

# Homework 5:

## Question 1:

- For implementing the given function `scale_down(image, resize_ratio)` we first will blur the image so it will reduce the high frequency of it so it will give us better scaling down and then we calculate the Fourier transform of the image so we will crop the frequencies to scale it down:  
`fshifted_im[from_height:to_height, from_width:to_width]`  
 and at the end we will get the image back using the inverse  
 and we did `cv2.normalize` for the final image to get the correct range of pixel values.
- The same as scale down but now we will not do blur and we will put the frequency of the image on a larger scale.
- No we will implement the `ncc_2d` the same way as we saw in the lecture :

$$\frac{\sum_{x,y \in N} [(u+x, v+y) - \bar{u}_v] [P(x,y) - \bar{P}]}{\left[ \sum_{x,y \in N} [(u+x, v+y) - \bar{u}_v]^2 \sum_{x,y \in N} [P(x,y) - \bar{P}]^2 \right]^{1/2}}$$

We used the `np.lib.stride_tricks.sliding_window_view(image, pattern.shape)` to create all the possible windows from the image and then we did the same as in the Formula given also we payed attention the the cases denominator is 0 we made sure to get a zero value.

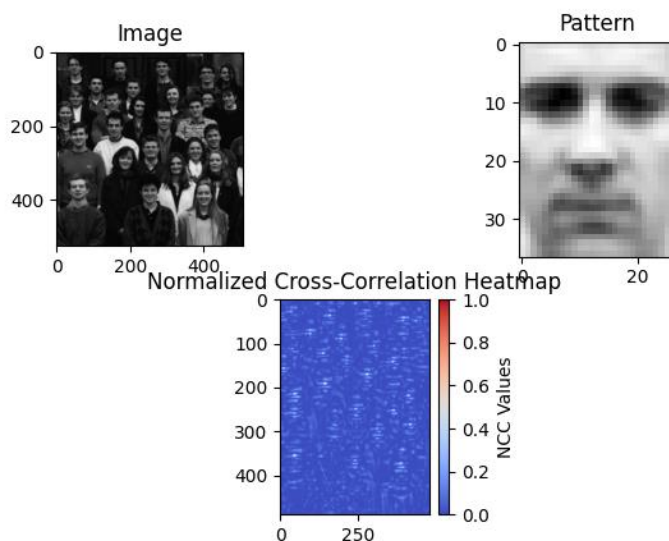
- +e

For the students picture we choose (we tried to choose values that matches the most faces at each picture did that in the painter app)

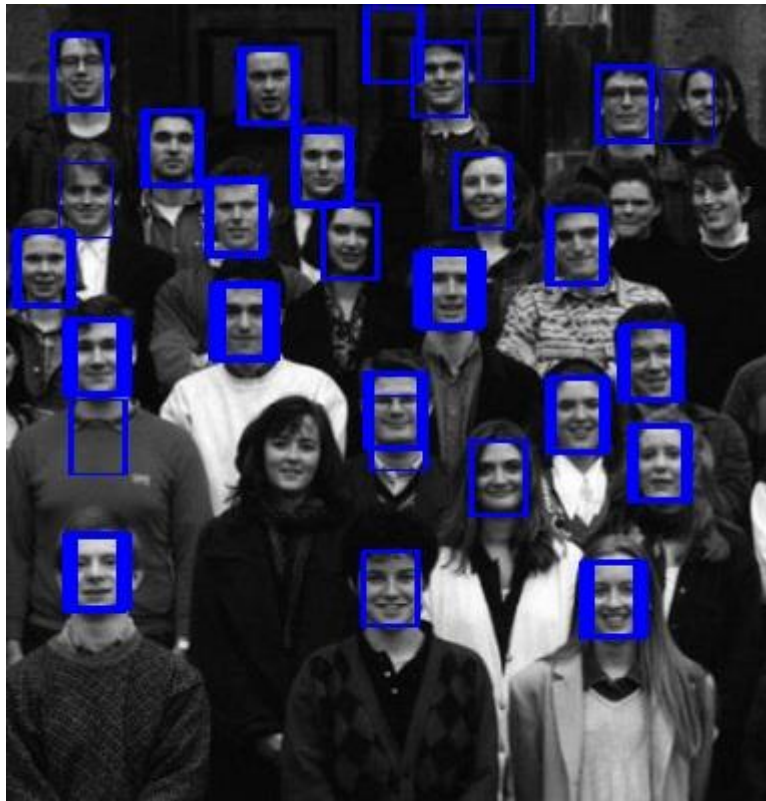
`image_scaled = scale_up(students_img, resize_ratio=1.32)`

