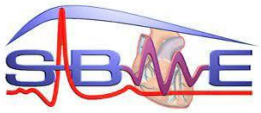
Second Year – 2nd Term

Systems & Biomedical Eng. Department

Cairo University



COMPUTER VISION
TASK 1: FILTERING AND EDGE DETECTION STUDIO REPORT

Team 5

Salma Ashraf

Sarah Mohamed

Habiba Salama

Hager Samir

Aya Eyad

March 20, 2024

Setup & Installation

To set up the Filtering and Edge Detection Studio environment, follow these steps:

1. **Install the Requirements**
   Open your terminal or command prompt and enter the following command to install the required packages: `pip install -r requirements.txt`
   This command will install the necessary libraries and dependencies for the studio environment.

2. **Run the Main Script**
   After installing the required packages, run the `main.py` file to start the Filtering and Edge Detection Studio. Use the following command: `python main.py`
   This command will execute the main script and launch the studio application.

The following libraries are included in the `requirements.txt` file and will be installed during the setup process:

- PyQt5
- pyqtgraph
- scipy
- opencv-python
- matplotlib
- numpy

Note: these libraries are essential for the functionalities and features of the Filtering and Edge Detection Studio.

Conversion to Greyscale

1. to_greyscale

Parameters:

- red_ch: a 2d numpy array containing pixel values of the red channel
- green_ch: a 2d numpy array containing pixel values of the green channel
- blue_ch: a 2d numpy array containing pixel values of the blue channel

Description:

Combines the 3 color channels into a single greyscale channel using the equation Y' = 0.2989 R + 0.5870 G + 0.1140 B, based on the way humans perceive color. The human eye is more sensitive to green light, and less sensitive to blue and red light. Therefore, when converting to grayscale, the green channel is given more weight than the red and blue channels.

Experiment:

Original image

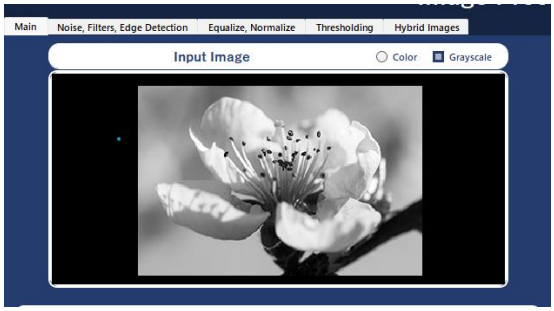


Image opened in greyscale mode

Histograms and Distribution Curves

1. get_frequencies

Parameters:

- Channel_array: A 2d array containing the pixel values

Purpose:

Counting the frequency of each pixel value to draw the histogram of one channel, to be used in both colored and greyscale histograms. It uses the numpy histogram function after flattening the array.

2. get_cumulative_frequencies

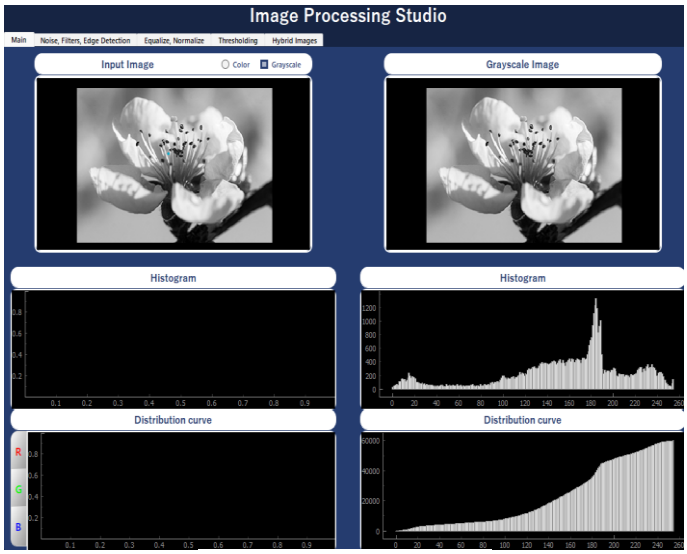
Parameters:

- frequencies: A 1d array containing the frequencies of pixel values, the output of the previous get_frequencies function.

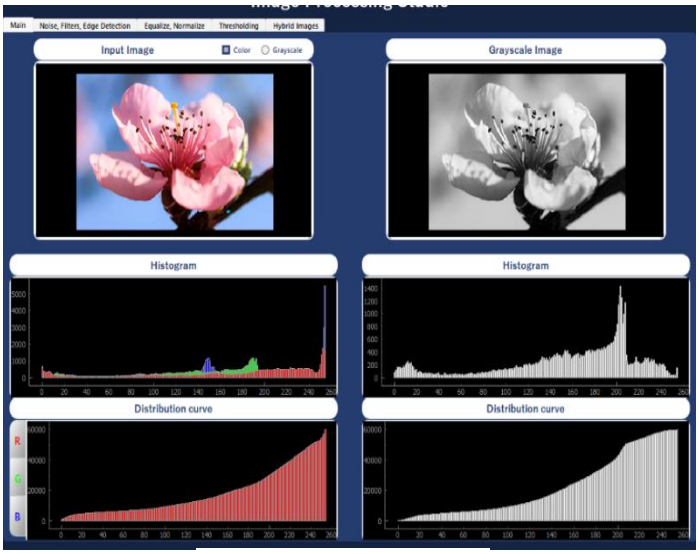
Purpose:

Accumulating the frequencies for the cumulative distrib.of a single channel.

Experiments



For a greyscale image



For a colored image

Note: The user can navigate between the cumulative distributions of the 3 color channels using buttons.

Noises

1. Uniform Noise

Parameters:

- img(ndarray): Input image to which uniform noise will be applied.
- noise_percentage(float): This parameter determines the range of the noise values. The specified range is chosen to control the intensity of the uniform noise added to the image. Higher values of 'noise_percentage' widen this range, leading to more intense noise as the possible noise values increase. Conversely, lower values of 'noise_percentage' narrow the range, resulting in less intense noise.

