Computer Vision CS-GY 6643 - Project 1

Sumedh Parvatikar, sp7479@nyu.edu

1 Histogram Equalisation

(a) Brief introduction of histogram of an image and what it does

> *Histogram of an image:* Histogram of an image comprises of a frequency count of the different intensities grouped by number of bins. Histograms help us analyze the distribution of the intensities and assist in further image processing. It helps us understand how we can proceed with image processing, for instance in the below case, we use histogram to understand the distribution and then process the image to brighten the dark pixels allowing us to capture more information. In addition, they also help us to identify number of peaks, uniformity allowing us to conclude about the brightness and contrast properties of the image.

(b) Implement the following functions: def create pdf(im in) which returns the pdf, def create cdf(pdf) which returns the cdf and def histogram equalization(im in) which returns the equalized image utilizing your previously implemented functions of pdf and cdf

> The code is implemented as part of the colab notebook and can be found there. I'll explain the reasoning for the approach.

- `def create_pdf(im_in)`
  For this particular function, we create a hashmap and count the intensity for each pixel in the image. We then return the image
- `def create_cdf(pdf)`
  We use the pdf and process it further to create a cumulative distribution. To perform this we keep updating the running sum of the pdf intensities starting from 0. As the pdf's retrieved are probabilities, we multiply with 255 in the end to stretch the range from 0 to 1 to 0 to 255. We then return the value.
- `def histogram_equalization(im_in)`
  We perform histogram equalization by utilizing the previously created functions. We first create a pdf which returns the frequency count for 256 intensities. We then divide the frequency count with the image size to convert it into a probability of pixels in that particular intensity. We then compute the cdf by using the function. We then multiply with 255 to increase the range from 0 to 1 to 0 to 255. We then round it to integer for better computation. Now, we replace the intensities of the old image with the processed cdf. This ensures that the intensity is uniform and distributed.

> The histogram equalized image is uploaded as part of figure 1b

(c) Plot pdf and cdf before and after histogram equalization. Plot them side-by-side for easy comparison.

> The graph is plot as part of figure 1c.

(d) Provide a comparative discussion of the PDF and CDF plots, and relate them to the image's contrast and intensity.

> We can observe from the plot above that the pdf and cdf of the image prior histogram equalization are different indicating that the image has changed. We can validate that through viewing the new image where we can effectively see that the man is sitting on a chair.*

> Before Histogram equalization : We can notice that the major PDF distribution of the intensities is mostly on the lower intensities highlighting that the overall image has lot of dark pixels making the image collectively dark. And because of this, in the cumulative distribution, we observe that we reach value 1 (because we are using probability) very early in the intensity.*

> After histogram equalization : We can notice from the PDF plot that the intensities are more distributed and spread out compared to prior processing indicating that we have eliminated some

Figure 1b

dark pixels. We also notice a uniform increase in the cumulative distribution when compare to the prior graph where we had many dark pixels. In this particular scenario, we have reduced the dark pixel intensities thus allowing us to notice some of the dark pixels roughly which were not evident early.

*The person is sitting on a revolving chair **

2 Image Thresholding

(a) Show histogram of the image.

> The histogram of the image can viewed as part of Figure 2a.

(b) Implement and apply Otsu's thresholding and generate a plot of the inter-class variance as a function of the chosen threshold (i.e., x-axis with each possible threshold from 0-255, y-axis with the resulting variance) and highlight the threshold value where it is maximised.

> Otsu's algorithm is given by

$$\sigma_v^2 = W_b * W_f(\mu_b - \mu_f)^2 \tag{1}$$

where W_b = No of pixels in the background; W_f = No of pixels in the foreground; μ_b = Mean intensity of the background and μ_f = Mean intensity of the foreground

