Introduction to Computer Vision

Computer vision is a multidisciplinary field of artificial intelligence and computer science that focuses on enabling computers to interpret and understand visual information from the world, similar to how humans perceive and comprehend their surroundings through vision. It involves the development of algorithms and techniques to extract meaningful insights, features, and knowledge from images or video data. Computer vision has a wide range of applications across various industries, including healthcare, automotive, robotics, entertainment, agriculture, and more.

Vision:

The term Vision is something we use in our daily life to see our surrounding. Vision provides the ability to see. It is one of the important sensory powers of human beings. Vision helps to create images of the surrounding environment in our brains. The interpretation of the visual images helps us to navigate our surroundings and better understand the world we live in.

Figure 1: Figure showing different objects

How machines interpret images and videos?

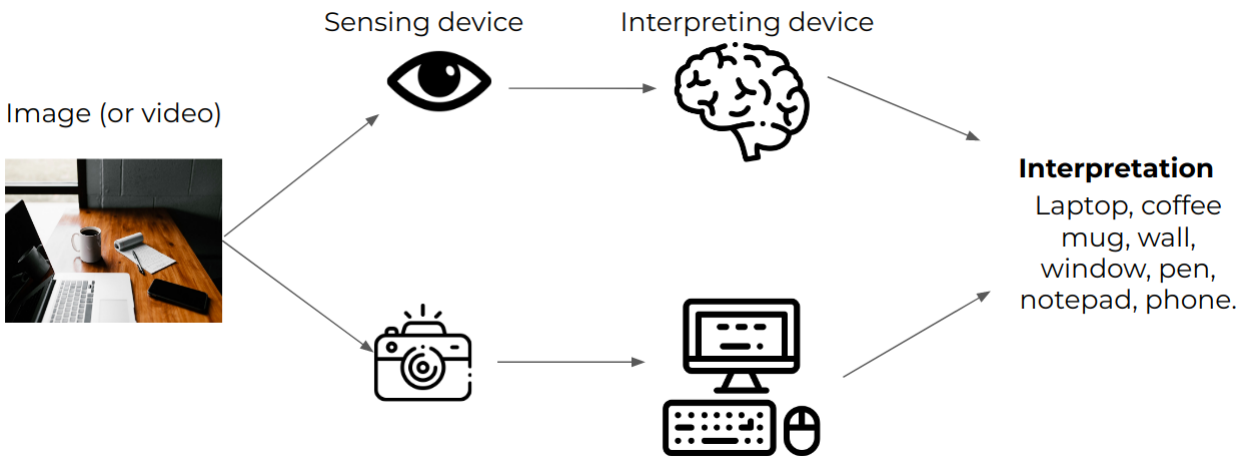


Figure 2: Natural and Artificial vision

In the figure above, we can see that in the case of human vision, we perceive the real world through our eye, and the brain carries complex action, processing, and interpreting visual data to give perception and formulate memories. While in the case of CV, you see that camera perceives the real world, and computer performs the processing and interpreting visual data to give perception and formulate memories.

In computer science, the computer sees the images as the nD-array of pixel values. We know videos are the sequence of images stacked together to form a moving picture, and hence computer also sees it as it sees the images. This is the difference between our human eyes and computer vision, where we see things and observe different features of them, but the computer only sees the pixel values, as shown in the figure below.

Pixels are the smallest unit of the digital image. Pixel value represents the color value. It ranges from 0 to 255. Through these pixel values, computers/machines try to understand the image provided to them.

One of the major goals of CV is to bridge the gap between the pixels and meaning. It tries to create meaningful information from the nD-array of pixels values

Image Processing Vs Computer Vision

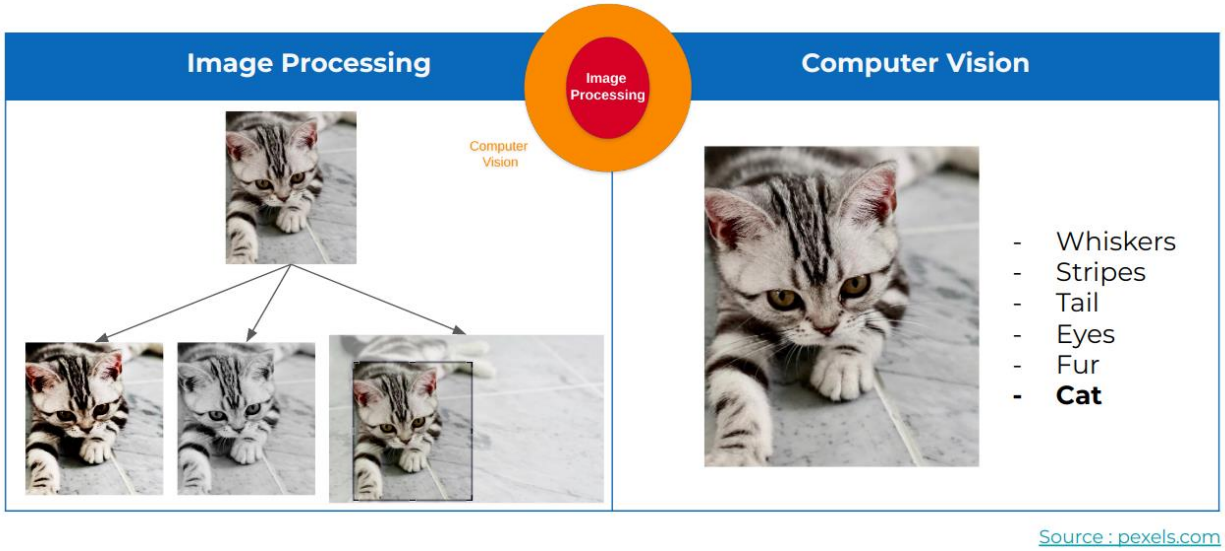


Figure: Image Processing vs Computer Vision

Image processing is, as its name implies, all about the processing of images. It takes an image as its input, processes them, and gives an out that is processed image.

From above, we can conclude that image processing divides into three steps:

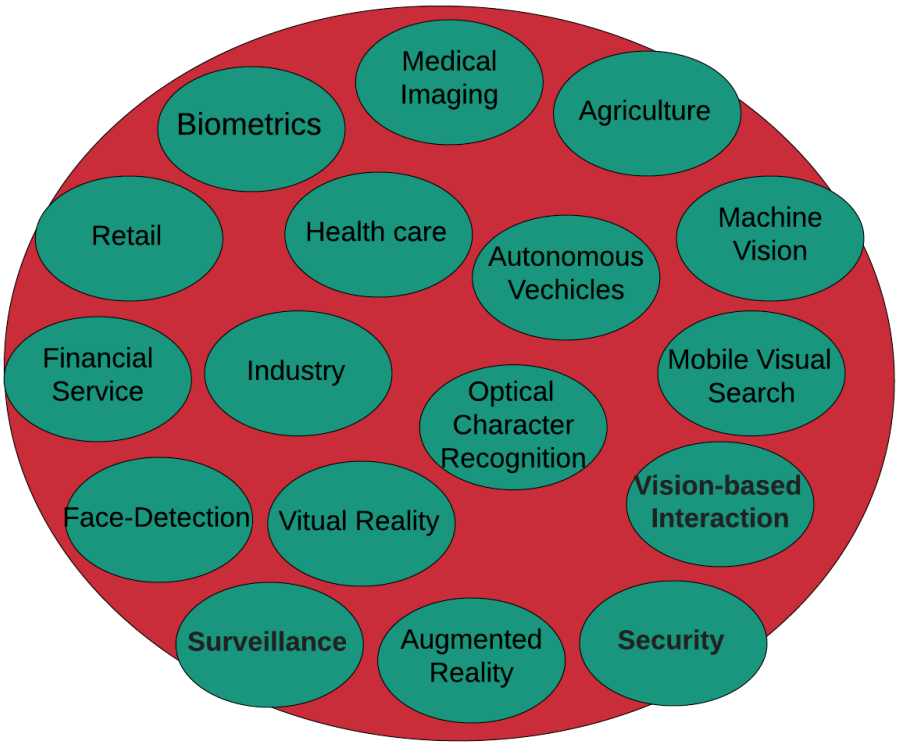
- importing the image via image acquisition tools
- Analyzing and manipulating images
- output the transformed image or report based on image analysis.

In image processing, we use two methods analog (hard copies like printouts, photographs) and digital (manipulation of images by computers) image processing. In digital image processing, we find that images go through three general phases, namely, pre-processing, enhancement, and display, information extraction. Some of the image processing methods are image filtering, noise removal, color processing, edge detection, and so on.

Image processing is a subset of computer vision, as shown in the figure above. A computer vision uses image processing algorithms to solve the computer vision problems like classification, recognition etc.

Why computer Vision?

Nowadays, in the streets, public places, smartphones, home, we can find the cameras everywhere. People are uploading the millions of images and videos in a day on the internet through different mediums like Instagram, Tik-Tok, YouTube, Vimeo, Facebook, and so on. It would help if we had a mechanism to deal with these images and videos and process them and carry out useful information from them. This mechanism is computer vision.



Computer Vision Applications

CV is a multi-disciplinary field. We can find its applications in different areas like Medical, Surveillance, Security, Virtual Reality, and many more. Nowadays, we find that field of CV is growing rapidly. It adapts itself to industries like Vehicles, Retail, Oil, and so on in different ways.

OpenCV:

OpenCV (**Open-Source Computer Vision Library**) is a popular open-source computer vision and image processing library that is widely used in the field of computer vision and machine learning. It provides a wide range of tools and functions for tasks such as image and video manipulation, object detection, image segmentation, face recognition, and more. OpenCV was built for maximum efficiency and performance of computing-intensive vision tasks. Therefore, it has a strong focus on real-time applications of AI vision.

Features and possible operations

It has a modular structure, i.e., its packages include several shared or static libraries. The modules available in the CV are as follows:

- **Core functionality (core):**

  It is a compact module defining basic data structure and dense multi-dimensional array and basic functions used by all other modules.

- **Image Processing(imgproc)**:

  It is an image processing module that includes linear and non-linear image filtering, geometrical image transformation, color space conversion, histograms, etc.

- **Video Analysis(video):**

  It is a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.

- **Camera Calibration and 3D Reconstruction(calib3d):**

  It includes basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and 3D reconstruction elements.

- **2D Features Framework(features2d):**

  It includes salient feature detectors, descriptors, and descriptor matchers.

- **Object Detection (obj detect):**

  It includes the detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).

- **High-level GUI (high Gui)**

  It is an easy-to-use interface to simple UI capabilities.

- **Video I/O (Video IO):**

  It is an easy-to-use interface for video capturing and video codecs.

Applications of OpenCV:

There are lots of applications which are solved using OpenCV, some of them are listed below

- Face recognition
- Automated inspection and surveillance
- Number of people – count
- Vehicle counting on highways along with their speeds
- Street view image stitching
- Video/image search and retrieval
- Robot and driver-less car navigation and control

OpenCV Functionality:

- Image/video I/O, processing
- Object/feature detection

- Geometry-based monocular or stereo computer vision
- Computational photography
- Machine learning & clustering
- CUDA acceleration

Reading an image in OpenCV using Python:

To read and display an image in OpenCV (Open-Source Computer Vision Library), we will need to follow these steps in Python:

1. **Import the necessary libraries**: First, we need to import the OpenCV library.

   ```
   Import cv2
   ```

2. **Read the image:** Using the cv2.imread() function to read the image from a file. We need to specify the file path as an argument.

   ```
   image = cv2.imread('path_to_your_image.jpg')
   ```

3. **Check if the image was read successfully**: It's a good practice to check whether the image was loaded correctly. If the file path is invalid or the image format is not supported, image will be None. We can use a simple conditional statement to check this:

   ```
   if image is None:
     print ("Image not found or invalid format")
     exit ()
   ```

4. **Display the image**: We can use the cv2.imshow() function to display the image in a window. This function takes two arguments: a window name (a string) and the image you want to display.

   ```
   cv2.imshow('Image', image)
   ```

Here's an example below:

```
import cv2

import matplotlib. pyplot as plt



# Specify the path to the image we want to read

image_path = 'path_to_your_image.jpg'

# Using the cv2.imread() function to read the image

# The below argument specifies how the image should be read:

# cv2.IMREAD_COLOR: Loads a color image (default)

# cv2.IMREAD_GRAYSCALE: Loads the image in grayscale

# cv2.IMREAD_UNCHANGED: Loads the image as is (including alpha/transparency channels)

image = cv2.imread(image_path, cv2.IMREAD_COLOR)

# Check if the image was successfully loaded

if image is not None:

    # Display the image using matplotlib

    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    plt.show()

else:

    print("Failed to load the image.")
```

Image Processing Technique

- Image processing is a method to perform operations on an image to extract information from it or enhance it.
- Image processing techniques play a crucial role in various machine learning projects, particularly in computer vision applications.
- Digital image processing has a broad range of applications such as image restoration, medical imaging, remote sensing, image segmentation, etc. Every process requires a different technique.
- In general, Image processing may include: Importing the image via image acquisition tools; Analyzing and manipulating the image; Output in which results are altered based on image analysis, etc.

  Image processing basically includes the following three steps:

  1. Importing the image
  2. Analyzing and manipulating the image
  3. Output in which result can be altered image or report that is based on image analysis

Image Processing operator: A general image processing operator is a function that takes one or more input images and produces an output image.

The image transformation can be seen as:

* Point operators (pixel transforms)

* Neighbourhood (area-based) operators/ filters

Point Operators: The Point operators are a simple image manipulation method in image processing in which output pixel value depends only on the corresponding input value.

It Includes: Brightness and contrast adjustment.

Color correction and color transformation.

Brightness and contrast adjustment

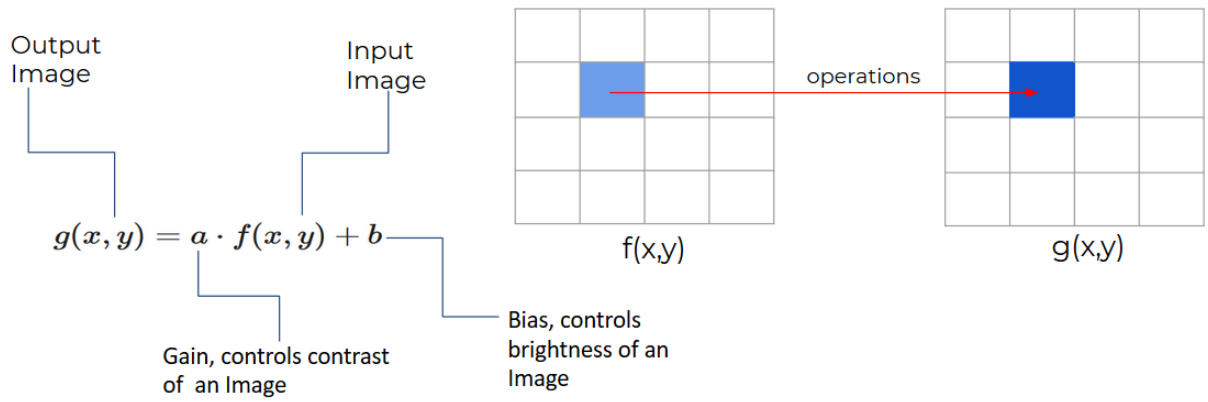


Figure 1: Brightness and Contrast adjustment using Point Operator

Image as a Matrix

Input Image: f (x, y)

Output Image: g (x, y)

where x and y are horizontal and vertical coordinates pointing to a pixel of an image.

The value of function **f (x, y)** gives the characteristic of the point on an image.

We can represent Images as a function too.

- Input Image: f(k)
- Output Image: g(k)

Where **K= (x, y)** we can change value of x and y because they are horizontal and vertical co-ordinates that gives pixel intensity value.

The brightness and Contrast adjustment using point operator can be represented as:

g (k) = a. f(k) + b

Where,

- a is gain and a>0 and controls contrast,
- b is bias and controls brightness

We can think of f(k) as the source image pixels and g(k) as the output image pixels. Then, more conveniently we can write the expression as

g (x, y) = a· f (x, y) + b

where x and y indicate that the pixel is located in the x-th row and y-th column.

To selectively darken the sky or when modeling vignetting in an optical system you can spatially change a and b.

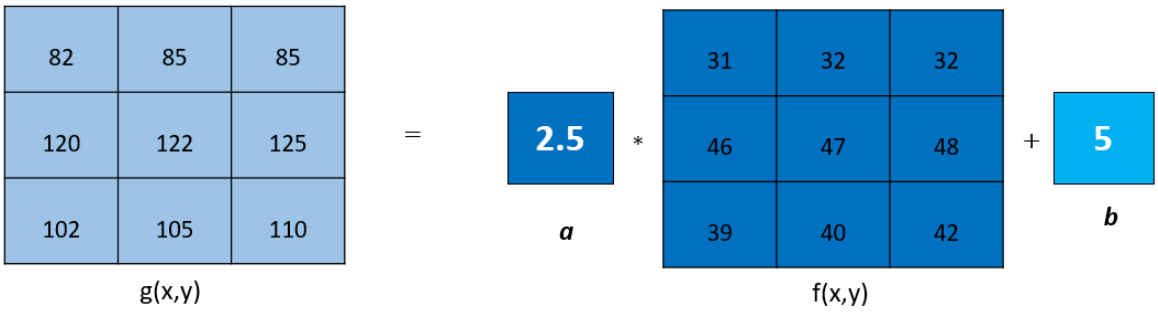


Figure 2: Working/Math of Point Operators

The value of a and b is broadcasted and the point operation is applied as shown in the figure.

- If b>0 Brightness increases else Brightness decreases.
- If a>1 contrast is increased.
- If a<1 contrast is reduced
- If a is negative dark area becomes light and light area becomes dark

Example: g (x, y) = a· f (x, y) + b

1. Image Restoration:

- Image restoration is a process in image processing and computer vision that aims to enhance or restore the quality of a degraded or damaged image.
- Image restoration in image processing refers to the process of enhancing the quality and clarity of an image by reversing or mitigating the effects of various forms of degradation that can occur during image acquisition, transmission, or storage.
- Image restoration is the operation of taking a corrupt/noisy image and estimating the clean, original image.
- In summary, image restoration is a vital field in image processing that aims to improve image quality, making images more suitable for analysis, interpretation, and visualization in a wide range of applications. It involves a combination of mathematical models, filters, and algorithms to recover or enhance the original information contained in a degraded or noisy image.

2. Linear filtering:

- Linear filtering is a fundamental image processing technique used to enhance or extract specific features from an image.
- It operates on the principle of applying a convolution kernel (also known as a filter or mask) to each pixel in the image.
- This kernel defines a weighted average of the pixel values in the local neighborhood of each pixel, resulting in a new pixel value in the output image.
- Linear filtering is widely used for tasks such as blurring, sharpening, edge detection, and noise reduction.

3. Independent Component Analysis:

- Independent Component Analysis (ICA) is a powerful technique in image processing for separating mixed sources within images and extracting meaningful components.

- It is particularly useful when dealing with images where multiple sources are mixed together, and the goal is to uncover the original sources or components.

Basic Principle of ICA:

- ICA assumes that observed images are linear combinations of independent sources. These sources can be thought of as the underlying components that make up an image.
- The key idea is to find a transformation that separates these mixed sources into their original, statistically independent components.

4. Pixelation:

- Pixelation is a technique used to blur or hide sensitive information in images by reducing the level of detail and replacing it with larger, blocky pixels.
- Pixelation occurs when resizing of the images are enlarged to a point where individual pixels can be observed or pixels stretch to the point beyond their original size.

5. Template matching:

- Template matching is a technique used in computer vision to find a sub-image (template) within a larger image.
- OpenCV provides a built-in function called **cv2.matchTemplate()** for performing template matching.
- In template matching, we slide the template image over the larger image as we do in the convolution process and find the matching part.

Introduction to Noise

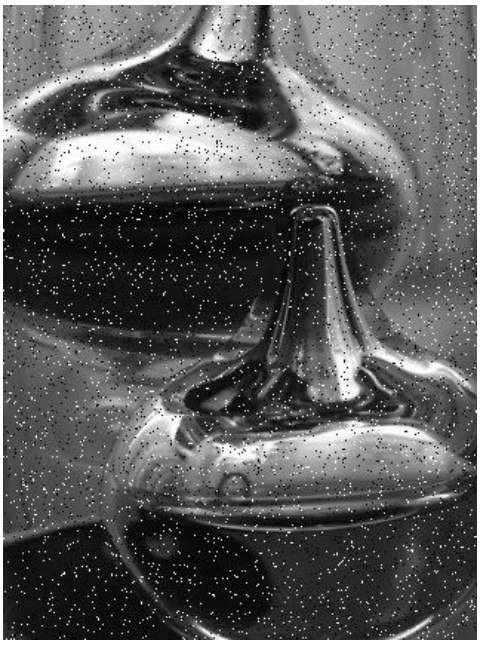


Figure: Noisy Image

Noise is a grainy film in a photograph, hiding details and making the picture look considerably worse. Images can be so noisy in some situations that they are unusable. Technically, there will always be some amount of noise in any picture. There is nothing to keep this from happening; it is a physical property of light and photography. It is a degradation in the image caused by external interference.

Images are corrupted during transmission

Images are corrupted during transmission principally by interference in the transmission. One of the most common noise types which corrupt images during transmission are impulse noise, also known as salt & pepper noise.

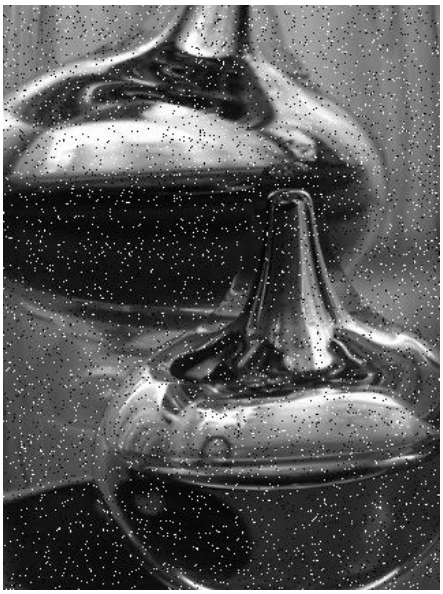


Figure: Image with salt and pepper noise

Models of Noise based on nature

Based on the nature of noise, there are two types of noise:

1. Additive noise
2. Multiplicative noise

Additive noise

$$g(x, y) = f(x, y) + \eta(x, y)$$

In additive noise, noise is added to the original Image. One typical example of additive noise is Gaussian noise.

Gaussian noise

Gaussian noise is the statistical noise having a probability distribution function equal to Gaussian distribution or normal distribution. The PDF of a Gaussian random variable Z is given by

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(z-\bar{z})^2}{2\sigma^2}}, -\infty < z < \infty$$

where,

- z represents intensity,
- $\bar{z}$ is the mean (average) value of z ,
- σ is the standard deviation.

Gaussian noise is the most commonly used noise model in image processing because it effectively describes the most random noise encountered in the image-processing pipeline. This type of noise is also known as additive noise. Gaussian noise is caused by natural sources such as the thermal vibration of atoms and the discrete nature of warm objects' radiation.

Denoising

An image may contain noise. Reducing noise in an image is a primary goal in Image processing. It produces meaningful Information. Reducing noise from an Image is called denoising. In denoising, loss of original features should be minimum. Denoising could inevitably lose some details.

Multiplicative noise

g (x, y) =f (x, y). η (x, y)

In Multiplicative noise, noise multiplies to the original Image. One typical example of Multiplicative noise is Speckle noise.

Image Thresholding

- Image thresholding is a simple form of image segmentation.
- It is a way to create a binary image from a grayscale or full-color image.
- Image Thresholding is an intensity transformation function in which the values of pixels below a particular threshold are reduced, and the values above that threshold are boosted.

Introduction to thresholds:

- Thresholding is a type of Image Segmentation in which we segment the image into the foreground and background. In the foreground, you have an object of interest, and in the background, you have everything except an object of interest. Separating the regions corresponding to the object of interest and background is useful in many computer vision applications. Binary masks for an image are generated using the thresholding process.
- Thresholding often provides an easy and convenient way to perform this segmentation based on the different intensities or colors in the foreground and background regions of an image. If an image can be segmented correctly or not, is determined by looking at the Image's intensity histogram
- In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255).
- Thresholding is a very popular segmentation technique, used for separating an object considered as a foreground from its background.
- A threshold is a value which has two regions on its either side i.e., below the threshold or above the threshold.

- In OpenCV with Python, the function cv2.threshold is used for thresholding.

  Syntax: cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)

  Parameters:

  -> source: Input Image array (must be in Grayscale).

  -> thresholdValue: Value of Threshold below and above which pixel values will change accordingly.

  -> maxVal: Maximum value that can be assigned to a pixel.

  -> thresholdingTechnique: The type of thresholding to be applied.

Simple Thresholding:

The basic Thresholding technique is Binary Thresholding. 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 different Simple Thresholding Techniques are:

- cv2.THRESH_BINARY: If pixel intensity is greater than the set threshold, value set to 255, else set to 0 (black).
- cv2.THRESH_BINARY_INV: Inverted or Opposite case of cv2.THRESH_BINARY.
- cv.THRESH_TRUNC: If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same.
- cv.THRESH_TOZERO: Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.
- cv.THRESH_TOZERO_INV: Inverted or Opposite case of cv2.THRESH_TOZERO.

Binary

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Inverted Binary

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

Truncated

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

To Zero

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

To Zero Inverted

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Adaptive Thresholding:

Adaptive thresholding is the method where the threshold value is calculated for smaller regions. This leads to different threshold values for different regions with respect to the change in lighting. We use cv2.adaptiveThreshold for this.

Syntax: cv2.adaptiveThreshold(source, maximum_value, adaptive_method, threshold_type, block_size, constant)

Parameters:

- source: Input Image array (Single-channel, 8-bit or floating-point)
- maximum_value: Maximum value that can be assigned to a pixel.
- adaptive_method: Adaptive method decides how threshold value is calculated.
- threshold_type: The type of thresholding to be applied.
- block_size: Size of a pixel neighborhood that is used to calculate a threshold value.
- constant: A constant value that is subtracted from the mean or weighted sum of the neighbourhood pixels.

cv2.ADAPTIVE_THRESH_MEAN_C: Threshold Value = (Mean of the neighbourhood area values – constant value). In other words, it is the mean of the blockSize×blockSize neighborhood of a point minus constant.

cv2.ADAPTIVE_THRESH_GAUSSIAN_C: Threshold Value = (Gaussian-weighted sum of the neighbourhood values – constant value). In other words, it is a weighted sum of the blockSize×blockSize neighborhood of a point minus constant.

Otsu Thresholding:

In Otsu Thresholding, a value of the threshold isn't chosen but is determined automatically. A bimodal image (two distinct image values) is considered. The histogram generated contains two peaks. So, a generic condition would be to choose a threshold value that lies in the middle of both the histogram peak values.

We use the Traditional cv2.threshold function and use cv2.THRESH_OTSU as an extra flag.

Syntax: cv2.threshold(source, threshold_value, maximum_value, thresholding_technique)

Parameters:

- source: Input Image array (must be in Grayscale).
- threshold_value: Value of Threshold below and above which pixel values will change accordingly.
- maximum_value: Maximum value that can be assigned to a pixel.
- thresholding_technique: The type of thresholding to be applied.

How does Otsu's Binarization work?

This section demonstrates a Python implementation of Otsu's binarization to show how it actually works.

Since we are working with bimodal images, Otsu's algorithm tries to find a threshold value (t) which minimizes the weighted within-class variance given by the relation:

$$\sigma_{2w}(t)=q_1(t)\sigma_{21}(t)+q_2(t)\sigma_{22}(t)$$

where

$$q_1(t)=\sum_{i=1t}P(i)\&q_2(t)=\sum_{i=t+1I}P(i)$$

$$\mu_1(t)=\sum_{i=1t}iP(i)q_1(t)\&\mu_2(t)=\sum_{i=t+1I}iP(i)q_2(t)$$

$$\sigma_{21}(t)=\sum_{i=1t}[i-\mu_1(t)]_2P(i)q_1(t)\&\sigma_{22}(t)=\sum_{i=t+1I}[i-\mu_2(t)]_2P(i)q_2(t)$$

It actually finds a value of t which lies in between two peaks such that variances to both classes are minimal.

Image Filtering:

Image filtering is the process of modifying an image by changing its shades or color of the pixels. It is also used to increase the brightness and contrast.

Filtering is a common image processing technique used to enhance or modify the appearance of an image. OpenCV is a popular library for image processing in Python, and it provides various functions for image filtering. You can perform filtering operations on an image using convolution with a kernel (also known as a filter) to apply various effects like blurring, sharpening, edge detection, and more.

These are the most commonly used OpenCV image smoothing filters-

- Averaging Filter
- Gaussian Filter
- Median Filter
- Bilateral Filter

Averaging Filter:

An averaging filter, also known as a mean filter or box filter, is a simple image filter that smoothens an image by replacing each pixel's value with the average value of its neighboring pixels. This filter is commonly used for noise reduction and blurring.

The functions **cv2.blur()** and **cv2.boxFilter()** can be used to perform the averaging filter. Both functions smooth an image using the kernel.

Syntax of cv2.blur()

cv2.blur(image, ksize)

Here, the image is the image source, and ksize is the size of blurring kernel.

Syntax of cv2.boxFilter()

cv2.boxFilter(src, dst, depth, ksize, anchor, normalize, border_type)

Here, the src is the image source, dst is the destination image of the same size, and depth denotes the output image depth. The anchor denotes the anchor points. By default, it is at the kernel point (coordinate (-1,1)). The ksize is the size of blurring kernel and normalize is the flag which specifies whether the kernel should be normalized or not. The border_type is an integer object that represents the type of the border used.

Gaussian Filter:

The OpenCV Gaussian filtering provides the **cv2.GaussianBlur()** method to blur an image by using a Gaussian Kernel. Each pixel in an image gets multiplied by a Gaussian Kernel. It means, a Gaussian Kernel is a square array of pixels.

Syntax of Gaussian Filter

cv2.GaussianBlur(src, ksize, sigma_x, dst, sigma_y, border_type)

src - the input image,

ksize - Gaussian kernel size (width and height), the width and height can have different values and must be positive and odd,

sigma_x - Gaussian kernel standard deviation along the X-axis,

dst - output image,

sigma_y - Gaussian kernel standard deviation along the Y-axis,

border_type - image boundaries.

Median Filtering:

Python OpenCV provides the **cv2.medianBlur()** function to blur the image with a median kernel. This is a non-linear filtering technique. It is highly effective in removing salt-and-pepper noise. This takes the median of all the pixels under the kernel area and replaces the central component with this median value. Since we are taking a middle, the output image will have no new pixel esteems other than that in the input image.

Syntax of Median Filter

cv2.medianBlur(image, ksize)

Here, the image represents the image for operation. The ksize is a size object representing the size of the kernel.

Bilateral Filter

Python OpenCV provides the **cv2.bilateralFilter()** function to blur the image with a bilateral filter. This function can be applied to reduce noise while keeping the edges sharp.

Syntax of Bilateral Filter

cv2.bilateralFilter(image, dst, d, sigma_color, sigma_space)

image- image source,

dst- destination image,

d- specifies the diameter of the pixel neighbourhood,

sigma_color- specifies the filter sigma in the color space.

Sigma_space- specifies the filter sigma in the coordinate space.

Edge Detection Using OpenCV

Edge detection is one of the fundamental image-processing tasks used in various Computer Vision tasks to identify the boundary or sharp changes in the pixel intensity. It plays a crucial role in object detection, image segmentation and feature extraction from the image. In Real-time edge detection, the image frame coming from a live webcam or video is continuously captured and each frame is processed by edge detection algorithms which identify and highlight these edges continuously.

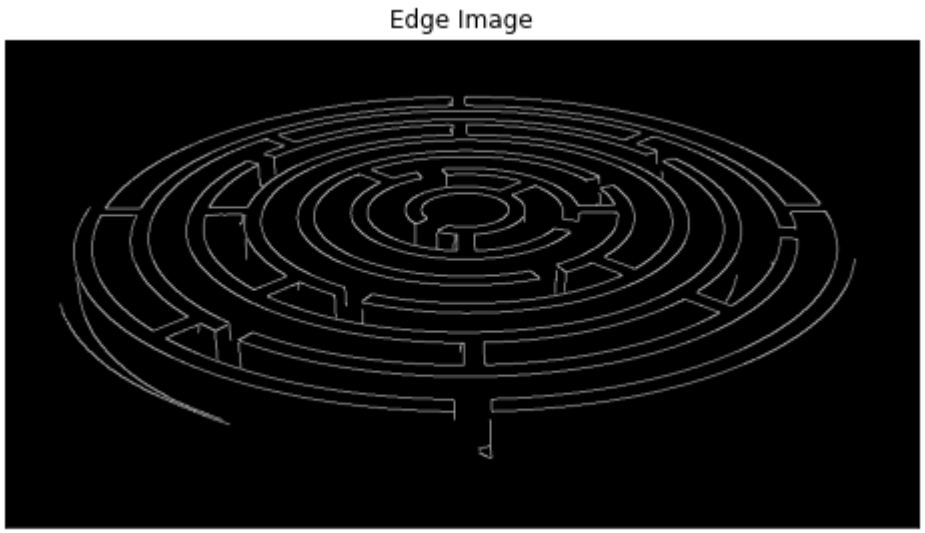


Fig: Output image after edge detection

A computer cannot visualize an image the way a human does. Each image is interpreted as an array of pixel values. Each pixel value can range from 0 to 255, representing black to white pixels, respectively. On this basis, the computer interprets an image in various ways.

The feature detection technique is one of the major approaches used for recognition that detects the most important features of the image. Some of the features are interest points, edges, lines, etc.

Among the many essential features of an image, as mentioned above, detecting an edge is crucial for various recognition tasks. Edges are the area where intensity between the pixels changes sharply. Although interest points carry an essential feature of an image, detecting interest points is not always a good strategy.

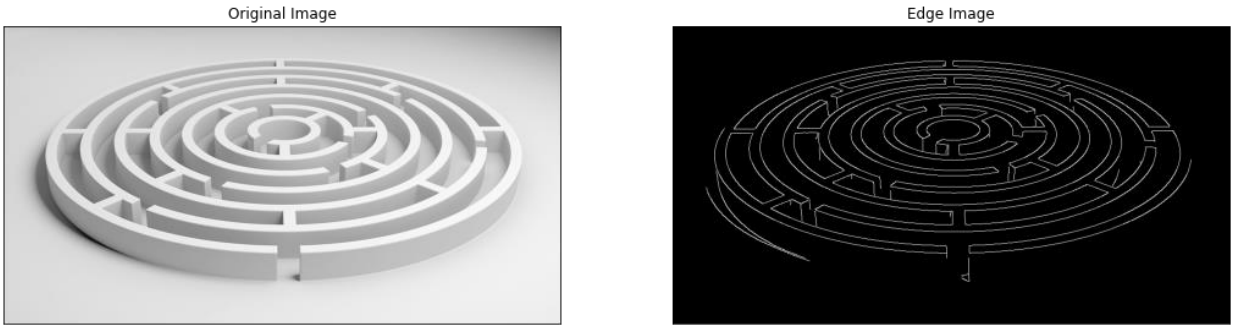


Fig.: (a) Original image (left) (b) Output image after edge detection (right)

Why to detect edges?

Detecting an edge plays very important role in various computer vision applications. Some of the purposes behind edge detection are,

- To create a line drawing of an image scene.
- To extract essential features of edges from an image (such as corners, curves, boundaries, lines, etc.).
- To generate features for various computer vision algorithms (such as object recognition)

Edges intensity modeling

Edges can be divided into different types based on their intensity, shown in the figure below.

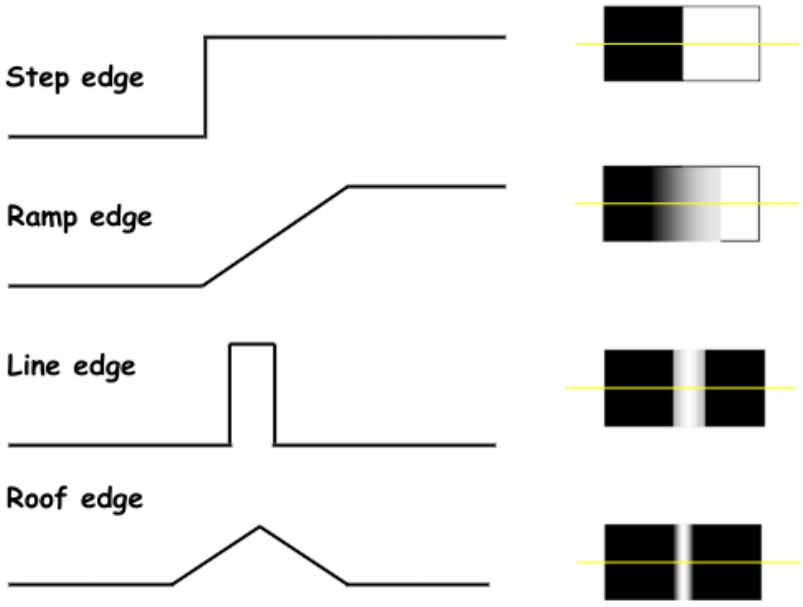


Fig.: Edge intensity profiles

1. Step edge: In step edge, abrupt changes in intensity from one area of an image to another area takes place.
2. Ramp edge: It is a kind of step edge where image intensity doesn't change suddenly but remains for a certain period.
3. Line edge: It is same as ramp edge but intensity changing period will be shorter than the ramp edge.
4. Roof edge: It is a kind of Line edge where image intensity doesn't change suddenly and occur very short distance.

Algorithm for edge detection

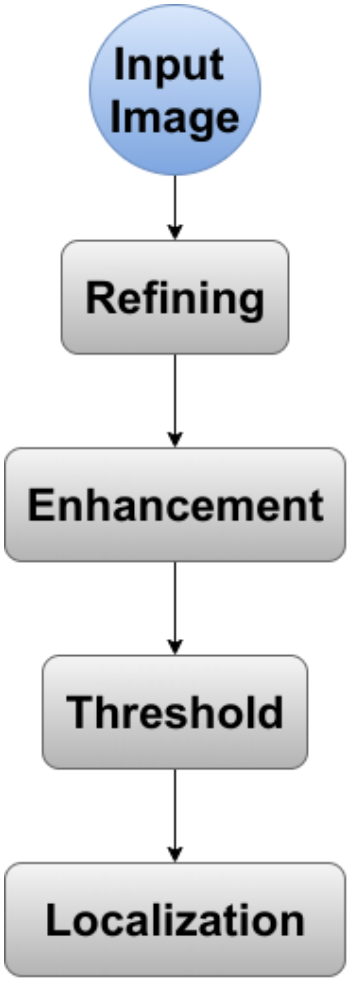


Fig: Edge detection algorithm

1. **Take an image:** First of all, the colored image should be taken for detection.
2. **Refining:** The purpose of refining is to remove the noises from the image without eliminating true edges.
3. **Enhancement:** In this approach, various filters or differentiation techniques are applied to enhance the edges' quality.

4. **Threshold:** In this step, certain threshold is set to eliminate noise and retain the pixels other than noise.
5. **Localization:** After identifying the true edge pixels we have to precisely locate these pixels in the edge boundary. The same thing is done in this localization step. For eg., Edge thinning, Edge linking, etc.

Types of Edge Detection

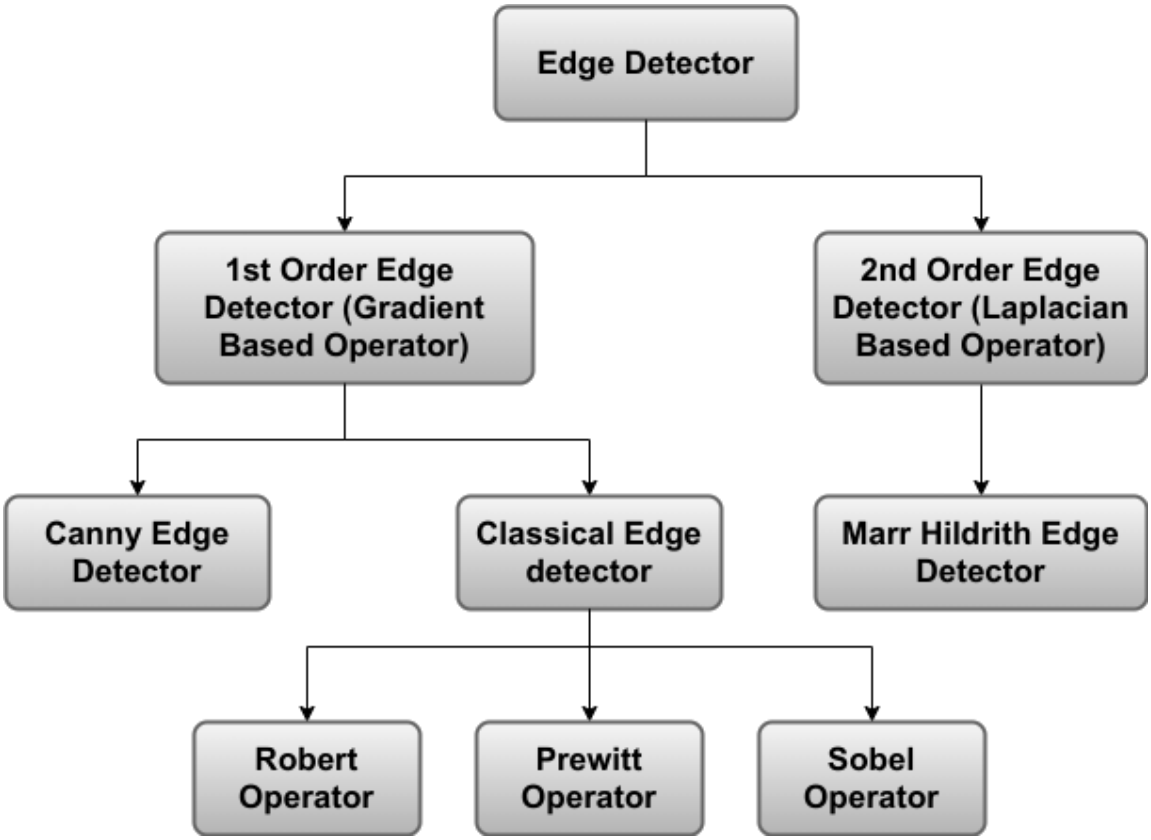


Fig: Edge Detection Types

Types of edge detection can be broadly divided into two major categories.

- First-order edge detector (Gradient-based operator)
- Second-order edge detector (Laplacian based operator)

Classical Edge Detector

Classical edge detector falls under the first-order edge detector category, which is also known as a gradient-based operator. The following three operators, Robert, Prewitt, and Sobel, are classified as classical operators. Although they are simple and easy to use, they are sensitive to the noise.

Sobel Edge Detection

Sobel Edge Detection is one of the most widely used algorithms for edge detection. Sobel edge detector is a gradient based method based on the first order derivatives. It calculates the first derivatives of the image separately for the X and Y axes.

The operator uses two 3X3 kernels which are convolved with the original image to calculate approximations of the derivatives- one for horizontal changes, and one for vertical.

These are the kernels used for Sobel Edge Detection:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad (1)$$

X-Direction Kernel

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (2)$$

Y-Direction Kernel

When these kernels are convolved with the original image, you get a 'Sobel edge image'.

- If we use only the Vertical Kernel, the convolution yields a Sobel image, with edges enhanced in the X-direction.

- Using the Horizontal Kernel yields a Sobel image, with edges enhanced in the Y-direction.

Let G_x and G_y represents the intensity gradient in the x and y directions respectively. If A and B denote the X and Y kernels defined above:

$$G_x = A * I \tag{3}$$

$$G_y = B * I \tag{4}$$

where * denotes the convolution operator, and I represent the input image.

$$G = \sqrt{G_x^2 * G_y^2} \tag{5}$$

And the orientation of the gradient can then be approximated as:

$$\Theta = arctan(G_y/G_x) \tag{6}$$

The following is the syntax for applying Sobel edge detection using OpenCV:

Sobel (src, ddepth, dx, dy)

The parameter ddepth specifies the precision of the output image, while dx and dy specify the order of the derivative in each direction. For example:

If dx=1 and dy=0, we compute the 1st derivative Sobel image in the x-direction.

If both dx=1 and dy=1, we compute the 1st derivative Sobel image in both directions

Canny Edge Detection:

Canny Edge Detection is one of the most popular edge-detection methods in use today because it is so robust and flexible. The algorithm itself follows a three-stage process for extracting edges from an image. Add to it image blurring, a necessary preprocessing step to reduce noise. This makes it a four-stage process, which includes:

- Reduce Noise using Gaussian Smoothing.
- Compute image gradient using Sobel filter.

- Apply Non-Max Suppression or NMS to just jeep the local maxima
- Finally, apply Hysteresis thresholding which that 2-threshold values T_upper and T_lower which is used in the Canny () function.

Canny Edge Detection algorithm

The algorithm consists of 5 steps. It only works once the image is converted to grayscale one. So, grayscale conversion is not included in this algorithm. In the figure below, input image refers to the grayscale image.

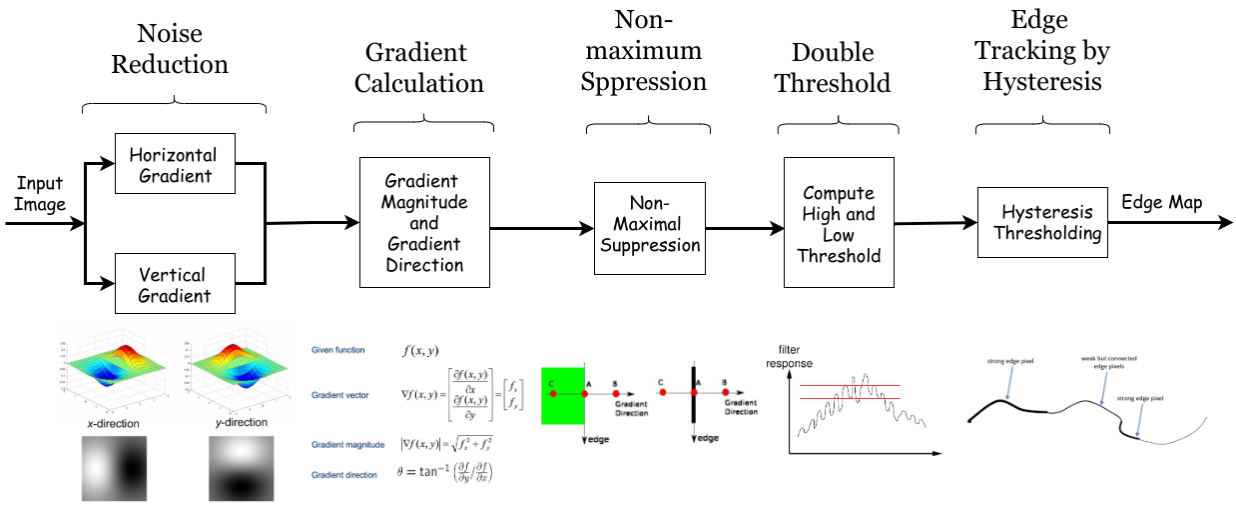


Fig: Diagram of Canny Edge Detector

Noise Reduction

Raw image pixels can often lead to noisy edges, so it is essential to reduce noise before computing edges. In Canny Edge Detection, a Gaussian blur filter is used to essentially remove or minimize unnecessary detail that could lead to undesirable edges

Calculating the Intensity Gradient of the Image

Once the image has been smoothed (blurred), it is filtered with a Sobel kernel, both horizontally and vertically. The results from these filtering operations are then used to calculate both the intensity gradient magnitude (G), and the direction (Θ)for each pixel, as shown below.

$$G = \sqrt{G_x^2 + G_y^2} \tag{7}$$

$$\Theta = tan^{-1}\left(\frac{G_x}{G_y}\right) \tag{8}$$

The gradient direction is then rounded to the nearest 45-degree angle.

Suppression of False Edges

After reducing noise and calculating the intensity gradient, the algorithm in this step uses a technique called non-maximum suppression of edges to filter out unwanted pixels (which may not actually constitute an edge). To accomplish this, each pixel is compared to its neighboring pixels in the positive and negative gradient direction. If the gradient magnitude of the current pixel is greater than its neighboring pixels, it is left unchanged. Otherwise, the magnitude of the current pixel is set to zero.

Hysteresis Thresholding – Edge Detection Using OpenCV

In this final step of Canny Edge Detection, the gradient magnitudes are compared with two threshold values, one smaller than the other.

- If the gradient magnitude value is higher than the larger threshold value, those pixels are associated with solid edges and are included in the final edge map.
- If the gradient magnitude values are lower than the smaller threshold value, the pixels are suppressed and excluded from the final edge map.

- All the other pixels, whose gradient magnitudes fall between these two thresholds, are marked as ‘weak’ edges (i.e., they become candidates for being included in the final edge map).
- If the ‘weak’ pixels are connected to those associated with solid edges, they are also included in the final edge map.

The following is the syntax for applying Canny edge detection using OpenCV:

Canny (image, threshold1, threshold2)

OpenCV Morphological Operations

Morphological operations are image processing techniques used to manipulate the shape and structure of objects in an image. OpenCV, a popular computer vision library in Python, provides functions to perform various morphological operations, including erosion, dilation, opening, closing, and more. These operations are often used in tasks such as noise reduction, object segmentation, and feature extraction.

Morphological operators are mainly based on the mathematical concepts from set-theory. These operators are useful for the analysis of binary images.

Morphological operators include the set theory concept like Subset (A⊆B), Union(A∪B), Intersection (A∩B), Complement(A^C), difference (A−B), reflection, and translation.

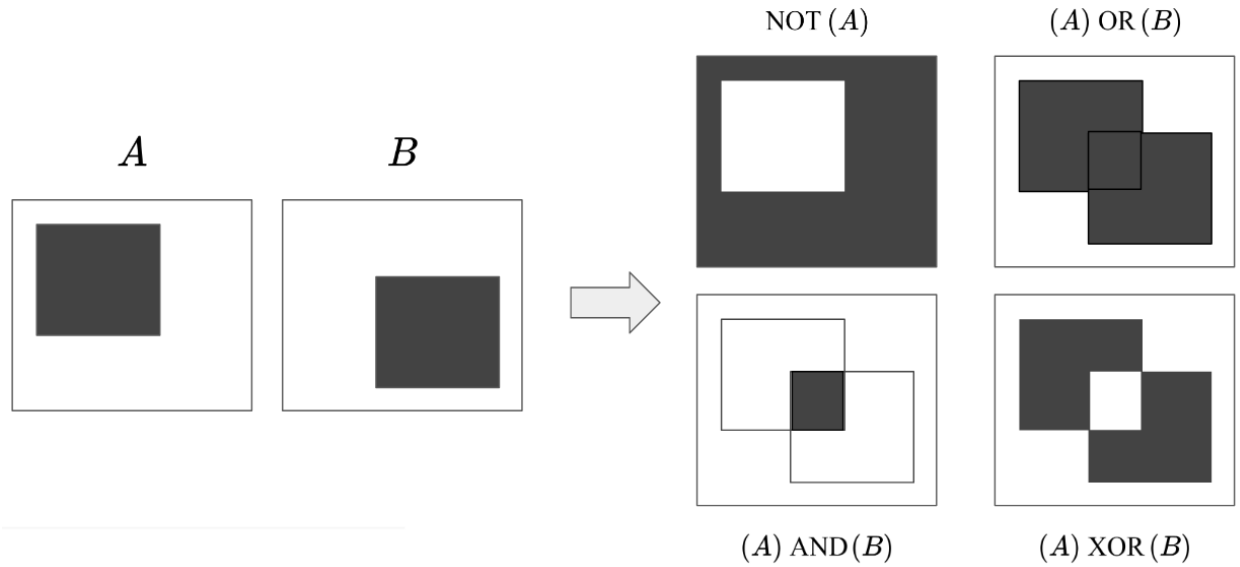


Figure: Morphological operations with logical operations

Morphological operators also include the logical operators like AND, OR, NOT, XOR, NAND for performing the logical operations on a pixel-by-pixel basis between corresponding pixels (bitwise

operations). Here, we know that logical operations are just a private case for binary set operations such as Intersection (AND), Union (OR), Complement (NOT). The above figure shows some basic morphological operations with logical operators like NOT, AND, OR, XOR. In the figure, A and B represents two binary images and we can see how different operations change the output images.

Structuring Elements:

As you know that, the morphological operations take two inputs: one is the image object, and the other is a structuring element, which we combine using morphological operators to produce the output.

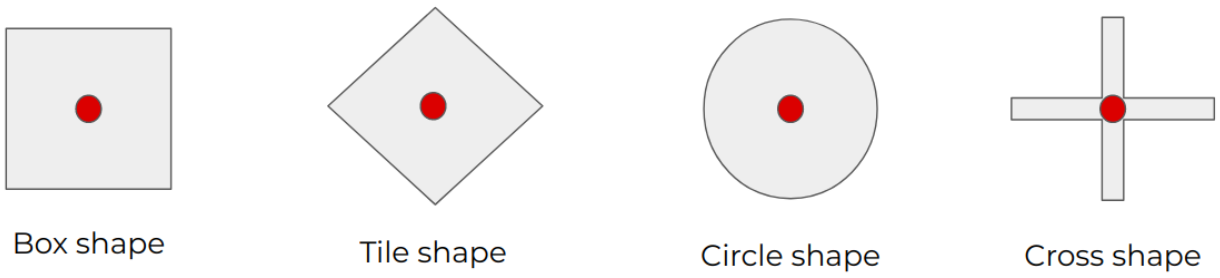


Figure: Structuring elements with various shapes

Structuring elements are the small shape or template that morphological techniques use to probe the images. The above figure shows various structuring elements such as box, tile, circle, and cross with small red circle as an origin. More specifically, it is the matrix of pixels, each with a value of 0 or 1. The matrix dimensions determine the size of the structuring element, while the pattern of ones and zeros determines its shape. Here, in the example below, you can see three 5×5

structuring elements with shapes like square-shape, diamond-shape, and cross-shaped. Among the structuring elements, one of its elements represents the origin of the structuring element - shown by red box, as shown in figure below. This origin can also be outside the structuring element.

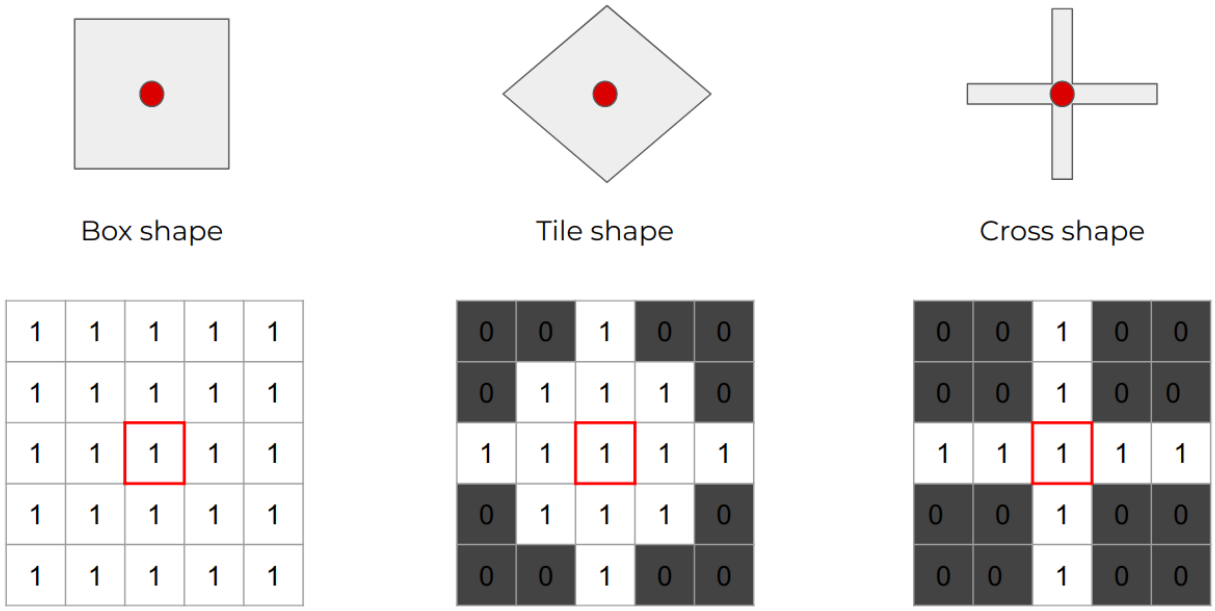


Figure: Structuring element with matrix representation

Morphological operations based on OpenCV are as follows:

- Erosion
- Dilation
- Opening
- Closing
- Morphological Gradient
- Top hat
- Black hat

Erosion:

Erosion primarily involves eroding the outer surface (the foreground) of the image. As binary images only contain two pixels 0 and 255, it primarily involves eroding the foreground of the image and it is suggested to have the foreground as white. The thickness of erosion depends on the size and shape of the defined kernel.

Let's see the mathematical definition of the erosion. For this, let us consider a set A that contains the foreground pixels. If B is a structuring element, and if there is a pixel z, which translates B into another position such that B is in A, then the set containing all such z is the eroded representation of A. Mathematically we can represent this as:

$$A \ominus B = \{z | (B)_z \subseteq A\}$$

In the case of image, I there are foreground pixels and background pixels (0's or black pixels). Therefore, inputs and outputs are images, and we can represent it as:

$$I \ominus B = \{z | (B)z \subseteq A \text{ and } A \subseteq I\} \cup \{Ac | Ac \subseteq I\}$$

This means that what we perform erosion on foreground pixels and combine all the background pixels, results in a complete image.

Working Process

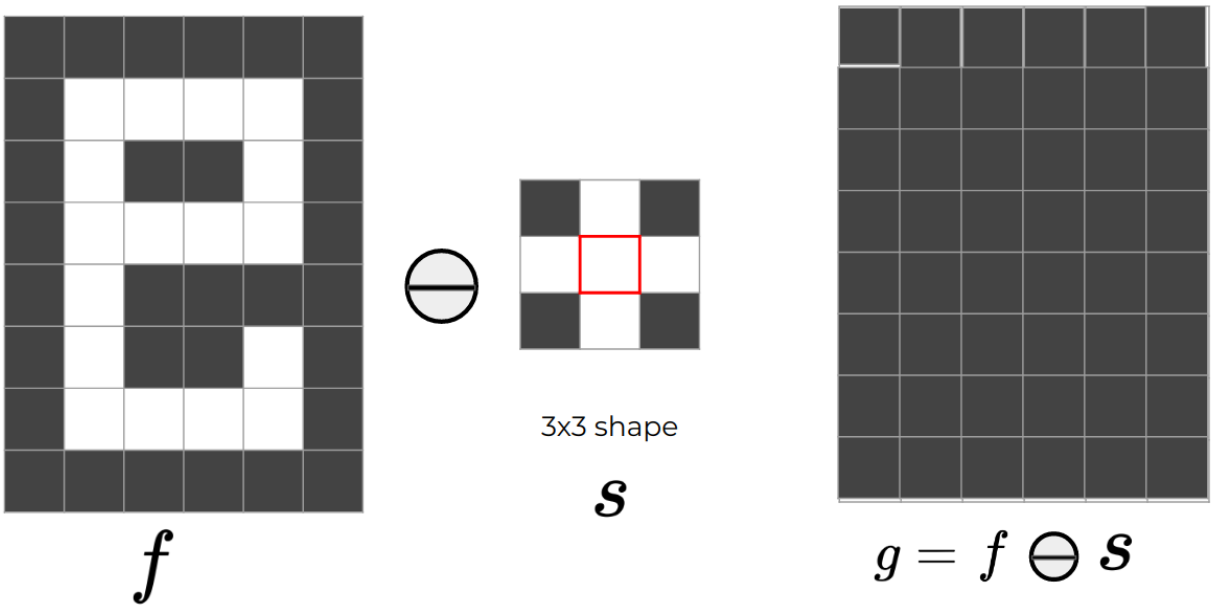


Figure: Structuring element with matrix representation

Here, the question arises on how we can do this calculation. The erosion operation performed above is straightforward. It follows the fit condition. For gentle we will be able to complete the erosion operation like this,

1. We need to overlap each pixel of the image with the origin of the structuring element and slide it through each pixel.
2. We need to see if the overlapping of the structuring element perfectly matches or not with the area that you overlap it. If it perfectly matches, the pixel value becomes 1 else 0.

In the above example, no pixels of the image matrix overlap perfectly with the 3x3 cross shaped structuring element matrix and that's why output image matrix contains all black pixels. Let's see this with another 5x5 structuring element.

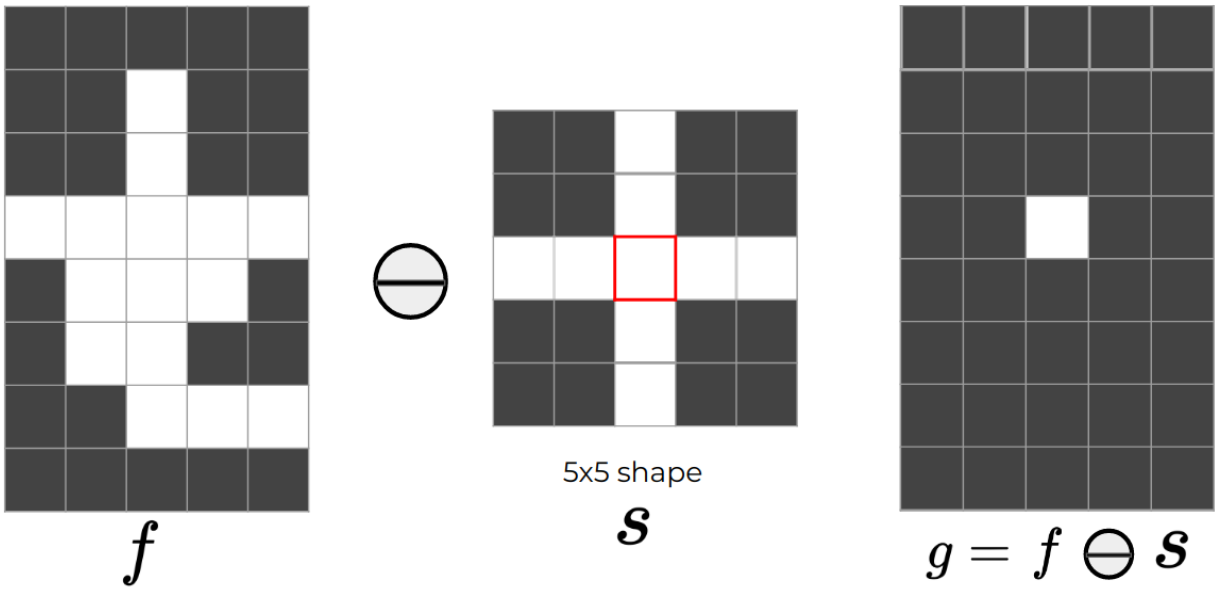


Figure: Structuring element with matrix representation

Here, we can see when structuring element's origin placed on that position it perfectly fits with the image pixels and gives 1 value and for all other position there is no perfect fits for 5x5 cross-shaped structuring element.

Properties

- Translation invariant

$$(A+x) \ominus B = (A \ominus B) + x = A \ominus (B + x)$$

Shifting in the image object or structuring element does not affect the erosion operation.

- Erosion is increasing

$$\text{If } A \subseteq C, A \ominus B \subseteq C \ominus B$$

where B is structuring element and A and C are image objects.

- Satisfy Decomposition theorem

$$A \ominus (B \cup C) = (A \ominus B) \cup (A \ominus C)$$

- Erosion also satisfies following condition:

$$(A \ominus B) \oplus C = A \ominus (B \oplus C)$$

where $\oplus$ represent dilation

Dilation

Dilation involves dilating the outer surface (the foreground) of the image. As binary images only contain two pixels 0 and 255, it primarily involves expanding the foreground of the image and it is suggested to have the foreground as white. The thickness of erosion depends on the size and shape of the defined kernel. We can make use of NumPy's ones () function to define a kernel.

Like in erosion, let's define it mathematically. If A is the set representing foreground pixels, and B is the structuring elements, we can represent dilation of A by B $(A \oplus B)$ as

$$A \oplus B = \{z | (B^)_z \cap A \neq \phi\}$$

This means if we reflect the element about its origin, and translate this reflection by pixel z, the set of all z consists of pixels where foreground elements of structure B overlap with at least one element of A.

Working Process

Similar to erosion, for explaining the working process of dilation in a binary image, we have an f as image objects and s as a structuring element, respectively, as shown below and perform the dilation operation.

Here, the calculation is same as erosion operation. Only difference is it follows the hit condition. We will be able to complete the dilation operation like this,

1. We need to overlap each pixel of the image with the origin of the structuring element and slide it through each pixel.
2. We need to see if at least one pixel of the structuring element activated by the white pixels of the image object. If this happens, the pixel value where origin is located becomes 1 else 0.

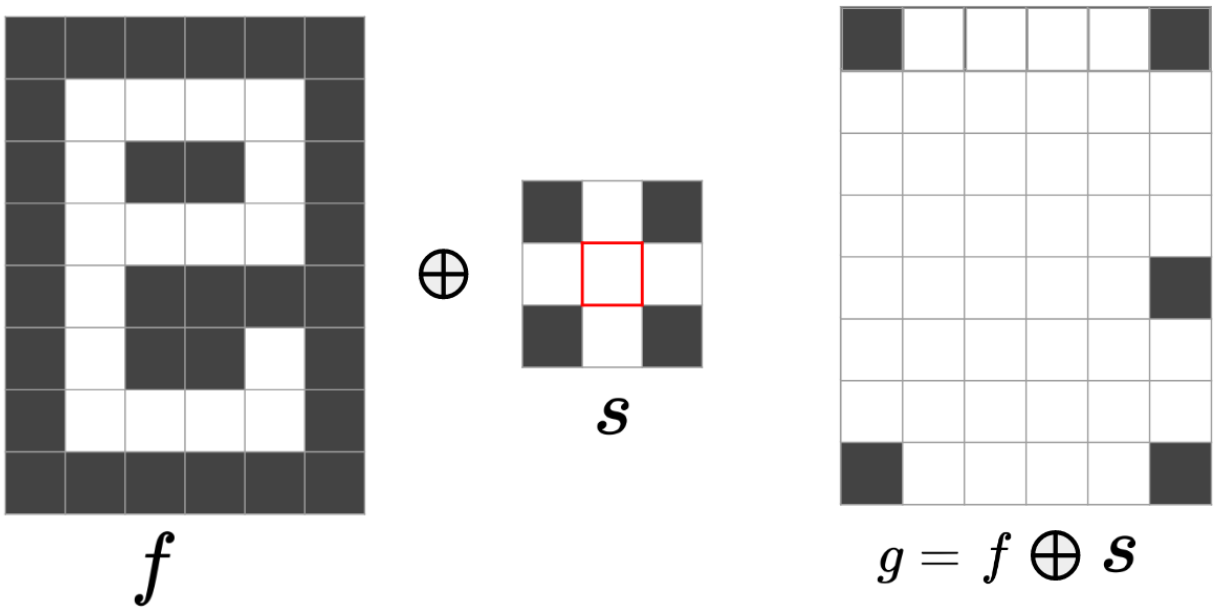


Figure 4. Structuring element with matrix representation

Let's see another example of dilating image object with 5×5 structuring element.

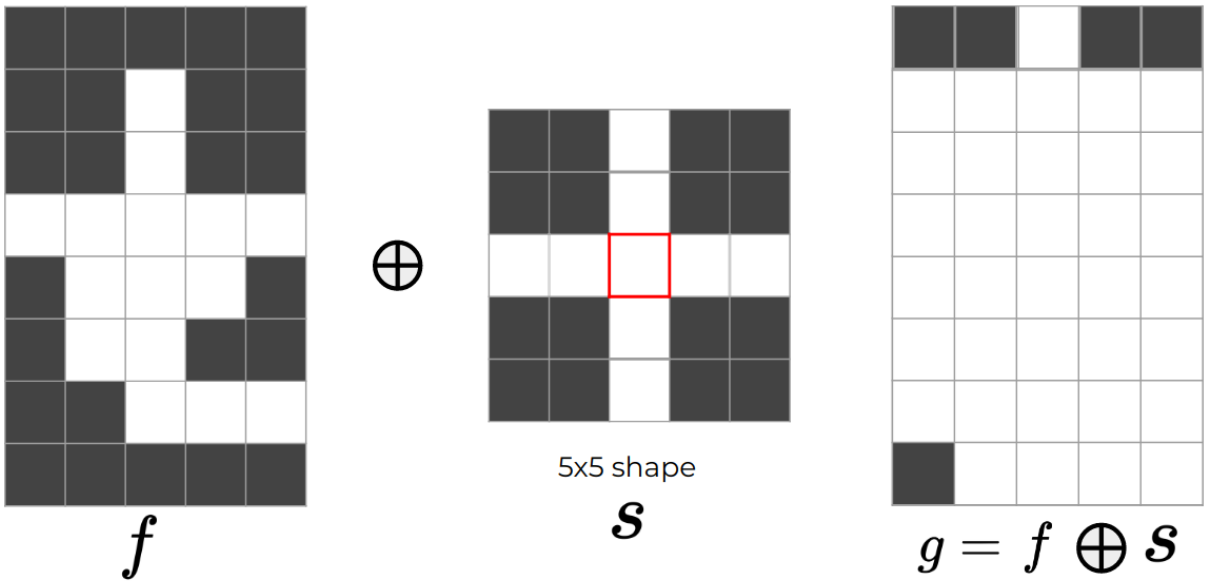


Figure: Structuring element with matrix representation

Here, we can see most of the pixels are white in the output image because of the hit condition.

Properties

- **Translation invariant**

  $$(A \oplus B) + x = A \oplus (B + x)$$

  Shifting in the image object or structuring element does not affect the dilation operation.

- **Dilation is increasing**

  $$\text{If } A \subseteq C,\ A \oplus B \subseteq C \oplus B$$

  where B is structuring element and A and C are image objects.

- **Satisfy Decomposition theorem**

  $$A \ominus (B \cup C) = (A \ominus B) \cup (A \ominus C)$$

- **Satisfy associative property**

$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$

- **Satisfy commutative property**

$$A \oplus B = B \oplus A$$

Opening

Opening involves erosion followed by dilation in the outer surface (the foreground) of the image. All the above-said constraints for erosion and dilation applies here. It is a blend of the two prime methods. It is generally used to remove the noise in the image.

An opening is a process in which erosion operation follows the dilation. We can see this clearly in its mathematical definition presented below.

For any set of foreground elements, A and structuring element B opening of A by B is

$$A \circ B = (A \ominus B) \oplus B$$

This means the opening is erosion, followed by dilation using the same structuring elements.

As we know that erosion removes some after of the elements from the foreground, and following the dilation, we refill these parts.

Using the opening, we can smooth surfaces, break narrow ridges, and eliminate protrusions. If we take a square structuring element of 5x5 pixels, we can remove all artifacts with less than 5.

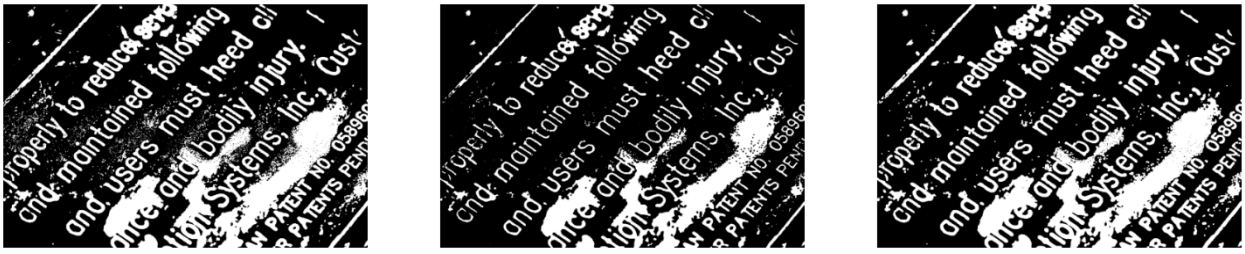


Figure: Structuring element with matrix representation

From the above figure, we can see the differences between erosion and opening. Erosion operation has eroded a lot of the character while Opening operation has thicked the character. The reason for it is that the main task of erosion operation is to erode the image object while opening does two actions: one is erosion, and the other is dilation. Here, the dilation process is overtaking the erosion action and making the character thicker.

Properties:

- **Idempotent**

$$(A \circ B) \circ B = A \circ B$$

- **Increasing monotonicity**

  If $A \subseteq C$, $A \circ B \subseteq C \circ B$, where A and C are image objects, and B is structuring element

- **Translation Invariant**

$$(A + x) \circ B = (A \circ B) + x$$

Closing

Closing involves dilation followed by erosion in the outer surface (the foreground) of the image. All the above-said constraints for erosion and dilation applies here. It is a blend of the two prime methods. It is generally used to remove the noise in the image.

Morphological Gradient

Morphological gradient is slightly different than the other operations, because, the morphological gradient first applies erosion and dilation individually on the image and then

computes the difference between the eroded and dilated image. The output will be an outline of the given image.

Top Hat

Top Hat is yet another morphological operation where Opening is performed on the binary image and the output of this operation is a difference between the input image and the opened image.

Black Hat

The black-hat operation is used to do the opposite, enhancing dark objects of interest on a bright background. The output of this operation is the difference between the closing of the input image and the input image.

Histogram Equalization:

A histogram is a representation of frequency distribution. It is the basis for numerous spatial domain processing techniques. Histogram manipulation can be used for image enhancement.

Contrast is defined as the difference in intensity between two objects in an image. If the contrast is too low, it is impossible to distinguish between two objects, and they are seen as a single object.

Histogram equalization is a widely used contrast-enhancement technique in image processing because of its high efficiency and simplicity. It is one of the sophisticated methods for modifying the dynamic range and contrast of an image by altering that image such that its intensity histogram has the desired shape. It can be classified into two branches as per the transformation function is used.

What is an image Histogram?

The histogram for any image acts as the graphical representation of the distribution of the intensity of the image. The histogram plots the number of pixels according to the intensity values. The image histogram describes an image by its intensity distribution. The image histogram for each intensity value considered quantifies the number of pixels.

What is Histogram Equalization in OpenCV?

Histogram equalization is an image processing method to adjust the contrast of an image using its intensity distribution histogram.

The equalization method increases the value of global contrast when the close contrast values represent the usable data for the image. Through the adjustments of the contrast values of the image, the intensity distribution of the image becomes clearer. Local regions of the image with lower contrast values gain a higher contrast value.

Histogram equalization method effectively spreads the frequent intensity values of the image. The histogram equalization method effectively spreads out the clustered intensity values. This improves the contrast of the image, while the intensity range of the image is stretched. Images with both dark or bright backgrounds and foregrounds benefit from this method.

cv2.equalizeHist() function in OpenCV performs histogram equalization to improve the contrast of the images and stretch the intensity range for the image histogram. The function takes a grayscale image as the input and returns an equalized image.

Histogram equalization of grayscale image

The cv2.equalizeHist() function takes a grayscale image as input. Therefore, we can simply provide a grayscale image for equalization. The equalized image in the output has improved contrast as compared to the original image. The cv2.equalizeHist() function returns an image with improved contrast and stretches the intensity range for the image histogram.

Histogram equalization of color image

The channels of the RGB color space represent the intensity of the associated color and not the brightness of the whole image. The application of histogram equalization on the color channels will not give us the desired results. The brightness of the image has to be separated from the color channels and then histogram equalization can be applied.

Color Spaces and Conversion in OpenCV

What are color spaces in image processing?

Colors spaces in image processing identify the combination of specific colors in various models and the mapping functions. Once identification of the color combination i.e., the color space is done, the associated color model will be recognized automatically. Color spaces represent the color channels of the image and represent the particular hues of the image.

There are numerous color spaces and their associated color models available for image processing tasks. Each color space has its significance.

Color spaces and color models

A color space is simply a grouping of colors that allows us to express and reproduce colors consistently. A color model is a mathematical representation of colors in the color space. RGB pixels of an image are represented by a three-integer tuple of red, green, and blue values.

A color space describes both the color model and the abstract mapping function that is utilized to define actual colors when taken as a whole. Informally, selecting a color space also indicates that we are selecting a color model**.**

Color channels

An image comprises color channels. The color channels contain information regarding various components of the image. These color channels merged, to form a complete image.

Importance of lighting conditions in Computer vision

Lighting conditions play a crucial role in computer vision and image processing tasks. The quality of images input to the system will determine the success of the computer vision algorithm, application, and system that has been produced and will be developed in the future.

We'll certainly be able to improve our systems' resiliency in the face of low lighting, but the quality of an image to be worked with cannot be assured. The computer vision algorithm's success or failure can be determined by the illumination conditions for the image.

We need to review three parameters to achieve the optimum lighting conditions while working with computer vision and image processing tasks.

1. **High contrast:** Attempts should be made to maximize the contrast between the image's regions of interest and the rest of the image. The region of interest should have distinguishable contrast from the rest of the image to be easily detectable and extracted.

2. **Generalizable:** The lighting circumstances should be steady, consistent, and reproducible from one object to the next.

3. **Stable:** The images for a certain computer vision application should have stable, consistent, and repeatable lighting conditions**.**

Color spaces available in OpenCV

There are various color spaces available in OpenCV.

1. **BGR/RGB Color Space**

The default color space in OpenCV is RGB. OpenCV by default reads images in the BGR format. The sequence for the color channel is blue, green, and red instead of red, green, and blue. Each color is stored in a channel in the BGR format. The BGR or RGB color space model is an additive model where the different intensities of different channels give different shades of color. The values for each color in the BGR color spaces is represented as Blue (255, 0, 0), Green (0, 255, 0), Red (0, 0, 255)

Properties of RGB color space:

The BGR color space is an additive model where the linear combinations of the red, green, and blue channels give different colors. The amount of light reflected from the surface defines the correlation between the three channels.

Image Representation using a Spatial Function

An image is a visual representation of a scene or object, often a photograph or two-dimensional picture. It is defined as a two-dimensional function, f (x, y), where x and y are coordinates and f at location (x, y) gives the intensity or gray level. In digital computers, continuous image signals are stored and processed in discrete levels, while discrete images are represented using spatial function f.

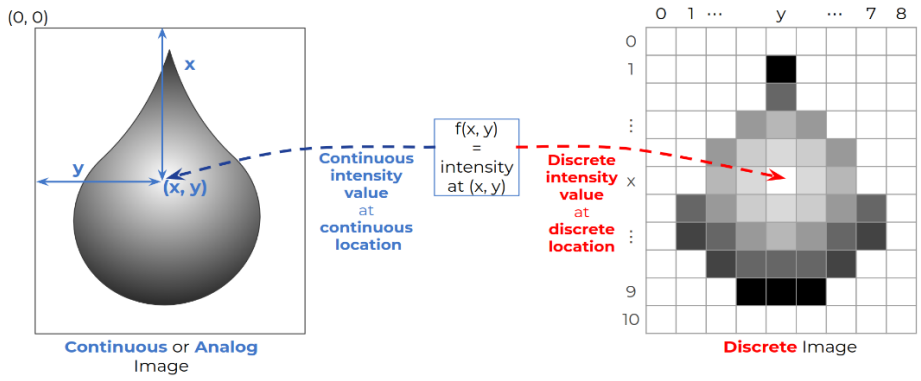


Figure: Image representation using spatial function f (x, y)

Digital images can be quantized into discrete images with M rows and N columns, representing the intensity level at spatial location (x, y) with f (x, y) as a M×N matrix.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Pixels are the smallest element of a discrete image, also known as a picture element. The intensity at the top-left pixel is given by f (0,0). An example of a discrete image is an 11x11 colorless image with three levels: gray, black, and white. The intensity value ranges from 0 to 1, with 0 representing the lowest intensity and 1 representing the highest. A matrix can represent the image with each intensity level.

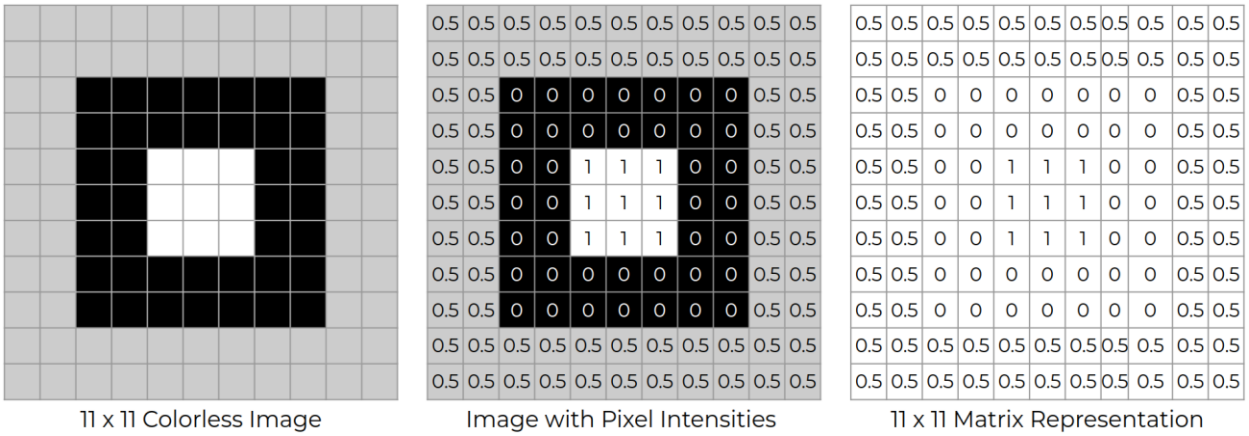


Figure: Image representation

Representation of Colored Image

The text discusses color formation in colored images using primary colors red, green, and blue. Mixing these colors yields different colors like yellow, magenta, cyan, white, and black. Different intensities of these colors can be denoted with scales from 0 to 255. This allows for the representation of 256×256×256=16,777,216 different colors. A color with a vector of length 3 is defined, and the RGB color space is then defined.

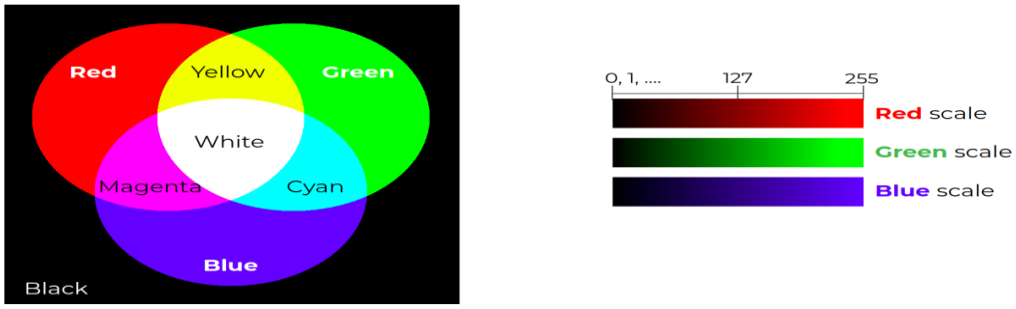


Figure: Mixing red, blue, and green colors to generate other colors

A color space is a coordinate system and subspace within it, with each point indicating a color. The RGB color space is used to create 3D coordinate systems with axes for red, green, and blue. Primary colors are located at the origin, while secondary colors are on the cube's corners.

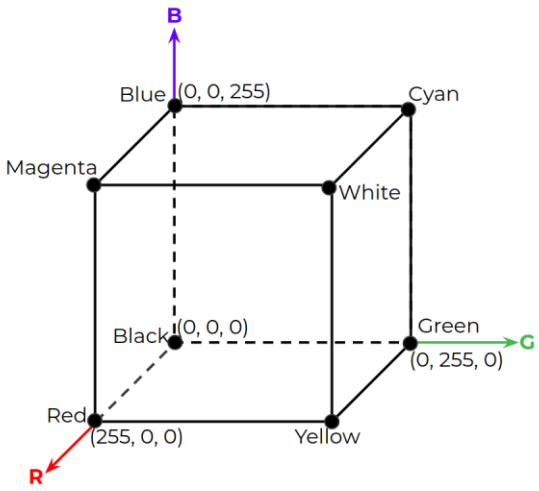


Figure: RGB colors pace

We said that different intensities of red, green, and blue colors could be mixed to form other colors. Hence, a colored image can be represented as a combination of red, green, and blue channels. The red channels contain the image's red component, while green and blue channels contain the image's green and blue components, respectively.

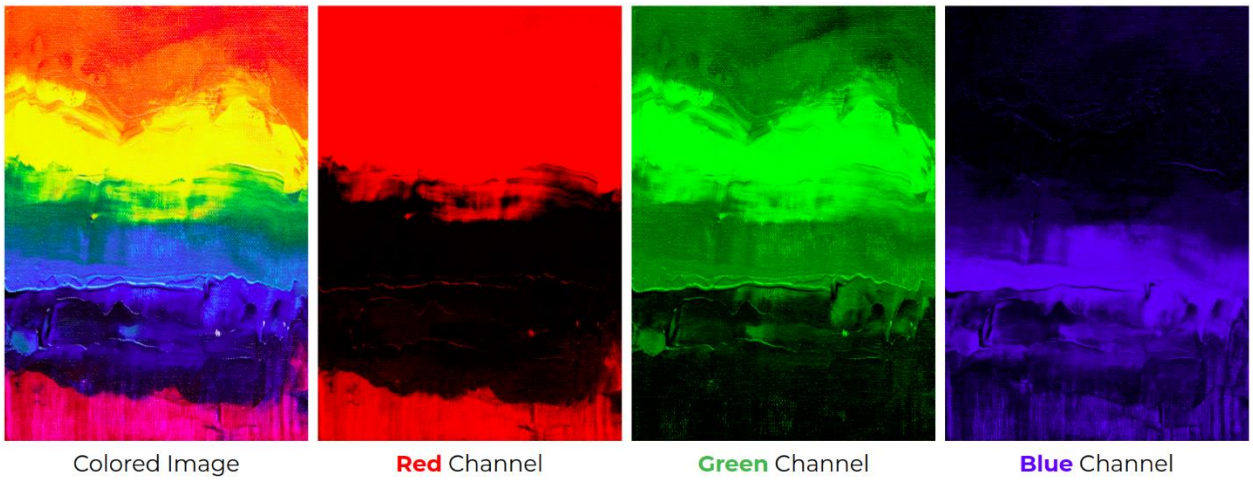


Figure: Decomposition of a colored image into red, green and blue channels

We can now represent a colored image as a stack of red, green, and blue channels. A pixel in the colored image is represented by the corresponding pixels in red, green, and blue channels. We can represent an image of height h, and width was a 3D tensor of shape (h, w, 3). Here the third dimension of length 3 corresponds to the color channels.

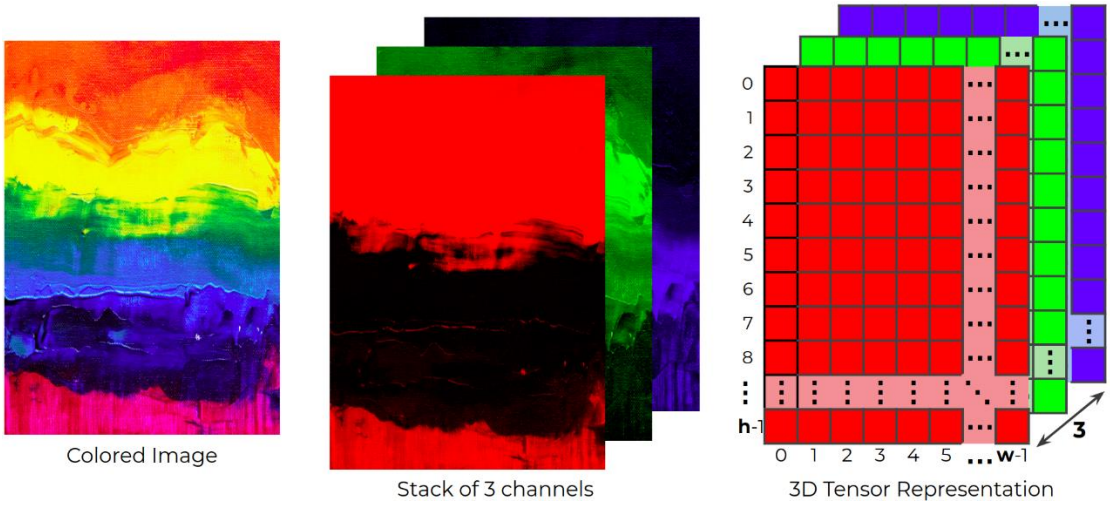


Figure: Representing colored image with 3D Tensor

Key Takeaways

- An image can be represented by a spatial function f (x, y), which gives the image's intensity at (x, y).
- A pixel is a discrete element of a digital image.

Image Type	Representation	Pixel Value
Binary image	2D Matrix	Binary - {0, 1}
Grayscale image	2D Matrix	Discrete - {0, 1, 2, ..., 255}
Colored image	3D Tensor	**R** - {0, 1, .., 255} **G** - {0, 1, .., 255} **B** - {0, 1, .., 255}

Table: Image Representation using matrix and tensor

2. **HSV Color space**

The HSV color space in OpenCV defines the hue, saturation, and value of brightness of the image. HSV expands to Hue, Saturation, and value. Hue refers to the dominant wavelength of the image, saturation refers to the purity of the shades of the colors and the value represents the brightness of the image.

The HSV color space stores information about the color as cylindrical represent of the color points of RGB color space. The value for hue ranges from 0-178, the value for saturation ranges from 0-255, and the value for the brightness of the image ranges from 0-255. HSV color spaces aid in color segmentation processes and provide better performance when compared to other color spaces in OpenCV.

Properties of HSV color space:

HSV color spaces make use of only one channel to describe color i.e., Hue, making it intuitive for specifying colors. The best thing is that it uses only one channel to describe color (H), making it very intuitive to specify color. HSV color space is dependent on the device.

3. HSL Color space

HSL stands for hue, saturation, and Luminance. The HSL Color space is a way of defining color more naturally. The Hue of the image specifies the base color, saturation defines the saturation of the color and luminance describes the brightness of the color.

Properties of HSL color space:

a. HSV color space tries to depict the colors as observed by the human eye.

b. The HSL color space describes color naturally.

4. Gray color space

Gray scaling is the process of converting an image from any color space to grayscale color space. The gray scaling process simply translates to converting an image to shades of gray. The shades of gray vary between complete black and complete white.

Properties of Gray color space:

a. Reduces the dimension of the images

b. Due to the dimension reduction, the information provided to each pixel is comparatively less.

c. Reduces the complexity of the model

d. Much more information is extracted for some images through gray scaling which might not be possible if the same algorithms or processes are applied to a color image.

e. Since a grayscale image is a single-channel image, hence it cannot be separated into different channels.

5. YCrCb Color space

YCrCb color space has been derived from the RGB color space. The color space has three components.

Y in the YCrCb color space represents the Luminance or the Luma. This component is obtained from the RGB color space after gamma correction.

Cr represents the difference between the R color channel of RGB color space and the luminance component. It describes the difference between the red component and Luma.

Cb represents the difference between the R color channel of RGB color space and the luminance component. It describes the difference between the blue component and Luma.

Properties of YCrCb color space:

a. Separation of luminance and chrominance into different color channels is done.

b. YCrCb color space is used for compression of the Cr and Cb components.

c. YCrCb color space is dependent on the device.

6. LAB Color space

In the RGB color space, the three-color channels encode both color and brightness information of the color for the image. In Lab color space, the encoding of color and brightness information is divided between channels. The LAB color space is three-dimensional and has three components.

The L channel encodes the brightness information of the color. The A channels stores information regarding the color components ranging from Green to Magenta color, while the B channels stores information regarding the color components ranging from Blue to Yellow color.

Properties of LAB color space:

a. It is a uniform color space and approximates how color is perceived.

b. The LAB color space is independent of the device.

Color Space Conversion in OpenCV

When we refer to color space conversion, we mean representing a color from one basis to another and converting an image from one color space to another while retaining as much similarity to the previous color space as possible to make the translated image look as similar to the original image as possible.

The cvtColor function is used for color space conversion in OpenCV. The **cvtColor()** function converts an image between color spaces. The OpenCV library offers more than 150 color space conversion functions.

Syntax:

cv2.cvtColor(src, code, dst, dstCn)

Parameters:

a. src: Source image or the image which has to be converted to another color space.

b. code: color space conversion codes provided by OpenCV

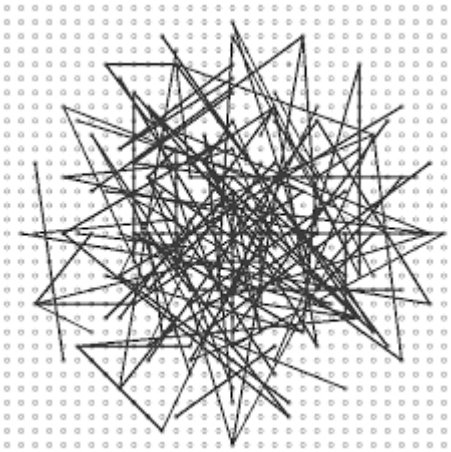
c. dst: Output image of the same size and depth as the source image.

d. dstCn: Number of channels of the output image

e. Return Value: The function returns a converted image

Introduction to BRIEF (Binary Robust Independent Elementary Features)

Feature point descriptors are now at the core of many Computer Vision technologies, such as object recognition, 3D reconstruction, image retrieval, and camera localization. Since applications of these technologies have to handle ever more data or to run on mobile devices with limited computational resources, there is a growing need for local descriptors that are fast to compute, fast to match, and memory efficient.



BRIEF is very fast both to build and to match. BRIEF easily outperforms other fast descriptors such as SURF and SIFT in terms of speed and terms of recognition rate in many cases.

Background

After detecting key point, we go on to compute a descriptor for every one of them. Feature descriptors encode interesting information into a series of numbers and act as a sort of numerical “fingerprint” that can be used to differentiate one feature from another. The defined neighborhood around pixel (key point) is known as a patch, which is a square of some pixel width and height.

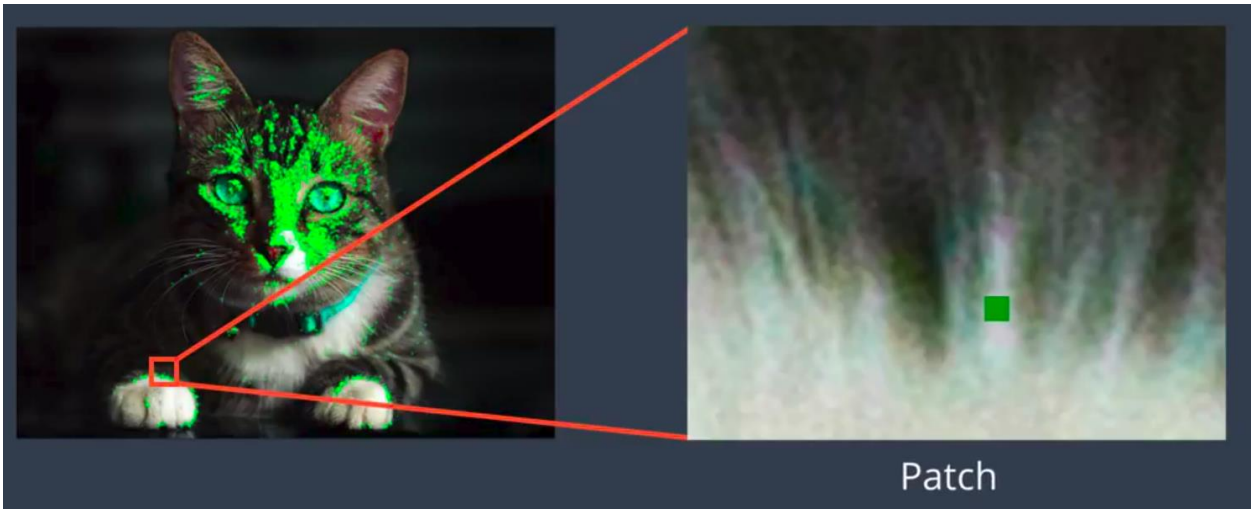
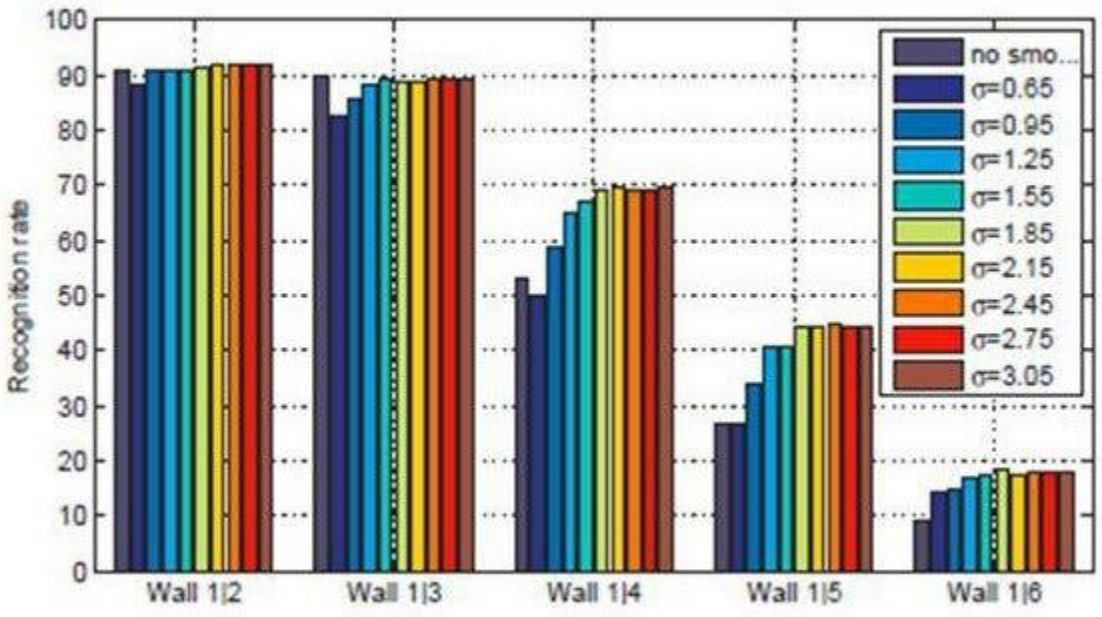


Image patches could be effectively classified on the basis of a relatively small number of pair-wise intensity comparisons. Brief convert image patches into a binary feature vector so that together they can represent an object. Binary features vector also known as binary feature descriptor is a feature vector that only contains 1 and 0. In brief each key point is described by a feature vector which is 128–512 bits string.

Smoothing Kernels

Brief deals with the image at pixel level so it is very noise-sensitive. By pre-smoothing the patch, this sensitivity can be reduced, thus increasing the stability and repeatability of the descriptors. It is for the same reason that images need to be smoothed before they can be meaningfully differentiated when looking for edges.

Brief uses Gaussian kernel for smoothing image. In the figure below we comparison of Gaussian smoothing on the recognition rates for variances of Gaussian kernel ranging from 0 to 3. The more difficult the matching, the more important smoothing becomes to achieving good performance. The recognition rates remain relatively constant in the 1 to 3 range and, in practice, we use a value of 2.



Smoothed Image to Binary Feature Vector

Now we have smoothed image patch, p. Now our target is to create a binary feature vector out of this patch. We simply create a binary feature vector of the binary test(τ) responses. A binary test τ is defined by:

$$Where\ \tau(p;x,y)\ is\ defined\ as:$$

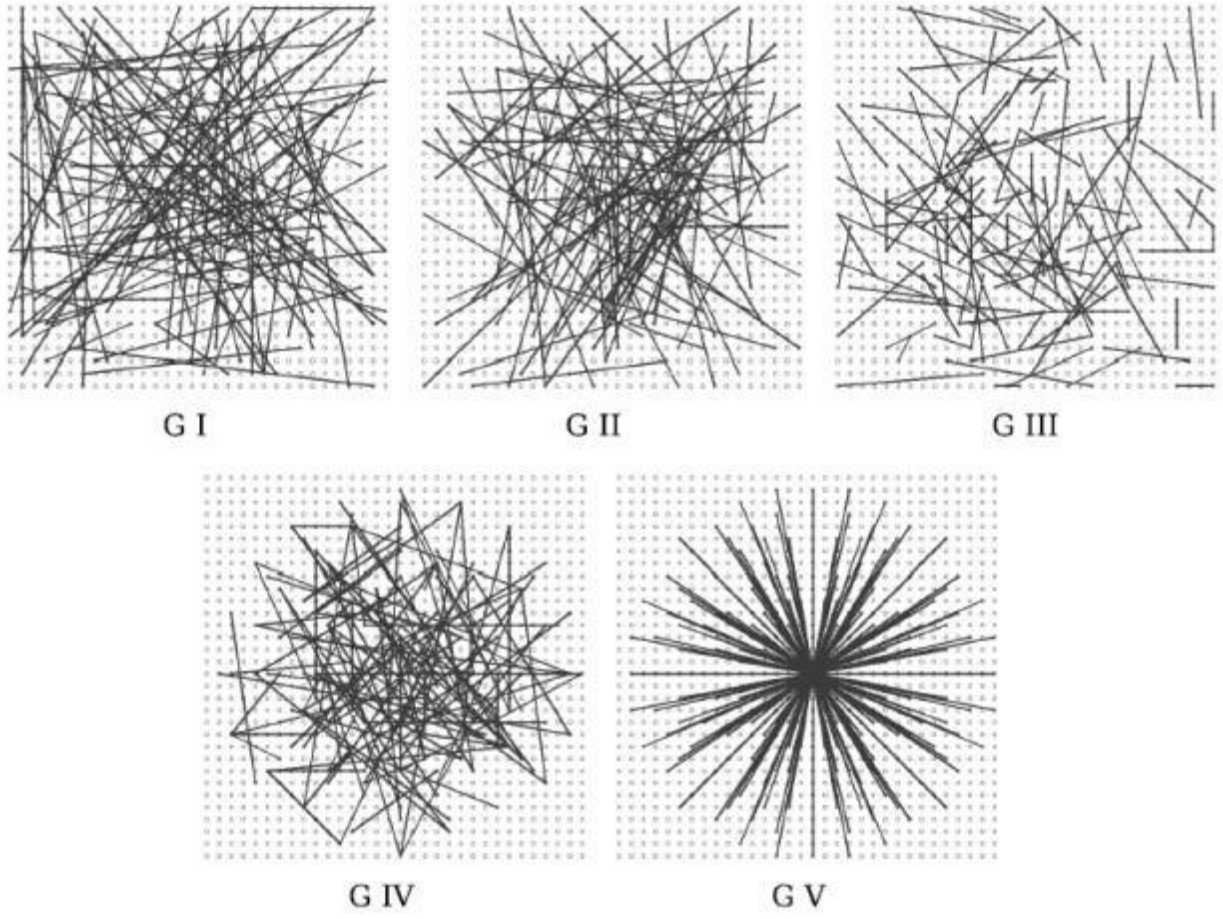
$$\tau(p;x,y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases}$$

$$p(x)\ is\ the\ \text{int}ensity\ value\ at\ pixel\ x.$$

where p(x) is the intensity of p at a point x. Choosing a set of **n (x, y)**-location pairs uniquely defines a set of binary tests. Where **n** is the length of the binary feature vector and it could be 128, 256, and 512.

How to select (x, y) pairs?

Generating a length **n** bit vector leaves many options for selecting the test locations ((x, y) pair). The (x, y) pair is also called random pair which is located inside the patch. Total we have to select **n** test (random pair) for creating a binary feature vector and we have to choose this **n** test from one of five approaches (Sampling Geometries) given below.



Let consider the size of patch p is (S x S) and assuming key point is located in the center of the patch.

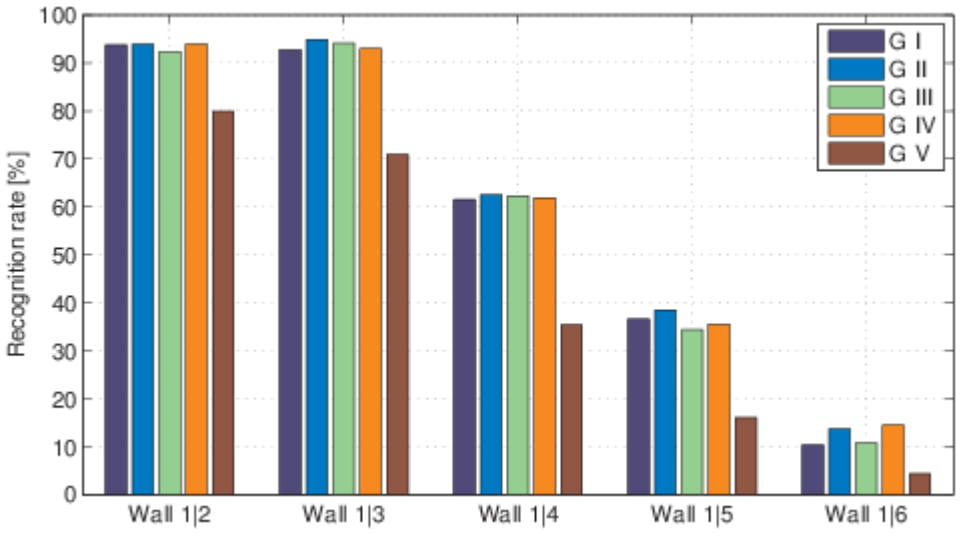
1. **Uniform (G I)**: Both x and y pixels in the random pair is drawn from a Uniform distribution or spread of **S/2 around key point**. The pair(test) can lie close to the patch border.

2. **Gaussian (G II)**: Both x and y pixels in the random pair is drawn from a Gaussian distribution or spread of **0.04 * S^2 around key point**.

3. **Gaussian (G III)**: The first pixel(x) in the random pair is drawn from a Gaussian distribution centered around the **key point** with a stranded deviation or spread of **0.04 * S^2**. The second pixel(y) in the random pair is drawn from a Gaussian distribution centered around the **first pixel(x) **with a standard deviation or spread of **0.01 * S^2**. This forces the test(pair) to be more local. Test(pair) locations outside the patch are clamped to the edge of the patch.

4. **Coarse Polar Grid (G IV):** Both x and y pixels in the random pair is sampled from discrete locations of a coarse polar grid introducing a spatial quantization.

5. **Coarse Polar Grid (G V):** The first pixel(x) in random pair is at (0, 0) and the second pixel(y) in the random pair is drawn from discrete locations of a coarse polar grid.



Finally, our BRIEF descriptor looks like:

$$f_{n_d}(\mathbf{p}) := \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i)$$

Advantages of BRIEF

Brief relies on a relatively small number of intensities difference tests to represent an image patch as a binary string. Not only is construction and matching for this descriptor much faster than for other state-of-the-art ones, but it also tends to yield higher recognition rates, as long as invariance to large in-plane rotations is not a requirement

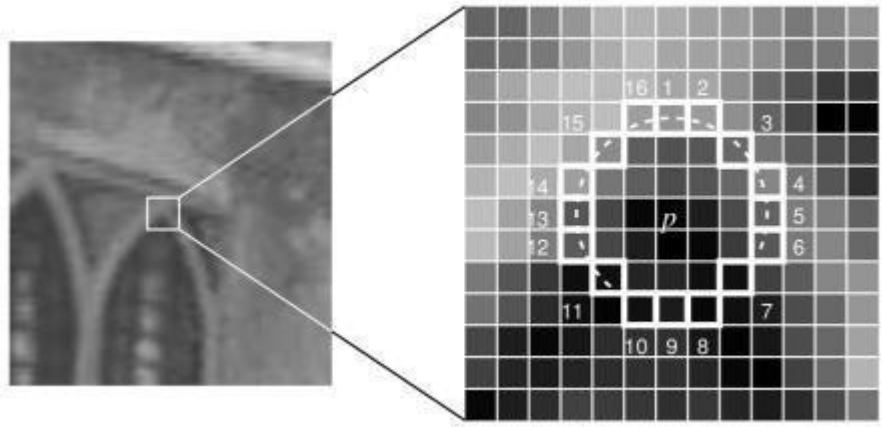
Introduction to FAST (Features from Accelerated Segment Test)

We have several feature detectors and many of them are really good. But when looking from a real-time application point of view, they are not fast enough. One best example would be SLAM (Simultaneous Localization and Mapping) mobile robot which has limited computational resources.

As a solution to this, *** Features from accelerated segment test (FAST) *** is a corner detection method, which could be used to extract feature points and later used to track and map objects in many computer vision tasks. The FAST corner detector was originally developed by Edward Rosten and Tom Drummond and was published in 2006. The most promising advantage of the FAST corner detector is its computational efficiency. Moreover, when machine learning techniques are applied, superior performance in terms of computation time and resources can be realized. The FAST corner detector is very suitable for real-time video processing application because of this high-speed performance.

Feature Detection using FAST

The algorithm is explained below:



- Select a pixel ***p*** in the image which is to be identified as an interest point or not. Let its intensity be ***Ip***.
- Select appropriate threshold value ***t***.
- Consider a circle of 16 pixels around the pixel under test. (This is a Brenham circle of radius 3.)
- Now the pixel*** p*** is a corner if there exists a set of ***n*** contiguous pixels in the circle (of 16 pixels) which are all brighter than ***Ip + t***, or all darker than ***Ip - t***. (The authors have used ***n***= 12 in the first version of the algorithm)

- To make the algorithm fast, first compare the intensity of pixels 1, 5, 9 and 13 of the circles with ***Ip***. As evident from the figure above, at least three of these four pixels should satisfy the threshold criterion so that the interest point will exist.

- If at least three of the four-pixel values — ***I1, I5, I9, I13*** are not above or below ***Ip + t***, then ***p*** is not an interest point (corner). In this case reject the pixel ***p*** as a possible interest point. Else if at least three of the pixels are above or below ***Ip + t***, then check for all 16 pixels and check if 12 contiguous pixels fall in the criterion.

- Repeat the procedure for all the pixels in the image.

There are a few limitations to the algorithm. First, for ***n***<12, the algorithm does not work very well in all cases because when ***n***<12 the number of interest points detected are very high. Second, the order in which the 16 pixels are queried determines the speed of the algorithm. A machine learning approach has been added to the algorithm to deal with these issues.

Machine Learning Approach

- Select a set of images for training (preferably from the target application domain)
- Run FAST algorithm in every image to find feature points.
- For every feature point, store the 16 pixels around it as a vector. Do it for all the images to get feature vector*** p***.
- Each pixel (say ***x***) in these 16 pixels can have one of the following three states:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t & \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t & \text{(similar)} \\ b, & I_p + t \le I_{p\to x} & \text{(brighter)} \end{cases}$$

- Depending on these states, the feature vector P is subdivided into 3 subsets Pd, Ps, Pb.
- Define a variable Kp which is true if ***p ***is an interest point and false if p is not an interest point.
- Use the ID3 algorithm (decision tree classifier) to query each subset using the variable Kp for the knowledge about the true class.
- The ID3 algorithm works on the principle of entropy minimization. Query the 16 pixels in such a way that the true class is found (interest point or not) with the minimum number of queries. Or in other words, select the pixel x, which has the most information about the pixel p. The entropy for the set P can be mathematically represented as:

$$
\begin{aligned}
H(P) &= (c+\bar{c})\log_2(c+\bar{c}) - c\log_2 c - \bar{c}\log_2 \bar{c} \\
\text{where} \quad c &= \left|\{p|K_p \text{ is true}\}\right| \quad \text{(number of corners)} \\
\text{and} \quad \bar{c} &= \left|\{p|K_p \text{ is false}\}\right| \quad \text{(number of non corners)}
\end{aligned}
$$

- This is recursively applied to all the subsets until its entropy is zero.
- The decision tree so created is used for fast detection in other images.

Non-maximal Suppression:

Detecting multiple interest points in adjacent locations is another problem. It is solved by using Non-maximum Suppression.

- Compute a score function, ***V*** for all the detected feature points. ***V*** is the sum of absolute difference between ***p*** and 16 surrounding pixels values.
- Consider two adjacent key points and compute their ***V*** values.

- Discard the one with lower ***V*** value.

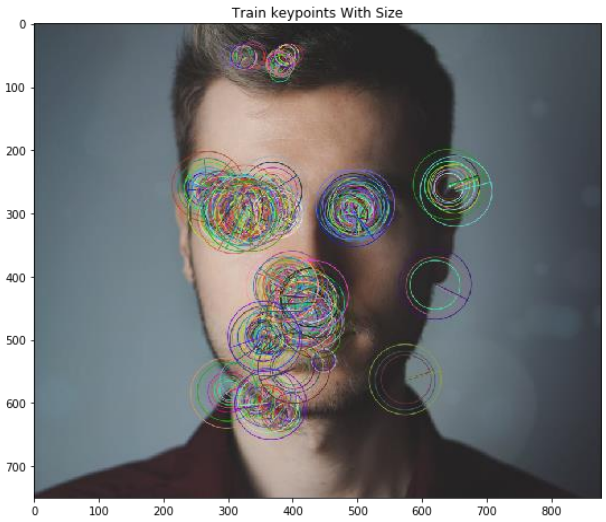
Limitations of the FAST Algorithm

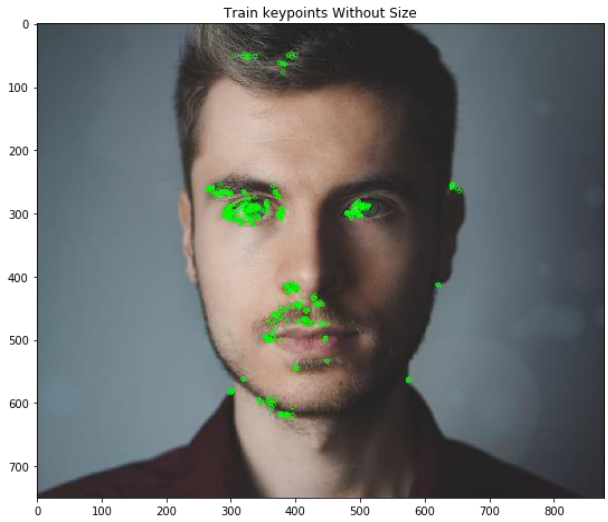
Other corner detection methods work very differently from the FAST method and a surprising result is that FAST does not detect corners on computer-generated images that are perfectly aligned to the ***x-axes** *and ***y-axes***. Since the detected corner must have a ring of darker or lighter pixel values around the center that includes both edges of the corner, crisp images do not work well.

This because the FAST algorithm requires a ring of contrasting pixels more than three-quarters around the center of corner. In the computer-generated image, both edges of a box at a corner are in the ring of the pixel used, so the test for a corner fails. A workaround to this problem is to add blur (by applying a Gaussian filter) to the image so that the corners are less precise but can be detected.

Introduction to ORB (Oriented FAST and Rotated BRIEF)

Oriented FAST and Rotated BRIEF (ORB) was developed at OpenCV labs by Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski in 2011, as an efficient and viable alternative to SIFT and SURF. ORB was conceived mainly because SIFT and SURF are patented algorithms.





ORB performs as well as SIFT on the task of feature detection (and is better than SURF) while being almost two orders of magnitude faster. ORB builds on the well-known FAST key point detector and the BRIEF descriptor. Both these techniques are attractive because of their good performance and low cost. ORB's main contributions are as follows:

- The addition of a fast and accurate orientation component to FAST
- The efficient computation of oriented BRIEF features
- Analysis of variance and correlation of oriented BRIEF features
- A learning method for decorrelating BRIEF features under rotational invariance, leading to better performance in nearest-neighbor applications

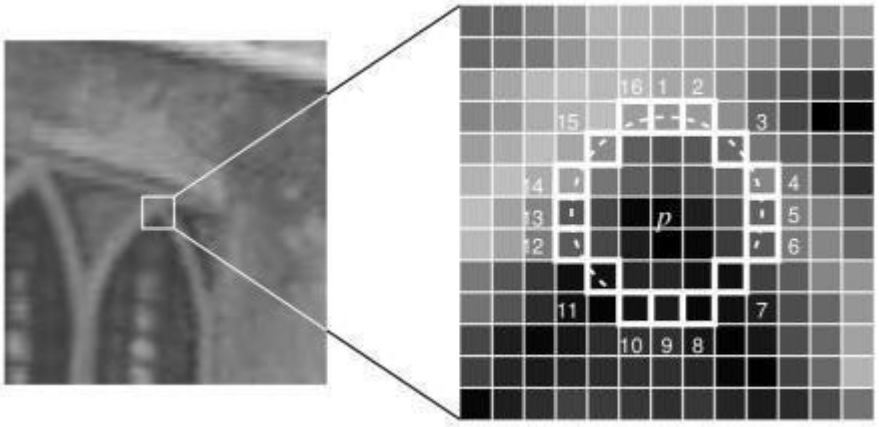
Basic Terms

Key point — Small region in an image that is particularly distinctive. Example: Corners where pixel values sharply change from light to dark.

Descriptors — A feature *descriptor* is an algorithm which takes an image and outputs *feature descriptors/feature vectors*. Feature descriptors encode interesting information into a series of numbers and act as a sort of numerical "fingerprint" that can be used to differentiate one feature from another. Ideally, this information would be invariant under image transformation, so we can find the feature again even if the image is transformed in some way.

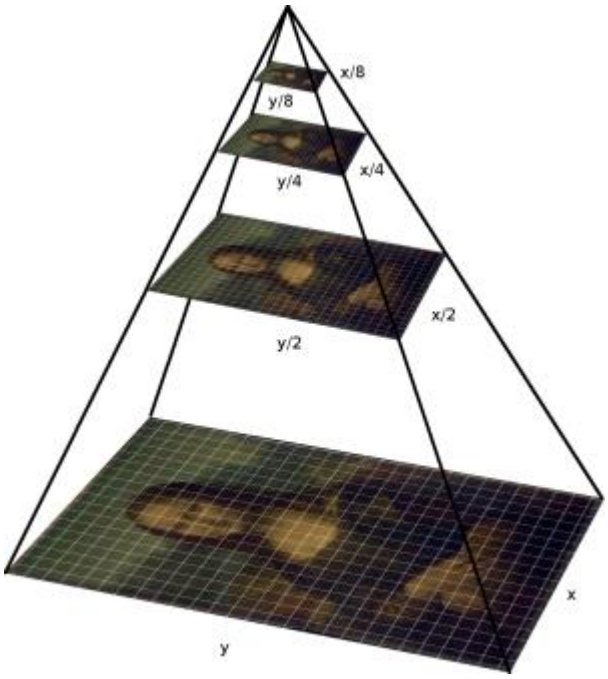
Fast (Features from Accelerated and Segments Test)

Given a pixel p in an array fast compares the brightness of p to surrounding 16 pixels that are in a small circle around p. Pixels in the circle is then sorted into three classes (lighter than p, darker than p or similar to p). If more than 8 pixels are darker or brighter than p than it is selected as a key point. So key points found by fast gives us information of the location of determining edges in an image.



However, FAST features do not have an orientation component and multiscale features. So, orb algorithm uses multiscale image pyramid. An image pyramid is a multiscale representation of a single image, that consist of sequences of images all of which are versions of the image at different resolutions. Each level in pyramid contains the down sampled version of the image than the previous level. Once orb has created

pyramid it uses the fast algorithm to detect key points in the image. By detecting key points at each level orb is effectively locating key points at a different scale. In this way, ORB is partial scale invariant.



After locating key points orb now assign an orientation to each key point like left or right facing depending on how the levels of intensity change around that key point. For detecting intensity change orb uses intensity centroid. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector may be used to impute an orientation.

First, the moments of a patch are defined as:

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y)$$

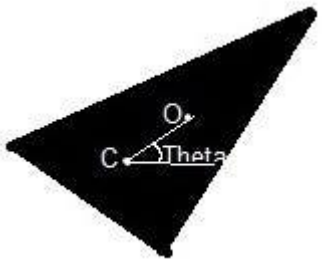
With these moments we can find the centroid, the "center of mass" of the patch as:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right)$$

We can construct a vector from the corner's center O to the centroid -OC. The orientation of the patch is then given by:

$$\theta = \text{atan2}(m_{01}, m_{10})$$

Here is an illustration to help explain the method:



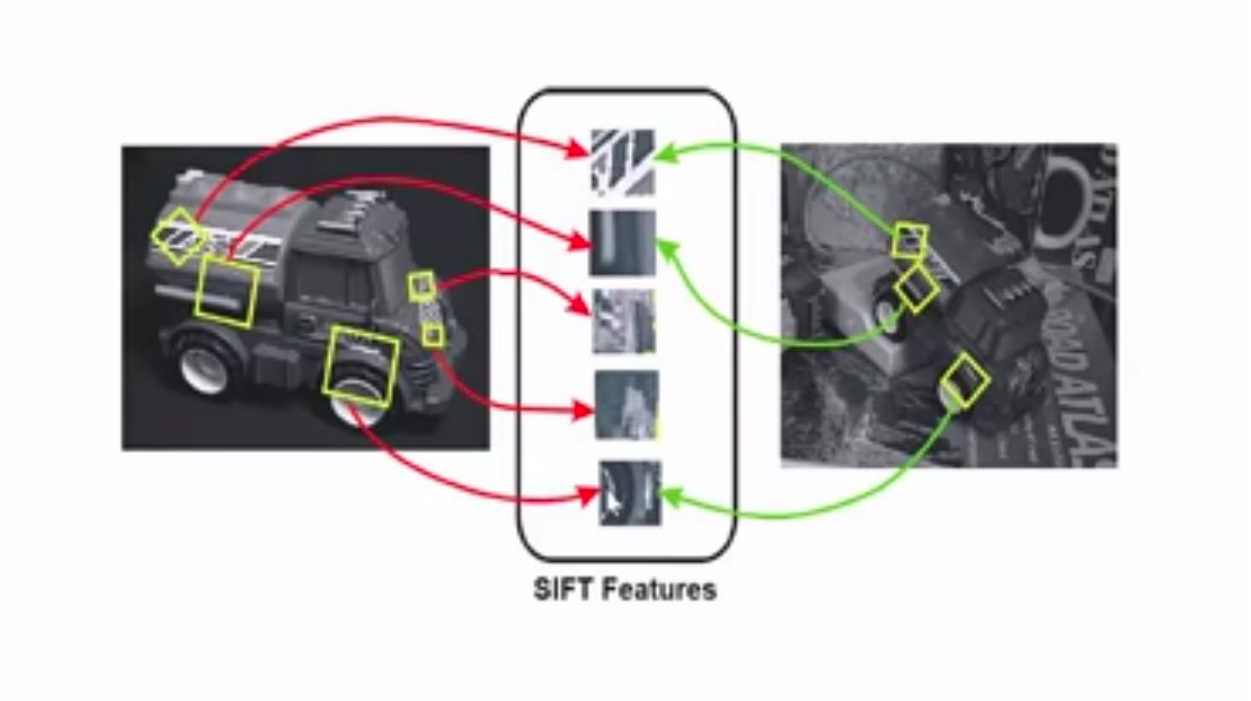
Once we've calculated the orientation of the patch, we can rotate it to a canonical rotation and then compute the descriptor, thus obtaining some rotation invariance.

Brief (Binary robust independent elementary feature)

Brief takes all key points found by the fast algorithm and convert it into a binary feature vector so that together they can represent an object. Binary features vector also known as binary feature descriptor is a feature vector that only contains 1 and 0. In brief, each key point is described by a feature vector which is 128–512 bits string.

Introduction to SIFT (Scale Invariant Feature Transform)

Scale Invariant Feature Transform (SIFT) was introduced by D. Lowe, a former professor at the University of British Columbia, in the year 2004. SIFT is a feature extraction method that reduces the image content to a set of points used to detect similar patterns in other images. This algorithm is usually related to computer vision applications, including image matching and object detection.



Key terminologies

Feature Extraction: These methods aim to reduce the number of features in the dataset by passing them through a mapping function.

Key points: An image's key points are spatial locations that are rotation and scale-invariant. These key points highlight what stands out in an image and which pixels are of utmost importance.

Descriptors: Descriptors are vectors that describe the local surroundings around the key points present in the image. These descriptors are used to make associations between different images.

Gaussian Blur: This is a method used to reduce the noise in the image, with the help of the Gaussian function, so that the key points can be detected efficiently.

The algorithm

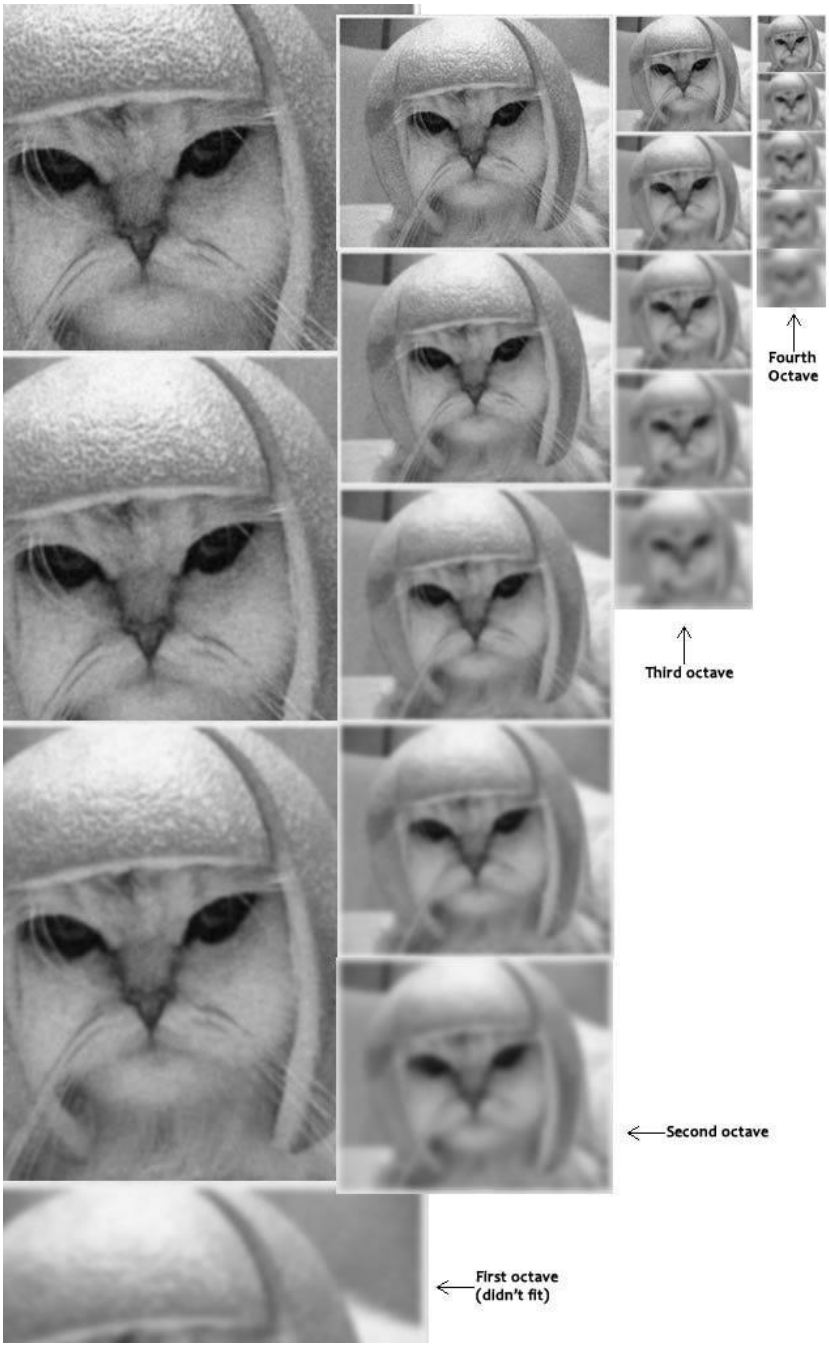
SIFT is quite an involved algorithm. There are four main steps involved in the SIFT algorithm. We will see them one by one.

- **Scale-space peak selection**: Potential location for finding features.
- **Key point Localization**: Accurately locating the feature key points.
- **Orientation Assignment**: Assigning orientation to key points.
- **Key point descriptor**: Describing the key points as a high dimensional vector.
- **Key point Matching**

Scale-space peak Selection

Scale-space

Real world objects are meaningful only at a certain scale. We might see a sugar cube perfectly on a table. But if looking at the entire milky way, then it simply does not exist. This multi-scale nature of objects is quite common in nature. And a scale space attempts to replicate this concept on digital images.



The scale space of an image is a function L (x, y, σ) that is produced from the convolution of a Gaussian kernel (Blurring) at different scales with the input image. Scale-space is separated into octaves and the number of octaves and scale depends on the size of the original image. So, we generate several octaves of the original image. Each octave's image size is half the previous one.

Blurring

Within an octave, images are progressively blurred using the Gaussian Blur operator. Mathematically, “blurring” is referred to as the convolution of the Gaussian operator and the image. Gaussian blur has a particular expression or “operator” that is applied to each pixel. What results is the blurred image.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

G is the Gaussian Blur operator and I is an image. While x, y are the location coordinates and σ is the “scale” parameter. Think of it as the amount of blur. Greater the value, greater the blur.

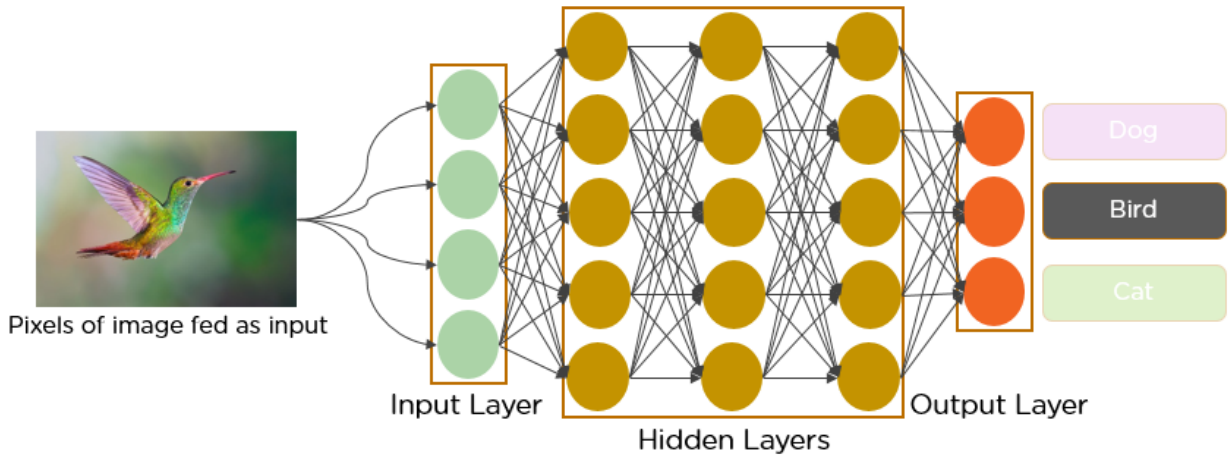
$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

DOG (Difference of Gaussian kernel)

Now we use those blurred images to generate another set of images, the Difference of Gaussians (DoG). These DoG images are great for finding out interesting key points in the image. The difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and *kσ*. This process is done for different octaves of the image in the Gaussian Pyramid.

Introduction to Convolutional Neural Networks (CNN)

In the past few decades, Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks (also known as CNN or ConvNet) in deep learning, especially when it comes to Computer Vision applications.



Since the 1950s, the early days of AI, researchers have struggled to make a system that can understand visual data. In the following years, this field came to be known as Computer Vision. In 2012, computer vision took a quantum leap when a group of researchers from the University of Toronto developed an AI model that surpassed the best image recognition algorithms, and that too by a large margin.

What Is a CNN?

Convolutional Neural Network (CNN) is the extended version of artificial neural networks (ANN) which is predominantly used to extract the feature from the grid-like matrix dataset. For example, visual datasets like images or videos where data patterns play an extensive role.

Convolutional Neural Network Architecture

A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

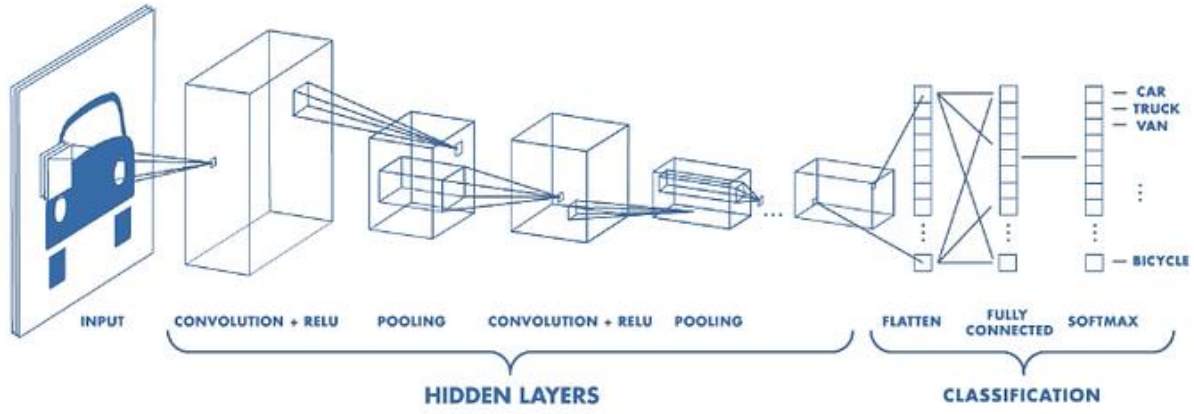


Figure: Architecture of a CNN

Application of CNN

Here are some applications of CNN.

- Object recognition within scene and videos
- Optical character recognition
- Computer vision
- Security surveillance
- Autonomous car
- Health care
- Agriculture
- Military

Convolution Layer

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load.

This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.4

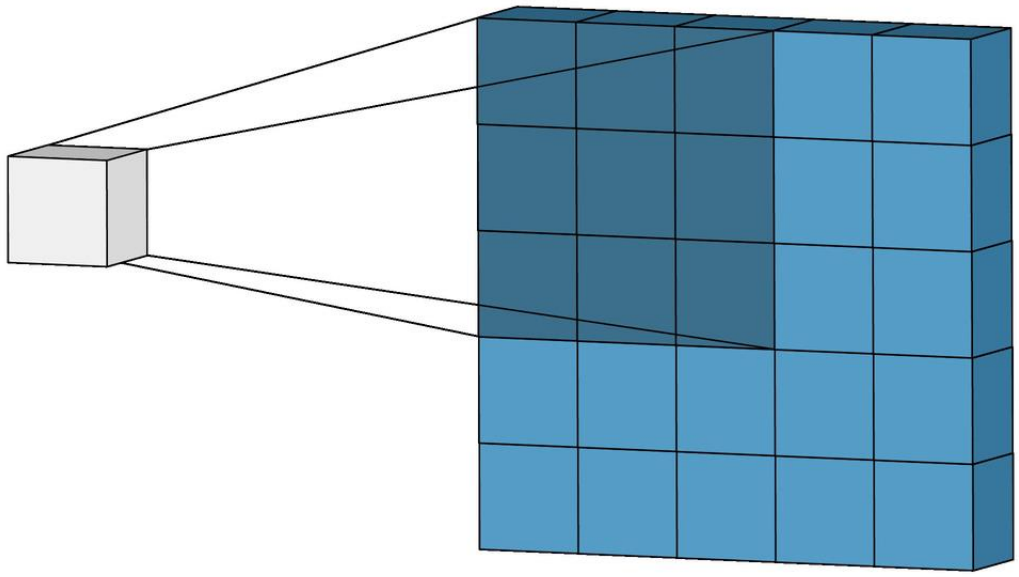


Illustration of Convolution Operation

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of

the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

If we have an input of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Formula for Convolution Layer

Relationship between inputs, filters and outputs

The main goal of convolution is featuring extraction from the give image data and reduce model parameters. In this section, we will learn how convolution reduces model parameters by establishing the relationship between the inputs, filters, and outputs.

The following quantities are taken into account while describing the relationship between the inputs, filters, and outputs.

- Input volume of size w1×h1×d1
- Hyperparameters:
    - Number of filters k,
    - The spatial extent of filter f (the depth of each filter is the same as the depth of each input: Convolution over Volume)
    - The stride s
    - The number of zero paddings p
- Output volume of size w2×h2×d2

Normal convolution

When stride one and valid padding, then the convolution is called normal convolution

Effect of kernel on output size

To understand the effect of filter size on output, consider the following example.

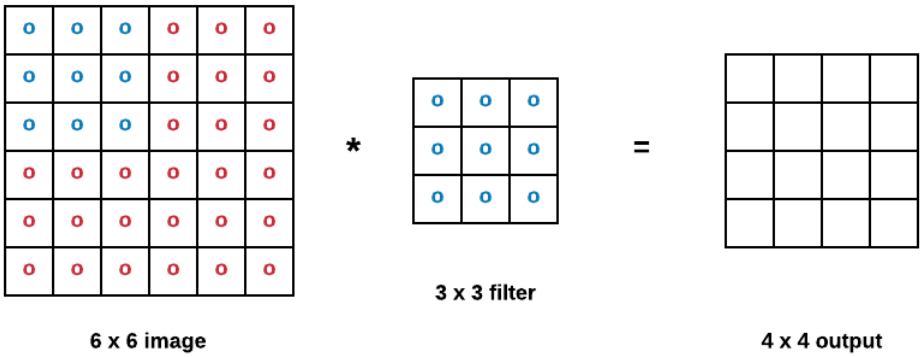


Figure 1: Normal convolution.

In example above, a 3×33×3 filter convolved across a 6×6 input image with stride one. The result is a 4×4output.

In general,

For an input volume of w1×h1×d1:

Output volume after normal convolution is calculated as,

- w2=w1−f+1
- h2=h1−f+1
- d2=k

Stride

Stride s refers to the number of pixels the filter is shifted after each convolution operation.

Effect of stride on output size

To understand the effect of stride on output, consider the following example.

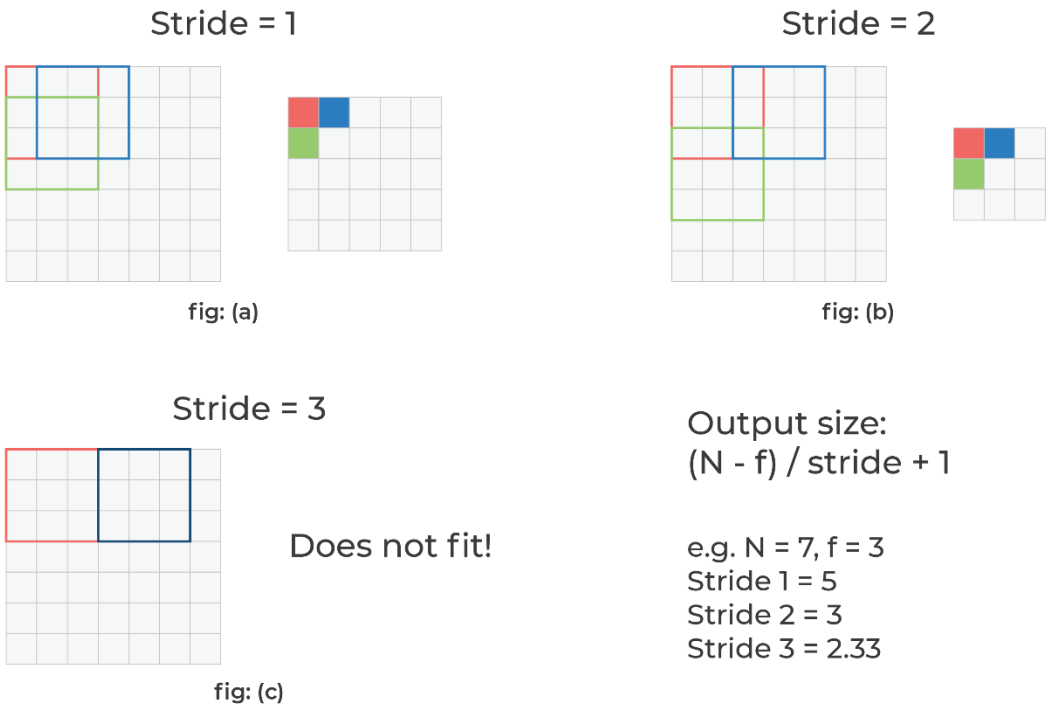


Figure - Illustration of stride

Stride controls how the kernel convolves around the input area. In the example above, in Figure-(a) kernel convolves around by shifting one unit at a time, and in Figure-(b) kernel convolves around by shifting two units at a time. Stride is usually set so that the output matrix is an integer and not a fraction.

In general,

For an input volume of $w_1 \times h_1 \times d_1$:

Output volume after stride with convolution is calculated as,

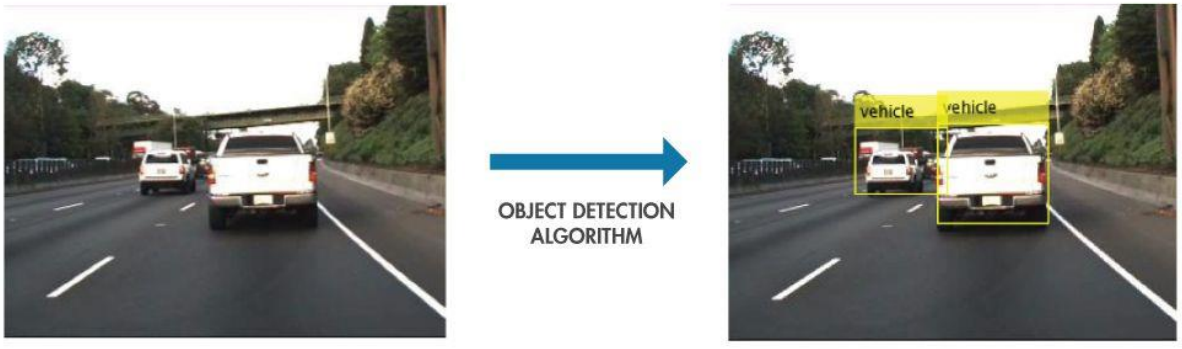
- $h_2 = \frac{w_1 - f}{s} + 1$
- $w_2 = \frac{h_1 - f}{s} + 1$
- d_2 remains unchanged

What Is Object Detection?

Object detection is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results. When humans look at images or video, we can recognize and locate objects of interest within a matter of moments. The goal of object detection is to replicate this intelligence using a computer.

Why Object Detection Matters

Object detection is a key technology behind advanced driver assistance systems (ADAS) that enable cars to detect driving lanes or perform pedestrian detection to improve road safety. Object detection is also useful in applications such as video surveillance or image retrieval systems.

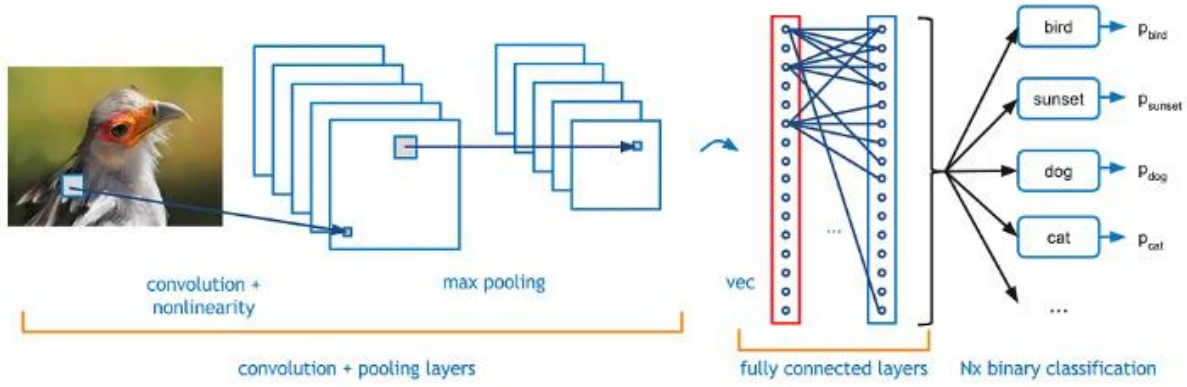


Using object detection to identify and locate vehicles.

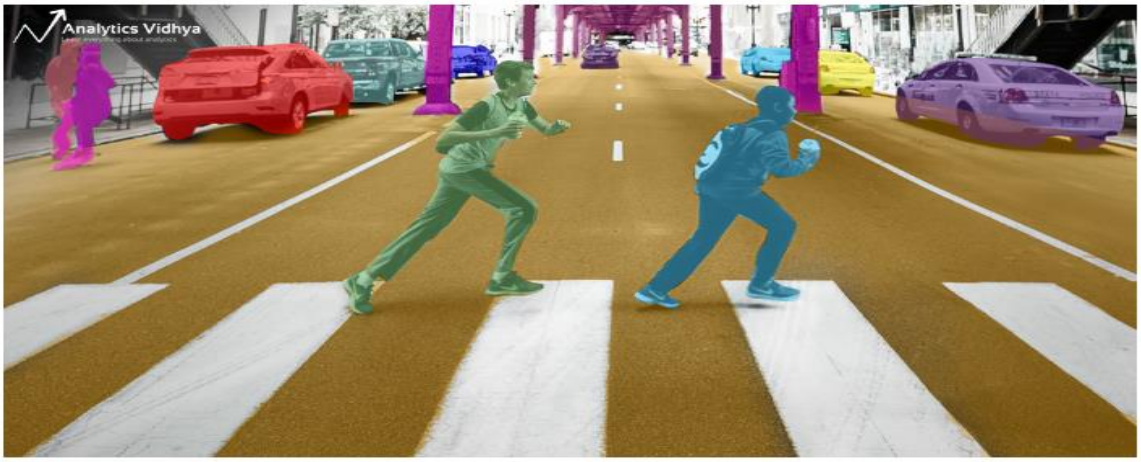
Object detection vs. other tasks

Let us break down the other three computer vision tasks individually for a greater understanding of each one:

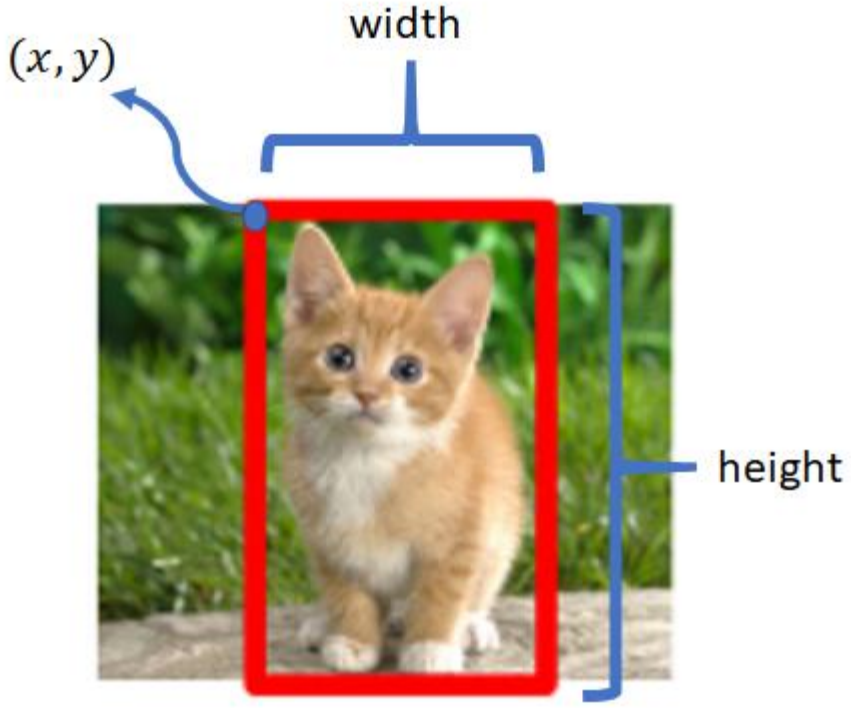
- **Image classification:** This is the prediction of the class of an item in an image. For example, when you carry out a reverse image search on Google, you are likely to receive a note that says “might include ‘x’, with the ‘x' being anything that the technology detects as the primary object of the image. Image classification can show that a particular object exists in the image, but it involves one primary object and does not provide the object’s location within the visual.



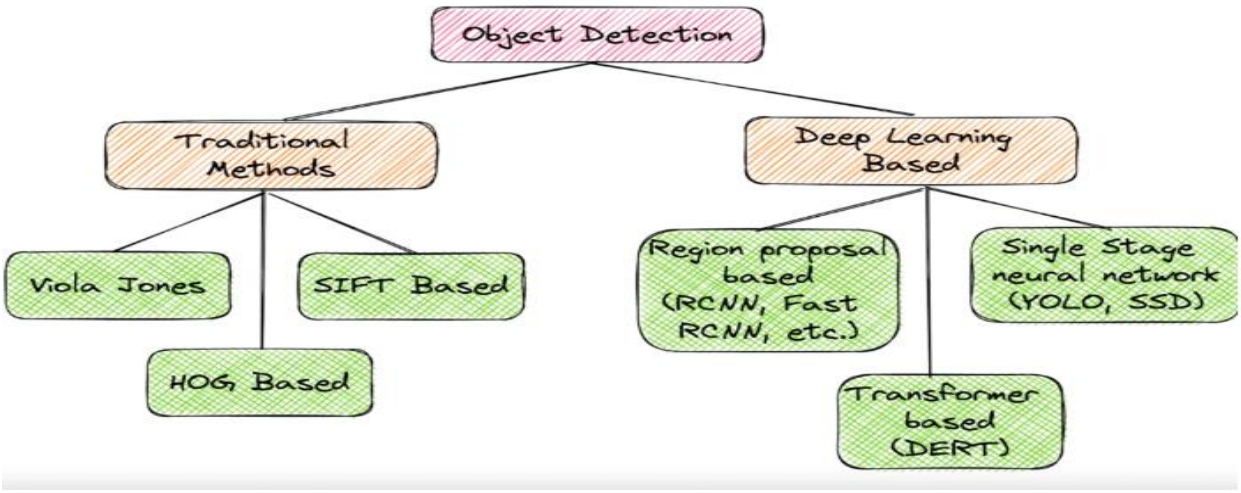
- **Segmentation:** Otherwise known as semantic segmentation, it is the task of grouping pixels with comparable properties together instead of bounding boxes to identify objects.



- **Object localization:** The difference with object detection is very subtle yet evident. Object localization seeks to identify the location of one or more objects in an image, whereas object detection identifies all objects and their borders without much focus on placement.



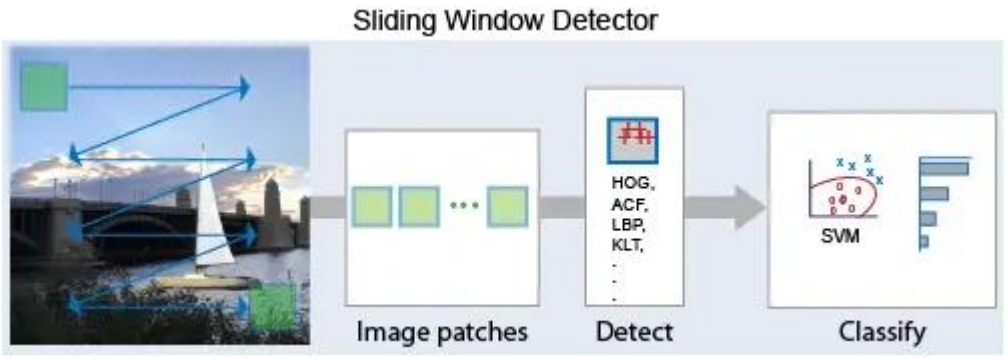
Early traditional approaches in object detection



Before the deep learning revolution took hold, object detection largely depended on manual feature extraction. These methods focused on crafting specific descriptors from images that were then used for detection tasks.

Sliding window approach

The sliding window approach is a technique traditionally used in object detection tasks before the invention of more advanced deep learning methods. The central idea is straightforward: a window (a bounding box) of a predefined size "slides" or moves across the image, covering all possible positions and scales. The sliding is done from left to right and top to bottom. Then, for each extracted bounding box hand, engineered features like HOG or SIFT are computed, which is then fed into a trained classifier, usually an SVM.



Objects of different sizes, or even the same objects of different scales, were this approach's main problem. This exact same operation was performed on several resized versions of the same image to detect objects of different scales, and the results were aggregated.

What is object detection with deep learning?

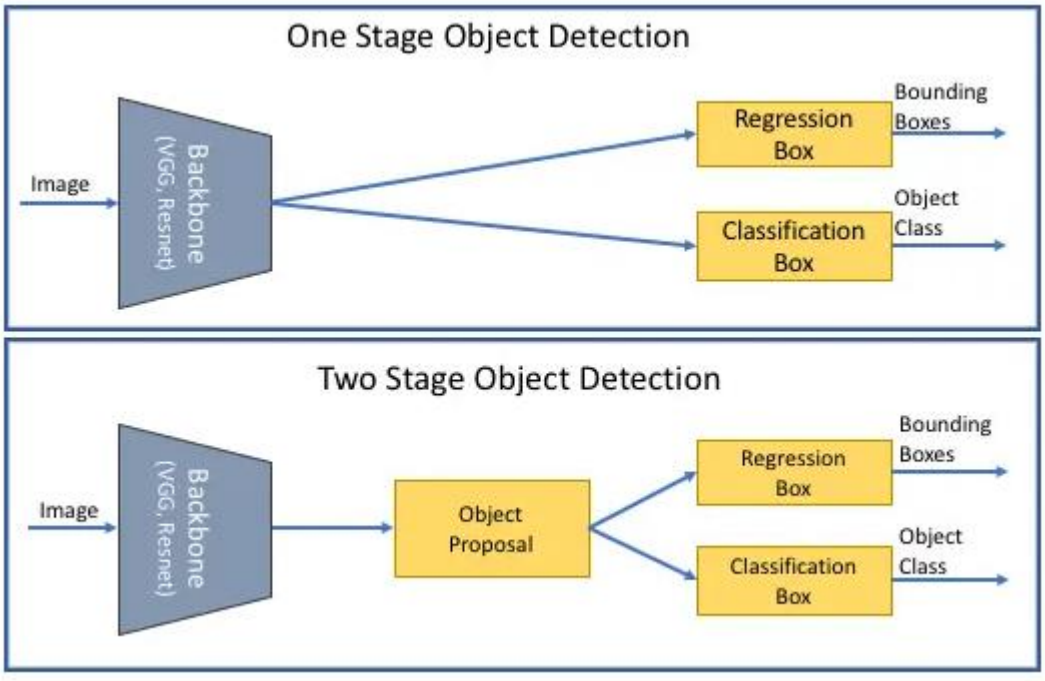
What sets object detection with deep learning apart from alternative approaches is the employment of convolutional neural networks (CNN). The neural networks mimic that of the complex neural architecture of the human mind. They primarily consist of an input layer, hidden inner layers, and an output layer. The learning for these neural networks can be supervised, semi-

supervised, and unsupervised, referring to how much of the is annotated, if at all (unsupervised). Deep neural networks for object detection yield by far the quickest and most accurate results for single and multiple object detection since CNNs are capable of automated learning with less manual engineering involved.

Methods and algorithms

Object detection is not possible without models designed especially for handling that task. These object detection models are trained with hundreds of thousands of visual content to optimize the detection accuracy on an automatic basis later on. Training and refining models are made efficient through the help of readily available datasets like COCO (Common Objects in Context) to help give you a head start in scaling your annotation pipeline.

Early deep learning-based object detection models were categorized into two classes: one-stage and two-stage detectors. One-stage object detectors direct prediction, eliminating completely the region proposal step. On the other hand, two-stage object detectors are composed of region proposals followed by classification and refinement of the proposal.

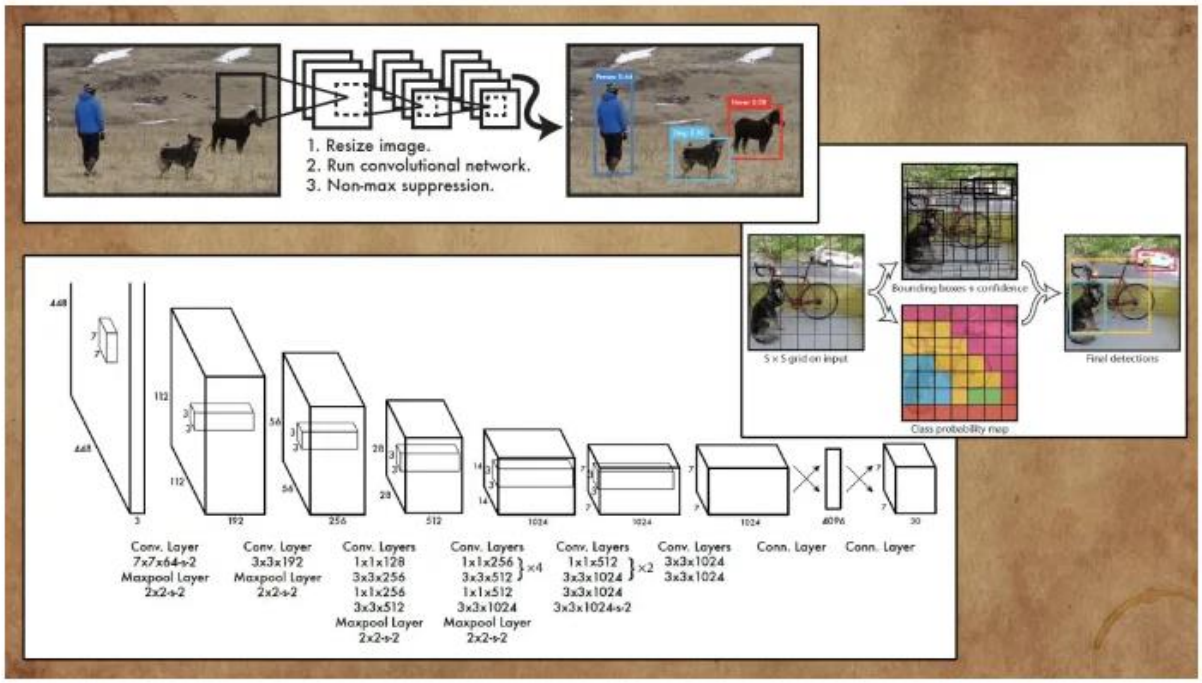


One-stage object detectors

One-stage object detection methods aim to simplify the object detection pipeline by predicting object class labels, and bounding box coordinates in a single pass, thereby often achieving faster processing speeds than two-stage methods. Given their efficiency, they are popular choices for real-time object detection.

YOLO: You Only Look Once

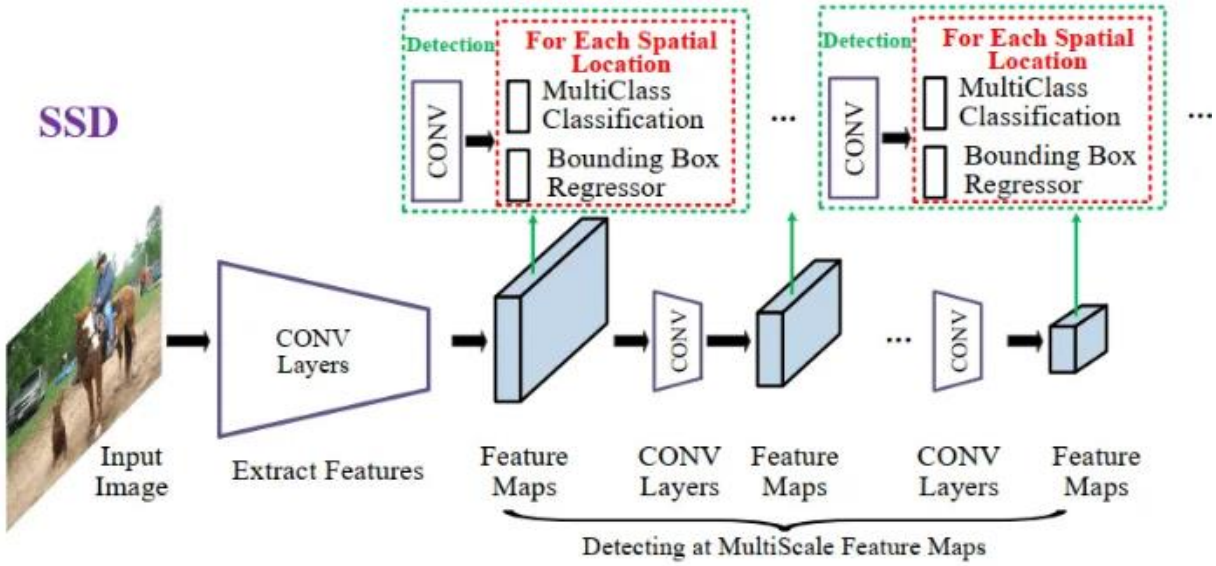
YOLO divides an image into a grid and predicts bounding boxes and class probabilities for each grid cell. Unlike sliding window and region proposal-based techniques, YOLO looks at the whole image only once, hence the name. The original YOLO has seen multiple versions, with YOLOv2 (YOLO9000), YOLOv3, and YOLOv4 introducing various improvements in accuracy, speed, and the ability to detect a wider range of object sizes. Its tiny versions are extremely fast, making it suitable for real-time object detection and even deployment on edge devices.



YOLO architecture

SSD: Single shot multibox detector

SSD predicts multiple bounding boxes and class scores for those boxes in one pass. It does this by using multiple feature maps from different network layers to predict detections at various scales. It uses a base network (often a VGG16 trained on ImageNet) for feature extraction. This network is truncated before the fully connected layers, allowing the model to take inputs of any size. One of SSD's key contributions is using feature maps from different layers in the network to predict detections at multiple scales.



Two Stage Object Detectors: The RCNN Family

The RCNN family of object detectors is very popular and has been the state-of-the-art object detection model for a long time. The evolution of the models in this family is very interesting and houses many insights that can be applied in any field of utilizing deep learning models. Let us get to an overview of the key developments in their architectures.

RCNN

The region-based convolutional neural network (RCNN) was proposed by Ross Girshick et al. in 2014 and was one of the first successful deep learning approaches to detect objects. RCNN operates by generating a set of region proposals using an external algorithm, applying a pre-trained convolutional neural network (CNN) to each proposal to extract features, and finally classifying each proposal using a support vector machine (SVM).