

دانشگاه علم و صنعت

تمرین دوم مبانی بینایی کامپیوتر

نام و نام خانوادگی:

فرناز خوش دوست آزاد

شماره دانشجویی:

99521253

نام استاد:

دکتر محمدرضا محمدی

(1)

الف) برای تصویر زیر (image1) ابتدا هیستوگرام را حساب کرده و رسم کنید و سپس فرایند کشش هیستوگرام و برش هیستوگرام را روی آن اعمال کنید و تصویر حاصل را بکشید و دوباره هیستوگرام را رسم کنید. (برای انتخاب محدوده برش هیستوگرام طبق نظر خودتان و برای بهتر شدن نتیجه عمل کنید) (۱۰)

[144 ,143 ,143 ,143 ,138 , 0 ,141]

[143 ,143 ,142 ,142 ,140 ,140 ,141]

[145 ,142 ,142 ,144 ,145 ,145 ,140]

[143 ,141 ,142 ,143 ,141 ,141 ,141]

[144 ,143 ,142 ,141 ,141 ,140 ,139]

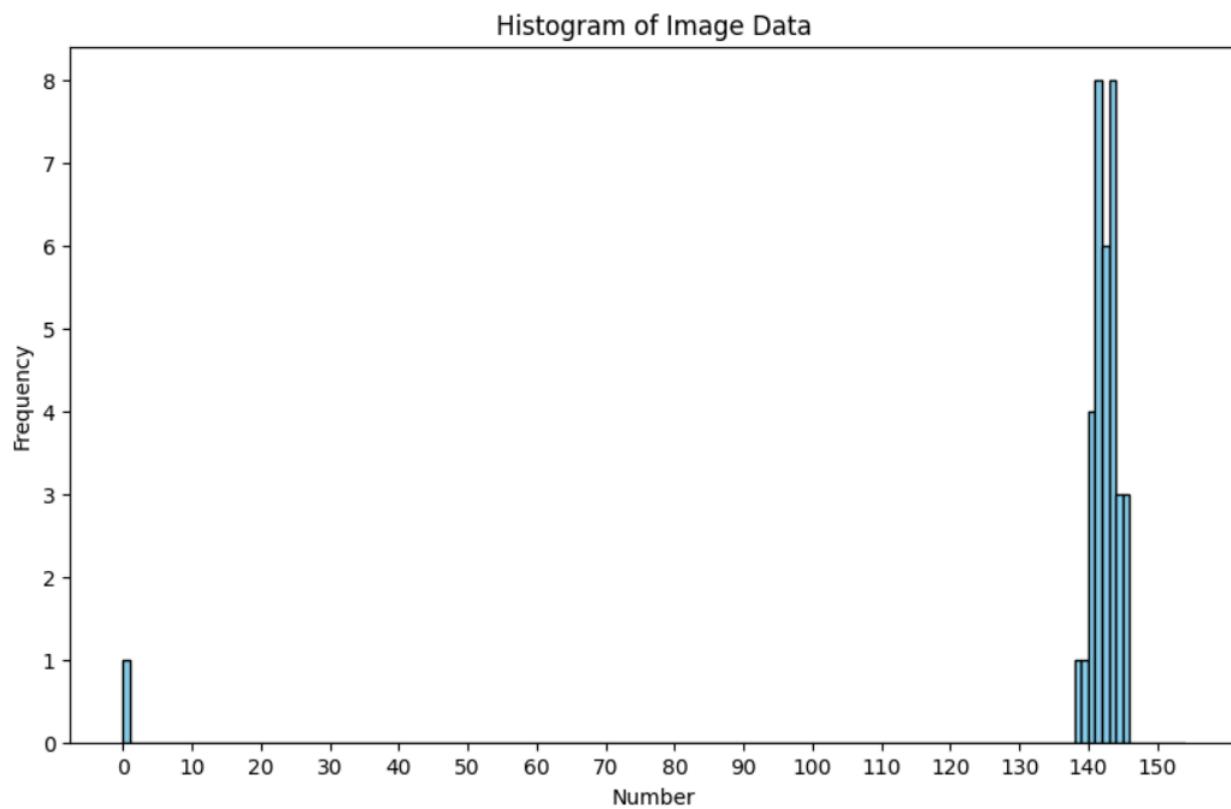
: پاسخ

```
1 #define image1 here
2 import numpy as np
3 import pandas as pd
4
5 image1_alaki = np.array([[141, 0, 138, 143, 143, 143, 144],
6 | | | | | | | [141, 140, 140, 142, 142, 143, 143],
7 | | | | | | | [140, 145, 145, 144, 142, 142, 145],
8 | | | | | | | [141, 141, 141, 143, 142, 141, 143],
9 | | | | | | | [139, 140, 141, 141, 142, 143, 144]])
10 unique, counts = np.unique(image1_alaki, return_counts=True)
11 df = pd.DataFrame({'Number': unique, 'Count': counts})
12 print(df)
```

برای محاسبه‌ی آن ابتدا باید تعداد هر کدام از پیکسل‌ها را به دست آوریم تا بتوانیم هیستوگرام آن را محاسبه کنیم.

Number	Count
0	0
1	138
2	139
3	140
4	141
5	142
6	143
7	144
8	145

و می دانیم که بقیه نیز مقدار صفر را دارا هستند و نمودار هیستوگرام آن نیز در زیر مشخص است.



کشش هیستوگرام یک تکنیک برای بهبود contrast تصویر است که چگالی مقادیر را در یک بازه بزرگتر پخش می کند.

این کار باعث می شود که وضوح در جزئیات در مناطق تیره و روشن تصویر بیشتر شود. بدین منظور ابتدا مقادیر maximum و minimum تصویر را محاسبه کرده و با استفاده از فرمول زیر این مقادیر را normalize می کنیم:

$$255 * (\text{original value} - \text{minimum}) / (\text{maximum} - \text{minimum})$$

$$\text{original value} = 0 \rightarrow \text{stretched value} = 0$$

$$\text{original value} = 138 \rightarrow \text{stretched value} = 242$$

$$\text{original value} = 139 \rightarrow \text{stretched value} = 244$$

$$\text{original value} = 140 \rightarrow \text{stretched value} = 246$$

$$\text{original value} = 141 \rightarrow \text{stretched value} = 247$$

$$\text{original value} = 142 \rightarrow \text{stretched value} = 249$$

$$\text{original value} = 143 \rightarrow \text{stretched value} = 251$$

$$\text{original value} = 144 \rightarrow \text{stretched value} = 253$$

$$\text{original value} = 145 \rightarrow \text{stretched value} = 255$$

New matrix =

$$[253, 251, 251, 251, 242, 0, 247]$$

$$[251, 251, 249, 249, 246, 246, 247]$$

$$[255, 249, 249, 253, 255, 255, 246]$$

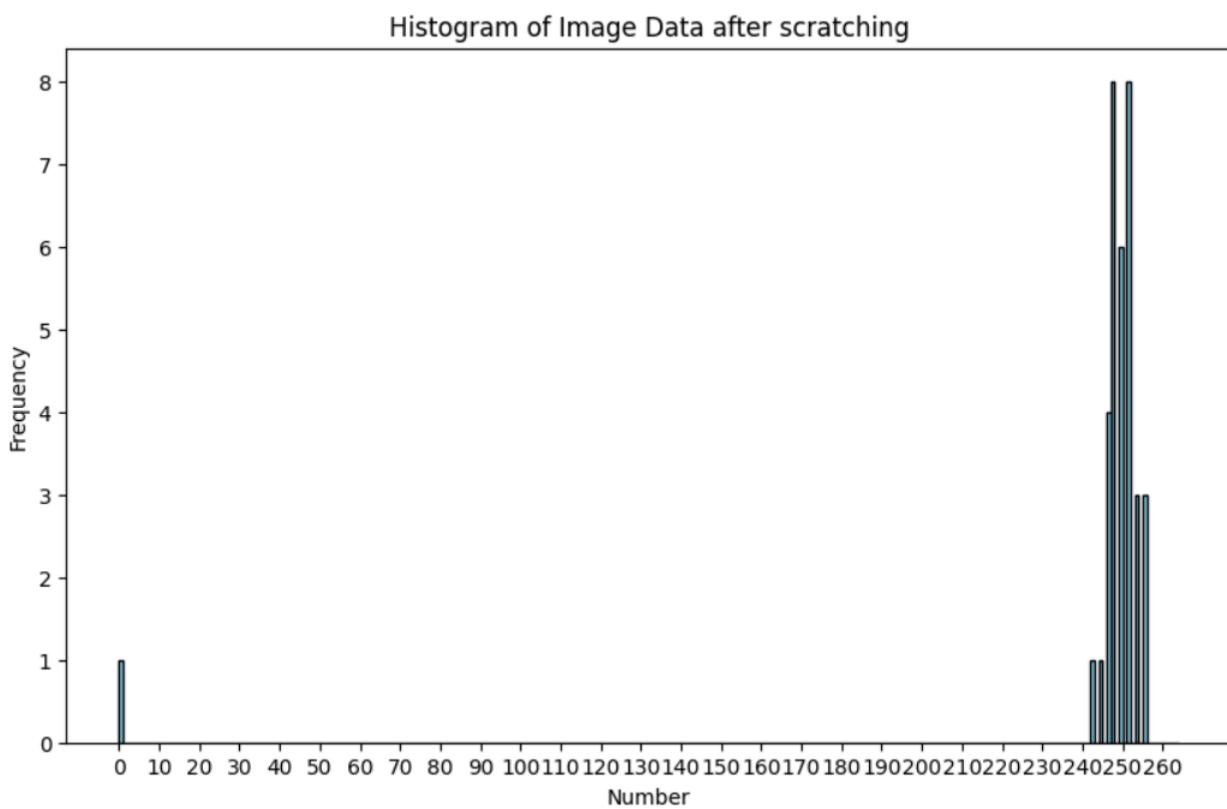
$$[251, 247, 249, 251, 247, 247, 247]$$

$$[253, 251, 249, 247, 247, 246, 244]$$

دوباره برای نمایش نمودار هیستوگرام، باید تعداد مقادیر هر یک را حساب کنیم:

Number	Count
0	0
1	242
2	244
3	246
4	247
5	249
6	251
7	253
8	255

و نمودار هیستوگرام آن در زیر مشخص شده است:



برای برش هیستوگرام ابتدا مقادیر کمینه و بیشینه را در نظر می گیریم؛ در ماتریس داده شده کمترین مقدار برابر 0 و بیشترین مقدار برابر 145 است. در مرحله بعد باید یک بازه را برای برش در نظر بگیریم. از آنجایی که اکثر مقادیر به یکدیگر نزدیکند، بازه ای را انتخاب می کنیم که بیشترین مقادیر را در برگیرد. بازه [130, 150] را در نظر می گیریم. اکنون برش هیستوگرام را اعمال می کنیم؛ برای

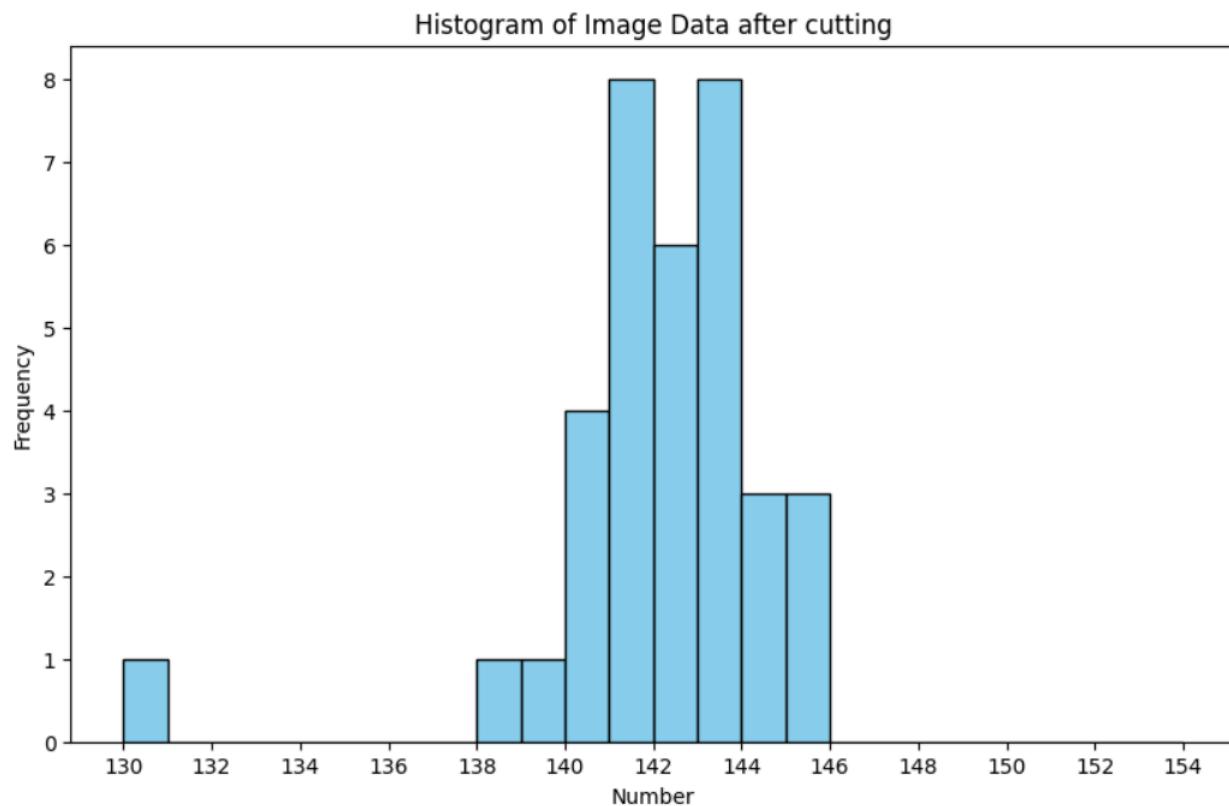
این کار مقادیر زیر 130 و بالای 150 را حذف می کنیم و مقدار بالای 150 را 150 و مقادیر کمتر از 130 را صفر می کنیم. حال داریم:

```

[[141 130 138 143 143 143 143 144]
 [141 140 140 142 142 143 143 143]
 [140 145 145 144 142 142 142 145]
 [141 141 141 143 142 141 141 143]
 [139 140 141 141 142 143 143 144]]
```

Number	Count	
0	130	1
1	138	1
2	139	1
3	140	4
4	141	8
5	142	6
6	143	8
7	144	3
8	145	3

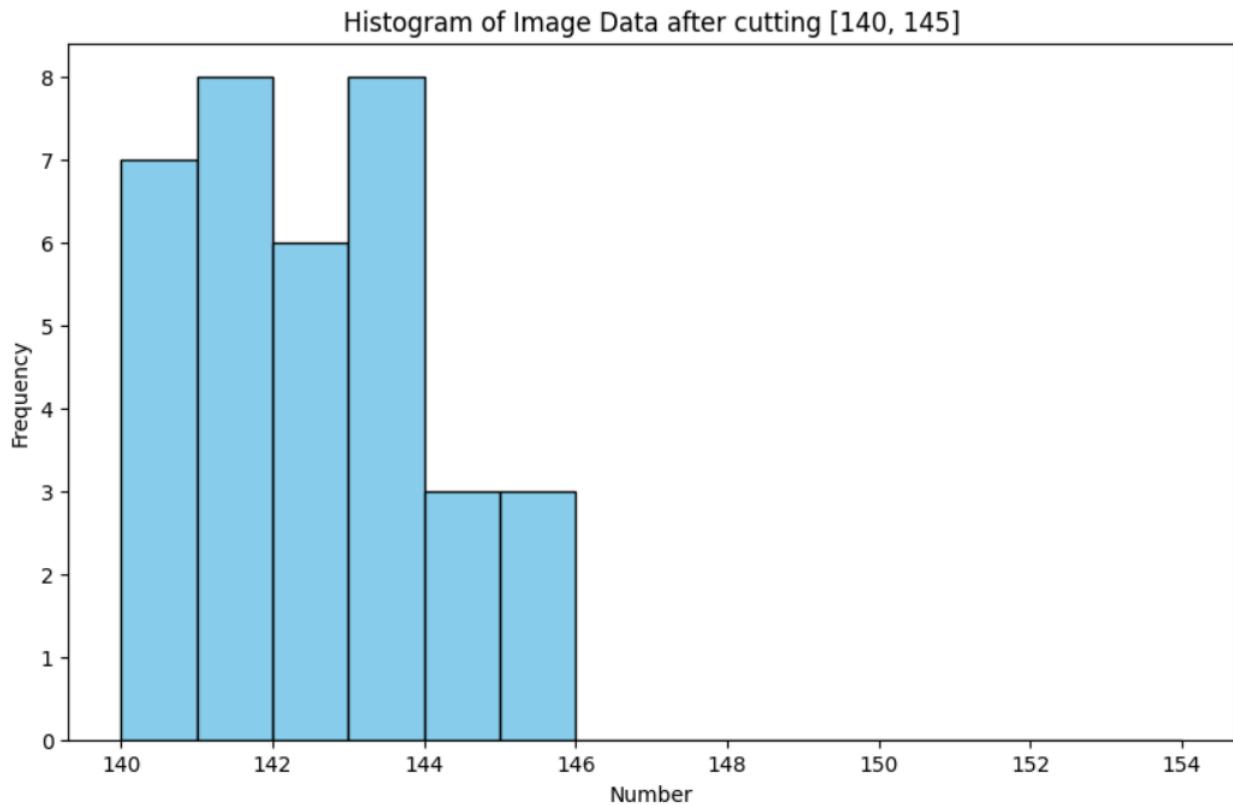
و نمودار هیستوگرام آن نیز به شرح زیر است:



حال می توانیم بازه‌ی خود را بین ۱۴۰ و ۱۴۵ در نظر بگیریم و خواهیم داشت:

[[141	140	140	143	143	143	144]
[141	140	140	142	142	143	143]
[140	145	145	144	142	142	145]
[141	141	141	143	142	141	143]
[140	140	141	141	142	143	144]]

Number	Count	
0	140	7
1	141	8
2	142	6
3	143	8
4	144	3
5	145	3



حال می بینیم که در بازه جدید مقادیر در بازه کوچکتری قرار می گیرند و مقادیر بیشتری تغییر می کنند که به این موضوع به معنای نزدیک تر بودن مقادیر تصویر و از دست رفتن برخی از اطلاعات آن

است. در بازه قبلی تنها مقدار ۰ به ۱۳۰ تبدیل شد و بقیه مقادیر نیز تغییری نکردند. اما در اینجا ما اطلاعات بیشتری را عوض کردیم.

کدهای آن نیز در نوتس بوک Q1.py موجود می‌باشد.

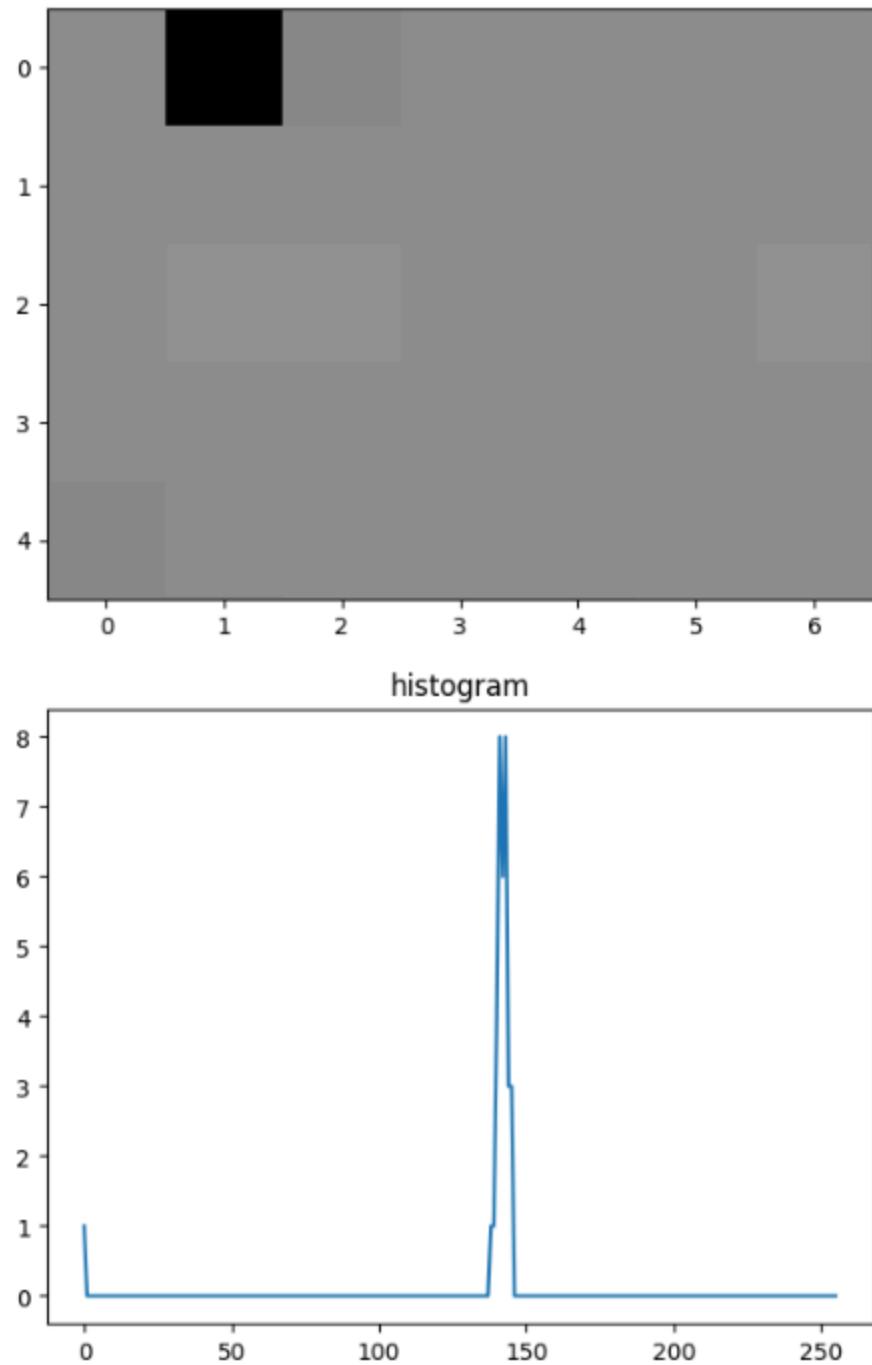
ب) در این بخش می‌خواهیم توابع هیستوگرام، کشش و برش هیستوگرام را تعریف کنیم. به نوبتیوک Q1 مراجعه کنید و توابع نوشته شده را کامل کنید. (۱۰)

پاسخ:

در ابتدا image1 را با استفاده از کتابخانه numpy به صورت یک آرایه در می‌آوریم و سپس تابع calc_hist را با استفاده از یک حلقه می‌نویسیم و خواهیم داشت:

```
4 def calc_hist(image):
5     ...
6     you are free to use libraries
7     calculate image histogram
8     input(s):
9         image (ndarray): input image
10    output(s):
11        hist (ndarray): computed input image histogram
12        ...
13    hist = np.zeros(256)
14    for row in image:
15        for pixel in row:
16            hist[pixel] += 1
17    return hist
```

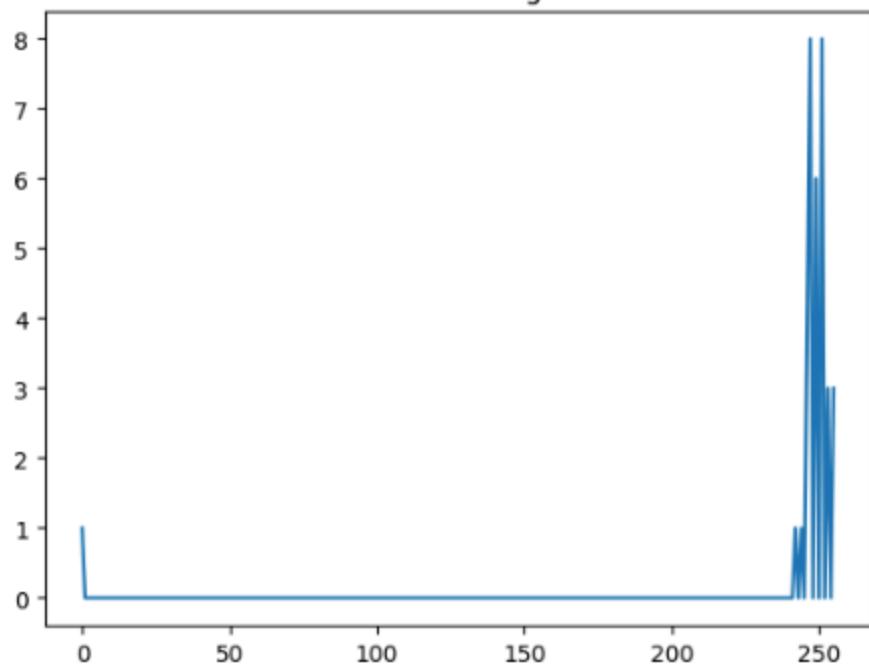
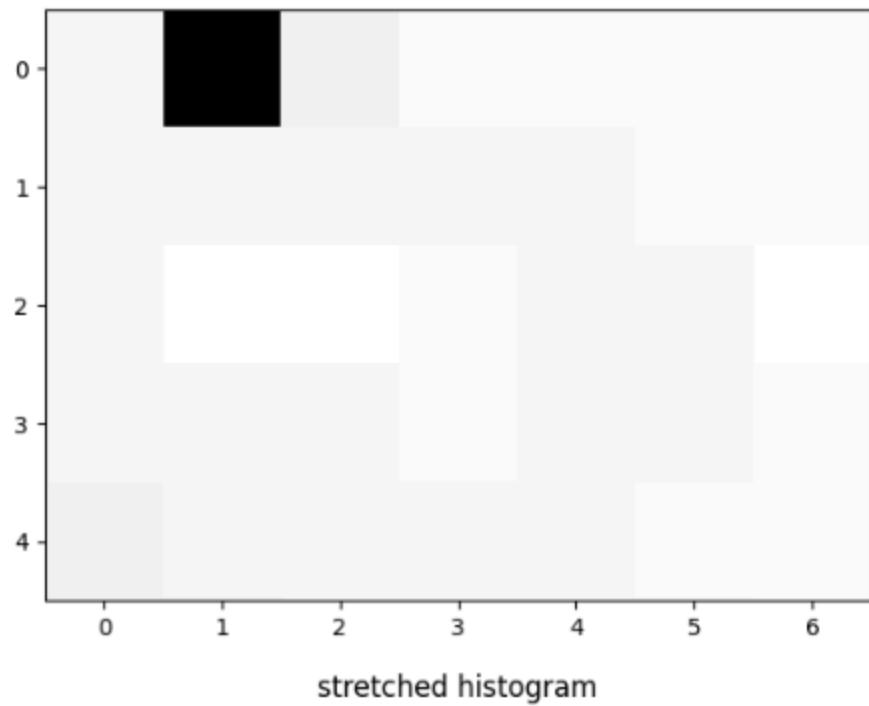
که نتیجه‌ی آن در زیر مشخص است:



و تابع کشش هیستوگرام را نیز به شرح زیر خواهیم داشت:

```
1 #code here
2 #define a function (stretch) for stretching(input:image , output: stretched image)
3
4 def stretch_hist(image):
5     """
6     don't use libraries
7     input(s):
8         image (ndarray): input image
9     output(s):
10        output_image (ndarray): enhanced image with histogram stretching
11    """
12
13     output_image = image.copy()
14     # Start
15     min_val = np.min(output_image)
16     max_val = np.max(output_image)
17     output_image = (255*((image - min_val) / (max_val - min_val))).astype('uint8')
18     # End
19     return output_image
```

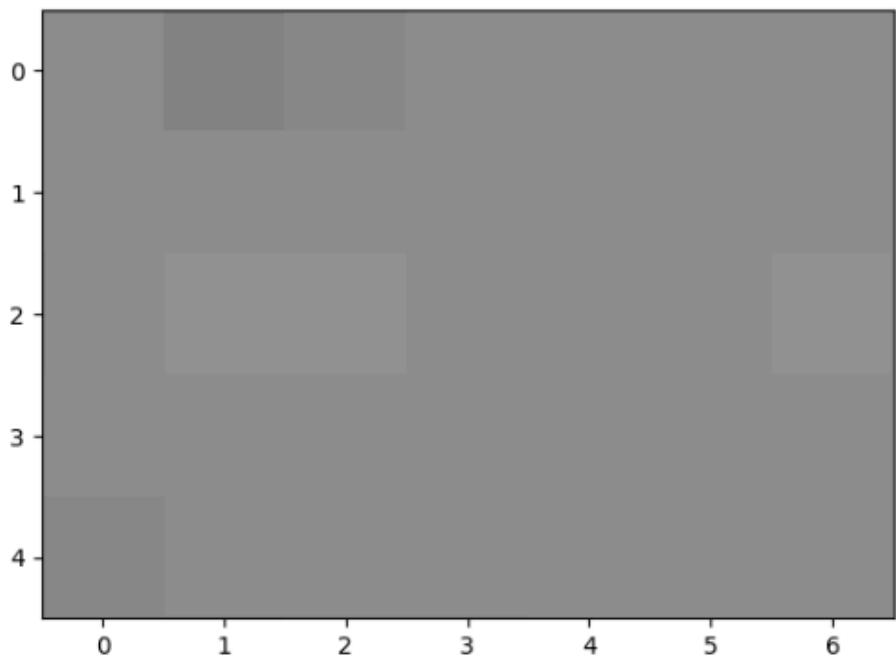
و نمودار های آن نیز به صورت زیر است:



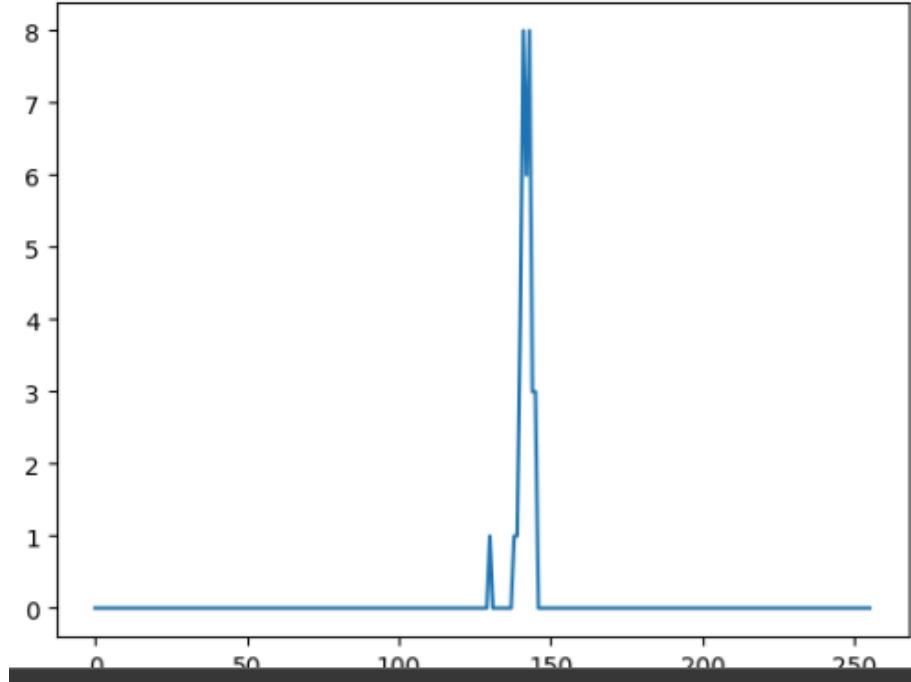
تابع مربوط به برش را نیز به صورت زیر پیاده سازی می کنیم.

```
1 #code here
2 #define a function (clip) for Clipping(input:image , output: clipped image)
3
4 def clip_hist(image, min_value = 30, max_value =120):
5     """
6     don't use libraries
7     input(s):
8         image (ndarray): input image
9         min_value : min value of the histogram which you wanna clip.
10        max_value : max value of the histogram which you wanna clip.
11    output(s):
12        output_image (ndarray): enhanced image with histogram clipping
13    """
14    output_image = image.copy()
15    # Start
16    output_image = np.clip(image, min_value, max_value)
17    # End
18    return output_image
19
```

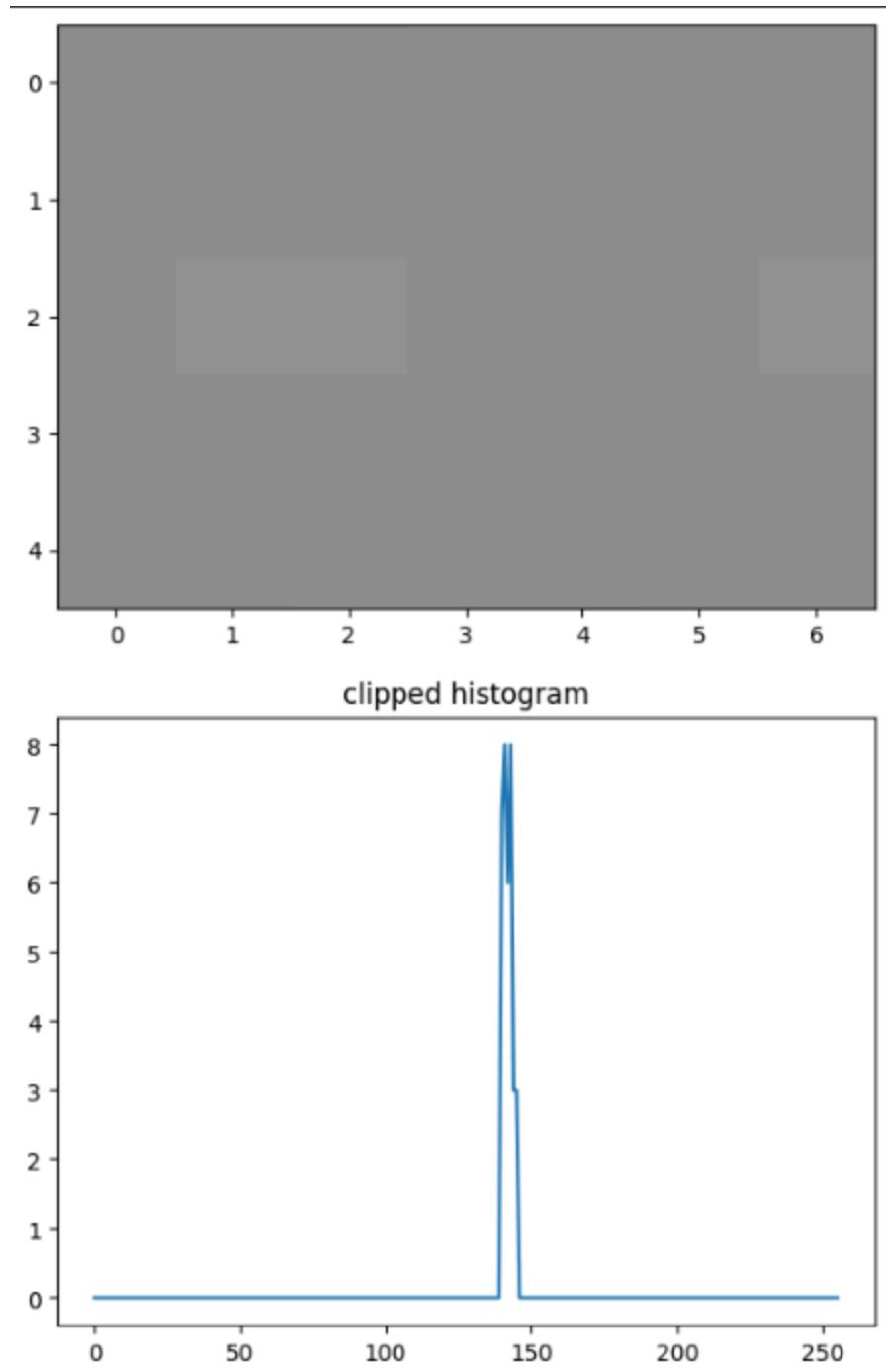
برای بازه‌ی بین {130 و 150} نیز جواب زیر را خواهیم داشت:



clipped histogram



برای بازه‌ی بین 140 تا 145 نیز داریم:



جواب آن در نوتبوک مربوطه موجود است.

ج) تصویر image2 را بخوانید و با استفاده از توابعی که در بخش ب پیاده سازی کردید، نتایج برش و کشش هیستوگرام را بر روی تصویر نشان دهید و در نهایت نتایج هر دو روش را مقایسه کنید و دلیل متفاوت بودن را شرح دهید.(5)

پاسخ:

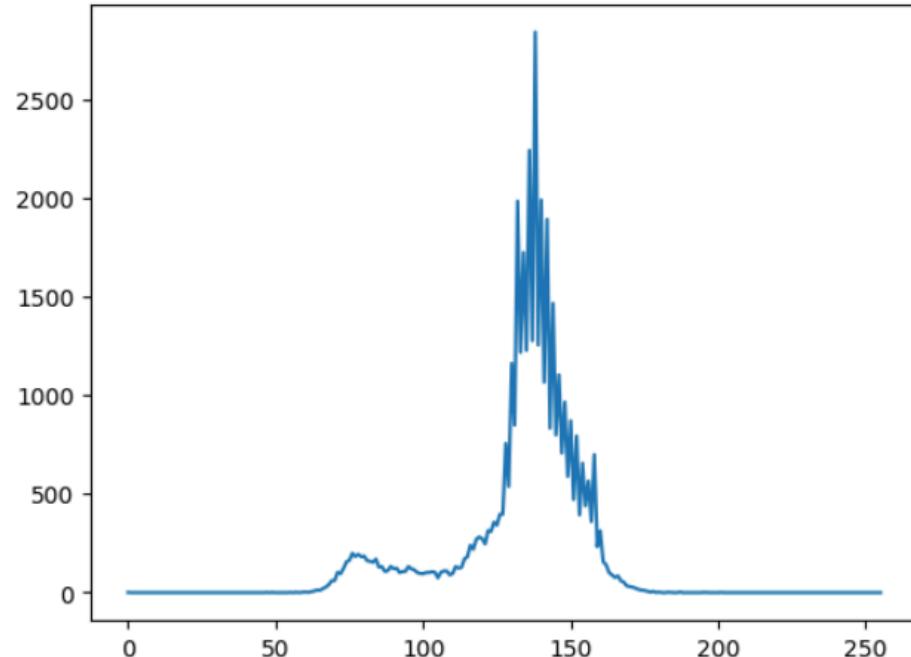
در ابتدا تصویر مربوطه را خوانده و سپس آن را به `grayscale` تبدیل می کنیم و خواهیم داشت:



```
1 plt.plot(calc_hist(clip_hist(input_image, 0, 255)))
2 plt.title('clipped histogram')
```

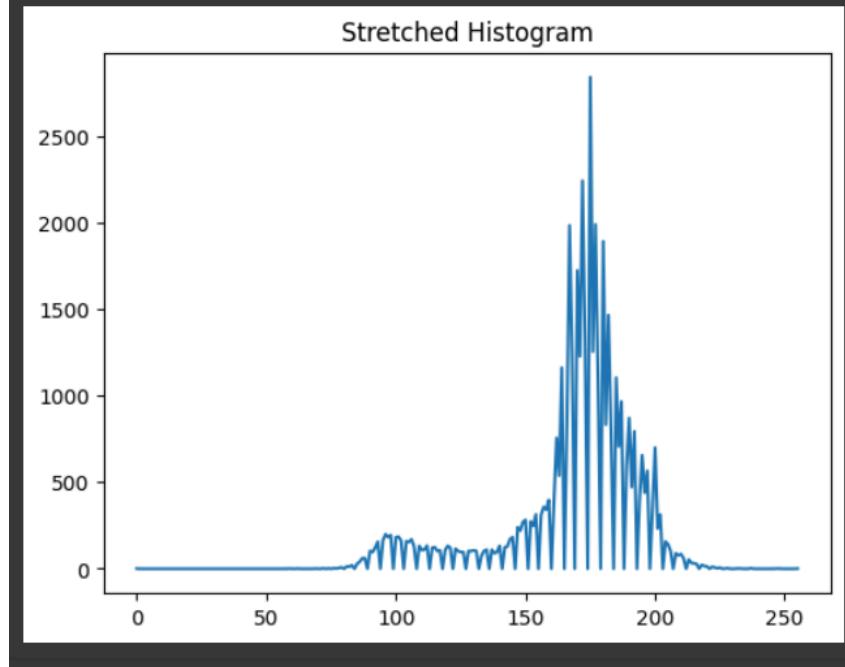
Text(0.5, 1.0, 'clipped histogram')

clipped histogram



حال نمودار مربوطه به عکس پس از کشش را نمایش می دهیم:

```
1 # And visualize the histogram of the stretched image
2
3 stretched_image = stretch_hist[input_image]
4 stretched_image_hist = calc_hist(stretched_image)
5 plt.figure()
6 plt.plot(stretched_image_hist)
7 plt.title('Stretched Histogram')
8 plt.show()
9
```

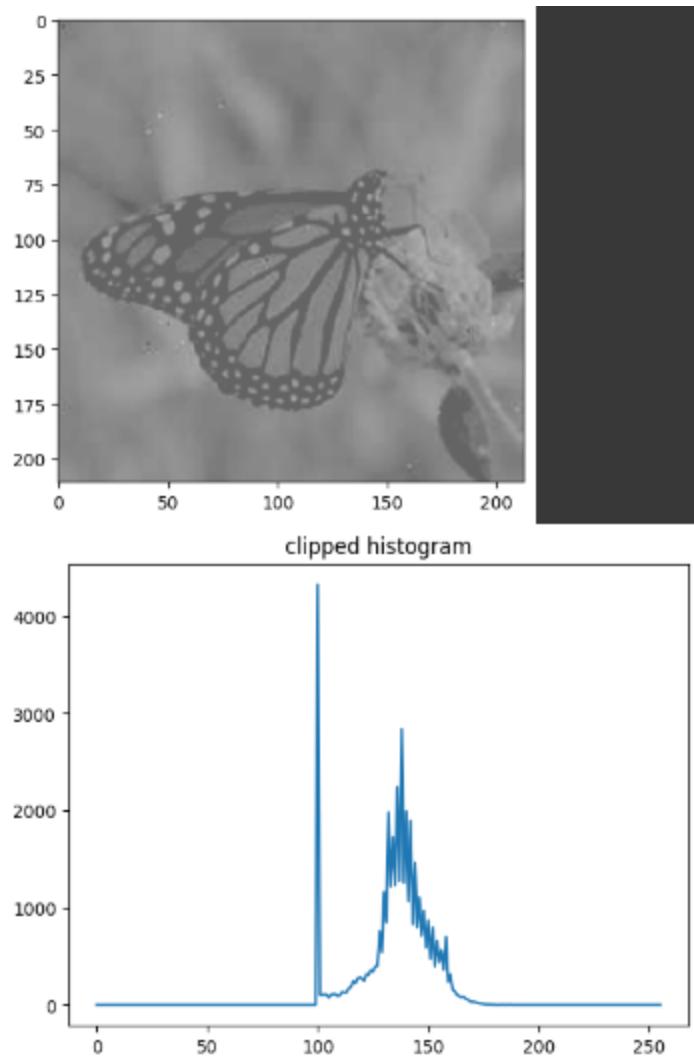


عکس مربوطه نیز به صورت زیر می باشد:

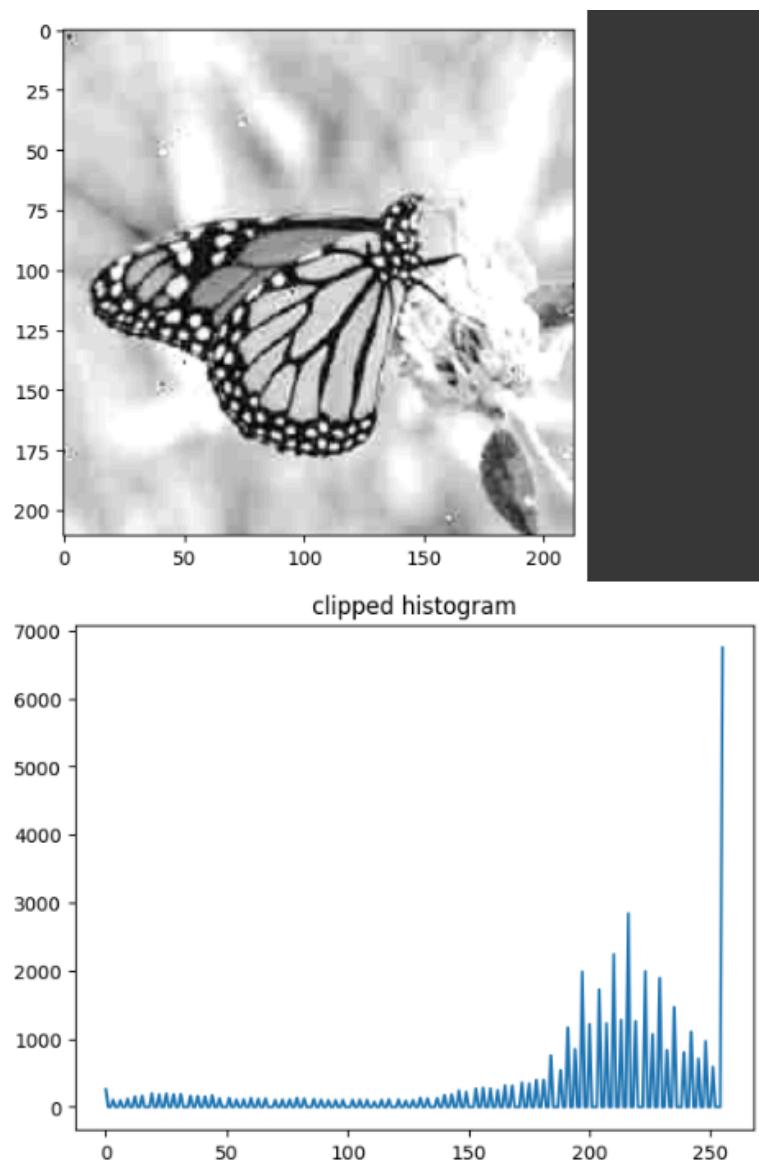
```
1 #use clip_hist function to improve quality of the image and show it
2
3 plt.imshow(stretched_image, cmap='gray', vmin=0, vmax=255)
4 plt.axis('off')
5 plt.show()
```



حال توابع مربوط به برش را در بازه‌ی 100 و 255 اعمال می‌کنیم و خواهیم داشت:



تصویر زیر نیز مربوط به stretching پس از انجام clipping در بازه‌ی 70 تا 150 می‌باشد که می‌بینیم بسیار به کیفیت تصویر افزوده شده است و تصویر از contrast و روشنی بیشتری برخوردار است.



و میدانیم که وقتی که یک هیستوگرام برش داده می‌شود، اطلاعات جزیی از تصویر از بین می‌روند. این باعث می‌شود که دقت و کیفیت تصویر کاهش یابد. با برش دادن هیستوگرام، جزئیات و اطلاعات مهم ممکن است از بین بروند و تصویر به شکل انحرافی تغییر کند و در روش کشش هیستوگرام، با انتقال انحرافی جهت کاهش فشردگی تصویر، کنتراست تصویر افزایش پیدا می‌کند. این باعث بهبود وضوح و تمایز بین مناطق مختلف تصویر می‌شود. همچنین اطلاعات تصویر حفظ شده و تلاش می‌شود تا با افزایش کنتراست، جزئیات بیشتری از تصویر حفظ شوند. بنابراین، هنگامی که از برش یا کشش

هیستوگرام بر روی یک تصویر استفاده می‌کنیم، باید به دقت تغییرات و تفاوت‌هایی که این دو روش در تصویر ایجاد می‌کنند توجه کنیم. انتخاب مناسب بین این دو روش بستگی به نیازهای خاص پروژه و هدف پردازش تصویر دارد.

2

الف) تطبیق هیستوگرام را بر روی دو تصویر زیر پیاده سازی کنید.(5)

4	7	6	5	2	5	7	3
4	7	6	5	2	5	7	3
4	7	6	5	2	5	7	3
4	7	6	5	2	5	7	3
4	7	6	5	2	5	7	3
4	7	6	5	2	5	7	3
4	7	6	5	2	5	7	3
4	7	6	5	2	5	7	3

Ref

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

src

به منظور پیاده سازی تطبیق هیستوگرام باید مقادیر پیکسل های تصویر src را به مقادیر مربوطه در تصویر reference، باید map کنیم. ابتدا باید cumulative distribution functions را برای هر دو تصویر و سپس مقادیر mapping را به دست آوریم. در ابتدا باید هیستوگرام تصویر src را به دست آوریم. سپس احتمال هر عدد را به دست می آوریم و که خواهیم داشت:

$$n(0) = 8, \quad p(0) = 8/64$$

$$n(0) = 8, \quad p(1) = 32/64$$

$$n(0) = 8, \quad p(2) = 24/64$$

حال باید مقدار cumulative sum را حساب کنیم و خواهیم داشت

$$\text{CumulativeSum}(0) = 8/64 = 0.125$$

$$\text{CumulativeSum}(1) = 40/64 = 0.625$$

$$\text{CumulativeSum}(0) = 64/64 = 1$$

حال مراحل بالا را برای تصویر ref انجام می دهیم و خواهیم داشت:

CumulativeSum(2) = 1/8

CumulativeSum(3) = 2/8

CumulativeSum(4) = 3/8

CumulativeSum(5) = 5/8

CumulativeSum(6) = 6/8

CumulativeSum(7) = 8/8

حال مقادیر متناظر بین دو نمودار را پیدا کرده و آنها را جایگزین می کنیم.

لذا با توجه به آن مقادیر صفر به 2 و مقدار 1 به 5 و مقدار 2 به مقدار 7 نگاشت می شود.

1 matched_image

```
array([[2, 2, 2, 2, 2, 2, 2, 2],  
       [5, 5, 5, 5, 5, 5, 5, 5],  
       [5, 5, 5, 5, 5, 5, 5, 5],  
       [7, 7, 7, 7, 7, 7, 7, 7],  
       [7, 7, 7, 7, 7, 7, 7, 7],  
       [7, 7, 7, 7, 7, 7, 7, 7],  
       [5, 5, 5, 5, 5, 5, 5, 5],  
       [5, 5, 5, 5, 5, 5, 5, 5]], dtype=uint8)
```

کد مربوطه در نوتبوک موجود می باشد.

(ب) به نوتبوک Q2 مراجعه کنید و قسمتهای خواسته شده را تکمیل کنید. در این قسمت از عکس های

موجود در فolder Q2 استفاده کنید و خروجی هر بخش را تحلیل کنید. توجه داشته باشید در این بخش

مجاز به استفاده از کتابخانه ای جز numpy نیستید. (۱۰)

در صورت پیاده سازی بدون استفاده از حلقه for و while و... نمره امتیازی دریافت میکنید. (۱۰)

ابندا تابع هیستوگرام را بدون استفاده از حلقه ها به صورت زیر می نویسیم:

```

1 def calc_hist(image):
2     ...
3     Do not use libraries
4     calculate image histogram
5     input(s):
6         | image (ndarray): input image
7     output(s):
8         | hist (ndarray): computed input image histogram
9         ...
10    hist = np.zeros(256,dtype=int)
11    #####
12    # your code here #
13    hist = np.bincount(image.flatten(),minlength = 256)
14
15    #####
16
17    return hist

```

حال باید تابع محاسبه‌ی `cdf` را پیاده سازی کنیم که خواهیم داشت:

```

1 def calc_cdf(channel):
2     ...
3     Do not use libraries
4     calculate image cdf
5     input(s):
6         | channel (ndarray): input image channel
7     output(s):
8         | cdf (ndarray): computed cdf for input image channel
9     ...
10
11    #####
12    # your code here #
13    cdf = np.zeros(256, dtype=int)
14    hist = calc_hist(channel)
15    cumulative_sum = 0
16    for i in range(0, 256):
17        cumulative_sum += hist[i]
18        cdf[i] = cumulative_sum / len(channel)

```

حال باید تابع تطبیق هیستوگرام را پیاده سازی کنیم و خواهیم داشت:

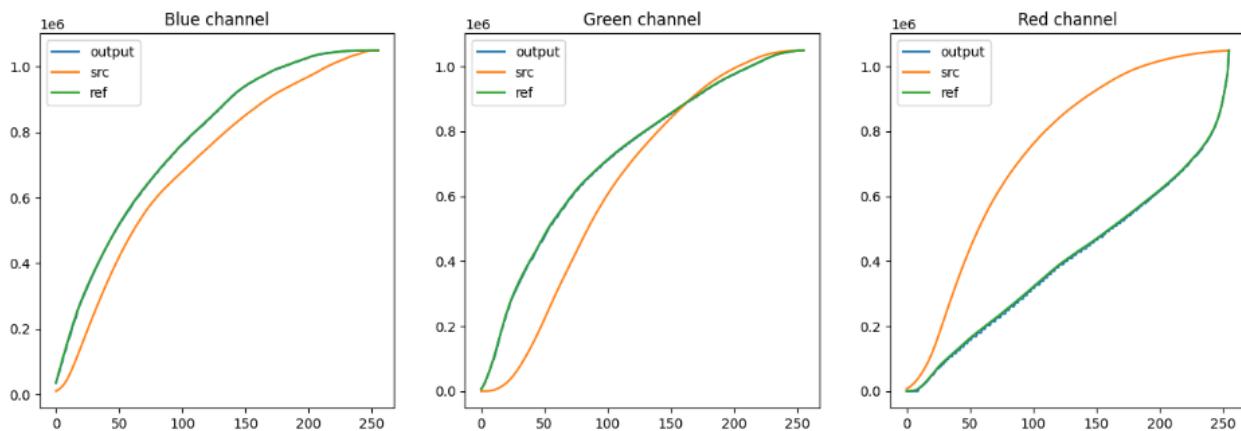
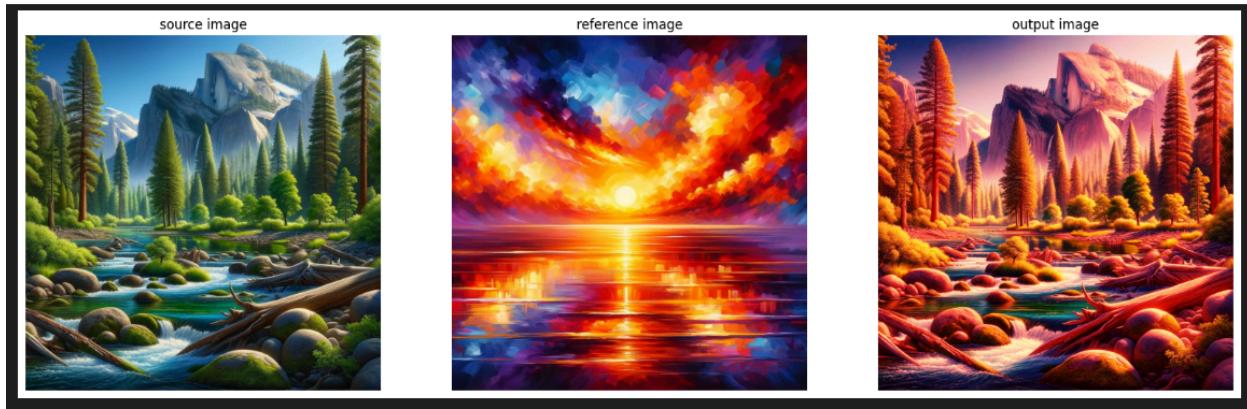
```
def hist_matching(src_image, ref_image):
    """
    don't use libraries
    input(s):
        src_image (ndarray): source image
        ref_image (ndarray): reference image
    output(s):
        output_image (ndarray): transformation of source image so that its histogram matches histogram of reference image
    """

    output_image = src_image.copy()
    channels = [(0, 'Blue channel'), (1, 'Green channel'), (2, 'Red channel')]
    for channel, title in channels:

        #####
        # your code here #
        src_hist = np.zeros(256, dtype=int)
        ref_hist = np.zeros(256, dtype=int)

        # Calculate histograms
        for pixel_value in src_image[:, :, channel].flatten():
            src_hist[pixel_value] += 1
        for pixel_value in ref_image[:, :, channel].flatten():
            ref_hist[pixel_value] += 1
        # Calculate CDFs
        src_cdf = src_hist.cumsum()
        ref_cdf = ref_hist.cumsum()
        # Normalize CDFs
        src_cdf = src_cdf / src_cdf[-1]
        ref_cdf = ref_cdf / ref_cdf[-1]
        # Create a mapping from src to ref values based on CDFs
        mapping = np.zeros(256, dtype=np.uint8)
        j = 0
        for i in range(256):
            while j < 256 and src_cdf[i] > ref_cdf[j]:
                j += 1
            if j < 256:
                mapping[i] = j
        # Apply the mapping to get the output image
        mapped_channel = mapping[src_image[:, :, channel]]
        output_image[:, :, channel] = mapped_channel
        #####
    return output_image
```

لذا در این تابع ابتدا مقدار هیستوگرام و cdf مربوط برای هر کانال محاسبه می شود و سپس از تطبیق آن دو برای match کردن استفاده می کنیم که نتیجه‌ی آن را در زیر مشاهده می کنیم که معلوم است که شکل تصویر عوض نشده است اما از نظر رنگ متفاوت شده است.



3

همانطور که در اسلایدها اشاره شد، روش‌های ارتقا تصویر مانند متعادل سازی، برش و کشش هیستوگرام سراسری هستند و روش‌هایی مانند ACE و CLAHE روش‌های ارتقا محلی هستند.

الف) به نوتبوک Q3 مراجعه کنید و ابتدا متعادل سازی هیستوگرام را با کتابخانه OpenCV پیاده سازی کنید. نتیجه را گزارش کنید. معایب این روش چیست؟

متعادل سازی هیستوگرام به صورت سراسری در بیشتر عکس‌ها منجر به کاهش کیفیت تصویر و بدتر شدن contrast آن می‌شود به طوری که عنوان مثال نواحی روشن، روشن‌تر می‌شوند و نواحی تیره‌تر نیز تیره‌تر می‌شوند و در عکس پایین نیز می‌بینیم که جزئیات این نواحی (مثال مجسمه) از بین میرود. و بسیاری از تصاویر (مانند این تصویر) نیاز دارند که ارتقا در آنها به صورت محلی صورت گیرد. علاوه بر این موضوع، روش سراسری می‌تواند باعث تقویت نویز در تصویر شود و کیفیت تصویر را کاهش دهد که در زیر مشاهده می‌شود.

```

1 image = cv2.imread('img_improvment.png',cv2.IMREAD_GRAYSCALE)
2 output_image = image.copy()
3 # equalize_image = ## your code here ##
4 equalize_image = cv2.equalizeHist(image)

```



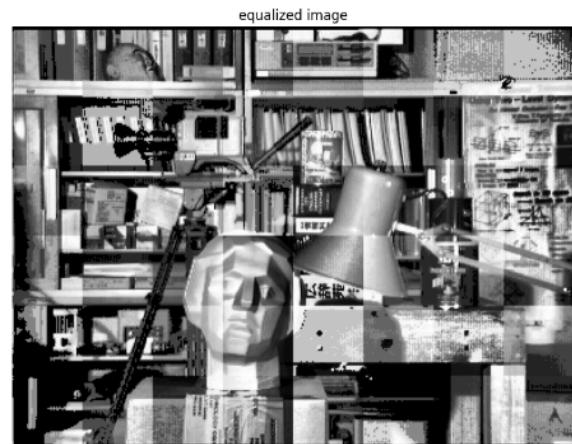
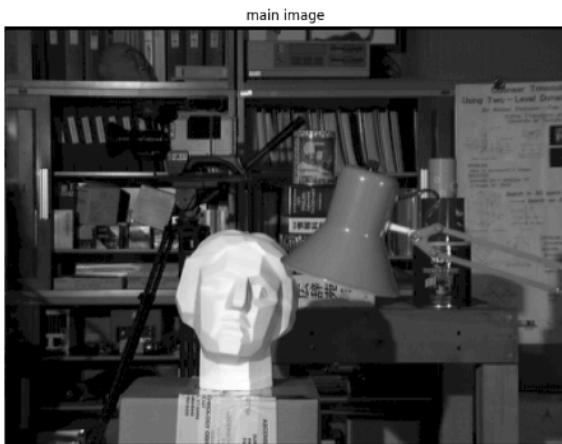
ب) حال به سراغ روش‌های بهبود محلی می‌رویم. توابع مربوط به ACE (هر دو روش) و CLAHE را تکمیل کنید و خروجی به دست آمده را تحلیل کنید و مزایا و معایب هر یک را ذکر کنید.

در روش اول تصویر را به بخش‌های کوچک‌تری تقسیم می‌کنیم و در نتیجه‌ی آن هر بخش بدون توجه به بخش‌های دیگر بهتر می‌شود. این کار باعث می‌شود که مرزهای هر بخش در خروجی مشخص باشند و یکپارچگی تصویر از بین برود که در تصویر زیر نیز مشاهده می‌شود.

```

1 def ACE1(image, gridSize):
2     """
3         you can use the equalize function of OpenCV for each grid
4         Use first method for ACE implementation (calculating transition function for each grid)
5         input(s):
6             image (ndarray): input image
7             gridSize (int): window size for calculating histogram equalization
8         output(s):
9             output (ndarray): improved image
10    ...
11    x,y = image.shape
12    output_image = image.copy()
13    #####
14    #    your code here    #
15    height, width = image.shape
16    output_image = np.zeros_like(image)
17    numGridsX = width // gridSize
18    numGridsY = height // gridSize
19
20    for x in range(0, numGridsX):
21        for y in range(0, numGridsY):
22            # Calculate the start and end points of the current grid
23            startX = x * gridSize
24            endX = (x + 1) * gridSize
25            startY = y * gridSize
26            endY = (y + 1) * gridSize
27            # Extract the current grid from the image
28            grid = image[startY:endY, startX:endX]
29            # Apply histogram equalization to the grid
30            equalized_grid = cv2.equalizeHist(grid)
31            # Place the processed grid back into the output image
32            output_image[startY:endY, startX:endX] = equalized_grid
33    #####
34    return output_image

```



در روش دوم ACH هر پیکسل را با توجه به پیکسل های مجاورش تغییر می دهیم. در این روش ممکن است نویز نیز تقویت شود زیرا در بین پیکسلهای مجاور که اختلاف رنگ چندانی ندارند، ممکن است اختلاف رنگ بیشتر شود و نویز محسوس تر شود. در این روش به ازای هر پیکسل یک grid در اطراف آن در نظر میگیریم و متعادل سازی را روی آن اعمال کرده و در تصویر جایگزین می کنیم. در هر صورت این روش نسبت به روش اول که باعث نمایان شدن مرزهای هر بخش میشد، در این روش نتایج بهتری قابل دستیابی هستند.

```

1 def ACE2(image, gridSize):
2     """
3         you can just use the equalize function of OpenCV for each grid
4         You can use OpenCV built-in tools for applying padding
5         Use second method for ACE implementation (calculating transition function for each pixel)
6         input(s):
7             image (ndarray): input image
8             gridSize (tuple): window size for calculating histogram equalization
9         output(s):
10            output (ndarray): improved image
11            ...
12
13            #####
14            # your code here #
15            # padHeight = (gridHeight - height % gridHeight) % gridHeight
16            # padWidth = (gridWidth - width % gridWidth) % gridWidth
17            paddedImage = cv2.copyMakeBorder(output, gridSize[0]//2, gridSize[0]//2, gridSize[1]//2,
18                                            [gridSize[1]//2, cv2.BORDER_REFLECT])
19            paddedHeight, paddedWidth = output.shape
20            for i in range(paddedHeight):
21                for j in range(paddedWidth):
22                    # grid = paddedImage[y:y+gridHeight, x:x+gridWidth]
23                    # equalizedGrid = cv2.equalizeHist(grid)
24                    x_start, x_end = i, i + gridSize[0]
25                    y_start, y_end = j, j + gridSize[1]
26
27                    pixel_grid = paddedImage[x_start: x_end, y_start: y_end]
28                    equalized_pixelGrid = cv2.equalizeHist(pixel_grid)
29                    output[i, j] = equalized_pixelGrid[gridSize[0] //2, gridSize[1] // 2]
30
31            #####
32
33    return output

```



ج) حال روش CLAHE را با استفاده توابع `opencv` پیاده سازی کنید. سپس می خواهیم تاثیر ابعاد پنجره و حد برش را بر روی خروجی این تابع بررسی کنیم. پارامترهای زیر را امتحان کنید و با توجه به خروجی مقایسه و نتیجه گیری کنید.

این روش دقیقاً مانند قسمت قبل است فقط باید به جای استفاده از تابع آماده `equalizer`، این تابع را خودمان تعریف می کنیم و در این روش نیز مشکل تقویت نویز نیز در روش قبل حل می شود و توزیع ما به نرمال نزدیکتر می شود. ابتدا هیستوگرام محاسبه شده و اختلاف با `limit` پس از نرمالایز شدن به هیستوگرام اصلی اضافه می شود. در انتها `cdf` جدید به صورت تصویر بازگردانده می شود.

4

الف) تابع `AddNoise` را طوری کامل کنید که به تصویر ورودی نویز نمک و فلفل اضافه کند. سپس تصویر `4image` را بخوانید و نویز نمک و فلفل را به آن اضافه کنید.

```

1  def Add_Noise(img):
2      """
3          Add salt and pepper noise to the input image.
4          Parameters:
5              image: Input image (numpy array).
6          Returns:
7              Image with salt and pepper noise added.
8      """
9      img_noisy = img.copy()
10     salt_pepper_ratio = 0.25 # This means 2% of the pixels will be affected
11     amount = salt_pepper_ratio * img_noisy.size # Total number of affected pixels
12     # Calculate the number of salt and pepper pixels
13     num_salt = np.ceil(amount * 0.5).astype(int)
14     num_pepper = np.ceil(amount * 0.5).astype(int)
15     # Add Salt noise (white pixels)
16     coords = [np.random.randint(0, i - 1, num_salt) for i in img.shape]
17     img_noisy[coords[0], coords[1]] = 255
18     # Add Pepper noise (black pixels)
19     coords = [np.random.randint(0, i - 1, num_pepper) for i in img.shape]
20     img_noisy[coords[0], coords[1]] = 0
21     return img_noisy
22     # return img

```

cat with noise



ب) در این سوال قصد داریم با فیلترهای هموارساز آشنا شویم. به نوت بوک Q4 مراجعه کنید و توابع نوشته شده که مربوط به سه فیلتر متوسطگیر، میانه و گوسی هستند را کامل کنید. نتیجه فیلترها بر روی تصویر و همچنین تأثیر سایز کرنل بر روی نتیجه را تحلیل کنید. توجه داشته باشید که برای اعمال فیلتر باید از Reflect101 استفاده کنید و مجاز به استفاده از توابع آماده نیستید.

سه فیلتر متوسط‌گیر، میانه و گاوی از فیلترهای پرکاربرد در پردازش تصویر هستند. هر یک از این فیلترها دارای ویژگی‌ها و عملکردهای مختلفی هستند و در پردازش تصویر برای کاهش نویز، حذف جزئیات غیر مهم و بهبود کیفیت تصویر استفاده می‌شوند.

تابع فیلتر میانگین:

```
1 def Averaging_Blurring(img, filter_size):
2     ...
3     Do not use libraries
4     input(s):
5         img (ndarray): input image
6         filter_size (ndarray): filter size
7     output(s):
8         result (ndarray): computed averaging blurring
9     ...
10    image = Reflect101(img, filter_size)
11    result = np.zeros((img.shape))
12    #####
13    # your code here #
14    for i in range(result.shape[0]):
15        for j in range(result.shape[1]):
16            neighborhood = image[i:i+filter_size, j:j+filter_size]
17            result[i, j] = np.sum(neighborhood) / (filter_size * filter_size)
18    #####
19    return result
```

تابع فیلتر میانه:

```
1 def Median_Blurring(img, filter_size):
2     ...
3     Do not use libraries
4     input(s):
5         img (ndarray): input image
6         filter_size (ndarray): filter size
7     output(s):
8         result (ndarray): computed median blurring
9     ...
10    image = Reflect101(img, filter_size)
11    result = np.zeros((img.shape))
12    #####
13    # your code here #
14    for i in range(result.shape[0]):
15        for j in range(result.shape[1]):
16            neighborhood = image[i:i+filter_size, j:j+filter_size]
17            result[i, j] = np.median(neighborhood)
18    #####
19    return result
```

تابع فیلتر گاوی:

```
1 def Gaussian_Blurring(img, filter_size, std):
2     """
3     Do not use libraries
4     input(s):
5         img (ndarray): input image
6         filter_size (tuple): filter size
7         std (float): std of gaussian kernel
8     output(s):
9         result (ndarray): computed gaussian blurring
10    """
11    kernel = np.zeros((filter_size,filter_size))
12    #####
13    # your code here #
14    image = Reflect101(img, filter_size)
15    result = np.zeros(img.shape)
16    kernel_center = (filter_size // 2, filter_size // 2)
17    for i in range(filter_size):
18        for j in range(filter_size):
19            distance = (i - kernel_center[0])**2 + (j - kernel_center[1])**2
20            kernel[i, j] = np.exp(-distance / (2 * std**2))
21    kernel /= np.sum(kernel)
22    for i in range(filter_size):
23        for j in range(filter_size):
24            neighborhood = image[i:i+filter_size, j:j + filter_size]
25            result[i, j] = np.sum(neighborhood * kernel[:neighborhood.shape[0], :neighborhood.shape[1]])
26    #####
27    output = img.copy()
28    result = cv2.filter2D(src = output, ddepth = -1, kernel = kernel)
29    return result
```

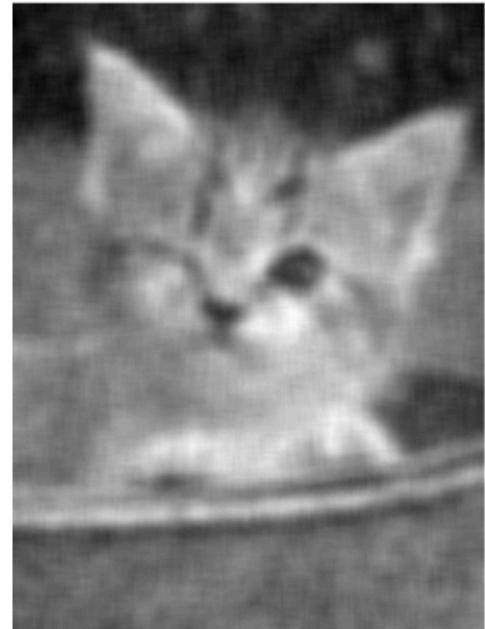
تصاویر آن به شرح زیر است:

عکس گربه با نویز:

main image



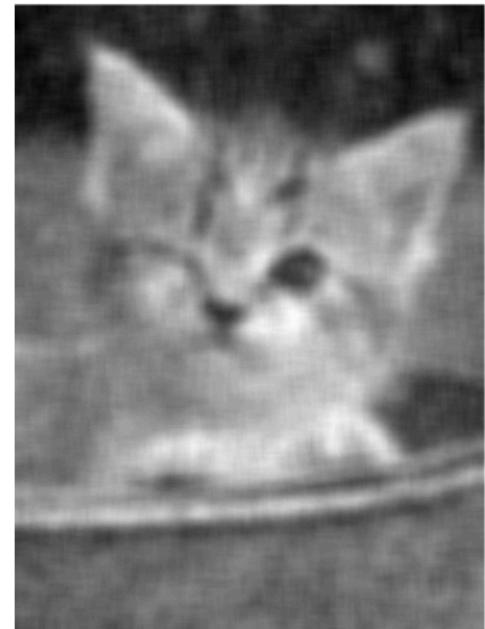
Averaging Blurring



Median Blurring



Gaussian Blurring



عکس گربه بدون نویز:



Median Blurring



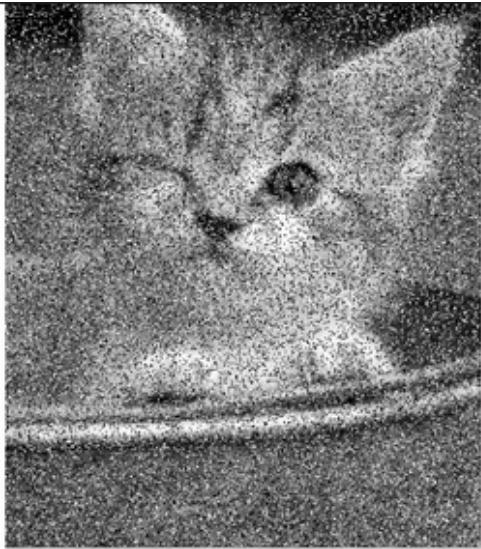
Gaussian Blurring



پارت سی نیز به شرح زیر است:

```
1 AveragingBlurring = cv2.blur(image, (15, 15))## your code here ##
2 MedianBlurring = cv2.medianBlur(image, 15)## your code here ##
3 GaussianBlurring = cv2.GaussianBlur(image, (15, 15), 40)## your code here ##
```

عکس گربه با نویز:



Median Blurring



Gaussian Blurring



عکس گربه بدون نویز:



Median Blurring



Gaussian Blurring



5

نتیجه اعمال عملگر لاپلاسین را بر روی تصویر زیر محاسبه کنید.

[10 ,10 ,10 ,10 ,10 ,10 ,10 ,10]

[10 ,10 ,10 ,10 ,10 ,10 ,10 ,10]

[10 ,10 ,10 ,10 ,10 ,10 ,10 ,10]

[10 ,10 ,10 ,10 ,10 ,10 ,10 ,10]
[10 ,10 ,10 ,12 ,10 ,10 ,10 ,10]
[10 ,10 ,10 ,10 ,10 ,10 ,10 ,10]
[10 ,10 ,10 ,10 ,10 ,10 ,10 ,10]
[10 ,10 ,10 ,10 ,10 ,10 ,10 ,10]

برای اعمال لaplacian کانولوشن تصویر را با ماتریس زیر محاسبه می کنیم:

[0 ,1- ,0]
[1- ,5 ,1-]
[0 ,1- ,0]

و برای انجام padding از روش replicate استفاده می کنیم. در این روش یک ردیف 10 به هر 4 قسمت تصویر اضافه می شود و سپس فرمول گوسین را روی آن پیاده می کنیم.

$$g(0,0) = \sum_{s=-1}^{+1} \sum_{t=-1}^{+1} w(s,t)f(-s,-t)$$

$$= 0 * 10 + (-1) * 10 + 0 * 10 + (-1) * 10 + 5 * 10 + (-1) * 10 + 0 * 10 + (-1) * 10 + 0 * 10 = 0 + -10 + 0 - 10 + 50 - 10 + 0 - 10 + 0 = 10$$

لذا مقادیر خانه های 10 همان 10 باقی می ماند و در بقیه خانه ها نیز داریم:

$$g(4, 3) = g(3, 4) = g(4, 5) = g(5, 4) = -10 - 10 + 50 - 12 - 10 = 8$$

$$g(4, 4) = -10 - 10 - 10 - 10 + 60 = 20$$

نقاطی که دارای مقادیر دیگری هستند با رنگ سبز نشان داده شده اند.

[10, 10, 10, 10, 10, 10, 10, 10, 10]
[10, 10, 10, 10, 10, 10, 10, 10, 10]
[10, 10, 10, 10, 10, 10, 10, 10, 10]
[10, 10, 10, 10, 8, 10, 10, 10]
[10, 10, 10, 8, 20, 8, 10, 10]
[10, 10, 10, 10, 8, 10, 10, 10]
[10, 10, 10, 10, 10, 10, 10, 10]
[10, 10, 10, 10, 10, 10, 10, 10]

مقدار پیکسل های یک تصویر 2 در 2 به صورت زیر است. تبدیل فوریه این تصویر را حساب کنید.

$$\begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 1 \\ \hline 1 & 2 \\ \hline \end{array}
 \quad F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

$$\xrightarrow{\frac{M=N}{N=1}} F(u, v) = \sum_{x=0}^1 \sum_{y=0}^1 f(x, y) e^{-j\pi(ux+vy)}$$

$$F(0, 0) = f(0, 0) + f(0, 1) + f(1, 0) + f(1, 1) = 2 + 1 + 2 + 1 = 6$$

$$F(0, 1) = f(0, 0) + f(0, 1) e^{-j\pi} + f(1, 0) + f(1, 1) e^{j\pi} =$$

$$2 + 1(-1) + 1 + 2(-1) = 0$$

$$F(1, 0) = f(0, 0) + f(0, 1) + f(1, 0) e^{-j\pi} + f(1, 1) e^{j\pi}$$

$$= 2 + 1 + 1(-1) + 2(-1) = 0$$

$$F(1, 1) = f(0, 0) + f(0, 1) e^{-j\pi} + f(1, 0) e^{-j\pi} + f(1, 1) e^{-2j\pi}$$

$$= 2 + 1(-1) + 1(-1) + 2 = 2$$

$$F(U, V) = (6, 0)$$

$$(0, 2)$$