Assignment_1

September 26, 2019

1 Assignment 1

In this assignment, we will go through basic image manipulation using python to get everyone on the same page for the prerequisite/necessary skills for this class.

One of the aims of this assignment is to get you to start getting comfortable searching for useful python library functions. So in many of the functions you will implement, you will have to look up helper functions.

[2]:
```
#Imports the print function from newer versions of python
from __future__ import print_function

#Setup

# The Random module for implements pseudo-random number generators
import random

# Numpy is the main package for scientific computing with Python.
# This will be one of our most used libraries in this class
import numpy as np


#Imports all the methods in the file: imageOperations.py
from imageOperations import *

#Matplotlib is a useful plotting library for python
import matplotlib.pyplot as plt
# This code is to make matplotlib figures appear inline in the
# notebook rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

%load_ext autoreload
%autoreload 2
```

1.1 1: Image Manipulation

Let's load some images and treat them as matrices and do some operations on them. By the end of this section, you will have implemented all the methods in `imageOperations.py`

[3]:
```
# Run this code to set the locations of the images we will be using.
# You can change these paths to point to your own images if you want to try␣
 ↪them out for fun.

image1_path = './Images/image1.jpg'
image2_path = './Images/image2.jpg'
def display(img):
    # Show image
    plt.figure(figsize = (5,5))
    plt.imshow(img)
    plt.axis('off')
    plt.show()
```

1.1.1 Question 1.1 (5 points)

Implement the load method in imageOperations.py and read the display method below. We will use these two methods through the rest of the notebook to visualize our work.

[4]:
```
image1 = load(image1_path)
image2 = load(image2_path)

display(image1)
display(image2)
```

1.1.2 Question 1.2 (10 points)

Implement the `dim_image()` method by converting images according to $x_n = 0.5 * x_p^2$ for every pixel, where x_n is the new value and x_p is the original value.

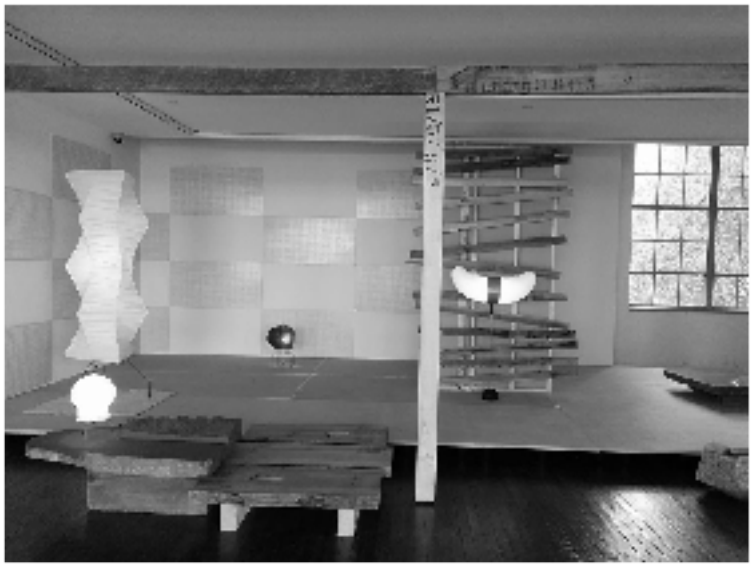
Note: Since all the pixel values of the image are in the range $[0, 1]$, the above formula will result in reducing these pixels values and therefore make the image dimmer.

[5]:
```
new_image = dim_image(image1)
display(new_image)
```

1.1.3 Question 1.3 (10 points)

Implement the convert_to_grey_scale method and convert the image into grey scale.

[6]:
```
grey_image = convert_to_grey_scale(image1)
display(grey_image)
```



1.1.4 Question 1.4 (10 points)

Implement the `rgb_exclusion()`, in which the input image is decomposed into the three channels: R, G and B and return the image excluding the specified channel.

[7]:
```
without_red = rgb_exclusion(image1, 'R')
without_blue = rgb_exclusion(image1, 'B')
without_green = rgb_exclusion(image1, 'G')

print("Below is the image without the red channel.")
display(without_red)

print("Below is the image without the green channel.")
display(without_green)

print("Below is the image without the blue channel.")
display(without_blue)
```

```
Below is the image without the red channel.
```

```
Below is the image without the green channel.
```

Below is the image without the blue channel.

1.1.5 Question 1.5 (10 points)

In mix_images method, create a new image such that the left half of the image is the left half of image1 and the right half of the image is the right half of image2. Exclude the specified channel for the given image.

You should see the left half of the monkey without the red channel and the right half of the house image with no green channel.

[8]:
```
image_mixed = mix_images(image1, image2, channel1='R', channel2='G')
display(image_mixed)
```

```
(600, 500, 3)
(600, 400, 3)
```

1.1.6 Extra credit (10 points)

The following questions are optional and will go towards you extra credit grade.

Implement `mix_quadrants` function in `imageOperations.py`.

[9]:
```
mixed_quadrants = mix_quadrants(image1)
display(mixed_quadrants)
```

1.2 2: Basic Image Processing

In this section you will load images using OpenCV and perform basic operations

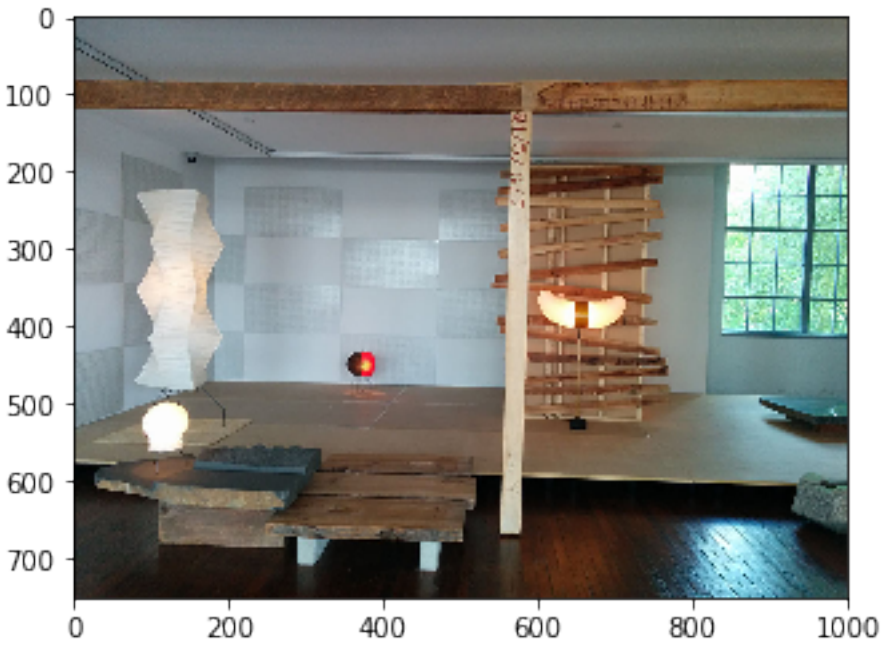
1.2.1 Question 2.1 (0 credit, the solution is givne below)

Read the image using OpenCV, convert image to RGB color for matplotlib, as OpenCV reads in inverted mode BGR, display the image in proper RGB format using matplotlib builtin function

[10]:
```
import matplotlib, cv2, os
import numpy as np
from skimage import color
import matplotlib.pyplot as plt

%matplotlib inline
image1_path = './Images/image1.jpg'
# read an image
img = cv2.imread(image1_path)

# show image format (basically a 3-d array of pixel color info, in BGR format)
#print(img)

# convert image to RGB color for matplotlib, as cv2 reads in inverted mode or␣
 ↪RGB which is BGR
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# show image with matplotlib
```

```
plt.imshow(img)
```

[10]: <matplotlib.image.AxesImage at 0x25e32891ef0>

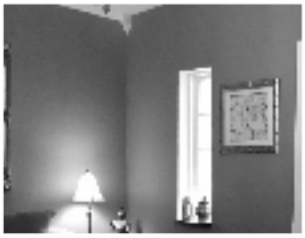


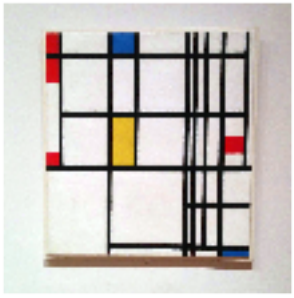
1.2.2 Question 2.2 (5 points)

Loop through all the images in the directory *'./Images/'* and display as thumbnails. Feel free to use any library function

[11]:
```
directory = './Images/'
maxSize = (120,120)
for file in os.listdir(directory):
    if file.endswith('jpg') or file.endswith('png'):
        o_image = load(directory+file)
        thumbNail = cv2.resize(o_image,None,fx=0.2,fy=0.2,interpolation=cv2.
↪INTER_CUBIC)
        plt.figure(figsize = (2,2))
        plt.imshow(thumbNail)
        plt.axis('off')
        plt.show()
```


