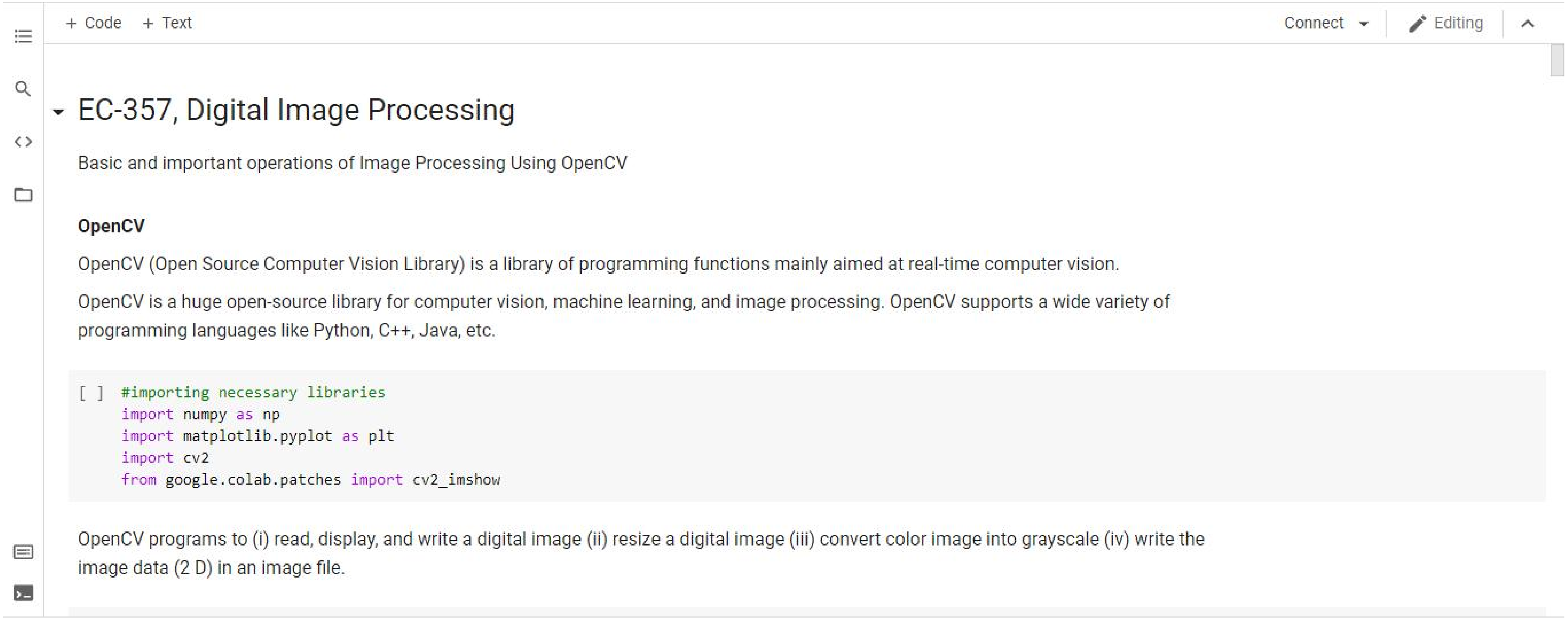


EC-357, Digital Image Processing

Basic and important operations of Image Processing Using OpenCV

OpenCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision.

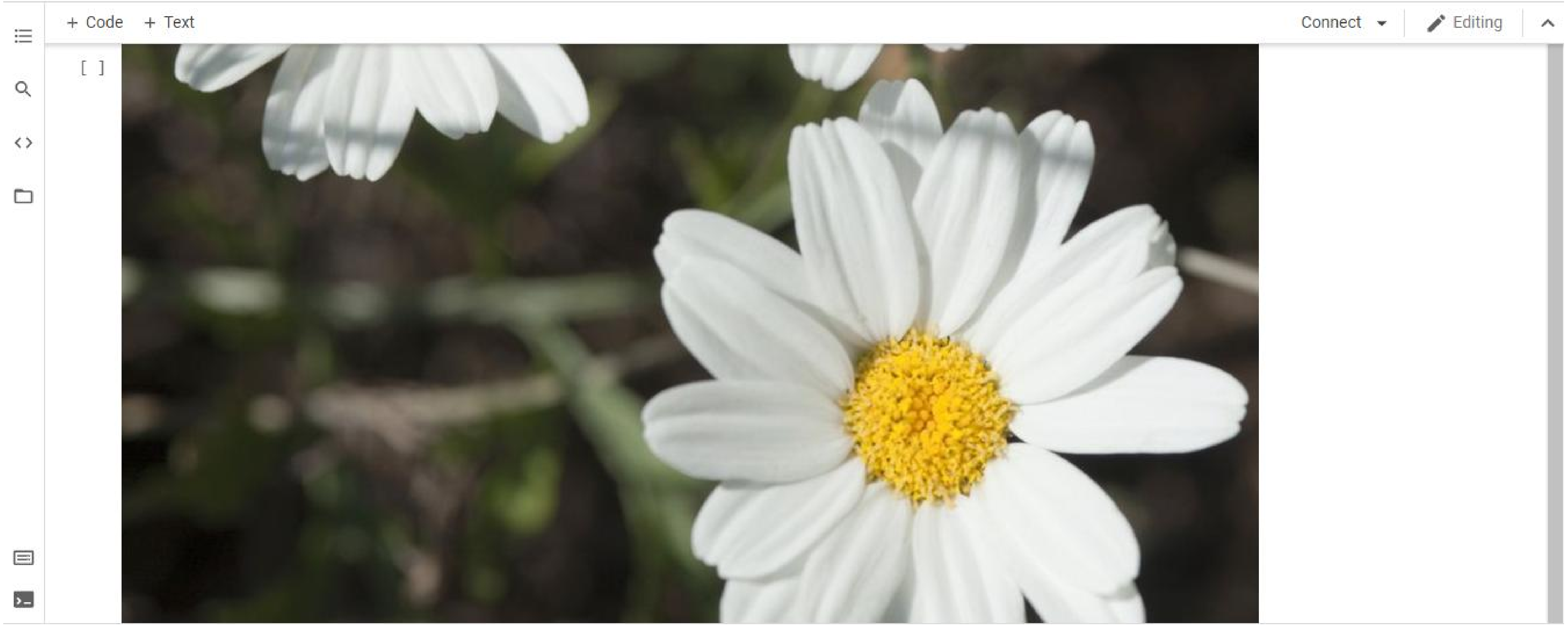
OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc.

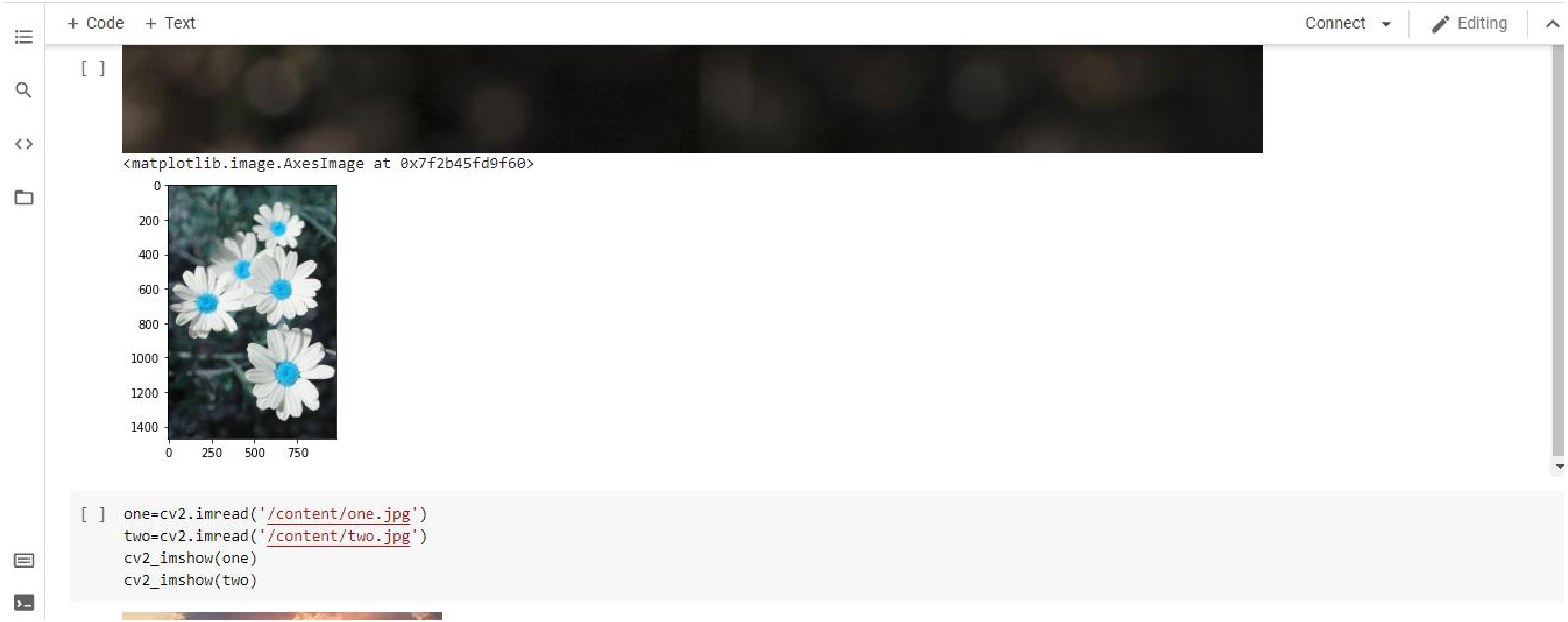
```
[ ] #importing necessary libraries
    import numpy as np
    import matplotlib.pyplot as plt
    import cv2
    from google.colab.patches import cv2_imshow
```

OpenCV programs to (i) read, display, and write a digital image (ii) resize a digital image (iii) convert color image into grayscale (iv) write the image data (2 D) in an image file.

OpenCV programs to (i) read, display, and write a digital image (ii) resize a digital image (iii) convert color image into grayscale (iv) write the image data (2 D) in an image file.

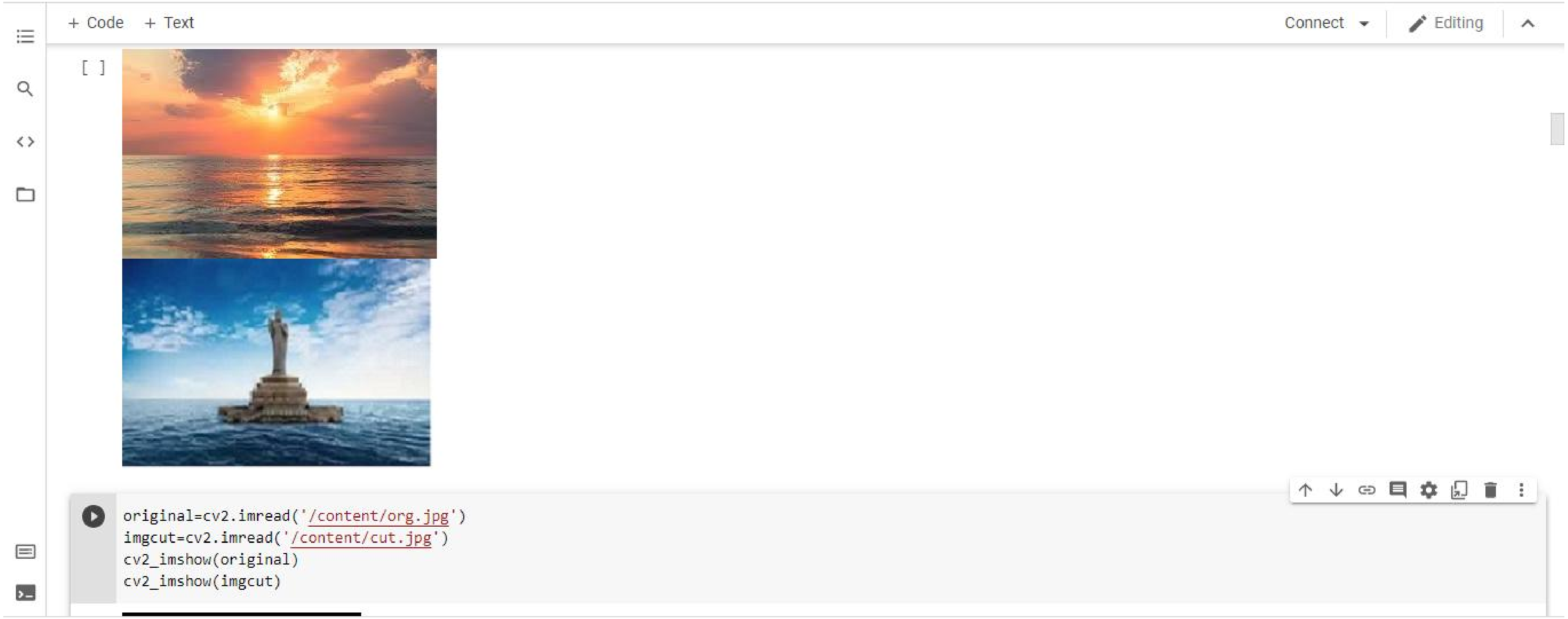
```
[ ] #reading and displaying all input images used
    img = cv2.imread('/content/flower.jpg',1)
    cv2_imshow(img)
    plt.imshow(img)
```


+ Code
+ Text
Connect
Editing
[ ]

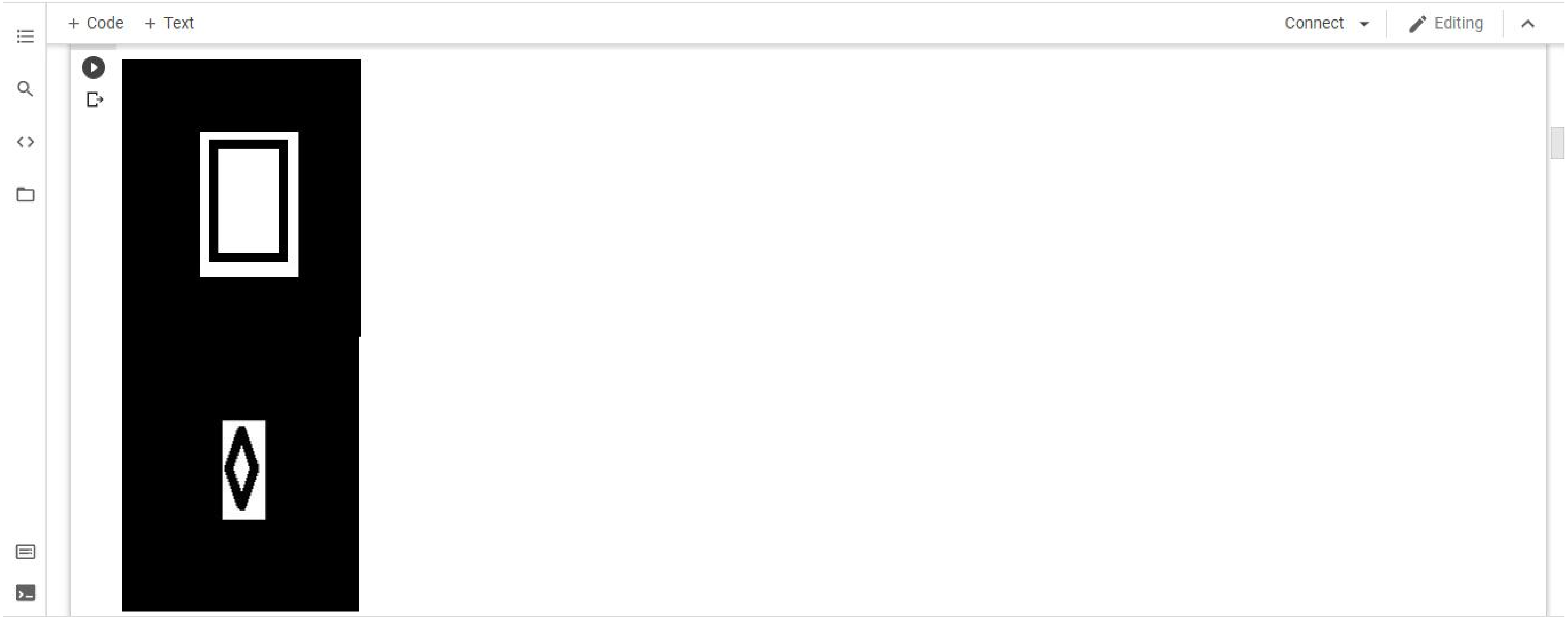


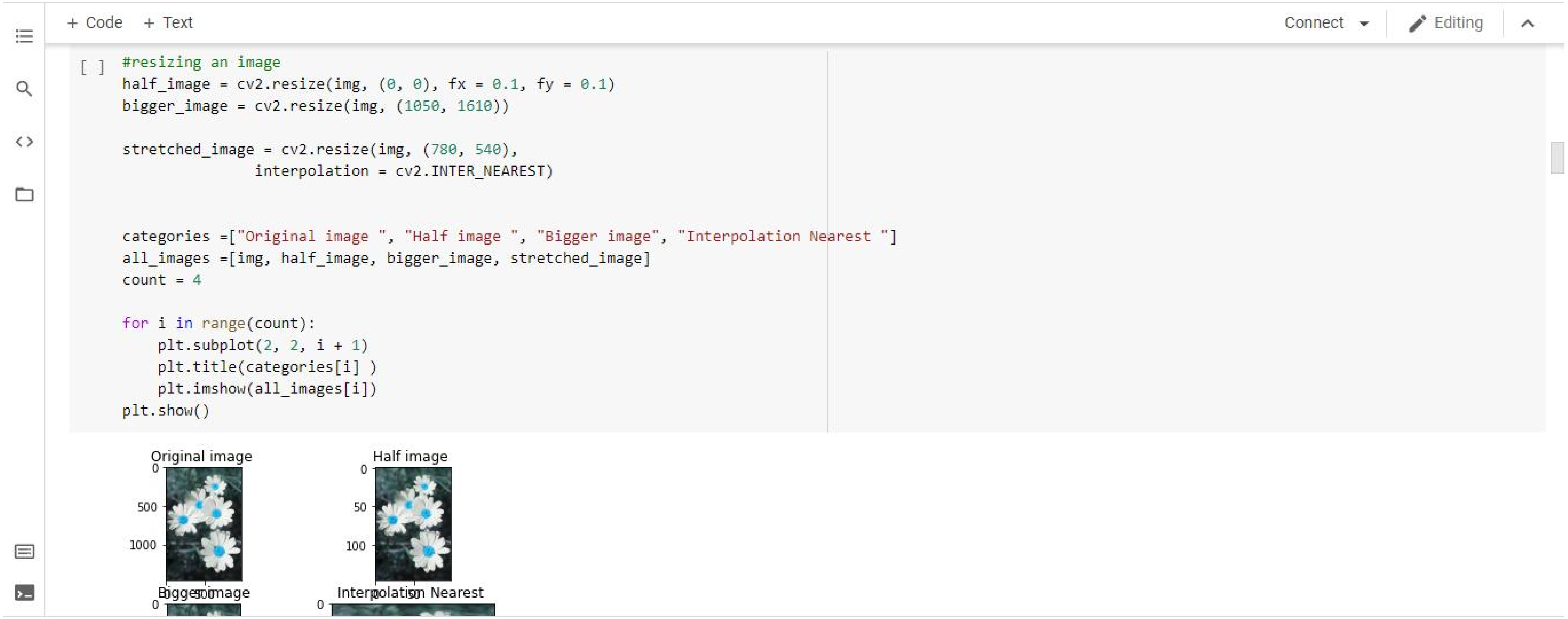
```
<matplotlib.image.AxesImage at 0x7f2b45fd9f60>
```

```
one=cv2.imread('/content/one.jpg')
two=cv2.imread('/content/two.jpg')
cv2_imshow(one)
cv2_imshow(two)
```

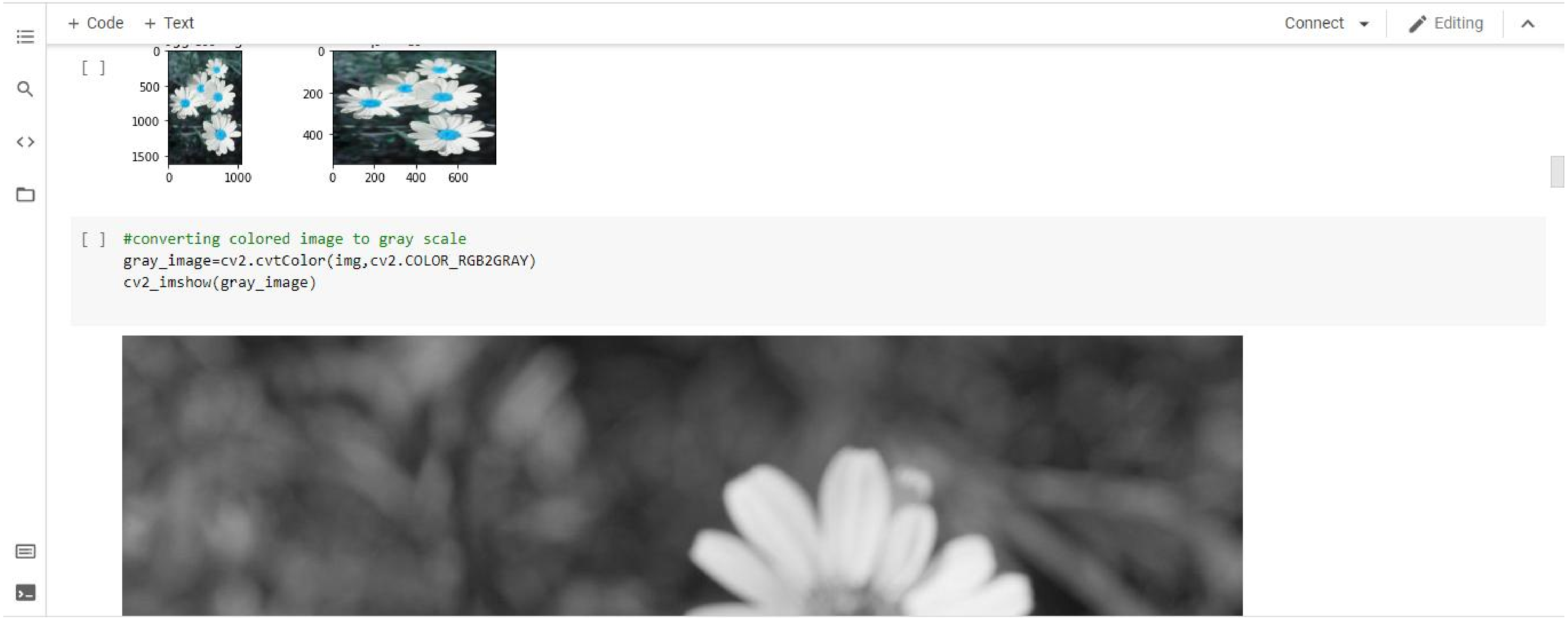


```
original=cv2.imread('/content/org.jpg')
imgcut=cv2.imread('/content/cut.jpg')
cv2_imshow(original)
cv2_imshow(imgcut)
```

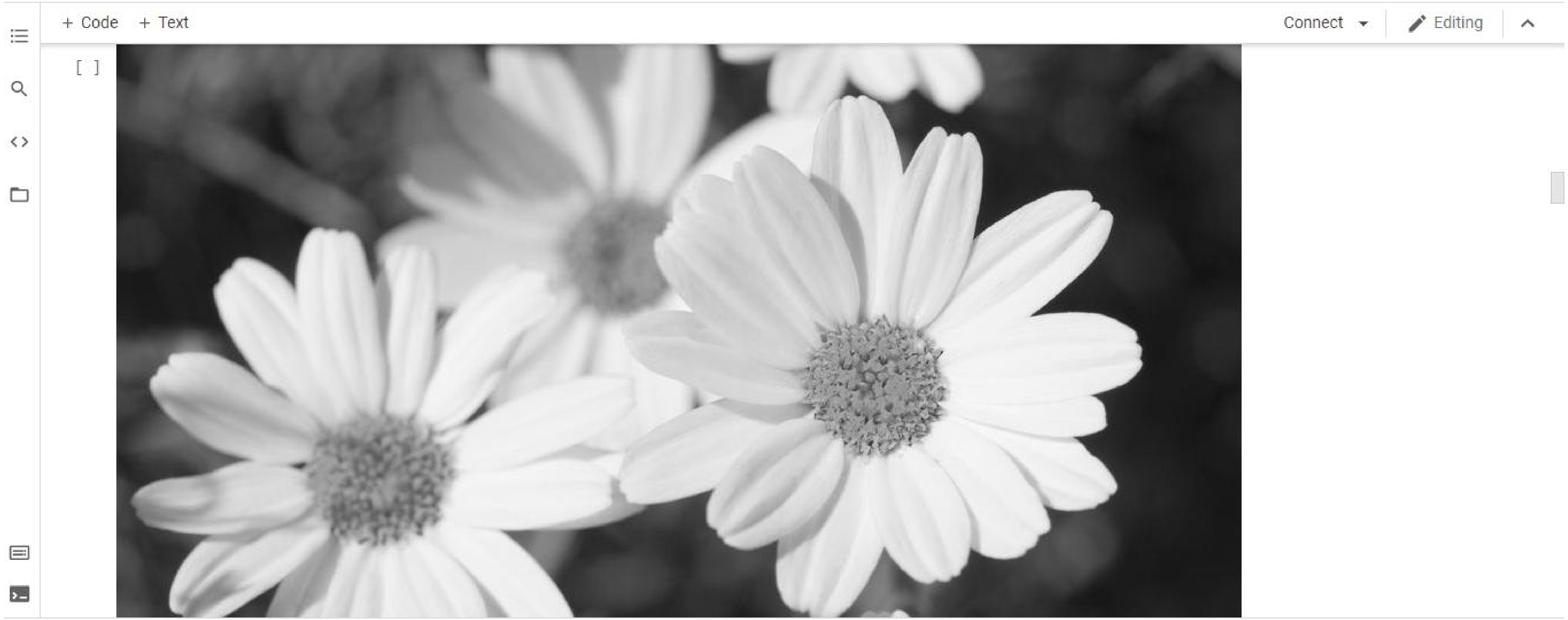

+ Code
+ Text
Connect
Editing



```
#resizing an image
half_image = cv2.resize(img, (0, 0), fx = 0.1, fy = 0.1)
bigger_image = cv2.resize(img, (1050, 1610))

stretched_image = cv2.resize(img, (780, 540),
               interpolation = cv2.INTER_NEAREST)



categories =["Original image ", "Half image ", "Bigger image", "Interpolation Nearest "]
all_images =[img, half_image, bigger_image, stretched_image]
count = 4

for i in range(count):
    plt.subplot(2, 2, i + 1)
    plt.title(categories[i] )
    plt.imshow(all_images[i])
plt.show()
```



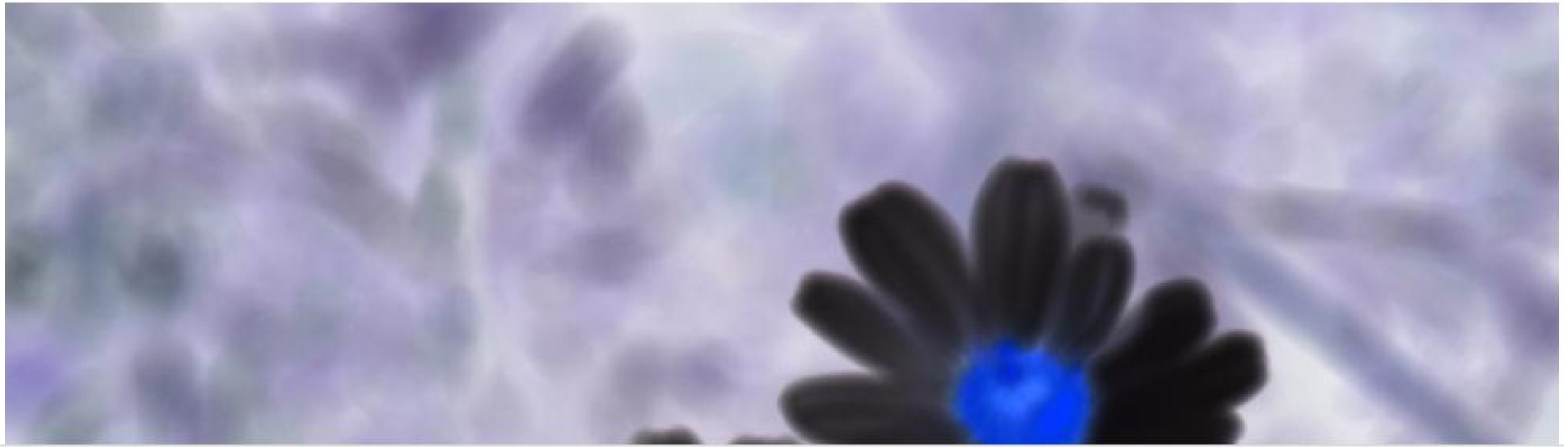
```
#converting colored image to gray scale
gray_image=cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
cv2_imshow(gray_image)
```

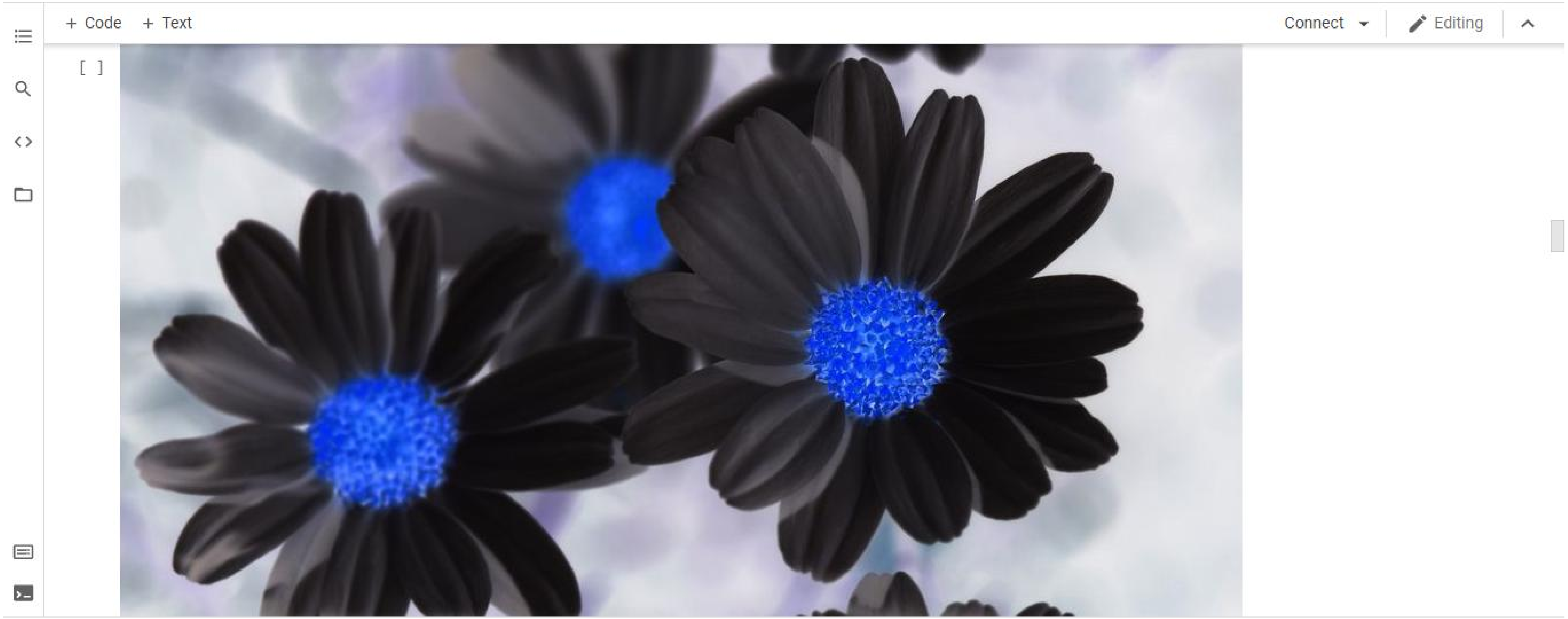

+ Code
+ Text
Connect
Editing
[ ]

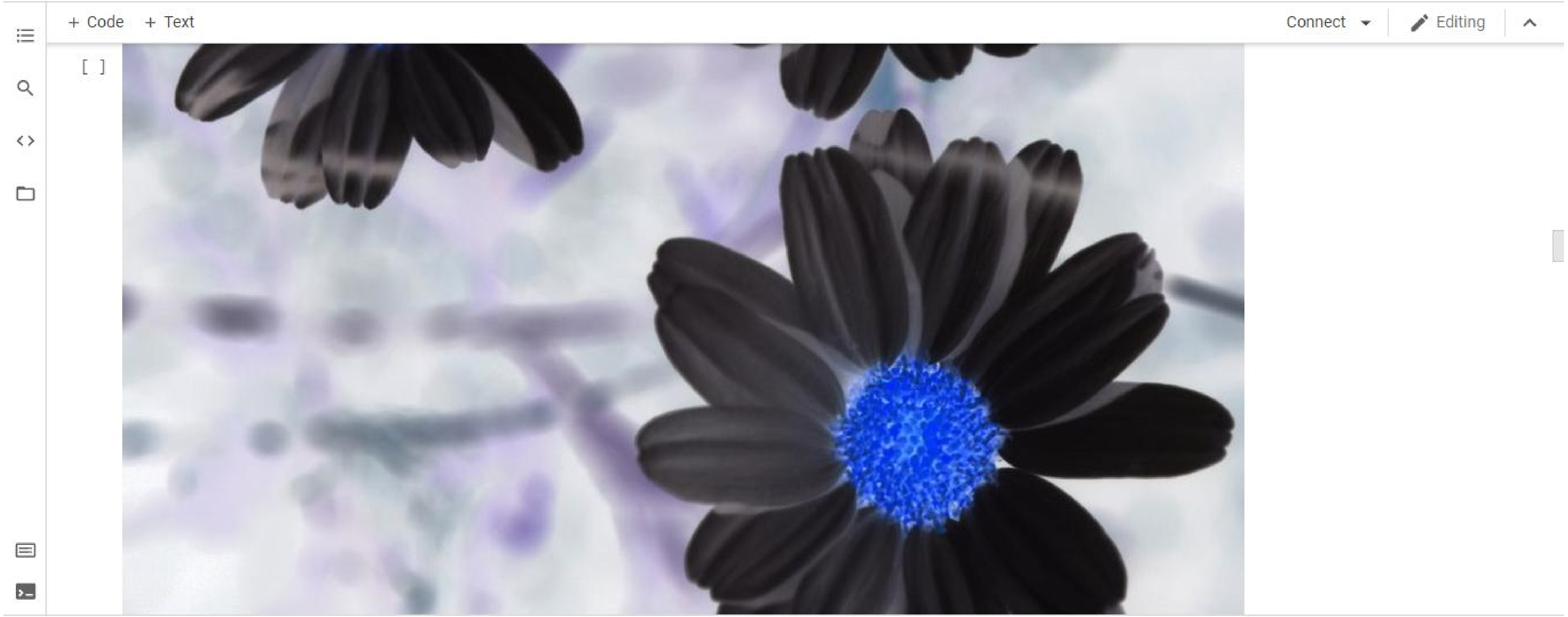
+ Code + Text Connect Editing

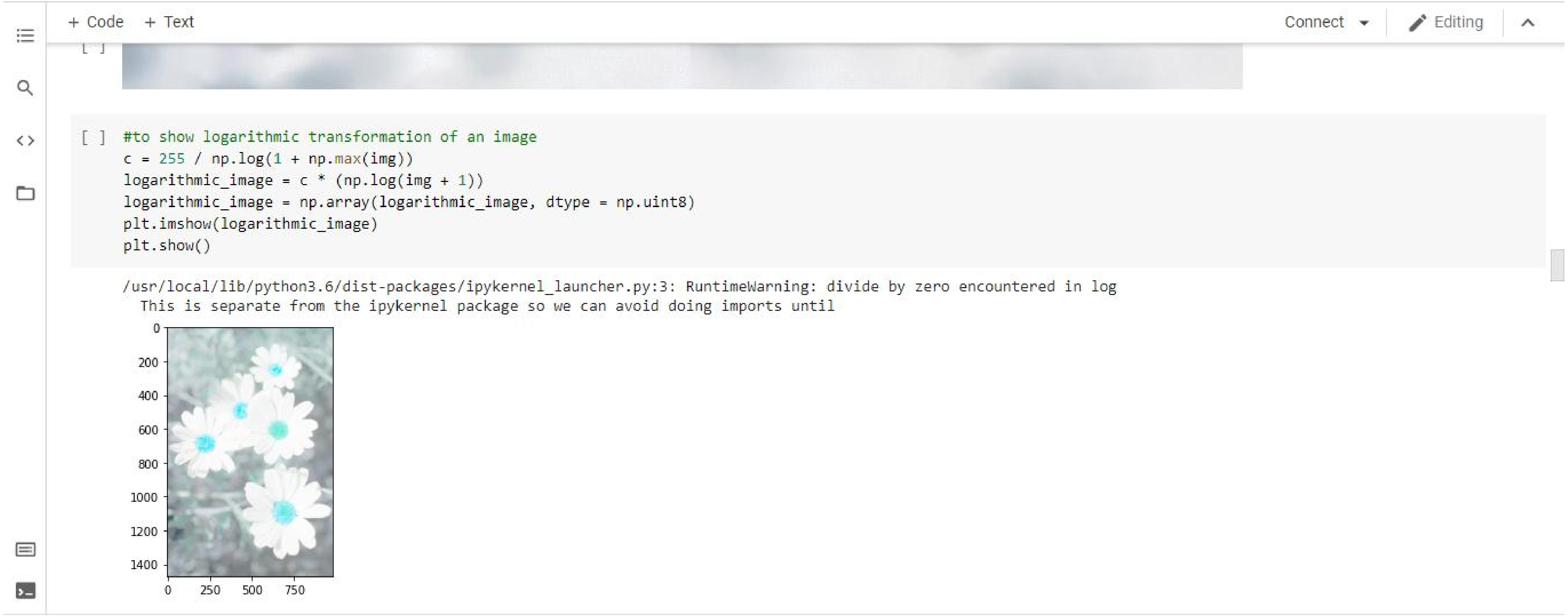
OpenCV programs to perform image enhancement (i) obtain negative of an image (ii) to show logarithmic transformation (iii) to show power transformation (iv) contrast stretching.

```
[ ] #obtain negative of an image
    n_image = cv2.bitwise_not(img)
    cv2_imshow(n_image)
```



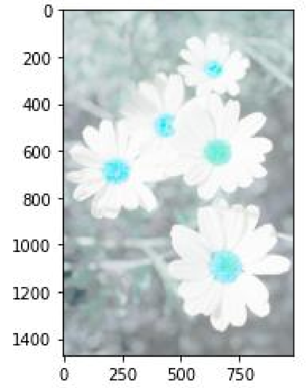

+ Code
+ Text
Connect
Editing
[ ]

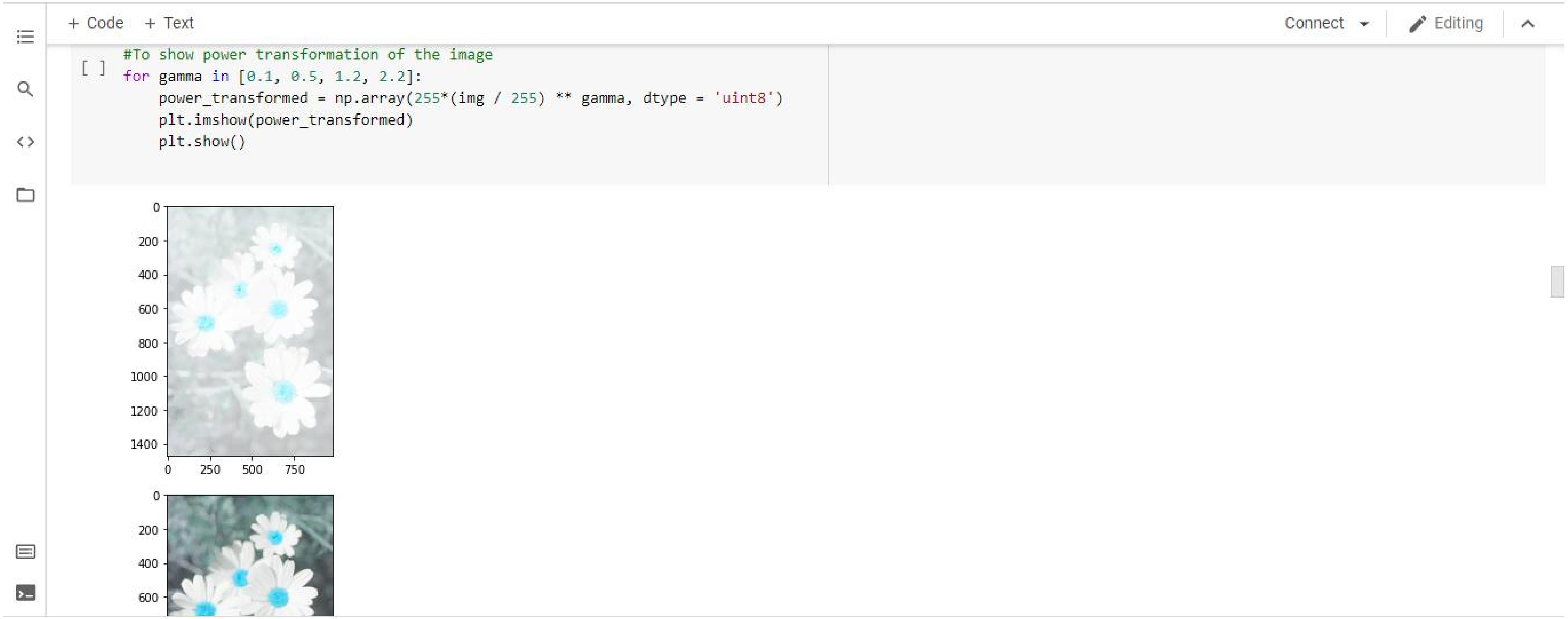

+ Code
+ Text
Connect
Editing
[ ]



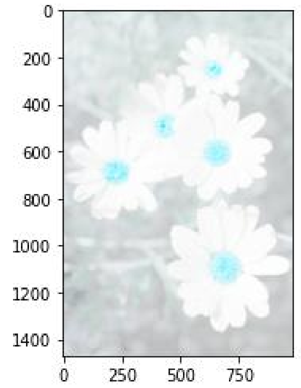
```
[ ]  #to show logarithmic transformation of an image
     c = 255 / np.log(1 + np.max(img))
     logarithmic_image = c * (np.log(img + 1))
     logarithmic_image = np.array(logarithmic_image, dtype = np.uint8)
     plt.imshow(logarithmic_image)
     plt.show()
```

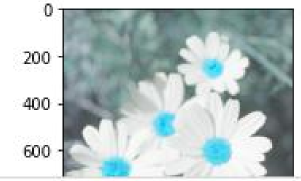
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero encountered in log
  This is separate from the ipykernel package so we can avoid doing imports until
```

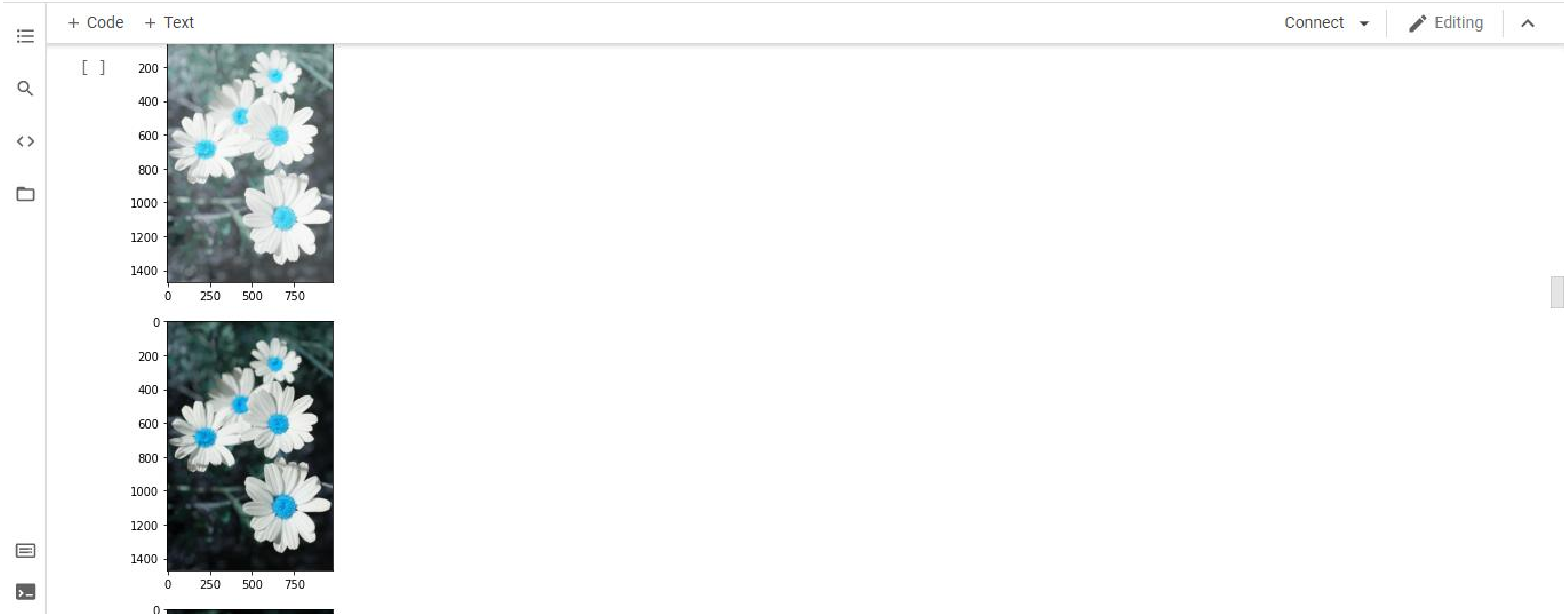


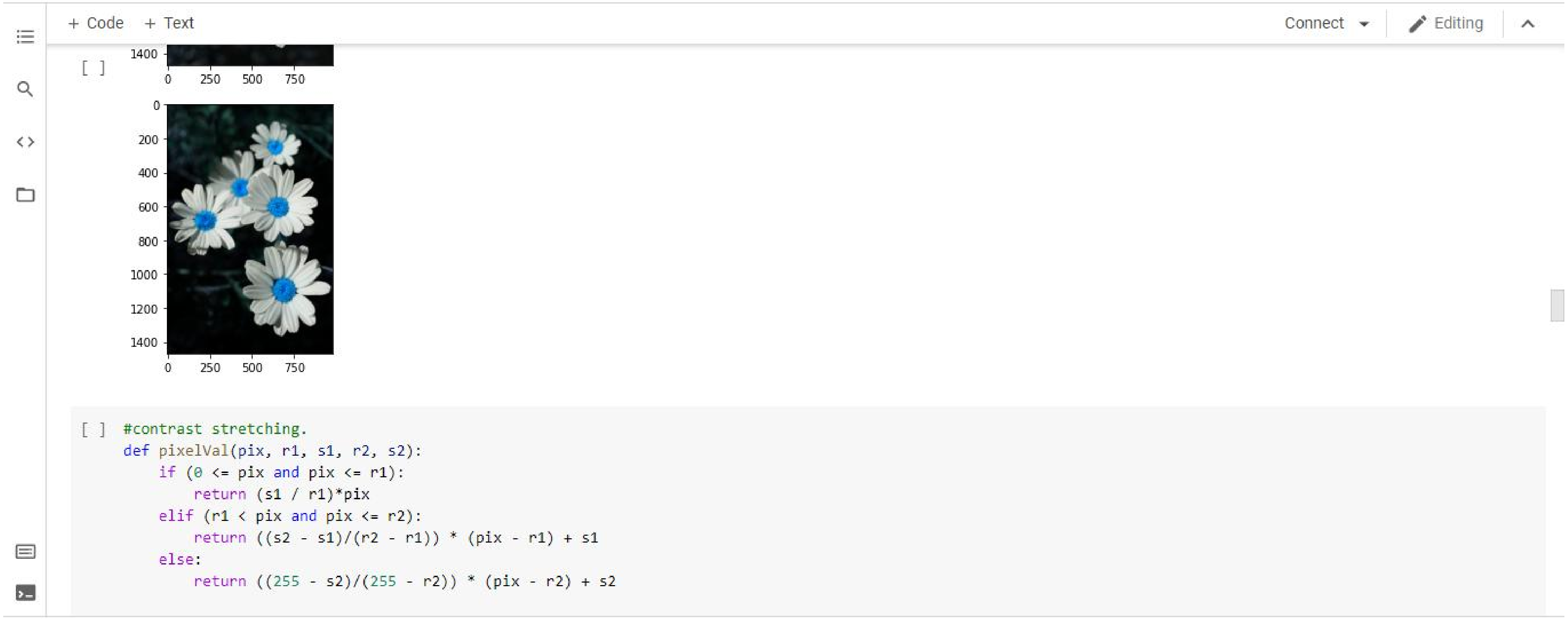


