# OpenCV (Open Source Computer Vision)

Friday, November 25, 2022 6:24 PM

**Introduction to OpenCV**

- OpenCV is computer vision and machine learning software library mostly used in Python but also available in C, C++ java and android.
- Releases in 2000
- Used for Digital image and video processing

**Images and digital images**

- A digital image can be considered as a large array of discrete dots, each of which has a brightness associated with it.
- These dots are called ***picture elements***, or more simply ***pixels***. The pixels surrounding by other pixels creates its ***neighbourhood***.
- A neighbourhood can be characterized by its shape in the same way as a matrix: we can speak of a **3 x 3** neighbourhood, or of a **5 x 7** neighbourhood.
- Neighbourhoods have odd numbers of rows and columns; this ensures that the current pixel is in the centre of the neighbourhood.

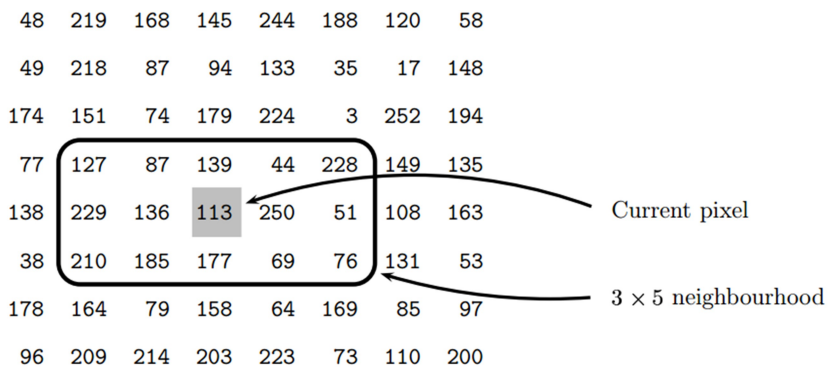



Figure 1.7: Pixels, with a neighbourhood

**Types of digital images**

- **Binary:**
  - Each pixel is just black or white.
  - Since there are only two possible values for each pixel, we only need one bit per pixel.
  - Such images can therefore be very efficient in terms of storage.
  - Images for which a binary representation may be suitable include text (printed or handwriting), fingerprints, or architectural plans.
  - White for the edges, and black for the background in given image.

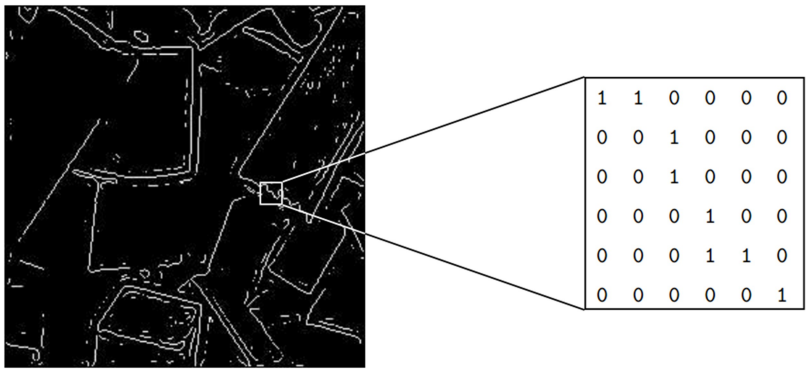



Figure 1.8: A binary image

- **Greyscale:**
  - Each pixel is a shades of grey, normally from 0 (black) to 255 (white).
  - This range means that each pixel can be represented by eight bits, or exactly one byte.
  - This is a very natural range for image file handling.
  - Such images arise in medicine (X-rays), images of printed works, and indeed 256 different grey levels is sufficient for the recognition of most natural objects.

$255 = 2^8$

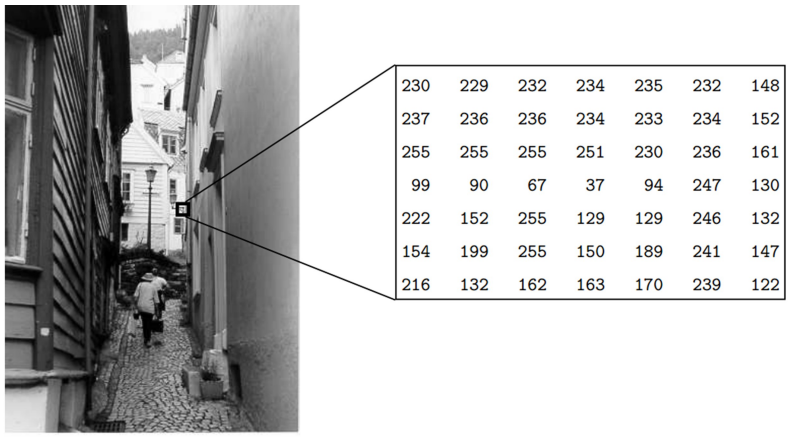



Figure 1.9: A greyscale image

- **True colour, or RGB:**
  - Here each pixel has a particular colour; that colour being described by the amount of red, green and blue in it.
  - If each of these components has a range 0–255, this give a total of $255^3 = 16,777,216$ different possible colors in the image. This is enough colors for any image.
  - Since the total number of bits required for each pixel is 24, such images are also called 24-*bit colour images*.
  - Such an image may be considered as consisting of a “stack” of three matrices; representing the red, green and blue values for each pixel.
  - This means that for every pixel there correspond three values.

