Introduction to Robotics

Manipulation and Programming

Unit 3: Sensors and Vision

CAMERA AND COLOR – PART 3: LOW LEVEL VISION

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- The interest in **computer vision** started with a quest to make machines comprehend images to enhance the possibilities of image processing.
- In this lab, we cover basic image processing techniques, with ready-to-use code, and examples to understand concepts better. Various low-level vision algorithms dealing with image features like edges, contours, etc are considered.
- **Python** and OpenCV will be our go-to resources. You may get stuck during installation, coding, compiling, or error-handling. You can refer to StackOverflow, Python docs, etc, to clarify to get solutions to the problems at hand. We find instructions for installation of OpenCV and its dependencies on the website mentioned in the references.

Load Image

SECTION 1

Load Image

- Let us understand the basic building block of an image – the pixels. The pixels are the most fundamental blocks in an image. A high definition (HD) image has higher volume of these pixels and a lower resolution image would have fewer pixels.
- The world is wrapped in a continuous field of reality, and the thought of pixelating the world is mind-boggling – which can you an understanding of how difficult it would be to apply concepts of computer vision for real world applications

Load Image

- Consider the two images: grayscale and color. In the former, each pixel ranges between 0 and 255, where 0 is black and 255 is white. In the latter, three-color channels together present the various colors present in the vast spectrum.

Load Image

- The first line imports the library OpenCV, and we are accessing a file named “messi.jpg” present in the same directory as the code. Download an image and name it as **messi.jpg**, and store it in the same directory as the one with the code in it. Only then will the above code work. Finally, we print the image onto the output screen with the dialog box named “Image”.
- Images are arrays of integers ranging between two numbers, depending on the number of bits used to represent each pixel. For example, in a 1-bit image, the number of color choices for each pixel is white(1) or black(0).

Demo Program:

loadimage.py

```
import cv2
image =cv2.imread("messi.jpg")
cv2.imshow("Image", image)

while True:
    if cv2.waitKey(100) == 27: break
cv2.destroyAllWindows()
```

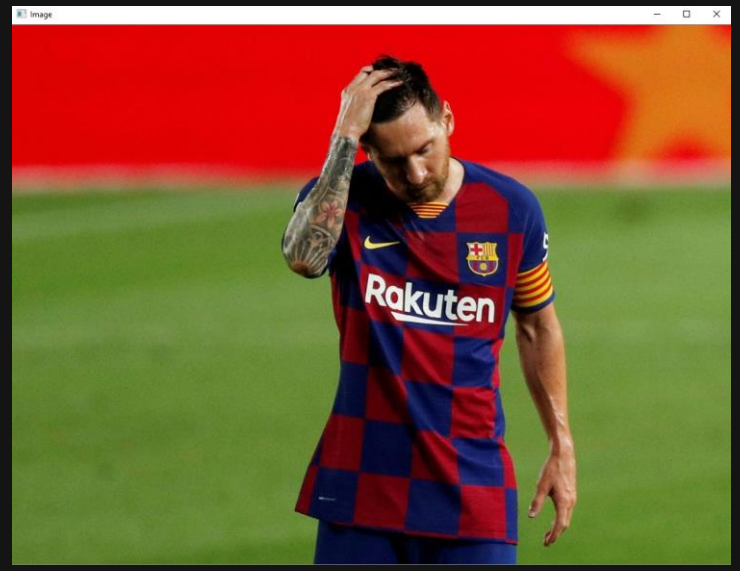


Image Transformation

SECTION 2

Image Transformation

There are common techniques applied to images, including translation, rotation, re-sizing, flipping, and cropping. Adobe Photoshop implements low-level and mid-level feature transformations. A few tools that Photoshop provides are easy to code. Let us explore a few of these.

1. Translation
2. Re-sizing
3. Rotation

Translation

SECTION 3

Translation

- The translation is a technique for shifting of an image along the x-axis and y-axis. Used to shift an image up, down, left, or right, along with any combination of the above translation is a good place to start.

Translation

- The actual translation takes place on lines 3-5, where we initially define our translation matrix **translation_matirx**. This matrix handles the number of pixels that shift in any of the directions. The translation matrix “translation_matrix“ is a floating-point array.
- OpenCV expects the matrix to be of floating-point type. The first row of the matrix is **[1, 0, shift_num_x]**, where **shift_num_x** is the number of pixels of shift for the image. Negative values of **shift_num_x** will shift the image to the left and positive values will shift the image to the right.
- The second row of the matrix is **[0, 1, shift_num_y],** where **shift_num_y** is the number of pixels the image shifts, up or down. The negative value of **shift_num_y** will shift the image up and positive values will shift the image down.