Snow

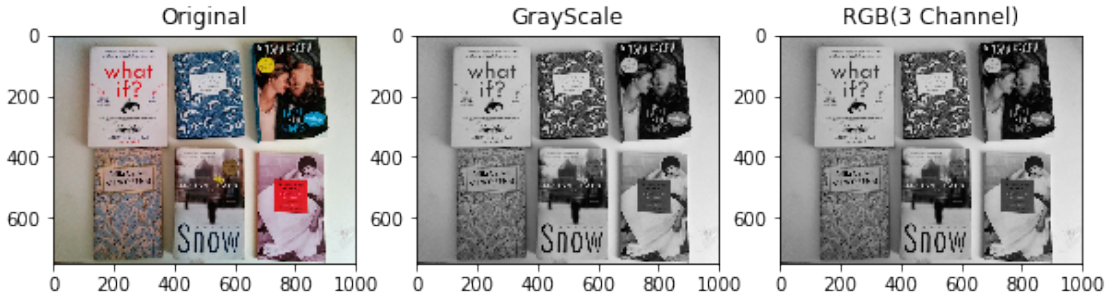


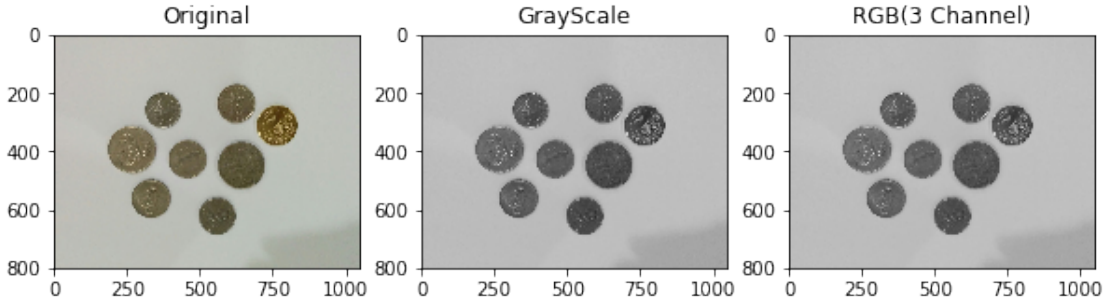


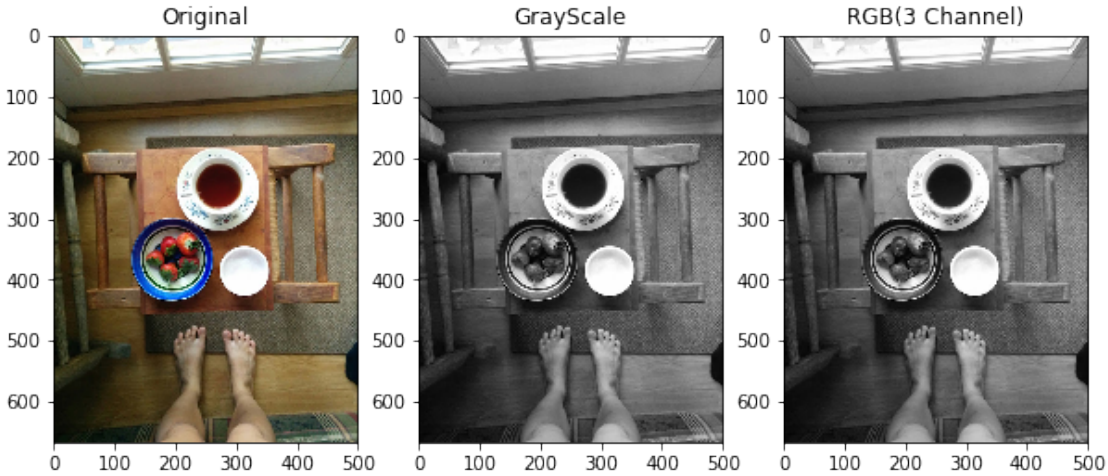
1.2.3 Question 2.3 (5 points)

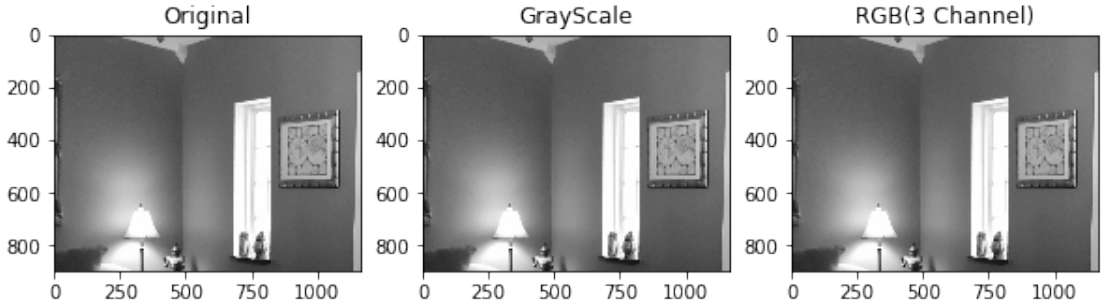
Perform following operations on a couple of images present in './Images/' directory. You can use OpenCV/Scipy/Scikit-image built-in functions - Convert the colored image into grayscale, and display. - Convert the grayscale image back to RGB image, and display.

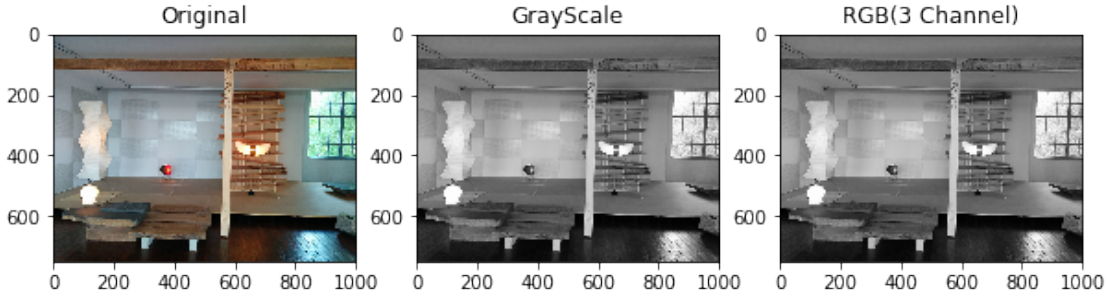
[22]:
```
for f in os.listdir('./Images/'):
    if f.endswith('jpg') or f.endswith('png'):
        img = load('./Images/'+f)
        grayImage = color.rgb2gray(img) #plt.imshow(grayImage, cmap=plt.cm.gray)
        back2RGB = color.gray2rgb(grayImage)
        plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
        fig = plt.figure()
        fig.add_subplot(1, 3, 1)
        plt.imshow(img)
        plt.title("Original")
        fig.add_subplot(1, 3, 2).imshow(grayImage)
        plt.title("GrayScale")
        fig.add_subplot(1, 3, 3).imshow(back2RGB)
        plt.title("RGB"+"("+"3 Channel"+")")
```

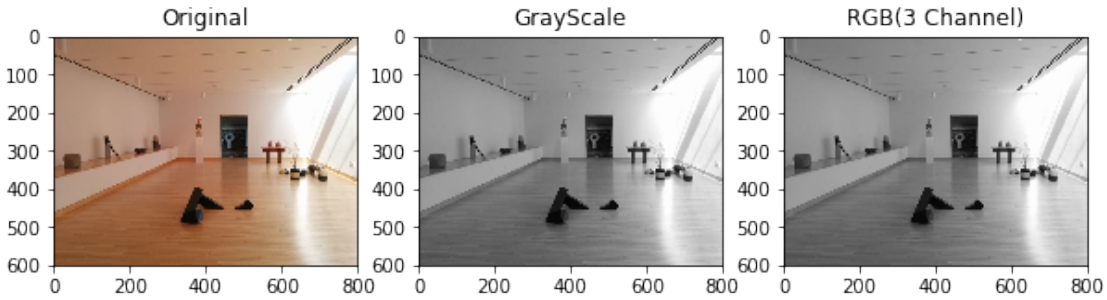


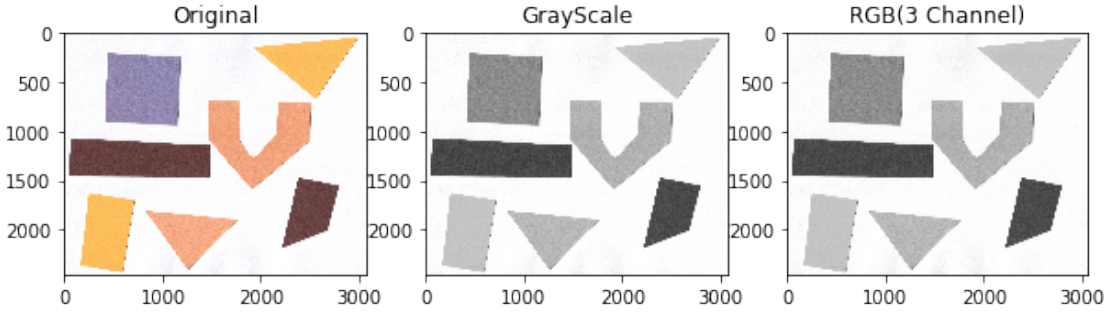


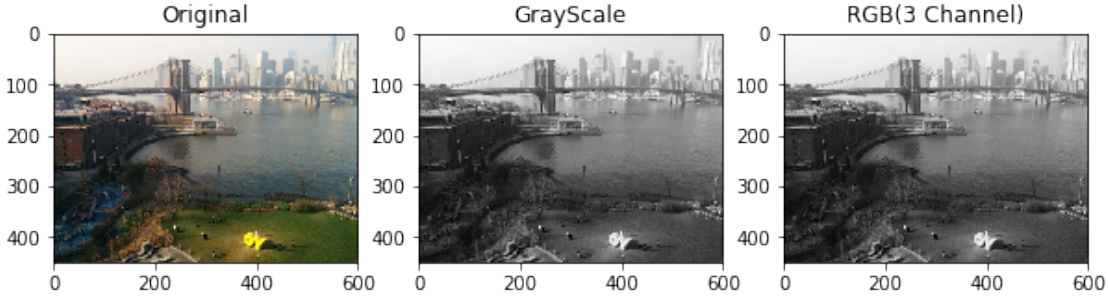

Original
GrayScale
RGB(3 Channel)

