Task 1 Report

Computer Vision

Images, Filters, Histograms, Gradients, Frequency

Team Members

Ahmed Abd Elraouf
Zeyad Amr
Mo’men Mohamed
Mazen Tarek
Michael Hany

Table of content

Introduction

It’s a web project that aims to enhance images using various image processing techniques. It covers noise removal, edge detection, and histogram analysis, as well as color to grayscale transformation and frequency domain filters. Local and global thresholding is also included, as well as hybrid image creation. The project is designed to provide a comprehensive understanding of image processing techniques and their practical applications.

The application consists of three tabs that allow the user to manipulate loaded images. In Tab 1, the user can apply various filters, select filter parameters, and see the output image. In Tab 2, the user can plot and analyze the histogram of an image, apply histogram equalization, and see the resulting output image and histogram. Tab 3 enables the user to create hybrid images by combining the high and low-frequency components of two loaded images.

Overall, the web application is user-friendly and offers a range of image processing options, from basic filtering to advanced techniques such as hybrid image creation.

Technologies

- Frontend: React ts
- Backend: Django
- Processing: Python, opencv, matplotlib, numpy

Features

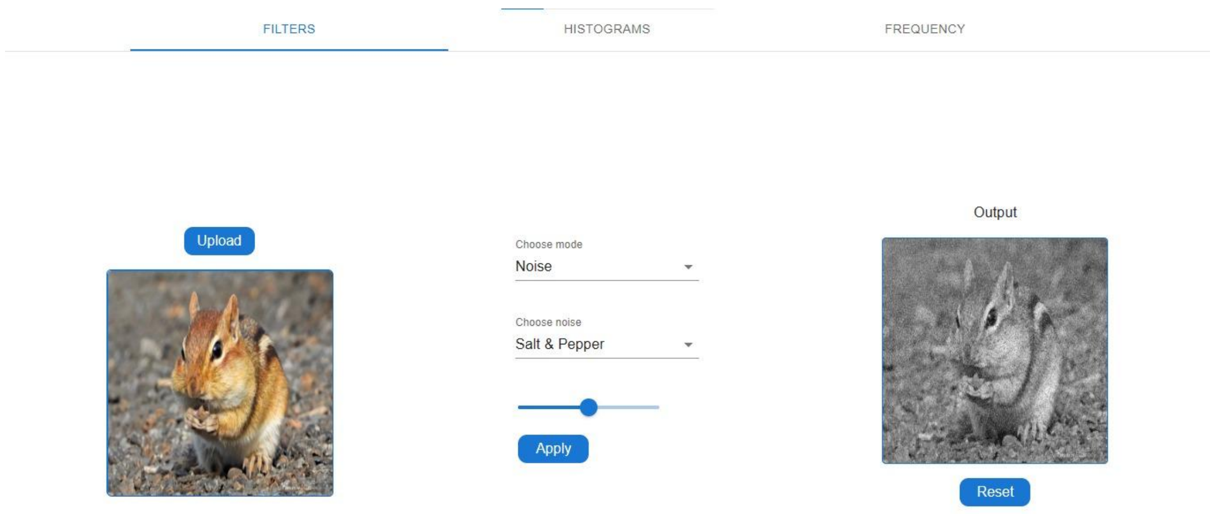
1. Filters:

The given class Filters contains various image processing algorithms for adding noise, applying smoothing filters and detecting edges to an image.

Add Noise Algorithms:

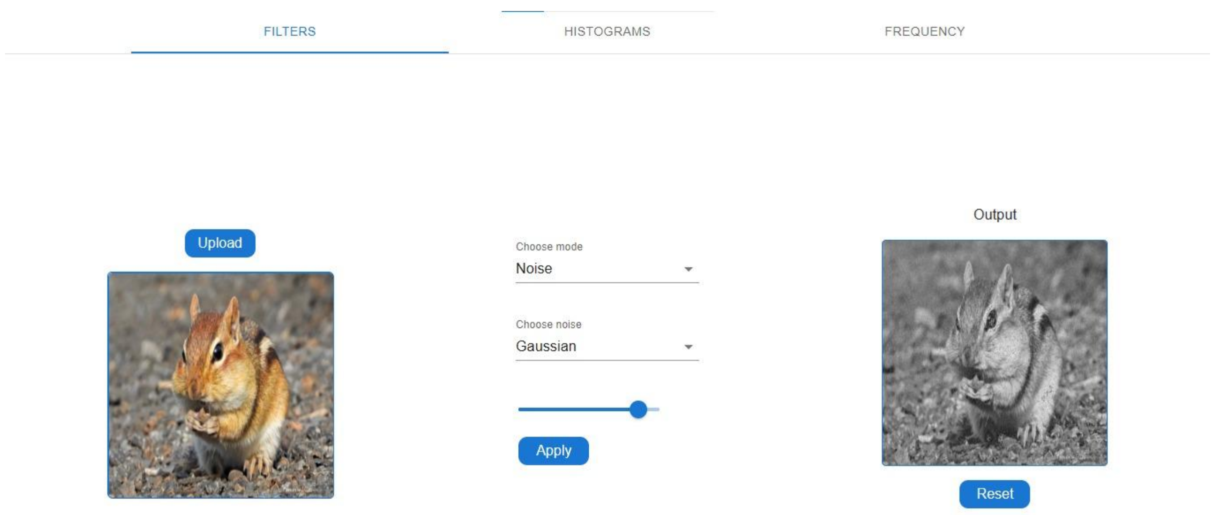
a. salt_pepper_noise(image, range):

- This function adds salt and pepper noise to the image. The function takes two arguments: the input image and the range of noise to be added. The noise range is a value between 0 and 255, where 0 means no noise and 255 means full noise.
- The algorithm generates a random matrix of the same size as the input image and multiplies it by 255. It then checks for values less than the given range and assigns 0 to those pixels (pepper noise) and values greater than 255 minus the range and assigns 255 to those pixels (salt noise).
- Finally, The function will add salt and pepper noise to the input image.


Choose mode: Noise
Choose noise: Salt & Pepper

b. gaussian_noise(image, range):

- This function adds Gaussian noise to the image. The function takes two arguments: the input image and the range of noise to be added. The noise range is a value that controls the variance of the Gaussian distribution.
- The algorithm generates a random Gaussian distribution of the same size as the input image with mean 0 and standard deviation equal to the given range. It then adds this noise to the input image.
- Finally, The function will add Gaussian noise to the input image.

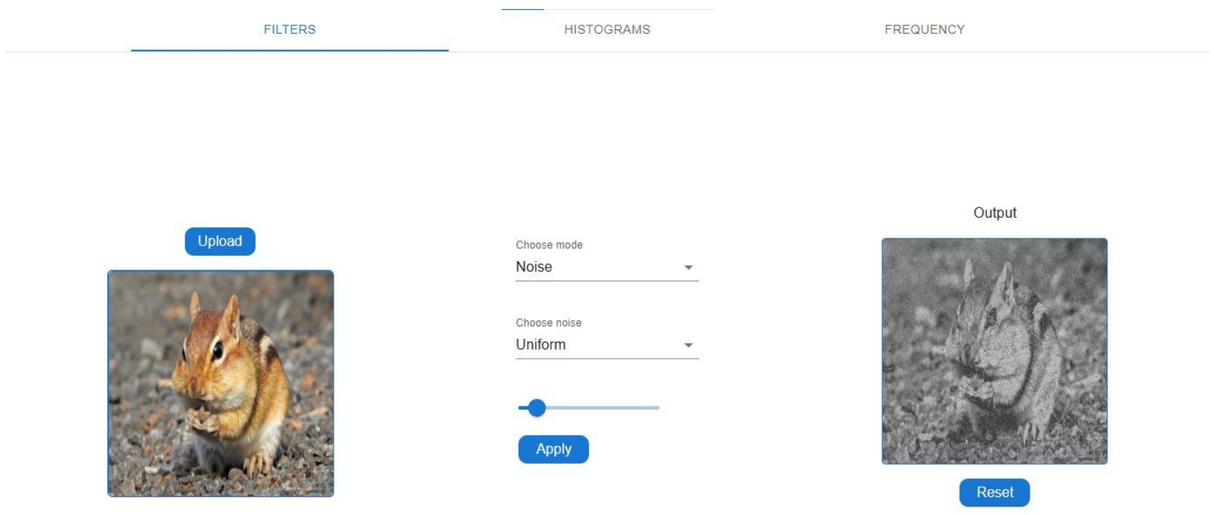


Choose mode
Noise

Choose noise
Gaussian

c. uniform_noise(image, range):

- This function adds uniform noise to the image. The function takes two arguments: the input image and the range of noise to be added. The noise range is a value that controls the range of uniform distribution.
- The algorithm generates a random uniform distribution of the same size as the input image with low equal to minus the given range and high equal to the given range. It then adds this noise to the input image.
- Finally, The function will add uniform noise to the input image.



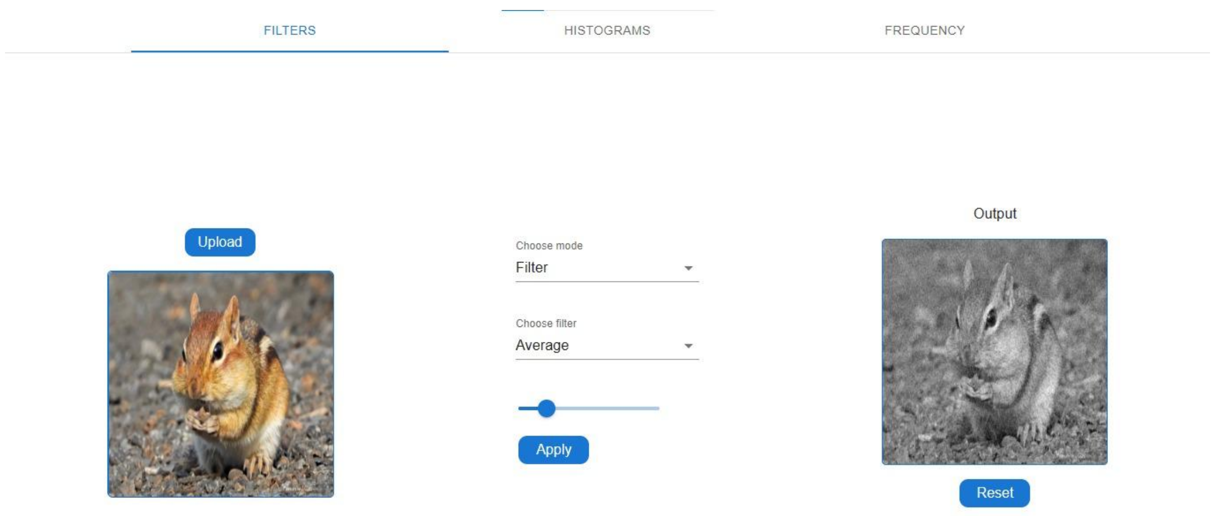
Choose mode: Noise

Choose noise: Uniform

Smoothing Filters Algorithms:

a. average_filter(image, kernel_size):

- This function applies an average filter to the image. The function takes two arguments: the input image and the kernel size of the filter.
- The algorithm convolves the kernel of the given size with the input image to obtain the filtered image. The value of each pixel in the filtered image is the average of the values of the corresponding pixels in the kernel.
- Finally, The function will apply an average filter to the input image.

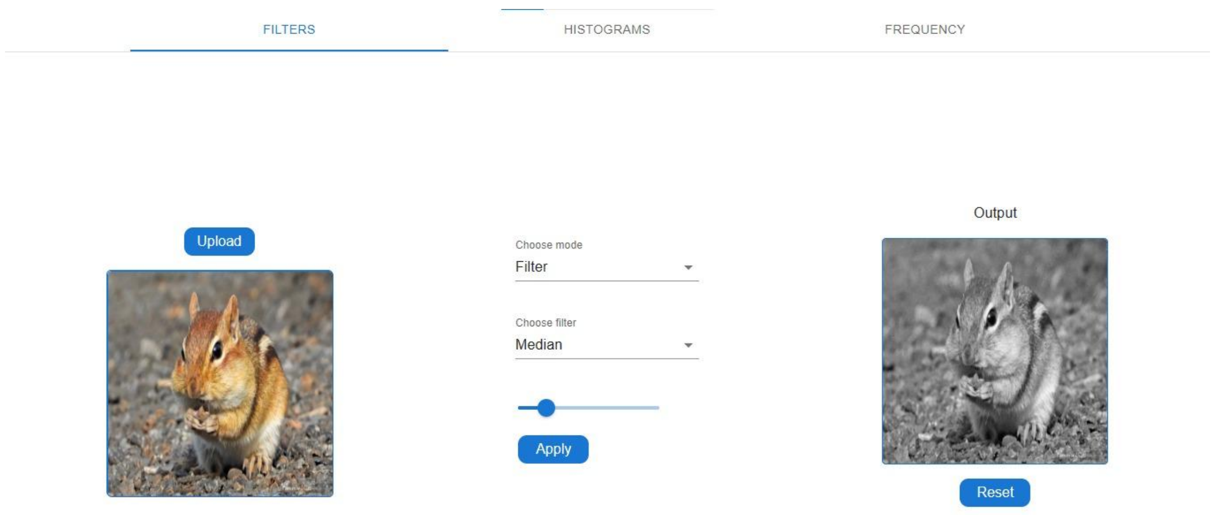


Choose mode: Filter

Choose filter: Average

b. median_filter(image, kernel_size):

- This function applies a median filter to the image. The function takes two arguments: the input image and the kernel size of the filter.
- The algorithm convolves the kernel of the given size with the input image to obtain the filtered image. The value of each pixel in the filtered image is the median of the values of the corresponding pixels in the kernel.
- Finally, The function will apply a median filter to the input image.

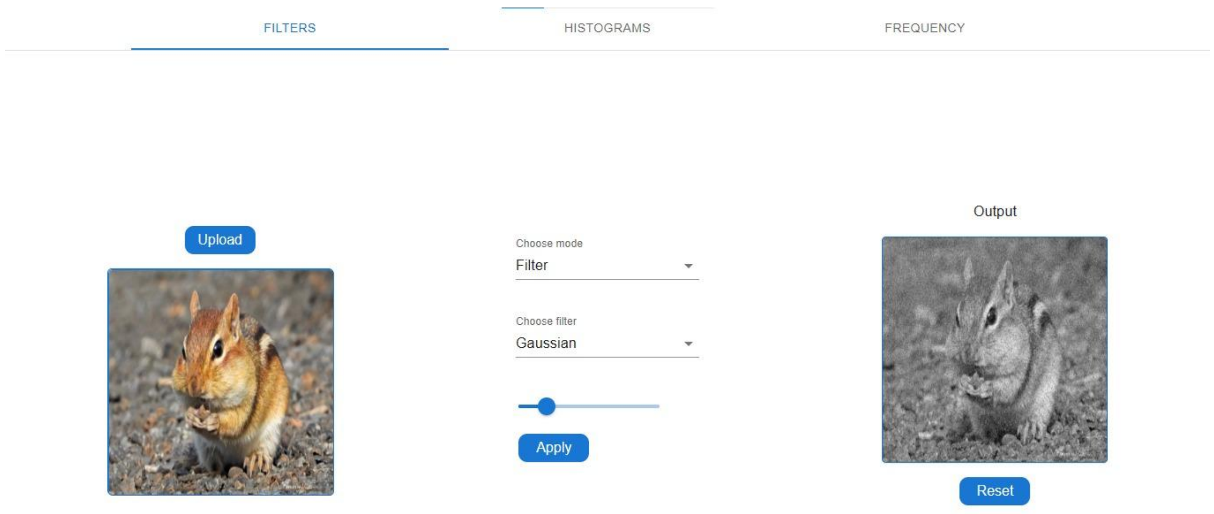


Choose mode: Filter

Choose filter: Median

c. gaussian_filter(image, kernel_size):

- This function applies a Gaussian filter to the image. The function takes two arguments: the input image and the kernel size of the filter.
- The algorithm generates a Gaussian kernel of the given size and standard deviation of 2. It then convolves this kernel with the input image to obtain the filtered image.
- Finally, The function will apply a Gaussian filter to the input image.



Choose mode: Filter

Choose filter: Gaussian

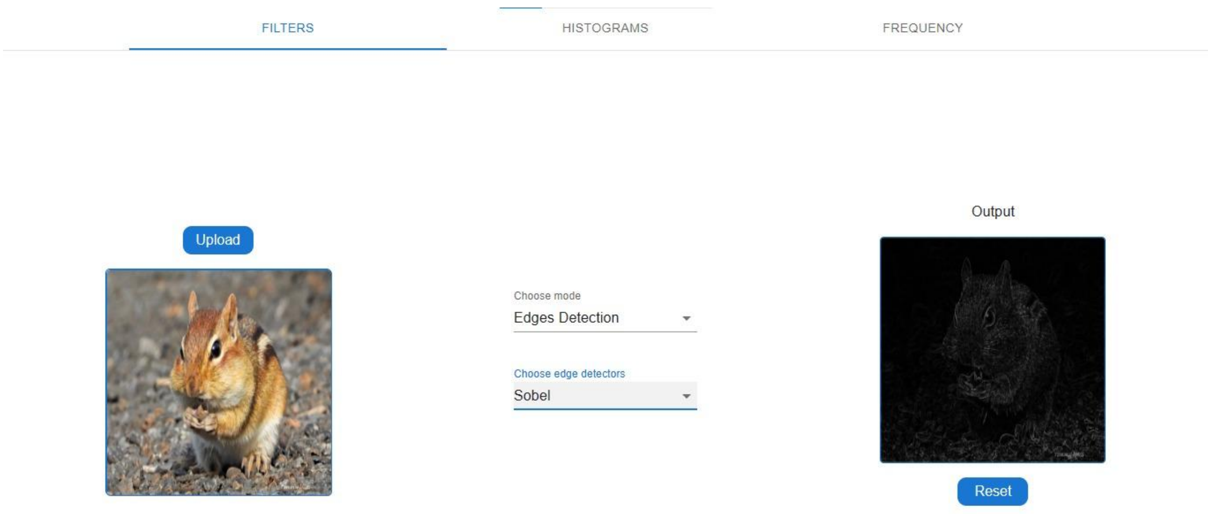
d. __gaussian_kernel(kernel_size, sigma):

- This function generates a Gaussian kernel of the given size and standard deviation.
- The algorithm first generates a 1D array with values ranging from -(kernel_size-1)/2 to (kernel_size-1)/2. It then creates a 2D meshgrid from the 1D array and calculates the values of the Gaussian function with the given standard deviation. Finally, it normalizes the image.

Edge Detection Algorithms:

a. sobel_edge_detector(image):

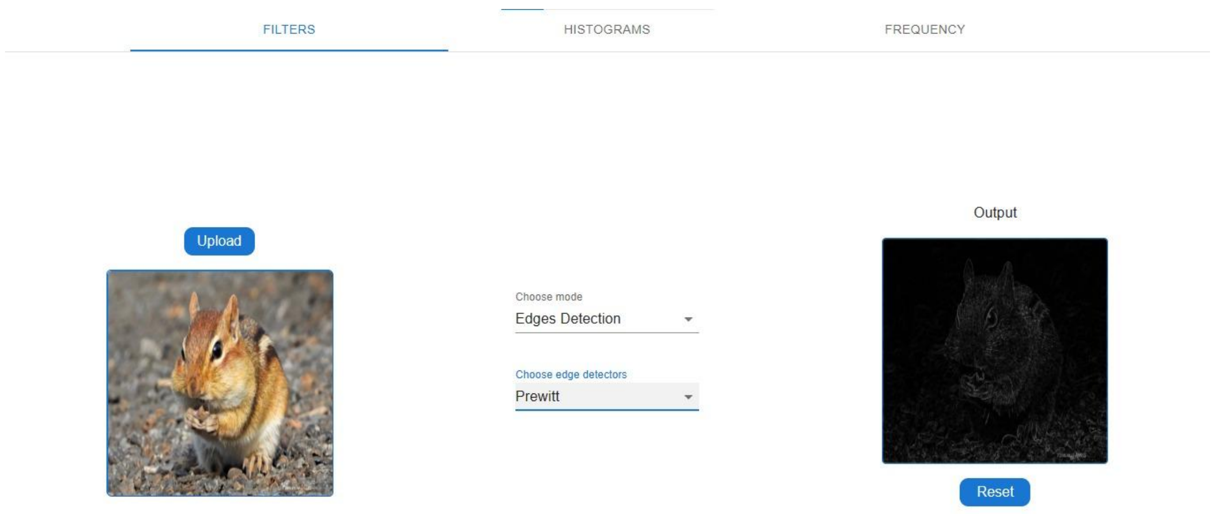
- This function performs edge detection using the Sobel operator.
- The method initializes two 3x3 filters, vertical_grad_filter and horizontal_grad_filter, which are the Sobel operators for detecting vertical and horizontal edges in an image, respectively.
- The filters are designed to give more weight to the central pixel and less weight to the surrounding pixels.
- Then, the method calls the __detect_edges_helper method with the input image and these two filters as arguments. The __detect_edges_helper method applies the filters on the image to detect edges and returns a gradient image that represents the strength of the edges and the angles of the gradients at each pixel.
- Finally, the sobel_edge_detector method returns the gradient image obtained from __detect_edges_helper.


Choose mode
Edges Detection

Choose edge detectors
Sobel

b. prewitt_edge_detector(image):

- This function performs edge detection using the Prewitt operator.
- The method initializes two 3x3 filters, vertical_grad_filter and horizontal_grad_filter, which are the Prewitt operators for detecting vertical and horizontal edges in an image, respectively.
- The filters are designed to give equal weight to all pixels in the image.
- Then, the method calls the __detect_edges_helper method with the input image and these two filters as arguments. The __detect_edges_helper method applies the filters on the image to detect edges and returns a gradient image that represents the strength of the edges and the angles of the gradients at each pixel.
- Finally, the sobel_edge_detector method returns the gradient image obtained from __detect_edges_helper.

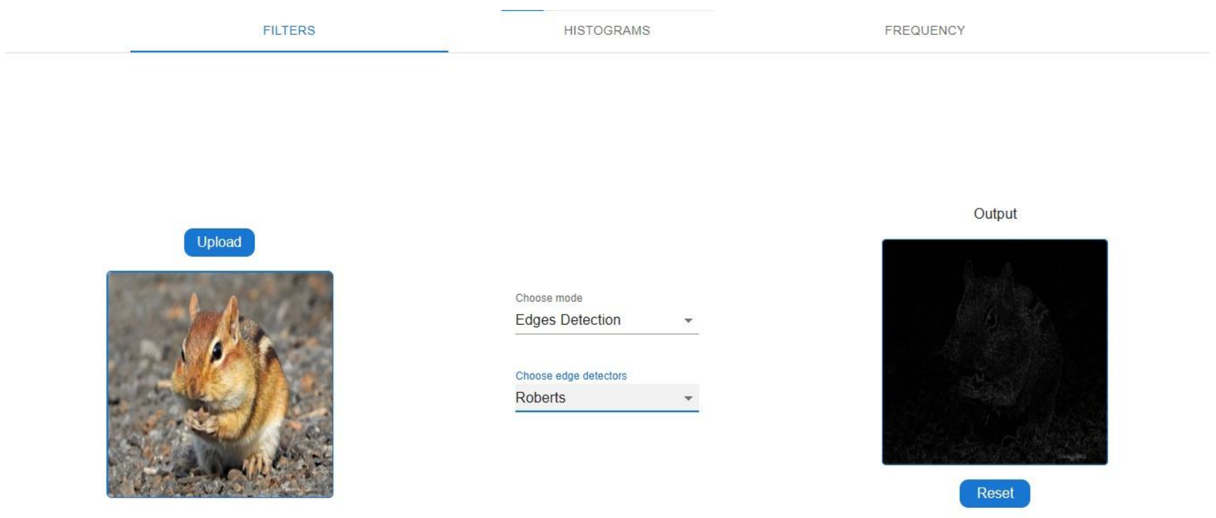


Choose mode
Edges Detection

Choose edge detectors
Prewitt

c. roberts_edge_detector(image):

- This function performs edge detection using the Prewitt operator.
- The method initializes two 3x3 filters, vertical_grad_filter and horizontal_grad_filter, which are the Prewitt operators for detecting vertical and horizontal edges in an image, respectively.
- The filters are designed to give more weight to the diagonal pixels and less weight to the surrounding pixels.
- Then, the method calls the __detect_edges_helper method with the input image and these two filters as arguments. The __detect_edges_helper method applies the filters on the image to detect edges and returns a gradient image that represents the strength of the edges and the angles of the gradients at each pixel.
- Finally, the sobel_edge_detector method returns the gradient image obtained from __detect_edges_helper.

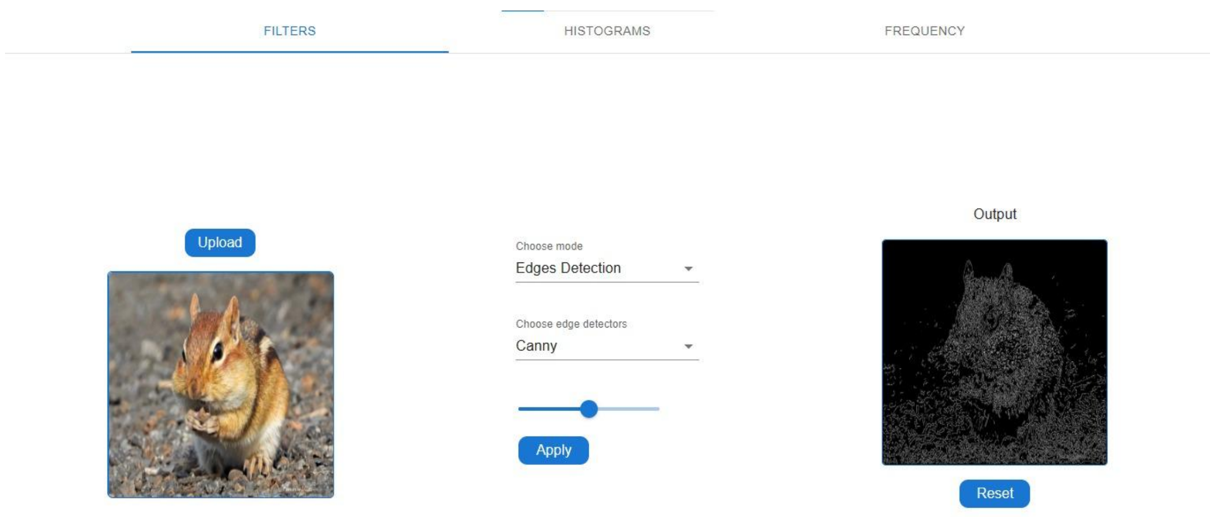


Choose mode
Edges Detection

Choose edge detectors
Roberts

c. canny_edge_detector(image):

- This function performs edge detection using the Canny edge detection algorithm on an input image. The steps of the algorithm are as follows:
- Gaussian filter: The input image is first smoothed using a Gaussian filter to reduce noise and remove small details.
- Sobel edge detector: The Sobel operator is applied to the smoothed image to obtain the gradient magnitude and direction at each pixel.
- Non-maximum suppression: The gradient magnitude is thinned down to a one-pixel wide edge by suppressing all gradients except the local maxima in the gradient direction.
- Double threshold and hysteresis: A double threshold is applied to the gradient magnitude image to classify each pixel as either an edge or non-edge pixel. The pixels with gradient magnitude above the high threshold are considered edge pixels, while those below the low threshold are considered non-edge pixels. The pixels between the two thresholds are classified as edge pixels only if they are connected to other edge pixels.
- The range parameter is used to set the high threshold of the double thresholding step. The low threshold is set to zero.
- Finally, the output image is returned. It will have white pixels indicating edges and black pixels indicating non-edges.



Choose mode
Edges Detection

Choose edge detectors
Canny

e. __detect_edges_helper (image, kernel_size):

- This function takes in an image and two optional filter arrays - vertical_grad_filter and horizontal_grad_filter - that are used to calculate the gradient in the vertical and horizontal directions.
- The image is normalized by dividing every pixel value by 255, which scales the pixel values to be between 0 and 1.
- The function initializes the width of the kernel as half the height of the vertical_grad_filter.
- It initializes two empty arrays - gradient and self.angles - to hold the gradient magnitude and direction information, respectively.
- The image is padded with zeros around its edges to ensure that the kernel can be applied to all pixels in the image.
- The function loops through every pixel in the image, skipping the padded edges.
- For each pixel, the function extracts a sub-image of size (kernel_width*2 + 1, kernel_width*2 + 1) centered at the pixel.
- The filter is applied to the sub-image by element-wise multiplication of the filter array with the sub-image, followed by a summation of the resulting array. This process is repeated for both the vertical and horizontal filters to obtain the vertical and horizontal gradient components.
- The gradient magnitude at the pixel is calculated by taking the square root of the sum of the squared vertical and horizontal gradient components.
- The gradient direction at the pixel is calculated using the arctan2 function in numpy, which returns the angle between the positive x-axis and the line passing through the origin and the pixel.
- The gradient magnitude and direction values are stored in the gradient and self.angles arrays, respectively.
- The gradient array is returned as the output of the function.

f. __non_maximum_suppression(image):

- This function performs non-maximum suppression on the gradient magnitude image to thin the edges and keep only the strongest edges. The input to the function is obtained from the output of the __detect_edges_helper function.
- The function first converts the angles from radians to degrees and removes negative angles. Then it creates a zero matrix of the same size as the input image to store the output. It then loops through the image and for each pixel, it checks the angle of the gradient and compares the pixel's value to its neighbors in the direction of the gradient.
- If the pixel's value is greater than or equal to the value of its neighbors in the direction of the gradient, the pixel's value is kept in the output. Otherwise, the pixel's value is set to zero. This process eliminates all pixels that are not local maxima in the direction of the gradient.
- Finally, the output image is scaled to have values between 0 and 255 and returned.

g. __double_threshold_hysteresis(image, low, high):

- This function performs hysteresis thresholding on the input image.
- After applying the Canny edge detection algorithm, the edges are classified as strong edges or weak edges. The thresholds for strong and weak edges are defined by the high and low values respectively.
- The code creates a zero matrix called result with the same dimensions as the input image. Then, it sets the pixels that are above the high threshold to strong and the pixels that are above the low threshold but below the high threshold to weak.
- To perform hysteresis, the code starts with a strong edge pixel and checks its 8 neighboring pixels in all directions using the dx and dy arrays. If a neighboring pixel is a weak edge pixel, it is marked as strong and added to the strong edge list. The process continues until all weak edge pixels that are connected to strong edge pixels have been marked as strong.
- Finally, all the pixels in the result matrix that are not marked as strong are set to 0. The result matrix is then returned.

2. Histogram:

The given class Histogram contains methods for performing image processing tasks such as histogram equalization, normalization, global and local thresholding, and color channel splitting.

a. getImg():

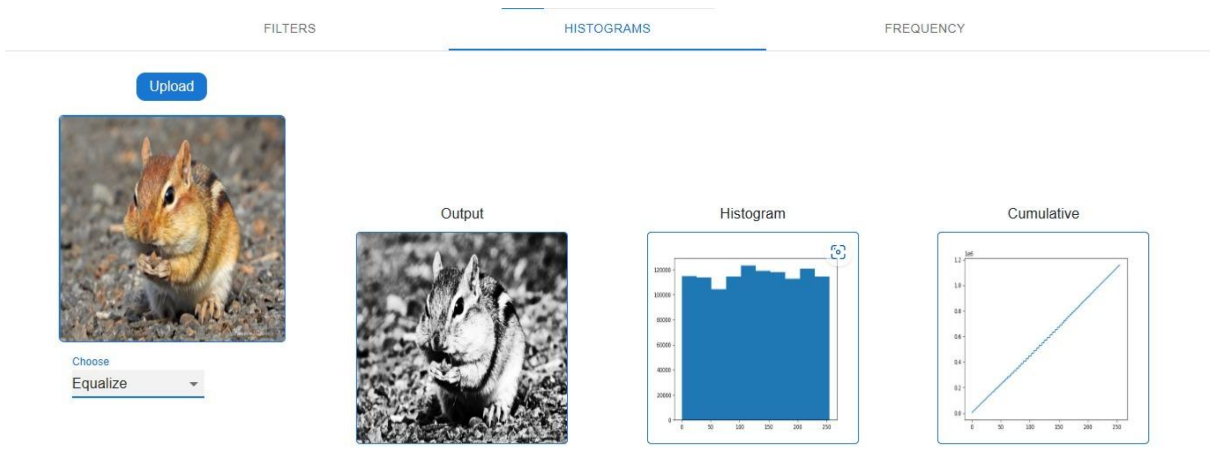
- This method returns the image array associated with the instance.

b. operateOn(img):

- This method sets the image array associated with the instance to the passed image array img.

c. equalize():

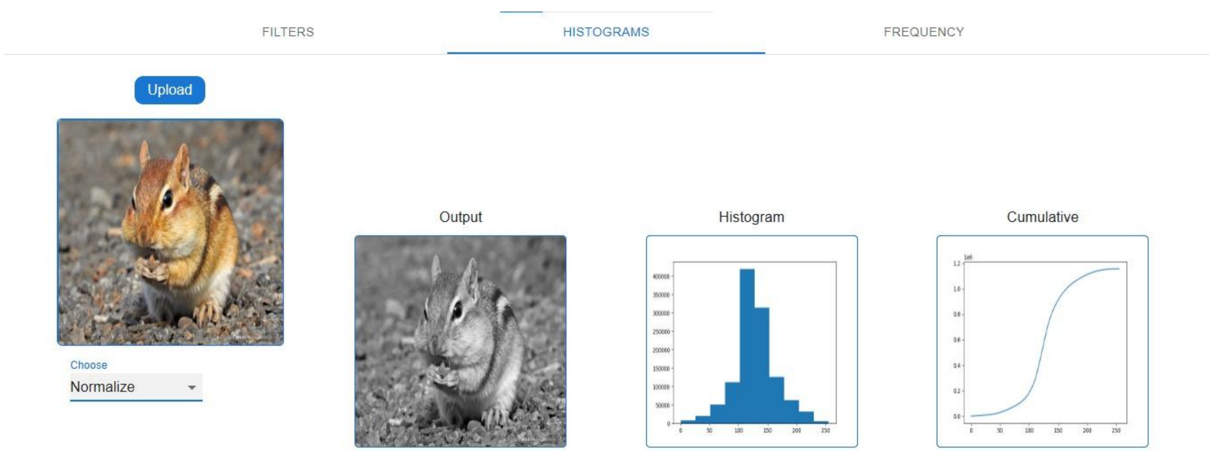
- This method performs histogram equalization on the image array. It computes the histogram and the cumulative sum of the pixel intensities and then scales the cumulative sum to the range of [0, 255].
- Then, it maps each pixel intensity of the input image to its corresponding intensity in the equalized image using the scaled cumulative sum.
- The output of the function is the equalized image.



Choose: Equalize

d. normalize():

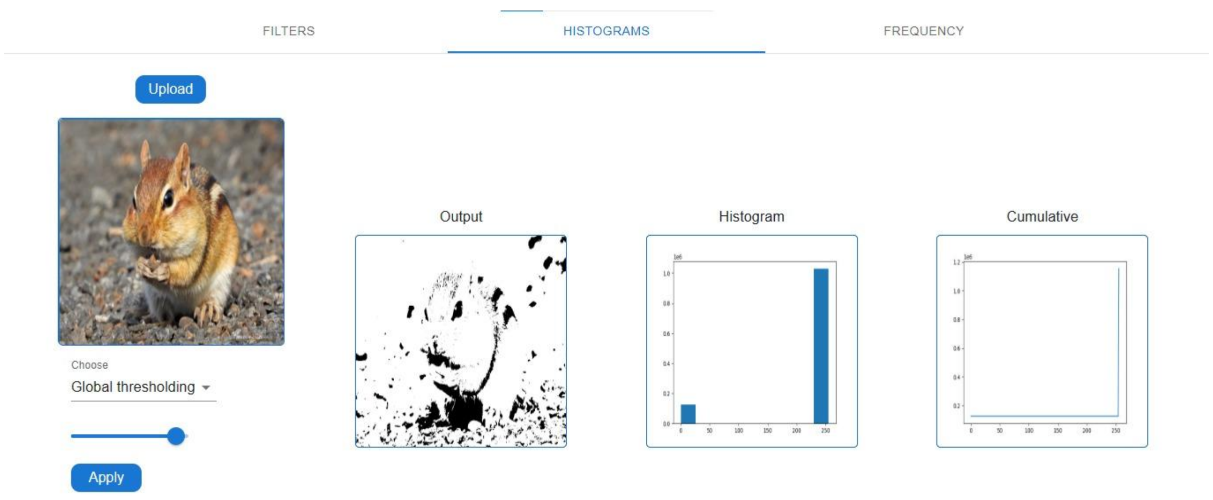
- This method normalizes the image array by dividing each pixel intensity by maximum intensity and then scaling it to a range of [0, 255].
- The output of the function is the normalized image.



Choose: Normalize

e. applyGlobalThreshold(threshold):

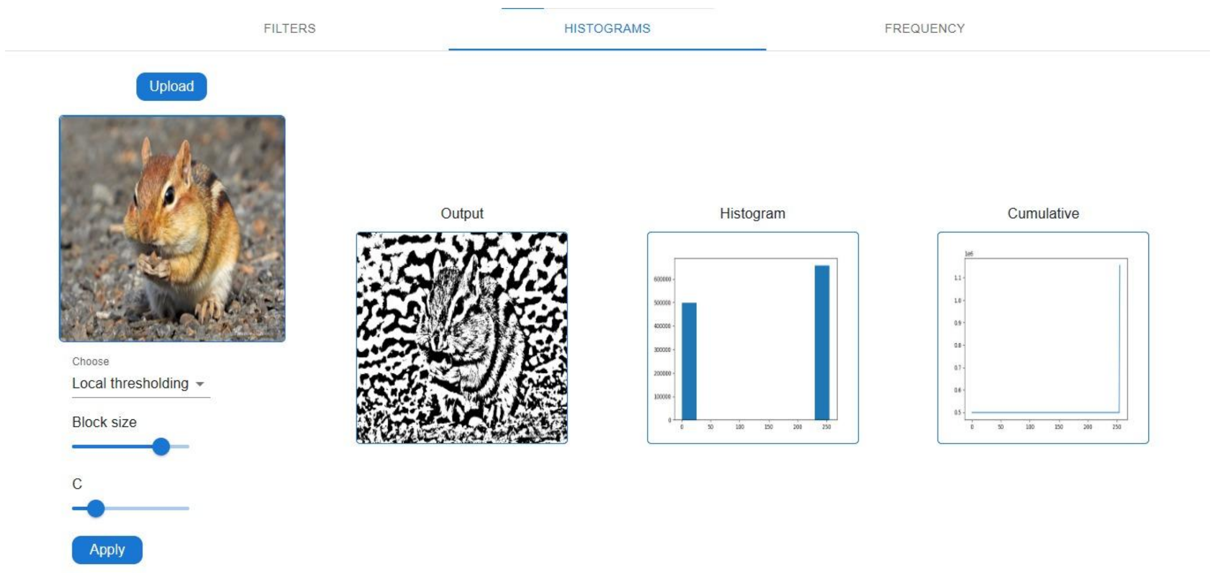
- This method applies a global thresholding operation on the image array.
- It sets all pixel intensities greater than the threshold to 255 and all other pixel intensities to 0.
- The output of the function is the thresholded image.



Choose: Global thresholding

f. applyLocalThreshold(blockSize=10, C=5):

- This method applies a local thresholding operation on the image array.
- It uses a sliding window of size blockSize to compute a local threshold for each pixel based on the mean intensity of the pixels in the window and a constant C.
- If the intensity of the pixel is greater than or equal to the local threshold, it is set to 255, otherwise it is set to 0.
- The output of the function is the thresholded image.



Choose: Local thresholding

Block size

C

g. split():

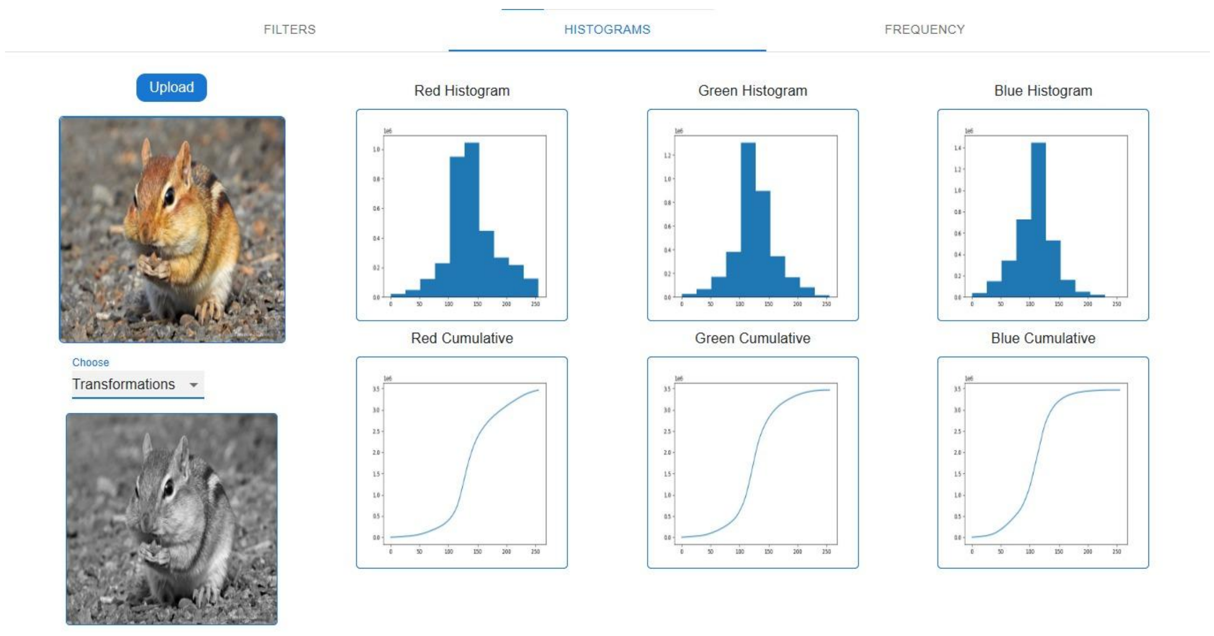
- This method splits the image array into its three color channels, i.e., red, green, and blue.
- It creates three new arrays of the same size as the input image and assigns the red, green, and blue values of each pixel to their corresponding arrays.
- The output of the function is a tuple containing the red, green, and blue arrays.

h. getCumSum(arr):

- This method takes a one-dimensional array arr and returns its cumulative sum.

i. grayScale() :

- This method is a function that returns a grayscale representation of an image. It achieves this by averaging the intensity values of the red, green, and blue color channels of the image.
- It first extracts the individual red, green, and blue color channels using the getRedFrame(), getGreenFrame(), and getBlueFrame() methods. It then divides each channel by 3 and adds them together to obtain a single grayscale value for each pixel.



j. getHistoGram(arr2d, bins=256):

- This method takes a two-dimensional array arr2d and computes its histogram using the specified number of bins.
- It returns a one-dimensional array representing the histogram.

k. flatten(arr2d):

- This method takes a two-dimensional array arr2d and returns a flattened one-dimensional array.

l. getCumulative2d(img):

- This is a helper method that takes a two-dimensional image array and returns its two-dimensional cumulative sum array.

m. getSumAndNum(cumulative, bottomRightX, bottomRightY, topLeftX, topLeftY):

- This is a helper method that takes a two-dimensional cumulative sum array cumulative and the coordinates of the top-left and bottom-right corners of a rectangular region and returns the sum of the pixel intensities in the region and the number of pixels in the region.

3. Frequency:

The given class Frequency contains methods for applying frequency-based image processing techniques, specifically Fourier transforms, high-pass/low-pass filtering and hybrid images.

a. high_pass_filter(img, filter_range):

- This method applies a high-pass filter on the input image img.
- It starts by resizing the image to a fixed size (512x512). It then applies a Fourier transform on the image using the np.fft.fft2 method.
- The Fourier transform shifts the low frequency components of the image towards the corners and high frequency components towards the center of the image.
- The next step is to define a filter mask which will be applied on the Fourier transformed image to remove the low frequency components from the image.
- The filter mask is a binary mask of size the same as input image with values 1 for pixels where the distance from the center of the image is greater than the filter_range and 0 otherwise. The mask is applied to the Fourier transformed image by element-wise multiplication with the np.fft.fftshift method.
- After applying the inverse Fourier transform to the filtered image using the np.fft.ifft2 method, it returns the absolute value of the filtered image.

b. low_pass_filter(img, filter_range):

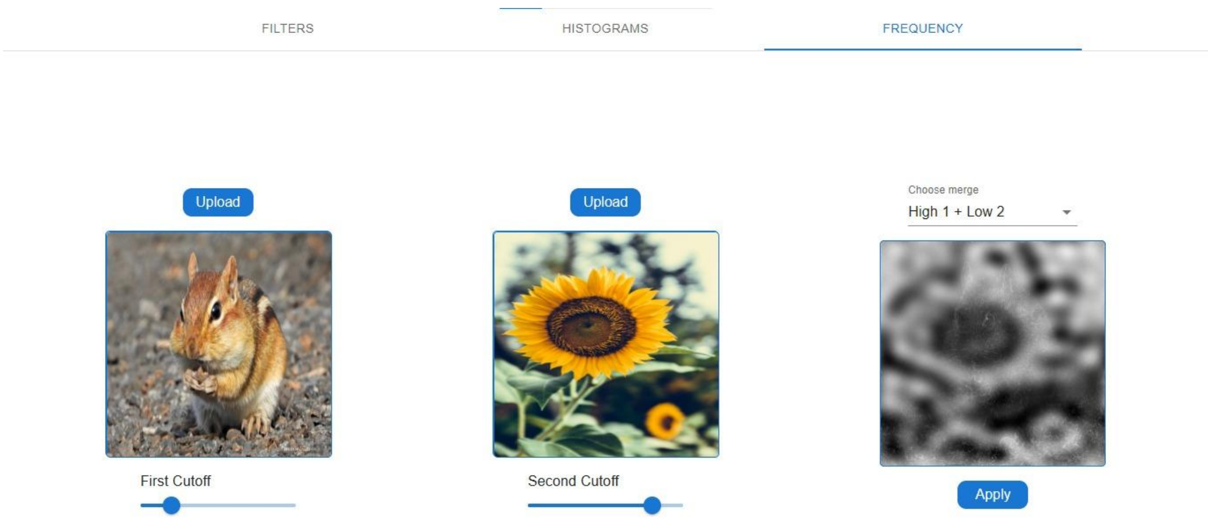
- This method applies a low-pass filter on the input image img. It starts by resizing the image to a fixed size (512x512).
- It then applies a Fourier transform on the image using the np.fft.fft2 method.
- The Fourier transform shifts the low frequency components of the image towards the corners and high frequency components towards the center of the image.
- The next step is to define a filter mask which will be applied on the Fourier transformed image to remove the high frequency components from the image.

- The filter mask is a binary mask of size the same as input image with values 1 for pixels where the distance from the center of the image is less than the filter_range and 0 otherwise.
- The mask is applied to the Fourier transformed image by element-wise multiplication with the np.fft.fftshift method.
- After applying the inverse Fourier transform to the filtered image using the np.fft.ifft2 method, it returns the absolute value of the filtered image.

c. hypridImages(img1, img2):

- This method takes two input images img1 and img2, adds them together and returns the result as the hybrid image.
- The hybrid image is normalized by dividing by 255 before returning.
- This method can be used to create hybrid images by combining the low frequency components of one image with the high frequency components of another image.

- Hybriding two images, high frequencies of the first first and low frequencies of the second.

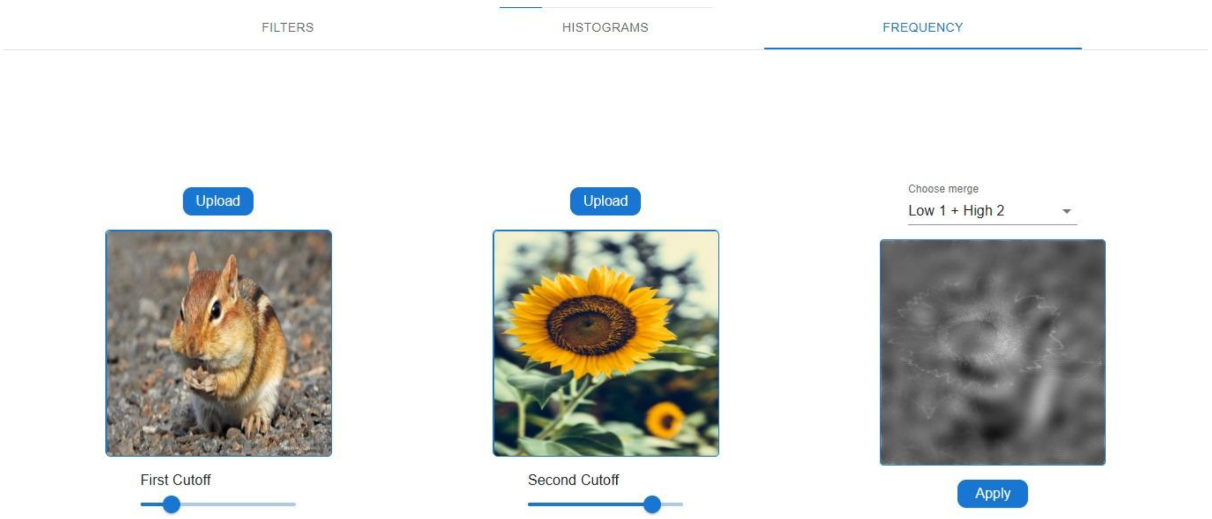


First Cutoff

Second Cutoff

Choose merge
High 1 + Low 2

- Hybriding two images, low frequencies of the first first and high frequencies of the second.



First Cutoff

Second Cutoff

Choose merge
Low 1 + High 2