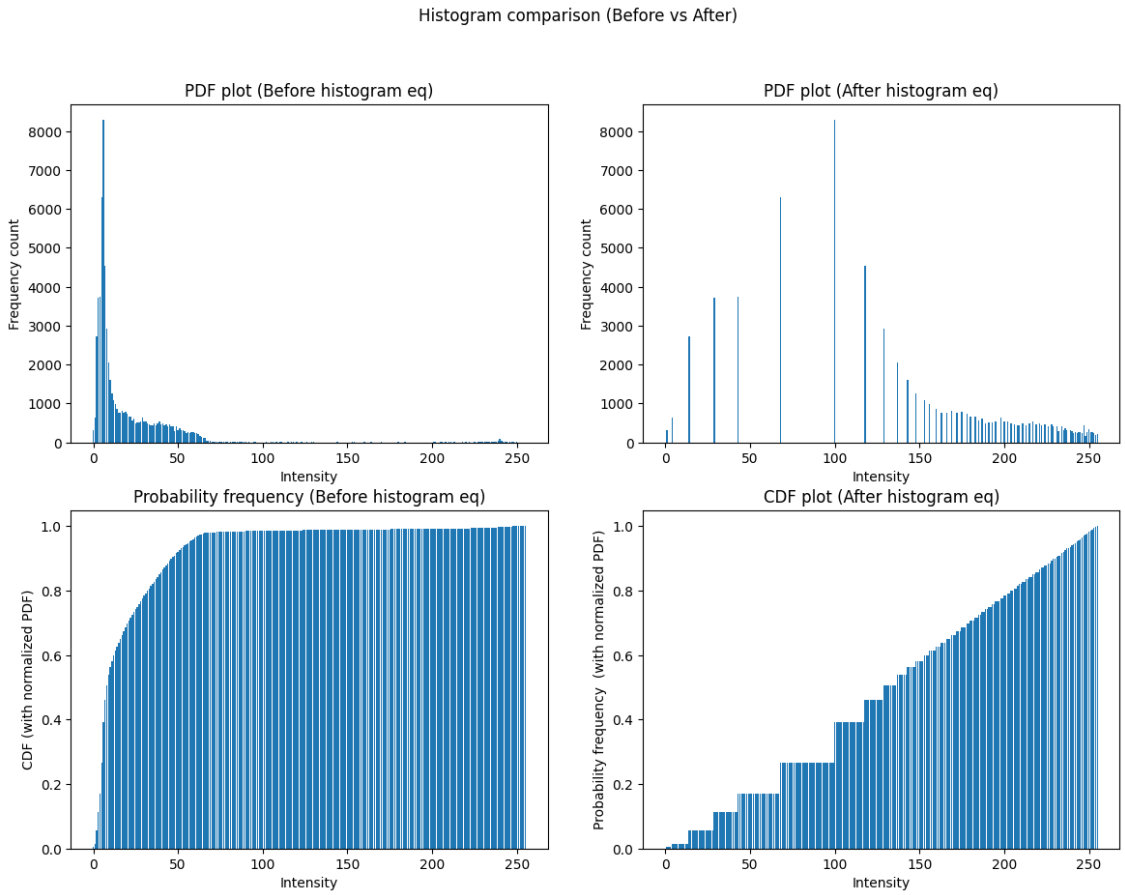

Figure 1c

We use the above formula to calculate the variance across all the possible thresholds. We do this from 0 to 255 and calculate the the variance for each threshold value. After performing computation for every threshold value, we then find the threshold with the maximum variance which will be an optimal value.

From figure 2b, we can notice that the threshold value computed is 85 which has the maximum variance. We use this threshold variance to filter the image and get the new image.

(c) Show the binary image after thresholding and discuss the results. Explain why do you think the results are they way they are.

> Otsu's threshold algorithm automatically finds an optimal threshold intensity by computing variance across all the possible threshold and using the one which has maximized variance across the image. In the above output, we are able to deduce cell borders only in the middle ignoring the cells in the darker background, because of the threshold value. This is not the most optimal result generated using the Otsu's algorithm.

Otsu's algortihm does not work well when the object intensities of some of the cells are dim characterizing them as a dark pixel after thresholding. As the algorithm is known to perform well as a global thresholding algorithm, but because of some local nuances in intensities and illumination we ignore them. The algorithm is also known to work well with images having a bi-modal distribution, and our image does not have a bimodal distribution.

From figure 2c, we can observe new image generated after applying the algorithm. We can also clearly see the dimmer area being darkened because of the threshold.