Description:

It's a consistent level of graininess or speckles throughout the image. The intensity of the noise will be uniform across the entire image, without any specific patterns.

Observations:

- ✓ Uniform noise appears as a constant level of noise added to the image, where each pixel's intensity is randomly altered within a specified range.
- ✓ Higher values of 'noise_percentage' widen this range, leading to more intense noise as the possible noise values increase. Conversely, lower values of 'noise_percentage' narrow the range, resulting in less intense noise.

2. Gaussian Noise

Parameters:

- image(ndarray): The input image to which Gaussian noise will be applied.
- scale(float): The standard deviation of the Gaussian noise distribution. A higher scale value results in more intense noise.

Description:

It's a subtle, smooth variation in pixel intensities. The noise tends to blend with the original image, resulting in a slightly blurred or softened appearance rather than distinct graininess.

Observations:

- ✓ This noise follows a normal distribution and appears as a smooth, continuous variation in pixel intensities around the original values.
- ✓ It's like a soft blur or fuzziness added to the image.
- ✓ It's not as harsh as uniform noise and creates a smoother transition between pixel values.

3. Salt-and-Pepper Noise

Parameters:

- img: ndarray - Input image (2D or 3D array) to which salt-and-pepper noise will be applied.
- noise_percentage: float - Percentage of pixels to be affected by the noise. Higher values of noise_percentage mean more pixels will be altered by the salt-and-pepper noise.

Description:

It's isolated white and black pixels randomly distributed across the image. These pixels represent extreme outliers compared to the surrounding pixels, leading to a speckled appearance with noticeable black and white dots.

Observations:

- ✓ Random white and black pixels (high- and low-intensity outliers) are scattered throughout the image.

✓ It's like sprinkling salt and pepper on your image.

