Lokmanya Tilak College of Engineering

Lab Manual

Subject - Digital Image Processing and Machine Vision **Sem-VI (R2019-C-Scheme)**

LOKMANYA TILAK COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION

LIST OF EXPERIMENTS

SUBJECT: DIPMV **CLASS:** TE **SEM:** VI

Sr.No.	Name of Experiment	CO's
1	To implement Point processing operation on gray scale image.	CO1
2	To implement low pass filter on given gray scale image.	CO2
3	To implement high pass filter on given gray scale image.	CO2
4	To perform histogram equalization on given image.	CO1
5	To analyze edge detection operators SOBEL and CANNY on given gray scale image.	CO4
6	To understand and implement Image resizing and Image cropping operation.	CO1
7	To perform morphological operations like EROSION, DILATION, OPENING, CLOSING on given gray scale image.	CO3
8	To perform adaptive thresholding on given gray scale image.	CO4
9	To perform classification using k-means algorithm.	CO6
10	Case Study: License plate recognition	CO6

Subject In charge

Dr. Rajeshree Shinde

Department of Electronics and Telecommunication

EXPERIMENT NO 1

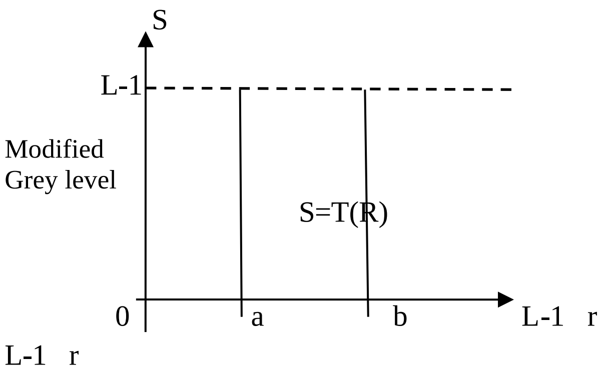
Aim: Point processing operation on gray level image.

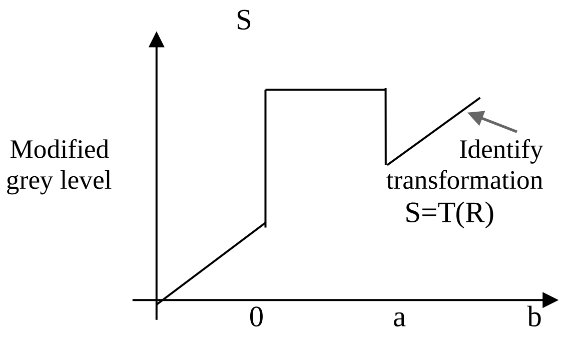
Problem statement: To implement gray level slicing, negative image and log transformation on gray scale image using PYTHON.

Description:

GREY LEVEL SLICING

1) It is used to highlight a specific range of grey levels in a image. For example mass of water in satellite image, enhancing he flows in a x-ray or a CT image.
2) There are two types of gray level slicing.





Original gray level

a)Slicing without background

S=250 IF 50 < r < 200
=0 otherwise

Original gray level

b) Slicing with background

S=250 IF 50 < r < 200
=0 otherwise

3) In grey level slicing without background, we have completely lost the background.

4) But in some applications, we not only need To enhance a band of grey levels, but also need to retain the background .this technique of retaining the background is called as grey level slicing with background.

EXAMPLE:

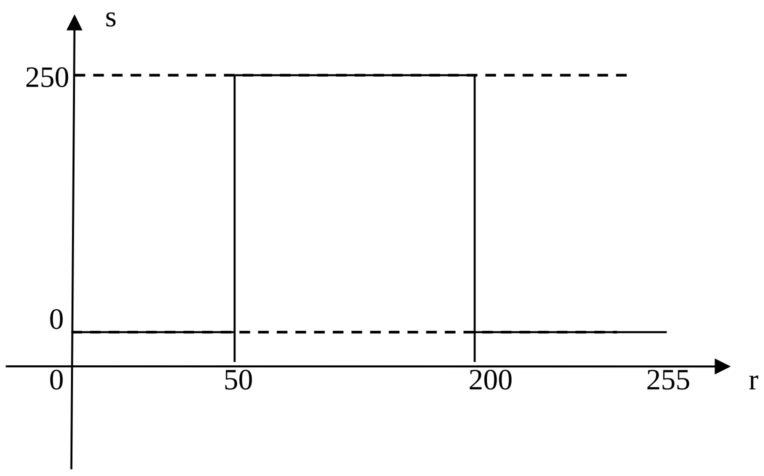
1) **Apply grey level slicing on given image by (a)&(b) technique.**

$$\begin{pmatrix} 0 & 10 & 60 & 80 \\ 200 & 210 & 135 & 125 \\ 160 & 255 & 20 & 40 \\ 100 & 120 & 140 & 244 \end{pmatrix}$$

