پروژه دوم درس سیستم های چندرسانه ای

آنوشا شریعتی ۹۹۲۳۰۴۱

مهشاد اکبری سریزدی ۹۹۲۳۰۹۳

سوال ۱:

در ابتدا کتابخانه های مورد نیاز برای حل این سوال را اضافه کرده و تابع imshow را تعریف میکنیم. سپس به صورت زیر تصویر اصلی و تغییر یافته را باز کرده ونمایش میدهیم. طول و عرض تصویر را محاسبه کرده و در طول برنامه آن را عدد ثابت (۹۷۴و۸۲۴) میگیریم.

adding the libraries

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

# Define our imshow function
def imshow(title = "Image", image = None, size = 10):
    w, h = image.shape[0], image.shape[1]
    aspect_ratio = w/h
    plt.figure(figsize=(size * aspect_ratio,size))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.show()
```

[183] ✓ 0.0s

```
image1 = cv2.imread('C:/Users/My/Desktop/HW_MULTI/Original_image.jpg')
image2 = cv2.imread('C:/Users/My/Desktop/HW_MULTI/transformed_image.jpg')
imshow("original",image1)
imshow("transformed",image2)
height, width = image1.shape[:2]
print(height,width)
```

✓ 0.9s

دراین سوال از ما خواسته شده تبدیل های مورد نیاز برای تبدیل تصویر اصلی به تصویر تغییر یافته را بیابیم و با اعمال عکس تبدیل ها روی تصویر تغییر یافته تصویر اصلی را به دست آوریم.

الگوریتم پیاده شده برای حل این سوال به این صورت است که در ابتدا کانتورهای شکل اصلی را یافته و آن را به عنوان region of interest تعریف میکنیم. همچنین با کمک مومنت مرکز مختصات کانتور را پیدا کرده وبه کمک bounding box ها طول و عرض کانتور را مشخص میکنیم. این اطلاعات در ادامه مسئله مورد نیاز واقع میشوند.

برای راحتی کار تابع find_contours را به گونه ای تعریف کردیم که خروجی آن علاوه بر کانتور بزرگتر، مرکز کانتور و طول و عرض آن هم باشد.

```python
def find_contours(image):

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    largest_contour = max(contours, key=cv2.contourArea)

    cv2.drawContours(image, [largest_contour], -1, (0, 255, 0), 3)

    M = cv2.moments(largest_contour)
    if M["m00"] != 0:
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
    else:
        cX, cY = 0, 0

    x, y, w, h = cv2.boundingRect(largest_contour)

    return (cX, cY), largest_contour, (x, y, w, h)
```

باید در نظر گرفت که تغییرات باید روی کانتور اعمال شوند نه روی کل تصویر. پس کانتور به دست آمده را کراپ میکنیم و تغییرات را روی آن اعمال میکنیم. تغییرات اعمال شده روی تصویر شامل translation,rotation,scaling,shearing,flipping میشوند که ترتیب و اندازه بردار تبدیل آن را باید با سعی وخطا اندازه گیری کرد. در نظر گرفته شود که تابع تبدیل هر تبدیل از فرمول های زیر به دست می آید.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

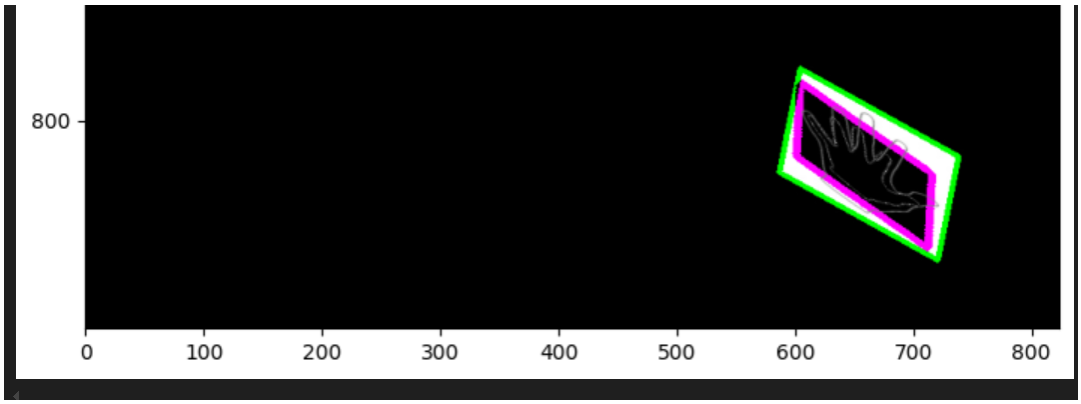
در ادامه کانتور های مربوط به شکل تغییر یافته را در آورده وبا تابع گفته شده مرکز کانتور را به دست می آوریم. با این کار میتوان سعی کرد تا مرکز مختصات شکل اصلی پس از هر تغییر به مرکز مختصات شکل تغییر یافته نزدیکتر شود و در آخر بر ان منطبق شود. با این روش میتوان ضرایب translation یا انتقال را اندازه گرفت. همچنین میتوان طول و عرض تصویر تغییر یافته را محاسبه کرد که با محاسبه نسبت طول و عرض ضریب های scaling را به دست اورد.

