CSL442_IVP_S21_Practice_Programs_Part-1

January 16, 2021

1 Pre-requisites and references

```
1. Minimum software installations: Python 3.6, numpy, matplotlib, cv2 (open computer vision lib
```

1.1 Image Processing Tools

1. OpenCV, 2. Python image library, 3. Scikit-image library

1.2 Python IDE

1. Jupyter, 2. Pycharm, 3. Spyder, 4. Visual Studio code

2 Introduction to Numpy.

1. NumPy is a Python open source package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

2. NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing.

3. Numpy package can be imported as " import numpy as np".

3 List vs Numpy

- List

```
1. The list can be homogeneous or heterogeneous.
2. Element wise operation is not possible on the list.
3. Python list is by default 1 dimensional. But we can create an N-Dimensional list. But then to
   storing another 1D list
4. Elements of a list need not be contiguous in memory.
```

- Numpy

```
1. We can create a N-dimensional array in python using numpy.array().
2. Array are by default Homogeneous, which means data inside an array must be of the same Dataty
3. Element wise operation is possible.
```

4. Numpy array has the various function, methods, and variables, to ease our task of matrix comp
5. Elements of an array are stored contiguously in memory. For example, all rows of a two dimens
   the same number of columns. Or a three dimensioned array must have the same number of rows an

3.0.1 *Example 1: Memory consumption between Numpy array and lists.*

[2]:
```
# importing numpy package
import numpy as np

# importing system module
import sys

# declaring a list of 1000 elements
S= range(1000)

# printing size of each element of the list
print("Size of each element of list in bytes: ",sys.getsizeof(S))

# printing size of the whole list
print("Size of the whole list in bytes: ",sys.getsizeof(S)*len(S))

# declaring a Numpy array of 1000 elements
D= np.arange(1000)

# printing size of each element of the Numpy array
print("Size of each element of the Numpy array in bytes: ",D.itemsize)

# printing size of the whole Numpy array
print("Size of the whole Numpy array in bytes: ",D.size*D.itemsize)
```

```
Size of each element of list in bytes:  48
Size of the whole list in bytes:  48000
Size of each element of the Numpy array in bytes:  4
Size of the whole Numpy array in bytes:  4000
```

3.0.2 *Example 2: Time comparison between Numpy array and Python lists.*

[1]:
```
# importing required packages
import numpy
import time

# size of arrays and lists
size = 10000000  # 1 crore elements

# declaring lists
list1 = range(size)
list2 = range(size)
```

```
# declaring arrays
array1 = numpy.arange(size)
array2 = numpy.arange(size)

# capturing time before the multiplication of Python lists
initialTime = time.time()

# multiplying  elements of both the lists and stored in another list
resultantList = [(a * b) for a, b in zip(list1, list2)]

# calculating execution time
print("Time taken by Lists to perform multiplication:",(time.time() -␣
 ↪initialTime),"seconds")

# capturing time before the multiplication of Numpy arrays
initialTime = time.time()

# multiplying  elements of both the Numpy arrays and stored in another Numpy␣
 ↪array
resultantArray = array1 * array2

# calculating execution time
print("Time taken by NumPy Arrays to perform multiplication:",(time.time() -␣
 ↪initialTime),"seconds")
```

```
Time taken by Lists to perform multiplication: 1.0252928733825684 seconds
Time taken by NumPy Arrays to perform multiplication: 0.0199434757232666 seconds
```

3.0.3 *Example 3: Effect of operations on Numpy array and Python Lists.*

[2]:
```
# importing Numpy package
import numpy as np

# declaring a list
ls =[1, 2, 3]

# converting the list into a Numpy array
arr = np.array(ls)

try:
    # adding 4 to each element of list
    ls = ls + 4

except(TypeError):
    print("Lists don't support list + int")
```

```
# now on array
try:
    # adding 4 to each element of Numpy array
    arr = arr + 4

    # printing the Numpy array
    print("Modified Numpy array: ",arr)

except(TypeError):
    print("Numpy arrays don't support list + int")
```

```
Lists don't support list + int
Modified Numpy array:  [5 6 7]
```

3.0.4 *Write a NumPy program to compute the inverse of a given matrix.*

[4]:
```
import numpy as np
m = np.array([[1,2],[3,4]])
print("Original matrix:")
print(m)
result =  np.linalg.inv(m)
print("Inverse of the said matrix:")
print(result)
```

```
Original matrix:
[[1 2]
 [3 4]]
Inverse of the said matrix:
[[-2.   1. ]
 [ 1.5 -0.5]]
```

3.0.5 *Write a NumPy program to get the dates of yesterday, today and tomorrow.*

[38]:
```
import numpy as np
yesterday = np.datetime64('today', 'D') - np.timedelta64(1, 'D')
print("Yestraday: ",yesterday)
today     = np.datetime64('today', 'D')
print("Today: ",today)
tomorrow  = np.datetime64('today', 'D') + np.timedelta64(1, 'D')
print("Tomorrow: ",tomorrow)
```

```
Yestraday:  2021-01-14
Today:  2021-01-15
Tomorrow:  2021-01-16
```

3.0.6 *Write a NumPy program to create a structured array from given student name, height, class and their data types. Now sort by class, then height if class are equal.*

[39]:
```
import numpy as np
data_type = [('name', 'S15'), ('class', int), ('height', float)]
students_details = [('James', 5, 48.5), ('Nail', 6, 52.5),('Paul', 5, 42.10),␣
 ↪('Pit', 5, 40.11)]
# create a structured array
students = np.array(students_details, dtype=data_type)
print("Original array:")
print(students)
print("Sort by class, then height if class are equal:")
print(np.sort(students, order=['class', 'height']))
```

```
Original array:
[(b'James', 5, 48.5 ) (b'Nail', 6, 52.5 ) (b'Paul', 5, 42.1 )
 (b'Pit', 5, 40.11)]
Sort by class, then height if class are equal:
[(b'Pit', 5, 40.11) (b'Paul', 5, 42.1 ) (b'James', 5, 48.5 )
 (b'Nail', 6, 52.5 )]
```

- Creating Numpy array ### There are a number of ways to initialize new numpy arrays, for example from: **- A Python list or tuples.**

- Using functions that are dedicated to generating numpy arrays, like arange, linspace, zeros, ones etc.

- Reading data from files.

- "rank" is defined as the size of the array.

[8]:
```
# Creating an array from list
import numpy as np

lst = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list

lst_arr = np.array(lst)  # convert a list to an array

print("\nArray created using passed list:\n", lst_arr)
```

```
Array created using passed list:
 [[1 2 3]
 [3 6 9]
 [2 4 6]]
```

[9]:
```
# Creating an array from tuple
tup_arr = np.array((1, 3, 2))
print("\nArray created using passed tuple:\n", tup_arr)
```

```
Array created using passed tuple:
 [1 3 2]
```

[12]:
```
# Creating an array using built-in functions
# np.range
a0=np.arange(1,100,2)# start , end , step

print(a0)
print(type(a0[0]))
print("\n")
```

```
[ 1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47
 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95
 97 99]
<class 'numpy.int32'>
```

[13]:
```
# np.linespace
a1=np.linspace(1,10,20) # i to 10 with    elements
print(a1)
print(type(a1[0]))
print("\n")

a2=np.linspace(1,10,20, dtype='float16')
print(a2)
print(type(a2[0]))
```

```
[ 1.          1.47368421  1.94736842  2.42105263  2.89473684  3.36842105
  3.84210526  4.31578947  4.78947368  5.26315789  5.73684211  6.21052632
  6.68421053  7.15789474  7.63157895  8.10526316  8.57894737  9.05263158
  9.52631579 10.        ]
<class 'numpy.float64'>


[ 1.     1.474  1.947  2.422  2.895  3.37   3.842  4.316  4.79   5.26
  5.74   6.21   6.684  7.156  7.633  8.1    8.58   9.055  9.52  10.   ]
<class 'numpy.float16'>
```

[15]:
```
# Example
a3 = np.zeros((3,4,2), dtype='uint8') # (no. of planes , rows, columns )
print ("\nAn array initialized with all zeros:\n", a3)
```

```
An array initialized with all zeros:
 [[[0 0]
  [0 0]
  [0 0]
  [0 0]]
```

```
 [[0 0]
  [0 0]
  [0 0]
  [0 0]]

 [[0 0]
  [0 0]
  [0 0]
  [0 0]]]
```

