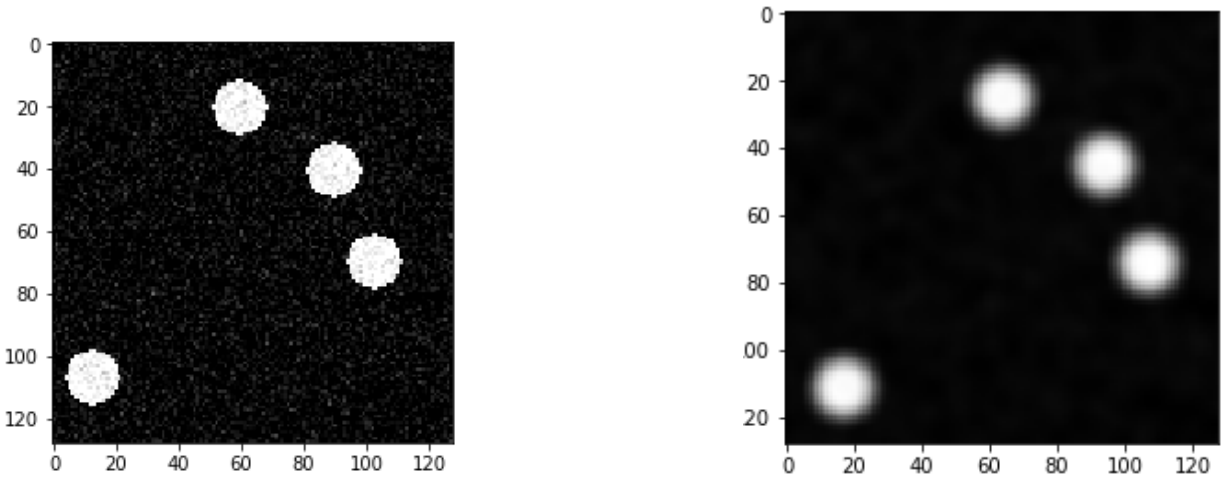
Hint: Implement Low pass filter using function defined in Task 3 and following Python class implemented in Task 2.

Answer key: **Python code: Class_lpf.py (Low pass filter using python class.)**

Task 5: Write a python code for testing low pass filter created in Task 3 by applying this low pass filter to a noisy image.

Hint: Look into skimage.util.random_noise function in scikit-image library.

Expected Result :



(a) Noisy Image.

(b) Low pass filtered image.

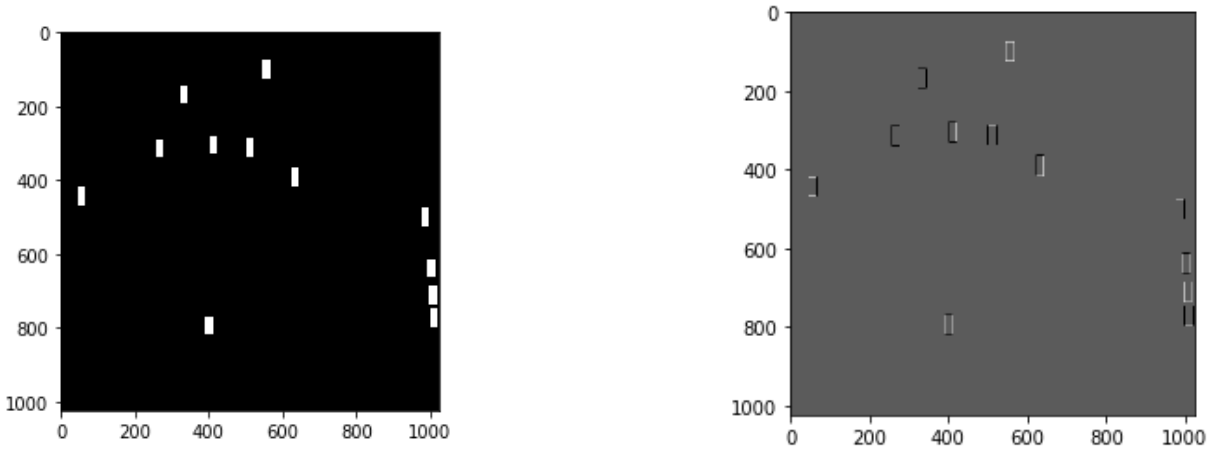
Figure 10: Result of applying low pass filter on Noisy image.

Answer key: **Python code: LPFforNoise.py (Adding noise to an image and applying low pass filter to it.)**

Task 6: Write a python code to create a 'high pass filter' using appropriate kernel. Apply this high pass filter to one of the geometric shape images created in Task 1.

Hint: High pass filter can also be implemented using convolution function.
Apply kernel used for edge detection.

Expected result:



(a) Original Image.

(b) High pass filtered image.

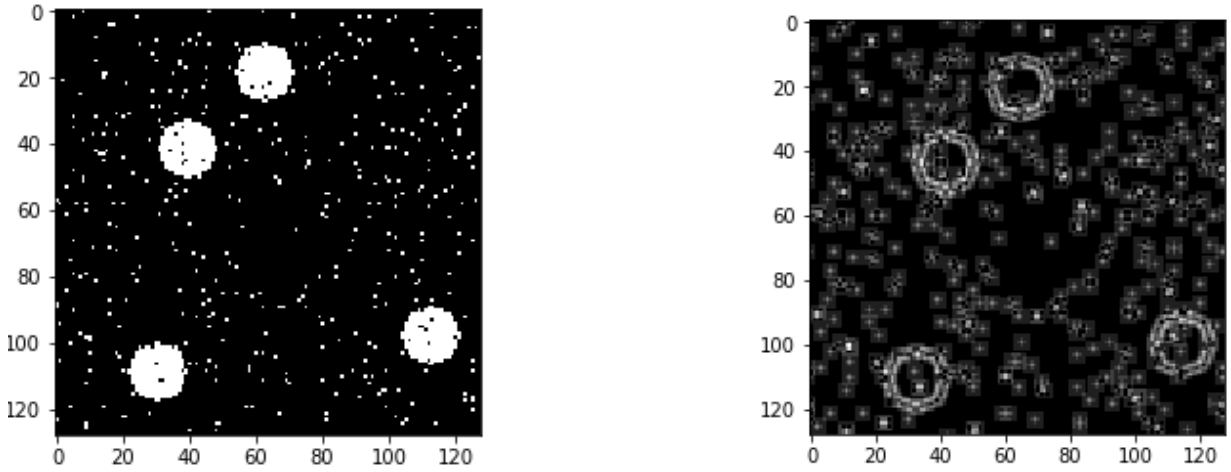
Figure 11: Result of applying high pass filter on Noisy image.

Answer key: **Python code: HPF.py (High Pass Filter)**

Task 7 : Write a python code for testing high pass filter created in Task 6 by applying this high pass filter to a noisy image.

Hint: To add noise to an image check python's scikit.util.random_noise library.

Expected Result:



(a) Original Image.

(b) High pass filtered image.

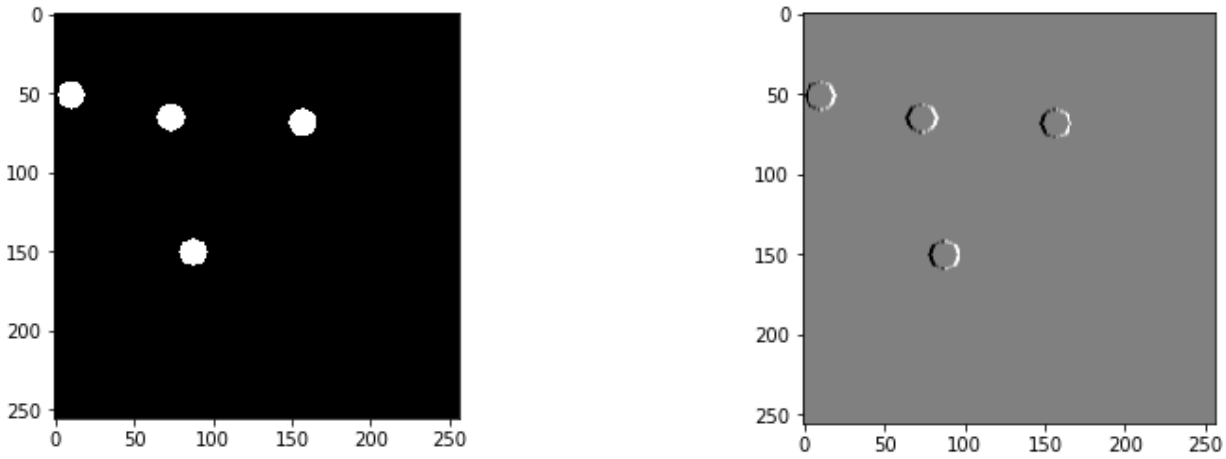
Figure 12: High pass filtered image.

Answer key: **Python code: HPFforNoise.py (To create noisy image and applying High Pass Filter to it.)**

Task 8: Write a python code to create a 'Band pass filter' using appropriate kernel. Apply this Band pass filter to one of the geometric shape images created in Task 1.

Hint : Use sobel kernels for applying band pass filter.

Expected Results:



(a) Original Image.

(b) Band pass filtered image.

Figure 13: Band pass filtered image.

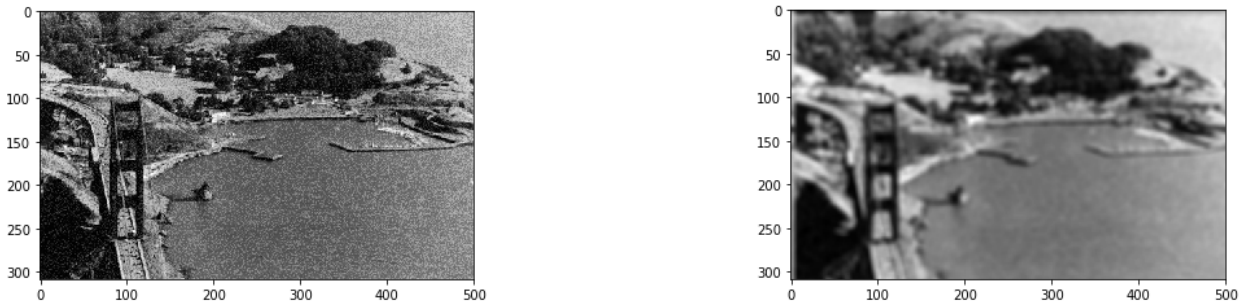
Answer key: **Python code: BPF.py (Band Pass Filter)**

Task 9: Write a python code to create a 'Weiner filter' using python's Scikit image processing library.

Hint:

- Import a noisy image. For example, image distorted due to 'salt and pepper' noise.
- Apply low pass filter to smooth the distortion.
- Use python's scipy.signal.wiener function for applying Weiner filter.

Input image and low pass filtered image :



(a) Original Noisy Image.

(b) Low pass filtered image.

Figure 14: Result of applying low pass filter on Noisy image.

Expected result:

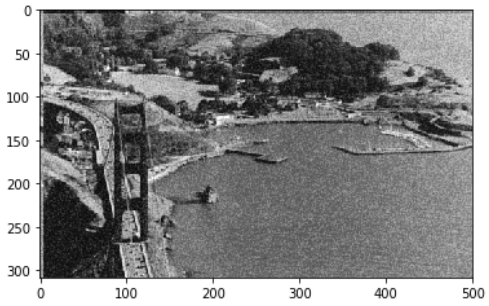


Figure 15: Weiner filtered image.

Answer key: **Python code: WienerFilter.py (Wiener Filter)**

9. **What is an Image Registration?**
   Answer:
   Image registration is the process of aligning two or more images of the same scene. This process involves designating one image as the reference image, also called the fixed image, and applying geometric transformations or local displacements to the other images so that they align with the reference.
   Images can be misaligned for a variety of reasons. Commonly, images are captured under variable conditions that can change the camera perspective or the content of the scene. Misalignment can also result from lens and sensor distortions or differences between capture devices.
   Image registration is often used as a preliminary step in other image processing applications. It enables you to compare common features in different images[4].

   Phase correlation or Cross-correlation.
   The example code given in scikit-image processing package has 'register_translation' function defined for phase correlation.
   The register_translation function uses cross-correlation in Fourier space, optionally employing an upsampled matrix-multiplication DFT to achieve arbitrary subpixel precision[5].

   In signal processing cross-correlation is a measure of similarity of two series as a function of the displacement of one relative to the other. This is also known as **Inner dot product**. The cross correlation similar in nature to the convolution of two functions.

   Applications: Cross-correlation has applications in pattern recognition, single particle analysis, Electron tomography, Averaging, Cryptanalysis and Neurophysiology[6].

Algorithm for phase correlation (using register_translation function from[5]) to identify the relative shift between two similar sized images. (This algorithm is based on the python code for phase-correlation given here[5].)

1. Import an image. (This is a reference image.)

2. Define a **shift** to be added. (To shift the reference image- This gives an offset-image.)

3. Obtain an **offset_image** by following steps:
   (a) Compute N-dimensional DFT of an image.
   (b) Apply a **fourier_shift** filter to N-dimensional DFT of an image shift. $\longrightarrow$ fourier_shift(np.fft.fftn(imag shift)

4. Obtain an inverse discrete fourier transform of **offset_image**.

5. Perform a **register_translation** operation on image and offset_image. This does efficient subpixel image translation registration be cross correlation.

6. Plot image and offset_image.

7. To show output of cross-correlation to show behind the scenes tasks done by algorithm.
   (a) Take 2-dimensional DFT of an image.
   (b) Take 2-dimensional DFT of conjugate of **offset_image**.
   (c) Obtain **'image_product'** by convolution of 1 and 2 above.

8. Obtain a **cc_image**(cross correlated image).
   (a) Take 2-Dimensional DFT of **'image_product'**.
   (b) Obtain a shifted array by shifting the zero frequency component to the center of the spectrum. This is done using **np.fft.fftshift** function on **'image_product'** obtained in above step. This gives the cross-correlated image.

9. For Subpixel precision perform **register_translation** operation on image and an offset_image with upsample_factor set to specific integer. This does efficient subpixel image translation registration for cross correlation.

Task 10: Write a python code for drift correction in a given image.

Hint: Follow the algorithm explained for phase correlation in the previous page. For this task use image 'Airplane.tiff'

Expected result:

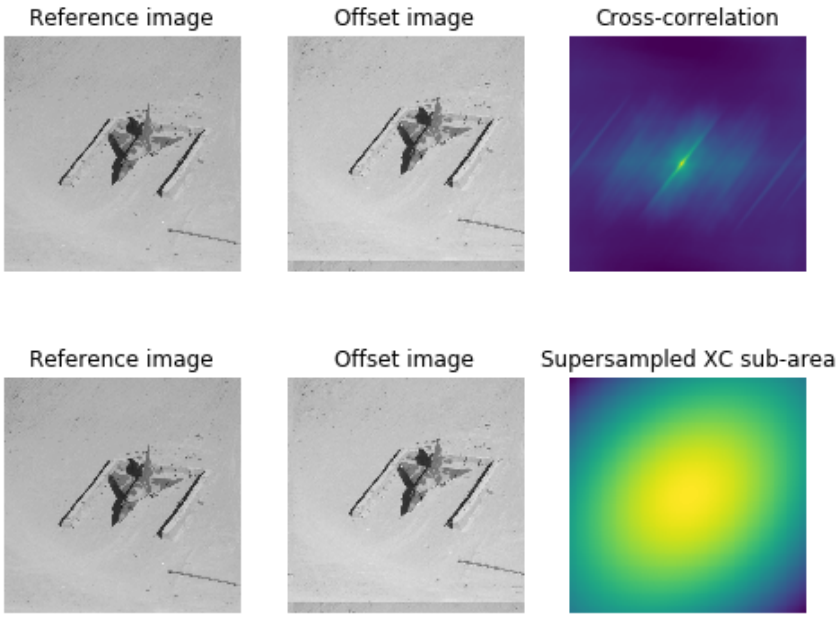


Figure 16: Phase Correlation.

Answer key: **Python code: DriftCorrection.py (Image registration using phase correlation method.)**

Thresholding and Binarization

1. **Binarization:**
   - Image binarization is the process of separation of pixel values into two groups viz.
     (a) White as background pixel and
     (b) Black as foreground pixel.
   - Thus the objective of binarization is to mark pixels that belong to true foreground regions with a single intensity and background regions with different intensities[11].

2. **Thresholding:**
   - Thresholding plays a major role in binarization of the images.
   - It is a simplest method of image segmentation.
   - It is used to create a binary image from a grayscale image.
   - The purpose of thresholding is to extract those pixels from some image which represent an 'object' (either text or other line image data). Though the information is binary the pixels represent a range of intensities.

   - The simplest thresholding methods replace,
     – pixel in an image with a black pixel if image intensity I_i, j is less than some fixed constant value T that is $I_{i,j} < T$ or
     – a white pixel if the image intensity is greater than that constant $I_{i,j} > T$ [10].

   - There are two types of thresholding algorithms viz. **Global thresholding** and **Local or adaptive thresholding**.

     (a) **Global thresholding:** For all the image pixels, single threshold value is used. When the pixel values of the components and that of background are fairly consistent in their respective values over the entire image, global thresholding could be used.
     (b) **Local or dynamic or Adaptive thresholding:** This thresholding method uses different threshold values for different local area of the image[11]. In case of large variation in the image background intensity, adaptive thresholding may produce better results[10].
   - **Otsu's thresholding method:**
     – It is the most commonly used method for image binarization using global thresholding method.
     – Used to automatically perform clustering-based image thresholding, or, the reduction of a graylevel image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal[20].

– 'Otsu's method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels that either fall in foreground or background. The aim is to find the optimum threshold value where the sum of foreground and background spreads is at its minimum.
– It is important in image processing to select an adequate threshold of gray level for extracting objects from their background. In an ideal case, the intensity histogram is a bi-modal which has a deep and sharp valley between two peaks representing foreground and background respectively. In such cases, the threshold separating those two classes can be chosen at the bottom of this valley[19].'
– **Maths of Otsu's method:**[20].

(a) In Otsu's method we exhaustively search for the threshold that minimizes the intra-class variance (the variance within the class), defined as a weighted sum of variances of the two classes:
$\sigma_w^2(t)$=$\omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2(t)$

Where Weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t and σ_0^2 and σ_1^2 are variances of these two classes.

(b) The class probability $\omega_{0,1}(t)$ is computed from $\boldsymbol{L}$ histograms:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i)$$
$$\omega_1(t) = \sum_{i=t}^{L-1} p(i)$$

(c) Otsu shows that minimizing the intra-class variance is the same as maximizing inter-class variance, which is expressed in terms of class probabilities ω and class means μ:

$$\begin{aligned}\sigma_b^2(t) &= \sigma^2 - \sigma_w^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \\ &= \omega_0(t)\omega_1(t)\left[\mu_0(t) - \mu_1(t)\right]^2\end{aligned}$$

(d) The class mean $\mu_{0,1,T}(t)$ is:

$$\mu_0(t) = \sum_{i=0}^{t-1} i\frac{p(i)}{\omega_0}$$
$$\mu_1(t) = \sum_{i=t}^{L-1} i\frac{p(i)}{\omega_1}$$
$$\mu_T = \sum_{i=0}^{L-1} ip(i)$$

(e) The following relations can be easily verified:

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T$$
$$\omega_0 + \omega_1 = 1$$

The class probabilities and class means can be computed iteratively. This idea yields an effective algorithm.

– **Algorithm for Otsu's method:**[19],[20].

(a) Read gray scale image.

(b) Compute histogram and probabilities of each intensity level.

(c) Select a threshold and refer it as t.

i. Calculate foreground variance.

ii. Calculate background variance.

(d) Calculate within-class variance.

(e) Repeat steps 3 and 4 for all possible threshold values t=1,....,maximum intensity.

(f) Final global threshold, **T** = Threshold in MIN(within-class variance). Desired threshold corresponds to the maximum $\sigma_b^2(t)$

(g) Binarized Image = gray scale image $>$ **T**

Task 11: Import grayscale image 'Nemo.jpg'. Write python code for converting this grayscale image into binary image using Otsu's thresholding method.

Hint: Follow algorithm for Otsu's thresholding method given on the previous page.

Expected result:

(a) Original image.

(b) Thresholded image.

Figure 17: Thresholded Image.

Answer key: **Python code: Thresholding.py**

Task 12: For the same image 'Nemo.jpg', write a python code to apply adaptive and global thresholding methods and compare both the output images.

Hint: Use reference http://scikit-image.org/docs/dev/auto_examples/xx_applications/plot_thresholding.html

Expected result:

(a) Adaptive Thresholding.

(b) Global Thresholding.

Figure 18: Comparision of Adaptive and Global Thresholding methods.

Answer key: **Python code: CompareThresholding.py**

Morphological Tools

- 'Morphological image processing is a collection of non-linear operations related to the shape or morphology of features in an image, such as boundaries, skeletons, etc. In any given technique, we probe an image with a small shape or template called a **structuring element**, which defines the region of interest or neighbourhood around a pixel.

- Morphological operations rely only on the relative ordering of pixel values, not on their numerical values, and therefore are especially suited to the processing of binary images. Morphological operations can also be applied to greyscale images such that their light transfer functions are unknown and therefore their absolute pixel values are of no or minor interest[21].'
- **Structuring element :**
  - 'The structuring element is a small matrix of pixels, each with a value of zero or one representing a small binary image. This matrix is positioned at all possible locations in the image and is compared with the corresponding neighbourhood of pixels. Some operations test whether the element "fits" within the neighbourhood, while others test whether it intersects the neighbourhood.
    Stucturing elements play in moprphological image processing the same role as convolution kernels in linear image filtering[21].
  - This small matrix dimensions specify the size of the structuring element.
  - The pattern of ones and zeros specifies the shape of the structuring element.
  - An origin of the structuring element is usually one of its pixels[21].'
  - The basic idea in binary morphology is to probe an image with a simple, pre-defined shape, drawing conclusions on how this shape fits or misses the shapes in the image. This simple "probe" is called the structuring element, and is itself a binary image (i.e., a subset of the space or grid)[22].

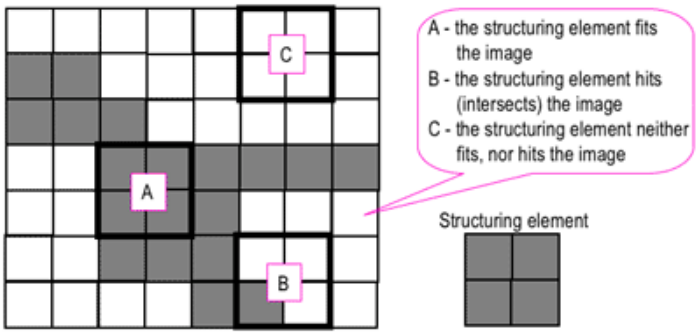


Figure 19: Fitting and missing of Shape in an image[21].

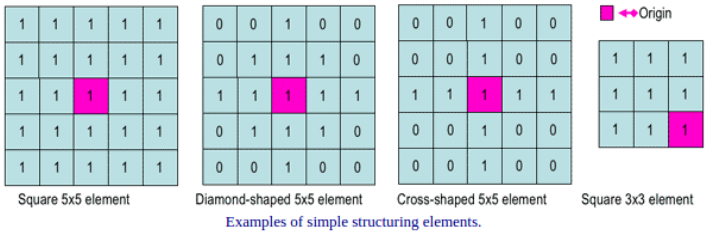
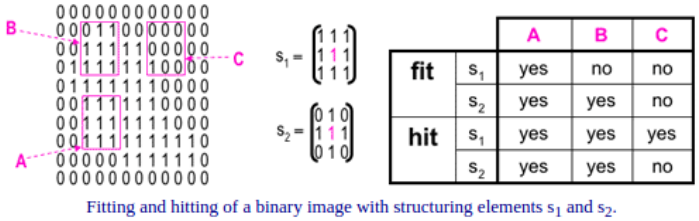


Figure 20: Simple structuring elements[21].

– 'When a structuring element is placed in a binary image, each of its pixels is associated with the corresponding pixel of the neighbourhood under the structuring element.

1. The structuring element is said to *fit* the image if, for each of its pixels set to 1, the corresponding image pixel is also 1.
2. Whereas, structuring element is said to *hit*, or *intersect*, an image if, at least for one of its pixels set to 1 the corresponding image pixel is also 1.
3. It's a *miss* if, at least for one of its pixels set to 1, none of corresponding image pixels is 1.



		A	B	C
fit	s_1	yes	no	no
	s_2	yes	yes	no
hit	s_1	yes	yes	yes
	s_2	yes	yes	no

Figure 21: Fitting and hitting of binary image with structural images S_1 and S_2[21].

– The hit and miss transform can be used for detecting specific shapes (spatial arrangements of object and background pixel values) if the two structuring elements present the desired shape, as well as for thinning or thickening of object linear elements.
– The size of the structuring element is most important to eliminate noisy details but not to damage objects of interest[21].'

Various structuring element available in scikit-image Morphology module.

1. *'ball(radius)* : Generates a ball-shaped structuring element.
   This is the 3D equivalent of a disk. A pixel is within the neighborhood if the euclidean distance between it and the origin is no greater than radius.
2. *cube(width)* : Generates a cube-shaped structuring element.
   This is the 3D equivalent of a square. Every pixel along the perimeter has a chessboard distance no greater than radius (radius=floor(width/2)) pixels.
3. *diamond(radius)* : Generates a flat, diamond-shaped structuring element. A pixel

is part of the neighborhood (i.e. labeled 1) if the city block/Manhattan distance between it and the center of the neighborhood is no greater than radius.

4. *disk(radius)* : Generates a flat, disk-shaped structuring element.
   A pixel is within the neighborhood if the euclidean distance between it and the origin is no greater than radius.
5. *octagon(m, n)* : Generates an octagon shaped structuring element.
   For a given size of (m) horizontal and vertical sides and a given (n) height or width of slanted sides octagon is generated. The slanted sides are 45 or 135 degrees to the horizontal axis and hence the widths and heights are equal.
6. *octahedron(radius)* : Generates a octahedron-shaped structuring element.
   This is the 3D equivalent of a diamond. A pixel is part of the neighborhood (i.e. labeled 1) if the city block/Manhattan distance between it and the center of the neighborhood is no greater than radius.
7. *rectangle(width, height)* : Generates a flat, rectangular-shaped structuring element.
   Every pixel in the rectangle generated for a given width and given height belongs to the neighborhood.
8. *square(width)* : Generates a flat, square-shaped structuring element.
   Every pixel along the perimeter has a chessboard distance no greater than radius (radius=floor(width/2)) pixels.
9. *star(a)* : Generates a star shaped structuring element.
   Start has 8 vertices and is an overlap of square of size 2*a + 1 with its 45 degree rotated version. The slanted sides are 45 or 135 degrees to the horizontal axis [28].'

'In this document we outline the following basic morphological operations:

1. Erosion
2. Dilation
3. Opening
4. Closing
5. White Tophat
6. Black Tophat

1. **Erosion and Dilation:**[23].

Morphological Erosion	Morphological Dilation
1. Sets a pixel at (i, j) to the minimum over all pixels in the neighbourhood centered at (i, j) 2. Enlarges dark regions and shrinks bright regions	1. Sets a pixel at (i, j) to the maximum over all pixels in the neighbourhood centered at (i, j) 2. Enlarges bright regions and shrinks dark regions.

2. **Opening and Closing:**[23].

Morphological Opening	Morphological Closing
1. It is defined as an erosion followed by a dilation 2. Opening can remove small bright spots (i.e. "salt") and connect small dark cracks.	1. It is defined as a dilation followed by an erosion. 2. Closing can remove small dark spots (i.e. "pepper") and connect small bright cracks.

3. **White-tophat and Black-tophat:**[23].

White-tophat	Black-tophat
1. It is defined as the image minus its morphological opening. 2. This operation returns the bright spots of the image that are smaller than the structuring element.	1. It is defined as morphological closing of image minus the original image. 2. This operation returns the dark spots of the image that are smaller than the structuring element.

For Tasks 12, 13 and 14 import image 'Butterfly.jpg'. Use structuring element 'disk' for all the tasks.

Task 12: Write python code to apply **erosion** and **dilation** function on given image.

Expected result :

(a) Erosion. (b) Dilation.

Figure 22: Eroded and dilated images.

Answer key : **Python code: ErosionNDilation.py**

Task 13: Write python code to remove small bright spots (**Opening**) and black spots (**Closing**) from given image.

Expected result :

(a) Opening. (b) Closing.

Figure 23: Application of Opening and closing function.

Answer key :**Python code: OpeningNClosing.py**

Task 14: Write python code to recover the bright spots (**White-tophat**) and dark spots (**black-tophat**) in the image given in Task 13. Take output images of Task 13 as input images.

Expected result :

(a) White-tophat.

(b) Black-tophat.

Figure 24: Application of white-tophat and black-tophat function.

Answer key : **Python code: wNbTophat.py**

Bonus Task: Use anyone of the following structuring elements : diamond, rectangle, square or star. Apply it to tasks 12, 13 and 14 and compare results with the outputs for structuring element 'disk' applied previously.

Image Segmentation

- Image segmentation is the process of subdividing a digital image into its component regions or categories which correspond to its object or parts of object[26].
- Segmentation algorithms generally are based on one of 2 basis properties of intensity values:
  Discontinuity : To partition an image based on sharp changes in intensity (such as edges).
  Similarity : To partition an image into regions that are similar according to a set of predefined criteria[27].
- Segmentation is often the critical step in image analysis: the point at which we move from considering each pixel as a unit of observation to working with objects (or parts of objects) in the image, composed of many pixels. If segmentation is done well then all other stages in image analysis are made simpler[26].
- 'The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image. Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic.
- By segmentation we get simplified image which is more meaningful and easier to analyse[25]'.
- Image segmentation can be done based on:
  1. Edge : In edge-based segmentation, an edge filter is applied to the image, pixels are classified as edge or non-edge depending on the filter output, and pixels which are not separated by an edge are allocated to the same category[26].
     Edge-based methods center around contour detection.

     Canny edge detector :
     - 'The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.
     - Canny edge detection technique is useful to extract structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems[29].'
     - The Canny filter uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients.The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude[30].
     - Python's scikit-image library provides a function *skimage.feature.canny(image, sigma=1.0, low_threshold=None, high_threshold=None, mask=None, use_quantiles=False)* for Edge filtering an image using the Canny algorithm. The Canny has three adjustable parameters: the width of the Gaussian (the noisier the image, the greater the width), and the low and high threshold for the hysteresis thresholding.

'Process of Canny edge detection algorithm:

(a) Apply Gaussian filter with sigma width to smooth the image in order to remove the noise.

(b) Find the intensity gradients of the image.

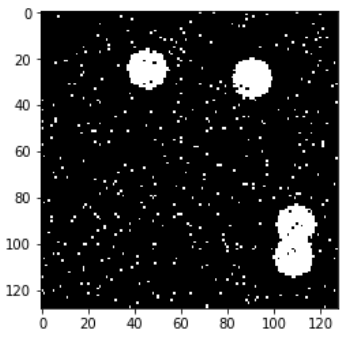
(c) Apply non-maximum suppression to get rid of spurious response to edge detection.

(d) Apply double threshold to determine potential edges.

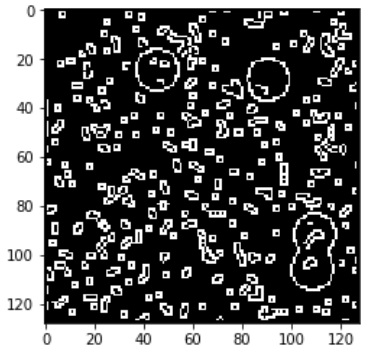
(e) Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges[29].'(First label all points above the high threshold as edges. Then recursively label any point above the low threshold that is 8-connected to a labeled point as an edge[30].)

Task 15: Write a python code to apply Canny edge detection filter to one of the geometric shapes created in Task 1.
Hint : Look into http://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html

Expected result:

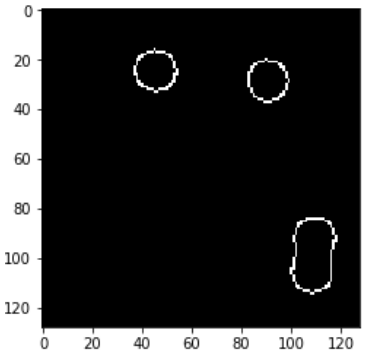


(a) Noisy Image.

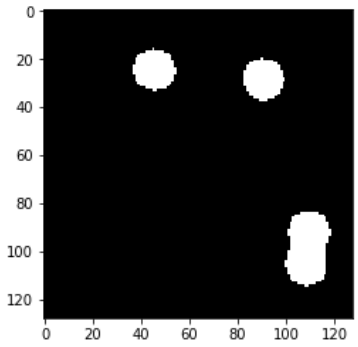


(b) Default Smoothing.

Figure 25: Canny Edge detection filtering.



(a) Canny Filter with increased smoothing.



(b) Filled interior.

Figure 26: Canny Edge detection filtering.

Answer key : **Python code: CannyFilter.py**

2. Region :
   - Region-based segmentation algorithms operate iteratively by grouping together pixels which are neighbours and have similar values and splitting groups of pixels which are dissimilar in value[26].
   - 'This segmentation may be considered as spatial clustering: *spatial*; as pixels in the same category form a single connected component. and
     clustering; as pixels with similar values (greylevel, texture, color etc.) are grouped together[26].'
   - Region based segmentation methods can be categorized into:
     (a) Methods that merge pixels: Initialize by giving all the pixels a unique label. All pixels in the image are assigned to a region. If two adjacent regions are collectively similar enough, merge them into single region. If two adjacent regions are collectively similar enough merge them likewise. This collective similarity is usually based on comparing the statistics of each region. Eventually this method will converge when no further such merging are possible[32].
     (b) Methods which split the image into regions.
     (c) Methods which both split and merge : Separate the image into regions based on a given similarity measure. Then merge regions based on the same or a different similarity measure[31].

   - **Watershed Algorithm:**
     * Watershed is a classical algorithm used for segmentation, that is, for separating different objects in an image[33].

     * 'This algorithm can be used on images derived from:
       · The intensity image
       · Edge enhanced image
       · Distance transformed image
       · Gradient of the image

* Consider an image as a 3D topographic surface, (x,y,intensity), with both valleys and mountains.

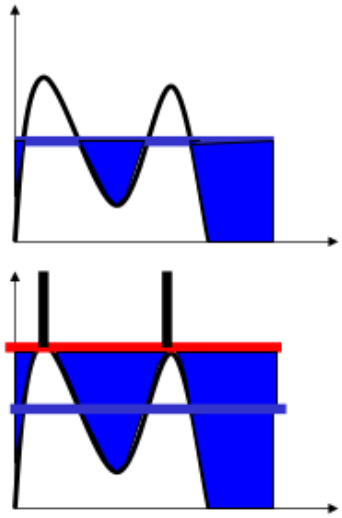


Figure 27

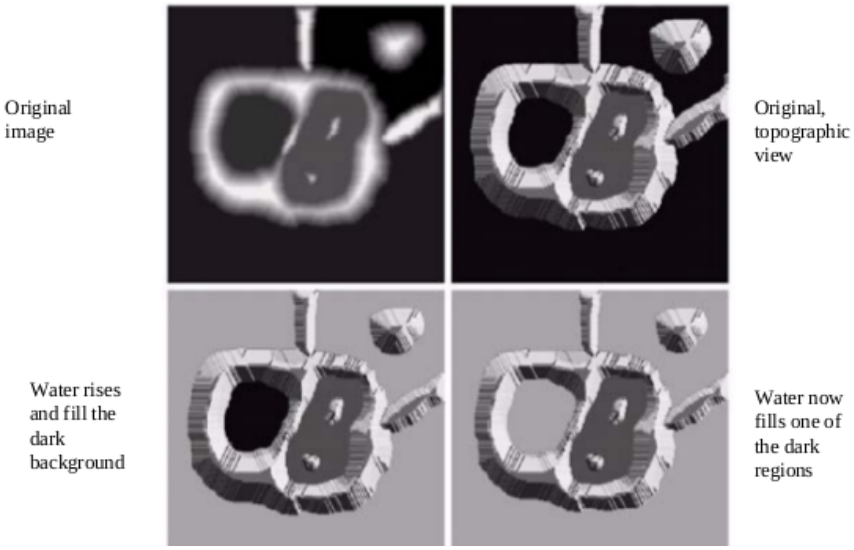


Figure 28

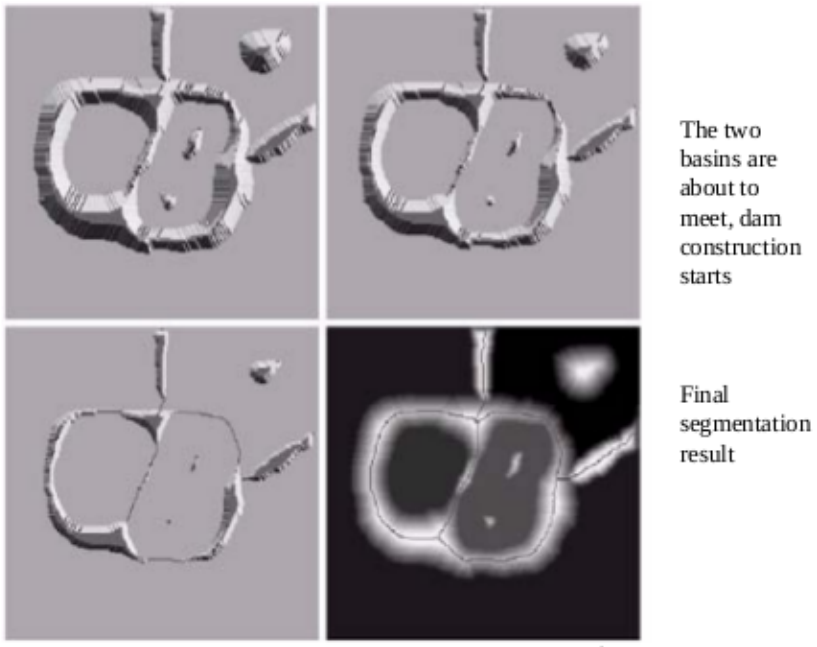


Figure 29: Watershed algorithm[31].

* Assume that there is a hole at each minimum, and that the surface is immersed into a lake.
* The water will enter through the holes at the minima and flood the surface.
* To avoid two different basins to merge, a dam is built.
* Final step: the only thing visible would be the dams.
* The connected dam boundaries correspond to the watershed lines[31].'
* 'Starting from user-defined markers, the watershed algorithm treats pixels values as a local topography (elevation).
* The algorithm floods basins from the markers, until basins attributed to different markers meet on watershed lines. In many cases, markers are chosen as local minima of the image, from which basins are flooded[33].'

Task 16: Write python code to perform watershed transform on one of the images created in Task 1.

Hint : Look into http://scikit-image.org/docs/dev/auto_examples/xx_applications/plot_coins_segmentation.html

Expected Result:

(a) Original Image.

(b) Watershed transformed image.

Figure 30: Watershed transformation-Region based image segmentation.

Answer key : **Python code: Watershed.py**

References

[1] Computational imaging.
http://training.scicomp.jic.ac.uk/docs/bioimaging_course_book/computational_imaging.html

[2] Introduction to image processing.
https://docs.google.com/viewer?url=https%3A%2F%2Fwww.spacetelescope.org%2Fstatic%2Fprojects%2Ffits_liberator%2Fimage_processing.pdf

[3] What is an Image?
http://www.cafeaulait.org/course/week9/23.html

[4] Image Processing Toolbox.
https://www.mathworks.com/help/images/approaches-to-registering-images.html

[5] Cross-Correlation(Phase Correlation).
http://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.register_translation

[6] Cross-correlation : Wikipedia.
https://en.wikipedia.org/wiki/Cross-correlation

[7] Wiener filter.
https://www.clear.rice.edu/elec431/projects95/lords/wiener.html

[8] Wiener filter.
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node15.html

[9] Python codes written by Prof. Duane Loh.(Center for BioImaging Sciences, National University of Singapore.)

[10] Thresholding.
https://en.wikipedia.org/wiki/Thresholding_%28image_processing%29

[11] Thresholding.
http://www.ancient-asia-journal.com/articles/10.5334/aa.06113/

[12] Programming Computer vision with Python.
ccdraft.pdf, Jan Erik Solem.

[13] Scikit-image.
http://www.scipy-lectures.org/packages/scikit-image/#scikit-image-and-the-scipy-ecosystem

[14] Python Libraries for Image processing.
http://www.datasciencecentral.com/profiles/blogs/9-python-libraries-which-can-help-you-in-

[15] Python Imaging Library.
http://python-guide-pt-br.readthedocs.io/en/latest/scenarios/imaging/

[16] ImageJ User Guide. IJ1.46r
Tiago Ferreira, Wayne Rasband.

[17] Digital Imaging.
https://micro.magnet.fsu.edu/primer/digitalimaging/digitalimagebasics.html

[18] Image processing toolbox.
http://www-rohan.sdsu.edu/doc/matlab/toolbox/images/intro6.html

[19] Threshold and binarization for document image analysis using otsu's Algorithm. Dr. Neeraj Bhargava, Anchal Kumawat, Dr. Ritu Bhargava. International Journal of Computer Trends and Technology (IJCTT) – volume 17 Number 5 Nov 2014.

[20] Ostu's Method.
https://en.wikipedia.org/wiki/Otsu%27s_method

[21] Morphological Tools.
https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm

[22] Structuring Element.
https://en.wikipedia.org/wiki/Mathematical_morphology#Structuring_element

[23] Morphological Tools.
http://scikit-image.org/docs/dev/auto_examples/xx_applications/plot_morphology.html

[24] Image Segmentation.
http://scikit-image.org/docs/dev/user_guide/tutorial_segmentation.html

[25] Image Segmentation-Wikipedia.
https://en.wikipedia.org/wiki/Image_segmentation

[26] Image Segmentation.
www.bioss.ac.uk/people/chris/ch4.pdf

[27] Image Segmentation.
http://slideplayer.com/slide/8143316/

[28] Morphology structuring elements.
http://scikit-image.org/docs/dev/api/skimage.morphology.html#skimage.morphology.cube

[29] Canny edge detector.
https://en.wikipedia.org/wiki/Canny_edge_detector

[30] Canny edge detector(scikit-image).
http://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html

[31] INF 4300-Digital Image Analysis.
'Region and edge based segmentation'. Fritz Albregtsen.

[32] Region based segmentation.
Lecture 18. Segmentation(Region based). Bryan S. Morse. Brigham Young university. 1998-2000.

[33] Watershed algorithm.
http://scikit-image.org/docs/dev/auto_examples/segmentation/plot_watershed.html