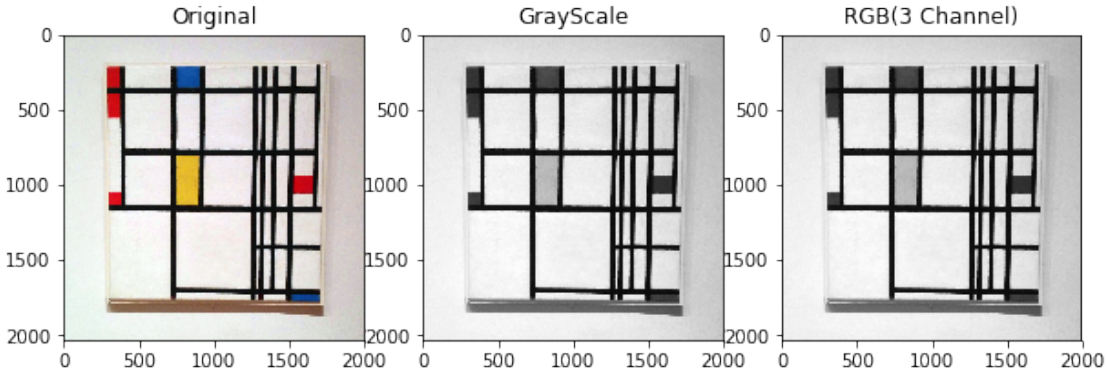

Original
GrayScale
RGB(3 Channel)

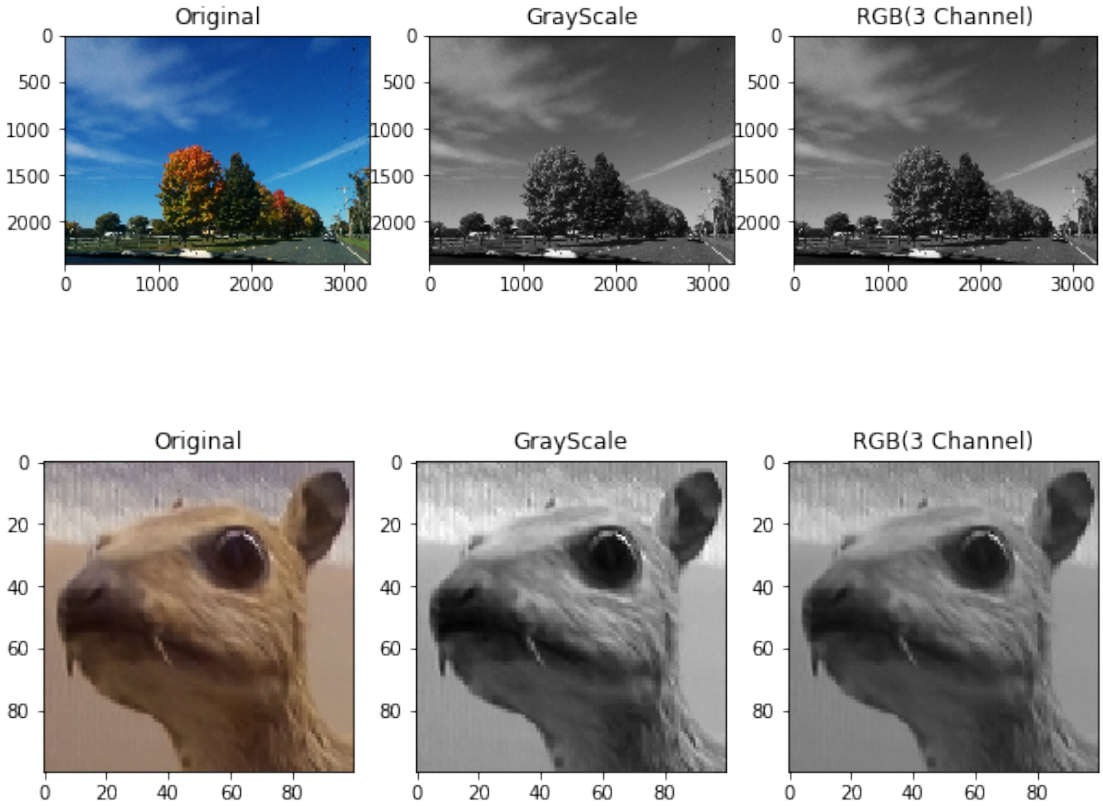

Original
GrayScale
RGB(3 Channel)


Original
GrayScale
RGB(3 Channel)


Original
GrayScale
RGB(3 Channel)


Original
GrayScale
RGB(3 Channel)


Original
GrayScale
RGB(3 Channel)



1.2.4 Question 2.4 (10 points)

Implement the function `myConvolve2d(image, kernel)`, in the file `ImageOperations.py`. In this function you need to implement the convolution operation from scratch. This function which takes an image and a kernel and returns the convolution of them. Necessary padding is provided within the image

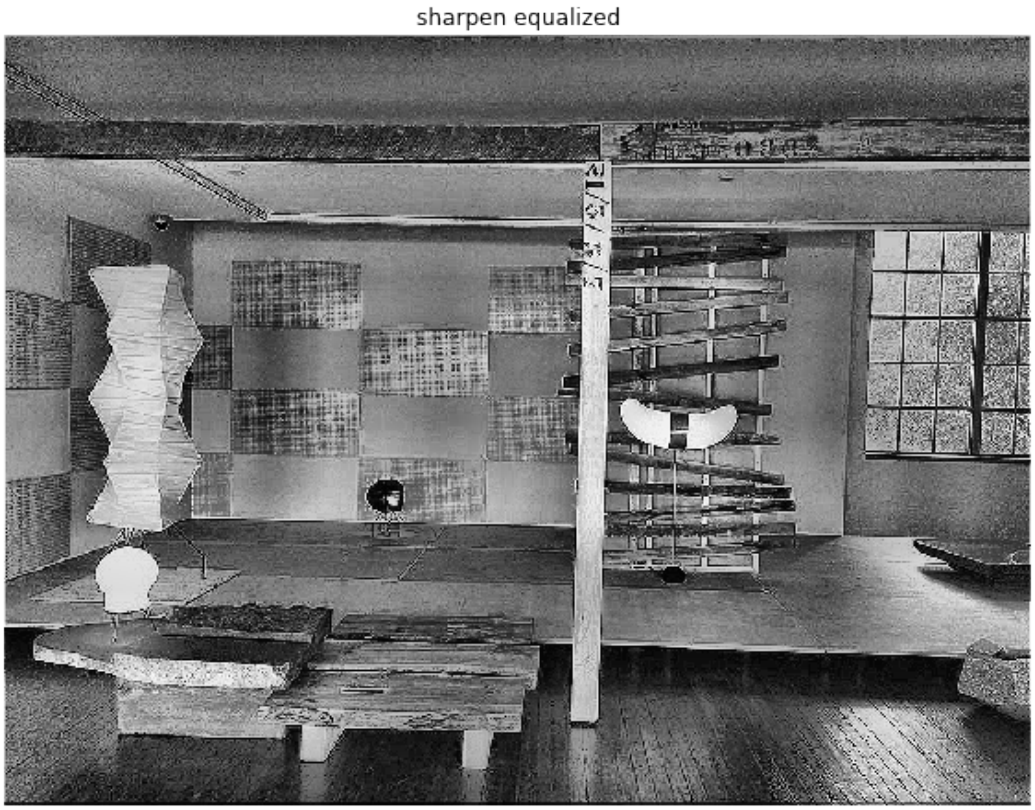
[23]:
```
from skimage import io
from skimage import exposure
from imageOperations import myConvolve2d
import pylab

image1_path = './Images/image1.jpg'

img = io.imread(image1_path)     # Load the image
img = color.rgb2gray(img)        # Convert the image to grayscale (1 channel)
plt.imshow(img, cmap=plt.cm.gray)
plt.axis('off')
plt.title('converted to grayscale')
plt.show()
```

```
# Convolve the sharpen kernel (laplacian) and the image
kernel = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])

#Call the function my_convolve2d from the file imageOperations.py, which you␣
 ↪have to implement.
image_sharpen = myConvolve2d(img,kernel)

print ('\n First 5 columns and rows of the image_sharpen matrix: \n',␣
 ↪image_sharpen[:5,:5]*255)

# Plot the filtered image
plt.imshow(image_sharpen, cmap=plt.cm.gray)
plt.title('Image sharpen')
plt.axis('off')
plt.show()

# Adjust the contrast of the filtered image by applying Histogram Equalization
image_sharpen_equalized = exposure.equalize_adapthist(image_sharpen/np.max(np.
 ↪abs(image_sharpen)), clip_limit=0.03)
plt.imshow(image_sharpen_equalized, cmap=plt.cm.gray)
plt.axis('off')
plt.title('sharpen equalized')
plt.show()
```

converted to grayscale

```
First 5 columns and rows of the image_sharpen matrix:
[[325.2853 238.1902 237.1902 201.1902 200.1902]
[200.1902 120.0951 123.0951 114.0951 108.0951]
[197.1902 106.0951 126.0951 126.0951 117.0951]
[201.1902 104.0951 114.0951 125.0951 114.0951]
[222.1902  99.0951  97.0951 118.0951 117.0951]]
```

Image sharpen



1.2.5 Question 2.5 (5 points)

Load a couple of images from './Images/' directory. Better to load them all in a loop.

