Assignment A4: Gaussian Pyramids and Normalized Correlation

Clinton Fernandes

u1016390

02/27/2016

A4

CS 6320

1. Introduction

The purpose of this assignment is to study the camera calibration process. For this, we need to develop Matlab functions implementing normalized correlation and Gaussian pyramids and then use these to develop a method to detect hands raised in surrender in an image and provide a Matlab function for this.

We need to optimize the hand detection method based on success (all raised hands in the image are found), and efficiency (lowest computational cost).

Hence the following questions can be asked through this exercise:

- How efficient is the method of hand detection?
- How many raised hands can be detected?

2. Method

To detect the hands in an image we require to find where the hand or the object lies in the image under consideration. This can be done using the Normalized correlation function. In this function, a template image of the hand is passed as the input, then it is checked where this template matches the areas in the image and the magnitude of correlation (-1 to 1) is obtained at every pixel.

With the correlation function itself, we are able to detect the hands in the image and hence the point of interest can be shown on the image. But this will not result in the optimum method, i.e., not all hands may be detected. Hence we need to use the Gaussian pyramid here. The Gaussian pyramids a collection of representations of an image. Each layer of the pyramid is half width and half the height of the previous layer, also each layer is smoothed by a symmetric Gaussian kernel and resampled to get the next layer.

The Gaussian pyramids are useful because, it helps to reduce the computation time required for hand detection, due to the smaller images at every successive layer. It helps to perform coarse to fine matching. We may be able to detect the hands better at a successive level of the smoothened image and also the computational time for a higher order level of the pyramid is lesser than it's previous.

Finally, having the normalized correlation function and the Gaussian pyramid function, we can make the hand detection function. In this, a normalized correlation is performed onto each image of the Gaussian pyramid, with the help of a hand template. Having detected the locations in each image of the Gaussian pyramid, we can display these regions onto the original image.

Also we can check out how efficiency and the success rate of the method in finding the hands, by comparing the number of hands detected in the different level of the Gaussian pyramid and also by checking the computation time required for every level of the Gaussian pyramid. We can use precision and recall method here.

CS5320_normcorr

To detect the location of the hands in the image, we require the correlation function. Input to the function is the Template and the image. Output of the function is the correlation coefficient matrix, which tells about the degree to which the template matches to the region/ location in the image. The template is assumed to have odd number dimensions, hence if the columns are 22, than an additional column number 23 with 0's is added. Similarly, for rows, an additional row is added to the top.

Algorithm:

```
Input to the function is Image im, and template T
[rows, cols] =  size(im)
kr = round(rowsT/2)-1;
kc = round(colsT/2)-1;
Now , we pad the image with constant values (0's), such that the offset of padding = k…
…store this new image as big_im
[temp_rows,temp_cols]=size(big_im);
Let big_im(1+k:rows+k,1+k:cols+k) = im
C_temp=zeros(temp_rows,temp_cols);
C = zeros(rows,cols);;

T = T - mean(T(:));

for i = 1+kr: temp_rows-kr
    for j= 1+kc: temp_cols-kc
        temp_im = [];
        temp_im = big_im((i-kr):(i+kr),(j-kc):(j+kc));
```

```
                temp_im = temp_im - mean(temp_im(:));
                numerator = sum(sum(temp_im.*T));
                denominator = sqrt(sum(sum(temp_im.*temp_im))*sum(sum(T.*T)));
                if denominator == 0
                        C_temp(i,j) = 0;
                        continue;
                end
                C_temp(i,j) = numerator / denominator;
        end
end
C = C_temp(1+kr:temp_rows-kr,1+kc: temp_cols-kc);
```

CS5320_G_pyramid

This function which gives a collection of representation of an image takes in 3 inputs; the image, k - size of the smoothing window if window is 2k+1 and sigma for Gaussian function. Output of the function is the collection of representation of the image, such that every image is half the width and half the height of the previous image and a Gaussian filter is applied to the image at every level.

The Gaussian pyramid can be achieved by applying a low pass filter to the original image then scaling down this image to half the dimensions.

Algorithm:

```
Input to the function is Image I, smoothing window size k, sigma for Gaussian function
Kernel size = k
[rows, cols] size (Image)
Now , we pad the image with constant values (0's), such that the offset of padding = k…
…store this new image as big_im
Initialize the G_pyramid = zeros (rows, cols , 5)
For each pyramid level from 2:5
        Let big_im(1+k:rows+k,1+k:cols+k) = I;
        for i = 1+k: temp_rows-k
            for j= 1+k: temp_cols-k
```

```
                Filtered_big_im = sum(sum(T.* big_im((i-k):(i+k),(j-k):(j+k))))
            End
        End

        Now the filtered image  = Filtered_big_im(1+k:temp_rows-k,1+k: temp_cols-k)
        %unpad the Filtered_big_im

        Now, we Subsample the image such that value of ith, jth pixel of new image is equal
        to the value of the 2ith, 2jth pixel of the filtered image;

        Add this sampled image to the current level of the gaussian pyramid
    End
```

CS5320_hands

The input to this function is the image from which the hands are to be detected. The image is inputted like ‘s1.jpg’. Output of the function is the RGB image of original with hands outlined in red.

To call the function for image 's1.jpg’:

```
    hands = CS5320_hands('s1.jpg')
```

Algorithm:

```
    First, we load the image using imread(im)
    Convert the image to grayscale using img = rgb2gray(im)
    Template T is made as
        T = zeros (23,15);    T(6:23,6:10) = 1;
    Apply a laplacian of Gaussian filter to the image
        G = fspecial('gaussian',7,2);
        Lap = fspecial('laplacian',1);
        LoG = filter2(Lap,G);
        img =filter2(LoG, double(img));
    Get the Image pyramid (Ipyr) and the Template pyramid (Tpyr) for the image and the …
    …template by calling the CS5320_G_Pyramid function.
```

```
For every level of the pyramid:
        Take the corresponding image and the template
        Consider only the unpadded region of the image and the template
        Run normalized correlation…
        …C = CS5320_normcorr(grad_T,grad_im);
        After obtaining C, the matrix for the normalized correlation coefficients, we should
        plot the pixels in red only at those places wherein the correlation coefficient is higher
        than some threshold value which is calculated as:
              ct = C(:);
              ct = sort(ct);
              *numX = round(0.9975*size(ct,1));
              threshold = ct(numX);
        Now set the R value of RGB of the corresponding pixel in the original image to 255
        **At this place, we have the image with the detected hands at the current level of
        Gaussian pyramid.
        At the end of this program, we can set the content of the variable hands to the best
        performance Gaussian level image (mostly this is the 1st or 2nd level image). Or, we
        can add all the image locations having red values.
End
```

*It is seen that different images give better result for a different value of threshold. The current value with the factor of 0. 9975 is selected for all images. If we select this number to be different for different images than we get much better result for every image. For example, setting this value to 0.991 for image 's3.jpg' we get 9 hands instead of the current 5 hands.

**benefit of taking a concatenated image of all the 5 resulting images via Gaussian pyramid is that some images detect hands at locations which others do not. Addition of all gives a better result (this is seen in 's3.jpg')

Disadvantage is that some hands are detected multiple times at different region of the same hand (this is seen in 's1.jpg').

Currently I have used result of all 5 images.

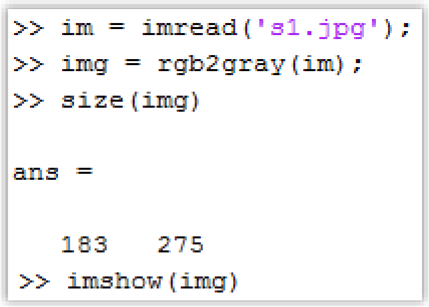
3. Verification

CS5320_normcorr

To verify this function, we can take an image, crop a small section of the image (such as hand or any other object). Then use this cropped section as the template.

After using the image and the template as the inputs to the function, we can check the output of the pixel at the center location in the image from which the template was cropped.

let us consider the image 's1.jpg'



```
>> im = imread('s1.jpg');
>> img = rgb2gray(im);
>> size(img)

ans =

   183   275
 >> imshow(img)
```

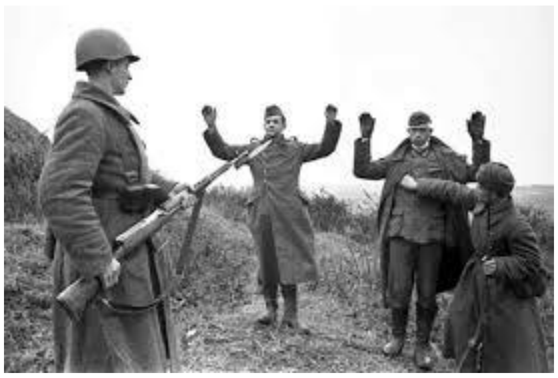
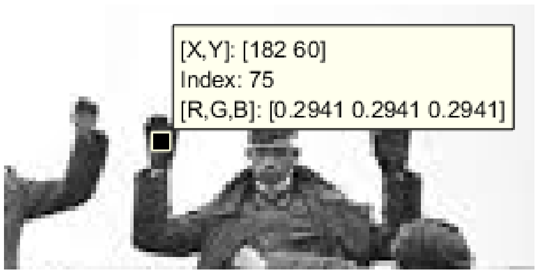


figure 1: image 's1.jpg'

Then we check thecenter pixels of the required tmplate



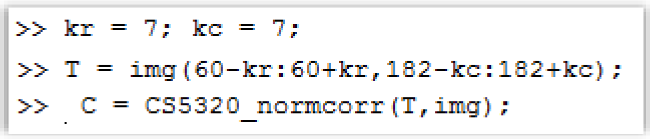
```
[X,Y]: [182 60]
Index: 75
[R,G,B]: [0.2941 0.2941 0.2941]
```

figure 2: selecting template location

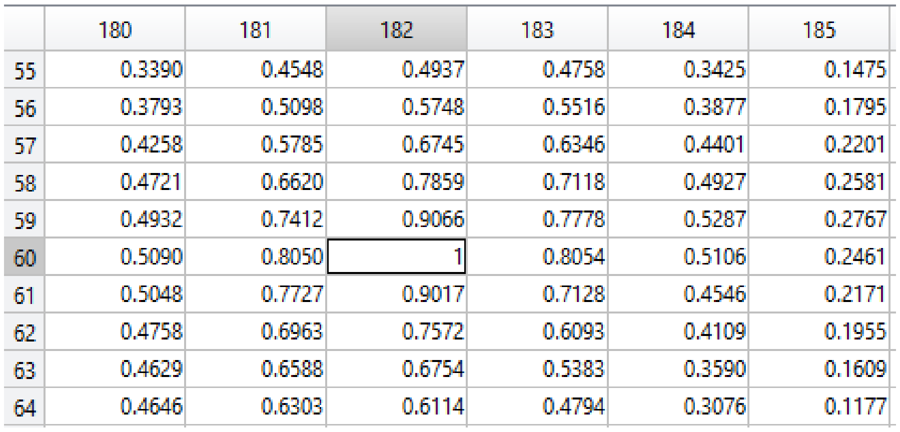
let us crop the template image at the 182,60

Using the template size as 15 x 15; 2k*+1 = 15

and applying the CS5320_normcorr function



```
>> kr = 7; kc = 7;
>> T = img(60-kr:60+kr,182-kc:182+kc);
>> C = CS5320_normcorr(T,img);
```



	180	181	182	183	184	185
55	0.3390	0.4548	0.4937	0.4758	0.3425	0.1475
56	0.3793	0.5098	0.5748	0.5516	0.3877	0.1795
57	0.4258	0.5785	0.6745	0.6346	0.4401	0.2201
58	0.4721	0.6620	0.7859	0.7118	0.4927	0.2581
59	0.4932	0.7412	0.9066	0.7778	0.5287	0.2767
60	0.5090	0.8050	1	0.8054	0.5106	0.2461
61	0.5048	0.7727	0.9017	0.7128	0.4546	0.2171
62	0.4758	0.6963	0.7572	0.6093	0.4109	0.1955
63	0.4629	0.6588	0.6754	0.5383	0.3590	0.1609
64	0.4646	0.6303	0.6114	0.4794	0.3076	0.1177

We are getting a value of 1 at exactly the location, from where we had cropped the template

CS5320_G_pyramid

We know that G_pyramid function will give a collection of images, such that the image at the 1st level is the original image and the ones at the successive levels are the images with Gaussian filter applied to scaled down from the previous image. Here, we can consider the image 's1.jpg'. The image is shown in *figure 1*. We can verify the function by checking the size of the image at every level of the pyramid, ensuring that it's dimensions are half that of the previous image. Also we can notice that every successive image is smoothened.

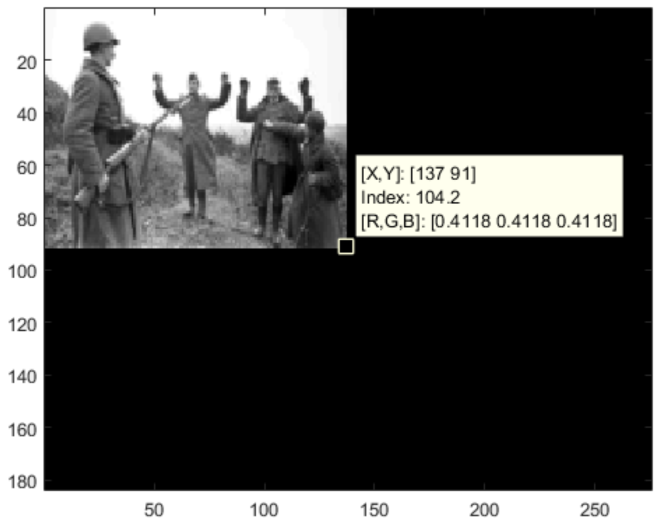


fig 3: level 2 image

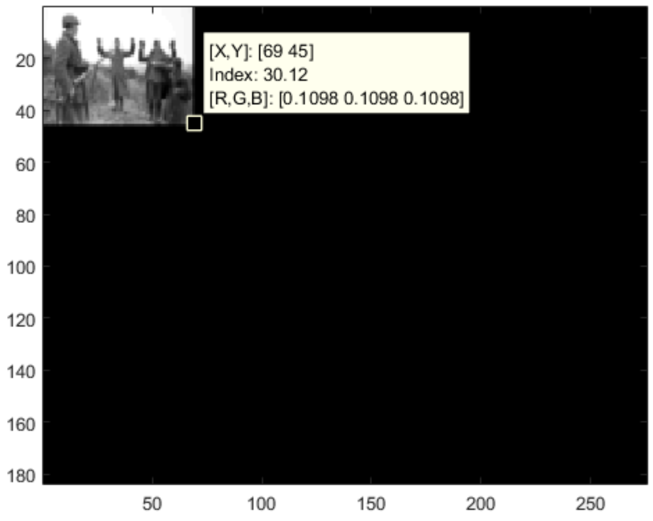
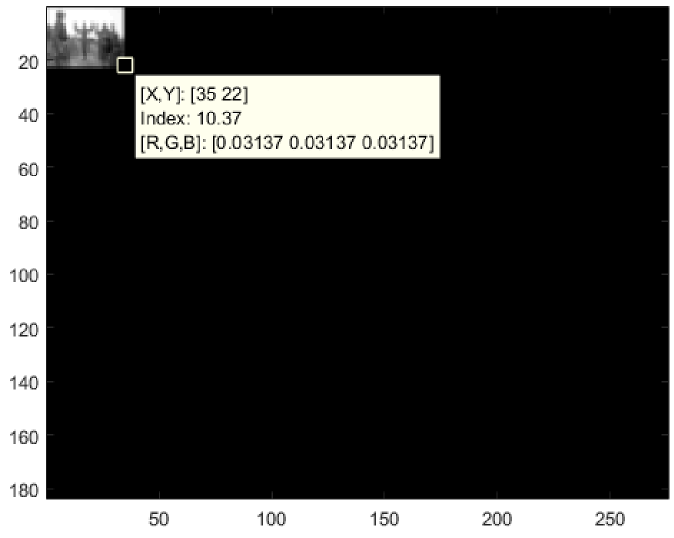


fig 4: level 3 image



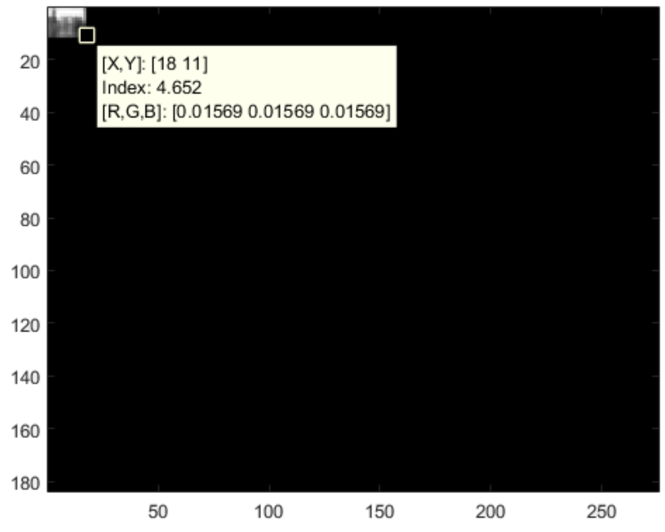
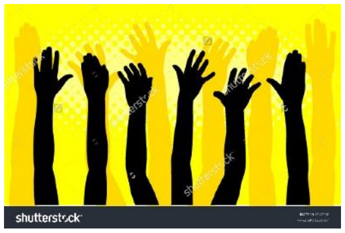
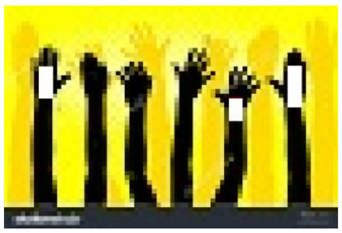


fig 5: level 4 image *fig 6: level 5 image*

CS5320_G_hands

Here, we can take a simple image and test if the function detects the object in the image;

Using the image:  

We are able to detect 3 hands out of the 6 (image on the right)

The detected locations are shown in white instead of red, for print purpose

4. Data

On use of the optimized CS5320_hands function on the 4 images, with the same threshold factor of 0. 9975 for all, the results were as follows:

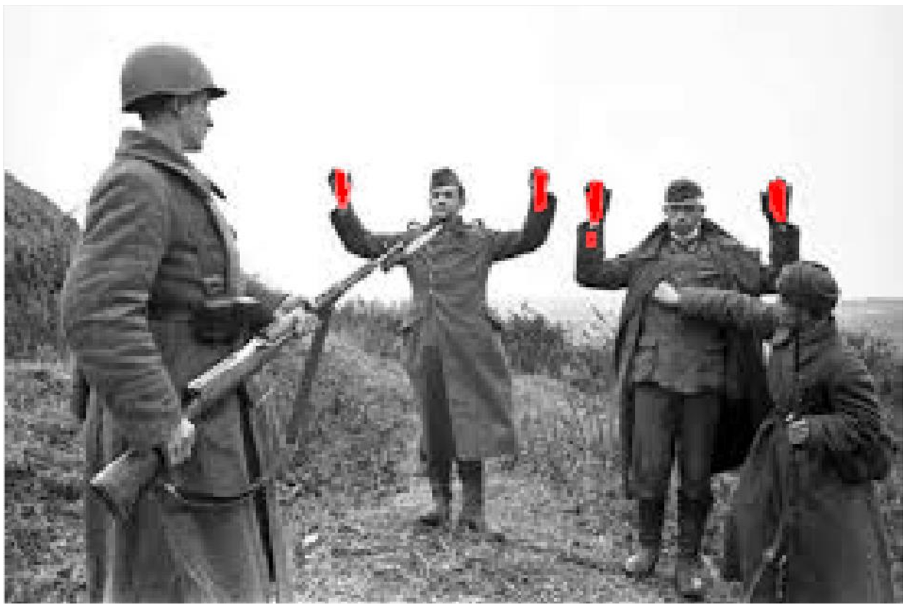


Fig: 's1.jpg' 4 hands detected

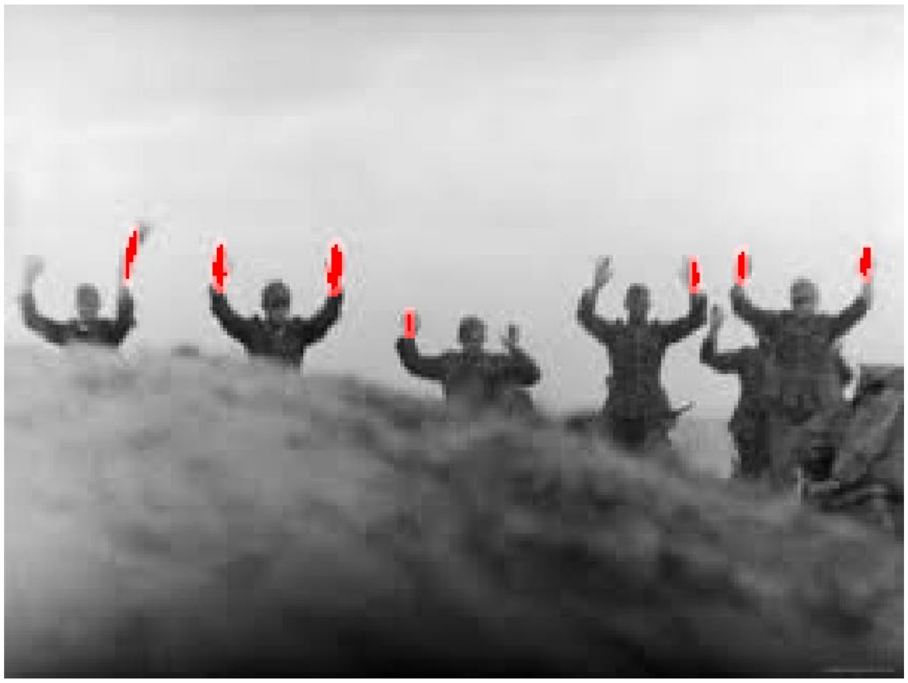


Fig: 's2.jpg' 7 hands detected

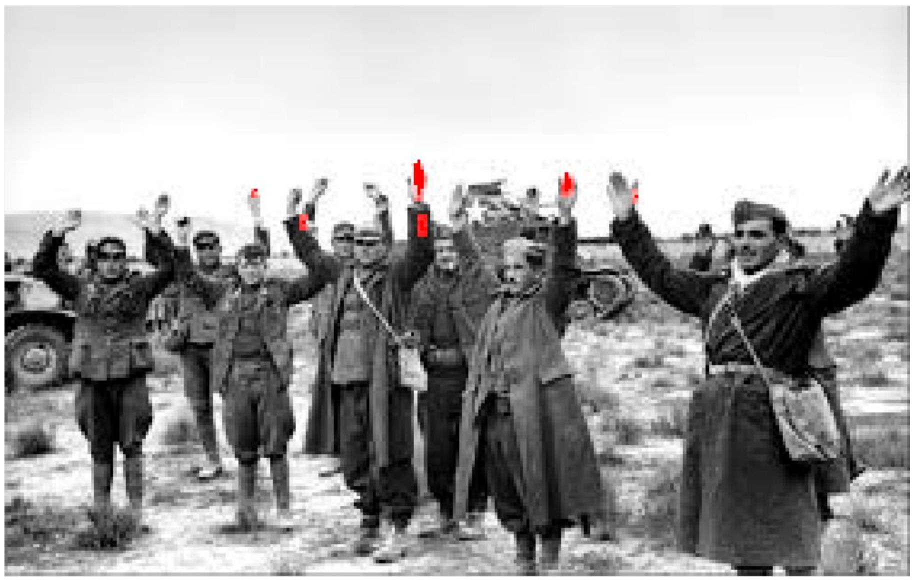


Fig: 's3.jpg' 5 hands detected

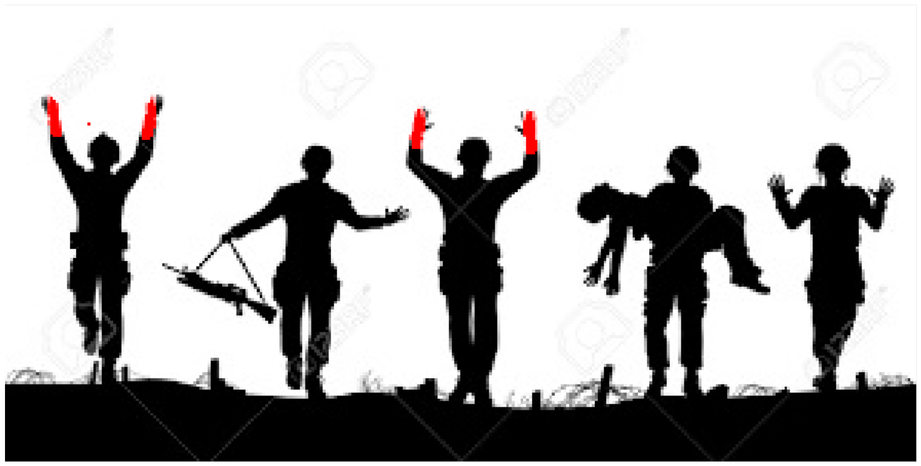


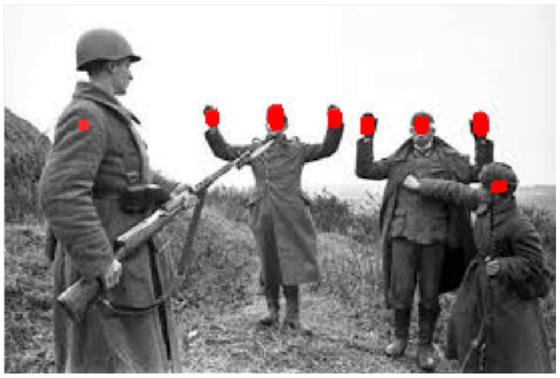
Fig: 's4.jpg' 4 hands detected

In the above case, the hands were detected without any other object being detected, except for s4 where an additional object was detected just above the head of the first Peron to the left and in the image s1, one hand was detected at two locations.

Optimization:

1. It was initially seen that the hands function would detect some regions from the sky. To minimize this error, only those regions were shown in the image whose grayscale pixel value was less than 150, since the sky region had a value above 150.

2. Also, since the average below a certain row did not have hands, only those detections were above in the image.



3. When the threshold was reduced, more hands were detected in the image along with other objects such the heads. To minimize this additional detection of heads, the white region width in the template was reduced so that it will mirror the hands more than the head which has a higher width.

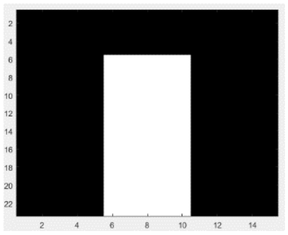
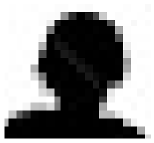


Figure: before reducing template width

4. Also it was seen that the heads were detected. This error was reduced by using a template of the head like the one given on the right. But this method only worked for image s4 which had its head shape of similar pattern while other images included heads of different shapes. Hence this was not used for the general hands function.



5. When the threshold at the various images was kept at different levels, for a particular threshold an image gave better results. Particularly for image s3.jpg wherein 11 hands were detected.

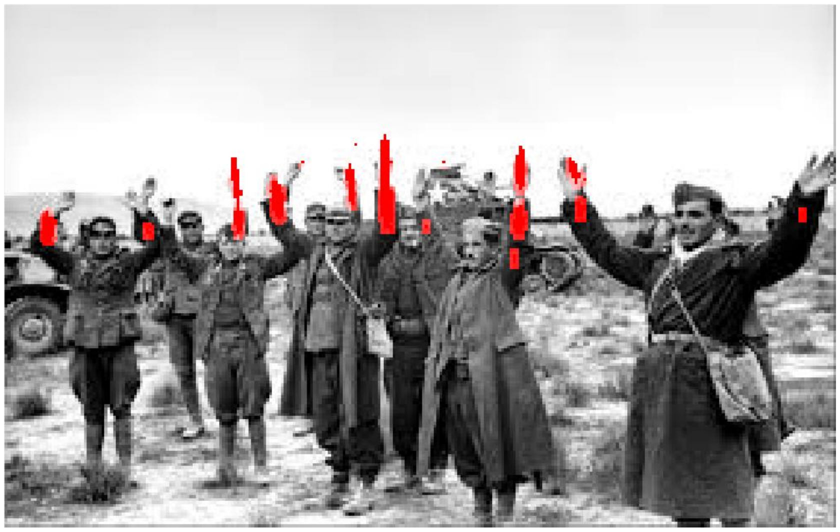


Fig: 's4.jpg' 11 hands detected with threshold factor of 0.983

6. Also, instead of applying the laplacian of Gaussian filter to the image at the initial stage of the hands function, gradient of the image was also taken to seen how the performance would vary. It was seen that the performance was similar.

5. Analysis

The confusion matrix can be used for analysis. To check the optimization of the function, let us use precision and recall.

Precision: How many detected hands are relevant

$$\text{Precision} = \frac{hands\ in\ image \cap hands\ found}{hands\ retreived}$$

Recall: how many relevant hands are detected

$$\text{Recall} = \frac{hands\ in\ image \cap hands\ found}{hands\ in\ image}$$

After optimization, with correlation threshold factor of 0.9975:

optimum performance		s1.jpg	s2.jpg	s3.jpg	s4.jpg
A	hands in image	4	11	18	7
B	hands found	4	7	5	5
A∩B	true hands found	4	7	5	4
A∩B/B	precision %	100%	100%	100%	80%
A∩B/A	Recall %	100%	64%	28%	57%

Average Computational time = 1.32

decrease in C threshold from 0.9975 to 0.983		s1.jpg	s2.jpg	s3.jpg	s4.jpg
A	hands in image	4	11	18	7
B	hands found	11	19	13	11
A∩B	true hands found	4	9	10	5
A∩B/B	precision %	36%	47%	77%	45%
A∩B/A	Recall %	100%	82%	56%	71%

Average Computational time = 1.325

Increase in Gaussian pyramid sigma to 2		s1.jpg	s2.jpg	s3.jpg	s4.jpg
A	hands in image	4	11	18	7
B	hands found	6	8	4	6
A∩B	true hands found	4	7	4	4
A∩B/B	precision %	67%	88%	100%	67%
A∩B/A	Recall %	100%	64%	22%	57%

Average Computational time = 1.3145

Decrease in laplacian of Gaussian from 7 to 3		s1.jpg	s2.jpg	s3.jpg	s4.jpg
A	hands in image	4	11	18	7
B	hands found	7	8	3	4
A∩B	true hands found	4	8	2	4
A∩B/B	precision %	57%	100%	67%	100%
A∩B/A	Recall %	100%	73%	11%	57%

Average Computational time = 1.3145

6. Interpretation

It can be seen that with the increase in correlation threshold, the success rate drops, but the precision improves.

With increase in Gaussian pyramid sigma, precision drops and success rate remains stable. Similar is the result with decrease of laplacian of Gaussian window.

Using Subsampled images instead of the original image for the hands function improves computational time but the results were good only in the first and 2nd level images after the second level pyramid image, less hands were detected.

7. Critique

It can be seen that a single hand was detected at multiple locations along the hand (along vertical direction). Here, an average of the y coordinates could be taken to show the hand at particular position.

I could have had used a rectangular box surrounding the detected hand instead of showing the hand in red color.

I should have had noted down many optimization techniques (although minor) which I have used while typing the code, although I have mentioned few in the report.

Instead of using the laplacian of Gaussian filter onto the image in the hands function before correlation, I could have used the edges in the image or else used gradient of the image.

It would be better if each image had its own correlation threshold.

Creating a generalized template to avoid detection of heads in the image could also be done.

8. Log

25 hours to code and to make report.