

Lecture Notes on OpenCV Functions (these are just a few of the many, many functions):

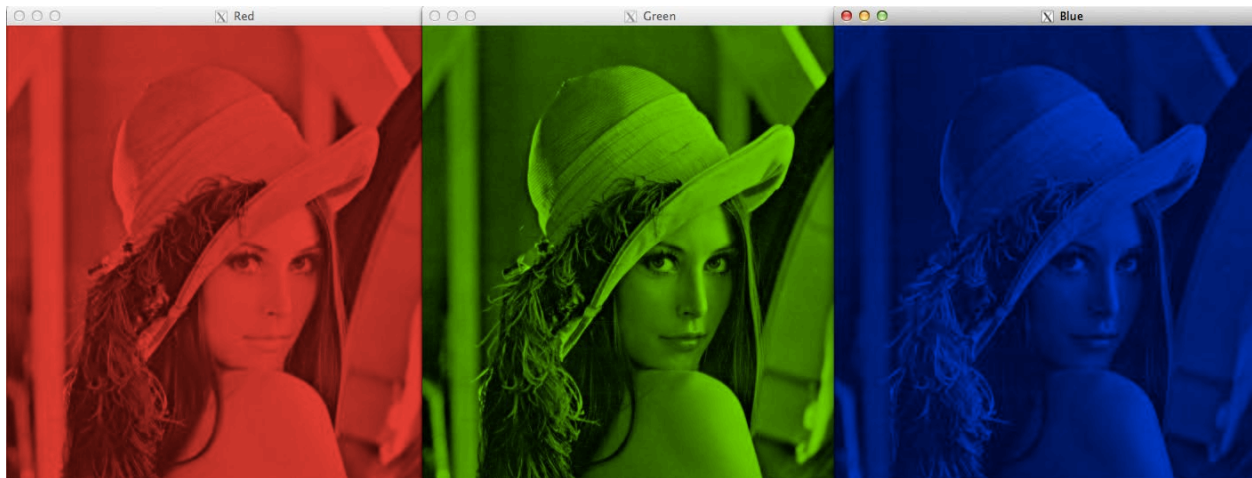
Today I am going to give this document that I have put together. I have found these notes and images from all over the Internet. This RGB picture below is a very famous image. It's been used as example for years on image processing, almost every textbook I've ever looked at has had this image. Recently it was discovered the woman in this image was a prostitute and her client, a computer scientist took the picture and started using it with his research. It spread from there and now the facts are out we have many algorithms and analysis of computer vision thanks to the field of prostitution. That sounds bad, prostitution is bad, but the history of it with computer vision is a little witty.

I am writing this in a hospital room so give me a break, I'm a little off my game.

In this document are several OpenCV functions that are important, many you will use in the next project. Others I'm giving you because they will be needed later when we do a little machine learning on images.

Next I'm going to write some source code, in Python, to demo how each of these work on one of my images.

RGB red green blue values- notice even though it's all blue there are different values of blue, it's kind of like a grayscale image with just the blue channel.



Converting from one format such as RGB to GrayScale

Resizing Images

Machine learning models work with a fixed sized input. The same idea applies to computer vision models as well. The images we use for training our model must be of the same size.

Now this might become problematic if we are creating our own dataset by scraping images from various sources. That's where the function of resizing images comes to the fore.

Images can be easily scaled up and down using OpenCV. This operation is useful for training deep learning models when we need to convert images to the model's input shape. Different interpolation and downsampling methods are supported by OpenCV, which can be used by the following parameters:

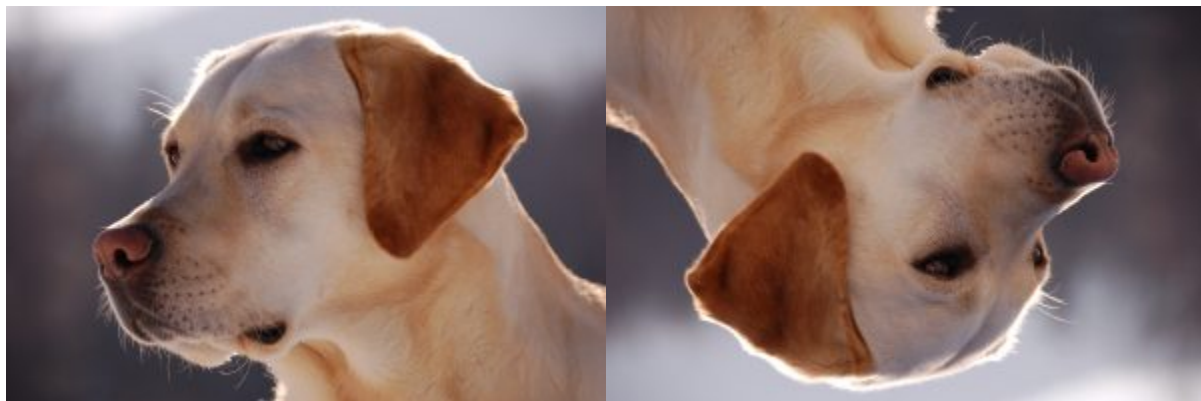
1. **INTER_NEAREST:** Nearest neighbor interpolation
2. **INTER_LINEAR:** Bilinear interpolation
3. **INTER_AREA:** Resampling using pixel area relation
4. **INTER_CUBIC:** Bicubic interpolation over 4×4 pixel neighborhood
5. **INTER_LANCZOS4:** [Lanczos interpolation](#) over 8×8 neighborhood

See source code for example.

Rotating Images

Remember the first program when I had you rotate an image to see how to do it, and what it entails, here is why we do this more than you think.....

Suppose we are building an image classification model for identifying the animal present in an image. So, both the images shown below should be classified as 'dog':



But the model might find it difficult to classify the second image as a Dog if it was not trained on such images. So what should we do?

Let me introduce you to the technique of data augmentation. This method allows us to generate more samples for training our deep learning model. Data augmentation uses the available data samples to produce the new ones, by applying image operations like rotation, scaling, translation, etc. This makes our model robust to changes in input and leads to better generalization.

Rotation is one of the most used and easy to implement data augmentation techniques. As the name suggests, it involves rotating the image at an arbitrary angle and providing it the same label as the original image. Think of the times you have rotated images in your phone to achieve certain angles – that's basically what this function does.

SEE the OPENCV Python Source code I will link for this lecture for an example

Image Translation

Image translation is a geometric transformation that maps the position of every object in the image to a new location in the final output image. After the translation operation, an object present at location (x,y) in the input image is shifted to a new position (X,Y):

$$X = x + dx$$

$$Y = y + dy$$

Here, dx and dy are the respective translations along different dimensions.

Image translation can be used to add shift invariance to the model, as by translation we can change the position of the object in the image give more variety to the model that leads to better generalizability which works in difficult conditions i.e. when the object is not perfectly aligned to the center of the image.

This augmentation technique can also help the model correctly classify images with partially visible objects. Take the below image for example. Even when the complete shoe is not present in the image, the model should be able to classify it as a Shoe.



This translation function is typically used in the image pre-processing stage. Check out the below code to see how it works in a practical scenario:

SEE the OPENCV Python Source code I will link for this lecture for an example

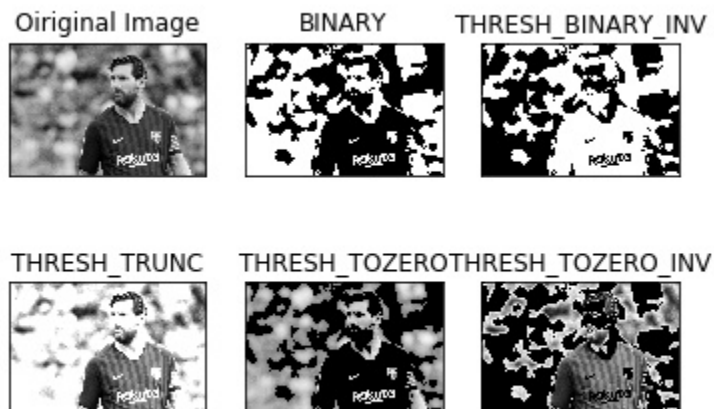
Simple Image Thresholding

[Thresholding](#) is an image **segmentation** method. It compares pixel values with a threshold value and updates it accordingly. OpenCV supports multiple variations of thresholding. A simple thresholding function can be defined like this:

```
if Image(x,y) :  
    threshold , Image(x,y) = 1  
else:  
    Image(x,y) = 0
```

Thresholding can only be applied to grayscale images.

A simple application of image thresholding could be dividing the image into its foreground and background.



Adaptive Thresholding

In case of adaptive thresholding, different threshold values are used for different parts of the image. This function gives better results for images with varying lighting conditions – hence the term “adaptive”.

[Otsu's binarization method](#) finds an optimal threshold value for the whole image. It works well for bimodal images (images with 2 peaks in their histogram).

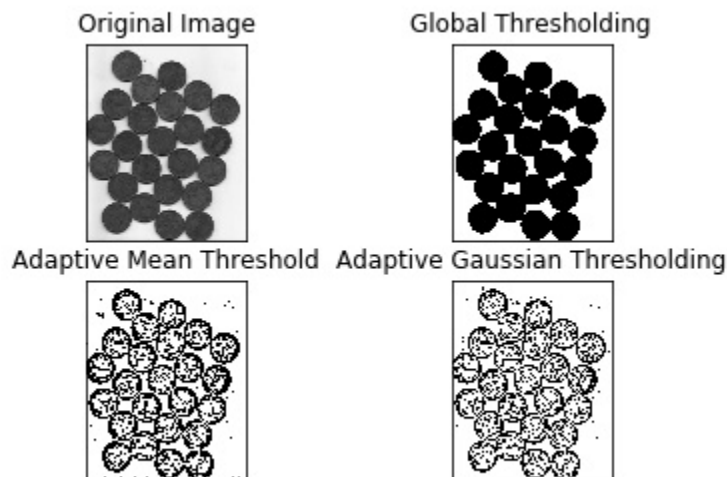


Image Segmentation (Watershed Algorithm)

Image segmentation is the task of classifying every pixel in the image to some class. For example, classifying every pixel as foreground or background. Image segmentation is important for extracting the relevant parts from an image.

The watershed algorithm is a classic image segmentation algorithm. It considers the pixel values in an image as topography. For finding the object boundaries, it takes initial markers as input. The algorithm then starts flooding the basin from the markers till the markers meet at the object boundaries.

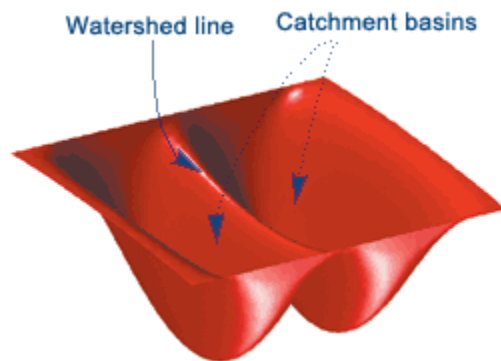


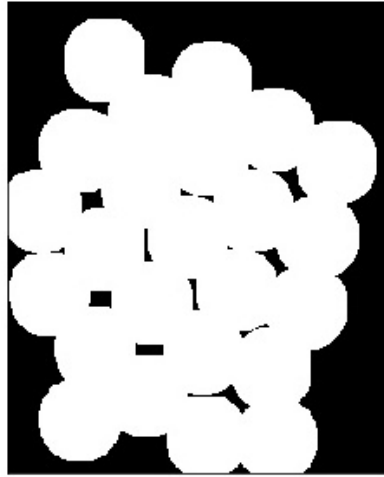
Image Source :- Mathworks

Let's say we have a topography with multiple basins. Now, if we fill different basins with water of different color, then the intersection of different colors will give us the object boundaries. This is the intuition behind the watershed algorithm.

INPUT IMAGE



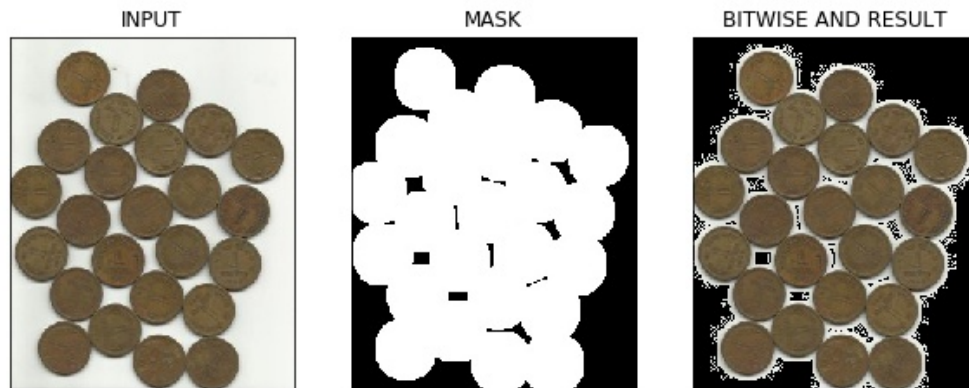
SEGMENTATION MASK



Bitwise Operations

[Bitwise operations](#) include AND, OR, NOT and XOR. You might remember them from your programming class! In computer vision, these operations are very useful when we have a mask image and want to apply that mask over another image to extract the region of interest.

[view rawBitwise Operations.py](#) hosted with [by GitHub](#)



In the above figure, we can see an input image and its segmentation mask calculated using the Watershed algorithm. Further, we have applied the bitwise 'AND' operation to remove the background from the image and extract relevant portions from the image. Pretty awesome stuff!

