A simple detector for crowd counting using OpenCV and Python

Martí Gelabert Gómez

November 24, 2022

Contents

1 Introduction

The assignment consists in **counting** the number of people that appear on the given images. There was not an stablished way to aproach this problem. In this case I have taken the decision of using the computer vision algorithms seen in class and not relay on deep learning or any kind of artificial inteligence algorithm. This is because not by de amount of data needed for training, due to there are really god datasets out there, like MS COCO, but because of the effort of building a good performing detector architecture. Use an already set framework as YOLO or Mask-RCNN would be the way to accomplish the task, but I would be thinking about it as "cheating". Therefore, this way, using the content seen in class, I will be more cautious about my decisions and It will be much easier to justify them.

In the following document we will be focusing on the process taken and the algorithms used, their implementation and their performance.

2 General Procedure

The program takes the following steps :

1. Import images as black and white.
2. Apply Adaptative Histogram equalization to the images.
3. Substract the background to the images using the image with the background.

4. Apply a thresholding algorithm to binarize the image.
5. Apply a dilation operation into the binarized images to expand the whites.
6. Use a contour algorithm to extract the diferent regions containing persons.
7. Obtain the bounding boxes from these regions.
8. Count them and compare the number of detections to the real cuantity.

3 algorithms

In this section we will discuss the algorithms selected and the output we obtain from them in the application of our problem.

3.1 Background Removal

Background removal is a technique that allows to remove the background from the image, this way, the output will be only the foreground highlighted in the form of a binarized image as it can be ilustrated on the figure 1.

(a) Output of background substraction using cv2.threshold(image, 100, 255, cv2.THRESH_BINARY)

(b) Original 1660320000.jpg

Figure 1: Reference images

In the following sections will be explained how the preprocessing of this technique was prepared and for what they are used for. They are ordered by

CLAHE

In the case of our problem, first we have imported all the images in black and white integer values using `cv2.imread(img,0)`. Then for a better extraction, it has been applyed an **algorithm of CLAHE** seen previously on the course to try to uniformize the ilumination of the images. With a more uniform images, we will be able to distinguish more precisely our foreground and artifact like shadows casted by objects on the images may have less impact on the **binarization process**.

(a) 1660284000.jpg with CLAHE applyed

(b) 1660287600.jpg with CLAHE applyed

(c) 1660302000.jpg with CLAHE applyed

(d) 1660284000.jpg gray

(e) 1660287600.jpg gray

(f) 1660302000.jpg gray

Figure 2: Images from Gelabert folder where the first row presents a more uniform ilumination across frames ompared with the second row images.

Background Image

For the selection of our background image, in a starting point the **average image** was computed as the main way for accomplish foreground extraction, but the resulting image generally was really noisy and with ghosting effect very present on it, which decreased the performance of the algorithm as it can bee seen on the Figure 3. Even with a gaussian blur applyed to try to counter the ghosty figures, they are still really present on the image. The image averaging is a good idea generally for noise reduction and in this case to try to obtain a neutral image were the background is predominant, but there is not enought images to obtain a more sharp image to really have a good background. Therefore, we **end up** using the image 1660284000.jpg with a gaussian blur applyed as our background, this way we obtain better results on the **binarization process**.

Gaussian Blur

With an image already selected for the image background, it is applyed a gaussian filtering with a kernel of size (7,7), this way we can obtain a more standard image, and get a more smuthered image that will allow the algorithm to contrast slightly better our foreground.

The ending size used for gaussian blur was obtained by trial and error experimentation, and could be changed to get different ending behaveours.

Substraction

Once we have selected our background and applyed the preprocessing, we should substract it to every image, this way, the result we end up with is the foreground (in this case the persons and some residuals) highlighted as we can see in the figure 5.

3.2 Binarization

Once the substraction has been aplyed to all images, the resulting foreground will contain hopefully our persons. All those whity areas should be further treated using a binarization progress, allowing a solid division between the background and our foreground. This is acomplish by the process of binaryzation. Depending of the process selected, the resulting image will transform into a binary black and white image from a gray scale one. In this assignment there has been some experimentation using OTSU and the binary threasholding function avaliable on cv2 library, but for the sake of the performance, the last one has been ultimately selected.