[16]:
```
# Example
a4 = np.zeros((3,3), dtype='uint8')
print ("\nAn array initialized with all zeros:\n", a4)
```

```
An array initialized with all zeros:
 [[0 0 0]
 [0 0 0]
 [0 0 0]]
```

[18]:
```
# Example
a5 = np.ones((3, 4))
print ("\nAn array initialized with all zeros:\n", a5)
a5copy=np.ones((3, 4), dtype='uint8')
print ("\nAn array initialized with all zeros in unsigned int datatype:\n",␣
 ↪a5copy)
```

```
An array initialized with all zeros:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

An array initialized with all zeros in unsigned int datatype:
 [[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
```

[19]:
```
# Example
# Create a constant value array of complex type

d = np.full((3, 4), 6, dtype = 'uint8')

print ("\nAn array initialized with all 6s. Array type is uint8:\n", d)
```

```
An array initialized with all 6s. Array type is uint8:
 [[6 6 6 6]
```

```
 [6 6 6 6]
 [6 6 6 6]]
```

[20]:
```
# Example
I = np.eye(3,dtype='uint8')
print ("\nAn identity matrix :\n", I)

print(type(I[0][0]))
print("\n")
```

```
An identity matrix :
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
<class 'numpy.uint8'>
```

[21]:
```
# Example
diaga=np.diag([1,2,3,4])
print(diaga)
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

3.0.7 - Accessing array elements

[22]:
```
# 1D array
import numpy as np

arr=np.arange(1,10,2)
print(arr)

print(arr[0]) # from first to last

print(arr[-2]) # from last to first

arr[2]=50
print(arr)
```

```
[1 3 5 7 9]
1
7
[ 1  3 50  7  9]
```

[23]:
```
# 2D array
import numpy as np
arr3 = np.array( [[ 1, 2, 3],
                  [ 4, 2, 5],
                  [ 4, 2, 6]] )
print(arr3[1][2])
print(arr3[2,2])
print(arr3[-1,-1])
```

```
5
6
6
```

[24]:
```
# 3D array
import numpy as np
arr4 = np.array([
                    [
                        [ 1, 2, 3],
                        [ 4, 2, 5],
                        [0,0,0]
                    ],
                    [
                        [ 41, 21, 51],
                        [121, 222,125],
                        [1,1,1]
                    ],
                    [
                        [ 9, 10, 11],
                        [12, 13, 15],
                        [2,2,2]
                    ]
                ] )
print(arr4.shape) # (dimention, rows, coloumn)
print("\n")
print(arr4)
print("\n")
print(arr4[1][1][1])
print(arr4[2][2][0])
print("\n")
arr4[2][2][2]=500
print("Updated array:\n\n",arr4)
```

```
(3, 3, 3)


[[[  1   2   3]
  [  4   2   5]
  [  0   0   0]]
```

```
 [[ 41  21  51]
  [121 222 125]
  [  1   1   1]]

 [[  9  10  11]
  [ 12  13  15]
  [  2   2   2]]]


222
2


Updated array:

 [[[  1   2   3]
  [  4   2   5]
  [  0   0   0]]

 [[ 41  21  51]
  [121 222 125]
  [  1   1   1]]

 [[  9  10  11]
  [ 12  13  15]
  [  2   2 500]]]
```

[40]:
```
# few Built-in functions in numpy

arr1=np.array([[1,2,3,4,5],[6,5,7,8,9]])

print(arr1)
print("\n")

rowsum=arr1.sum(axis=1) # row wise

print("Row sum:",rowsum)
print("\n")

colsum=arr1.sum(axis=0)
print("Col sum:",colsum)

print()
print("Matrix dimension:",arr1.shape)
print("max value:",arr1.max())
print("min value:",arr1.min())
```

```
print("max value at index:",arr1.argmax())
print("min value at index:",arr1.argmin())
print("mean value:",arr1.mean())
print("median value:",np.median(arr1[0]))
```

```
[[1 2 3 4 5]
 [6 5 7 8 9]]


Row sum: [15 35]


Col sum: [ 7  7 10 12 14]

Matrix dimension: (2, 5)
max value: 9
min value: 1
max value at index: 9
min value at index: 0
mean value: 5.0
median value: 3.0
```

[26]:
```
# Sort Using 'mergesort'

arr = np.array([6, 1, 4, 2, 18, 9, 3, 4, 2, 8, 11])

sortedArr0 = np.sort(arr, kind='mergesort')
print("Mergesort:",sortedArr0)
print()
sortedArr1 = np.sort(arr) # default is " Quicksort"
print("Default is (Quicksort):",sortedArr1)
```

```
Mergesort: [ 1  2  2  3  4  4  6  8  9 11 18]

Default is (Quicksort): [ 1  2  2  3  4  4  6  8  9 11 18]
```

[27]:
```
arrs=np.array([[8,5,1,100],[4,2,0,9],[1,6,8,58]])

arrsort0=np.sort(arrs,axis=None) # sort without any axis

arrsort1=np.sort(arrs,axis=0) # Sort along axis 0 i.e. sort contents of each␣
 ↪Column in numpy array

arrsort2=np.sort(arrs,axis=1) #Sort along axis 1 i.e. sort contents of each Row␣
 ↪in numpy array

print("\n Actual 2D array\n",arrs)
print("\n Sort with out axis:",arrsort0)
```

```
print("\n Along first axis (each Column) : \n",arrsort1)
print("\n Along none axis (each Row) : \n",arrsort2)
```

```
Actual 2D array
[[  8   5   1 100]
 [  4   2   0   9]
 [  1   6   8  58]]

Sort with out axis: [  0   1   1   2   4   5   6   8   8   9  58 100]

Along first axis (each Column) :
[[  1   2   0   9]
 [  4   5   1  58]
 [  8   6   8 100]]

Along none axis (each Row) :
[[  1   5   8 100]
 [  0   2   4   9]
 [  1   6   8  58]]
```

3.0.8 - Slicing numpy array elements

[29]:
```
a=np.arange(10)
print(a)
a[1:8:2]
```

```
[0 1 2 3 4 5 6 7 8 9]
```

[29]:
```
array([1, 3, 5, 7])
```

[30]:
```
# We can also combine assignment and slicing
a=np.arange(10)

a[5:]=10

print(a)
```

```
[ 0  1  2  3  4 10 10 10 10 10]
```

[32]:
```
b=np.arange(5)
a[5:]=b[::-1] # start : end : -1 indicates reverse order  :4 (strat:end) ( start:
 ↪ end : step)
print(a)
```

```
[0 1 2 3 4 4 3 2 1 0]
```

[34]:
```
# crop sub array
# Initial Array
```

```
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

print("Initial Array: ")
print(arr)

# Printing a range of Array
# with the use of slicing method

sliced_arr = arr[:2, ::2] # if single : (start : end) , if double :: (strat:end:
 ↪step)

print ("Array with first 2 rows and"
    " alternate columns(0 and 2):\n", sliced_arr)
```

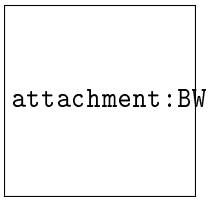
```
Initial Array:
[[-1.   2.   0.   4. ]
 [ 4.  -0.5  6.   0. ]
 [ 2.6  0.   7.   8. ]
 [ 3.  -7.   4.   2. ]]
Array with first 2 rows and alternate columns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]
```

[37]:
```
# reverse using Colon

arr=np.array([[1,2,3,4],[5,6,7,8],[10,11,12,13]])

print("Actual array:\n",arr)

print("\nDisplay using Colon:\n",arr[:,:])

print("\n Row wise reverse:\n",arr[::-1,:]) #arr[row,col]

print("\n column wise reverse:\n",arr[:,::-1]) #arr[row,col]

print("\n Row and column wise reverse:\n",arr[::-1,::-1]) #arr[row,col]
```

```
Actual array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [10 11 12 13]]

Display using Colon:
 [[ 1  2  3  4]
 [ 5  6  7  8]
```

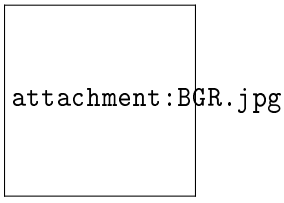
```
[10 11 12 13]]

Row wise reverse:
[[10 11 12 13]
[ 5  6  7  8]
[ 1  2  3  4]]

column wise reverse:
[[ 4  3  2  1]
[ 8  7  6  5]
[13 12 11 10]]

Row and column wise reverse:
[[13 12 11 10]
[ 8  7  6  5]
[ 4  3  2  1]]
```

4 *Introduction to image processing*



1. Black&White Image

2. Gray Image



3. Color Image

[22]:
```
import cv2
image3=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\smallimage.jpg",0)

cv2.imshow("small image",image3)

print(image3)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
[[255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]
 [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]
 [255 255 255 255 255   0   0   0   0   0 255 255 255 255 255]
 [255 255 255 255   0   0   0   0   0   0   0 255 255 255 255]
 [255 255 255   0   0   0   0   0   0   0   0   0 255 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255]
 [255 255   0   0   0   0   0   0   0   0   0   0   0 255 255]
 [255 255 255   0   0   0   0   0   0   0   0   0 255 255 255]
 [255 255 255 255   0   0   0   0   0   0   0 255 255 255 255]
 [255 255 255 255 255   0   0   0   0   0 255 255 255 255 255]
 [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]
 [255 255 255 255 255 255 255 255 255 255 255 255 255 255 255]]
```

[21]:
```
import cv2

img=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_gray_256.jpg",0)
image3 = cv2.resize(img, (36,36))
cv2.imshow("Lena image(256x256)",img)
cv2.waitKey(0)
cv2.imshow("Lena image(16x16)",image3)
cv2.waitKey(0)
print(image3)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
[[159 157 156 ... 123 125 125]
 [156 155 165 ... 127 109  55]
 [158 163 166 ... 108  54  52]
 ...
 [ 76 183 125 ...  99  98  66]
 [ 98 178 119 ...  62 106  74]
 [ 51 200 119 ... 122  76  57]]
```

4.0.1 Write a python program to read, display and save an image. (cv2.imread(), cv2.imshow(), and cv2.imwrite())

[49]:
```
import cv2
image = cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\Veggi.png")
img_gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
print("width: {} pixels".format(image.shape[1]))
print("height: {} pixels".format(image.shape[0]))
print("channels: {}".format(image.shape[2]))

cv2.imwrite("E:\\VNIT Stuff\\CV_W21\\practice Programs\\graytest.png",img_gray)

cv2.imshow("Grayimg",img_gray)
cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
width: 400 pixels
height: 302 pixels
channels: 3
```

4.0.2 Create synthesized white and black image filters and ramp image

[19]:
```
black_mask=np.zeros([512,512,1],dtype='uint8')

white_mask=np.ones([512,512,1],dtype='uint8')*255

ramp=np.zeros([256,256,1],dtype='uint8')


for i in range(1,256):
    for j in range(1,256):
        ramp[i,j]=j-1;

cv2.imshow("whiteimage",white_mask)
cv2.waitKey(0)

cv2.imshow("blackimage",black_mask)
cv2.waitKey(0)

cv2.imshow("Rampimage",ramp)
cv2.waitKey(0)



cv2.imwrite('E:\\VNIT Stuff\\CV_W21\\practice Programs\\whiteimage.
 ↪jpg',white_mask)
```

```
cv2.imwrite('E:\\VNIT Stuff\\CV_W21\\practice Programs\\blackimage.
 ↪jpg',black_mask)
cv2.imwrite('E:\\VNIT Stuff\\CV_W21\\practice Programs\\rampimage.jpg',ramp)


cv2.destroyAllWindows()
```

4.0.3 Display multiple Images together

[20]:
```
# Display multiple Images as stack
img1=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\blackimage.jpg")
img2=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\whiteimage.jpg")
img3=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_gray_256.jpg")
img4=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\rampimage.jpg")

print(img1.shape)
print(img2.shape)

#img1=cv2.resize(img1,(0,0),None,0.5,0.5)
#img2=cv2.resize(img2,(0,0),None,0.5,0.5)

ver=np.vstack((img3,img4))
hor=np.hstack((img1,img2))

cv2.imshow('Verticle',ver)
cv2.waitKey(0)

cv2.imshow('Horizontal',hor)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

```
(512, 512, 3)
(512, 512, 3)
```

4.0.4 Crop sub portion of image.

[23]:
```
image1=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\cameraman.tif')

image2=image1

subimage=image1[50:300,0:450] #; [rowstart:rowend,colstart:colend]


cv2.imshow("main image",image1)
cv2.waitKey(0)
```

```
cv2.imshow("sub image",subimage)
cv2.waitKey(0)


cv2.destroyAllWindows()
```

4.0.5 Image flip

[25]:
```
image1=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_gray_256.jpg')

cv2.imshow("original image",image1)
cv2.waitKey(0)

i2=image1[::-1,:]
cv2.imshow("Vertical filp image",image1[::-1,:]) # reverse the row in this image
cv2.waitKey(0)

i3=image1[:,::-1]
cv2.imshow("Horizontal filp image",image1[:,::-1]) # reverse the coloumns in␣
 ↪this image
cv2.waitKey(0)

i4=image1[::-1,::-1]
cv2.imshow("Horizontal & Vertical filp image",image1[::-1,::-1]) # reverse the␣
 ↪rows and coloumns in this image
cv2.waitKey(0)

hor1=np.hstack((image1,i2))
hor2=np.hstack((i3,i4))

ver=np.vstack((hor1,hor2))

cv2.imshow('Verticle',ver)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

4.0.6 Display image properties

[26]:
```
image2=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_color_256.
 ↪tif')

image3=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_gray_256.
 ↪jpg',0)

print("Gray Image")
```

```
print(type(image3))
print(image3.shape)
print(image3.size)
print(image3.dtype)
height, width = image3.shape[:2]
print(height,width)

print("------------------------------")
print("Color Image")
print(type(image2))
print(image2.shape)
print(image2.size)
print(image2.dtype)
height, width = image2.shape[:2]
print(height,width)
```

```
Gray Image
<class 'numpy.ndarray'>
(256, 256)
65536
uint8
256 256
------------------------------
Color Image
<class 'numpy.ndarray'>
(256, 256, 3)
196608
uint8
256 256
```

4.0.7 Image color channels and conversion

[27]:
```
import cv2
import numpy as np

image2=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_color_256.
 ↪tif')
print(image2.shape)

zeros=np.zeros(image2.shape[:2],dtype="uint8")

# direct method in OpenCV is b,g,r=cv2.spilt()
red = image2[:,:,2]
green = image2[:,:,1]
blue = image2[:,:,0]

cv2.imshow("Oringinal image",image2)
```

```
cv2.waitKey(0)
cv2.imshow("Blue image",cv2.merge([blue,zeros,zeros]))
cv2.waitKey(0)
cv2.imshow("Green image",cv2.merge([zeros,green,zeros]))
cv2.waitKey(0)
cv2.imshow("Red image",cv2.merge([zeros,zeros,red]))

cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
(256, 256, 3)
```

4.0.8 Negative image

[28]:
```
image3=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_gray_256.jpg')
image5=image3.copy()

image5[::1]=abs(255-image3[::1])

cv2.imshow("gray image",image3)
cv2.waitKey(0)
cv2.imshow("gray image nagative",image5)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

4.0.9 Image padding

[29]:
```
# Image padding
import cv2
import numpy as np

img1=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_color_256.tif')

constant= cv2.copyMakeBorder(img1,10,10,10,10,cv2.BORDER_CONSTANT,value=0)

cv2.imshow(" Pad image",constant)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.0.10 Image Scalar Opertions

[31]:
```
s1_img=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_gray_256.jpg')
img1=cv2.cvtColor(s1_img,cv2.COLOR_BGR2GRAY)

scalaradd=img1+25;
```

```
scalarsub=img1-25;
scalarmul=img1*0;
scalardiv=img1//2;

cv2.imshow("Original image",img1)
cv2.waitKey(0)
cv2.imshow("scalar image addition",scalaradd)
cv2.waitKey(0)
cv2.imshow("scalar image sub",scalarsub)
cv2.waitKey(0)
cv2.imshow("scalar image mul",scalarmul)
cv2.waitKey(0)
cv2.imshow("scalar image division",scalardiv)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