Translation

- Now that we define the translation matrix, the actual translation takes place on Line 4 using the cv2.warpAffine function.
- What are the parameters for the cv2.**warpAffine** function?
- The first argument is the image of interest and the second argument is the translation matrix **translation_matrix**.
- The dimensions (width and height) of our image are provided in place of the third argument. Line 5 shows the results of the translation.

cv2.**warpAffine**

- Translation is the shifting of object's location. If you know the shift in (x,y) direction, let it be (t_x, t_y), you can create the transformation matrix $\boldsymbol{M}$ as follows:

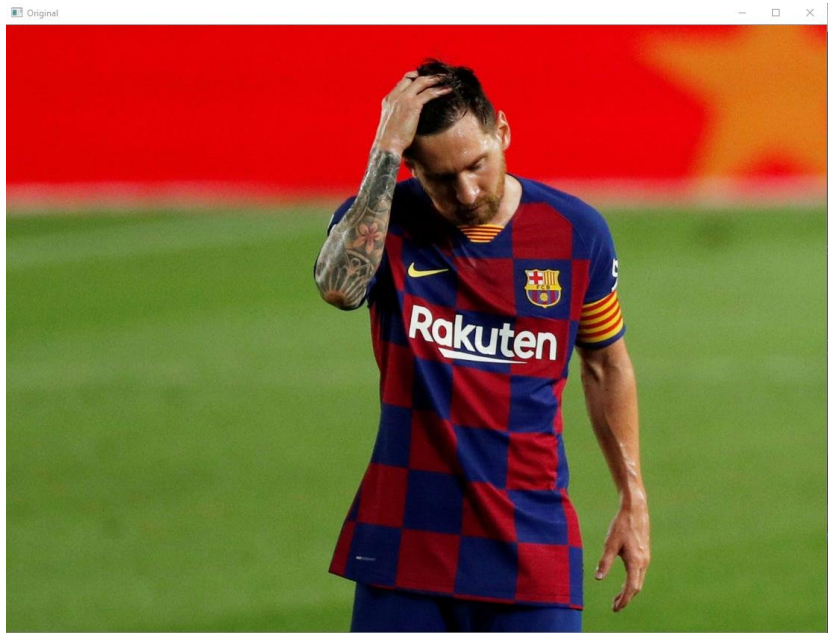
$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

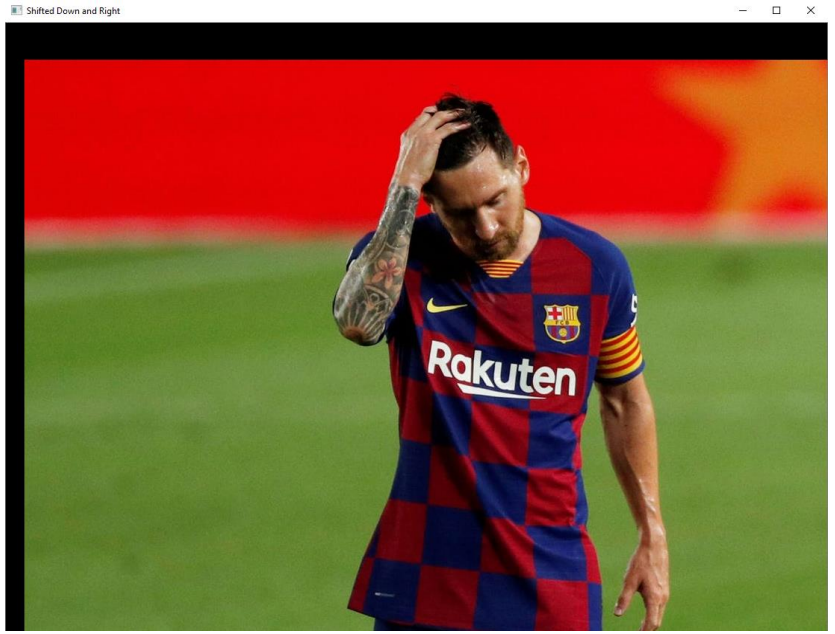
- You can take make it into a Numpy array of type np.float32 and pass it into **cv2.warpAffine()** function. See below example for a shift of (100,50):

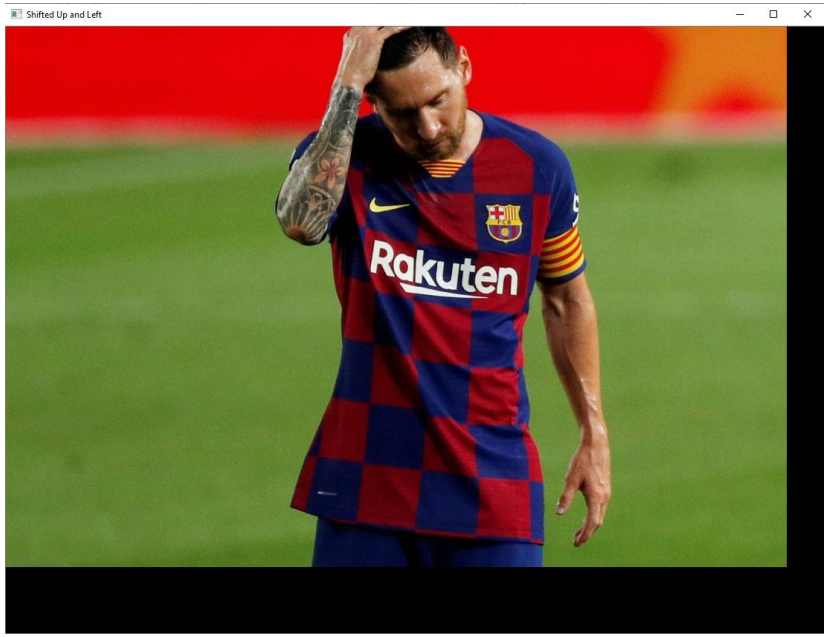
Demo Program:

translation.py

```
import cv2                  #(numOfCol, numOfRow)=(image.shape[1], image.shape[0])
import numpy as np

image = cv2.imread("messi.jpg")
cv2.imshow("Original", image)
translation_matrix = np.float32([[1, 0, 25], [0, 1, 50]])
shifted = cv2.warpAffine(image, translation_matrix, (image.shape[1],
image.shape[0]))
cv2.imshow("Shifted Down and Right", shifted)

M = np.float32([[1, 0, -50], [0, 1, -90]])
shifted = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
cv2.imshow("Shifted Up and Left", shifted)

while True:
    if cv2.waitKey(100) == 27: break
cv2.destroyAllWindows()
```


Original
Rakuten


Shifted Down and Right
Rakuten


Shifted Up and Left
Rakuten

Resizing

SECTION 4

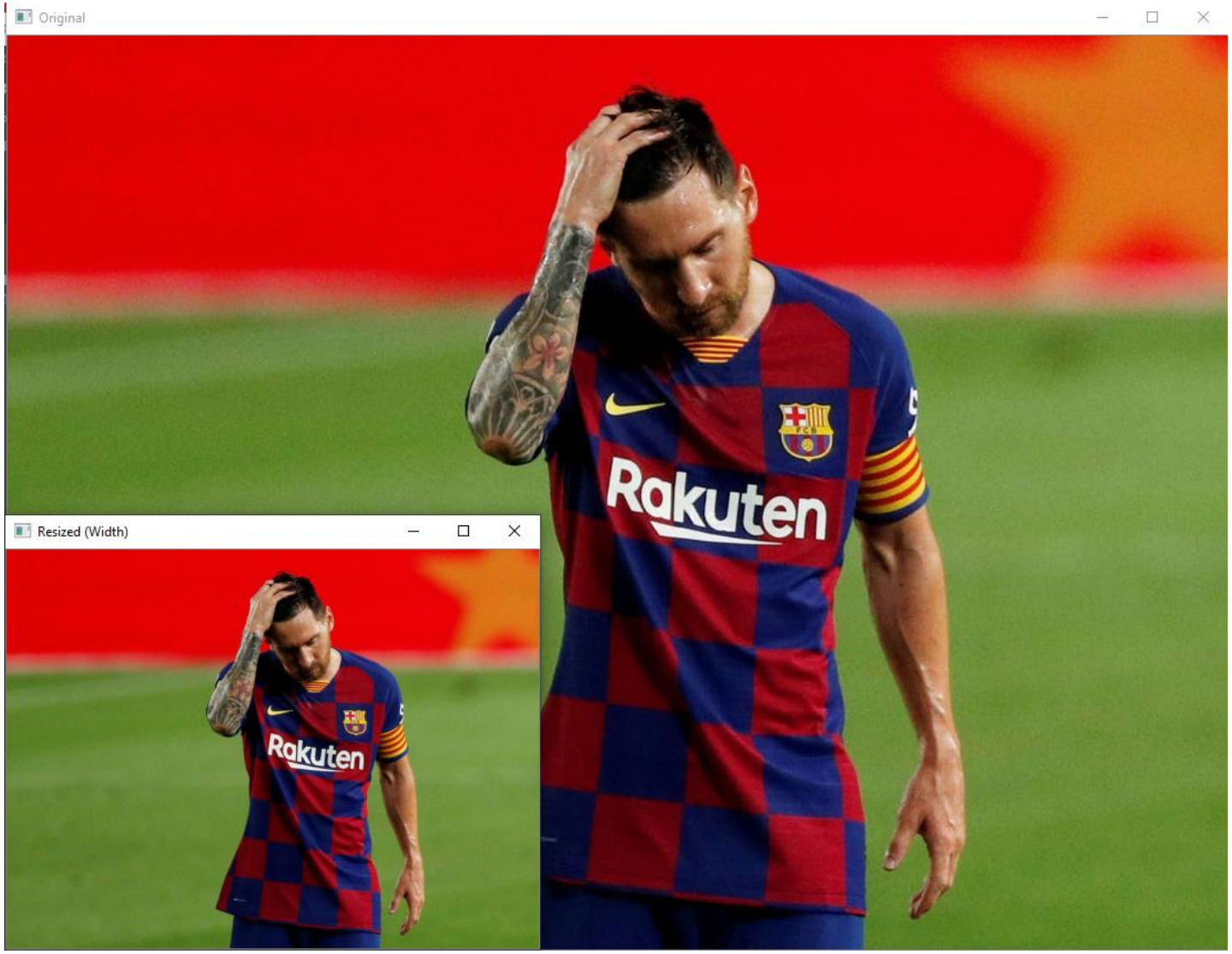
Resize

- Re-sizing is the next transformation. The inbuilt function provided by OpenCV accomplishes this task. An example always makes it easier to understand.
- In the above code, re-sizing happens in lines 6-8. In line 6, we calculate the aspect ratio of the image.

Resize

Demo Program: resize.py

```
import cv2
import numpy as np

image = cv2.imread("messi.jpg")
cv2.imshow("Original", image)
r = 480.0 / image.shape[1]
dim = (480, int(image.shape[0] * r))
resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
cv2.imshow("Resized (Width)", resized)

while True:
    if cv2.waitKey(100) == 27: break
cv2.destroyAllWindows()
```


Original
Resized (Width)
Rakuten
Rakuten

Rotation

SECTION 5

Rotation

- Rotation is the transformation for rotating the image by an angle theta. It is useful to rotate images, as implemented in the photo editors on our phones and computers. Consider implementing this on your own, with the knowledge from the earlier two examples.
- Given this problem of rotation, how do you approach it? The solution always lies in the questions we ask ourselves, right? We mention the code below, in case you get stuck.

Rotation

- The point around which anything rotates decides everything about the rotation and hence is of utmost importance. The same applies for images, and the point of rotation is the point of concern. We usually rotate the image around the center of an image; However, OpenCV allows us to specify any arbitrary point of rotation.
- We shall go ahead with rotation around the center of the image. Lines 8 and 9 calculates the center of the image by dividing the length and width of the image by 2. Integer division represented by "//" ensures integer-type results.
- A matrix like the one used to translate an image will be required here. Instead of manually constructing the matrix using NumPy, a call to the cv2.**getRotationMatrix2D** method on Line 9 does the job.

cv2.**getRotationMatrix2D**

The cv2.getRotationMatrix2D function works with three arguments:

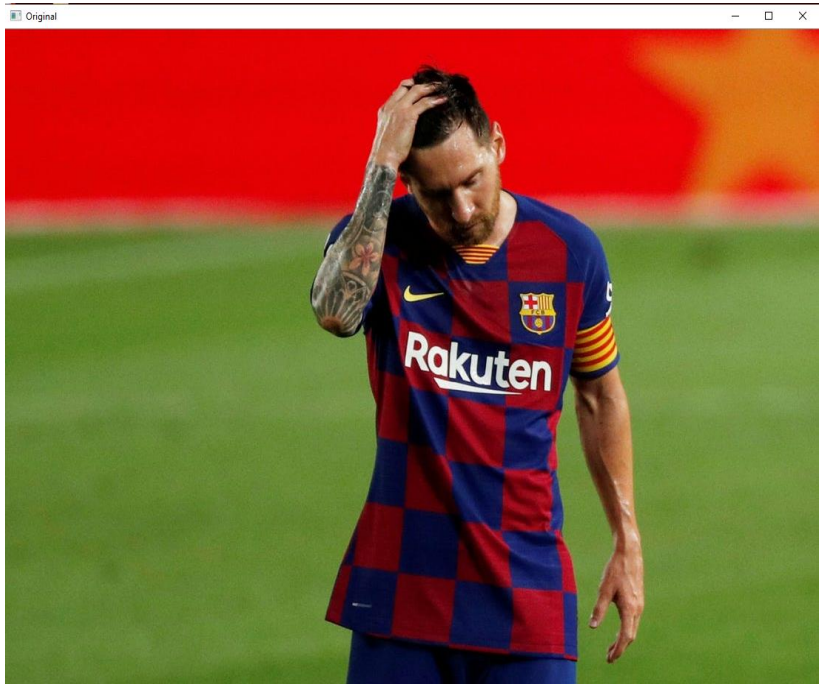
- **p**: the point of rotation
- **theta**: the angle of rotation(60 in this case)
- **scale**: the scale of the image.

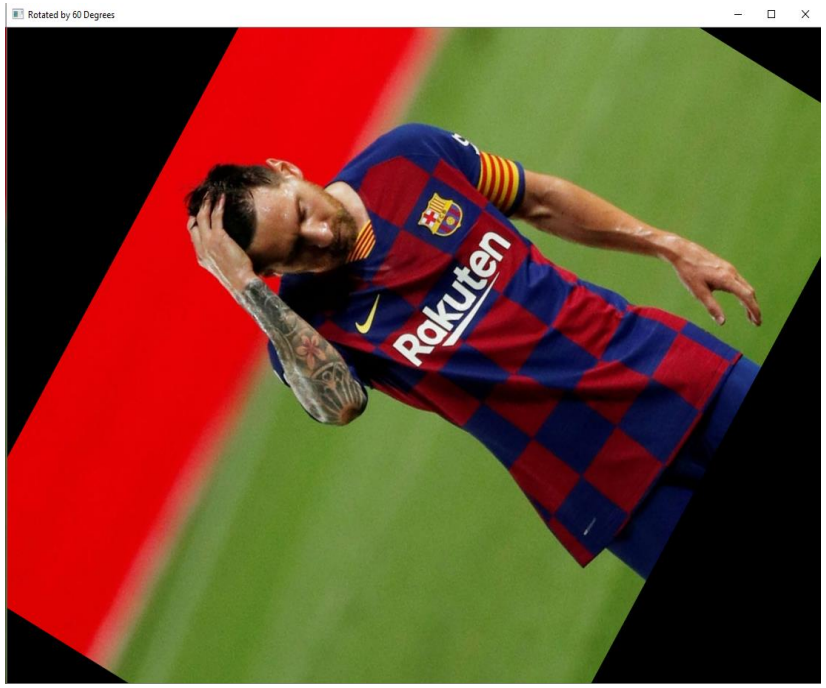
Similarly, there are various other transformations that you can experiment with.

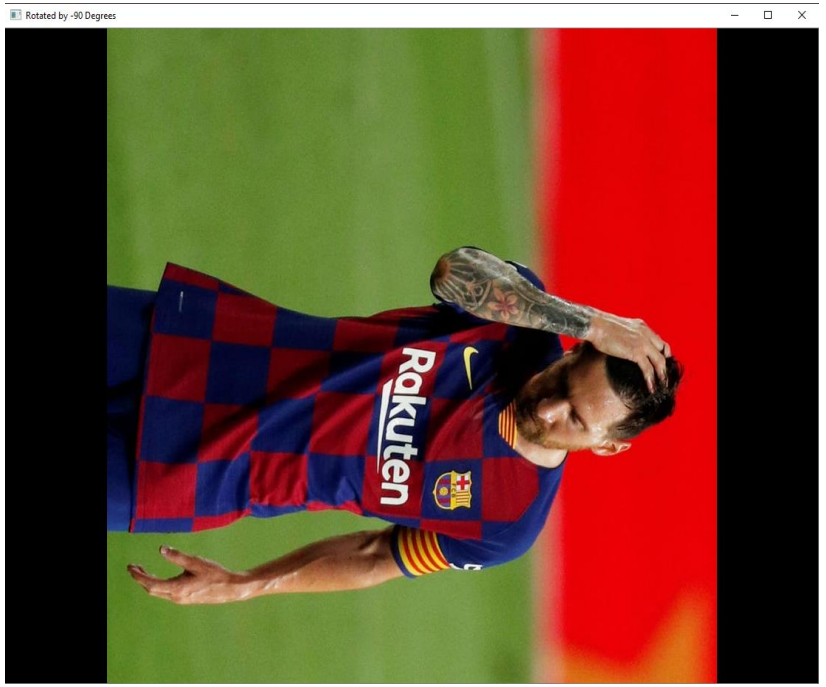
Rotation

Demo Program: rotation.py

```
import cv2

image = cv2.imread("messi.jpg")
cv2.imshow("Original", image)
(h, w) = image.shape[:2]
center = (w // 2, h // 2)
matrix = cv2.getRotationMatrix2D(center, 60, 1.0)
rotated = cv2.warpAffine(image, matrix, (w, h))
cv2.imshow("Rotated by 60 Degrees", rotated)
matrix = cv2.getRotationMatrix2D(center, -90, 1.0)
rotated = cv2.warpAffine(image, matrix, (w, h))
cv2.imshow("Rotated by -90 Degrees", rotated)

while True:
    if cv2.waitKey(100) == 27: break
cv2.destroyAllWindows()
```


Original
Rakuten


Rotated by 60 Degrees
Rakuten


Rotated by -90 Degrees
Rakuten

Color

SECTION 6

Color Spaces

- Color spaces refer to the coordinate system of the geometry of colors in the image. For example, BGR and grayscale are two examples of the same.
- There are multiple color spaces and many methods available in OpenCV that do the job of conversion between color spaces.

Where is Gray-color Space useful?

- Three-color spaces are in major use in the modern-day computer vision society: **Gray**, **RGB**, Hue, Saturation, Value (**HSV**). **Gray** works with one feature channel and thus eliminates the role of color, translating to shades of grey.
- It is used heavily during the intermediate processing phase, such as face detection.
- BGR is the blue-green-red color space. Web developers work with this color space with a slight difference in the order of colors: RGB. The ordering of BGR is unique to OpenCV. Finally, with HSV, the hue is the color tone, saturation is the intensity of a color, and the value is its darkness. This is a color space that is counter-intuitive and working with this color space shall deepen our understanding of the subject.

Frequency Domain

SECTION 7

The Fourier Transform

- Image processing is a digital task because of the involvement of processors. The processors need to complete the tasks fast, and for high efficiency and throughput, one must have highly efficient algorithms. Processing images in their raw form of multidimensional arrays is not an efficient choice as attaining top processing speeds is a challenge.
- Thus, the conversion of these digital signals into the frequency domain is of utmost importance.

What are the advantages derived in the frequency domain?

- In the frequency domain, we alter the characteristics of the image in a way that enables parallel processing and a lesser number of computations to arrive at the same results. We represent a signal as a sum of intermediate signals, thus enabling parallel processing. The Fourier Transform converts images in the spatial domain to the frequency domain.

Who discovered the Fourier Transform?

- Joseph Fourier was an 18th-century French mathematician. He discovered that we can express waveforms as the sum of simple waveforms of various frequencies.
- The waveforms we see are the sum of waveforms of different frequencies equal to or smaller than the actual waveform's frequency.

How do we use the Fourier Transform in digital image processing?

- This concept is useful in manipulating images because it allows us to find regions in images where an image pixel changes a lot and regions where the change is less. Using these characteristics, we can then term these as the region of interest(ROI), or noise, or the foreground of an image or background.
- These divisions can help in faster processing and extrapolation of interesting patterns and results.

Filters

SECTION 7

High-pass filter

- A high-pass filter (HPF) allows only higher frequencies to pass through and it blocks the lower frequencies. The threshold set for the filter is called the cut-off point. An HPF is a filter that examines a region of an image and increases the intensity of pixels.
- How does an HPF work?
- It increases the intensity of pixels in a region based on the difference in the values of the intensity in the ROI.

High-pass filter

- Consider the following kernel:

```
[[0, -1, 0],
 [-1, 4, -1],
 [0, -1, 0]]
```

- This kernel, when convoluted with an image, gives a mask that has information about all the high-frequency signals in the image.

What information do high-frequency signals convey?

- High-frequency signals add details to the image, for example, edges. Thus, the addition of this mask to the original image results in highlighting the edges present and enhancement of the image.

What is a kernel?

- A kernel is a set of weights when applied to ROI results in a single pixel in the destination image. For example, a kernel size of 7 implies that 49 (7 x 7) source pixels are considered in generating each destination pixel.
- The intensity of the central pixel is boosted (or not), after finding large variations in the sum of differences of the intensities. If a pixel stands out from the surrounding pixels, it will get boosted.

What is a kernel?

- Both high pass and low-pass filters use a property called the radius. Radius is the reach of the kernel in the image in one stride. Below is an example for an HPF:

```
import cv2
import numpy as np
import copy
from scipy import ndimage

kernel_3x3 = np.array([[-1, -1, -1],
                       [-1, 8, -1],
                       [-1, -1, -1]])
```

- The kernel mentioned here is convoluted with the original image to get a mask.

What is a mask?

- The mask has the information that the filter has derived from the image via convolution.

Low Pass Filter

- While HPF works with the intensity difference in images, a low-pass filter (LPF) smoothens the pixels by averaging the ROI. The output is a single-pixel value which is the average of the pixels in the kernel. Applications for LPF are blurring and denoising. An example of the same is the Gaussian filter used to blur the images.

Low Pass Filter

- LPF kernel example

$$\frac{1}{8}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Edge Detection

SECTION 7

Edge Detection

- Edges are the lowest level of features, and they are also the first piece of information that we as humans infer about through our visual system. Usually, most MOOCs start with the human visual system. It creates a sense of intrigue in anyone who reads about it. Software, too, can reason about edges, poses, and archetypes.

Edge Detection

- How do we implement filters on OpenCV?
- OpenCV lists many edge detecting filters.
  1. Laplacian()
  2. Sobel()
  3. Scharr()
- These filters are faulty, not in terms of the implementation, but by their algorithms itself. The expected outcome is to turn non-edge regions to black while turning regions having edges to saturated colours. The faulty nature arises when the algorithms work with noisy images, wherein it identifies noise as edges.

Why is the noise identified as an edge?

- Noise can be of many types, like uniform noise, impulsive noise, and Gaussian noise. With impulsive noise, the values of the pixels transform and thus the difference in the intensity of pixels is high, thus getting misclassified as an edge.

How do we prevent misclassification of noise as an edge?

- We can eradicate this flaw. Logically thinking, we need to remove the impulsive increase in the intensity values occurring in the image. This can be done by passing the image through an LPF, which ends up blurring the image before trying to find its edges. OpenCV provides a range of blurring filters, including **blur()**, **medianBlur()**, and **GaussianBlur()**. The arguments are implementation-dependent, and we suggest you read the documentation to get a better understanding of the arguments in each of these filters.

How do we prevent misclassification of noise as an edge?

- For blurring, let us use **medianBlur()**, which is effective in removing digital video noise, especially in colour images. For edge-finding, let us use **Laplacian()**, which produces bold edge lines, especially in grayscale images. After applying **medianBlur()**, but before applying **Laplacian()**, we should convert the image from BGR to grayscale.

How do we prevent misclassification of noise as an edge?

- Once we have the result of **Laplacian()**, we can invert it to get black edges on a white background. Then, we can normalize it (so that its values range from 0 to 1) and multiply it with the source image to darken the edges. We carry out the function that gets the task done:

How do we prevent misclassification of noise as an edge?

- Note that we allow kernel sizes to be specified as arguments for **strokeEdges()**. We use the **blurKsize** argument as kernel size for **medianBlur()**, while **edgeKsize** is used as kernel size for **Laplacian()**.


Rakuten


Rakuten

Contour Detection

SECTION 8

Contour Detection

- Another important computer vision technique is called contour detection that has gained popularity among scientists and researchers. It deals with the aspect of detecting contours in ROI in an image. It gains importance because of its derivative operations connected with identifying contours.

Contour Detection

- What are the operations related to contour detection?

1. generating bounding boxes
2. approximating shapes
3. Calculating ROIs

- ROIs simplify the complexity involved with the interaction with images. We use these concepts in the fields of object detection and face detection.

Contour Detection

- First, create an empty black image that is 200×200 pixels in size. Then, a black square is placed in the center of the square. Thresholding is done, and we call the **findContours()** function.
- What are the parameters of **findContours()** function in OpenCV?
    1. the selected image to be fed in as input
    2. type of hierarchy
    3. the contour approximation method.

Contour Detection

- Several aspects are of particular interest in this function. The hierarchy tree returned by the function is important: cv2.RETR_TREE will find all the contours present in the image and thus enable us to find relationships between contours. If we only want to retrieve the most external contours, we make use of cv2.**RETR_EXTERNAL**. This is useful when we want to drop contours that are a subset of other contours.

What is the return type of findContours function?

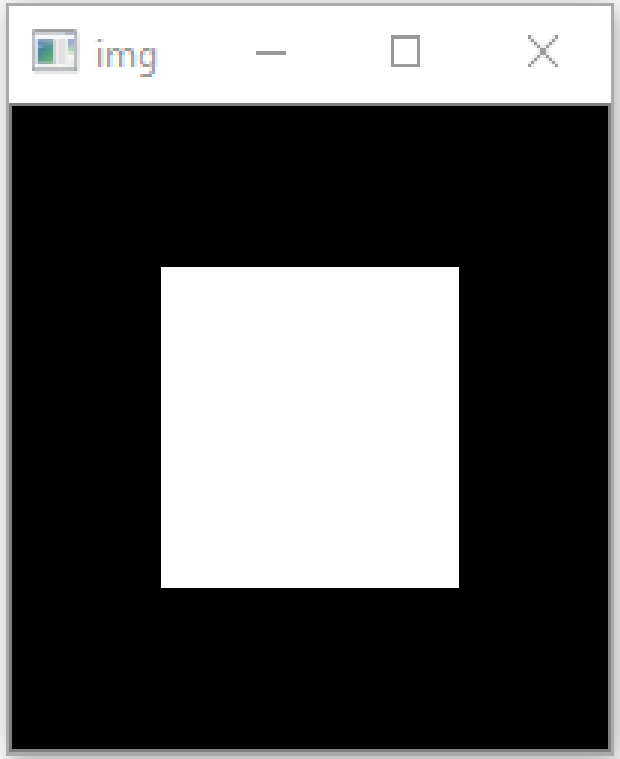
- It returns two elements:
    1. Contours
    2. Hierarchy
- We use the contours to draw on the color version of the image (so we can draw contours in green) and display it. The result is a white square with its contour drawn in green.

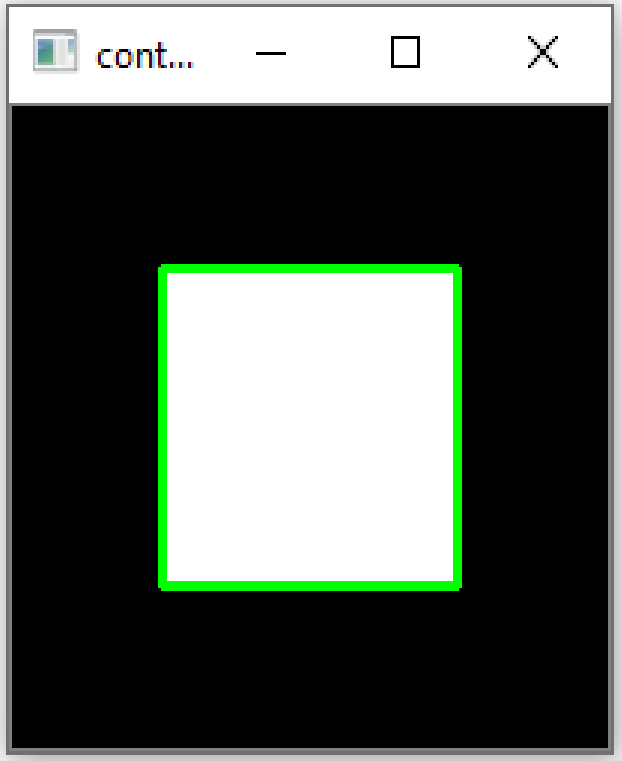
Finding Contour

Demo Program: contour.py

```
import cv2
import numpy as np

img = np.zeros((200, 200), dtype=np.uint8)
img[50:150, 50:150] = 255
cv2.imshow("img", img)

ret, thresh = cv2.threshold(img, 127, 255, 0)
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
img = cv2.drawContours(color, contours, -1, (0, 255, 0), 2)
cv2.imshow("contours", color)

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

Finding Contour





Summary

SECTION 9

Summary

- We covered various algorithms related to the basics of image processing. We hope you enjoy implementing the same and have fun with the many algorithms learned in the article. These basics learned well shall take us a long way in our journey in image processing and computer vision.
- This is part 1 in a 4-article series on Computer Vision. Be sure to read the next article that deals with the basics of **deep learning applied in the domain of computer vision**.