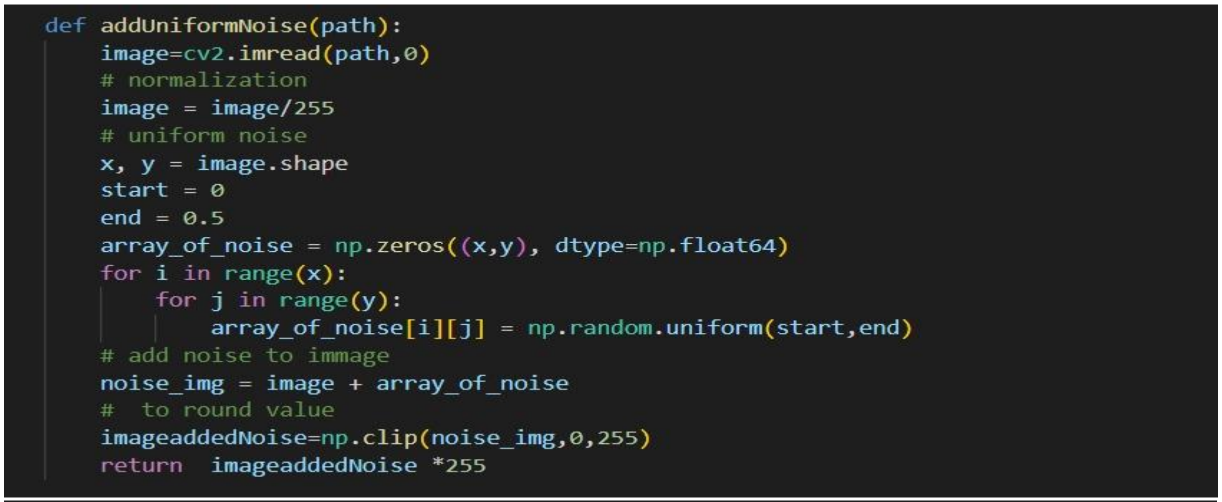
Computer Vision

Phase 1

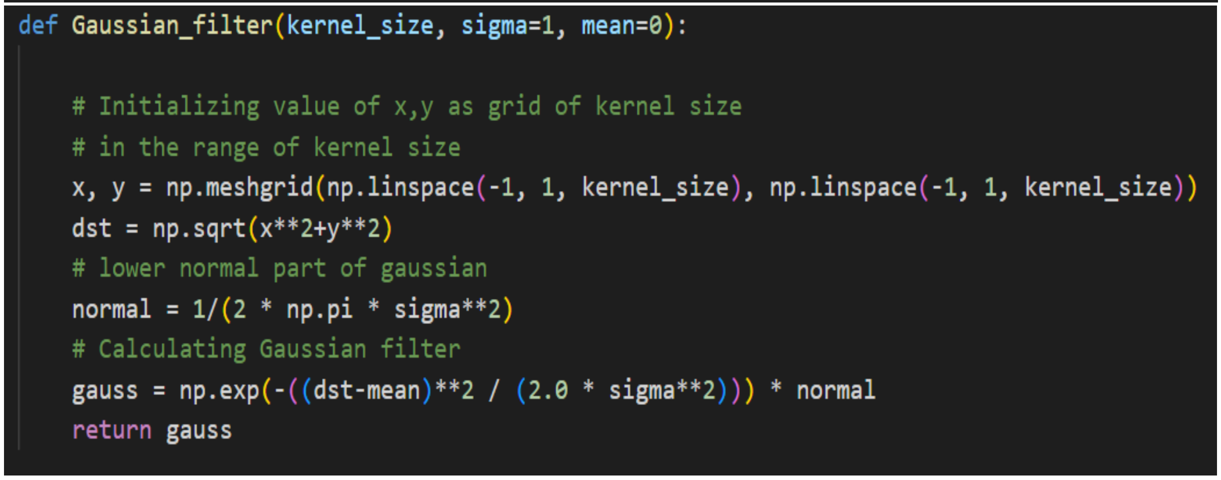
Team Member Names	Section	Bench Number
Osama Faisal AbdulLatef	1	11
Beshara Safwat Fahim	1	22
Shuaib AbdulSalam Ahmed	1	48
AbdElRahman Sameh Mohamed	1	53
Mariam Mounier AbdElRehim	2	35

1. Implementation Of Uniform Noise

It works by reading the image in a 2D array first then normalizing it by diving by 255, Then taking the shape of x and y of it in two variables. We initialize array of zeros with same size of image, after that we begin fill it by uniform random noise and returning random noised image.



```
def addUniformNoise(path):
    image=cv2.imread(path,0)
    # normalization
    image = image/255
    # uniform noise
    x, y = image.shape
    start = 0
    end = 0.5
    array_of_noise = np.zeros((x,y), dtype=np.float64)
    for i in range(x):
        for j in range(y):
            array_of_noise[i][j] = np.random.uniform(start,end)
    # add noise to immage
    noise_img = image + array_of_noise
    #  to round value
    imageaddedNoise=np.clip(noise_img,0,255)
    return  imageaddedNoise *255
```

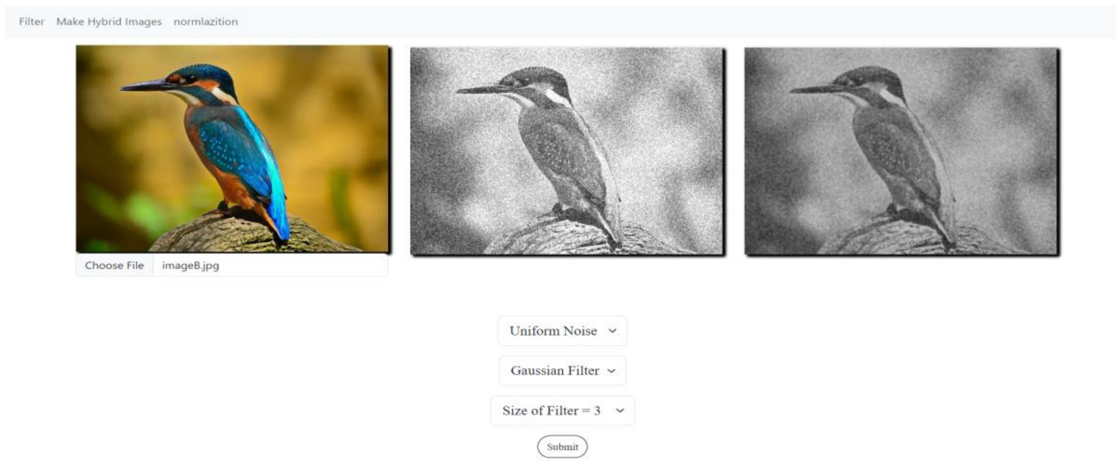


```
def Gaussian_filter(kernel_size, sigma=1, mean=0):

    # Initializing value of x,y as grid of kernel size
    # in the range of kernel size
    x, y = np.meshgrid(np.linspace(-1, 1, kernel_size), np.linspace(-1, 1, kernel_size))
    dst = np.sqrt(x**2+y**2)
    # lower normal part of gaussian
    normal = 1/(2 * np.pi * sigma**2)
    # Calculating Gaussian filter
    gauss = np.exp(-((dst-mean)**2 / (2.0 * sigma**2))) * normal
    return gauss
```

We remove noise by applying the Gaussian window by substituting in the following equation:

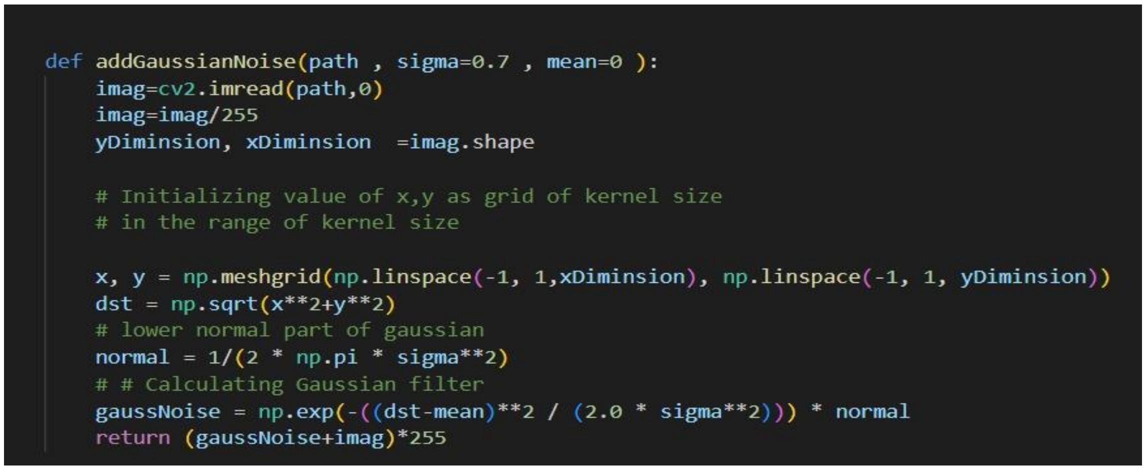
$$\frac{1}{2\pi(\sigma)^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2. Output of uniform Noised Image With Gaussian Filter



3. Implementation Of Gaussian Noise

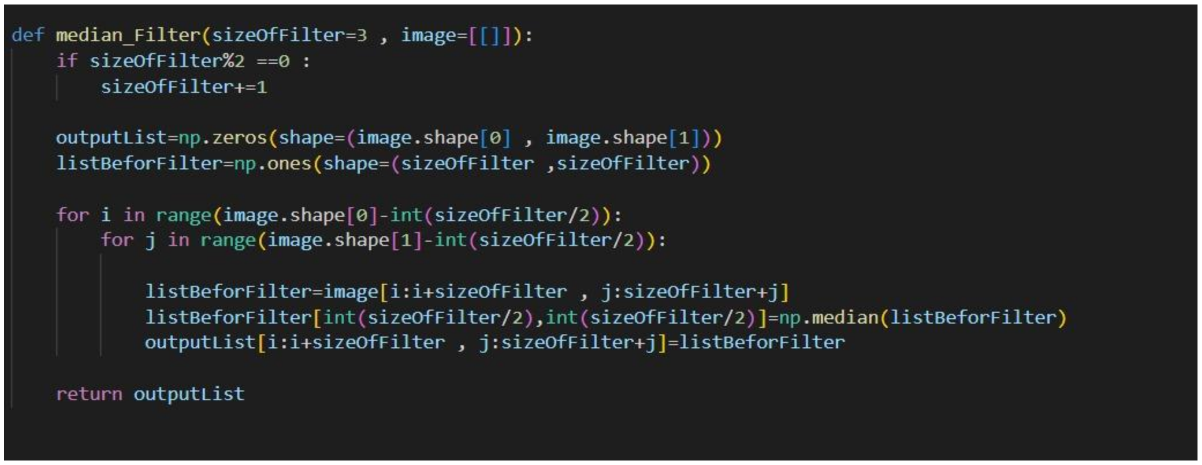
We read the image just as before then we begin to create the kernel (window) of noise with the same shape of image and we substitute in the past told equation. In the end we return the Gaussian Noised Image.



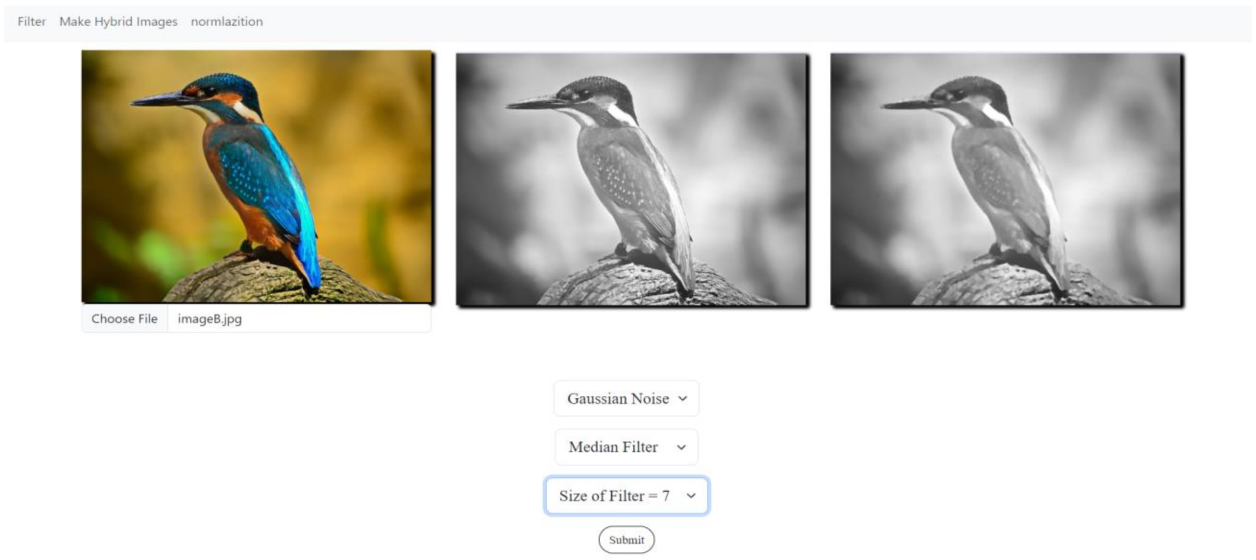
```
def addGaussianNoise(path , sigma=0.7 , mean=0 ):
    imag=cv2.imread(path,0)
    imag=imag/255
    yDiminsion, xDiminsion  =imag.shape

    # Initializing value of x,y as grid of kernel size
    # in the range of kernel size

    x, y = np.meshgrid(np.linspace(-1, 1,xDiminsion), np.linspace(-1, 1, yDiminsion))
    dst = np.sqrt(x**2+y**2)
    # lower normal part of gaussian
    normal = 1/(2 * np.pi * sigma**2)
    # # Calculating Gaussian filter
    gaussNoise = np.exp(-((dst-mean)**2 / (2.0 * sigma**2))) * normal
    return (gaussNoise+imag)*255
```

4. Implementation of Median Filter and Gaussian Noise Output

Median filter works by filling an array of zeros by pixels values then getting the median of these values. After getting the median we replace the center of filter with that median value found in the image and then applying filter to image.



```
def median_Filter(sizeOfFilter=3 , image=[[]]):
    if sizeOfFilter%2 ==0 :
        sizeOfFilter+=1

    outputList=np.zeros(shape=(image.shape[0] , image.shape[1]))
    listBeforFilter=np.ones(shape=(sizeOfFilter ,sizeOfFilter))

    for i in range(image.shape[0]-int(sizeOfFilter/2)):
        for j in range(image.shape[1]-int(sizeOfFilter/2)):

            listBeforFilter=image[i:i+sizeOfFilter , j:sizeOfFilter+j]
            listBeforFilter[int(sizeOfFilter/2),int(sizeOfFilter/2)]=np.median(listBeforFilter)
            outputList[i:i+sizeOfFilter , j:sizeOfFilter+j]=listBeforFilter

    return outputList
```

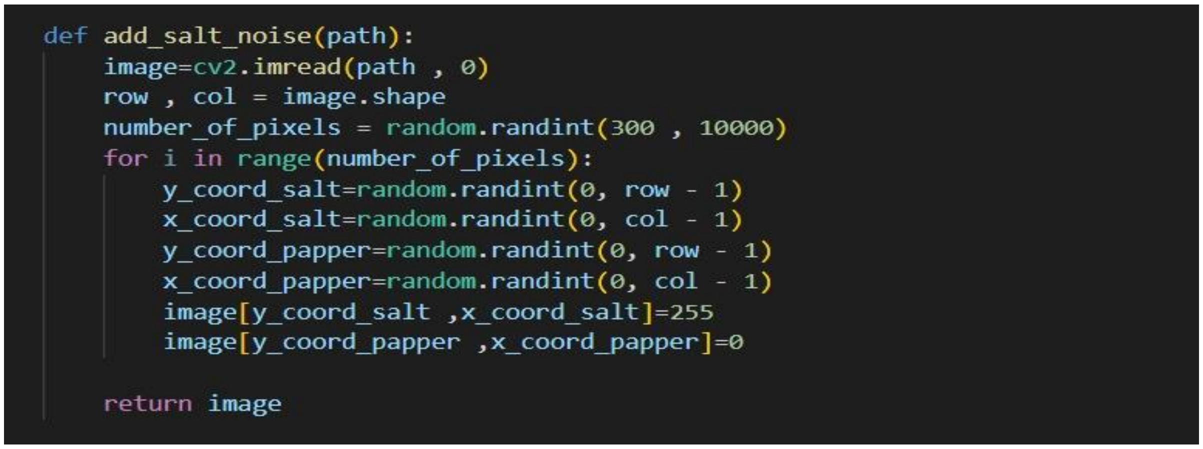


5. Implementation Of Salt and Pepper Noise

After reading the image we create variable within the range 300 to 10.000 with random values and we loop on it.

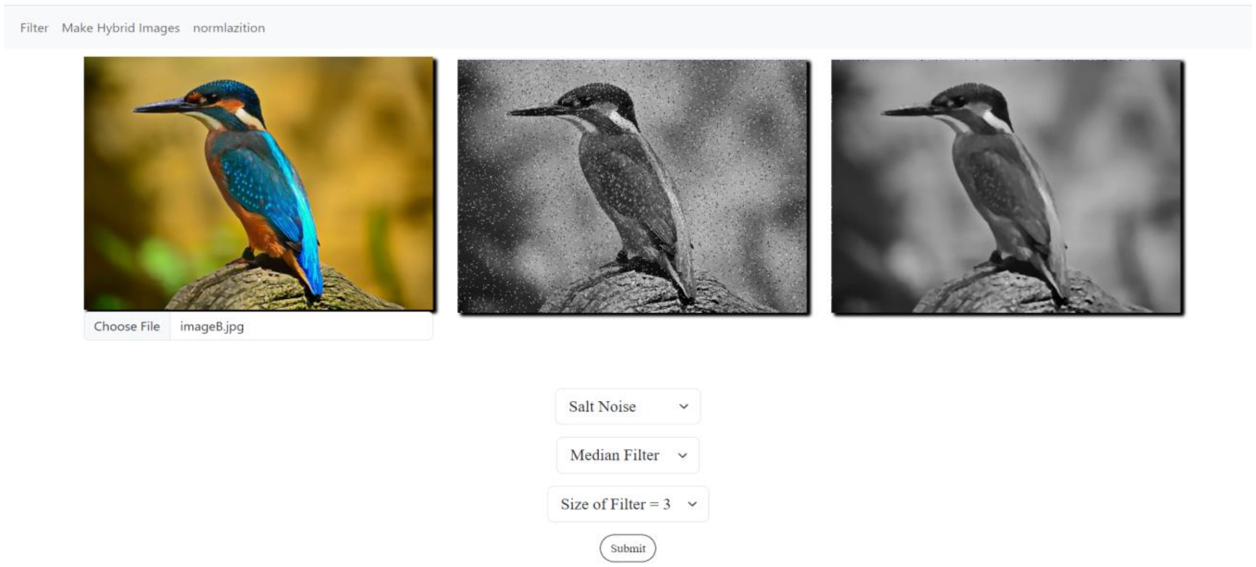
In the loop we begin to fill the x and y coordinates with random variables within shape of image.

We take the x and y coordinates and multiply it with 255 to be white in grey scale and same coordinates and multiply it with 0 to be grey in grey scale.



```python
def add_salt_noise(path):
    image=cv2.imread(path , 0)
    row , col = image.shape
    number_of_pixels = random.randint(300 , 10000)
    for i in range(number_of_pixels):
        y_coord_salt=random.randint(0, row - 1)
        x_coord_salt=random.randint(0, col - 1)
        y_coord_papper=random.randint(0, row - 1)
        x_coord_papper=random.randint(0, col - 1)
        image[y_coord_salt ,x_coord_salt]=255
        image[y_coord_papper ,x_coord_papper]=0

    return image
```

6. Output Of Salt and Pepper Noise and Output Of Median filter



Salt Noise

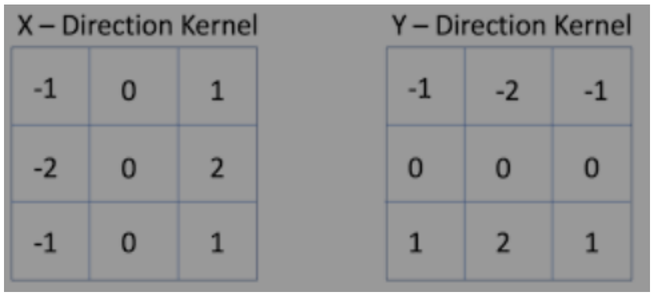
Median Filter

Size of Filter = 3

7. Edge Detection by Sobel operator:

Edges are detected by convolving the image with the operator of choice

- Sobel operator: better approximation to gradient magnitude, produces thin edges

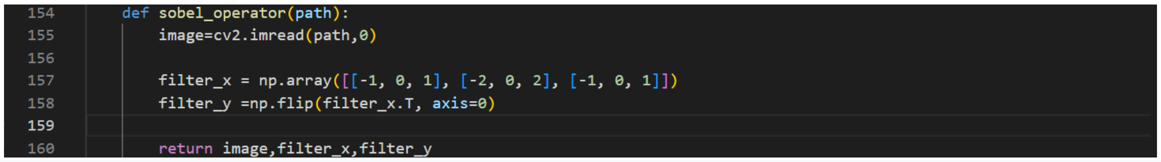


X – Direction Kernel

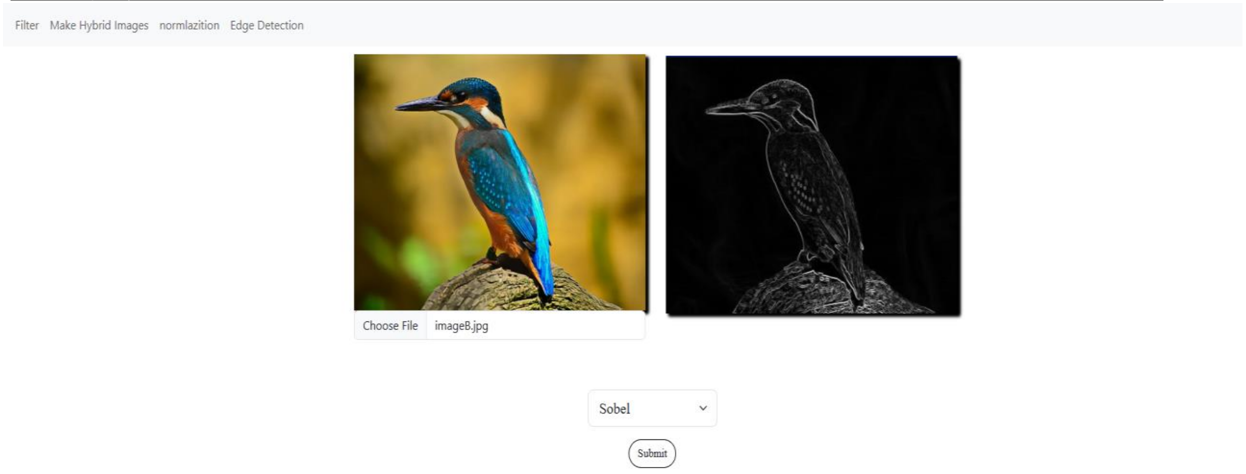
-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

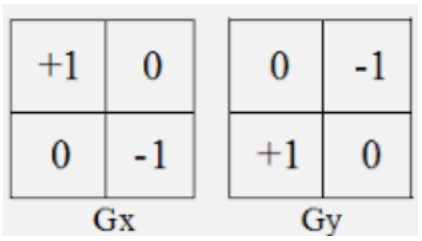
-1	-2	-1
0	0	0
1	2	1



```python
def sobel_operator(path):
    image=cv2.imread(path,0)

    filter_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    filter_y =np.flip(filter_x.T, axis=0)

    return image,filter_x,filter_y
```



- Roberts operator: simple, fast and detects thicker edges

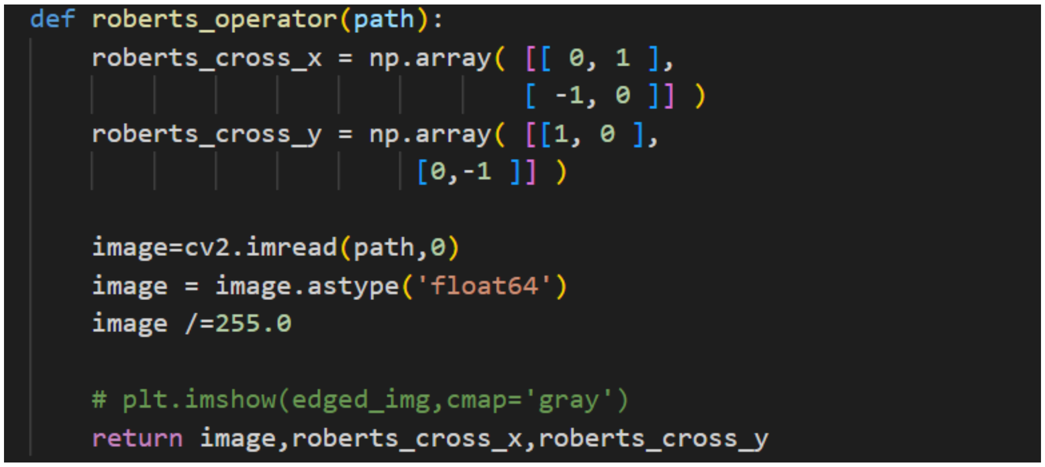


+1	0
0	-1

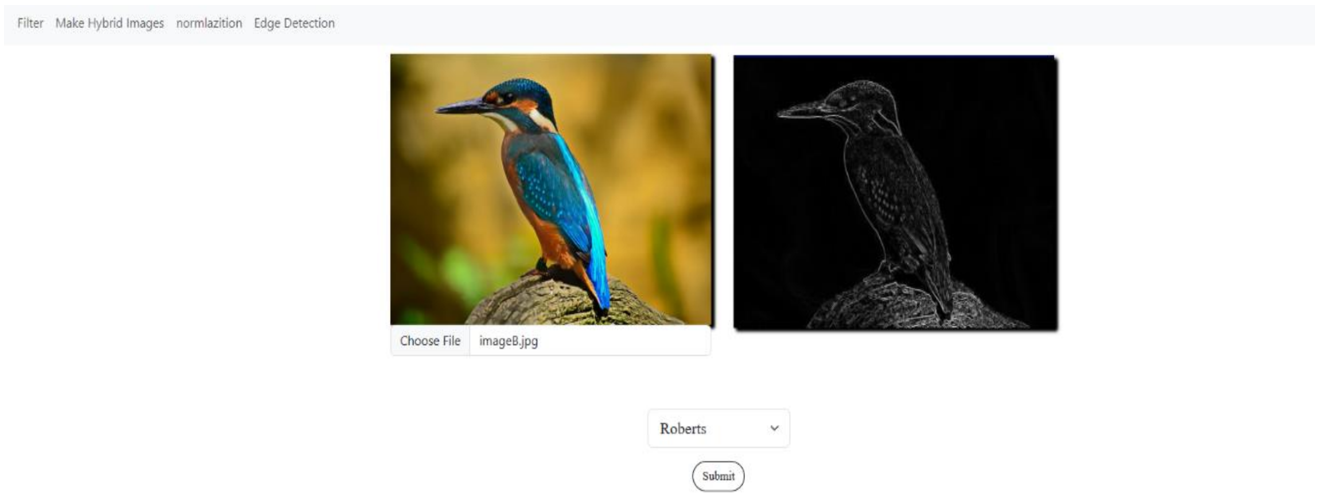
Gx

0	-1
+1	0

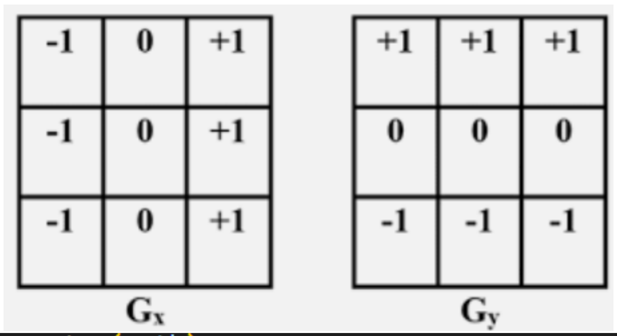
Gy



```python
def roberts_operator(path):
    roberts_cross_x = np.array( [[ 0, 1 ],
                                 [ -1, 0 ]] )
    roberts_cross_y = np.array( [[1, 0 ],
                          [0,-1 ]] )

    image=cv2.imread(path,0)
    image = image.astype('float64')
    image /=255.0

    # plt.imshow(edged_img,cmap='gray')
    return image,roberts_cross_x,roberts_cross_y
```



- Prewitt operator: more sensitive to horizontal and vertical edges

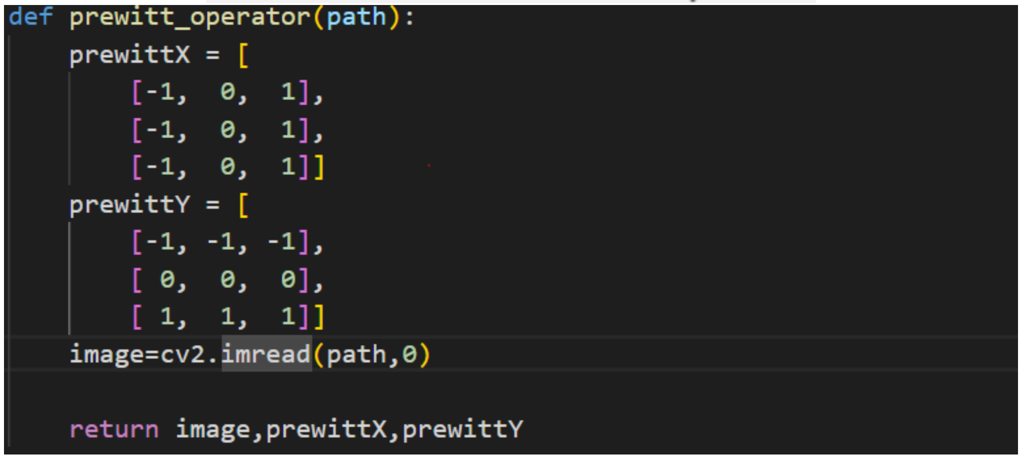


-1	0	+1
-1	0	+1
-1	0	+1

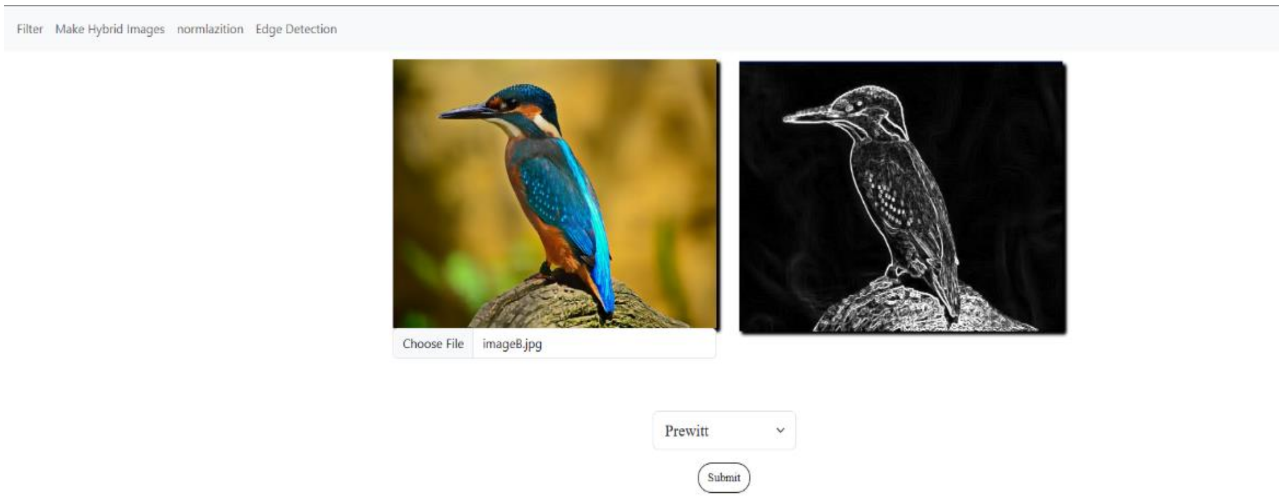
G_x

+1	+1	+1
0	0	0
-1	-1	-1

G_y

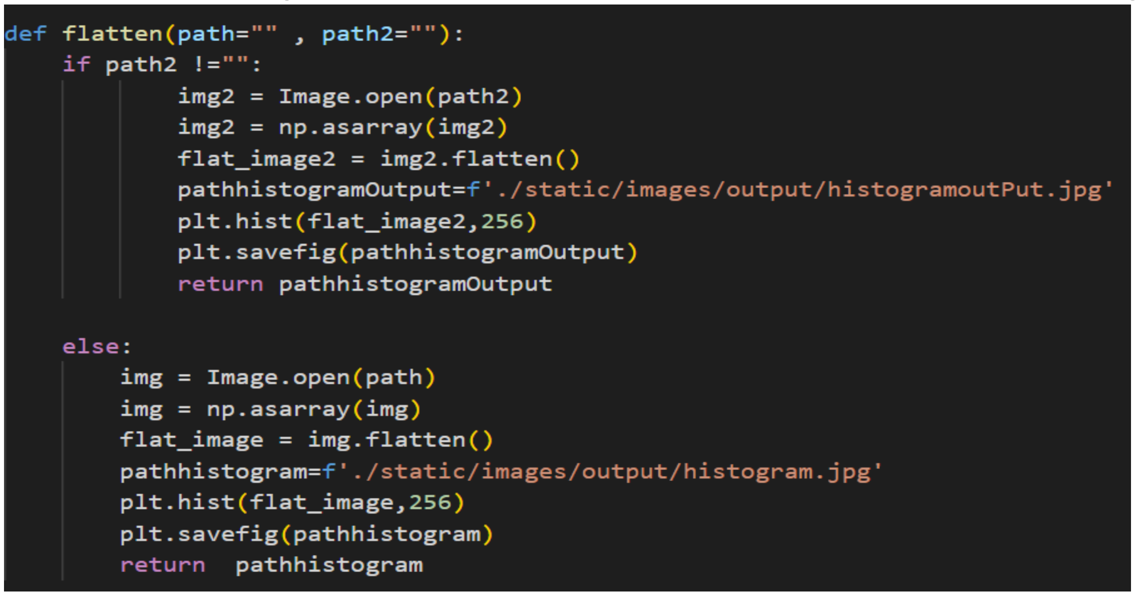


```
def prewitt_operator(path):
    prewittX = [
        [-1,  0,  1],
        [-1,  0,  1],
        [-1,  0,  1]]
    prewittY = [
        [-1, -1, -1],
        [ 0,  0,  0],
        [ 1,  1,  1]]
    image=cv2.imread(path,0)

    return image,prewittX,prewittY
```



8. Histogram and Normalization and Equalization

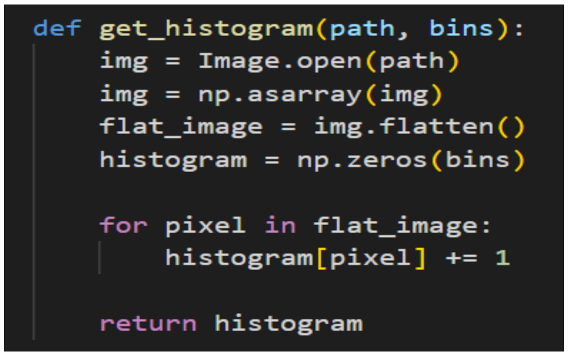
We begin by flatting the image (changing it to 1D array) and here one time we return the path of input histogram and the second we return the path of output histogram.



```
def flatten(path="" , path2=""):
    if path2 !="":
            img2 = Image.open(path2)
            img2 = np.asarray(img2)
            flat_image2 = img2.flatten()
            pathhistogramOutput=f'./static/images/output/histogramoutPut.jpg'
            plt.hist(flat_image2,256)
            plt.savefig(pathhistogramOutput)
            return pathhistogramOutput

    else:
        img = Image.open(path)
        img = np.asarray(img)
        flat_image = img.flatten()
        pathhistogram=f'./static/images/output/histogram.jpg'
        plt.hist(flat_image,256)
        plt.savefig(pathhistogram)
        return  pathhistogram
```

9. Function for getting histogram

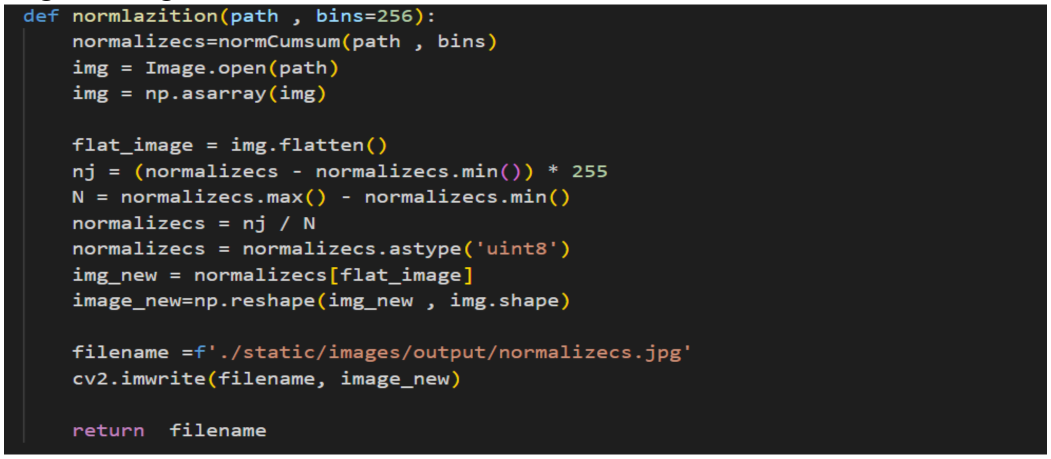
After reading and taking the image in 2D array we flatten it and then initiate arrays of zeros to fill it with pixels.



```python
def get_histogram(path, bins):
    img = Image.open(path)
    img = np.asarray(img)
    flat_image = img.flatten()
    histogram = np.zeros(bins)

    for pixel in flat_image:
        histogram[pixel] += 1

    return histogram
```

10. Normalization

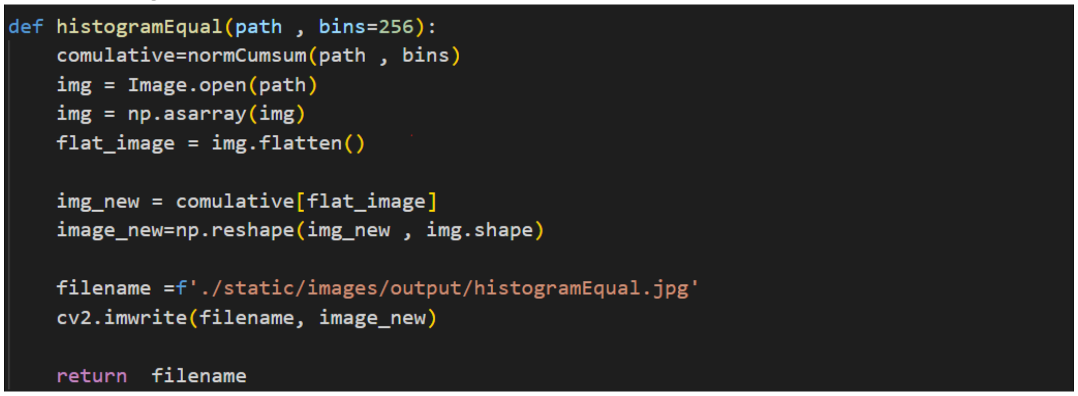
    Normalizing the values of the cumulative sum to make sure they are laying between 0-255. And then use this normalized cumulative sum to modify the intensity values of our original image.



```python
def normlazition(path , bins=256):
    normalizecs=normCumsum(path , bins)
    img = Image.open(path)
    img = np.asarray(img)

    flat_image = img.flatten()
    nj = (normalizecs - normalizecs.min()) * 255
    N = normalizecs.max() - normalizecs.min()
    normalizecs = nj / N
    normalizecs = normalizecs.astype('uint8')
    img_new = normalizecs[flat_image]
    image_new=np.reshape(img_new , img.shape)

    filename =f'./static/images/output/normalizecs.jpg'
    cv2.imwrite(filename, image_new)

    return  filename
```

11. Histogram equalization

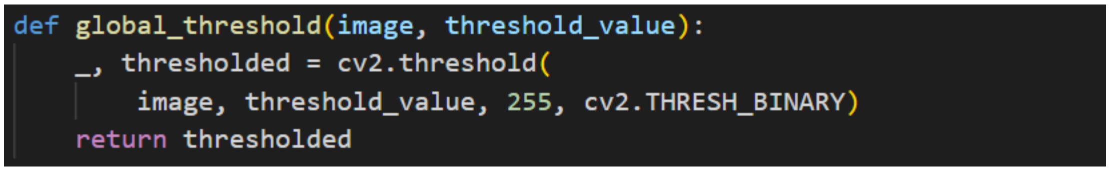
    Applying the equalization by using the flat image array as indices of the cumulative sum so that we get the new intensity value of every particular pixel.



```python
def histogramEqual(path , bins=256):
    comulative=normCumsum(path , bins)
    img = Image.open(path)
    img = np.asarray(img)
    flat_image = img.flatten()

    img_new = comulative[flat_image]
    image_new=np.reshape(img_new , img.shape)

    filename =f'./static/images/output/histogramEqual.jpg'
    cv2.imwrite(filename, image_new)

    return  filename
```

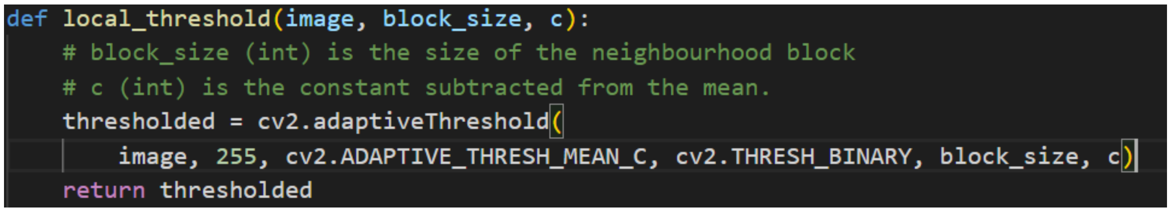
12. Local and global threshold

    Global Threshold works on the whole image and the pixels of value below the determined threshold are black (0), and the above ones are white (255).



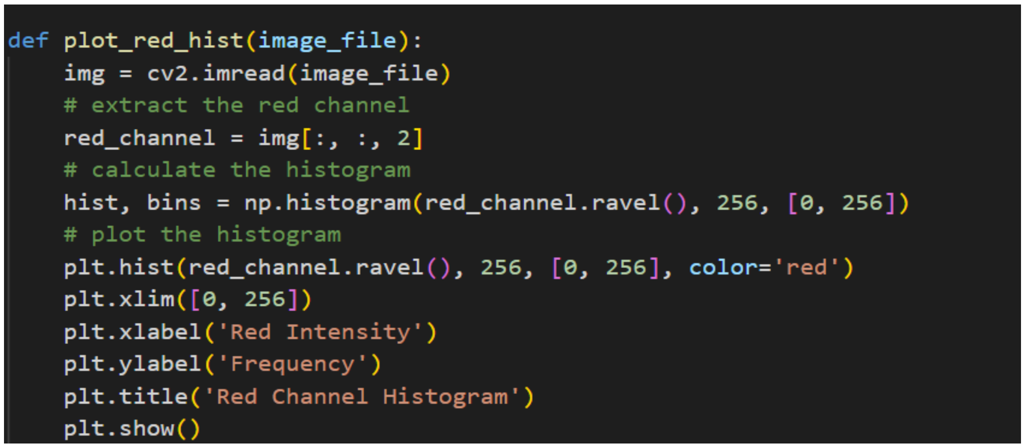
```python
def global_threshold(image, threshold_value):
    _, thresholded = cv2.threshold(
        image, threshold_value, 255, cv2.THRESH_BINARY)
    return thresholded
```

Local threshold applies the same concept but on a window or defined region of the image.

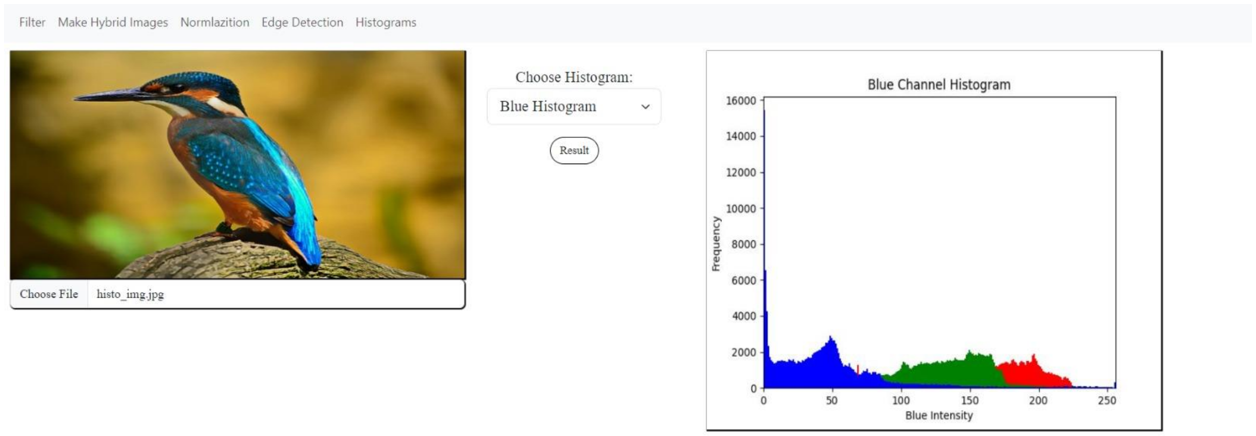


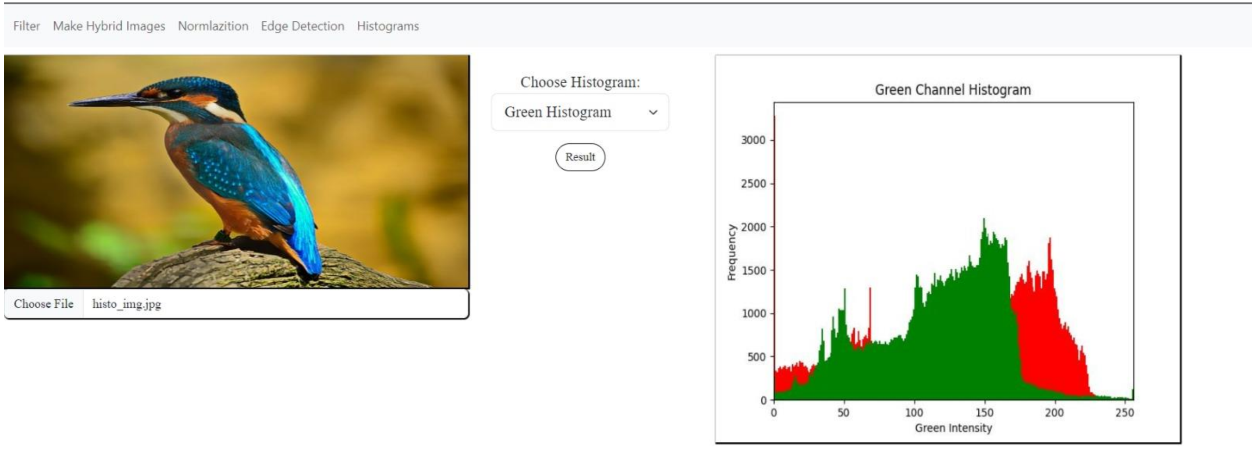
```python
def local_threshold(image, block_size, c):
    # block_size (int) is the size of the neighbourhood block
    # c (int) is the constant subtracted from the mean.
    thresholded = cv2.adaptiveThreshold(
        image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, block_size, c)
    return thresholded
```

13. RGB Histograms



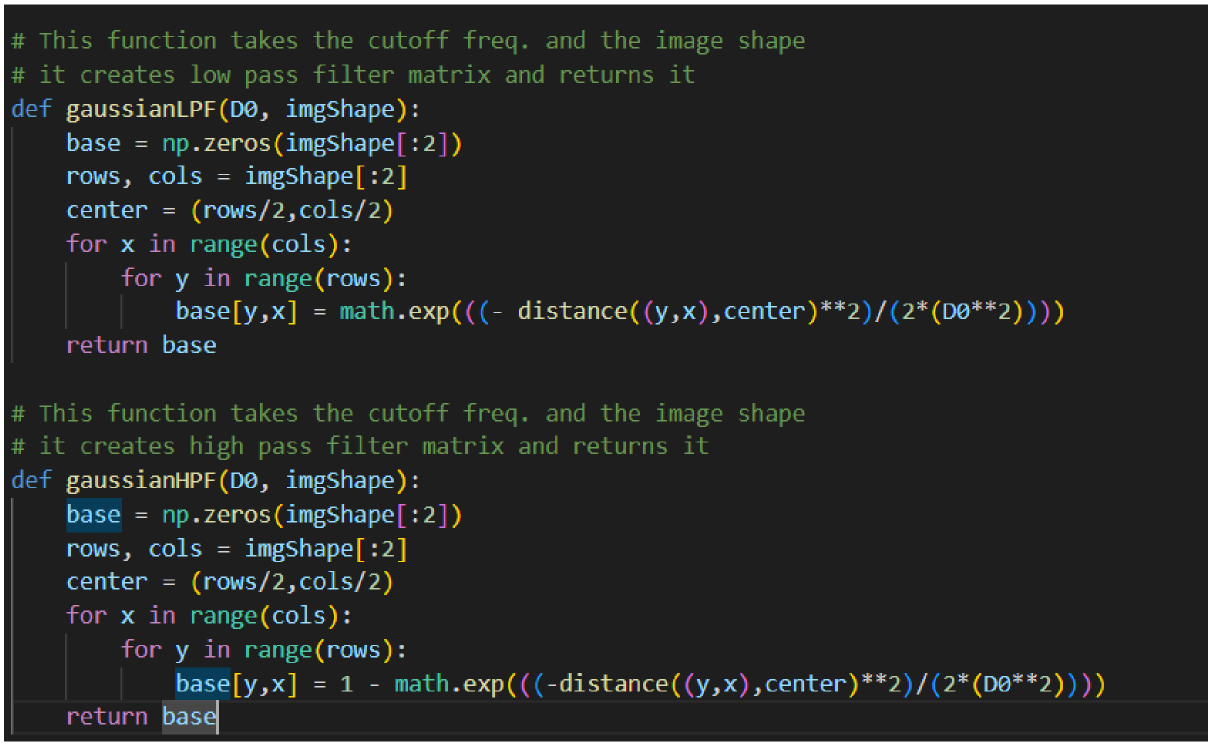
```python
def plot_red_hist(image_file):
    img = cv2.imread(image_file)
    # extract the red channel
    red_channel = img[:, :, 2]
    # calculate the histogram
    hist, bins = np.histogram(red_channel.ravel(), 256, [0, 256])
    # plot the histogram
    plt.hist(red_channel.ravel(), 256, [0, 256], color='red')
    plt.xlim([0, 256])
    plt.xlabel('Red Intensity')
    plt.ylabel('Frequency')
    plt.title('Red Channel Histogram')
    plt.show()
```





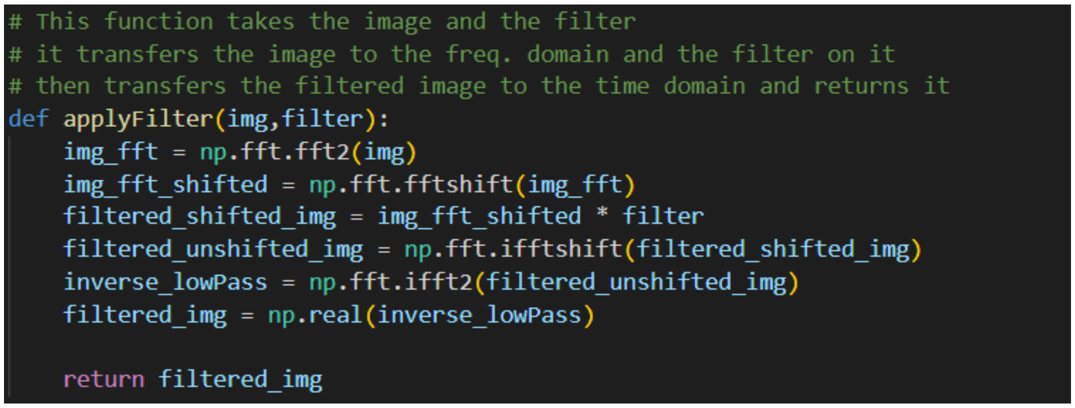
14. Low-pass, High-pass filters and Hybrid image

Getting the low-pass or high-pass filters by getting the cutoff frequency or the diameter you want to include in your result.



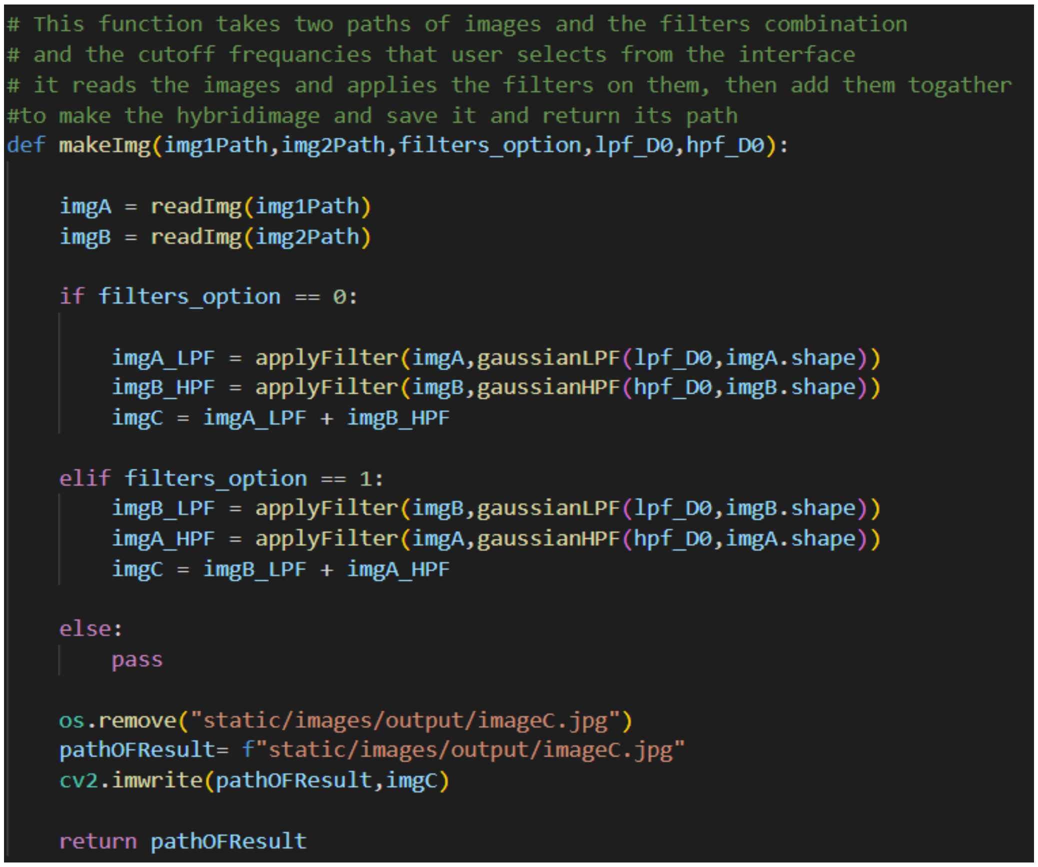
```python
# This function takes the cutoff freq. and the image shape
# it creates low pass filter matrix and returns it
def gaussianLPF(D0, imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2,cols/2)
    for x in range(cols):
        for y in range(rows):
            base[y,x] = math.exp(((- distance((y,x),center)**2)/(2*(D0**2))))
    return base

# This function takes the cutoff freq. and the image shape
# it creates high pass filter matrix and returns it
def gaussianHPF(D0, imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2,cols/2)
    for x in range(cols):
        for y in range(rows):
            base[y,x] = 1 - math.exp(((-distance((y,x),center)**2)/(2*(D0**2))))
    return base
```

Applying Fourier transform on images and shifting the values to center them and multiply the shifted image with the filter and do the inverse shifting Fourier transform



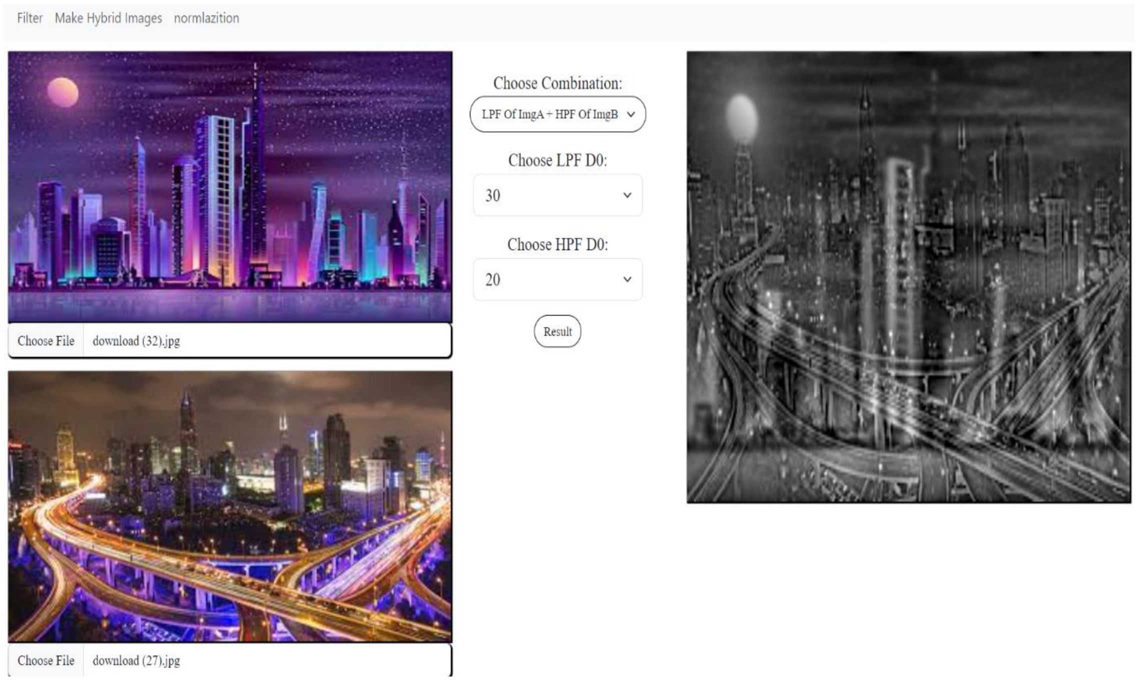
```python
# This function takes the image and the filter
# it transfers the image to the freq. domain and the filter on it
# then transfers the filtered image to the time domain and returns it
def applyFilter(img,filter):
    img_fft = np.fft.fft2(img)
    img_fft_shifted = np.fft.fftshift(img_fft)
    filtered_shifted_img = img_fft_shifted * filter
    filtered_unshifted_img = np.fft.ifftshift(filtered_shifted_img)
    inverse_lowPass = np.fft.ifft2(filtered_unshifted_img)
    filtered_img = np.real(inverse_lowPass)

    return filtered_img
```

After that we create the hybrid image by combining the high-pass of one image and the low-pass of the other.



```python
# This function takes two paths of images and the filters combination
# and the cutoff frequancies that user selects from the interface
# it reads the images and applies the filters on them, then add them togather
#to make the hybridimage and save it and return its path
def makeImg(img1Path,img2Path,filters_option,lpf_D0,hpf_D0):

    imgA = readImg(img1Path)
    imgB = readImg(img2Path)

    if filters_option == 0:

        imgA_LPF = applyFilter(imgA,gaussianLPF(lpf_D0,imgA.shape))
        imgB_HPF = applyFilter(imgB,gaussianHPF(hpf_D0,imgB.shape))
        imgC = imgA_LPF + imgB_HPF

    elif filters_option == 1:
        imgB_LPF = applyFilter(imgB,gaussianLPF(lpf_D0,imgB.shape))
        imgA_HPF = applyFilter(imgA,gaussianHPF(hpf_D0,imgA.shape))
        imgC = imgB_LPF + imgA_HPF

    else:
        pass

    os.remove("static/images/output/imageC.jpg")
    pathOFResult= f"static/images/output/imageC.jpg"
    cv2.imwrite(pathOFResult,imgC)

    return pathOFResult
```

Output of the hybrid images:



Choose File download (32).jpg

Choose File download (27).jpg

Choose Combination: LPF Of ImgA + HPF Of ImgB

Choose LPF D0: 30

Choose HPF D0: 20