(d) Implement and apply Niblack thresholding algorithm and show how changing the filter shape affect the results. Show the thresholded binary image with 3 examples of changing filter size

> Niblack algorithm is given by

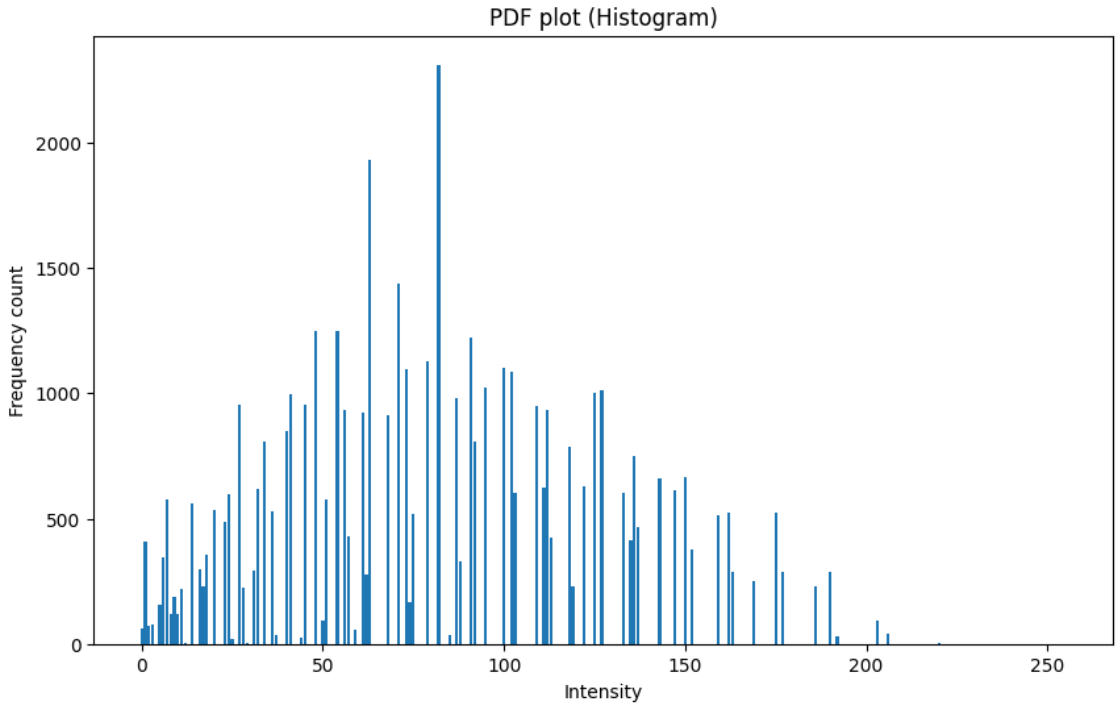

Figure 2a

$$t = \mu + k * \sigma_{std} \tag{2}$$

where μ = mean of the neighborhood; σ_{std} = standard deviation of neighborhood and k = constant value, default to -0.2.

This algorithm works on local thresholding i.e., based on a window size provided. This window size determines how the threshold is calculated for a locality. We initially pad the image to cover the edge pixels and perform computation with a particular window size. Once the threshold is calculated, we compare whether the central pixel value and update the pixel value to either 255 or 0. We perform the Niblack algorithm on the same TEM image but with various window sizes. We perform the algorithm for window sizes [5, 15, 21, 33, 45] and the results can be seen as part of figure 2d. The images are generated in the same order as the window sizes mentioned earlier.

In the images, We can notice the borders of the image in all the outputs above. We can infer that as we increase the window sizes, we incorporate more data to compute the local thresholding allowing us to capture the borders more effectively We can observe the image becoming more clearer with less noise.

(e) Comparatively discuss the two thresholding based on the results you got.

> The two thresholding algorithms have their own strengths. Otsu's algorithm known as a global thresholding algorithm determines the threshold by accounting the entire image and calculating a threshold value. But sometimes, the areas of the image which are dim are ignored because of this and create problems when retrieving key information from the image. This can be observed from the image output before the Niblack algorithm section.