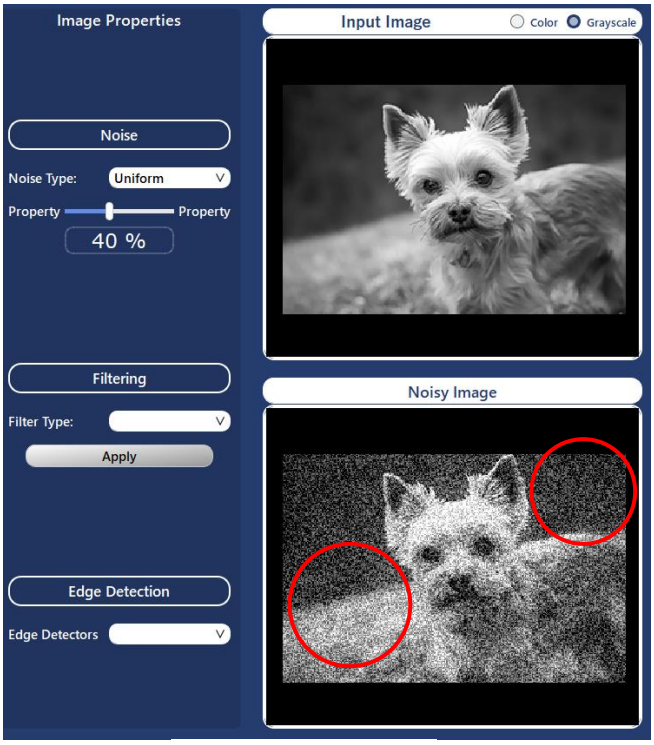


Fig.1 *40% uniform*

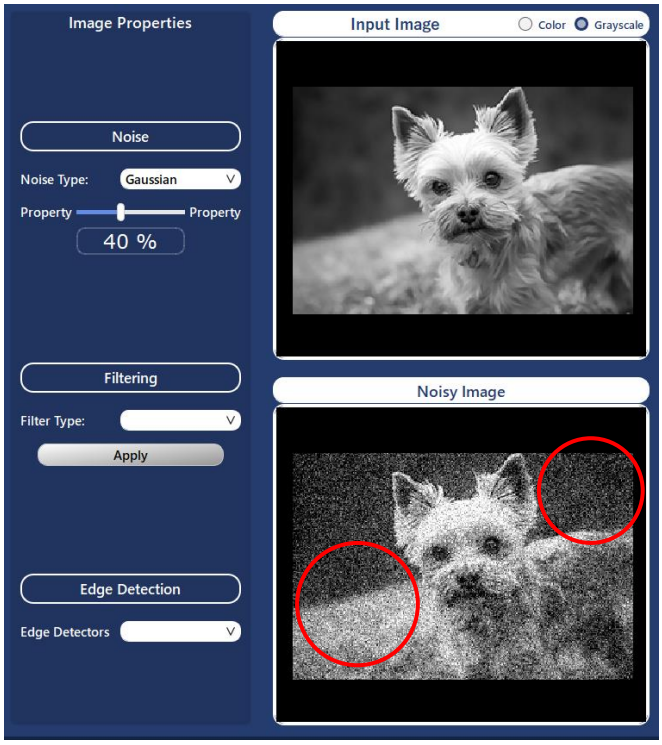


Fig.2 *40% Gaussian noise*

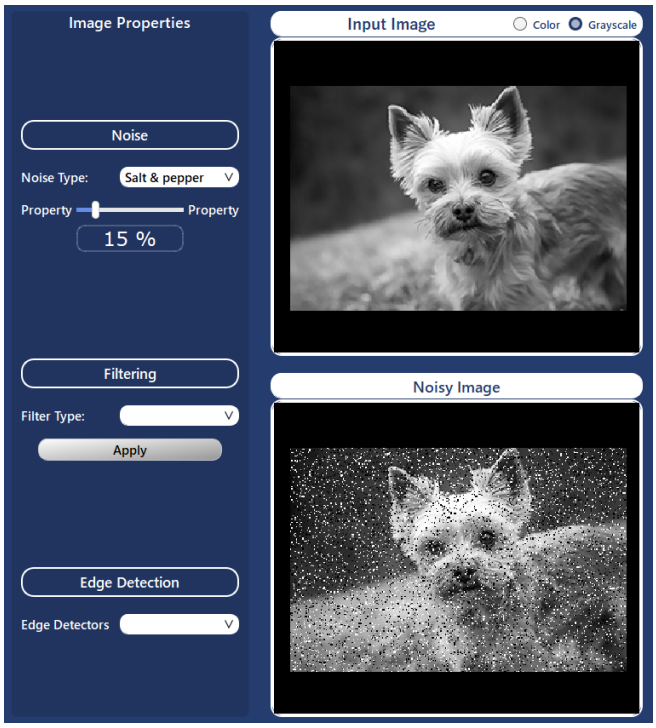


Fig.3 *15% Salt-and-Pepper noise*

Fig.1, **Fig.2,** and **Fig.3** observe three distinct types of noise affecting the grayscale pictures of a dog. **Fig.1** is subjected to *uniform noise*, resulting in a consistent grainy overlay that maintains the overall shape but adds a textured appearance. In contrast, **Fig.2** is affected by *Gaussian noise*, where the irregular distribution of noise creates a more disruptive visual effect, obscuring finer details and resulting in a less uniform presentation. **Fig.3** is subjected to *salt-and-pepper noise* introducing random white and black pixels, disrupting the smoothness and causing noticeable speckles. This makes the gradients or smooth transitions appear noisier and less uniform. By comparing the highlighted areas in **Fig.1** and **Fig.2**, we notice that *Gaussian noise* slightly smoothes out gradients and transitions, making them appear less sharp but without introducing distinct graininess like *uniform noise*.

Filters

1. convv

Parameters:

- img: The input image to be convolved.
- kernel_type: The convolution kernel used for the convolution operation.

Purpose:

These two parameters are essential for performing convolution. The img parameter represents the image on which the convolution operation will be applied, while the kernel_type parameter specifies the convolution kernel, which determines the behavior of the convolution operation.

Observations:

- ✓ Utilizes nested loops to iterate over image pixels and perform the convolution operation.
- ✓ Implements zero-padding to handle border pixels.

2. gaussian_kernel

Parameters:

- kernel_size (int): The size of the kernel.
- sigma (float): The standard deviation of the Gaussian distribution.

Purpose:

Generates a Gaussian kernel for Gaussian filtering.

Observations:

- ✓ Constructs the Gaussian kernel using the provided size and standard deviation.
- ✓ Utilizes nested loops to compute the Gaussian weights.

3. Gaussian_filter

Parameters:

- img (numpy.ndarray): The input image.
- kernel_size (int): The size of the median filter kernel.
- sigma (float): The standard deviation of the Gaussian distribution.
- is_gray (bool, optional): Indicates whether the image is grayscale.

Description:

The Gaussian filter applies a convolution operation using a Gaussian kernel. It is commonly used for smoothing or blurring images while preserving edges.

Suitability for Noise:

- Gaussian Noise: Effective in reducing Gaussian noise as it blurs the image, smoothing out pixel intensity variations.
- Uniform Noise: This can reduce uniform noise to some extent but may not be as effective as it is primarily designed for Gaussian noise.
- Salt-and-Pepper Noise: Not the most suitable choice for removing salt-and-pepper noise, as it tends to blur sharp variations rather than remove isolated high-intensity spikes.

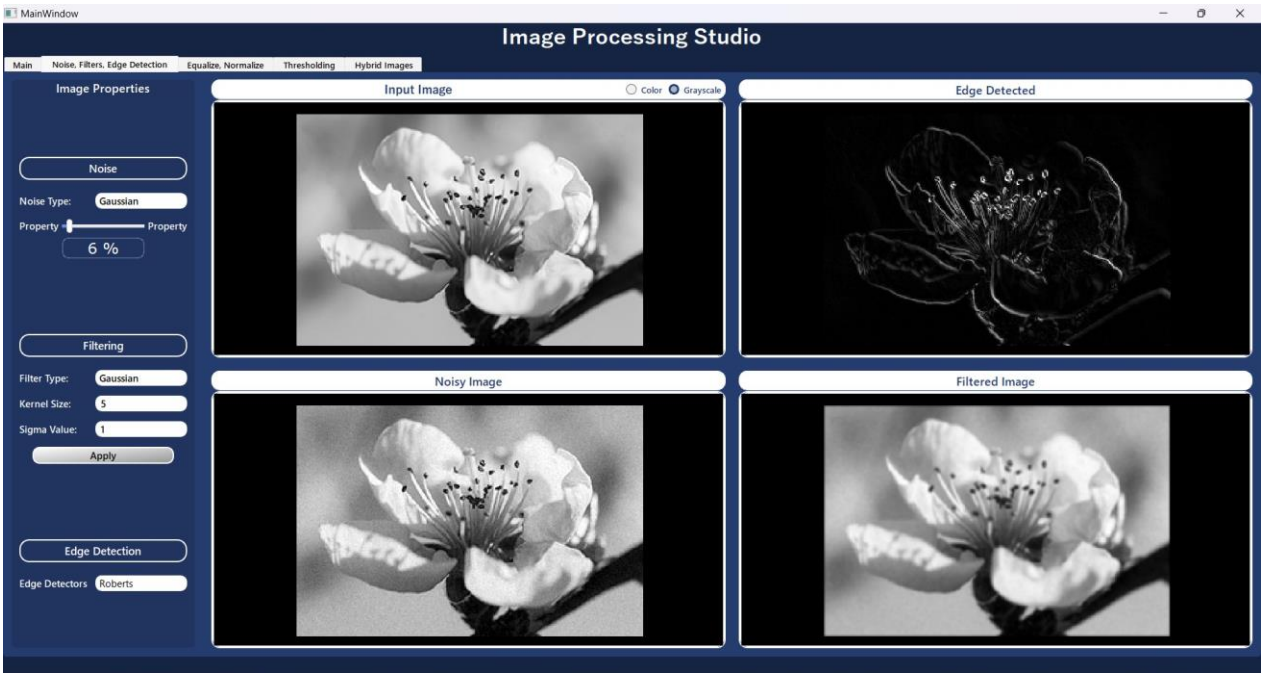
Observations:

- ✓ Perfect for removing Gaussian Noise
- ✓ As Kernel_size increases the blurring of the image increase
- ✓ As sigma increases more blurred and filtered image

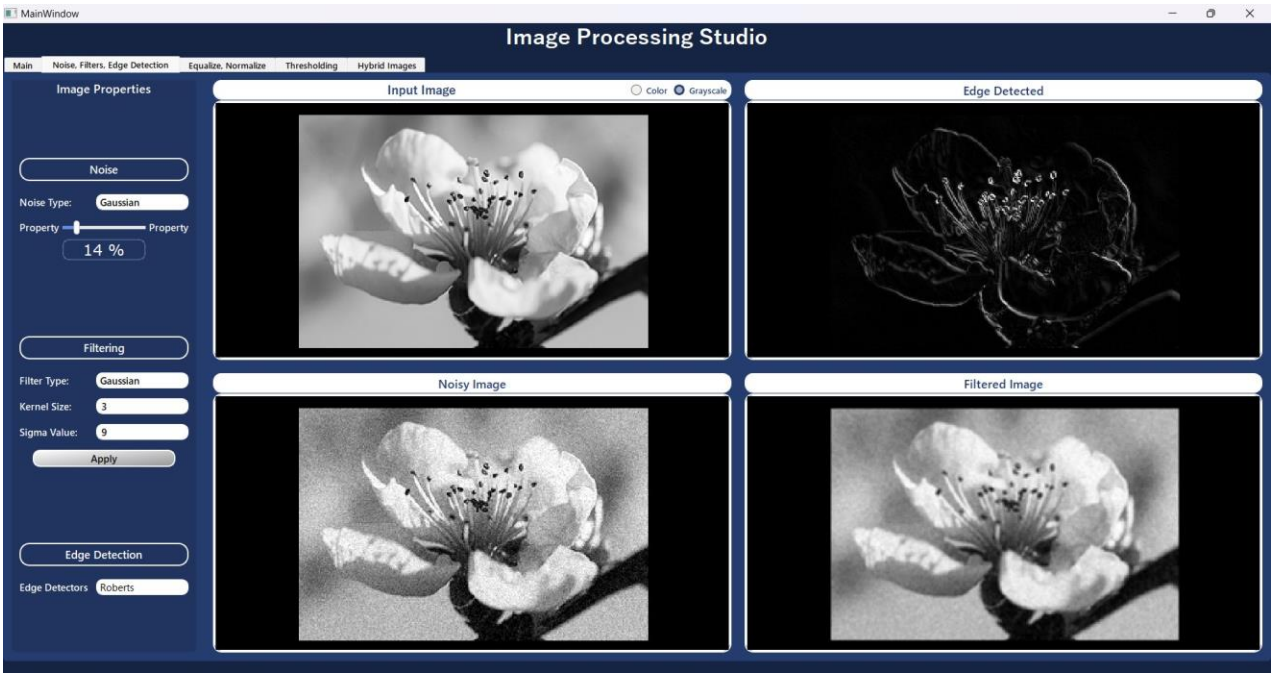
Experiments:

- **Apply on Gaussian noisy image with Kernel_size 3 & Sigma 1:**

- **Apply on Gaussian noisy image with Kernel_size 5& Sigma 1(more blurred):**



➢ **Apply on Gaussian noisy image with Kernel_size 3& Sigma 9(more filtered):**



4. Avg_filter

Parameters:

- img (numpy.ndarray): The input image.
- kernel_size (int): The size of the median filter kernel.
- is_gray (bool, optional): Indicates whether the image is grayscale.

Description:

Applies average filtering to the input image. The average filter calculates the average intensity within a kernel window and replaces the central pixel's intensity with the computed average.

Suitability for Noise:

- Gaussian Noise: Effective in reducing Gaussian noise by averaging pixel intensities, leading to smoother transitions between neighboring pixels.
- Uniform Noise: Suitable for reducing uniform noise as it averages out pixel intensity variations within the kernel window.
- Salt-and-Pepper Noise: Can partially remove salt-and-pepper noise by averaging neighboring pixel intensities, but may not completely eliminate isolated high-intensity spikes.

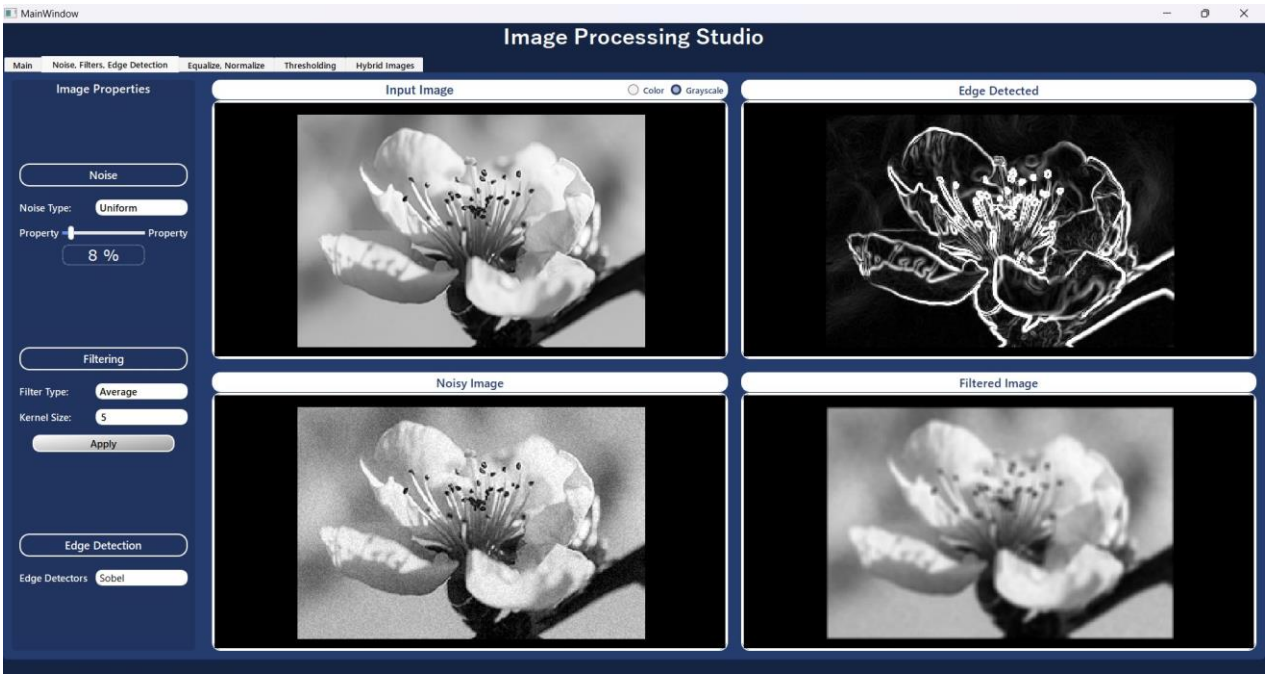
Observations:

- ✓ Perfect for removing Gaussian & Uniform noise
- ✓ As Kernel_size increases blurring of image increase

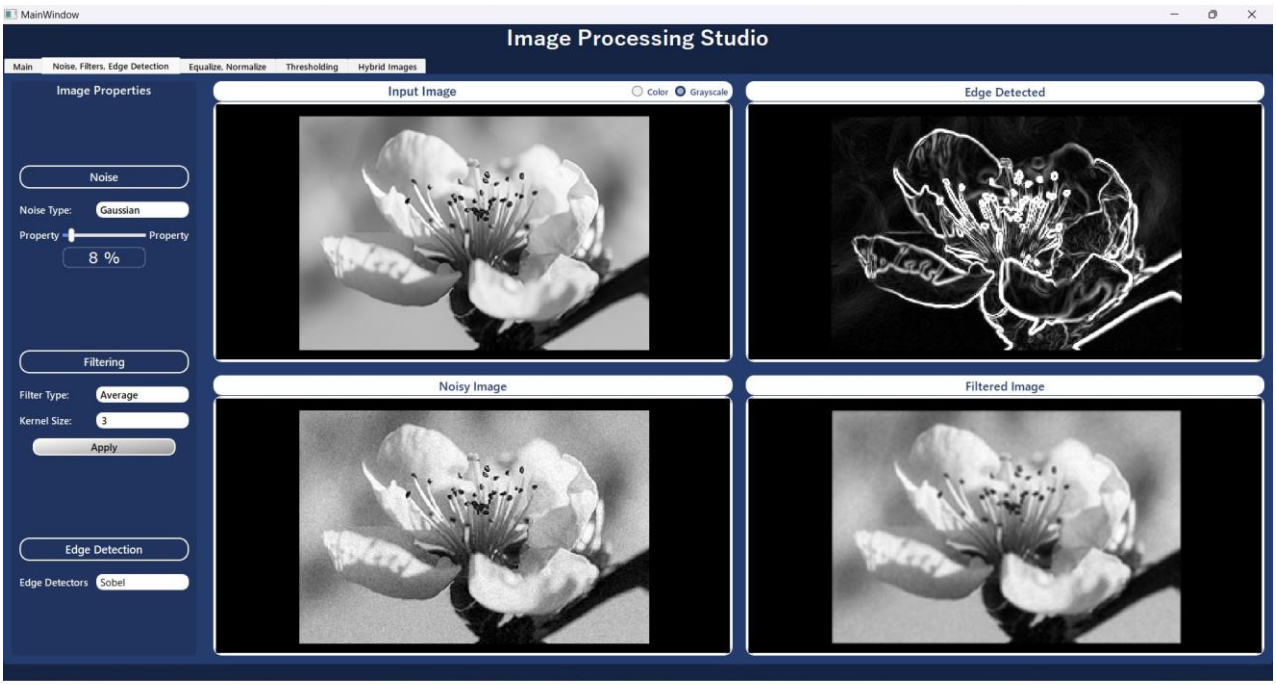
Experiments:

- **Apply on Uniform noisy image with Kernel_size 3:**

- **Apply on Uniform noisy image with Kernel_size 5(more blurred):**



- **Apply on Gaussian noisy image with Kernel_size 3:**



5. Median_filter

Parameters:

- img (numpy.ndarray): The input image.
- kernel_size (int): The size of the median filter kernel.
- is_gray (bool, optional): Indicates whether the image is grayscale.

Description:

Applies median filtering to the input image. The median filter replaces each pixel's intensity with the median value of the pixel intensities within a kernel window.

Suitability for Noise:

- Gaussian Noise: Moderately effective in reducing Gaussian noise while preserving edges, as it replaces each pixel with the median intensity of its neighbors.
- Uniform Noise: Effective in reducing uniform noise by replacing pixel intensities with the median value within the kernel window.
- Salt-and-Pepper Noise: Highly effective in removing salt-and-pepper noise, as it replaces outlier pixel intensities with the median value of neighboring pixels, effectively smoothing out isolated spikes.

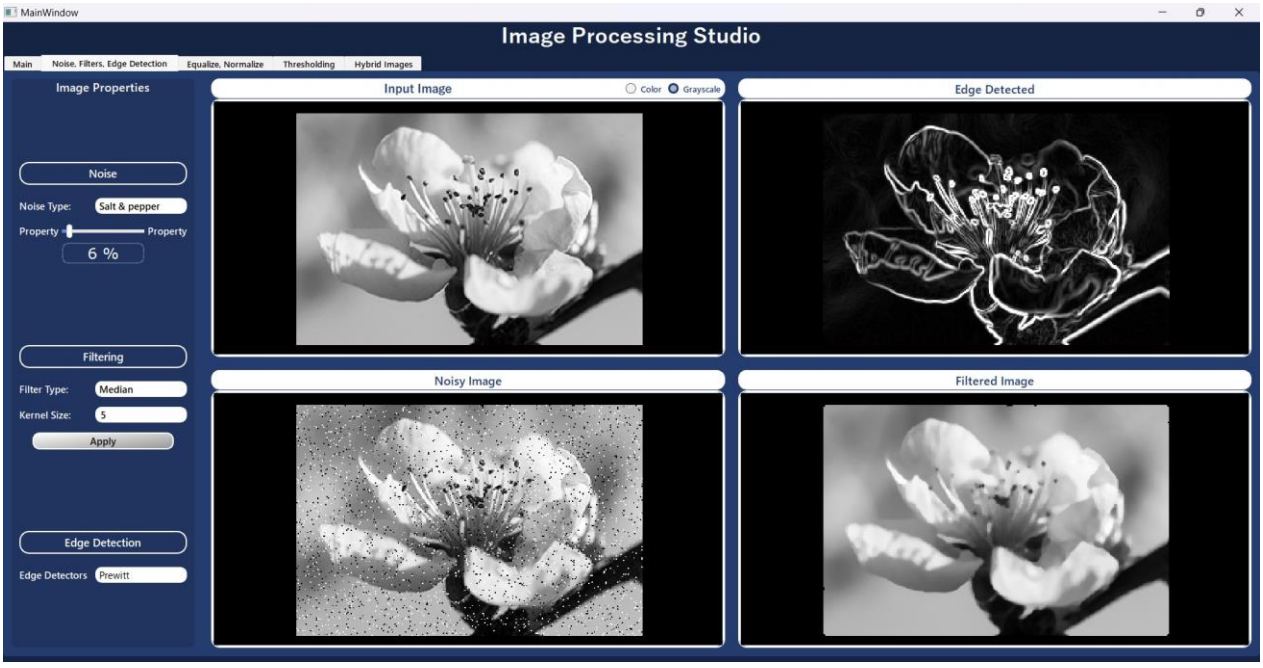
Observations:

- ✓ Utilizes the np.median function to compute the median value within the filter window.
- ✓ Perfect for removing Salt & Pepper noise
- ✓ As Kernel_size increases blurring of image increase

Experiments:

- **Apply on Salt&Pepper noisy image with Kernel_size 3:**

- **On Salt&Pepper noisy image with Kernel_size 5(more blurred):**



Conclusion for Filters:

- **For Gaussian Noise:** Both Gaussian and Average filters can be effective, but Gaussian filter is more commonly used.
- **For Uniform Noise:** Average filter is recommended.
- **For Salt-and-Pepper Noise:** Median filter is highly recommended for its superior performance in removing isolated high-intensity spikes.

Edge Detection

1. Apply_kernel

Parameters:

- img: The input image.
- horizontal_kernel: The horizontal kernel for convolution.
- vertical_kernel: The vertical kernel for convolution.
- is_gray (optional): Indicates whether the image is grayscale. Defaults to True.
- return_edge (optional): Whether to return the edges of the image with the magnitude of the image.

Description:

Applies the given horizontal and vertical kernels to the image for edge detection using convolution. It calculates the magnitude of edges.

Observation:

- ✓ This function is a fundamental step in edge detection algorithms. It performs convolution with the given kernels to detect horizontal and vertical edges in the image.

2. Prewitt_edge

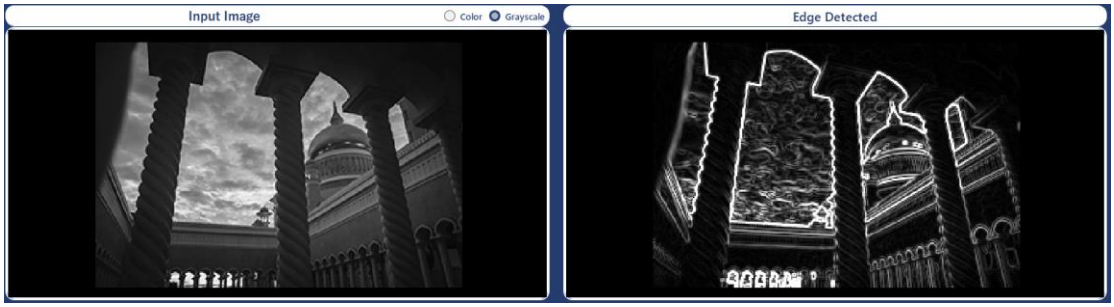
Parameters:

- img: The input image.
- is_gray (optional): Indicates whether the image is grayscale.

Description:

Applies Prewitt edge detection to the image, which is a simple edge detection method based on convolution with predefined (1st derivative) kernels. Prewitt edge detection emphasizes edges in images by calculating the gradient magnitude in both horizontal and vertical directions.

Experiment:



3. Roberts_edge

Parameters:

- img: The input image.
- is_gray (optional): Indicates whether the image is grayscale.

Description:

Applies Roberts edge detection to the image, another simple edge detection method based on convolution with predefined kernels. Roberts edge detection calculates the gradient using a pair of 2x2 convolution kernels to detect edges.

Experiment:

4. Sobel_edge

Parameters:

- img: The input image.
- is_gray (optional): Indicates whether the image is grayscale.
- get_magnitude (optional): Whether to return the magnitude of gradients.
- get_direction (optional): Whether to return the direction of gradients.

Description:

Applies Sobel edge detection to the image. It calculates the gradient magnitude and optionally the direction. Sobel edge detection is widely used due to its effectiveness in detecting edges with reduced sensitivity to noise.

Experiment:

Conclusion For Edges:

- Each edge detection method has its unique characteristics and advantages.
- **Prewitt and Roberts methods:** are simpler and computationally efficient.
- **Sobel method:** stands out for its robustness against noise.
- Depending on the specific requirements of the application and the nature of the images being processed, the choice of edge detection method may vary.

Thresholding

1. apply_threshold

Parameters:

- img: The input image.
- threshold (int): The required threshold value.
- low_value (int): The value to replace the pixel values lower than the threshold defaults to zero.
- high_value (int): The value to replace the pixel values higher than the threshold defaults to 255.

Description:

Applies thresholding (binary) to a greyscale image setting values lower than the given threshold to low_value and higher to high_value.

Purpose:

Used as it is for global thresholding after getting the threshold from the user via a slider, the mean value of pixel values is also recommended for use as a threshold to the user on the window as well as providing the image histogram to help choose the optimum threshold. Local thresholding will be explained in the next function.

2. local_thresholding

Parameters:

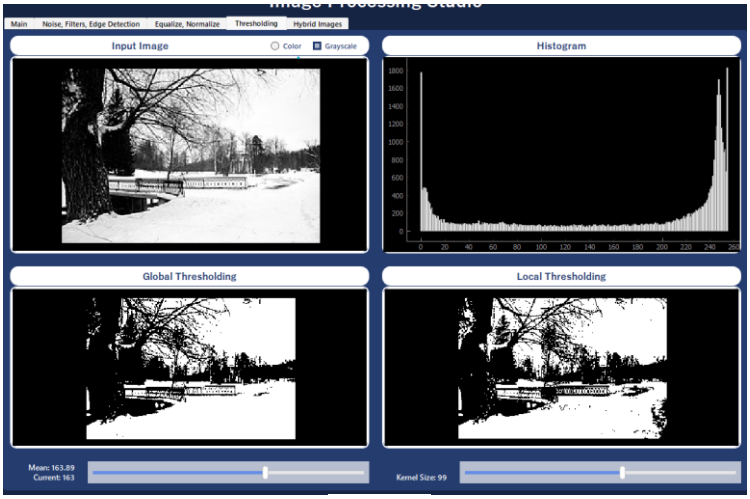
- img: The input image.
- region_size (int): Size of the local kernel used for thresholding. (square kernel)

Description:

Applies thresholding (binary) to a greyscale image with the threshold as the mean value of each local region. It accepts the region_size as an input from the user. Looping on each region of the image, finding the mean and using the apply_threshold function mentioned previously.

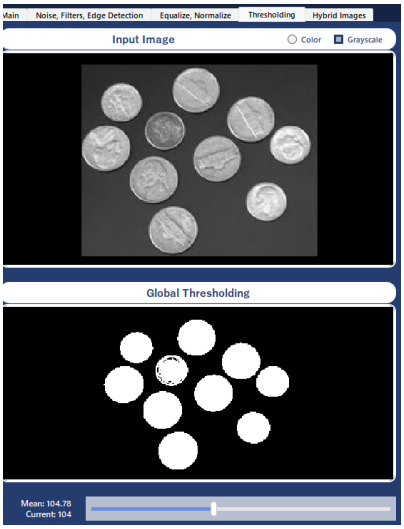
Experiments:

Img1

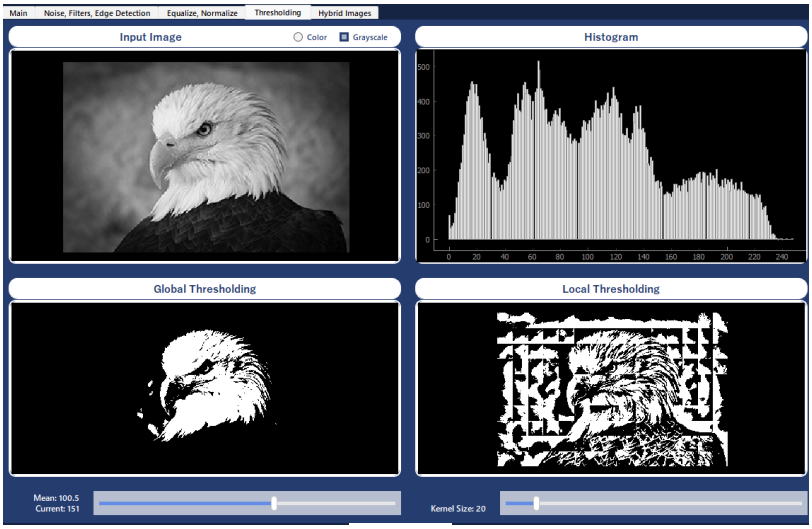


Img2

Note: Using the mean as a global threshold in the leftmost image.



Img3



Img4

- We can see that **global thresholding** does quite well when the whole background and foreground are quite distinct as in **img2** and **img3**.
- As for **img1**, we can see that the **mean** might not always be the best choice hence the need for more sophisticated thresholding algorithms.
- Also having one global threshold was not a suitable choice since the foreground and background have a lot of the same light grey shades.
- Too small of a kernel size in local thresholding is hardly a good choice as it loses sense of the whole picture as in **img4**.

Histogram Equalization

1. histogram_equalization

Parameters:

- image(ndarray): Input image (grayscale or colored) on which histogram equalization will be performed.
- is_grey(bool): Flag indicating whether the input image is grayscale (True) or colored (False).

Description:

It's a technique used to enhance the contrast of an image by redistributing the intensity levels across the entire range. It works particularly well for images with poor contrast or those that appear too dark or too bright.

Purpose:

- Histogram equalization enhances the visibility of details in an image by spreading out the intensity values.
- It can be especially useful for improving the appearance of medical images, satellite imagery, and low-contrast photographs.
- By redistributing the intensity levels, it makes the image more visually appealing and easier to analyze.

How it works?

- The process involves transforming the pixel intensities in such a way that the resulting histogram becomes more uniform.
- Here are the steps involved in our method of histogram equalization:
  1. Convert the input image to the HSV (Hue, Saturation, Value) color space.
  2. Compute the histogram of the Value (V) channel (which represents the brightness) in the HSV image.
  3. Calculate the cumulative distribution function (CDF) from the histogram.
  4. Normalize the CDF to ensure it spans the entire intensity range from 0 to 255.
  5. Create a mapping function that maps the original pixel intensities to their equalized values based on the normalized CDF.
  6. Apply this mapping to each pixel in the V channel of the image.
  7. Merge the equalized V channel back into the HSV image.
  8. Convert the modified HSV image back to the original color space.

Equalization in HSV color space:

Performing histogram equalization in the HSV (Hue, Saturation, Value) color space offers several advantages:

1. Separation of Luminance and Chrominance:
   In the HSV color model, the V channel (Value) represents the luminance or brightness of the image. The H (Hue) and S (Saturation) channels represent color information. By working with the V channel separately, we can focus specifically on adjusting the brightness without affecting color information.
2. Preserving Color Information:
   Histogram equalization directly applied to the RGB channels can alter the color balance and introduce color shifts. By equalizing only the V channel, we maintain the original color information in the H and S channels.
3. Better Handling of both Colored and Grayscale Images:
   When working with colored images (RGB) or grayscale images (where R = G = B), for code optimization we generalized the color space where we performed a histogram equalization process in the HSV color space. Equalizing the V channel in such cases enhances the overall contrast without affecting color.

Observations:

- ✓ The resulting image exhibits improved contrast and better visibility of details.
- ✓ Dark regions become brighter, and bright regions become darker, leading to a more balanced overall appearance
- ✓ However, keep in mind that histogram equalization is not always suitable for all types of images. In some cases, it may amplify noise or artifacts.

Experiments:

Original image

Original image

Equalized image

Equalized image

Image Normalization

1. normalize

Parameters:

- image(ndarray): Input image (grayscale or colored) to be normalized.
- is_grey(bool): Flag indicating whether the input image is grayscale (True) or colored (False).

Description:

Normalization is a process used to scale the pixel values of an image to a specific range. It ensures that the pixel intensities fall within a desired interval (often [0, 255]).

Purpose:

- Dynamic Range Adjustment:
  Normalization ensures that the pixel values span the entire intensity range (0 to 255). This is crucial for consistent processing and display.
- Consistent Scaling:
  When working with images, algorithms, or neural networks, having consistent pixel value ranges simplifies computations and improves convergence.
- Avoiding Saturation:
  Normalization prevents pixel values from exceeding the valid range, avoiding issues like clipping.

How it works?

- The normalization process is applied to the V channel (brightness) of an image in the HSV color space.
- Here are the steps involved:
  1. Convert the input image to the HSV color space.
  2. Extract the V channel from the HSV image.
  3. Calculate the minimum and maximum pixel values in the V channel.
  4. Normalize the pixel values of the V channel to the range [0, 255] using the following formula:

$$Normalized\_Pixel\_Value = New\_Min + \frac{(Pixel\_Value - Old\_Min)}{(Old\_Max - Old\_Min)} * (New\_Max - New\_Min)$$

Where:

$Pixel\_Value$ is the original value of the pixel.
Old_Min and Old_Max are the minimum and maximum pixel values in the original image.
New_Min and New_Max are the desired minimum and maximum values for the normalized image.

  5. Replace the original V channel with the normalized V channel.
  6. Convert the modified HSV image back to the original color space.

Observations:

- ✓ After normalization, the V channel will have pixel values ranging from 0 to 255.
- ✓ The overall brightness of the image remains unchanged, but the pixel values are now appropriately scaled.

Hybrid Images

1. resize_images

Parameters:

- image1(ndarray): First input image (grayscale or colored) to be resized.
- image2(ndarray): Second input image (grayscale or colored) to be resized.

Description:

It's a function to adjust the dimensions of two input images (image1 and image2) so that they have the same size. It checks the dimensions of both images and resizes the larger image to match the dimensions of the smaller one.

Purpose:

- Ensure that two images have the same dimensions, which can be useful for various image processing tasks.
- Having images of equal size allows for easier comparison, combination, or manipulation of the images in subsequent operations.
- It's often necessary for both images to have the same dimensions to achieve the desired visual effect. Therefore, it helps standardize the dimensions of input images for further processing.

2. generate_hybrid_image

Parameters:

- image1(ndarray): First input image (grayscale or colored) used to generate the hybrid image.
- image2(ndarray): Second input image (grayscale or colored) used to generate the hybrid image.
- segma1(int): (optional) - Standard deviation for the Gaussian filter applied to image1 (default value: 20).
- segma2(int): (optional) - Standard deviation for the Gaussian filter applied to image2 (default value: 1).
- kernel_size1(int): (optional) - Size of the Gaussian kernel for filtering image1 (default value: 7).
- kernel_size2(int): (optional) - Size of the Gaussian kernel for filtering image2 (default value: 3).
- is_grey1(bool): (optional) - Flag indicating whether image1 is grayscale (True) or colored (False) (default value: True).
- is_grey2(bool): (optional) - Flag indicating whether image2 is grayscale (True) or colored (False) (default value: True).

Description:

Hybrid images are a combination of two input images into a single image. When viewed from different distances, hybrid images reveal distinct features:

- Up close, we perceive the high-frequency details (fine textures, edges).
- From a distance, we see the low-frequency content (overall structure and global features).

How it works?

- The process involves blending two input images in the following steps:
  1. Retrieve Input Images:
     Obtain two input images that will be used to create the hybrid image.
  2. Adjust Parameters:
     Determine the parameters required for generating the hybrid image. These parameters include:
     a. Sigma values: Standard deviations for Gaussian blurring.

b. Kernel sizes: Sizes of the Gaussian kernels used for blurring.
c. Color mode: Decide whether to process the images in grayscale or color.

3. Process Input Images:
   a. Apply Gaussian blurring to one input image using the specified parameters. This step involves convolving each image with a Gaussian kernel to blur it and form a low frequency content image.
   b. Get high frequency content of the second image by blurring the image, then subtracting the output from the original image.
4. Combine Images:
   Combine both images by adding the blurred image with low frequency content, with the image with high frequency content.

Observations:

- ✓ The parameters of the Gaussian filter significantly impact the output:
  a. Kernel size: Determines the size of the Gaussian kernel. Larger values result in stronger blurring.
  b. Sigma: Controls the spread of the Gaussian distribution. Larger sigma values lead to more extensive blurring.
- ✓ When the output hybrid image is viewed up close, we primarily see the high-frequency details, while at a distance, we notice the low-frequency components more.

Experiments: