Assignment No.1
Computer Vision

By:
Ahmed Mohamed,
Ahmed Mahmoud,
Eman Emad,
Rawan Shoaib

I. Introduction

IN this file we explore noise addition, filtering, edge detection, transformation from color image to gray scale image, local and global thresholding and frequency domain transformations for clearer vision.

mds
Febrary 18, 2025

II. Tasks to implement

A. Add additive noise to the image:

Given a grayscale image represented as a 2D NumPy array of shape (H,W) with pixel values in the range [0,255], we aim to generate a noisy image by applying different types of noise: Uniform Noise: Adds random values from a uniform distribution. Gaussian Noise: Adds random values from a normal distribution. Salt-and-Pepper Noise: Randomly replaces pixels with 0 (black) or 255 (white) based on probability.

1) Algorithm :

- **Input**
  img: grayscale image 2D NumPy array of shape (H,W) with pixel values in the range [0,255]
  intensity: Maximum intensity for uniform noise.
  mean: Mean of the Gaussian noise.
  std: Standard deviation of the noise.
  salt_prob:Probability of white noise.
  pepper_prob: Probability of black noise.

- **Convert Image Format**
  To prevent overflow, convert grayscale image to (int16) a higher bit-depth format.
- **Apply Uniform Noise**
  Generate a noise matrix of shape (H,W) from a uniform distribution. And add this noise to the image.
- **Apply Gaussian Noise**
  Generate a noise matrix of shape (H,W) from a normal distribution. And add this noise to the image.
- **Apply Salt-and-Pepper Noise**
  Generate two random masks for salt and pepper noise and set pixel values to 255 (salt noise) where salt mask is True. Set pixel values to 0 (pepper noise) where pepper mask is True.
- **Clip and Convert Back**
  To ensure all pixel values remain in the valid range [0,255]. Convert back to an 8-bit format.
- **Output**
  noisy_img: Noisy grayscale image.

B. Filter the noisy image using low pass filters:

Given an input grayscale image of size (H×W), we want to obtain a filtered image by applying three different types of smoothing filters. These filters are Average filter (Mean filter), Gaussian filter and Median filter. Each filter operates using a convolution process, but the filtering method differs based on the kernel used.

1) Algorithm :

- **Input**
  img: grayscale image 2D NumPy array of shape (H,W) with pixel values in the range [0,255]
  kernel_size: Size of the low pass filter kernel (odd intege).
  sigma: Standard deviation of the Gaussian distribution (only for the Gaussian filter).

- **Apply the Average Filter**
  The average filter replaces each pixel value with the mean of its surrounding pixels within a K×K window. It creates a kernel of size K×K with uniform values then performs convolution on grayscale image.
- **Generate and Apply Gaussian Filter**
  Gaussian filter smooths the image using a weighted average, giving more importance to the central pixels. It creates a 1D Gaussian distribution along an axis, then computes the 2D Gaussian kernel using the outer product. And Normalize kernel so that it sums to 1, then convolves it with the image.
- **Apply the Median Filter**
  Median filter replaces each pixel with the median value of its surrounding K×K window. It Pads the image with edge values to avoid border issues, then iterates over each pixel to extract the K×K neighborhood, compute the median of all pixels in the region and assign the median to the output pixel. Finally returns the filtered image.

- **Output**
  0utput: Filtered grayscale image.

C. Detect edges in the image:

This set of functions implements multiple edge detection algorithms, including: Sobel Operator, Prewitt Operator, Roberts Operator and Canny Edge Detection. Each method follows a step-by-step filtering process using convolution, gradient computation, thresholding, and non-maximum suppression.

1) Algorithm :

- **Input**
  image: Input image (BGR or grayscale).
  kernel_size: Size of the Gaussian kernel for smoothing.
  sigma: Standard deviation for Gaussian blur.
  low_threshold: Lower threshold for edge detection.
  high_threshold: Upper threshold for edge detection.
  threshold, value: Threshold value for binary edge map.

- **Convert Image to Grayscale**
  If the input image is BGR, convert it to grayscale.
- **Apply Gaussian Smoothing**
  Applies a Gaussian filter to reduce noise. This step enhances edge detection by reducing noise effects.
- **Sobel Edge Detection**
  Compute Sobel Gradients (Horizontal & Vertical). Convolve the image with Sobel kernels. Compute the gradient magnitude. Normalize to 0-255 range for visualization.
- **Prewitt Edge Detection**
  Use Prewitt Kernels instead of Sobel. Compute Gradients & Magnitude (same as Sobel). Apply global thresholding to highlight significant edges.
- **Roberts Edge Detection**
  Apply Roberts Cross Operator. Compute Gradients & Magnitude. Return Edge Map.
- **Canny Edge Detection**
  Apply Gaussian Smoothing. Compute Sobel Gradients. Compute Edge Direction (Theta) using the arctangent of gradients. performe Non-Maximum Suppression which Suppresses non-edge pixels by comparing neighbors. Apply Double Thresholding to classify edges as strong or weak. Apply Hysteresis Tracking so that weak edges connected to strong edges are retained.
- **Displaying Results**
  Function display_edge_detection() shows: Original image, detected edges using a selected method and uses Matplotlib for visualization.
- **Output**
  Edge Detected Image: Grayscale Edge Map.

D. Draw histogram and distribution curve:

Given a dataset of an image, computes its histogram, and overlays a normal distribution curve fitted to the data. The goal is to visualize the distribution of the dataset and compare it to an ideal normal distribution.

1) Algorithm :

- **Input**
  data: A numerical dataset.
  bin: The number of bins for the histogram.

- **Plot the Histogram**
  Compute the histogram using histogram binning (Divides data into bins number of intervals), normalization to ensures the histogram represents a probability density and edge color to make bin edges more visible.
- **Compute the Normal Distribution Curve**
  Compute normal distribution parameters the mean and standard deviation. To compute the curve it generates 100 evenly spaced values from the minimum to maximum of the dataset. And computes the probability density function of the normal distribution.
- **Output**
  x: A set of evenly spaced values used to plot the normal distribution curve.
  y: The corresponding probability density function (PDF) values for the normal curve.

E. Equalize and Normalize the image:

Given a dataset of an image, this set of functions is designed to enhance image contrast through histogram equalization and normalization for both grayscale and color images.

1) Algorithm :

- **Input**
  image: A grayscale or color image (2D NumPy array, shape (height, width), dtype uint8).

- **equalize_grayscale_image**
  Compute the histogram of the grayscale image. Compute the cumulative distribution function (CDF) from the histogram. Normalize the CDF to map old intensity values to new ones. Apply histogram equalization to get the final equalized image.
- **equalize color image**
  Split the BGR image into three channels: Blue, Green, Red. Apply equalize_grayscale_image() to each channel separately. Merge the equalized channels back into a single BGR image.
- **normalize grayscale image**
  Compute the minimum and maximum pixel values. Apply min-max normalization. Convert to 8-bit integer format.
- **normalize color image**
  Normalizes each BGR color channel separately. Process each channel (Blue, Green, Red) using normalize_grayscale_image(). Merge normalized channels back to form a BGR image.
- **process image**
  Detect if the image is grayscale or color. Apply equalization and/or normalization based on user input. Return the processed image.
- **Output**
  equalize_grayscale_image: Equalized grayscale image.
  equalize_color_image: Equalized color image (BGR).
  process_image: Processed image.
  normalize_color_image: Normalized color image (BGR).
  normalize_grayscale_image: Normalized grayscale image.

F. Local and global thresholding:

Given an input grayscale image we convert it into a binary image by thresholding. We perform binary image thresholding using either a global threshold or a local adaptive method (Niblack's thresholding).

1) Algorithm :

- **Input**
  img: grayscale image 2D NumPy array of shape (H,W) with pixel values in the range [0,255]
  kernal: Neighborhood window size for local thresholding.
  T: Global threshold (default 128, or computed from the image mean).
  local: Boolean flag to enable local adaptive thresholding.
  k: Niblack's parameter controlling threshold sensitivity.

- **Select Thresholding Method**
  Check if local thresholding is enabled: If True, apply Niblack's local thresholding (localthresholding). If False, use global thresholding (globalthresholding). If no global threshold T is given, compute it as the image mean.
- **Global Thresholding**
  This method applies a fixed threshold T globally to all pixels. It converts the condition into a binary mask and multiply by 255 to obtain a black-and-white image.
- **Local Thresholding (Niblack's Method)**
  This method computes an adaptive threshold for each pixel based on its neighborhood. It ensures kernel size is odd, pads the image to handle edge pixels, iterates through each pixel, extract its neighborhood, compute threshold and Applies it to the image.
- **Output**
  binary_img: Binary (black & white) thresholded image.

G. Transformation from color image to gray scale:

Given a BGR image of size H×W×3, convert it into a grayscale image using a perceptual weighted sum of the Red (R), Green (G), and Blue (B) channels (luminance calculation).

1) Algorithm :

- **Input**
  image: Color image in BGR format (NumPy array of shape H×W×3).

- **Convert BGR to RGB**
  Since OpenCV loads images in BGR format, convert it to RGB This ensures that color channels are correctly mapped before grayscale conversion.
- **Compute Weighted Grayscale Intensity**
  Extract the Red, Green, and Blue components. This separates the 3 color channels into individual matrices. Apply the weighted sum formula this formula is derived from human perception of brightness: Green contributes the most (0.587) because the human eye is most sensitive to green. Red contributes moderately (0.299). Blue has the least impact (0.114).
- **Convert to uint8 and Return**
  Since pixel values must be in the range [0, 255], convert the result.
- **Output**
  gray_image: Grayscale image (NumPy array of shape H×W, dtype uint8).

H. Frequency domain filters:

Given an input grayscale image the goal is to compute its Discrete Fourier Transform. Apply an ideal frequency filter (low-pass or high-pass). Transform the image back to the spatial domain.

1) Algorithm :

- **Input**
  img: grayscale image 2D NumPy array of shape (H,W) with pixel values in the range [0,255]
  cutoff: Cutoff frequency (default = 10).
  type: Filter type (”lp” for low-pass, ”hp” for high-pass).
  dft_shifted: (NumPy array, shape: H×W) The shifted DFT representation of the image (complex values).
  mask: A binary mask that defines which frequencies are retained (1) or removed (0).

- **Compute the DFT of the Image**
  The input image is transformed into the frequency domain using DFT.
- **Generate the Ideal Frequency Filter**
  A binary mask is created in the frequency domain. Initialize a mask of the same size as dft_shifted, filled with zeros, then Find center of frequency domain. Iterate over each pixel to decide whether it should be part of the filter.
- **Apply the Filter and Compute Inverse DFT**
  Apply the filter in frequency domain. Shift the zero frequency component back. Compute the inverse DFT to return to the spatial domain. Take magnitude (remove imaginary part). Clip pixel values to valid range [0,255].
- **Output**
  image_filtered: Filtered Image in the spatial domain.

I. Hybrid images:

This algorithm creates a hybrid image by combining two filtered images in the frequency domain using ideal filtering.

1) Algorithm :

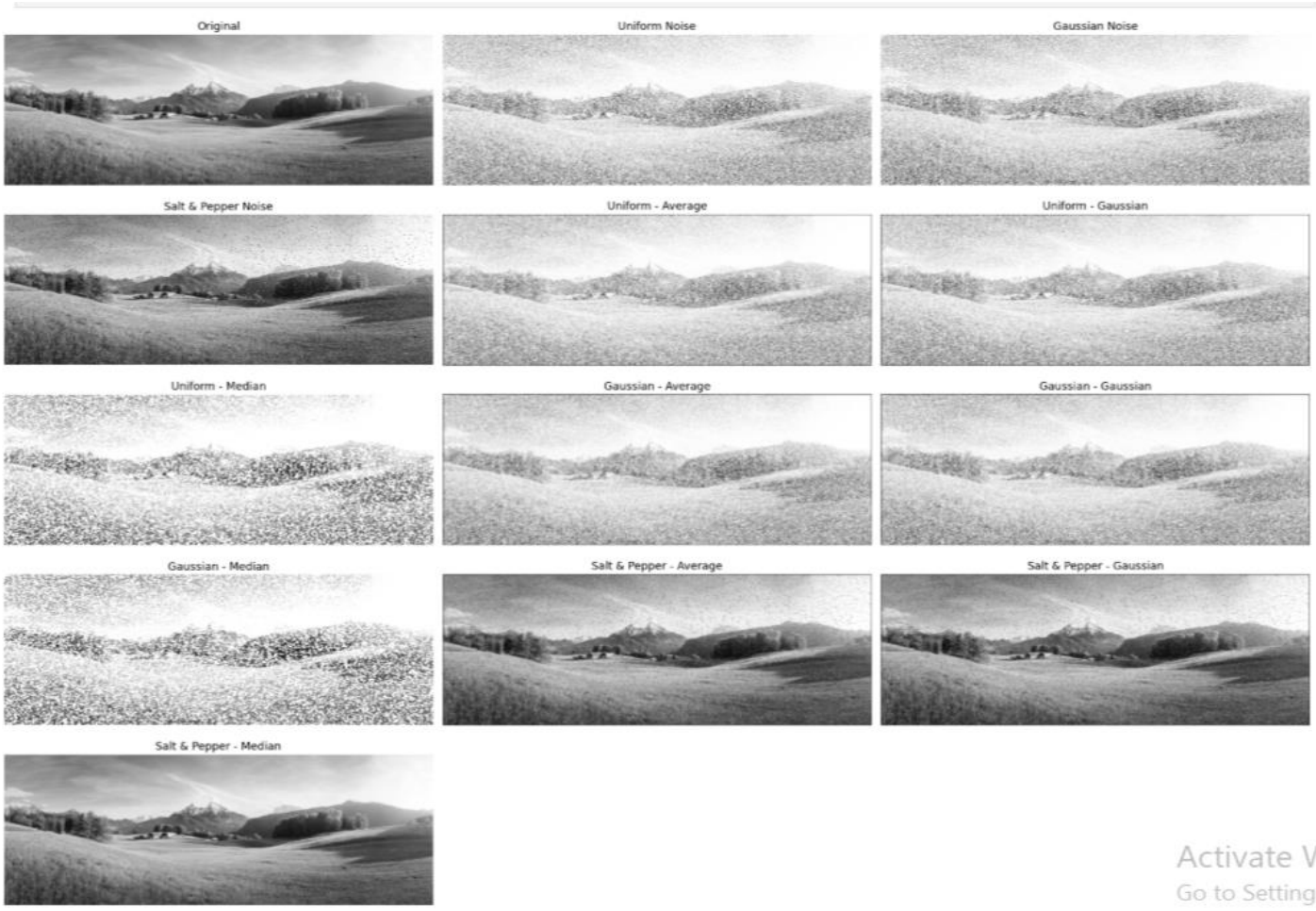
- **Input**
  img_1: First image (grayscale, NumPy array, shape H×W).
  img_2: Second image (grayscale, NumPy array, shape H×W).
  cutoff1: Defines the cutoff frequency for the first image.
  cutoff2: Defines the cutoff frequency for the second image.

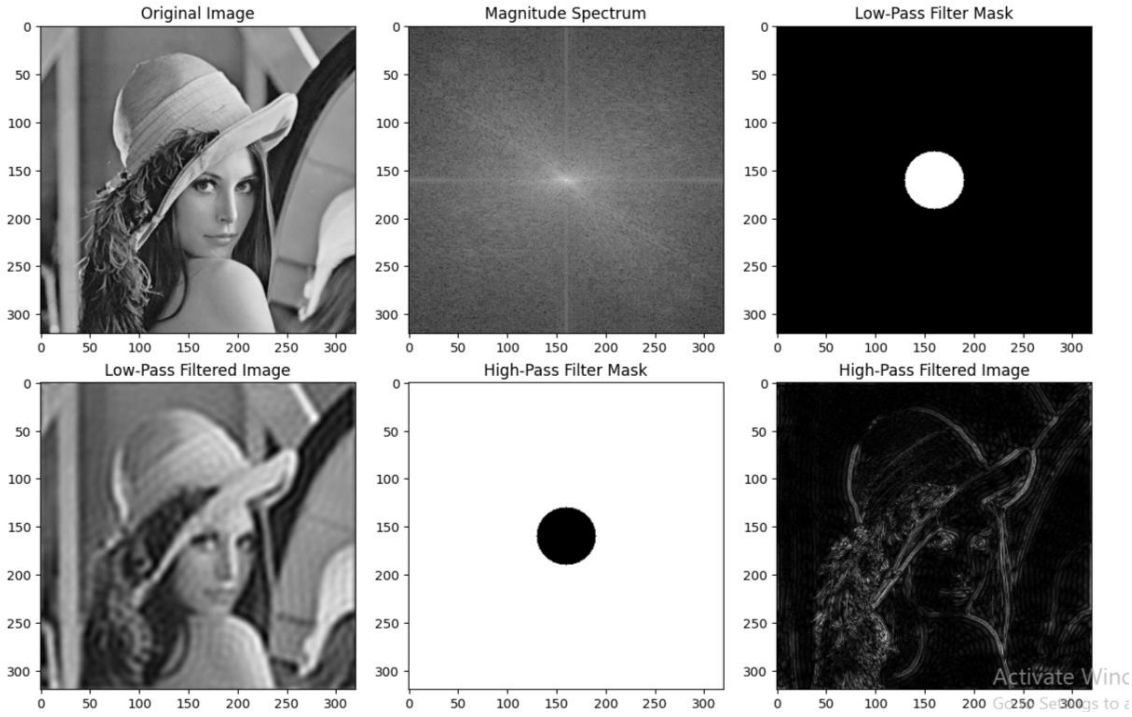
  type1: Defines the type of filter applied to the first image (”lp” for low-pass, ”hp” for high-pass).
  type2: Defines the type of filter applied to the second image (”lp” for low-pass, ”hp” for high-pass).

- **Apply Ideal Frequency Filtering**
  The function ideal_filter is applied to both images. It retains low frequencies (blurred image) or high frequencies (sharp edges) depending on the type of filter.
- **Combine the Filtered Images**
  The two filtered images are added together. Creates a hybrid effect where low frequencies dominate at a distance and high frequencies dominate up close.

- **Normalize and Clip the Output**
  The final hybrid image is clipped to the valid pixel range [0,255] and converted to uint8 to ensures pixel values are within the valid grayscale range.
- **Output** hybrid_image: NumPy array of size (H,W).

III. Examples





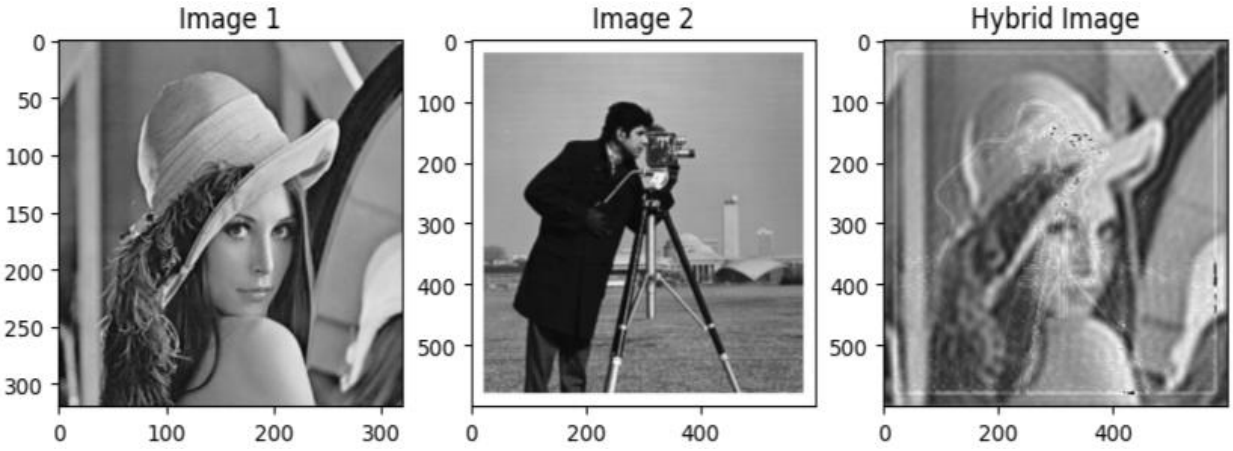


Image 1
Image 2
Hybrid Image