Assignment 5

Samih Warwar, 324888155 – Merry Shalabi, 324007202

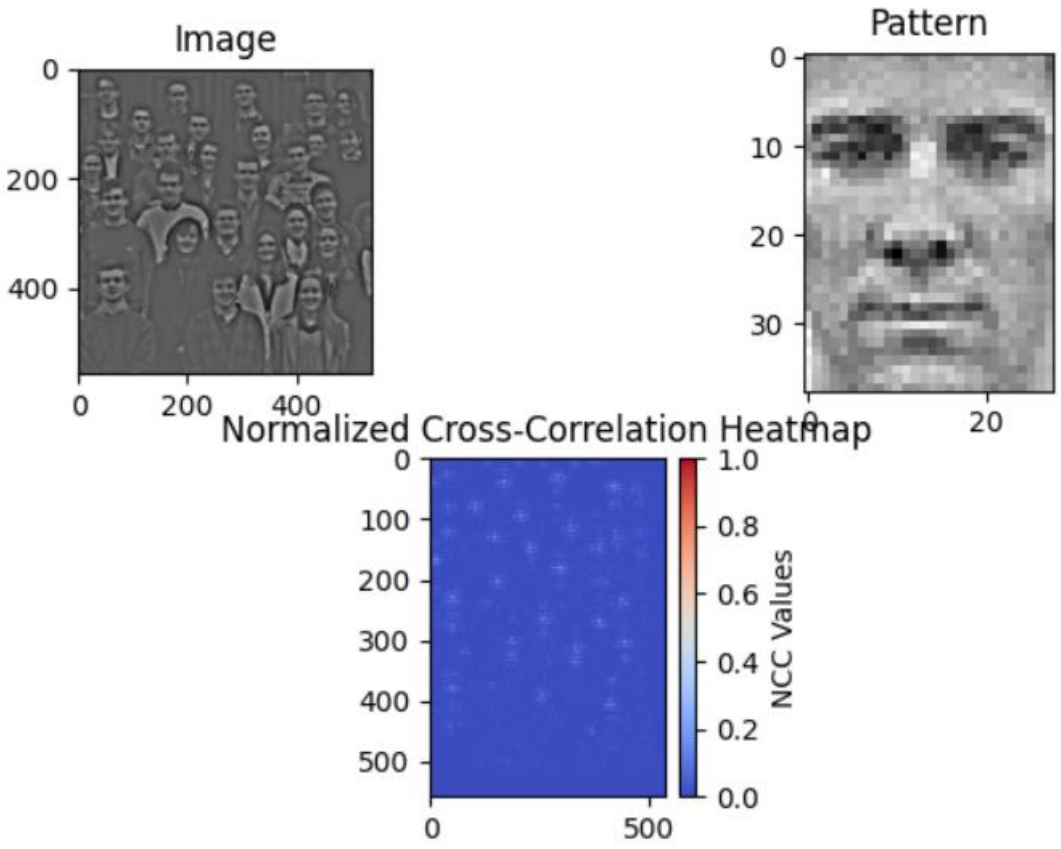
## Problem 1:

First, we started implementing the scale up and down functions using Furrier transformation. To scale up an image we padded the Furrier image equally from all edges in a way that new colls = colls*resize_ratio and new rows= rows*resize_ratio and then transformed it back. In the same way to scale down an image we move to the Furrier version of the image and take the center in a way that new colls = colls*resize_ratio and new rows= rows*resize_ratio and then transformed it back.

Next, we had to implement the NCC function that calculates the normalized cross-correlation between an image and a pattern we move using a window size of the pattern all over the image and calculate the normalized cross correlation using the equation from the lecture and save them in an array.

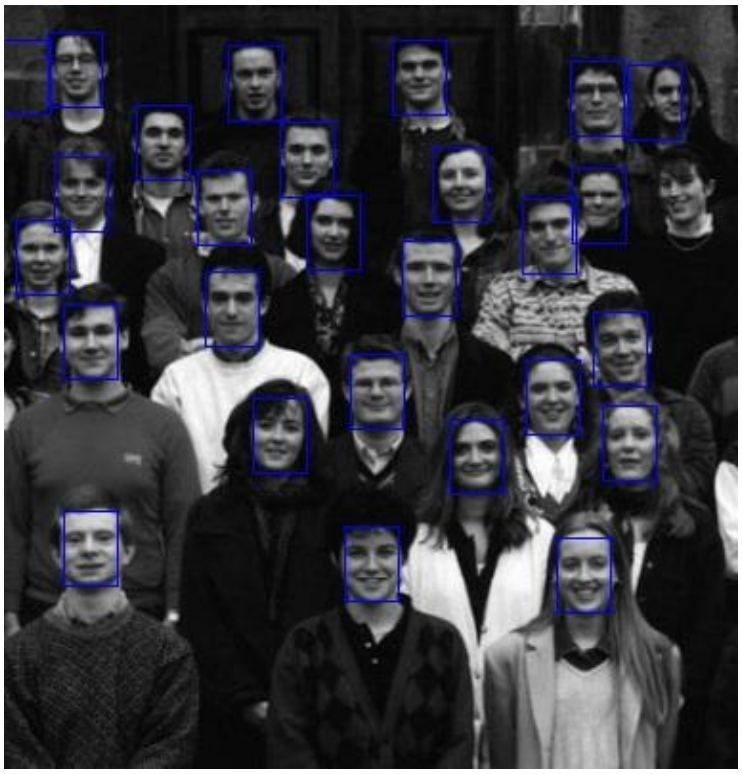
Once we had these functions in addition to the display and draw matches functions, we started to try parameters to get the best result to detect the faces in the “students” image. Eventually we decided to scale up the image and scale down the pattern to match the size of the faces in the image so it will reduce False positives thus making the matching process more precise and scaling up the image is done to ensure that the features of the pattern are not lost during matching, in top of that we saw that applying sharpening filter on the image and the pattern helps in localizing the object more accurately.

An example of the heat map we got of a good result:



After deciding the threshold of the NCC after choosing different resizing options and using maximum filter to eliminate the duplicated squares we got three good results:

1.One missing face and one False Positive:

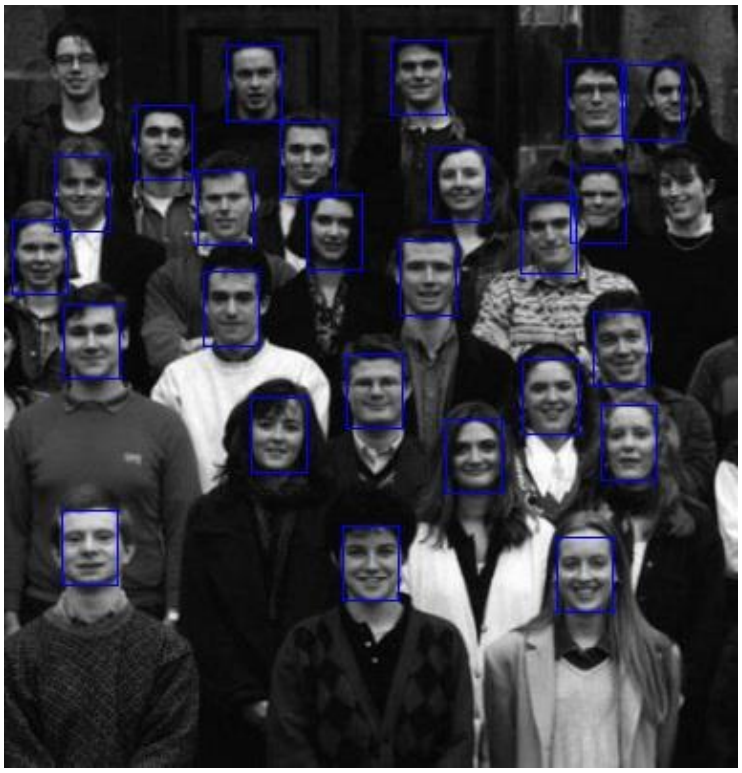


image_scaling=1.4

pattern_scaling=0.78

threshold=0.36

2. Two missing faces and no False Positives:

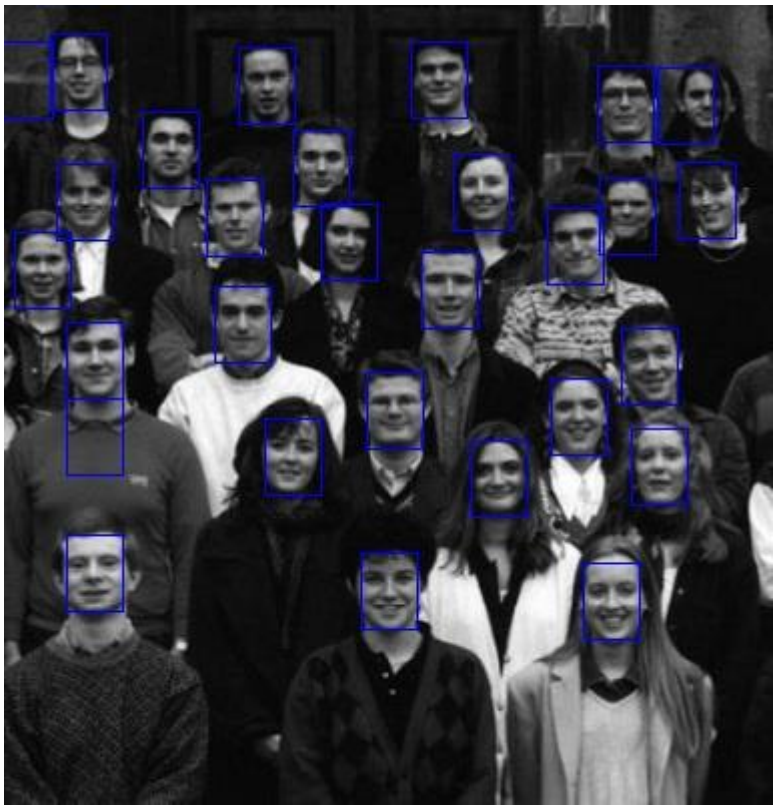


image_scaling=1.4

pattern_scaling=0.79

threshold=0.37

3. Found all the faces but we got two False Positives:



image_scaling=1.4

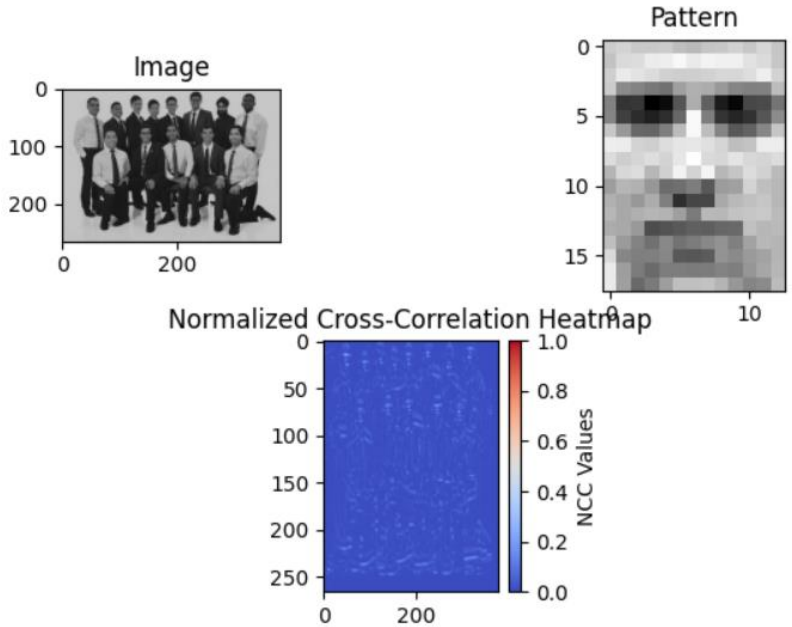
pattern_scaling=0.77

threshold=0.335

---

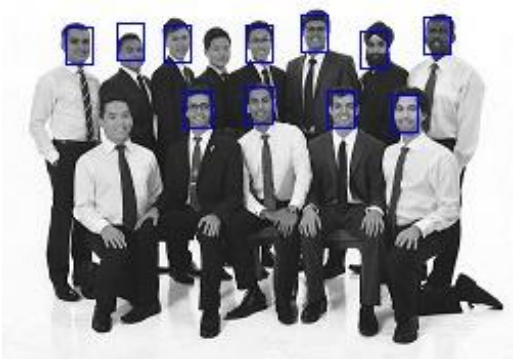
Now we move to the second image "**crew**" image:

Like the previous image we also scaled up the image and scaled down the pattern for the same reasons but here we used other parameters due to the difference of the size in the image itself and the faces, we got heatmap like:



After deciding the threshold of the NCC after choosing different resizing options and using maximum filter to eliminate the duplicated squares we got two good results:

1. two missing faces:

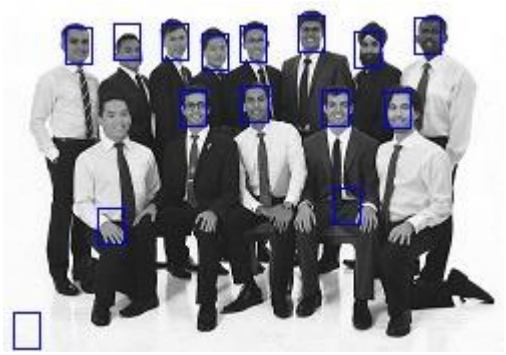


image_scaling=1.52

pattern_scaling=0.36

threshold=0.44

2. one missing face and three False Positives:



mage_scaling=1.52

pattern_scaling=0.36

threshold=0.44

---

---

## Problem 2:

To build the Laplacian pyramid first we chose to build the Gaussian pyramid by using a helper function my_pyrDown(image).

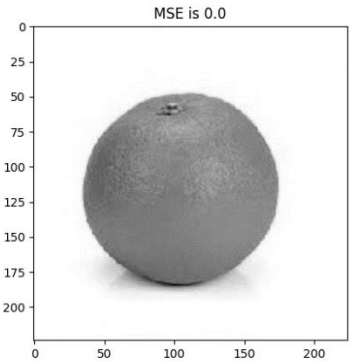
My_pyrDown: we applied gaussian blur to the image we get and then down sample it by taking every 2nd pixel in each row and column.

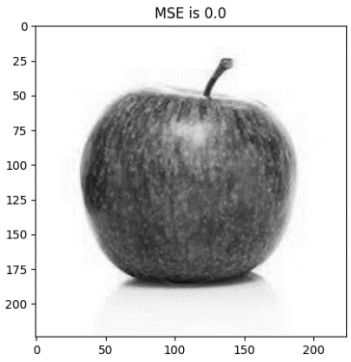
get_gaussian_pyramid: we called My_pyrDown the numbers of levels we want the pyramid to be.

Then implemented **get_laplacian_pyramid** function by getting the gaussian pyramid and going over each 'i' from 0 to level-1 and subtract gaussian_pyramid[i] from the expanded version of the gaussian_pyramid[i + 1] leaving behind the high-frequency components or details.

To restore an image from Laplacian pyramid we moved from top to bottom and added the expanded version of the level itself to the next level and then expand the result and add it to the next level till the end.

To check if we did the build and the restoration correct, we tested it on the orange and the apple images and got MSE=0:




Now to blend the two images we will blend the two Laplacian pyramids (orange , apple) by moving on each level on both pyramids and build a new pyramid according to the following:

```
For each level i do:
    initiate a mask with 0 and size of the current level
    put 1 on the first "cols // 2 - int((levels-i)*1.5)"
    # here we have first couple colls of the mask =1
    # and last couple colls of the mask =0
    # now we move to fill the midlle using the givving function
    # where we start at 0.9 and end with 0
    for c in range(cols // 2 - int((levels-i)*1.5),
                   cols // 2 + int((levels-i)*1.5)+1):
        mask[:, c] = 0.9 - 0.9 * (c-(cols // 2 - int((levels-
            i)*1.5)))/ (2 * (levels-i))
        blended_level = laplacian_pyramid_orange[i] *
                (1-mask) + laplacian_pyramid_apple[i] * mask
```

Eventually we got oranpple:

With

MSE with original apple image: 35.50673628826531

MSE with original orange image: 37.04087611607143

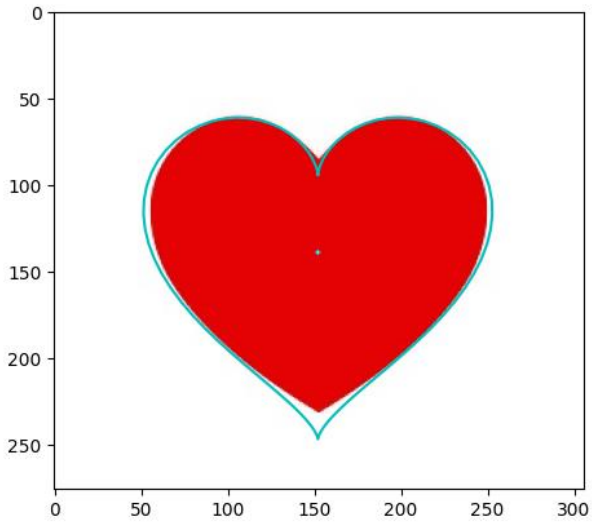
## Bonus:

First, we started by defining the parametric_x and y according to the giving equation. Then using the find_hough_shape function we can get the possible heart matches. for each edge point, it calculates the corresponding center point (x_center, y_center) for each shape candidate and updates the accumulator by incrementing the vote count, then it sorts the accumulator based on the number of votes in descending order and according to the giving threshold. For each candidate shape in the sorted accumulator, If the percentage of votes exceeds the bin threshold, it adds the shape to out_shapes. Then it removes nearby duplicate shapes and finally draws the detected heart shapes on the output image using the parametric equations to generate points along the heart's contour.

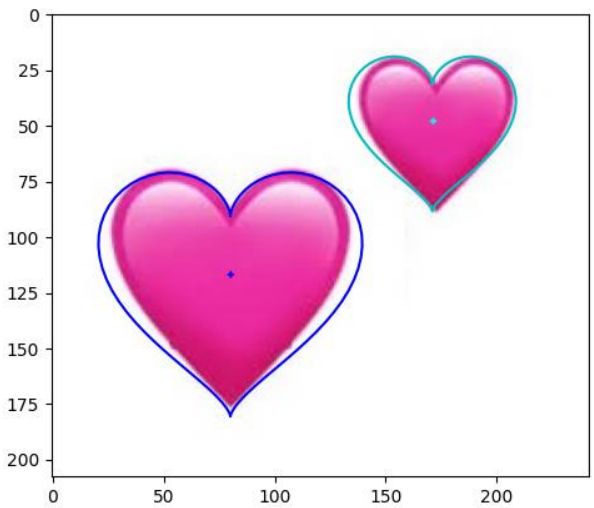
Now for all images we built the edge version and passed it to the function in addition to the image itself and the r_max, r_min and threshold.

Simple: After matching the radius of the heart, we got 6.95 < r < 7 and the best

threshold= 0.15




Medium: 2.1 < r < 5.1  threshold=0.29



Hard: 2.8 < r < 11.5 threshold=0.2447