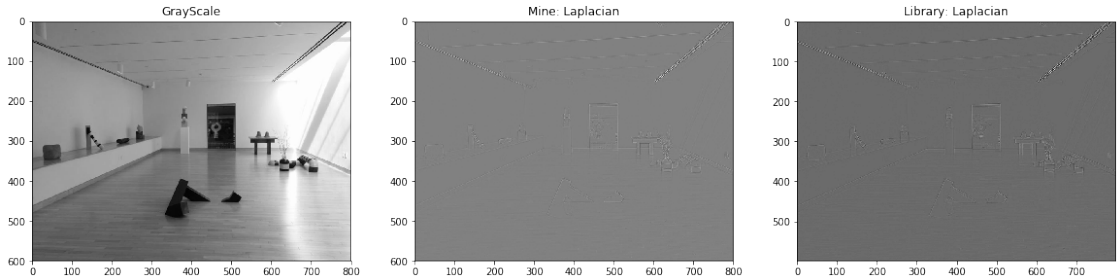
Apply the kernals given in the code snippet below to them by two methods. 1: use your implemented method `my_convolve2d(img, kernel)` which you have implemented in `ImageOperation.py` 2: Use builtin methods for convolution, available in various python packages, which includes scipy method `scipy.signal.convolve2d(img, kernel, 'valid')` , OpenCV method `cv2.filter2D(... , .. , ...)`.

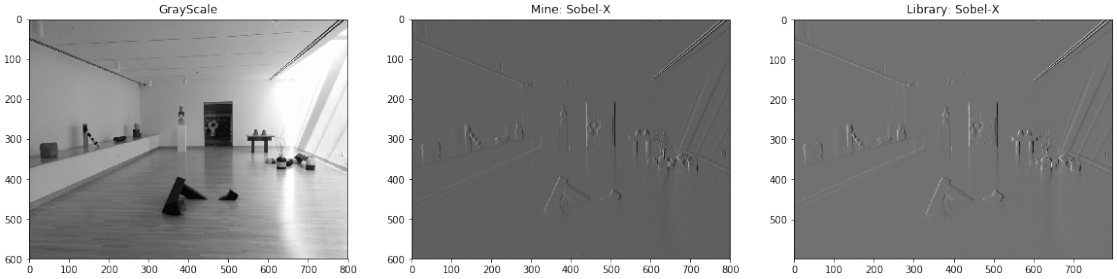
Validate the results obtained by your implementation of `myConvolve2d()`with those obyained by builtin metohds, by displaying them side by side. For that purpose, you can use filter kernels given in the code snipped below

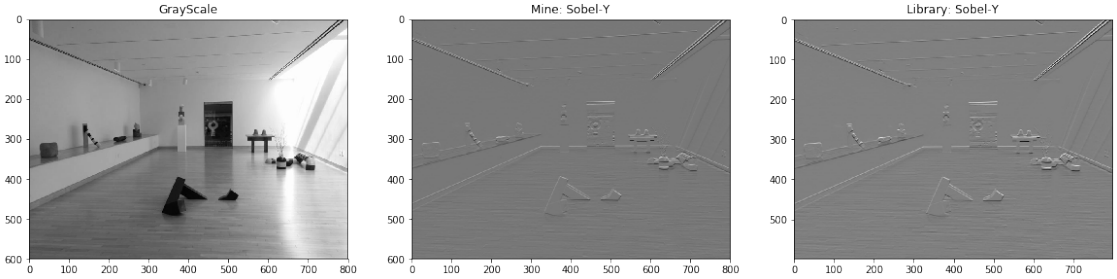
[24]:
```
from scipy import signal

def comparison(img,laplacian,sobelX,sobelY):
    #converting to grayscale
    img = color.rgb2gray(img)
    #Laplcian part
    myLaplacian = myConvolve2d(img,laplacian)
    libLaplacian = signal.convolve2d(img, laplacian, 'valid')
```

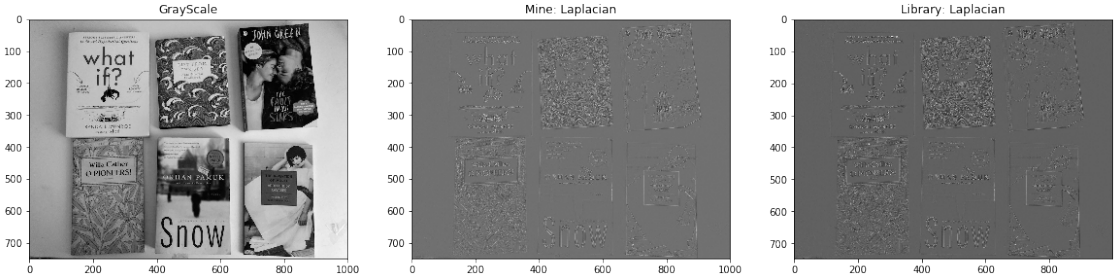
```
    plt.rcParams['figure.figsize'] = (20.0, 16.0) # set default size of plots
    fig = plt.figure()
    fig.add_subplot(1, 3, 1)
    plt.imshow(img)
    plt.title("GrayScale")
    fig.add_subplot(1, 3, 2)
    plt.imshow(myLaplacian)
    plt.title("Mine: Laplacian")
    fig.add_subplot(1, 3, 3).imshow(libLaplacian)
    plt.title("Library: Laplacian")

    #Sobel X part
    mySobelX = myConvolve2d(img,sobelX)
    libSobelX = signal.convolve2d(img, sobelX, 'valid')
    plt.rcParams['figure.figsize'] = (20.0, 16.0) # set default size of plots
    fig = plt.figure()
    fig.add_subplot(2, 3, 1)
    plt.imshow(img)
    plt.title("GrayScale")
    fig.add_subplot(2, 3, 2)
    plt.imshow(mySobelX)
    plt.title("Mine: Sobel-X")
    fig.add_subplot(2, 3, 3).imshow(libSobelX)
    plt.title("Library: Sobel-X")

    #Sobel Y part
    mySobelY = myConvolve2d(img,sobelY)
    libSobelY = signal.convolve2d(img, sobelY, 'valid')
    plt.rcParams['figure.figsize'] = (20.0, 16.0) # set default size of plots
    fig = plt.figure()
    fig.add_subplot(3, 3, 1)
    plt.imshow(img)
    plt.title("GrayScale")
    fig.add_subplot(3, 3, 2)
    plt.imshow(mySobelY)
    plt.title("Mine: Sobel-Y")
    fig.add_subplot(3, 3, 3).imshow(libSobelY)
    plt.title("Library: Sobel-Y")



# Laplacian kernel used to detect edge-like regions of an image
laplacian = np.array((
        [0, 1, 0],
        [1, -4, 1],
        [0, 1, 0]), dtype="int")
```

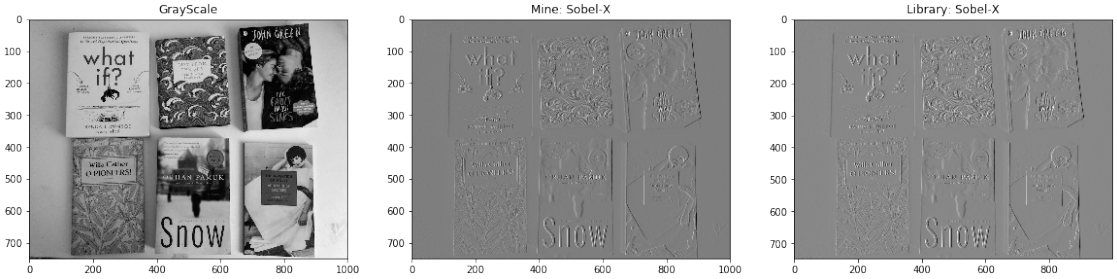
```
# Sobel x-axis kernel
sobelX = np.array((
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]), dtype="int")

# Sobel y-axis kernel
sobelY = np.array((
        [-1, -2, -1],
        [0, 0, 0],
        [1, 2, 1]), dtype="int")

#for f in os.listdir('./Images/'):
    #if f.endswith('jpg') or f.endswith('png'):
img = load('./Images/image2.jpg')
comparison(img,laplacian,sobelX,sobelY)

img = load('./Images/books.jpg')
comparison(img,laplacian,sobelX,sobelY)

img = load('./Images/oy.jpg')
comparison(img,laplacian,sobelX,sobelY)

img = load('./Images/junk.jpg')
comparison(img,laplacian,sobelX,sobelY)

img = load('./Images/Vd-Orig.png')
comparison(img,laplacian,sobelX,sobelY)
```