4.0.11 Image Opertions

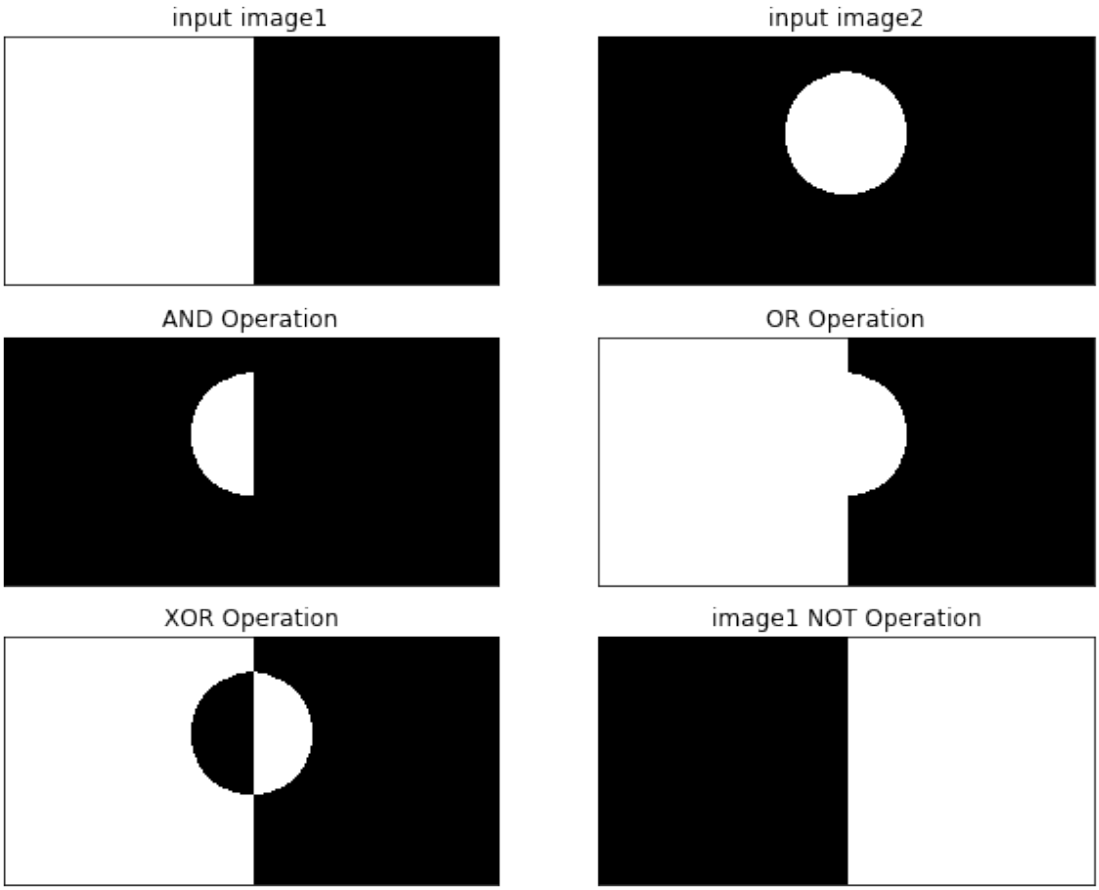
[32]:
```
s1_img=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_color_512.
 ↪tif',0)
s2_img=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\Myimage1111.jpg',0)

#image addition
addimg=cv2.add(s1_img,s2_img)

#image Substration
imagesubtract=cv2.subtract(s1_img,s2_img)

#image Multiplication
imagemulti=cv2.multiply(s1_img,s2_img)

#image Division
imagediv=cv2.divide(s1_img,s2_img)

cv2.imshow("image1",s1_img)
cv2.waitKey(0)
cv2.imshow("image2",s2_img)
cv2.waitKey(0)
cv2.imshow("added image",addimg)
cv2.waitKey(0)

cv2.imshow("cv2 subtration image",imagesubtract)
cv2.waitKey(0)

cv2.imshow("cv2 multiplication image",imagemulti)
cv2.waitKey(0)
```

```
cv2.imshow("cv2 Division image",imagediv)
cv2.waitKey(0)


cv2.destroyAllWindows()
```

4.0.12 Bitwise Operations on images

[35]:
```
import matplotlib.pyplot as plt



def convert_rgb(img):
    imgg=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    return imgg

img1=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\1bit1.png",0)
ret,img1 = cv2.threshold(img1,127,255,cv2.THRESH_BINARY)

img2=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\2bit2.png",0)
ret,img2 = cv2.threshold(img2,127,255,cv2.THRESH_BINARY)

and_img= cv2.bitwise_and(img2, img1, mask = None)
or_img= cv2.bitwise_or(img2, img1, mask = None)
xor_img=cv2.bitwise_xor(img1, img2, mask = None)
not_img1=cv2.bitwise_not(img1, mask = None)

and_img=convert_rgb(and_img)
or_img=convert_rgb(or_img)
xor_img=convert_rgb(xor_img)
not_img1=convert_rgb(not_img1)

plt.figure(figsize=(10,8))
plt.subplot(3,2,1)
img1=convert_rgb(img1)
plt.imshow(img1)
plt.title('input image1')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,2)
img2=convert_rgb(img2)
plt.imshow(img2)
plt.title('input image2')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,3)
```

```
plt.imshow(and_img)
plt.title('AND Operation')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,4)
plt.imshow(or_img)
plt.title('OR Operation')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,5)
plt.imshow(xor_img)
plt.title('XOR Operation')
plt.xticks([])
plt.yticks([])
plt.subplot(3,2,6)
plt.imshow(not_img1)
plt.title('image1 NOT Operation')
plt.xticks([])
plt.yticks([])
plt.show()
```



4.0.13 - Image Geometric transformations

Translation

[67]:
```
# Image Geometrical Operations # Traslate


image=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\jeep.jpg')

def translate(image,x,y): # shifting
    trans_matrix=np.float32([[1,0,x],[0,1,y]])
    shifted=cv2.warpAffine(image,trans_matrix,(image.shape[1],image.shape[0]))
    return shifted;

#Image Translate
trans1=translate(image,0,50)
trans2=translate(image,0,-50)
trans3=translate(image,50,0)
trans4=translate(image,-50,0)

cv2.imshow("original image",image)
cv2.waitKey(0)
cv2.imshow("Down shift image",trans1)
cv2.waitKey(0)
cv2.imshow("Top shift image",trans2)
cv2.waitKey(0)
cv2.imshow("right shift image",trans3)
cv2.waitKey(0)
cv2.imshow("left shift image",trans4)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Resize

[68]:
```
# Image Geometrical Operations # resize

image=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\jeep.jpg')


def resize(image,width=None,height=None):
    (h, w) = image.shape[:2]
    if width is None and height is None:
        return image
    if width is None:
        r = height / float(h)
        dim = (int(w * r), height)
```

```
    else:
        r = width / float(w)
        dim = (width, int(h * r))

    resized=cv2.resize(image,dim,interpolation=cv2.INTER_AREA)
    # interpolation method, which is the algorithm working behind the scenes to␣
 ↪handle how the actual image is resized.
    #   cv2.INTER_LINEAR, cv2.INTER_CUBIC, and cv2.INTER_NEAREST.
    return resized;

# Image resize
resizeimg1=resize(image,width=250,height=250)
resizeimg2=resize(image,width=800,height=800)
cv2.imshow("Original Image",image)
cv2.waitKey(0)
cv2.imshow("Resized Image1:",resizeimg1)
cv2.waitKey(0)
cv2.imshow("Resized Image2:",resizeimg2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Rotation