Ans-

a) Grey level slicing without background we used following tarsformation for this method

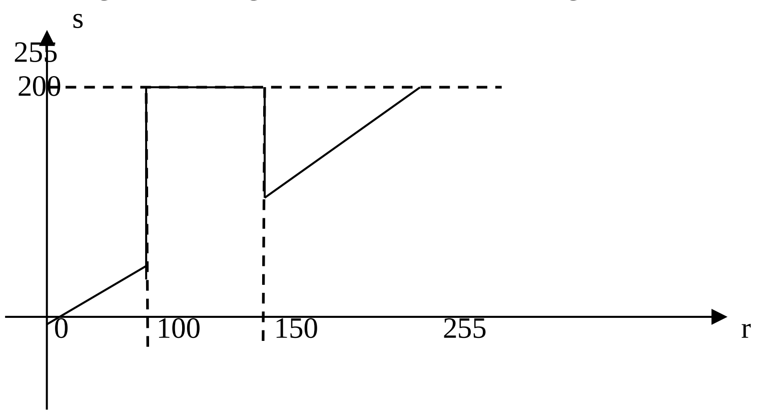
$$S = \begin{cases} 250 & \text{IF } 50 \le r < 200 \\ 0 & \text{otherwise} \end{cases}$$



Output image after grey level slicing without background will be

$$\begin{pmatrix} 0 & 0 & 250 & 250 \\ 250 & 0 & 250 & 250 \\ 250 & 0 & 0 & 0 \\ 250 & 250 & 250 & 250 \end{pmatrix}$$

b) Gray level slicing with background we used following transformation for this method.



