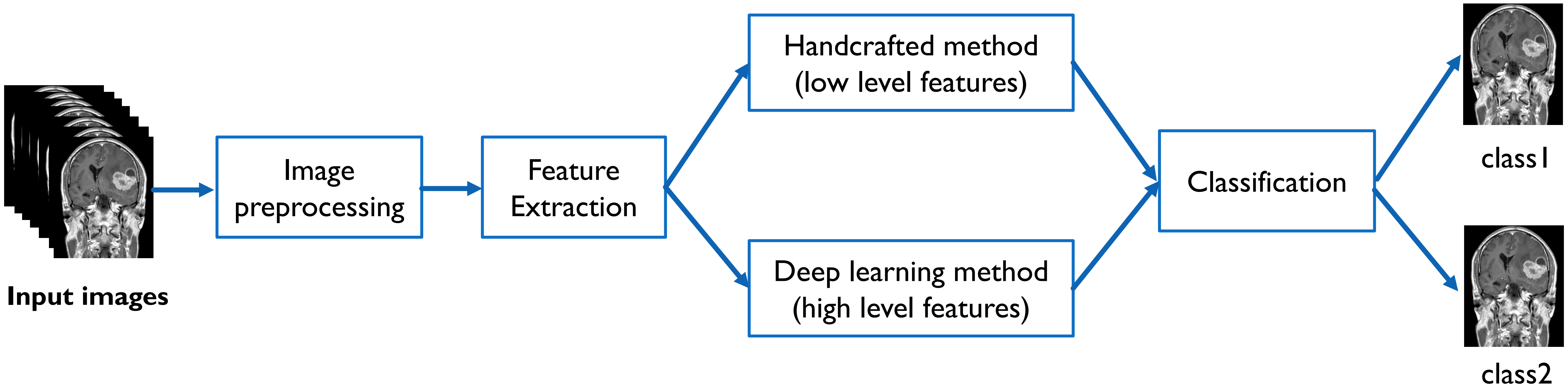
Artificial intelligence

Diseases Classification Stages

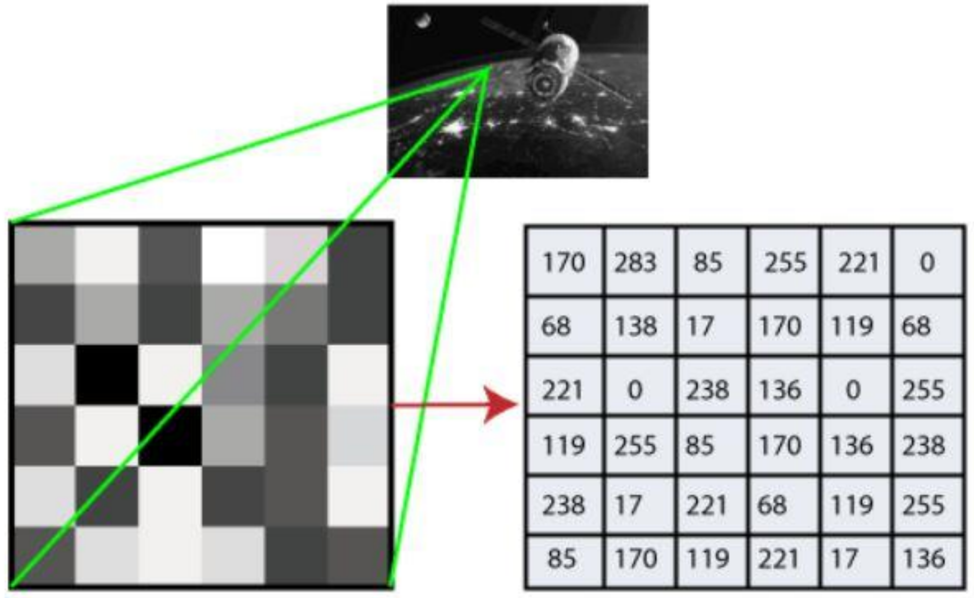

Input images
Image preprocessing
Feature Extraction
Handcrafted method (low level features)
Deep learning method (high level features)
Classification
class1
class2

COMPUTER VISION

- **Computer vision** is a process by which we can understand the images and videos how they are stored and how we can manipulate and retrieve data from them. Computer Vision is the base or mostly used for Artificial Intelligence. Computer-Vision is playing a major role in self-driving cars, robotics as well as in photo correction apps.
- **Applications of Computer Vision:**
  - **Medical Imaging:** Computer vision helps in MRI reconstruction, automatic pathology, diagnosis, and computer aided surgeries and more.
  - **AR/VR:** Object occlusion, outside-in tracking, and inside-out tracking for virtual and augmented reality.
  - **Smartphones:** All the photo filters (including animation filters on social media), QR code scanners, panorama construction, Computational photography, face detectors, image detectors like (Google Lens, Night Sight) that we use are computer vision applications.
  - **Internet:** Image search, Mapping, photo captioning, Ariel imaging for maps, video categorization and more.

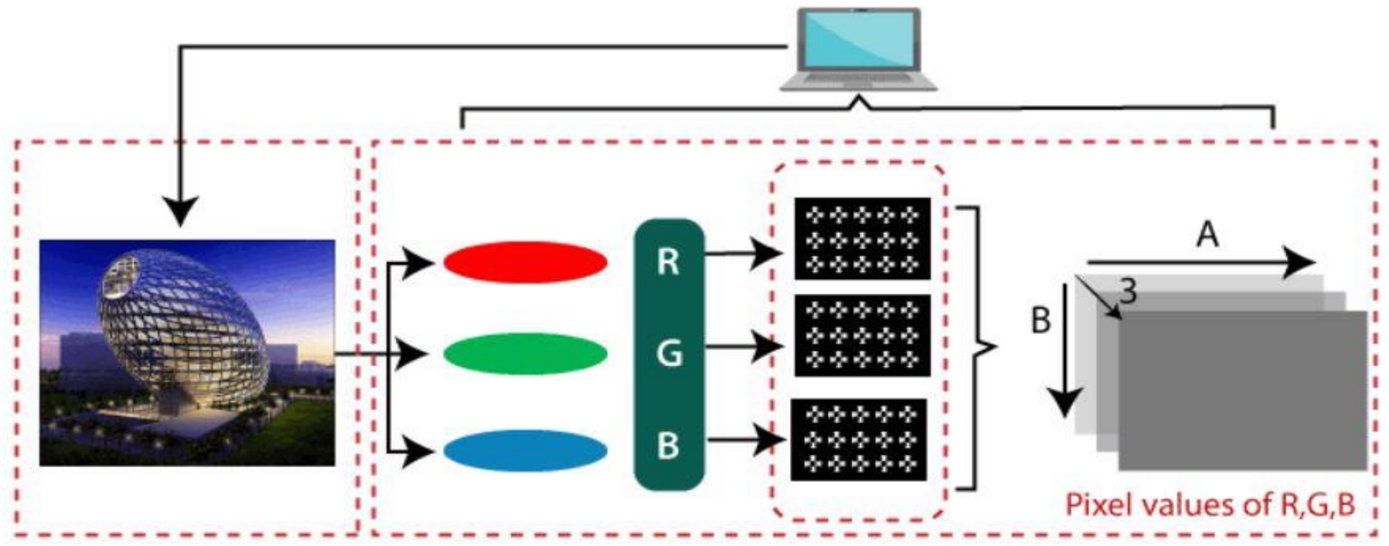
HOW DOES COMPUTER RECOGNIZE ?

- Human eyes provide lots of information based on what they see.
- Machines are facilitated with seeing everything, convert the vision into numbers and store in the memory.
- Here the question arises how computer convert images into numbers. So the answer is that the pixel value is used to convert images into numbers.
- A pixel is the smallest unit of a digital image or graphics that can be displayed and represented on a digital display device.
- A pixel has two values: location and intensity.



170	283	85	255	221	0
68	138	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

- There are two common ways to identify the images:
  - **1. Grayscale**: Grayscale images are those images which contain only two colors black and white. The contrast measurement of intensity is black treated as the weakest intensity, and white as the strongest intensity. When we use the grayscale image, the computer assigns each pixel value based on its level of darkness.
  - **2. RGB**: is a combination of the red, green, blue color which together makes a new color. The computer retrieves that value from each pixel and puts the results in an array to be interpreted.



COMPUTER VISION WITH OPENCV

- **OpenCV (Open Source Computer Vision)**, a cross-platform and free to use library of functions is based on real-time Computer Vision which supports Deep Learning frameworks that aids in image and video processing. In Computer Vision, the principal element is to extract the pixels from the image to study the objects and thus understand what it contains. Below are a few key aspects that Computer Vision seeks to recognize in the photographs:
  - **Object Detection:** The location of the object.
  - **Object Recognition:** The objects in the image, and their positions.
  - **Object Classification:** The broad category that the object lies in.
  - **Object Segmentation:** The pixels belonging to that object.

OPENCV LIBRARY:

- **OpenCV** is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations.
- The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both **academic** and **commercial** use.
- In this course, OpenCV version was 4.0.
- Packages for standard desktop environments (Windows, macOS, almost any GNU/Linux distribution)

  Option 1 - Main modules package: pip install opencv-python

  Option 2 - Full package (contains both main modules and contrib/extra modules): pip install opencv-contrib-python

INSTALLATION ON WINDOWS

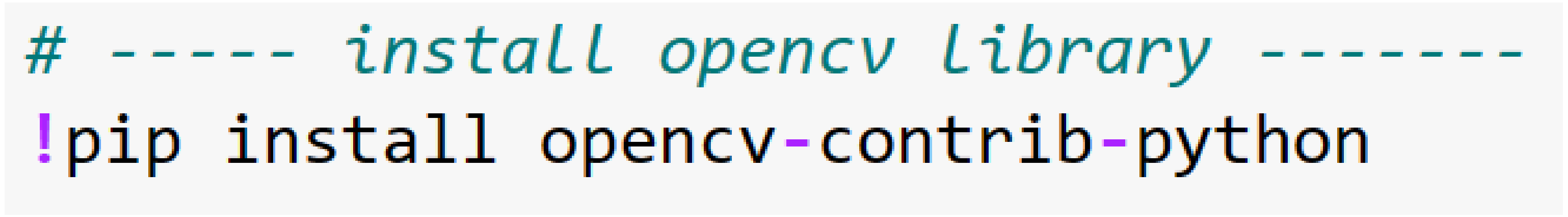
- Follow the same previous steps in installing the numpy library but choose **opencv-python library**.
- Another way to install opencv library, use the commend line

```
pip install opencv-python
pip install opencv-contrib-python
```

- To check if OpenCV is correctly installed, just run the following commands to perform a version check:

```
>>>import cv2
 >>>print(cv2.__version__)
```

- Install opencv library in jupyter notebook :



```
# ----- install opencv library -------
!pip install opencv-contrib-python
```

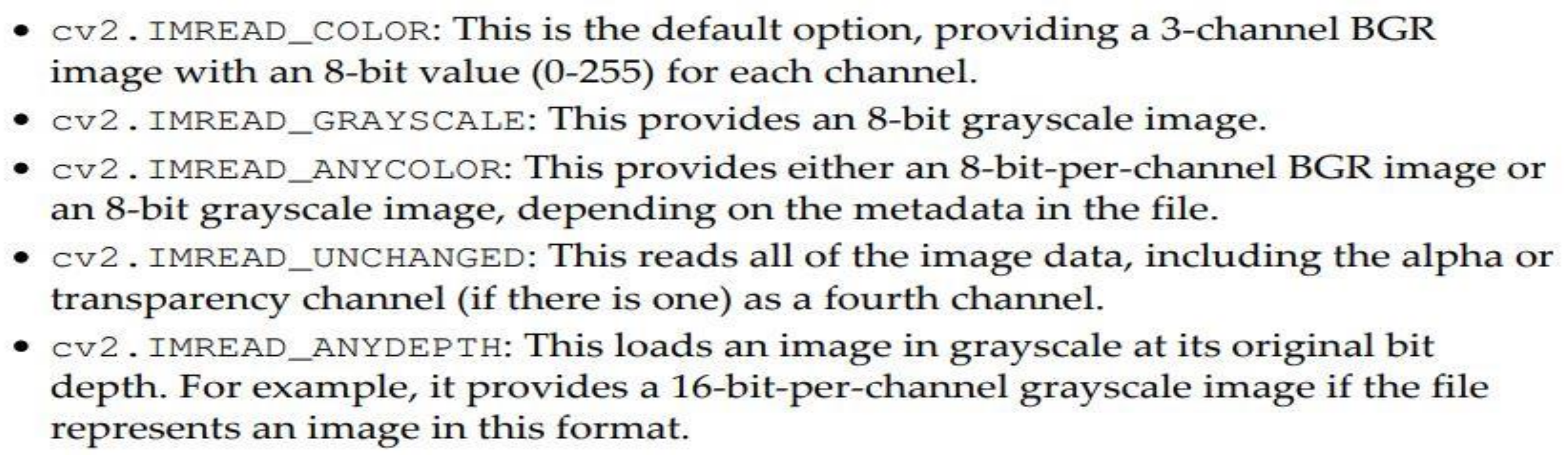
IMAGE READING

- The imread() function loads image from the specified file and returns it. The syntax is:

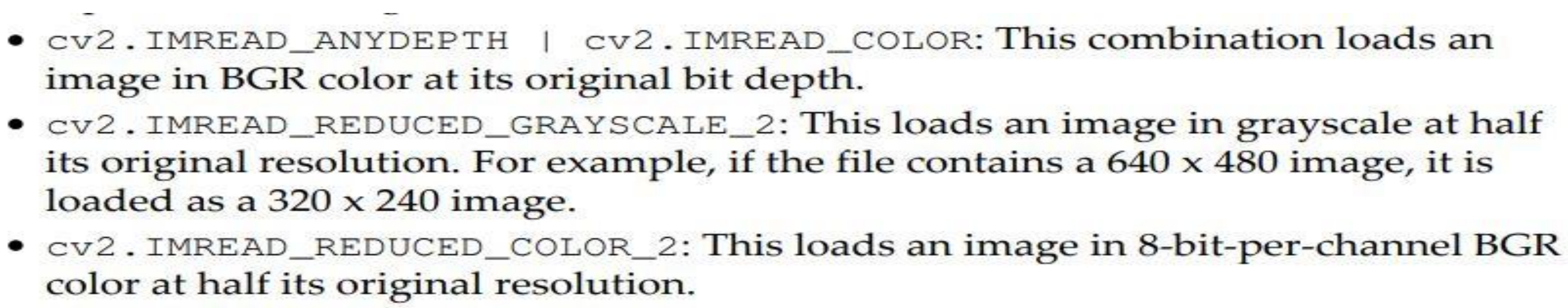
    cv2.imread(filename, mode of image read)

    filename: Name of the file to be loaded

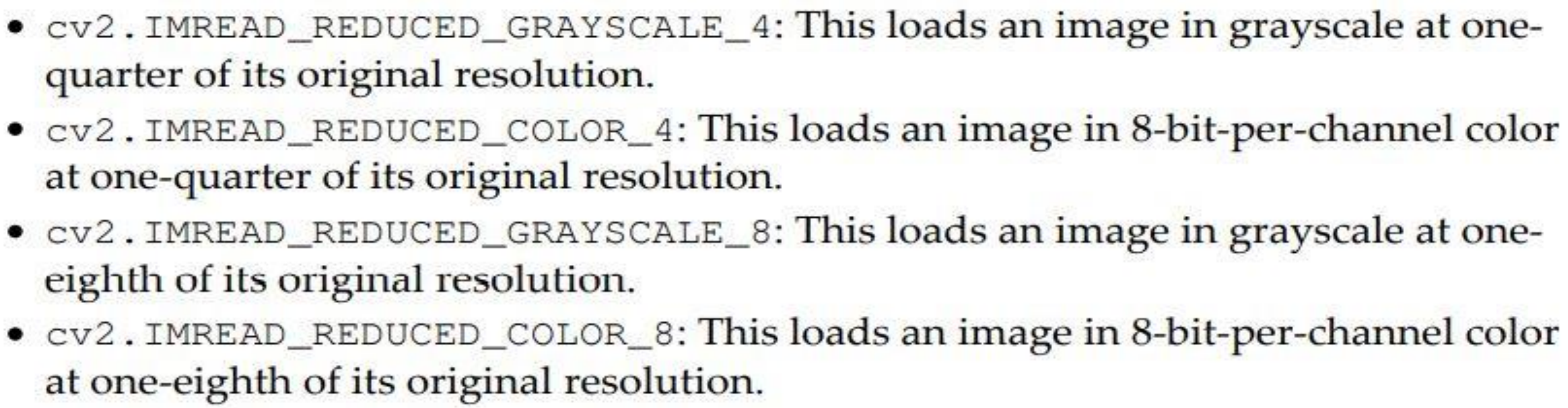
    Mode of image read:



- `cv2.IMREAD_COLOR`: This is the default option, providing a 3-channel BGR image with an 8-bit value (0-255) for each channel.
- `cv2.IMREAD_GRAYSCALE`: This provides an 8-bit grayscale image.
- `cv2.IMREAD_ANYCOLOR`: This provides either an 8-bit-per-channel BGR image or an 8-bit grayscale image, depending on the metadata in the file.
- `cv2.IMREAD_UNCHANGED`: This reads all of the image data, including the alpha or transparency channel (if there is one) as a fourth channel.
- `cv2.IMREAD_ANYDEPTH`: This loads an image in grayscale at its original bit depth. For example, it provides a 16-bit-per-channel grayscale image if the file represents an image in this format.