Edge Detection

Edges are the points in an image where the image brightness changes sharply or has discontinuities. Such discontinuities generally correspond to:

- Discontinuities in depth
- Discontinuities in surface orientation
- Changes in material properties
- Variations in scene illumination

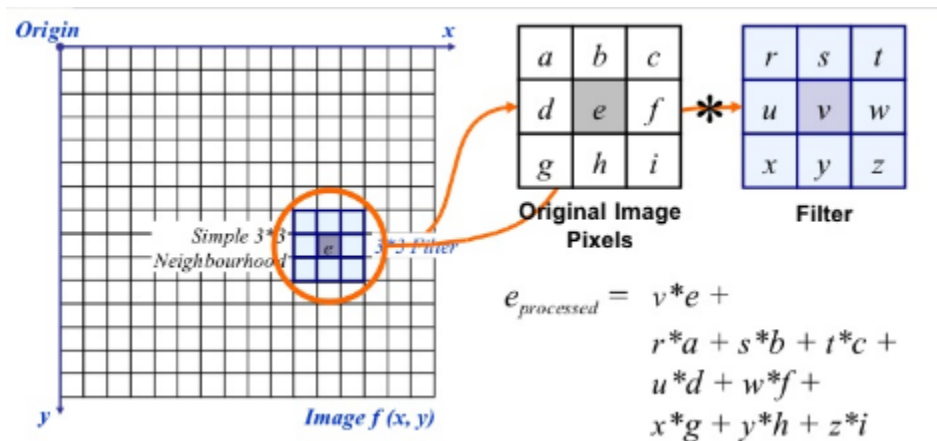
Edges are very useful features of an image that can be used for different applications like classification of objects in the image and localization. Even deep learning models calculate edge features to extract information about the objects present in image.

Edges are different from contours as they are not related to objects rather they signify the changes in pixel values of an image. Edge detection can be used for image segmentation and even for image sharpening.

Image Filtering

In image filtering, a pixel value is updated using its neighboring values. But how are these values updated in the first place?

Well, there are multiple ways of updating pixel values, such as selecting the maximum value from neighbors, using the average of neighbors, etc. Each method has its own uses. For example, averaging the pixel values in a neighborhood is used for image blurring. Another thing you did in the first assignment.



Gaussian filtering is also used for image blurring that gives different weights to the neighboring pixels based on their distance from the pixel under consideration.

For image filtering, we use kernels. Kernels are matrices of numbers of different shapes like 3 x 3, 5 x 5, etc. A kernel is used to calculate the dot product with a part of the image. When calculating the new value of a pixel, the kernel center is overlapped with the pixel. The neighboring pixel values are multiplied with the corresponding values in the kernel. The calculated value is assigned to the pixel coinciding with the center of the kernel.

ORIGINAL



AFTER GAUSSIAN KERNEL



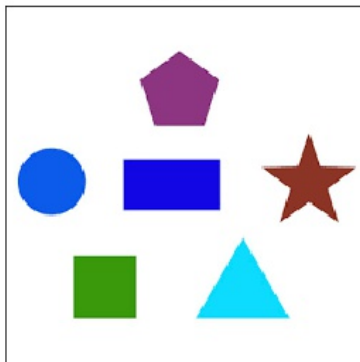
Image Contours

A contour is a closed curve of points or line segments that represents the boundaries of an object in the image. Contours are essentially the shapes of objects in an image.

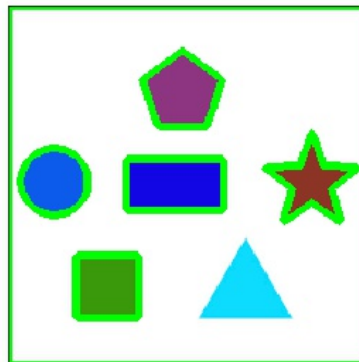
Unlike edges, contours are not part of an image. Instead, they are an abstract collection of points and line segments corresponding to the shapes of the object(s) in the image.

We can use contours to count the number of objects in an image, categorize objects on the basis of their shapes, or select objects of particular shapes from the image.

ORIGINAL



WITH CONTOURS



Scale Invariant Feature Transform (SIFT)

Keypoints is a concept you should be aware of when working with images. These are basically the points of interest in an image. Keypoints are analogous to the features of a given image.

They are locations that define what is interesting in the image. Keypoints are important, because no matter how the image is modified (rotation, shrinking, expanding, distortion), we will always find the same keypoints for the image.

Scale Invariant Feature Transform (SIFT) is a very popular keypoint detection algorithm. It consists of the following steps:

- Scale-space extrema detection
- Keypoint localization
- Orientation assignment
- Keypoint descriptor
- Keypoint matching

Features extracted from SIFT can be used for applications like image stitching, object detection, etc. The below code and output show the keypoints and their orientation calculated using SIFT.



I have many more of these, but I'm going to stop there for today. We will use many of these features on Friday's Project this week.