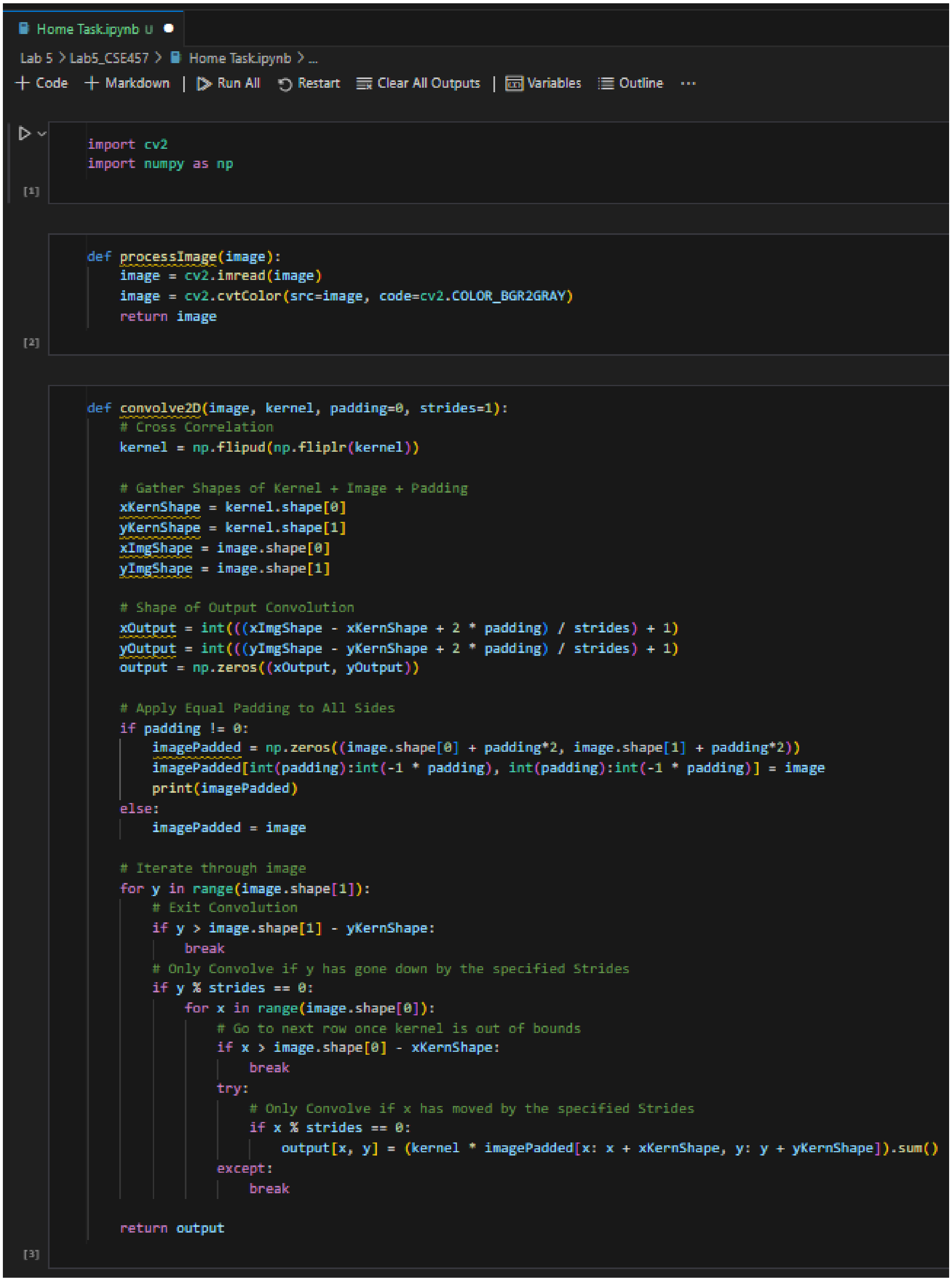
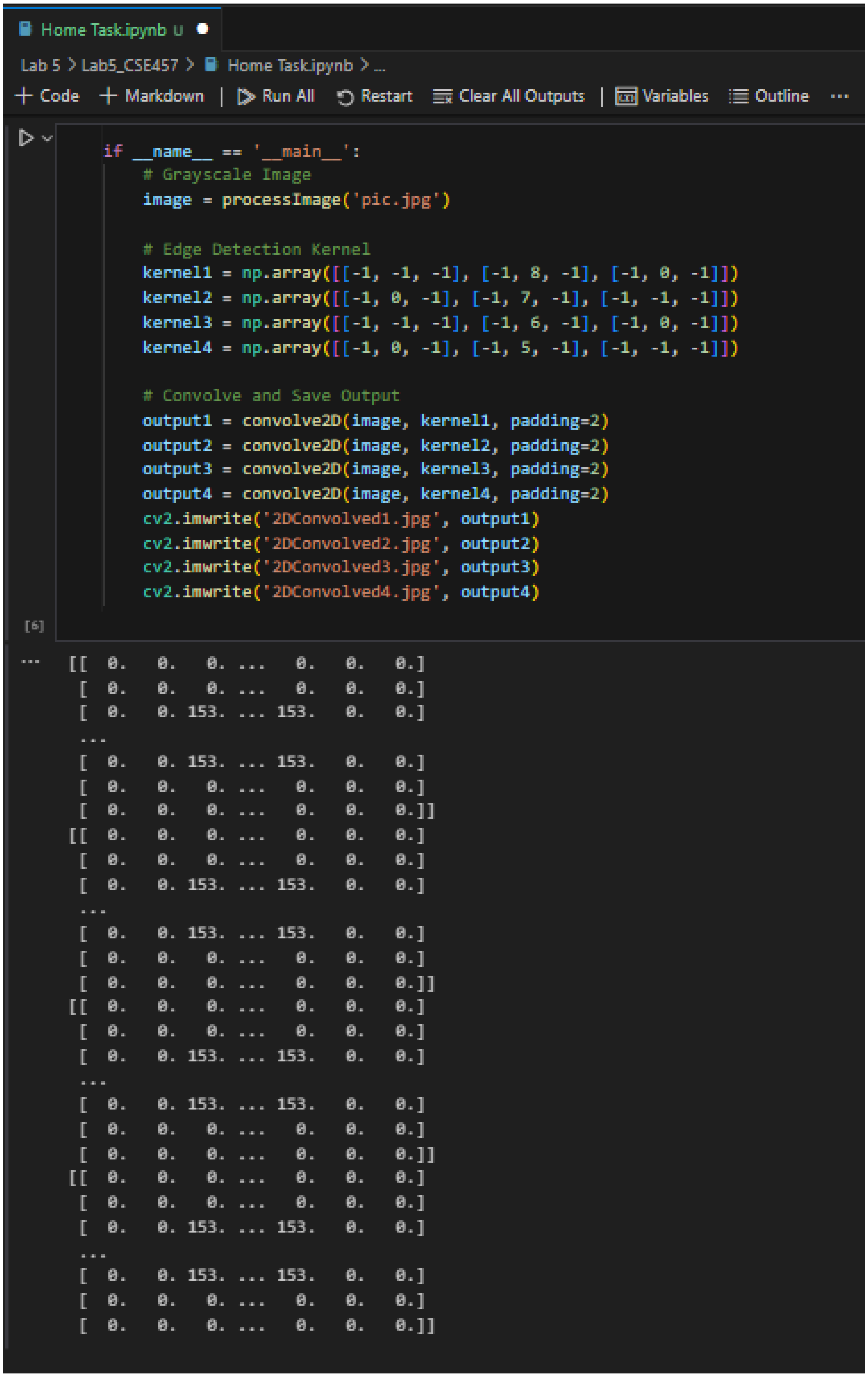
* **Spatial Filtering**
* **Edge Detections**



```python
import cv2
import numpy as np
```