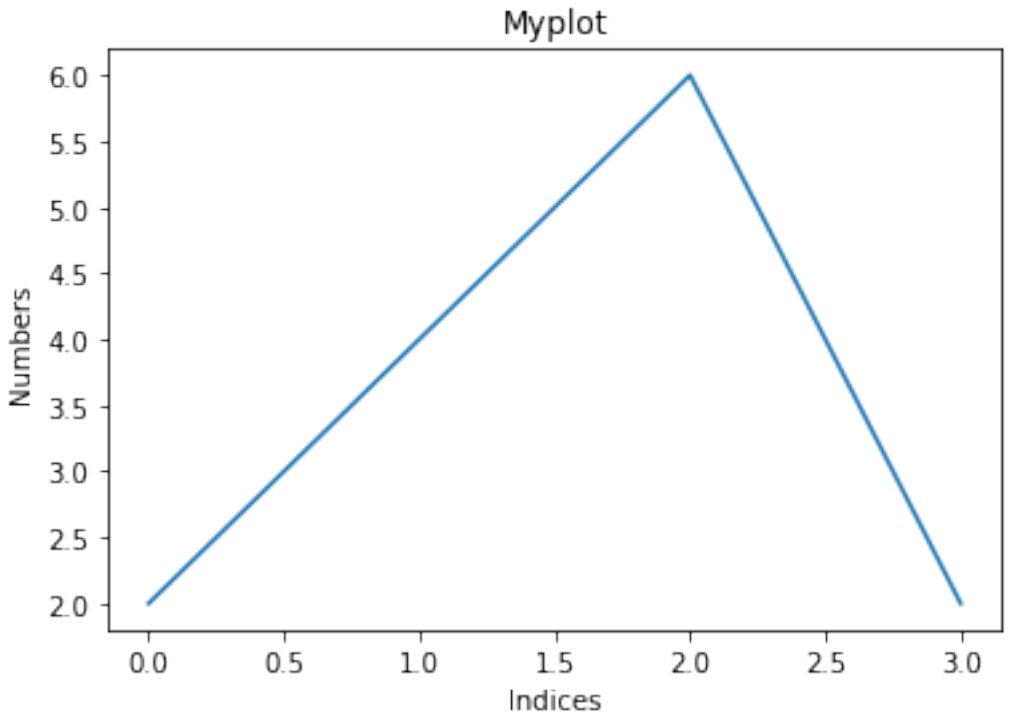
[75]:
```
# Image Geometrical Operations rotation
import cv2

image=cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\jeep.jpg')

def rotate(image,angle,scale):
    (w,h)=image.shape[:2]

    rotate_m=cv2.getRotationMatrix2D((w//2,h//2),angle,scale) # (ratation␣
 ↪point,angle,
    rotated=cv2.warpAffine(image,rotate_m,(w,h))

    return rotated;

# Image Rotation
j=0;
for i in range(0,365,30):
    rotate1=rotate(image,-i,1.0) # +ve i Anti-clockwise
    cv2.imshow("Rotated images",rotate1)
    cv2.waitKey(0)
    if j==12:
        break;
    j+=1;

cv2.destroyAllWindows()
```

4.0.14 Drawing Commands

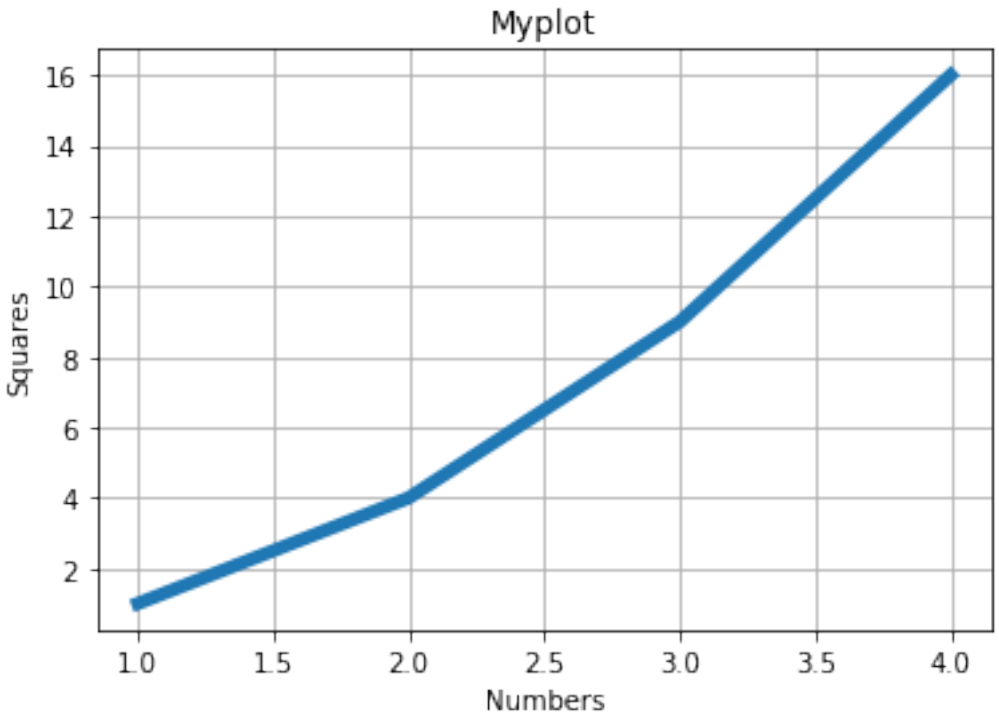
[76]:
```
# Draw Shapes on image

sys_img=np.ones([512,512,3],dtype='uint8')*255

cv2.line(sys_img,(0, 0), (512, 512), (0,255,0),5)  #color channel (b,g,r)
#cv2.line(sys_img,(512, 0), (0, 512), (255,0,0),5)
cv2.rectangle(sys_img,(0, 0),(512, 512),(0,0,255),5)
for r in range(0,100,10):
    cv2.circle(sys_img, (256, 100), r, (0,0,0),2)
cv2.circle(sys_img, (256, 400),80,(0,255,0),-1)
cv2.rectangle(sys_img,(30,200),(150,300),(0,0,255),-1)
cv2.rectangle(sys_img,(330,200 ),(450, 300),(255,0,0),-1)

#cv2.imwrite("output file path",sys_img)

cv2.imshow("fig",sys_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.1 - Introduction to Matplotlib.pyplot

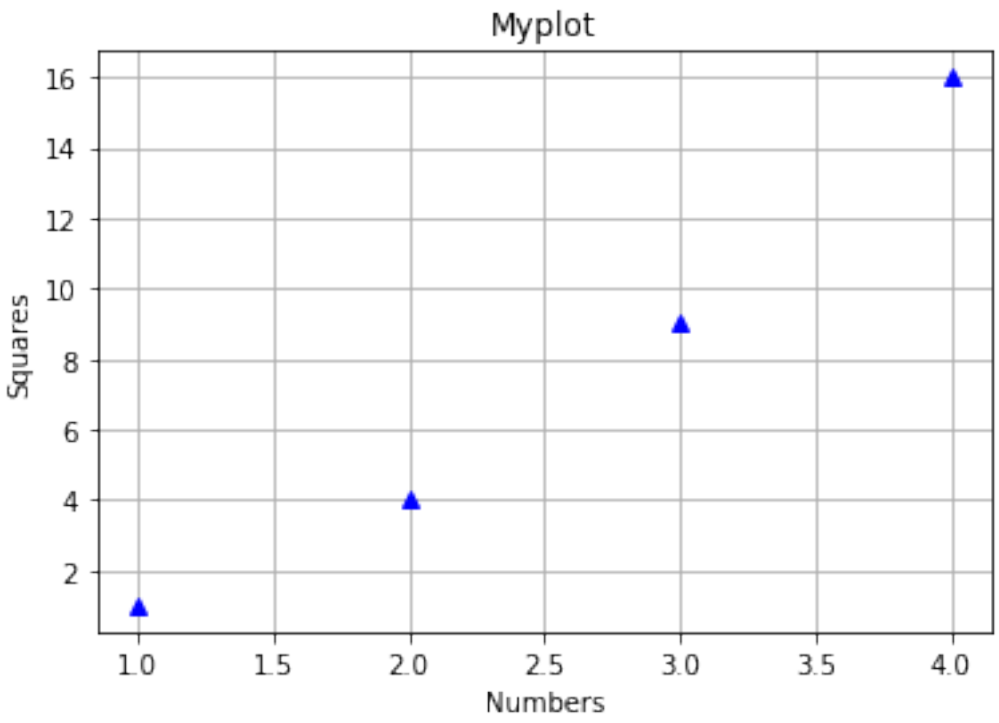
[77]:
```
import matplotlib.pyplot as plt