```
#To show power transformation of the image
for gamma in [0.1, 0.5, 1.2, 2.2]:
    power_transformed = np.array(255*(img / 255) ** gamma, dtype = 'uint8')
    plt.imshow(power_transformed)
    plt.show()
```

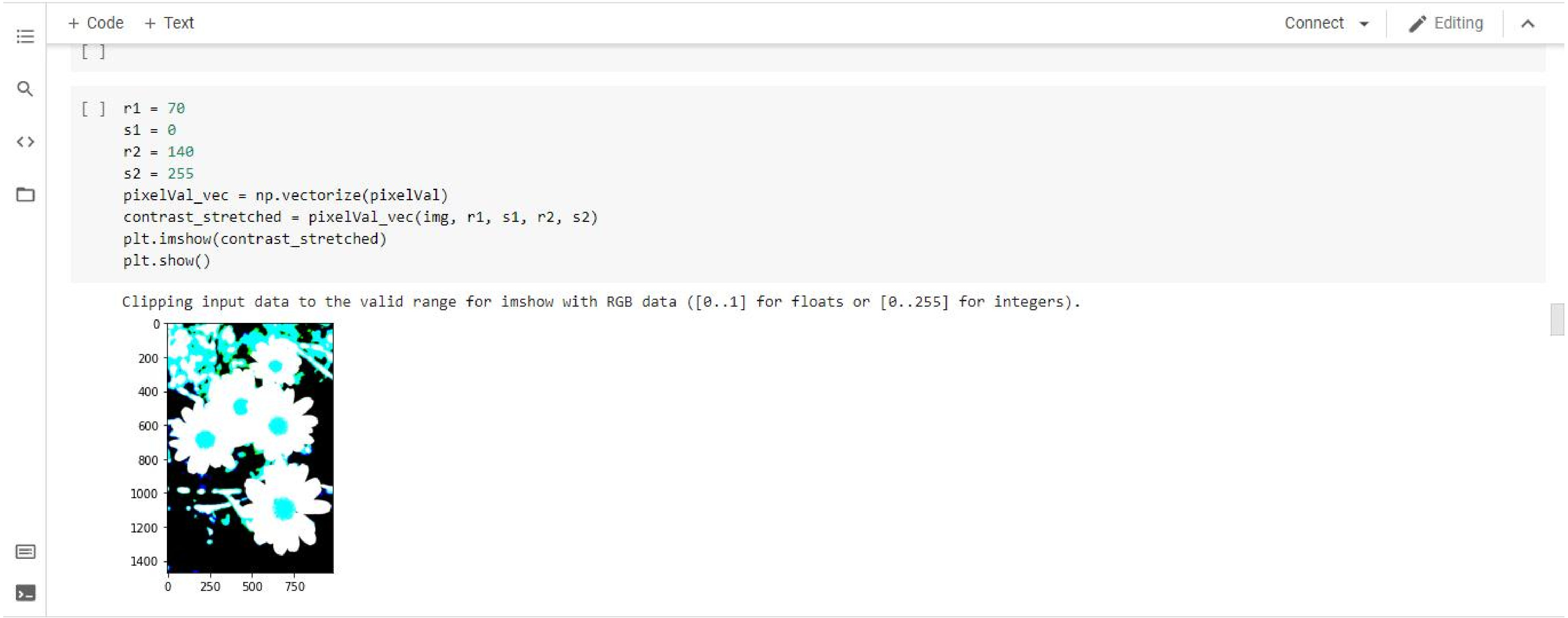





+ Code
+ Text
Connect
Editing
[ ]
200
400
600
800
1000
1200
1400
0
250
500
750
0
200
400
600
800
1000
1200
1400
0
250
500
750
0



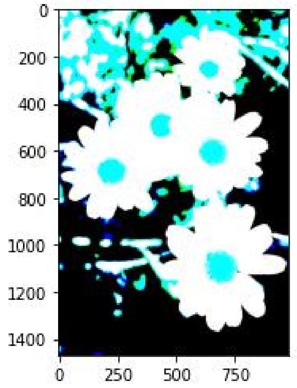
```
#contrast stretching.
def pixelVal(pix, r1, s1, r2, s2):
    if (0 <= pix and pix <= r1):
        return (s1 / r1)*pix
    elif (r1 < pix and pix <= r2):
        return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
    else:
        return ((255 - s2)/(255 - r2)) * (pix - r2) + s2
```

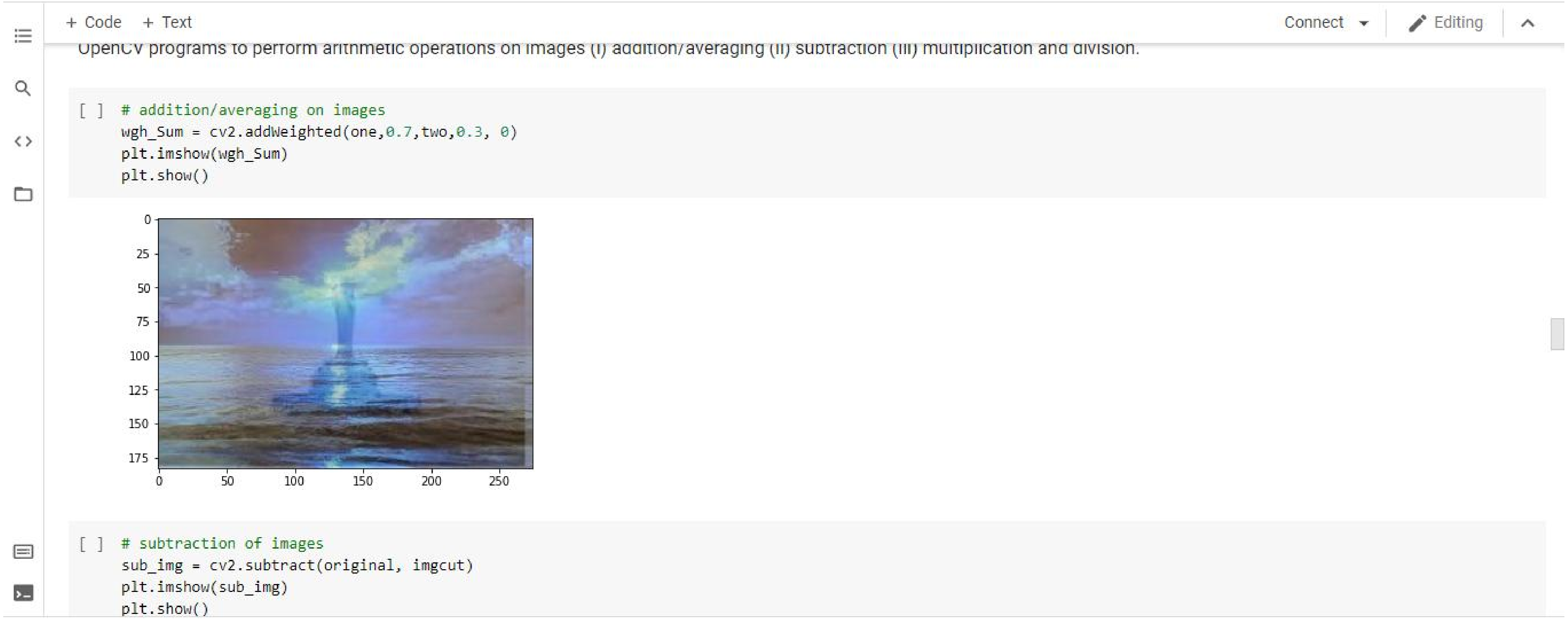


```
[ ]
```

```
[ ] r1 = 70
    s1 = 0
    r2 = 140
    s2 = 255
    pixelVal_vec = np.vectorize(pixelVal)
    contrast_stretched = pixelVal_vec(img, r1, s1, r2, s2)
    plt.imshow(contrast_stretched)
    plt.show()
```

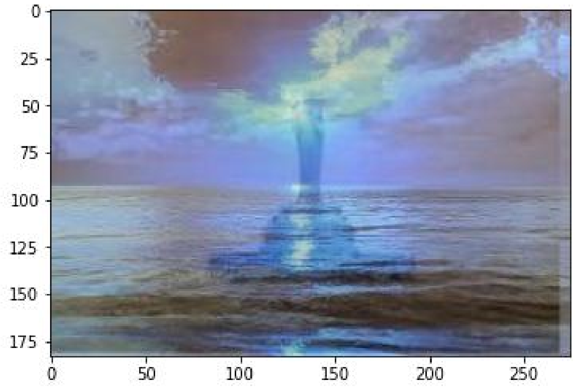
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



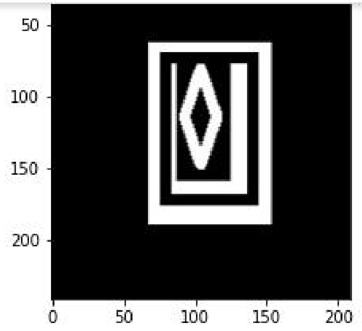


OpenCV programs to perform arithmetic operations on images (i) addition/averaging (ii) subtraction (iii) multiplication and division.

```
[ ] # addition/averaging on images
    wgh_Sum = cv2.addWeighted(one,0.7,two,0.3, 0)
    plt.imshow(wgh_Sum)
    plt.show()
```

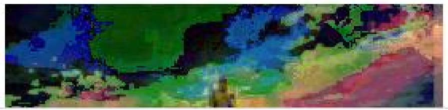


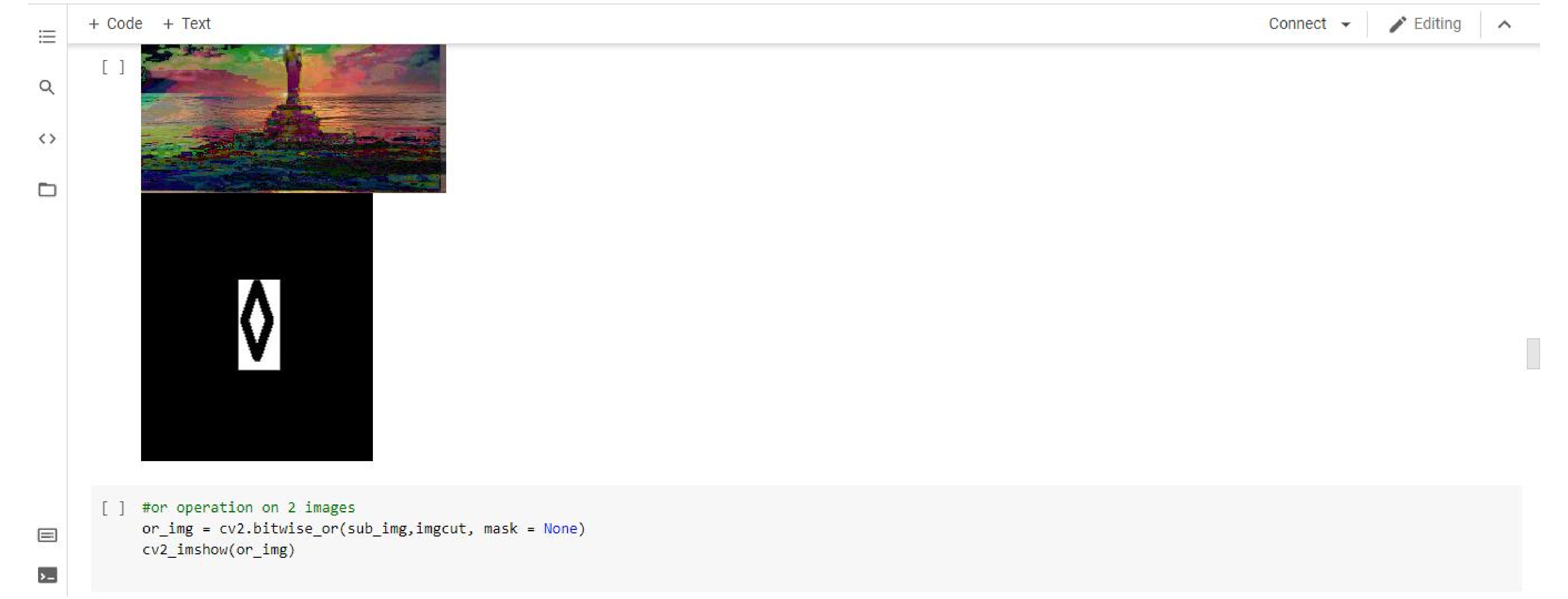
```
[ ] # subtraction of images
    sub_img = cv2.subtract(original, imgcut)
    plt.imshow(sub_img)
    plt.show()
```

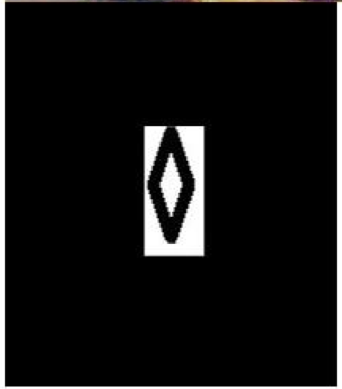


OpenCV programs to perform logical operations between two images (i) AND operation (ii) OR operation (iii) NOT Operation (iv) EX-OR operation and hence watermarking.

```
#AND operation on 2 images
and_Img1 = cv2.bitwise_and(one,two, mask = None)
and_Img2=cv2.bitwise_and(original,imgcut,mask=None)
cv2_imshow(and_Img1)
cv2_imshow(and_Img2)
```

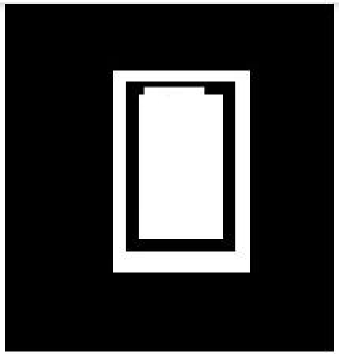




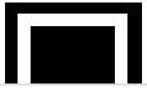


```
#or operation on 2 images
or_img = cv2.bitwise_or(sub_img,imgcut, mask = None)
cv2_imshow(or_img)
```

+ Code + Text Connect Editing

[ ]


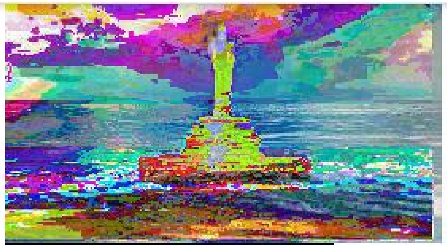
```
[ ] #Not operation on 2 images
    not_Img1 = cv2.bitwise_not(original, mask = None)
    not_Img2 = cv2.bitwise_not(two, mask = None)
    cv2_imshow(not_Img1)
    cv2_imshow(not_Img2)
```

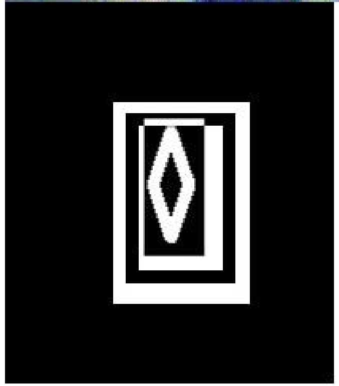


+ Code + Text Connect Editing

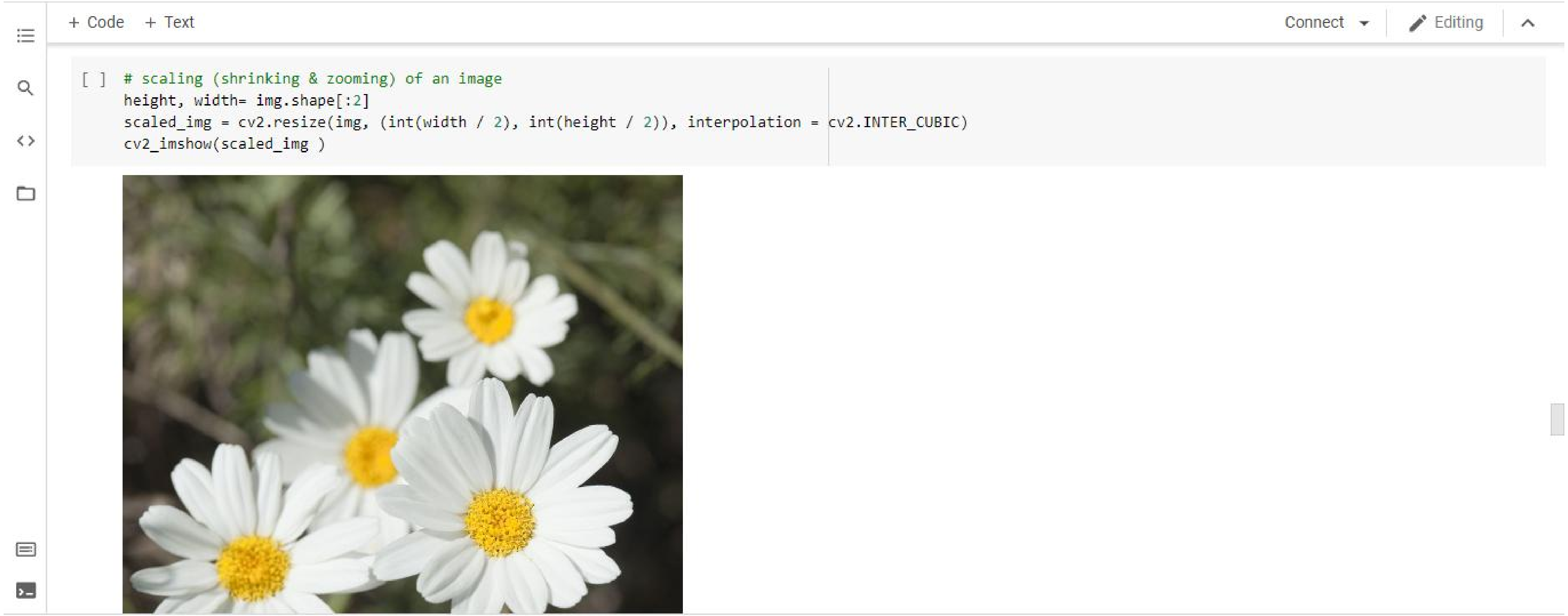
[ ]

```
[ ] #ex-or operation on 2 image
    xor_Img1 = cv2.bitwise_xor(one ,two, mask = None)
    xor_Img2=cv2.bitwise_xor(original,imgcut,mask=None)
    cv2_imshow(xor_Img1)
    cv2_imshow(xor_Img2)
```

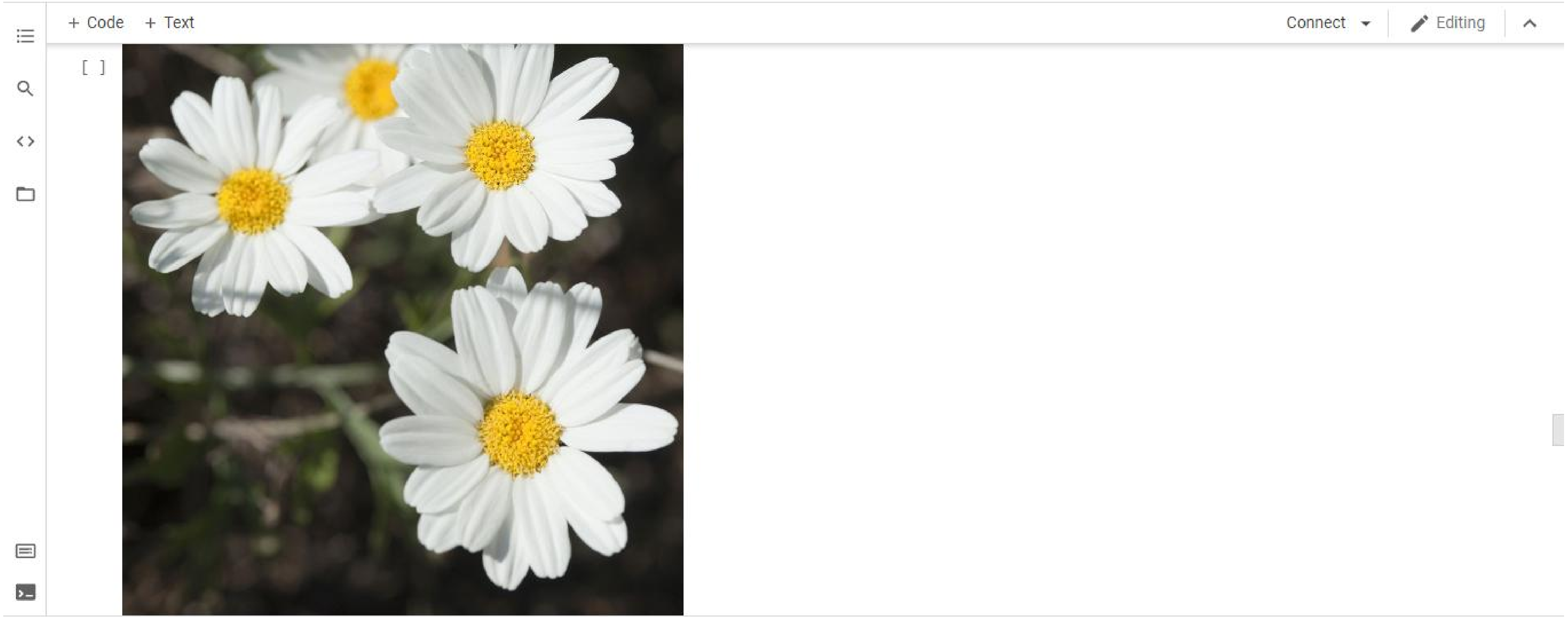


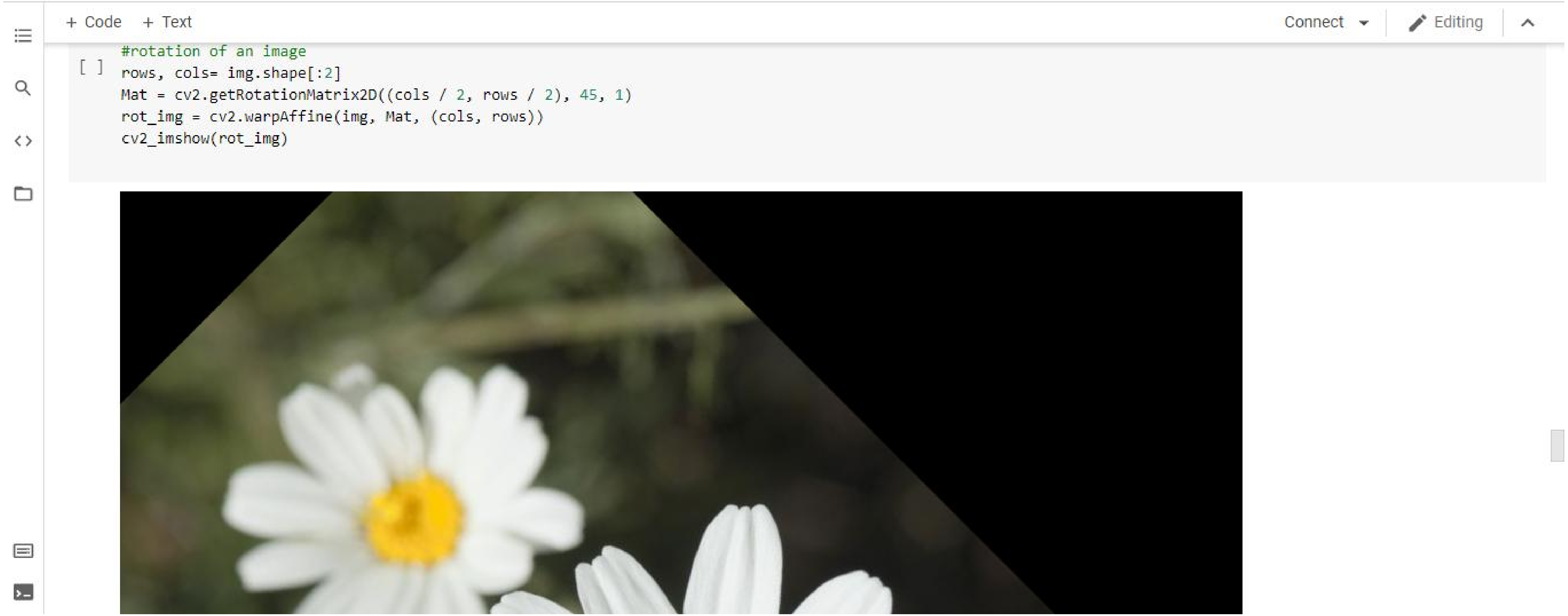


OpenCV programs showing geometric transformation of an image (i) scaling (shrinking & zooming) (ii) translation (iii) rotation (iv) shear (horizontal & vertical).

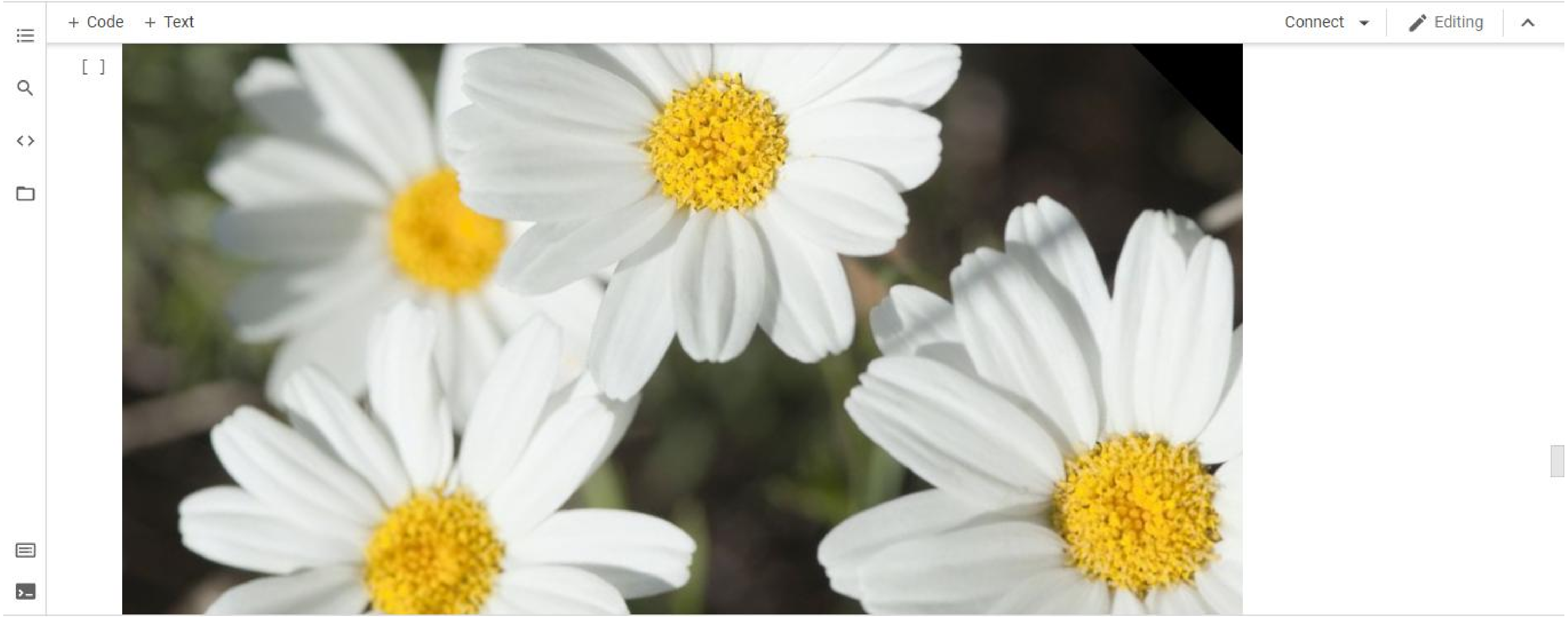


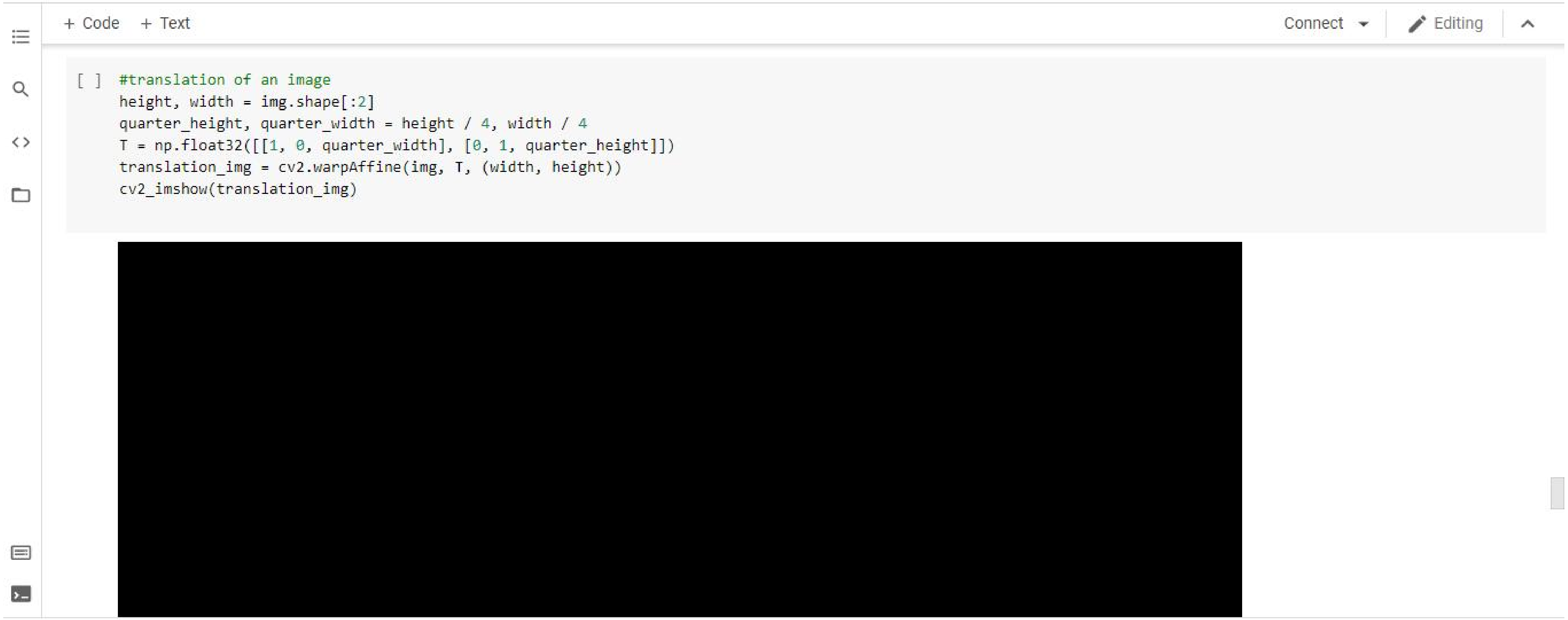
```
# scaling (shrinking & zooming) of an image
height, width= img.shape[:2]
scaled_img = cv2.resize(img, (int(width / 2), int(height / 2)), interpolation = cv2.INTER_CUBIC)
cv2_imshow(scaled_img )
```


+ Code
+ Text
Connect
Editing
[ ]

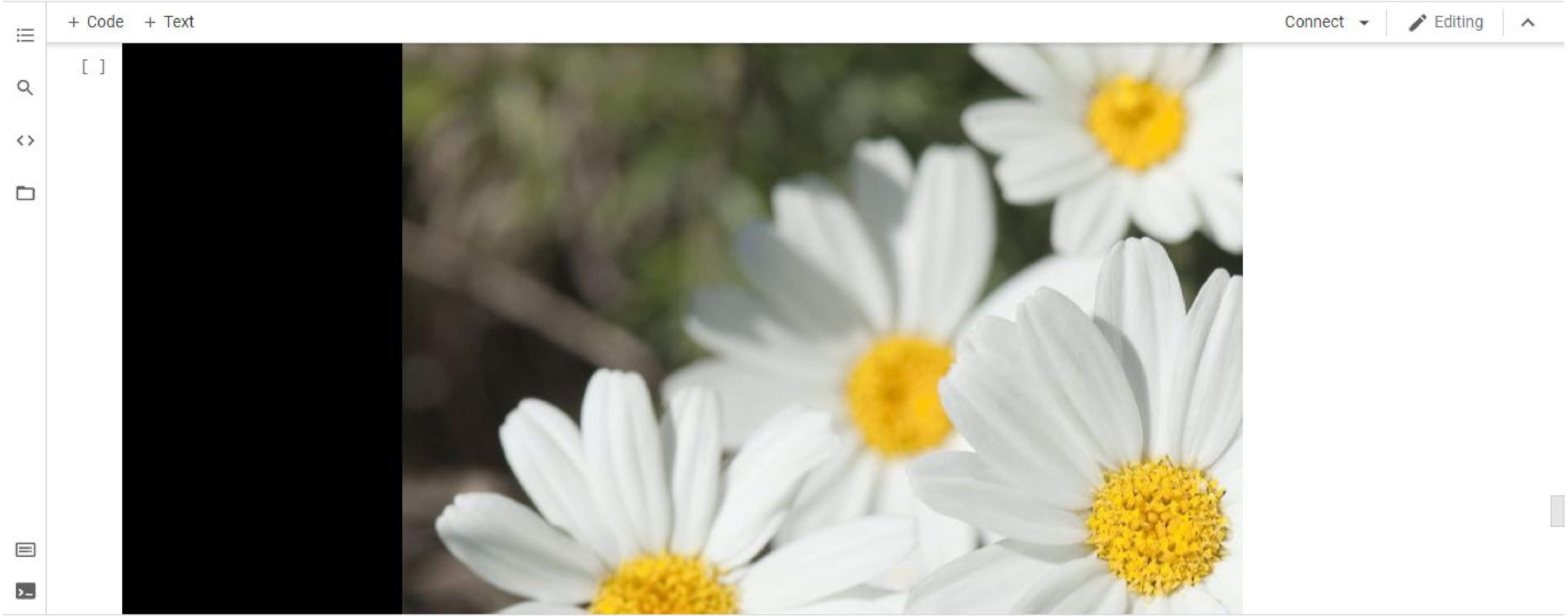


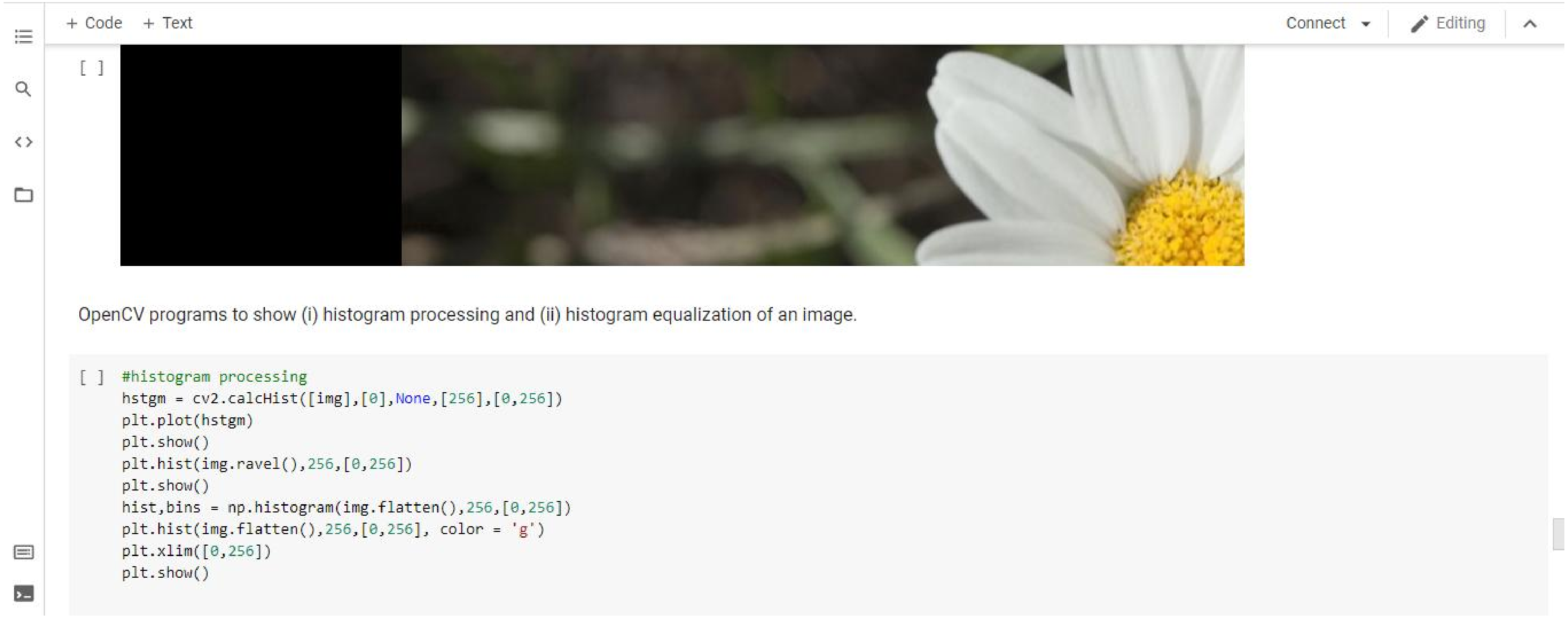
```
#rotation of an image
rows, cols= img.shape[:2]
Mat = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
rot_img = cv2.warpAffine(img, Mat, (cols, rows))
cv2_imshow(rot_img)
```


+ Code
+ Text
Connect
Editing
[ ]



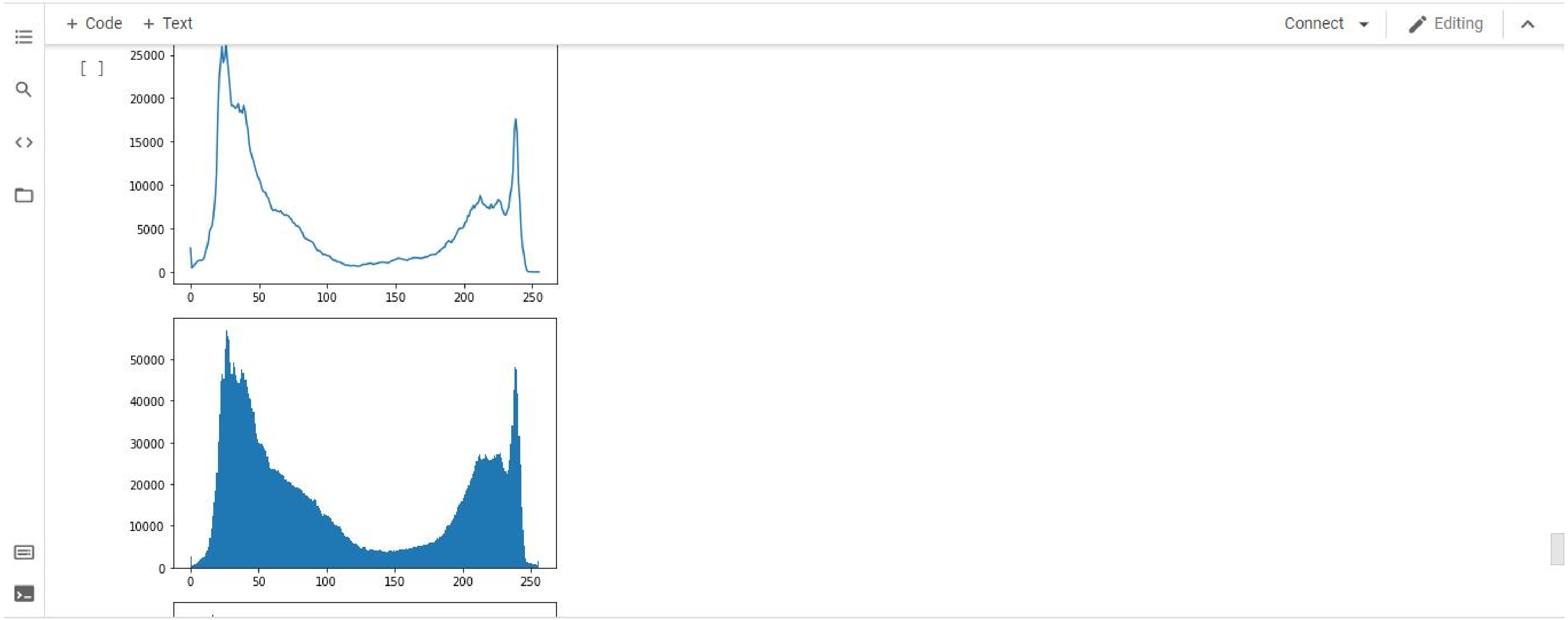
```
[ ] #translation of an image
    height, width = img.shape[:2]
    quarter_height, quarter_width = height / 4, width / 4
    T = np.float32([[1, 0, quarter_width], [0, 1, quarter_height]])
    translation_img = cv2.warpAffine(img, T, (width, height))
    cv2_imshow(translation_img)
```

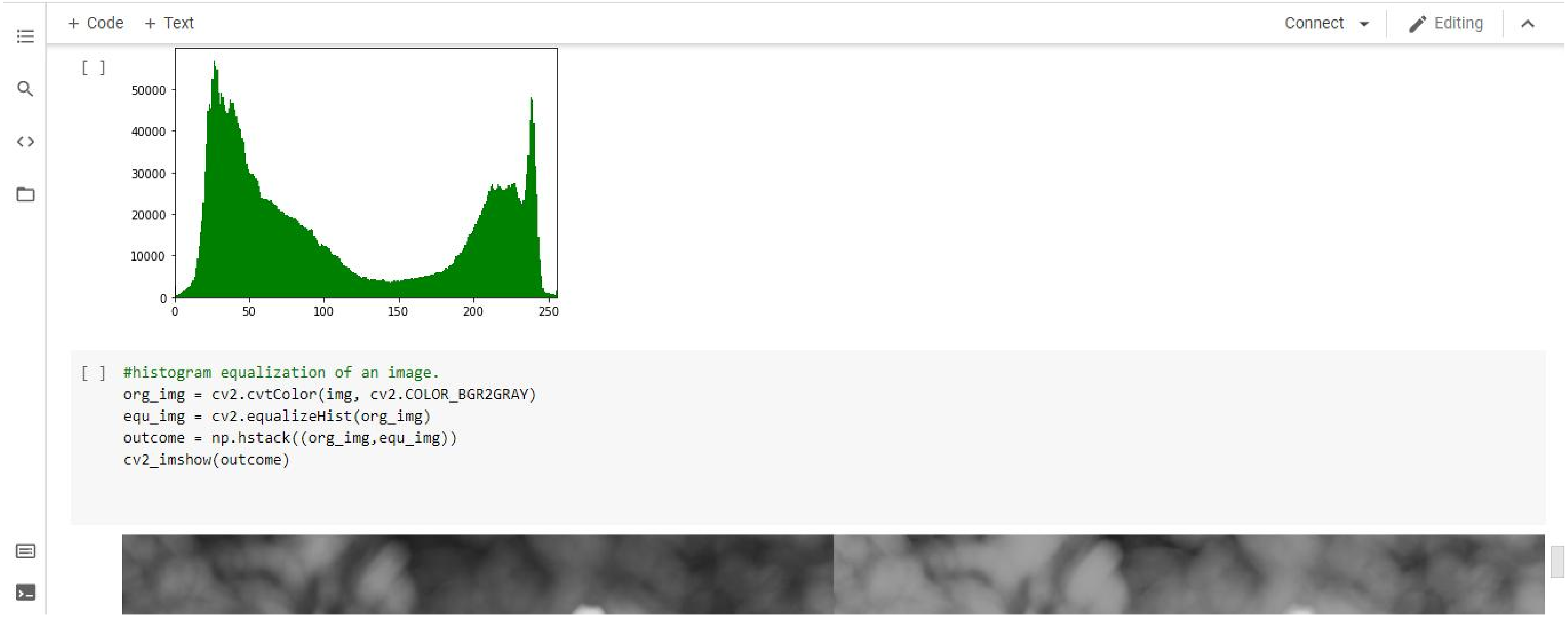

+ Code
+ Text
Connect
Editing
[ ]



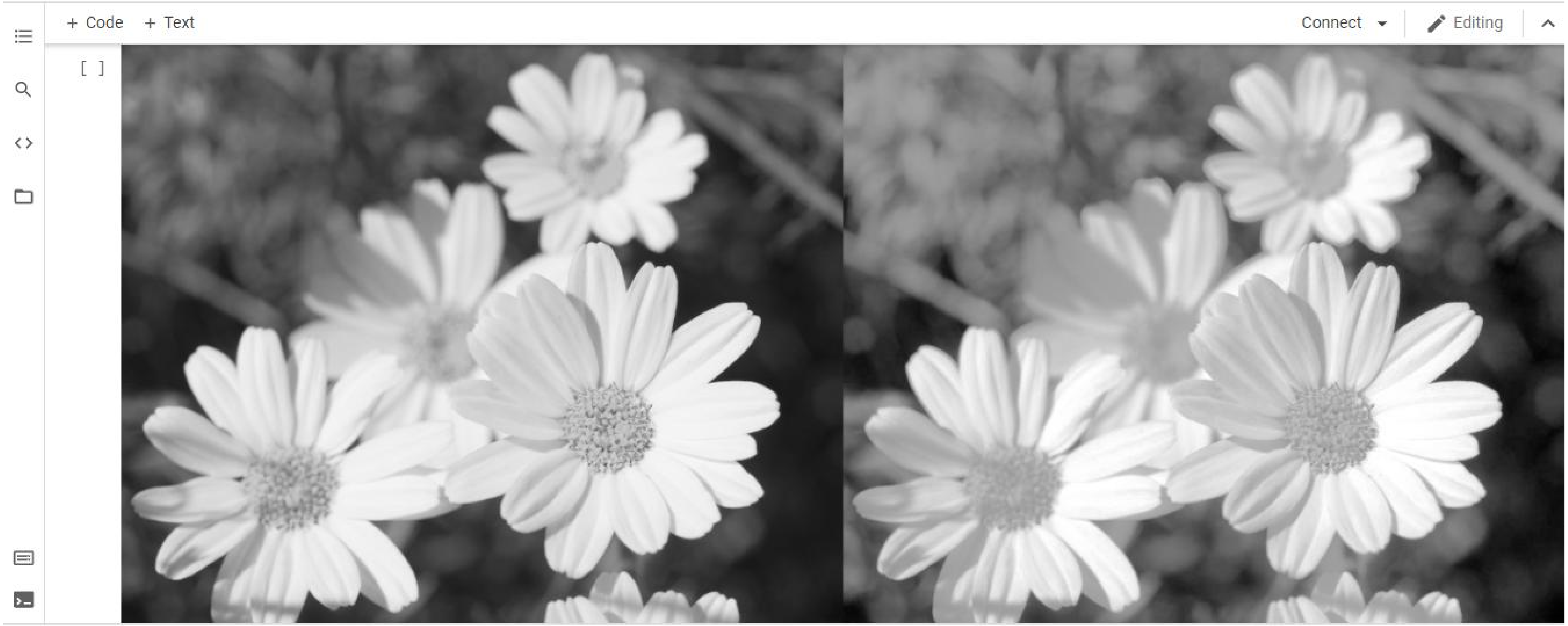
OpenCV programs to show (i) histogram processing and (ii) histogram equalization of an image.

```
[ ] #histogram processing
    hstgm = cv2.calcHist([img],[0],None,[256],[0,256])
    plt.plot(hstgm)
    plt.show()
    plt.hist(img.ravel(),256,[0,256])
    plt.show()
    hist,bins = np.histogram(img.flatten(),256,[0,256])
    plt.hist(img.flatten(),256,[0,256], color = 'g')
    plt.xlim([0,256])
    plt.show()
```


+ Code
+ Text
Connect
Editing
[ ]
25000
20000
15000
10000
5000
0
0
50
100
150
200
250
50000
40000
30000
20000
10000
0
0
50
100
150
200
250



```
#histogram equalization of an image.
org_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
equ_img = cv2.equalizeHist(org_img)
outcome = np.hstack((org_img,equ_img))
cv2_imshow(outcome)
```


+ Code
+ Text
Connect
Editing
[ ]