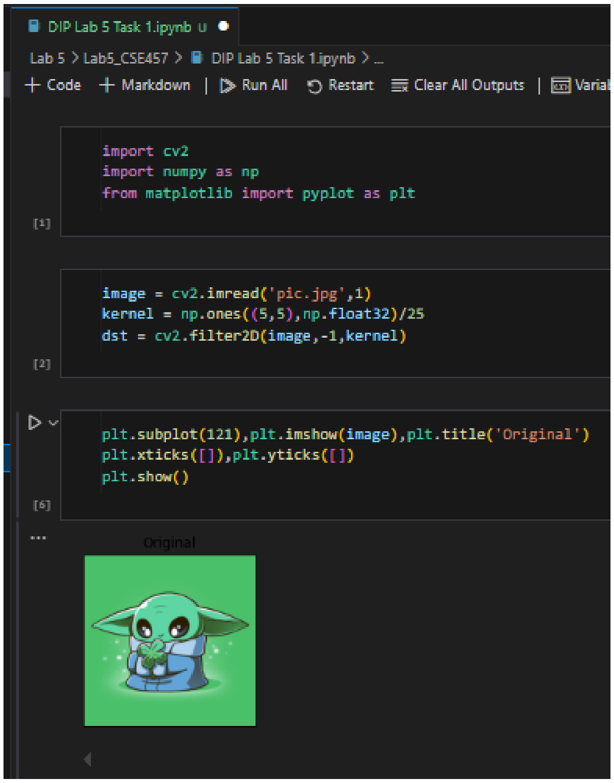
```python
def processImage(image):
    image = cv2.imread(image)
    image = cv2.cvtColor(src=image, code=cv2.COLOR_BGR2GRAY)
    return image
```

```python
def convolve2D(image, kernel, padding=0, strides=1):
    # Cross Correlation
    kernel = np.flipud(np.fliplr(kernel))

    # Gather Shapes of Kernel + Image + Padding
    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    # Shape of Output Convolution
    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
    output = np.zeros((xOutput, yOutput))

    # Apply Equal Padding to All Sides
    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 * padding)] = image
        print(imagePadded)
    else:
        imagePadded = image

    # Iterate through image
    for y in range(image.shape[1]):
        # Exit Convolution
        if y > image.shape[1] - yKernShape:
            break
        # Only Convolve if y has gone down by the specified Strides
        if y % strides == 0:
            for x in range(image.shape[0]):
                # Go to next row once kernel is out of bounds
                if x > image.shape[0] - xKernShape:
                    break
                try:
                    # Only Convolve if x has moved by the specified Strides
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKernShape]).sum()
                except:
                    break

    return output
```



```python
if __name__ == '__main__':
    # Grayscale Image
    image = processImage('pic.jpg')

    # Edge Detection Kernel
    kernel1 = np.array([[-1, -1, -1], [-1, 8, -1], [-1, 0, -1]])
    kernel2 = np.array([[-1, 0, -1], [-1, 7, -1], [-1, -1, -1]])
    kernel3 = np.array([[-1, -1, -1], [-1, 6, -1], [-1, 0, -1]])
    kernel4 = np.array([[-1, 0, -1], [-1, 5, -1], [-1, -1, -1]])

    # Convolve and Save Output
    output1 = convolve2D(image, kernel1, padding=2)
    output2 = convolve2D(image, kernel2, padding=2)
    output3 = convolve2D(image, kernel3, padding=2)
    output4 = convolve2D(image, kernel4, padding=2)
    cv2.imwrite('2DConvolved1.jpg', output1)
    cv2.imwrite('2DConvolved2.jpg', output2)
    cv2.imwrite('2DConvolved3.jpg', output3)
    cv2.imwrite('2DConvolved4.jpg', output4)
```