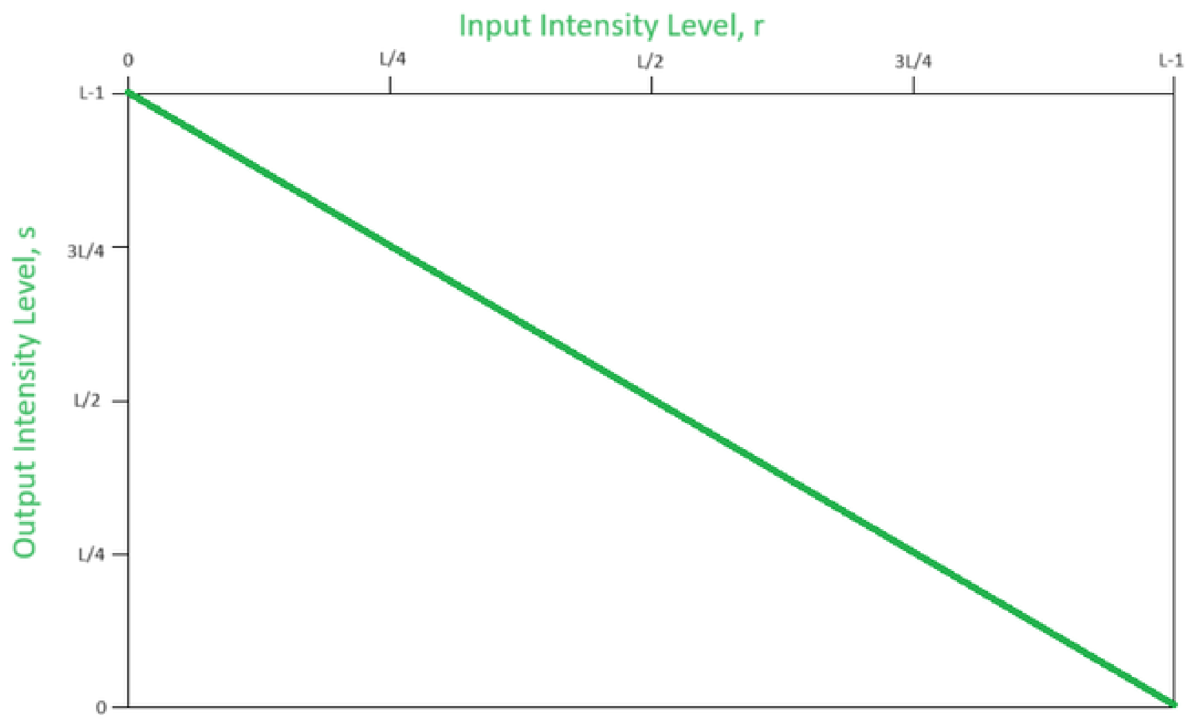
S= 200 ; for 100 < r < 150
=r otherwise

Output image after grey level slicing with background will be

$$\begin{pmatrix} 0 & 10 & 60 & 80 \\ 200 & 210 & 200 & 200 \\ 160 & 255 & 20 & 40 \\ 100 & 200 & 200 & 244 \end{pmatrix}$$

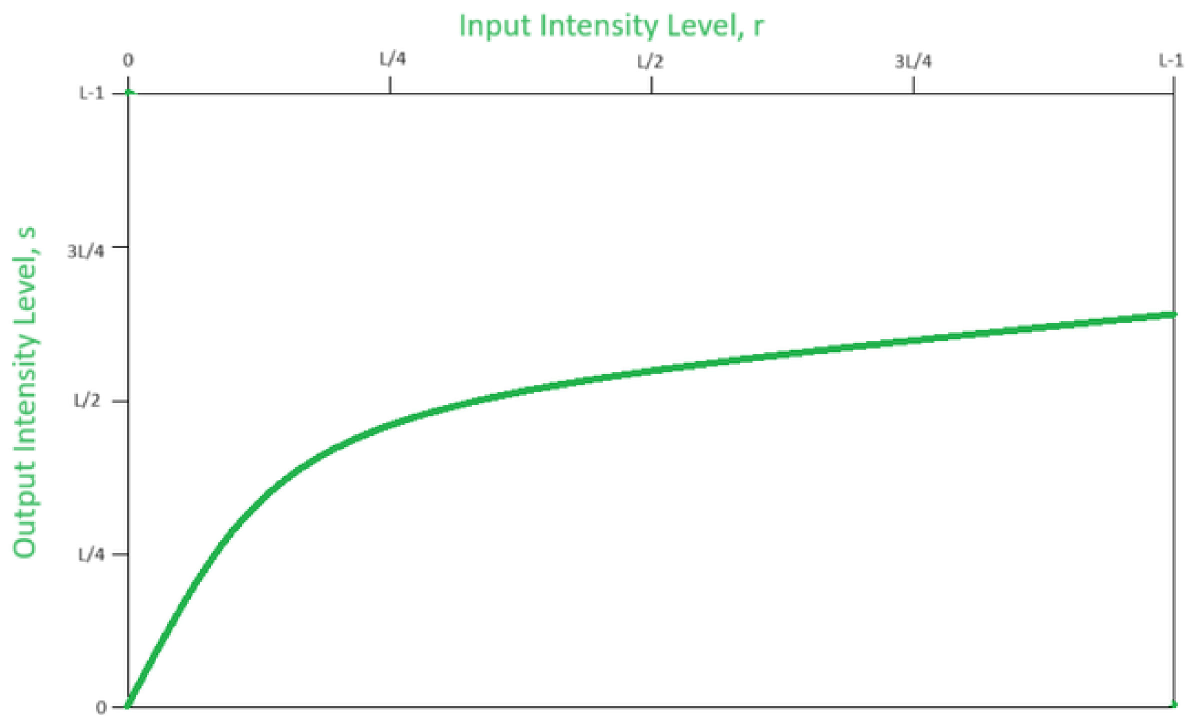
Negative Image

The negative of a image having intensity range [0, L - 1] can be found using the following transformation:
s = T(r) = L - r - 1



Negative Image
This technique is used to enhance white or light color detail in dark background, as dark color in light background is easily visible than light color in dark background. Here dark becomes light and light becomes dark.

Log Transformation
The log transformation is of the form:
$s = T(r) = c \log (1 + r)$
where, c is constant and $r \geq 0$.