`pattern_scaled = scale_down(pattern, resize_ratio=0.75)`

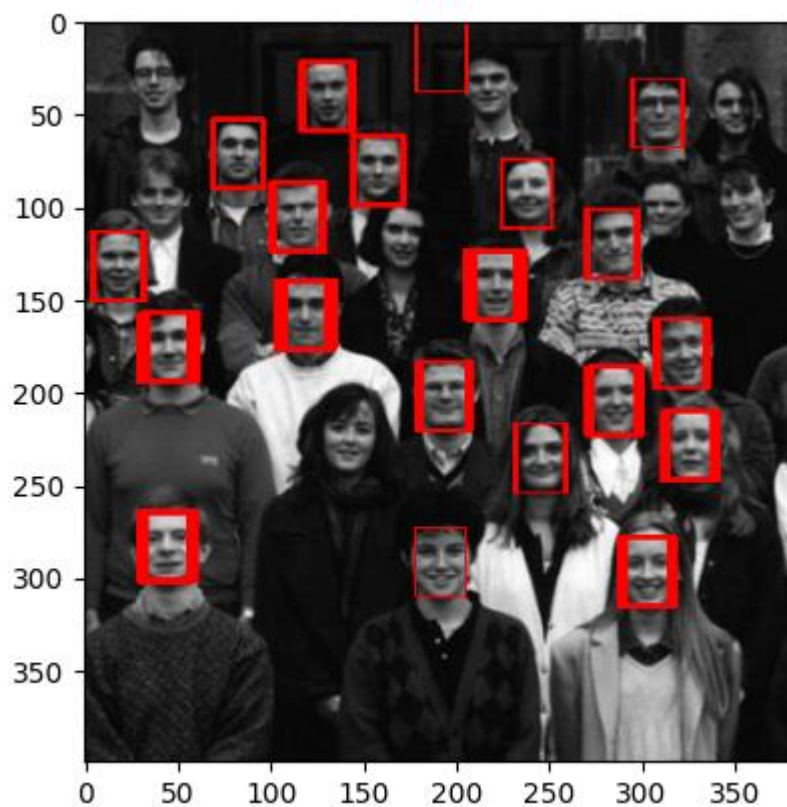
so we got this:



we clearly can see the point of matches and we got this result :



as we can see there is three false positive but we almost got all of the faces we can get ride of that by increasing the threshold a bit and we will get:



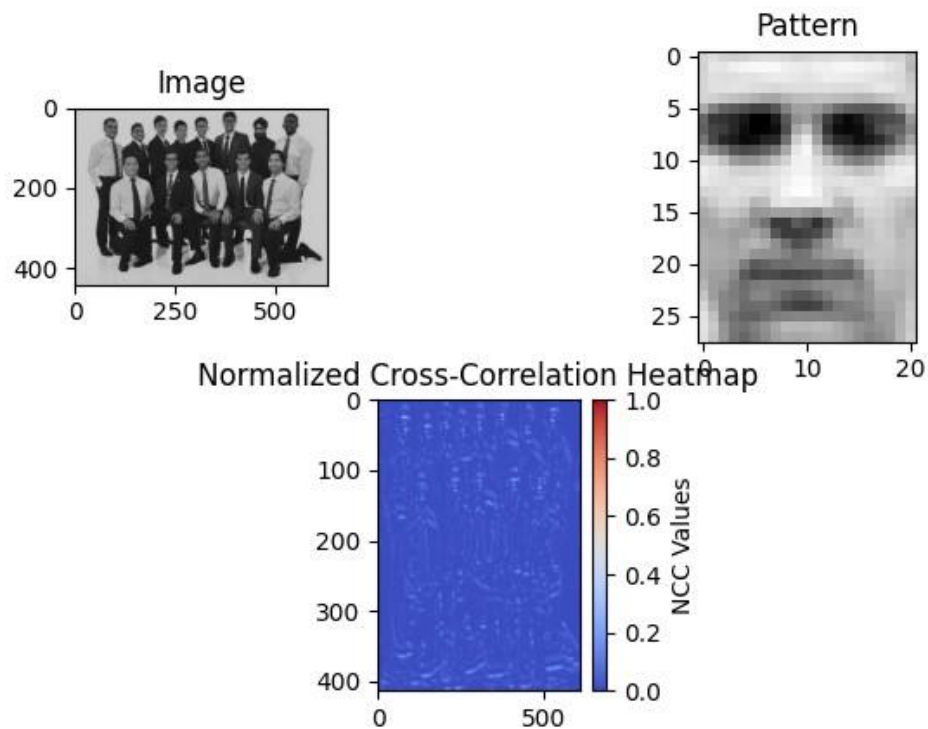
as we can see that we got less faces.

In the next picture the crew:

```
image_scaled = scale_up(crew_image, resize_ratio=2.52)
```

```
pattern_scaled = scale_down(pattern, resize_ratio=0.58)
```

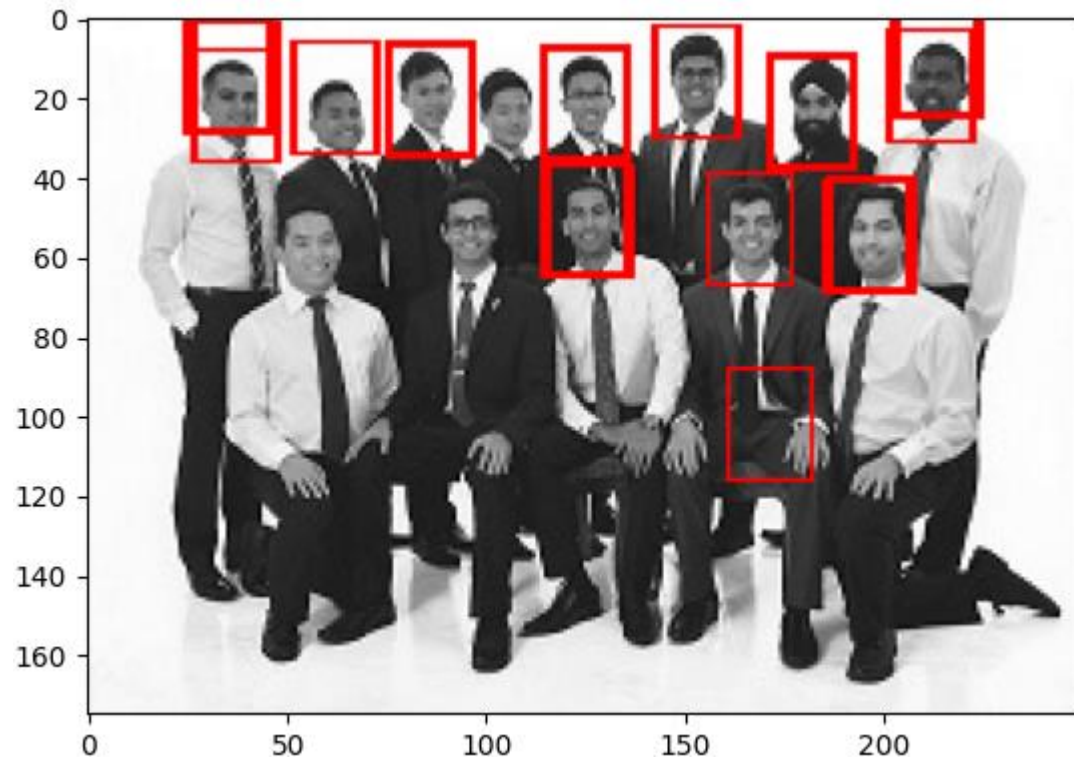
we used this values to scall the pictures we manage that using the painter to get the wanted size:



And the detected faces result like this:



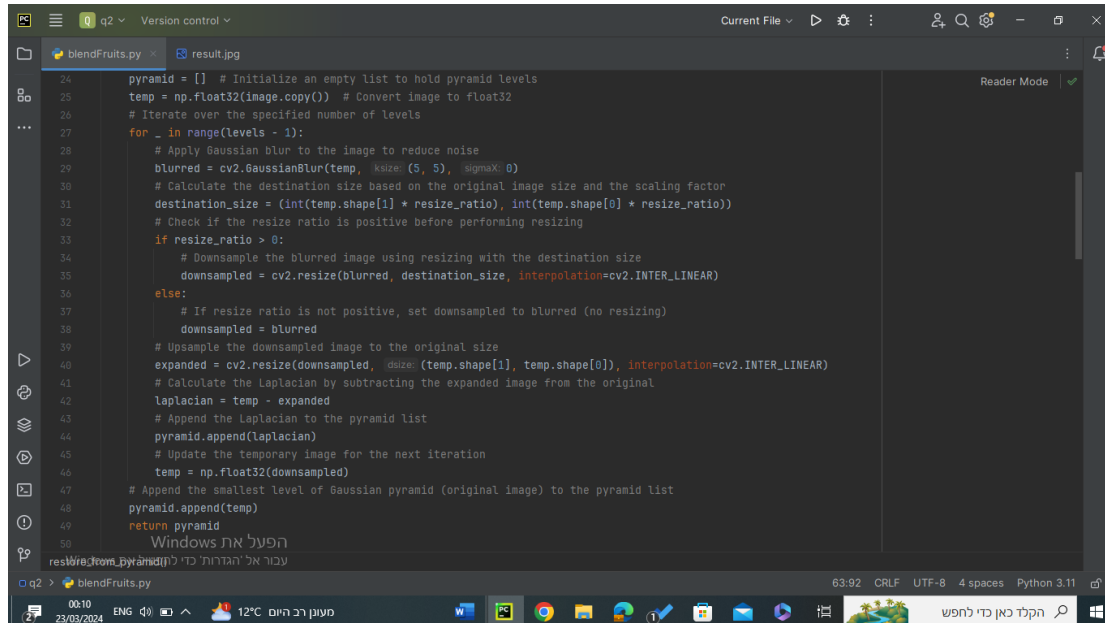
We have a small amount of false positive that we can get rid from by increasing the threshold:



Also we got less amount of detected faces.

## Question 2:

### a) Implementing get\_laplacian\_pyramid:



```
24 pyramid = [] # Initialize an empty list to hold pyramid levels
25 temp = np.float32(image.copy()) # Convert image to float32
26 # Iterate over the specified number of levels
27 for _ in range(levels - 1):
28     # Apply Gaussian blur to the image to reduce noise
29     blurred = cv2.GaussianBlur(temp, ksize=(5, 5), sigmaX=0)
30     # Calculate the destination size based on the original image size and the scaling factor
31     destination_size = (int(temp.shape[1] * resize_ratio), int(temp.shape[0] * resize_ratio))
32     # Check if the resize ratio is positive before performing resizing
33     if resize_ratio > 0:
34         # Downsample the blurred image using resizing with the destination size
35         downsampled = cv2.resize(blurred, destination_size, interpolation=cv2.INTER_LINEAR)
36     else:
37         # If resize ratio is not positive, set downsampled to blurred (no resizing)
38         downsampled = blurred
39     # Upsample the downsampled image to the original size
40     expanded = cv2.resize(downsampled, dsize=(temp.shape[1], temp.shape[0]), interpolation=cv2.INTER_LINEAR)
41     # Calculate the Laplacian by subtracting the expanded image from the original
42     laplacian = temp - expanded
43     # Append the Laplacian to the pyramid list
44     pyramid.append(laplacian)
45     # Update the temporary image for the next iteration
46     temp = np.float32(downsampled)
47 # Append the smallest level of Gaussian pyramid (original image) to the pyramid list
48 pyramid.append(temp)
49 return pyramid
50 הפעל את Windows
עבור אל הגדרות כדי לנקות את המערכת
```

The **get\_laplacian\_pyramid** function constructs a Laplacian pyramid from a given image. A Laplacian pyramid is a multi-scale representation of an image that decomposes the image into different levels of detail. Here's an explanation of each part of the function:

#### 1)Parameters:

**image:** This parameter represents the input image, which is expected to be a numpy array.

**levels:** Specifies the number of levels in the Laplacian pyramid.

**resize\_ratio:** (Optional) It determines the ratio by which the image is resized at each level. The default value is set to 0.5, indicating that the image is resized to half of its original size at each level.

#### 2>Returns:

It returns a list containing the laplacian pyramid levels

#### 3)Functionality:

**pyramid = []**: Initializes an empty list to hold the Laplacian pyramid levels.

**temp = np.float32(image.copy())**: Converts the input image to **float32** format to ensure numerical stability during computations.

Iterates over the specified number of levels and does the following:

1) **blurred = cv2.GaussianBlur(temp, (5, 5), 0)**: Applies Gaussian blur to the image to reduce noise

2) Computes the destination size for resizing based on the original image size and the resize ratio.

3) Checks if the resize ratio is positive and performs resizing accordingly

4) Upsamples the downsampled image to the original size.

5) Calculates the Laplacian by subtracting the expanded image from the original.

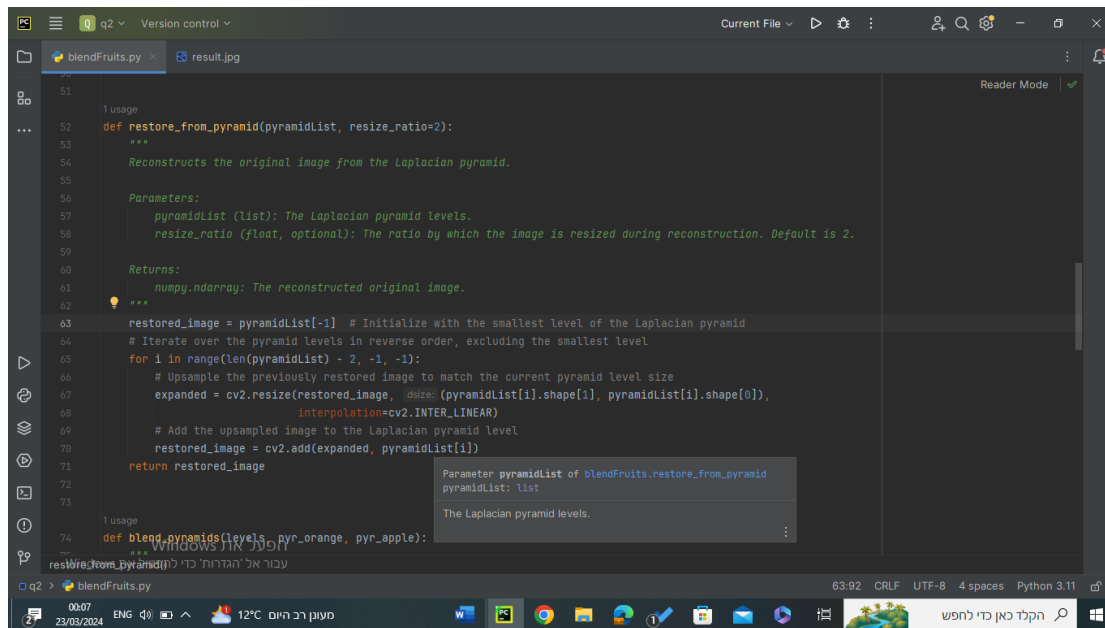
6) Appends the Laplacian to the **pyramid** list.

7) Updates the temporary image (**temp**) for the next iteration.

After we are done with the loop the function Appends the smallest level of the Gaussian pyramid (original image) to the **pyramid** list.

Returns the constructed Laplacian pyramid.

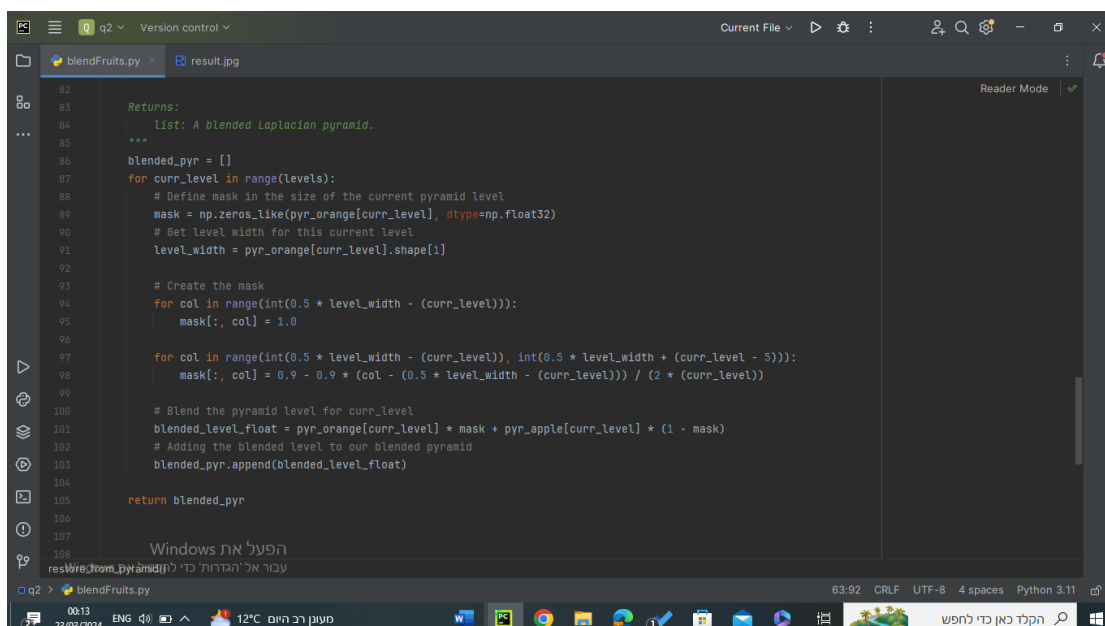
## b) Implementing restore from pyramid:



```
51 1 usage
52 def restore_from_pyramid(pyramidList, resize_ratio=2):
53     """
54     Reconstructs the original image from the Laplacian pyramid.
55
56     Parameters:
57         pyramidList (list): The Laplacian pyramid levels.
58         resize_ratio (float, optional): The ratio by which the image is resized during reconstruction. Default is 2.
59
60     Returns:
61         numpy.ndarray: The reconstructed original image.
62     """
63     restored_image = pyramidList[-1] # Initialize with the smallest level of the Laplacian pyramid
64     # Iterate over the pyramid levels in reverse order, excluding the smallest level
65     for i in range(len(pyramidList) - 2, -1, -1):
66         # Upsample the previously restored image to match the current pyramid level size
67         expanded = cv2.resize(restored_image, dsize=(pyramidList[i].shape[1], pyramidList[i].shape[0]),
68                               interpolation=cv2.INTER_LINEAR)
69         # Add the upsampled image to the Laplacian pyramid level
70         restored_image = cv2.add(expanded, pyramidList[i])
71     return restored_image
72
73 1 usage
74 def blend_pyramids(levels, pyr_orange, pyr_apple):
75     """
76     Windows את הודות כדי להקדיר
77     restore_from_pyramid
78     עבר אל הודות כדי להקדיר
79
80     Parameters:
81         levels (list): A list of Laplacian pyramid levels.
82         pyr_orange (numpy.ndarray): The Laplacian pyramid levels for the orange image.
83         pyr_apple (numpy.ndarray): The Laplacian pyramid levels for the apple image.
84
85     Returns:
86         list: A blended Laplacian pyramid.
87
88     """
89     blended_pyr = []
90     for curr_level in range(levels):
91         # Define mask in the size of the current pyramid level
92         mask = np.zeros_like(pyr_orange[curr_level], dtype=np.float32)
93         # Get level width for this current level
94         level_width = pyr_orange[curr_level].shape[1]
95
96         # Create the mask
97         for col in range(int(0.5 * level_width - (curr_level))), int(0.5 * level_width + (curr_level - 5))):
98             mask[:, col] = 0.9 - 0.9 * (col - (0.5 * level_width - (curr_level))) / (2 * (curr_level))
99
100         # Blend the pyramid level for curr_level
101         blended_level_float = pyr_orange[curr_level] * mask + pyr_apple[curr_level] * (1 - mask)
102         # Adding the blended level to our blended pyramid
103         blended_pyr.append(blended_level_float)
104
105     return blended_pyr
106
107 Windows את הודות
108 restore_from_pyramid
109 עבר אל הודות כדי להקדיר
```

as we can see to restore the image from the pyramid we need to do something called a collapse where we start from the smallest level and then iterate over the pyramid levels in reverse order (to get from smallest to highest ), we basically use a for loop where we upsample the previously stored image and add it to the current laplacian pyramid level.

## C) implementing blend pyramids:



```
82
83 Returns:
84     list: A blended Laplacian pyramid.
85 """
86 blended_pyr = []
87 for curr_level in range(levels):
88     # Define mask in the size of the current pyramid level
89     mask = np.zeros_like(pyr_orange[curr_level], dtype=np.float32)
90     # Get level width for this current level
91     level_width = pyr_orange[curr_level].shape[1]
92
93     # Create the mask
94     for col in range(int(0.5 * level_width - (curr_level))), int(0.5 * level_width + (curr_level - 5))):
95         mask[:, col] = 0.9 - 0.9 * (col - (0.5 * level_width - (curr_level))) / (2 * (curr_level))
96
97     for col in range(int(0.5 * level_width - (curr_level)), int(0.5 * level_width + (curr_level - 5))):
98         mask[:, col] = 0.9 - 0.9 * (col - (0.5 * level_width - (curr_level))) / (2 * (curr_level))
99
100     # Blend the pyramid level for curr_level
101     blended_level_float = pyr_orange[curr_level] * mask + pyr_apple[curr_level] * (1 - mask)
102     # Adding the blended level to our blended pyramid
103     blended_pyr.append(blended_level_float)
104
105     return blended_pyr
106
107 Windows את הודות
108 restore_from_pyramid
109 עבר אל הודות כדי להקדיר
```

For each level (we chose 5 levels because it gives us the best answers) in the pyramids we do the following:

1) define mask the size of the current pyramid level.

2) get the level width for this level

3) creating the mask using advanced indexing techniques to get the smooth blurring between the apple and orange image.

4) then we blend the 2 pyramids on this level using this line `blended_level_float = pyr_orange[curr_level] * mask + pyr_apple[curr_level] * (1 - mask)`

5) we add the blended level to our blended pyramid