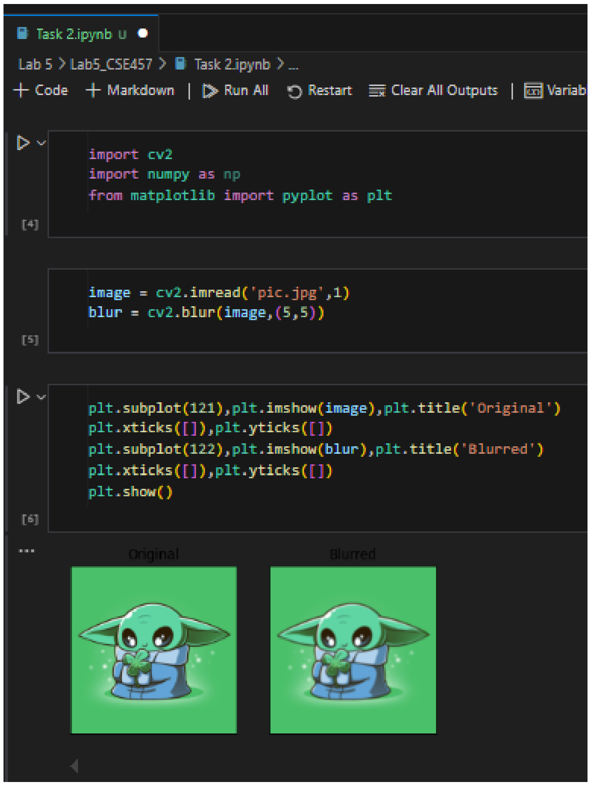
```
[[  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0. 153. ... 153.   0.   0.]
 ...
 [  0.   0. 153. ... 153.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]]
[[  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0. 153. ... 153.   0.   0.]
 ...
 [  0.   0. 153. ... 153.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]]
[[  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0. 153. ... 153.   0.   0.]
 ...
 [  0.   0. 153. ... 153.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]]
[[  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0. 153. ... 153.   0.   0.]
 ...
 [  0.   0. 153. ... 153.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]
 [  0.   0.   0. ...   0.   0.   0.]]
```


