Practical No. 1 & 2

Aim : Consider any image with size 1024*1024. Modify the image to the sizes 512*512, 256*256, 128*128, 64*64 and 32*32 using subsampling technique. Create the original image from all the above subsampled images using resampling technique. Read any image. Display the histogram, Equalized histogram, and image with equalized histogram.

Objective: To learn the resize/scaling/ interpolation concept in Image processing • Input – RGB images of size 1024*1024.

Theory:

Image digitization (Sampling and quantization) –

In Digital Image Processing, signals captured from the physical world need to be translated into digital form by "Digitization" Process. In order to become suitable for digital processing, an image function f(x,y) must be digitized both spatially and in amplitude. This digitization process involves two main processes called-

1. Sampling: Digitizing the co-ordinate value is called sampling.
2. Quantization: Digitizing the amplitude value is called quantization

The subsampling was accomplished by deleting the appropriate number of rows and columns from the original image. For example, the 512*512 image was obtained by deleting every other row and column from the 1024*1024 image. The 256*256 image was generated by deleting every other row and column in the 512*512 image, and so on. The number of allowed gray levels was kept at 256. These images show the dimensional proportions between various sampling densities, but their size differences make it difficult to see the effects resulting from a reduction in the number of samples. The simplest way to compare these effects is to bring all the subsampled images up to size 1024*1024 by row and column pixel replication.

Quantization is opposite to sampling. It is done on y axis. When you are quantizing an image, you are actually dividing a signal into quanta(partitions). On the x axis of the signal, are the co-ordinate values, and on the y axis, we have amplitudes. So digitizing the amplitudes is known as Quantization.

Procedure:

1. Read and load the images obtained in the previous step of sizes 512*512, 256*256, 128*128, 64*64, 32*32. (One by one).
2. Set the output size as: 1024*1024.
3. Resize the input image to resample it, using the cv2.resize() function by passing the input image and output size as parameters.
4. Also, set the parameter 'interpolation' to cv2.INTER_LINEAR so that it follows linear interpolation while resampling. (Other types of interpolation can also be tested).
5. Display the resampled images.

Histogram Equalization –

The histogram technique that is used to enhance the brightness and contrast of an image is histogram equalization. The goal of histogram equalization is to distribute the gray levels within an image so that every gray level is equally likely to occur. In other words, histogram equalization takes an image's histogram and produces a new image with a histogram that is uniformly distributed. Histogram equalization will increase the brightness and contrast of a dark and low contrast image, making features observable that were not visible in the original image. Since histogram equalization distributes an image's gray levels uniformly about the range of gray levels, all images will have approximately the same brightness and contrast, hence allowing images to be compared equally without a bias due to perceived contrast and brightness differences.

Conclusion: We have successfully modified the image to the sizes 512*512, 256*256, 128*128, 64*64 and 32*32 using subsampling technique and create the original image from all the subsampled images using resampling technique and also display equalized histogram for the images.

Practical No. 3

Aim : Read any image. Display the histogram, Equalized histogram, and image with equalized histogram

Objective: To learn to plot image in the form histogram and apply histogram equalization.

Theory:

What we can explore from Histogram? Why Histogram equalization?

We can deduce a great deal about the appearance of the image from its histogram –

i) In a dark image, the gray levels would be clustered at lower ends
ii) In an uniformly bright image, the gray levels would be clustered at the upper end
iii) In a well contrasted image, the gray levels would be well spred out over much of the image

Applications:

For image enhancement
a) Displaying Histogram of input image:
1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Obtain the histogram of the grayscale image using the cv2.calcHist() function.
   Set the following parameters:
   Input image
   Channels: [0] (since the image is grayscale).
   Mask: None (since we require the histogram of the entire image and not a specific part).
   histSize: [256] (represents the bin count; 256 in case of grayscale image).
   range: [0, 256] (the range of values that a pixel can have).
4. Plot the histogram using the matplotlib library.

Histogram :

The histogram of an image is a plot of the number of occurrences of gray levels in the image against gray level values. Histograms are the basis for numerous spatial domain processing techniques. It plays an important role in enhancement of perceived brightness and contrast of an image. It specifies the number of pixels having each gray level, but gives no hint as to where those pixels are located within the image. The histogram of an N × M image is defined as the percentage of pixels within the image at a given gray level:

$$h_i = \frac{n_i}{MN} \qquad for\ 0 \le i \le G_{\max}$$

where n_i is the number of pixels at gray level i,
NM is the total number of pixels within the image and
G_{max} is the maximum gray level value of the image.