- `cv2.IMREAD_ANYDEPTH | cv2.IMREAD_COLOR`: This combination loads an image in BGR color at its original bit depth.
- `cv2.IMREAD_REDUCED_GRAYSCALE_2`: This loads an image in grayscale at half its original resolution. For example, if the file contains a 640 x 480 image, it is loaded as a 320 x 240 image.
- `cv2.IMREAD_REDUCED_COLOR_2`: This loads an image in 8-bit-per-channel BGR color at half its original resolution.

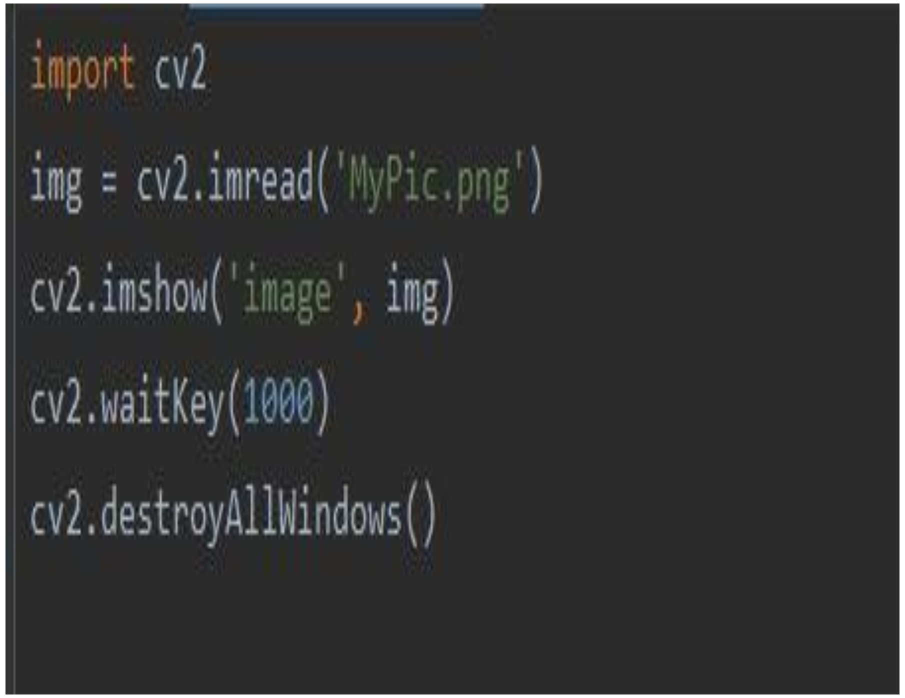


- `cv2.IMREAD_REDUCED_GRAYSCALE_4`: This loads an image in grayscale at one-quarter of its original resolution.
- `cv2.IMREAD_REDUCED_COLOR_4`: This loads an image in 8-bit-per-channel color at one-quarter of its original resolution.
- `cv2.IMREAD_REDUCED_GRAYSCALE_8`: This loads an image in grayscale at one-eighth of its original resolution.
- `cv2.IMREAD_REDUCED_COLOR_8`: This loads an image in 8-bit-per-channel color at one-eighth of its original resolution.

EXAMPLE FOR READING IMAGE:

- Code:

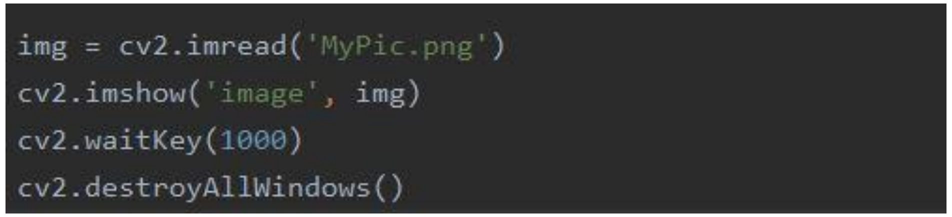
Output:



```
import cv2
img = cv2.imread('MyPic.png')
cv2.imshow('image', img)
cv2.waitKey(1000)
cv2.destroyAllWindows()
```

CODE EXAMPLES

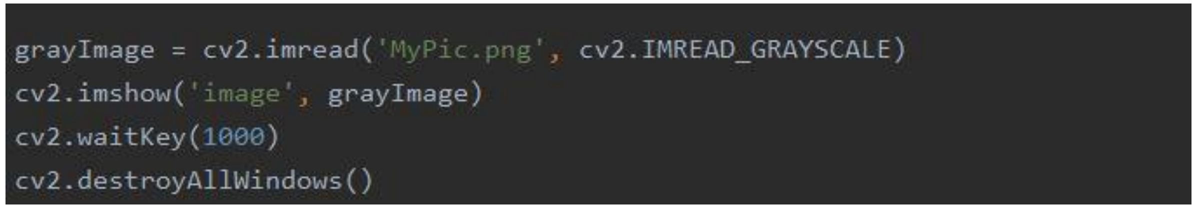
Example(1): Code



```
img = cv2.imread('MyPic.png')
cv2.imshow('image', img)
cv2.waitKey(1000)
cv2.destroyAllWindows()
```

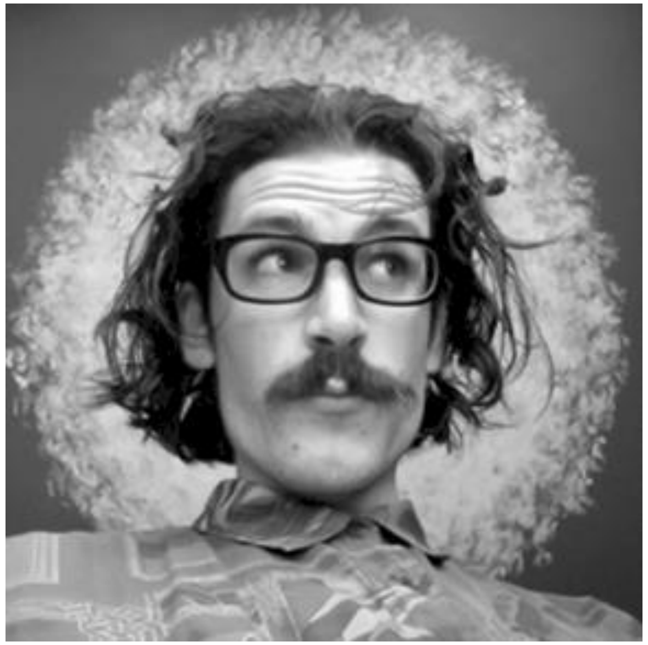
Output:

Example(2): Code



```
grayImage = cv2.imread('MyPic.png', cv2.IMREAD_GRAYSCALE)
cv2.imshow('image', grayImage)
cv2.waitKey(1000)
cv2.destroyAllWindows()
```

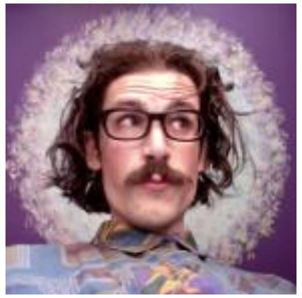
Output:



- Example(3): Code

```
image_reduced_color2 = cv2.imread('MyPic.png', cv2.IMREAD_REDUCED_COLOR_2)
cv2.imshow('image', image_reduced_color2)
cv2.waitKey(1000)
cv2.destroyAllWindows()
```

- Output:



Example(4): Code

```
image_reduced_grayscale8 = cv2.imread('MyPic.png', cv2.IMREAD_REDUCED_GRAYSCALE_8)
cv2.imshow('image', image_reduced_grayscale8)
cv2.waitKey(1000)
cv2.destroyAllWindows()
```

Output:

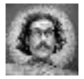
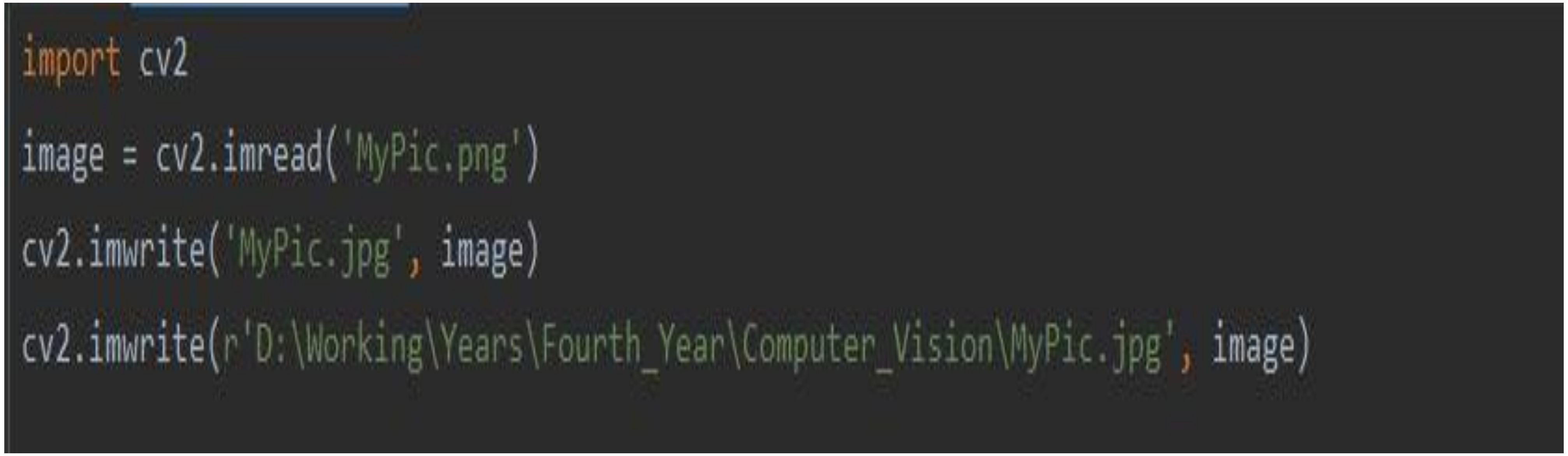


IMAGE WRITING

- Save image in project or in another location in your computer.



```
import cv2
image = cv2.imread('MyPic.png')
cv2.imwrite('MyPic.jpg', image)
cv2.imwrite(r'D:\Working\Years\Fourth_Year\Computer_Vision\MyPic.jpg', image)
```

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.

COLOR MODELS

1. Grayscale :
   - which pixel in a grayscale image is represented by a single 8-bit value, ranging from 0 for black to 255 for white.
2. BGR :
   - which each pixel has a triplet of values representing the blue, green, and red components.
   - each pixel is represented by a triplet of 8-bit values, such as [0, 0, 0] for black, [255, 0, 0] for blue, [0, 255, 0] for green, [0, 0, 255] for red, and [255, 255, 255] for white.
3. HSV :
   - uses a different triplet of channels.
   - Hue is the color's tone.
   - saturation is its intensity.
   - value represents its brightness.

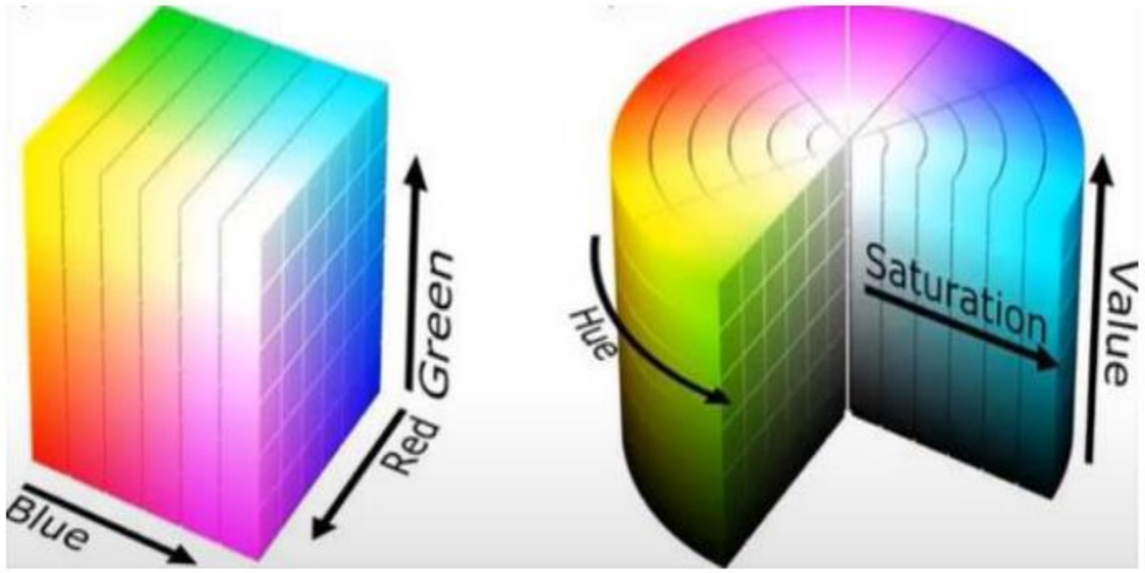
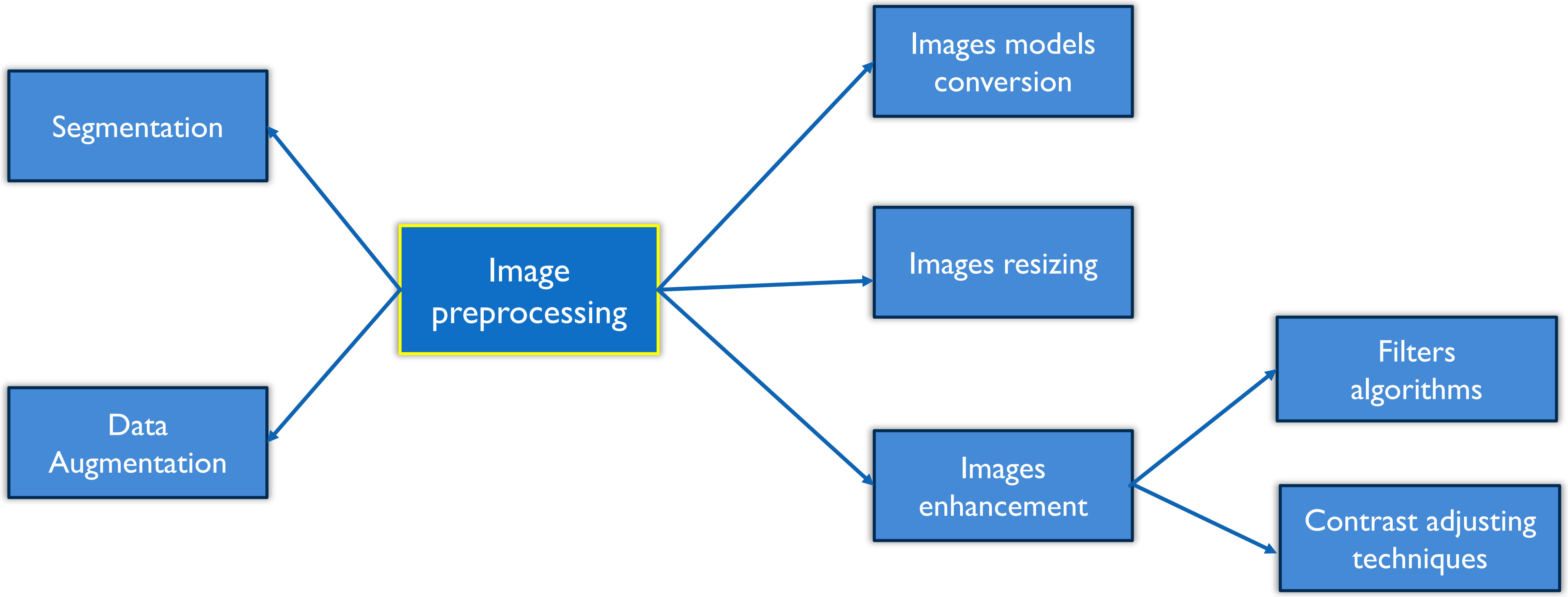


Image Processing

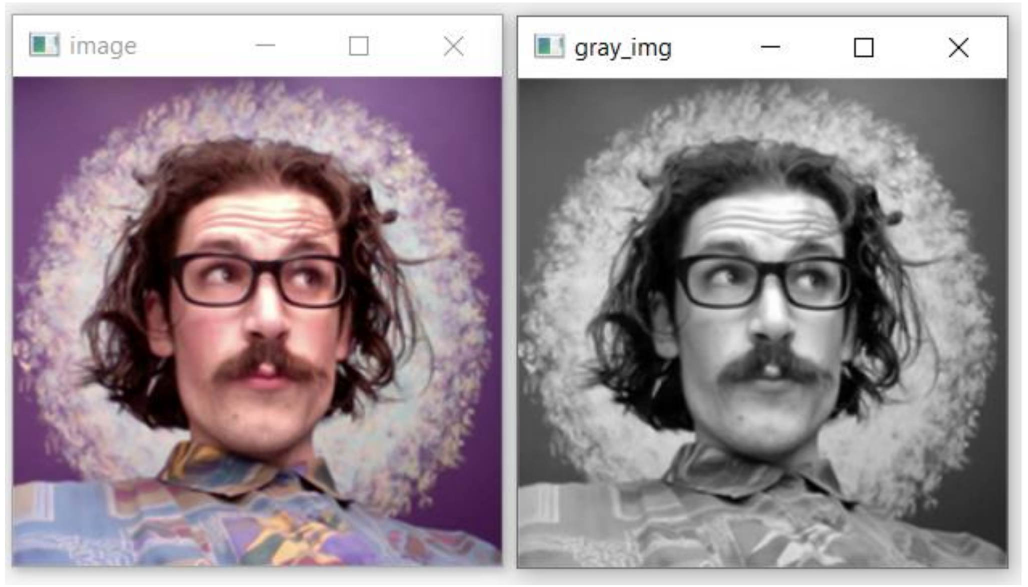


CONVERT FROM BGR TO GRAYSCALE.

Code

```
import cv2
img = cv2.imread("Picture1.png")
cv2.imshow("image", img)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("gray_img", gray_img)
cv2.waitKey()
cv2.destroyAllWindows()
```

Output

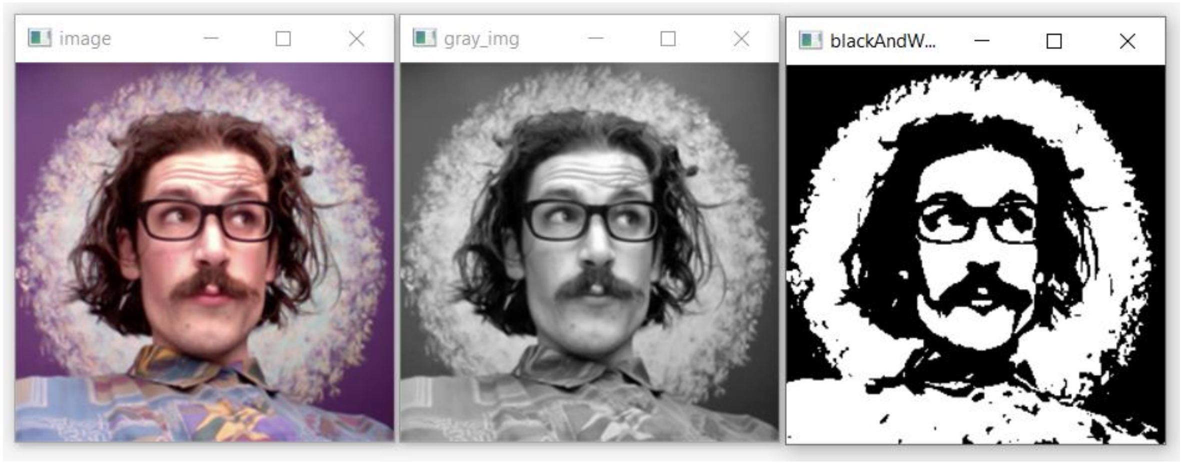


CONVERT FROM GRAY TO BINARY

Code

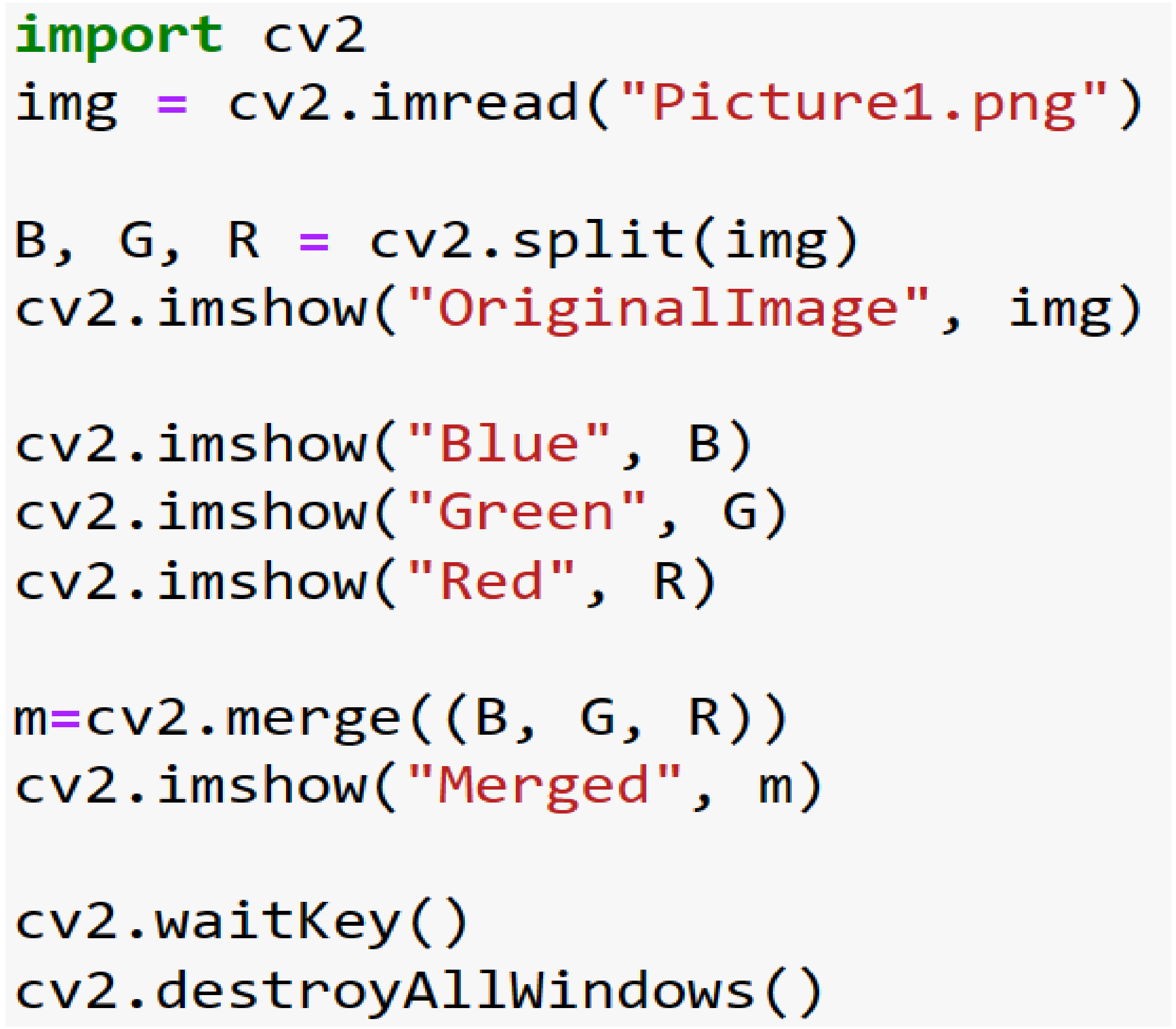
```
import cv2
img = cv2.imread("Picture1.png")
cv2.imshow("image", img)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("gray_img", gray_img)
(thresh, blackAndWhiteImage) = cv2.threshold(gray_img, 127, 255,
cv2.THRESH_BINARY)
cv2.imshow("blackAndWhiteImage", blackAndWhiteImage)
cv2.waitKey()
cv2.destroyAllWindows()
```

Output



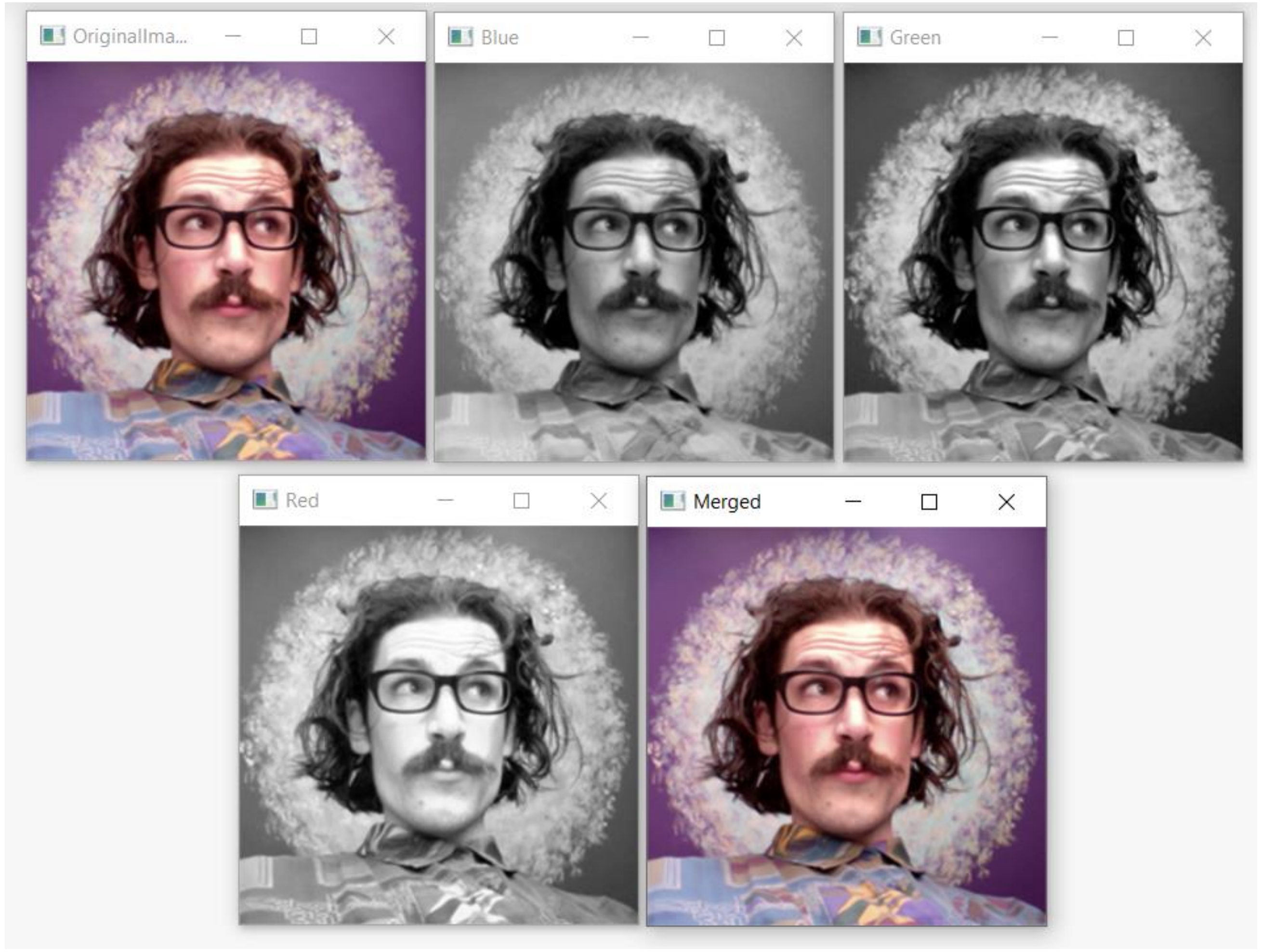
SPLIT $ MERGE

Code



```python
import cv2
img = cv2.imread("Picture1.png")

B, G, R = cv2.split(img)
cv2.imshow("OriginalImage", img)

cv2.imshow("Blue", B)
cv2.imshow("Green", G)
cv2.imshow("Red", R)

m=cv2.merge((B, G, R))
cv2.imshow("Merged", m)

cv2.waitKey()
cv2.destroyAllWindows()
```

Output

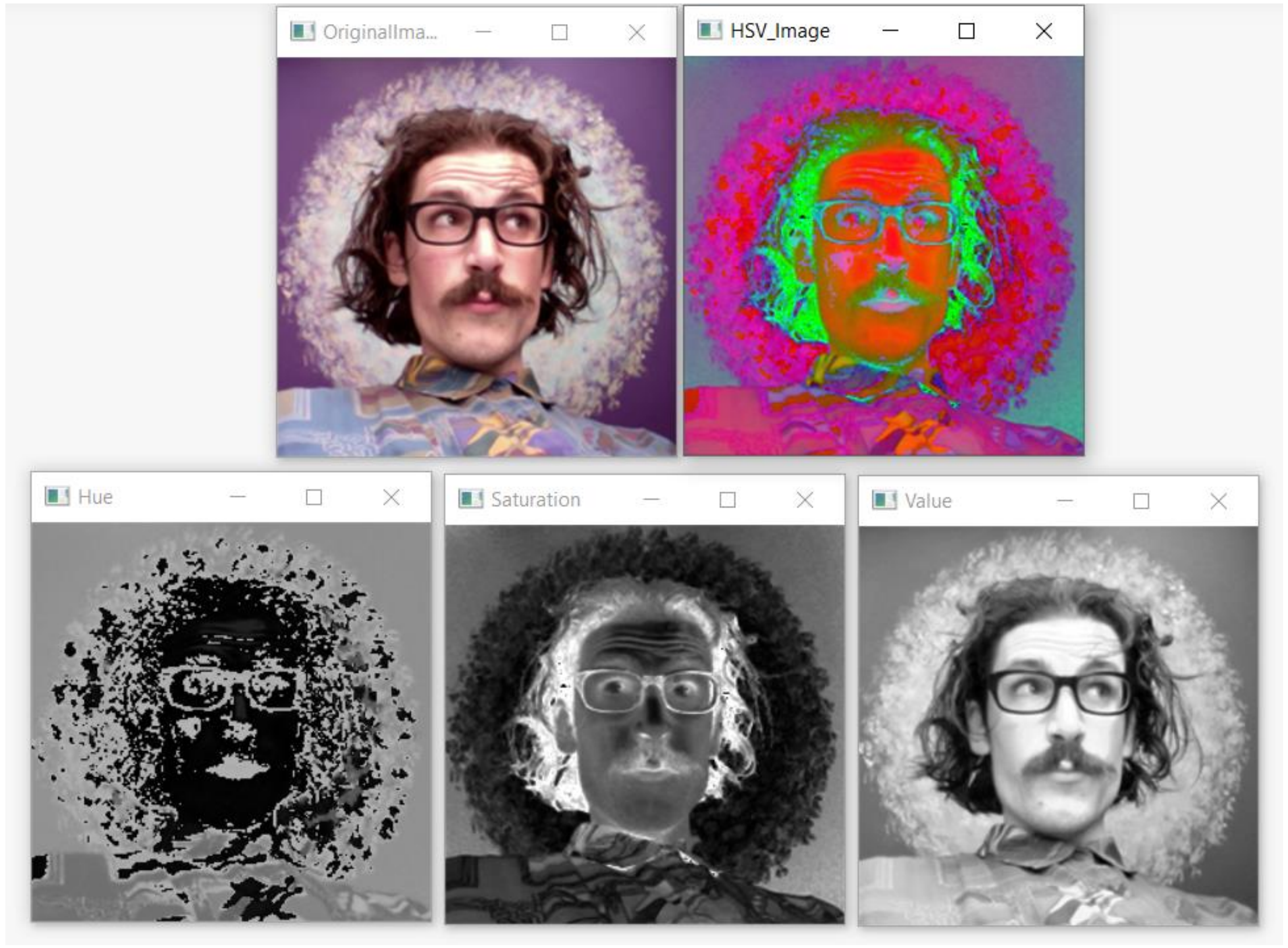


BGR TO HSV

Code

```
import cv2
img = cv2.imread("Picture1.png")
cv2.imshow("OriginalImage", img)

HSV_Image=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.imshow("HSV_Image", HSV_Image)

H=HSV_Image[:,:,0]
S=HSV_Image[:,:,1]
V=HSV_Image[:,:,2]
cv2.imshow("Hue", H)
cv2.imshow("Saturation", S)
cv2.imshow("Value", V)

cv2.waitKey()
cv2.destroyAllWindows()
```

Output



RESIZING IMAGE

Code

```
import cv2
img = cv2.imread("Picture1.png")
cv2.imshow("OriginalImage", img)

Half = cv2.resize(img, (0, 0), fx = 0.5, fy = 0.5)
Bigger = cv2.resize(img, (1050, 1610))
Stretch_near = cv2.resize(img, (780, 540),
interpolation = cv2.INTER_NEAREST)

cv2.imshow("Half", Half)
cv2.imshow("Big", Bigger)
cv2.imshow("Stretch", Stretch_near)

cv2.waitKey()
cv2.destroyAllWindows()
```

Output

Original Image

Half

Bigger

Stretch

IMAGE FILTERING

- To enhance the quality of image and remove the existing noise.
- There are two types of filters:
  - Low Pass Filter
  - High Pass Filter

- ✓ **Blurring image using Low Pass Filter (LPF).**

  - **Advantages of LPF:**
    - It helps in Noise removal. As noise is considered as high pass signal so by the application of low pass filter kernel, we restrict noise.
    - It helps in smoothing the image.
    - Low intensity edges are removed.
    - It helps in hiding the details when necessary.
  - **Types:**
    - Gaussian Blurring.
    - Median Blur.
    - Bilateral Blur.
    - Custom kernel.

IMAGE BLURRING (IMAGE SMOOTHING)

- Image blurring is achieved by convolving the image with a low-pass filter kernel.
- It is useful for removing noise. It actually removes high frequency content (eg: noise, edges) from the image.
- So edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges). OpenCV provides four main types of blurring techniques.
- https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html

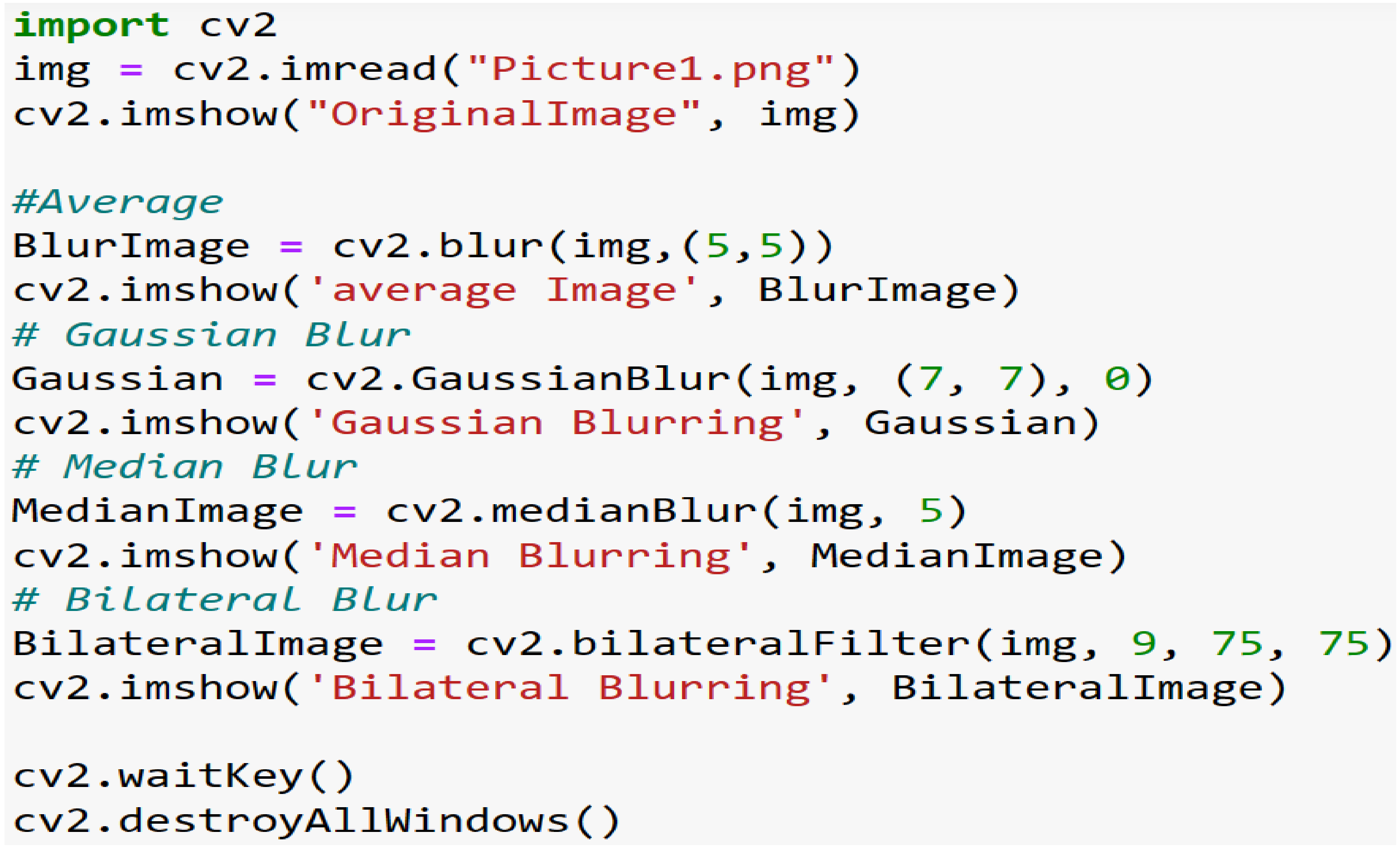
- **1. Averaging**
- This is done by convolving an image with a normalized box filter. It simply takes the average of all the pixels under the kernel area and replaces the central element. This is done by the function **cv.blur()** or **cv.boxFilter()**. Check the docs for more details about the kernel. We should specify the width and height of the kernel. A 3x3 normalized box filter would look like the below:

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- **2. Gaussian Blurring**
- In this method, instead of a box filter, a Gaussian kernel is used. It is done with the function, **cv.GaussianBlur()**. We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as the same as sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian blurring is highly effective in removing Gaussian noise from an image.
- If you want, you can create a Gaussian kernel with the function, **cv.getGaussianKernel()**.

- **3. Median Blurring**
- Here, the function cv.medianBlur() takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. But in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer.

- **4. Bilateral Filtering**
- **cv.bilateralFilter()** is highly effective in noise removal while keeping edges sharp. But the operation is slower compared to other filters. We already saw that a Gaussian filter takes the neighbourhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone, that is, nearby pixels are considered while filtering. It doesn't consider whether pixels have almost the same intensity. It doesn't consider whether a pixel is an edge pixel or not. So it blurs the edges also, which we don't want to do.
- Bilateral filtering also takes a Gaussian filter in space, but one more Gaussian filter which is a function of pixel difference. The Gaussian function of space makes sure that only nearby pixels are considered for blurring, while the Gaussian function of intensity difference makes sure that only those pixels with similar intensities to the central pixel are considered for blurring. So it preserves the edges since pixels at edges will have large intensity variation.



```
import cv2
img = cv2.imread("Picture1.png")
cv2.imshow("OriginalImage", img)

#Average
BlurImage = cv2.blur(img,(5,5))
cv2.imshow('average Image', BlurImage)
# Gaussian Blur
Gaussian = cv2.GaussianBlur(img, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)
# Median Blur
MedianImage = cv2.medianBlur(img, 5)
cv2.imshow('Median Blurring', MedianImage)
# Bilateral Blur
BilateralImage = cv2.bilateralFilter(img, 9, 75, 75)
cv2.imshow('Bilateral Blurring', BilateralImage)

cv2.waitKey()
cv2.destroyAllWindows()
```

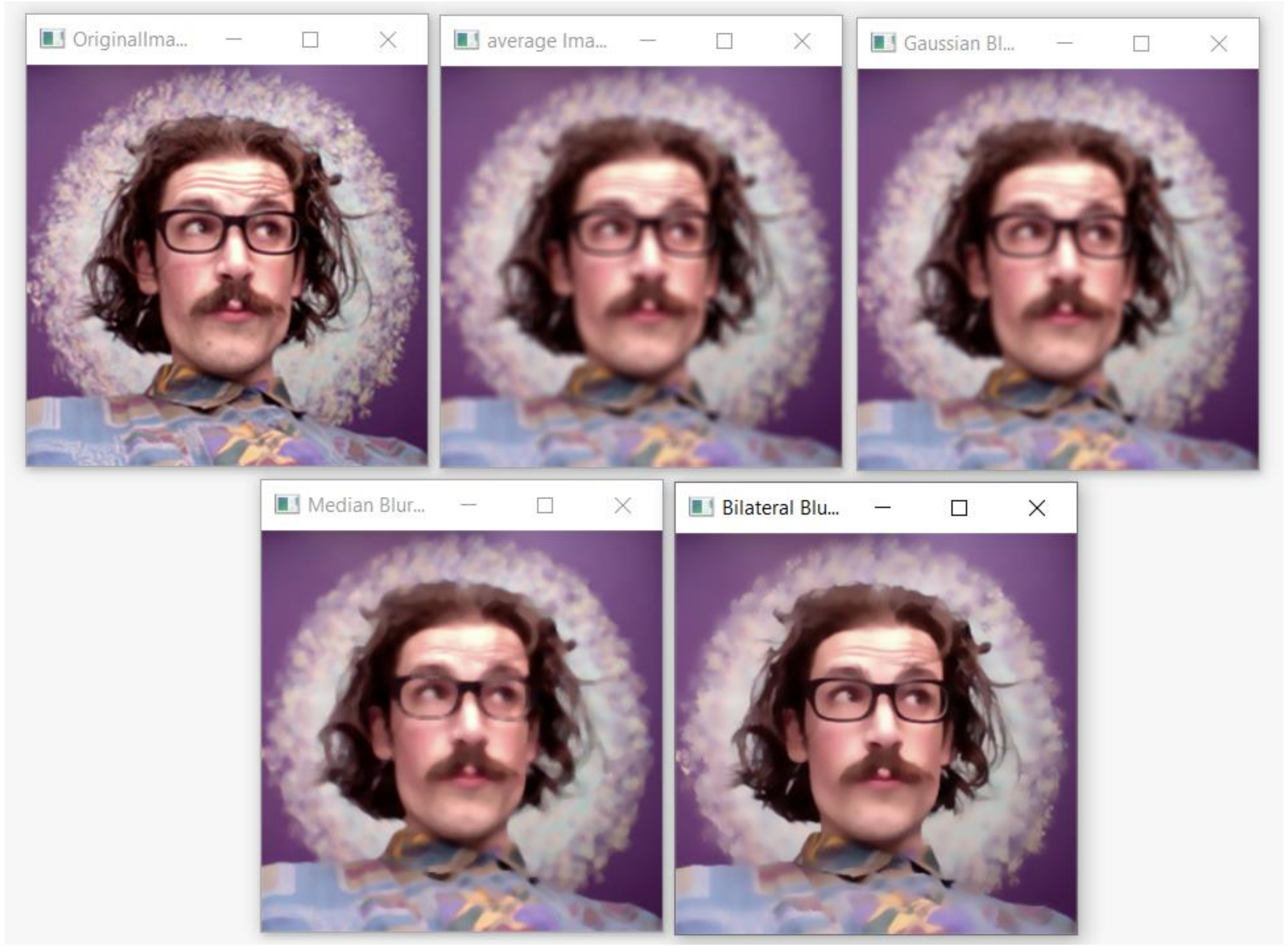

OriginalIma...
average Ima...
Gaussian Bl...
Median Blur...
Bilateral Blu...

IMAGE ENHANCEMENT

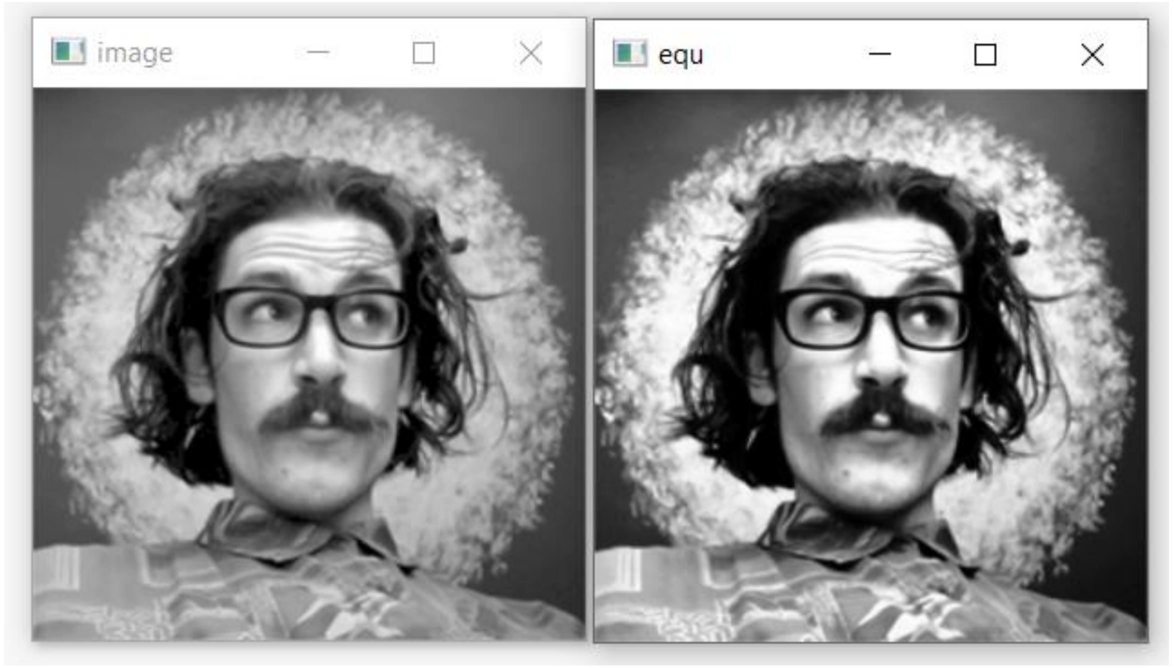
- Image enhancement is the process of improving the quality and appearance of an image. It can be used to correct flaws or defects in an image, or to simply make an image more visually appealing.
- Image enhancement techniques can be applied to a wide range of images, including photographs, scans, and digital images.
- Some common goals of image enhancement include increasing contrast, sharpness, and colorfulness; reducing noise and blur; and correcting distortion and other defects.
- Image enhancement techniques can be applied manually using image editing software, or automatically using algorithms and computer programs such as OpenCV.

HISTOGRAMS EQUALIZATION IN OPENCV

- https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html#histogram-equalization
- OpenCV has a function to do this, **cv2.equalizeHist()**. Its input is just grayscale image and output is our histogram equalized image.
- Histogram equalization is good when histogram of the image is confined to a particular region. It won’t work good in places where there is large intensity variations where histogram covers a large region, ie both bright and dark pixels are present. Please check the SOF links in Additional Resources.

```python
# ------ Histograms Equalization --------
import cv2
import numpy as np
img = cv2.imread("Picture1.png", 0)
cv2.imshow("image", img)
equ = cv2.equalizeHist(img)
cv2.imshow("equ", equ)
cv2.waitKey(3000)
cv2.destroyAllWindows()
```

Code



Output

Feature Extraction

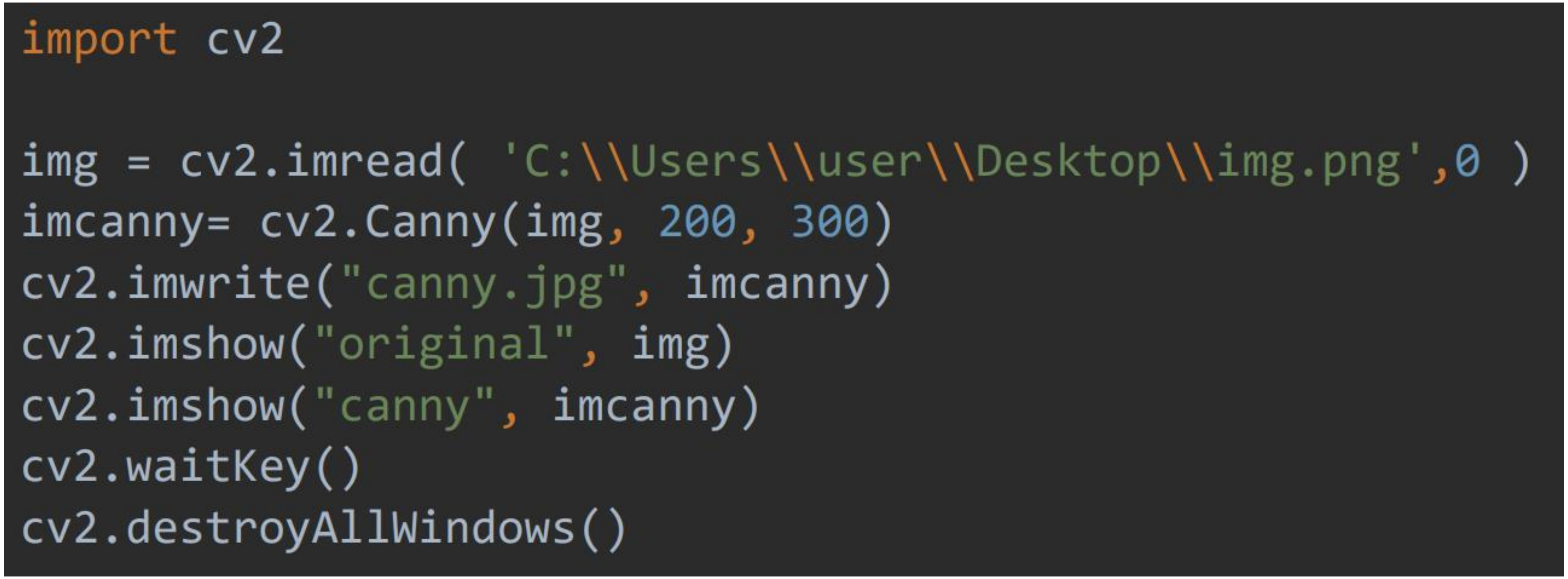
- Feature is a piece of information which is relevant for solving computational tasks.
- Types:
  - Edges
  - Corners.
  - Blobs.
  - Ridges.

Edge Detection

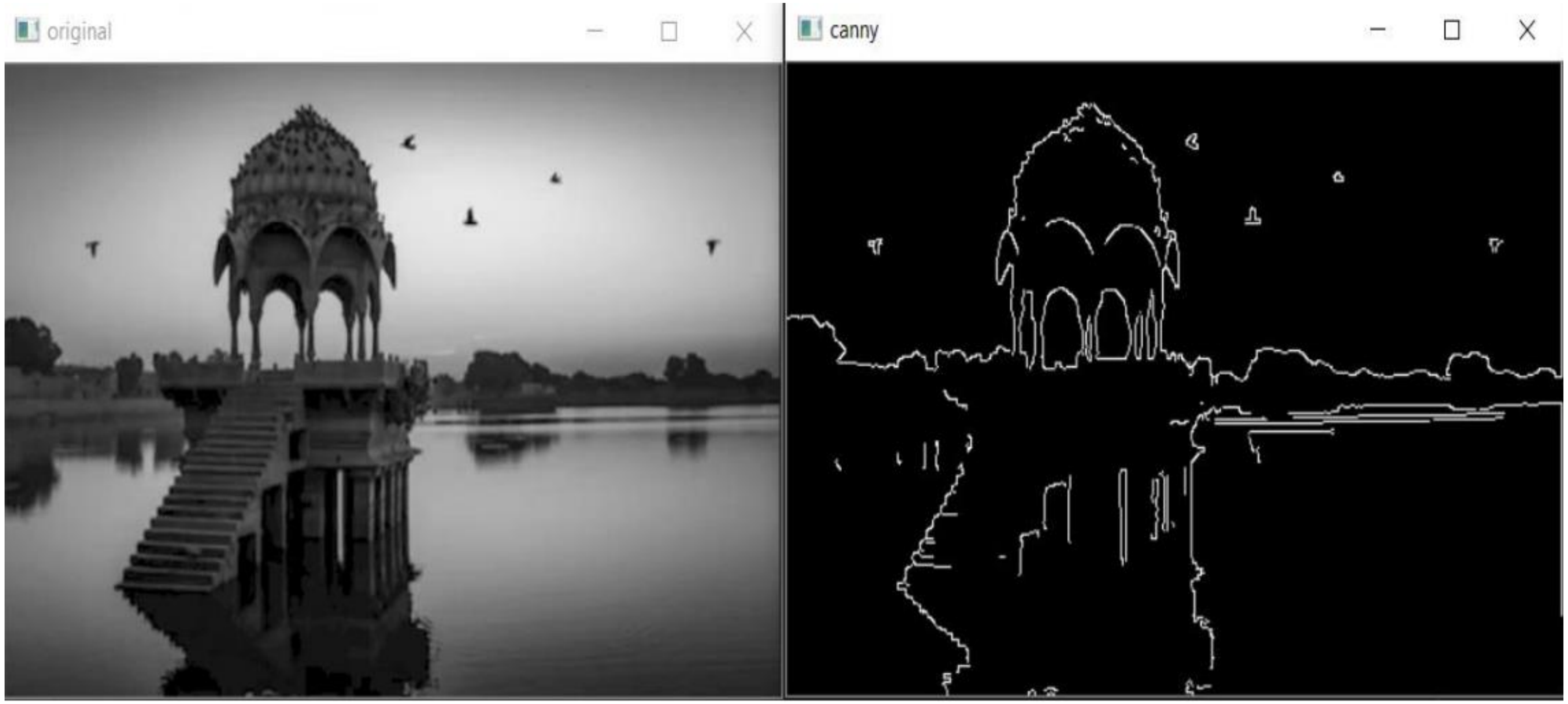
- Goal: Identify sudden changes(discontinuities) in an image.
- Most semantic and shape information from the image can be encoded in the edges.
- More compact than pixels.

- Edge Detection methods:
    - Custom kernel.
    - Laplacian Derivatives.
    - Sobel Derivatives.
    - Scharr Derivatives.
    - Canny.

- Canny detector:

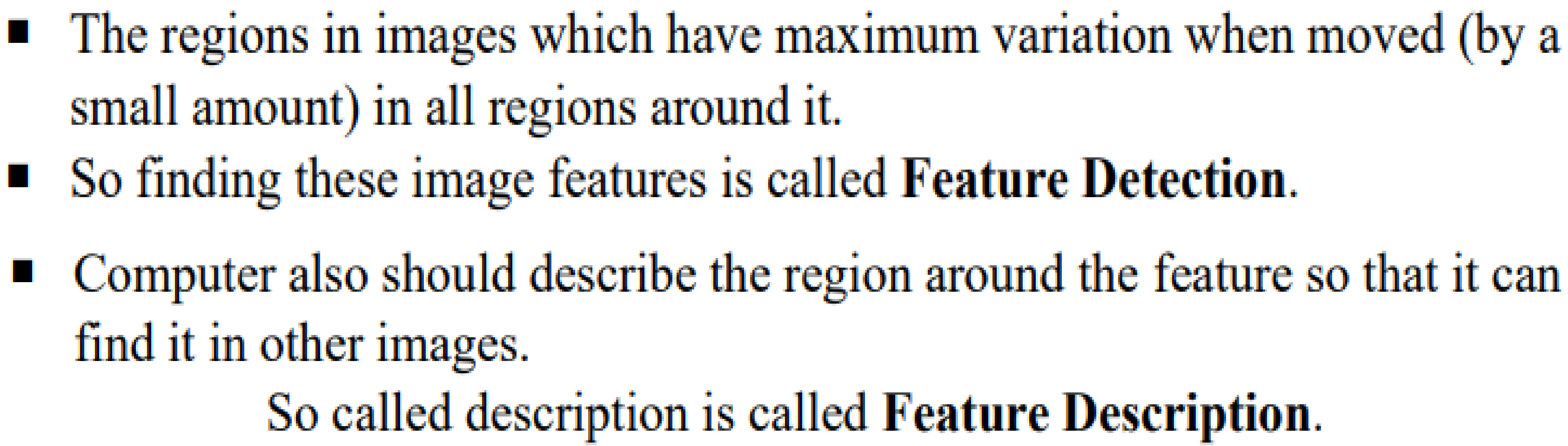


```
import cv2

img = cv2.imread( 'C:\\Users\\user\\Desktop\\img.png',0 )
imcanny= cv2.Canny(img, 200, 300)
cv2.imwrite("canny.jpg", imcanny)
cv2.imshow("original", img)
cv2.imshow("canny", imcanny)
cv2.waitKey()
cv2.destroyAllWindows()
```



Output

Corner Detection



- The regions in images which have maximum variation when moved (by a small amount) in all regions around it.
- So finding these image features is called **Feature Detection**.
- Computer also should describe the region around the feature so that it can find it in other images.

  So called description is called **Feature Description**.

- Harris Corner Detection:

A. Determine which windows produce very large variations in intensity when moved in x and y direction.

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \, [\underbrace{I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

where

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

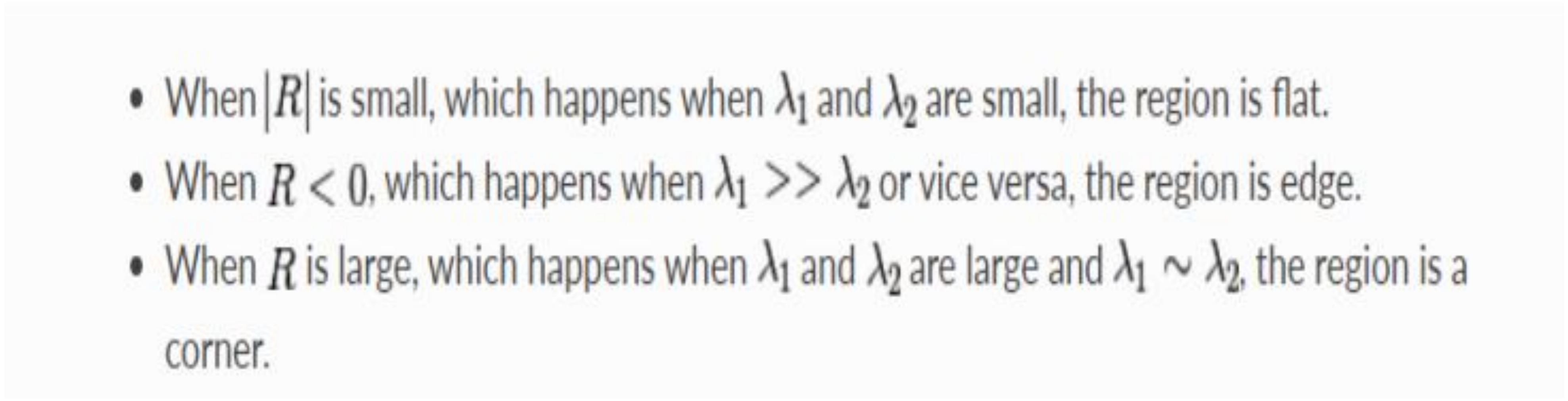
B. With each such window found, a score R is computed.

$$R = det(M) - k(trace(M))^2$$

where

- $det(M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigen values of M

C. After applying a threshold to this score, important corners are selected& marked.

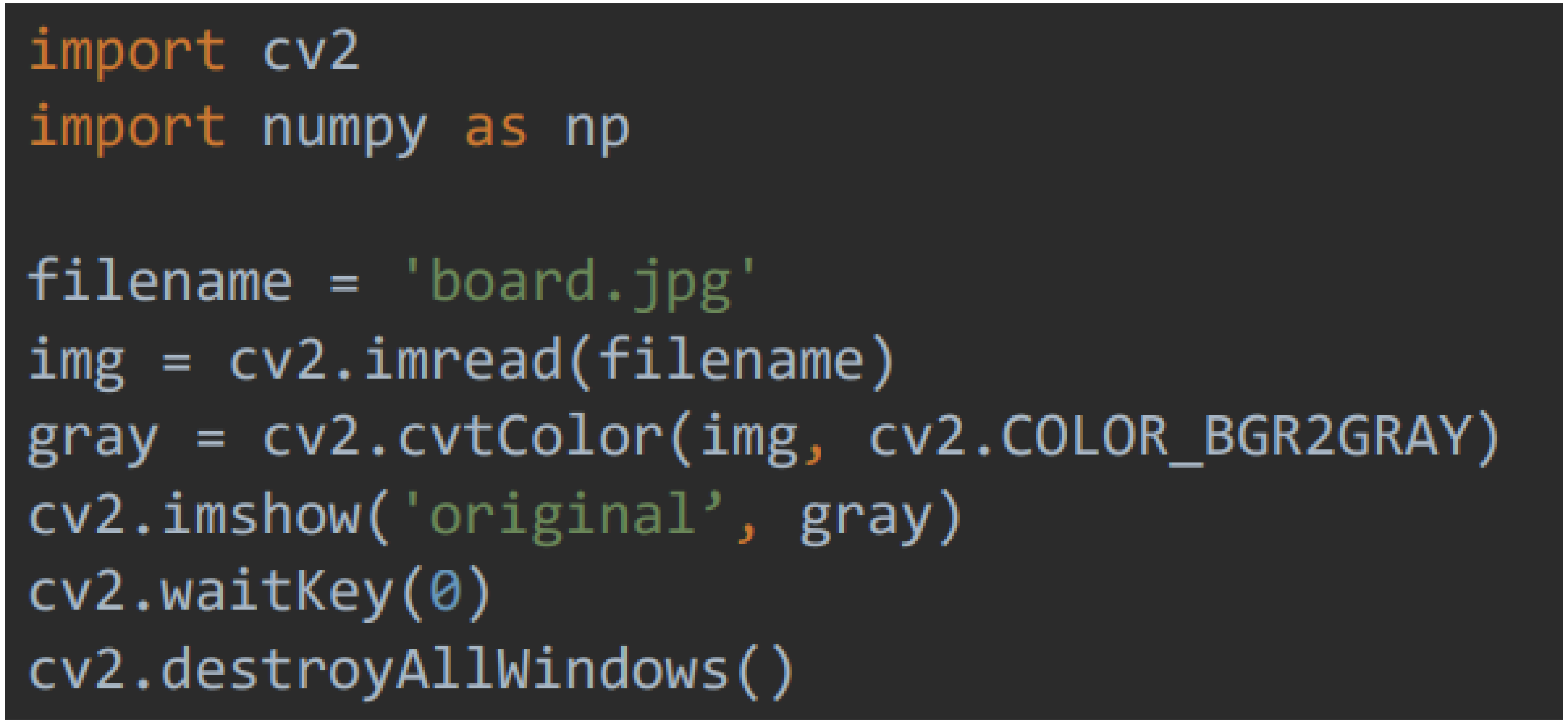


- When $|R|$ is small, which happens when λ_1 and λ_2 are small, the region is flat.
- When $R < 0$, which happens when $\lambda_1 >> \lambda_2$ or vice versa, the region is edge.
- When R is large, which happens when λ_1 and λ_2 are large and $\lambda_1 \sim \lambda_2$, the region is a corner.

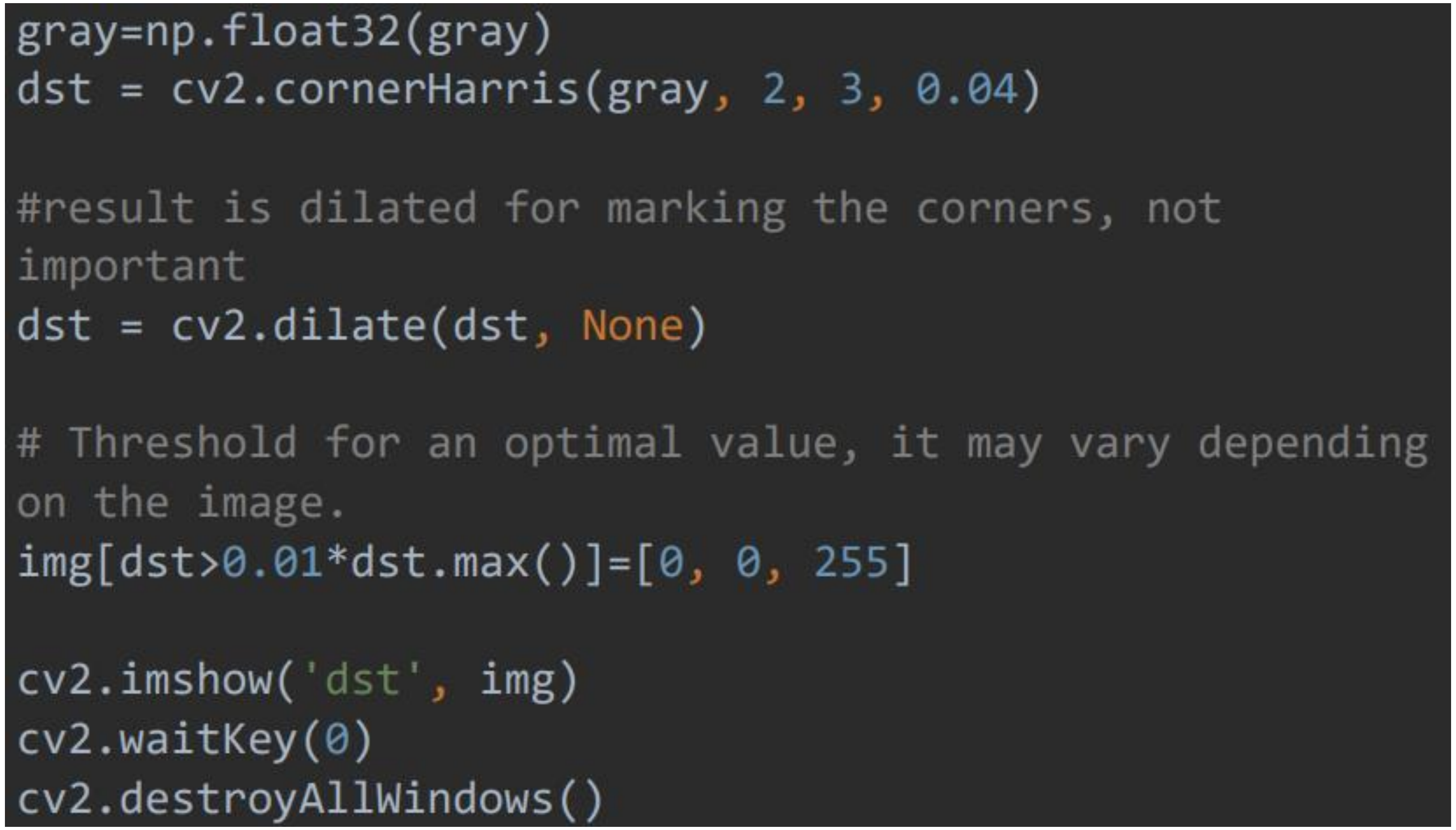
➢ Harris Corner Detection (OpenCV)

- OpenCV has the function cv2.cornerHarris() for this purpose. Its arguments are :

❑ **img** - Input image, it should be grayscale and float32 type.
❑ **blockSize** - It is the size of neighborhood considered for corner detection
❑ **ksize** - Aperture parameter of Sobel derivative used.
❑ **k** - Harris detector free parameter in the equation.

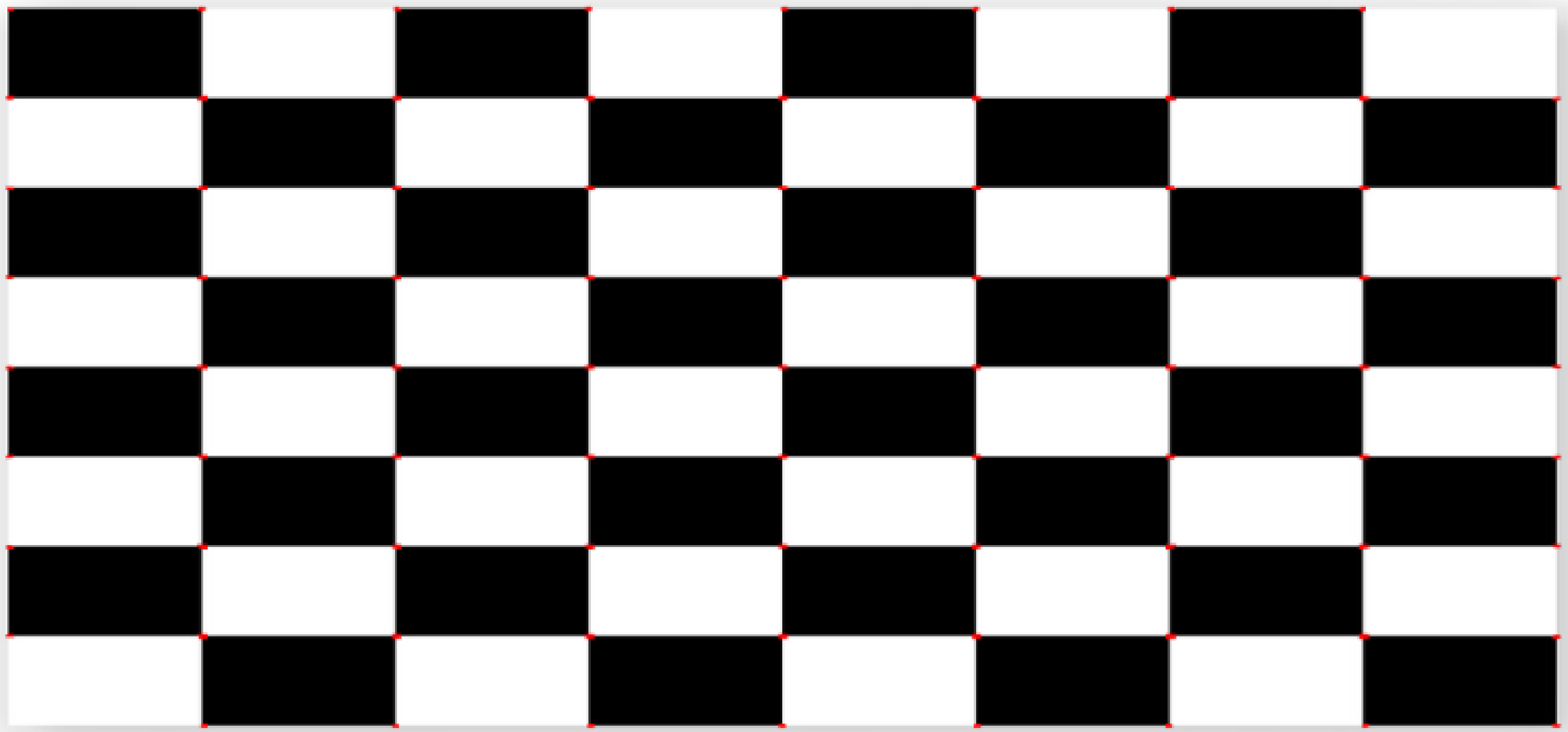


```
import cv2
import numpy as np

filename = 'board.jpg'
img = cv2.imread(filename)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('original', gray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

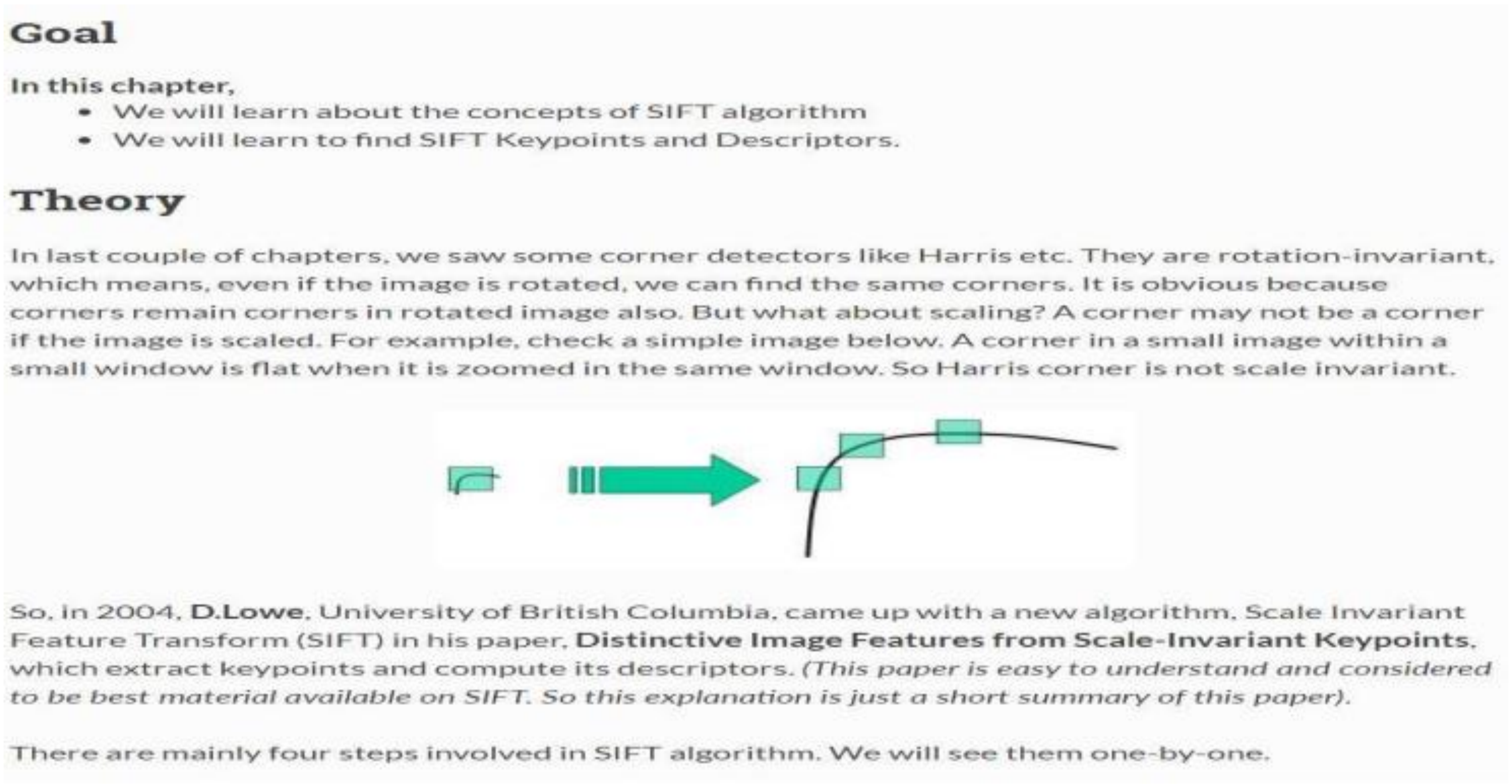


```
gray=np.float32(gray)
dst = cv2.cornerHarris(gray, 2, 3, 0.04)

#result is dilated for marking the corners, not
important
dst = cv2.dilate(dst, None)

# Threshold for an optimal value, it may vary depending
on the image.
img[dst>0.01*dst.max()]=[0, 0, 255]

cv2.imshow('dst', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The output:



SIFT (SCALE-INVARIANT FEATURE TRANSFORM)



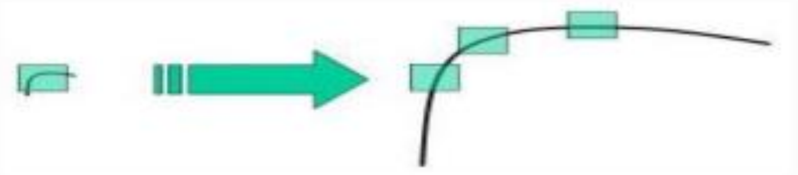
Goal

In this chapter,

- We will learn about the concepts of SIFT algorithm
- We will learn to find SIFT Keypoints and Descriptors.

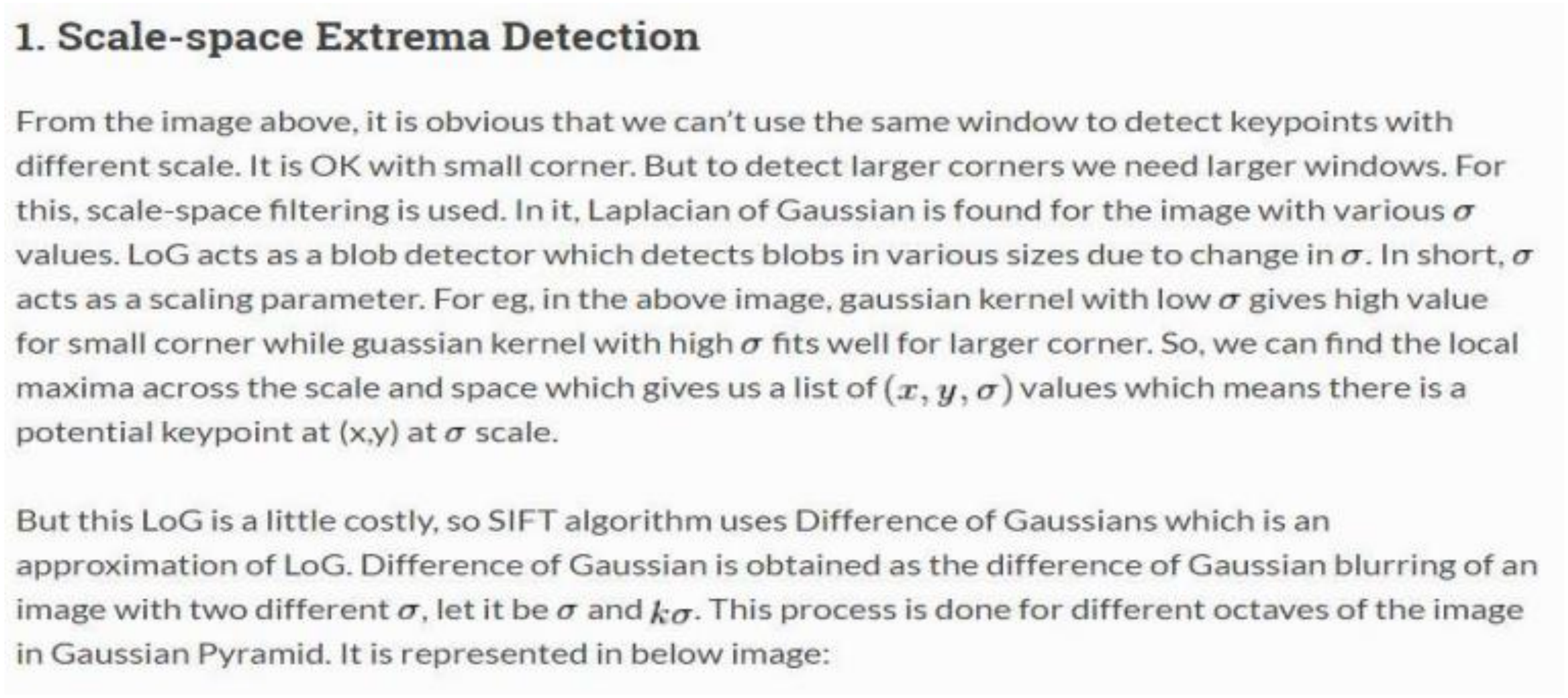
Theory

In last couple of chapters, we saw some corner detectors like Harris etc. They are rotation-invariant, which means, even if the image is rotated, we can find the same corners. It is obvious because corners remain corners in rotated image also. But what about scaling? A corner may not be a corner if the image is scaled. For example, check a simple image below. A corner in a small image within a small window is flat when it is zoomed in the same window. So Harris corner is not scale invariant.



So, in 2004, **D.Lowe**, University of British Columbia, came up with a new algorithm, Scale Invariant Feature Transform (SIFT) in his paper, **Distinctive Image Features from Scale-Invariant Keypoints**, which extract keypoints and compute its descriptors. *(This paper is easy to understand and considered to be best material available on SIFT. So this explanation is just a short summary of this paper).*

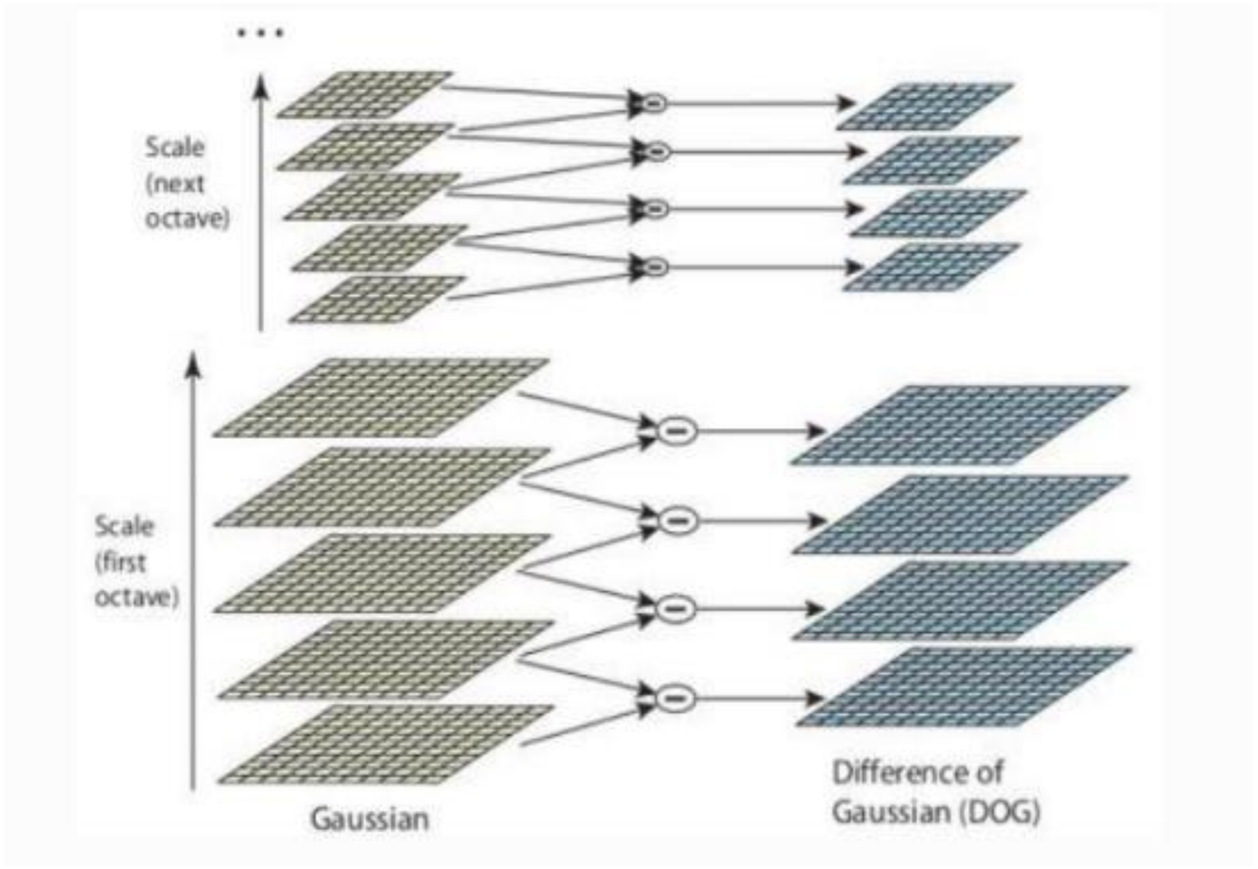
There are mainly four steps involved in SIFT algorithm. We will see them one-by-one.



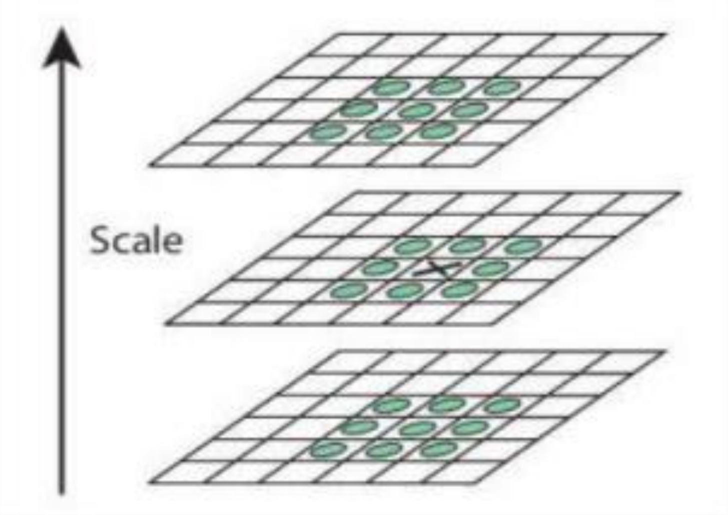
1. Scale-space Extrema Detection

From the image above, it is obvious that we can't use the same window to detect keypoints with different scale. It is OK with small corner. But to detect larger corners we need larger windows. For this, scale-space filtering is used. In it, Laplacian of Gaussian is found for the image with various σ values. LoG acts as a blob detector which detects blobs in various sizes due to change in σ. In short, σ acts as a scaling parameter. For eg, in the above image, gaussian kernel with low σ gives high value for small corner while guassian kernel with high σ fits well for larger corner. So, we can find the local maxima across the scale and space which gives us a list of (x, y, σ) values which means there is a potential keypoint at (x,y) at σ scale.

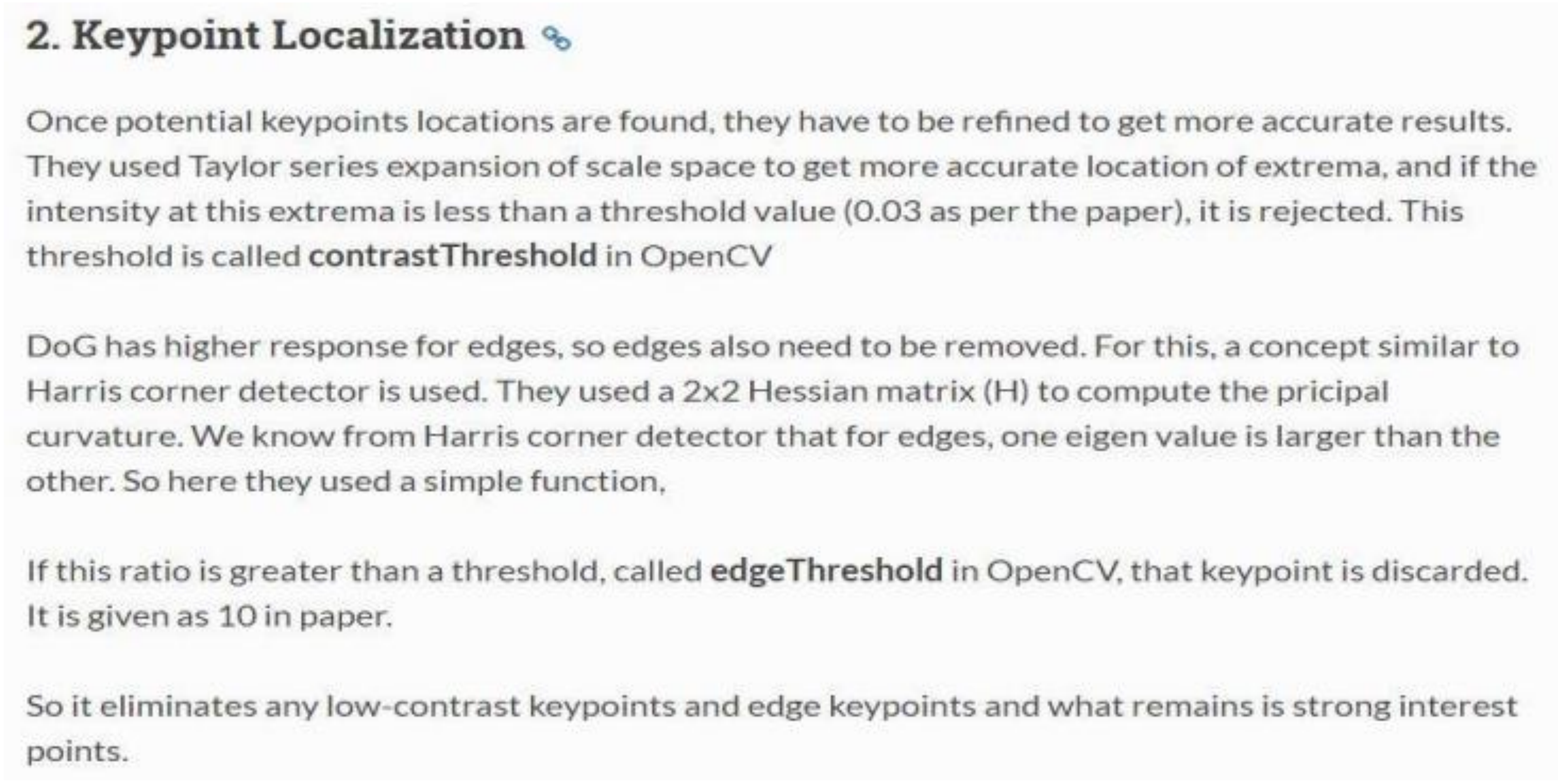
But this LoG is a little costly, so SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and $k\sigma$. This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image:


. . .
Scale (next octave)
Scale (first octave)
Gaussian
Difference of Gaussian (DOG)

Once this DoG are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale. It is shown in below image:



Regarding different parameters, the paper gives some empirical data which can be summarized as, number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6, k = \sqrt{2}$ etc as optimal values.



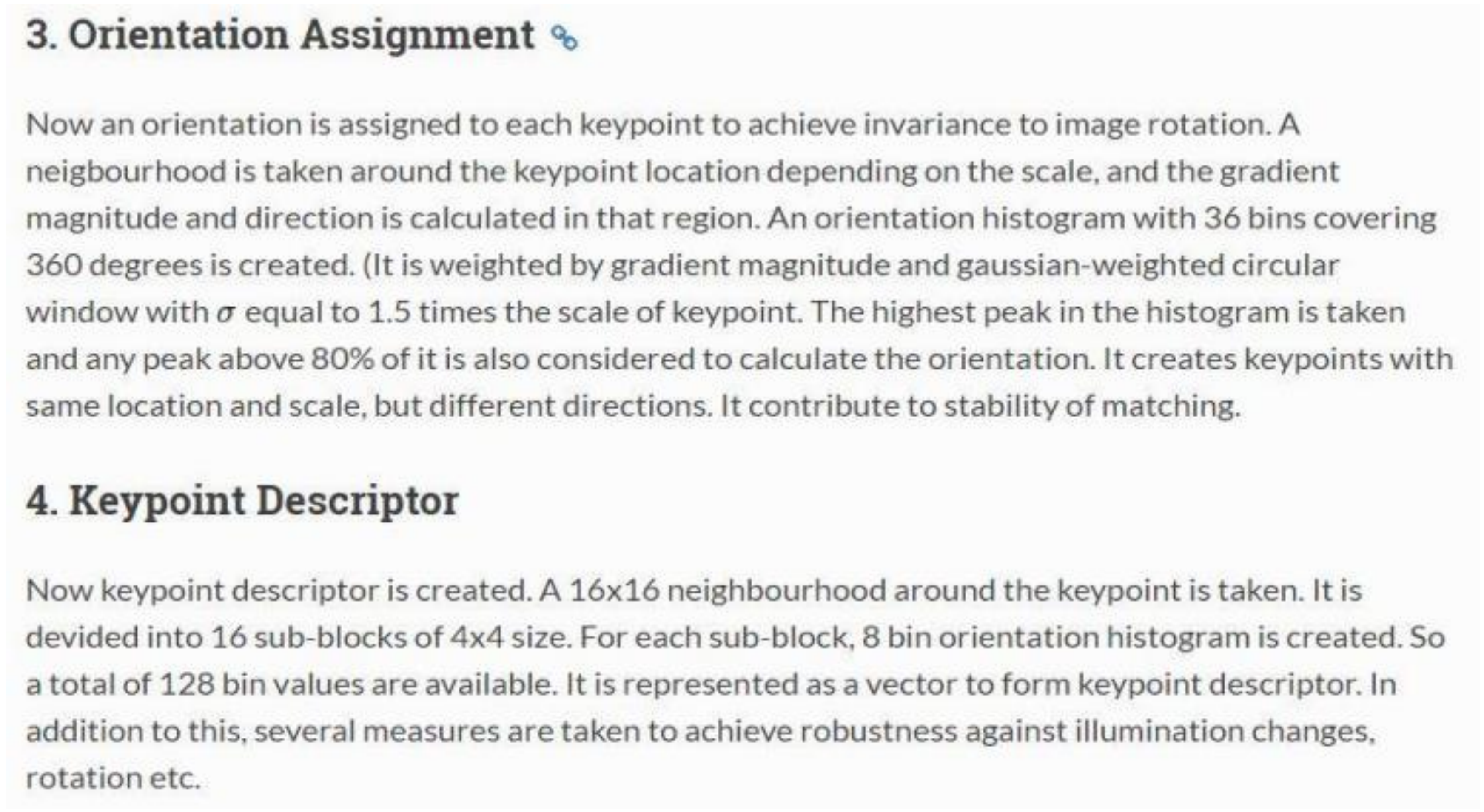
2. Keypoint Localization

Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called **contrastThreshold** in OpenCV

DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a 2x2 Hessian matrix (H) to compute the pricipal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function,

If this ratio is greater than a threshold, called **edgeThreshold** in OpenCV, that keypoint is discarded. It is given as 10 in paper.

So it eliminates any low-contrast keypoints and edge keypoints and what remains is strong interest points.

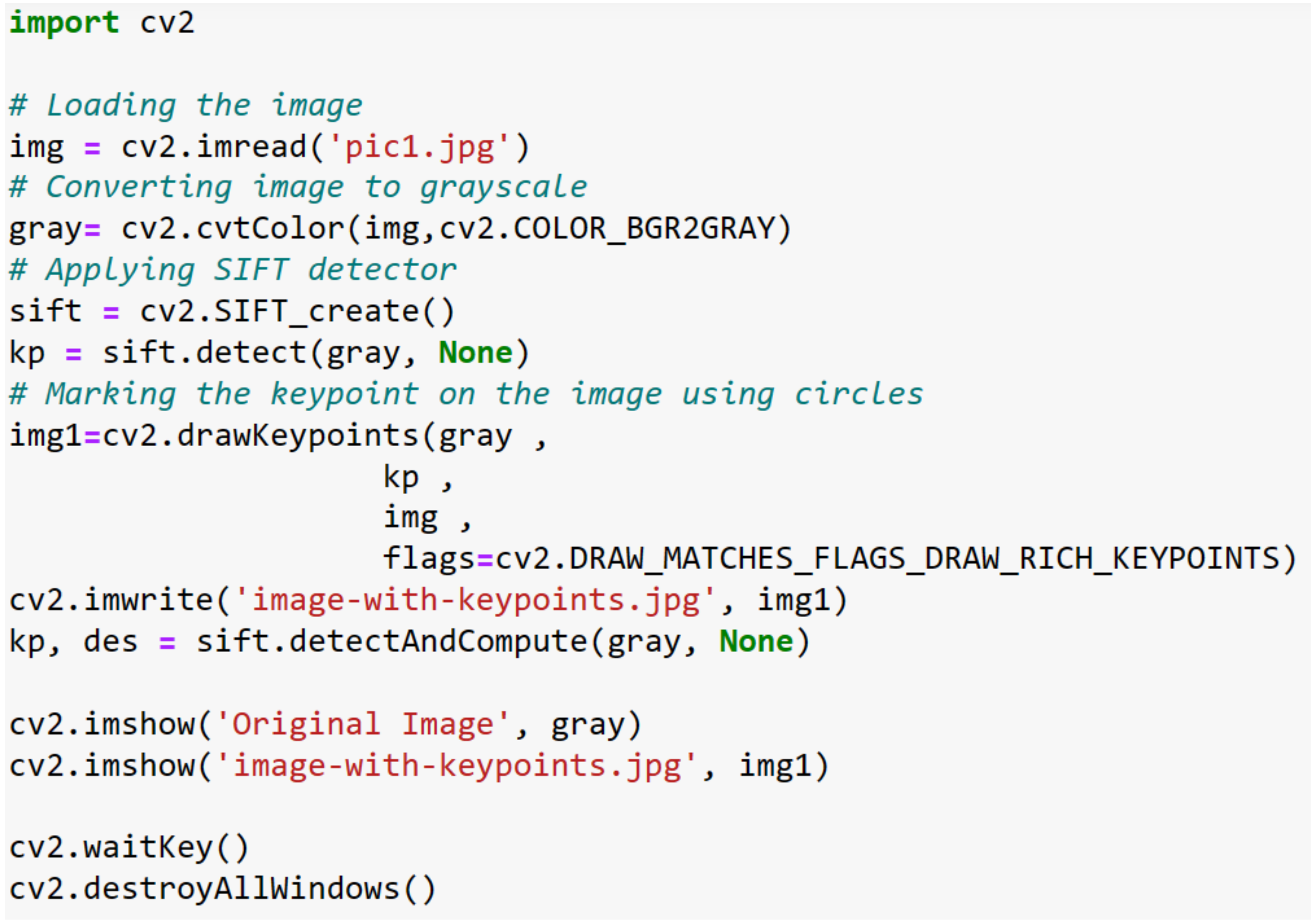


3. Orientation Assignment

Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neigbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. 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 contribute to stability of matching.

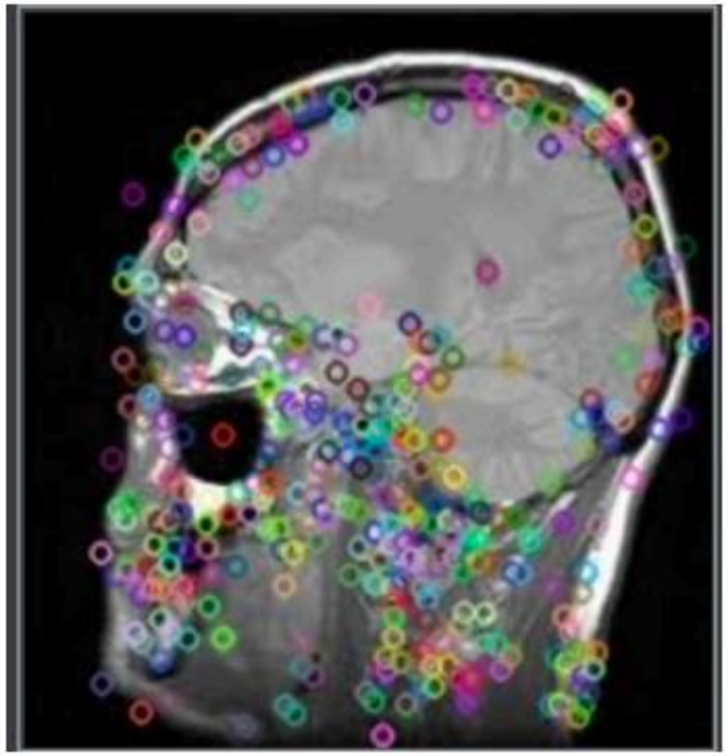
4. Keypoint Descriptor

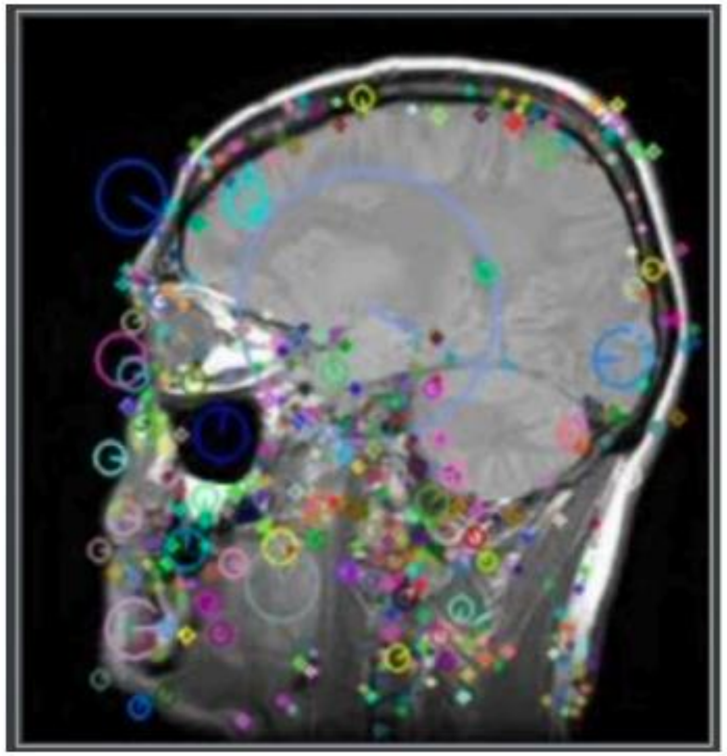
Now keypoint descriptor is created. A 16x16 neighbourhood around the keypoint is taken. It is devided into 16 sub-blocks of 4x4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.



```python
import cv2

# Loading the image
img = cv2.imread('pic1.jpg')
# Converting image to grayscale
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
# Applying SIFT detector
sift = cv2.SIFT_create()
kp = sift.detect(gray, None)
# Marking the keypoint on the image using circles
img1=cv2.drawKeypoints(gray ,
                    kp ,
                    img ,
                    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imwrite('image-with-keypoints.jpg', img1)
kp, des = sift.detectAndCompute(gray, None)

cv2.imshow('Original Image', gray)
cv2.imshow('image-with-keypoints.jpg', img1)

cv2.waitKey()
cv2.destroyAllWindows()
```

The output:

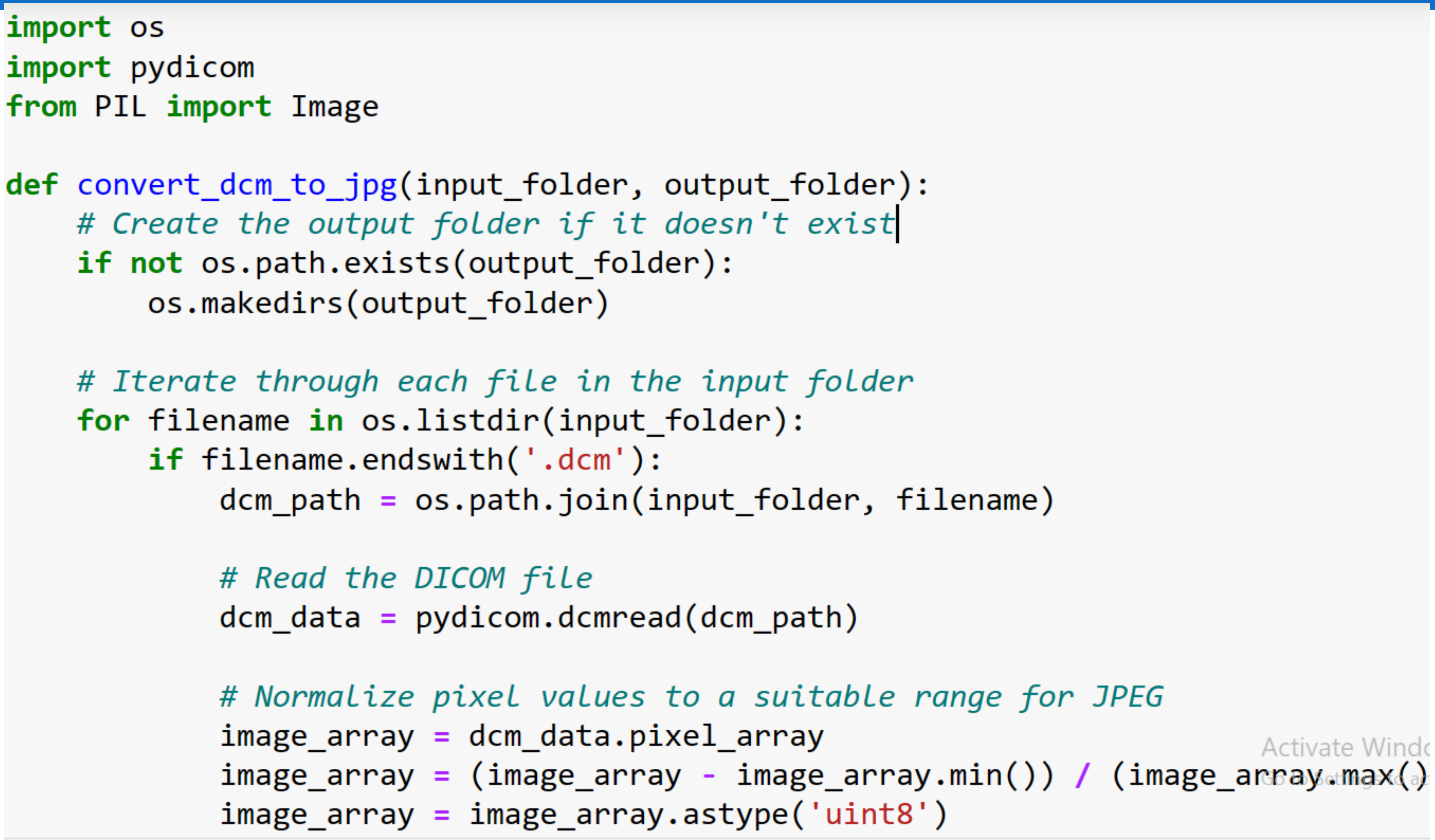




Read Jpg Images From Folder

```python
import os
import cv2

rootdir = r"C:\Users\lenovo\DL Course\Brain Tumor Images"

for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        if file.endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp')):
            # Check if the file is an image
            image_path = os.path.join(subdir, file)
            print("Image Name:", file)
            frame = cv2.imread(image_path)
            cv2.imshow(file, frame)
            cv2.waitKey(1000)
            cv2.destroyAllWindows()
```

Read DICOM Images From Folder And Convert Into Jpg Images



```python
import os
import pydicom
from PIL import Image

def convert_dcm_to_jpg(input_folder, output_folder):
    # Create the output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Iterate through each file in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith('.dcm'):
            dcm_path = os.path.join(input_folder, filename)

            # Read the DICOM file
            dcm_data = pydicom.dcmread(dcm_path)

            # Normalize pixel values to a suitable range for JPEG
            image_array = dcm_data.pixel_array
            image_array = (image_array - image_array.min()) / (image_array.max()
            image_array = image_array.astype('uint8')
            # Convert array to PIL Image
            image = Image.fromarray(image_array)

            # Save as JPEG in the output folder
            output_path = os.path.join(output_folder,
                                       f"{os.path.splitext(filename)[0]}.jpg")
            image.save(output_path, "JPEG")

            print(f"Converted {filename} to JPEG")

# Replace 'input_folder' with the path to your DICOM images folder
input_folder = r"C:\Users\lenovo\DL Course\dicom_dir"
# Replace 'output_folder' with the path where you want to save the JPEG images
output_folder = r"C:\Users\lenovo\DL Course\jpg_dicom_dir"

convert_dcm_to_jpg(input_folder, output_folder)
```

Thank you