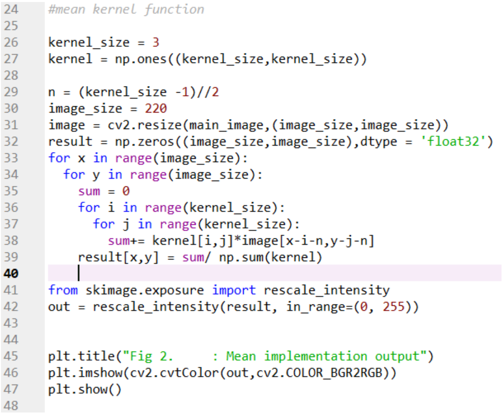
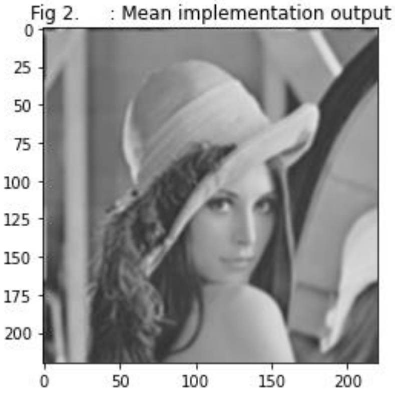
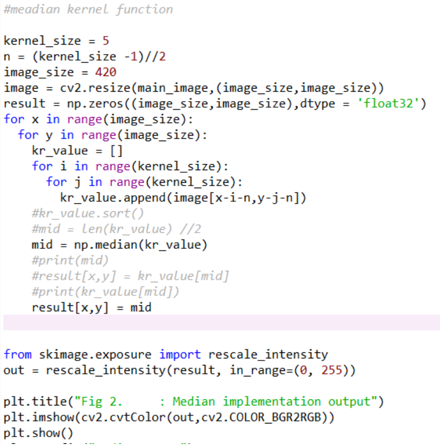
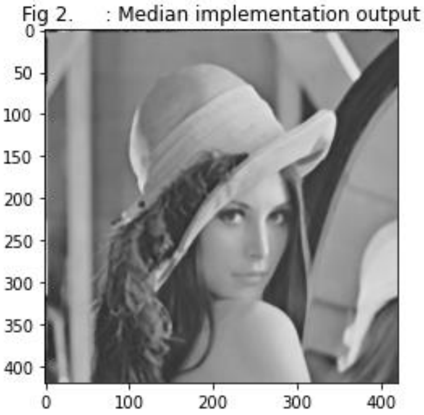


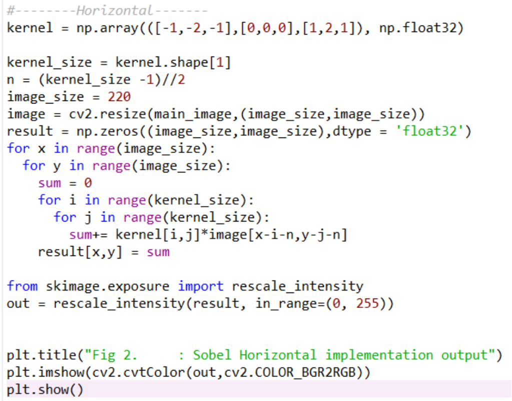
```
#mean kernel function

kernel_size = 3
kernel = np.ones((kernel_size,kernel_size))

n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum/ np.sum(kernel)
    
from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))


plt.title("Fig 2.      : Mean implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```



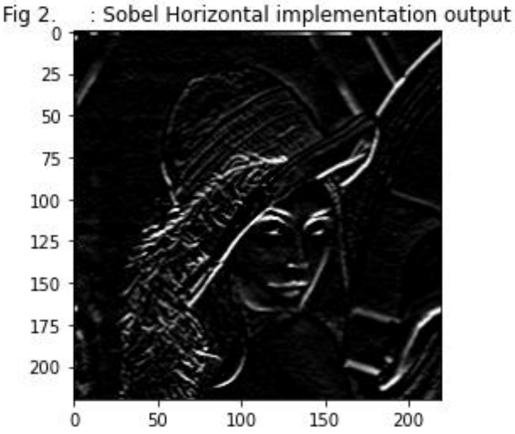


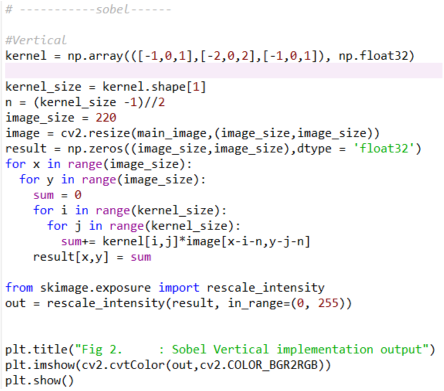
```
#meadian kernel function