Histogram Equalization :

The histogram technique that is used to enhance the brightness and contrast of an image is histogram equalization. The goal of histogram equalization is to distribute the gray levels within an image so that every gray level is equally likely to occur. In other words, histogram equalization takes an image's histogram and produces a new image with a histogram that is uniformly distributed. Histogram equalization will increase the brightness and contrast of a dark and low contrast image, making features observable that were not visible in the original image. Since histogram equalization distributes an image's gray levels uniformly about the range of gray levels, all images will have approximately the same brightness and contrast, hence allowing images to be compared equally without a bias due to perceived contrast and brightness differences.

Procedure :

1. Read and load the grayscale image generated in the previous step.
2. Obtain the equalized histogram of the grayscale image using the cv2.equalizeHist() function.
3. Obtain the corresponding histogram which can be displayed for the equalized histogram using the cv2.calcHist() function. Set the parameters as set above.
4. Plot the equalized histogram using the matplotlib library.
5. Display the image corresponding to the equalized histogram using cv2 library.

Conclusion: We have successfully read the image and display the histogram, Equalized histogram, and image with equalized histogram.

Practical No. 4

Aim : Read any image. Display the outputs of contrast stretching, intensity level slicing.

Objective : To Study Various Methods for Image Enhancement using Spatial and Frequency Domain.

Theory:

Image enhancement –

The principal objective of enhancement is to process an image so that the resultis more suitable than the original image for a specific application. Image enhancement approaches fall into two broad categories: spatial domain methods and frequency domain methods. The term spatial domain refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image. Frequency domain processing techniques are basedon modifying the Fourier transform of an image.

Image enhancement is the process of adjusting digital images so that the results are more suitable for display or further image analysis. For example, you can remove noise, sharpen, or brighten an image, making it easier to identify key features.

Contrast Stretching:

One of the simplest piecewise linear functions (function whose pieces are linear) is a contrast stretching transformation. The form of the piecewise function can be arbitrary complex. A practical implementation of some important transformation can be formulated only as piecewise function. It increases the dynamic range of the gray levels in the image being processed. Low contrast images are resulted from poor illumination, lack of dynamic range in the imaging sensor, or even wrong setting of a lens aperture of image acquisition.

Transformation function is shown in the graph. Locations of the points (r1, s1) and (r2, s2) used to control the shape of the transformation function. If r1=s1 and r2=s2 transformation is linear function and produces no changes in the gray level. If r1=r2, s1=0 and s2=L-1, transformation becomes a thresholding function that results a binary image. Intermediate values of (r1, s1) and (r2, s2) produce various degrees of spread in the gray levels of the output image, thus affecting its contrast.

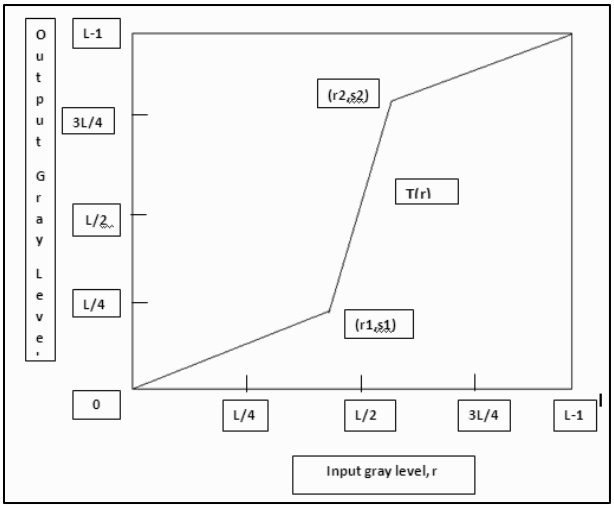
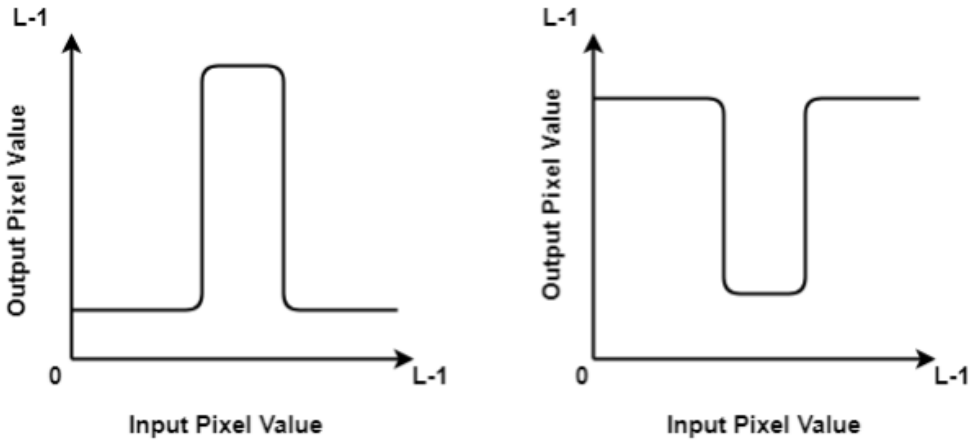


Fig : Contrast Stretching

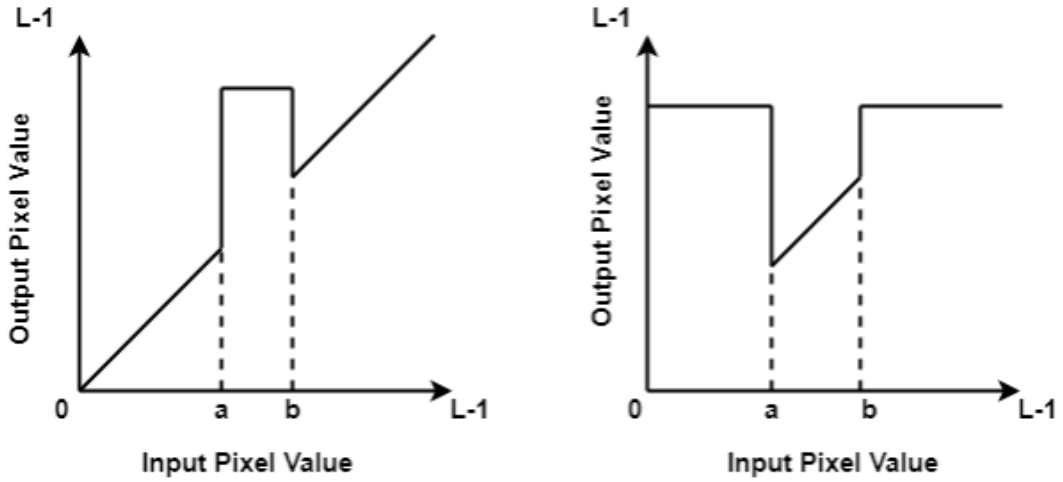
Intensity level slicing-

Intensity level slicing means highlighting a specific range of intensities in an image. In other words, we segment certain gray level regions from the rest of the image. Suppose in an image, your region of interest always take value between say 80 to 150, So, intensity level slicing highlights this range and now instead of looking at the whole image, one can now focus on the highlighted region of interest. Since, one can think of it as piecewise linear transformation function so this can be implemented in several ways. Here, we will discuss the two basic type of slicing that is more often used.

In the first type, we display the desired range of intensities in white and suppress all other intensities to black or vice versa. This results in a binary image. The transformation function for both the cases is shown below.



In the second type. we brighten or darken the desired range of intensities(a to b as shown below) and leave other inten- sities unchanged or vice versa. The transformation function for both the cases, first where the desired range is changed and second where it is unchanged, is shown below.



Procedure:

a) Contrast Stretching -

Applying contrast stretching to the input image:

1. Read and load the input RGB image.
2. Split the R, G, B channels of the image.
3. For each channel:
   • Initialise the stretch factors.
   • Determine the min and max values in each channel (max_val, min_val).
   • Calculate the stretch factor for each channel using the following formula:
   • stretch_factor_1 = (stretch_max - stretch_min) / (max_val - min_val)
   • stretch_factor_2 = stretch_min - stretch_factor_1 * min_val
4. Initialize the stretched image with zeroes.
5. Obtain the three stretched channels one by one using cv2.convertScaleAbs() function by passing the above calculated stretch factors to it.
6. Store the stretched channels as parts of the stretched image.
7. 7. Display the contrast stretched image using cv2 library.

b) Intensity level Slicing

Applying intensity level slicing to the input image:

1. 1.Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Set the lower and upper intensity levels.
4. Create a mask with the intensity range using the intensity levels set above and the obtained grayscale image. 5. Apply this mask to the original image by performing a bitwise AND to obtain the intensity sliced image.
5. 6. Display the intensity sliced image using the cv2 library.

Conclusion: We have successfully read the image and display the outputs of contrast stretching, intensity level slicing.

Practical No. 5

Aim : Compare the results of any three edge detection algorithms on the same image dataset and do the analysis of the result.

Objective : To Learn Classification Techniques for Image Segmentation.

Theory:

Image Segmentation

The first step in image analysis is to segment the image. Segmentation is the process to subdivide the image into its constituent parts or objects. The level to which subdivision is carried depends on the problem being solved.

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).When applied to a stack of images, typical in medical imaging, the resulting contours after image segmentation can be used to create 3D reconstructions with the help of interpolation algorithms like Marching cubes.

Segmentation algorithms for monochrome images are generally based on one of the two basic properties of grey level values:

1. Discontinuity: In this category the approach is to partition the image based on abrupt changes in grey level.
2. Similarity: In this category the approaches are based in thresholding, region growing, region splitting and merging.

Edge Detection :

Edge detection is by far the most common approach for detecting meaningful discontinuities in gray level. The reason is that isolated points and the thin lines are not frequent occurrences in most practical applications. An edge is the boundary between two regions with relative distinct gray level properties. Note from the profile that an edge (transition from dark to light) is modeled as a smooth, rather than as an abrupt, change of gray level. This model reflects the fact that edges in digital images are generally slightly blurred as a result of sampling.

1. Gradient Operator :

The gradient of an image f (x,y) at the location (x,y) is defined as

$$\nabla \mathbf{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}.$$

Gradient vector always points in the direction of maximum rate of change of f at co-ordinates (x,y). Magnitude of the vector is the main quantity in edge detection and given by,

$$\nabla f = \text{mag}(\nabla \mathbf{f}) = \left[G_x^2 + G_y^2\right]^{1/2}.$$

Different Gradient Operators:

Use of different gradient operators for the following 3x3 grey level image is discussed below.

Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

3x3 Grey Level Image

2. Roberts Operator:

One of the simplest way to implement first order partial derivative is to use Roberts cross-gradient operator.The mask for Roberts operator are given by,

-1	0
0	-1

0	-1
1	0

The two cross differences for x and y gradient components are given by equation,

$$Gx = Z9 - Z5 \text{ and } Gy = Z6 - Z8.$$

Then we can implement first order derivative ▼f at point Z5.

Mask of size 2x2 is awkward to implement because they don't have clear center. So generally, 3x3 masks are used.

Sobel operator edge detection algorithm –

1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Apply the Sobel operator to find the x and y gradients of the image using the cv2.Sobel() function.
4. Calculate the magnitude and angle of the gradients using the following formulae:
   - magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
   - angle = np.arctan2(sobel_y, sobel_x)
5. Normalize the magnitude to the range [0, 255].
6. Create a binary image based on the gradient magnitudes.
7. Display the binary image with the detected edges using the cv2 library.

Canny edge detection algorithm –

1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Apply Gaussian blur to reduce noise.
4. Use the Canny edge detector to find the edges by setting the upper and lower thresholds and using the cv2.Canny() function.
5. Create a binary image using the obtained edges.
6. Display the binary image with the detected edges using the cv2 library.

Laplacian operator edge detection algorithm –

1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Apply Gaussian blur to reduce noise.
4. Use the Laplacian operator edge detector to find the edges using the cv2.Laplacian() function.
5. Normalize the magnitude to the range [0, 255].
6. Create a binary image using the obtained edges.
7. Display the binary image with the detected edges using the cv2 library.

Conclusion : We have successfully compare the results of edge detection algorithms on the image dataset and analysis of the result.

Practical No. 6

Aim : Compare the result of any two image segmentation algorithm on the same image data set.

Objectives : To Learn Classification Techniques for Image Segmentation.

Theory:

Image Segmentation

The first step in image analysis is to segment the image. Segmentation is the process to subdivide the image into its constituent parts or objects. The level to which subdivision is carried depends on the problem being solved.

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).When applied to a stack of images, typical in medical imaging, the resulting contours after image segmentation can be used to create 3D reconstructions with the help of interpolation algorithms like Marching cubes.

Segmentation algorithms for monochrome images are generally based on one of the two basic properties of grey level values:

1. Discontinuity: In this category the approach is to partition the image based on abrupt changes in grey level.
2. Similarity: In this category the approaches are based in thresholding, region growing, region splitting and merging.

Threshold based image segmentation algorithm –

1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Set the threshold value to differentiate between regions to create a binary image.
4. E.g.: if the threshold value is set to 120, then the pixels with intensity values greater than 120 are set to 255 (white) while pixels with intensity values less than 120 are set to 0 (black).
5. Create a binary image based on the generated regions.
6. Display the binary image using the cv2 library.

Watershed image segmentation algorithm –

1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Apply Gaussian blur to reduce noise.
4. Threshold the image to create a binary image.
5. Create a kernel for the morphological operation.
6. Perform morphological opening to remove small objects using the function: cv2.morphologyEx().
7. Create a mask for the background.
8. Create a mask for the foreground.
9. Find the markers for the watershed transformation.
10. Add one to all labels so that the background is not 0, but 1.
11. Set the background to 0.
12. Perform the watershed transformation using the function: cv2.watershed().
13. Create a binary image based on the generated regions.
14. Display the binary image using the cv2 library.

Conclusion: We have successfully compare the result of two image segmentation algorithm on the image.

Practical No. 7

Aim : Write a program for image compression using any three compression techniques and compare the results.

Objectives : To Understand Image Compression.

Theory:

What is Image Compression?

In the field of Image processing, the compression of images is an important step before we start the processing of larger images or videos. The compression of images is carried out by an encoder and output a compressed form of an image. In the processes of compression, the mathematical transforms play a vital role. A flow chart of the process of the compression of the image can be represented as:

Why Do We Need Image Compression?

Consider a black and white image that has a resolution of 1000*1000 and each pixel uses 8 bits to represent the intensity. So the total no of bits required = 1000*1000*8 = 80,00,000 bits per image. And **consider if it is a video with 30 frames per second of the above-mentioned type images then the total bits for a video of 3 secs is: 3*(30*(8, 000, 000))=720, 000, 000 bits**

As we see just to store a 3-sec video we need so many bits which is very huge. So, we need a way to have proper representation as well to store the information about the image in a minimum no of bits without losing the character of the image. Thus, image compression plays an important role.

Basic steps in image compression:

- Applying the image transform
- Quantization of the levels
- Encoding the sequences.

Run Length Encoding (RLE) image compression algorithm –

1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Initialize an empty list to store the compressed image.
4. Get the shape of the image.
5. Iterate over the rows of the image, for every row:
   - initialize a variable to store the current pixel value.
   - initialize a variable to store the current run length.
   - iterate over the columns of the image, for every column:
   - get the pixel value.
   - check if the pixel value is equal to the current pixel value and increment the current run length.
   - else if the pixel value is different, append the current run to the compressed image and update the current pixel value and reset the current run length.
6. Append the last run to the compressed image.
   - (This compressed image can be saved as a file).
7. Decompress the image to check if it appears like the original image. (The original grayscale and decompressed images should be identical if the compression and decompression were successful.)
8. Display the decompressed image using the cv2 library.

Discrete Cosine Transform (DCT) Image Compression algorithm –

1. Read and load the input RGB image.
2. Convert the image to grayscale using the cv2.cvtColor() function.
3. Create a function to compress a region of the image using the DCT image compression technique for a given region:
   • get the shape of the region
   • apply the DCT to the region
   • set the low frequency coefficients to zero
   • apply the inverse DCT to the region
   • return the region
4. create a copy of the image to store the compressed image.
5. iterate over the blocks in the image.
6. get the current block.
7. compress the block using the DCT image compression technique (calling the above defined function).
8. insert the compressed block into the compressed image.
9. Display the compressed image using the cv2 library.

Scalar quantization Image Compression algorithm –

1. Read and load the input RGB image.
2. Flatten the image into a 1D array.
3. Find the min and max pixel values: min_val, max_val.
4. Divide the range of pixel values into equally spaced intervals.
5. Find the mean value of each interval.
6. Quantize the pixel values by replacing them with the mean value of their interval.
7. Reshape the quantized image back into its original shape.
8. Display the compressed image using the cv2 library.

Conclusion: We have successfully implement a program for image compression using three compression techniques and compare the results.

Mini Project

Mini project 1 –

Implement visual surveillance applications and detect moving objects using object detection and tracking algorithm.

Mini project 2–

Implement any medical image processing application for freely available medical image dataset.

Mini project 3 –

Implement image segmentation to detect object in the background of image.