Laboratory Manual for

EC725–Computer Vision & Machine Learning

B. Tech.

SEM. VII (EC)

Department of Electronics & Communication
Faculty of Technology
Dharmsinh Desai University
Nadiad

TABLE OF CONTENTS

P A R T I

LAB MANUAL

EXPERIMENT – 1

BASIC FUNCTIONS OF IMAGE PROCESSING WITH OpenCV

OBJECTIVE: To understand and implement basic OpenCV functions for Image Processing.

THEORY:

OpenCV, arguably the most widely used computer vision library, includes hundreds of ready-to-use imaging and vision functions and is used in both academia and industry. As cameras get cheaper and imaging features grow in demand, the range of applications using OpenCV increases significantly, both for desktop and mobile platforms. Initially developed by Intel, OpenCV (Open Source Computer Vision) is a free cross-platform library for real-time image processing.

The applications for OpenCV cover areas such as segmentation and recognition, 2D and3D feature toolkits, object identification, facial recognition, motion tracking, gesture recognition, image stitching, high dynamic range (HDR) imaging, augmented reality, and so on. Moreover, to support some of the previous application areas, a module with statistical machine learning functions is included.

OpenCV-Python:

Python is a general purpose programming language started by Guido van Rossum, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

And the support of Numpy makes the task easier. Numpy is a highly optimized library for numerical operations. It gives MATLAB/Scilab-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this. So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems.

SAMPLE PROGRAM:

(1) Read, Write and Displaying an Images

```
import cv2
imgpath = "/home/ec/Desktop/gray.jpg"
img = cv2.imread(imgpath, 0)# ‘0’ is for gray scale image

outpath = "/home/ec/Desktop/gray_out1.jpg"

cv2.imshow('Lena', img)
cv2.imwrite(outpath, img)
cv2.waitKey(0) #Wait until key strike from keyboard
cv2.destroyAllWindow()#Close all windows
```

RESULT:

Gray Scale Image

(2) Display Image Properties

```
import cv2
imgpath = "/home/ec/Desktop/gray.jpg"
img1 = cv2.imread(imgpath, 0)

print(type(img1))
print('Image Data Type: ', img1.dtype)
print('Row Column: ', img1.shape)
print('Dimension : ', img1.ndim)
print('Image Size: ', img1.size)

(nr,nc) = img1.shape  # to access row and column of image

print('No. of Row: ',nr)
print('No. of Column: ', nc)

cv2.imshow('Lena', img1)
cv2.waitKey(0) #Wait until key strike from keyboard
cv2.destroyAllWindow()()#Close all windows
```

RESULT:

```
Image Data Type: uint8
Row Column: 512, 512
Dimension: 2
Image Size: 2097987
```

(3) Color Image to Gray Scale Conversion

```
import cv2

imgpath1 =  "/home/ec/Desktop/index.jpg"

img1 = cv2.imread(imgpath1, 1)
cv2.imshow('Color Image', img1)
cv2.waitKey(0)
```

```
cv2.destroyWindow('Color Image')

gray1 = cv2.cvtColor(img1, cv2.COLOR_RGB2GRAY)
cv2.imshow('Gray Image', gray1)
cv2.waitKey(0) #Wait until key strike from keyboard
cv2.destroyWindow('Gray Image')#Close window
```

RESULT:

(4) Display Multiple Images

```
import cv2
import matplotlib.pyplot as plt
imgpath1 =  "/home/ec/Desktop/index1.jpg"
imgpath2 =  "/home/ec/Desktop/index2.jpg"
imgpath3 =  "/home/ec/Desktop/index3.jpg"
img1 = cv2.imread(imgpath1, 0)
img2 = cv2.imread(imgpath1, 0)
img3 = cv2.imread(imgpath1, 0)

titles = ['Gray Image1', 'Gray Image2', 'Gray Image3']
images = [img1, img2, img3]

for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])

plt.show()
```

RESULT:

(5) Pixel Value Accessing and Editing

```
import cv2
imgpath = "/home/ec/Desktop/gray.jpg"
img = cv2.imread(imgpath, 0)# ‘0’ is for gray scale image
# accessing RED value
X=img.item(10,10,2)
Print(‘x:’ x)
# modifying RED value
img.itemset((10,10,2),100)
y=img.item(10,10,2)
Print(‘y:’ y)
```

RESULT:

```
X: 87
Y: 100
```

(6) Image Resize

```
import cv2
import matplotlib.pyplot as plt
imgpath1 =  "/home/ec/Desktop/index1.jpg"

img1 = cv2.imread(imgpath1, 1)

cv2.imshow('Original Image', img1)
cv2.waitKey(0)
(row1,column1) = img1.shape  # to access row and column of image
row2=150
column2=150
img2=cv2.resize(img1,(row2,column2), cv2.INTER_AREA)#bilinear interpolation,
INTER_NEAREST, INTER_CUBIC
cv2.imshow('Resized Image', img2)
cv2.waitKey(0)

print(‘Original Image Size:’,img1.shape)
```

print('Resize Image Size:',img2.shape)

RESULT:

CONCLUSION:

EXPERIMENT – 2

FUNDAMENTAL IMAGE PROCESSING

OBJECTIVE:

I. Find out average intensity of grayscale image.
II. Apply negative intensity transformation on any grayscale image.

SAMPLE PROGRAM:

I. Find out average intensity of grayscale image

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

imgpath = "/home/IP_Python 2018-19/ippythonprograms/gray.jpg"
img1 = cv2.imread(imgpath, 0)

(nr,nc) = img1.shape
print('No. of Row: ',nr)
print('No. of Column: ', nc)

cv2.imshow('Lena', img1)
cv2.waitKey(0)
cv2.destroyWindow('Lena')

temp=0;

for i in range(nr):
    for j in range(nc):
        temp=temp+img1[i][j];

AvgIntensity=temp/(nr*nc);
print('Average Intensity:',AvgIntensity);
```

RESULT:

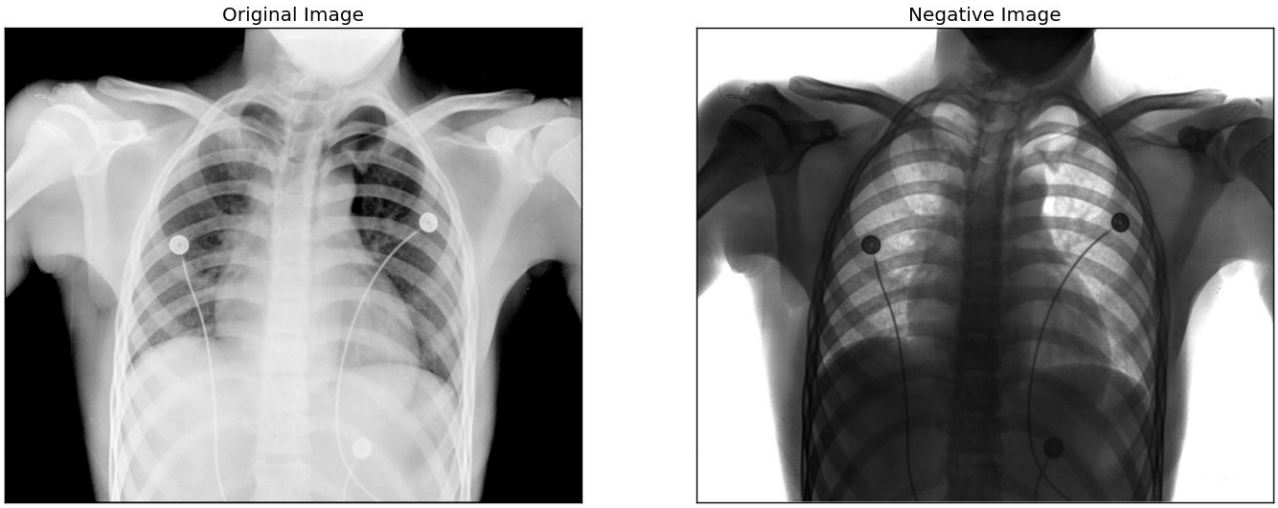
```
Average Intensity: 90.786542
```

II. Apply negative intensity transformation on grayscale image

```
import cv2
import matplotlib.pyplot as plt

imgpath1 =  "/home/IP_Python 2018-19/ippythonprograms/image2.tif"
img1 = cv2.imread(imgpath1, 0)
```

```
cv2.imshow('Original Image', img1)
cv2.waitKey(0)

img2 = abs(255-img1)
cv2.imshow('Negative Image', img2)
cv2.waitKey(0)

cv2.destroyAllWindows()

titles = ['Original Image', 'Negative Image']
images = [img1, img2]

for k in range(2):
    plt.subplot(1, 2, k+1)
    plt.imshow(images[k],cmap='gray')
    plt.title(titles[k])
    plt.xticks([])
    plt.yticks([])

plt.show()
```

RESULT:



ASSIGNMENT:

(I) Read images having following characteristics and calculate average intensity of the same. (I) Darker Image (II) Brighter Image. Justify your answer.
(II) W.A.P to perform Image Flipping operation on digital image.
(III) W.A.P to perform to apply log transform on grayscale image.
(IV) W.A.P to perform gamma transform on grayscale image.

CONCLUSION:

EXPERIMENT – 3

CONTRAST ENHANCEMENT ALGORITHMS

OBJECTIVE:

I. To observe histogram of grayscale image
II. To apply contrast stretching intensity transformation on grayscale image
III. To observe bit-plane slicing of any grayscale image

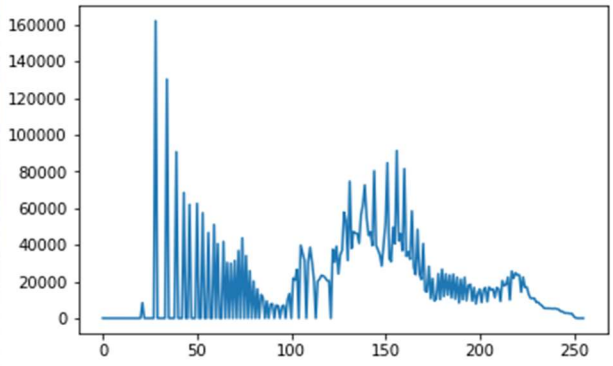
SAMPLE PROGRAM:

I. To observe histogram of grayscale image

```
import cv2
from matplotlib import pyplot as plt
img = cv2.imread('first.png',0)
# find frequency of pixels in range 0-255
histr = cv2.calcHist([img],[0],None,[256],[0,256])
plt.plot(histr)
plt.show()
```

RESULTS:

Input Image



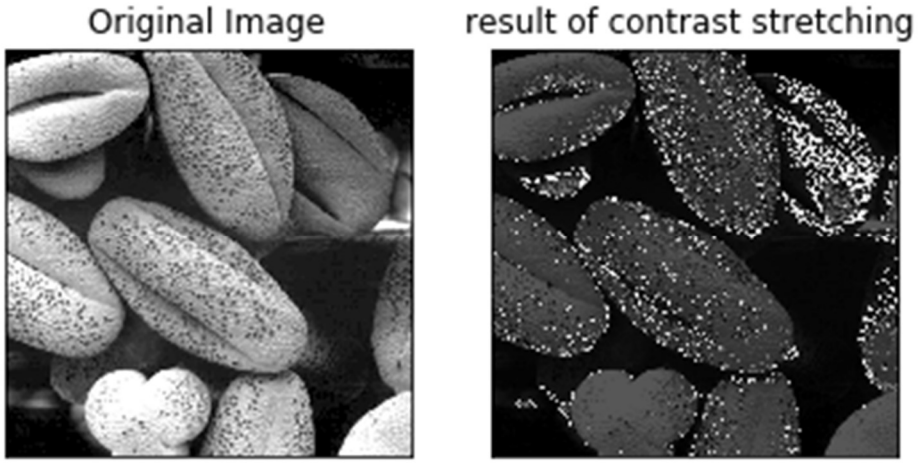
Histogram

II. To apply contrast stretching intensity transformation on grayscale image

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('Fig0320(3)(third_from_top).tif',0)
final = np.zeros((img.shape[0],img.shape[1]),dtype = 'float16')
r1=95
r2=135
s1=25
s2=215
m1=(s1)/r1;
m2=(s2-s1)/(r2-r1);
```

```
m3=(256-s2)/(256-r2);
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if img[i][j]<=r1:
            final[i,j] = m1*img[i][j]
        elif img[i][j]<=r2:
            final[i,j] = m2*img[i][j]
        else:
            final[i,j] = m3*img[i][j]
im1 = np.array(final,dtype=np.uint8)


titles = ['Original Image', 'result of contrast stretching ']
images = [img,im1]
no=2
for k in range(no):
        plt.subplot(1, no, k+1)
        plt.imshow(images[k],cmap='gray')
        plt.title(titles[k])
        plt.xticks([])
        plt.yticks([])

plt.show()
```

RESULT:

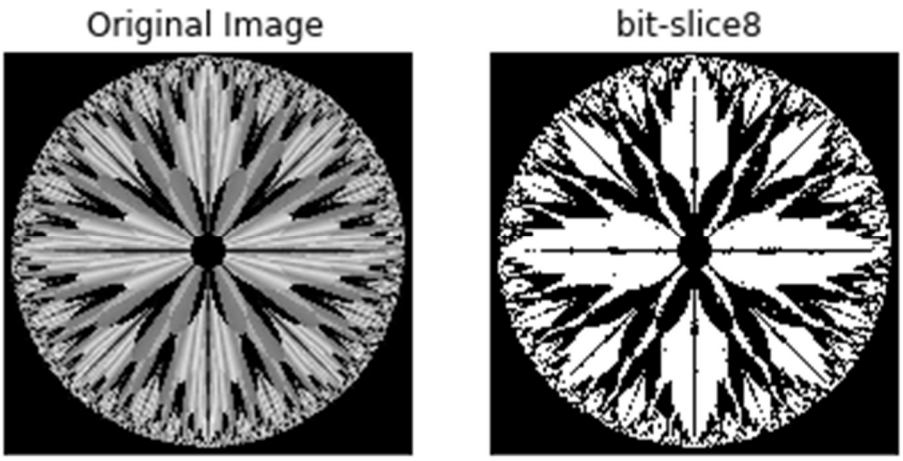


III. To observe bit-plane slicing of grayscale image

```
import cv2
import numpy as np
from matplotlib import pyplot as plt


img = cv2.imread('fractal.jpg',0)
(nr,nc) = img.shape
lst = []
```

```
for i in range(nr):
        for j in range(nc):
                lst.append(np.binary_repr(img[i][j] ,width=8)) # width = no. of bits
eight_bit_img=(np.array([int(i[0]) for i in lst],dtype = np.uint8)*128).reshape(nr,nc)
seven_bit_img=(np.array([int(i[1]) for i in lst],dtype = np.uint8)*64).reshape(nr,nc)
six_bit_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8)*32).reshape(nr,nc)
five_bit_img= (np.array([int(i[3]) for i in lst],dtype = np.uint8)*16).reshape(nr,nc)
four_bit_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8)*8).reshape(nr,nc)
three_bit_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8)*4).reshape(nr,nc)
two_bit_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8)*2).reshape(nr,nc)
one_bit_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8)*1).reshape(nr,nc)
titles = ['Original Image', 'bit-slice8 ']
images = [img,eight_bit_img]

no=2
for k in range(no):
      plt.subplot(1, no, k+1)
      plt.imshow(images[k],cmap='gray')
      plt.title(titles[k])
      plt.xticks([])
      plt.yticks([])

plt.show()
```

RESULT:



ASSIGNMENT:

(I) W.A.P. to perform histogram equalization on digital image.

CONCLUSION:

EXPERIMENT – 4

EFFECT OF SAMPLING & QUANTIZATION ON AN IMAGE

OBJECTIVE:

I. To observe checker-board effect due to different sampling rate
II To observe false contouring effect due to number of gray level

SAMPLE PROGRAM:

I. To observe checker-board effect due to different sampling rate

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('Fig0809(a).tif',0)
a=[]
lst=[]
samplerate=2
x=int(img.shape[0]/samplerate)
y=int(img.shape[1]/samplerate)
for i in range(0,img.shape[0],samplerate):
    for j in range(0,img.shape[1],samplerate):
        a.append(img[i][j])

b=np.reshape(a,(x,y))
titles = ['Original Image', 'Sampling rate-2 ']
images = [img,b]

no=2
for k in range(no):
        plt.subplot(1, no, k+1)
        plt.imshow(images[k],cmap='gray')
        plt.title(titles[k])
        plt.xticks([])
        plt.yticks([])

plt.show()
```

RESULT:

II.To observe false contouring effect due to number of gray level

```
import cv2
import numpy as np
img = cv2.imread('Fig0809(a).tif',0)

# Create zeros array to store the stretched image
final1 = np.zeros((img.shape[0],img.shape[1]),dtype = 'uint8')
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        if (img[i][j]>=0 and img[i][j]<=25) :
            final1[i,j] = 13
        elif(img[i][j]>25 and img[i][j]<=50) :
            final1[i,j] = 38
        elif(img[i][j]>50 and img[i][j]<=75) :
            final1[i,j] = 63
        elif(img[i][j]>75 and img[i][j]<=100) :
            final1[i,j] = 88
        elif(img[i][j]>100 and img[i][j]<=125) :
            final1[i,j] = 113
        elif(img[i][j]>125 and img[i][j]<=150) :
            final1[i,j] = 138
        elif(img[i][j]>150 and img[i][j]<=175) :
            final1[i,j] = 163
        elif(img[i][j]>175 and img[i][j]<=200) :
            final1[i,j] = 188
        elif(img[i][j]>200 and img[i][j]<=225) :
            final1[i,j] = 213
        elif(img[i][j]>225 and img[i][j]<=255) :
            final1[i,j] = 238

titles = ['Original Image', 'False Contouring With 10 Intensity Level ']
images = [img,final1]
```

```
no=2
for k in range(no):
        plt.subplot(1, no, k+1)
        plt.imshow(images[k],cmap='gray')
        plt.title(titles[k])
        plt.xticks([])
        plt.yticks([])

plt.show()
```

RESULT:

ASSIGNMENT:

(I) W.A.P. to observe False contouring effect due to no. of gray level. Keep only 4 gray levels for observation.
(II) W.A.P. to observe checkerboard effect for different sampling rates

CONCLUSION:

EXPERIMENT – 5

SPATIAL FILTERS (SMOOTHING & SHARPNING)

OBJECTIVES:

I. To observe effect of various order of averaging filter (Linear Filter) on grayscale image.
II. W.A.P. to perform median filtering operation on a given noisy image.
II. Comparison performance of median and averaging filters to remove salt & pepper noise.
III. To observe effect of various type of filtering mask on grayscale image.

THEORY:

Filtering is a technique for modifying or enhancing an image. Spatial domain operation or filtering (the processed value for the current pixel processed value for the current pixel depends on both itself and surrounding pixels). Hence, filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is some set of pixels, defined by their locations relative to that pixel. Filters are used for noise removal also.

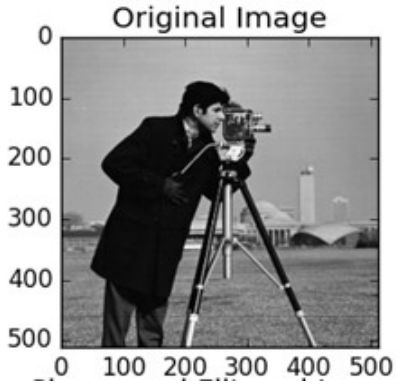
Spatial filtering is of two types: (1) linear filtering in which response is decided by a mask is convolved with respective image pixel values. E.g. smoothing filters, sharpening filters, (2) Non-linear filtering where response is decided by arranging pixels values of neighborhood in order and selecting pixel as per requirement. E.g. median filters, min or max filters.

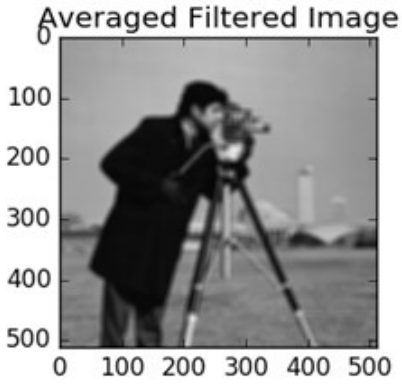
SAMPLE PROGRAM:

I. Observing the effect of smoothing filters.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

path = "/home/shital/oc/Python-OpenCV3-master/Dataset/"
imgpath =  path + "cameraman.tif"
img = cv2.imread(imgpath, 1)
#    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
k1 = np.array(np.ones((11, 11), np.float32))/121 #average filter
print(k1)                                        #printing mask
output = cv2.filter2D(img, -1, k1)               #here change mask variable

plt.subplot(1, 2, 1)
plt.imshow(img)
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(output)
plt.title('Filtered Image')
plt.show()
```

RESULTS:





MODIFICATION:

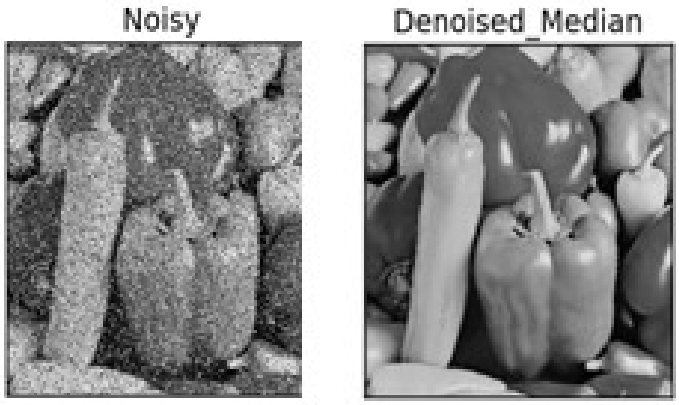
1. Compare the sharpening filter results with mask [-1, -1, -1], [-1, 9, -1], [-1, -1, -1]
2. Vary the mask size of average filter and see effect on an image.

II. Median Filters.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import random

path = "/home/shital/oc/Python-OpenCV3-master/Dataset/"
imgpath =  path + "4.2.07.tiff"
img = cv2.imread(imgpath, 1)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

noisy = np.zeros(img.shape, np.uint8)

p = 0.2
for i in range(img.shape[0]):
  for j in range(img.shape[1]):
        r = random.random()
        if r < p/2:
                noisy[i][j] = 0
        elif r < p:
                noisy[i][j] = 255
        else:
                noisy[i][j] = img[i][j]

denoised_median = cv2.medianBlur(noisy, 5)
#For Average Filtering
k1 = np.array(np.ones((11, 11), np.float32))/121        #average filter
print(k1)                                               #printing mask

Averaged = cv2.filter2D(noisy, -1, k1)                  #here change mask variable
```

```
output = [img, noisy, denoised_median]
titles = ['Original', 'Noisy', 'Denoised_Median']

for i in range(2):
        plt.subplot(1, 2, i+1)
        plt.imshow(output[i],'gray')
        plt.title(titles[i])
        plt.xticks([])
        plt.yticks([])
plt.show()
```

RESULTS:

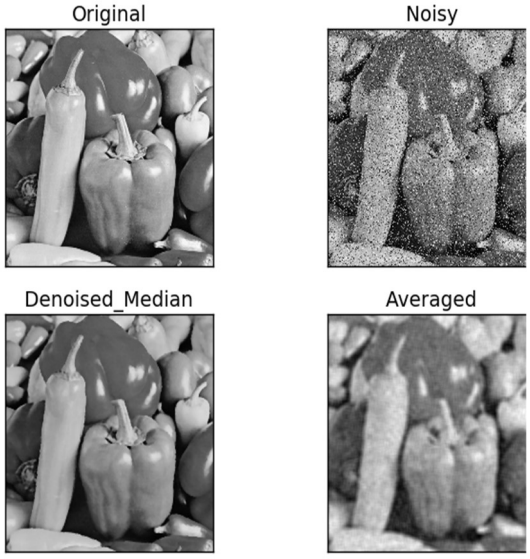


III. Comparison of averaging and median filters.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import random

path = "/home/shital/oc/Python-OpenCV3-master/Dataset/"
imgpath =  path + "4.2.07.tiff"
img = cv2.imread(imgpath, 1)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

noisy = np.zeros(img.shape, np.uint8)
p = 0.2
for i in range(img.shape[0]):
  for j in range(img.shape[1]):
        r = random.random()
  if r < p/2:
        noisy[i][j] = 0
  elif r < p:
        noisy[i][j] = 255
  else:
        noisy[i][j] = img[i][j]
```

```
denoised_median = cv2.medianBlur(noisy, 5)
#For Average Filtering
k1 = np.array(np.ones((11, 11), np.float32))/121        #average filter
print(k1)                                               #printing mask

Averaged = cv2.filter2D(noisy, -1, k1)                  #here change mask variable

output = [img, noisy, denoised_median, Averaged]
titles = ['Original', 'Noisy', 'Denoised_Median' , 'Averaged']

for i in range(4):
   plt.subplot(2, 2, i+1)
   plt.imshow(output[i],'gray')
   plt.title(titles[i])
   plt.xticks([])
   plt.yticks([])
plt.show()
```

RESULTS:



MODIFICATIONS:

I. Compare median and average filter with size of mask.
II. Vary amount of noise and compare output of both filters.
III. To observe the effect of laplacian sharpening filter on grayscale image.
IV. Apply sobel and prewait mask on digital image to detect edges.

CONCLUSION :

EXPERIMENT – 6

IMAGE RESTORATION

OBJECTIVE:

I. Implement the Alpha trimmed Order Statistic Restoration filter.
II. Implement the Following Mean Restoration filters: Arithmetic, Contra Harmonic.

SAMPLE PROGRAM:

I. To implement the Alpha- trimmed Order Statistic Restoration filter.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

from numpy import mean
def trimmeanval(arr, d):
    n = len(arr)
    k = int(d/2)
    return mean(arr[k:n-k])

# Reading the input image
imgpath = 'C:/1_YKM/MANUAL/IMAGE_PROCESSING/B.TECH/2019-20/YKM/EXP. 6/test.tif'
img1 = cv2.imread(imgpath, 0)

##### Gaussian Filter
max_val=np.max(img1);
print(max_val);

img2=(img1/max_val); # Normalization

a=0;
b=0.01;
(nr,nc) = img1.shape

x=a+(b*np.random.normal(a,b,(nr,nc))); # Gaussian Distribution with mean (a) and variance (b)
y=a+((b-a)*np.random.rand(nr,nc));     # Uniform Distribution

img_guassian=(img2+x)*max_val;

##########Salt & Pepper Noise
max_val=np.max(img_guassian);
print(max_val);

img3=(img_guassian/max_val); # Normalization

pa=0.05;
pb=0.05;
(nr,nc) = img_guassian.shape

 R = np.uint8(np.zeros((nr,nc),dtype = 'uint8')+0.11);
 x=np.random.rand(nr,nc);
```

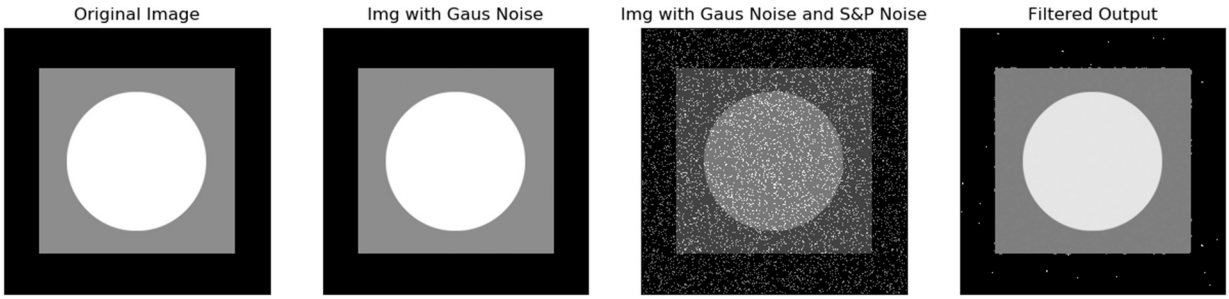
```
[r,c]=np.where(x<=pa);
for i in range(len(r)):
    R[r[i]][c[i]]=np.uint8(0);
u=pa+pb;

[r,c]=np.where(x<=u);
for i in range(len(r)):
    R[r[i]][c[i]]=np.uint8(255);

img_noise=img_guassian+R;
```

Alpha Trimmed Filter

```
img = img_noise;

(nr,nc) = img_noise.shape  # to access row and column of image

print('No. of Row: ',nr)
print('No. of Column: ', nc)

output=np.zeros((nr,nc),dtype='uint8');
ary=np.zeros(9,dtype='uint8');
ary1=np.zeros(9,dtype='uint8');

d=8

for i in range(1,nr-1):
    for j in range(1,nc-1):
        temp=0;
        for x in range(i-1,i+2):
            for y in range(j-1,j+2):
                ary[temp] = img_noise[x,y]
                temp +=1

        ary.sort()
        ary1 = ary
        output[i][j] = trimmeanval(ary1,d);


plt.subplot(1, 4, 1)
plt.imshow(img1,cmap='gray')
plt.title('Original Image')
plt.xticks([])
plt.yticks([])


plt.subplot(1, 4, 2)
plt.imshow(img_guassian,cmap='gray')
plt.title('Img with Gaus Noise')
plt.xticks([])
plt.yticks([])

plt.subplot(1, 4, 3)
plt.imshow(img_noise,cmap='gray')
plt.title('Img with Gaus Noise and S&P Noise')
plt.xticks([])
```

```
plt.yticks([])

plt.subplot(1, 4, 4)
plt.imshow(output,cmap='gray')
plt.title('Filtered Output')
plt.xticks([])
plt.yticks([])

plt.show()

cv2.waitKey(0) #Wait until key strike from keyboard
cv2.destroyAllWindows()#Close all windows
```

RESULT:



ASSIGNMENT:

(I) Implement the Following Order Statistic Restoration filters: Median, Max, Min, Mid-Point.

II. Implement the Following Mean Restoration filters: Arithmetic, Contra-Harmonic.

Arithmetic Filter:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Reading the input image
imgpath = 'C:/1_YKM/MANUAL/IMAGE_PROCESSING/B.TECH/2019-20/YKM/EXP. 6/test.tif'

img1 = cv2.imread(imgpath, 0)
```

Gaussian Filter

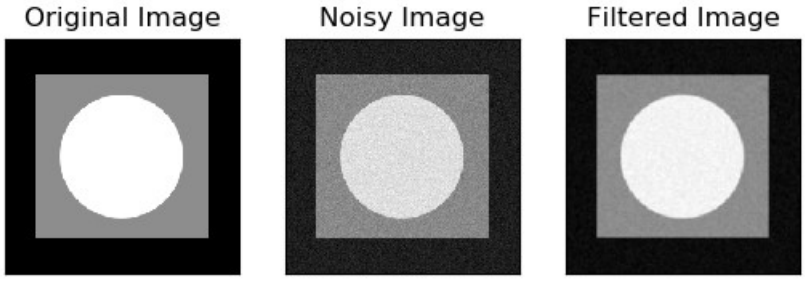
```
max_val=np.max(img1);
print(max_val);

img2=(img1/max_val); # Normalization

    a=0;
    b=0.2;
```

```
(nr,nc) = img1.shape
 # Gaussian Distribution with mean (a) and variance (b)
x=a+(b*np.random.normal(a,b,(nr,nc)));
# Uniform Distribution
y=a+((b-a)*np.random.rand(nr,nc));

img_guassian=(img2+x)*max_val;
```

Arithmetic Mean Filter

```
filterSize = 3
k1 = np.array(np.ones((filterSize, filterSize), np.float32))/(filterSize * filterSize) #average
filter

print(k1)                                          #printing mask
output = cv2.filter2D(img_guassian, -1, k1)        #here change mask variable

plt.subplot(1, 3, 1)
plt.imshow(img1,cmap='gray')
plt.title('Original Image')
plt.xticks([])
plt.yticks([])
plt.subplot(1, 3, 2)
plt.imshow(img_guassian,cmap='gray')
plt.title('Noisy Image')
plt.xticks([])
plt.yticks([])
plt.subplot(1, 3, 3)
plt.imshow(output,cmap='gray')
plt.title('Filtered Image')
plt.xticks([])
plt.yticks([])
plt.show()
cv2.waitKey(0) #Wait until key strike from keyboard
cv2.destroyAllWindows()#Close all windows
```

RESULT:

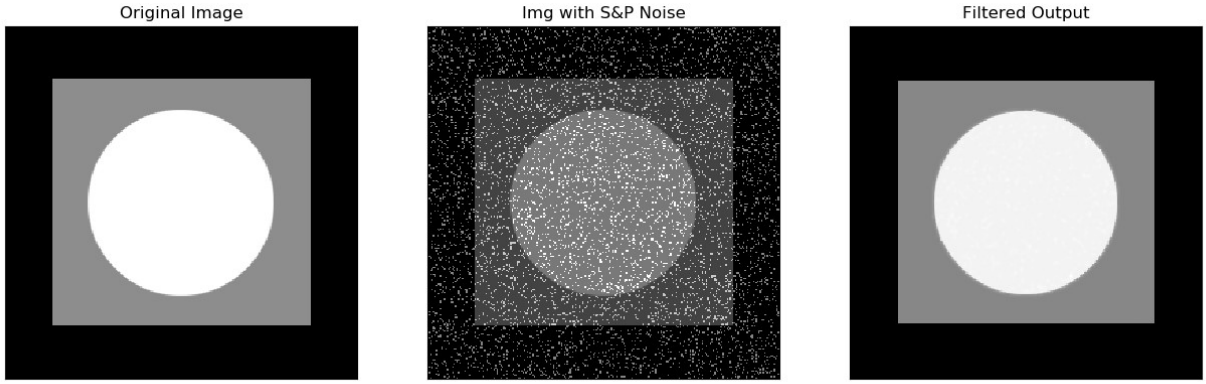


Contra-Harmonic Filter:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


# Reading the input image
imgpath = 'C:/1_YKM/MANUAL/IMAGE_PROCESSING/B.TECH/2019-20/YKM/EXP. 6/test.tif'

img1 = cv2.imread(imgpath, 0)

##########Salt & Pepper Noise

max_val=np.max(img1);
#print(max_val);

img3=(img1/max_val); # Normalization

pa=0.05;
pb=0.05;
(nr,nc) = img1.shape

R = np.float32(np.zeros((nr,nc),dtype = 'float32')+0.11);
x=np.random.rand(nr,nc);
[r,c]=np.where(x<=pa);
for i in range(len(r)):
   R[r[i]][c[i]]=np.uint8(0);
u=pa+pb

[r,c]=np.where(x<=u);
for i in range(len(r)):
   R[r[i]][c[i]]=np.uint8(255);

img_noise=img1+R;

###### Contra-Harmonic Filter

(nr,nc) = img_noise.shape  # to access row and column of image

print('No. of Row: ',nr)
print('No. of Column: ', nc)

output=np.zeros((nr,nc),dtype='uint8');

Q=-5

for i in range(1,nr-1,1):
   for j in range(1,nc-1,1):
      num=0;denom=0;
      for x in range(i-1,i+2):
         for y in range(j-1,j+2):
            num = num + (pow(img_noise[x][y],(Q+1)))
            denom = denom + (pow(img_noise[x][y],Q))
      if denom != 0:
```

```
output[i][j]= num / denom;


 plt.subplot(1, 3, 1)
 plt.imshow(img1,cmap='gray')
 plt.title('Original Image')
 plt.xticks([])
 plt.yticks([])

 plt.subplot(1, 3, 2)
 plt.imshow(img_noise,cmap='gray')
 plt.title('Img with S&P Noise')
 plt.xticks([])
 plt.yticks([])

 plt.subplot(1, 3, 3)
 plt.imshow(output,cmap='gray')
 plt.title('Filtered Output')
 plt.xticks([])
 plt.yticks([])
 plt.show()

 cv2.waitKey(0) #Wait until key strike from keyboard
 cv2.destroyAllWindows()#Close all windows
```

RESULT:



ASSIGNMENT:

(I) Implement the Following Mean Restoration filters: Geometric, Harmonic.

CONCLUSION:

EXPERIMENT – 7

COLOR IMAGE PROCESSING

OBJECTIVE:

I. Split and merge the R, G, B component from the color image.
II. Observe effect saturation of red, green and blue color component in the blank image.

SAMPLE PROGRAM:

I. Split and merge the R, G, B component from the color image

```
import cv2
import matplotlib.pyplot as plt

imgpath1 =  "/home/IP_Python 2018-19/ippythonprograms/index.jpg"
img1 = cv2.imread(imgpath1, 1)

cv2.imshow('Original Color Image', img1)
cv2.waitKey(0)

b,g,r = cv2.split(img1)
rgb_img = cv2.merge([b,g,r])
cv2.imshow('Color Image', rgb_img)
cv2.waitKey(0)

cv2.imshow('Blue Component', b)
cv2.waitKey(0)

cv2.imshow('Green Component', g)
cv2.waitKey(0)

cv2.imshow('Red Component', r)
cv2.waitKey(0)

cv2.destroyAllWindows()
titles = ['RED Component', 'GREEN Component', 'BLUE Component']
images = [r, g, b]
for k in range(3):
        plt.subplot(1, 3, k+1)
        plt.imshow(images[k],cmap='gray')
        plt.title(titles[k])
        plt.xticks([])
        plt.yticks([])

plt.show()
```

RESULT:

II. Observe effect saturation of red, green and blue color component in the blank image

```
import cv2
import numpy as np
def emptyFunction():
    pass
def main():
    img1 = np.zeros((512, 512, 3), np.uint8)
    windowName = 'OpenCV BGR Color Palette'
    cv2.namedWindow(windowName)

    cv2.createTrackbar('B', windowName, 0, 255, emptyFunction)
    cv2.createTrackbar('G', windowName, 0, 255, emptyFunction)
    cv2.createTrackbar('R', windowName, 0, 255, emptyFunction)

    while(True):
        cv2.imshow(windowName, img1)
     #  cv2.waitKey(0)
        if cv2.waitKey(1) == 27:   # ASCII Code for the ESC key
            break
              blue = cv2.getTrackbarPos('B', windowName)
        green = cv2.getTrackbarPos('G', windowName)
        red = cv2.getTrackbarPos('R', windowName)

        img1[:] = [blue, green, red]
        print (blue, green, red)

    cv2.destroyAllWindows()

if __name__ == "__main__":
        main()
```

III. Write a program to convert the RGB color model to HSI model.

```
import cv2
import converter

img = cv2.imread('index.jpg', 1);
img=cv2.resize(img,(300,300),cv2. INTER_AREA);
```

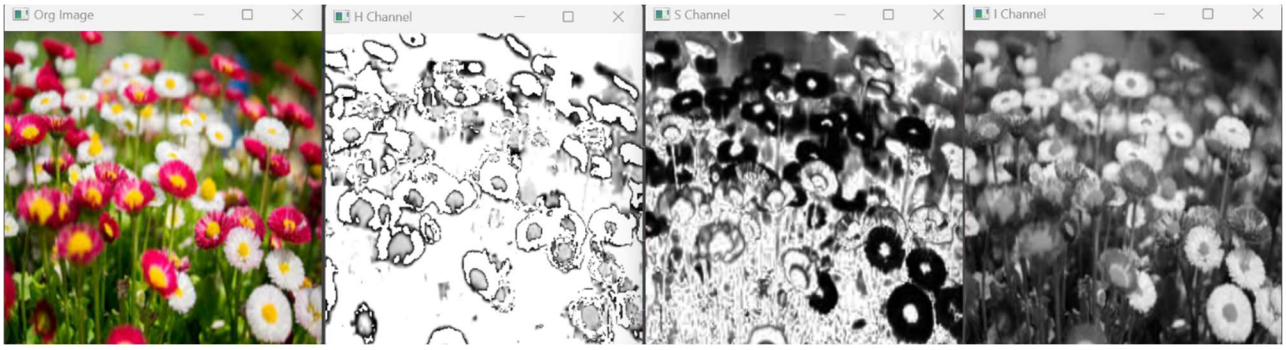
```
cv2.imshow('Org Image', img)
cv2.waitKey(0)

hsi = converter.RGB_TO_HSI(img)

# Display HSV Image
cv2.imshow('HSI Image', hsi)
cv2.waitKey(0)
# The three value channels

cv2.imshow('H Channel', hsi[:, :, 0])
cv2.waitKey(0)

cv2.imshow('S Channel', hsi[:, :, 1])
cv2.waitKey(0)

cv2.imshow('I Channel', hsi[:, :, 2])

# Wait for a key press and then terminate the program
cv2.waitKey(0)
cv2.destroyAllWindows()
```

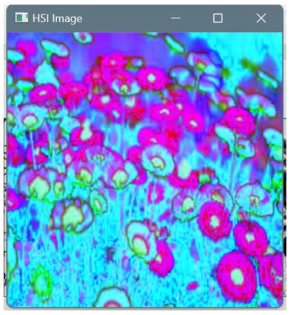
Convert Function

```
import cv2
import numpy as np
import math

def RGB_TO_HSI(img):

   with np.errstate(divide='ignore', invalid='ignore'):

      #Load image with 32 bit floats as variable type
      bgr = np.float32(img)/255

      #Separate color channels
      blue = bgr[:,:,0]
      green = bgr[:,:,1]
      red = bgr[:,:,2]

      #Calculate Intensity
      def calc_intensity(red, blue, green):
         return np.divide(blue + green + red, 3)

      #Calculate Saturation
      def calc_saturation(red, blue, green):
         minimum = np.minimum(np.minimum(red, green), blue)
         saturation = 1 - (3 / (red + green + blue + 0.001) * minimum)

         return saturation

      #Calculate Hue
      def calc_hue(red, blue, green):
         hue = np.copy(red)
```

```
                for i in range(0, blue.shape[0]):
                    for j in range(0, blue.shape[1]):
                        hue[i][j] = 0.5 * ((red[i][j] - green[i][j]) + (red[i][j] - blue[i][j])) / \
                                math.sqrt((red[i][j] - green[i][j])**2 +
                                        ((red[i][j] - blue[i][j]) * (green[i][j] - blue[i][j])))
                        hue[i][j] = math.acos(hue[i][j])

                        if blue[i][j] <= green[i][j]:
                            hue[i][j] = hue[i][j]
                        else:
                            hue[i][j] = ((360 * math.pi) / 180.0) - hue[i][j]

                return hue

            #Merge channels into picture and return image
             hsi = cv2.merge((calc_hue(red, blue, green), calc_saturation(red, blue, green),
calc_intensity(red, blue, green)))
            return hsi
```

OUTPUT:





CONCLUSION:

EXPERIMENT –08

IMAGE SEGMENTATION & MORPHOLOGICAL OPERATION

OBJECTIVE:

I. To find optimum threshold value by maximizing between class variance.
II. Perform erosion operation on binary image with various structuring elements.
III. Perform dilation operation on binary image with various structuring elements.
IV. Illustrating morphological open by erosion followed by dilation.
V. Illustrating morphological close by dilation followed by erosion.

IMAGE SEGMENTATION

SAMPLE PROGRAM:

I. **To find optimum threshold value by maximizing between class variance**.

```
# Calculate the histogram
import cv2
imgpath1 = "G:\\DEPT_2021\\ODDSEM\\IP_Labs\\tour1.jpg"
img1 = cv2.imread(imgpath1, 0)

#Between class variance
hist = plt.hist(img1.ravel(),256,[0,256])
# Total pixels in the image
total = np.sum(hist[0])
# calculate the initial weights and the means
left, right = np.hsplit(hist[0],[0])
left_bins, right_bins = np.hsplit(hist[1],[0])
# left weights
w_0 = 0.0
# Right weights
w_1 = np.sum(right)/total
# Left mean
mean_0 = 0.0
weighted_sum_0 = 0.0
# Right mean
weighted_sum_1 = np.dot(right,right_bins[:-1])
mean_1 = weighted_sum_1/np.sum(right)
def recursive_otsu1(hist, w_0=w_0, w_1=w_1, weighted_sum_0=weighted_sum_0,
weighted_sum_1=weighted_sum_1, thres=1, fn_max=-np.inf, thresh=0, total=total):
    if thres<=255:
        # To pass the division by zero warning
        if np.sum(hist[0][:thres+1]) !=0 and np.sum(hist[0][thres+1:]) !=0:
            # Update the weights
            w_0 += hist[0][thres]/total
            w_1 -= hist[0][thres]/total
            # Update the mean
            weighted_sum_0 += (hist[0][thres]*hist[1][thres])
            mean_0 = weighted_sum_0/np.sum(hist[0][:thres+1])
            weighted_sum_1 -= (hist[0][thres]*hist[1][thres])
            if thres == 255:
                mean_1 = 0.0
```

```
else:
    mean_1 = weighted_sum_1/np.sum(hist[0][thres+1:])
# Calculate the between-class variance
out = w_0*w_1*((mean_0-mean_1)**2)
# # if variance maximum, update it
if out>fn_max:
            fn_max = out
            thresh = thres
       return recursive_otsu1(hist, w_0=w_0, w_1=w_1, weighted_sum_0=weighted_sum_0,
weighted_sum_1=weighted_sum_1, thres=thres+1, fn_max=fn_max, thresh=thresh,
total=total)
    # Stopping condition
    else:
       return fn_max,thresh

# Check the results
var_value, thresh_value = recursive_otsu1(hist, w_0=w_0, w_1=w_1,
weighted_sum_0=weighted_sum_0, weighted_sum_1=weighted_sum_1, thres=1,
fn_max=-np.inf, thresh=0, total=total)
print(var_value, thresh_value)
# threshold the image
ret, thresh_Between = cv2.threshold(img1,0,255,thresh_value)
# Otsu's thresholding using inbuilt function
retval, thresh_Otsu = cv2.threshold(img1,0,255,cv2.THRESH_OTSU)
print(retval)
```

RESULT :

2180.6899678456916 117

114.0

MORPHOLOGICAL OPERATION

THEORY:

Morphology word commonly denotes a branch of biology that deals with the form and structure of animals and plants. Using the same word in context of image processing, mathematical morphology is a tool for extracting image components that are useful in the representation and description of region shape, such as boundaries, skeletons, and the convex hull. The language of mathematical morphology is set theory. It was originally defined for binary images, later being extended to grayscale images. The basic idea in binary morphology is to probe an image with a simple, pre-defined shape, drawing conclusions on how this shape fits or misses the shapes in the image. This simple "probe" is called structuring element, and is itself a binary image (i.e., a subset of the space or grid).

The basic morphological operators are erosion, dilation, opening and closing.

1. Erosion:

Erosion of set A by structuring element B is denoted as A Θ B, and is defined as,

$$A \ominus B = \left\{ Z \mid (\hat{B})_Z \subseteq A \right\}$$

Erosion shrinks or thins objects in a binary image.

2. Dilation:

Dilation of set A by structuring element B is defined by the following equation.

$A \oplus B = \left\{ Z \mid (\hat{B})_Z \cap A \neq \emptyset \right\}$

Unlike erosion, dilation grows. Dilation is used for expanding an element A by using structuring element B.

3. Opening:

Theopening of setA by structuring element B, denoted A ° B, is defined as,

A ° B = (A Θ B) ⊕ B

The opening A by B is the erosion of A by B, followed by a dilation of the result by B. Opening generally smoothes the contour of an object, breaks narrow isthmuses, and eliminates thin protrusions.

4. Closing:

Closing of set A by structuring element B is defined as,

A • B = (A ⊕ B) Θ B

The closing A by B is the dilation of A by B, followed by an erosion of the result by B. Closingalso tends to smooth sections of contours but, as opposed to opening, it generally fuses narrow breaks and long thin gulfs, eliminates small holes, and fills gaps in the contour.

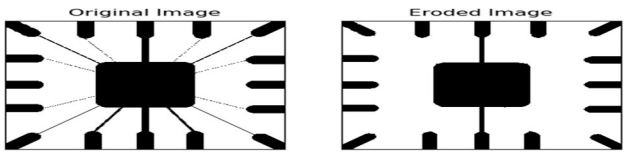
SAMPLE PROGRAM:

I. Perform erosion operation on binary image with various structuring elements

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Reading the input image
imgpath = 'C:/1_YKM/MANUAL/IMAGE_PROCESSING/B.TECH/2019-20/YKM/EXP. 9/input.tif'
img = cv2.imread(imgpath, 0)# ‘0’ is for gray scale image

# Taking a matrix of size 5 as the kernel
kernel = np.ones((5,5), np.uint8)

img_erosion = cv2.erode(img, kernel, iterations=1)

#cv2.imshow('Input', img)
#cv2.imshow('Eroded Output', img_erosion)

                    # Writing the output image
```

```
outpath = 'C:/MANUAL/IMAGE_PROCESSING/B.TECH/2019-20/EXP. 9/erosionOut.tif'
cv2.imwrite(outpath, img)

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='binary')
plt.title('Original Image')
plt.xticks([])
plt.yticks([])
plt.subplot(1, 2, 2)
plt.imshow(img_erosion, cmap='binary')
plt.title('Eroded Image')
plt.xticks([])
plt.yticks([])

plt.show()

cv2.waitKey(0) #Wait until key strike from keyboard
cv2.destroyAllWindows()#Close all windows
```

RESULTS:



MODIFICATIONS:

(i) Perform dilation operation on binary image with various structuring elements.
Hint: in-built function is: dilate

(ii) Illustrating morphological open by erosion followed by dilation.
Hint: in-built function is:
img_open = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

(iii) Illustrating morphological open by dilation followed by erosion.
Hint: in-built function is:
img_close = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

ASSIGNMENTS:

I. Apply (i) different structuring elements, (ii) different iterations, and write your observations.

II. Modify the program without using in-built erode and dilate functions.

III. Modify the open and close programs using erosion and dilation functions.

CONCLUSION:

EXPERIMENT – 09

IMAGE CLASSIFICATION USING K-MEANS CLUSTERING ALGORITHM

OBJECTIVE:

To implement image classification using the K-Means clustering algorithm, which is an unsupervised learning technique.

Theory:

K-Means Clustering: K-Means is a popular unsupervised learning algorithm used for clustering data into K distinct groups based on their features. The algorithm aims to partition the dataset into K clusters, where each data point belongs to the cluster with the nearest mean. The K-Means algorithm minimizes the within-cluster variance, i.e., the sum of the squared distances between data points and their corresponding cluster centroids.

Steps in K-Means Clustering:

1. **Initialization:** Choose K initial cluster centroids randomly.
2. **Assignment :** Assign each data point to the cluster whose centroid is closest to it.
3. **Update Step:** Recalculate the centroids by taking the mean of all data points assigned to each cluster.
4. **Repeat:** Continue the assignment and update steps until the centroids no longer change or a maximum number of iterations is reached.
5. **Labeling Clusters:** Assign labels to the clusters based on the majority class of data points within each cluster.

Advantages of K-Means:

- Simple and easy to implement.
- Efficient for large datasets.
- Works well when the clusters are well-separated.

Disadvantages of K-Means:

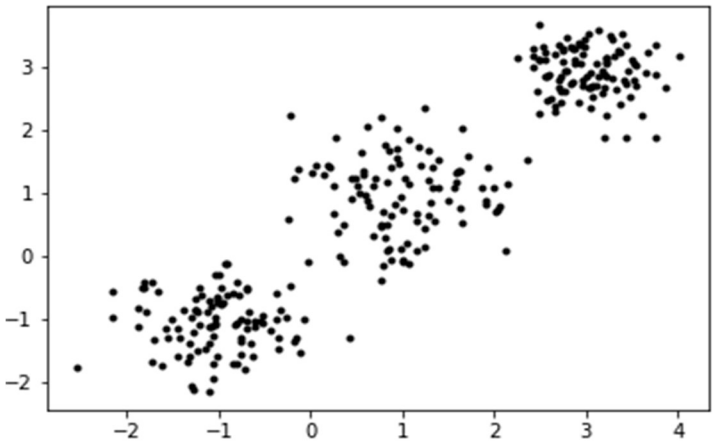
- Requires the number of clusters K to be specified in advance.
- Sensitive to the initial placement of centroids.
- May converge to a local minimum rather than a global minimum.

SAMPLE PROGRAM:

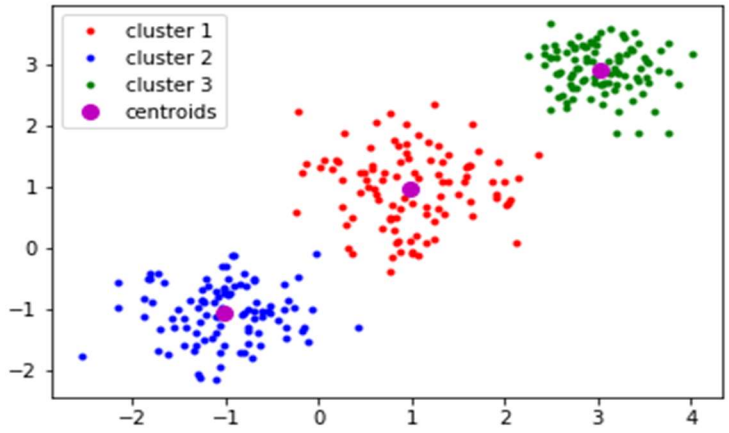
(i) K-Means clustering applied on randomly generated data.

```
import numpy as np
import matplotlib.pyplot as plt


from sklearn.cluster import KMeans
np.random.seed(7)

#generate 2D random data for three clusters and represent them

def generate_data():
    np.random.seed(7)
    x1 = np.random.standard_normal((100,2))*0.6+np.ones((100,2))
    x2 = np.random.standard_normal((100,2))*0.5-np.ones((100,2))
    x3 = np.random.standard_normal((100,2))*0.4-2*np.ones((100,2))+5
    X = np.concatenate((x1,x2,x3),axis=0)
    return X

#generate the k=3 initial centroids with the function
n = 3
X = generate_data()
plt.plot(X[:,0],X[:,1],'k.')
plt.show()

k_means = KMeans(n_clusters=n)
model = k_means.fit(X)
centroids = k_means.cluster_centers_
labels= k_means.labels_
plt.figure()
plt.plot(X[labels==0,0],X[labels==0,1],'r.', label='cluster 1')
plt.plot(X[labels==1,0],X[labels==1,1],'b.', label='cluster 2')
plt.plot(X[labels==2,0],X[labels==2,1],'g.', label='cluster 3')

plt.plot(centroids[:,0],centroids[:,1],'mo',markersize=8, label='centroids')

plt.legend(loc='best')
plt.show()
```

OUTPUT:



Raw data points



data clustered in three groups

MODIFICATIONS:

I. Generate different number of clusters and try with different number of centroids. Write your observations.

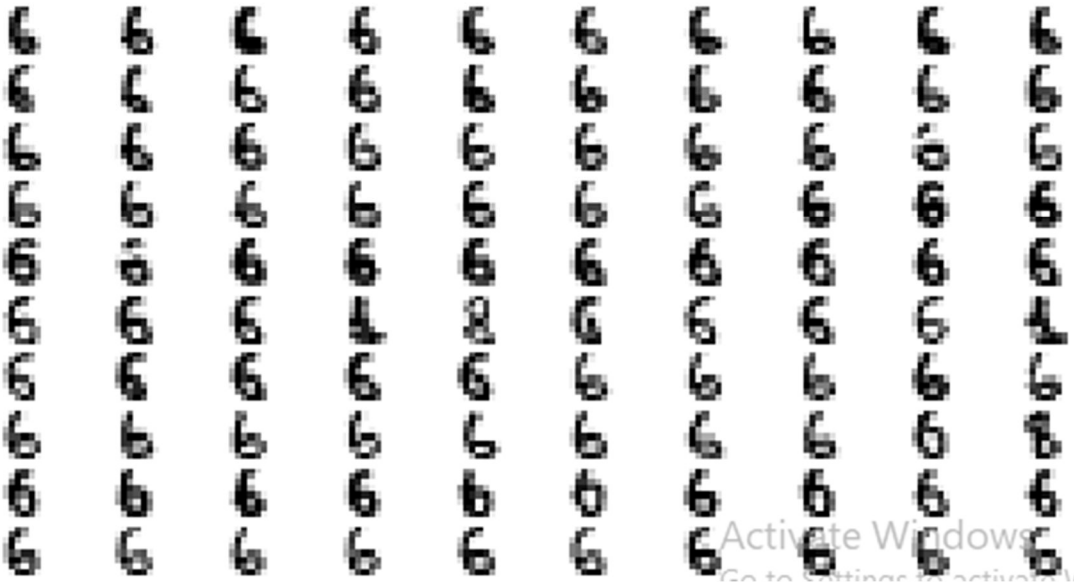
(ii) Kmeans clustering in MNIST dataset classification.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits

digits = load_digits()
data = digits.data
print(data.shape)

#To improve the visualization, we invert the colors

data = 255-data
np.random.seed(1)
```

```
n=10
kmeans = KMeans(n_clusters=n,init='random')
kmeans.fit(data)
Z = kmeans.predict(data)
for i in range(0,n):

    row = np.where(Z==i)[0]       # row in Z for elements of cluster i
    num = row.shape[0]            #  number of elements for each cluster
    r = int(np.floor(num/10.))    # number of rows in the figure of the cluster

    print("cluster "+str(i))
    print(str(num)+" elements")

    plt.figure(figsize=(10,10))
    for k in range(0, num):
        plt.subplot(r+1, 10, k+1)
        image = data[row[k], ]
        image = image.reshape(8, 8)
        plt.imshow(image, cmap='gray')
        plt.axis('off')
    plt.show()
```

OUTPUT:

```
cluster 0
182 elements
```



You will get similar outputs for all other clusters.

CONCLUSION:

EXPERIMENT – 10

PERCEPTRON AND GRADIENT DECENT ALGORITHMS

OBJECTIVE:

I. To evaluate weight for different logic gates using perceptron algorithm.
II. To apply gradient decent algorithm.

PERCEPTRON ALGORITHM

THEORY:

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. Single-layer perceptrons are only capable of learning linearly separable patterns.

SAMPLE PROGRAM:

I. To evaluate weight for OR logic gate using perceptron algorithm.

```
import numpy as np
atributes = np.array([ [0, 0], [0, 1], [1, 0], [1, 1]])
labels = np.array([0, 1, 1, 1])
w = [1, 1]
threshold = 0.5
bias = 0
alpha = 0.2
epoch = 50
print("learning rate: ", alpha,", threshold: ", threshold)
for i in range(0, epoch):
        print("epoch ", i+1)
        global_delta = 0
        for j in range(len(atributes)):
                actual = labels[j]
                sum = atributes[j][0]*w[0] + atributes[j][1]*w[1] + bias
                if sum > threshold:
                        predicted = 1
                else:
                        predicted = 0
                delta = actual - predicted
                global_delta = global_delta + abs(delta)
                for k in range(0, 2):
                        w[k] = w[k] + delta * alpha
                print(atributes[j][0]," ", labels, " ", atributes[j][1], " -> actual: ", actual, ", predicted: ",
                 predicted, " (w: ",w[0],")")
                if global_delta == 0:
                        break
                print("------------------------------")
```

RESULTS:

```
learning rate:  0.2 , threshold:  0.5
epoch  1
0   [0 1 1 1]   0  -> actual:  0 , predicted:  0  (w:  1.0 )
0   [0 1 1 1]   1  -> actual:  1 , predicted:  1  (w:  1.0 )
1   [0 1 1 1]   0  -> actual:  1 , predicted:  1  (w:  1.0 )
1   [0 1 1 1]   1  -> actual:  1 , predicted:  1  (w:  1.0 )
```

MODIFICATIONS:

1. To evaluate weight for AND logic gate using perceptron algorithm.
2. To evaluate weight for XOR logic gate using perceptron algorithm and write your observations.

ASSIGNMENTS:

I. Apply different initial weight and bias values and write your observations.
II. Apply different learning rate and write your observations.

(II) GRADIENT DECENT ALGORITHM

THEORY:

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. It trains machine learning models by minimizing errors between predicted and actual results. Training data helps these models learn over time.

SAMPLE PROGRAM:

I. To apply gradient decent algorithm for y = x^2

```
from numpy import asarray
from numpy import arange
from numpy.random import rand
from matplotlib import pyplot

def objective(x):
  return x**2.0

def derivative(x):
  return x * 2.0

def gradient_descent(objective, derivative, bounds, n_iter, step_size):
  solutions, scores = list(), list()
  solution = bounds[:, 0] + rand(len(bounds)) * (bounds[:, 1] - bounds[:, 0])
  for i in range(n_iter):
        gradient = derivative(solution)
        solution = solution - step_size * gradient
        solution_eval = objective(solution)
        solutions.append(solution)
        scores.append(solution_eval)
        print('>%d f(%s) = %.5f' % (i, solution, solution_eval))
  return [solutions, scores]
```

```
bounds = asarray([[-1.0, 1.0]])
n_iter = 5
step_size = 0.0001

solutions, scores = gradient_descent(objective, derivative, bounds, n_iter, step_size)
inputs = arange(bounds[0,0], bounds[0,1]+0.1, 0.1)
results = objective(inputs)
pyplot.plot(inputs, results)
pyplot.plot(solutions, scores, '.-', color='red')
pyplot.show()
```

RESULTS:

```
>0 f([-0.11063056]) = 0.01224
>1 f([-0.11060843]) = 0.01223
>2 f([-0.11058631]) = 0.01223
>3 f([-0.1105642]) = 0.01222
>4 f([-0.11054208]) = 0.01222
```

MODIFICATIONS:

- **I.** Change the number of iteration and write your observations.
- **II.** Apply different step size and write your observations.

ASSIGNMENTS:

- **I.** To apply gradient decent algorithm for y = 2(x^2)

CONCLUSION:

P A R T II

APPENDIX

COOLPIX 8400

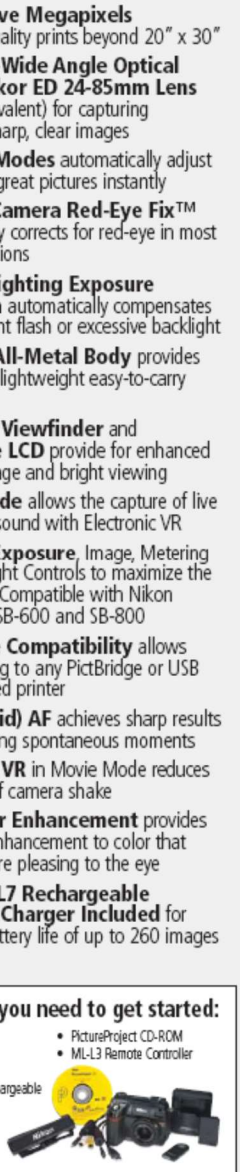
DIGITAL CAMERA

Key Features and Benefits

- **8.0 Effective Megapixels** for photo-quality prints beyond 20" x 30"
- **3.5x Ultra-Wide Angle Optical Zoom-Nikkor ED 24-85mm Lens** (35mm equivalent) for capturing incredible, sharp, clear images
- **15 Scene Modes** automatically adjust controls for great pictures instantly
- **Nikon In-Camera Red-Eye Fix™** automatically corrects for red-eye in most typical situations
- **Nikon D-Lighting Exposure Correction** automatically compensates for insufficient flash or excessive backlight
- **Compact All-Metal Body** provides a strong yet lightweight easy-to-carry camera
- **Electronic Viewfinder** and **Vari-Angle LCD** provide for enhanced frame coverage and bright viewing
- **Movie Mode** allows the capture of live action with sound with Electronic VR
- **Superior Exposure**, Image, Metering and Speedlight Controls to maximize the possibilities. Compatible with Nikon speedlights SB-600 and SB-800
- **PictBridge Compatibility** allows direct printing to any PictBridge or USB direct enabled printer
- **Fast (Hybrid) AF** achieves sharp results when shooting spontaneous moments
- **Electronic VR** in Movie Mode reduces the effects of camera shake
- **Auto Color Enhancement** provides automatic enhancement to color that makes it more pleasing to the eye
- **New EN-EL7 Rechargeable Battery & Charger Included** for extended battery life of up to 260 images on a charge

Everything you need to get started:

- Strap
- USB Cable
- Audio Video Cable
- EN-EL7 Li-ion Rechargeable Battery
- MH-56 Battery Charger
- PictureProject CD-ROM
- ML-L3 Remote Controller



Canon

Canon EOS 650D
Specification Sheet

IMAGE SENSOR		
	Type	22.2 x 14.8mm CMOS
	Effective Pixels	Approx. 18.0 megapixels
	Total Pixels	Approx. 18.5 megapixels
	Aspect Ratio	3:2
	Low-Pass Filter	Built-in/Fixed
	Sensor Cleaning	EOS integrated cleaning system
	Colour Filter Type	Primary Colour
IMAGE PROCESSOR		
	Type	DIGIC 5
LENS		
	Lens Mount	EF/EF-S
	Focal Length	Equivalent to 1.6x the focal length of the lens
FOCUSING		
	Type	TTL secondary image forming, phase difference detection with AF dedicated CMOS sensor
	AF System/ Points	9 cross-type AF points (f/2.8 at centre)
	AF working Range	EV -0.5 - 18 (at 23ºC & ISO100)
	AF Modes	AI Focus One Shot AI Servo
	AF Point Selection	Automatic selection, Manual selection
	Selected AF Point Display	Superimposed in viewfinder and indicated on LCD monitor
	Predictive AF (1)	Yes, up to 10m
	AF Lock	Locked when shutter button is pressed half way
	AF Assist Beam	Intermittent firing of built-in flash or emitted by optional dedicated Speedlite
	Manual Focus	Selected on lens
EXPOSURE CONTROL		
	Metering Modes	TTL full aperture metering with 63-zone SPC Evaluative metering (linked to all AF points) Partial metering at centre (approx. 9% of viewfinder) Spot metering (approx. 4% of viewfinder at centre) Centre weighted average metering
	Metering Range	EV 1-20 (at 23ºC with 50mm f/1.4 lens ISO100)
	AE Lock	Auto: In One-shot AF mode with evaluative metering exposure is locked when focus is achieved Manual: By AE lock button in creative zone modes
	Exposure Compensation	+/-5 EV in 1/3 or 1/2 stop increments (can be combined with AEB)
	AEB	3 shots +/- 2 EV, 1/2 or 1/3-stop increments
	ISO Sensitivity (2)	AUTO(100-6400), 100-12800 in 1-stop increments ISO can be expandable to H: 25600)
SHUTTER		
	Type	Electronically-controlled focal-plane shutter
	Speed	30-1/4000 sec (1/2 or 1/3 stop increments), Bulb (Total shutter speed range. Available range varies by shooting mode)
WHITE BALANCE		
	Type	Auto white balance with the imaging sensor
	Settings	AWB, Daylight, Shade, Cloudy, Tungsten, White Fluorescent light, Flash, Custom White balance compensation: Blue/Amber +/-9 Magenta/ Green +/-9
	Custom White Balance	Yes, 1 setting can be registered

	WB Bracketing	+/-3 levels in single level increments 3 bracketed images per shutter release Selectable Blue/Amber bias or Magenta/ Green bias
VIEWFINDER		
	Type	Pentamirror
	Coverage (Vertical/Horizontal)	Approx. 95%
	Magnification	Approx. 0.85x (3)
	Eyepoint	Approx. 19mm (from eyepiece lens centre)
	Dioptre Correction	-3 to +1 m-1 (dioptre)
	Focusing Screen	Fixed
	Mirror	Quick-return half mirror (Transmission: reflection ratio of 40:60, no mirror cut-off with EF600mm f/4 or shorter)
	Viewfinder Information	AF information: AF points, focus confirmation light Exposure information: Shutter speed, aperture value, ISO speed (always displayed), AE lock, exposure level/compensation, spot metering circle, exposure warning, AEB Flash information: Flash ready, high-speed sync, FE lock, flash exposure compensation, red-eye reduction light Image information: Highlight tone priority (D+), monochrome shooting, maximum burst (1 digit display), White balance correction, SD card information
	Depth of Field Preview	Yes, with Depth of Field preview button
	Eyepiece Shutter	On strap
LCD MONITOR		
	Type	Touch screen vari angle 7.7cm (3.0") 3:2 Clear View II TFT, approx. 1040K dots
	Coverage	Approx. 100%
	Viewing Angle (horizontally/vertically)	Approx 170°
	Coating	Anti smudge
	Brightness Adjustment	Adjustable to one of seven levels
	Display Options	Quick Control Screen Camera settings
FLASH		
	Built-in Flash GN (ISO 100, Meters)	13
	Built-in Flash Coverage	Up to 17mm focal length (35mm equivalent: 28mm)
	Built-in Flash Recycle Time	Approx. 3 seconds
	Modes	Auto, Manual flash, Integrated Speedlite Transmitter
	Red-Eye Reduction	Yes - with red eye reduction lamp
	X-sync	1/200sec
	Flash Exposure Compensation	+/- 2EV in 1/2 or 1/3 increments
	Flash Exposure Bracketing	Yes, with compatible External Flash
	Flash Exposure Lock	Yes
	Second Curtain Synchronisation	Yes
	HotShoe/ PC Terminal	Yes/ No
	External Flash Compatibility	E-TTL II with EX series Speedlites, wireless multi-flash support
	External Flash Control	Via camera menu screen

SHOOTING		
	Modes	Scene Intelligent Auto, No Flash, Creative Auto, Portrait, Landscape, Close-up, Sports, Night Portrait, Handheld Night Scene, HDR Backlight Control, Program AE , Shutter priority AE, Aperture priority AE, Manual
	Picture Styles	Auto, Standard, Portrait, Landscape, Neutral, Faithful, Monochrome, User Defined (x3)
	Colour Space	sRGB and Adobe RGB
	Image Processing	Highlight Tone Priority Auto Lighting Optimizer (4 settings) Long exposure noise reduction High ISO speed noise reduction (4 settings) Multi Shot Noise Reduction Auto Correction of Lens Peripheral illumination Basic+ (Shoot by ambience selection, Shoot by lighting or scene type) Creative filters (Art Bold, Water painting, Grainy B/W, Soft focus, Toy camera, Miniature effect, Fish-eye) - during image Playback only
	Drive Modes	Single, Continuous, Self timer (2s, 10s+remote, 10s + continuous shots 2-10)
	Continuous Shooting	Max. Approx. 5fps for approx. 22 JPEG images [4], 6 images RAW [5][6]
	Intervalometer	Approx. 99% (horizontally and vertically)
LIVE VIEW MODE		
	Type	Electronic viewfinder with image sensor
	Coverage	Approx. 99% (horizontally and vertically)
	Frame Rate	30 fps
	Focusing	Manual Focus (Magnify the image 5x or 10x at any point on screen) Autofocus: Hybrid CMOS AF (Face detection and Tracking AF, FlexiZone-Multi, FlexiZone-Single), Phase detection AF (Quick mode)
	Metering	Real-time evaluative metering with image sensor. Evaluative metering, partial metering, spot metering, centre-weighted average metering
	Display Options	Grid overlay, Histogram
FILE TYPE		