kernel_size = 5
n = (kernel_size -1)//2
image_size = 420
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    kr_value = []
    for i in range(kernel_size):
      for j in range(kernel_size):
        kr_value.append(image[x-i-n,y-j-n])
    #kr_value.sort()
    #mid = len(kr_value) //2
    mid = np.median(kr_value)
    #print(mid)
    #result[x,y] = kr_value[mid]
    #print(kr_value[mid])
    result[x,y] = mid


from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Fig 2.      : Median implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```



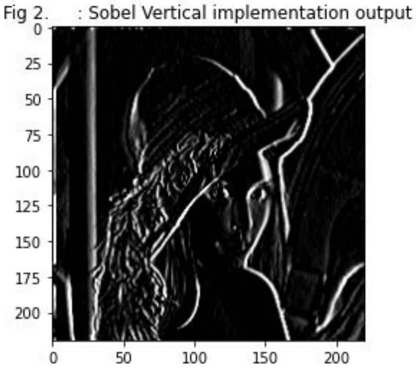


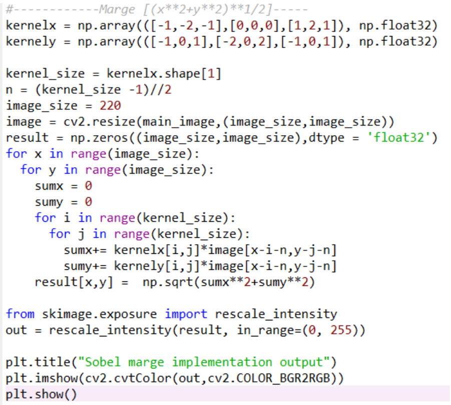
```
#--------Horizontal-------
kernel = np.array(([-1,-2,-1],[0,0,0],[1,2,1]), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))


plt.title("Fig 2.      : Sobel Horizontal implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```



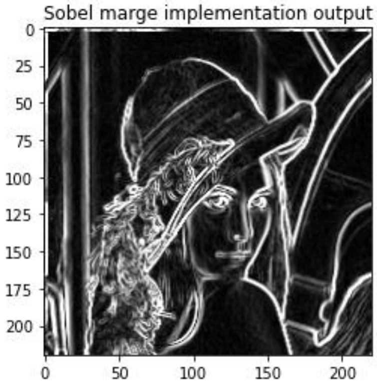


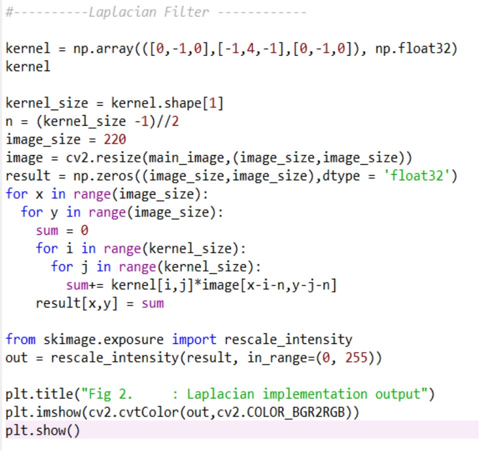
```
# -----------sobel------

#Vertical
kernel = np.array(([-1,0,1],[-2,0,2],[-1,0,1]), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))


plt.title("Fig 2.     : Sobel Vertical implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```



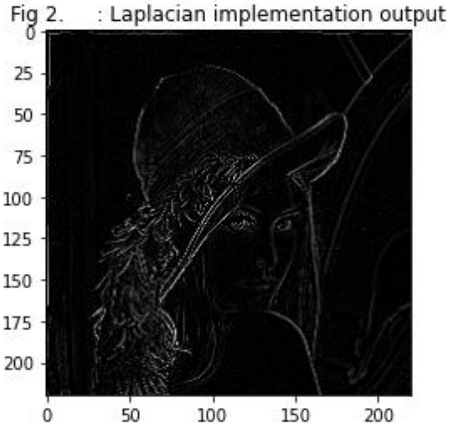


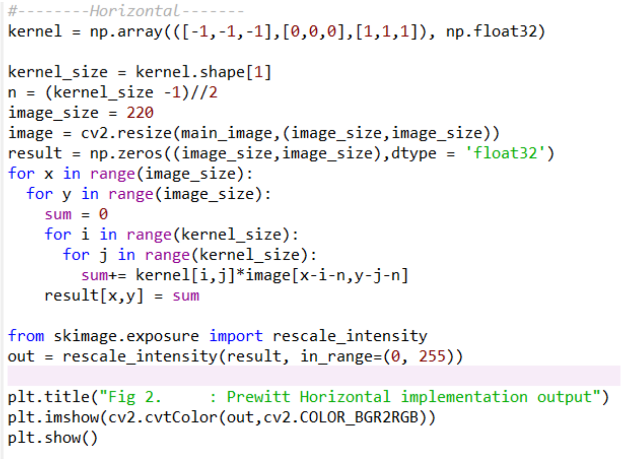
```
#-----------Marge [(x**2+y**2)**1/2]-----
kernelx = np.array(([-1,-2,-1],[0,0,0],[1,2,1]), np.float32)
kernely = np.array(([-1,0,1],[-2,0,2],[-1,0,1]), np.float32)

kernel_size = kernelx.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sumx = 0
    sumy = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sumx+= kernelx[i,j]*image[x-i-n,y-j-n]
        sumy+= kernely[i,j]*image[x-i-n,y-j-n]
    result[x,y] =  np.sqrt(sumx**2+sumy**2)

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Sobel marge implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```





```
#----------Laplacian Filter ------------

kernel = np.array(([0,-1,0],[-1,4,-1],[0,-1,0]), np.float32)
kernel

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Fig 2.     : Laplacian implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```





```
#--------Horizontal-------
kernel = np.array(([-1,-1,-1],[0,0,0],[1,1,1]), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Fig 2.     : Prewitt Horizontal implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

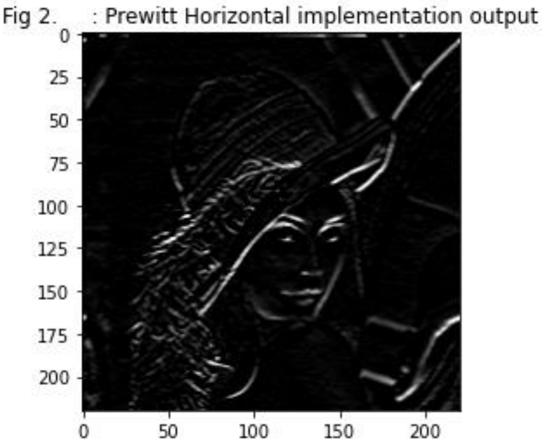
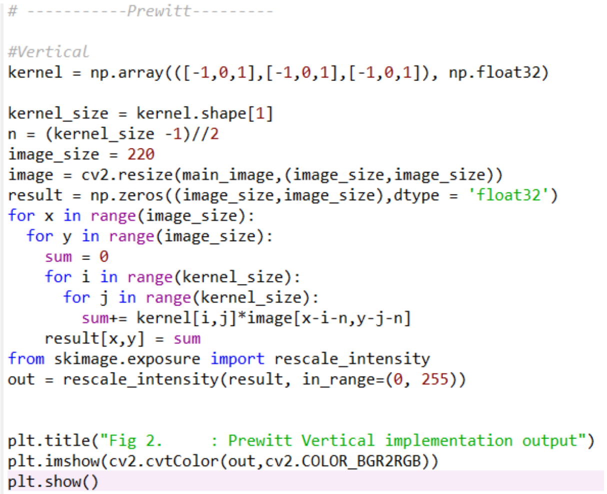


Fig 2.   : Prewitt Horizontal implementation output



