Computer Vision (SBE3230)

Task 1

Name	Sec	BN
Dina Hussam Assem	1	28
Omar Ahmed Anwar	2	2
Omar Saad Mohamed	2	3
Mohamed Ahmed Ismail	2	16
Neveen Mohamed Ayman	2	49

Supervised By

Dr. Ahmed Badwy

TA: eng. Peter Emad & eng. Lila Abbas

Table of Contents

Introduction:

It's an image processing implementation functions project implemented in C++ with a desktop application (Qt) that consists of four tabs that let the user add noise to an image, filter the added noise, view different types of histograms, apply thresholds to an image, and create hybrid images.

Algorithms Implemented And How They Work:

1) Add Noise:

We implemented three types of noise:

- Uniform Noise:
  Algorithm:
  - We create a matrix the same size as the image that enters the function, and this matrix will be filled with random values using the random uniform distribution.
  - Copy the image that enters the function that we want to add noise to into the destination image (that will be the output image).
  - Nested for loop to loop in every pixel in the copied image and compare if the pixel is greater than the threshold that we entire it to the function we put in this pixel 255, but if it is smaller than the threshold put in this pixel zero.
  - Then add the matrix that we made in the first step to the destination image that we made in the second step, and then the noise is applied.

- Gaussian Noise:
  Algorithm:
  - We create a matrix the same size as the image that enters the function, and this matrix will be filled with random values using the Gaussian distribution.
  - Copy the image that enters the function that we want to add noise to into the destination image (that will be the output image).
  - Then add the matrix that we made in the first step to the destination image that we made in the second step, and then the noise is applied.

- Salt and Pepper:
  Algorithm:
  - Function parameters: image that we will apply noise to, output image that will be saved in, salt and pepper amount (0:1), and this number represents the pepper amount desired to be added, so salt will be (1-pepper).
  - Nested for loop to loop in every pixel in the image and generate a random value and compare the random value, if it is smaller than pepper we put in this pixel zero and if the random value is greater than salt we put in this pixel one, and if anything else will be the same value of pixel.

2) Filter The Noise:

We implemented three types of low pass filters:

- Average Filter:
  Algorithm:
  - Copy the image that enters the function that we want to add noise to into the destination image (that will be the output image).
  - Create Matrix with length 9 (that represents 3×3 kernel).
  - Sum the kernel values and then divide them by 9 and this represent the mean.
  - Put the mean in the middle pixel in the kernel, and the kernel will move; then, repeat the same process.

- Gaussian Filter:
  Algorithm:
  - Create array with length 9 (that represents 3×3 kernel).
  - The kernel makes convolutions with the image.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Median Filter:
  Algorithm:
  - Create array with length 9 (that represents 3×3 kernel).
  - Copy the image that enters the function that we want to add noise to into the destination image (that will be the output image).
  - Nested for loop to loop in every pixel in the image with kernel and then sort it with bubble sort, take the median of it, and put it in middle pixel in kernel and kernel move and repeat the same process.

3) Edge Detection:

Edge detection Algorithms:

We implemented four types of edge detection (Sobel, Robert , Prewitt) :

Sobel, Robert, and Prewitt Edge detection depend on convolution of image with difference filters (kernels) in the vertical and horizontal directions then calculating the magnitude gradient of vertical and horizontal gradients.

Sobel uses 3 x 3 kernels for x and y directions:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Prewitt uses 3 x 3 kernels for x and y directions:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Robert uses 2 x 2 kernels for x and y directions:

-1	0
0	+1

G_x

0	-1
+1	0

G_y

Canny Detection: Canny edge detector is probably the most used and most effective method.

It is a multi-stage algorithm:

1: Gaussian blur: First, the input image is smoothed using a Gaussian filter to remove noise and small details from the image.

2: Gradient calculation: Next, the edges in the image are identified by calculating the gradient of the image. The gradient magnitude and gradient phase (direction) are calculated at each pixel using Sobel gradient operators.

3: Non maximum suppression: In this stage, the algorithm goes through each pixel in the gradient magnitude image and suppresses non-maximum values in the direction of the gradient. The purpose of this step is to thin out the edges so that only the most prominent ones are preserved.

4: Double thresholding: The gradient magnitude image is then thresholded into two levels: a low threshold and a high threshold. Pixels with gradient magnitudes above the high threshold are considered strong edges, pixels below the low threshold are considered non-edges, and pixels with magnitudes between the two thresholds are considered weak edges.

5: Edge tracking by hysteresis: Finally, weak edges that are connected to strong edges are also considered as edges. This is done by tracing along the edges using a process called hysteresis. If a weak edge pixel is adjacent to a strong edge pixel, it is considered part of the edge, and if it is not adjacent to a strong edge pixel, it is considered noise and is discarded.

4) Draw Histogram and Distribution Curve Of Image:

Algorithm:

- In Gray Scale:
  Loop in an image for intensity from 0 to 255 and count how many pixels there are for each intensity. The count for each intensity is the y-axis, and the intensities are the x-axis. To draw the histogram, we draw the points (x, y) in a bar plot and the distribution curve in a scatter plot.
- In RGB Scale:
  The same in Gray Scale but should split the image first to 3 channels B, G and R and make the same steps for every channel.

5) Equalize The Image:

It is a method that improves the contrast in an image.

Algorithm:

- Using the calculate_histogram function, calculate the frequency of each intensity of the image.
- For loop to get the maximum frequency among pixels intensity in order to calculate probability distribution function (PDF)
- Using the map structure that is in the for loop to get the new level of intensities by multiplying the cumulative distribution function (CDF)* 255 (maximum intensity) and rounding the values, insert in the map the new gray level as a value of the old gray level as a key.
- Lastly, Nested for loop on the image and replace each intensity as a key at the map created with corresponding value at the map.

6) Normalize The Image:

Image normalization is a typical process in image processing that changes the range of pixel intensity values. Its normal purpose is to convert an input image into a range of pixel values that are more familiar or normal to the senses, hence the term normalization.

Algorithm:

$$\text{Normalized Image} = \frac{\text{Image} - \text{Image min}}{\text{Image max} - \text{Image min}} \times 255$$

7) Thresholding:

We implemented two types of Thresholding:

- Local Thresholding:
  Algorithm:
  - Copy the image that enters the function that we want to add noise to into the destination image (that will be the output image).
  - Apply padding to the copied image in all directions with pixels equal to the border pixels of the source image.
  - Create Matrix with length 9 (that represents 3×3 kernel).
  - Nested for loop to loop in every pixel in the image with kernel and take the average of it, and compare if the pixel is greater than the average, we put 255 in this pixel, but if it is smaller than the average, we put zero in this pixel in the copied image. and the kernel move and repeat the same process.
- Global Thresholding:
  Algorithm:
  - Take the input from the user.
  - Copy the image that enters the function that we want to add noise to into the destination image (that will be the output image).
  - Nested for loop to loop in every pixel in the image and compare if the pixel is greater than the threshold that we entire it to the function we put in this pixel 255, but if it is smaller than the threshold put in this pixel zero in copied image.

8) Transform RGB Image To Gray Image:

We implement weighted method, also called the luminosity method, weighs red, green, and blue according to their wavelengths.

Algorithm:

$$\text{Grayscale} = 0.299\text{R} + 0.587\text{G} + 0.114\text{B}$$

9) Frequency Domain Filters:

- Filtering in the frequency domain can be much faster – especially for large images.
- It consists of modifying the FT of an input image and then finding the IFT to get the output image.
- Mathematically it's given by:

$$G(u,v) = H(u,v)F(u,v)$$

- A low-pass filter attenuates high frequencies while passing low frequencies, and is used for blurring (smoothing) but a High-pass filter attenuates low frequencies while passing high frequencies and is used for sharpening.
- Ideal low pass (very sharp) and high pass filters have been applied in our task.
- We took the following steps in the code to get what's needed :
  - Construct a function with the low pass filter that cuts off all high-frequency components that are at a distance greater than a specified distance (D0) from the center or apply the opposite of what was said on the high pass filter.

Low Filter

$$H(u,v) = \begin{cases} 1 \text{ if } D(u,v) \le D_0 \\ 0 \text{ if } D(u,v) > D_0 \end{cases}$$

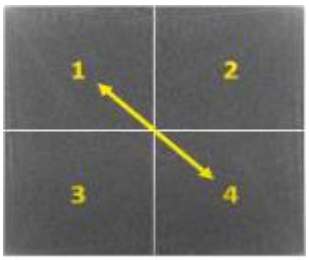
High Filter

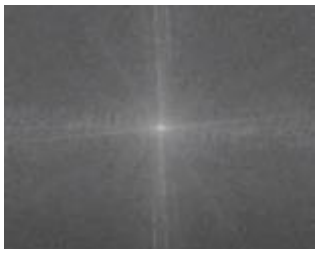
$$H(u,v) = \begin{cases} 1 \text{ if } D(u,v) \le D_0 \\ 0 \text{ if } D(u,v) > D_0 \end{cases}$$

Where:

- D(u,v) is the distance from (u,v) to the center of the frequency rectangle.
- Image size: MxN.
- Center of the frequency rectangle: (u,v) = (M/2, N/2)
- Distance to the center: D(u,v) = [(u – M/2)2 + (v – N/2)2]1/2

  - Compute the Fourier Transform of the image F(u,v).
  - Apply to shift with Fourier image.



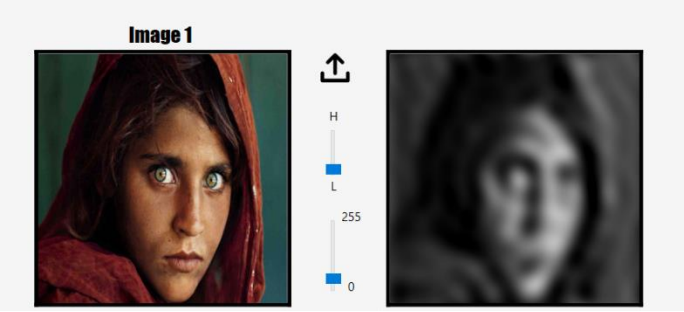


- Multiply the result by a filter transfer function.
- Take the shift back again before Inverse Transform.
- Take the inverse transform (filtered image g(x,y)).

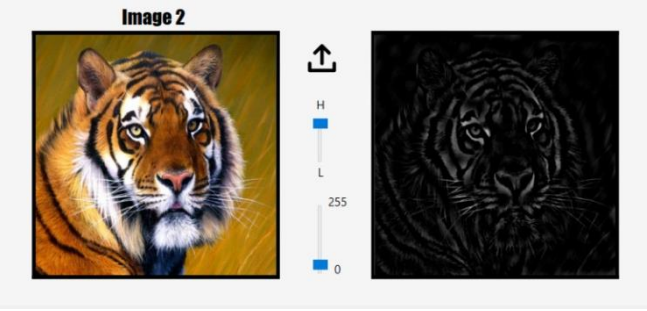
- The center of the picture always holds the most important information about it (that is, most of the picture information), while the details are distributed.
- The larger the radius of D0, the closer we get to the original:

Radius	total image power %
5	92.0
15	94.6
30	96.4
80	98
230 pixels	99.5

- The Ideal Low Pass Filter has sharp cutoffs or discontinuities which cause ringing so it has a sink function behavior in the spatial domain.
- The center of the lobe is the cause for blurring but the outer smaller lobes cause ringing so we want to achieve blurring with little ringing.



Low Pass Filter

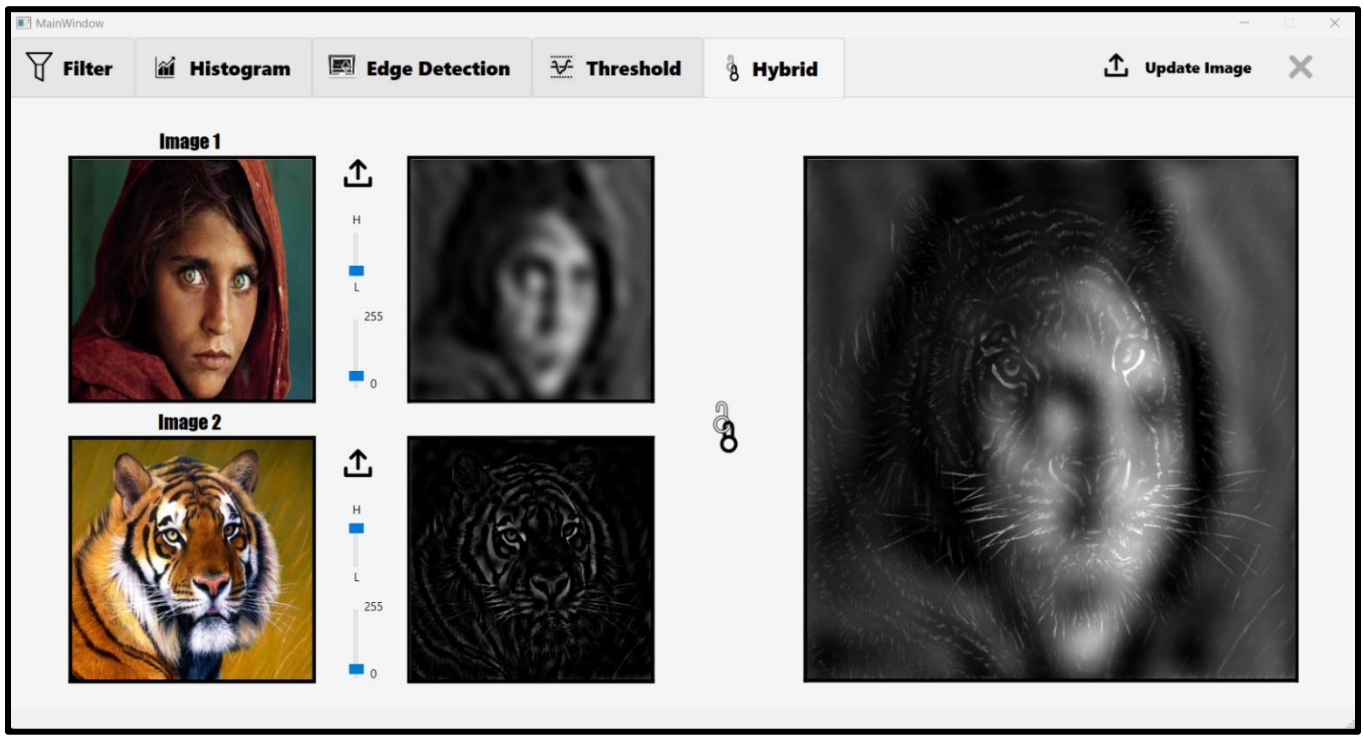


High Pass Filter

10) Hybrid Images:

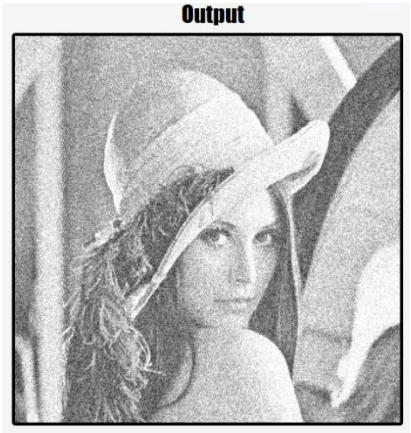
Algorithm:

It just adds the low-pass filter image to the high-pass filter image in the Fourier domain and then applies the Fourier inverse to the image.

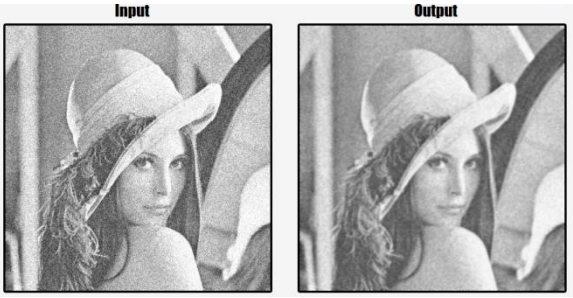
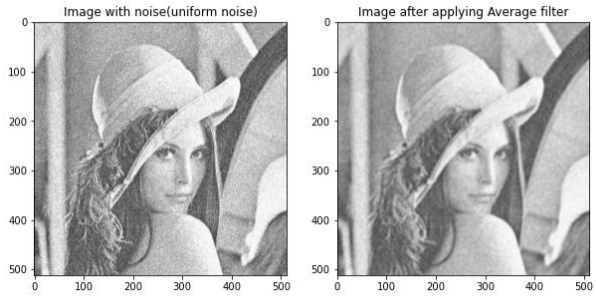
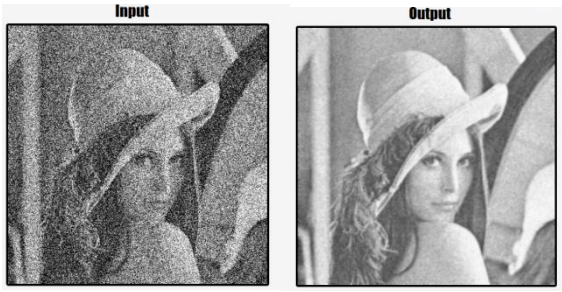
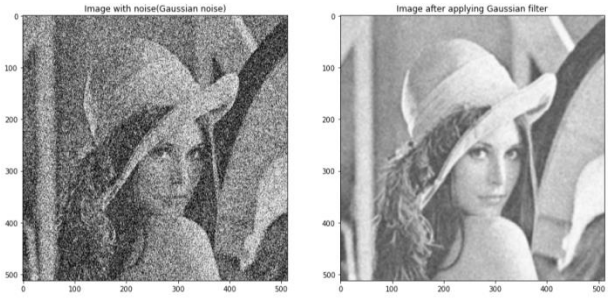
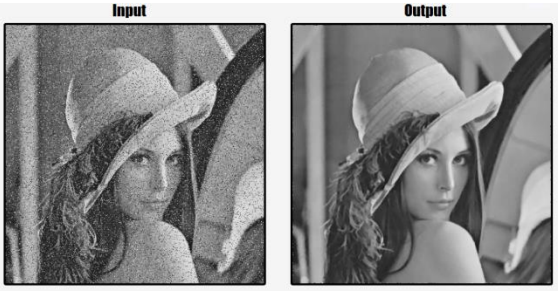
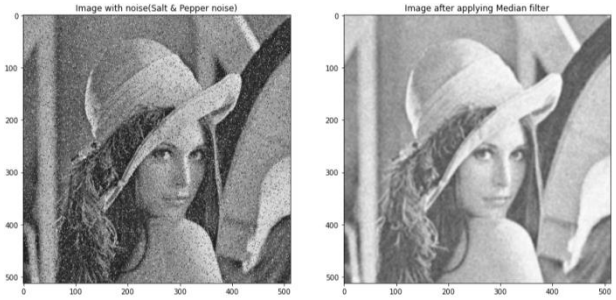


Compare To Library:

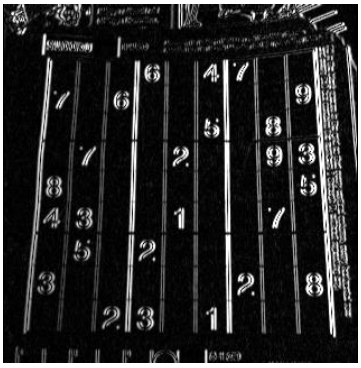
1) Add Noise:

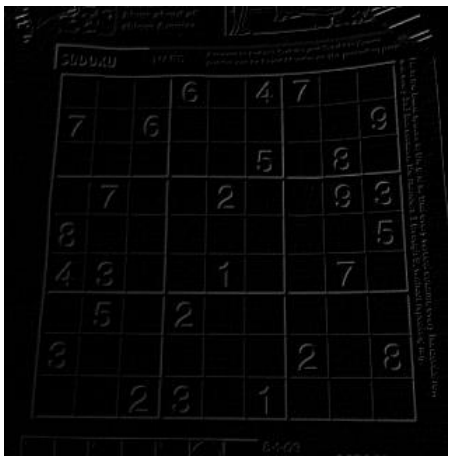
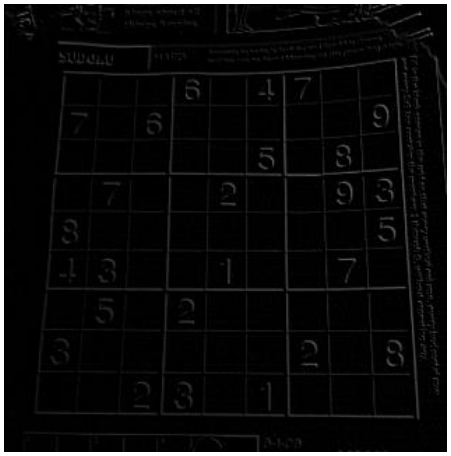
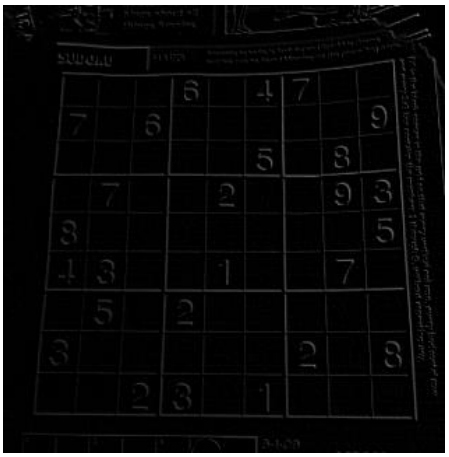
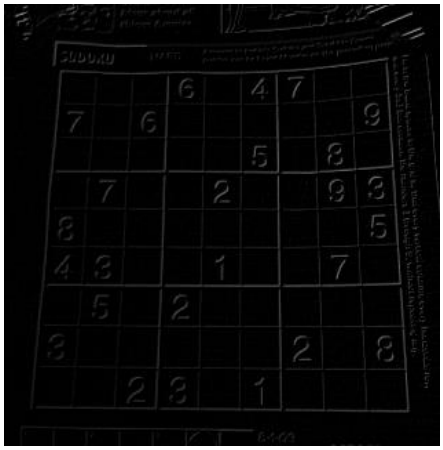
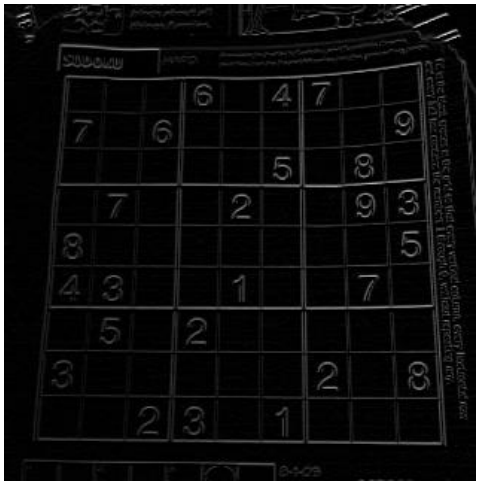
	C++ Implementation	Library
Uniform Noise		
Gaussian Noise		
Salt and Pepper		

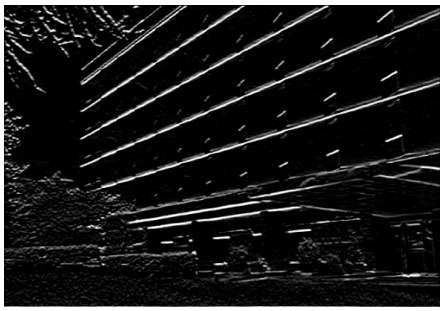
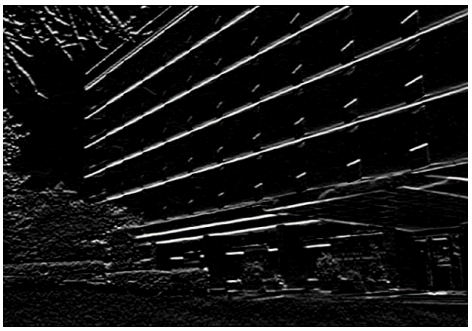
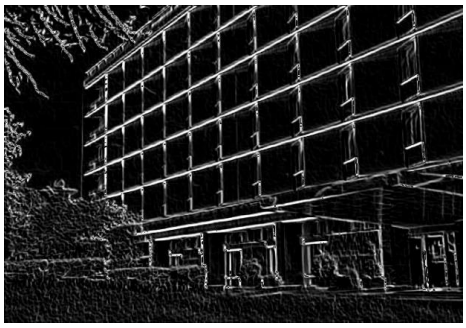
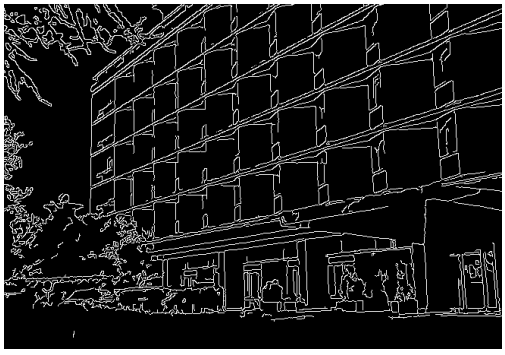
2) Filter The Noise:

	C++ Implementation	Library
Average Filter		
Gaussian Filter		
Median Filter		

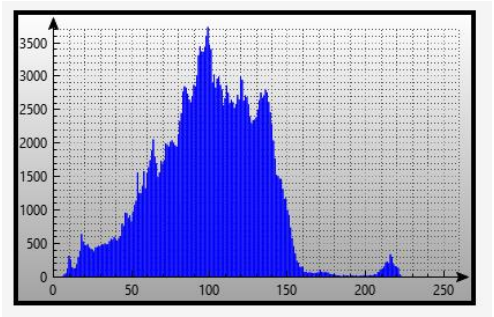
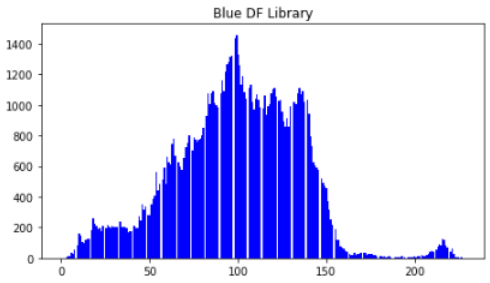
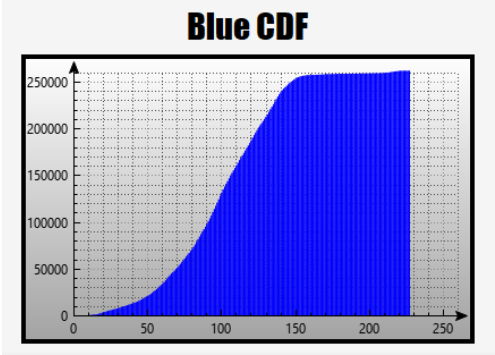
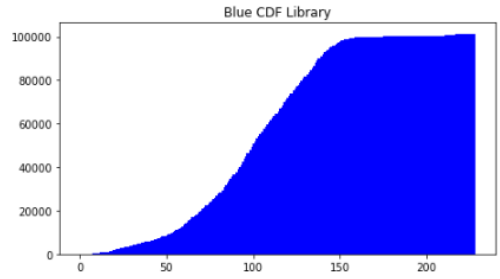
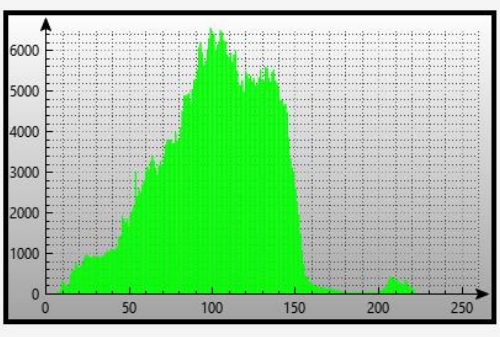
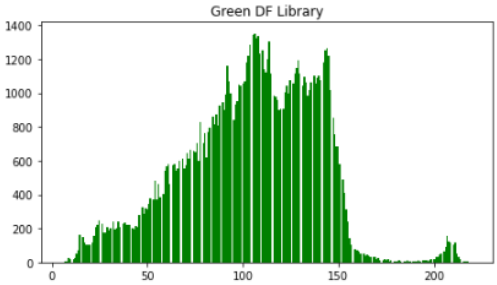
3) Edge Detection:

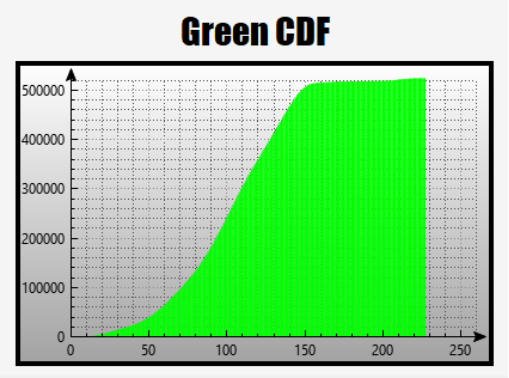
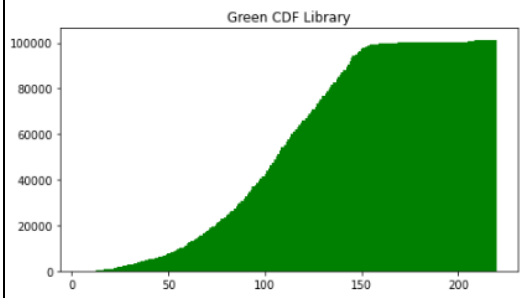
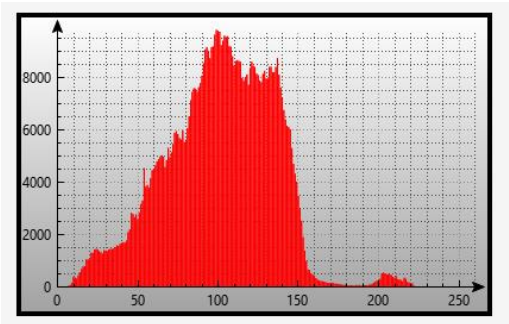
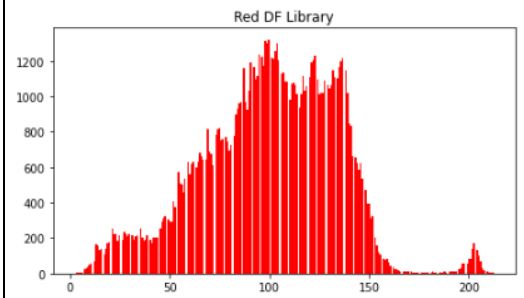
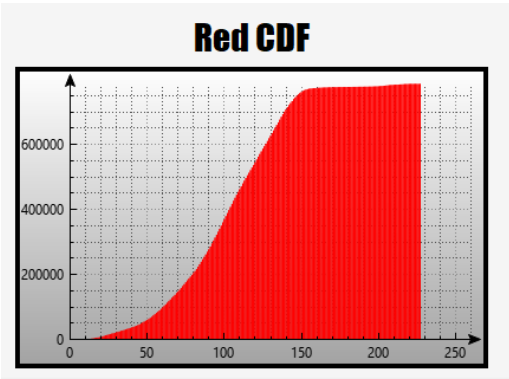
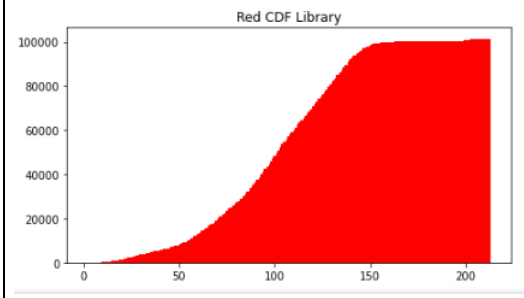
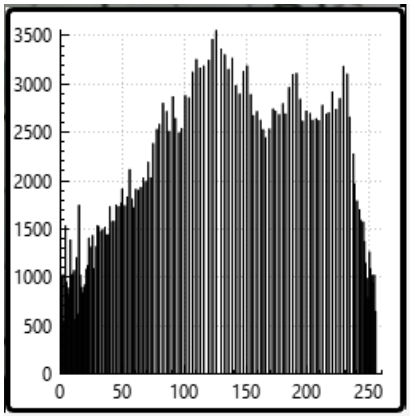
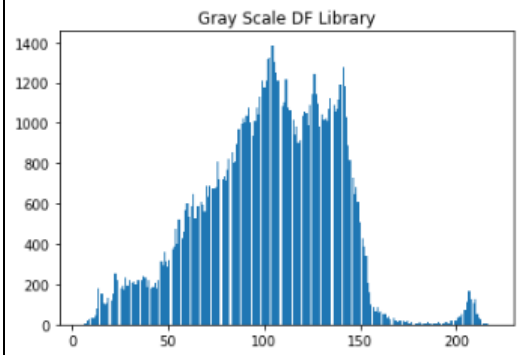
		C++ Implementation	Library
Sobel Detection	Sobel X		
	Sobel Y		
	Sobel		

Robert Detection	**Robert X**		
	Robert Y		
	Robert		

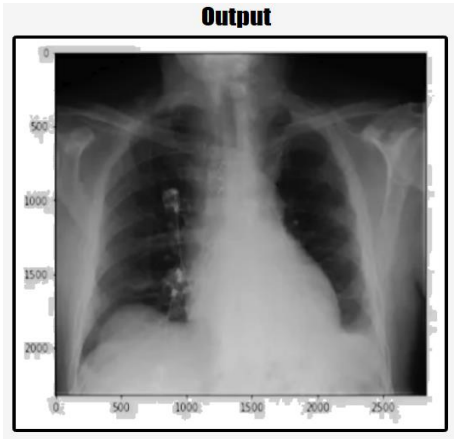
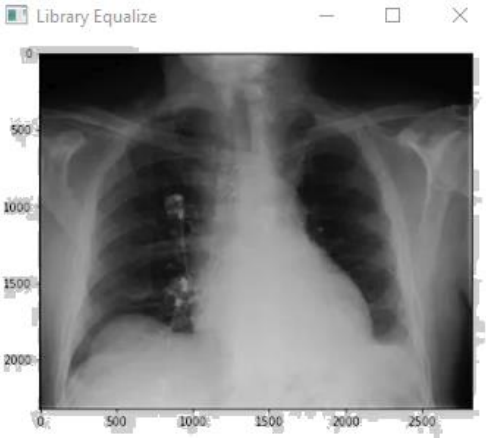
Prewitt Detection	**Prewitt X**		
	Prewitt Y		
	Prewitt		
Canny Detection			

4) Draw Histogram and Distribution Curve Of Image:

	C++ Implementation	Library
Blue DF		
Blue CDF		
Green DF		

Green CDF		
Red DF		
Red CDF		
Gray Scale DF		

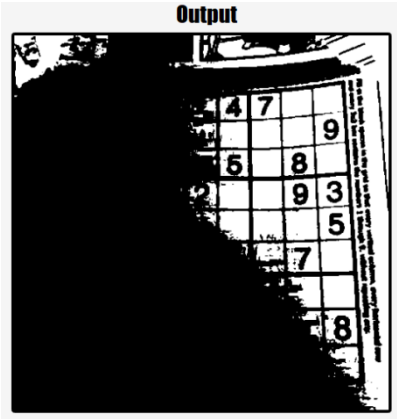
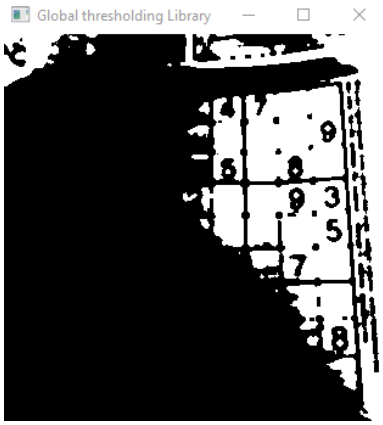
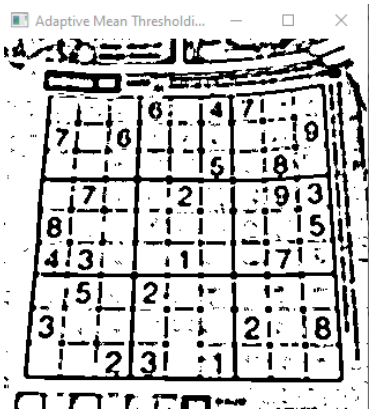
5) Equalize The Image:

	C++ Implementation	Library
Equalized Image		

6) Normalize The Image:

	C++ Implementation	Library
Normalized Image		

7) Thresholding:

	C++ Implementation	Library
Global Thresholding		
Local Thresholding		

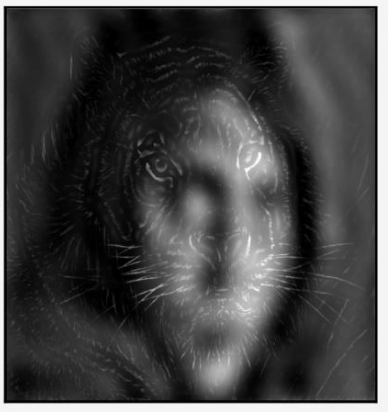
8) Transform RGB Image To Gray Image:

	C++ Implementation	Library
Gray Scale Image		

9) Frequency Domain Filters:

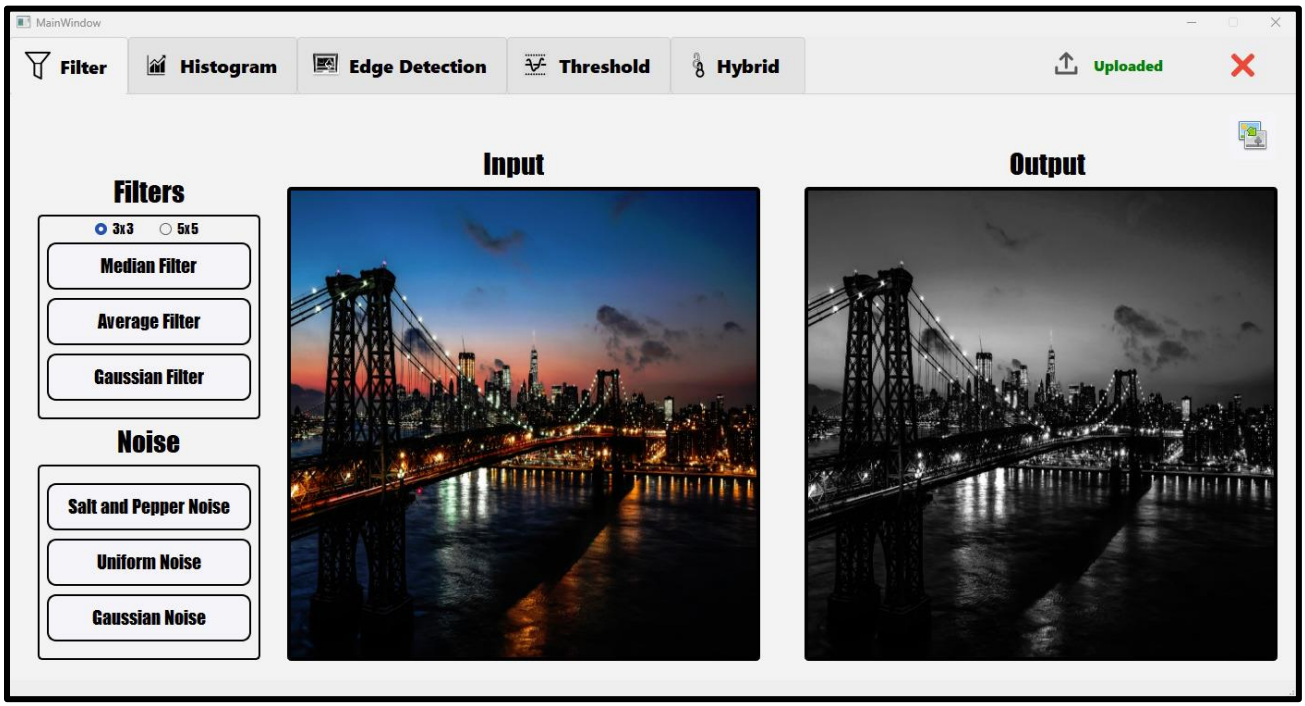
	C++ Implementation	Library
High Pass Filter		
Low Pass Filter		

10) Hybrid Images:

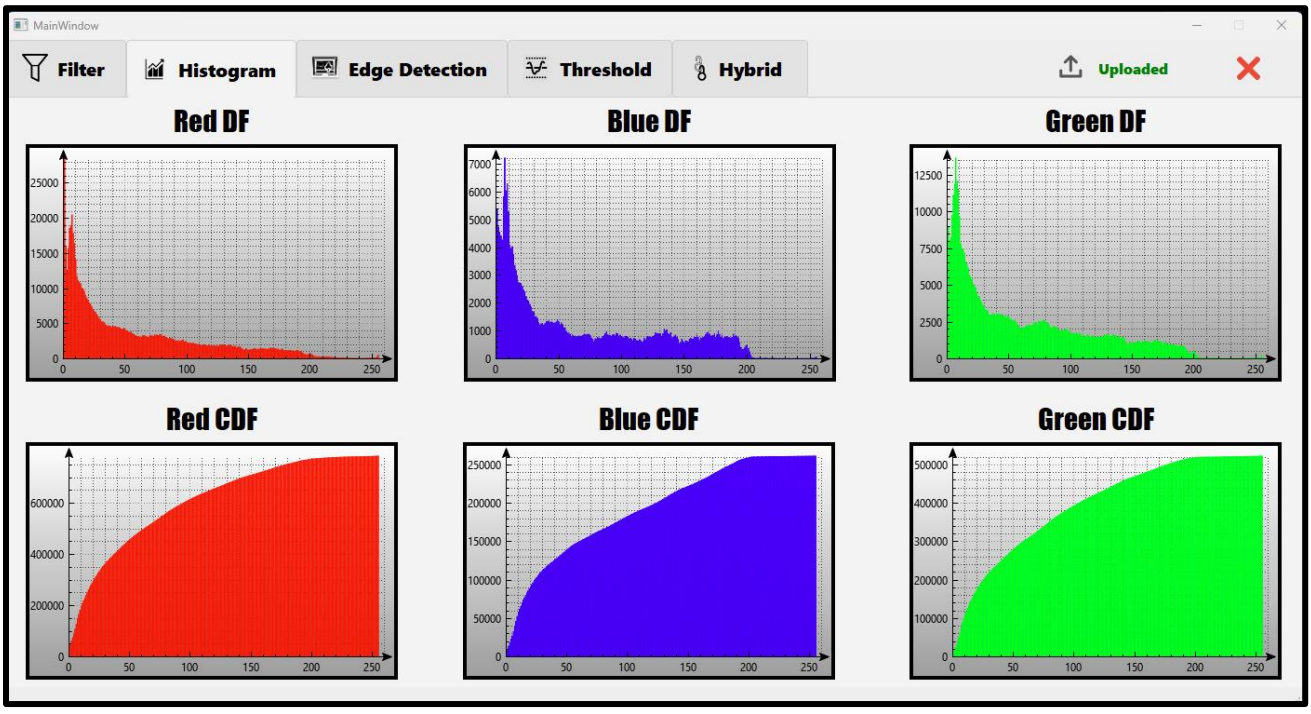
	C++ Implementation	Library
Hybrid Image		

Application Screenshots:

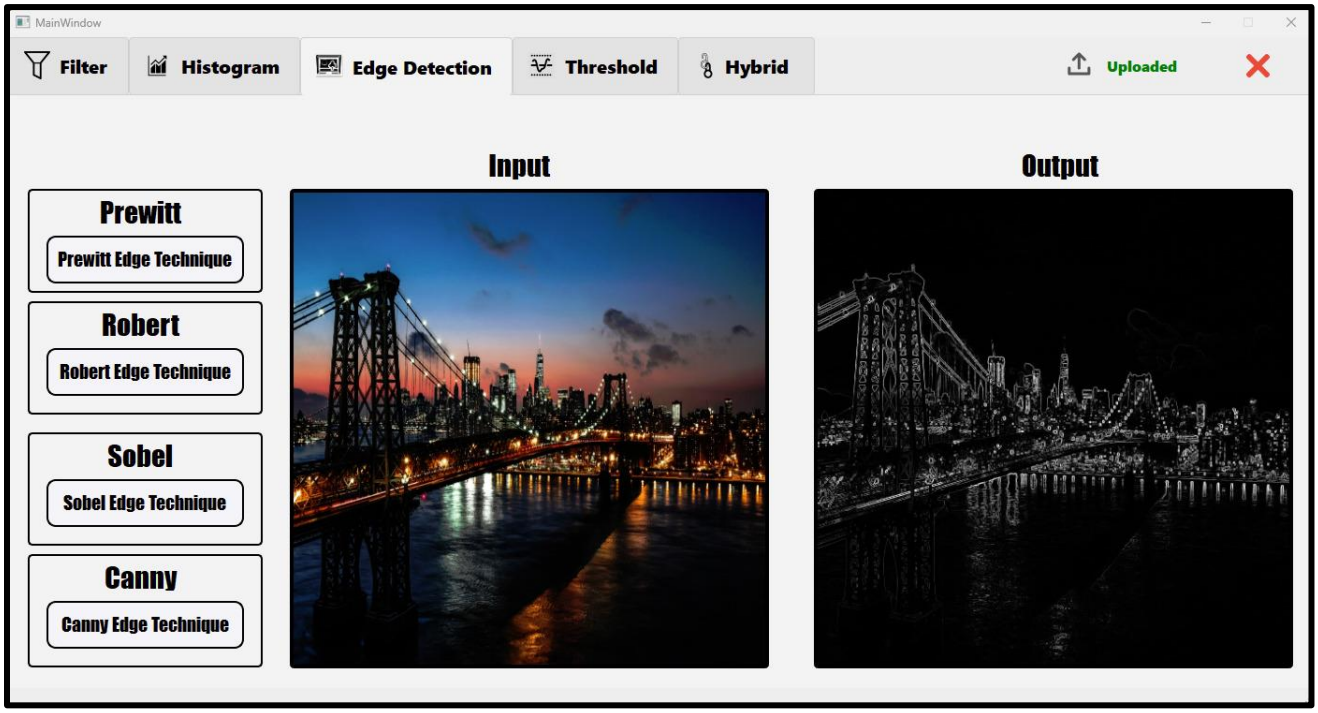
1) Filter Tab:



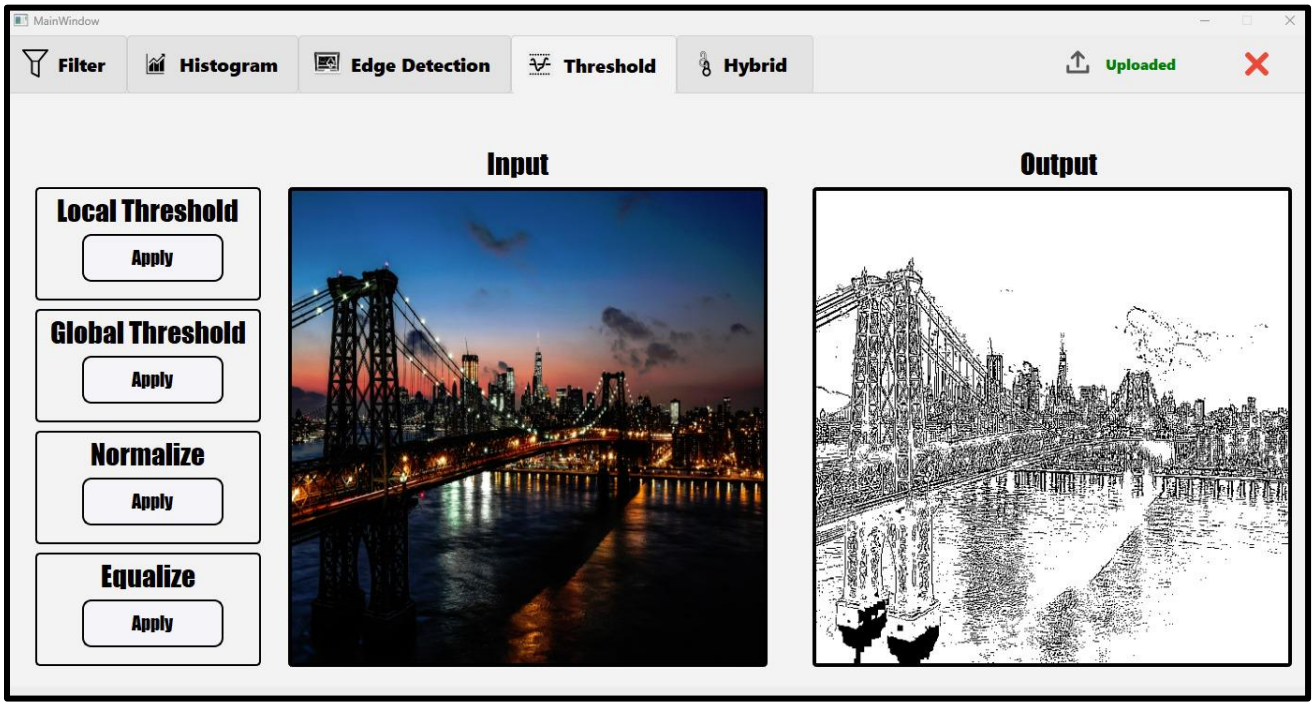
2) Histogram Tab:



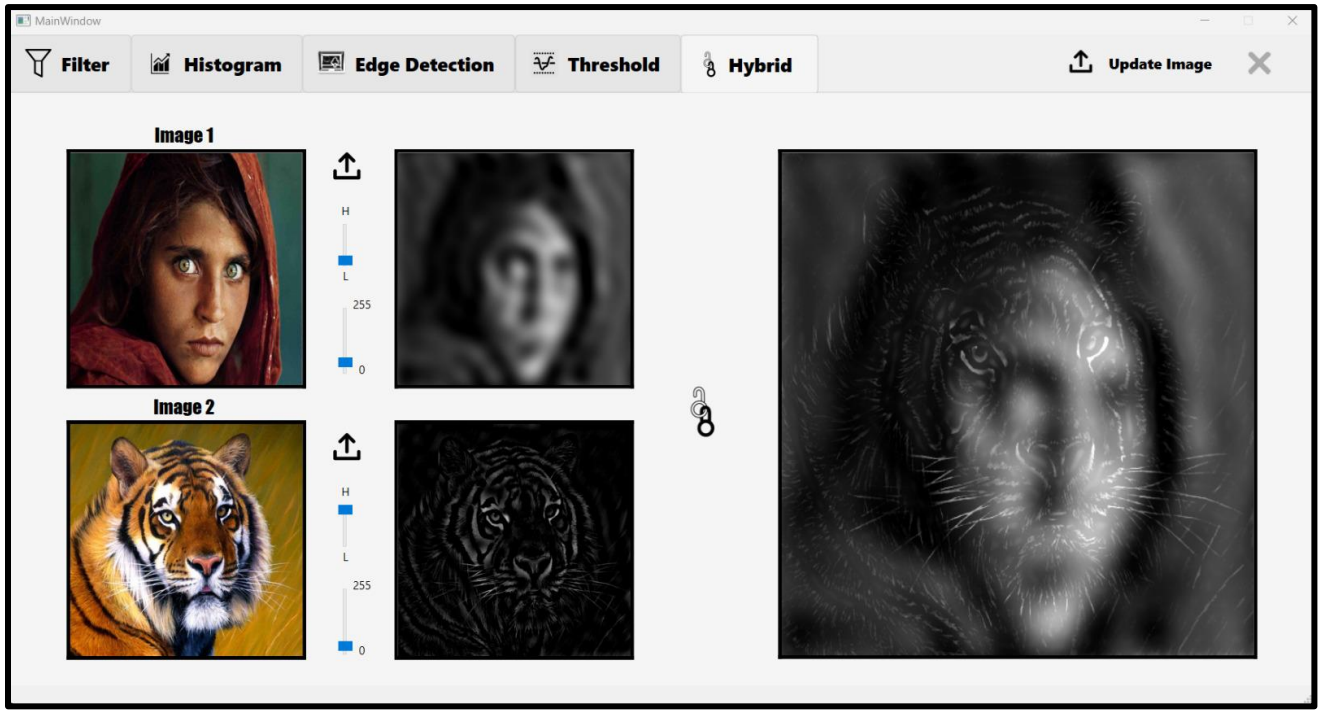
3) Edge Detection Tab:



4) Threshold Tab:



5) Hybrid Tab:


MainWindow
Filter
Histogram
Edge Detection
Threshold
Hybrid
Update Image
Image 1
H
L
255
0
Image 2
H
L
255
0