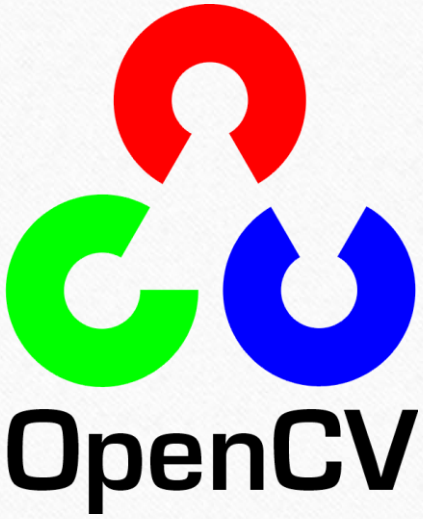
دوره آموزشی بینایی ماشین کاربردی

آکادمی ربوتک – آزمایشگاه تعامل انسان و ربات

جلسه 4 – لبه یابی+ الگوریتم های تشخیص ویژگی



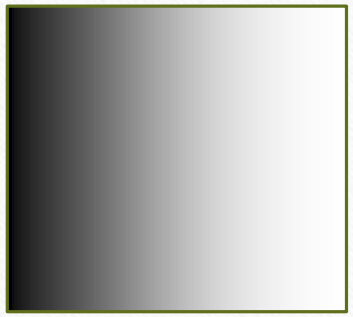
چرا لبه ها مهم هستند ؟

عملگر گرادیان (Gradient Operator)

ورودی تصویر و خروجی مشتق تصویر

این عملگر دارای دو مولفه در راستای x و y است.

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$



$$\nabla f = [\frac{\partial f}{\partial x}, 0]$$

$$\nabla f = [0, \frac{\partial f}{\partial y}]$$

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

مفهوم مشتق در یک تصویر:

مشتق پیوسته :

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon,\ y) - f(x,y)}{\varepsilon}$$

مشتق گسسته (در تصویر) :

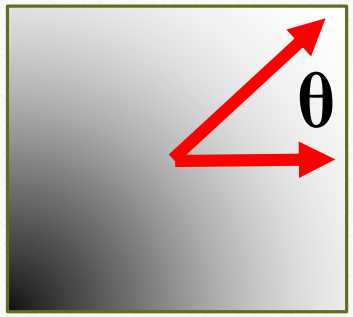
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1} \approx f(x+1,\ y) - f(x,y)$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+1) - f(x,y)}{1} \approx f(x,\ y+1) - f(x,y)$$

جهت و اندازه گرادیان

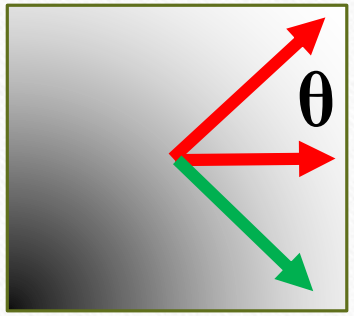
جهت گرادیان:

$$\theta = \tan^{-1}\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right)$$