Figure 3: Average image with a gaussian blur applyed with a kernel of (11,11)

Figure 4: Gaussian Blur Applyed with a kernel of size (7,7)

Figure 5: Image obtained by applying cv2.subtract(background, image)

The problem with using OTSU is that for each images it obtains a threshold selection automatically, and this should not be a problem, and sometimes could perform better than a fixed one. But for the samples tested there are a lot of artifacts related to the shadow casting of the image, as it can be seen in the figure 6. The problems are seen on the bottom of the image, where there are an amount of shadows casted by the sand crests which can be a bit annoying for the next steps.

Figure 6: Binarized image using OTSU and the empty beach image

By using a set thresholding we are loosing some flexibility and a fixed values could not be the wiser idea, but in this case using a tigth threshold may be the best decision as may reduce the number of false detections considerably. In this case the values selected are setted in the instruction `cv2.threshold(substracted, 100, 255, cv2.THRESH_BINARY)`. The resulting image can be shown in figure 7.

Masking

For a better filtrage of our data, we will be using a binary mask to get rid of centain areas of the image where we can get non-persons in the foreground (i.e the quay part) and those areas where there is to much information to get clear data about it. We can see the last case in the umbrella section of the beach and the pavement area, where there are too many persons overlaping each others. The mask applyed can be seen in the figure 8.

3.3 Dilation

Now, with the image binarized we will a apply a dilation to the image. The dilation will expand the white color on a binarized image, obtaining as results the highlighted areas in form of white chunks as it can be seen on figure 9. With these chunks the countourn detection will obtain better results because they are more clear shapes. This could merge figures into one, so the parameters of the kernel size may be tinkered with to obtain the best performance possible.

Figure 7: Binarized image using a fix thresholding method

Figure 8: Binary image

3.4 Find contours

Once a binarized image is obtained, there should be a process of contour detection. Here we use the dilated images over the function that openCV provides `cv2.findContours()`. From this functions, we can extract the bounding boxes that contain each of the object contours. In the figure 10 appears the image with some bounding boxes applyed.

3.5 Matching Algorithm

For convenience, we will cuantify our detection accuracy using the some "incorrect" assumptions. We will be checking if the bounding box contain labels, only one label containing will be assigned to that bounding box. In the case we would have a really big box keeping a lot of detection we will not count that as correct, but in the cases where it may contain a region of 4 or 5 persons because they are really close to each other, we would be more flexibles.

For checking the dimensionality of the bounding box, we would assume that width or height higher than a third of the image will not be acceptable, and the same for the ones no more than two pixels size.

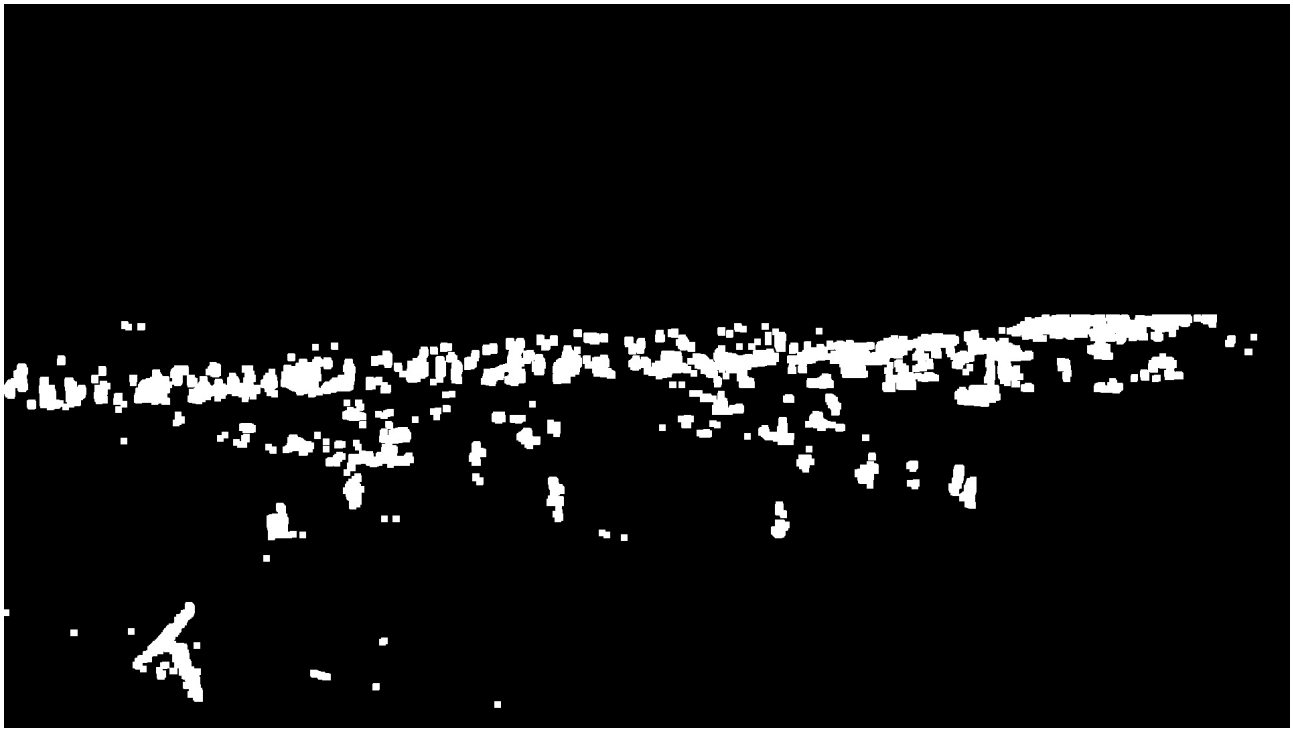

Figure 9: Dilation applyed to the binarized image

Figure 10: Output of foreground detector, as it can be seen there are some big detections due to contourn detecting multiple forms as one shape, and some

4 Results Analysis

Table 1: Performance metrics using the propossed algorithm

files	precission	recall	f1 score	gt	detected	matched
1660309200.jpg	0.529	0.700	0.603	90	119	63
1660302000.jpg	0.446	0.563	0.498	103	130	58
1660294800.jpg	0.535	0.736	0.620	72	99	53
1660320000.jpg	0.770	0.770	0.770	135	135	104
1660287600.jpg	0.275	0.647	0.386	17	40	11
1660298400.jpg	0.663	0.594	0.627	106	95	63
1660305600.jpg	0.595	0.653	0.623	101	111	66
1660316400.jpg	0.888	0.856	0.872	139	134	119
1660291200.jpg	0.667	0.654	0.660	52	51	34

5 Is a different aproach capable of improving our results?

For some of the test that were executed for this assignment, generally the effort was not useful. The following

Ending up in situations where a precise modification to the parameters gave as output really good results only in a set of the images. Usually the over-tunning of some function parameters made really inconsisten outputs that provoked a bad performance.

Color Spacing

For example, one of the aproches taken for background removal, was the use of colored images in the color space of LAB instead of gray scale. On the paper one may think that the use of color will improve our performance, because of the extra information we are just acquired from free. But in this case the process to obtain our foreground sections were in general less precice, giving as result a lot more objects as foreground, which is not ideal for our case, been a lot of those object just umbrellas and cast shadows.

The problem with shadows

In general, the sample images follow a consisting landscape withouth heavy cast shadows on the sand, but it is noticeable in some of them and usually they will give you more than one headache.

We could try to counter some of the cases using a sharpening processes or even more filtering (i.e median filtering), to avoid certain annoience for shadows, but doesn't mean that it will be worth it to apply. We could have the situation again where is really useful for 1 case, but then the rest of images just got obliterated just for applying those transformations.

6 Conclusions

As we could be expecting, the performance is not really good compared to a neural network. There are some cases where we can be completely sure about classifing incorrectly our detection because of how the ground truth is placed, some detections are actually correct but slightly shifted and withouth a manually intervention it is safer to just leave it as 'incorrect'. In general, the amount of false positives increase in function of which process are used but overall the false positives generally concetrates in distance areas where there is a lot of information compacted and the dock area.

The use of background subtraction to try to extract completely our foreground is a tricky task, because using an averaged image could allow to have an image with the solid background on it, but it works better when we dispose with a lot of images. With just the images from the 'Gelabert' folder the image generated was not clear enought (even with some pprocessing) to accomplish the objective and in the end jsut using the emtyes image was the one given the best performance.