On the other hand, Niblack algorithm performs thresholding by only accounting for local thresholding thus adapting and identifying information from the dim regions of the images. The window sizes can be adjusted according to the requirement and from the results above we can notice that Niblack algorithm performs better for this particular image because of that reason.

3 Template Matching

(a) Implement cross-correlation with the original image and the template as mask.

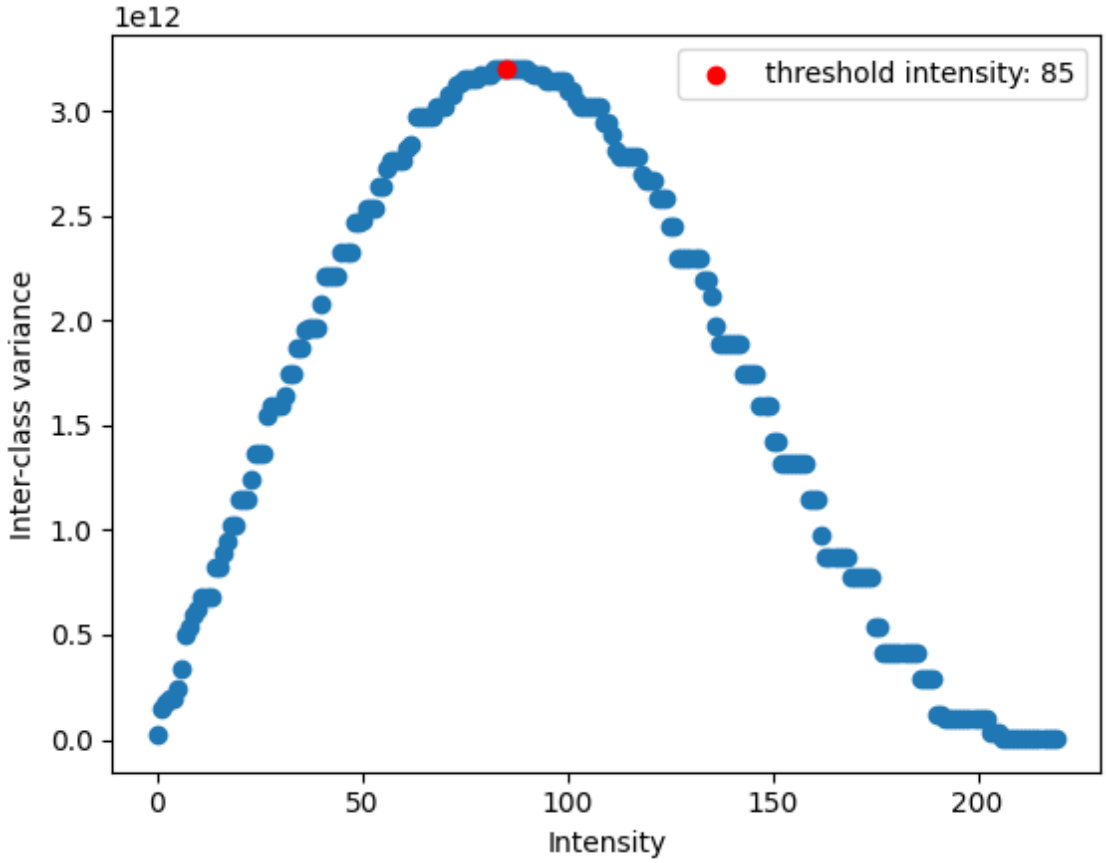

Figure 2b

> The logic for template matching is implemented as part of

```
def template_matching(im_in, template)
```

Please look at the code implementation for more details. The way we perform template matching using cross-correlation is to use the template image and calculate a similarity score which is the cross correlation and then use the best score as the match for the template image. As a note, before we start the scanning, to avoid noisy brightness and illumination, we perform 0-mean on the image as well as the template to get a normalized intensities.

(b) Parse the correlation image to detect the maximum peak value and its (x,y) location. You may mark this location with an overlay of a circle or just manually painting an arrow or else. Discuss if this location matches your expectations. Discuss the appearance and cause of other peaks, and how they compare to the global peak, you may just manually check other peaks and annotate them on the peak image.

> Yes the peak matches the expectations from the template image. From the above image, after thresholding for top 5 correlation values, we see a white dot mark close to the center in the image highlighting there is a match. Note: The top 5 correlation values were all near each other and hence we see only one point with a higher intensity. Because there was only a single match, and hence we see only one white dot in the image. If we reduce the threshold value, and there are good matches, we will see more white dots in the image as we reduce the threshold indicating the close matches with the template image.

In the second image where we have used a different threshold (not the highest peak), we can see so many peaks implying that there are so many potential matches with that threshold value. But with the best threshold value, we do not see this. The best location is also marked in the original image with a black dot. All these details can be seen as part of Figure 3b.

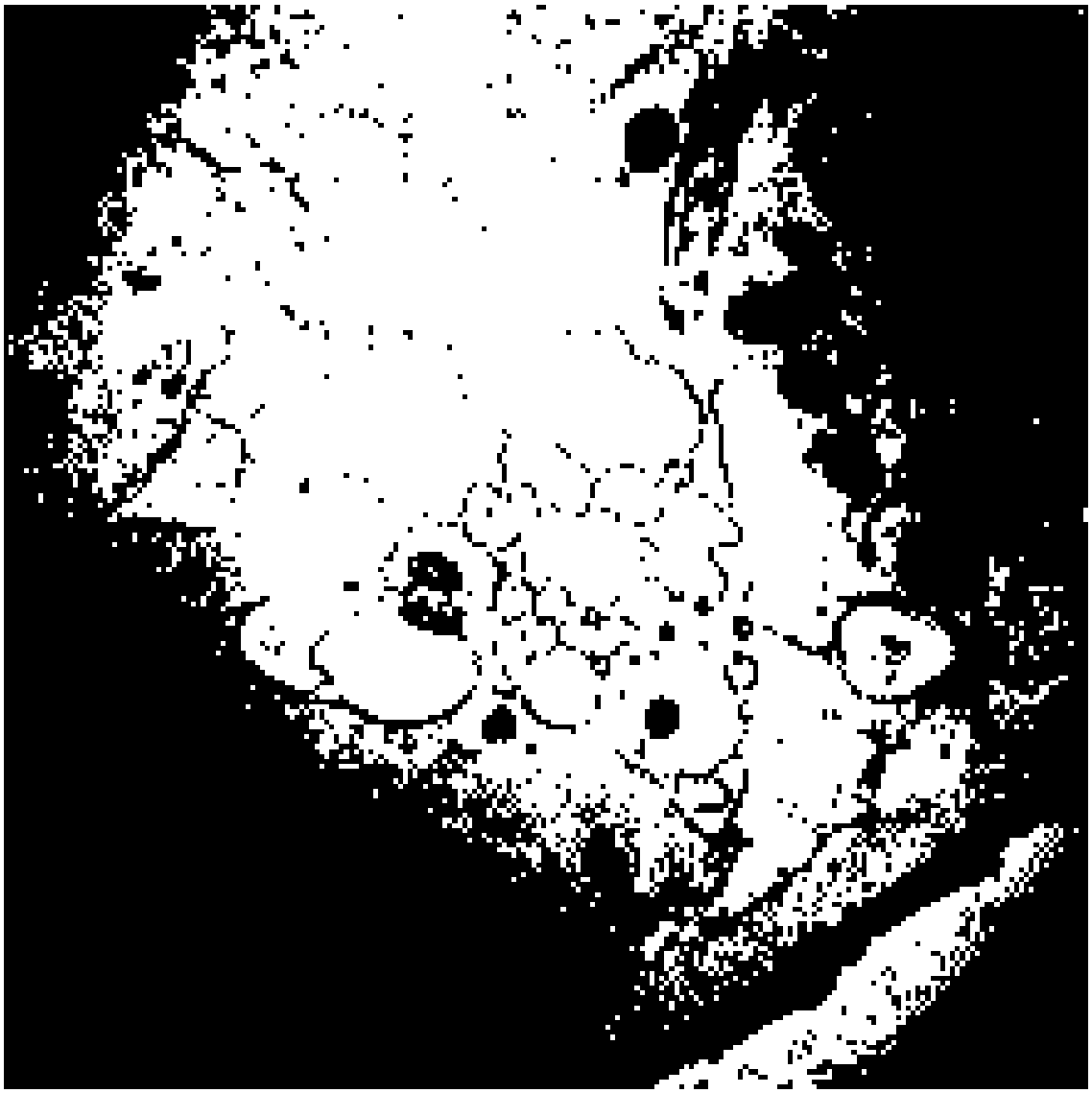

Figure 2c

(c) Show the image which helps you identify the peaks (cross-correlation output) with a mark on the highest peak

> The image which helps us identify the peaks is displayed as part of Figure 3c.

