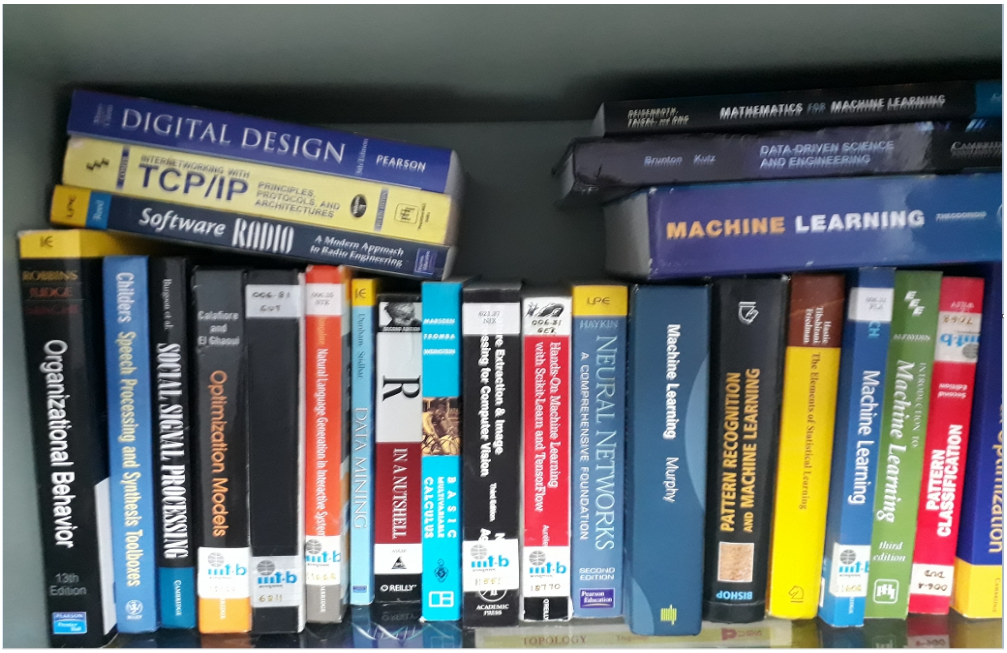
Problem Statement: We are given an image of bookshelf and we are asked to find the count of books in that shelf.

Given Image:



Solution:

As we are dealing with books, which comes in various colors like single book can have multiple colors on it hence we can't recognize books from their colors. **So we can't do any color related operations.**

The other thing which can be done is counting contours on this image and number of books will be equal to number of contours we got. **As we know contours are actually an outline of the object like bounding box.** Hence it seems promising at first.

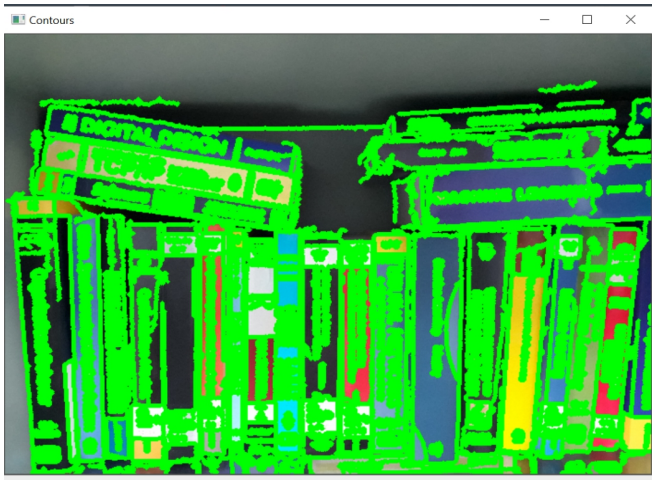
Assumption:

So the assumption for this method is, we will get bounding box around each book and hence number of books will be equal to number of boxes.

Approach 1 [Only finding contours]:

Step-1: First convert image into gray scale and then using canny edge detection found edges on the image.
Step-2: Now on this output image, find contours.
Step-3: And than found the total number of contours which is 1500 around. So to find out what the problem is I had plotted the contours on the image.

Problem with this approach:



To visualize I had plotted contours on the image and I found that, it is actually counting the **written text as contours**. So i need something which makes that **written text blurred.**

Approach 2[Blurring the written text]:

Step-1: First simply blur the given image using bilateral blurring **(because it doesn't blur the edge).**
Step-2: Now to remove text, **apply erosion with kernel size** of 5*5 with number of iterations as 1. (this size and number of iterations was found by try and error).
Step-3: After this again found the contours using the same method as discussed before.