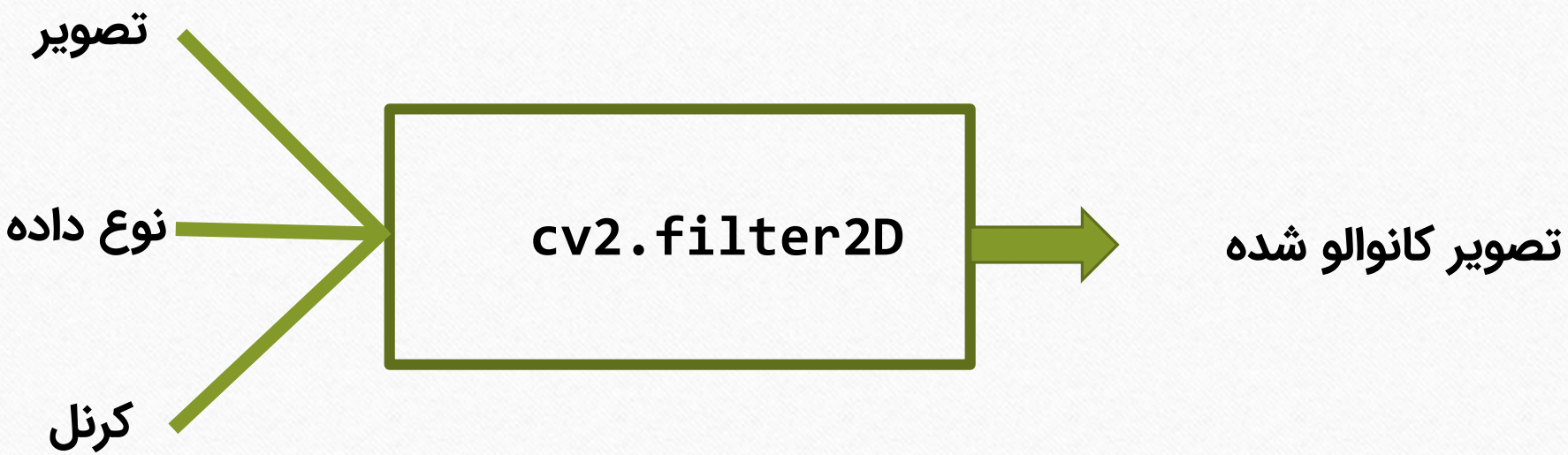
اندازه گرادیان:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



جهت لبه ⟵ عمود بر جهت گرادیان

یاد آوری : کانولوشن در OpenCV



مثال :

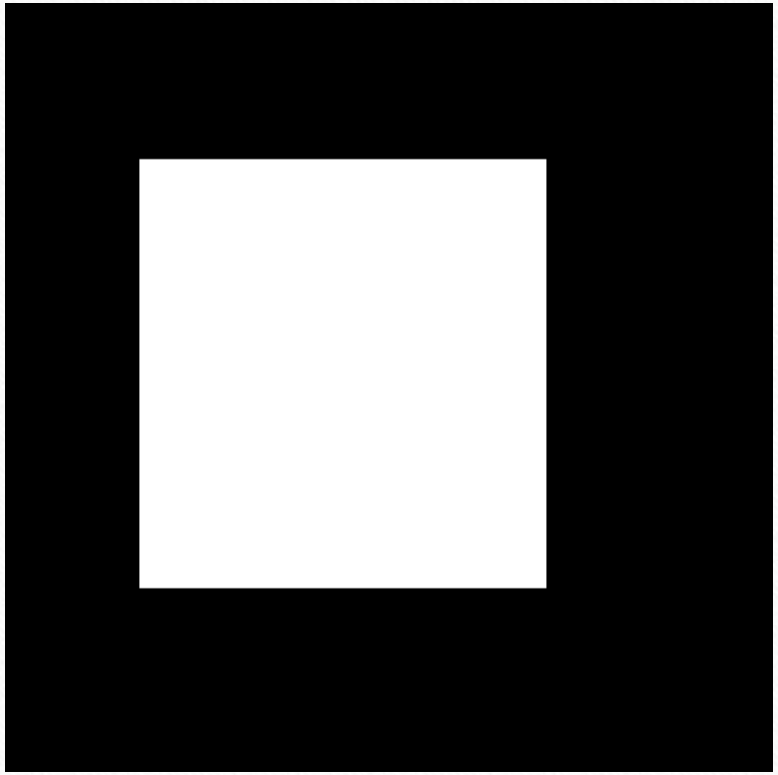
```
conv_img = cv2.filter2D(img, cv2.CV_8U, kernel)
```

- کرنل یک ماتریس است که میتوانیم آن را با کتابخانه numpy بسازیم.

حل یک مثال

Kernel = [-1, 1]

Ddepth = -1 / cv2.CV_8U / cv2.CV_64F



❑ سوال : نتیجه خروجی چه خواهد بود ؟

دستور ConvertScaleAbs:31

تصویر غیر استاندارد

cv2.convertScaleAbs

تصویر استاندارد

مثال :

```
std_img = cv2.convertScaleAbs(img)
```

لبه یابی در عمل

چگونه لبه ها را پیدا کنیم ؟

ایده : مشابه مرحله قبل یک کرنل به تصویر اعمال کنیم و نتیجه را مشاهده کنیم

عملگر Prewitt

-1	0	+1
-1	0	+1
-1	0	+1

-1	0	+1
-1	0	+1
-1	0	+1

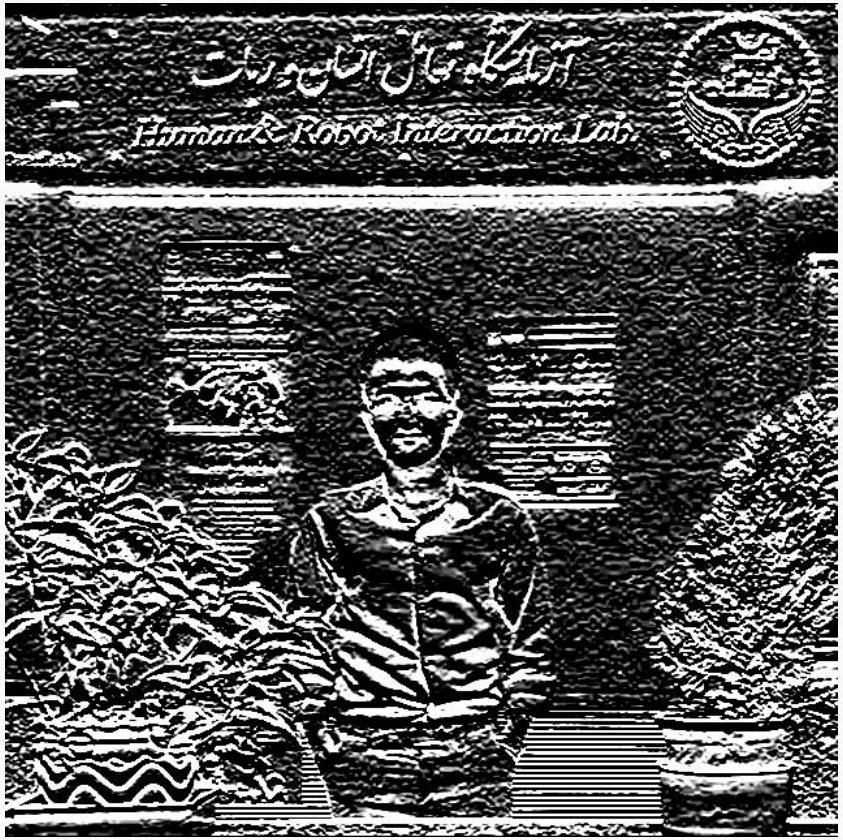
عملگر Sobel

-1	0	+1
-2	0	+2
-1	0	+1

