Computer Vision: Open CV

Manual Template

Author: Layth Ayache

Contents

[1. Introduction 2](#_Toc200207996)

[2. OpenCV Installation 2](#_Toc200207997)

[2.1 Environment Setup 2](#_Toc200207998)

[2.2 Installation Steps 3](#_Toc200207999)

[2.3 Verification 3](#_Toc200208000)

[3. Getting Started with Images 4](#_Toc200208001)

[3.1 Loading & Display 4](#_Toc200208002)

[3.2 Image Properties 6](#_Toc200208003)

[4. Basic Image Manipulation 9](#_Toc200208004)

[4.1 Color Channels 9](#_Toc200208005)

[4.2 Color Conversions 10](#_Toc200208006)

[Conclusion 11](#_Toc200208007)

# 1. Introduction

The objective of this report is to showcase the full extent of the OpenCV library, and to guide any person that reads this report on how to use it and do the following :

1. OpenCV Installation
2. Getting Started with Images
3. Basic Image Manipulation
4. Image Annotation
5. Image Enhancement
6. Accessing the Camera
7. Video Writing

The content of this report were directly taken from the OpenCV bootcamp course :” <https://courses.opencv.org/courses/course-v1:OpenCV+Bootcamp+CV0/course/> “

# 2. OpenCV Installation

## 2.1 Environment Setup

A **Python environment** is an isolated workspace containing its own Python interpreter and libraries, so different projects don’t conflict. **Anaconda** is a popular distribution that includes the Conda package manager plus many scientific libraries, making it easy to create and manage these environments.

**Alternatives to Anaconda**

* Python’s built-in venv module
* virtualenv (stand-alone virtual environments)
* pyenv (manages multiple Python versions & environments)

the following will be the steps for anaconda in windowns:

PS: these steps differ depending on the OS being used (MAC OS is different than Linux and is different than Windows)

First step is to download the Anaconda installer and run it:

**Anaconda** or **Miniconda** installed (download from <https://docs.conda.io/>)

Then you want to create a Conda environment:  
**Open** your terminal (macOS/Linux) or Anaconda Prompt (Windows):

And type:   
conda create -n vision python=3.9 < - -this will create an environment called vision using python 3.9

Then type:  
conda activate vision < - - this will activate the environment

## 2.2 Installation Steps

The next step of our process is actually downloading the OpenCV library in the vision environment:  
conda install -c conda-forge opencv < - - this will install the latest installation of OpenCV

And believe it or not, this is enough to download opencv!!!

## 2.3 Verification

Now to verify that openCv was correctly downloaded  
open a python file and try to import cv2,   
if it works then OpenCV has been correctly downloaded on your environment

# 3. Getting Started with Images

## 3.1 Loading & Display

First we need to import the libraries, then we need to load and display the images,

In python:

*# Display any image.*

Image(filename**=**"NameofImage.formatofimage")

Now lets Use OpenCV:  
#display any image using cv2

cv2.imread( filename[, flags] )

you can add a variable

retval = cv2.imread( filename[, flags] )

retval: Is the image if it is successfully loaded. Otherwise it is None. This may happen if the filename is wrong or the file is corrupt.

The function has **1 required input argument** and one optional flag:

1. filename: This can be an **absolute** or **relative** path. This is a **mandatory argument**.
2. Flags: These flags are used to read an image in a particular format (for example, grayscale/color/with alpha channel). This is an **optional argument** with a default value of cv2.IMREAD\_COLOR or 1 which loads the image as a color image.

Before we proceed with some examples, let's also have a look at some of the flags available.

**Flags**

1. **cv2.IMREAD\_GRAYSCALE** or **0**: Loads image in grayscale mode
2. **cv2.IMREAD\_COLOR** or **1**: Loads a color image. Any transparency of image will be neglected. It is the default flag.
3. **cv2.IMREAD\_UNCHANGED** or **-1**: Loads image as such including alpha channel.

Example:  
 this is the picture we are using for this example, named “checkerboard\_18x18.png”

# Read image as gray scale.

cb\_img = cv2.imread("checkerboard\_18x18.png", 0) < - - - the 0 flag means gray scale

# Print the image data (pixel values), element of a 2D numpy array.

# Each pixel value is 8-bits [0,255] < - - - since it is in gray scale

print(cb\_img)

the output will be :  
[[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 255 255 255]

[255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 255 255 255]

[255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 255 255 255]

[255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 255 255 255]

[255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 255 255 255]

[255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 255 255 255]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]

[ 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0]]

## 3.2 Image Properties

Now after we learned how to load the image, lets study it’s properties

# print the size  of image  
print("Image size (H, W) is:", cb\_img.shape)

🡪Image size (H, W) is: (18, 18)

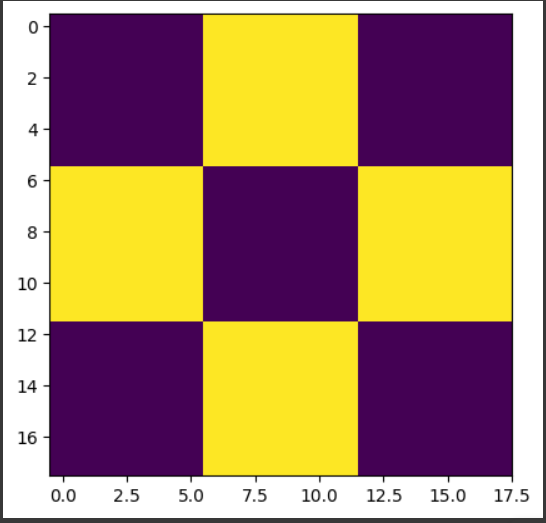
# print data-type of image  
print("Data type of image is:", cb\_img.dtype)

🡪Data type of image is: uint8

Display Images using Matplotlib

# Display image.

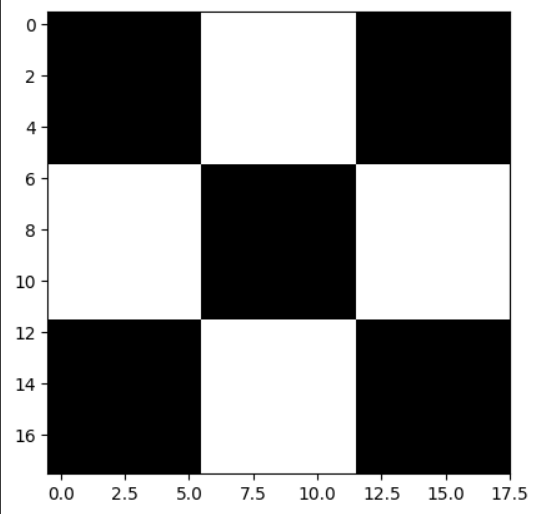
plt.imshow(cb\_img)



Even though the image was read in as a gray scale image, it won't necessarily display in gray scale when using imshow(). matplotlib uses different color maps and it's possible that the gray scale color map is not set.

# Set color map to gray scale for proper rendering.

plt.imshow(cb\_img, cmap="gray")



Until now, we have been using gray scale images in our discussion. Let us now discuss color images:  
# Read and display Coca-Cola logo.

Image("coca-cola-logo.png")



Let us read a color image and check the parameters. Note the image dimension.  
# Read in image

coke\_img = cv2.imread("coca-cola-logo.png", 1)

# print the size  of image

print("Image size (H, W, C) is:", coke\_img.shape)

# print data-type of image

print("Data type of image is:", coke\_img.dtype)

🡪Image size (H, W, C) is: (700, 700, 3)

🡪Data type of image is: uint8

PS: note that opencv expect the numbers as BGR and matplotlib expects them in RGB

# 4. Basic Image Manipulation

## 4.1 Color Channels

* **cv2.split()** Divides a multi-channel array into several single-channel arrays.
* **cv2.merge()** Merges several arrays to make a single multi-channel array. All the input matrices must have the same size.

# Split the image into the B,G,R components

img\_NZ\_bgr = cv2.imread("New\_Zealand\_Lake.jpg", cv2.IMREAD\_COLOR)

b, g, r = cv2.split(img\_NZ\_bgr)

# Show the channels

plt.figure(figsize=[20, 5])

plt.subplot(141);plt.imshow(r, cmap="gray");plt.title("Red Channel")

plt.subplot(142);plt.imshow(g, cmap="gray");plt.title("Green Channel")

plt.subplot(143);plt.imshow(b, cmap="gray");plt.title("Blue Channel")

# Merge the individual channels into a BGR image

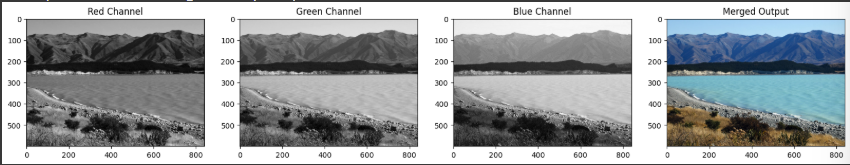
imgMerged = cv2.merge((b, g, r))

# Show the merged output

plt.subplot(144)

plt.imshow(imgMerged[:, :, ::-1])

plt.title("Merged Output")



## 4.2 Color Conversions

**cv2.cvtColor()** Converts an image from one color space to another. The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

dst = cv2.cvtColor( src, code )

dst: Is the output image of the same size and depth as src.

The function has **2 required arguments**:

1. src input image: 8-bit unsigned, 16-bit unsigned ( CV\_16UC... ), or single-precision floating-point.
2. code color space conversion code (see ColorConversionCodes).

# OpenCV stores color channels in a differnet order than most other applications (BGR vs RGB).

img\_NZ\_rgb = cv2.cvtColor(img\_NZ\_bgr, cv2.COLOR\_BGR2RGB)

plt.imshow(img\_NZ\_rgb)

# Image manipulation

## Accessing Individual Pixel

Let us see how to access a pixel in the image.

For accessing any pixel in a numpy matrix, you have to use matrix notation such as matrix[r,c], where the r is the row number and c is the column number. Also note that the matrix is 0-indexed.

For example, if you want to access the first pixel, you need to specify matrix[0,0]. Let us see with some examples. We will print one black pixel from top-left and one white pixel from top-center.

# print the first pixel of the first black box  
print(cb\_img[0, 0])

# print the first white pixel to the right of the first black box  
print(cb\_img[0, 6])

## Modifying Image Pixels

# Conclusion

Throughout this manual, we’ve walked through the core capabilities of OpenCV—from setting up a clean Python environment and verifying your installation, to loading and displaying images, manipulating pixels and color spaces, annotating and enhancing imagery, and capturing live video streams. You’ve seen how simple it is to perform resizing, cropping, drawing, filtering, and color conversion with just a few lines of code, and how to build a robust capture loop for both webcam input and video file output.

By mastering these foundational techniques, you now have the tools to preprocess data, implement custom computer-vision pipelines, and prototype real-time applications. As you move forward, consider exploring more advanced topics such as feature detection (e.g., SIFT, ORB), object tracking, camera calibration and 3D reconstruction, deep-learning integration with DNN modules, and GPU acceleration for high-performance workloads.

OpenCV’s rich ecosystem and active community mean that you can continually extend your solutions—whether you’re building interactive demos, automated inspection systems, or full-scale AI pipelines. Keep experimenting with new modules and sample code from the official OpenCV documentation and courses, and leverage what you’ve learned here to tackle increasingly complex vision challenges.