**Image processing Final project**



**By: Noam Bassat**

Table Of Content

[Visible Watermark 3](#_Toc73020753)

[Invisible Watermark 5](#_Toc73020754)

[Check changes on the original image: 7](#_Toc73020755)

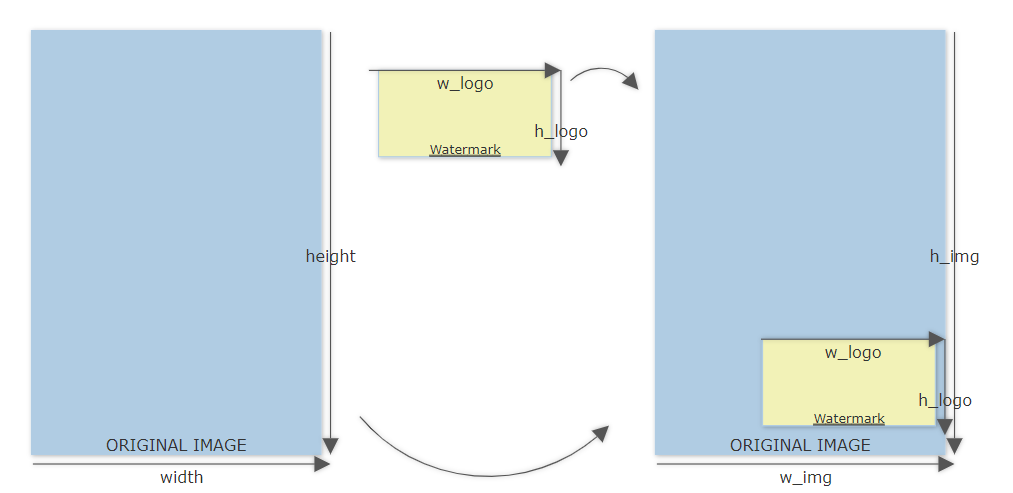
## Visible Watermark

In this section, I have created a function that gets as inputs: An image, a watermark image, an intensity value, a height position and a width position and returns the same image with a visible watermark.

def add\_visible\_watermark(img, logo, intensity, w\_img, h\_img):  
 h\_logo, w\_logo, c\_logo = logo.shape  
 y\_up = h\_img + int(h\_logo/2)  
 x\_left = w\_img - int(w\_logo/2)  
 y\_down = h\_img -int(h\_logo/2)  
 x\_right = w\_img + int(w\_logo/2)+1  
  
 print("y\_down: ", y\_down)  
 print("y\_up: ", y\_up)  
 print("x\_left: ", x\_left)  
 print("x\_right: ", x\_right)  
 print("shape: ", img.shape)  
 img[y\_down : y\_up, x\_left: x\_right] = img[y\_down : y\_up, x\_left: x\_right] - (logo-intensity)  
 return img

**Description:  
Inputs:**"img" = The main image that we want to add watermark into; "logo" – the watermark image that needs to be added; intensity – how strongly the watermark will appear on the image; 'w\_img' and 'h\_img' – the width and height positions for adding the watermark on.

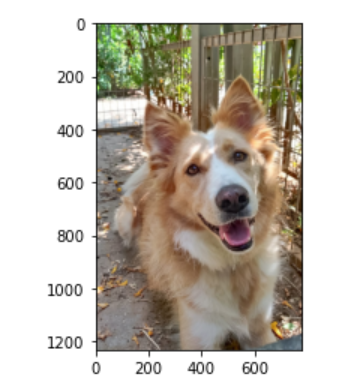
**Output:**The original image with visible watermark in the desired position.

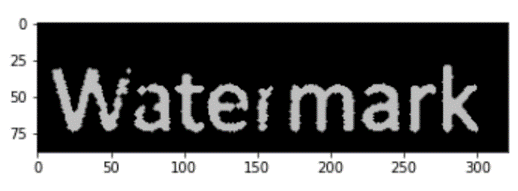
**Functionality:**1. Calculate h\_logo, w\_logo and c\_logo as the height, width and color of the watermark.  
2. Set the correct position for pasting the watermark:  
y\_up position is calculated by adding to the original image height, half of the watermark height value.  
x\_left position is calculated by subtracting from the original image width, half of the watermark width value.  
y\_down position is calculated by subtracting from the original image height, half of the watermark height value.  
y\_right position is calculated by adding to the original image width, half of the watermark width value.  
3. Finally, subtract the intensity from the logo, and then subtract the new logo from the positions above, in the original image.

h\_img

w\_img

**Before:**

**  
Original image Watermark   
Oםם**



**Results**:

img2 = add\_visible\_watermark(img, watermark, **0.25, 600, 1100**)

Image with visible watermark



y\_down: 1056

y\_up: 1144

x\_left: 440

x\_right: 761

shape: (1233, 778, 3)

img3 = add\_visible\_watermark(img, watermark, **0.5, 400, 1100**)

y\_down: 1056

y\_up: 1144

x\_left: 240

x\_right: 561

shape: (1233, 778, 3)

### Invisible Watermark

In this section, I have created a function that gets as inputs: an image and a watermark image, and returns the same image with an invisible watermark.

def add\_invisible\_watermark(img,watermark):  
 i=0  
 data = "0"  
 for x in watermark:  
 for y in x:  
 for z in y:  
 data += str(z)  
 with Image.open("Lotus.jpg") as inv\_wa\_img:  
 width, height = inv\_wa\_img.size  
 for x,y,n in itertools.product(range(width), range(height), range(3)):  
 pixel = list(inv\_wa\_img.getpixel((x, y)))  
 if(i<len(data)):  
 pixel[n] = pixel[n] & ~1 | int(data[i]) # (pixel[n] and not 1) or data  
 i+=1  
 inv\_wa\_img.putpixel((x,y), tuple(pixel))  
 inv\_wa\_img.save("source\_secret.png", "PNG")  
 plt.imshow(inv\_wa\_img)  
 return inv\_wa\_img

**Functionality:**1. First, disassemble the watermark, into binary bits data string. Declare an integer 'i'=0 that presents the current index in the data string.  
2. Loop over the image's width and height, and make a list if its pixels.  
3. For each pixel in the three color channels (RGB), which is a type of UINT8, store the value in the pixel's list at the 'n' position (**pixel[n]**).  
4. For each pixel[n]: pixel[n] = pixel[n] AND ~1 OR data[i]:  
~1 = 1complete = 11111110  
(UINT8)X AND ~1 => Reset the value of the LSB. For example:  
 (DEC)7 =(UINT8)0000111; ~1 = 11111110 -> 0000111 (**AND**) 11111110 = 0000110 = (DEC)6.  
 In this way, the least significant bit is deleted. Then, adding the "OR" condition to it, so this bit becomes 1 again or stays zero, depending on the data[i] current value.

5. Finally, add the new changed pixels into the original image and return it.

Before: After:

Before: After:



### Check changes on the original image:

In this section, I have created a function that compares the lower pixels of a given image to those of the original one (with the invisible watermark).

def check\_changes(im2):  
 with Image.open("source\_secret.png") as im1:  
 width, height = im1.size  
 if (im1.size!= im2.size): return "Different images!"  
 for x,y in itertools.product(range(width), range(height)):

pixel = list(im1.getpixel((x, y)))  
 pixel2 =list(im2.getpixel((x, y)))  
 for n in range(0,3):  
 if(pixel[n]!=pixel2[n]): return "Changed!!!"  
 return "Pure Image!"

**Description:  
Inputs:**"img2" = The same image as the original one, with a minor filter added.

**Output:**A string that indicates whether the original image has been changed or not.

**Functionality:**1. If the images sizes are different, returns "Different images!"  
2. Else, looping over the images height and width and storing the original image's pixels into pixel list, and the other image into pixel2 list.  
3. For each pixel in the three color channels (RGB), which is a type of UINT8, store the value in the pixel's list at the 'n' position (**pixel[n]** for the original image and **pixel2[n]** for the other image.)  
4. Compare each pixel with the parallel pixel2, if they are not equal, return "Changed!!!"  
5. Else, continue until the end, if there were not any different pixels, return "Pure Image".



### A system for counting and identifying objects from the same category

In this section, I have created a system for identifying and counting pens, in a given image.  
In order to check the system's performance, I sent images that shows both pens and markers.

def object\_in\_img(image, blur):  
 plt.imshow(image)  
 df = pd.DataFrame(index=range(6), columns=["Threshold", "Num\_of\_Pens", "FP", "FN"])  
 image = image[:, :, 1]  
 thresh = threshold\_otsu(image) - 10  
 for i in range(14):  
 thresh += 10  
 df["Threshold"][i] = thresh  
 bw = closing(image > thresh, square(4))  
 cleared = clear\_border(bw)  
  
 # label image regions  
 label\_image = label(cleared)  
 image\_label\_overlay = label2rgb(label\_image, image=image, bg\_label=0)  
  
 fig, ax = plt.subplots(figsize=(10, 6))  
 ax.imshow(image\_label\_overlay)  
 counter = 0  
 for region in regionprops(label\_image):  
 # take regions with large enough areas  
 if region.area >= 500:  
 minr, minc, maxr, maxc = region.bbox  
 max\_line = max(maxc - minc, maxr - minr)  
 min\_line = min(maxc - minc, maxr - minr)  
 prop = min\_line / max\_line  
  
 if (max\_line < 250 or min\_line > 70): continue  
 rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,  
 fill=False, edgecolor='red', linewidth=2)  
 ax.add\_patch(rect)  
 counter += 1  
 print("Number of pens = ", counter)  
 df["Num\_of\_Pens"][i] = counter  
 ax.set\_axis\_off()  
 plt.tight\_layout()  
 plt.show()  
 return df

**thresh = threshold\_otsu(image) : Return threshold value based on Otsu’s method.**

**cleared = clear\_border(bw): remove artifacts connected to image border  
 rect = mpatches.Rectangle: draw rectangle around segmented pens**

**analyzing\_data\_to\_gender MyProject-Copy1**