DIP Lab 5 Task 1.ipynb U
Lab 5 > Lab5_CSE457 > DIP Lab 5 Task 1.ipynb > ...
+ Code + Markdown | Run All Restart Clear All Outputs | Varial
[1]
[2]
[6]
Original

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

```
image = cv2.imread('pic.jpg',1)
kernel = np.ones((5,5),np.float32)/25
dst = cv2.filter2D(image,-1,kernel)
```

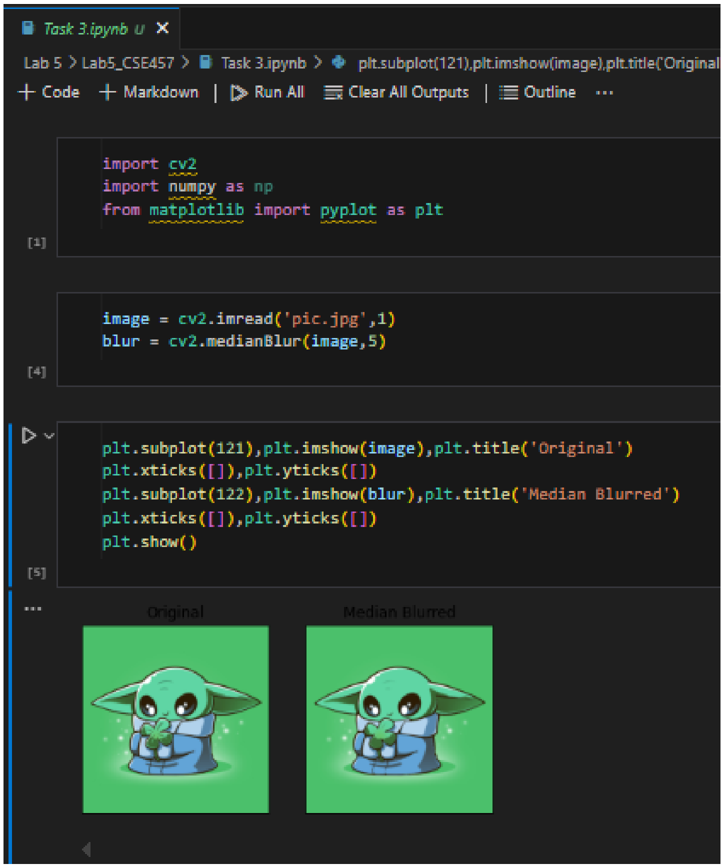
```
plt.subplot(121),plt.imshow(image),plt.title('Original')
plt.xticks([]),plt.yticks([])
plt.show()
```


Task 2.ipynb U
Lab 5 > Lab5_CSE457 > Task 2.ipynb > ...
+ Code + Markdown | Run All Restart Clear All Outputs | Variab
[4]
[5]
[6]
Original
Blurred

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

```
image = cv2.imread('pic.jpg',1)
blur = cv2.blur(image,(5,5))
```

```
plt.subplot(121),plt.imshow(image),plt.title('Original')
plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]),plt.yticks([])
plt.show()
```

