Amirreza Akbari | 810899045

Task 1:

Task 1 encompassed a series of image manipulation techniques applied to an initial image to explore various aspects of image processing.

1. **Image Format Conversion:** The initial image underwent transformation into binary, grayscale, and RGB formats using OpenCV's conversion functions (cv2.cvtColor and cv2.threshold).

2. **Contrast Adjustment:** Contrast adjustment was performed on the grayscale and RGB images using a custom function (adjust_contrast). This function utilized the median of the gray values as a reference point and adjusted the contrast by applying linear functions with different increase or decrease factors.

3. **Pixel Manipulation:** All gray values of the image were increased by a specified amount using simple arithmetic operations. This manipulation, implemented through addition (np.clip), aimed to uniformly shift the intensity levels of pixels in the image.

4. **Noise Addition and Reduction:** Salt and pepper noise was added to the initial image using a custom function (add_salt_pepper_noise), simulating random noise commonly encountered in images.

5. **Noise Reduction:** Noise reduction was achieved for the output image of the previous step by applying a median filter (cv2.medianBlur), aiming to remove outliers and preserve image structure.

6. **Edge Detection:** Edge detection algorithms, including the Sobel and Canny operators (cv2.Sobel, cv2.Canny), were employed to identify and extract edges within the image. These operators convolved the image with kernels to highlight areas of rapid intensity change, revealing object boundaries and outlines.

7. **Image Sharpening:** An appropriate sharpening operator was applied to enhance image clarity and detail (cv2.detailEnhance). This operation emphasized edges and finer details within the image, resulting in a more visually appealing output.

8. **Image Blurring:** Average and Gaussian blurring operations (cv2.blur, cv2.GaussianBlur) were performed to smooth out the image and reduce noise. These operations convolved the image with a kernel to average pixel values, effectively reducing high-frequency components and producing a softer, more uniform appearance.

Task 2:

Task 2 involved various image decomposition, manipulation, and analysis techniques applied to the initial image saved from Task 1.

1. **Image Decomposition:** The initial image was decomposed into its color components in RGB, HSV, and YCrCb color formats using OpenCV's conversion function (cv2.cvtColor). The decomposed images were saved individually for further analysis (cv2.imwrite).
2. **Grayscale Image Operations:** Various image filtering operations were applied to the grayscale version of the initial image using functions such as cv2.blur, cv2.GaussianBlur, cv2.Sobel, cv2.Canny, cv2.medianBlur, cv2.dilate, cv2.erode, and custom functions for FIR and IIR filters.

These operations aimed to perform tasks such as blurring, edge detection, noise reduction, and frequency domain filtering.

3. **Histogram Analysis:** Histograms of both the grayscale and RGB versions of the initial image were plotted using matplotlib (plt.hist). These histograms provided insights into the distribution of pixel intensities in the image and were analyzed in terms of contrast.
4. **Histogram Equalization:** Histogram equalization was applied to the grayscale image using cv2.equalizeHist to enhance contrast. The equalized histogram and the corresponding image were plotted and saved for comparison.
5. **Fourier Transform Analysis:** The Fourier transform of the grayscale image was computed using numpy's fft module. The magnitude spectrum of the Fourier transform was visualized to analyze the frequency components present in the image.

Task 3:

Task 3 involved analyzing the initial image to determine its format, resolution, compression status, and associated attributes. This analysis was facilitated by various steps implemented in the provided code.

1. **Header Bytes Extraction and Image Format Identification:** The initial image's file path was provided, and the image was loaded using OpenCV's imread function. The code segment utilizing open function and read method was responsible for extracting the first 100 bytes of the image file to obtain header information. By examining the extracted header bytes, the code determined the image format based on predefined signature patterns to check for JPEG, PNG, and BMP signatures.
2. **Resolution and Compression Analysis:** The resolution of the image (width x height) was obtained using the shape attribute of the image array. The file size of the image was determined using the os.path.getsize function. The code assessed the compression status of the image by comparing its file size with the size that would be occupied by uncompressed pixel data. For compressed images, the compression ratio was calculated by dividing the compressed size by the uncompressed size.
3. **Compression Algorithm and Type Determination:** If the image was compressed, the code identified the compression algorithm and type (lossy or lossless) based on the image format. JPEG images were associated with the Discrete Cosine Transform (DCT) compression algorithm, while PNG images used the DEFLATE (LZ77-based) algorithm.

Task 4:

Task 4 involved comparing different images of the user's face captured under various conditions with a reference image of the user's front posture. This comparison aimed to calculate the similarity between each image and the reference image using an appropriate template matching algorithm, specifically Histogram Matching.

1. **Image Preparation:** The user captured images of their face under four different conditions: front posture, closer to the camera, with a 45-degree rotation, and with some part of the face covered by an object. These images were loaded into the program using OpenCV's imread function and converted to grayscale for further processing.

2. **Histogram Matching:** Histogram Matching, an effective template matching algorithm, was applied to the grayscale images of the user's face under different conditions. This process involved using the Contrast Limited Adaptive Histogram Equalization (CLAHE) method to enhance image contrast and improve histogram matching accuracy. Histograms of pixel intensities were calculated for each image using the calcHist function.
3. **Histogram Plotting:** The histograms of pixel intensities for the reference image and the three comparison images (closer, diagonal, covered) were plotted to visualize their distributions. The histograms represented the frequency of pixel intensities in each image, allowing for a qualitative analysis of their similarities and differences.
4. **Histogram Normalization:** To facilitate meaningful comparison, the histograms were normalized using OpenCV's normalize function. Normalization scaled the histograms to the range [0, 1], ensuring consistent comparison across images with varying intensity distributions.
5. **Histogram Similarity Calculation:** Histogram similarity scores were computed using the compareHist function, which employed the correlation coefficient metric (cv2.HISTCMP_CORREL) to quantify the similarity between pairs of histograms. This metric measured the correlation between the histograms, with values closer to 1 indicating higher similarity.
6. **Results and Analysis:** The similarity scores between each image and the reference image were calculated and printed. These scores provided quantitative measures of how closely each image resembled the reference image in terms of pixel intensity distributions. The comparison results enabled the assessment of image similarity under different conditions, offering insights into the effectiveness of Histogram Matching as a template matching algorithm for facial recognition and analysis.

Task 5:

Task 5 involved analyzing brain MRI image samples of different patients to detect the presence and location of brain lesions (tumors). This task comprised two main steps: plotting histograms of the left and right hemispheres of the brain and applying Histogram Matching technique to identify lesions, followed by lesion detection using an appropriate algorithm.

1. **Histogram Analysis:** The first step involved plotting histograms of the left and right hemispheres of the brain to analyze their intensity distributions. The histograms provided insights into the pixel intensity characteristics of each hemisphere, serving as a basis for lesion detection. To accurately assess similarity between histograms and identify lesions, background pixels were removed from the images and their histograms.
2. **Histogram Matching Technique:** Histogram Matching technique was applied to compare the intensity distributions of the left and right hemispheres and detect the presence of brain lesions. By determining suitable threshold values for dissimilarity, deviations from expected intensity distributions indicative of lesions were identified. Histogram Matching facilitated the quantitative assessment of similarity between histograms, enabling the detection of abnormal patterns associated with brain lesions.
3. **Lesion Detection Algorithm:** Following histogram analysis and lesion identification, an appropriate algorithm was employed to detect the location of brain lesions (tumors). The algorithm utilized image processing techniques to analyze image features and identify regions of interest corresponding to lesions. By leveraging characteristic properties of lesions, such as shape, texture, and intensity contrast with surrounding tissue, the algorithm accurately localized and delineated the extent of lesions within the brain MRI images.