Log Transformation

In log transformation, the higher range of intensity level is mapped to a lesser range of intensity level at the brighter side, whereas the lesser range of intensity level is mapped to higher range of intensity level at the darker side. The log transformation expands the dark pixels. Log transformation is applied when the intensity levels are very large, for example, 0 to 10^6. This is because transformation compresses the intensity levels of input level.

The inverse logarithmic transformation is called exponential transformation. It has the opposite behaviour that of logarithmic transformation. The higher range of intensity level is mapped to higher range of intensity level at the brighter side, whereas the lesser range of intensity level is mapped to a lesser range of intensity level at the darker side.

Program and output:

Conclusion:

EXPERIMENT NO 2

Aim: To implement low pass filter on given gray scale image using PYTHON.

Problem statement: To implement averaging filter and analyze the difference on similar gray scale image using PYTHON.

Description:

Low Pass Filtering:

Low pass filtering as the name suggests remove high frequency content from the image .it is used to remove noise present in the image. noise is basically a high frequency signal and low pass filtering eliminate the noise

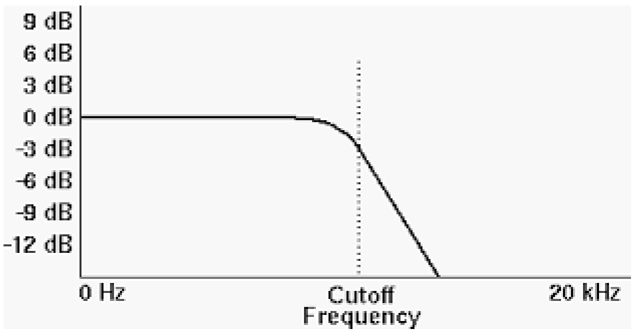


Fig. Low pass filter frequency response

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

3 × 3 Averaging mask

Apply a low pass mask on the following if X={2 3 4 3 4 5 6} & W={-1 0 1} perform median filtering.

Soln :

W = {-1 0 1} simply means that the size of the mask is 1×3 and term 0 indicate the position from where filtering starts.

Steps 1:

2 3 4 3 4 5 6

-1 0 1

Boundary value= 2

steps 2:

2 3 4 3 4 5 6

-1 0 1

Median {2 3 4} = 3

Steps 3:

2 3 4 3 4 5 6

-1 0 1

Median { 3 4 3} = 3

Steps 4:

2 3 4 3 4 5 6

-1 0 1

Median { 4 3 4} = 4

Steps 5:

2 3 4 3 4 5 6

-1 0 1

Median { 3 4 5} = 4

Steps 6:

2 3 4 3 4 5 6

$\boxed{-1\ 0\ 1}$

Median { 4 5 6} = 5

Steps 7:

$\boxed{2\ 3\ 4\ 3\ 4\ 5\ 6}$

$\boxed{-1\ 0\ 1}$

Boundary value =6

Hence the final answer is y= {2 3 3 4 4 5 6}.The principle function of median filtering is to force points with distinct intensities to be more like their neighbors

Program and output:

Conclusion:

EXPERIMENT NO 3

To implement high pass filter on given gray scale image using PYTHON.

Problem statement: To implement high pass filter and analyze the difference on similar gray scale image using PYTHON. **Aim:**

Description:

High Pass Filtering:

High pass filtering eliminates the low frequency region while retaining or enhancing high frequency region. An image which is high passed would have no background (as background are low frequency region) and would have enhanced edges hence high pass filter are used to sharpen blurred images. Once we have understood how the mask moves over the entire image for the averaging filter the same applies for the high pass filter only mask coefficient change

The important thing to note is that sum of the coefficient of the high pass has to be equal to zero .this is because when we place this mask over the low frequency region .The result should be zero

Let us take example of a pseudo image

10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100

HIGH PASS MASK

-1/9	-1/9	-1/9
-1/9	8/9	-1/9
-1/9	-1/9	-1/9

The background, which is low frequency get eliminated while the edges get enhanced

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-270	-270	-270	-270	-270	-270	-270	-270
-270	-270	-270	-270	-270	-270	-270	-270
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

-270 is a value that is lower than 0.that is ,it is supposed to be darker than the darkest value i.e 0 .we cannot take MOD [-270] because it will yield +270, which is definitely not darker than the zero value hence we write -270 as zero

The result using the high pass mask and negative values as zero is

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
30	30	30	30	30	30	30	30
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Program and output:

Conclusion:

EXPERIMENT NO 4

Aim: To perform histogram equalization on given image using PYTHON.

