Functions implemented in the assignment
1. For reading and conversion to gray.

def **show_image**(img_name, img, cmap):
The function saves or shows the image

def **convert_to_grayscale**(image):
This function converts the image into grayscale using below formula:
img_gray = 0.2989*R + 0.5870*G + 0.1140*B

2. apply median filtering and Gaussian filtering (of standard deviation 4) for denoising. Display the original, noisy and all filtered images. Compute the PSNR of the noisy, and filtered images with respect to the original image in each case. - Analysis (10 marks), downsampling (5 marks), thresholding (10 marks), upsampling (5 marks), synthesis (10 marks), Gaussian filtering (5 marks), median filtering (5 marks), computing PSNRs (5 marks), display images (5 marks)

def **gaussian_noise**(img):
The function adds gaussian noise to the image.

Median filter is usually used to reduce noise in an image. We will be dealing with salt and pepper noise in example below. Median_Filter method takes 2 arguments, Image array and filter size. Lets say you have your Image array in the variable called img_arr, and you want to remove the noise from this image using 3x3 median filter. Thats how we do it.

PSNR is defined as follows:

$$PSNR = 10log_{10}(\frac{(L-1)^2}{MSE}) = 20log_{10}(\frac{L-1}{RMSE})$$

Here, L is the number of maximum possible intensity levels (minimum intensity level suppose to be 0) in an image.

MSE is the mean squared error & it is defined as:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(O(i,j) - D(i,j))^2$$

3. On the original gray image apply the LoG (Laplacian of Gaussian) operator of standard deviation 5, and find edge pixels by computing its zero crossings

Steps:
We will define a function for the filter
Then we will make the mask
Then we will define function to iterate that filter over the image(mask)
We will make a function for checking the zeros as explained
And finally a function to bind all this together
One thing to keep in mind is that to create a combination of the filters, i.e. Laplacian over gaussian filter (LoG), then we can use the following formula to combine both of them.

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

As we are breaking down the filter, we would also need some additional functions. Those are:

A function to take the filter and convolve around the image.
A function for creating the actual mask.
And finally a function to check the zeros.

Function(s) logic:
Create_log function:

Calculate w by using the size of filter and sigma value
Check if it is even of not, if it even make it odd, by adding 1
Iterate through the pixels and apply log filter and then append those changed pixels into a new array, and then reshape the array.
Convolve function:

Extract the heights and width of the image
Now make a range of pixels which are covered by the mask (output of filter)
Make an array of zeros (res_image)
Now iterate through the image and append the values of product of mask and the original image and append it to the res_image array.
Zeros function (To mark the edges):

Make a zeros array of same dimensions of the log_image (zc_image)
Now iterate over the image and check two values, if they are equal to zero or not. If they are not equal to zero then check if values are positive or negative.
If the pixel value is zero, that means there an edge exist.
If none of the pixels, are zero, that means there is no edge.
Now whichever pixels were zero, append those coordinates to zc_image array.
Make all the pixels in zc_image as 1, meaning white. That will show the edges.

4. Perform histogram equalization on the original grayscale image and perform the Otsu thresholding operation

The mathematical formula from which we'll base our solution is:

$$P_x(j) = \sum_{i=0}^{j} P_x(i)$$

Now we have our histogram, and we can take the next step towards equalization by computing the cumulative sum of the histogram. The cumulative sum is exactly as it sounds—the sum of all values in the histogram up to that point, taking into account all previous values.
We're making progress! We now have the cumulative sum, but as you can see, the values are huge (> 6,000,000). We're going to be matching these values to our original image in the final step, so we have to normalize them to conform to a range of 0–255. Here's one last formula for us to code up:

$$s_k = \sum_{j=0}^{k} \frac{n_j}{N}$$

That's better—our values are now normalized between 0-255. Now, for the grand finale. We can now use the normalized cumulative sum to modify the intensity values of our original image. The code to do this can look a bit confusing if you've never used numpy before. In fact, it's anticlimactically simple.

We'll take all of the values from the flat array and use it as the index to look up related value in the cs array. The result becomes the new intensity value which will be stored in img_new for that particular pixel.

And there we have it—the original image has been equalized. We're practically radiologists now! Notice the difference in contrast throughout the whole image. The most important thing to remember about histogram equalization is that it adjusts the intensities at a global level, taking into account all pixels. That process works well for images like the one above but may perform poorly on other images.

1. def **filter2D**(image, kernel)- Takes a grayscale image and a kernel as numpy 2D arrays. Convolves the kernel on the image with stride = 1 in both dimensions and returns the output as numpy array.

2. def **scaling**(image, sigmag = 3, k = 5)- Takes a grayscale image, sigma for gaussian and kernel size (k). Performs simple gaussian filtering and returns the output image.
3. def **sharpen**(image)- Sharpens the grayscale image using Laplacian filter. It first filters the image using the Laplacian. The filtered image is then added back to the original image using proper scaling.
4. def **edge_sobel**(image)- Takes an image and detects edges using sobel operators for X and Y directions. Returns the gradient magnitude image, along with X and Y derivatives.
5. def **otsu**(image)- Performs optimum global thresholding using Otsu's method. Returns a binary image thresholded as per the obtained value

5. Perform Harris corner detection on the gray image and display the detected corners
def **harris**(img, threshold=1e-2, nms_size=10)- Computes harris corners and draws them on the supplied image as circles and returns it. Nms_size indicates the window size used for the Non- maxima suppression step.