Image Analysis Project

The aim of this project is detecting people from beach images. The dataset consists of 10 images, 8 of them have people in them with various intensities and two of them are completely empty versions of the beach. The first task has been done was annotating the images manually by using the makesense ai tool.

Data Annotation

875 people identified manually in 8 pictures and their pixel data in bounding box saved in a csv file. All the images have the same size as 1920x1080 pixels. The structure of the file is seen in Figure I. **bbox_x** and **bbox_y** are representing the x and y coordination's of the upper right pixel of the bounding box. Width and height of the boxes also stored in the dataset.

	label_name	bbox_x	bbox_y	bbox_width	bbox_height	image_name	image_width	image_height
190	Person	432	543	10	20	1660471200.jpg	1920	1080
161	Person	1168	554	8	45	1660467600.jpg	1920	1080
493	Person	1214	648	31	20	1660478400.jpg	1920	1080
648	Person	956	529	12	20	1660482000.jpg	1920	1080
561	Person	1816	941	98	28	1660482000.jpg	1920	1080
301	Person	295	629	13	58	1660474800.jpg	1920	1080
496	Person	1154	491	7	20	1660478400.jpg	1920	1080
340	Person	353	550	4	17	1660474800.jpg	1920	1080
709	Person	716	508	5	3	1660482000.jpg	1920	1080
669	Person	1622	546	7	18	1660482000.jpg	1920	1080

Figure I: Annotated data examples

The most crowded picture is image 1660482000.jpg with 159 people and the distribution is shown in a bar chart in Figure II.

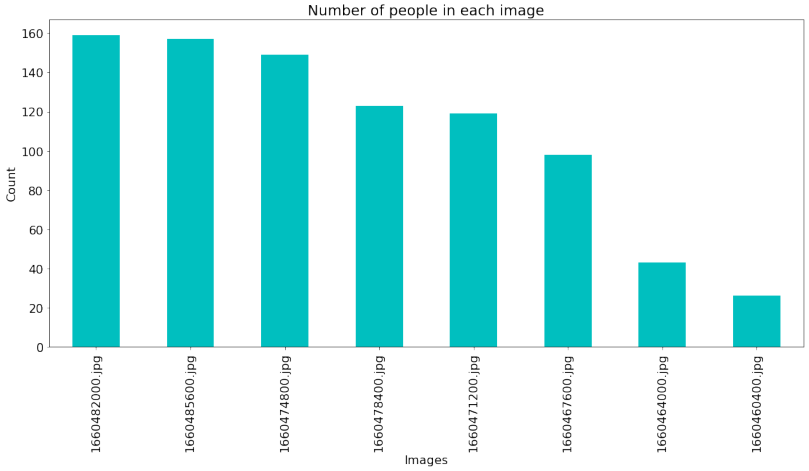

Figure II: Number of people in each image

Algorithm Design and Implementation

Since the empty image of the beach is provided, it is possible to implement an image averaging algorithm with pixel-wise comparison. From the two empty images, one of them is brighter and one is darker. Therefore, the brighter beach images compared with the brighter empty beach image and the darker ones are compared with the darker beach image. One of the resulted images by pixel-wise comparison is given in Figure III. Python's pixelmatch framework is used to implement this functionality. Several parameters are tried to improve the results and threshold value decreased to 0.05 from the default 0.10 value.

Figure III: Pixel-wise comparison result

Comparing pixel by pixel takes relatively long time (around 10-15 seconds per image) and the result is poor if there is a light difference that causes shadows. In addition to that, if there were no empty images of the beach, it would not be possible to implement this solution. Because of that, another method which is background removal has been implemented. For background removal image is blurred with a (11,11) filter to reduce the noise and otsu thresholding is used to extract the background. The result is shown in Figure IV.

Figure IV: Example result of background removal algorithm

Contour Detection

A function is implemented named as findContours. It takes as parameters the original image and img_tec which can be the result of pixel-wise comparison or background removal. First, it transforms image into grayscale and uses OpenCV's Canny. The bounding boxes are drawn with certain limitations. The boxes bigger than 10 pixels width and height values, and smaller than 100 pixels kept only. Another limitation is not taking the upper part of image into consideration since that part only consists of boats and mountains. The rectangles drawn in pink and the function also returns a dataframe consisting of all bounding boxes details in addition to the resulted image. Figure V shows one of the resulted images.

Figure V: An example of contour detection result

Validation

Validation of the images consists of two levels. Person level with accuracy and precision calculation. Image level with mean squared error calculation. For the person level validation, a function named as validateBoxes developed. This function returns validated images as in Figure VI format and the dataframe with accuracy, precision, number of True Positives and False Positives, number of actual boxes and image name.

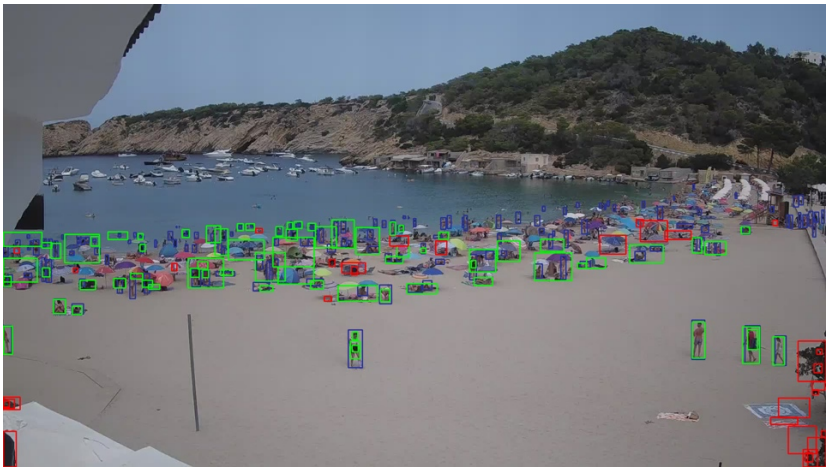

Figure VI: Validated Image

If the resulted boxes match with the actual boxes, then the resulted box colored as green and if there is no match the resulted box colored as red. The blue colored ones are the actual annotated ones. The results with pixel-comparison had accuracy around 20% and precision around 30%. On the other hand, the results with background removal had 31.5% average accuracy and 41.5% average precision. The highest accuracy for an image is around 40% and the precision is 65%. The lowest is as low as 8% accuracy and the reason is the poor performance for a highly shadowed image.

Accuracy is calculated as true positives divided by sum of true and false positives. Precision is calculated as true positives divided by number of actual values.

For the image level validation, mean square error is calculated based on the center points of bounding boxes. For each resulted boxes' center, the Euclidean distance to the closest actual box is calculated. The mean squared error is as low as 5.3 for the best resulted image but as high as 283 for the worst resulted image. However, the average of all the images is 57.3 considering the image's pixel size is 1920x1080. The mean squared error is not high and the algorithm is performing good.

Conclusion

Although the average accuracy and precision values are not very high, the boxes which are not matching with the actual ones are still close to them. Because the calculated error value is less than 30 pixels for most of the images. The difficult part of this project is distinguishing umbrellas, bags, or towels from humans. Because it is not possible to train a good algorithm with only 10 images. The result is still satisfactory without using any deep learning method.