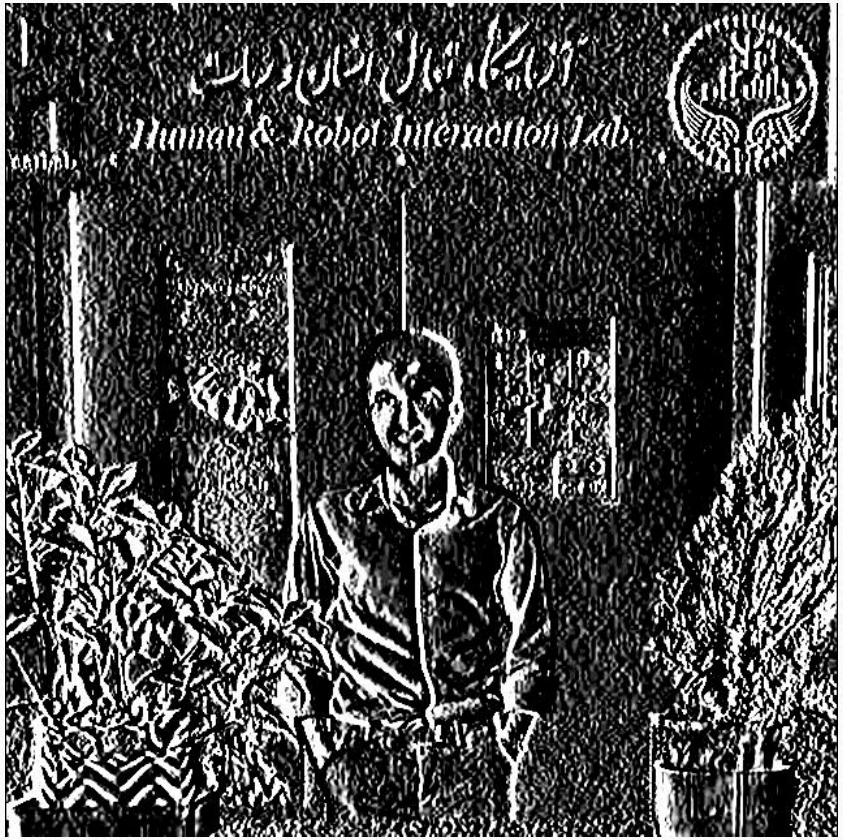
-1	-2	-1
0	0	0
+1	+2	+1

مشتق در راستای X و Y

مشتق در راستای Y



مشتق در راستای X



آشنایی با عملگر Sobel

کرنل برای مشتق در جهت X

$$\frac{1}{8}\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

کرنل برای مشتق در جهت Y

$$\frac{1}{8}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

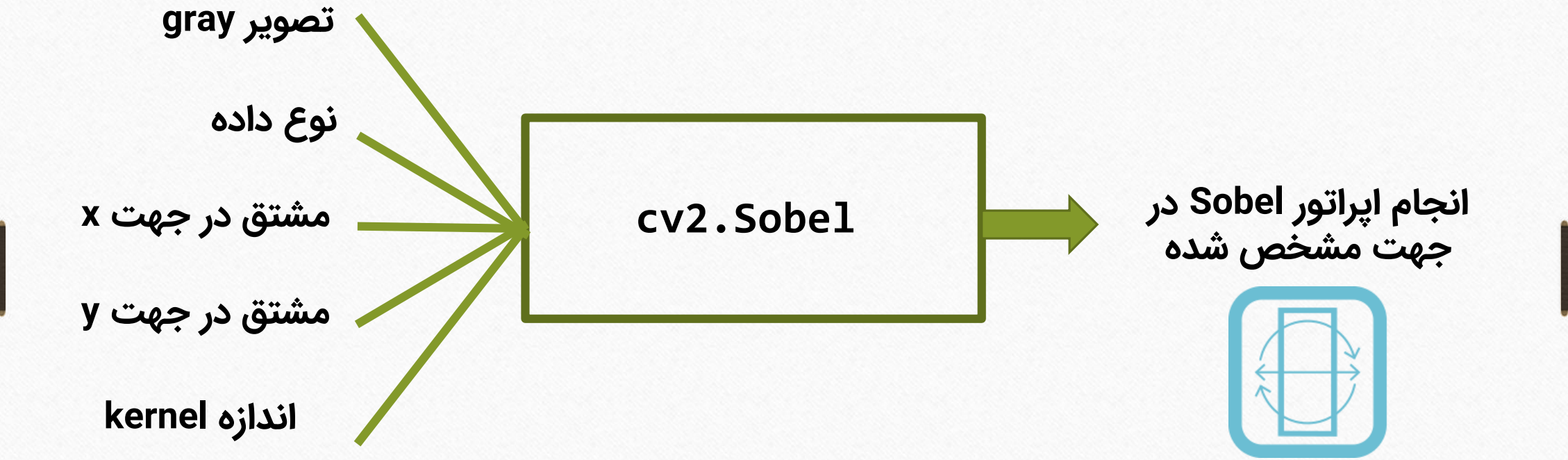
اندازه و جهت

$$\nabla I = [g_x \quad g_y]$$

$$\|\nabla f\| = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \tan^{-1}(g_x / g_y)$$