```
# -----------Prewitt---------

#Vertical
kernel = np.array(([-1,0,1],[-1,0,1],[-1,0,1]), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum
from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))


plt.title("Fig 2.     : Prewitt Vertical implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

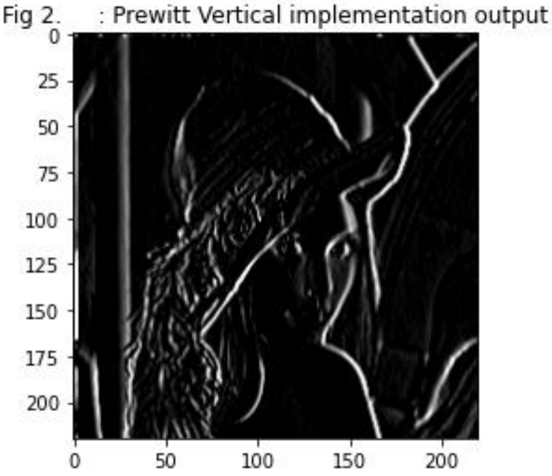
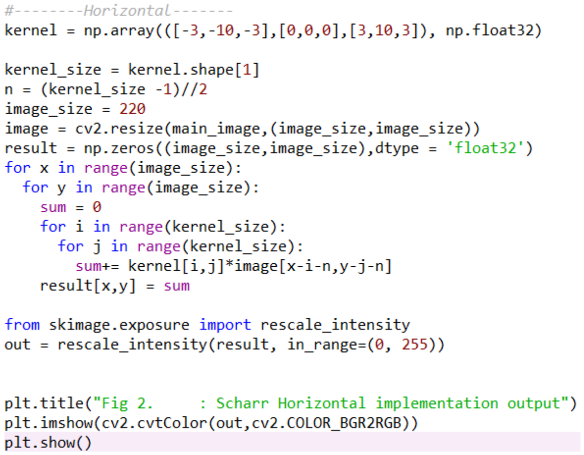


Fig 2.   : Prewitt Vertical implementation output



```
#--------Horizontal-------
kernel = np.array(([-3,-10,-3],[0,0,0],[3,10,3]), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))


plt.title("Fig 2.     : Scharr Horizontal implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

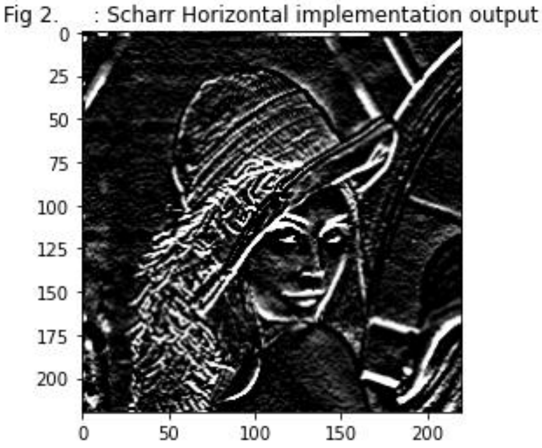
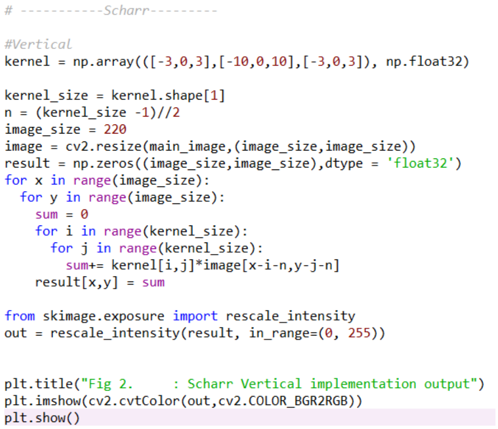
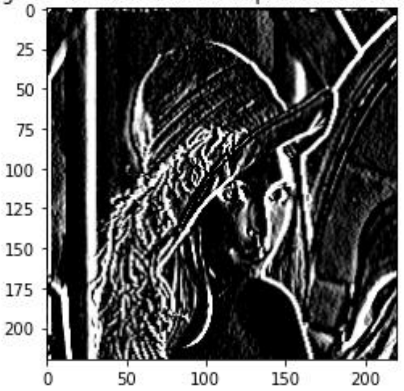


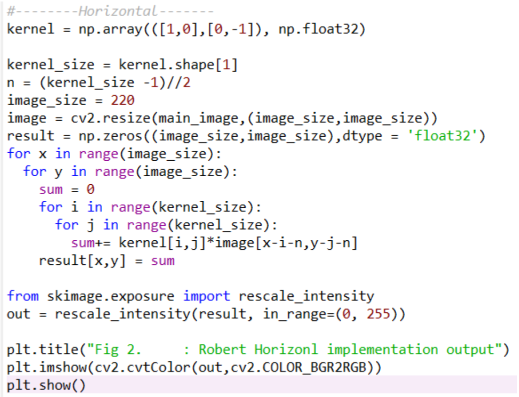
Fig 2.   : Scharr Horizontal implementation output



```
# -----------Scharr---------

#Vertical
kernel = np.array(([-3,0,3],[-10,0,10],[-3,0,3]), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))


plt.title("Fig 2.     : Scharr Vertical implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

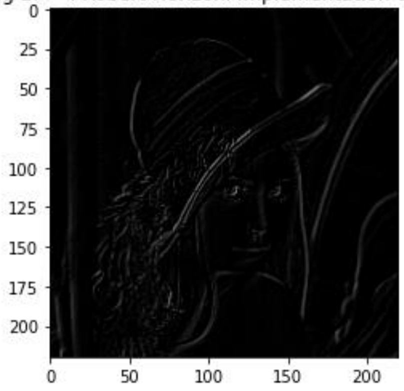
Fig 2.   : Scharr Vertical implementation output

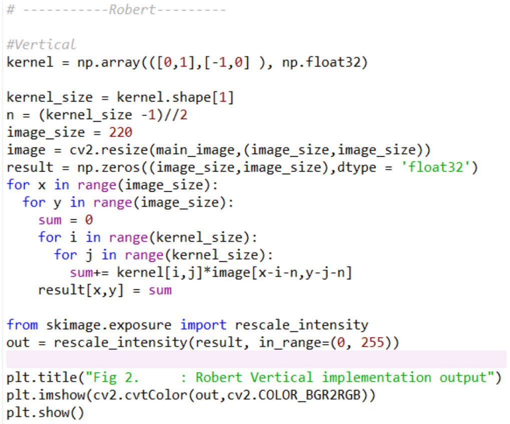




```
#--------Horizontal-------
kernel = np.array(([1,0],[0,-1]), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Fig 2.     : Robert Horizonl implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

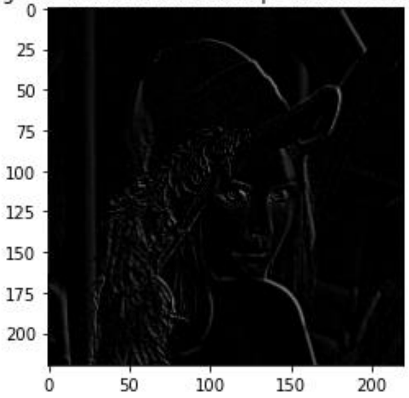
Fig 2.   : Robert Horizonl implementation output

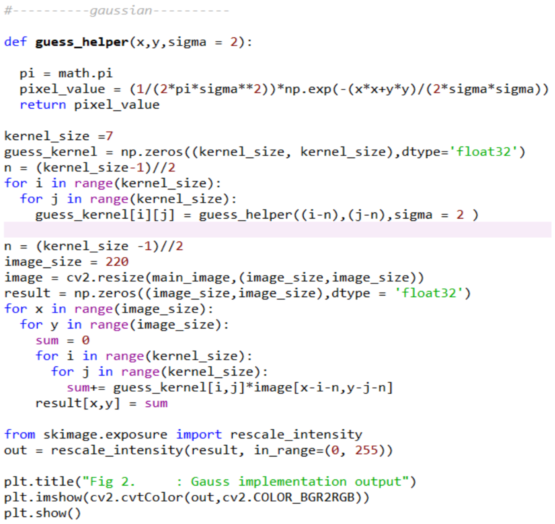




