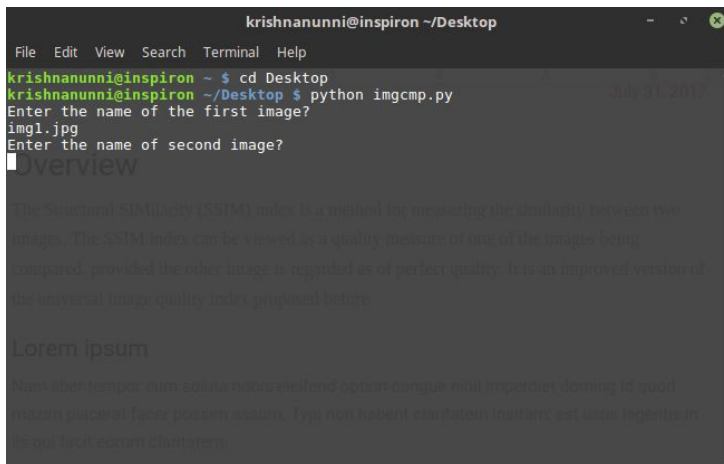


Using Structural Similarity for Image Differentiation

July 31, 2017

Overview



```
krishnanunni@inspiron ~/Desktop
File Edit View Search Terminal Help
krishnanunni@inspiron ~ $ cd Desktop
krishnanunni@inspiron ~/Desktop $ python imgcmp.py
Enter the name of the first image?
img1.jpg
Enter the name of second image?
Overview
The Structural Similarity (SSIM) index is a method for measuring the similarity between two
images. The SSIM index can be viewed as a quality measure of one of the images being
compared, provided the other image is regarded as of perfect quality. It is an improved version of
the universal image quality index proposed before.
Lorem ipsum
Sams albet tempor cum soluta notae eieffend opunt conque nihil imperdiet donec in quid
namque pascunt facis possum statim. Typi non habent euismod invidiam sed utam repente in
re qui facilis eorum claritatem.
```

The Structural SIMilarity (SSIM) index is a method for measuring the similarity between two images. The SSIM index can be viewed as a quality measure of one of the images being compared, provided the other image is regarded as of perfect quality. It is an improved version of the universal image quality index proposed before. This program makes

use of SSIM and contouring to find differences in two images.

Libraries and Modules Used

- matplotlib>=1.3.1
- numpy>=1.11
- scipy>=0.17.0
- pillow>=2.1.0
- imutils>=0.4.3
- Cython>=0.23.4
- OpenCV>=2.4.9

Program Overview

This is an concise description of the working of the code

```
from skimage.measure import compare_ssim  
  
import sys  
  
import imutils  
  
import cv2  
  
import Image
```

The libraries required for this program is imported.

```
def displayarray(data):  
    img = Image.fromarray(data)  
    img.show()
```

A function is defined to create images from a numpy array and display it, using the Image library.

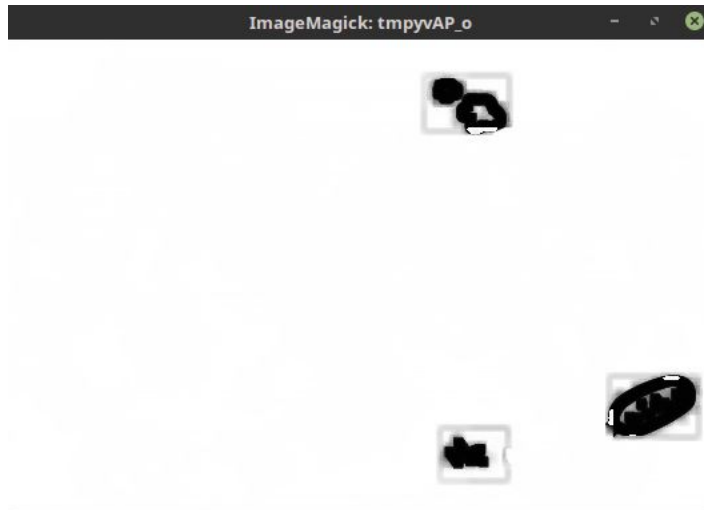
```
img_name_one = raw_input("Enter the name of the first image?\n")  
img_name_two = raw_input("Enter the name of second image?\n")  
imageA = cv2.imread(img_name_one)  
imageB = cv2.imread(img_name_two)
```

The names of the two image files is taken as input from the user, OpenCV function imread loads this into image objects.

```
grayA = cv2.cvtColor(imageA, cv2.COLOR_BGR2GRAY)  
grayB = cv2.cvtColor(imageB, cv2.COLOR_BGR2GRAY)
```

The two image objects are then converted to grayscale using the OpenCV function cvtColor and COLOR_BGR2GRAY. The image is converted to grayscale in order to ignore the difference in brightness and saturation.

```
(score, diff) = compare_ssim(grayA, grayB, full=True)  
diff = (diff * 255).astype("uint8")
```



`compare_ssim` function returns a tuple containing a score between 0 and 1 which indicate the similarity of the images and also a numpy array `diff` which consists of values between 0 and 1. This `diff` array is multiplied by 255 in order convert it to RGB values.

```
thresh = cv2.threshold(diff, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
```



This is BINARY INVERTED THRESHOLDING in this thresholding if the intensity of pixel is higher than a threshold value then intensity of that pixel is set to 0, if it is lower than the threshold value intensity is set to 255.

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

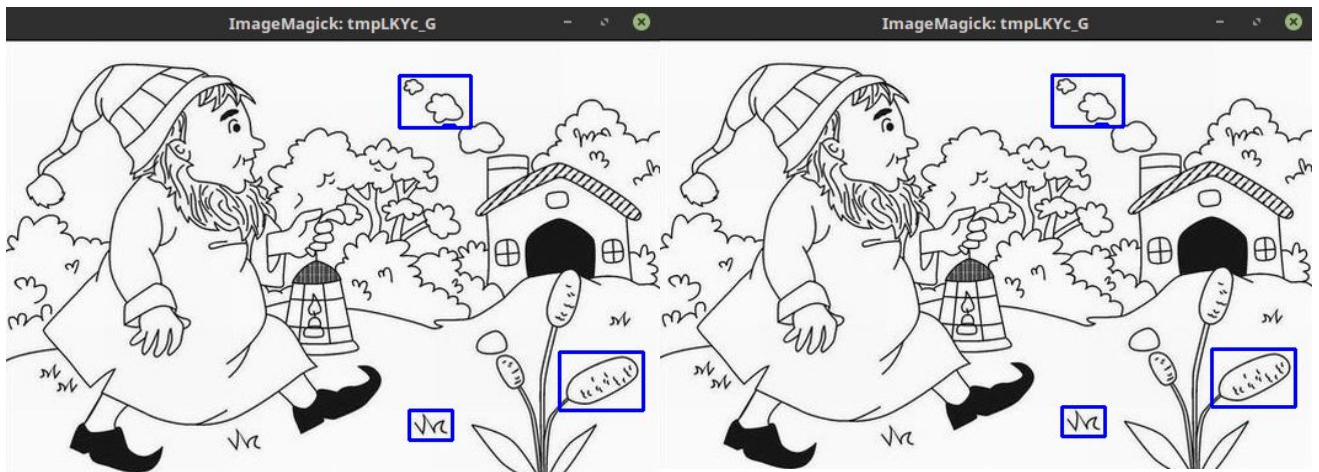
findContour function is used to find curves in the image that has been already simplified using thresholding.

```
for c in cnts:  
    (x, y, w, h) = cv2.boundingRect(c)  
    cv2.rectangle(imageA, (x, y), (x + w, y + h), (0, 0, 255), 2)  
    cv2.rectangle(imageB, (x, y), (x + w, y + h), (0, 0, 255), 2)
```

All the contours are looped through and a rectangles are drawn in the original images corresponding to the contour using the rectangle function.

```
dispararray(imageA)  
dispararray(imageB)  
cv2.waitKey(0)
```

Both the images are displayed using the earlier defined function and the program further waits for keystroke to exit.



The final result, the difference in both the images are bounded by rectangles.