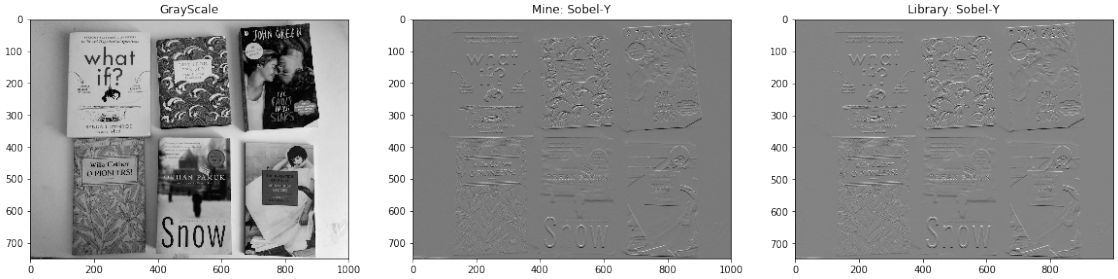


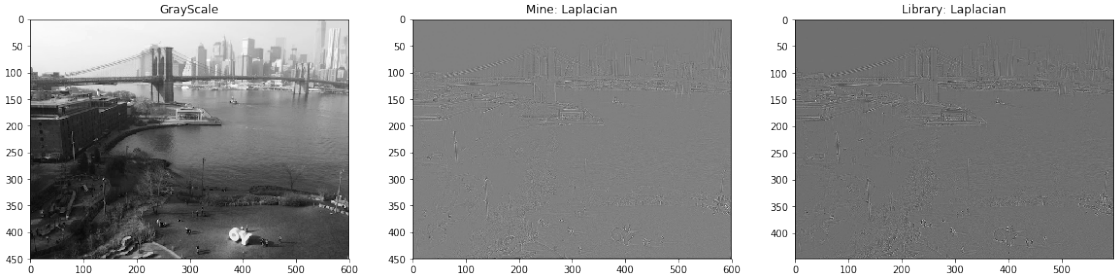

GrayScale
Mine: Sobel-X
Library: Sobel-X

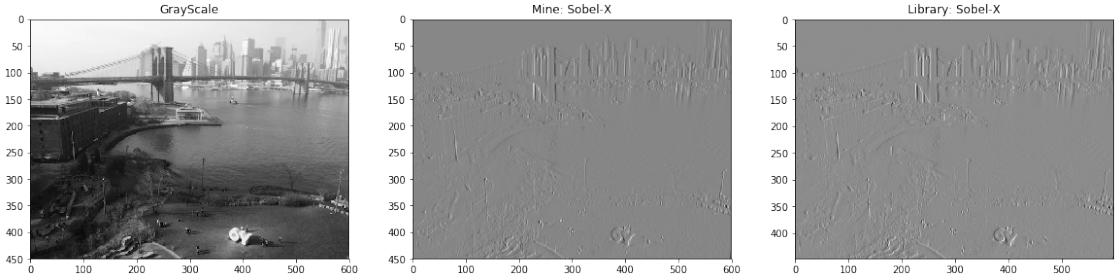

GrayScale
Mine: Sobel-Y
Library: Sobel-Y

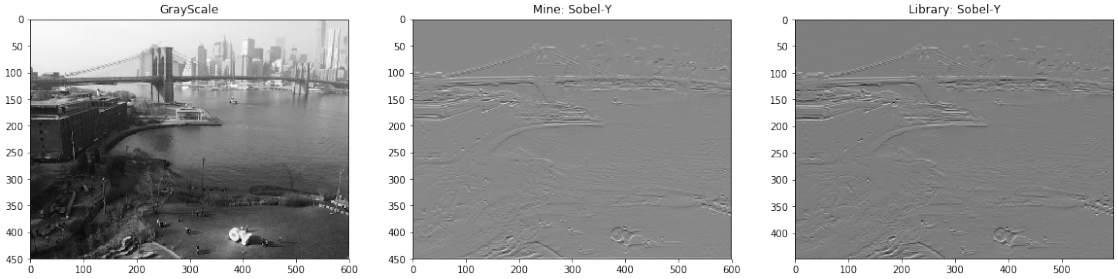

GrayScale
Mine: Laplacian
Library: Laplacian
what if?
Snow

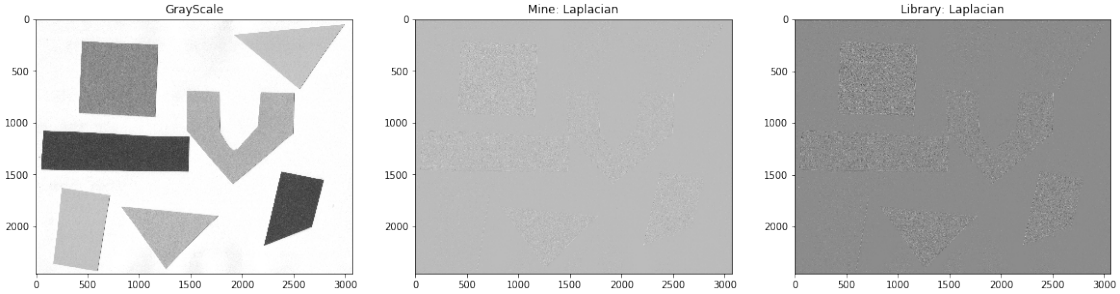

GrayScale
Mine: Sobel-X
Library: Sobel-X
what if?
Snow

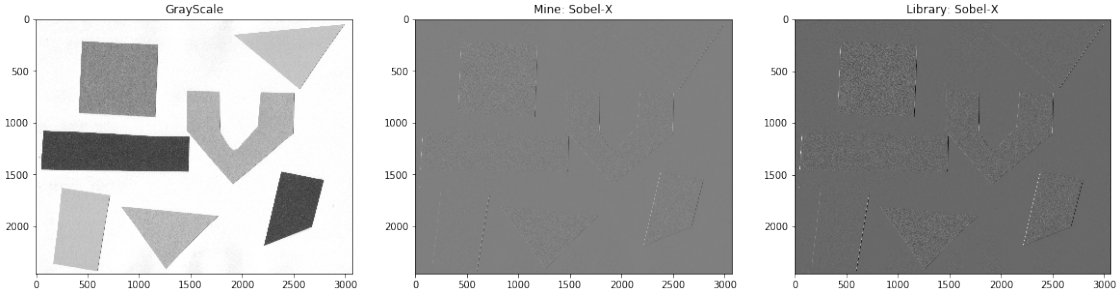

GrayScale
Mine: Sobel-Y
Library: Sobel-Y

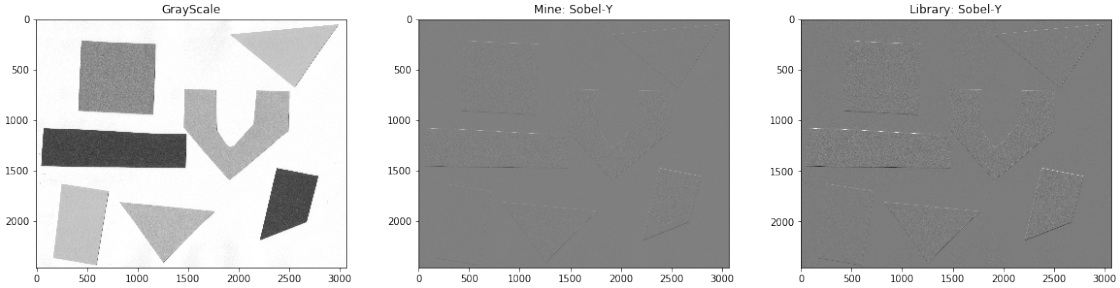

GrayScale
Mine: Laplacian
Library: Laplacian

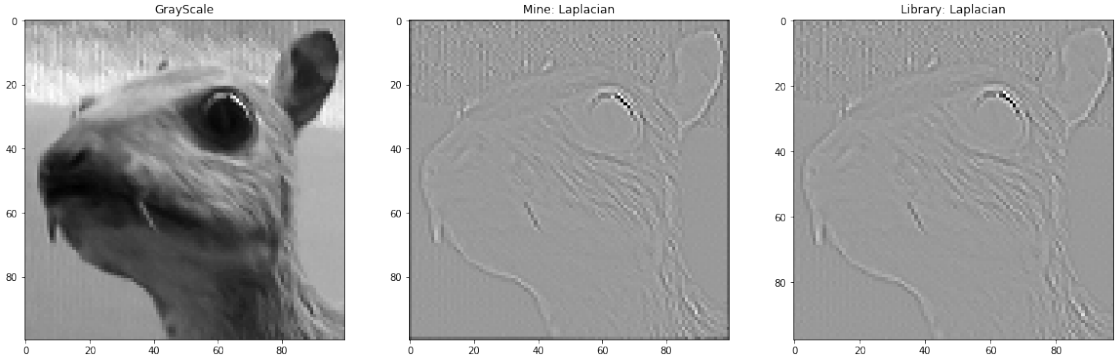

GrayScale
Mine: Sobel-X
Library: Sobel-X

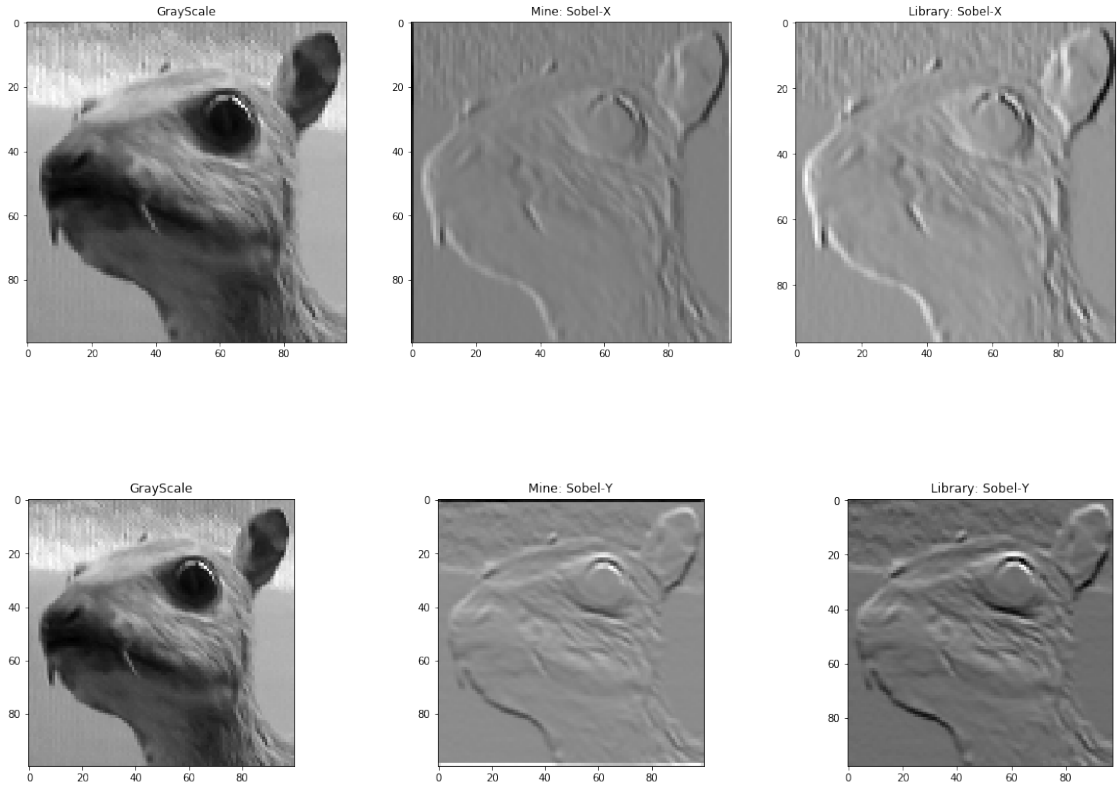

GrayScale
Mine: Sobel-Y
Library: Sobel-Y


GrayScale
Mine: Laplacian
Library: Laplacian


GrayScale
Mine: Sobel-X
Library: Sobel-X


GrayScale
Mine: Sobel-Y
Library: Sobel-Y


GrayScale
Mine: Laplacian
Library: Laplacian



1.2.6 Question 2.6 (10 points)

Load a couple of images from './Images/' directory. - Apply box filter using convolution, and display the resultant image - Apply Gaussian filter to the image, with varying sigma values. - Add Gausian Noise and Salt and Pepper Noise to them. - Apply Gaussian Filter and Median Filters. - Apply Sobel operator, computer gradient magnitude and display the results (original image, gradient images and gradient magnitude image) - Apply Canny Edge Deteciton Operator and display the results.

For application of filters, you can use builtin methods, available in various python packages. The results should be displayed in three columns i.e Original Image, Corrupted Image and the Filtered Image.

You are encouraged to play with the different parameter values

[25]:
```
from skimage import util,filters,feature
from scipy import ndimage

#return an image after applying gaussian,salt and pepper noise
def applyNoise(img):
    img = util.random_noise(img, mode='gaussian',seed=None, clip=True)
    img = util.random_noise(img, mode='salt',seed=None, clip=True)
    img = util.random_noise(img, mode='pepper',seed=None, clip=True)
```

```
    return img


def applyTasks(img):
    #noising
    corrupted = applyNoise(img)
    #display(corrupted)

    #box filter without noise
    img_boxFilter = cv2.blur(img,(10,10)) #10x10 kernal size
    fig = plt.figure()
    fig.add_subplot(1, 2, 1)
    plt.imshow(img)
    plt.title("Original")
    fig.add_subplot(1, 2, 2)
    plt.imshow(img_boxFilter)
    plt.title("Box Filter")

    #box filter with noise
    img_boxFilter = cv2.blur(img,(10,10)) #10x10 kernal size
    fig = plt.figure()
    fig.add_subplot(1, 3, 1)
    plt.imshow(img)
    plt.title("Original")
    fig.add_subplot(1, 3, 2)
    plt.imshow(corrupted)
    plt.title("Corrupted: Added Noise: gaussian,salt and pepper")
    fig.add_subplot(1, 3, 3)
    plt.imshow(img_boxFilter)
    plt.title("Box Filter: Blurred")


    #gaussian filter without noise
    img_gaussianFilter = cv2.GaussianBlur(img, (5,5),sigmaX=1)
    fig = plt.figure()
    fig.add_subplot(1, 2, 1)
    plt.imshow(img)
    plt.title("Original")
    fig.add_subplot(1, 2, 2)
    plt.imshow(img_gaussianFilter)
    plt.title("Gaussian Filter, sigma=1")

    #gaussian with noise
    img_gaussianFilter = cv2.GaussianBlur(corrupted, (5,5),sigmaX=1)
    fig = plt.figure()
    fig.add_subplot(1, 3, 1)
    plt.imshow(img)
```

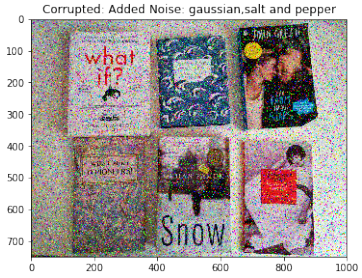
```
    plt.title("Original")
    fig.add_subplot(1, 3, 2)
    plt.imshow(corrupted)
    plt.title("Corrupted: Added Noise: gaussian,salt and pepper")
    fig.add_subplot(1, 3, 3)
    plt.imshow(img_gaussianFilter)
    plt.title("Gaussian Filter, sigma=1")

    #Median filter
    #opencv medianfilter is resulting into error
    #img_med = cv2.medianBlur(img,5)

    img_med = ndimage.median_filter(img,5)
    fig = plt.figure()
    fig.add_subplot(1, 2, 1)
    plt.imshow(img)
    plt.title("Original")
    fig.add_subplot(1, 2, 2)
    plt.imshow(img_gaussianFilter)
    plt.title("Median Filter")

    #Median filter without noise
    img_med = ndimage.median_filter(corrupted,5)
    fig = plt.figure()
    fig.add_subplot(1, 3, 1)
    plt.imshow(img)
    plt.title("Original")
    fig.add_subplot(1, 3, 2)
    plt.imshow(corrupted)
    plt.title("Corrupted: Added Noise: gaussian,salt and pepper")
    fig.add_subplot(1, 3, 3)
    plt.imshow(img_med)
    plt.title("Median Filter")


    #sobel

    #img_sobel = ndimage.sobel(color.rgb2gray(img))
    #print('library sobel')

    #sx--> horizontal der
    img_sobelX = ndimage.sobel(color.rgb2gray(img),axis=0)
    #sy--> vertical der
    img_sobelY = ndimage.sobel(color.rgb2gray(img),axis=1)
    gradient_magnitude = np.sqrt((np.power(img_sobelX,2)+np.
↪power(img_sobelY,2)))
    np.hypot(img_sobelX,img_sobelY)
```

```
    fig = plt.figure()
    fig.add_subplot(1, 5, 1)
    plt.imshow(img)
    plt.title("Original")
    fig.add_subplot(1, 5, 2)
    plt.imshow(color.rgb2gray(img))
    plt.title("Gray")
    fig.add_subplot(1, 5, 3)
    plt.imshow(img_sobelX)
    plt.title("sX: derivate in horizontal direction")
    fig.add_subplot(1, 5, 4)
    plt.imshow(img_sobelY)
    plt.title("sY: derivate in vertical direction")
    fig.add_subplot(1, 5, 5)
    plt.imshow(gradient_magnitude)
    plt.title("Gradient Magnitude")


    #canny edge detector
    img_canny = feature.canny(color.rgb2gray(img))
    fig = plt.figure()
    fig.add_subplot(1, 3, 1)
    plt.imshow(img)
    plt.title("Original")
    fig.add_subplot(1, 3, 2)
    plt.imshow(color.rgb2gray(img))
    plt.title("Gray")
    fig.add_subplot(1, 3, 3)
    plt.imshow(img_canny)
    plt.title("Canny Edge")



img = load('./Images/books.jpg')
applyTasks(img)

img = load('./Images/coins.jpg')
applyTasks(img)

img = load('./Images/image.jpg')
applyTasks(img)
```