```
# -----------Robert---------

#Vertical
kernel = np.array(([0,1],[-1,0] ), np.float32)

kernel_size = kernel.shape[1]
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Fig 2.     : Robert Vertical implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

Fig 2.   : Robert Vertical implementation output





```
#----------gaussian----------

def guess_helper(x,y,sigma = 2):

  pi = math.pi
  pixel_value = (1/(2*pi*sigma**2))*np.exp(-(x*x+y*y)/(2*sigma*sigma))
  return pixel_value

kernel_size =7
guess_kernel = np.zeros((kernel_size, kernel_size),dtype='float32')
n = (kernel_size-1)//2
for i in range(kernel_size):
  for j in range(kernel_size):
    guess_kernel[i][j] = guess_helper((i-n),(j-n),sigma = 2 )

n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(main_image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= guess_kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Fig 2.     : Gauss implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

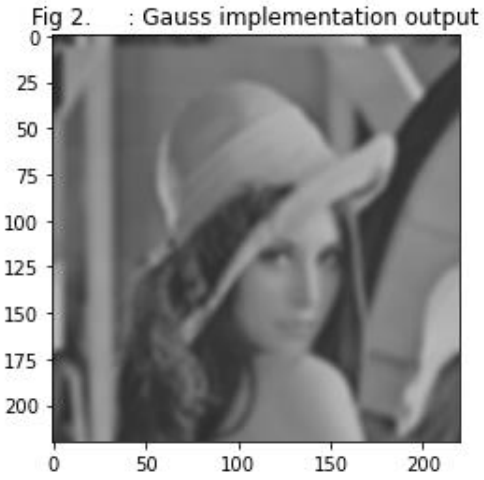
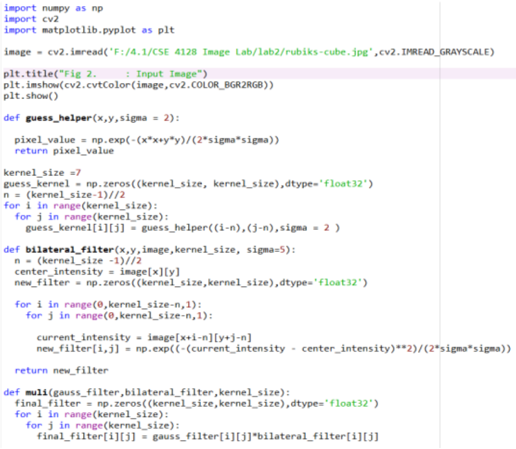
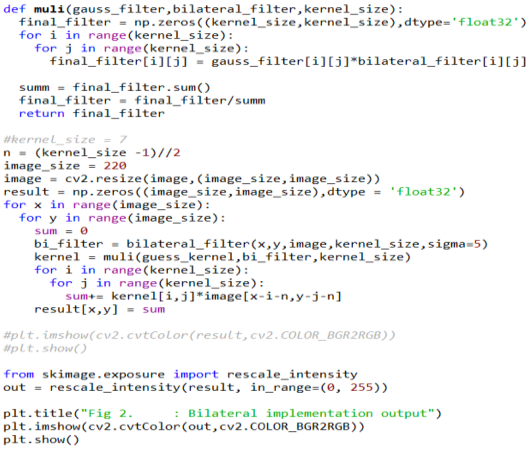


Fig 2.   : Gauss implementation output



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

image = cv2.imread('F:/4.1/CSE 4128 Image Lab/lab2/rubiks-cube.jpg',cv2.IMREAD_GRAYSCALE)

plt.title("Fig 2.     : Input Image")
plt.imshow(cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
plt.show()

def guess_helper(x,y,sigma = 2):

  pixel_value = np.exp(-(x*x+y*y)/(2*sigma*sigma))
  return pixel_value

kernel_size =7
guess_kernel = np.zeros((kernel_size, kernel_size),dtype='float32')
n = (kernel_size-1)//2
for i in range(kernel_size):
  for j in range(kernel_size):
    guess_kernel[i][j] = guess_helper((i-n),(j-n),sigma = 2 )

def bilateral_filter(x,y,image,kernel_size, sigma=5):
  n = (kernel_size -1)//2
  center_intensity = image[x][y]
  new_filter = np.zeros((kernel_size,kernel_size),dtype='float32')

  for i in range(0,kernel_size-n,1):
    for j in range(0,kernel_size-n,1):

      current_intensity = image[x+i-n][y+j-n]
      new_filter[i,j] = np.exp((-(current_intensity - center_intensity)**2)/(2*sigma*sigma))

  return new_filter

def muli(gauss_filter,bilateral_filter,kernel_size):
  final_filter = np.zeros((kernel_size,kernel_size),dtype='float32')
  for i in range(kernel_size):
    for j in range(kernel_size):
      final_filter[i][j] = gauss_filter[i][j]*bilateral_filter[i][j]
```



```
def muli(gauss_filter,bilateral_filter,kernel_size):
  final_filter = np.zeros((kernel_size,kernel_size),dtype='float32')
  for i in range(kernel_size):
    for j in range(kernel_size):
      final_filter[i][j] = gauss_filter[i][j]*bilateral_filter[i][j]

  summ = final_filter.sum()
  final_filter = final_filter/summ
  return final_filter

#kernel_size = 7
n = (kernel_size -1)//2
image_size = 220
image = cv2.resize(image,(image_size,image_size))
result = np.zeros((image_size,image_size),dtype = 'float32')
for x in range(image_size):
  for y in range(image_size):
    sum = 0
    bi_filter = bilateral_filter(x,y,image,kernel_size,sigma=5)
    kernel = muli(guess_kernel,bi_filter,kernel_size)
    for i in range(kernel_size):
      for j in range(kernel_size):
        sum+= kernel[i,j]*image[x-i-n,y-j-n]
    result[x,y] = sum

#plt.imshow(cv2.cvtColor(result,cv2.COLOR_BGR2RGB))
#plt.show()

from skimage.exposure import rescale_intensity
out = rescale_intensity(result, in_range=(0, 255))

plt.title("Fig 2.     : Bilateral implementation output")
plt.imshow(cv2.cvtColor(out,cv2.COLOR_BGR2RGB))
plt.show()
```

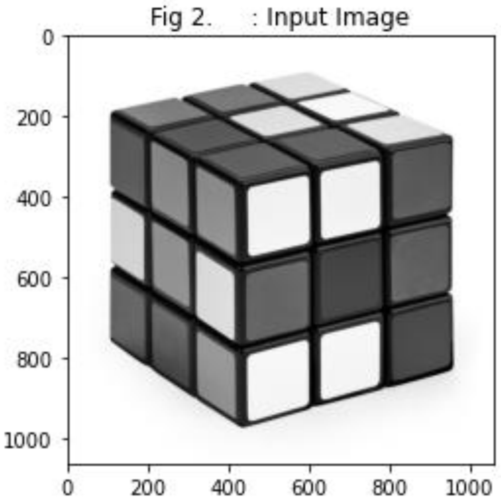


Fig 2.   : Input Image

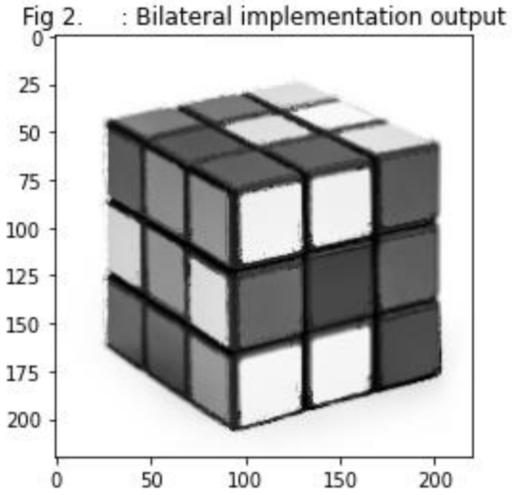


Fig 2.   : Bilateral implementation output