(d) Overlay the found peak on the original image

> The match on the original image is displayed as part of Figure 3d. There is a rectangle drawn at the match location.

4 Creative Section

(a) Write code to find Waldo in the image. Make your code generic enough, such that it accepts two images(original image and template image) and returns a bounding box around the matched template on the original image.

> The code can be found as part of colab notebook. The output of Waldo can be found as part of Figure 4. The time taken to execute and return the results was around 3.4 microseconds on an average of 5 runs. The previous approach for template matching takes O(n*m) time complexity as it has to look for image at every location, where as in the new approach, we apply fourier transform into the frequency domain which takes O(n*logn) time complexity. We utilize a property of Fourier Transform and convolution theorem that convolution in the spatial domain is equivalent to the

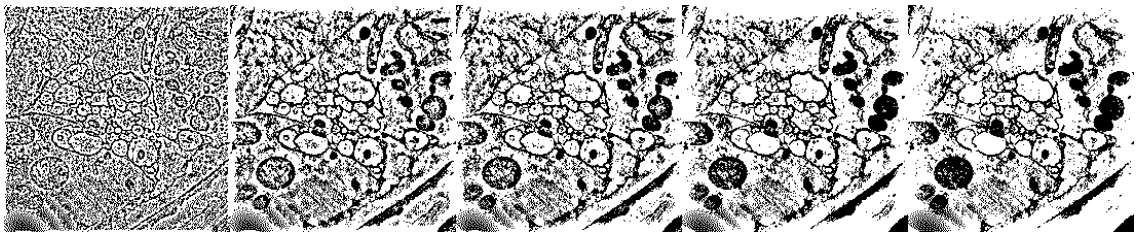

Figure 2d

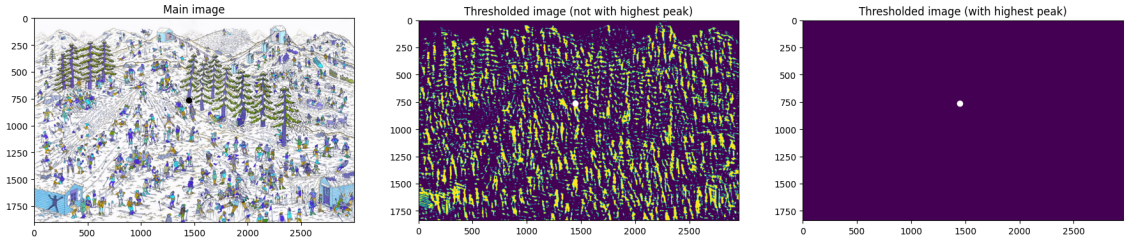

Figure 3b

multiplication in the frequency domain. This paves way for faster execution especially in larger images. To begin with, Firstly, we normalize the main image and the template image, then we convert the images into frequency domain using fourier transform. When converting using fourier transform, we need to maintain the same shape. So we pad the template image to maintain the shape of the original image. Once we get the transformed images, we multiply the main image in frequency domain with the conjugate of the transformed template image to get the correlation matrix. We then perform inverse operation thus giving us the correlation matrix in spatial form. In the next step, we take the absolute values to avoid imaginary numbers. We perform these steps for all the 3 channels of the image and normalize the correlation values among the images. Finally, we iterate to find the location for the maximum correlation and use that to generate the rectangular box in the image.

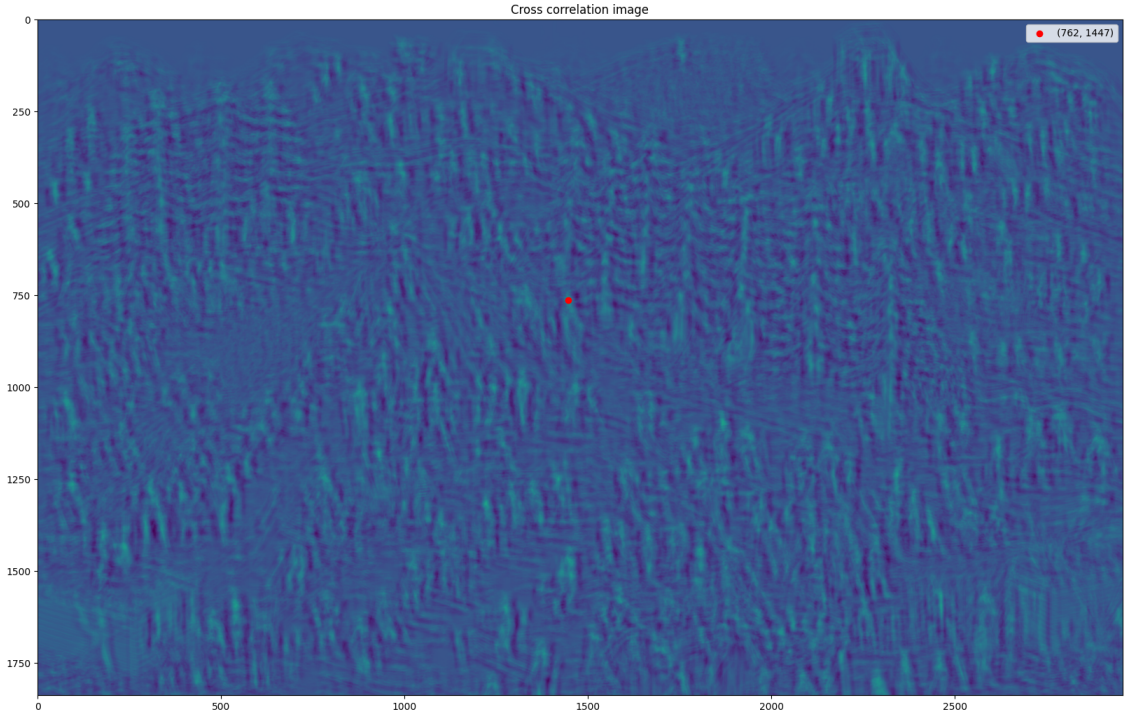

Figure 3c

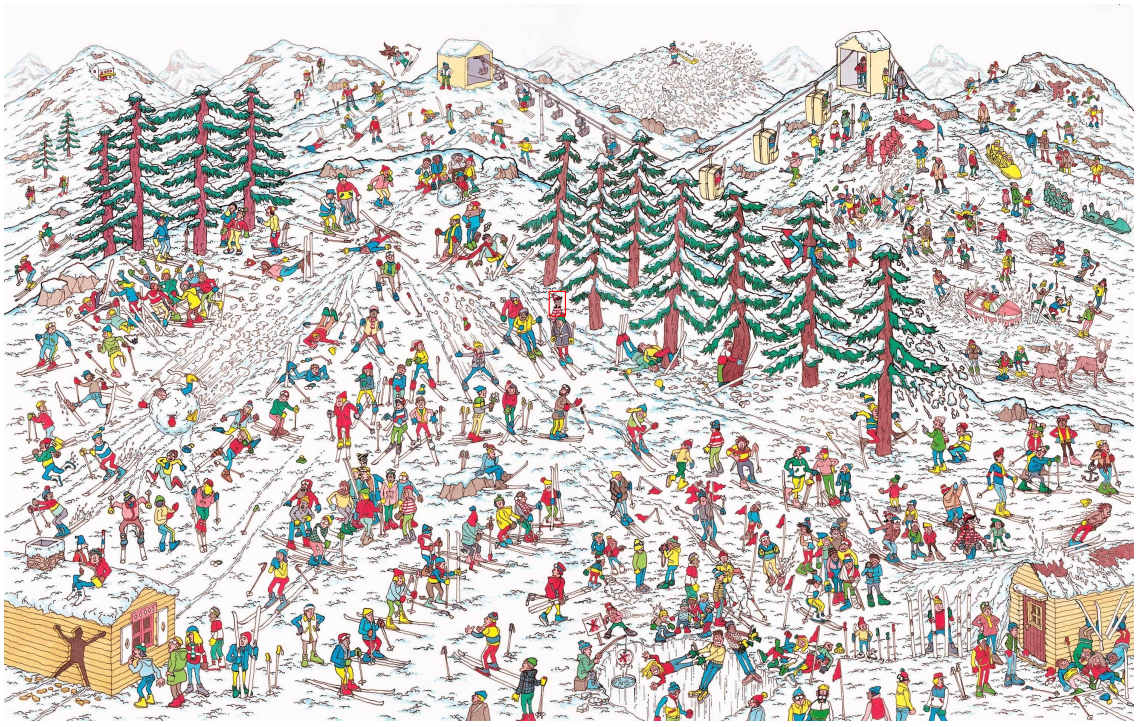

Figure 3d

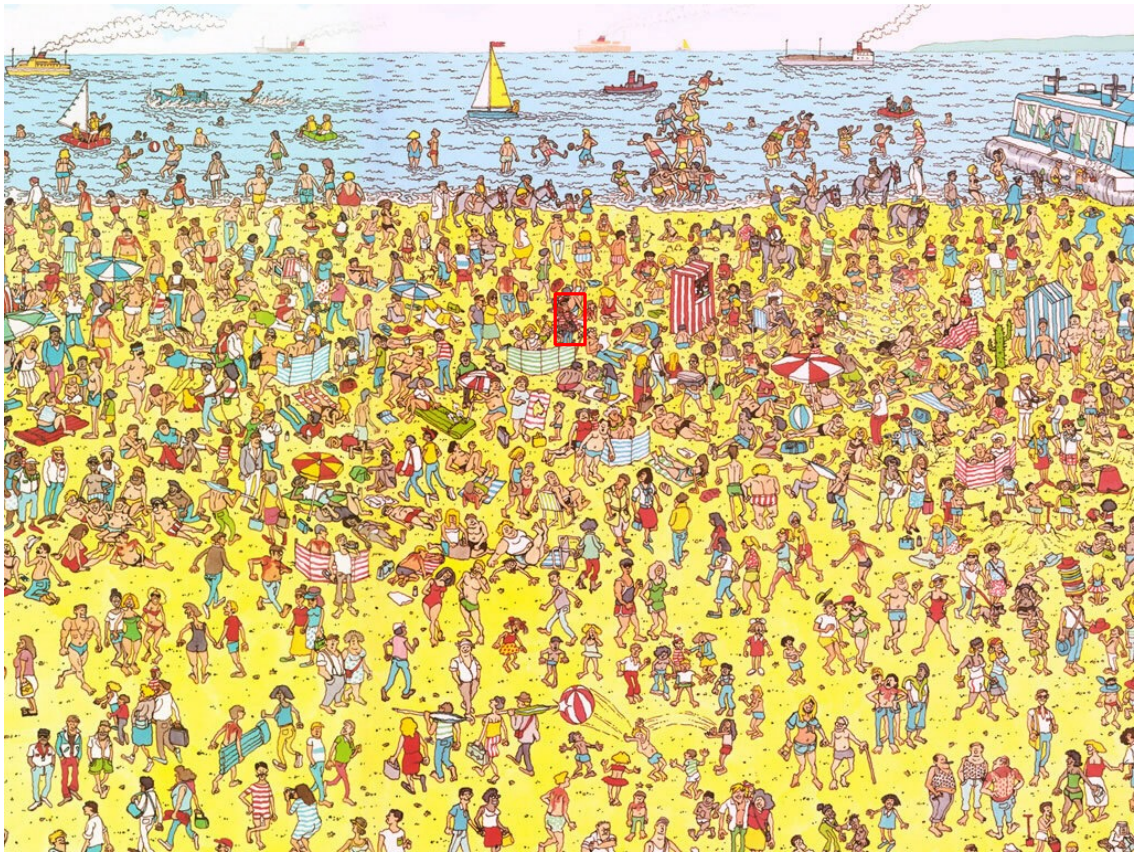

Figure 4