3. OpenCV Basics

This chapter will introduce some basic operations supported by OpenCV, such as opening image or video files and displaying them, converting color images to grayscale, or black-white images, connecting to a webcam of a laptop and showing the videos captured by it. This chapter will also introduce the fundamentals of digital images, like pixels, and color spaces.

Section 3.1 is to load and display a color image and a grayscale image.

Section 3.2 is to load and display videos. Section 3.3 to connect a webcam of the laptop and show it.

Section 3.4 is to explain the fundamentals of images including the pixels and color spaces.

Section 3.5 is to draw shapes including lines, rectangles, circles, ellipses and polylines. And section 0 to draw texts.

In section 3.7, we will apply the knowledge learned in this chapter to draw a graph that is similar to the OpenCV icon.

Enjoy the OpenCV basics.

3.1. Load and Display Images

```
Source:          ShowImage.py
```

After the PyCharm IDE is successfully installed and configured to use Python with OpenCV, now we are going to load an image and display it.

3.1.1. Load Color Images

In PyCharm, if you have the Github project loaded, then click on the *ShowImage.py* file, the image file is located in the *res* folder called *flower004.jpg*.

Alternatively, if you do not use the source files in the Github repository, create a new Python file called *ShowImage.py*, or whatever you like, copy and paste the following codes, and make sure you have your image file in your PyCharm project folder and correctly point to it.

```
1 import cv2
2 img = cv2.imread("../res/flower004.jpg")
3 cv2.imshow("Image", img)
4 cv2.waitKey(0)
5 cv2.destroyAllWindows()
```

Explanations:

Line 1	*`import cv2` tells the Python to include the `cv2` (OpenCV) library. Every time when using OpenCV, must import the `cv2` package. This is typically always the first statement in the code file.*
Line 2	*`cv2.imread()` is the function to load an image, the image file path is specified in the argument.*
Line 3	*`cv2.imshow()` is the function to show the image. The first argument is the title of the window that displays the image, the second argument is the image that returned from `cv2. imread()` function.*
Line 4	*Wait for a keystroke. If do not wait for a keystroke, the `cv2.imshow()` will display the window and go to the next immediately, the execution will complete and the window will disappear, this happens very quickly so you can hardly see the result. So `cv2.waitKey()` is typically added here to wait for a user to press a key.*

Line 5	*Destroy all windows before the execution completes.*

Run the code, the loaded image is displayed, as shown in Figure 3.1:

Figure 3.1 Show Color Image

Please note, all images included in this book are for reference only, they might not be the same as the results of executing the codes, it's highly recommended to set up the working environment on your PC. You will be able to see the actual images on the screen by executing the codes.

`cv2.imread()` is an OpenCV function used to load the image from a file, here is its syntax:

Syntax	*cv2.imread(path, flag)*
Parameters	**path:** *The path of the image to be read.* **flag:** *Specifies how to read the image, the default value is cv2.IMREAD_COLOR.* **cv2.IMREAD_COLOR:** *Load a color image. Any transparency of the image will be neglected. It is the default flag. You can also use 1 for this flag.* **cv2.IMREAD_GRAYSCALE:** *Load an image in grayscale mode. You can also use **0** for this flag.* **cv2.IMREAD_UNCHANGED:** *Load an image as such including alpha channel. You can use **-1** for this flag.*
Return Value	*The image that is loaded from the image file specified in the `path`*

	parameter

3.1.2. Load Grayscale Images

Now let's load this image file in grayscale mode and display it.

Figure 3.2 Show Grayscale Image

Just simply replace the line 2 in above codes with the following, it tells the `cv2.imread()` to load image in grayscale mode.

```
2   img = cv2.imread("../res/flower004.jpg",
                     cv2.IMREAD_GRAYSCALE)
```

Execute the code and the grayscale image is shown as Figure 3.2.

3.1.3. Convert Color Image to Grayscale

An alternative way to have a grayscale image is to load the original image first, then use `cv2.cvtColor()` function to convert it to a grayscale image, this way we will have both original and grayscale images available for further processing. This is useful because in the future when we do the image processing, we want to process the image in grayscale mode while displaying the original color image.

Now replace the code line 2 and 3 in section 3.1.1 with the following:

```
1  img = cv2.imread("../res/flower004.jpg")
2  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
3  cv2.imshow("Image", img)
4  cv2.imshow("Image Gray", gray)
```

`cv2.cvtColor()` is an OpenCV function used to convert an image from one color space to another, here is its syntax:

Syntax	*cv2.cvtColor(src, code[, dst[, dstCn]])*
Parameters	**src:** source image to be converted. **code:** color space conversion code. **dst:** optional, the output image of the same size and depth as *`src`* image. **dstCn:** optional, the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from *`src`* image. **cv2.COLOR_BGR2HSV/COLOR_HSV2BGR:** *Convert between BGR color image and HSV space.* **cv2.COLOR_BGR2GRAY/COLOR_GRAY2RGB:** *Convert between BGR color image and grayscale image* *Other conversion codes are not listed here, please reference OpenCV documents.*
Return Value	*The image that is converted from the source image*

Execute the codes, the color image and grayscale image are displayed side by side, as shown below.