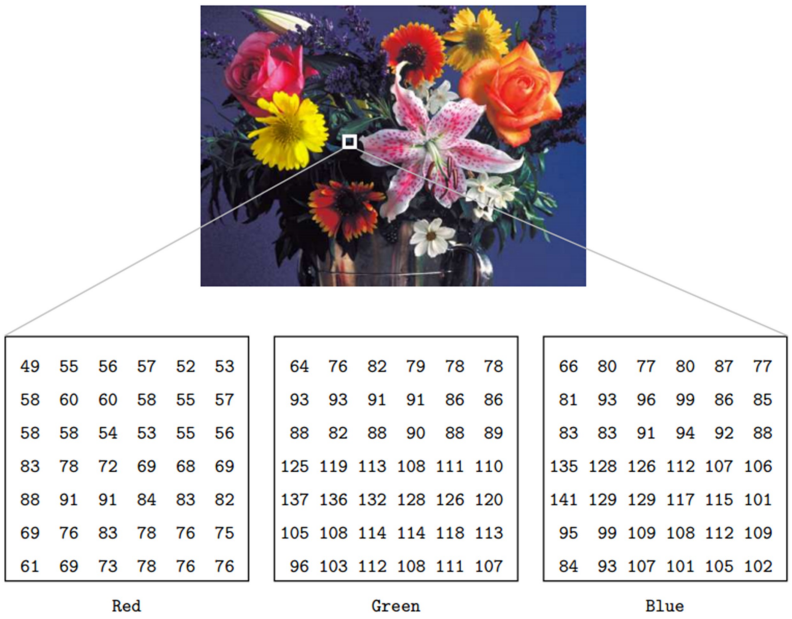
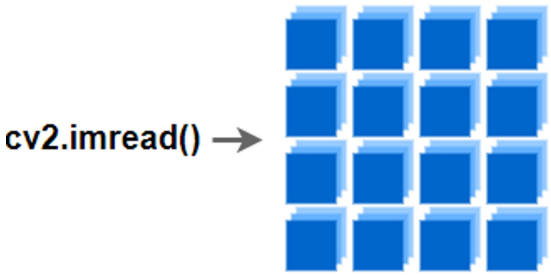



Figure 1.10: A true colour image

- **OpenCV imread() Function:**
  - imread() function of OpenCV library used to read an image in arrays, and numpy library is used to handle the arrays.
  - The array contains pixel level data. And as per the requirement, you may modify the data of the image at a pixel level by updating the array values.
  - imread() returns a 2D or 3D matrix based on the number of color channels present in the image. For a binary or grey scale image, 2D array is sufficient. But for a colored image, you need 3D array.




  - **Syntax**

```
cv2.imread(path, flag)
```

**Optional**, but some time use to convert colored image into gray during reading the image

For that you have to set flag **cv2.IMREAD_GRAYSCALE**

```
"C:\\Users\\hp\\Google Drive\\Fiverr Work
\\2022\\15. Teaching OpenCV to Client
\\pictures"
```

**Reading binary, grayscale and color image**

```
import cv2 as cv
```

Path of directory where images reside

```
path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\pictures"

img = cv.imread(path + "\\cameraman.tif")

print(img)
```

Name of image

**Now read colored image and convert it to grayscale during reading with imread function**

```
import cv2 as cv

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\pictures"

img = cv.imread(path + "\\zia.png", cv2.IMREAD_GRAYSCALE)

print(img)
```

This is colored image but readied as grayscale due to

- **OpenCV imshow() and waitkey() Function:**
  - Display an image in an OpenCV window
  - Syntax

```
cv.imshow("image", img)
```

  - The first argument is the title of the window and the second argument is the object or image that will be shown.
  - If you read an image using imread() function and then display using imshow(), and run the program the window appears for a second and disappear.
  - So, in order to display image along with imshow() function, you have to use OpenCV waitkey() function.

  - **waitkey() Function:**
    - Function whose only parameter is just how long should it wait for a user input (measured in milliseconds). Zero means to wait forever. The return value is the key that was pressed.
    - Syntax

```
cv.waitKey()
```

  - Complete Code to display an image

```
import cv2 as cv

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\pictures"

img = cv.imread(path + "\\zia.png")

cv.imshow("image", img)
cv.waitKey()

cv.destroyAllWindows()
```

- **cv2.destroyAllWindows()**
  - simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argument.

- **OpenCV resize() Function:**

  - In order to resize an image cv2.resize(source, (width x height)) is used.

```
img = cv.imread(path + "\\eyes.jpg")
imgResized = cv.resize(img, (1200, 700))

cv.imshow("image", img)
cv.imshow("Resized Image", imgResized)
cv.waitKey()

cv.destroyAllWindows()
```

- **OpenCV cvtColor() Function:**

  - In order to resize an image cv2.cvtColor(source, cv2.COLOR_CODE) is used.
  - cv2.COLOR_CODE have specific code which is integer value you can give that code instead of writing this `cv.COLOR_BGR2GRAY`.
  - `Print(cv.COLOR_BGR2GRAY) to see the code of this.`

```
img = cv.imread(path + "\\eyes.jpg")
imgResized = cv.resize(img, (1200, 700))

imgGrayScale = cv.cvtColor(imgResized, cv.COLOR_BGR2GRAY)

cv.imshow("image", img)
cv.imshow("Grayscale Image", imgGrayScale)
cv.waitKey()

cv.destroyAllWindows()
```

  - Example of changing the colors

```
import cv2 as cv

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching
OpenCV to Client\\pictures"

img = cv.imread(path + "\\eyes.jpg")
imgResized = cv.resize(img, (1200, 700))

# each color code has integer value
colorName = [cv.COLOR_BGR2RGB, cv.COLOR_BGR2RGBA, cv.COLOR_BGR2BGRA,
cv.COLOR_BGR2GRAY,
             cv.COLOR_BGR2HSV,
cv.COLOR_BGR2HLS_FULL,cv.COLOR_BGR2HSV_FULL,
             cv.COLOR_BGR2YCrCb, cv.COLOR_BGR2LAB, cv.COLOR_BGR2XYZ,
             cv.COLOR_BGR2LUV]

for colorNumber in colorName:

    colorImage = cv.cvtColor(imgResized, colorNumber)
```

```
cv.imshow(f"Image of color number {colorNumber}", colorImage)
cv.waitKey(2000)

cv.destroyAllWindows()
```

- **OpenCV shape() Function:**
  - The dimensions of a given image like the height of the image, width of the image and number of channels in the image are called the shape of the image.
  - In order to find the shape of a given image, we make use of a function in OpenCV called shape() function.
  - The shape() function can provide the dimension of a given image.
  - The shape() function stores each of the dimension of the image like the height of the image, width of the image and number of channels in the image at different indices.
  - The height of the image is stored at the index 0.
  - The width of the image is stored at index 1.
  - The number of channels in the image is stored at index 2.

```
img = cv.imread(path + "\\zia.png")

dimension = img.shape

height = dimension[0]
width = dimension[1]
channel = dimension[2]

print(f"The dimension of given image is {dimension}")
print(f"The height of given image is {height}")
print(f"The width of given image is {width}")
print(f"The channel of given image is {channel}")

cv.imshow("image", img)
cv.waitKey()
```

  - **len() Function:**
    - Only return height of an image len(imageSource)
  - **Now accessing the channels of colored image separately**

```
import cv2 as cv
Import numpy as cv

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching
OpenCV to Client\\pictures"

img = cv.imread(path + "\\eyes.jpg")
imgResized = cv.resize(img, (400, 400))

dimension = imgResized.shape
print(f"\nDimension of image {dimension}\n")

# Blue channel 0 represents blue, 1 for green, and 2 for red channel
b = imgResized[:,:,0]
print(f"{b} \n\n Dimension of blue Channel is: {b.shape}")
```

```
g = imgResized[:,:,1]
print(f"{g} \n\n Dimension of green Channel is: {g.shape}")
r = imgResized[:,:,2]
print(f"{r} \n\n Dimension of red Channel is: {r.shape}")

# we can do same thing by using split function of OpenCV
# but this function is costly in terms of time so that's why above
methods preferred to split
blue, green, red = cv.split(imgResized)
print(f"{blue} \n {green} \n {red}")

cv.imshow("image", img)
cv.imshow("Resized Image", imgResized)
cv.imshow("blue channel image", b)
cv.imshow("green channel image", g)
cv.imshow("red channel image", r)

# you can also merge these channel to make original image
mergedImage = cv.merge([b, g, r])
cv.imshow("Merged Image", mergedImage)


# for accessing actual blue, green, and red you have to merge blue and
with zero matrices of same size
# as green and red
# create an array of same length as blue or green contain with data type
mentioned also

zeroChannel = np.zeros((dimension[0], dimension[1]), "uint8")

blueImage = cv.merge([b, zeroChannel, zeroChannel])
greenImage = cv.merge([zeroChannel, g, zeroChannel])
redImage = cv.merge([zeroChannel, zeroChannel, r])

cv.imshow("actual blue channel", blueImage)
cv.imshow("actual green channel", greenImage)
cv.imshow("actual red channel", redImage)



cv.waitKey()
cv.destroyAllWindows()
```

- **OpenCV namedWindow() Function:**
  - is used to create a window with a suitable name and size to display images and videos on the screen.

    ***Syntax:*** cv2.namedWindow(window_name, flag)

  - ***window_name:*** *Name of the window that will display image/video*
  - ***flag:*** *Represents if window size is automatically set or adjustable.*
    ***Some of the flag values are:***
  - ***WINDOW_NORMAL –*** *Allows to manually change window size*
  - ***WINDOW_AUTOSIZE(Default) –*** *Automatically sets the window size*
  - ***WINDOW_FULLSCREEN –*** *Changes the window size to fullscreen*