plt.plot([2,4,6,2])
plt.ylabel("Numbers")
plt.xlabel("Indices")
plt.title("Myplot")
plt.show()
```



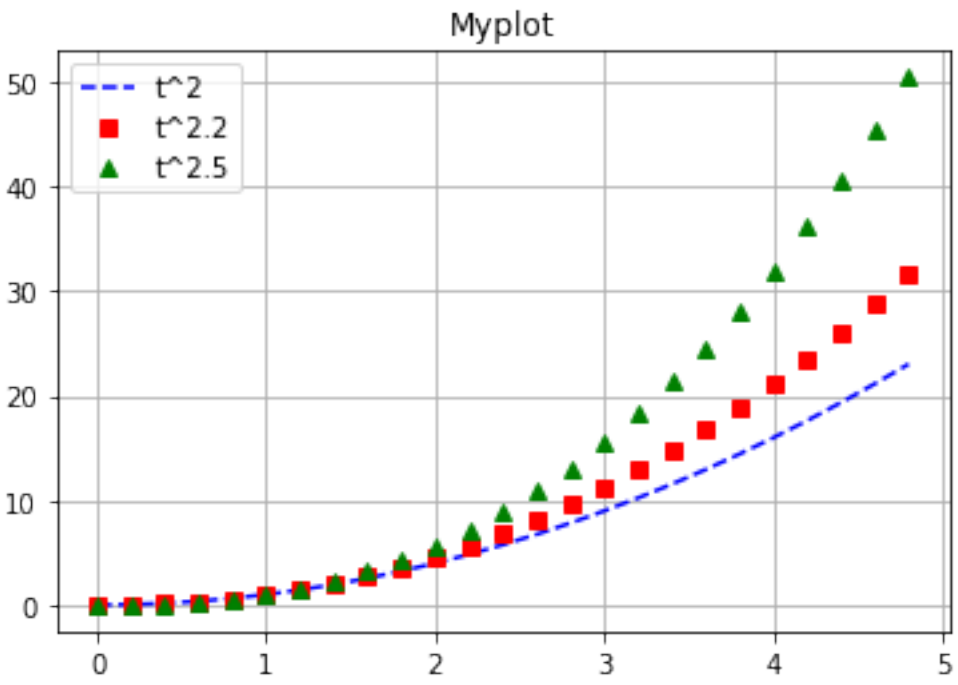
[78]:
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4],[1,4,9,16],linewidth=5.0)
plt.xlabel("Numbers")
plt.ylabel("Squares")
plt.title("Myplot")
plt.grid()
plt.show()
```



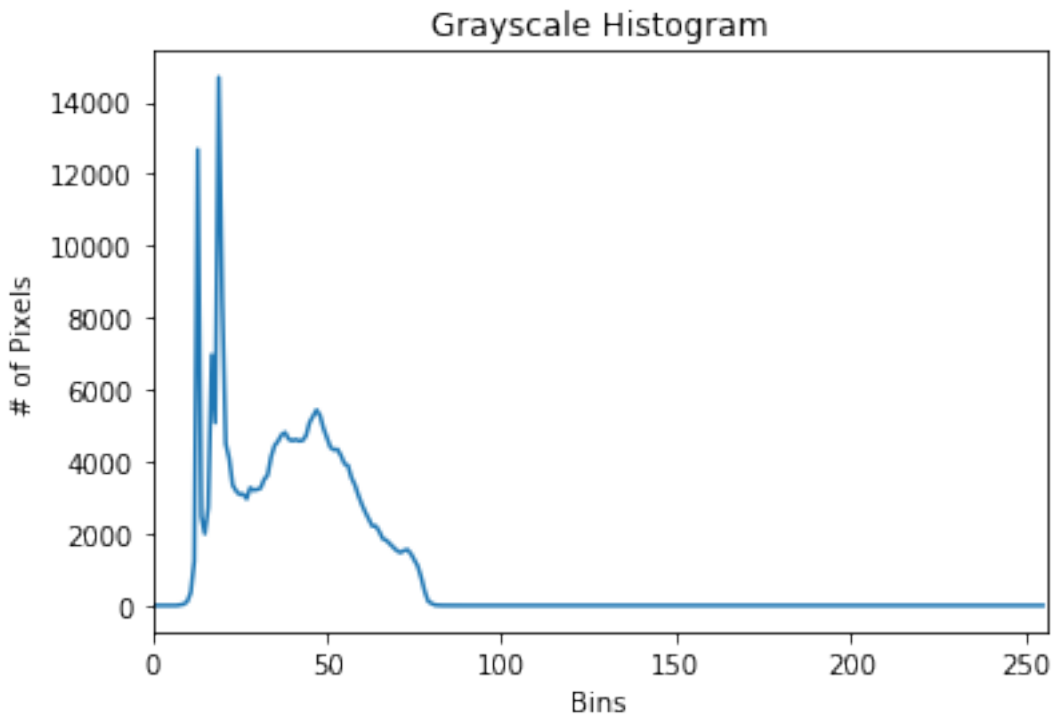
[79]:
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4],[1,4,9,16],'b^')
plt.xlabel("Numbers")
plt.ylabel("Squares")
plt.title("Myplot")
plt.grid()
plt.show()
```



[80]:
```
import matplotlib.pyplot as plt

t=np.arange(0.,5.,0.2)

plt.plot(t,t**2,'b--',label='t^2')
plt.plot(t,t**2.2,'rs',label='t^2.2')
plt.plot(t,t**2.5,'g^',label='t^2.5')

plt.title("Myplot")
plt.grid()
plt.legend()
plt.show()
```

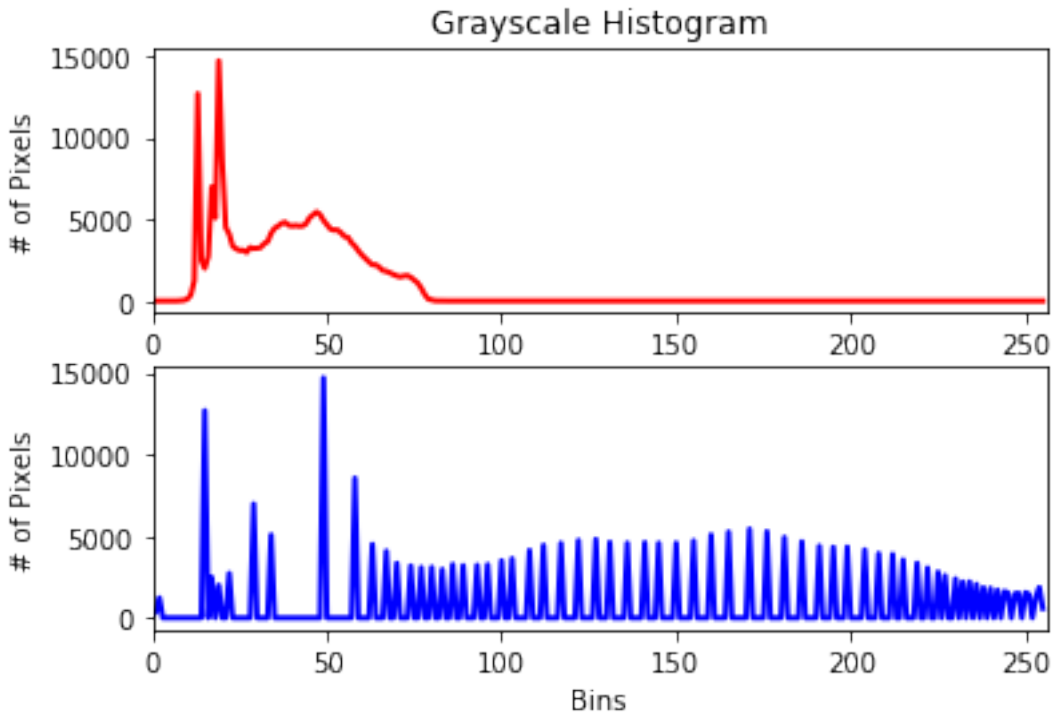


[38]:
```
import cv2
from matplotlib import pyplot as plt

image = cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\123456.tiff')

cv2.imshow("Original", image)


hist = cv2.calcHist([image], [0], None, [256], [0, 256])


plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist)
plt.xlim([0, 256])
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```



[39]:
```
import cv2
from matplotlib import pyplot as plt


image = cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\123456.tiff',0)

histimg=cv2.equalizeHist(image)

#cv2.calcHist(images,channels,mask,histSize,ranges)

hist1 = cv2.calcHist([image], [0], None, [256], [0, 256])

hist2 = cv2.calcHist([histimg], [0], None, [256], [0, 256])

cv2.imshow("Original image", image)
cv2.waitKey(0)
cv2.imshow("Histrogram equalized image ", histimg)
cv2.waitKey(0)

plt.figure(1)
plt.subplot(211)
plt.title("Grayscale Histogram")
plt.xlabel("Bins")
```

```
plt.ylabel("# of Pixels")
plt.plot(hist1,'r',linewidth=2)
plt.xlim([0, 256])

plt.subplot(212)
#plt.title("Grayscale Histogram Equalize")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist2,'b',linewidth=2)
plt.xlim([0, 256])
plt.show()

cv2.waitKey(0)
cv2.destroyAllWindows()
```



5 *Practice programs on basic image processing oprations using OpenCV libraries.*

[11]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from IPython.core.display import Image
```

```
%matplotlib inline
```

5.0.1 1. Write a python program to play a video file in mirror image using opencv.

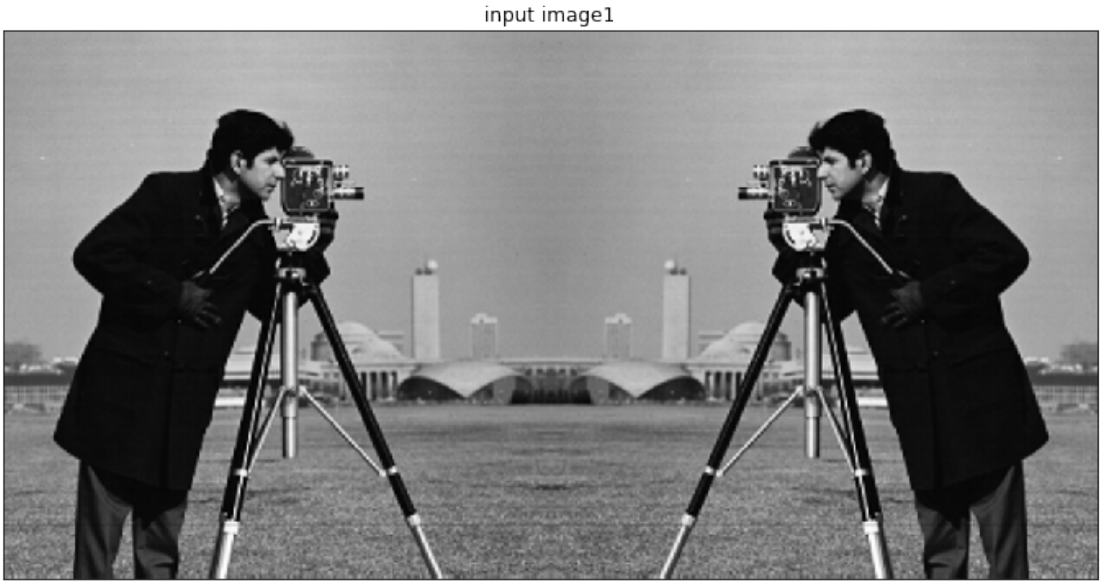
[85]:
```
# importing libraries
import cv2
import numpy as np

# Create a VideoCapture object and read from input file
cap = cv2.VideoCapture(0)  # 0 for Webcam or 1 for external cam or give path of␣
 ↪a video file to play video

# Check if camera opened successfully
if (cap.isOpened()== False):
    print("Error opening video  file")
print("Press Q for stop")
# Read until video is completed
while(cap.isOpened()):
    # Capture frame-by-frame
    ret, frame = cap.read()
    if ret == True:

        mirrorframe=frame[:,::-1]
        # Display the resulting frame
        cv2.imshow('Frame', frame)
        cv2.imshow('Mirror Frame', mirrorframe)

        # Press Q on keyboard to  exit
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    # Break the loop
    else:
        break

# When everything done, release
# the video capture object
cap.release()

# Closes all the frames
cv2.destroyAllWindows()
```

```
Press Q for stop
```

5.0.2 2Q. Write a program to create below display image. Use cameraman.tif image.

[41]:
```
import cv2
import matplotlib.pyplot as plt

img1=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\cameraman_flip.tif")
#Your Code Starts here

#read cameraman.tif image

# apply appropriate operation to create resultant image and display it


#Your code ends here

plt.figure(figsize=(12,8))
plt.subplot(1,1,1)
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
plt.imshow(img1)
plt.title('input image1')
plt.xticks([])
plt.yticks([])
```

[41]:
```
([], <a list of 0 Text yticklabel objects>)
```



5.0.3 3Q. Write a function BitQuantizeImage which takes an 8-bit image im and k, the number of bits to which the image needs to be quantized to and returns the k-bit quantized image.

[45]:
```
def BitQuantizeImage(im, k):
    # n = 8
    # Error checking part
    if k >= 8 or k <= 0:
        print ("Error detected: invalid k value")
        return

    img = np.zeros(im.shape, dtype=np.uint8)

    for row in range(im.shape[0]):
        for col in range(im.shape[1]):
            if len(im.shape) == 2:
                temp = int(im[row][col])
                for i in range(2**k):
                    if (temp >= 2**(8-k) * i) and (temp < 2**(8-k) * (i+1)):
                        img[row][col] = int(2**(8-k-1) * (2 * i + 1))
                        break
            else:
                for chan in range(im.shape[2]):
                    #size of new partition = 2 ^ (8-k)
                    #no of new partitions = 2^k
                    temp = int(im[row][col][chan])
                    for i in range(2**k):
                        if (temp >= 2**(8-k) * i) and (temp < 2**(8-k) * (i+1)):
                            img[row][col][chan] = int(2**(8-k-1) * (2 * i + 1))
                            break
    return img

im3 = cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\lena_gray_256.jpg")
bits=int(input("Enter number of bits:"))
img3 = BitQuantizeImage(im3,bits)
cv2.imshow("fig",img3)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Enter number of bits:3
```

5.0.4 4Q. You are given images with unknown gamma correction parameters. With the help of below given power-law transformation program choose appropriate gamma for each image. Avaliable gamma values are 0.1,0.5,1,1.5,2.

1. 

5.0.5 2. 

5.0.6 3. 

5.0.7 4. 

5.0.8 5. 

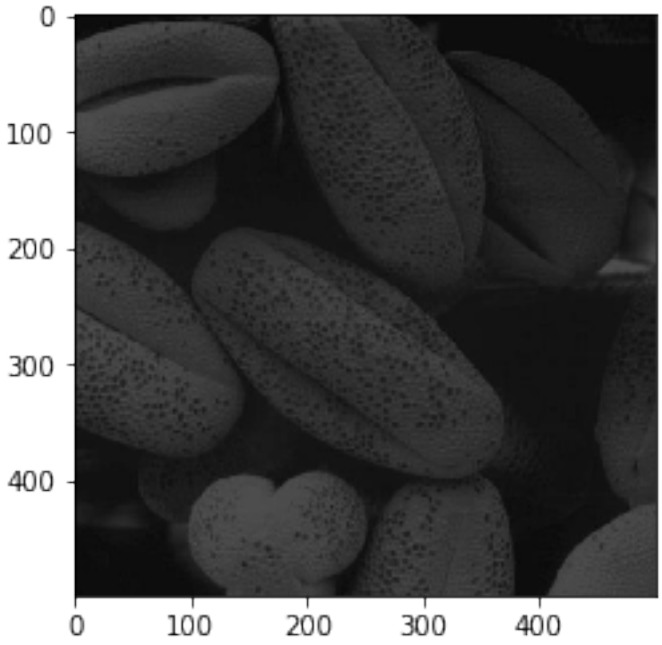
[99]:
```
#Power-law transformation program

def gamma_correction(img,gamma):
    gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')
    return gamma_corrected

in_img1=cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\img1.jpg") # read␣
 ↪original image

gamma=float(input("Enter the gamma value: ")) #enter the gamma value and match␣
 ↪the appropriate gamma value to corresponding image

corredted_img=gamma_correction(in_img1,gamma) # call gamma_correction function
```

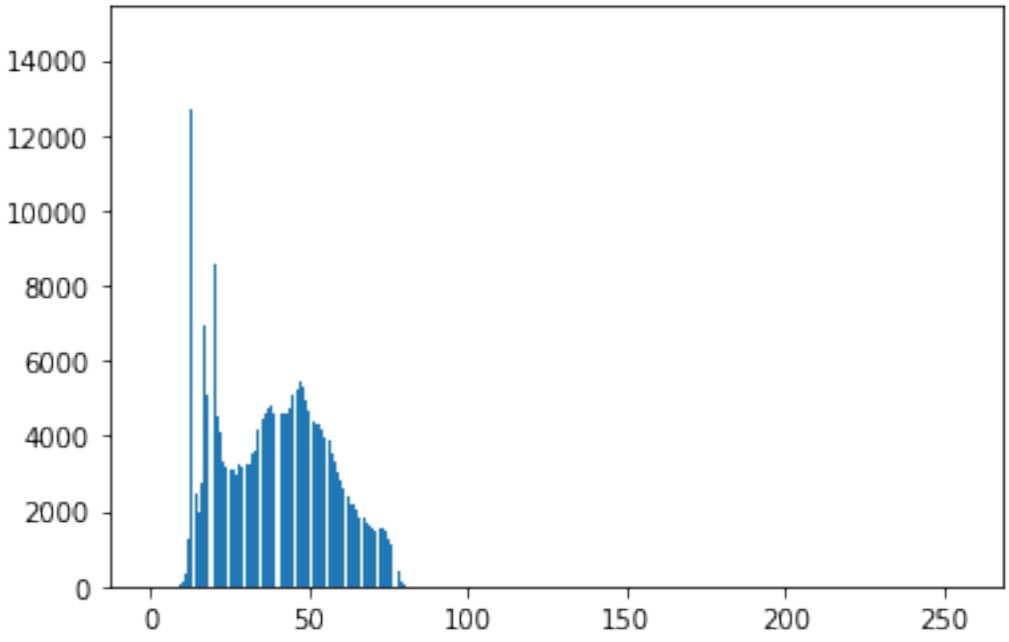
```
cv2.imshow("figure",corredted_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Enter the gamma value: 0.1
```

5.0.9 5Q. Write a function plotHistogram(image) to generate histogram of a gray image. Using this function display histograms for the images histogram1.jpg, histogram2.jpg and histogram3.jpg. notedone your observations.

[46]:
```
def plotHistogram(im) :
    if len(im.shape) != 2:
        im = im[:,:,0]

    mapp = [0] * 256
    for row in range(im.shape[0]):
        for col in range(im.shape[1]):
            key = im[row][col]
            mapp[key] += 1

    #pixel_count = sum(mapp)
    #mapp = [(val * 1.0)/pixel_count for val in mapp]
    return mapp

im = cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\123456.tiff")

plt.figure()
plt.imshow(im)

hist=plotHistogram(im)

plt.figure()
plt.bar(range(256),hist)
plt.show()
```


0
100
200
300
400
0
100
200
300
400


14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250

5.0.10 6Q. Write a program in Python to experiment different inbuilt smoothing methods by different filter sizes (3X3,5X5 and 7X7) and observe its blurring effect on the given noise image.

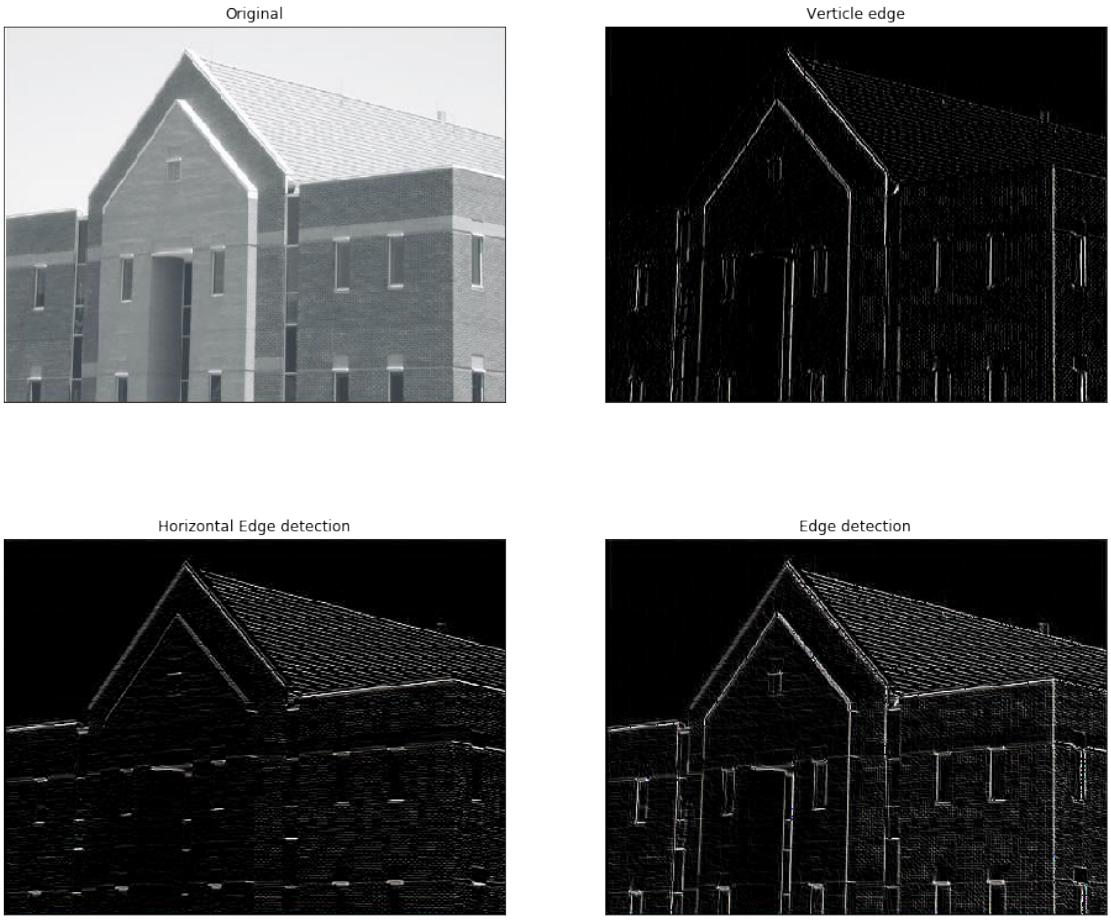
[48]:
```
import cv2
import numpy

# using imread()
img = cv2.imread("E:\\VNIT Stuff\\CV_W21\\practice Programs\\noise_img.tif",0)

# change filter sizes in the send parameter position in the below funtion call.

# simple average filter
dst = cv2.blur(img,(3,3),cv2.BORDER_DEFAULT) # Change filter size(2nd parameter)␣
 ↪and observe the output result

#weighted average filter
dst1 = cv2.GaussianBlur(img,(3,3),cv2.BORDER_DEFAULT) # Change filter size(2nd␣
 ↪parameter) and observe the output result

#median filter
dst2 = cv2.medianBlur(img,3,cv2.BORDER_DEFAULT) # Change filter size(2nd␣
 ↪parameter) and observe the output result

cv2.imshow('inbuilt simple Avg filter Vs Weighted Avg filter Vs Median filter',␣
 ↪numpy.hstack((dst,dst1,dst2)))

cv2.waitKey(0);
cv2.destroyAllWindows();
```

5.0.11 7Q.Write a program to illustrate sobal filters effect using cv2.filter2D function.

[49]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('E:\\VNIT Stuff\\CV_W21\\practice Programs\\Building.jpg')

ker_ver=np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
ker_hor=np.array([[1,2,1],[0,0,0],[-1,-2,-1]])

print(ker_ver)
print(ker_hor)

#kernel = np.ones((5,5),np.float32)/25

ver_img = cv2.filter2D(img,-1,ker_ver)
```

```
#2nd parameter is desired depth of the output image. If it is negative, it will␣
 ↪be the same as that of the input image.
hor_img = cv2.filter2D(img,-1,ker_hor)

final_img=ver_img+hor_img

plt.figure(figsize=(16,14))
plt.subplot(221),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(222),plt.imshow(ver_img),plt.title('Verticle edge')
plt.xticks([]), plt.yticks([])
plt.subplot(223),plt.imshow(hor_img),plt.title('Horizontal Edge detection')
plt.xticks([]), plt.yticks([])
plt.subplot(224),plt.imshow(final_img),plt.title('Edge detection')
plt.xticks([]), plt.yticks([])
plt.show()
cv2.destroyAllWindows()
```

```
[[-1  0  1]
 [-2  0  2]
 [-1  0  1]]
[[ 1  2  1]
 [ 0  0  0]
 [-1 -2 -1]]
```



5.0.12 8Q.Real-Time Edge Detection using OpenCV in Python | Canny edge detection method.

[50]:
```
# OpenCV program to perform Edge detection in real time
# import libraries of python OpenCV

# where its functionality resides
import cv2

# np is an alias pointing to numpy library
import numpy as np

 #capture frames from a camera
cap = cv2.VideoCapture(0)

print("Press q for exit")

 # loop runs if capturing has been initialized
```

```
while(1):

    # reads frames from a camera
    ret, frame = cap.read()

    # converting BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of red color in HSV
    lower_red = np.array([30,150,50])
    upper_red = np.array([255,255,180])

    # create a red HSV colour boundary and
    # threshold HSV image
    mask = cv2.inRange(hsv, lower_red, upper_red)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame,frame, mask= mask)

    # Display an original image
    cv2.imshow('Original',frame)

    # finds edges in the input image image and
    # marks them in the output map edges
    edges = cv2.Canny(frame,100,200)

    # Display edges in a frame
    cv2.imshow('Edges',edges)

    # Wait for Esc key to stop
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break


# Close the window
cap.release()

# De-allocate any associated memory usage
cv2.destroyAllWindows()
```

Press q for exit