Color Image *Grayscale Image*

Figure 3.3 Convert a Color Image to Grayscale Image

3.2. Load and Display Videos

> Source: ShowVideo.py

We were able to load and display an image, now we are going to work with videos, and see how OpenCV can process videos.

Open the *ShowVideo.py* file in PyCharm. If you are not using the Github project, then create a new Python file, and make sure you have a video file available. There is an mp4 video file called "*Sample Videos from Windows.mp4*" in the Github project, it will be loaded and displayed in this example. Below is the code,

```
1  #
2  # Show a video from local file
3  #
4  import cv2
5
6  cap = cv2.VideoCapture("../res/Sample Videos from
   Windows.mp4")
7
8  success, img = cap.read()
```

```
 9   while success:
10       cv2.imshow("Video", img)
11       # Press ESC key to break the loop
12       if cv2.waitKey(15) & 0xFF == 27:
13           break
14       success, img = cap.read()
15   cap.release()
16   cv2.destroyWindow("Video")
```

Explanations:

Line 6	*Use `cv2.VideoCapture()` to load a video stream, the function returns a video capture object.*
Line 8	*Read the first frame from the video capture object, the frame image is stored in `img` variable, and the result is stored in `success` variable indicating True or False.*
Line 9-14	*Loop frame by frame until all frames in the video object are read, within the loop the image frames are processed one by one throughout the video.*
Line 10	*Use `cv2.imshow()` to display a frame. Each frame is an image.*
Line 12-13	*Wait for 15 milliseconds and accept a keystroke, if ESC key (keycode is 27) is pressed then break the loop. Changing the `cv2.waitKey()` parameter will change the speed of playing the video.*
Line 14	*Same as Line 8, load subsequent frames from the video capture object.*
Line 15	*Release the video capture object to release the memory.*
Line 16	*Close the Video window*

Execute the codes, it will load the video and play it in a window called `Video`. It will play until either the end of the video or ESC key is pressed.

The video is a sample from Windows Vista, it's located at: *https://github.com/jchen8000/OpenCV/raw/master/res/Sample%20Videos%20from%20Windows.mp4*.

The original video is from *https://www.youtube.com/watch?v=K1ShYerq6lg*

In Line 12 the `cv2.waitKey(15)` will wait for 15 milliseconds between each frame, changing the parameter value will change the speed of playing the video. If the parameter value is smaller then the play speed is faster, larger then slower.

Inside the loop from Line 9 to 14 you can process each image before showing it, for example you can convert each image to grayscale and display it, you will play the video in grayscale.

Replace the above Line 10 and 11 with the following two lines, it will convert the image from color to grayscale for every image frame inside the video, as a result, the video will be played in grayscale.

```
10      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
11      cv2.imshow("Video", gray)
```

Similarly, you can do other image processing inside the loop, for example recognize the people or faces and highlight them in the video. These will be explained in later this book.

3.3. Display Webcam

```
Source:          ShowWebcam.py
```

Like displaying videos, a similar technique is used to display webcam. Replace the above Line 6 with `cap = cv2.VideoCapture(0)`, it will load the laptop/desktop's default webcam and display it.

In the above section the parameter of `cv2.VideoCapture()` function was the path of the video file, now pass the index of the webcam device as parameter, here 0 is used as the default webcam, it will connect to the default webcam.

Some video properties can also be set here, as below,

```
1   import cv2
2
```

```
3   cap = cv2.VideoCapture(0)
    # read from default webcam
4
5   # Set video properties
6   cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
    # set width
7   cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
    # set height
8   cap.set(cv2.CAP_PROP_BRIGHTNESS, 180)
    # set brightness
9   cap.set(cv2.CAP_PROP_CONTRAST, 50)
    # set contrast
10
11  success, img = cap.read()
12  while success:
13      cv2.imshow("Webcam", img)
14
15      # Press ESC key to break the loop
16      if cv2.waitKey(10) & 0xFF == 27:
17          break
18      success, img = cap.read()
19
20  cap.release()
21  cv2.destroyWindow("Webcam")
```

Explanation

Line 3	*Load video from default Webcam.*
Line 6	*Set width of the camera video.*
Line 7	*Set height of the camera video.*
Line 8	*Set brightness of the camera video.*

Line 9	*Set contrast of the camera video.*

For references, this is the list that can be used as parameter for `cap.set()`:

0	*CAP_PROP_POS_MSEC*	*Current position of the video file in milliseconds.*
1	*CAP_PROP_POS_FRAMES*	*0-based index of the frame to be decoded/captured next.*
2	*CAP_PROP_POS_AVI_RATIO*	*Relative position of the video file*
3	*CAP_PROP_FRAME_WIDTH*	*Width of the frames in the video stream.*
4	*CAP_PROP_FRAME_HEIGHT*	*Height of the frames in the video stream.*
5	*CAP_PROP_FPS*	*Frame rate.*
6	*CAP_PROP_FOURCC*	*4-character code of codec.*
7	*CAP_PROP_FRAME_COUNT*	*Number of frames in the video file.*
8	*CAP_PROP_FORMAT*	*Format of the Mat objects returned by retrieve() .*
9	*CAP_PROP_MODE*	*Backend-specific value indicating the current capture mode.*
10	*CAP_PROP_BRIGHTNESS*	*Brightness of the image (only for cameras).*
11	*CAP_PROP_CONTRAST*	*Contrast of the image (only for cameras).*
12	*CAP_PROP_SATURATION*	*Saturation of the image (only for cameras).*
13	*CAP_PROP_HUE*	*Hue of the image (only for cameras).*
14	*CAP_PROP_GAIN*	*Gain of the image (only for cameras).*
15	*CAP_PROP_EXPOSURE*	*Exposure (only for cameras).*
16	*CAP_PROP_CONVERT_RGB*	*Boolean flags indicating whether images should be converted to RGB.*
17	*CAP_PROP_WHITE_BALANCE*	*Currently unsupported*
18	*CAP_PROP_RECTIFICATION*	*Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x*

		backend currently)

Make sure a webcam is attached to the laptop/desktop computer and enabled, execute the code you will see a window displaying whatever the webcam captures. Press ESC key to terminate.

3.4. Image Fundamentals

```
Source:          ImageFundamental.py
```

3.4.1. Pixels

Pixels (short for "picture elements") are the smallest units of a digital image. Each pixel represents a tiny point of color that, when combined with other pixels, forms the complete image.

Pixels are typically arranged in a grid within a rectangle or square, with each pixel having a specific location within the image. The color of each pixel is represented by a combination of numerical values that represent the intensity of the three primary colors of blue, green, and red (BGR).

The resolution refers to the number of pixels in the image, usually expressed in terms of the width and height of the image in pixels. The higher the resolution, the more detail the image can contain, as there are more pixels available to represent the image. However, higher resolution images also require more storage space and processing power.

Typically, a digital image is made of thousands or millions of pixels, which are organized in rows and columns. For example, for an image of 640 x 480, there are a total of 307,200 pixels, and they are located in 480 rows and 640 columns. The coordinates of a pixel specify the location of the pixel, say a pixel with coordinates of (100, 100) means it is in column number of 100 and row number of 100.

Unlike a mathematics coordinate system, the digital image's coordinate of the origin (0,0) is located at the top left corner of the image. x-axis represents the columns and y-axis represents the rows.

As a color image shown in Figure 3.4, x-axis is the horizontal arrow at the top facing right, and y-axis is the vertical arrow at the very left and facing down. A pixel

can be identified by a pair of integers specifying a x value (in column number) and a y value (in row number). In below Figure 3.4, the pixel at (100, 100) is identified and highlighted.

Figure 3.4 Pixels in a Color Image

For a 24-bit image, each pixel has 24 bits, it is made of blue, green and red values, each has 8 bits, which value is from 0 to 255. For example, the pixel at (100, 100) in above Figure 3.4 has blue value of 151, green of 82 and red of 234. The color of this pixel, shown as pink, is determined by these three values.

3.4.2. BGR Color Space and Channels

A digital image is represented in different color spaces, the color space refers to a specific way of representing colors in an image. It is a three-dimensional model that describes the range of colors that can be displayed or printed. There are several color spaces used in digital imaging, and each has a different range of colors and is used for specific purposes. Here we will introduce the BGR and HSV color spaces in this section and the next, both are commonly and widely used in image processing.

BGR stands for Blue, Green, and Red. It is a color space used to represent colors on electronic screens, like computer monitors, TVs, and smartphones. In this space, colors are represented by three primary colors: Red, Green, and Blue. Each primary color has a range of 0 to 255, meaning each color can have 256 possible values, which makes a total of 16.7 million (= 256 · 256 · 256) possible colors.

Each primary color is called a channel, a channel has the same size as the original image. Therefore, an image in BGR color space has three channels, blue, green and red. Figure 3.5 shows the idea of how the three channels compose a color image.

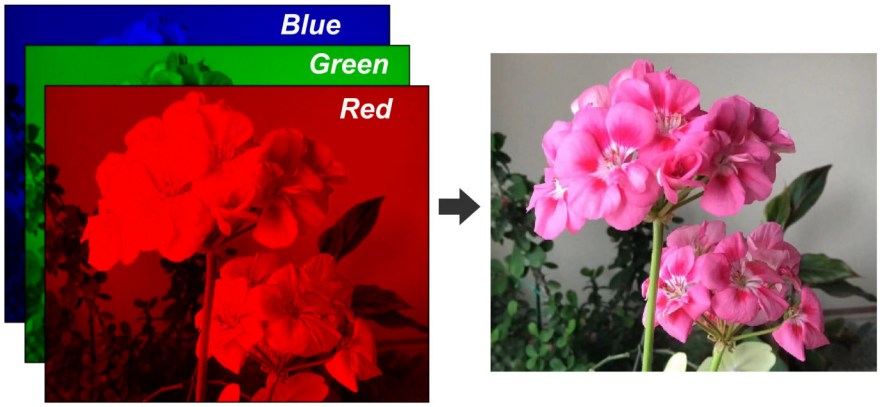

Figure 3.5 Blue, Green and Red Channels to Produce an Image

A single channel does not have any colors, it's a grayscale image. Because the three primary colors can build up a color, a single channel only has one value, which can only represent a grayscale, not a color.

Therefore, the above Figure 3.5 explains the concept, but not quite correct, because the blue, green and red channels are all in grayscale without colors. The above red channel is shown in red, looks like it is red, but that is not the case, it should be in grayscale. Similarly, the green and blue channels should be also in grayscale.

Figure 3.6 is the correct one, the blue, green and red channels are all in grayscale, they are mixed together to produce the color image.

Each channel is represented by an 8-bit value ranging from 0 to 255, the combination of the three primary colors at their maximum intensity (255, 255, 255) results in white, while (0, 0, 0) results in black, anything in between results in different colors. The same value in three channels, such as (125, 125, 125), represents a gray color.

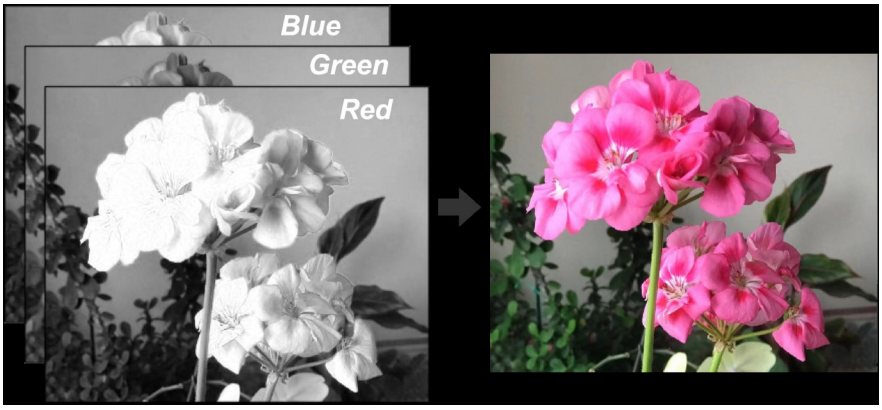

Figure 3.6 Blue, Green and Red Channels to Produce an Image

3.4.3. HSV Color Space and Channels

In addition to the BGR color space, an image can also be represented by HSV (Hue, Saturation, Value) color space, also known as HSB (Hue, Saturation, Brightness), which is a cylindrical color space that describes colors based on three attributes: hue, saturation, and value/brightness, as shown in Figure 3.7. The three attributes are also represented in channels.

In the HSV color space, colors are represented as a point in a cylindrical coordinate system. The *hue* is represented by the angle on the horizontal axis, the *saturation* is represented by the radius or distance from the center, and the *value*/*brightness* is represented by the vertical axis.

The HSV is often used in graphics software for color selection and manipulation because it allows users to easily adjust the hue, saturation, and value/brightness of color separately. For example, changing the hue will change the color family, while changing the saturation or brightness will alter the intensity or lightness/darkness of the color.

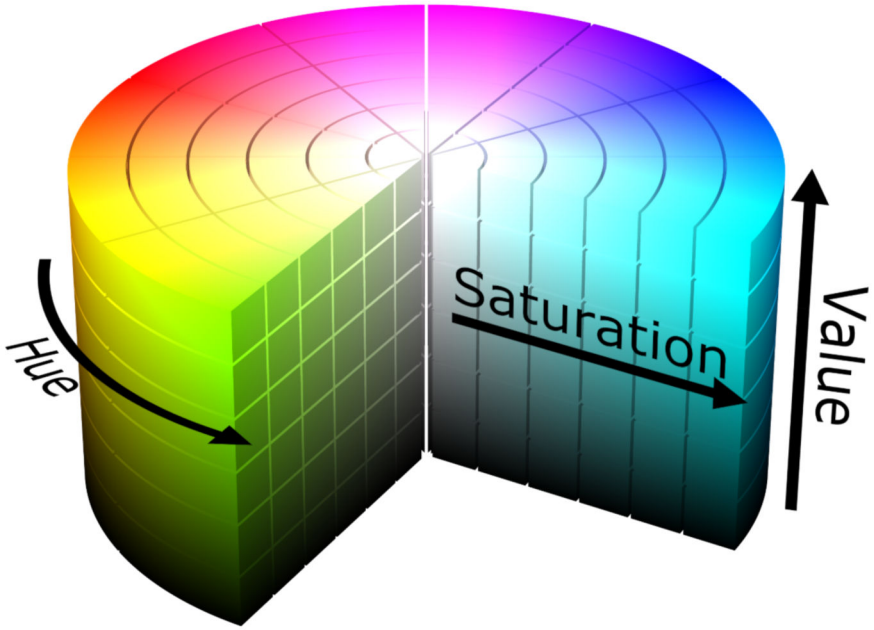

Figure 3.7 HSV Color Space

SharkD, CC BY-SA 3.0 via Wikimedia Commons https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSV_color_solid_cylinder_saturation_gray.png

Hue represents the color portion of the image, which is described as an angle on a color wheel ranging from 0 to 359 degrees. Figure 3.8 shows the color wheel, different colors are distributed around a circle with red at 0 degree, green at 120 degrees, and blue at 240 degrees. Hue value at different angles represent different colors.

Normally the hue value is from 0 to 359 representing an angle of the circle, however in OpenCV the values of hue are different, since the values are stored in an 8-bit datatype with the range of [0, 255], which can not store the entire hue value of [0, 359]. OpenCV is using a trick to resolve it, the hue value is divided by 2 and stored in the 8-bit datatype. Therefore, the hue in OpenCV is [0, 179].

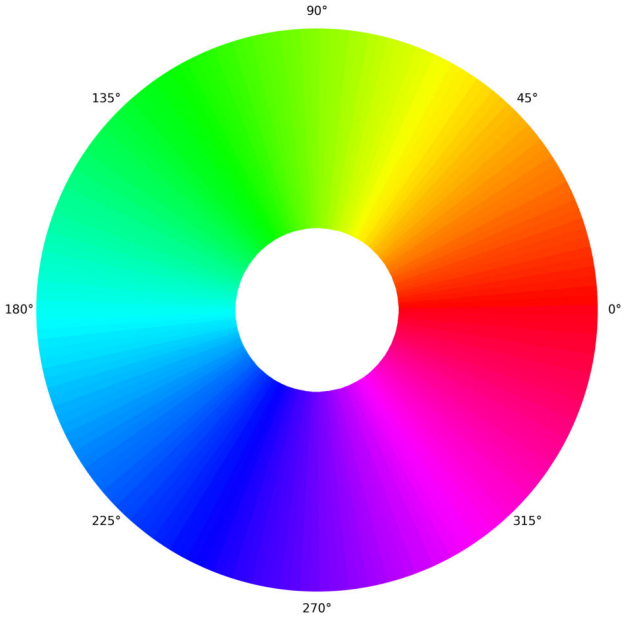

Figure 3.8 Color Wheel

Generated by source codes at common/color_wheel.py

The below table shows the Hue value at different angles and the corresponding color name and code:

Hue	Color Code	Color Name
0°	#FF0000	red
15°	#FF4000	vermilion
30°	#FF8000	orange
45°	#FFBF00	golden yellow
60°	#FFFF00	yellow
75°	#BFFF00	yellowish green
90°	#80FF00	chartreuse
105°	#40FF00	leaf green
120°	#00FF00	green
135°	#00FF40	cobalt green
150°	#00FF80	emerald green

165°	#00FFBF	bluish green
180°	#00FFFF	cyan
195°	#00BFFF	cerulean blue
210°	#0080FF	azure
225°	#0040FF	blue, cobalt blue
240°	#0000FF	blue
255°	#4000FF	hyacinth
270°	#8000FF	violet
285°	#BF00FF	purple
300°	#FF00FF	magenta
315°	#FF00BF	reddish purple
330°	#FF0080	ruby red, crimson
345°	#FF0040	carmine

The above table is from https://en.wikipedia.org/wiki/Hue

Saturation represents the intensity or purity of the color, the value is defined from 0 to 100 percent, where 0 is gray and 100 percent is the pure color. As the saturation increases the color appears to be purer, a highly saturated image is more vivid and colorful. As the saturation decreases the color appears to be faded out, a less saturated image appears towards a grayscale one.

In OpenCV, however, its value range is extended to [0, 255] instead of [0, 100].

Value/Brightness represents the overall brightness of the color, the value is defined from 0 to 100 percent, where 0 is black and 100 is the brightest level of the color. Similar to saturation, in OpenCV the range for Value/Brightness is [0, 255], instead of [0, 100].

Same as BGR channels, the HSV is also separated into three channels as well, each in grayscale. Figure 3.9 illustrates how the Hue, Saturation and Value/Brightness channels can compose a color image.

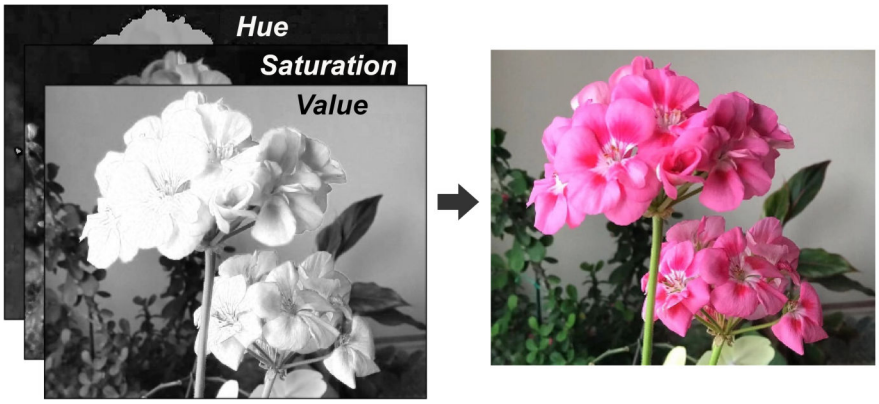

Figure 3.9 Hue, Saturation and Value Channels to Compose an Image

In summary, the HSV color space is a cylindrical color model that describes colors based on their hue, saturation, and value/brightness. It's widely used in various applications that involve color selection, manipulation, and analysis.

3.5. Draw Shapes

```
Source:          DrawShapes.py

Library:         common/Draw.py
```

In this section we will use OpenCV functions to draw the following shapes on an empty canvas,

- Lines
- Rectangles
- Circles
- Ellipses
- Polylines

3.5.1. Create an Empty Canvas

So far, the images we have used are coming from the image files, now we are going to create an empty canvas from `numpy` library for drawing. Remember when we installed OpenCV in section 2.3.2, *numpy* is also installed together with

opencv_python, so it's already available to our project. If for some reason it is not installed, install it following the descriptions in section 2.3.2.

`numpy` is a popular Python library for numerical computing that provides a powerful multi-dimensional array object and various functions for performing mathematical operations on the arrays. It provides efficient storage and manipulation of arrays and allows for fast mathematical operations on the entire array without the need for loops. It is one of the fundamental libraries for scientific computing in Python and is widely used in fields such as data science, image processing, machine learning, and engineering.

As we know an image is made of pixels in rows and columns and channels, in another word 3-dimensional array. Therefore, `numpy` is good for supporting this type of operation.

Now import `numpy` in the beginning of the code.

```
1  import cv2
2  import numpy as np
```

Then create a canvas with size of 480 in width and 380 in height, a canvas is an empty image. Remember a color image has three channels representing BGR color space, so the array we are going to create using *numpy* should have three dimensions – 380, 480, and 3. The datatype of the array is *uint8*, it contains 8-bit values ranging from 0 to 255.

```
3  canvas = np.zeros((380, 480, 3), np.uint8)
4
5  cv2.imshow("Canvas", canvas)
6  cv2.waitKey(0)
7  cv2.destroyAllWindows()
```

Explanation

Line 3	*`np.zeros()` create an array that has 380 rows, 480 columns, and 3 channels corresponding to blue, green and red. `np.zeros()` will fill the array with all 0, as we explained earlier all 0 means a black color.*

Execute the above code, the result is a window with the black canvas with the size of 480 x 380.

Now we want to paint the canvas with some color. See line 4 in the below codes, it will set values of (235, 235, 235) to the array, which means to set a color to the canvas image, this color code is blue = 235, green = 235 and red = 235, it represents light gray.

```
4  canvas[:] = 235,235,235
```

The canvas is painted in light gray, as Figure 3.10, we will draw shapes on it. If you want to paint it with different color, simply change the values in line 4.

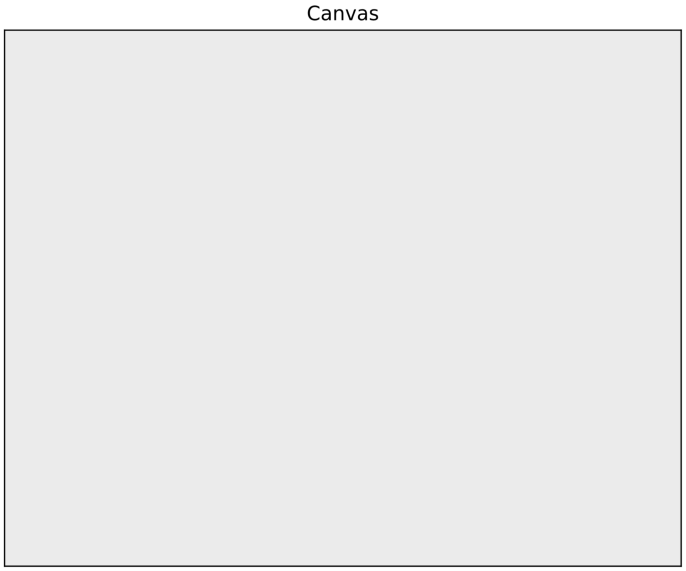

Figure 3.10 Canvas in Light Gray

3.5.2. Draw a Line

`cv2.line()` function is used to draw a line segment between start point and end point,

Syntax	*cv.line(img, pt1, pt2, color, thickness, line_type)*
Parameters	*img The canvas image.* *pt1 Start point of the line segment.* *pt2 End point of the line segment.* *color Line color.*

	thickness *Line thickness.* *line_type* *Type of the line, below are the values for line types:* *cv2.FILLED* *cv2.LINE_4* *4-connected line algorithm* *cv2LINE_8* *8-connected line algorithm* *cv2.LINE_AA* *antialiased line algorithm*

Create a function `draw_line()` to wrap the `cv2.line()` function, and set default values for `color`, `thickness` and `line_type`, when this function is invoked later don't have to specify these parameters because these default values will be used.

```
1  def draw_line(image, start, end,
2                color=(255,255,255),
3                thickness=1,
4                line_type=cv2.LINE_AA):
5        cv2.line(image, start, end, color, thickness,
                  line_type)
```

Call this function to draw a line,

```
 7  # Draw a line
 8  draw_line(canvas,
 9            start=(100, 100),
10            end=(canvas.shape[1]-100,
                   canvas.shape[0]-100),
11            color=(10, 10, 10),
12            thickness=10)
```

The result looks like Figure 3.11:

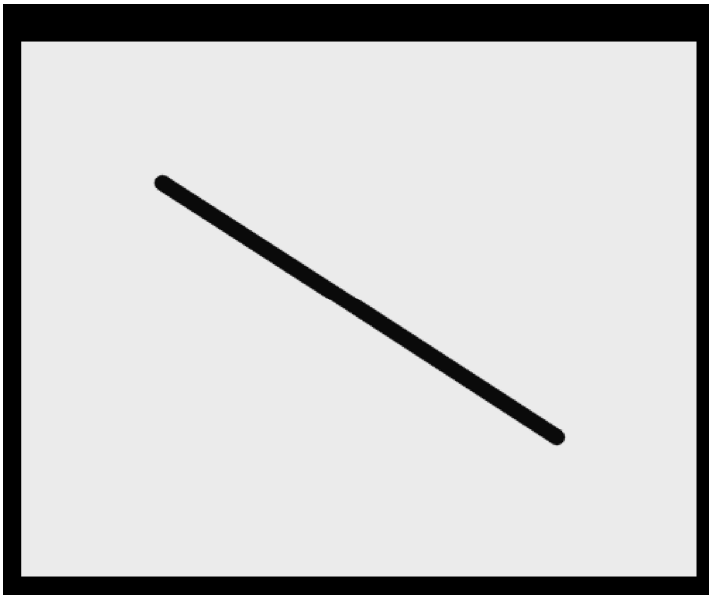

Figure 3.11 Draw a Line on the Canvas

3.5.3. Draw Rectangles, Circles, Ellipses and Polylines

Similarly draw other shapes, now begin with defining our wrapper functions for drawing shapes, like the above `draw_line()` function.

```
def draw_rectangle(image, top_left,
                   bottom_right,
                   color=(255,255,255),
                   thickness=1,
                   line_type=cv2.LINE_AA):
    cv2.rectangle(image, top_left,
                  bottom_right,
                  color, thickness, line_type)

def draw_circle(image, center, radius,
                color=(255,255,255),
                thickness=1,
                line_type=cv2.LINE_AA):
```

```
12      cv2.circle(image, center, radius, color,
                   thickness, line_type)
13
14  def draw_ellipse(image, center, axes, angle,
                     start_angle, end_angle,
15                   color=(255,255,255),
16                   thickness=1,
17                   line_type=cv2.LINE_AA):
18      cv2.ellipse(image, center, axes, angle,
                    start_angle, end_angle,
19                  color, thickness, line_type)
20
21  def draw_polylines(image, points,
                       is_closed=True,
22                     color=(255,255,255),
23                     thickness=1,
24                     line_type=cv2.LINE_AA ):
25      cv2.polylines(image, points, is_closed,
26                    color, thickness, line_type)
```

The results look like Figure 3.12:

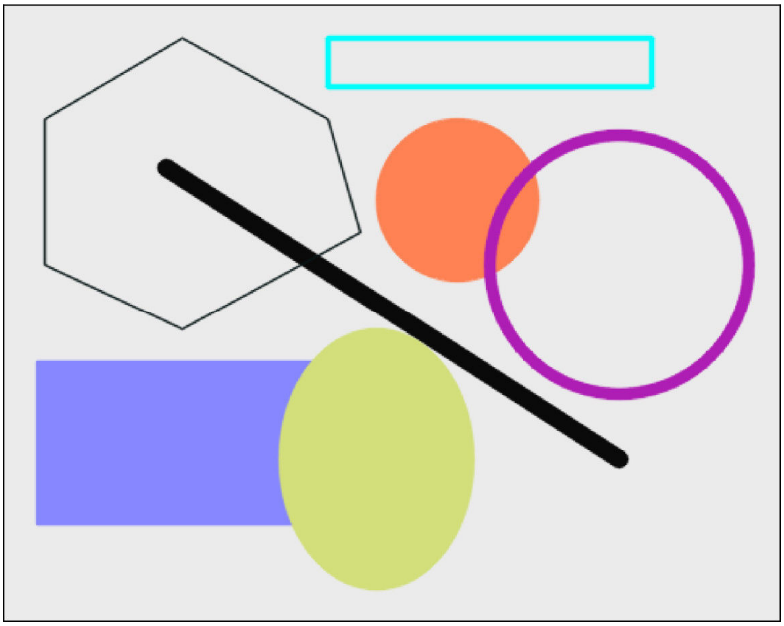

Figure 3.12 Draw Shapes on Canvas

Open *DrawShapes.py* file from the Github repository, it has all the codes to draw the shapes, execute it and the results look like Figure 3.12 above.

We don't explain the functions in the above codes one by one; they are straightforward and similar to the above `cv2.line()` function.

For details, please reference OpenCV documents for drawing functions at *https://docs.opencv.org/4.7.0/d6/d6e/group__imgproc__draw.html*

3.6. Draw Texts

```
Source:          DrawTexts.py

Library:         common/Draw.py
```

OpenCV provides functions not only for drawing shapes, but also for texts. `cv2.putText()` function is used for drawing texts.

Similarly, define a wrapper function `draw_text()`:

```
1  def draw_text(image, text, org,
2                font_face=cv2.FONT_HERSHEY_COMPLEX,
```

```
                font_scale=1,
                color=(255,255,255),
                thickness=1,
                line_type=cv2.LINE_AA):
    cv2.putText(image, text, org, font_face,
                font_scale, color, thickness,
                line_type )
```

Then create a canvas, paint it with light gray color, then call `draw_ text()` function to draw the texts.

```
canvas = np.zeros((380, 480, 3), np.uint8)
canvas[:] = 235,235,235
#Draw a text
draw_text(canvas, "Hello OpenCV", (50, 100),
          color=(125, 0, 0),
          font_scale=1.5,
          thickness=2)
cv2.imshow("Hello OpenCV", canvas)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The result is shown as Figure 3.13:

Figure 3.13 Draw Texts on Canvas

This version of OpenCV supports a limited set of fonts, below table shows the supported fonts.

Font Code	Description
FONT_HERSHEY_SIMPLEX	normal size sans-serif font
FONT_HERSHEY_PLAIN	small size sans-serif font
FONT_HERSHEY_DUPLEX	normal size sans-serif font (more complex than FONT_HERSHEY_SIMPLEX)
FONT_HERSHEY_COMPLEX	normal size serif font
FONT_HERSHEY_TRIPLEX	normal size serif font (more complex than FONT_HERSHEY_COMPLEX)
FONT_HERSHEY_COMPLEX_SMALL	smaller version of FONT_HERSHEY_COMPLEX
FONT_HERSHEY_SCRIPT_SIMPLEX	hand-writing style font
FONT_HERSHEY_SCRIPT_COMPLEX	more complex variant of FONT_HERSHEY_SCRIPT_SIMPLEX
FONT_ITALIC	flag for italic font

Reference OpenCV documents for more details at *https://docs.opencv.org/4.7.0/d6/d6e/group__imgproc__draw.html*.

3.7. Draw an OpenCV-like Icon

```
Source:          DrawTexts.py

Library:         common/Draw.py
```

Now let's use our wrapper functions to draw a complicated image, not exactly same but similar to the OpenCV Icon, like Figure 3.14.

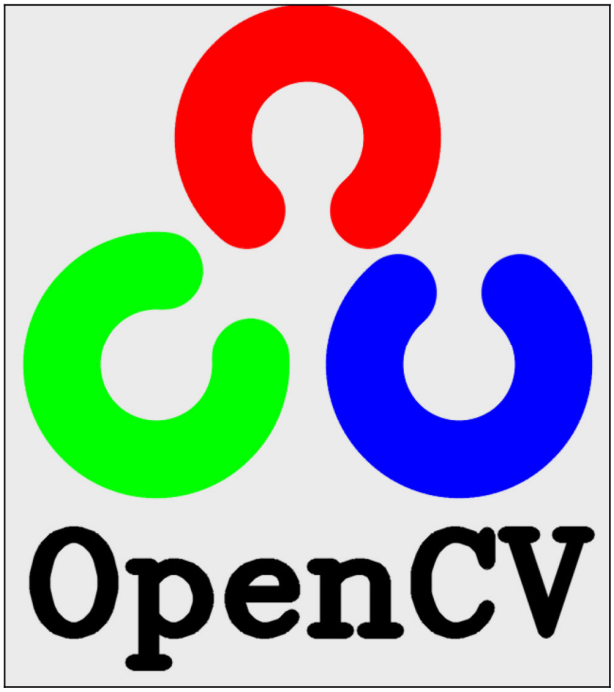

Figure 3.14 OpenCV-like Icon

A light-gray colored empty canvas of size 360 x 320 is created, same as above.

`cv2.ellipse()` function will be used to draw the three non-closed circles, a start angle and an end angle can be specified for the ellipse.

`cv2.putText()` function will be used to draw the texts at the bottom.

Below are the code snippets, the axes of the ellipse are defined as (50, 50) so it appears as a circle instead of an ellipse. The center of the three circles and start and end angles of each are defined based on the position of the OpenCV-like icon.

```
1   def draw_opencv_icon(image):
2       axes = (50, 50)
3       center_top_circle = (160, 70)
4       center_lowerleft_circle = (
            center_top_circle[0]-80,
```

```
5           center_top_circle[1]+120 )
6       center_lowerright_circle = (
            center_top_circle[0]+80,
7           center_top_circle[1]+120 )
8       angle_top_circle = 90
9       angle_lowerleft_circle = -45
10      angle_lowerright_circle = -90
11      start_angle = 40
12      end_angle = 320
13      draw_ellipse(image,center_top_circle,
                     axes,
14                   angle_top_circle,
                     start_angle, end_angle,
15                   color=(0, 0, 255),
16                   thickness=40)
17      draw_ellipse(image,
                     center_lowerleft_circle,
                     axes,
18                   angle_lowerleft_circle,
                     start_angle, end_angle,
19                   color=(0, 255, 0),
20                   thickness=40)
21      draw_ellipse(image,
                     center_lowerright_circle,
                     axes,
22                   angle_lowerright_circle,
                     start_angle, end_angle,
23                   color=(255, 0, 0),
24                   thickness=40)
25      draw_text(image, "OpenCV", (10,330),
                     color=(0,0,0),
```

```
                    font_scale=2.4, thickness=5)

if __name__ == '__main__':
    canvas = np.zeros((360,320,3), np.uint8)
    canvas[:] = 235,235,235
    draw_opencv_icon(canvas)
    cv2.imshow("Canvas", canvas)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Execute the code, the result is shown in Figure 3.14.