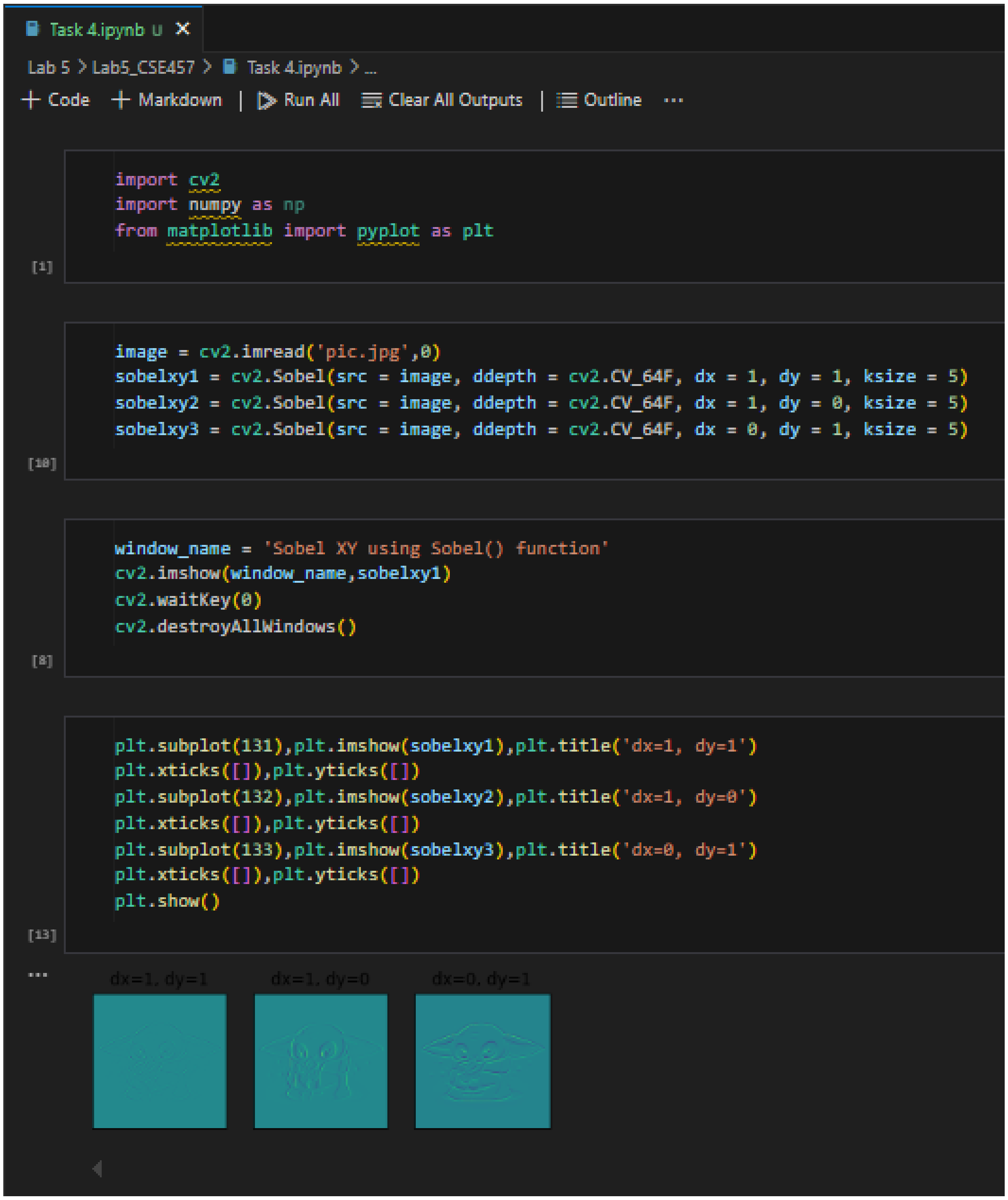

Task 3.ipynb U
Lab 5 > Lab5_CSE457 > Task 3.ipynb > plt.subplot(121),plt.imshow(image),plt.title('Original
+ Code + Markdown | Run All Clear All Outputs | Outline ...
[1]
[4]
[5]
Original
Median Blurred

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

```
image = cv2.imread('pic.jpg',1)
blur = cv2.medianBlur(image,5)
```

```
plt.subplot(121),plt.imshow(image),plt.title('Original')
plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Median Blurred')
plt.xticks([]),plt.yticks([])
plt.show()
```

* Sobel Edge detection



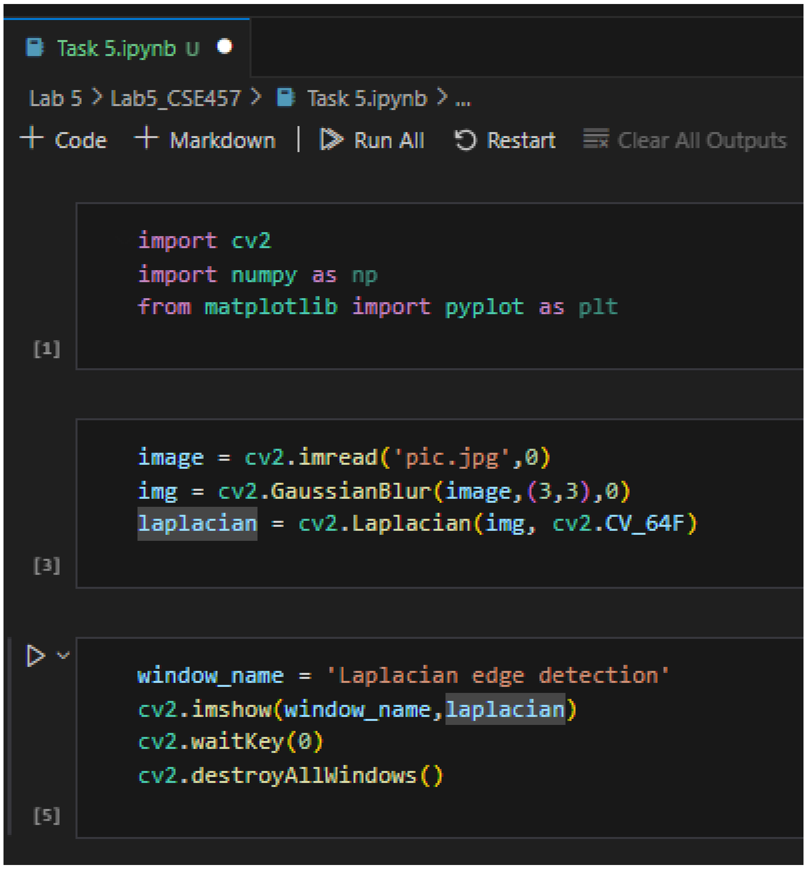
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

```
image = cv2.imread('pic.jpg',0)
sobelxy1 = cv2.Sobel(src = image, ddepth = cv2.CV_64F, dx = 1, dy = 1, ksize = 5)
sobelxy2 = cv2.Sobel(src = image, ddepth = cv2.CV_64F, dx = 1, dy = 0, ksize = 5)
sobelxy3 = cv2.Sobel(src = image, ddepth = cv2.CV_64F, dx = 0, dy = 1, ksize = 5)
```

```
window_name = 'Sobel XY using Sobel() function'
cv2.imshow(window_name,sobelxy1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
plt.subplot(131),plt.imshow(sobelxy1),plt.title('dx=1, dy=1')
plt.xticks([]),plt.yticks([])
plt.subplot(132),plt.imshow(sobelxy2),plt.title('dx=1, dy=0')
plt.xticks([]),plt.yticks([])
plt.subplot(133),plt.imshow(sobelxy3),plt.title('dx=0, dy=1')
plt.xticks([]),plt.yticks([])
plt.show()
```

* Laplacian Edge detection



[1]

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

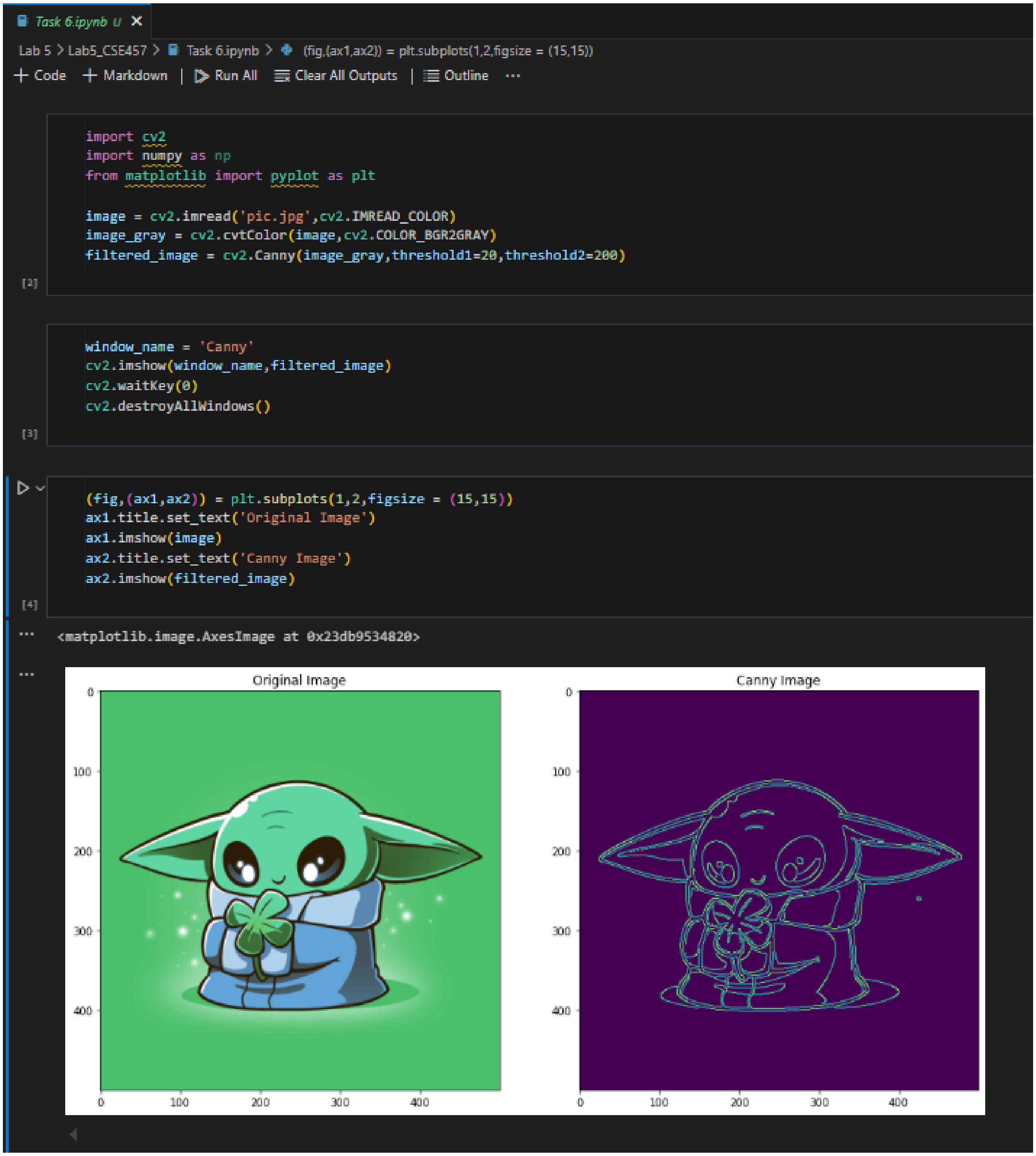
[3]

```
image = cv2.imread('pic.jpg',0)
img = cv2.GaussianBlur(image,(3,3),0)
laplacian = cv2.Laplacian(img, cv2.CV_64F)
```

[5]

```
window_name = 'Laplacian edge detection'
cv2.imshow(window_name,laplacian)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

* Canny edge Detection



```
[2]
import cv2
import numpy as np
from matplotlib import pyplot as plt

image = cv2.imread('pic.jpg',cv2.IMREAD_COLOR)
image_gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
filtered_image = cv2.Canny(image_gray,threshold1=20,threshold2=200)
```

```
[3]
window_name = 'Canny'
cv2.imshow(window_name,filtered_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
[4]
(fig,(ax1,ax2)) = plt.subplots(1,2,figsize = (15,15))
ax1.title.set_text('Original Image')
ax1.imshow(image)
ax2.title.set_text('Canny Image')
ax2.imshow(filtered_image)
```

```
<matplotlib.image.AxesImage at 0x23db9534820>
```