UNIT 4

Computer Vision

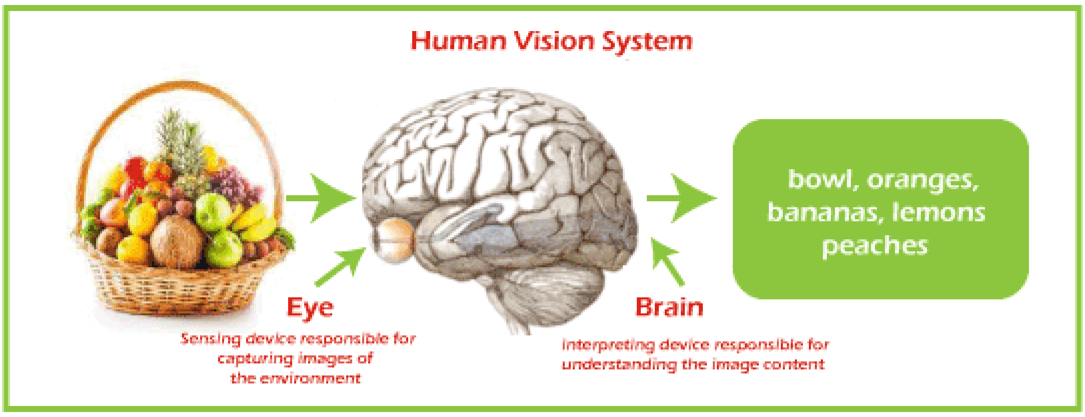
- Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information.
- Computer vision is a field of AI that trains computers to capture and interpret information from image and video data. By applying machine learning (ML) models to images, computers can classify objects and respond—like unlocking your smartphone when it recognizes your face.
- If AI enables computers to think, computer vision enables them to see, observe and understand.
- Computer vision needs lots of data.
- It runs analyses of data over and over until it discerns distinctions and ultimately recognize images. For example, to train a computer to recognize automobile tires, it needs to be fed vast quantities of tire images and tire-related items to learn the differences and recognize a tire, especially one with no defects.

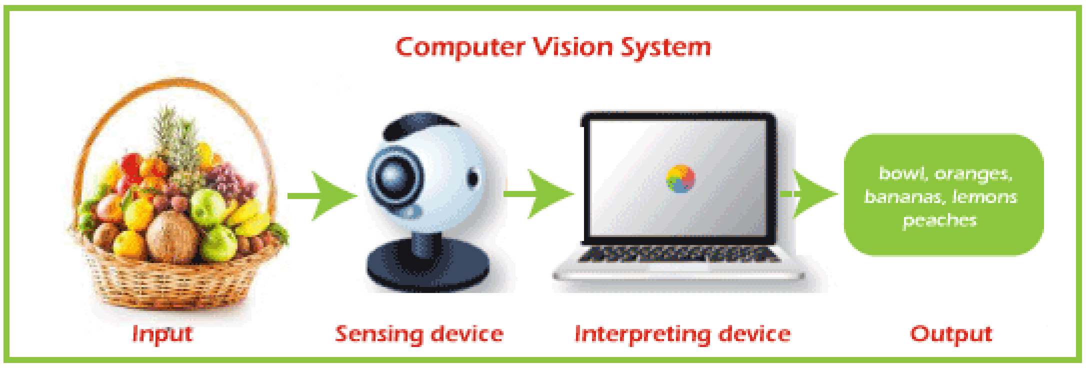
History of Computer Vision

The evolution of computer vision is classified as follows:

- ***1959:*** The first experiment with computer vision was initiated in 1959, where they showed a cat as an array of images. Initially, they found that the system reacts first to hard edges or lines, and scientifically, this means that image processing begins with simple shapes such as straight edges.
- ***1960:*** In 1960, artificial intelligence was added as a field of academic study to solve human vision problems.
- ***1963:*** This was another great achievement for scientists when they developed computers that could transform 2D images into 3-D images.
- ***1974:*** This year, optical character recognition (OCR) and intelligent character recognition (ICR) technologies were successfully discovered. The OCR has solved the problem of recognizing text printed in any font or typeface, whereas ICR can decrypt handwritten text. These inventions are one of the greatest achievements in document and invoice processing, vehicle number plate recognition, mobile payments, machine translation, etc.
- ***1982:*** In this year, the algorithm was developed to detect edges, corners, curves, and other shapes. Further, scientists also developed a network of cells that could recognize patterns.
- ***2000:*** In this year, scientists worked on a study of object recognition.
- ***2001:*** The first real-time face recognition application was developed.
- ***2010:*** The ImageNet data set became available to use with millions of tagged images, which can be considered the foundation for recent Convolutional Neural Network (CNN) and deep learning models.
- ***2012:*** CNN has been used as an image recognition technology with a reduced error rate.
- ***2014:*** COCO has also been developed to offer a dataset for object detection and support future research.

<u>How does Computer Vision Work?</u>


Human Vision System
Eye
Sensing device responsible for capturing images of the environment
Brain
Interpreting device responsible for understanding the image content
bowl, oranges, bananas, lemons peaches

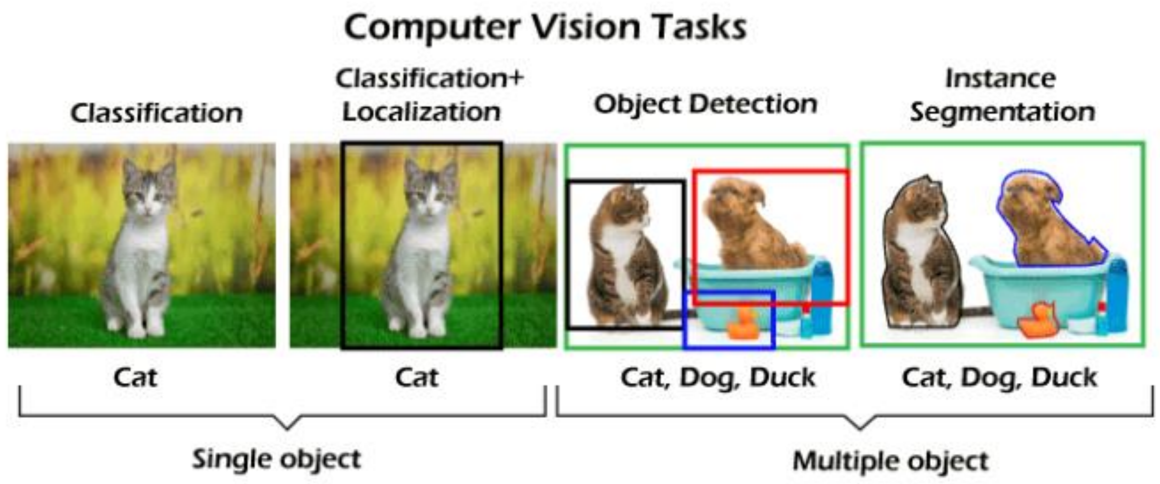


- Firstly, a vast amount of visual labeled data is provided to machines to train it.
- This labeled data enables the machine to analyze different patterns in all the data points and can relate to those labels. E.g., suppose we provide visual data of millions of dog images.
- In that case, the computer learns from this data, analyzes each photo, shape, the distance between each shape, color, etc., and hence identifies patterns similar to dogs and generates a model.
- As a result, this computer vision model can now accurately detect whether the image contains a dog or not for each input image.

<u>Task Associated with Computer Vision</u>

- ***Object classification:*** Object classification is a computer vision technique/task used to classify an image, such as whether an image contains a dog, a person's face, or a banana. It analyzes the visual content (videos & images) and classifies the object into the defined category. It means that we can accurately predict the class of an object present in an image with image classification.
- ***Object Identification/detection***: Object identification or detection uses image classification to identify and locate the objects in an image or video. With such detection and identification technique, the system can count objects in a given image or scene and determine their accurate location and labeling. For example, in a given image, one dog, one cat, and one duck can be easily detected and classified using the object detection technique.

- ***Object Verification:*** The system processes videos, finds the objects based on search criteria, and tracks their movement.
- ***Object Landmark Detection***: The system defines the key points for the given object in the image data.
- ***Image Segmentation:*** Image segmentation not only detects the classes in an image as image classification; instead, it classifies each pixel of an image to specify what objects it has. It tries to determine the role of each pixel in the image.
- ***Object Recognition:*** In this, the system recognizes the object's location with respect to the image.



<u>Applications of computer vision</u>

Below are some most popular applications of computer vision:

- ***Facial recognition:*** Computer vision has enabled machines to detect face images of people to verify their identity. Initially, the machines are given input data images in which computer vision algorithms detect facial features and compare them with databases of fake profiles. Popular social media platforms like Facebook also use facial recognition to detect and tag users. Further, various government spy agencies are employing this feature to identify criminals in video feeds.
- ***Healthcare and Medicine:*** Computer vision has played an important role in the healthcare and medicine industry. Traditional approaches for evaluating cancerous tumors are time-consuming and have less accurate predictions, whereas computer vision technology provides faster and more accurate chemotherapy response assessments; doctors can identify cancer patients who need faster surgery with life-saving precision.
- ***Self-driving vehicles:*** Computer vision technology has also contributed to its role in self-driving vehicles to make sense of their surroundings by capturing video from different angles around the car and then introducing it into the software. This helps to detect other cars and objects, read traffic signals, pedestrian paths, etc., and safely drive its passengers to their destination.

- ***Optical character recognition (OCR):***Optical character recognition helps us extract printed or handwritten text from visual data such as images. Further, it also enables us to extract text from documents like invoices, bills, articles, etc.
- ***Machine inspection:*** Computer vision is vital in providing an image-based automatic inspection. It detects a machine's defects, features, and functional flaws, determines inspection goals, chooses lighting and material-handling techniques, and other irregularities in manufactured products.
- ***Retail (e.g., automated checkouts):*** Computer vision is also being implemented in the retail industries to track products, shelves, wages, record product movements into the store, etc. This AI-based computer vision technique automatically charges the customer for the marked products upon checkout from the retail stores.
- ***3D model building:*** 3D model building or 3D modeling is a technique to generate a 3D digital representation of any object or surface using the software. In this field also, computer vision plays its role in constructing 3D computer models from existing objects. Furthermore, 3D modeling has a variety of applications in various places, such as Robotics, Autonomous driving, 3D tracking, 3D scene reconstruction, and AR/VR.
- ***Medical imaging:*** Computer vision helps medical professionals make better decisions regarding treating patients by developing visualization of specific body parts such as organs and tissues. It helps them get more accurate diagnoses and a better patient care system. E.g., Computed Tomography (CT) or Magnetic Resonance Imaging (MRI) scanner to diagnose pathologies or guide medical interventions such as surgical planning or for research purposes.
- ***Automotive safety:*** Computer vision has added an important safety feature in automotive industries. E.g., if a vehicle is taught to detect objects and dangers, it could prevent an accident and save thousands of lives and property.
- ***Surveillance:*** It is one of computer vision technology's most important and beneficial use cases. Nowadays, CCTV cameras are almost fitted in every place, such as streets, roads, highways, shops, stores, etc., to spot various doubtful or criminal activities. It helps provide live footage of public places to identify suspicious behavior, identify dangerous objects, and prevent crimes by maintaining law and order.
- ***Fingerprint recognition and biometrics***: Computer vision technology detects fingerprints and biometrics to validate a user's identity. Biometrics deals with recognizing persons based on physiological characteristics, such as the face, fingerprint, vascular pattern, or iris, and behavioral traits, such as gait or speech. It combines Computer Vision with knowledge of human physiology and behavior.

OpenCV

- (Open Source Computer Vision Library) is a powerful open source software widely used in image processing, deep learning and computer vision applications.
- Using the library, we can process images and videos to extract key-features which help in detecting objects or regions of interest (ROI) in an image.
- It provides support for multiple programming languages such as C++, Java and Python and seamlessly integrates with libraries, such as NumPy, to process images as numerical arrays.

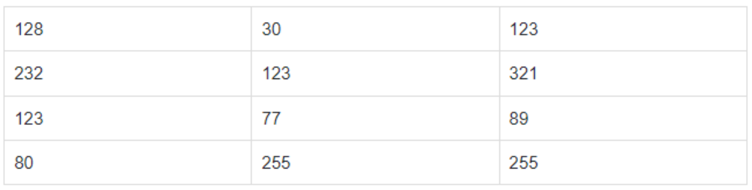
OpenCV with Python could be the most preferred choice for beginners due to its flexibility, simple syntax, and versatility. Various reasons make Python the best programming language for computer vision, which is as follows:

- **Easy-to-use:** Python is very famous as it is easy to learn for entry-level persons and professionals. Further, Python is also easily adaptable and covers all business needs.
- **Most used programming language:** Python is one of the most popular programming languages as it contains complete learning environments to get started with machine learning, artificial intelligence, deep learning, and computer vision.
- **Debugging and visualization:** Python has an in-built facility for debugging via **'PDB'** and visualization through Matplotlib.

Introduction to Image Processing

- ***Image processing*** is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image.
- An ***image*** is nothing more than a two dimensional signal. It is defined by the mathematical function f(x,y) where x and y are the two co-ordinates horizontally and vertically.
- The value of f(x,y) at any point is gives the pixel value at that point of an image.
- ***Pixel*** is the smallest element of an image. Each pixel correspond to any one value. In an 8-bit gray scale image, the value of the pixel between 0 and 255.
- The value of a pixel at any point correspond to the intensity of the light photons striking at that point. Each pixel store a value proportional to the light intensity at that particular location.

- The above figure is an example of digital image that you are now viewing on your computer screen. But actually , this image is nothing but a two dimensional array of numbers ranging between 0 and 255.

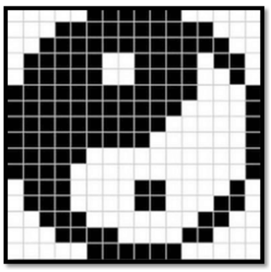


128	30	123
232	123	321
123	77	89
80	255	255

- Each number represents the value of the function f(x,y) at any point. In this case the value 128 , 230 ,123 each represents an individual pixel value. The dimensions of the picture is actually the dimensions of this two dimensional array.

<u>Types of Images</u>

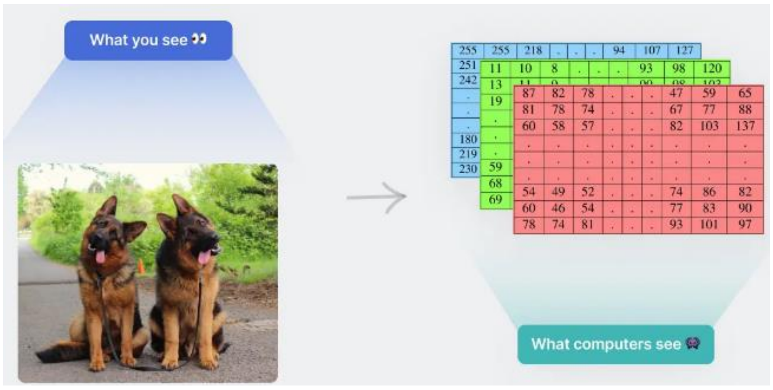
- ***<u>Binary Images:</u>***
  - It is the simplest type of image.
  - It takes only two values i.e, Black and White or 0 and 1.
  - The binary image consists of a 1-bit image and it takes only 1 binary digit to represent a pixel.
  - Binary images are mostly used for general shape or outline.
  - For Example: Optical Character Recognition (OCR).
  - Binary images are generated using threshold operation.
  - When a pixel is above the threshold value, then it is turned white('1') and which are below the threshold value then they are turned black('0')



- ***<u>Gray-scale images:</u>***
  - Grayscale images are monochrome images, Means they have only one color.
  - Grayscale images do not contain any information about color.
  - Each pixel determines available different grey levels.
  - A normal grayscale image contains 8 bits/pixel data, which has 256 different grey levels ranging between 0 and 255.

- **<u>Colour images</u>**
  - Colour images are three band monochrome images in which, each band contains a different color and the actual information is stored in the digital image.
  - The color images contain gray level information in each spectral band.
  - 8-bit color is used for storing image information in a computer's memory or in a file of an image.
  - In this format, each pixel represents one 8 bit byte.
  - It has 0-255 range of colors, in which 0 is used for black, 255 for white and 127 for gray color.

- The images are represented as red, green and blue (RGB images). And each color image has 24 bits/pixel means 8 bits for each of the three color band(RGB).
- ***16-bit color format:*** The 16-bit color format is also known as high color format. It has 65,536 different color shades. It is used in the system developed by Microsoft.
- ***24-bit color format:*** The 24-bit color format is also known as the true color format. The 24-bit color format is also distributed in Red, Green, and Blue. As 24 can be equally divided on 8, so it is distributed equally between 3 different colors like 8 bits for R, 8 bits for G and 8 bits for B.



Learn to extract and load the image

- *Install OpenCV:* `pip install opencv-python`
- *Importing Libraries* — To get started, we first need to import all necessary packages.

```
import cv2
```

- *Read the image* – Read any image by giving the name of that image or specify the path of the image with its file type extension using the keyword *imread.* Save the image as the variable.

```
img = cv2.imread('IMAGE1.jpg')  (OR)  img = cv2.imread('D://test//IMAGE1.png')
```

- Display the saved image using *imshow*

```
cv2.imshow('original', img)
```

Operating on images using OpenCV- Python

a) Accessing and Modifying pixel values

- You can access a pixel value by its row and column coordinates.
- For BGR image, it returns an array of Blue, Green, Red values.
- For grayscale image, just corresponding intensity is returned.

Code:
```
import cv2
img = cv2.imread('messi5.jpg')
px = img[100,100]
print( px )
```
Output:
```
[157 166 200]
```

Code:
```
# accessing only blue pixel
blue = img[100,100,0]
print( blue )
```
Output:
```
157
```

- You can modify the pixel values the same way.

Code:
```
img[100,100] = [255,255,255]
print( img[100,100] )
```
Output:
```
[255 255 255]
```

- Better pixel accessing and editing method :

Code:
```
# accessing RED value
img.item(10,10,2)
```
Output:
```
59
```

Code:
```
# modifying RED value
img.itemset((10,10,2),100)
img.item(10,10,2)
```

Output:
```
100
```

b) Accessing Image Properties:

- The shape of an image is accessed by img.shape. It returns a tuple of the number of rows, columns, and channels (if the image is color):

Code:
```
print( img.shape )
```
Output:
```
(342, 548, 3)
```

- Total number of pixels is accessed by

Code:
```
print( img.size )
```
Output:
```
562248
```

- Image datatype is obtained by `img.dtype`

Code:

```
print( img.dtype )
Output:
uint8
```

c) Splitting and Merging Image Channels:

- Sometimes you will need to work separately on the B,G,R channels of an image. In this case, you need to split the BGR image into single channels. In other cases, you may need to join these individual channels to create a BGR image. You can do this simply by:

```
b,g,r = cv2.split(img)
img = cv2.merge((b,g,r))
```

Or

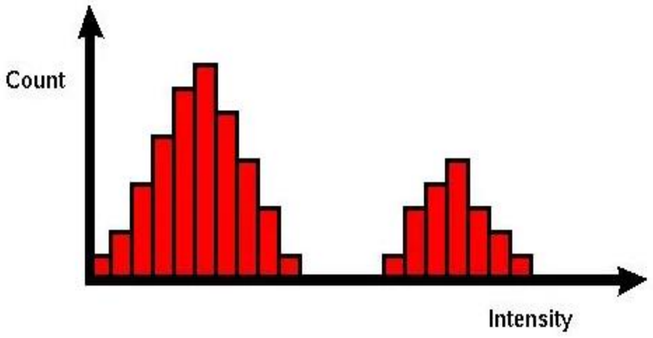
```
b = img[:,:,0]
```

- Suppose you want to set all the red pixels to zero - you do not need to split the channels first. Numpy indexing is faster:

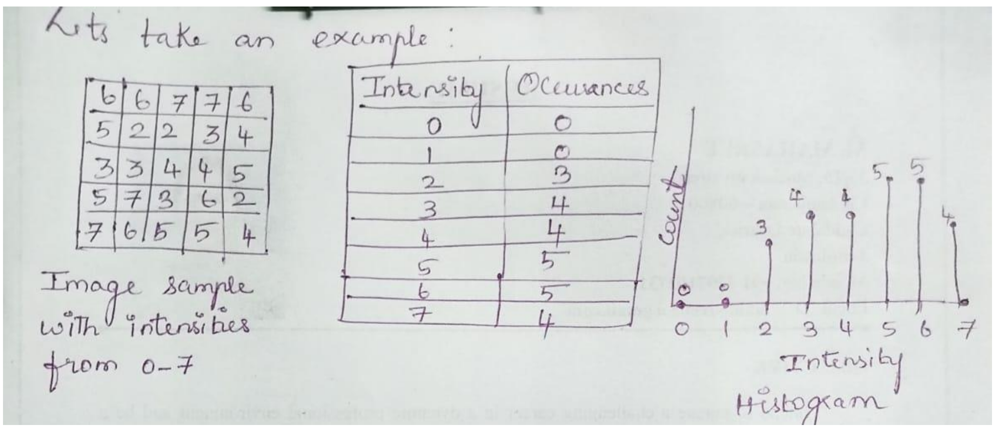
```
img[:,:,2] = 0
```

Histogram

- A histogram of an image is the graphical interpretation of the image's pixel intensity values. It can be interpreted as the data structure that stores the frequencies of all the pixel intensity levels in the image.



- Here, X-axis represents the pixel intensity levels of the image. The intensity level usually ranges from 0 to 255. For a gray-scale image, there is only one histogram, whereas an RGB colored image will have three 2-D histograms — one for each color. The Y-axis of the histogram indicates the frequency or the number of pixels that have specific intensity values.



Lets take an example :

6	6	7	7	6
5	2	2	3	4
3	3	4	4	5
5	7	3	6	2
7	6	5	5	4

Image sample with intensities from 0-7

Intensity	Occurances
0	0
1	0
2	3
3	4
4	4
5	5
6	5
7	4

Histogram equalization

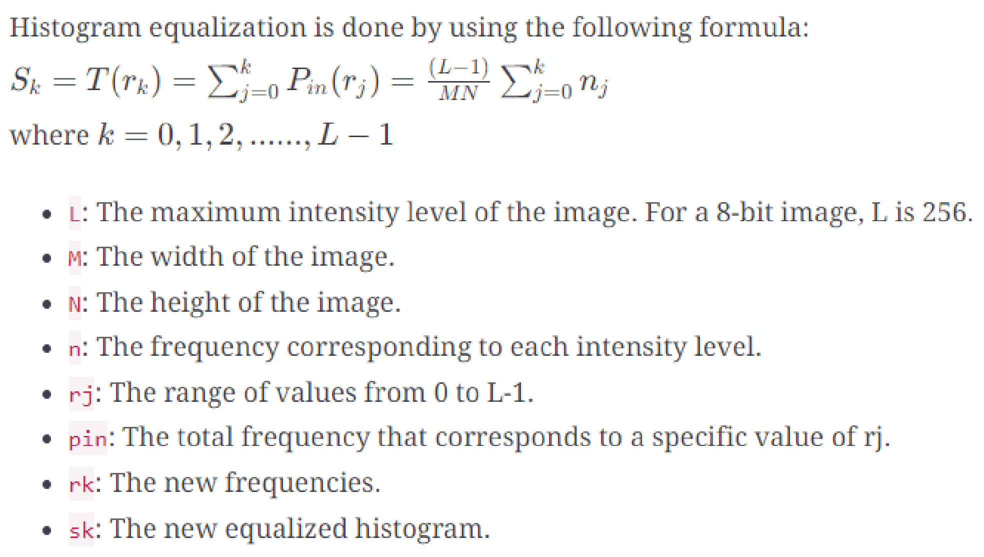
- Histogram Equalization is an image processing technique that adjusts the contrast of an image by using its histogram.
- To enhance the image’s contrast, it spreads out the most frequent pixel intensity values or stretches out the intensity range of the image.
- By accomplishing this, histogram equalization allows the image’s areas with lower contrast to gain a higher contrast.

Steps Involved

- Get the input image
- Generate the histogram for the image
- Find the local minima of the image
- Divide the histogram based on the local minima
- Have the specific gray levels for each partition of the histogram
- Apply the histogram equalization on each partition

Algorithm

- Compute the histogram of pixel values of the input image. The histogram places the value of each pixel $f[x,y]$ into one of L uniformly-spaced buckets $h[i]$ where L=2^8 and the image dimension is $M\times N$
- Calculate the cumulative distribution function
- Scale the input image using the cumulative distribution function to produce the output image.
- Where CDFmin is the smallest non-zero value of the cumulative distribution function

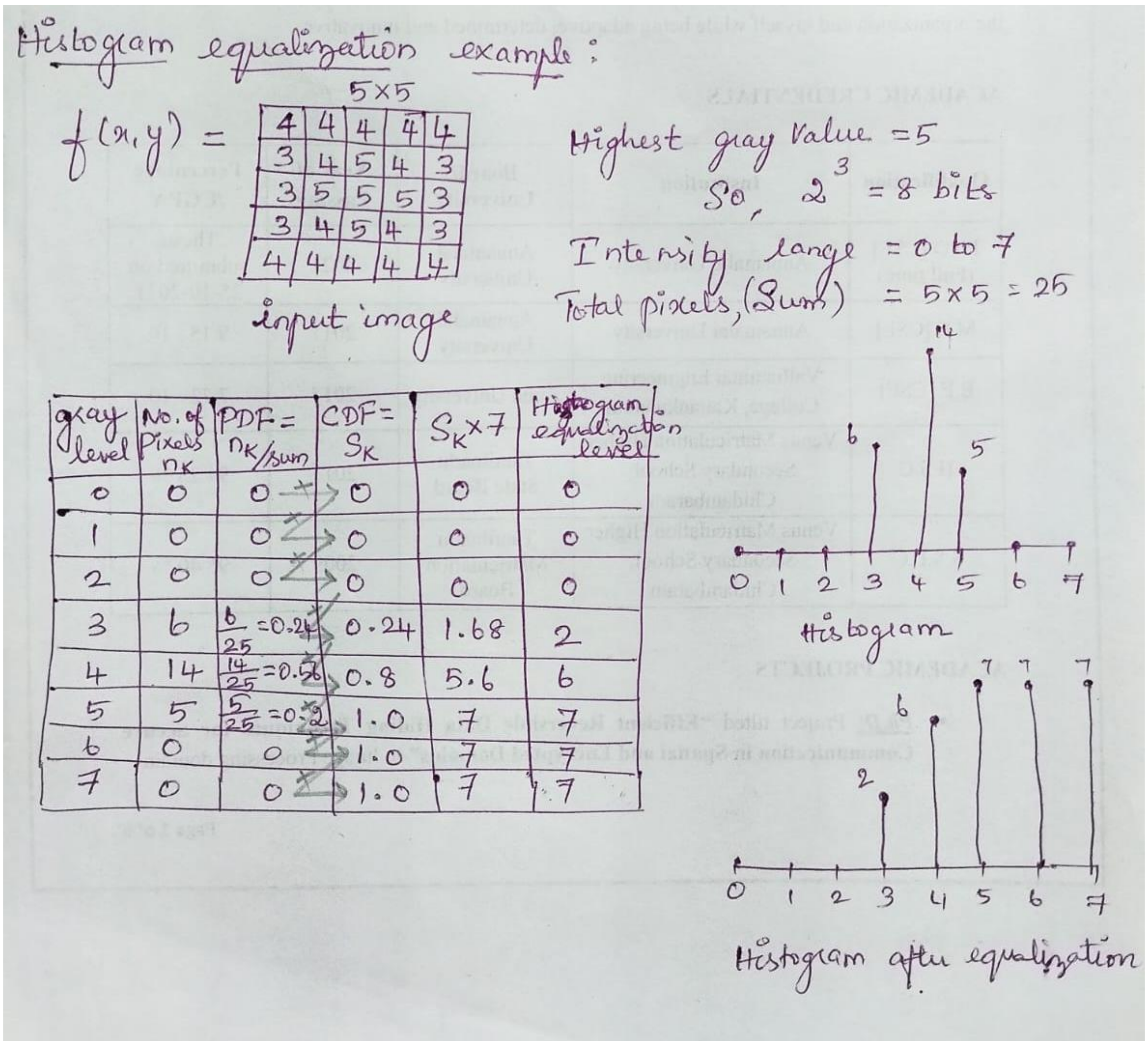


Histogram equalization is done by using the following formula:

$$S_k = T(r_k) = \sum_{j=0}^{k} P_{in}(r_j) = \frac{(L-1)}{MN} \sum_{j=0}^{k} n_j$$

where $k = 0, 1, 2, \ldots\ldots, L-1$

- `L`: The maximum intensity level of the image. For a 8-bit image, L is 256.
- `M`: The width of the image.
- `N`: The height of the image.
- `n`: The frequency corresponding to each intensity level.
- `rj`: The range of values from 0 to L-1.
- `pin`: The total frequency that corresponds to a specific value of rj.
- `rk`: The new frequencies.
- `sk`: The new equalized histogram.



Histogram equalization example:

5×5

$$f(x,y) = \begin{array}{|c|c|c|c|c|} \hline 4 & 4 & 4 & 4 & 4 \\ \hline 3 & 4 & 5 & 4 & 3 \\ \hline 3 & 5 & 5 & 5 & 3 \\ \hline 3 & 4 & 5 & 4 & 3 \\ \hline 4 & 4 & 4 & 4 & 4 \\ \hline \end{array}$$

input image

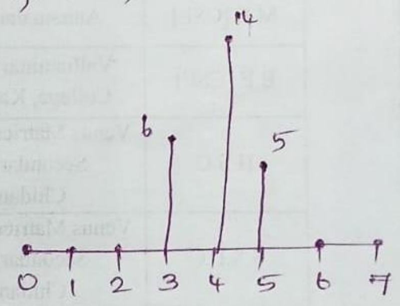
Highest gray value = 5

So, 2^3 = 8 bits

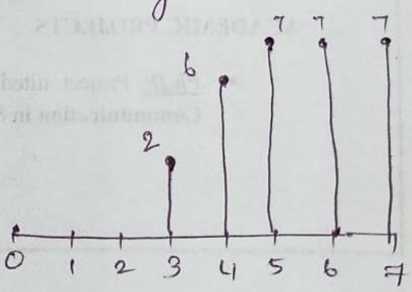
Intensity range = 0 to 7

Total pixels, (Sum) = 5×5 = 25

gray level	No. of Pixels n_K	PDF = n_K/sum	CDF = S_K	S_K×7	Histogram equalization level
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	6	$\frac{6}{25}$=0.24	0.24	1.68	2
4	14	$\frac{14}{25}$=0.56	0.8	5.6	6
5	5	$\frac{5}{25}$=0.2	1.0	7	7
6	0	0	1.0	7	7
7	0	0	1.0	7	7

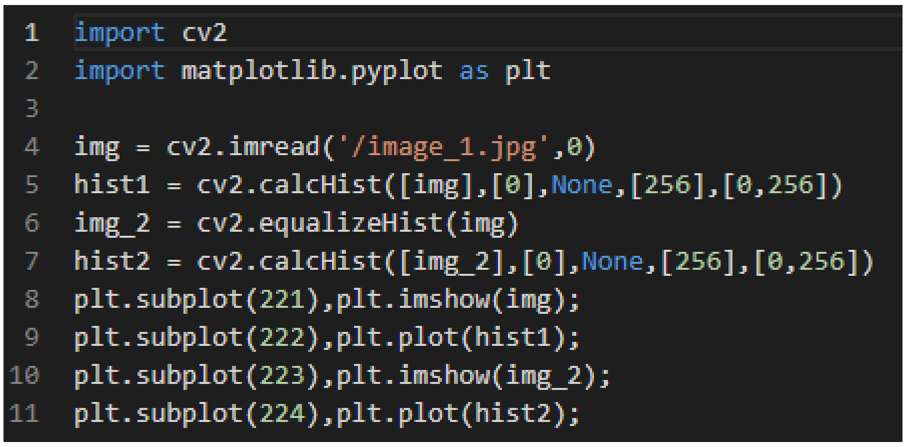


Histogram



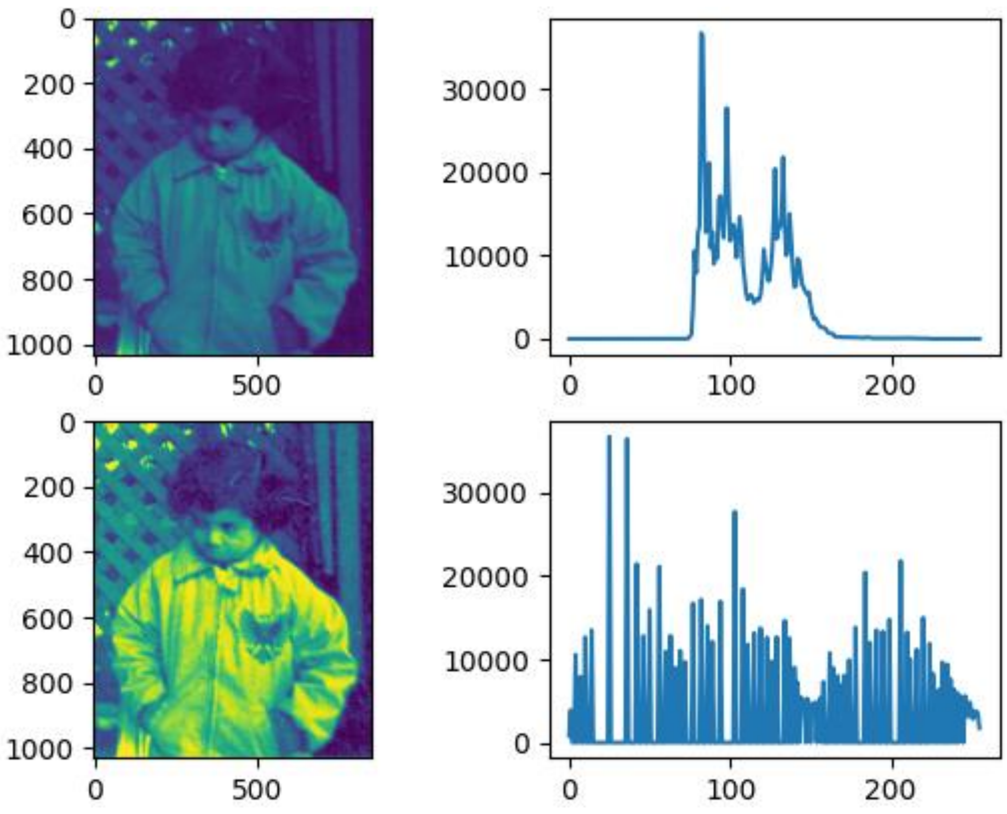
Histogram after equalization

Python Code



```python
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('/image_1.jpg',0)
hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
img_2 = cv2.equalizeHist(img)
hist2 = cv2.calcHist([img_2],[0],None,[256],[0,256])
plt.subplot(221),plt.imshow(img);
plt.subplot(222),plt.plot(hist1);
plt.subplot(223),plt.imshow(img_2);
plt.subplot(224),plt.plot(hist2);
```

Output



Edge Detection

- Edge detection is a technique of image processing used to identify points in a digital image with discontinuities, simply to say, sharp changes in the image brightness. These points where the image brightness varies sharply are called the edges (or boundaries) of the image.

Why do we need Edge Detection?

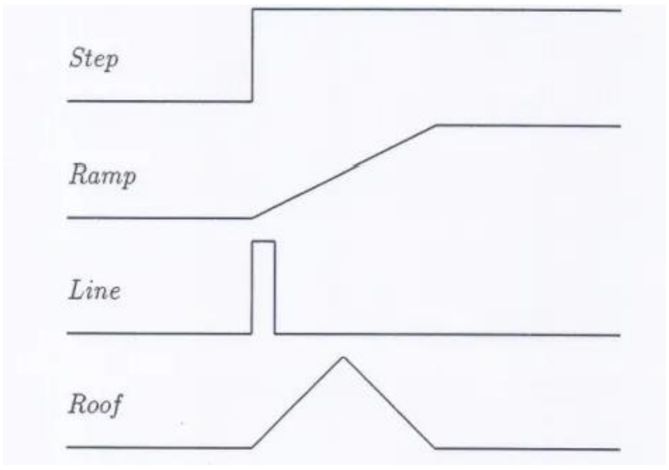
- Discontinuities in depth, surface orientation, scene illumination variations, and material properties changes lead to discontinuities in image brightness. We get the set of curves that indicate the boundaries of objects and surface markings, and curves that correspond to discontinuities in surface orientation.
- Thus, applying an edge detection algorithm to an image may significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant while preserving the important structural properties of an image.

Edge types/profiles:

- Discontinuities in the image intensity can be either
  - *Step discontinuities*, where the image intensity abruptly changes from one value on one side of the discontinuity to a different value on the opposite side,
  - *Line discontinuities*, where the image intensity abruptly changes value but then returns to the starting value within some short distance. However, step and line

edges are rare in real images. Because of low-frequency components or the smoothing introduced by most sensing devices, sharp discontinuities rarely exist in real signals.

- Step edges become *ramp edges* and line edges become *roof edges*, where intensity changes are not instantaneous but occur over a finite distance.



Applications of Image Edge Detection:

- medical imaging, study of anatomical structure
- locate an object in satellite images
- automatic traffic controlling systems
- face recognition, and fingerprint recognition

There are many different edge detection methods, the majority of which can be grouped into two categories:

- *Gradient* - The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image.
- *Laplacian* - The Laplacian method searches for zero crossings in the second derivative of the image.

Edge detection is a major application for convolution. Images contain noise, which also generates sudden transitions of pixel values. Usually there are three steps in the edge detection process:

1) Noise reduction

   - Suppress as much noise as possible without removing edges.

2) Edge enhancement

   - Highlight edges and weaken elsewhere.

3) Edge localisation

   - Look at possible edges and eliminate spurious edges

Sobel Filter

- The Sobel filter is used for edge detection.
- It works by calculating the gradient of image intensity at each pixel within the image.
- It finds the direction of the largest increase from light to dark and the rate of change in that direction.
- The result shows how abruptly or smoothly the image changes at each pixel, and therefore how likely it is that that pixel represents an edge.
- It also shows how that edge is likely to be oriented.
- The sobel filter uses two 3 x 3 kernels. One for changes in the horizontal direction, and one for changes in the vertical direction.

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

- The two kernels are convolved with the original image to calculate the approximations of the derivatives.
- If we define Gx and Gy as two images that contain the horizontal and vertical derivative approximations respectively, the computations are:

$$G_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} * A \quad \text{and} \quad G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * A$$

Where A is the original source image.

- At each pixel in the image, the gradient approximations given by Gx and Gy are combined to give the gradient magnitude, using:

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

- The gradient's direction is calculated using:

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$

- A value of 0 would indicate a vertical edge that is darker on the left side.

Advantages:

- Simple and time efficient computation
- Very easy at searching for smooth edges

Limitations:

- Diagonal direction points are not preserved always
- Highly sensitive to noise
- Not very accurate in edge detection
- Detect with thick and rough edges does not give appropriate results

A working example of Sobel filter

- Let’s calculate the image gradient using the Sobel filter. Suppose we have the same 4*4 image.

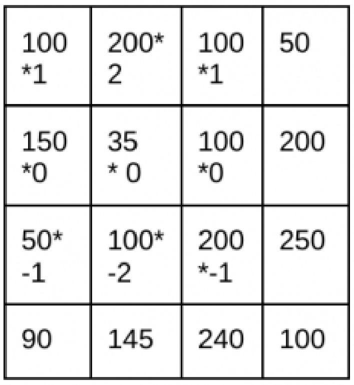
100	200	100	50
150	35	100	200
50	100	200	250
90	145	240	100

- Take both the Gx and Gy filters and convolute them over the image.
- The gradient in x direction :

100 *-1	200 *0	100 *1	50
150 *-2	35 * 0	100 *2	200
50* -1	100 *0	200 *1	250
90	145	240	100

```
Gx = 100 *(-1) + 200*0 + 100*1 + 150*(-2) + 35*0 + 100*2 + 50*(-1) + 100*0 + 200*1
Gx = 50
```

- The gradient in y direction :



100 *1	200* 2	100 *1	50
150 *0	35 * 0	100 *0	200
50* -1	100* -2	200 *-1	250
90	145	240	100

```
Gy = 100 *1 + 200*2 + 100*1 + 150*0 + 35*0 +100*0 + 50*(-1) + 100*(-2) + 200*(-1)
Gy = 150
```

- Calculate gradient strength and gradient orientation:

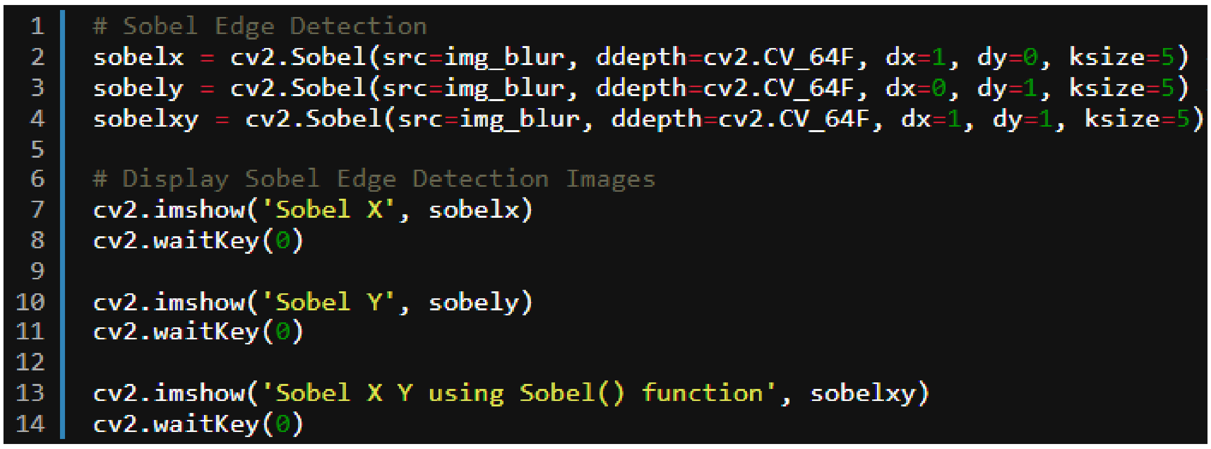
  Gradient magnitude = √(Gx)^2 + (Gy)^2

  = √(50)^2 + (150)^2 = √ ≅ 58

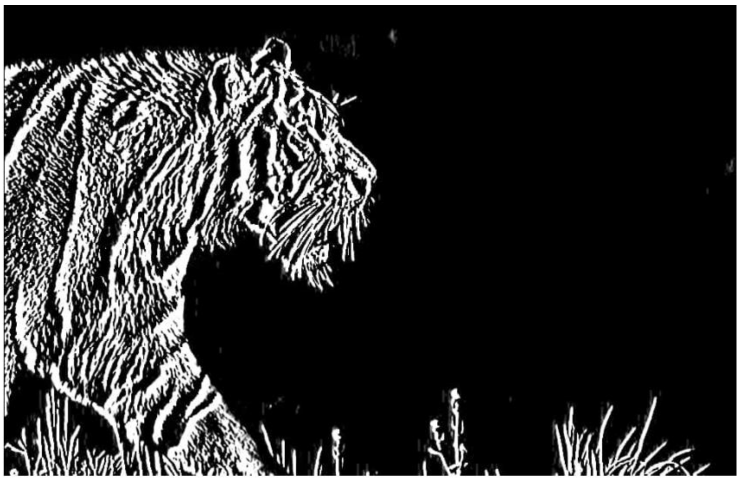
- Using the arctan2 function of NumPy to find the gradient orientation

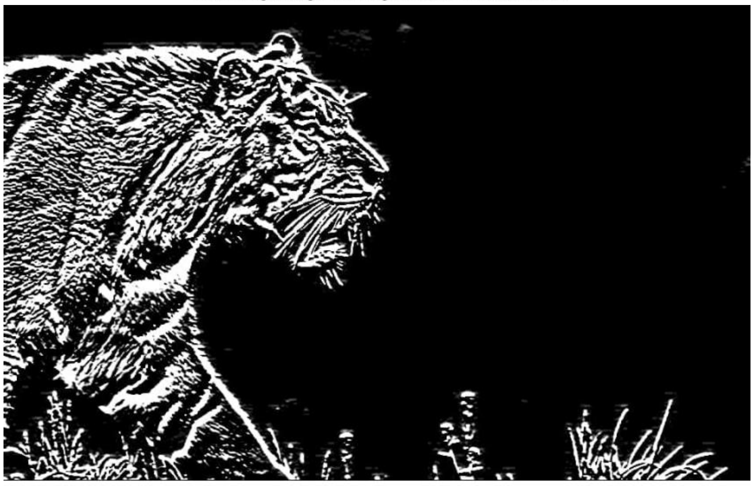
  Gradient Orientation = np.arctan2(Gy / Gx) * (180/ π)

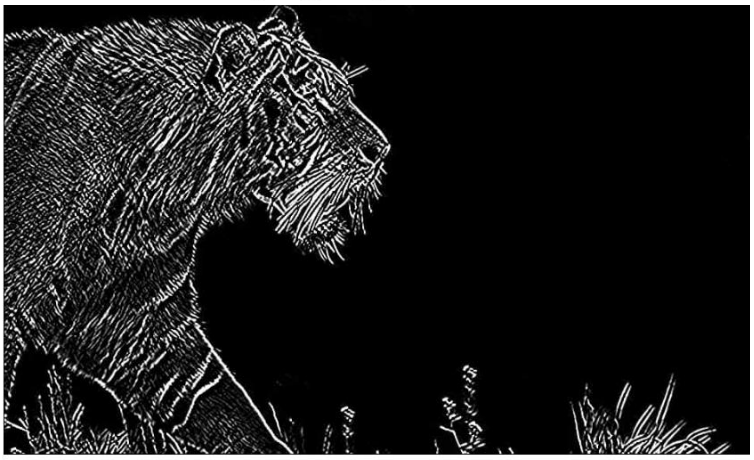
  = 71.5650



```python
# Sobel Edge Detection
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5)
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5)
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5)

# Display Sobel Edge Detection Images
cv2.imshow('Sobel X', sobelx)
cv2.waitKey(0)

cv2.imshow('Sobel Y', sobely)
cv2.waitKey(0)

cv2.imshow('Sobel X Y using Sobel() function', sobelxy)
cv2.waitKey(0)
```

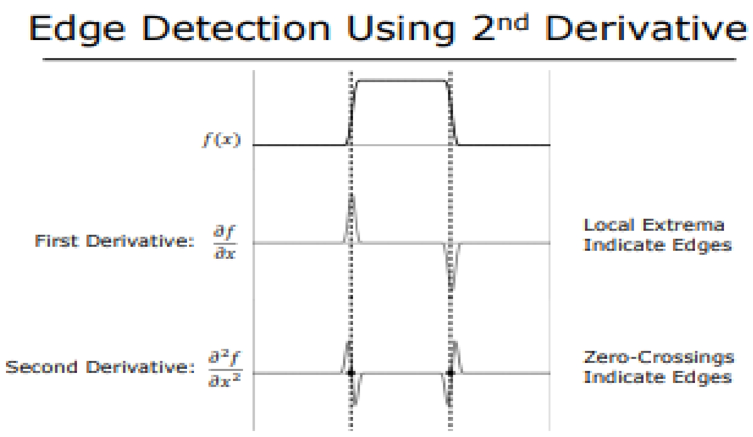

Sobel edge image, with edge enhanced in X-direction


Sobel Edge image, with edge enhanced in Y-direction


Sobel Edge image, with edge enhanced in XY-direction

Laplacian edge detector

- Laplacian Operator is also a derivative operator which is used to find edges in an image.
- The major difference between Laplacian and other operators like Prewitt, Sobel, Robinson and Kirsch is that these all are first order derivative masks but Laplacian is a second order derivative mask.



Edge Detection Using 2nd Derivative
$f(x)$
First Derivative: $\frac{\partial f}{\partial x}$
Local Extrema Indicate Edges
Second Derivative: $\frac{\partial^2 f}{\partial x^2}$
Zero-Crossings Indicate Edges

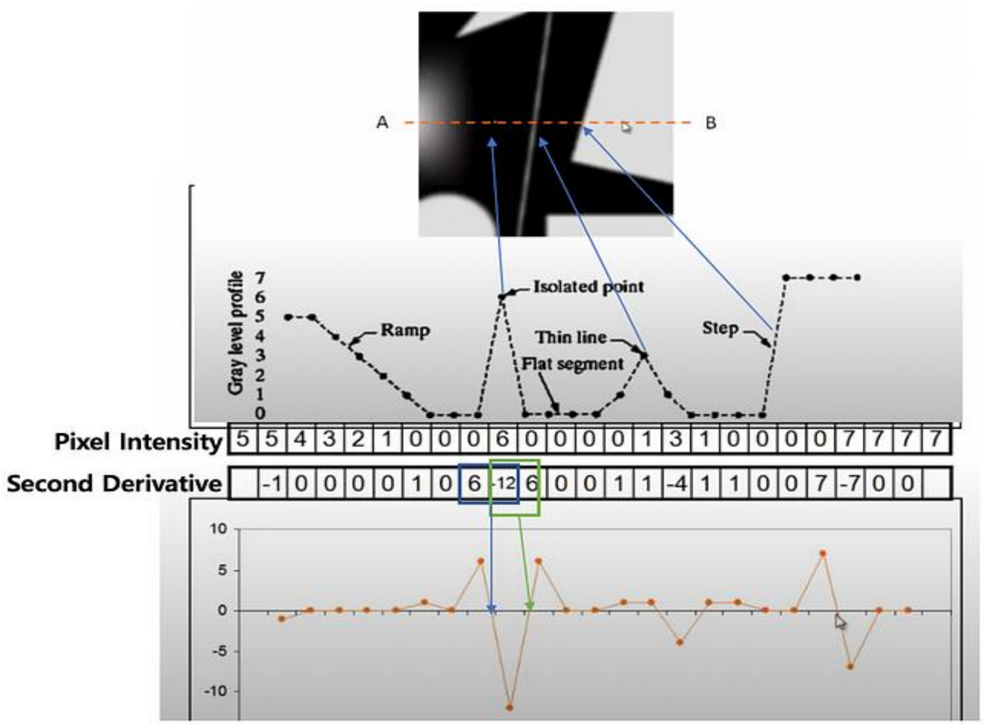
- When the first derivative is increasing, the second derivative will have positive values, and where the first derivative is decreasing, it will have negative values.
- In between, when the first derivative reaches a peak, the second derivative will be zero. Therefore, exactly at the location of the edge, we get a sharp change from positive to negative values, called a *zero-crossing*.

Laplacian as edge detector:

- The mathematical definition of Laplacian is as follows.

$$
\begin{aligned}
Laplacian\ \ \nabla^2 f &= \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \\
&= f(x+1,y) + f(x-1,y) - 2f(x,y) + f(x,y+1) + f(x,y-1) - 2f(x,y) \\
&= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)
\end{aligned}
$$

- Laplacian operator, also called "del-squared" operator is simply the sum of the second derivative of the image with respect to x and the second derivative of the image with respect to y



Pixel Intensity	5	5	4	3	2	1	0	0	0	6	0	0	0	0	1	3	1	0	0	0	0	7	7	7	7
Second Derivative		-1	0	0	0	0	1	0	6	-12	6	0	0	1	1	-4	1	1	0	0	7	-7	0	0	

- First, compute the second derivative for each pixel.
- The goal is to locate the edge locations (pixels), find the Maxima of first derivatives with second derivative. The answer is, edges are at between pixels whose second derivatives sharply transition from one sign to the other sign (zero-crossing).
- The two detected edges, among all edges, are marked with blue and green box in Fig above.
- Two further classifications: one is Positive Laplacian Operator and other is Negative Laplacian Operator.
- Laplacian didn't take out edges in any particular direction but it take out edges in following classification.
    - Inward Edges
    - Outward Edges

Positive Laplacian Operator

In Positive Laplacian we have standard mask in which center element of the mask should be negative and corner elements of mask should be zero. Positive Laplacian Operator is use to take out outward edges in an image.

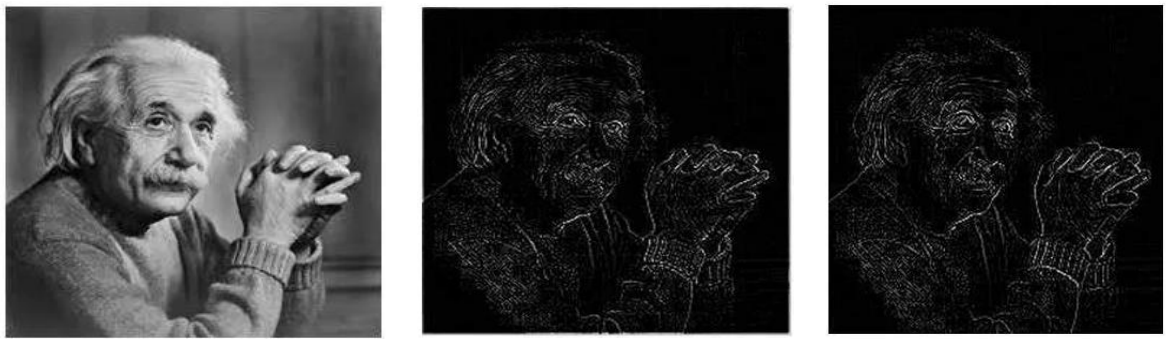
Negative Laplacian Operator

In negative Laplacian operator we also have a standard mask, in which center element should be positive. All the elements in the corner should be zero and rest of all the elements in the mask should be -1. Negative Laplacian operator is use to take out inward edges in an image.

0	1	0
1	−4	1
0	1	0

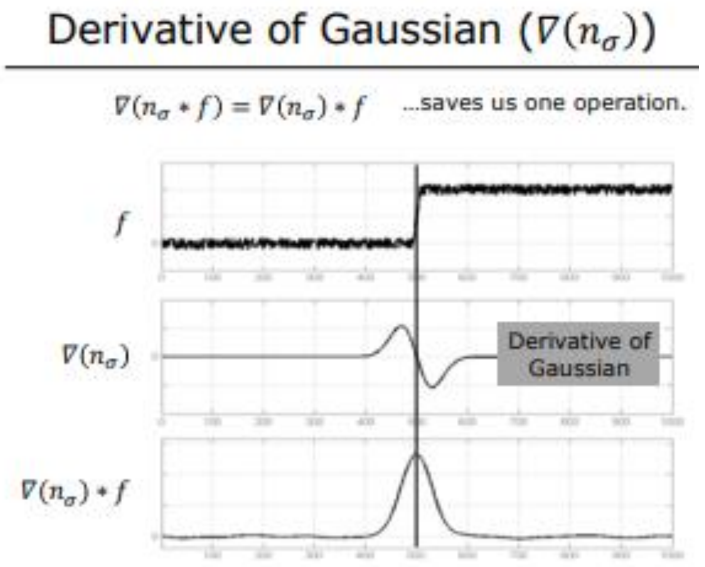
0	−1	0
−1	4	−1
0	−1	0

After applying Positive and Negative Laplacian Operator



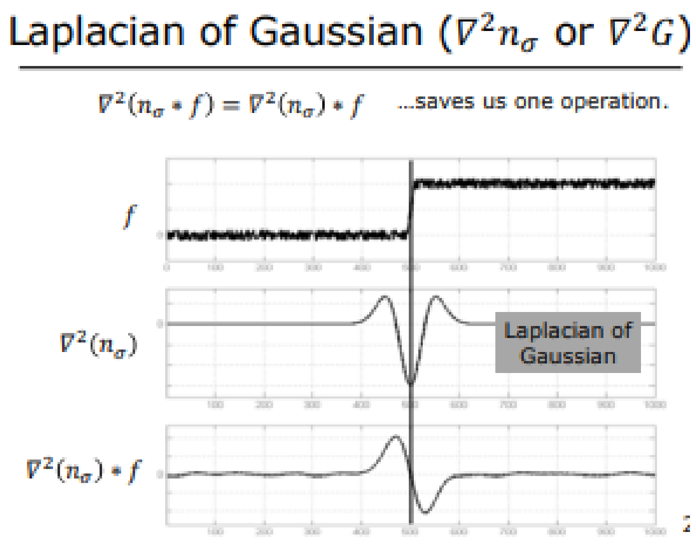
<u>Derivative of Guassian</u>

- The signal f(x) is convolved it with a Gaussian to reduce the noise, and then found its first derivative.
- We can achieve exactly the same result in a different way. We know that the first derivative is a linear operation, and we know that Gaussian smoothing is linear as well.
- Therefore, we can find the first derivative of the Gaussian, which gives us a new operator (filter) that we can convolve our signal with. Comparing the results in below diagram and the previous one, we see that they are exactly the same.



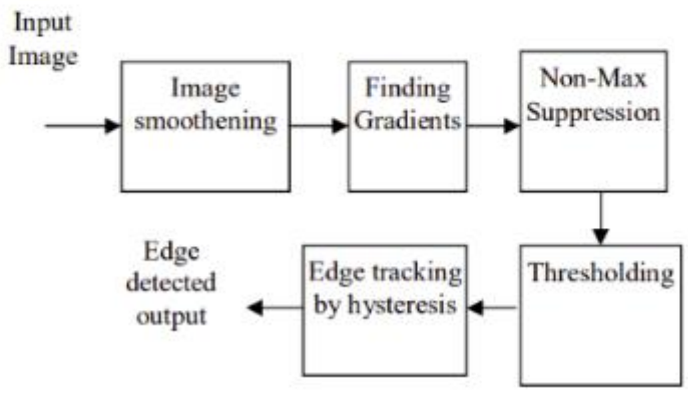
<u>Laplacian of Gaussian</u>

- Instead of applying the Laplacian operator to the input signal after it is smoothed, we can apply the Laplacian operator to the Gaussian to get what is called a Laplacian of Gaussian (LoG) operator, and then simply apply that to the input signal.
- Then, we end up with a strong zero-crossing at the location of the edge.



Canny edge detector

- The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. The Canny edge detection algorithm is composed of 5 steps:
  - *Noise reduction;*
  - *Gradient calculation;*
  - *Non-maximum suppression;*
  - *Double threshold;*
  - *Edge Tracking by Hysteresis.*
- The input image is smoothened, Sobel filter is applied to detect the edges of the image.
- Then we apply non-max suppression where the local maximum pixels in the gradient direction are retained, and the rest are suppressed.
- We apply thresholding to remove pixels below a certain threshold and retain the pixels above a certain threshold to remove edges that could be formed due to noise.
- Later we apply hysteresis tracking to make a pixel strong if any of the 8 neighboring pixels are strong.



- ***Noise reduction / Smoothing:*** Blurring of the image to remove noise.
  - One way to get rid of the noise on the image, is by applying Gaussian blur to smooth it. To do so, image convolution technique is applied with a Gaussian Kernel (3x3, 5x5, 7x7 etc…).
  - The kernel size depends on the expected blurring effect. Basically, the smallest the kernel, the less visible is the blur. For example, we will use a 5 by 5 Gaussian kernel.
  - The equation for a Gaussian filter kernel of size (2k+1)×(2k+1) is given by:

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

Original image (left) — Blurred image with a Gaussian filter (sigma=1.4 and kernel size of 5×5)

- ***Gradient Calculation:*** The edges should be marked where the gradients of the image has large magnitudes
  - The Gradient calculation step detects the edge intensity and direction by calculating the gradient of the image using edge detection operators.
  - Edges correspond to a change of pixels' intensity. To detect it, the easiest way is to apply filters that highlight this intensity change in both directions: horizontal (x) and vertical (y)
  - When the image is smoothed, the derivatives Ix and Iy w.r.t. x and y are calculated. It can be implemented by convolving I with Sobel kernels Kx and Ky, respectively:

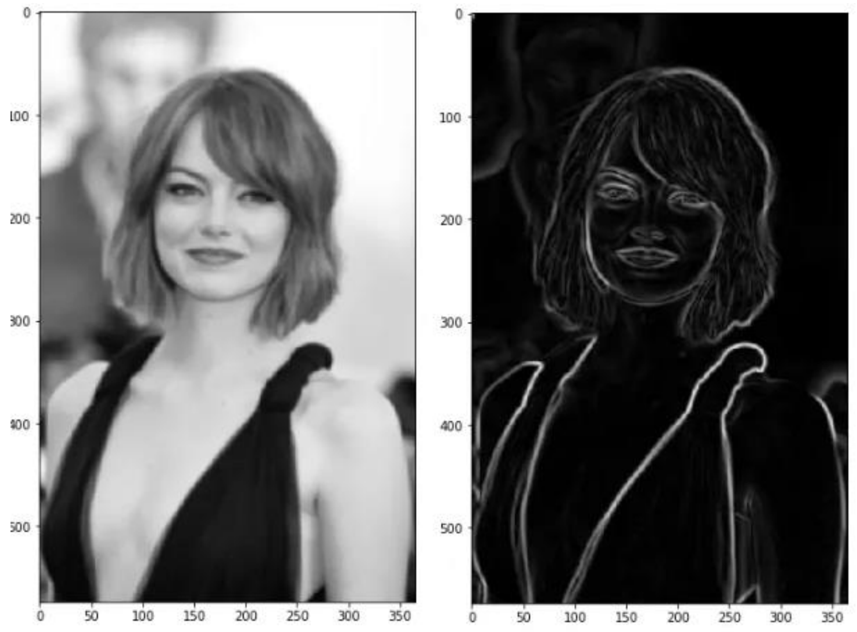
$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Sobel filters for both direction (horizontal and vertical)

  - Then, the magnitude G and the slope θ of the gradient are calculated as follow:

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x, y) = arctan\left(\frac{I_y}{I_x}\right)$$

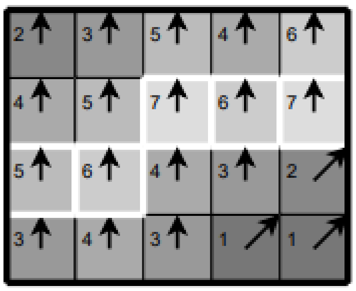
Gradient intensity and Edge direction

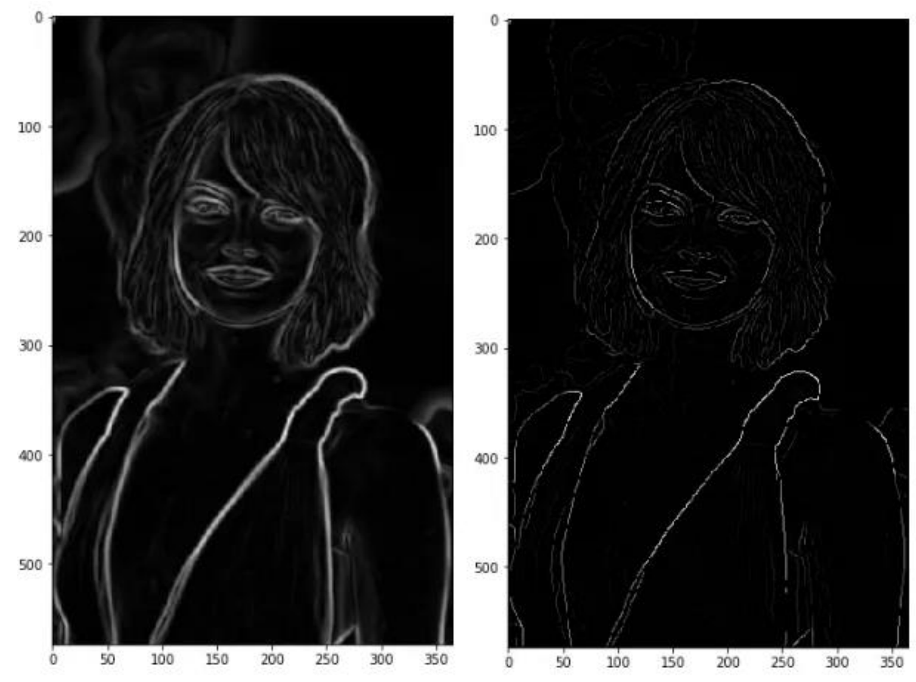


Blurred image (left) — Gradient intensity (right)

- ***Non-maximum suppression:*** Only local maxima should be marked as edges

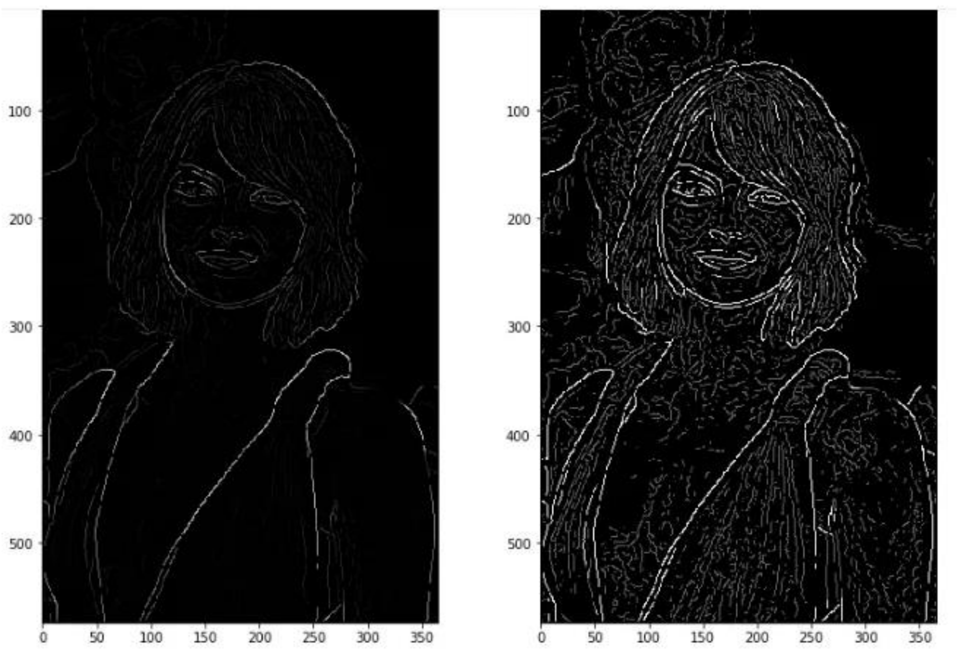
- The purpose of this step is to convert the “blurred” edges in the image of the gradient magnitudes to “sharp” edges.
- Basically this is done by preserving all local maxima in the gradient image, and deleting everything else.
- The algorithm is for each pixel in the gradient image:
  1. Round the gradient direction θ to nearest 45◦ corresponding to the use of an 8-connected neighbourhood.
  2. Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction. i.e. if the gradient direction is north (theta = 90◦), compare with the pixels to the north and south.
  3. If the edge strength of the current pixel is largest; preserve the value of the edge strength. If not, suppress (i.e. remove) the value.
- A simple example of non-maximum suppression is shown in Figure below.
- Almost all pixels have gradient directions pointing north. They are therefore compared with the pixels above and below.
- The pixels that turn out to be maximal in this comparison are marked with white borders. All other pixels will be suppressed.





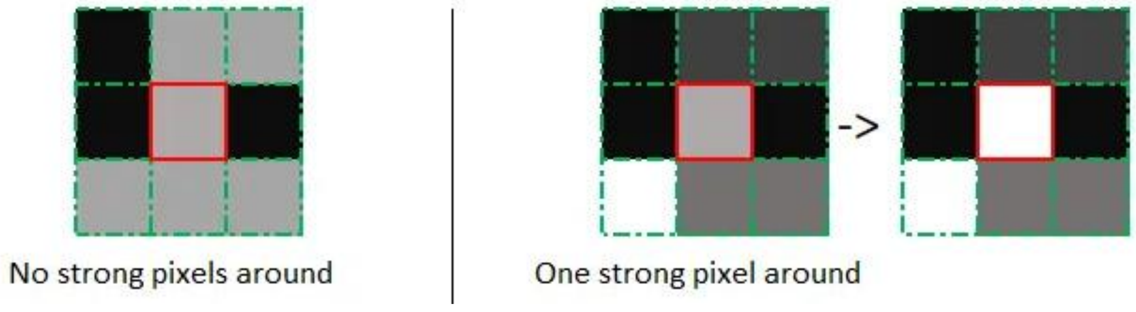
Result of the non-max suppression.

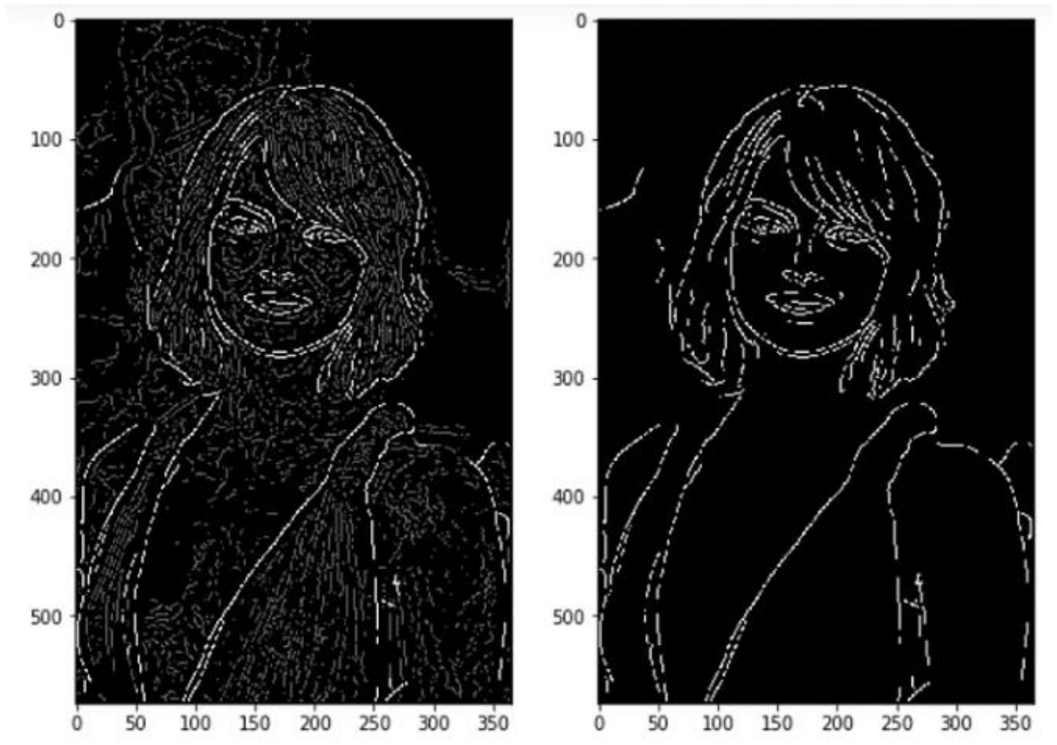
- ***Double thresholding:*** Potential edges are determined by thresholding.
  - The Canny edge detection algorithm uses double thresholding. Edge pixels stronger than the high threshold are marked as strong; edge pixels weaker than the low threshold are suppressed and edge pixels between the two thresholds are marked as weak.
  - The result of this step is an image with only 2 pixel intensity values (strong and weak):



Non-Max Suppression image (left) — Threshold result (right): weak pixels in gray and strong ones in white.

- ***Edge tracking by hysteresis:*** Final edges are determined by suppressing all edges that are not connected to a very certain (strong) edge.
    - Strong edges are interpreted as “certain edges”, and can immediately be included in the final edge image.
    - Weak edges are included if and only if they are connected to strong edges.





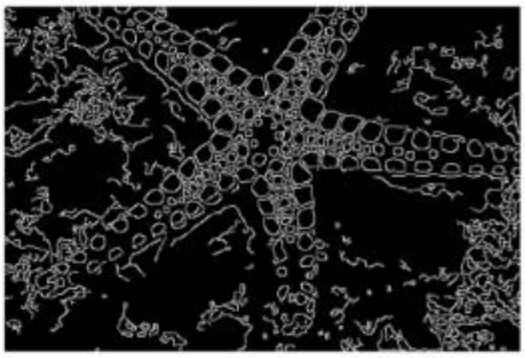
The following code applies a canny edge detector on an image of starfish

```
img_path = 'starfish.png'
#Reading the image
image = cv2.imread(img_path)
(H, W) = image.shape[:2]
# convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# blur the image
```

```
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
# Perform the canny operator
canny = cv2.Canny(blurred, 30, 150)
```

Let’s see the output of the canny edge detector

```
fig,ax =  plt.subplots(1,2,figsize=(18, 18))
ax[0].imshow(gray,cmap='gray')
ax[1].imshow(canny,cmap='gray')
ax[0].axis('off')
ax[1].axis('off')
```



Corner Detection

- Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image.
- A corner can be defined as the intersection of two edges. A corner can also be defined as a point for which there are two dominant and different edge directions in a local neighbourhood of the point.
- *Applications:* Corner detection is frequently used in motion detection, image registration, video tracking, image mosaicing, panorama stitching, 3D reconstruction and object recognition.
- Commonly used methods are Harris Corner Detection and Shi-Tomasi Corner Detection techniques.
- Refer the links below:

https://medium.com/data-breach/introduction-to-harris-corner-detector-32a88850b3f6

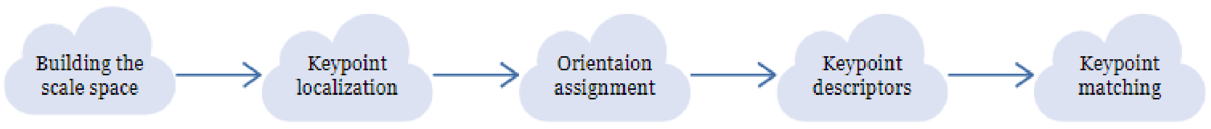
https://www.geeksforgeeks.org/python-corner-detection-with-shi-tomasi-corner-detection-method-using-opencv/

https://www.tutorialspoint.com/implementing-shi-tomasi-corner-detector-in-opencv-python

<u>SIFT feature detection</u>

- ***Scale Invariant Feature Transform (SIFT)*** was introduced by D. Lowe, a former professor at the University of British Columbia, in the year 2004.
- It is a technique for detecting salient, stable feature points in an image.
- For every such point, it also provides a set of "features" that "characterize/describe" a small image region around the point.
- These features are invariant to rotation and scale.

Steps of execution



Overview of the SIFT Algorithm

1. Building the scale space:

- We need to identify the most distinct features in a given input image while ignoring any noise. Additionally, we need to ensure that the features are not scale-dependent.
- For every pixel in an image***, the Gaussian Blur*** calculates a value based on its neighboring pixels with a certain sigma value.
- After applying the Gaussian blur, the texture and minor details are removed from the image, and only the relevant information, like the shape and edges, remain.

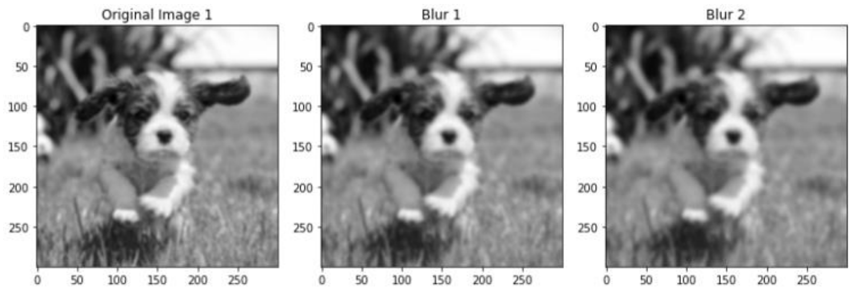
$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

Gaussian Blur operator

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

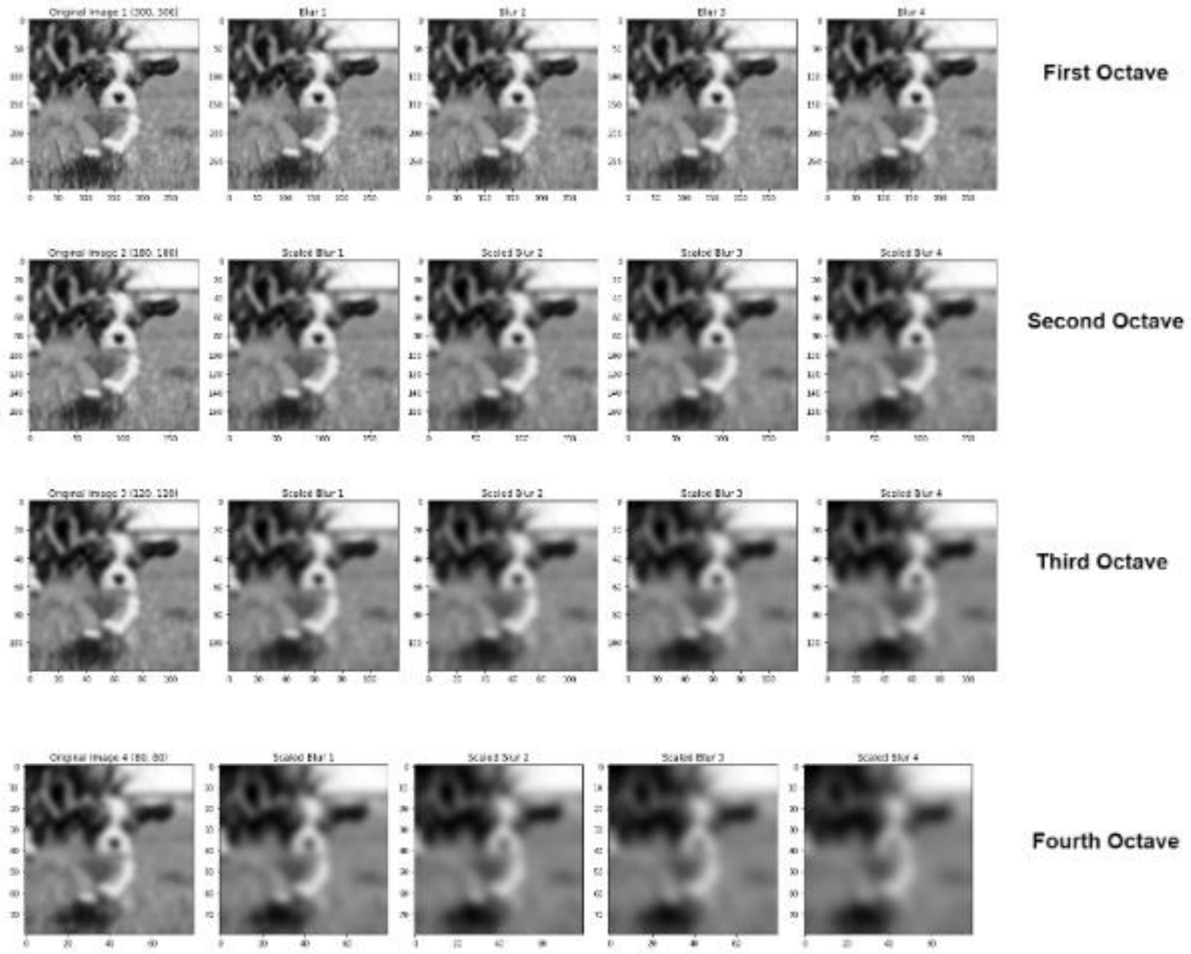
Blurred image

The symbols:
L is a blurred image
G is the Gaussian Blur operator
I is an image
x,y are the location coordinates
σ is the "scale" parameter. Think of it as the amount of blur. Greater the value, greater the blur.
* is the convolution operation in x and y. It "applies" gaussian blur G onto image I.



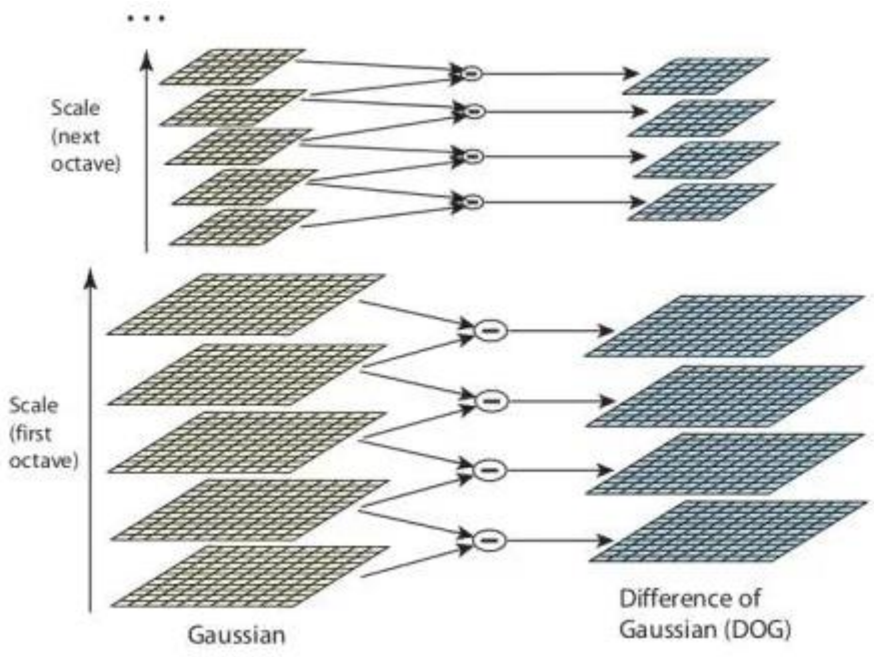
Test image after Gaussian blur is applied

- Scale space is a collection of images having different scales, generated from a single 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.



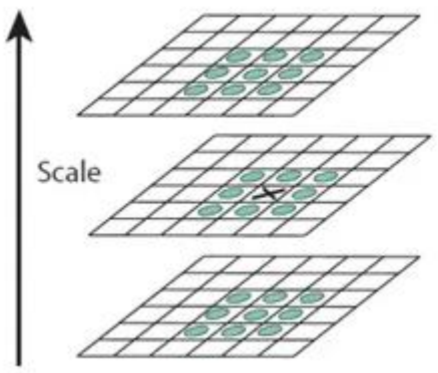
Visualization of 4 Octaves

- After we create the scale-space, we must now amplify the features using a technique called ***Difference of Gaussian (DoG).***
- DoG subtracts the higher blurred version of an original image from the lower blurred version, which creates another set of images in the same scale for each octave, respectively.



2. Key point localization:

- Once the images have been created, the next step is to find the important keypoints from the image that can be used for feature matching.
- The idea is to find the local maxima and minima for the images. This part is divided into two steps:
    - Find the local maxima and minima
    - Remove low contrast keypoints (keypoint selection)
- To locate the local maxima and minima, we go through every pixel in the image and compare it with its neighboring pixels.
- This means that every pixel value is compared with 26 other pixel values to find whether it is the local maxima/minima called extrema.
- For example, in the below diagram, we have three images from the first octave. The pixel marked x is compared with the neighboring pixels (in green) and is selected as a keypoint or interest point if it is the highest or lowest among the neighbors:



- If it is a local extrema, it is a ***potential keypoint***. It basically means that keypoint is best represented in that scale.

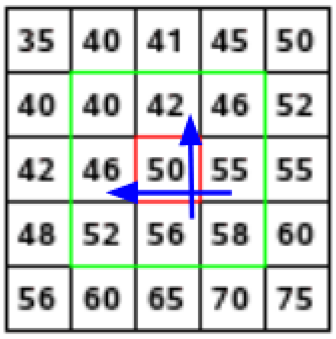
3. Orientation Assignment:

- At this stage, we have a set of stable keypoints for the images. The next step is to assign an orientation to each of these keypoints so that they are invariant to rotation. It is again divided into two smaller steps:
    - Calculate the magnitude and orientation
    - Create a histogram for magnitude and orientation
- *Calculate Magnitude and Orientation*
    - The formula used for calculating the magnitude and orientation is as follows:

$$Magnitude = \sqrt{(G_x^2 + G_y^2)}$$

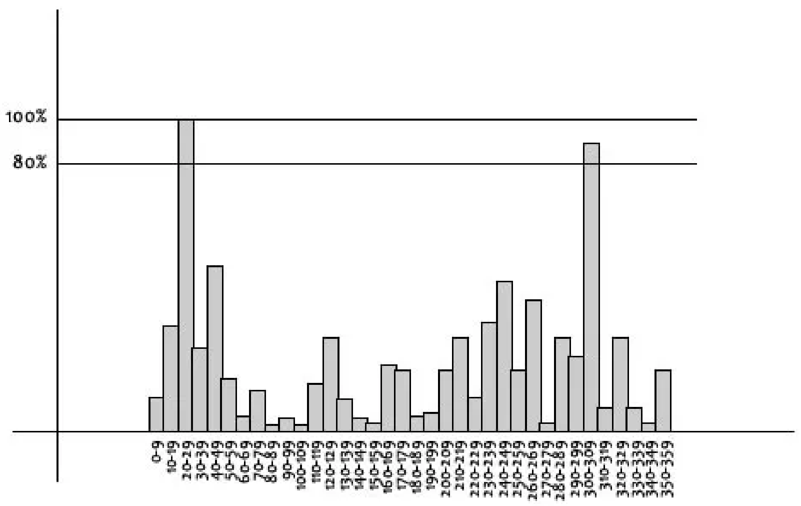
$$Orientation = arctan(G_y/G_x)$$

    - Consider the sample image shown below:



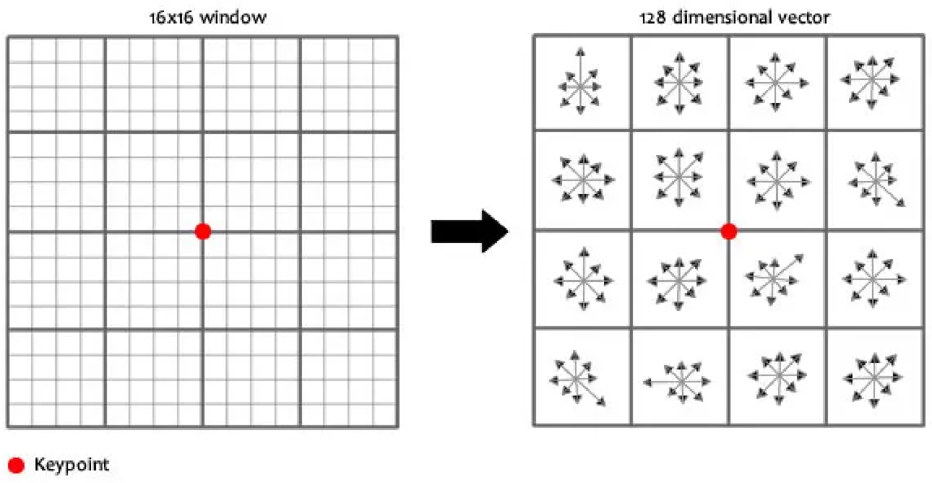
35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

    - Let's say we want to find the magnitude and orientation for the pixel value in red. For this, we will calculate the gradients in the x and y directions by taking the difference between 55 & 46 and 56 & 42. This comes out to be Gx = 9 and Gy = 14, respectively.
    - Once we have the gradients, we can find the magnitude and orientation using the following formulas:
        - Magnitude = √[(Gx)2+(Gy)2] = 16.64
        - Φ = atan(Gy / Gx) = atan(1.55) = 57.17
    - The magnitude represents the intensity of the pixel and the orientation gives the direction for the same.
- *Creating a Histogram for Magnitude and Orientation*
    - On the x-axis, we will have bins for angle values, like 0-9, 10 – 19, 20-29, and up to 360.
    - Since our angle value is 57, it will fall in the 6th bin. The 6th bin value will be in proportion to the magnitude of the pixel, i.e. 16.64. This is repeated for all the pixels around the keypoint.
    - This is shown in below histogram.
    - The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contributes to the stability of matching.

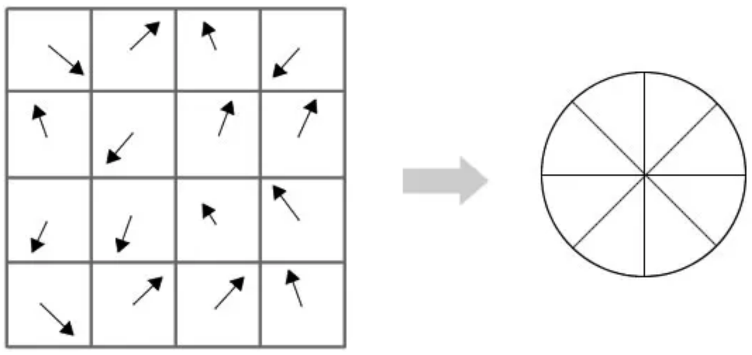


4. Keypoint descriptor:

- At this point, each keypoint has a location, scale, orientation. Next is to compute a descriptor for the local image region about each keypoint that is highly distinctive and invariant as possible to variations such as changes in viewpoint and illumination.
- To do this, a 16x16 window around the keypoint is taken. It is divided into 16 sub-blocks of 4x4 size.



- For each sub-block, 8 bin orientation histogram is created.



- So 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 X 8 directions give 128 bin values. It is represented as a feature vector to form ***keypoint descriptor.***
- This feature vector introduces a few complications. Before finalizing, get rid of them.
- *Rotation dependence:* The feature vector uses gradient orientations. Clearly, if you rotate the image, everything changes. All gradient orientations also change. To achieve rotation independence, the keypoint's rotation is subtracted from each orientation. Thus each gradient orientation is relative to the keypoint's orientation.
- *Illumination dependence:* If we threshold numbers that are big, we can achieve illumination independence. So, any number (of the 128) greater than 0.2 is changed to 0.2. This resultant feature vector is normalized again. And now we have an illumination independent feature vector.

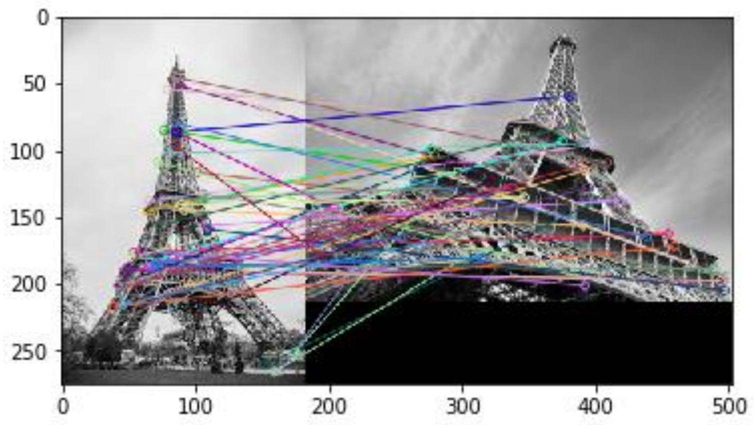
5. Keypoint Matching:

- Keypoints between two images are matched by identifying their nearest neighbors.
- But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, the ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected.
- It eliminates around 90% of false matches while discards only 5% correct matches

Python Code

```
Import cv2
import matplotlib.pyplot as plt
%matplotlib inline
# read images
img1 = cv2.imread('eiffel_2.jpeg')
img2 = cv2.imread('eiffel_1.jpg')
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
#sift
sift = cv2.xfeatures2d.SIFT_create()
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)
#feature matching
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)
matches = bf.match(descriptors_1,descriptors_2)
matches = sorted(matches, key = lambda x:x.distance)
img3 = cv2.drawMatches(img1, keypoints_1, img2, keypoints_2, matches[:50], img2, flags=2)
plt.imshow(img3),plt.show()
```

Output



<u>Star Feature Detector</u>

- Star Feature Detector is derived from CenSurE (Center Surrounded Extrema) detector. While CenSurE uses polygons such as Square, Hexagon and Octagons as a more computable alternative to circle.
- Star mimics the circle with 2 overlapping squares: 1 upright and 1 45-degree rotated. These polygons are bi-level. The can be seen as polygons with thick borders. The borders and the enclosed area have weights of opposing signs.
- It works in gray scale only

<u>Detection</u>

- *Build Integral Image*
  - Square and/or Slanted-variants (trapezoids). The slanted ones are used to compute polygon shaped filters.
- *LoG (Laplacian of Gaussian)*
  - Approximate LoG (Laplacian of Gaussian) with choice of the bi-level polygons. The outer strip and inner strip has opposing weights. It is supposed to produce zero DC value. Meaning the ratio of the weights should somehow be related to the area under the 2 levels.
  - Let m, n be parameters that defines of the inner region and outer boundary (thickness, sort-of).
  - A set of filters will be defined, based on the pairs of (m, n) values.
  - Repeat the convolution on the same input image with each of the scaled filter. There will be no-subsampling, therefore no need to localize points from higher-scales.
- *Non Maxima Suppression*
  - Feature corners are filter-response maximas in a 3x3x3 neighbors.
  - Eliminate weak feature points - with values below a filter response threshold.
- *Edge Suppression*
  - Remove unstable points on edges by examining each feature corner with Harris measure under a 9x9 window.
  - Eliminate points having the high ratio of the two highest Principle Curvatures, above a predefined threshold.

<u>Descriptor</u>

- The CenSurE paper suggests to use improved Upright SURF: MU-SURF. That reduces the 'boundary effect' by making oversized masks when calculating Haars wavelet responses.
- It also weighted the wavelet responses twice with Gaussian.
- Shares the characteristics of SURF descriptors in terms of quick matching with +/- signs.

<u>Sample (detector_simple)</u>

- *Parameters (by observation)*
    - **maxSize** - choose the number of filters to be applied, the parameter value set the maximum size (last filter).
    - **responseThreshold -** eliminate weak corners.
    - **lineThresholds:** maximum ratio between some factors that made up the Harris messure
    - **lineThresholdProjected** (first test) - Harris of responses
    - **lineThresholdBinarized** (second test) - Harris of sizes (refer to size1[]). Seems like size1 is simply a copy of size0 up to the max scale currently used.
    - Has to satisfy both tests to avoid elimination.
    - **suppressNonMaxSize** - window size (n-by-n) to apply the non-maximal suppression.
- *Observations*
    - Detect fewer points on human faces with default parameters comparing to SURF/SIFT.
    - **maxSize** @ default value (45) detects the most points on Kobe's Picture. Not apply to all images. But the values towards both ends detects much fewer points.
    - **responseThreshold** - when threshold set to relatively high, only circular objects or good corners like 30-45 degrees (such as necklace pendants, polo-shirt collars) remains.
    - **suppressNonMaxSize** - increasing the window size remove feature points that are close to each other.
    - **increase/decrease the lineThreshold** pairs with the same ratio, seeing fewer/more points along the hairlines.

<u>Performance</u>

- 572 points detected in 1 second