	Still Image Type	JPEG: Fine, Normal (Exif 2.30 compliant) / Design rule for Camera File system (2.0), RAW: RAW (14bit, Canon original RAW 2nd edition), Digital Print Order Format [DPOF] Version 1.1 compliant
	RAW+JPEG Simultaneous Recording	Yes, RAW + Large JPEG
	Image Size	JPEG 3:2: (L) 5184x3456, (M) 3456x2304, (S1) 2592x1728, (S2) 1920x1280, (S3) 720x480 JPEG 4:3: (L) 4608x3456, (M) 3072x2304, (S1) 2304x1728, (S2) 1696x1280, (S3) 640x480 JPEG 16:9: (L) 5184x2912, (M) 3456x1944, (S1) 2592x1456 (S2) 1920x1080, (S3) 720x400 JPEG 1:1: (L) 3456x3456, (M) 2304x2304, (S1) 1728x1728, (S2) 1280x1280, (S3) 480x480 RAW: (RAW) 5184x3456, (M-RAW) 3888x2592, (S-RAW) 2592x1728
	Movie Type	MOV (Video: H.264, Sound: Linear PCM, recording level can be manually adjusted by user)
	Movie Size	1920 x 1080 (29.97, 25, 23.976 fps) 1280 x 720 (59.94, 50 fps) 640 x 480 (30, 25 fps)
	Movie Length	Max duration 29min 59sec, Max file size 4GB
	Folders	New folders can be manually created and selected
	File Numbering	Consecutive numbering Auto reset Manual reset

OTHER FEATURES		
	Custom Functions	8 Custom Functions with 34 settings
	Metadata Tag	User copyright information (can be set in camera) Image rating (0-5 stars)
	Intelligent Orientation Sensor	No
	Playback Zoom	1.5x - 10x enabled in 15steps
	Display Formats	Single image with information (2 levels) Single image 4 image index 9 image index Jump Display
	Slide Show	Image selection: All images, by Date, by Folder, Movies, Stills Playback time: 1/2/3/5 seconds Repeat: On/Off
	Histogram	Brightness: Yes RGB: Yes
	Highlight Alert	Yes
	Image Erase/Protection	Erase: Single image, All images in folder, Checkmarked images, unprotected images Protection: Erase protection of one image at a time
	Menu Categories	Shooting menu (x5) Playback menu (x2) Setup menu (x3) Custom Functions menu My Menu
	Menu Languages	25 Languages English, German, French, Dutch, Danish, Portuguese, Finnish, Italian, Norwegian, Swedish, Spanish, Greek, Russian, Polish, Czech, Hungarian, Romanian, Ukrainian, Turkish, Arabic, Thai, Simplified Chinese, Traditional Chinese, Korean and Japanese
	Firmware Update	Update possible by the user
INTERFACE		
	Computer	Hi-Speed USB
	Other	Video output (PAL/ NTSC) (integrated with USB terminal), HDMI mini output (HDMI-CEC compatible), External microphone (3.5mm Stereo mini jack)
DIRECT PRINT		
	Canon Printers	Canon Compact Photo Printers and PIXMA Printers supporting PictBridge
	PictBridge	Yes
STORAGE		
	Type	UHS-1, SD, SDHC or SDXC card
SUPPORTED OPERATING SYSTEM		
	PC & Macintosh	Windows XP inc SP3 / Vista inc SP1 and SP2 (excl. Starter Edition) / 7 (excl. Starter Edition) Mac OS X 10.6 -10.7 (Intel processor required)
SOFTWARE		
	Browsing & Printing	ImageBrowser EX
	Image Processing	Digital Photo Professional
	Other	PhotoStitch, EOS Utility (inc. Remote Capture), Picture Style Editor
POWER SOURCE		
	Batteries	1 x Rechargeable Li-ion Battery LP-E8
	Battery Life	Approx. 440 (at 23ºC, AE 50%, FE 50%) (7) Approx. 400 (at 0ºC, AE 50%, FE 50%)
	Battery Indicator	4 levels
	Power Saving	Power turns off after 30sec or 1, 2, 4, 8 or 15mins
	Power Supply & Battery Chargers	AC Adapter Kit ACK-E8, Battery charger LC-E8, LC-E8E

PHYSICAL SPECIFICATIONS		
	Body Materials	Stainless Steel and polycarbonate resin with glass fibre
	Operating Environment	0 – 40 °C, 85% or less humidity
	Dimensions (WxHxD)	133.1 x 99.8 x 78.8 mm
	Weight (Body Only)	Approx. 575g (CIPA testing standard, including battery and memory card)
ACCESSORIES		
	Viewfinder	Eyecup Ef, E-series Dioptric Adjustment Lens with Rubber Frame Ef, Eyepiece Extender EP-EX15II, Angle Finder C
	Case	Semi-Hard Case EH22-L
	Wireless File Transmitter	Compatible with Eye-Fi cards
	Lenses	All EF and EF-S lenses
	Flash	Canon Speedlites (220EX, 270EX, 270EX II, 320EX, 420EX, 430EX, 430EX II, 550EX, 580EX, 580EX II, 600EX, 600EX-RT, Macro-Ring-Lite, MR-14EX, Macro Twin Lite MT-24EX, Speedlite Transmitter ST-E2, Speedlite Transmitter ST-E3-RT)
	Remote Controller/ Switch	Remote Switch RS-60E3, Remote Controller RC-6
	Other	Hand Strap E2

Canon

Delighting You Always India

Personal

Business

Contact Us
Support and Downloads
About Canon
E-Store
News and Press

PIXMA MG2570

Affordable All-In-One printer with basic printing, copying and scanning functions.

- Colour inkjet printer, copier and scanner
- ISO standard print speed (A4): up to 8.0ipm mono / 4.0ipm colour

Full Specifications Compare

Product compared by others

PIXMA

PIXMA MG2570

INR 4,150.00

Call us: 1800-180-3366, 1800-208-3366

Images are for illustration purpose only.

Compact and lightweight

Allow the printer to easily fit into even the smaller spaces and shelves in a home.

Auto Power ON

PIXMA MG2570

Auto Power ON detects a print command and will automatically switch the printer ON with a USB connection.

My Image Garden

PIXMA MG2570

A software that automatically creates various appealing collages and calendars using the photos stored on a PC to make suggestions to the user.

Specifications for PIXMA MG2570

Print		
Maximum Printing Resolution	4800 (horizontal)*1 x 600 (vertical)dpi	
Print Head / Ink	Type:	FINE Cartridge
	Number of Nozzles:	Total 1,280 nozzles
	Ink Droplet Size (minimum):	2pl
	Ink Cartridge:	PG-745, CL-746 (PG-745XL, CL746XL - Optional)
Print Speed Based on ISO / IEC 24734. Click here for summary report Click here for Document Print and Copy Speed Measurement Conditions	Document: Colour*2: ESAT / Simplex:	Approx. 4.0ipm
	Document: B/W*2: ESAT / Simplex:	Approx. 8.0ipm

Printable Width	Up to 203.2 mm (8 inch)	
	Bordered Printing:	Top margin: 3mm, Bottom margin: 16.7mm, Left / Right margin: each 3.4mm (LTR / LGL: Left: 6.4mm, Right: 6.3mm)
Recommended Printing Area	Top Margin:	31.6mm
	Bottom Margin:	29.2 mm
Paper Size	A4, A5, B5, LTR, LGL, 4 x 6", 5 x 7", Envelopes (DL, COM10), Custom size (width 101.6mm - 215.9mm, length 152.4mm - 676mm)	
Paper Handling (Rear Tray)	Plain Paper	A4, A5, B5, LTR = 60, LGL = 10
(Maximum Number)	Photo Paper Plus Glossy II (PP-201)	4 x 6" = 20
	Glossy Photo Paper "Everyday Use" (GP-501)	4 x 6" = 20
	Glossy Photo Paper "Everyday Use" (GP-601)	
	Envelope	European DL / US Com. #10 = 5
Paper Weight	Cassette:	Plain Paper: 64 - 105 g/m^2, Canon specialty paper: max paper weight: approx. 275 g/m^2 (Photo Paper Plus Glossy II (PP-201)
Ink End Sensor	Dot count	
Print Head Alignment	Manual	
Scan[4]		
Scanner Type	Flatbed	
Scanning Method	CIS (Contact Image Sensor)	
Optical Resolution[5]	600 x 1200dpi	
Selectable Resolution[6]	25 - 19200dpi	
Scanning Bit Depth (Input / Output)	Grayscale:	16 bits / 8 bits
	Colour:	48 bits / 24 bits (RGB each 16 bits / 8 bits)
Line Scanning Speed[7]	Grayscale:	1.2 ms/line (300dpi)
	Colour:	3.5 ms/line (300dpi)
Scanning Speed[8]	Reflective: A4 Colour / 300dpi	Approx. 14secs.
Maximum Document Size	Flatbed:	A4 / LTR (216 x 297mm)
Copy[9]		
Maximum Document Size	A4 / LTR (216 x 297mm)	
Compatible Media	Size:	A4, LTR
	Type:	Plain Paper
Image Quality	Plain Paper:	Fast, Standard
Copy Speed[10] Based on ISO / IEC 29183. Click here for summary report Click here for Document Print and Copy Speed Measurement Conditions	Document: Colour: sFCOT / Simplex:	Approx. 31secs.
	Document: Colour: sESAT / Simplex:	Approx. 1.6ipm
Multiple Copy	Black / Colour	1 - 21 pages
System Requirements (Please visit www.canon-asia.com to check OS compatibility and to download the latest driver updates.)	Windows:	Windows 8 / Windows 7 / Window Vista / Windows XP
	Macintosh:	Mac OS X v10.6.8 or later
General		
Interface	USB 2.0 Hi-Speed	
Operating Environment	Temperature:	5 - 35°C
	Humidity:	10 - 90% RH (no dew condensation)
Storage Environment	Temperature:	0 - 40°C
	Humidity:	5 - 95% RH (no dew condensation)
Power	AC 100 - 240V, 50 / 60Hz	
Power Consumption	Standby (scanning lamp is off) (USB connection to PC):	Approx. 1.0W
	OFF:	Approx. 0.4W
	Copying[11]: (USB connection to PC)	Approx. 9W
Environment	Regulation:	RoHS (EU, China), WEEE (EU)
	Eco-Label:	Energy Star, EPEAT
Dimension (W x D x H)	Approx. 426 x 306 x 145mm	
Weight	Approx. 3.5kg	

PART III

SESSIONAL QUESTION PAPER