Perception for Autonomous Systems

Lecture 1 - Image Processing (01/02/2021)

Introduction Questions

1. What is "Perception"?

2. What is an "Autonomous System"?

3. Why Autonomous Systems need Perception?

Outline/Content:

- What is Image Processing?
- Image
- Color
- Linear Filtering
- Non-linear Filters / Thresholding
- Morphology
- Connrected Components Analysis
- Summary

> Topics and Reading Sources:
>
> - Color
>   - Book A, Chapter 2.3.2
> - Linear Filtering
>   - Book A, Chapters 3.1, 3.2
> - Non-linear Filters, Morphology, Thresholding and Connected Components Analysis
>   - Book A, Chapter 3.3

What is Image Processing?

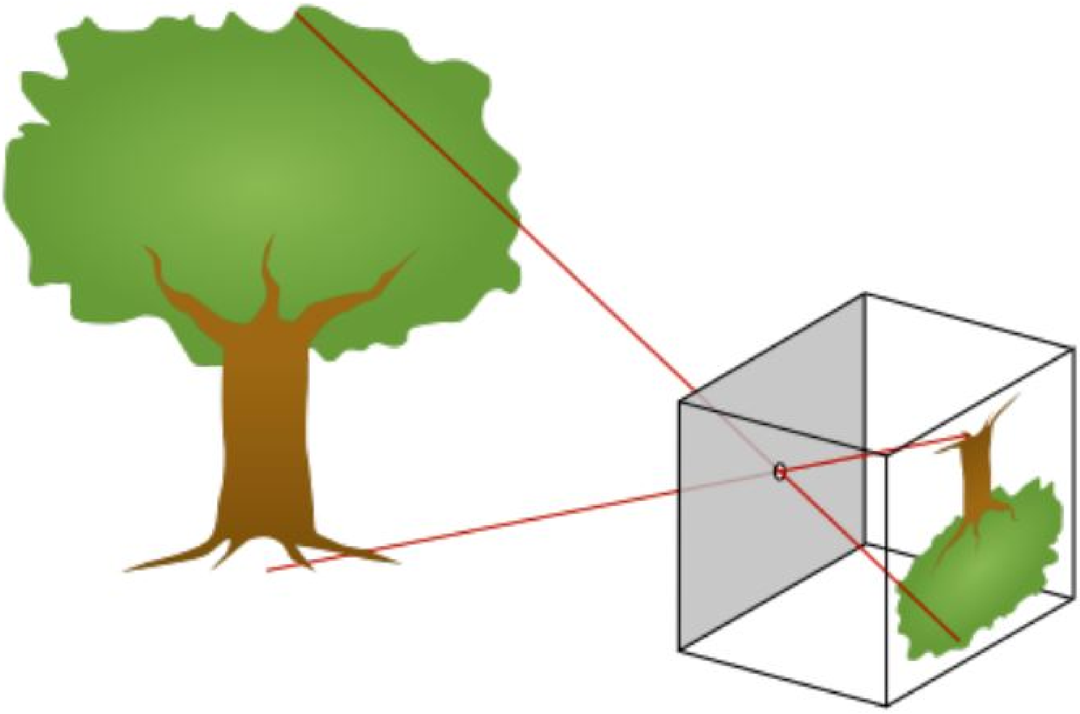
Image processing are operations we perform on an image to change or enhance it and make it suitable for further analysis. So that useful information can be highlighted or get extracted from it.

> MAYBE ADD MORE POINTS HERE

Image

Pinhole model **Youtube Link**

The pinhole model describes the process of reducing the incidence of light to such an extent that the section of the image in front of you can be seen (upside down). This process is used in all given cameras and is known as "aperture".



Pixels

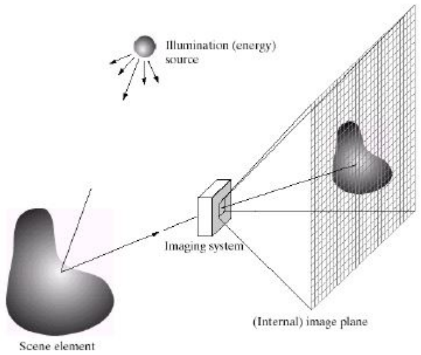
Pixels are the raw building blocks of an image. Each image consists of a set of pixels. There is no finer granularity than the pixel. Normally, a pixel is considered to be the "color" or the "intensity" of the light that appears at a particular location in our image.

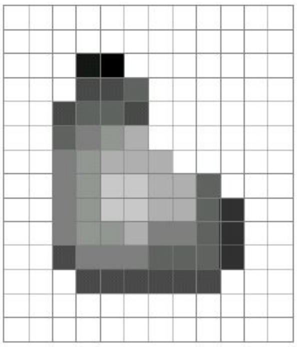
Most pixels are represented in two ways:

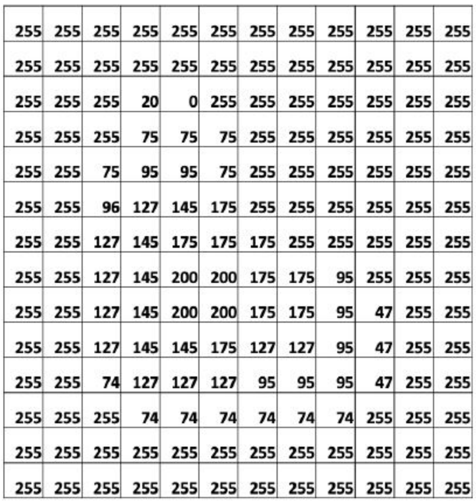
- Grayscale: single channel
- Color: three channels

What is a (digital) image?

The resolution of an image is given by the representation of columns_number (width) x rows_number (height) of pixels. For example, an image with a resolution of 1,000 x 750 is 1,000 pixels wide and 750 pixels high. We can think of an image as a (multi-dimensional) matrix with the form W x H x D (dimensions/channels). In this case, our matrix has 1,000 columns with 750 rows. Overall, there are 1,000 x 750 = 750,000 total pixels in this image. Each cell can take discrete/integer values (quantized samples ranging from 0 to 255).







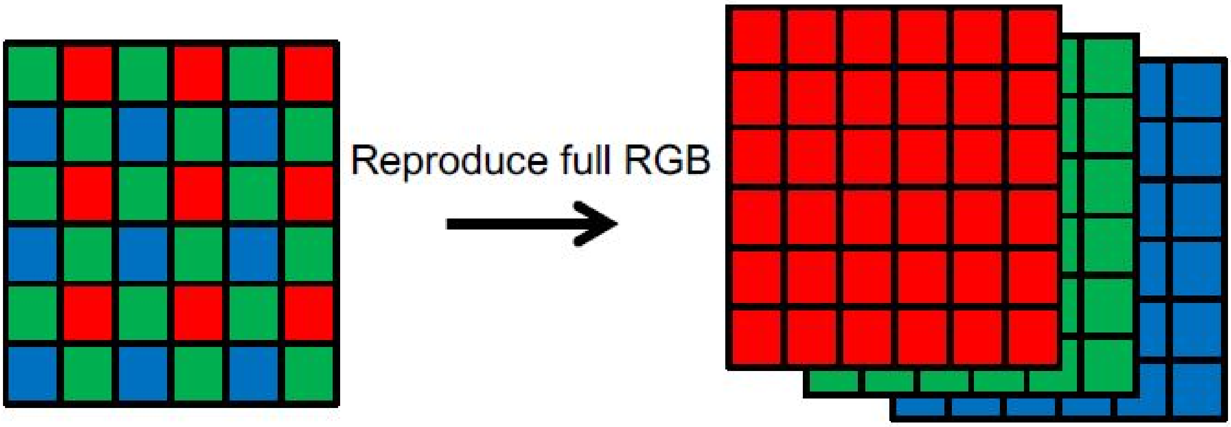
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

> HINT: Image processing libraries such as OpenCV and scikit-image represent RGB images as multidimensional NumPy arrays with shape (height, width, depth). So, height and width are reversed.

Color

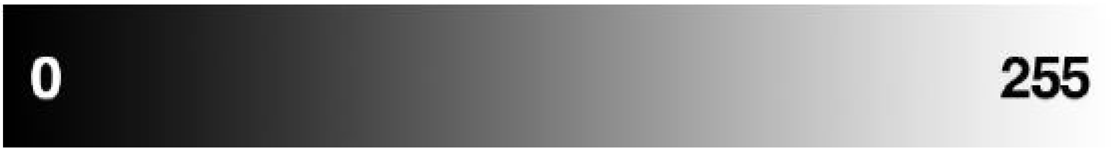
Bayer Color filter

The Bayer pattern places green filters over half of the sensors (in a checkerboard pattern), and red and blue filters over the remaining ones. The reason that there are twice as many green filters as red and blue is because the luminance signal is mostly determined by green values and the visual system is much more sensitive to high frequency detail in luminance than in chrominance.



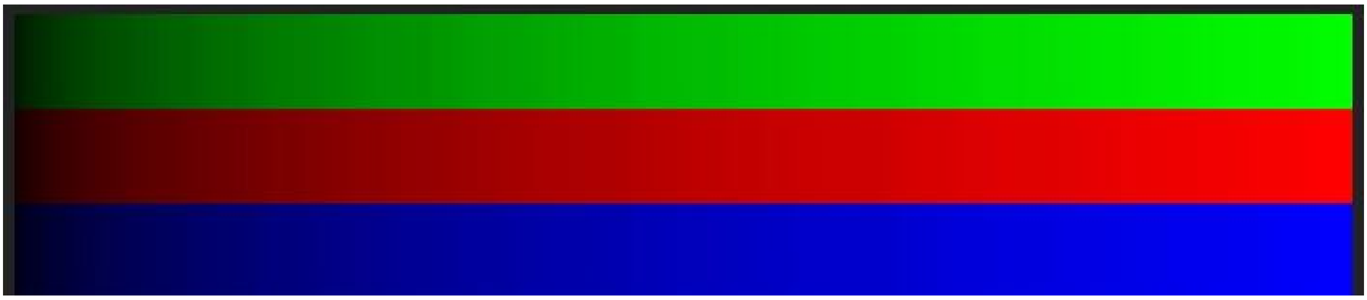
Grayscale

In a grayscale image, each pixel is a scalar (single-channel) value between 0 and 255, where zero corresponds to "black" and 255 to "white". Values in between are different shades of gray, with values closer to 0 being darker and values closer to 255 being lighter.



RGB

In a colored image, each pixel is a representation of three 2D images: R(x,y), G(x,y), B(x,y) with values ranging from 0 to 255. The values of a specific pixel (x,y) in the 3 images R, G, B describe the red-ness, green-ness and blue-ness of that particular pixel. Meaning 0 indicates the absence of the color, therefore "black" whereas 255 indicates the full saturation of that color. Given our three Red, Green, and Blue values, we can combine them into an RGB tuple in the form (red, green, blue). This tuple represents a given color in the RGB color space.



RGB Color Space (additive color space)

- A image consists of 25% red, 50% green and 25% blue pixels
- all colors can be reproduced by mixing Red, Green and Blue
- the more of each color is added, the brighter the pixel becomes and closer to white
- There is a maximum of 16.777.216 unique colors in the RGB color space (256 x 256 x 256)

Drawbacks of thr RGB Color Space:

- It's additive nature makes it a bit unintuitive for humans to easily define shades of color without using a "color picker" tool
- It doesn't mimic how humans perceive color

However, despite these drawbacks, nearly all images you'll work with will be represented (at least initally) in the RGB color space.

> Many other Color Spaces exist e.g. L*a*b* color space or HSI/HSV (Hue-Saturation-Value) Color spaces.

Linear Filtering

Questions to ask are:

- What are Linear Filters in Image Processing?
- What are they used for?

Filters

Filters form a new image whose pixels are a combination of the original pixels - Why?

- To get useful information from images
  - E.g., extract edges or contours (to understand shape)
- To enhance the image
  - E.g., blur to remove noise
  - E.g., sharpen to enhance image

Application of Filters: When using a filter we have two matrices. One big matrix representing the image and a small kernel or convolution matrix representing the filter of choice. Essentially, this tiny kernel sits on top of the image matrix and slides from left-to-right and top-to-bottom, applying a mathematical operation (i.e., a convolution) at each (x;y)-coordinate of the original image.

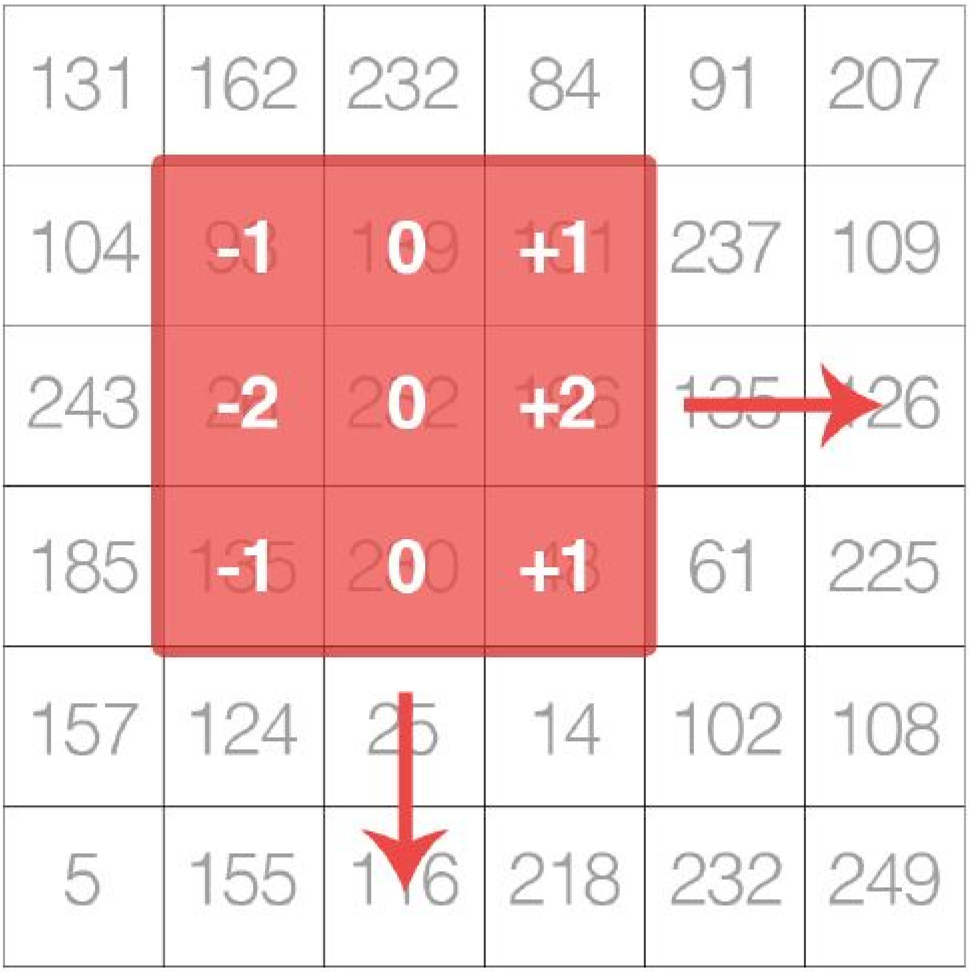
> SIDE NOTE: The Initial starting position within the pixel grid is the left upper corner at (0, 0).

Usually all the kernel cells need to have valid values. Meaning, we don't start at the corner of the images. This will introduce some loss as we are shrinking the image. There are ways to counter that, like zero padding etc.

Kernel

As already mentioned before we're sliding the kernel from left-to-right and top-to-bottom along the original image. At each (x,y)-coordinate of the original image, we stop and examine the neighborhood of pixels located at the center of the image kernel. We then take this neighborhood of pixels, convolve them with the kernel, and obtain a single output value. The output value is stored in the output image at the same (x,y)-coordinates as the center of the kernel.

Odd kernel sizes are required to ensure that there is a valid integer (x,y)-coordinate at the center of the image. Let's take a 3x3 matrix for example. Here you have the center at (1,1 => valid integer) if we start at the left upper corner. Whereas a 2x2 matrix mathematically would have it's center point at (0.5, 0.5) which is no valid expression. And visually it's even more obvious that there is no center point in a 2x2 matrix.



Convolution and cross-correlation

Application of Convolution:

- Blurring:
  - average/mean smoothing
  - gaussian smoothing
- Edge detection:
  - Laplacian
  - Sobel
  - Scharr
  - Prewitt
- Sharpen
- other filters

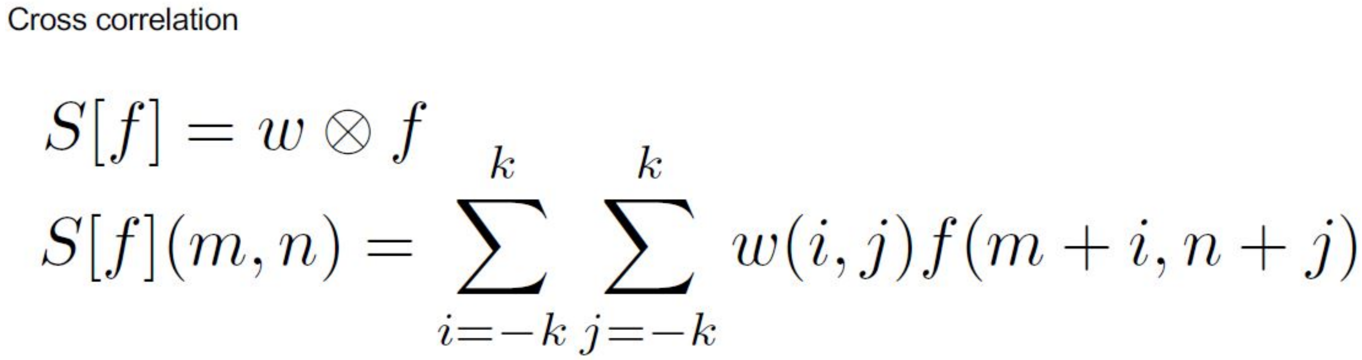
> SIDE NOTE: To sharpen an image, you blur the image and substract it from the original image and then add that result onto the original image again.

Convolutions are one of the most critical, fundamental building-blocks in computer vision and image processing. They're actually quite easy to understand. It's an element-wise multiplication of two matrices followed by a sum.

What's the difference between Convolution and cross-correlation?

Mathematically it's the same equation where simply a sign change took place. This effects the way we access the coordinates of the image (i.e., we don't have to "flip" the kernel relative to the input when applying cross-correlation).

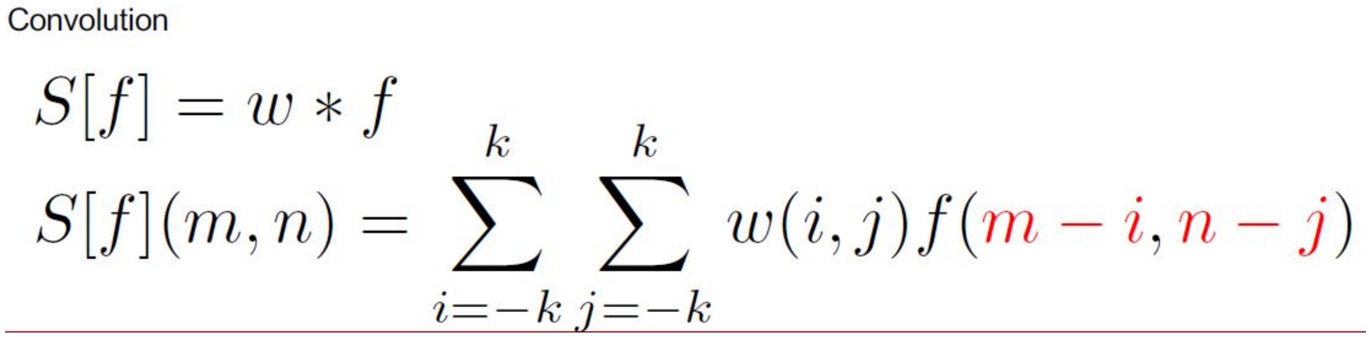
Therefore, speaking about convolution altough using cross-correlation is by design.



Cross correlation

$$S[f] = w \otimes f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$



Convolution

$$S[f] = w * f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m-i, n-j)$$

Putting everything together

A convolution requires three components:

1. An input image.
2. A kernel matrix that we are going to apply to the input image.
3. An output image to store the output of the image convolved with the kernel.

Convolution (or cross-correlation) is actually very easy. All we need to do is:

1. Select an (x;y)-coordinate from the original image.
2. Place the center of the kernel at this (x;y)-coordinate.
3. Take the element-wise multiplication of the input image region and the kernel, then sum up the values of these multiplication operations into a single value. The sum of these multiplications is called the kernel output.
4. Use the same (x;y)-coordinates from Step #1, but this time, store the kernel output at the same (x;y)-location as the output image.

Below you can find an example of convolving (denoted mathematically as the **star** operator) a 3x3 region of an image with a 3x3 kernel used for blurring:

$$O_{i,j} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \star \begin{bmatrix}93 & 139 & 101\\ 26 & 252 & 196\\ 135 & 230 & 18\end{bmatrix} = \begin{bmatrix}1/9\times 93 & 1/9\times 139 & 1/9\times 101\\ 1/9\times 26 & 1/9\times 252 & 1/9\times 196\\ 1/9\times 135 & 1/9\times 230 & 1/9\times 18\end{bmatrix}$$

Therefore,

$$O_{i,j} = \sum \begin{bmatrix}10.3 & 15.4 & 11.2\\ 2.8 & 28.0 & 21.7\\ 15.0 & 25.5 & 2.0\end{bmatrix} \approx 132.$$

After applying this convolution, we would set the pixel located at the coordinate (i; j) of the output image O to Oi; j = 132. That’s all there is to it! Convolution is simply the sum of element-wise matrix multiplication between the kernel and neighborhood that the kernel covers of the input image.

Mean filtering

Objective: Introduce blur to smoothen the image and remove noise.

Take the sum of each cell within the filter size and divide by the number of cells. For example:

3x3 Filter Size

x1	x2	x3
0	0	0
10	40	0
10	0	0

=> (0 + 0 + 0 + 10 + 40 + 0 + 10 + 0 + 0) / 9 = 6.66 = 7

> Pixels only take integer values. Therefore, 6.66 is being rounded to 7.

> There are two kinds of operations to apply a filter on a pixel. There are point operators or point processes that manipulate each pixel independently of its neighbors (Brightness, contrast, color correction/transformation). And then there are area-based operators, where each new pixel value depends on the value of it's neighbors.

Non-linear filters: Thresholding

By using non-linear filters we set a threshold which acts as our if condition . For example, we could set the threshold to 200 and set everything to 255 if true and 0 otherwise. This will result in a pure black & white image.

What if the threshold could be adaptive (instead of a pre-defined value)?

- Otsu's method performs automatic image tresholding

    - The algorithm exhaustively searches for the threshold that minimizes the instra-class variance, defined as a weighted sum of variances of the two classes.

Non-linear filters: Rectification

- g(m,n) = max(f(m,n), 0)
- crucial component of modern convolutional networks

Non-linear filters

- Sometimes mean filters does not work
- mean is sensitive to outliers
- median filter: Replace pixel by median of neighbors

Morphology PyImageSearch, Youtube

The most common binary image operations are called Morphological Operations

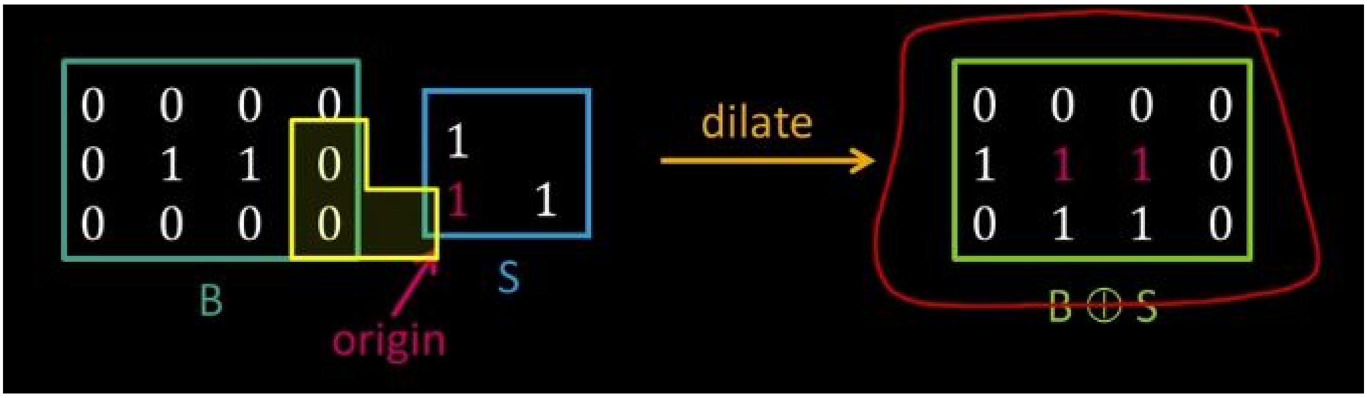
Basic Morphological Operations:

- Dilation
    - Increase white pixels, decrease black pixels
        - Case 1: Perfect match replace origin with 1
        - Case 2: Partial match replace origin with 1
        - Case 3: No match replace origin with 0
- Erosion
    - Decrease white pixels, increase black pixels
        - Case 1: Perfect match replace origin with 1
        - Case 2: Partial match replace origin with 0
        - Case 3: No match replace origin with 0
- Opening
    - Erosion then Dilation
- Closing
    - Dilation then Erosion

Structuring Element A shape mask used in basic morphological ops.

- Any shape, size that is digitally representable (Box, hexagon, disk, rectangle, etc.)
- With a defined origin

Application: The structuring element represents a pixel shape matrix with ones and usually has it's origin (highlighted) in the center of the shape. Then slide the shape from left to right, top to bottom and enable or disable pixels at the current position of the origin whenever there is any pixel match depending on the morphological operations.



> There are also Grayscale Morphological Operations, apart from Binary ones.

Connected Components Analysis

- Connected Component Analysis checks each pixel of an image for connectivity with it's neighboring pixels
- Each group of connected pixels are considered as one component and are assigned to the same label
- Connectivity is established if two neighboring pixels share same or similar intensity/color value.
- The method works on binary, grayscale, or color images.
- Different measures of connectivity arone possible (4-connectivity, or 8-connectivity are typical)