```
#transformed image
center2, contour2,(x2,y2,w2,h2) = find_contours(image2)
cv2.drawContours(image2, contour2, -1, (0,255,0), thickness = 1)
imshow("contours", image2)
print("transforned center:",center2)
print(w2,h2)
```

در ادامه تبدیلات را به ترتیب روی شکل اصلی پیاده سازی میکنیم. هرکدام از ضرایب با سعی وخطا و اندازه گیری های پی درپی محاسبه شده است و کد زیر نتیجه نهایی است. بعد از هرمرحله برای مشاهده میزان تطابق میتوان از عملگرهای بیت به بیت مانند or استفاده کرد. این دستور شکل به دست آمده از تبدیل وشکل نهایی را روی هممی اندازد و میتوان بر اساس میزان خطا ضرایب تبدیل ها را اصلاح نمود.

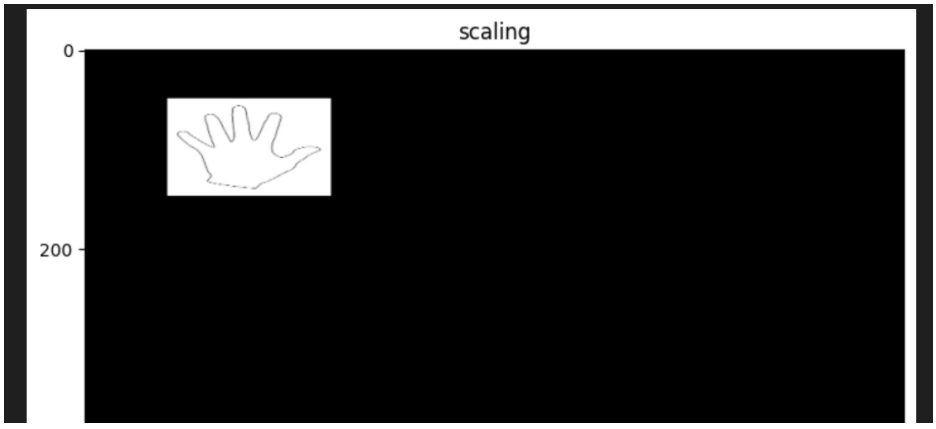
```
bitwiseor = cv2.bitwise_xor(img_translation, image2)
imshow("bitwiseXor", bitwiseor)
```

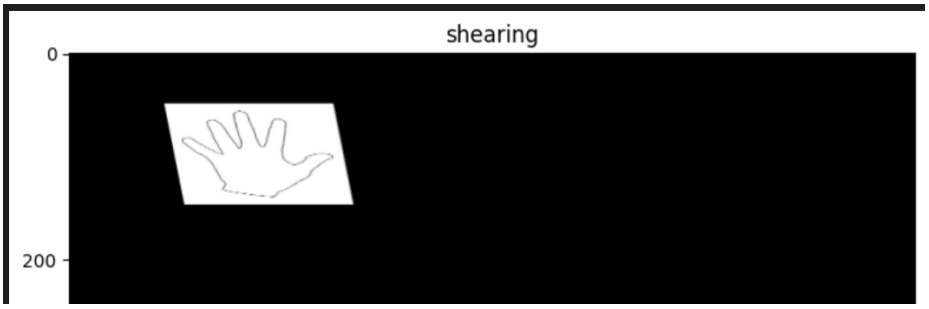
[172] ✓ 0.3s

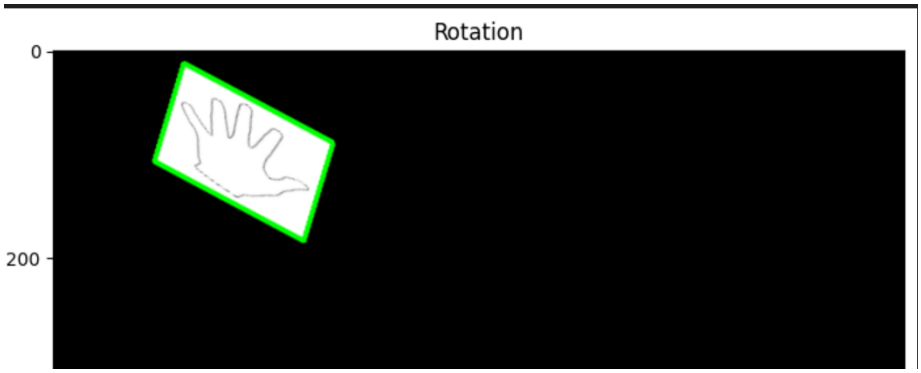


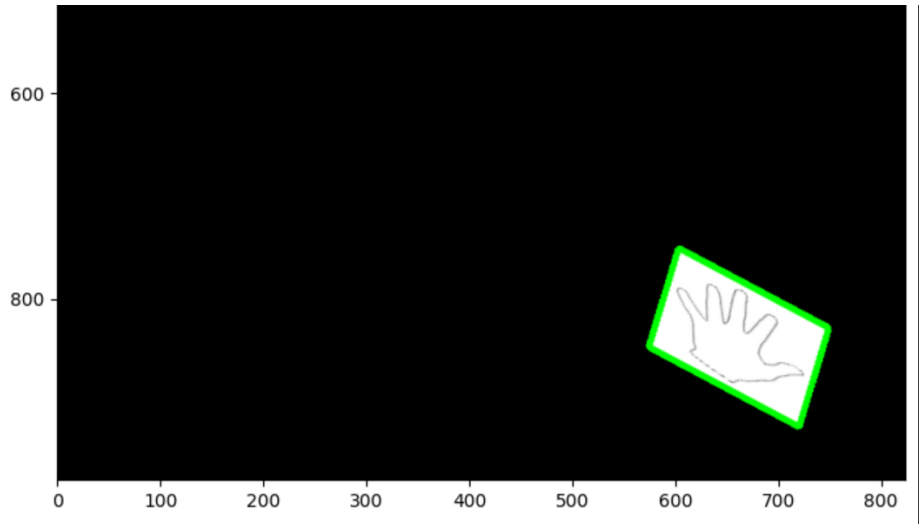
تبدیلات به این ترتیب انجام میشوند. ابتدا تصویر حول محور عمودی قرینه شده سپس با ماتریس اسکیلینگ T1 اسکیل میشود. سپس shearing با ماتری ضرایب t0 انجام میشود. چوندر مرحله بعد دوران را انجام میدهیم و نیاز داریم دوران حول مرکزشکل باشد پس در این مرحله با اجرای تابع تعریف شده پیدا کردن کانتور مرکز مختصات را به دست می آوریم. در مرحله بعد حول مختصات به دست آمده شکل را دوران میدهیم.مرحله آخر انتقال است. برای ان که بفهمیم چقدر انتقال داریم میتوانیم مرکز مختصات کانتور در این مرحله را به دست آورده و از مرکز مختصات حساب شده برای شکل تغییر یافته کم کنیم تا ماتریس ضرایب انتقال به دست آید. در ادامه تصویر به دست آمده بعد از هر مرحله تبدیل آورده شده است و در انتها تصویر نهایی با تصویر تبدیل شده داده شده در پروژه مقایسه شده است.

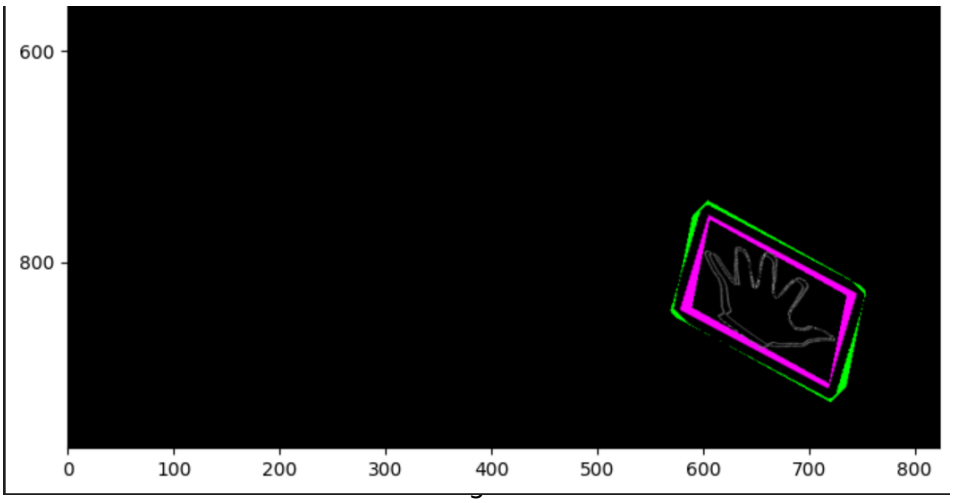
```
#flip
flipped = cv2.flip(image1, 1)
imshow("Horizontal Flip", flipped)
#scale
T1 = np.float32([[0.4, 0, 0 ], [0, 0.2,0]])
scaling = cv2.warpAffine(flipped, T1, (824,976))
imshow("scaling", scaling)
#shear
T0 = np.float32([[1, .2, 0 ], [ 0, 1,0]])
shear = cv2.warpAffine(scaling, T0, ( 824,976))
imshow("shearing", shear)
center, contour,(x,y,w,h) = find_contours(shear)
cv2.drawContours(shear, contour, -1, (0,255,0), thickness = 1)
print("after shear center:",center)
#rotate
rotation_matrix = cv2.getRotationMatrix2D(center, -28, 1)
rotated_image = cv2.warpAffine(shear, rotation_matrix, ( 824,976))
imshow("Rotation", rotated_image)
center3, contour3,(x3,y3,w3,h3) = find_contours(rotated_image)
cv2.drawContours(rotated_image, contour3, -1, (0,255,0), thickness = 1)
print("before transformation center:",center3)
#translate
T2 = np.float32([[1, 0, 477], [0, 1,740]])
img_translation = cv2.warpAffine(rotated_image, T2, ( 824,976))
imshow("Translation", img_translation)
center4, contour4,(x4,y4,w4,h4) = find_contours(img_translation )
cv2.drawContours(image1, contour4, -1, (0,255,0), thickness = 1)
```




shearing
0
200

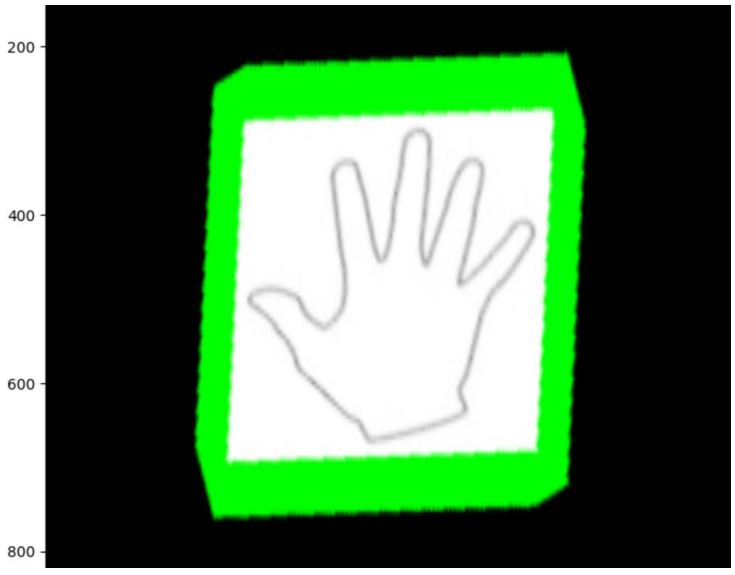

Rotation
0
200


600
800
0
100
200
300
400
500
600
700
800


600
800
0
100
200
300
400
500
600
700
800

برای به دست آوردن تصویر اصلی از روی تصویر تغییر یافته باید همین مراحل را از پایین به بالا و با ضرایب معکوس انجام داد و نتیجه به صورت زیر به دست می آید.

```
#translate
T2 = np.float32([[1, 0, -477], [0, 1,-740]])
img_translation = cv2.warpAffine(image2, T2, ( 824,976))
imshow("Translation", img_translation)
center5, contour5,(x5,y5,w5,h5) = find_contours(img_translation)
cv2.drawContours(rotated_image, contour5, -1, (0,255,0), thickness = 1)
print("before transformation center:",center5)

#rotate
rotation_matrix = cv2.getRotationMatrix2D(center5, +28, 1)
rotated_image = cv2.warpAffine(img_translation, rotation_matrix, ( 824,976))
imshow("Rotation", rotated_image)

#shear
T0 = np.float32([[1, -.2, 0 ], [ 0, 1,0]])
shear = cv2.warpAffine(rotated_image, T0, ( 824,976))
imshow("shearing", shear)

#scale
T1 = np.float32([[2.5, 0, 0 ], [0, 5,0]])
scaling = cv2.warpAffine(shear, T1, (824,976))
imshow("scaling", scaling)

#flip
flipped = cv2.flip(scaling, 1)
imshow("Horizontal Flip", flipped)
✓ 2.1s
```



سوال ۲: image enhancement

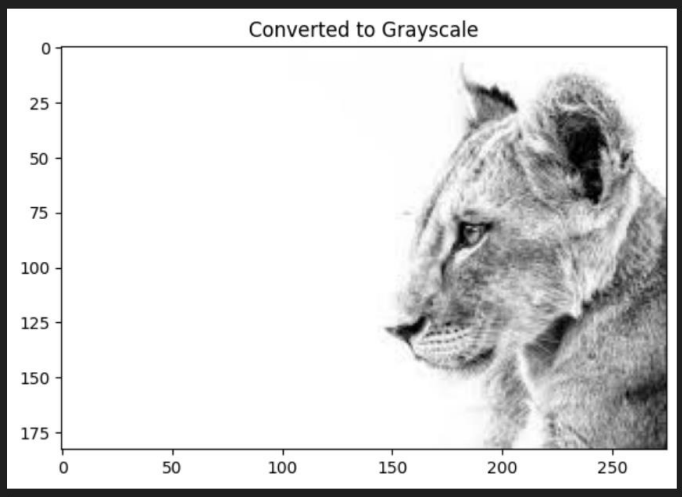
در ابتدا کتابخانه های مورد نیاز را اضافه کرده وتابع imshow را تعریف میکنیم و در ادامه تصویر مورد نظر را خوانده و با تابع imshow نمایش میدهیم. همانطورکه مشاهده میشود تصویر دارای طول و عرض ۱۸۳ در ۲۷۵ است. در ادامه تصویر را از حالت bgr به grayscale تبدیل میکنیم زیرا تبدیلات مربوط به enhancement روی تصویر grayscale پیاده میشوند.تصویرgrayscale شده به صورت زیر است.

adding the libraries

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

# Define our imshow function
def imshow(title = "Image", image = None, size = 10):
    w, h = image.shape[0], image.shape[1]
    aspect_ratio = w/h
    plt.figure(figsize=(size * aspect_ratio,size))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.show()
```

[183] ✓ 0.0s

```
Height of Image: 183 pixels
Width of Image: 275 pixels
Depth of Image: 3 colors components
```



(183, 275)

reading the image

```
image = cv2.imread('C:/Users/My/Desktop/HW_MULTI/image1.jfif')
imshow("image1", image)
print(image.shape)
print('Height of Image1: {} pixels'.format(int(image.shape[0])))
print('Width of Image1: {} pixels'.format(int(image.shape[1])))
print('Depth of Image1: {} colors components'.format(int(image.shape[2])))

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

imshow("Converted to Grayscale", gray_image)
gray_image.shape
```

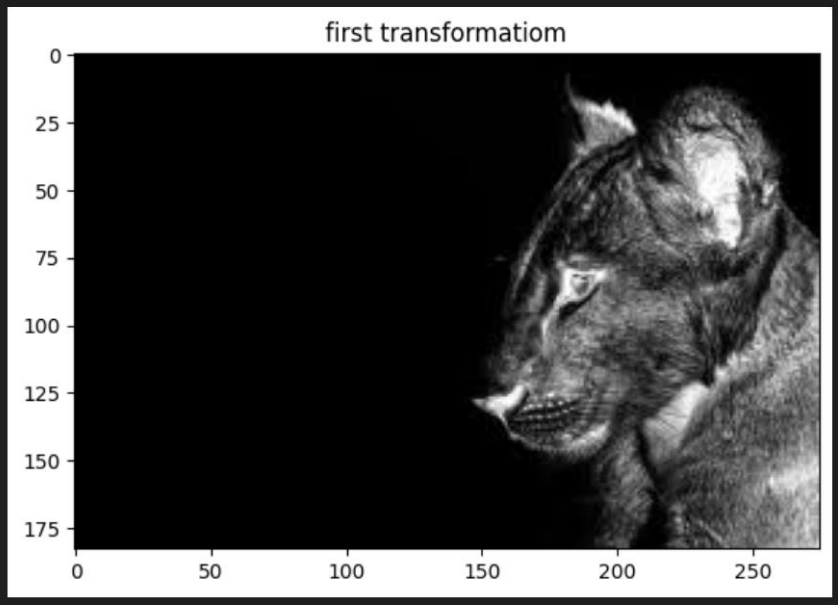
[184] ✓ 0.7s

تبدیل اول:

نمودار تبدیل اول y=-x+1 را نشان میدهد. یعنی مقدار هرپیکسل عکس grayscale از ۲۵۵ کم میشود و در تصویر نتیجه ذخیره میود. همانطورکه در تصویرنتیجه محاسبه میشود جای سفید وسیاه عوض شده است.

```
image = cv2.imread('C:/Users/My/Desktop/HW_MULTI/image1.jfif')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image1= 255 - (gray_image)
imshow("first transformatiom", image1)
```

185] ✓ 0.2s

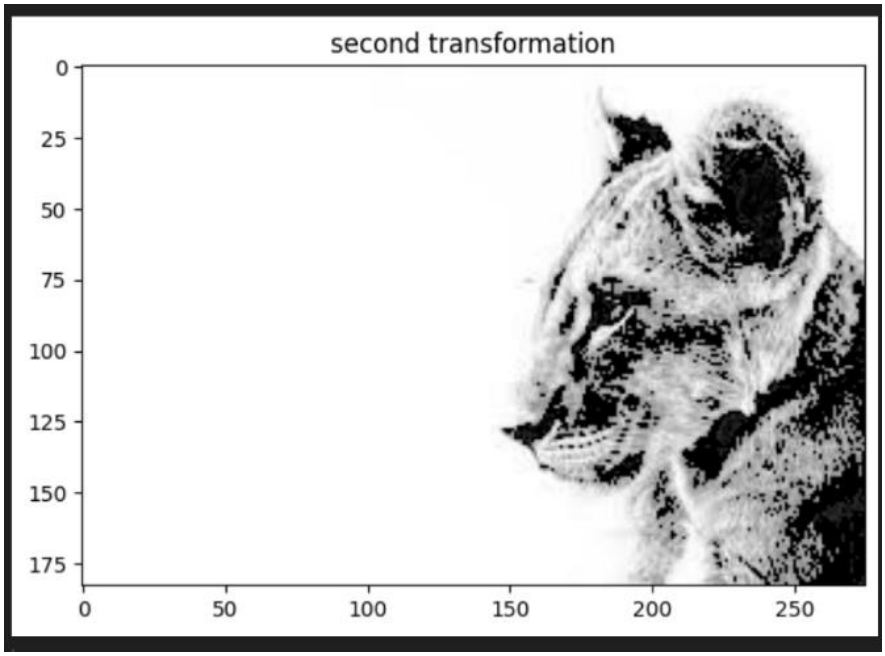


تبدیل دوم:

نمودار مربوط به تبدیل دوم یک تبدیل خطی را نشان میدهد که از ۰ تا ۰.۲ واز ۰.۵۵ تا ۱ نمودار به صورت x=y است وبین این مقادیر مقدار ۰ را دارد. با استفاده از دو for تودرتو روی تک تک پیکسل های تصویر grayscale شرط چک میشود و مقدار مورد نظر را میگیرد. تصویر نتیجه باید مقداری از طیف خاکستری میانی را به سیاه تبدیل کند که طبق نتیجه زیر درست عمل میکند.

```
image = cv2.imread('C:/Users/My/Desktop/HW_MULTI/image1.jfif')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image2=gray_image
for i in range(183):
    for j in range(275):
        if gray_image[i][j]>(255*0.2) and gray_image[i][j]<(255*0.55):
            image2[i][j]=0
        else :
            image2[i][j]=gray_image[i][j]
imshow("second transformation",image2)
cv2.imwrite('output2.jpg', image2)
```

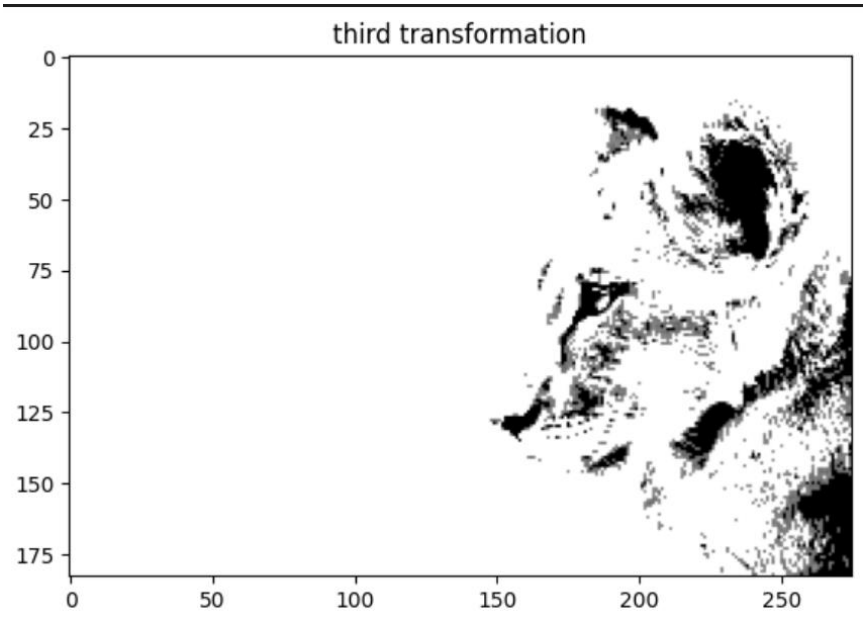
[198] ✓ 0.4s



تبدیل سوم:

در تبدیل سوم از ۰تا ۰.۴ خروجی ۰ است یعنی پیکسل ها مشکی اند و از ۰.۵۵ تا ۱ مقدار ۱ است یعنی پیکسل ها مقدار ۲۵۵ دارند وسفید اند. و بین این دومقدار رابطه y= x برقرار است. مانند قسمت قبل با دو حلقه تودرتو تک تک پیکسل ها چک میشوند و مقادیر تعریف شده به انها اختصاص میابد. انتظار میرود تصویر خروجی بیشتر سیاه و سفید باشد و فقط طیف کوچکی از خاکستری را شامل شود.

```
third transformation

image = cv2.imread('C:/Users/My/Desktop/HW_MULTI/image1.jfif')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image3=gray_image
for i in range(183):
    for j in range(275):
        if gray_image[i][j]<255*0.4:
            image3[i][j]=0
        elif gray_image[i][j]>255*0.55:
            image3[i][j]=255
        elif gray_image[i][j]<255*0.55 and gray_image[i][j]>255*0.4 :
            image3[i][j]=gray_image[i][j]

imshow("third transformation",image3)
cv2.imwrite('output3.jpg', image3)
[199]  ✓  0.4s
```



تبدیل چهارم:

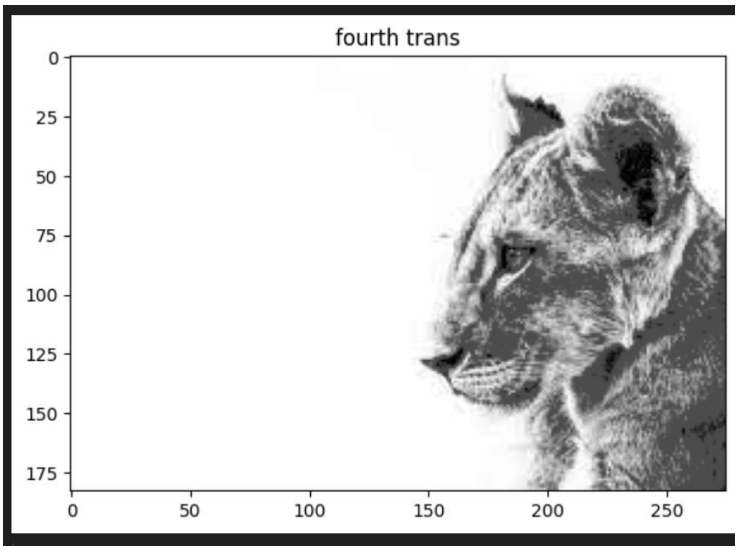
در تبدیل خطی چهارم بین ۰تا ۰.۲ معادله به صورت y=1.5x است و بین ۰.۲ تا ۰.۵۵ مقدار ثابت ۰.۳ را دارد. و برای بقیه معادله به صورت $\frac{0.7}{0.45}x + \left(1 - \frac{0.7}{0.45}\right) = y$ است.

برای پیاده سازی تبدیل بالا باید در نظر گرفت که مقدار قبلی پیکسل ها اولباید تقسیم بر ۲۵۵ شده تا نورمالایز شود و بین ۰و۱ در آید وبعد در معادله بالا گذاشته شود ودر نهایت مقدار آندر ۲۵۵ ضرب شود.

همچنین برای مقادیر بالا ممکن است سرریزرخ دهد یعنی جواب معادله از ۲۵۵ بیشتر شود. برای این که این مورد رخ ندهد میتوان مینیمم ۲۵۵و عدد به دست آمده را به عنوان خروجی گرفت یعنی اگر خروجی از ۲۵۵ بیشتر شود مقدار کمتر که ۲۵۵ است را در پیکسل میریزد. انتظار میرود که خروجی بیشتر سامل طیف های خاکستری باشد.

forth transformation

```
image = cv2.imread('C:/Users/My/Desktop/HW_MULTI/image1.jfif')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


image4=gray_image
for i in range(183):
    for j in range(275):
        if gray_image[i][j]<255*0.2:
            image4[i][j]=1.5*gray_image[i][j]
        elif gray_image[i][j]>255*0.2 and gray_image[i][j]<255*0.55:
            image4[i][j]=0.3*255
        elif gray_image[i][j]>255*0.55:
            image4[i][j]=min(255,(((0.7/0.45)*gray_image[i][j]/255)+(1-(0.7/0.45)))*255)



imshow("fourth trans",image4)
cv2.imwrite('output4.jpg', image4)
```



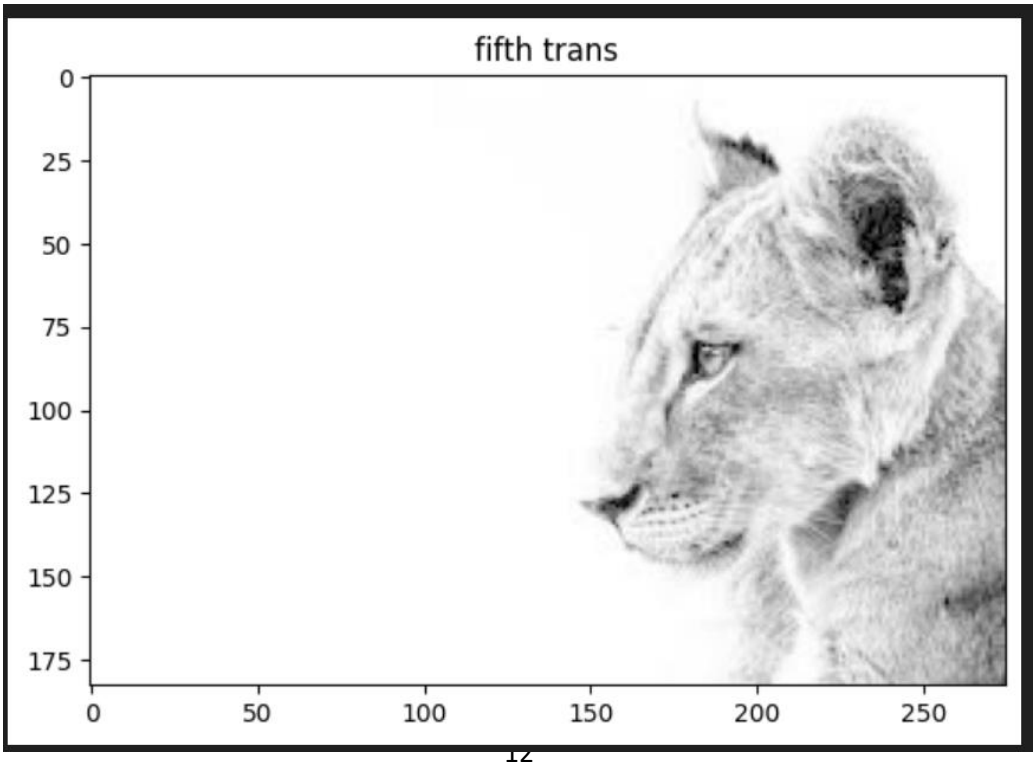
تبدیل پنجم:

در این تبدیل خروجی ریشه دوم ورودی است. در حلقه فور به این صورت عمل میشود که ابتدا مقدار هر پیکسل را با تقسیم بر ۲۵۵ نورمالایز کرده سپس آن را به توان یک دوم میرسانیم سپس در ۲۵۵ ضرب میکنیم و مینیمم جواب به دست آمده و ۲۵۵ را در پیکسل تصویر خروجی میریزیم. انتظار میرود در خروجی تصویر کمرنگتر شود.

fifth transformation

```
image = cv2.imread('C:/Users/My/Desktop/HW_MULTI/image1.jfif')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image5=gray_image
for i in range(183):
    for j in range(275):
        image5[i][j]=min(255,((gray_image[i][j]/255)**(1/2))*255)
imshow("fifth trans",image5)
cv2.imwrite('output5.jpg', image5)
```

[201] ✓ 0.4s



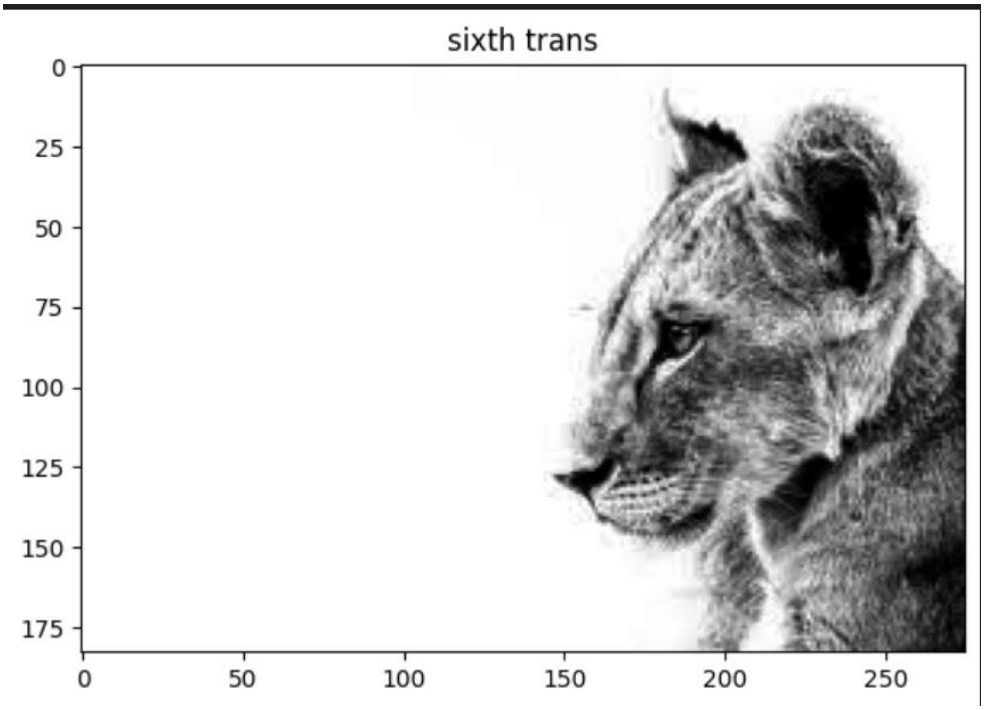
تبدیل ششم:

در این تبدیل خروجی توان دوم ورودی است. در حلقه فور به این صورت عمل میشود که ابتدا مقدار هر پیکسل را با تقسیم بر ۲۵۵ نورمالایز کرده سپس آن را به توان دو میرسانیم سپس در ۲۵۵ ضرب میکنیم و مینیمم جواب به دست آمده و ۲۵۵ را در پیکسل تصویر خروجی میریزیم. انتظار میرود در خروجی تصویر پررنگتر شود.

sixth transformation

```
image = cv2.imread('C:/Users/My/Desktop/HW_MULTI/image1.jfif')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image6=gray_image
for i in range(183):
    for j in range(275):
        image6[i][j]=min(255,((image6[i][j]/255)**2)*255)
imshow("sixth trans",image6)
cv2.imwrite('output6.jpg', image6)
```

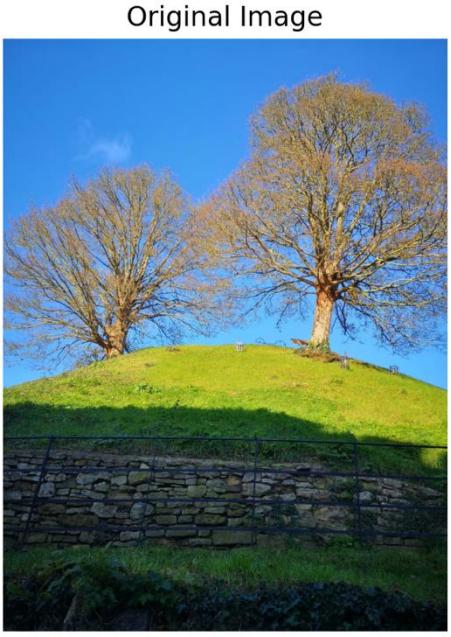
2] ✓ 0.4s



سوال ۳:

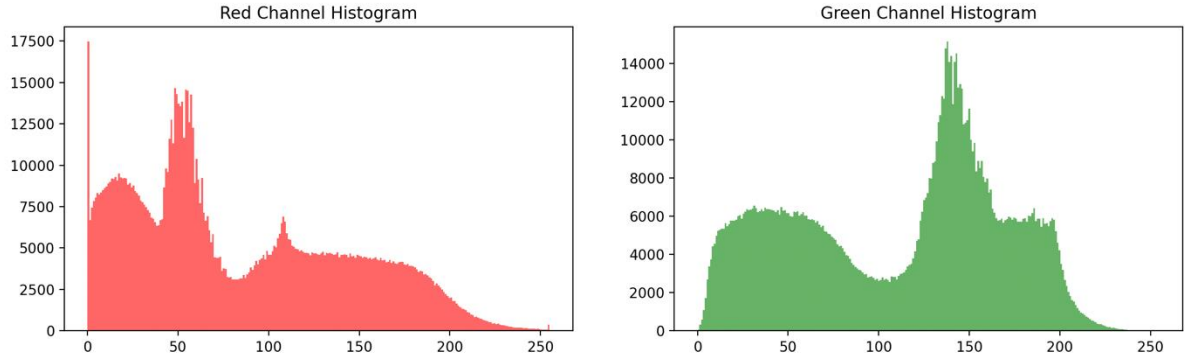
ابتدا کانال های رنگی تصویر را جدا کرده و سپس هیستوگرام هر کدام را رسم کردیم و همچنین هیستوگرام کلی تصویر را نیز به نمایش در آوردیم ’که تصاویر به صورت زیر حاصل شد .

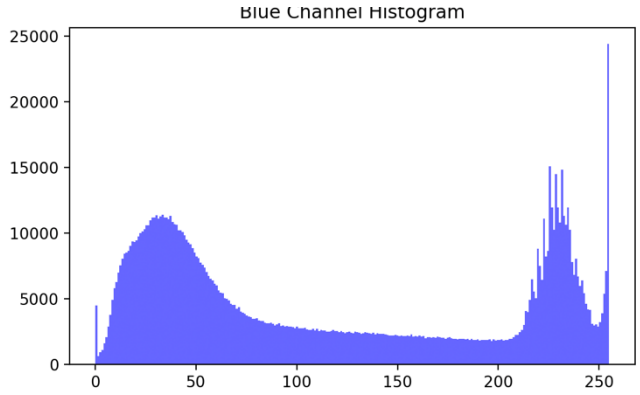
```
# Using PIL to split the image channels
r, g, b = image.split()
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
axes[0].imshow(r, cmap='Reds')
axes[0].axis('off')
axes[0].set_title('Red Channel')

axes[1].imshow(g, cmap='Greens')
axes[1].axis('off')
axes[1].set_title('Green Channel')

axes[2].imshow(b, cmap='Blues')
axes[2].axis('off')
axes[2].set_title('Blue Channel')
plt.show()
```



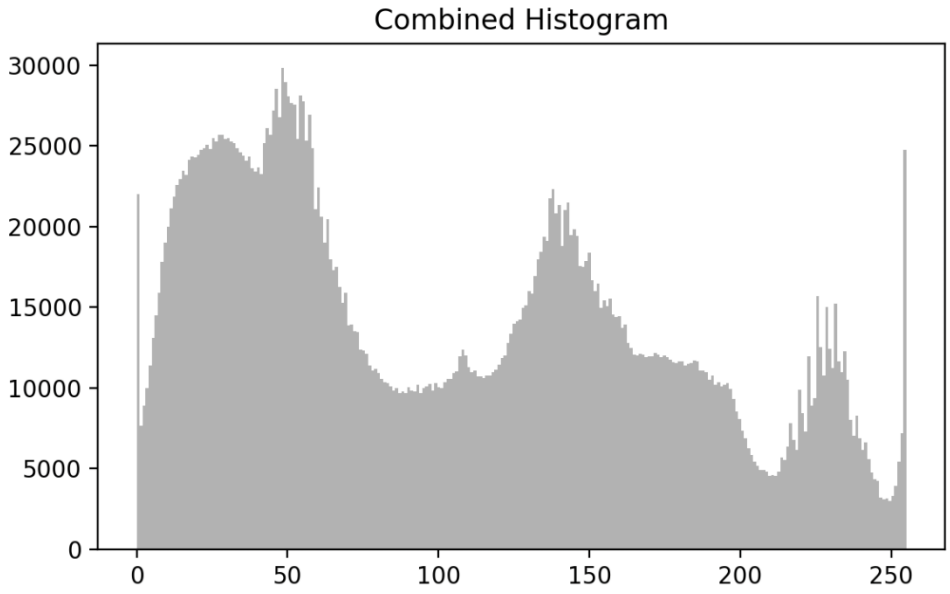
شکل ۱ - تصویر اصلی

شکل ۲ - کانال های رنگی جدا شده

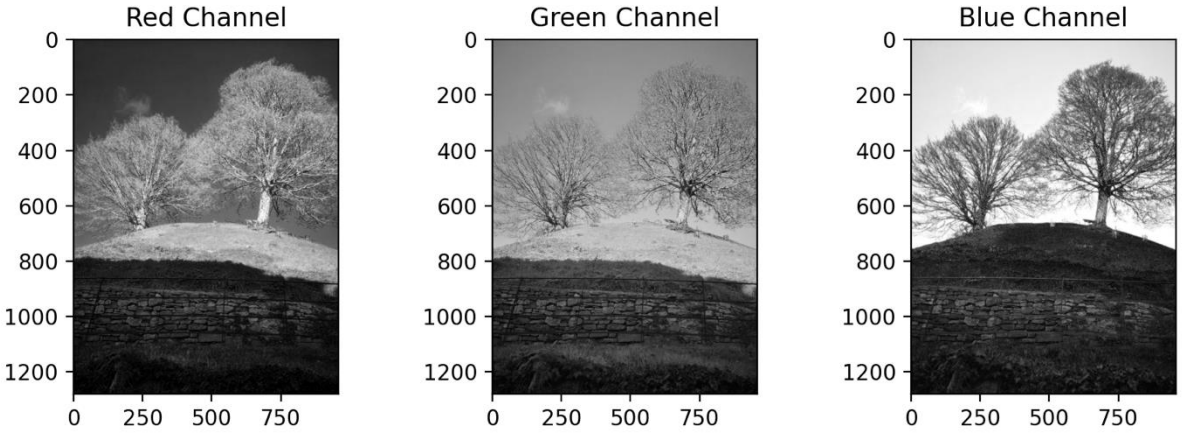




شکل ۳- هیستوگرام کانال قرمز و سبز و آبی



شکل ٤- هیستوگرام کلی

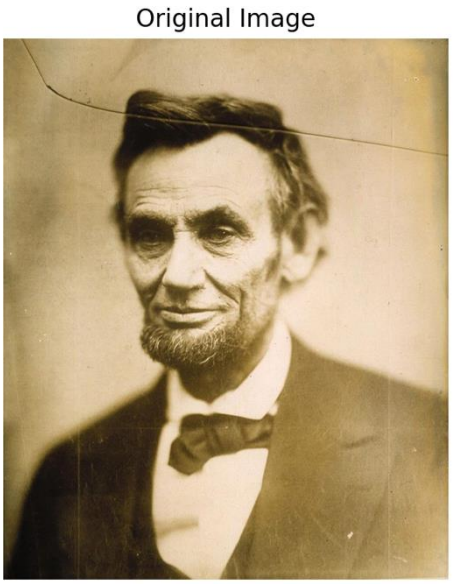


هنگامی که از opencv برای جداسازی کانال استفاده می کنیم تصاویری به صورت بالا داریم . که درواقع نقاطی که روشنایی بیشتری دارند بیانگر میزان بیشتر رنگ آن کانال در آن نقطه هستند . برای مثال در کانال سبز می بینیم روشنایی روی تپه بیشتر است و با مقایسه با تصویر اصلی متوجه می شویم رنگ سبز در این نقطه شدید تر می باشد .

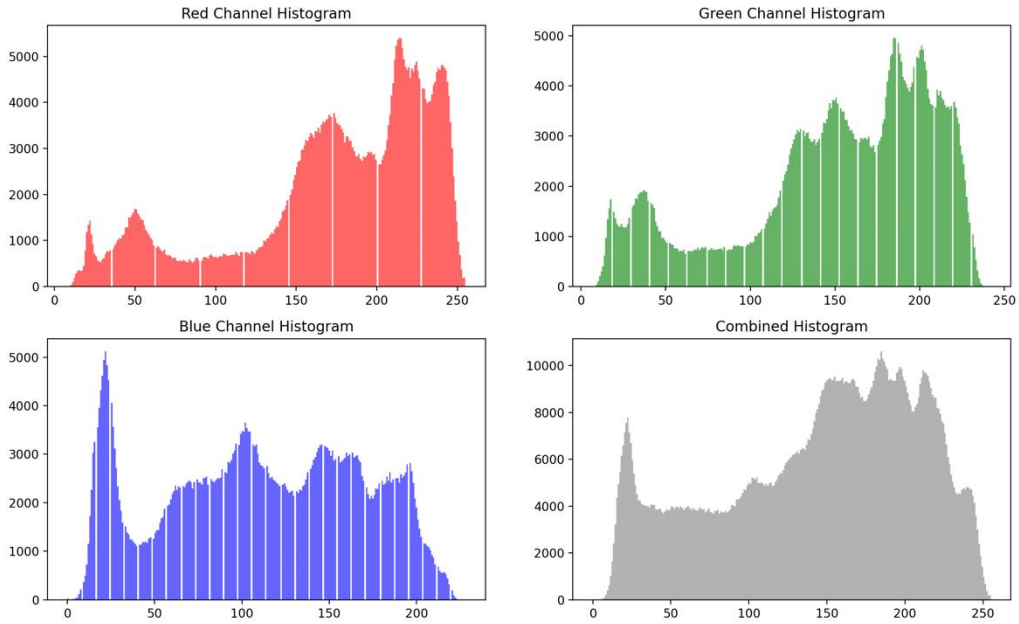
برای این که بتوانیم هیستوگرام کلی را با توجه به روشنایی تصویر داشته باشیم بهتر است تا تصویر را به سیاه و سفید تبدیل کنیم و بعد با استفاده از کتابخانه PIL آن را رسم کنیم .

تصویر دوم : abraham

ابتدا تصویر اصلی آن را نمایش میدهیم و سپس هیستوگرام آن را رسم میکنیم که به صورت زیر می باشد . از آنجایی که تصویر رنگی است هر کانال رنگی آن را جدا می کنیم و هیستوگرام هرکدام را به صورت جداگانه و همچنین مجموع کلی را رسم میکنیم . که تصاویر به صورت زیر است .



شکل ٦ - تصویر اصلی



شکل ٥ - هیستوگرام تصویر

کد مربوط به بخش رسم هیستوگرام :

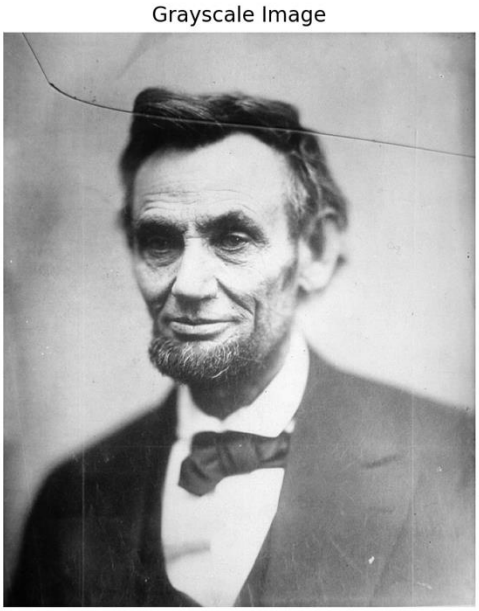
```
image_path = '/Users/digitcrom/Desktop/multimedia/Multimedia_HW2/abraham.jpg'
image = Image.open(image_path)
plt.imshow(image)
plt.axis('off')
plt.title('Original Image')
plt.show()

# Split the channels using PIL
r, g, b = image.split()
r_hist = np.array(r).flatten()
g_hist = np.array(g).flatten()
b_hist = np.array(b).flatten()
image_hist = np.array(image).flatten()
fig, axes = plt.subplots(2, 2, figsize=(16, 10))

axes[0, 0].hist(r_hist, bins=256, color='red', alpha=0.6)
axes[0, 0].set_title('Red Channel Histogram')

axes[0, 1].hist(g_hist, bins=256, color='green', alpha=0.6)
axes[0, 1].set_title('Green Channel Histogram')

axes[1, 0].hist(b_hist, bins=256, color='blue', alpha=0.6)
axes[1, 0].set_title('Blue Channel Histogram')

axes[1, 1].hist(image_hist, bins=256, color='gray', alpha=0.6)
axes[1, 1].set_title('Combined Histogram')

plt.show()
```

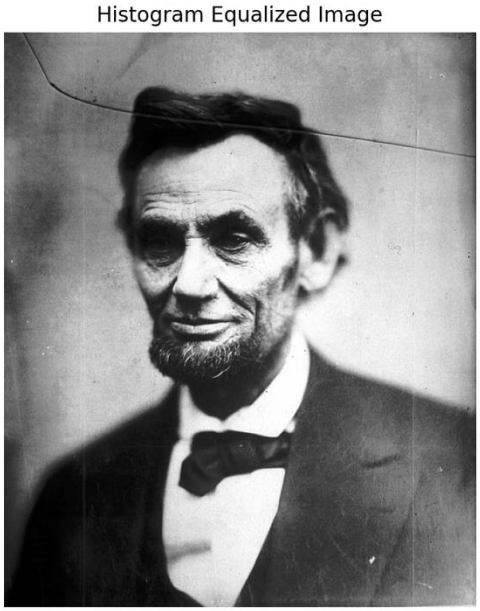
ابتدا از آدرس ذخیره شده' تصویر را لود کردیم و سپس آن را نمایش دادیم . برای نمایش هیستوگرام آن ابتدا کانال های تصویر را جدا کردیم و سپس هیستوگرام مربوط به هر کانال و همچنین هیستوگرام شامل تمامی کانال ها را رسم کردیم . مقدار آلفا میزان شفافیت هر بین در رسم را نشان میدهد که عددی بین ۰ تا ۱ می باشد .

حال همانطور که سوال خواسته است روش histogram equalization را بر روی تصویر اعمال میکنیم و اینبار نیز هیستوگرام مربوط به آن را نیز رسم میکنیم . این روش بر تک کانال اعمال می شود' بنابراین ابتدا تصویر را سیاه سفید می کنیم و بعد آن را اعمال می کنیم

```
gray_image =image.convert('L')
equalized_image = ImageOps.equalize(gray_image)

# Display the original and equalized images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(equalized_image, cmap='gray')
plt.title('Histogram Equalized Image')
plt.axis('off')
plt.show()
```

بعد از تبدیل تصویر به سیاه و سفید روش equalization را اعمال میکنیم و بعد تصویر سیاه سفید و تصویر equalized شده را کنار هم نمایش می دهیم .

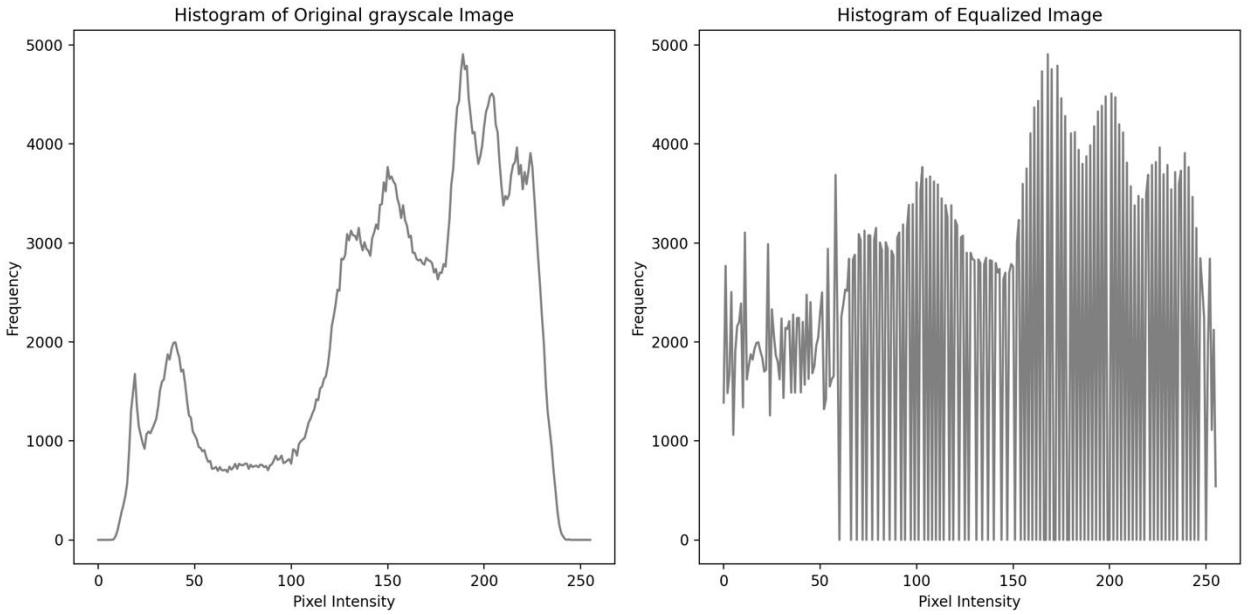




نتیجه به صورت بالا می باشد . مشاهده می شود که روش Histogram Equalization کنتراست تصویر را افزایش داده است که باعث شده جزئیات بیشتری نسبت به تصویر اصلی نمایان شود. برای مثال در تصویر جدید خطوط صورت بیشتر برجسته شده اند .

```
hist_original = gray_image.histogram()
hist_equalized = equalized_image.histogram()

# Plotting the histograms
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(hist_original, color='gray')
plt.title('Histogram of Original grayscale Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.plot(hist_equalized, color='gray')
plt.title('Histogram of Equalized Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

حال به رسم هیستوگرام تصویر سیاه سفید شده و هیستوگرام تصویر equalize شده می پردازیم و هر دو را در یک تصویر نمایش میدهیم تا بتوانیم آن هارا با هم مقایسه کنیم .



هیستوگرام ها به صورت بالا می باشند و مشاهده میشود که در روش equalized در واقع به نوعی مقادیر نرمالایز می شوند و در رنج مشخی قرار میگیرند ’ درواقع نشان دهنده توزیع یکنواخت تری از پیکسل ها است و پیکسل ها به طور یکنواخت تری پخش شده اند و این باعث افزایش کنتراست تصویر می شود . ولی هیستوگرام تصویر اصلی نشان می‌دهد که بیشتر پیکسل‌ها در محدوده‌ی خاکستری قرار دارند و پراکندگی زیادی در محدوده‌ی تیره یا روشن ندارند که نشان‌دهنده‌ی کم بودن کنتراست تصویر است.

سوال ۴ :

ابتدا تصویر را خوانده و سپس فیلتر های خواسته شده را به ترتیب بر روی تصویر اجرا می کنیم.

```
image_path = '/Users/digitcrom/Desktop/multimedia/Multimedia_HW2/AerialView.jpeg'
image = Image.open(image_path)

# Apply Gaussian Filter
gaussian_filtered = image.filter(ImageFilter.GaussianBlur(radius=2))

# Apply Median Filter
median_filtered = image.filter(ImageFilter.MedianFilter(size=3))

# Apply Sharpening Filter
sharpened = image.filter(ImageFilter.SHARPEN)
```

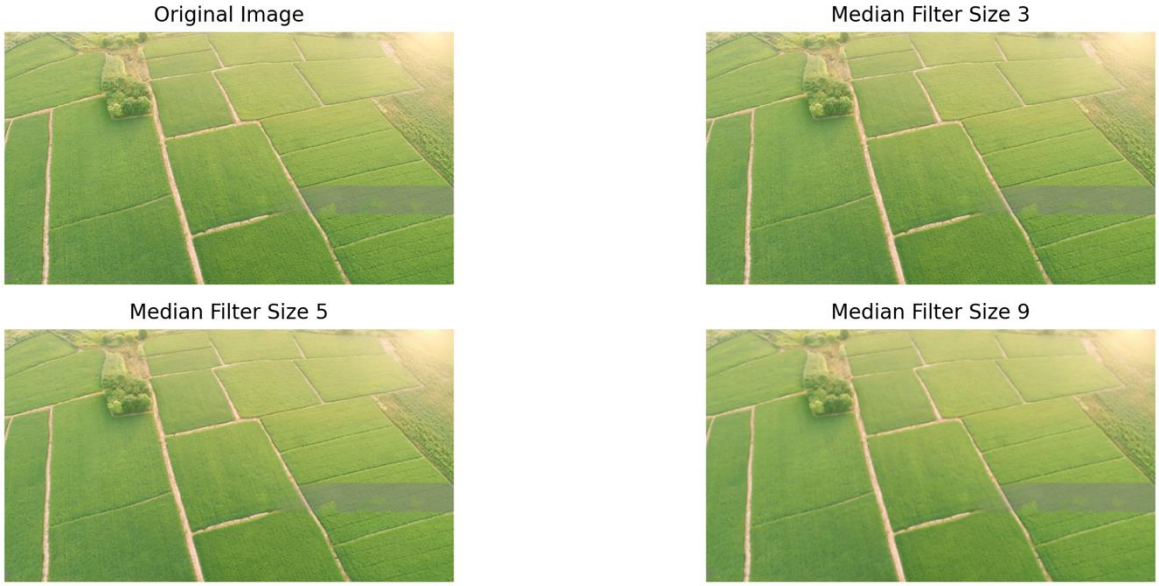
فیلتر گاوسین :

فیلتر گاوسین بر اساس تابع گاوسی کار می‌کند. در فیلتر گاوسین، یک کرنل بر اساس تابع گاوسی داریم که این کرنل دارای وزن است که بیشترین وزن در مرکز قرار دارد و به تدریج که از مرکز دور می‌شویم، وزن‌ها کمتر می‌شود. فیلتر به تصویر اعمال می ود و به این صورت است که هر پیکسل تصویر و همسایگان آن با وزن‌های متناسب با کرنل گاوسین ضرب شده و مجموع آن‌ها به عنوان مقدار جدید پیکسل در نظر گرفته می‌شود

.فیلتر گاوسین پارامتر radius دارد که می توانیم مقدار آن را تغییر دهیم . این پارامتر میزان بلور شدن تصویر را مشخص می کند . اگر مقدار این پارامتر کم باشد تقریبا تغییر خاصی نسبت به تصویر اصلی نداریم و کمی کاهش در نویز خواهیم داشت . اگر مقدار این پارامتر را متوسط در نطر بگیریم (مثلا بین ۵ تا ۱۰)مقدار بلور در تصویر قابل تشخیص خواهد شد و لبه ها از تیزی در میایند . در مقدار پارامتر های بزرگ تصویر را بسیار smooth می کند و کاملا بلوری و بیشتر محتوای تصویر را از دست میدهیم و تنها در تغییر رنگ تصویر قابل تشخیص است . در این جا مقدار radius را مقادیر مختلفی قرار دادیم و تصویر پایین تفاوت این مقادیر را نشان میدهد .

فیلتر مدین :

برای هر پیکسل در تصویر، همسایگی اطراف آن پیکسل بر اساس اندازه مشخص شده (مثلاً ۳*۳) انتخاب می‌شود.پیکسل‌های درون این همسایگی بر اساس شدت روشنایی‌شان مرتب می‌شوند. و یکسل میانی از فهرست مرتب‌شده انتخاب می‌شود و جایگزین پیکسل فعلی در تصویر نهایی می‌شود.در این فیلتر پارامتر سایز قابل تغییر می باشد و درواقع سایز فیلتری که برروی تصویر اعمال می شود را کنترل می کند . وقتی سایز پارامتر را کوچک در نظر میگیریم درواقع پیکسل های همسایه کمتری را برای میانگین گیری در نظر می گیرد و نویز کمتری را کاهش می دهد . هنگامی که مقدار پارامتر را متوسط در نظر می گیریم اثر smoothing واضح تر می باشد و مقدار sharpening نسبت به تصویر اصلی کاهش می یابد . پارامتر در مقدار های بزرگتر باعث می شود که نویز بیشتری از بین برود ولی همچنین می تواند باعث از بین رفتن جزئیات تصویر نیز بشود و تصویر بسیار smooth می شود . با اعمال مقادیر مختلف پارامتر size تصاویر زیر را مشاهده می کنیم و میتوانیم تغییرات را ببینیم .



فیلتر sharpening :

این فیلتر دارای یک کرنل از پیش تعریف شده می باشد اما میتوانیم کرنل آن را نیز خودمان به صورت دستی به آن اعمال کنیم . ما سه کرنل متفاوت را بر روی تصویر اعمال می کنیم .

```
# Apply Sharpening Filter
sharpened = image.filter(ImageFilter.SHARPEN)

# Define different kernels
sharpen_kernel = ImageFilter.Kernel(
    size=(3, 3),
    kernel=[
        -1, -1, -1,
        -1,  9, -1,
        -1, -1, -1
    ],
    scale=None,
    offset=0
)

edge_kernel = ImageFilter.Kernel(
    size=(3, 3),
    kernel=[
        0, -1,  0,
       -1,  4, -1,
        0, -1,  0
    ],
    scale=None,
    offset=0
)

blur_kernel = ImageFilter.Kernel(
    size=(3, 3),
    kernel=[
        1/9, 1/9, 1/9,
        1/9, 1/9, 1/9,
        1/9, 1/9, 1/9
    ],
    scale=None,
    offset=0
)
```

کرنل اول درواقع باعث ایجاد کنتراست بیشتر و تشخیص راحتتر لبه می شود .کرنل دوم لبه را بسیار پررنگ میکند .کرنل سوم درواقع میانگین مقادیر پیکسل های همسایه را می گیرد و عمل بلور کردن را انجام میدهد تا sharpening .نتیجه کرنل ها تصویر را به حالت های زیر در می آورد . در حالت کرنل دوم درواقع دور تصویر به عنوان لبه شناخته میشود.

Original Image

Sharpened Image

Edge Detection

Blurred Image

با توجه به نتایج بالا میتوانیم این برداشت را داشته باشیم که فیلتر گاوسین عملکرد بهتری در smoothing دارد و هنگامی که پارامتر radius آن در مقدار متوسط (۳-۵) باشد smoothing بهتری ارائه می کند و بعد از آن فیلتر مدین در smoothing بهتر از فیلتر sharpening عمل میکند .

حال به اعمال الگوریتم های لبه یابی بر روی تصویر اصلی و تصاویر فیلتر شده می پردازیم . برای این کار ابتدا تصاویر را به ارایه numpy تبدیل میکنیم و بعد از آن تصاویررا به سیاه و سفید تبدیل می کنیم

```
# Convert images to numpy arrays for edge detection
original_np = np.array(image)
gaussian_np = np.array(gaussian_filtered)
median_np = np.array(median_filtered)
sharpened_np = np.array(sharpened)

# Convert to grayscale for edge detection
original_gray = cv2.cvtColor(original_np, cv2.COLOR_BGR2GRAY)
gaussian_gray = cv2.cvtColor(gaussian_np, cv2.COLOR_BGR2GRAY)
median_gray = cv2.cvtColor(median_np, cv2.COLOR_BGR2GRAY)
sharpened_gray = cv2.cvtColor(sharpened_np, cv2.COLOR_BGR2GRAY)
```

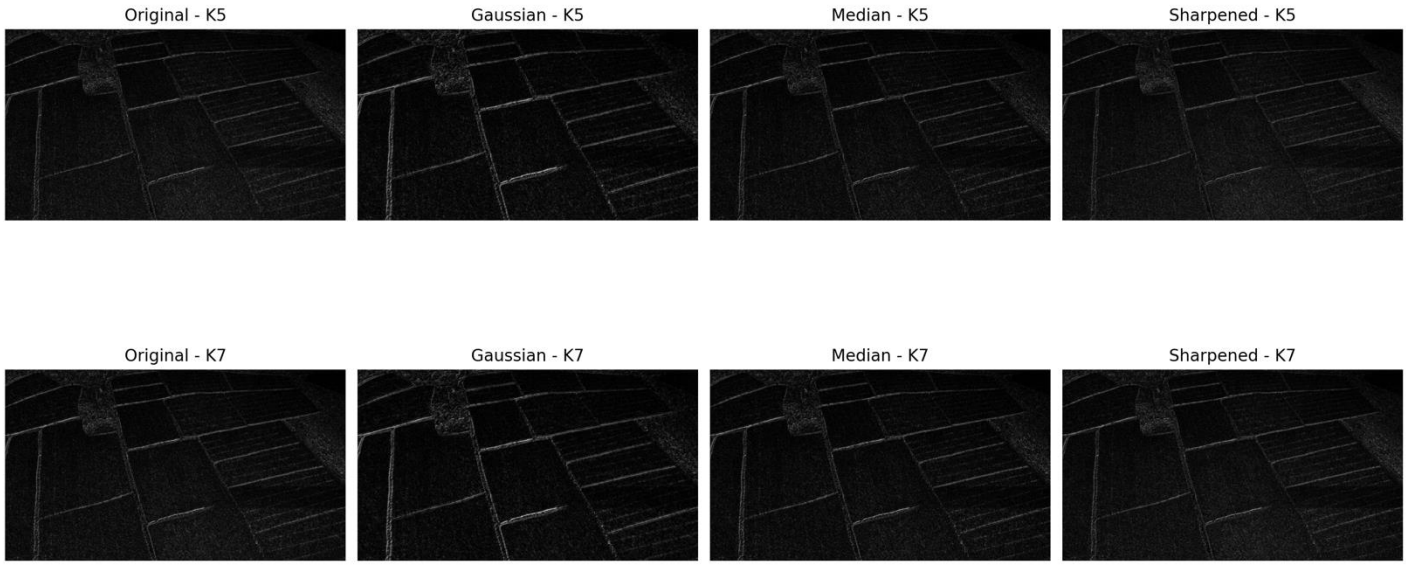
و حال تصاویر اماده برای اعمال الگوریتم لبه یابی هستند .

اعمال الگوریتم لبه یابی sobel :

الگوریتم را به این صورت تعریف می کنیم و اعداد ۱و۰ و برعکس نشان دهنده یافتن لبه در جهت محور افقی و سپس عمودی می باشد و همچنین میتوانیم سایز کرنل را نیز مشخص کنیم .

```
# Apply Sobel edge detection
sobel_original = cv2.Sobel(original_gray, cv2.CV_64F, 1, 0, ksize=3) + cv2.Sobel(original_gray, cv2.CV_64F, 0, 1, ksize=3)
sobel_gaussian = cv2.Sobel(gaussian_gray, cv2.CV_64F, 1, 0, ksize=3) + cv2.Sobel(gaussian_gray, cv2.CV_64F, 0, 1, ksize=3)
sobel_median = cv2.Sobel(median_gray, cv2.CV_64F, 1, 0, ksize=3) + cv2.Sobel(median_gray, cv2.CV_64F, 0, 1, ksize=3)
sobel_sharpened = cv2.Sobel(sharpened_gray, cv2.CV_64F, 1, 0, ksize=3) + cv2.Sobel(sharpened_gray, cv2.CV_64F, 0, 1, ksize=3)
```

الگوریتم sobel را برای کرنل با سایز های ۵ و ۷ اعمال کردیم و نتیجه به صورت زیر در آمد :



با توجه به تصاویر لبه ها در حالت کرنل ۷ و فیلتر گاوسین بهتر نمایش داده شده اند .

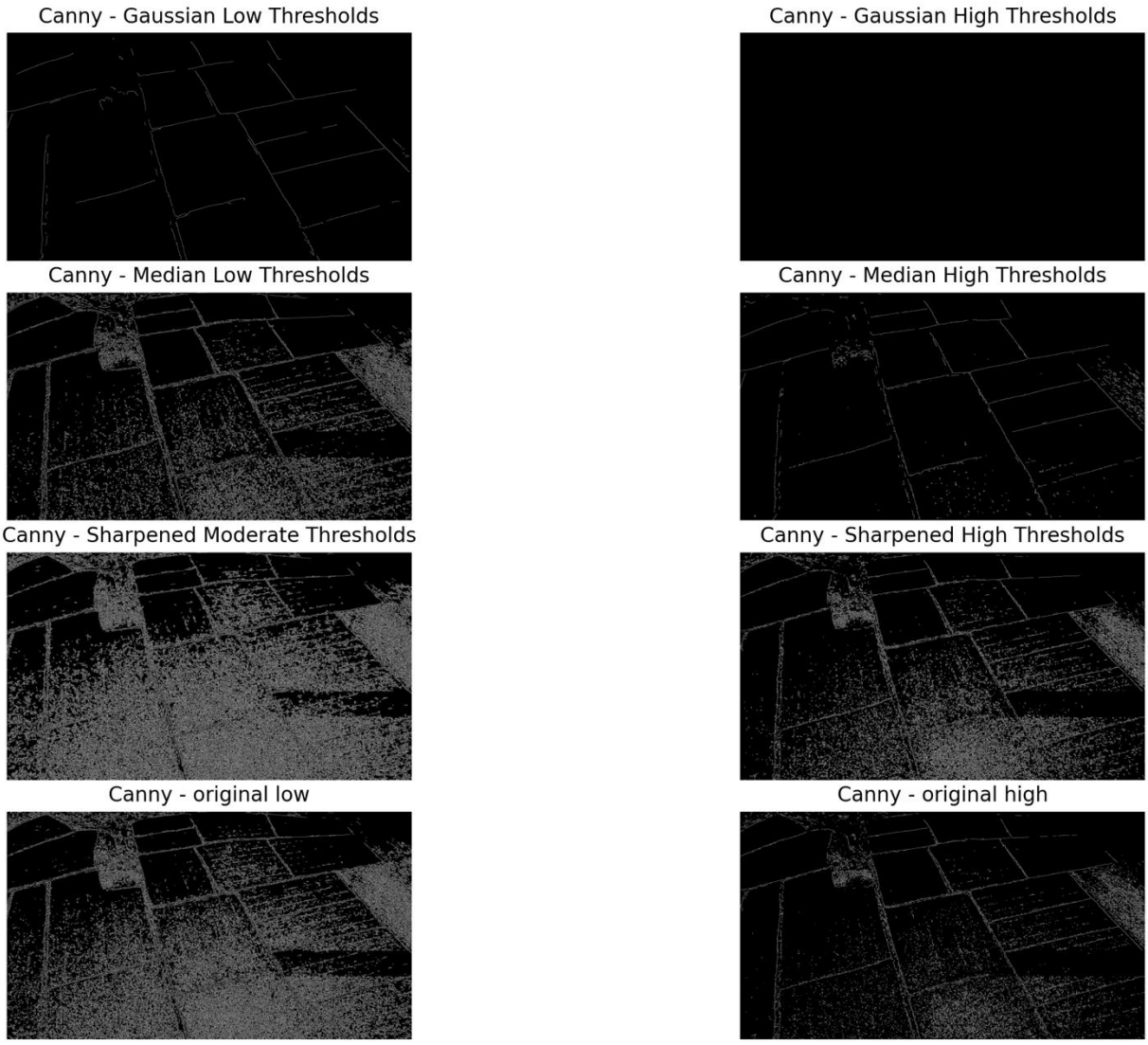
اعمال الگوریتم canny :

الگوریتم canny را به صورت زیر اعمال میکنیم و برای مقادیر آستانه متفاوت آن را امتحان می کنیم .

```
canny_gaussian_low = cv2.Canny(gaussian_gray, 50, 100)  # Lower thresholds
canny_gaussian_high = cv2.Canny(gaussian_gray, 100, 200)  # Higher thresholds


canny_median_low = cv2.Canny(median_gray, 50, 100)  # Lower thresholds
canny_median_high = cv2.Canny(median_gray, 100, 200)  # Higher thresholds


canny_sharpened_low = cv2.Canny(sharpened_gray, 50, 150)  # Moderate thresholds
canny_sharpened_high = cv2.Canny(sharpened_gray, 100, 200)  # Higher thresholds
```

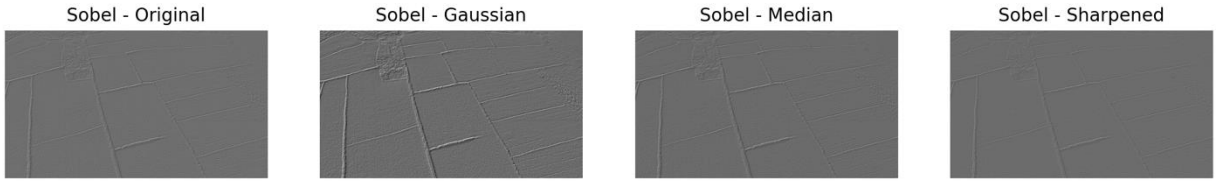
نتیجه اعمال این الگوریتم بر تصویر ها با مقادیر آستانه متفاوت به صورت زیر می باشد .

در اینجا مشاهده می شود که مدین در آستانه بالا به خوبی عمل میکند . همینطور که میبینیم وقتی استانه پایین باشد نقاط زیادی به عنوان لبه شناسایی می شوند .



- ضخامت لبه های تشخیص داده شده در الگوریتم sobel بیشتر از canny بود .
- مشاهده می شود که هنگامی که تصویر بدون فیلتر است الگوریتم canny در تشخیص لبه بهتر عمل میکند .
- در هر دو الگوریتم لبه ها تا حد خوبی به درستی نشان داده شده اند . البته در استانه های پایین canny این طور نیست که آن هم به دلیل گرفتن مقدار پایین استانه می باشد .

مشاهد می شود در sobel هنگامی که ابتدا فیلتر گاوسین اعمال می شود تشخیص لبه بهتری داریم .
درواقع ابتدا نویز تصویر کاهش پیدا می کند .



در canny مشاهده می شود که median و گاوسین با تنظیم مقدار آستانه بهتر عمل میکنند .