Problem statement: To implement histogram equalization on gray scale image using PYTHON.

Description:

Histogram Equalization:

In Histogram equalization, we convert low contrast image into high contrast image by increasing grey levels and decreasing or increasing the amplitude of histogram.

Histogram of this image: The histogram of this image has been shown below.

Now we will perform histogram equalization on the image which is given bellow.

7	6	7	7	5	5	5	5	5
3	0	6	2	7	4	2	5	5
6	6	6	6	6	4	7	5	7
1	6	7	1	7	4	4	4	5
6	3	6	2	5	7	2	4	7
7	3	7	3	5	3	0	4	3
3	7	7	1	5	7	2	4	4
6	6	3	2	6	6	1	7	4

PMF:

First we have to calculate the PMF (probability mass function) of all the pixels in this image. If you do not know how to calculate PMF, please visit our tutorial of PMF calculation.

CDF:

Our next step involves calculation of CDF cumulative distributive function. Again if you do not know how to calculate CDF , please visit our tutorial of CDF calculation.

Calculate CDF according to gray levels

Let’s for instance consider this, that the CDF calculated in the second step looks like this.

Gray level value	CDF
0	0.11
1	0.22
2	0.55
3	0.66
4	0.77
5	0.88
6	0.99
7	1

Then in this step you will multiply the CDF value with Graylevels - minus 1 .

Considering we have an 3 D image. Then number of levels we have are 8. And 1 subtracts 8 is 7. So we multiply CDF by 7. Here what we got after multiplying.

Gray Level Value	**CDF**	**CDF *** Levels−1
0	0.11	0
1	0.22	1
2	0.55	3
3	0.66	4
4	0.77	5
5	0.88	6
6	0.99	6
7	1	7

Now we have is the last step , in which we have to map the new gray level values into number of pixels.

Lets assume our old gray levels values has these number of pixels.

Gray level value	Frequency
0	2
1	4
2	6
3	8
4	10
5	12

6	14
7	16

Now if we map our new values to , then this is what we got.

Gray level value	New gray level value	Frequency
0	0	2
1	1	4
2	3	6
3	4	8
4	5	10
5	6	12
6	6	14
7	7	16

Now map these new values you are onto histogram , and you are done.Lets apply this technique to our original image. After applying we got the following image and its following histogram.

Equalized Output Image

7	6	7	7	6	6	6	6	6
4	0	6	3	7	5	3	6	6
6	6	6	6	6	5	7	6	7
1	6	7	1	7	5	5	5	6
6	4	6	3	6	7	3	5	7

7	4	7	4	6	4	0	5	4
4	7	7	1	6	7	3	5	5
6	6	4	3	6	6	1	7	5

Program and output:

Conclusion:

EXPERIMENT NO 5

Aim: To analyze edge detection operators SOBEL and CANNY on given gray scale image using PYTHON.

Problem statement: To implement sobel edge detection and canny edge detection on gray scale image using PYTHON.

Description:

THEORY:

Edges are significant local intensity changes in the image and are important features to analyse an image. They are important clues to separate regions within an object or to identify changes in illumination or colour. They are an important feature in the early vision stages in the human eye.
There are many different geometric and optical physical properties in the scene that can originate an edge in an image.

Edge detection is essential the operation of detecting intensity variations. 1st and 2nd derivative operators give the information of this intensity changes. For instance, the first derivative of a step edge has a maximum located of the position of the edge.

The gradient of the image intensity is the vector:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^t = \left[G_x, G_y\right]^t$$

The magnitude and direction of the gradient are:

$$G = \sqrt{G_x^2 + G_y^2} \qquad \theta = \tan^{-1}\frac{G_y}{G_x}$$

Is common practice to approximate by

$$G \approx |G_x| + |G_y| \quad or \quad G \approx \max(|G_x| + |G_y|$$

Filtering: gradient operators are very sensitive to noise. It is important to improve the signal to
noise ratio, by filtering previously the image. However, there is a trade-off between edge strength and noise reduction.

Enhancement: To emphasize pixels with a significant change in local intensity, using a gradient operator.
Detection: Label the edge points. Thresholding the gradient magnitude is a common technique.

Localization: The location of the edge can be estimated with sub pixel resolution.

Edge operators are:

1. Robert operator

The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point.

The operator consists of a pair of 2×2 convolution kernels as shown in Figure. One kernel is simply the other rotated by 90°. This is very similar to the Sobel operator.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these *Gx* and *Gy*). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

although typically, an approximate magnitude is computed using:

$$|G| = |Gx| + |Gy|$$

which is much faster to compute.

The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by:

$$\theta = \arctan(Gy/Gx) - 3\pi/4$$

2. Sobel operator

The operator consists of a pair of 3×3 convolution kernels as shown in Figure 1. One kernel is simply the other rotated by 90°.

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

Gx

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

Gy

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these *Gx* and *Gy*). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Typically, an approximate magnitude is computed using:

$$|G| = |Gx| + |Gy|$$

which is much faster to compute.

The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by:

$$\theta = \arctan(Gy/Gx)$$

3. Prewitt operator

Prewitt operator is similar to the Sobel operator and is used for detecting vertical and horizontal edges in images.

$$G_x \approx \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y \approx \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

4. Canny Edge Detector

The process of Canny edge detection algorithm can be broken down to five different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply gradient magnitude thresholding or lower bound cut-off suppression to get rid of spurious response to edge detection

4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Program and output:

Conclusion:

EXPERIMENT NO 6

Aim: To understand and implement Image resizing and Image cropping operation using PYTHON.

Problem statement: To implement resizing and cropping on gray scale image using PYTHON.

Description:

Resizing and Cropping Images to Standard Dimensions

Resizing and cropping your images is an important first step in image preprocessing.

Images come in all shapes and sizes, but machine learning algorithms typically require a standard size. You'll want to resize and crop your images to square dimensions, often 224x224 or 256x256 pixels.

In Python, you can use the OpenCY or Pillow library for resizing and cropping. With OpenCV, use the resize() function. For example:

import cv2

img = cv2.imread ('original.jpg')

resized = OV2.resize(img, (224, 224))

This will resize the image to 224x224 pixels.

To crop an image to a square, you can calculate the center square crop size and use crop() in OpenCV with the center coordinates. For example:

height, width = img.shape[2]

size = min (height, width)

x = (width - size) /| 2

y = (height - size) /| 2

cropped = ingly: y+size, xix+size)

With Pillow, you can use the Image. open () and resize() functions. For example:

```
from PIL import Image
img = Image.open('original.jpg')
resized = img.resize((224, 224))
```

To crop the image, use img. crop(). For example:

```
width, height = img.size
size = min (width, height)
left = (width - size) / 2
top = (height - size) / 2
right = (width + size) / 2
bottom = (height + size) / 2
cropped = img.crop((left, top, right, bottom))
```

Program and output:

Conclusion:

EXPERIMENT NO 7

Aim: To perform morphological operations like EROSION, DILATION, OPENING, CLOSING on given gray scale image using PYTHON.

Problem statement: To implement morphological operations like EROSION, DILATION, OPENING, CLOSING on given gray scale image using PYTHON.

Description:

1. Erosion

Assume that f(x, y) is a grey-scale image and b(x, y) is a structuring element and both functions are discrete. Similarly to binary morphology, the structuring elements are used to examine a given image for specific properties. The erosion of f by a flat structuring element b at any location (x, y) is defined as the minimum value of the image in the region coincident with b when the origin of b is at (x, y). Therefore, the erosion at (x, y) of an image f by a structuring element b is given by:

$$[f \ominus b](x, y) = \min_{(s,t)\in b} \{f(x+s, y+t)\}$$

Erosion is a fundamental morphological operation that reduces the size of objects in a binary image. It works by removing pixels from the boundaries of objects.

- Purpose: To remove small noise, detach connected objects, and erode boundaries.
- How it Works: The structuring element slides over the image, and for each position, if all the pixels under the structuring element match the foreground, the pixel in the output image is set to the foreground. Otherwise, it is set to the background.

2. Dilation

The dilation of f by a flat structuring element b at any location (x, y) is defined as the maximum value of the image in the window outlined by b ˆ when the origin of b ˆ is

$$[f \oplus b](x, y) = \max_{(s,t)\in b} \{f(x-s, y-t)\}$$

where we used that

$$\hat{b} = b(-x, -y)$$

Dilation is the opposite of erosion and is used to increase the size of objects in an image.

- Purpose: To join adjacent objects, fill small holes, and enhance features.
- How it Works: The structuring element slides over the image, and for each position, if any pixel under the structuring element matches the foreground, the pixel in the output image is set to the foreground.

3. Opening

$$A \circ B = (A \ominus B) \oplus B$$

Opening is a compound operation that involves erosion followed by dilation.

- Purpose: To remove small objects or noise from the image while preserving the shape and size of larger objects.
- How it Works: First, the image undergoes erosion, which removes small objects and noise. Then, dilation is applied to restore the size of the remaining objects to their original dimensions.

4. Closing

$$A \cdot B = (A \oplus B) \ominus B$$

Closing is another compound operation that consists of dilation followed by erosion.

- Purpose: To fill small holes and gaps in objects while preserving their overall shape.
- How it Works: First, dilation is applied to the image, filling small holes and gaps. Then, erosion is applied to restore the original size of the objects.

Program and output:

Conclusion:

EXPERIMENT NO 8

Aim: To perform adaptive thresholding on given gray scale image using PYTHON.

Problem statement: To implement Otsu algorithm for adaptive thresholding on given gray scale image using PYTHON.

Description:

In many vision applications, it is useful to be able to separate out the regions of the image corresponding to objects in which we are interested, from the regions of the image that correspond to background. Thresholding often provides an easy and convenient way to perform this segmentation on the basis of the different intensities or colours in the foreground and background regions of an image. In addition, it is often useful to be able to see what areas of an image consist of pixels whose values lie within a specified range, or *band* of intensities (or colours). Thresholding can be used for this as well.

The input to a thresholding operation is typically a gray scale or colour image. In the simplest implementation, the output is a binary image representing the segmentation. Black pixels correspond to background and white pixels correspond to foreground (or *vice versa*). In simple implementations, the segmentation is determined by a single parameter known as the *intensity threshold*. In a single pass, each pixel in the image is compared with this threshold. If the pixel's intensity is higher than the threshold, the pixel is set to, say, white in the output. If it is less than the threshold, it is set to black.

In more sophisticated implementations, multiple thresholds can be specified, so that a *band* of intensity values can be set to white while everything else is set to black. For colour or multi-spectral images, it may be possible to set different thresholds for each colour channel, and so select just those pixels within a specified cuboid in RGB space. Another common variant is to set to black all those pixels corresponding to background, but leave foreground pixels at their original colour/intensity (as opposed to forcing them to white), so that that information is not lost.

Not all images can be neatly segmented into foreground and background using simple thresholding. Whether or not an image can be correctly segmented this way can be determined by looking at an intensity histogram of the image. We will consider just a grayscale histogram here, but the extension to colour is trivial.

If it is possible to separate out the foreground of an image on the basis of pixel intensity, then the intensity of pixels within foreground objects must be distinctly different from the intensity of pixels within the background. In this case, we expect to see a distinct peak in the histogram corresponding to foreground objects such that thresholds can be chosen to isolate this peak accordingly. If such a peak does not exist, then it is unlikely that simple thresholding will produce a good segmentation. In this case, adaptive thresholding may be a better answer.

Figure 1 shows some typical histograms along with suitable choices of threshold.

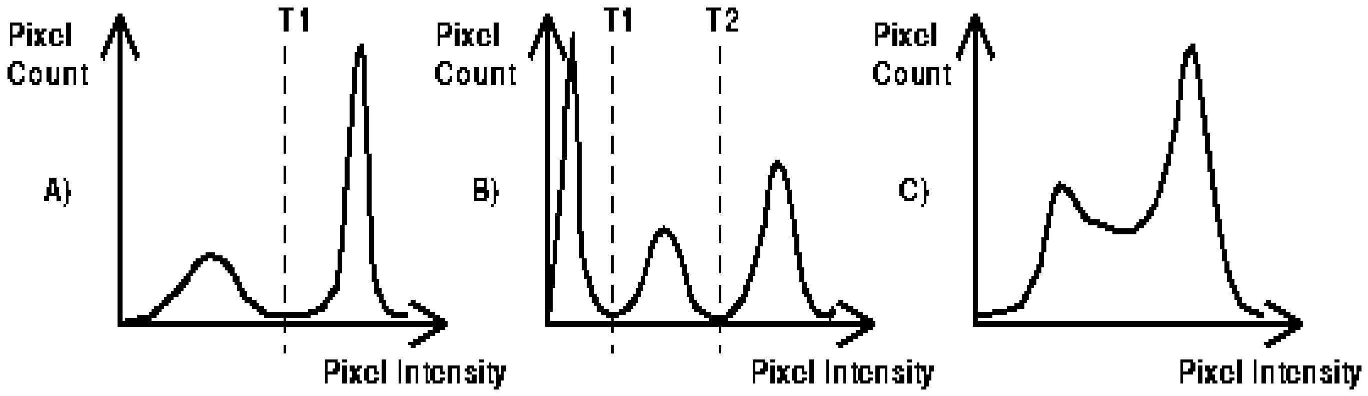


Figure 1 **A)** **B)** **C)**

Figure 1 A) shows a classic bi-modal intensity distribution. This image can be successfully segmented using a single threshold *T1*. **B)** is slightly more complicated. Here we suppose the central peak represents the objects we are interested in and so threshold segmentation requires two thresholds: *T1* and *T2*. In **C)**, the two peaks of a bi-modal distribution have run together and so it is almost certainly not possible to successfully segment this image using a single global threshold

Program and output:

Conclusion:

EXPERIMENT NO 9

Aim: To perform classification using k-means algorithm.

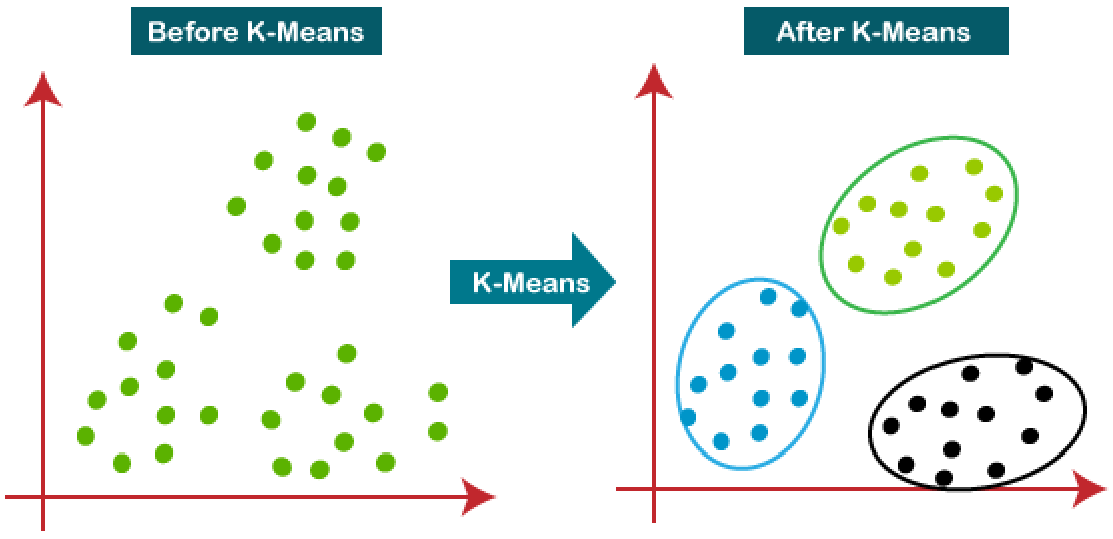
Problem statement: To implement classification using k-means algorithm given data set using PYTHON.

Description:

K-means is an unsupervised learning method for clustering data points. The algorithm iteratively divides data points into K clusters by minimizing the variance in each cluster.
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.



The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Program and output:

Conclusion:

EXPERIMENT NO 10

Aim: Case Study: License plate recognition

Problem statement: To implement License plate recognition from a given image using PYTHON.

Description:

Open cmd and install OpenCV and imutils using the following commands-

```
pip install opencv-contrib-python
```

OpenCV will be used here for various pre-processing techniques and for displaying the image.

```
pip install imutils
```

imutils library contains a series of basic pre-processing functions and here it will be used for resizing the image.

Now for installing pytesseract, head over to https://github.com/UB-Mannheim/tesseract/wiki and download and install it.

The tesseract library is an optical character recognition (OCR) tool for Python. That is, it can recognize and read the text embedded from any image. So we'll use it for identifying the characters inside the number plate.

Program and output:

Conclusion: