گزارش تمارین

سروش باقرنژاد

اطلاعات گزارش	چکیده
تاریخ:	فیلتر ها در پردازش تصویر ابزار هایی هستند که به ما کمک میکنند تصاویرمان را بر حسب نیاز تغییر دهیم و یا اطلاعات مهم تصویر را مثل لبه ها ی یک تصویر را استخراج کنیم. از فیلتر های پرکاربرد میتوان به فیلتر میانگین که در حذف نویز کاربرد دارد اشاره کرد.
واژگان کلیدی: فیلتر میانگین فیلتر باکس Robert operation تشخیص لبه تبدیلات فیلتر 5*5	

1-مقدمه

در سوال 2 قسمت 1 روشی برای بهبود فیلتر میانگین خواسته شده که ما این کار را به کمک خصوصیات فیلتر box و همچنین ساده سازی فیلتر convolution انجام میدهیم . در این روش، میانگین گیری با تقریباتی انجام میشود.

2-شرح تکنیکال

شرح قسمت 2.1 :

در سوال 2.1 از ما خواسته شده که بهبودی برای فیلتر میانگین ارایه دهیم، اگ بخواهیم فیلتر میانگین را بر روی تصویری به ابعاد M*N پیاده سازی کنیم ابتدا باید ماسکی در ابعاد m*n ایجاد کنیم و این ماسک را به ازای هر پیکسل از تصویر روی تصویر بلغزانیم تا میانگین پیکسل های اطراف آن را به عنوان خروجی به ما بدهد .

فرض کنیم ماسک ما ابعاد 3*3 دارد . تصویر زیر نمایانگر این ماسک میباشد :

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}.$$

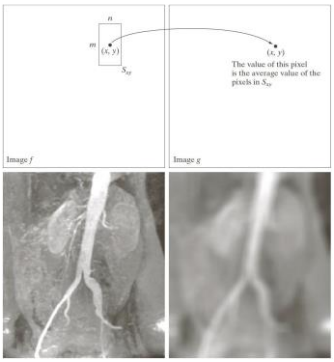
در این نوع ماسک ما دو پارامتر مهم به نام r که شعاع را نشان میدهد از مرکز ماسک و N که تعداد پیکسل ها را نمایش میدهد ، داریم . در این مثال 3*3 :

r=1 و N=9 میباشد.

اگر بخواهیم با این ماسک مقدار جدید پیکسل در درایه ی (i,j) را بدست بیاوریم باید میانگین 9 درایه ی اطراف آن را بگیریم به ازای هر پیکسل ما 9 عمل ضرب داریم و در واقع ما به ازای هر پیکسل m*n عمل ضرب انجام میدهیم

و اگر بخواهیم برای کل پیکسل های تصویر این کار را انجام دهیم M*N*m*n بار عمل ضرب تکرار میشود پس در این روش ما مرتبه ی زمانی O(M*N*m*n) را داریم که برای تصویر هایی با رزولوشن بالا و ماسک هایی با ابعاد بزرگ، پیچیدگی زمانی بالایی میباشد .

تصویر زیر نحوه ی اجرای این الگوریتم و تاثیر آن بر روی تصویر را نشان میدهد که تصویر smooth شده است.



- **بهبود الگوریتم**

از آنجایی که فیلتر میانگین فیلتری جدا شونده (separable) هست ما میتوانیم آن را به صورت فیلتر هایی ساده تر بنویسیم . در واقع فیلتر 2 بعدی کانولوشن به صورت دو فیلتر یک بعدی جدا میشود . این کار مرتبه زمانی ما را کاهش میدهد .

نحوه ی جدا کردن فیلتر میانگین 3*3 به دو فیلتر 3*1 در شکل زیر نمایش داده شده است .

$$\frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} * \frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix} = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

چون دو طرف معادله با هم برابرند در نتیجه ضرب ماتریس تصویر ما در هر کدام از این طرفین نتیجه یکسانی میدهد اما تفاوت آن در تعداد ضرب هاست . همانطور که از شکل پیداست در سمت چپ ما تنها 6 عمل ضرب لازم داریم یا به عبارت دیگر m+n عمل نیاز داریم در صورتی که اگر از ماسک سمت راست استفاده کنیم m*n عمل ضرب نیاز داریم در نتیجه مرتبه زمانی ما به O(M*N*(m+n)) کاهش پیدا میکند .

در تصویر زیر نوع دیگری از کاهش ابعاد فیلتر را برای فیلتر میانگین وزن دار مشاهده میکنید .

$$\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$$

در حالت اولیه ما با مرتبه زمانی O(M*N*m*n) یا $O(Nr^2)$ طرف بودیم اما با کاهش ابعاد ماسک به 2r+1 آرایه یک بعدی ما به مرتبه زمانی O(Nr) میرسیم که مربته زمانی نمایی را به چند جمله ای کاهش دادیم اما هنوز هم میشود این مرتبه ی زمانی را بهبود داد .

- **بهبود مرتبه زمانی با روش dp**

با استفاده از متد های رایج dynamic programming میتوانیم از جمع کردن تکراری جلوگیری کنیم و با داشتن یک جدول حاصل جمع ها را ذخیره و فقط پیکسل های قدیمی را از آن کم و پیکسل های جدید را به آن اضافه کنیم .

با این کار ما مرتبه زمانی را به O(N) کاهش میدهیم و در مرتبه خطی میانگیر گیری را انجام میدهیم .

شرح قسمت 2.2 :

Robert operator : یک عملگر بر پایه گرادیان کاهشی است که حاصل جمع مجذور های بردار های قطری یک تصویر را بدست میاورد . این عملگر از دو کرنل زیر برای محاسبات استفاده میکند .

$$M_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad M_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

برای پیدا کردن لبه با این روش ابتدا عکس ورودی که mosque هست را grayscale تبدیل میکنیم سپس عملگر Robert را بر آن اعمال میکنیم و برای بهتر مشخص کردن لبه ها عملیات thresholding را انجام میدهیم و مقادیر لبه ها را افزایش میدهیم .

استفاده از این متد مزایا و معایبی دارد . از مزایای آن میتوان به پیدا کردن راحت لبه ها و لبه های قطری اشاره کرد اما این روش زیاد دقیق نیست و نسبت به نویز بسیار حساس هست به طوری که کارایی آن هنگام مواجهه با نویز به شدت کاهش پیدا میکند .

اگر به تصویر اصلی باکس فیلتر را اضافه کنیم با توجه به اینکه این فیلتر ها عمل میانگین گیری را انجام میدهند، لبه های تصویر دیگر قابل مشاهده نخواهد بود .

شرح قسمت 2.3 و 2.4:

فیلتر های میانگین فیلترهایی هستند که با عملیات کانولوشن بر روی تصویر اعمال میشوند و تصویر را smooth تر میکنند . دو بار اعمال یک فیلتر 3*3 با اعمال یک فیلتر میانگین 5*5 بر روی تصویر برابر نیست زیرا که هر بار اعمال فیلتر 3*3 یکبار میانگین 9 پیکسل را میگیرد و اگر بار دیگر آن را اعمال کنیم تصویر قبلی که smooth شده بار دیگر smooth میشود اما اعمال فیلتر 5*5 برای هر پیکسل 25 پیکسل داخل کرنل را میانگین گیری میکند و برای اینکه نتیجه اخر اعمال فیلتر 5*5 با اعمال دو بار فیلتر 3*3 یکی شود دو راه پیش رو داریم .

راه اول :

از آنجا که میخواهیم تصویر حاصل از هر دو فیلتر به ما پیکسل های یکسانی را بدهد ابتدا فیلتر 3*3 را دو بار بر روی تصویر اعمال میکنیم . اگر تصویر حاصل را image_result بنامیم . یک ماتریس 5*5 به دلخواه از تصویر انتخاب میکنیم، اگر برای هر پیکسل از این ماتریس 5*5 ما فیلتر میانگینی با کرنل 5*5 که همه ی 25 درایه آن مجهول هستند اعمال کنیم باید به ماتریس متناظر در image_result برسیم . از این رو ما 25 مجهول داریم که وزن های ما در فیلتر 5*5 هستند و با یک دستگاه 25 معادله و 25 مجهول باید به ذنبال جواب این ماتریس باشیم.

احتمال جواب داشتن چنین دستگاهی کم است و دقت آن نیز پایین است (از آنجا که به ماتریس 5*5 انتخابی از تصویر اصلی وابسته است و این ماتریس ممکن است همه ی درایه های آن 0 باشند و معادله بینهایت جواب داشته باشد) .

پس احتمالا این روش تنها در موارد محدودی به ما جواب بدهد و اگر هم جواب بدهد ما از فیلتر بدست آمده اطمینان نداریم و همچنین بار محاسباتی زیادی را دارد .

روش دوم :

فیلتر 3*3 را دوبار اعمال میکنیم و تصویر بدست امده را result1 مینامیم . فیلتر 5*5 را نیز در تصویر اصلی اعمال میکنیم و تصویر حاصل را result2 مینامیم .

این دو تصویر را از هم کم میکنیم تا ببینیم این فیلترها در چه چیزی با هم تفاوت دارند .

Result3= result2 – result1

ما باید مقدار وزن ها در فیلتر 5*5 را طوری تغییر دهیم که حاصل result3 ماتریسی از 0 ها بشود.

3-بحث و نتایج

نتایج قسمت 2.1 :

تصویر اصلی که برای قسمت اول سوال 2 استفاده شده "child" میباشد .

تصویر 1 - تصویر اصلی

ابتدا بر روی تصویر box filter اعمال شده .

تصویر 2 - اعمال box filter

اعمال این فیلتر 10.5 ثانیه زمان برد .

```
import time
start=time.time()
dst=box_filter(img,3)
end=time.time()
time_elapsed_boxfilter=end - start
print("time elapsed is :" + str(time_elapsed_boxfilter))
```

```
time elapsed is :10.540462493896484
```

سپس فیلتر بهینه شده (optimized box filter) اعمال شد . در این فیلتر یک بار کل تصویر به صورت افقی با کرنل 1*3 که به صورت [1 1 1] است میانگین گیری انجام میشود و سپس کل تصویر به صورت عمودی با کرنل 3*1 که ترانهاده ای از کرنل بالاست ، میانگین گیری انجام میشود.

تصویر 3 - اعمال فیلتر optimized box filter

در این روش زمان اجرا به 9.3 ثانیه کاهش داشت .

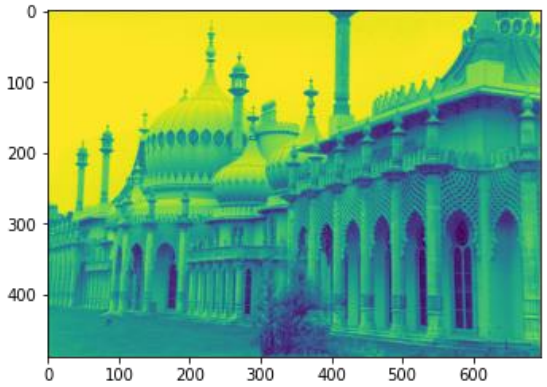
```
[12] import time
     start=time.time()
     dst2=optimized_boxfilter(img)
     end=time.time()
     time_elapsed_boxfilter=end - start
     print("time elapsed for optimized box filter is :" + str(time_elapsed_boxfilter))

     (1030, 830)
     time elapsed for optimized box filter is :9.39016342163086
```

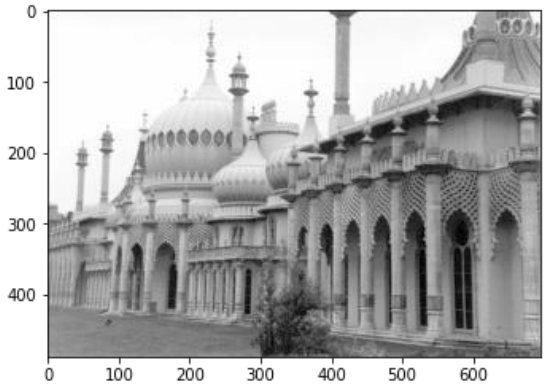
تصویر 4- زمان اجرای تابع بعد از بهینه سازی

همانطور که در بالا هم گفته شد این زمان را با روش های مرسوم dp میتوان به طور چشم گیری کاهش داد .

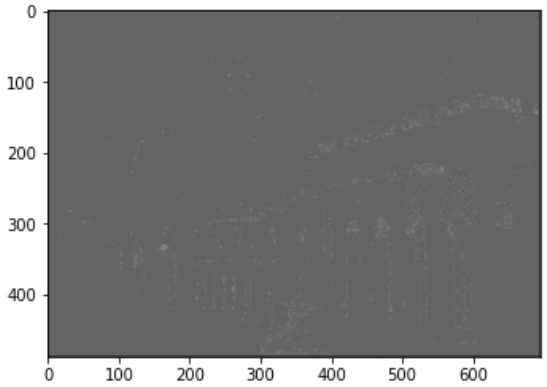
نتایج قسمت 2.2 :



تصویر 5-تصویر اصلی



تصویر 6-تصویر grayscale شده

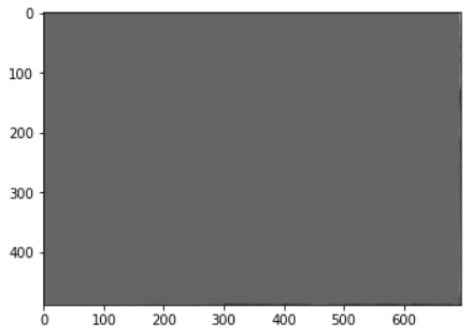


تصویر 7-تصویر بدست امده پس از اعمال فیلتر و thresholding

همانطور که مشاهده میشود ستون ها و لبه سقف قابل تشخیص است . همچنین با تغییر threshold میتوان این لبه هارا بهتر نمایش داد .

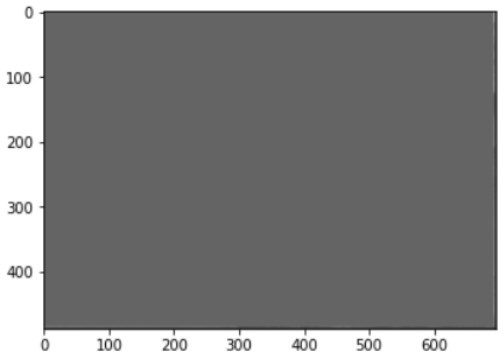
```
[19] filtered_3x3= box_filter(img,3)
```

```
[22] robert_operation(filtered_3x3)
```



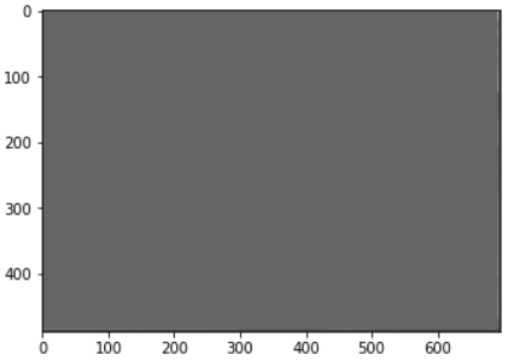
تصویر 8- لبه های تصویر پس از استفاده از فیلتر 3*3

```
filtered_5x5= box_filter(img,5)
robert_operation(filtered_5x5)
```



تصویر 9-لبه های تصویر پس از استفاده از 5*5

```
filtered_7x7= box_filter(img,7)
robert_operation(filtered_7x7)
```



تصویر 10-لبه های تصویر پس از استفاده از 7*7

نتایج قسمت 2.3 و 2.4 :

پس از اعمال 2 بار فیلتر 3*3 و یک بار فیلتر 5*5 به تصویر زیر میرسیم :

تصویر 11- اختلاف دو فیلتر

انتظار میرفت که اگر دو فیلتر باهم برابر بودند تصویر کامل سیاه میشد اما الان لبه ها ی تصویر مقدار دارند پس باید فیلتری طراحی کنیم که لبه ها را smooth کند و مقدار آن ها را کاهش دهد در حالی که بقیه تصویر دست نخورده باقی بماند .

برای اینکار دوبار فیلتر 3*3 را در هم ضرب ماتریسی میکنیم تا به کرنل زیر برسیم .

```
[1 2 3 2 1]
[2 4 6 4 2]
[3 6 9 6 3] * 1/81
[2 4 6 4 2]
[1 2 3 2 1]
```

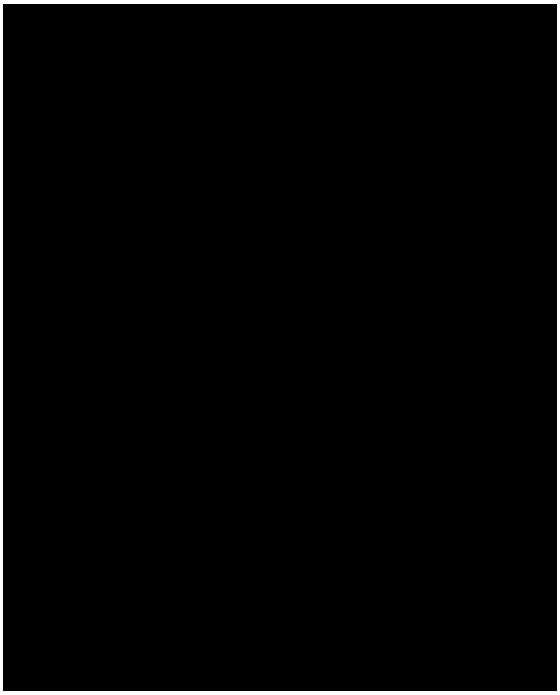
تصویر 12 -کرنل 5*5

هر خانه از این کرنل نمایانگر میزان مشارکت آن در یک فیلتر 3*3 است . برای مثال درایه اول فقط یکبار موقع پیمایش ظاهر میشود پس تنها عدد 1 را میگیرد اما درایه وسط مقدار 9 را میگیرد چون هر 9 بار در محاسبات میاید .

تصویری که با اعمال این کرنل بر تصویر اصلی بدست میاید دقیقا برابر است با تصویری که با اعمال دوبار کرنل 3*3 بدست میاید .

تصویر 13- استفاده از کرنل 3*3

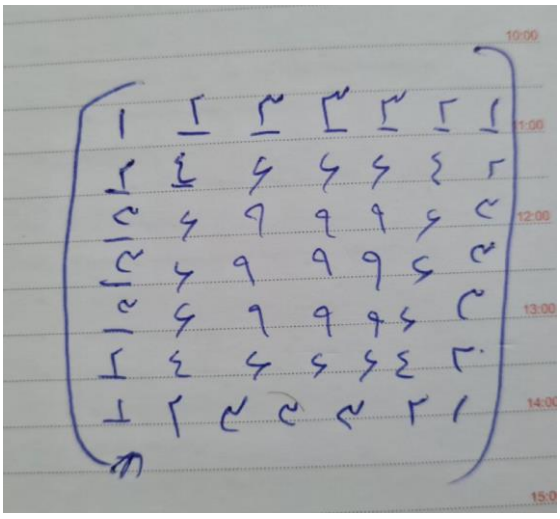
تصویر 14- استفاده از کرنل 5*5



تصویر 15- اختلاف دو ماتریس

همان طور که مشاهده میشود هر دو تصویر بدست آمده باهم برابرند .

برای کرنل 7*7 که با سه بار کانولوشن کرنل 3*3 برابر باشد باید مثل نتیجه بالا عمل کنیم و تعداد ظاهر شدن یک درایه هنگام پیمایش در فیلتر 3*3 را بنویسیم. عدد بدست آمده طبیعتا بین 1 تا 9 است .
سپس این فیلتر را اعمال میکنیم روی تصویر و تصویر بدست آمده را تقسیم بر 729 میکنیم .



4- پیوست (کد برنامه)

```
def box_filter(src, kernel_size):

    height, width = src.shape

    pad_size = kernel_size // 2
    src_padded = pad(src, pad_size)
    height_padded, width_padded = src_padded.shape
    src_padded[:, :] = 0
    src_padded[pad_size : height + pad_size, pad_size : width + pad_size] = src

    dst = np.zeros_like(src_padded, dtype=float)

    for i in range(height_padded):
        for j in range(width_padded):
            if i < height_padded - kernel_size and j < width_padded - kernel_size:
                dst[i + 1, j + 1] = np.mean(src_padded[i: i + kernel_size, j: j + kernel_size])

    return dst[pad_size : height + pad_size, pad_size : width + pad_size]
```

تصویر -16 باکس فیلتر

```
[7] def optimized_boxfilter(src):
      height, width = src.shape
      mask1=np.ones([1, 3], dtype = int)/3
      mask2=np.ones([3,1], dtype = int)/3
      pad_size = 1
      src_padded = pad(src, pad_size)
      height_padded, width_padded = src_padded.shape
      src_padded[:, :] = 0
      src_padded[pad_size : height + pad_size, pad_size : width + pad_size] = src
      print(src_padded.shape)
      dst = np.zeros_like(src_padded, dtype=float)
      for i in range(1,height):
        for j in range(1,width):
          dst[i,j]=np.mean(src_padded[i,j-1:j+1])


      return dst[pad_size : height + pad_size, pad_size : width + pad_size]
```

تصویر 17 بهینه شده باکس فیلتر

```
def box_filter_equivalent(src,kernel_size):
    height, width = src.shape

    pad_size = kernel_size // 2
    src_padded = pad(src, pad_size)
    height_padded, width_padded = src_padded.shape
    src_padded[:, :] = 0
    src_padded[pad_size : height + pad_size, pad_size : width + pad_size] = src

    dst = np.zeros_like(src_padded, dtype=float)
    kernel=np.array([[1,2,3,2,1],[2 ,4 ,6,4,2],[3 ,6 ,9, 6, 3],[2,4,6,4,2],[1 ,2, 3 ,2 ,1]])
   #kernel=np.ones((5,5),dtype=int)
    for i in range(height_padded):
        for j in range(width_padded):
            if i < height_padded - kernel_size and j < width_padded - kernel_size:
                dst[i + 1, j + 1] = np.mean(np.multiply(src_padded[i: i + kernel_size, j: j + kernel_size],kernel))

    return dst[pad_size : height + pad_size, pad_size : width + pad_size]
```

```
[122] img=cv2.imread('child.jpg',0)
      res=box_filter_equivalent(img,5)
```

```
[123] imshow(res*25/81)
```

تصویر 18- معادل دو فیلتر 3*3 برای تغییر به 7*7 فقط کافیست کرنل عوض شود

```
def box_filter_equivalent(src,kernel_size):
    height, width = src.shape

    pad_size = kernel_size // 2
    src_padded = pad(src, pad_size)
    height_padded, width_padded = src_padded.shape
    src_padded[:, :] = 0
    src_padded[pad_size : height + pad_size, pad_size : width + pad_size] = src

    dst = np.zeros_like(src_padded, dtype=float)
    kernel=np.array([[1,2,3,2,1],[2 ,4 ,6,4,2],[3 ,6 ,9, 6, 3],[2,4,6,4,2],[1 ,2, 3 ,2 ,1]])
   #kernel=np.ones((5,5),dtype=int)
    for i in range(height_padded):
        for j in range(width_padded):
            if i < height_padded - kernel_size and j < width_padded - kernel_size:
                dst[i + 1, j + 1] = np.mean(np.multiply(src_padded[i: i + kernel_size, j: j + kernel_size],kernel))

    return dst[pad_size : height + pad_size, pad_size : width + pad_size]
```

تصویر 19- معادل سازی فیلتر 5*5 بجای دو فیلتر 3*3

```
def robert_operation(img):
  filtered_image = np.zeros(img.shape)
  # Robert Operator Mask
  Mx = [[1,0],[0,-1]]
  My = [[0,1], [-1,0]]
  dm1_length = img.shape[0]
  dm2_lenght = img.shape[1]
  thresholdValue = 100
  for i in range(1,dm1_length - 1):
    for j in range(1, dm2_lenght - 1):
      Gx = sum(sum(np.multiply(Mx,img[i:i+2, j:j+2])))
      Gy = sum(sum(np.multiply(My,img[i:i+2, j:j+2])))
      point = max(math.sqrt(Gx**2 + Gy**2),thresholdValue)
      filtered_image[i,j] = point
      arr = np.asarray(filtered_image)
  plt.imshow(arr, cmap='gray', vmin=0, vmax=255)
  plt.show()
```

تصوير 20- robert operation

مراجع

R. Boyle and R. Thomas *Computer Vision: A First Course*, Blackwell Scientific Publications, 1988, pp 32 - 34.

E. Davies *Machine Vision: Theory, Algorithms and Practicalities*, Academic Press, 1990, Chap. 3.

D. Vernon *Machine Vision*, Prentice-Hall, 1991, Chap. 4.

گزارش تمارین

سروش باقرنژاد

اطلاعات گزارش

تاریخ:

واژگان کلیدی:
فیلتر میانگین
فیلتر باکس
Un-sharp masking
Sharp masking
بهبود جزییات تصویر

چکیده

هیستوگرام یک تصویر به ما نشان میدهد در یک سطح خاکستری خاص چه نعداد پیکسل داریم . هیستوگرام ها اطلاعات 2 بعدی را تبدیل به یک بعد میکنند و این عمل غیر قابل بازگشت است . از هیستوگرام ها در زمینه های مختلفی از جمله افزایش یا کاهش کنتراست تصویر کاربرد دارند .

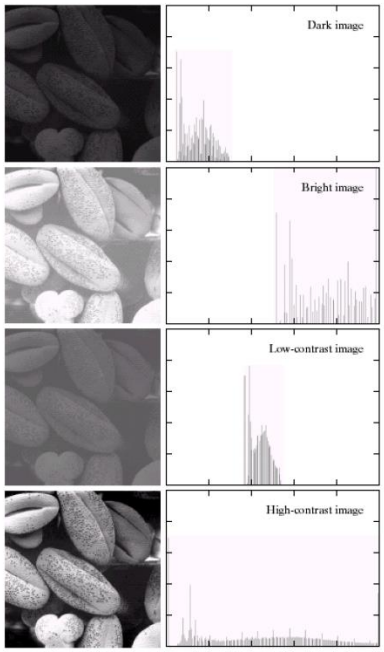
1-مقدمه

در سوال 3 از ما خواسته شده تا روی تصاویر he1,he2,he3,he4 همسان سازی هیستوگرام را انجام دهیم و در قسمت های بعدی همسان سازی محلی خواسته شده است .

2-شرح تکنیکال

هیستوگرام نمایانگر تعداد تکرار هر سطح خاکستری در تصویر دیجیتال است و ابزاریست برای تبدیل اطلاعات 2 بعدی تصویر و نمایش آن در یک بعد .
هدف از همسان سازی هیستوگرام ها ، بهبود کنتراست تصویر است . در همسان سازی هیستوگرام، سطح هایی با کنتراست کمتر ، مقدار بیشتری بگیرند .

تصویر زیر هیستوگرام تصاویر مختلف را نشان میدهد .



تصویر 1- هیستوگرام تصاویر مختلف

در یک تصویر با کیفیت و کنتراست بالا ، پیکسل ها در همه ی سطوح خاکستری به خوبی پخش شده اند اما در مقابل در یک تصویر با کنتراست پایین مانند تصویر سوم، پیکسل ها در قسمت کوچکی از سطوح خاکستری پخش میشوند . برای انجام همسان سازی هیستوگرام مراحل زیر را طی میکنیم :

ابتدا تعداد بیت های لازم برای نمایش بالاترین مقدار پیکسل در تصویر را پیدا میکنیم . برای مثال اگر بیشترین پیکسل ما مقدار 200 را داشته باشد ما به 8 بیت نیاز داریم سپس ما پراکندگی پیکسل ها را در دو به توان 8 پیکسل حساب میکنیم . پس از بدست اوردن پراکندگی ها ما به PDF نیاز داریم که مقدار آن برابر است با تقسیم مقدار پیکسل هر سطح خاکستری بر کل پیکسل ها . سپس باید مقدار CDF(cumulative distribution) را حساب کنیم. مقدار CDF به صورت زیر حساب میشود :

$$cdf_0 = pdf_0$$

و

$$cdf_i = pdf_i + cdf_{i-1}$$

سپس مقادیر بدست امده cdf را در دو به توان تعداد بیت منهای یک ضرب میکنیم . در مثال ما در 255 ضرب میشود و در مرحله اخر اعداد بدست آمده به سمت بالا گرد میشوند و مقادیر بدست امده را بجای مقادیر قبلی میگذاریم.

برای درک بهتر به جدول زیر نگاه کنید .

Discrete CDF of input histogram

Input	CDF input	CDF desired	New Value
0	1 (1.33)	1 (1.05)	3
1	3 (3.08)	2 (2.45)	4
2	5 (4.55)	5 (4.55)	5
3	6 (5.67)	6 (5.95)	6
4	6 (6.23)	6 (5.95)	6
5	7 (6.65)	7 (7.00)	7
6	7 (6.86)	7 (7.00)	7
7	7 (7.00)	7 (7.00)	7

تصویر 2- همسان سازی هیستوگرام

در جدول بالا ستون new value در واقع خروجی تابع ما میشود و هر پیکسل که ورودی تابع ماست با توجه به مقادیر به دست آمده map میشوند .

در قسمت دوم سوال خواسته شده تا تصویر بهبود یافته به کمک همسان سازی هیستوگرام را با فرمول داده شده به ازای مقادیر مختلف الفا بدست بیاریم .

$$g = \alpha.f + (1-\alpha)f_{he}$$

نتایج این قسمت و پیاده سازی آن در پایین پیوست شده است.

3-بحث و نتایج

تصویر 3- تصویر اصلی he1

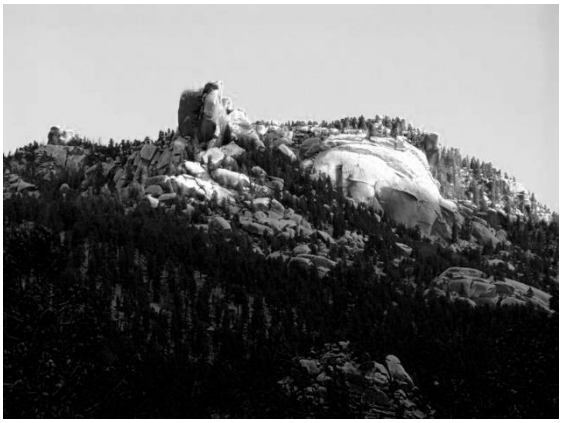
تصویر 4- تصویر he1 پس از همسان سازی

تصویر 5-تصویر اصلی he2

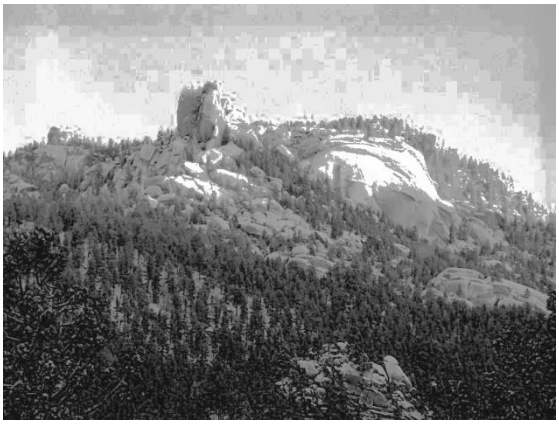
تصویر 6- تصویر he2 پس از همسان سازی

تصویر 7- تصویر اصلی he3

تصویر 8- تصویر he3 پس از همسان سازی



تصویر 9- تصویر اصلی he4



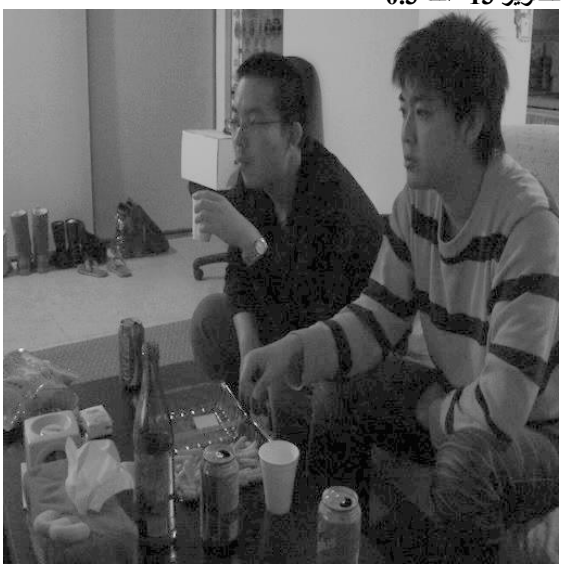
تصویر 10-تصویر he4 پس از همسان سازی

همانطور که مشاهده میشود، تصاویر تیره و dark که فقط چند سطح grayscale در نزدیک 0 دارند پس از همسان سازی هیستوگرام به تصویر هایی با کنتراست بالا تبدیل میشوند که پیکسل ها در سطوح خاکستری مختلف پخش شده اند و تصویر با کیفیت تر شده است .

تصویر 11- آلفا 0.1

تصویر 12 - آلفا 0.2

تصوير 13- آلفا 0.3



تصوير 14- آلفا 0.4

تصوير 15-آلفا 0.5

تصوير 16- آلفا 0.1

تصوير 17 - آلفا 0.2

تصوير 18 - آلفا 0.3

تصوير 19- آلفا 0.4

تصویر 20- آلفا 0.5

همانطور که مشاهده میشود با افزایش آلفا تصویر تیره تر میشود اما نویز تصویر کاهش پیدا میکند .

4- پیوست (کد برنامه)

```
import numpy as np
import pandas as pd
from scipy.spatial import distance_matrix
import cv2
import numpy as np
import math



# Read the image
he1 = cv2.imread('he1.jpg', 0)
he2 = cv2.imread('he2.jpg', 0)
he3 = cv2.imread('he3.jpg', 0)
he4 = cv2.imread('he4.jpg', 0)
# Obtain number of rows and columns
# of the image
img=he3
m, n = img.shape
print(n)
maxElement=max(map(max, img))
print(maxElement)
```

تصویر 21-لود کردن تصاویر

```
def next_power_of_2(x):
    return 1 if x == 0 else 2**(x - 1).bit_length()

def fillInputFrequency(myList,inputFrequency):
  for i in range(len(myList)):
    for j in range(len(myList[i])):
      inputFrequency[myList[i][j]]+=1

  return inputFrequency

def frequecy_sum(myList):
  sum=0
  for i in myList:
    sum+=i
  return sum
```

تصویر 22- توابع مورد نیاز برای همسان سازی

```
nextPowerOfTwo=next_power_of_2(int(maxElement))
inputFrequency=[0] * nextPowerOfTwo
inputFrequency=fillInputFrequency(img,inputFrequency)
freqSum=frequecy_sum(inputFrequency)
inputPDF=[0]*nextPowerOfTwo
inputCDF=[0]*nextPowerOfTwo
inpudCDFMultpPower=[0]*nextPowerOfTwo
```

```
[48] inputPDF=[x/freqSum for x in inputFrequency]
```

```
[49] inputCDF[0] = inputPDF[0]
     inpudCDFMultpPower[0] = int(round(inputCDF[0] * (nextPowerOfTwo-1)))

     for i,j in enumerate(inputPDF[1:], start=1):
       inputCDF[i] = inputPDF[i] + inputCDF[i-1]
       inpudCDFMultpPower[i] = int(round(inputCDF[i] * (nextPowerOfTwo-1)))
```

تصویر 23- انجام همسان سازی با استفاده از توابع بالا

```
from google.colab.patches import  cv2_imshow as imshow
resultImg=img.copy()
for i in range(len(resultImg)):
    for j in range(len(resultImg[i])):
      resultImg[i][j] = inpudCDFMultpPower[resultImg[i][j]]

imshow(resultImg)
```

مراجع

R. Boyle and R. Thomas *Computer Vision: A First Course*, Blackwell Scientific Publications, 1988, pp 32 - 34.

E. Davies *Machine Vision: Theory, Algorithms and Practicalities*, Academic Press, 1990, Chap. 3.

D. Vernon *Machine Vision*, Prentice-Hall, 1991, Chap. 4.

گزارش تمارین

سروش باقرنژاد

اطلاعات گزارش	چکیده
تاریخ:	sharpening به عملیاتی گفته میشود که در آن به کمک روش هایی اعم از shap masking و یا un-sharp masking میتوان جزییات یک تصویر را بهبود بخشید و اصطلاحا آن تصویر را شارپ تر کرد .
واژگان کلیدی: فیلتر میانگین فیلتر باکس Un-sharp masking Sharp masking بهبود جزییات تصویر	

1-مقدمه

در سوال 4 از ما خواسته شده تا روی تصویر child فیلتر های un-sharp را پیاده سازی کنیم . این کار به سادگی با کمک توابعی که در سوال 2 نوشته شده و روابط موجود در کتاب انجام پذیر است .

2-شرح تکنیکال

شرح قسمت 4.1 :

هدف عمده فیلترهای شارپ کننده برجسته کردن جزییات در یک تصویر یا بهبود جزییاتی است که تار شده اند .
برای شارپ کردن تصویر و در نتیجه بهبود تصویر ما نیاز داریم تا از تصویر تفاصل بگیریم . این کار با استفاده از مشتق و فیلتر لاپلاسیان انجام پذیر است . در مشتق اول بله های ضخیم تر پیدا میشوند و تغییرات پله ای سطوح خاکستری تاثیرگذارند و اگر مشتق مرتبه دوم بگیریم به جزییات دقیق تصویر میرسیم ازین جهت مشتق مرتبه دوم برای تیز کردن تصویر(sharpening) مناسب تر است .
متشق مرتبه دوم یا لاپلاسین :

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

با اعمال این فیلتر بر روی تصویر اصلی و کم کردن آن میتونیم به ماسکینگ تیز دست یابیم .

$$g(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y) \\ f(x,y) + \nabla^2 f(x,y) \end{cases}$$

اما در مقابل ما روشی دیگری به نام un-sharp masking داریم که به جای استفاده از لاپلاسین، نسخه ای غیر شارپ از تصویر تهییه میکنیم و آن را از تصویر اصلی کم میکنیم و ماسک حاصل را با ضریب k به تصویر اصلی اضافه میکنیم .

$$g_{mask}(x,y) = f(x,y) - \bar{f}(x,y)$$

در معادله ی بالا f تصویر اصلی ، $\bar{f}$ تصویر غیر شارپ است که با اعمال یک فیلتر میانگین (ساده یا وزن دار) به تصویر اصلی بدست میاد، g هم ماسک غیر تیز بدست آمده میباشد.

$$g(x,y) = f(x,y) + k.g_{mask}(x,y)$$

در معادله ی بالا k ضریبی است که به ماسک غیر تیز اعمال میشود و k=1 باشد به آن ماسکینگ غیر شارپ و اگر k>1 باشد از نام فیلتر تقویت بالا استفاده میشود.

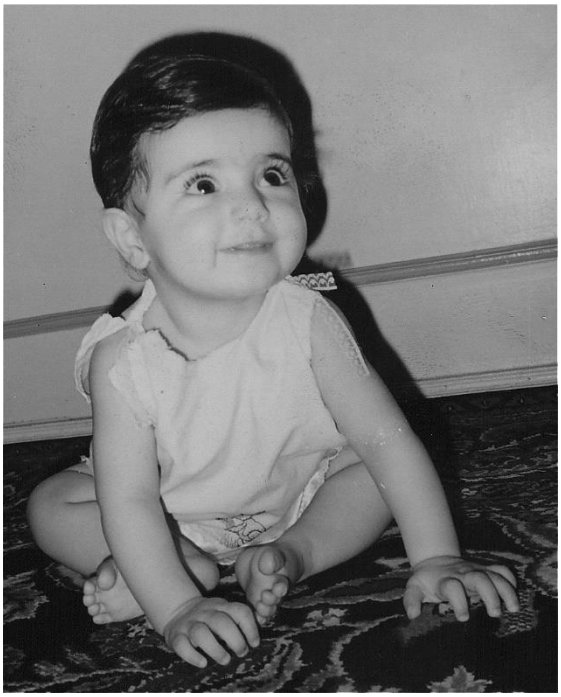
در قسمت اول سوال ما ابتدا تصویر را لود میکنیم و روش گفته شده در بالا را به تصویر اعمال میکنیم. نتیجه این بخش در قسمت نتایج پیوست شده است .

3-بحث و نتایج

تصویر اصلی که برای سوال 4 استفاده شده "child" میباشد .

تصویر 1 - تصویر اصلی

ابتدا بر روی تصویر box filter اعمال شده و از تصویر اصلی کم شده تا ماسک g بدست بیاید . سپس این ماسک غیر تیز را با k=1 از تصویر اصلی کم میکنیم تا به تصویر زیر برسیم



تصویر 2- پس از اعمال un-sharp masking

همانطور که مشاهده میشود این تصویر شارپ تر شده و از دید انسان وضوح بهتری دارد و جزییات آن مانند کنار لب و خطوط نازک صورت بهتر دیده میشوند همچنین لبه ها شارپ تر شده و اگر به کنار دست ها و یا لبه سایه کودک نگاه کنیم به راحتی برای ما از سایر اجزای تصویر قایل تفکیک است .

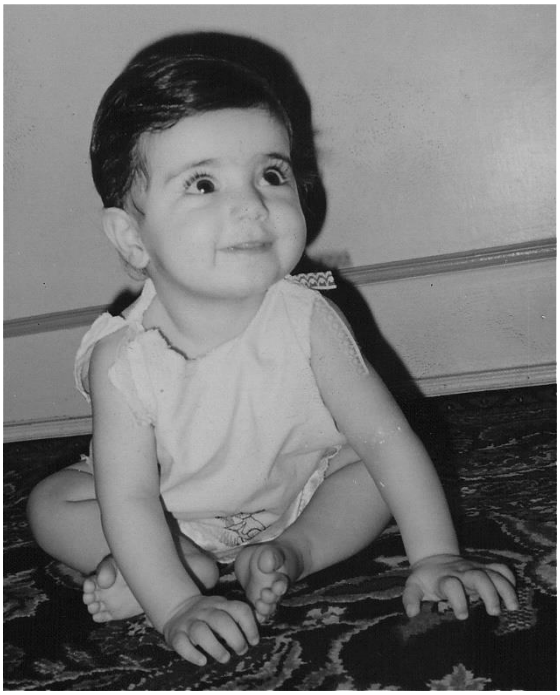
برای قسمت دوم سوال 4 ما بجای کرنل

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

که بدون وزن هست، کرنل وزن دار زیر را بر تصویر جهت میانگین گیری و Smooth کردن اعمال میکنیم .

$$\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

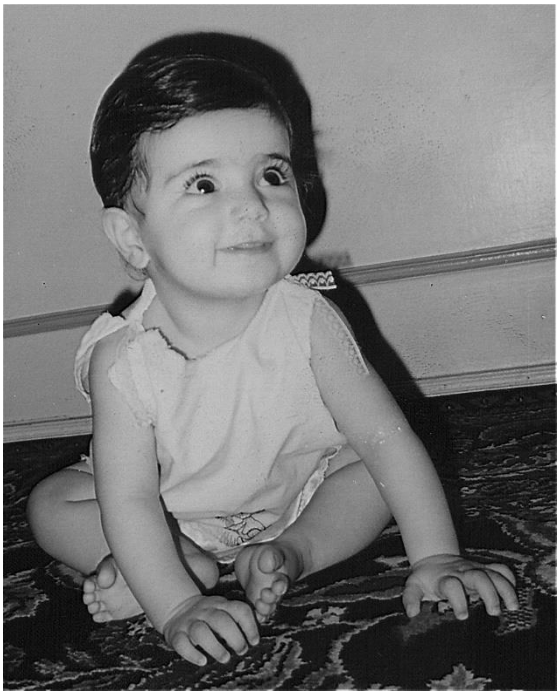
پس از اعمال این کرنل و انجام عملیات ها مشابه قسمت قبل به تصویر زیر میرسیم .



تصویر 3- un-sharp masking با کرنل وزن دار

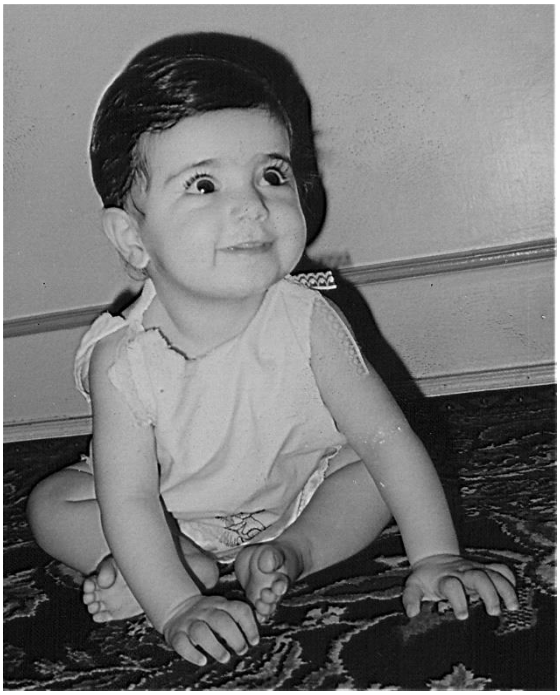
همانطور که مشاهده میشود استفاده از کرنل وزن دار تاثیر چندانی در بهبود و تیز کردن تصویر ندارد .

در قسمت سوم سوال از ما خواسته شده تا بجای کرنل 3*3 از کرنل های 5*5 ، 7*7 و 9*9 استفاده کنیم و نتیجه را مورد بحث قرار دهیم . در اینجا به ترتیب نتایج گذاشته میشود و به تغییرات هر تصویر پس از تغییر کرنل اشاره میشود.



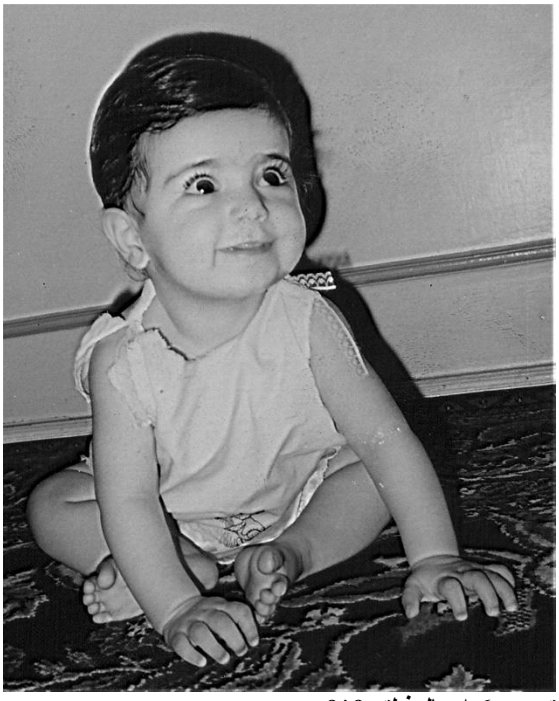
تصویر 4 - اعمال کرنل 5*5

هنگامی که کرنل 5*5 اعمال میکنیم لبه های تصویر بیشتر مشخص میشوند و تصویر شارپ تر میشود . برای مثال در قسمت دست ها کاملا کناره ی دست ها با خط سیاهی از قسمت پا ی کودک جدا شده و قابل تفکیک است .



تصویر 5-اعمال کرنل 7*7

با افزایش سایز کرنل جزییات ریز به خوبی قابل دیدن هست. در این تصویر برآمدگی گونه کودک به خوبی دیده میشود در صورتی که کرنل های کوچک تر این ویژگی را به ما نمیدادند.



تصویر 6-اعمال فیلتر 9*9

هنگامی که از کرنل 9*9 استفاده میکنیم دیگر کودک به طور کامل از پس زمینه جدا شده ست و لبه های تصویر در قسمت های روشن با نواری سیاه و در قسمت های تیره مانند مو با نواری سفید مشخص شده ولی مشکلی که ایجاد شده این است که در قسمت سایه کودک از موی آن قابل تفکیک نیست .

4- پیوست (کد برنامه)

```
import cv2
import numpy as np


# Read the image
img = cv2.imread('child.jpg', 0)
from google.colab.patches import cv2_imshow as imshow
imshow(img)
```

تصویر 7- لود کردن تصویر اصلی

```
  def pad(src, pad_size):
      return np.pad(src, ( (pad_size, pad_size), ( pad_size, pad_size) ) )
```

```
def box_filter(src, kernel_size):

    height, width = src.shape

    pad_size = kernel_size // 2
    src_padded = pad(src, pad_size)
    height_padded, width_padded = src_padded.shape
    src_padded[:, :] = 0
    src_padded[pad_size : height + pad_size, pad_size : width + pad_size] = src

    dst = np.zeros_like(src_padded, dtype=float)

    for i in range(height_padded):
        for j in range(width_padded):
            if i < height_padded - kernel_size and j < width_padded - kernel_size:
                dst[i + 1, j + 1] = np.mean(src_padded[i: i + kernel_size, j: j + kernel_size])

    return dst[pad_size : height + pad_size, pad_size : width + pad_size]
```

تصویر 8- تابع پدینگ و باکس فیلتر

```
[ ]  f=img
     f_unsharp_3x3=box_filter(img,3)
     g_mask=f-f_unsharp_3x3
     k=1
     g=f+k*g_mask
```

تصویر 9- ماسکینگ غیر تیز

```
[ ]  def weighted_average(src,kernel,kernel_size):
         height, width = src.shape

         pad_size = kernel_size // 2
         src_padded = pad(src, pad_size)
         height_padded, width_padded = src_padded.shape
         src_padded[:, :] = 0
         src_padded[pad_size : height + pad_size, pad_size : width + pad_size] = src

         dst = np.zeros_like(src_padded, dtype=float)
            #kernel=np.ones((5,5),dtype=int)
         for i in range(height_padded):
             for j in range(width_padded):
                 if i < height_padded - kernel_size and j < width_padded - kernel_size:
                     dst[i + 1, j + 1] = np.mean(np.multiply(src_padded[i: i + kernel_size, j: j + kernel_size],kernel))

         return dst[pad_size : height + pad_size, pad_size : width + pad_size]
```

تصویر 10- فیلتر میانگین با کرنل دلخواه

```
kernel=np.array([[1,2,1],[2,4,2],[1,2,1]],dtype=int)
f_unsharp_3x3_wa=weighted_average(img,kernel,3)*9/16
f2=img
g_mask2=f2-f_unsharp_3x3_wa
k=1
g2=f2+k*g_mask2
```

تصویر 11- un-sharp masking با کرنل وزن دار

```
f3=img
f_unsharp_5x5=box_filter(img,5)
g_mask3=f3-f_unsharp_5x5
k=1
g3=f3+k*g_mask3
```

```
[ ]  imshow(g3)
```

تصویر 12- unsharp-masking با کرنل 5*5

```
[ ]  f4=img
     f_unsharp_7x7=box_filter(img,7)
     g_mask4=f4-f_unsharp_7x7
     k=1
     g4=f4+k*g_mask4
```

```
imshow(g4)
```

تصویر 13- unsharp-masking با کرنل 7*7

```
f5=img
f_unsharp_9x9=box_filter(img,9)
g_mask5=f5-f_unsharp_9x9
k=1
g5=f5+k*g_mask5
```

```
imshow(g5)
```

تصویر 14- unsharp-masking با کرنل 9*9

مراجع

R. Boyle and R. Thomas *Computer Vision: A First Course*, Blackwell Scientific Publications, 1988, pp 32 - 34.

E. Davies *Machine Vision: Theory, Algorithms and Practicalities*, Academic Press, 1990, Chap. 3.

D. Vernon *Machine Vision*, Prentice-Hall, 1991, Chap. 4.