International Journal of Research in Engineering, Science and Management
Volume-3, Issue-3, March-2020
www.ijresm.com | ISSN (Online): 2581-5792

Image processing using OpenCV and Python

R. Jayashree[1], D. G. Savitha[2], D. Sharanya[3]
[1,2,3]*UG Student, Department of Computer Science and Engineering, GITAM University, Dodballapura, India*

***Abstract*: As humans, we are generally being excellent at finding the difference during a picture. However, for computers, this is often not such a simple task. Computers can only learn from what we train their models on. Some great models will classify batch images rather well, like Google's TensorFlow and Keras.**

Thanks to image classifier libraries. Now we will create complicated models. However, this project is to make a picture classifier that will tell how similar two images are. For that, there's no need for any complicated libraries like TensorFlow or image classification models. There are two ways to seek out if a picture is analogous to a different image. We are printing the values of SSIM, MSE and we are comparing both.

***Keywords*: Image difference, OpenCV, SSIM, Python.**

1. Introduction

There are many metrics out there for evaluating whether two images look like or how much they appear. Generally, the question is said to human's perception of images, so each algorithm has its support on human sensory system traits. Among them, SSIM is the easiest method to calculate and its overhead is additionally small.

The (structural similarity index metric) SSIM is an algorithm which we will be used for predicting the difference between two images. SSIM is mainly used for measuring the similarity between two images. The SSIM index is a full reference metric. i.e., the measurement or prediction of image quality is predicated on an initial uncompressed or distortion-free image as a reference. SSIM is used to enhance traditional methods of mean squared error (MSE) and peak signal to noise (PSNR).

Let's define what is calculating. MSE will calculate the mean square error between each pixel for the 2 images we are comparing. Whereas SSIM will do the other and appearance for similarities within pixels; i.e. if the pixels within the two images line up and or have similar pixel density values. the sole issues are that MSE tends to possess arbitrarily high numbers so it's harder to standardize it. The more the MSE value the images differ from each other, if the MSE value between images differs appears randomly, it'll be difficult for us to inform anything. SSIM on the opposite hand puts everything on a scale of -1 to 1. A score of 1 meant they're very similar and a score of -1 meant they're very different. In my opinion, this is often a far better metric of measurement. We are using CV2 (part of OpenCV) to read and edit images. we will also use matplot lib's imread instead.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

First, we load the pictures that are saved in our directory. Second, we've to form sure they're all an equivalent size as otherwise, we'll get a dimension error. The difficulty with this is often this will distort image so fiddle till you discover the right numbers. Next, we do another function so we will see what our pictures appear as if.

Now to check and see if our MSE and SSIM are functioning by comparing one image to itself. If it works, then we should always get MSE of 0 and SSIM of 1.

2. Existing system

In the existing system, we will take the original image and fake image (modified image). We will pass both images as parameters to the program that will give the SSIM and MES values. SSIM value will be in the range of -1 to 1. This system will give numeric which will tell us the difference between the two images and its similarity.

3. Proposed system

1. Import the specified packages.
2. Read in our two images.
3. Convert the pictures to grayscale.
4. Consistent with SSIM, how similar are the pictures.
5. Threshold the diff image, and find contours which can showcase the regions within the images that are different
6. Loop through the contours and make bounding boxes on our two images.
7. Print our images by highlighting the difference in it.

The process is often broadly classified into following steps,

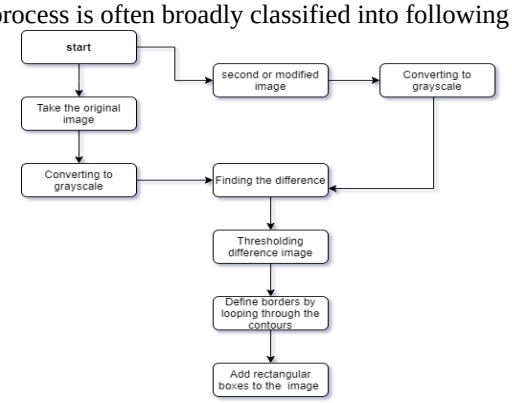

Fig. 1. Flowchart

1) Import the specified packages

Use the function cv2.imread() to read the image. The image should be within the working directory or a full path of image should be given.

img=cv2.imread('path/image_name')img=cv2.imread('path/image_name')

2) Read in our two images

If images are of equivalent shape and dimension, we can load the pictures or If not, we got to resize or crop them. PIL library will help to try it in Python. If they are taken with an equivalent setting and the same device, they are probably similar. Resizing the pictures check out zero patting

img1= cv2.resize(img_1, (10000, 10000))

3) Converting to grayscale

Grayscaling is the process of converting a picture from other color spaces e.g. RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white.

cv2.cvtColor(original,cv2.COLOR_BGR2GRAY)

We have 150 color-space conversion methods in OpenCV. But we only need two which are used most generally, BGR ↔ Gray and BGR ↔ HSV.

We use the function cv2.cvtColor(input_original_image, flags) for colour conversion where flags determines the sort of conversion. For BGR → Gray conversion we use the flags cv2.COLOR_BGR2GRAY.

Importance of grayscaling

- Dimension reduction: e.g. There are three colors in RGB channels and has three dimensions and grayscaled images are a single dimension which will be in black and white format.
- In training neural article on RGB images of 10x10x3 pixel. 300 input nodes will have the input. On the other hand, but grayscaled images need only 100 input nodes in an equivalent neural network
- For other algorithms to figure: Many algorithms are customized to work only on grayscaled images e.g. Canny Edge Detection is a function that is pre-implemented works on Grayscaled images in OpenCV library.

This score would suggest that the 2 pictures are extremely similar.

4) Consistent with SSIM, how similar are the pictures (1 being identical, -1 is completely different)

(score, diff) = compare_ssim(grayA, grayB, full=True)

diff = (diff * 255).astype("uint8")

print("SSIM: {}".format(score))

>> 0.997413297969084

We have used the compare_ssim function to get a score, but also an object called diff. This diff object represents the particular differences within the image. to hold on processing this object using OpenCV we multiply the values by 255 and alter it to uint8 format

The score represents the structural similarity index between the 2 input images. This value can fall into the range [-1, 1] with a worth of 1 being a “perfect match”.

The diff image contains the particular image differences between the 2 input images that we want to see. The difference image will be represented currently as a floating-point data type within the range [0, 1] so we first convert the array to 8-bit unsigned integers within the range [0, 255] before we will further process it

5) Threshold the diff image, and find contours showing the regions in the different images

If the pixel value is higher than the threshold value, some value is assigned (say white), otherwise, another value is assigned (say black).

cv2.threshold is the function used in the program. Grayscale image of the original image, threshold value, maximum Val if the pixel value is greater than the threshold value, then it represents the value are the arguments respectively. OpenCV includes different thresholding types and the fourth parameter of the method decides on this. Different types of thresholding are:

cv2.THRESH_BINARY
cv2.THRESH_BINARY_INV
cv2.THRESH_TRUNC
cv2.THRESH_TOZERO
cv2.THRESH_TOZERO_INV

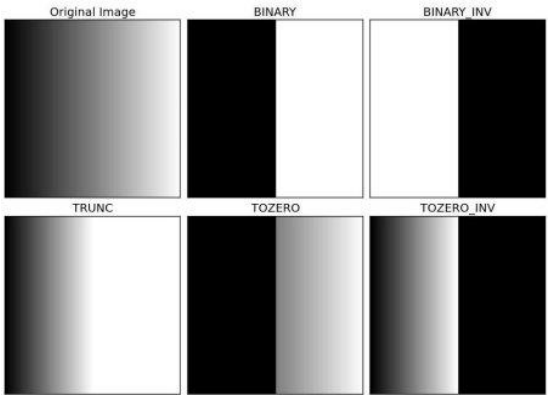

Fig. 2. Different types of thresholding

6) Loop through the contours and create bounding boxes on our two images

To detect contours in a picture, we’ll need to use the cv2.find Contours function.

A curve that connects all the continuous points, along with the boundary, having the same strength and color is called counters. Shape analysis, recognition of objects and object detection for all these contours are a useful tool.

We need to use binary images for better accuracy. So, we will apply threshold or canny edge detection before finding contours.

In OpenCV, finding contours is same as finding white object from black background. So, consider an object to be found should be white and background should be black.

cnts = cv2.findContours(thresh.copy(),
cv2.RETR_EXTERNAL,

cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

The first one is the source image, second is contour retrieval mode, third is contour approximation method and it generates the image, contours, and hierarchy. 'Contours' is a Python collection of all image contours. Numpy array of (a, b) coordinates of boundary points of the object is called individual counter.

The contours are the boundaries of a shape with the same intensity. They store the (a, b) coordinates of the boundary of a shape. But is it holding all the coordinates? This method of approximation to the contour specifies that.

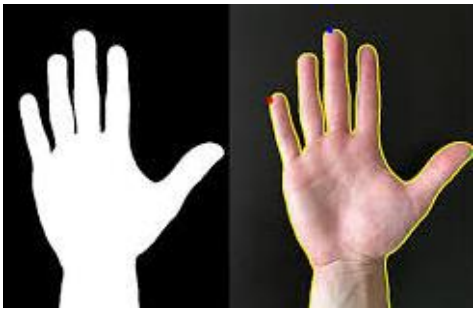

Fig. 3. Contours in OpenCV

Upon passing the third argument,

cv2.CHAIN_APPROX_NONE, all the boundary points are stored. And the need for points here is, for eg., if we have to find the contour of a straight line. There is a need for just two endpoints for that line. cv2.CHAIN_APPROX_SIMPLE in OpenCV will save memory and which compresses the contour and eliminates all redundant points.

for c in cnts:
(a, b, e, d) = cv2.boundingRect(c)
cv2.rectangle(imageA, (a, b), (a + e, b + d), (0, 0, 255), 2)
cv2.rectangle(imageB, (a, b), (a + e, b + d), (0, 0, 255), 2)

we loop over the contours, cnts. First, we will compute the bounding box around the contour by using the cv2.boundingRect function later we store the width/height of the rectangle as e and d as well as specific (a, b)-coordinates as a and b.

Then we use the values to draw a red rectangle with cv2.rectangle on each image.

At last, we will display the original images by showing the difference in it, and we will display the thresholded image and the difference image by showing the difference in it.

We make a call to cv2.waitKey function which makes the program wait until a key is pressed.

7) Show our images - and uncover the differences

we can quickly and easily highlight differences between two images using this script,

cv2.imshow("Original", imageA)

4. Conclusion

In this paper, we have discussed how to compute image differences using OpenCV, Python, and scikit-image's Structural Similarity Index (SSIM). Based on the image difference how to represent the different regions in two images. We addressed in detail about thresholding and contours. We have summarized how we can use digital image processing using OpenCV.

References

[1] https://www.pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python/
[2] https://stackabuse.com/introduction-to-image-processing-in-python-with-opencv/
[3] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html
[4] https://medium.com/datadriveninvestor/image-processing-using-python-open-cv-part-1-b5e83b5c2398
[5] Mahamkali Naveenkumar and Vadivel A. (2015). OpenCV for Computer Vision Applications.
[6] https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html
[7] https://www.youtube.com/watch?v=LNzC4NYYWdg
[8] https://www.youtube.com/watch?v=WOH7hDXrfwc
[9] https://www.youtube.com/watch?v=fUfvBnREBFc
[10] https://www.youtube.com/watch?v=9mQznoHk4mU