Introduction to OpenCV

Rishab Shah

29 October 2017

1 Introduction

OpenCV is one of the most popular libraries used to develop Computer Vision applications. It enables us to run many different Computer Vision algorithms in real time. It has been around for many years, and it has become the standard library in this field. One of the main advantages of OpenCV is that it is highly optimized and available on almost all the platforms.

There are more than 2500 algorithms that are optimized for state-of-the-art computer vision algorithms like detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

2 Images

2.1 What constitutes an RGB image digitally?

Digitally, a RGB image is a discretized representation of a continuous scene in the visual spectrum that is represented in the form of a collection or a matrix of pixels.

In image representation one is concerned with characterization of the quantity that each picture-element (also called pixel or pet) represents. An image could represent luminances of objects in a scene (such as pictures taken by ordinary camera), the absorption characteristics of the body tissue (X-ray imaging), the radar cross section of a target (radar imaging), the temperature profile of a region (infrared imaging), or the gravitational field in an area (in geophysical imaging). In general, any two-dimensional function that bears information can be considered an image. Image models give a logical or quantitative description of the properties of this function.

Let's say one pixel is part of a 2-bit image. Two measly bits. This pixel can hold only four values. (An 8-bit pixel can hold 256 values, and a 10-bit pixel can hold 1024 values). A pixel needs all three values to make a color. In this respect, we call each color a Channel. Think of a channel as a pipe (or a channel of water). Pipe R transports red water, pipe G transports green water and pipe B transports blue water.

3 OpenCV Primitive Data Types

OpenCV has several primitive data types. These data types are not primitive from the point of view of C, but they are all simple structures, and we will regard them as atomic. You can examine details of the structures described in what follows (as well as other structures) in the *cxtypes.h* header file, which is in the *.../OpenCV/cxcore/include* directory of the OpenCV install.

The simplest of these types is ***CvPoint***. ***CvPoint*** is a simple structure with two integer members, x and y. ***CvPoint*** has two siblings: ***CvPoint2D32f*** and ***CvPoint3D32f***. The former has the same two members x and y, which are both floating-point numbers. The latter also contains a third element, z.

CvSize is more like a cousin to ***CvPoint***. Its members are width and height, which are both integers. If you want floating-point numbers, use ***Cv-Size***'s cousin ***CvSize2D32f***.

CvRect is another child of ***CvPoint*** and ***CvSize***; it contains four members: x, y, *width*, and *height*.

Last but not least is ***CvScalar***, which is a set of four double-precision numbers. When memory is not an issue, ***CvScalar*** is often used to represent one, two, or three real numbers (in these cases, the unneeded components are simply ignored). ***CvScalar*** has a single member val, which is a pointer to an array containing the four double-precision floating-point numbers.

All of these data types have constructor methods with names like ***cvSize()*** (generally* the constructor has the same name as the structure type but with the first character not capitalized). Remember that this is C and not C++, so these "constructors" are just inline functions that take a list of arguments and return the desired structure with the values set appropriately.

cvScalar() is a special case: it has three constructors. The first, called ***cvScalar()***, takes one, two, three, or four arguments and assigns those arguments to the corresponding elements of val[ ]. The second constructor is ***cvRealScalar()***; it takes one argument, which it assigns to val[0] while setting the other entries to 0. The final variant is ***cvScalarAll()***, which takes a single argument but sets all four elements of val[ ] to that same argument.

4 cvMat Matrix Structure

There is no "vector" construct in OpenCV. Whenever we want a vector, we just use a matrix with one column (or one row, if we want a transpose or conjugate vector).

For example, the routine that creates a new two-dimensional matrix has the following prototype:

cvMat cvCreateMat (int rows, int cols, int type);*

Here type can be any of a long list of predefined types of the form:

CV_ <bit_depth>(S|U|F) C<number_of_channels>.

Thus, the matrix could consist of 32-bit fl oats (CV_32FC1), of unsigned integer 8-bit triplets (CV_8UC3), or of countless other elements. An element of a ***CvMat*** is not necessarily a single number. Being able to represent multiple values for a single entry in the matrix allows us to do things like represent multiple color channels in an RGB image. For a simple image containing red, green and blue channels, most image operators will be applied to each channel separately (unless otherwise noted).

4.1 *Exercise 1*

1. How does a program read the cvMat object, in particular, what is the order of the pixel structure?

5 Colorspaces

There are five major models of colorspaces that sub-divide into others, which are:
CIE, RGB, YUV, HSL/HSV, and CMYK.

- The three coordinates of **CIELAB** represent the lightness of the color (L* = 0 yields black and L* = 100 indicates diffuse white; specular white may be higher), its position between red/magenta and green (a*, negative values indicate green while positive values indicate magenta) and its position between yellow and blue (b*, negative values indicate blue and positive values indicate yellow).

- **RGB** (Red, Green, Blue) describes what kind of light needs to be emitted to produce a given color. Light is added together to create form from darkness. **RGB** stores individual values for red, green and blue. **RGB** is not a color space, it is a color model. There are many different **RGB** color spaces derived from this color model, some of which appear below. An **RGB** color space is any additive color space based on the **RGB** color model.
- **YIQ** was formerly used in NTSC (North America, Japan and elsewhere) television broadcasts for historical reasons. This system stores a luma value with two chroma or chrominance values, corresponding approximately to the amounts of blue and red in the color. It corresponds closely to the **YUV** scheme used in PAL (Australia, Europe, except France, which uses SECAM) television except that the **YIQ** color space is rotated 33° with respect to the **YUV** color space. **YPbPr** is a scaled version of **YUV**. It is most commonly seen in its digital form, **YCbCr**, used widely in video and image compression schemes such as MPEG and JPEG.
- **HSV** and **HSL** are transformations of a Cartesian RGB colorspace (usually sRGB), and their components and colorimetry are relative to the **RGB** colorspace from which they are derived. **HSV** (hue, saturation, value), is often used by artists because it is often more natural to think about a color in terms of hue and saturation than in terms of additive or subtractive color components.
- **CMYK** is used in the printing process, because it describes what kind of inks need to be applied so the light reflected from the substrate and through the inks produces a given color.

5.1 ***Exercise 2***

1. *ColorImage.cpp* is a program that takes a look at colorspace conversions in OpenCV. Run the code in *ColorImage.cpp* and comment on the outputs. Implement the same thing in Python and save each image.
2. Print out the values of the pixel at (20,25) in the RGB, YCbCr and HSV versions of the image. What are the ranges of pixel values in each channel of each of the above mentioned colorspaces?

6 Smoothing

Smoothing, also called blurring, is a simple and frequently used image processing operation. There are many reasons for smoothing, but it is usually done to reduce noise or camera artifacts. Smoothing is also important when we wish to reduce the resolution of an image in a principled way. As in one-dimensional signals, images also can be filtered with various low-pass filters(LPF), high-pass filters(HPF) etc. LPF helps in removing noises, blurring the images etc. HPF

filters helps in finding edges in the images. Some of the different kinds of noise present in an image are:
Gaussian noise, Salt-and-pepper noise, Poisson noise and Speckle noise.
Some of the common smoothing/ blurring filters are:

- **Box Filter** - The simple box blur operation, as exemplified by CV_BLUR, is the simplest case. Each pixel in the output is the simple mean of all of the pixels in a window around the corresponding pixel in the input. Simple blur supports 1–4 image channels and works on 8-bit images or 32-bit floating-point images.
- **Gaussian Blurring** - The next smoothing filter, the Gaussian filter, is probably the most useful though not the fastest. Gaussian filtering is done by convolving each point in the input array with a Gaussian kernel and then summing to produce the output array.
- **Median Filter** - The median filter replaces each pixel by the median or "middle" pixel (as opposed to the mean pixel) value in a square neighborhood around the center pixel. Median filter will work on single-channel or three-channel or four-channel 8-bit images, but it cannot be done in place. Simple blurring by averaging can be sensitive to noisy images, especially images with large isolated outlier points (sometimes called "shot noise"). Large differences in even a small number of points can cause a noticeable movement in the average value. Median filtering is able to ignore the outliers by selecting the middle points.

6.1 ***Exercise 3***

1. Look at the code in *Noise.cpp* and implement the code in Python. Also, print the results for different noise values in the Gaussian case, mean = 0, 5, 10, 20 and sigma = 0, 20, 50, 100 and for the salt-and-pepper case, pa = 0.01, 0.03, 0.05, 0.4 and pb = 0.01, 0.03, 0.05, 0.4.
2. Change the kernel sizes for all the filters with all different values for noises and print the results for 3x3, 5x5 and 7x7 kernels. Comment on the results. Which filter seems to work "better" for images with salt-and-pepper noise and gaussian noise?

7 Thresholding

Frequently we have done many layers of processing steps and want either to make a final decision about the pixels in an image or to categorically reject those pixels below or above some value while keeping the others. The OpenCV function ***cvThreshold()*** accomplishes these tasks . The basic idea is that an array is given, along with a threshold, and then something happens to every element of the array depending on whether it is below or above the threshold.

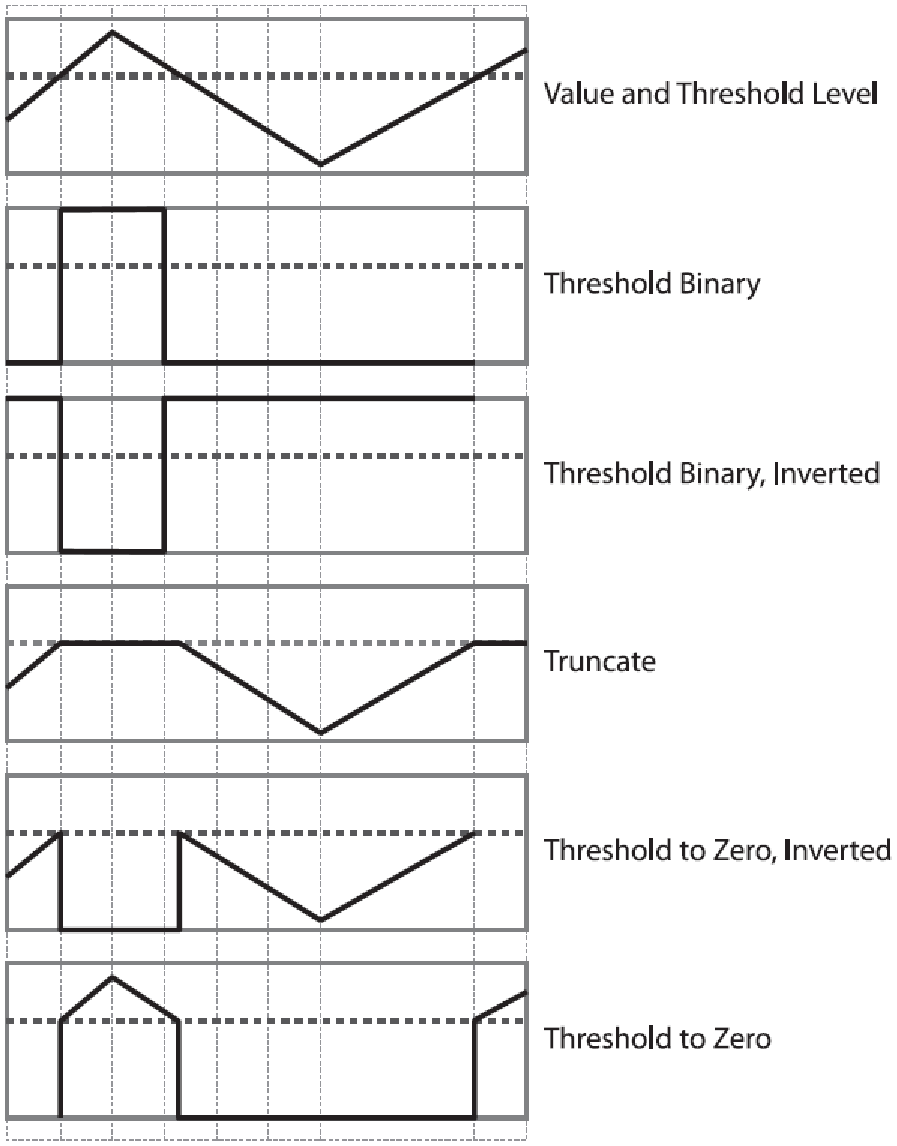

Figure 1: Results of varying the threshold type in cvThreshold(). Th e horizontal line through each chart represents a particular threshold level applied to the top chart and its effect for each of the five types of threshold operations below.

cvAdaptiveThreshold() allows for two different adaptive threshold types depending on the settings of adaptive_method. In both cases the adaptive threshold T(x, y) is set on a pixel-by-pixel basis by computing a weighted average of the b-by-b region around each pixel location minus a constant, where b is given by block_size and the constant is given by param1. For more details, look at the OpenCV documentation.

7.1 *Exercise 4*

1. Look at *Threshold.cpp* and implement the code in Python, and observe the results for different threshold values. Comment on the results.
2. What are the disadvantages of binary threshold?
3. When is Adaptive Threshold useful?

8 Advanced Methods

8.1 Template Matching

Refer to Jinyuan's PDF.

9 Object tracking

Simply put, locating an object in successive frames of a video is called tracking.

The definition sounds straight forward but in computer vision and machine learning, tracking is a very broad term that encompasses conceptually similar but technically different ideas. For example, all the following different but related ideas are generally studied under Object Tracking:

- ***Dense Optical flow***: These algorithms help estimate the motion vector of every pixel in a video frame. Sparse optical flow: These algorithms, like the Kanade-Lucas-Tomashi (KLT) feature tracker, track the location of a few feature points in an image.
- ***Kalman Filtering***: A very popular signal processing algorithm used to predict the location of a moving object based on prior motion information. One of the early applications of this algorithm was missile guidance!
- ***Meanshift and Camshift***: These are algorithms for locating the maxima of a density function. They are also used for tracking.
- ***Single object trackers***: In this class of trackers, the first frame is marked using a rectangle to indicate the location of the object we want to track. The object is then tracked in subsequent frames using the tracking algorithm. In most real life applications, these trackers are used in conjunction with an object detector.

- ***Multiple object track finding algorithms***: In cases when we have a fast object detector, it makes sense to detect multiple objects in each frame and then run a track finding algorithm that identifies which rectangle in one frame corresponds to a rectangle in the next frame.

9.1 *Exercise 5*

1. Run from the OpenCV python tutorials, the Face detection example, in the tutorials page. Then answer the following question.
2. Object detection vs. Tracking: After looking at OpenCV face detection, you know that it works in real time and you can easily detect the face in every frame. So, why do you need tracking in the first place?
3. Comment on the object tracking example shown in class (Hint: Look at the code and comment on the results.)

10 Appendix

- Install OpenCV by following the instructions on the website or install in your favorite environment (e.g., Anaconda).
  Go to `https://www.opencv.org`
- To run the C++ code, let's use an online OpenCV IDE (Click here) by Shuqun Zhang from CSI, CUNY (Zhang's website)