Original


Box Filter


Original


Corrupted: Added Noise: gaussian,salt and pepper


Box Filter: Blurred


Original

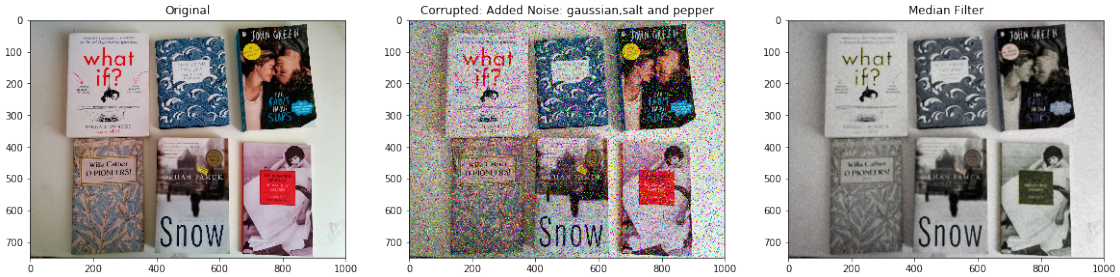

Gaussian Filter, sigma=1

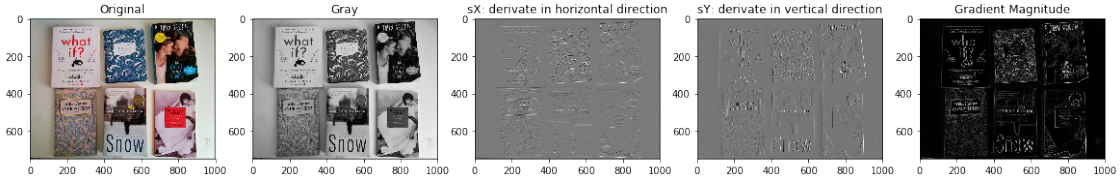

Original


Corrupted: Added Noise: gaussian,salt and pepper


Gaussian Filter, sigma=1

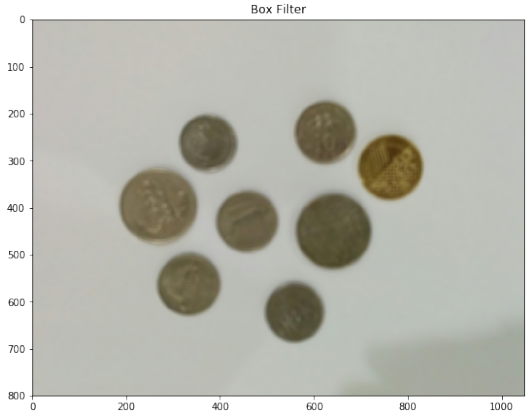

Original
Median Filter

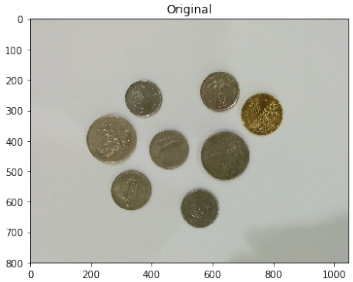

Original
Corrupted: Added Noise: gaussian,salt and pepper
Median Filter

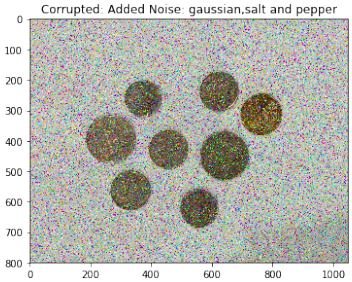

Original
Gray
sX: derivate in horizontal direction
sY: derivate in vertical direction
Gradient Magnitude

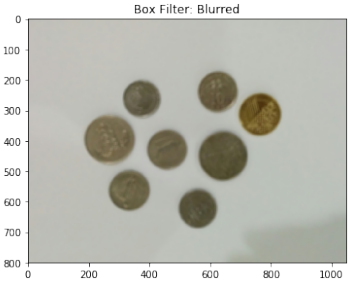

Original
Gray
Canny Edge


Original

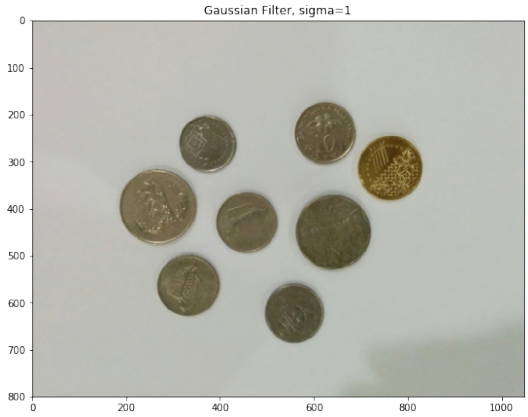

Box Filter

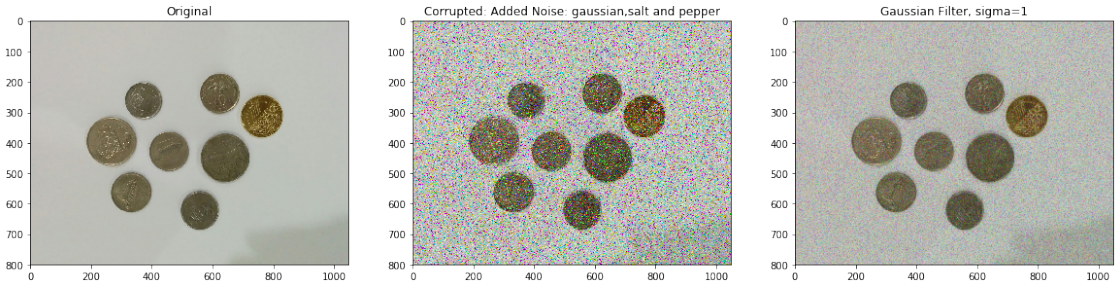

Original

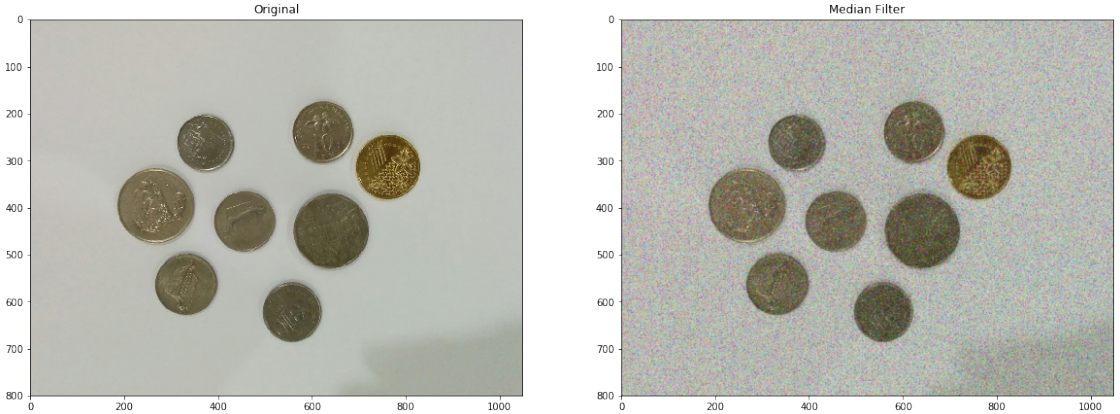

Corrupted: Added Noise: gaussian,salt and pepper

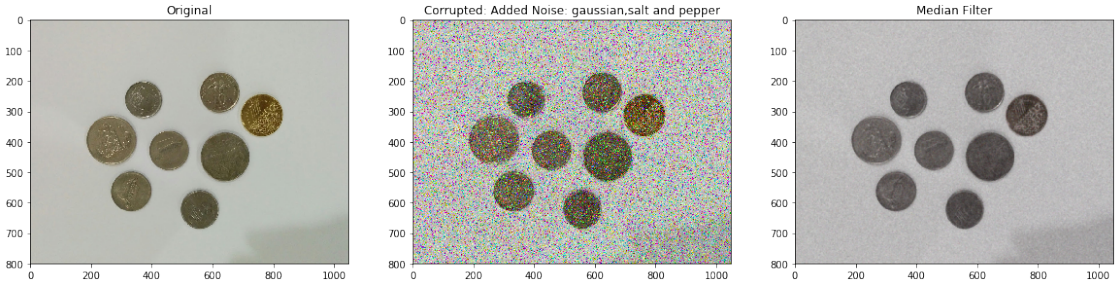

Box Filter: Blurred

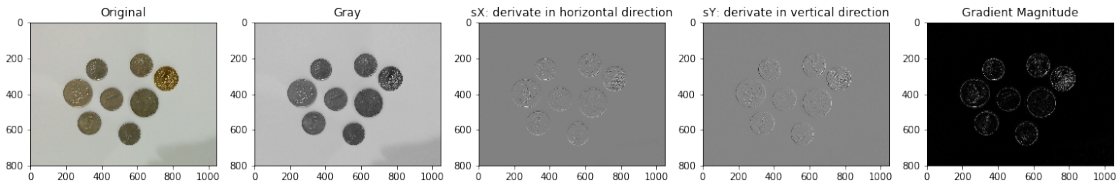

Original

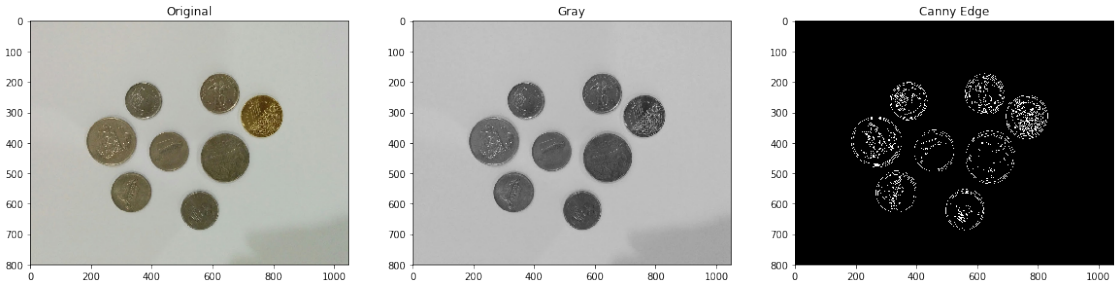

Gaussian Filter, sigma=1

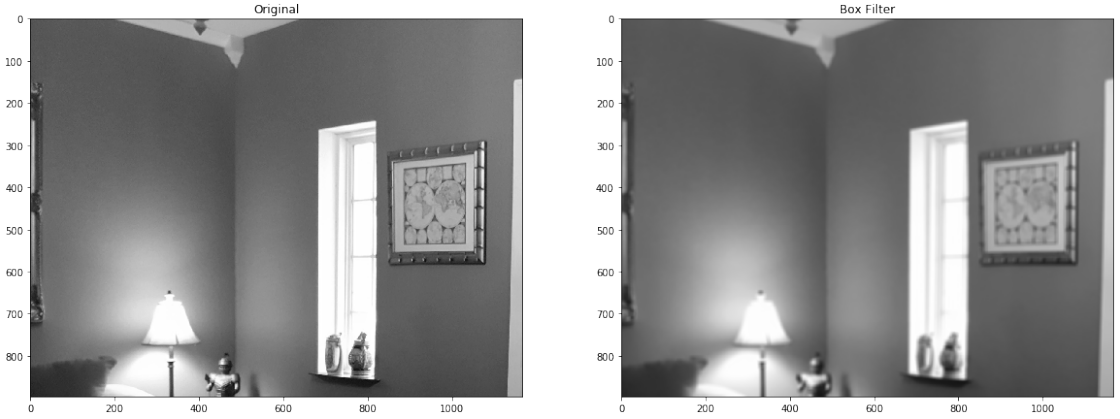

Original
Corrupted: Added Noise: gaussian,salt and pepper
Gaussian Filter, sigma=1

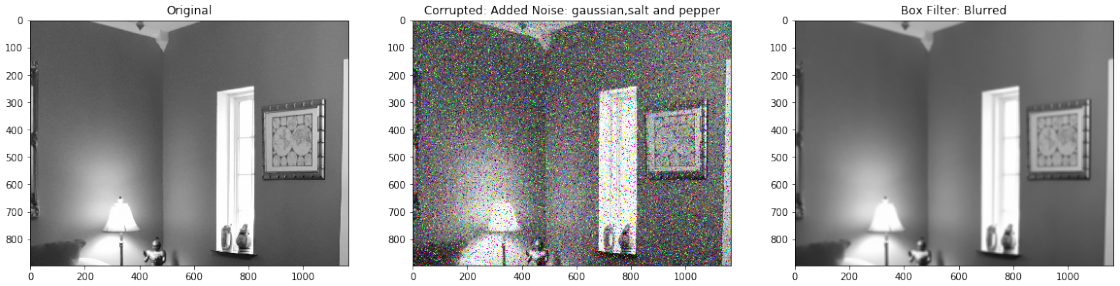

Original
Median Filter


Original
Corrupted: Added Noise: gaussian,salt and pepper
Median Filter


Original
Gray
sX: derivate in horizontal direction
sY: derivate in vertical direction
Gradient Magnitude


Original
Gray
Canny Edge

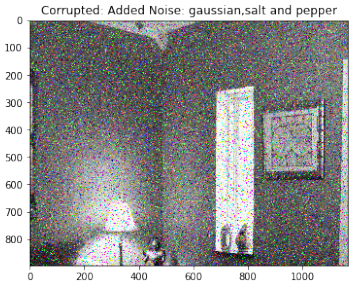

Original
Box Filter

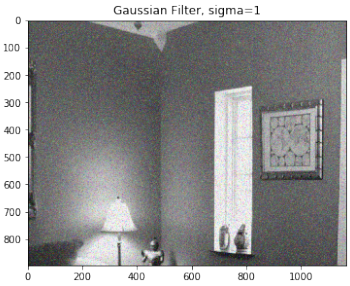

Original
Corrupted: Added Noise: gaussian,salt and pepper
Box Filter: Blurred


Original

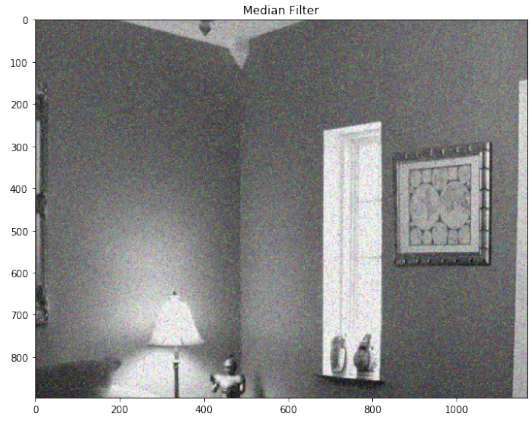

Gaussian Filter, sigma=1

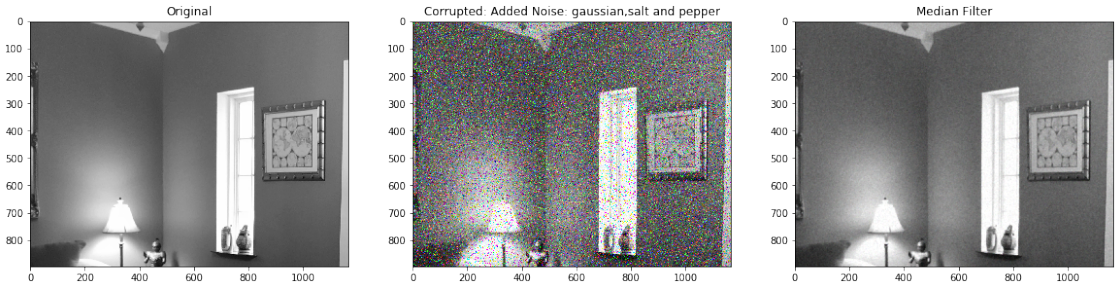

Original

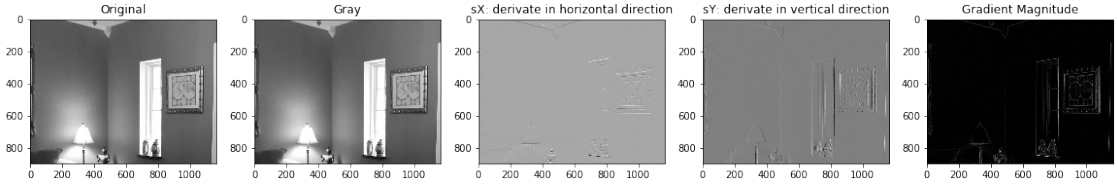

Corrupted: Added Noise: gaussian,salt and pepper

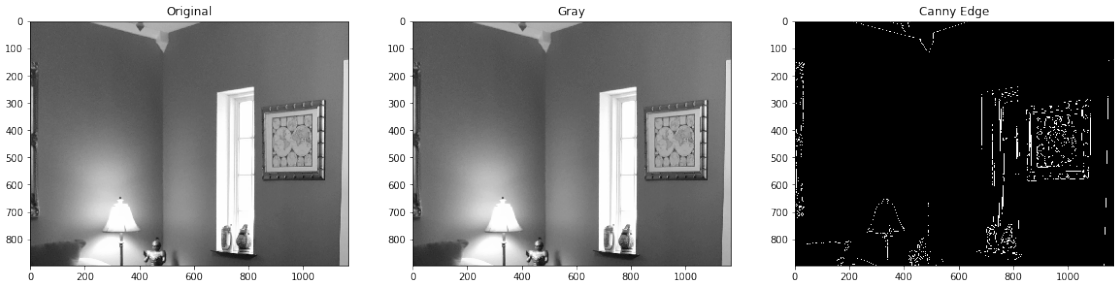

Gaussian Filter, sigma=1


Original


Median Filter







1.2.7 Question 2.7 (10 points : Extra Credit)

Read the video from webcam and apply any of the filtering operation (blurring, gradient magnitude, edge detection e.t.c on the video stream in the real time and display the resultant video stream, showeing the effect of image filterering operation in real time.

Hint: You can use ***ipywebrtc*** *library*

[26]:
```
from ipywebrtc import VideoStream
#video = VideoStream.from_file('./images/Bunny.mp4')
#video
```

[ ]:

[ ]:

[ ]: