Faculty of Engineering
Systems & Biomedical Engineering Department

Computer Vision-Task1

A Third Year Systems & Biomedical Engineering Task Report

Done by Team 5

Name	Section	BN
Ibrahim Mohamed	1	2
Omnia Sayed	1	14
Marina Nasser	2	12
Mahmoud Yaser	2	30
Maye Khaled	2	40

Submitted in Partial Fulfilment of the Requirements for
Computer Vision (SBE3230)

Submitted to:

Eng. Peter Emad

Eng. Laila Abbas

March 2023

1.Noise Additives

- **Algorithms**

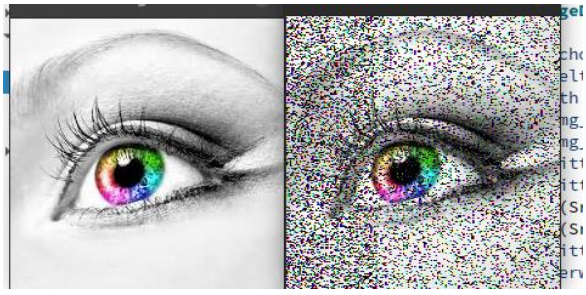
1. **Uniform**
   For each pixel in the image, we generate a random number that follows a rectangle distribution. Then we simply add the random number to the pixel's original value.

2. **Gaussian**
   Gaussian Noise is a statistical noise having a probability density function equal to normal distribution, also known as Gaussian Distribution. Random Gaussian function is added to Image function to generate this noise. It is also called as electronic noise because it arises in amplifiers or detectors.
   - **Output Sample**

3. **Salt & Pepper**

   This algorithm is implemented when the given pixel is noted to be changed. Instead of the original value of the pixel, it is replaced by the random number between 1 and 256.By randomizing the noise values, the pixels can change to a white, black, or gray value, thus adding the salt and pepper colors.

   - **Output Sample**



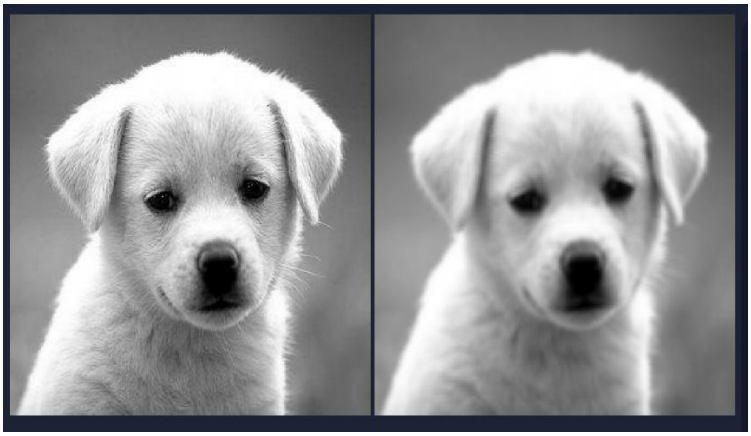
2. Image Smoothers

- **Algorithms**

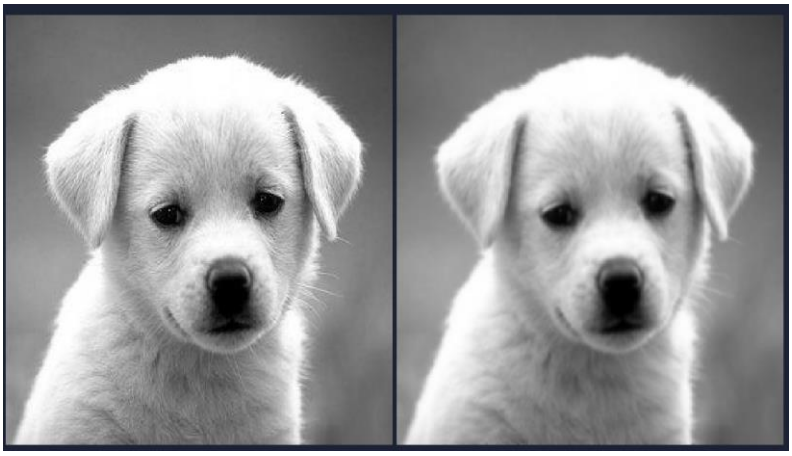
1. **applyAverageFilter:** Takes an input image and the size of the kernel as parameters. It applies the average filter on the input image using the given kernel size. The function first makes a copy of the input image, then iterates over each pixel of the image. For each pixel, it calculates the sum of the pixels around it with the size of the kernel and divides the sum by the number of pixels to get the average. Finally, the value of the pixel is set to the average value.
2. **applyGaussianFilter:** Takes an input image, the size of the kernel, and the standard deviation as parameters. It applies the Gaussian filter on the input image using the given kernel size and standard deviation. The function splits the input image into its color channels, applies the Gaussian filter on each color channel, and merges the filtered channels back to form the smoothed image. For each color channel, the function iterates over each pixel of the image. For each pixel, it calculates the sum of the pixels around it with the size of the kernel, weighted by the Gaussian function, and sets the value of the pixel to the weighted sum.
3. **applyMedianFilter:** Takes an input image and the size of the kernel as parameters. It applies the median filter on the input image using the given kernel size. The function first creates an output image with the same size and type as the input image. It then iterates over each pixel of the input image. For each pixel, it creates a window around the pixel with the size of the kernel, calculates the median value of the pixels in the window, and sets the value of the output pixel to the median value.

- **Output Samples**

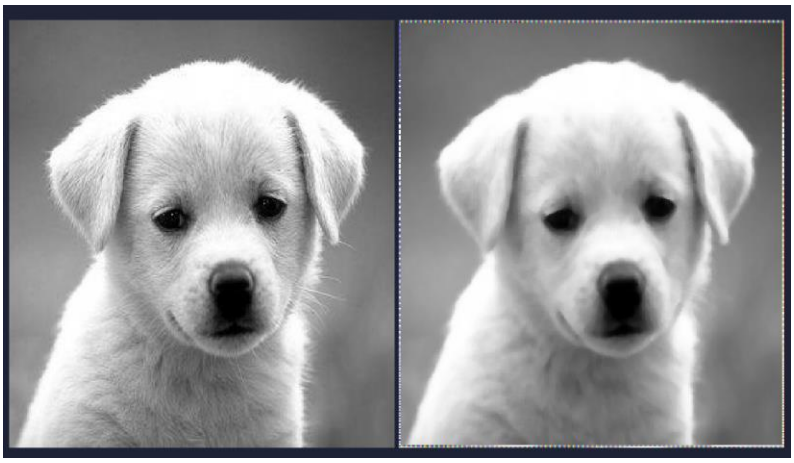
1. Average Filter



2. Gaussian Filter



3. Median Filter



3. Edge Detectors

1.Canny Edge Detector:

- **Apply a Gaussian Filter**
  A Gaussian filter is used to smooth the source image so as to remove image noise that may cause false-positive edge detection

- **Compute the image gradient**
  An image gradient is the two-dimensional gradient vector representing the directional change in intensity (brightness) of an image.
  These intensity values provide the information necessary to determine the positions and polarities of edges.

The image gradient is computed from convolving the source image with a derivative filter to get the first-order partial derivatives, which together, form the (image) gradient vector representation. A potential edge is simply identified by the values with the highest rates of change, so the derived values with the highest magnitude are potential edge candidates.

- **Apply non-maximum suppression**
  Non-maximum suppression is an edge thinning technique used to remove extraneous edge candidates. It works by iterating through all pixel values, comparing the current value with the pixel value in the positive and negative gradient directions, and suppressing the current value if it does not have the highest magnitude relative to its neighbors. For our implementation, we will be using a set of discrete gradient directions.

- **Apply double thresholding**
  Double thresholding is used to categorize the remaining edge pixels into three categories using a low and high threshold value. If an edge pixel value is greater than the high threshold value, it is categorized as a strong edge pixel, with a high probability of being an edge. If an edge pixel value is less than the high threshold value, but greater than the low threshold value, it is categorized as a weak edge pixel, with *some* probability of being an edge. If an edge pixel value is less than both the high and low threshold values, it is categorized as having a very low probability of being an edge, and the value is suppressed.

  - **Output Sample**

2.Robert Edge Detector

Measures a 2-D spatial gradient on an image in a straightforward, quick-to-compute manner. As a result, strong spatial gradient zones, which frequently correspond to edges, are highlighted. The operator's input and output are both grayscale images in their most typical configuration. The estimated absolute magnitude of the input image's spatial gradient at that position is represented by pixel values at each place in the output.

3.Sobel Edge Detector

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.In theory at least, the operator consists of a pair of 3×3 convolution kernels .One kernel is simply the other rotated by 90°. This is very similar to the Roberts Cross operator.

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

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

- **Output Sample**

4.Perwitt Edge Detector:

The Prewitt operator detects image edges by convolution with two filter masks. One for horizontal and one for vertical direction. At this way we can extract the coefficients of Prewitt masks are:

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * I; \quad Gy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * I$$

Prewitt operator with 3×3 masks

- **Output Sample**

4. Equalization

1. **equilization:** This function performs histogram equalization on the input grayscale image. The function first calculates the scaling factor alpha as 255.0 divided by the total number of pixels in the image. Then, the function scales the cumulative histogram cumhistogram to obtain the mapping function Sk, which maps each intensity value to its corresponding equalized intensity value. The function then creates a copy of the input image and applies the mapping function to each pixel intensity value to obtain the equalized image.

2. **equalizedHistogram:** This function calculates the equalized histogram of the input grayscale image, given its histogram and the mapping function sk obtained from histogram equalization. The function first calculates the probability density function (PDF) of each intensity value in the image histogram. Then, the function calculates the cumulative distribution function (CDF) of the PDF, and uses the mapping function sk to obtain the equalized histogram.

- **Output Samples**

1. Original Image

2. Equalized Image

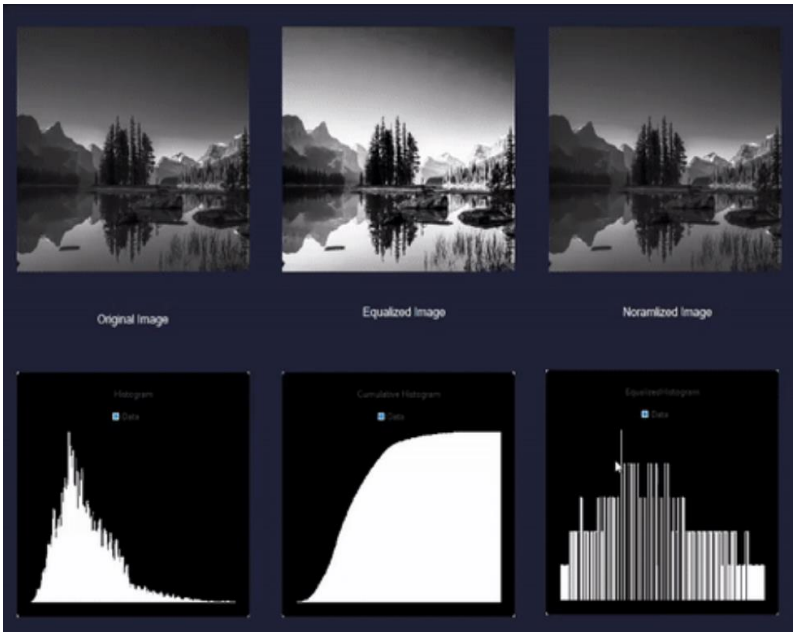
5. Histograms

- **Algorithms**
    1. **Histo:** This function calculates the histogram of the input grayscale image and stores the histogram values in the histogram array. The function initializes all intensity values to 0, and then iterates over each pixel in the image, calculates its intensity value, and increments the corresponding value in the histogram array.

    2. **cumHist:** This function calculates the cumulative histogram. The function initializes the first value of the cumulative histogram as the first value of the input histogram, and then iterates over each histogram value to calculate the corresponding cumulative histogram value.

    3. **histDisplay:** function creates a graphical representation of the input histogram. The function first creates an empty grayscale image with dimensions hist_h x hist_w, and then normalizes the input histogram values between 0 and hist_h. The function then iterates over each histogram value to draw a vertical line on the image representing the intensity value and its corresponding histogram value.

- **Output Samples**



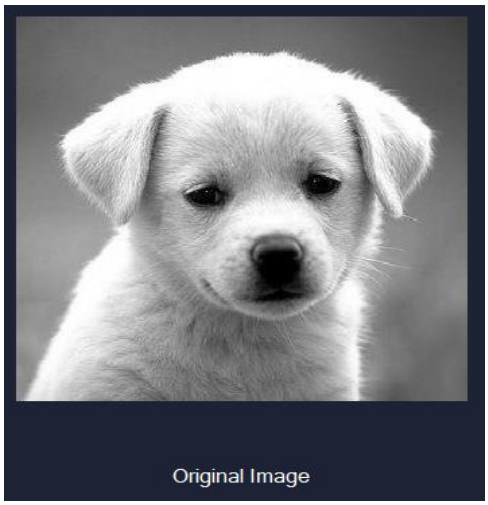
6. Normalized Image

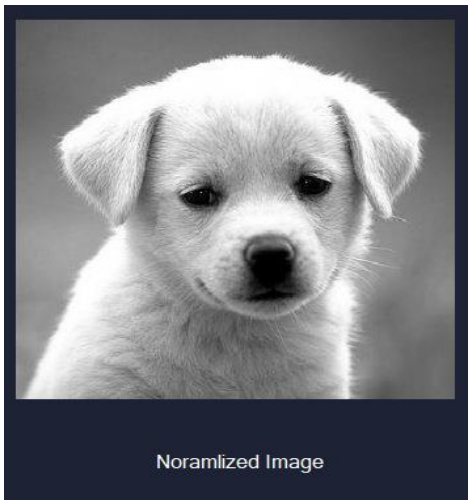
- **Algorithms**

1. **normalizeMat:** The function takes in a "cv::Mat object inputMat" and two double values "minVal" and "maxVal" as input parameters
2. It creates a new "cv::Mat object outputMat" with the same size and type as the inputMat
3. It then checks the type of the inputMat.
4. If the inputMat is a grayscale image, the function performs the following steps:
    a. Initializes the variables currentMinVal and currentMaxVal to the first pixel value of the inputMat
    b. Iterates over each pixel in the inputMat to find the minimum and maximum pixel values
    c. Calculates the normalized value for each pixel using the formula " (currentVal - currentMinVal) * (maxVal - minVal) / (currentMaxVal - currentMinVal) + minVal)"
    d. Assigns the normalized value to the corresponding pixel in the outputMat
5. If the inputMat is a BGR color image, the function performs the following steps:
    a. Separates the inputMat into its three color channels (blue, green, and red).
    b. For each color channel, initializes the variables currentMinVal and currentMaxVal to the first pixel value of the channel.

c. Iterates over each pixel in the channel to find the minimum and maximum pixel values.
d. Calculates the normalized value for each pixel in the channel using the formula "(currentVal - currentMinVal) * (maxVal - minVal) / (currentMaxVal - currentMinVal) + minVal)"
e. Assigns the normalized value to the corresponding pixel in the outputMat and returns the normalized image to it.

- **Output Samples**





7. Global and Local Thresholding

- **Algorithms**

1. Global Thresholding

The **GlobalthresholdImage** function takes an input image, a threshold value, and a reference to a final image as arguments. It creates an output image with the same size as the input image and applies the threshold value to each pixel of the input image. If the pixel value is greater than or equal to the threshold value, the output pixel value is set to 255, otherwise it is set to 0. The final thresholded image is stored in the final image reference.

2. Local Thresholding

The LocalThresholdImage function takes an input image, a block size, a k value, and a reference to a final image as arguments. It creates an output image with the same size as the input image and pads the input image to handle edge cases. It then applies a threshold to each pixel of the input image using a local thresholding method. For each

pixel, it computes the mean and standard deviation of the surrounding block of pixels with size equal to the block size. It then computes the threshold value for the pixel using the local mean and standard deviation and the k value. If the pixel value is greater than or equal to the threshold value, the output pixel value is set to 255, otherwise it is set to 0. The final thresholded image is stored in the final image reference.

- **Output Samples**

8. RGB to Grayscale Histogram

- **Algorithms**

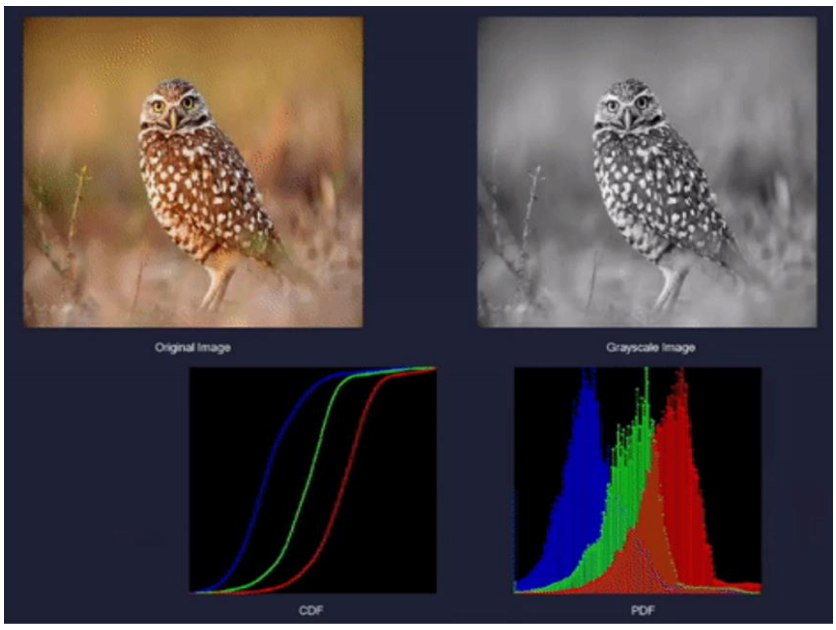
1. CDF Histogram
    - Define a function named plotRGBHistogramCDF that takes an input image img of type Mat and returns a histogram image of type Mat.
    - Create a Mat variable named histogram to store the histogram.
    - Split the input image into its separate red, green, and blue color channels using the split function and store them in a vector named bgrChannels.
    - Define the histogram parameters, including the number of bins, the range of pixel values, and whether the bin sizes are uniform or not.
    - Create Mat variables bHist, gHist, and rHist to store the histograms for each color channel.
    - Calculate the histograms for each color channel using the calcHist function.
    - Create Mat variables bCDF, gCDF, and rCDF to store the cumulative distribution function (CDF) for each color channel.
    - Copy the histograms for each channel to their corresponding CDF variables.
    - Calculate the CDF for each channel by adding the current bin value to the previous bin value for each histogram.
    - Create a Mat variable histogram to store the histogram image.
    - Define the histogram image dimensions and bin width based on the number of bins and desired image size.

- Normalize the CDFs to fit within the histogram image using the normalize function.
- Draw histogram lines for each color channel using the line function and the CDF values for each bin.
- Return the histogram image

2. PDF Histogram

- Define a function named plotRGBHistogramPDF that takes an input image img of type Mat and returns a histogram image of type Mat.
- Create a Mat variable named histogram to store the histogram.
- Split the input image into its separate red, green, and blue color channels using the split function and store them in a vector named bgrChannels.
- Define the histogram parameters, including the number of bins, the range of pixel values, and whether the bin sizes are uniform or not.
- Create Mat variables bHist, gHist, and rHist to store the histograms for each color channel.
- Calculate the histograms for each color channel using the calcHist function.
- Create a Mat variable histogram to store the histogram image.
- Define the histogram image dimensions and bin width based on the number of bins and desired image size.
- Normalize the histograms to fit within the histogram image using the normalize function.
- Draw histogram points and vertical lines for each color channel using the line function and the PDF values for each bin.
- Return the histogram image.

- **Output Samples**



9. Filtering in Frequency Domain

- **Algorithms**

  1. The main steps of the algorithm are reading the input image, creating a Mat object where the filtered output image will be stored, selecting the filter you want to apply, and defining D_0 (cutoff frequency) that is used to construct the filter.
  2. The input image is read using OpenCV's imread function and stored in a Mat object called "imgIn". The image is read in grayscale mode by passing the "0" flag as the second argument.
  3. The input image is resized to a predefined size ("imageSize") using OpenCV's resize function.
  4. The input image is converted to a 32-bit floating point format using OpenCV's "convertTo" function as the filtering operation requires floating point numbers.
  5. The Discrete Fourier Transform (DFT) of the input image is calculated using a custom function called "calculateDFT". The result is stored in a Mat object called "DFT_image".
  6. A filter is constructed based on the specified filter type and the value of D0. The construction of the filter is done using a custom function called "construct_H". The resulting filter is stored in a Mat object called "H".
  7. The input image is filtered in the frequency domain using the filter "H" and the DFT of the input image "DFT_image". The filtering operation is done using a custom function called "filtering". The result is stored in a Mat object called "complexIH".
  8. The Inverse Discrete Fourier Transform (IDFT) of the filtered image "complexIH" is calculated using OpenCV's "dft" function with the flags "DFT_INVERSE" and "DFT_REAL_OUTPUT". The result is stored in the Mat object "imgOut".
  9. The output image "imgOut" is normalized using OpenCV's normalize function with the arguments 0 and 1, and the resulting image is stored in "imgOut".
  10. The function returns with the filtered output image stored in the "imgOut" parameter passed by reference.

- **Implementation of Filters**

  1. Ideal Low Pass Filter

  It works by iterating through every pixel in the filter matrix "H", and for each pixel, calculate the distance D between the pixel and the center of the matrix (the point scr.rows/2, scr.cols/2). If D is **greater** than a threshold value D_0 (cutoff frequency), then the corresponding pixel in the H matrix is set to 0. The ideal D_0 value in our implementation is 30 but it can be changed by the user.

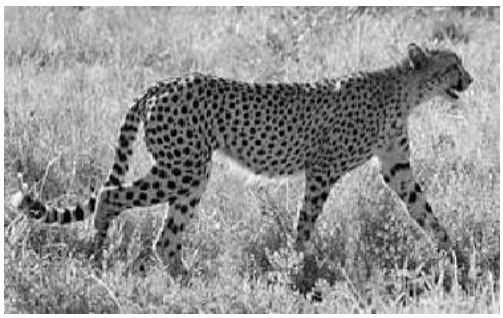
  2. Ideal High Pass Filter

  It works by iterating through every pixel in the H matrix, and for each pixel, calculate the distance D between the pixel and the center of the matrix. If D is **less** than or equal to a

threshold value D_0, then the corresponding pixel in the H matrix is set to 0. This has the effect of filtering out low-frequency components in the image and leaving high-frequency components.

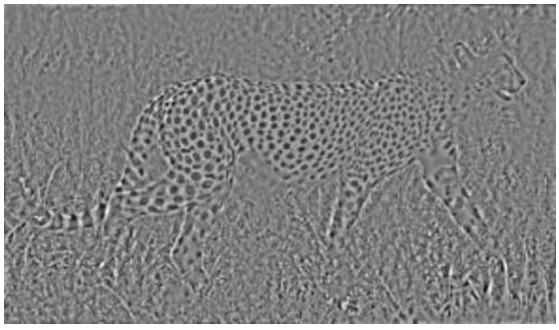
3. Gaussian Filter
It works by iterating through every pixel in the H matrix, and for each pixel, calculate the distance D between the pixel and the center of the matrix. Then, the corresponding pixel in the H matrix is set to the value of exp(-D*D / (2 * D_0*D_0)), where D_0 is a parameter that controls the width of the Gaussian filter. The expression exp(-D*D / (2 * D_0*D_0)) is the Gaussian function, which is used to weight the contribution of each pixel in the image to the filtered output. Pixels farther away from the center of the filter have a smaller weight, which results in a smoother output.

- **Output Samples**
  1. Original Images

2. LPF applied on elephant at D_0=30

3. HPF applied on tiger at D_0=30

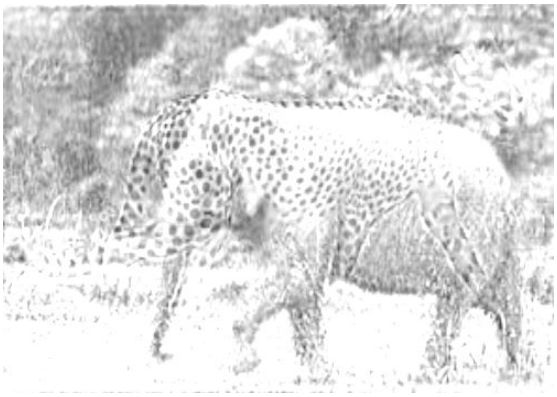


10. Hybrid Images

- **Algorithm**

Hybrid image is obtained using the function "mix_images", which blends a low-frequency filtered image (in frequency domain) with a high-frequency filtered image (both are filtered in frequency domain from the previous functions). The function takes three arguments: imgLow and imgHigh, both of type Mat, representing the low-frequency and high-frequency images, respectively; and imgHybrid, a Mat object representing the output hybrid image.

- **Output Sample**



That is the output image from the low and high frequency filtered images obtained from the previous step.