**Count the number of boxes in the images**

**Introduction**

Here I have built a python program to count the number of boxes present in the image.

**Library**

**Opencv**

A library of Python bindings designed to solve computer vision problems.

**Matplotlib**

It is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy.

**Numpy**

It allows fast element wise operations on matrices and offers several arithmetic operations on arrays.

**Reading the image**

I have used the OpenCV, python library to read the image. For thresholding the image should be grayscale.

**Thresholding of image**

We need to clearly separate objects from the background for that we use Thresholding segmentation technique. It is a technique in OpenCV, which is used to assign pixel values in relation with the provided threshold value. Here each pixel is compared with the threshold value. If the pixel value is smaller than the threshold value, it is set to 0(black), otherwise it is set to a maximum value 255(white).

Here I have used Otsu’s Binarization which will automatically assign the best thresholding value that would work for the image.

**Inverting the image**

By inverting the image, it will ensure that the input image does not contain an alpha channel. So it will increase the accuracy for computing contours and corners of the box present in the image.

**Contours**

It is a curve joining all the continuous points along the boundary. The contours are a useful tool for shape analysis and object detection and recognition. In OpenCV, finding contours is like finding white object from a black background. I have used the findContours() function to locate the contours present in the image. It will give two output arrays.

First one is the contours which is a python list of all the contours in the image. Each individual contour is a numpy array of (x,y) coordinates of boundary points of the object. Second output is a contour approximation method. Which store minimum boundary points to draw the object. Which is used to save memory.

To draw the contours I have used drawContours() function. For a better contour image I have considered contours which have area less than 150 only. To draw all contours I have passed -1 in the function.

**Corners**

Contours are useful for any kind of image to find it’s boundary locations. But we have another more proficient OpenCV function to find the corners of the boxes. Which is more accurate than contours as well. Here I have used goodFeaturesToTrack() function. It takes four input parameters to compute the corners present in the image. First one is grayscale Image. Second one is the maximum number of corners we want as output (for all corners give negative value). Third one is Quality level parameter and Last one is Maximum distance.

This function was derived from the Shi-Tomasi Corner Detection.

They have used basic intuition that corners can be detected by looking for significant changes in all directions. Consider a small window on the image then scan the whole image, looking for corners.

Mathematically, for a window(W) located at (X,Y) with pixel intensity

I(X,Y), formula for Shi-Tomasi Corner Detection is

f(X, Y) = Σ (I(Xk, Yk) - I(Xk + ΔX, Yk + ΔY))2 where (Xk, Yk) ϵ W

Calculation of f(X,Y) will be really slow. Hence we use Taylor expansion to simplify the scoring function, R.

R = min(λ1, λ2) where λ1, λ2 are eigenvalues of resultant matrix

Using the cv2.circle() function, draw the circles at each corner.

**Number of Boxes**

To calculate the number of boxes I have simply put the mathematical formula. Which is the total number of corners divided by 4 as each box has 4 corners.

Here may someone come up with the problem while using the above intuition to calculate the number of boxes. As two boxes may have the same corner. THen the above formula does not hold true. But the Shi-Tomasi corner detection method does not calculate that overlapping corner as one corner. As it is based on a sliding mechanism window over the image. There is a slight difference between the location of that corner for two different boxes.

For one of the corners it calculated two different locations, [[ 936. 34.]] & [[ 929. 34.]]. Here we can clearly see the difference between the location for the same corner point.