Problem with this approach:

It is still counting small chunks as contours and also certain books are not considered which might be because we have done erosion.

So to deal with this both problems, I came up with the final approach.

Final Approach:

- First apply bilateral blurring to the input image, which will reduce noise upto some extent.
- After that apply erosion with kernel size of 5*5 and number of iterations as 1.
- After that convert the image to gray scale and than find edges in that image using canny edge detection algorithm.
- Now as there are opened edges in the output, so to connect this edges apply morphological closing operation with same kernel size as we had previously used and number of iterations as 1.
- Now find the contours on this image and to decide whether it is book or not find area of each contours.
- Store area of each contours in array and than sort it and pick the 60^{th} largest area size as our threshold value (This value is found by try and error). Because as books are almost of similar thickness so like if area is small than it's not a book.
- So now finally iterate over contours array and if area of that contour is more than threshold than append that contour points in new array.
- **But on seeing the area values, I found out that the last few values were extremely large, it might be because they are contours on more than one book. So we will not consider that contours.**

Algorithm:

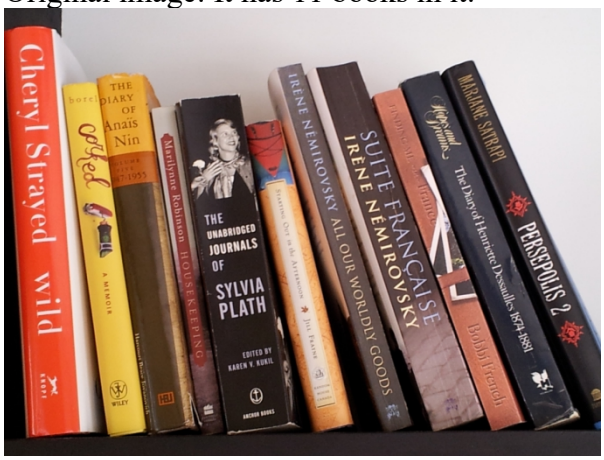
Step 1: Import necessary packages and read image and resize it.
Step 2: Blur the input image, using Bilateral blurring as it does not blur the edges.
Step 3: Do erosion operation, so the content return on the books gets removed.
Step 4: Now convert the eroded image into gray scale and find the edges in it.
Step 5: Now before finding contours first try connecting the disconnected portions in the edged image. So apply morphological closing.
Step 6: Find contours.
Step 7: Now to decide whether given contour is around book or not, first find area of all contours and than sort that array. And consider threshold as [length of array - 80] indexed value of array.
Step 8: Using this threshold, create new contour array whose area is more than threshold.
Step 8: Now finally print the length of contour array as output and to visualize it draw contours on image.

Output:

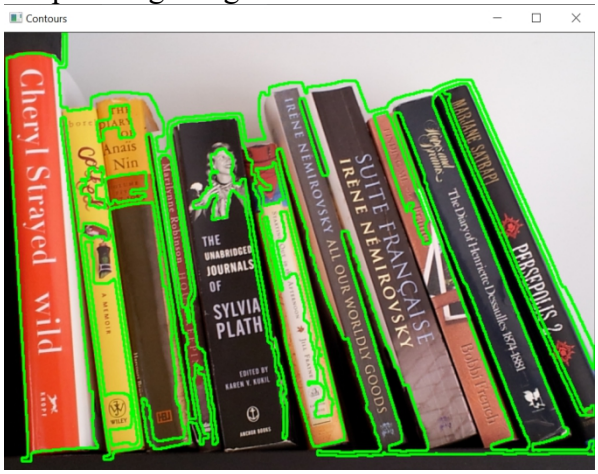
Books count by this approach is 21 which is close to the original count i.e 25. And the contours formed on this were as shown below.

Trying it on another image:

Original image: It has 11 books in it.



Output image: It gives 15 as book count.



Conclusion:

Because of the written text on books and multiple books are considered as single contour this approach is not working accurately. Moreover we have set that threshold which is hard-coded hence it won't generalize well.

Problem Statement: We are given an image of Rishab Pant, we need to mark pixels of faces.

Given image:

Solution:

Approach 1:

Initial Assumptions:

As here we have to cluster all liked pixels, so the first thing came into my mind was to apply **k-means clustering.**

So we can think of it as clustering all faces pixels together and keeping rest all pixels together. On looking at the image we can see that there are basically 3 color things in the image, **1. Face color, 2. White jersey, 3. Background blur top portion** so I had used **K = 3.** So that there will be 3 cluster centers related to this portions.

Algorithm Steps:

Step 1: First do blurring to reduce noise in background.
Step 2: We need to provide flatten array as input to cv2.kmeans() function, so flatten the image.
Step 3: Now define stopping criteria like max iterations, accuracy and define k value.
Step 4: Now finally call the cv2.kmeans() function. This will give us cluster centers.
Step 5: Reconstruct the image back from the cluster centers we got from above step.

Output:

Observations:

The result seems quite good, it has marked all face pixels but at the same time it has also considered background blur faces as the face pixels which is correct but we were interested in marking only Rishab's face pixels. Moreover it has also considered handle of bat as face. And also the beard on face is not considered as face pixels, but this is not a major problem.

Also the it is not able to handle illumination factor, like look at the left ear of above picture, as there is more brightness in that region, it is classified as T-Shirt picture.

Modified this approach based on observation:

So it seems like if we can reduce background noise than this approach will work good.

So to do so, I had tried various blurring methods and also morphological operation like erosion and dilation. But none worked well. And the above mentioned result was the best result I could come up with this method.

Where will this work and where will it won't:

This method will work well in case where we have almost no background noise and there are less colors in the frame and also person should not wear any accessories. There should be exactly same illumination on complete face of the person.

But it will fail miserably when there are plenty of colors or person is wearing clothes whose color is close to skin color or something like that.

Approach-2:

As we are asked to mark pixels of face and pixels of face is of **skin color,** so we can actually do a skin detection.

Initial Assumption:

As skin color value belongs to particular range and this range may vary with the change in the color space in which image is defined. So initially I thought this approach will mark all the skin color pixels and other pixels as black.

So to apply this method I had tried it on two different color spaces that are HSV and YCRCB.

Algorithm steps:

Step 1: Define min and max array of skin color pixel values based on the color space used (This values are found with the help of google and try and error).
Step 2: Convert the given BGR image into respected color space.
Step 3: Now apply cv2.inRange() function with min and max values as parameters. By doing this all the pixels in this range will be marked as white and others will be marked as black.
Step 4: Using the output of above step as mask, apply this mask on the input image that is will do bit-wise AND.

Output:

YCRCB

HSV

Observation:

First of all it is clear that results are better in YCRCB color space as compared to HSV space. But even in this space, the background blurred images of other players are still visible. So it's clear that we need to some how remove the background noise completely.

Moreover here it seems it will work under different illumination conditions, which was not possible in above approach.

Approach-3:

From above observations, we are clear that we need to remove that background noise completely.

So to do so, we will create an mask image of same size as that of input image with circle in between of white color and rest portion as black. And than will apply bit-wise ANDing between this mask and original image.

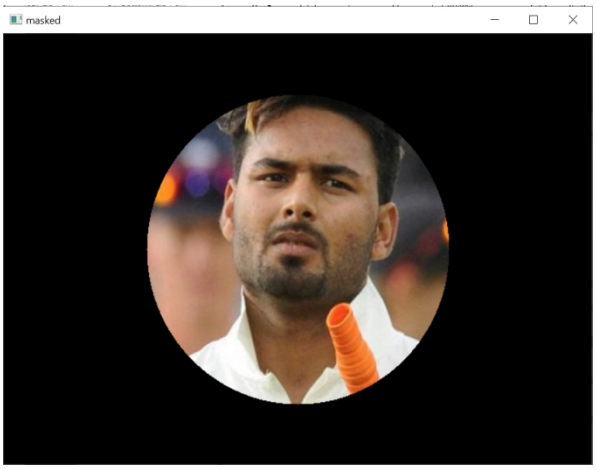
And after that, we will apply same algorithm as mentioned above to detect skin in that image.

By doing this we are removing background completely.

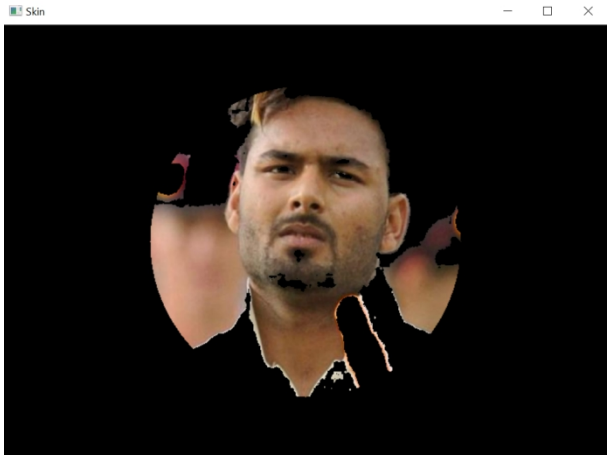
Algorithm:

Step 1: Create a complete black image of same size as that of input image.
Step 2: Draw filled white circle on this black image in the middle.
Step 3: Now do bit-wise ANDing with input image with mask as out new image.
Step 4: Now define min and max array of skin color pixel values based on the color space used (This values are found with the help of google and try and error).
Step 5: Convert the given BGR image into respected color space.
Step 6: Now apply cv2.inRange() function with min and max values as parameters. By doing this all the pixels in this range will be marked as white and others will be marked as black.
Step 7: Using the output of above step as mask, apply this mask on the input image that is will do bit-wise AND.

Output:



Output image from step-3.



Final output

Observations:

The results are quite good, but still we are not able to **extract the face perfectly**. But still this method will work good in the case where there are no skin color objects in close proximity to the face image.

Limitations:

As we are drawing circle in middle, this method is not at all invariant to scaling or position of the face. Like if face is in left or right border it won't crop it. Hence we need some mechanism to actually crop only the face portion, which is invariant to location of face and scale of image.

Approach-4 [Final]

In this method we will find contours on the given image and from this contours we will extract the one around the face. Like will create a mask as above but the difference is in this mask the white portion will be of same shape as that of face contour. And than will apply this mask on the input image.

But as in the first problem we saw that contours will also be around small regions like eyes, hairs, lips and many more other objects. So to deal with this first of all we will use a threshold so all small area contours will get removed.

The other thing **was how to know that this contour is around face ?** So the straight away solution was to assume, the largest contour will be around face. But this assumption was wrong because largest contour was considering the shoulders. So to just consider face, we will apply another threshold that will restrict on width.

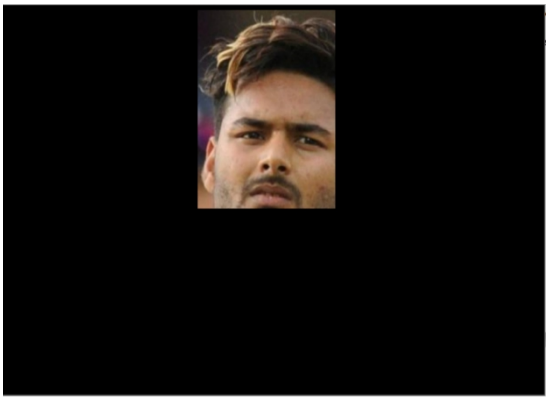
Algorithm:

Step 1: Convert the image into gray scale and do canny edge detection on it.
Step 2: Now as there were few open edges, apply morphological closing operation on it.
Step 3: Find contours on the output of step 2 image.
Step 4: Create a new_contours array by considering only those contours whose area size is more that certain threshold.(Value found by try and error.)
Step 5: Now iterate through new_contours array and create a bounding rectangle around this contours as close as possible using cv2.boundinRect() function.
Step 6: The above step will return width, height, x and y position for each box, now check if width and height are greater than certain threshold and also width is smaller than certain value (to restrict shoulder contour) than calculate area of that box and this way find the maximum area box.
Step 7: Now create a mask using the coordinates of face we found from above step.
Step 8: Apply this mask on the input image and than again do the skin detection as done in approach-1.
Step 9: And than finally convert that image into gray scale and mark all pixels which are not black as white. (Face pixel as white and rest as black)

NOTE: If we remove that thresholds in step-6 and save all images that are generated than we can manually pick the one which has best result.

Output:

Mask created from step-7



Final output image.

Trying with my photo: (For this I had changed width threshold at step-7)

Input image:

Output image:

Conclusion: If there are some bigger contours in background than this method will consider it, so to make it generalizable images need to be picked manually. And also if the picture has skin colored object too close to face, than it will mark it as face.