دستور Sobel : 32

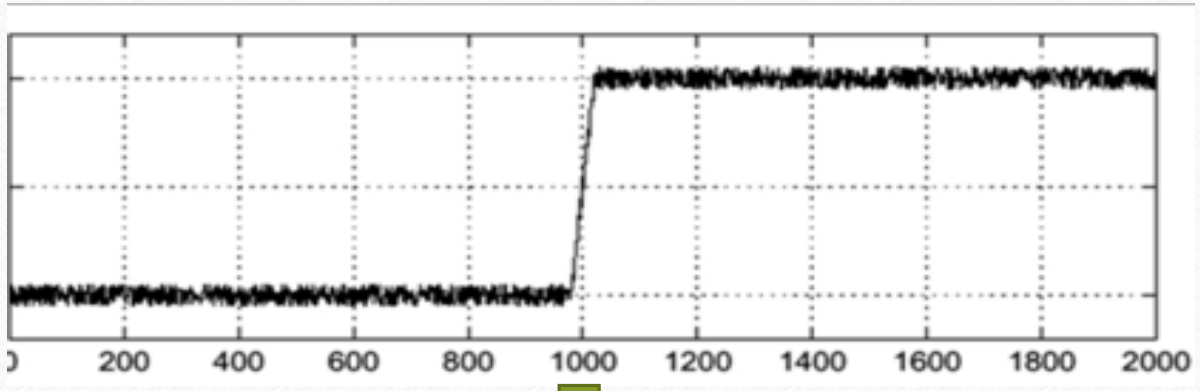


مثال :

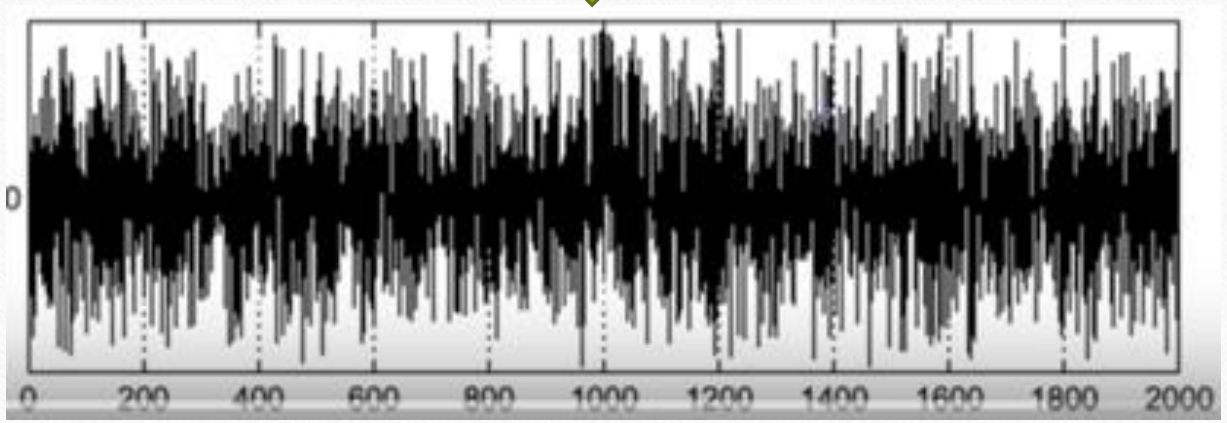
```
sobelX = cv2.Sobel(gray, cv2.CV_64F, dx=0, dy=1, ksize = 3)
```

و اما در دنیای واقعی !

$f(x)$

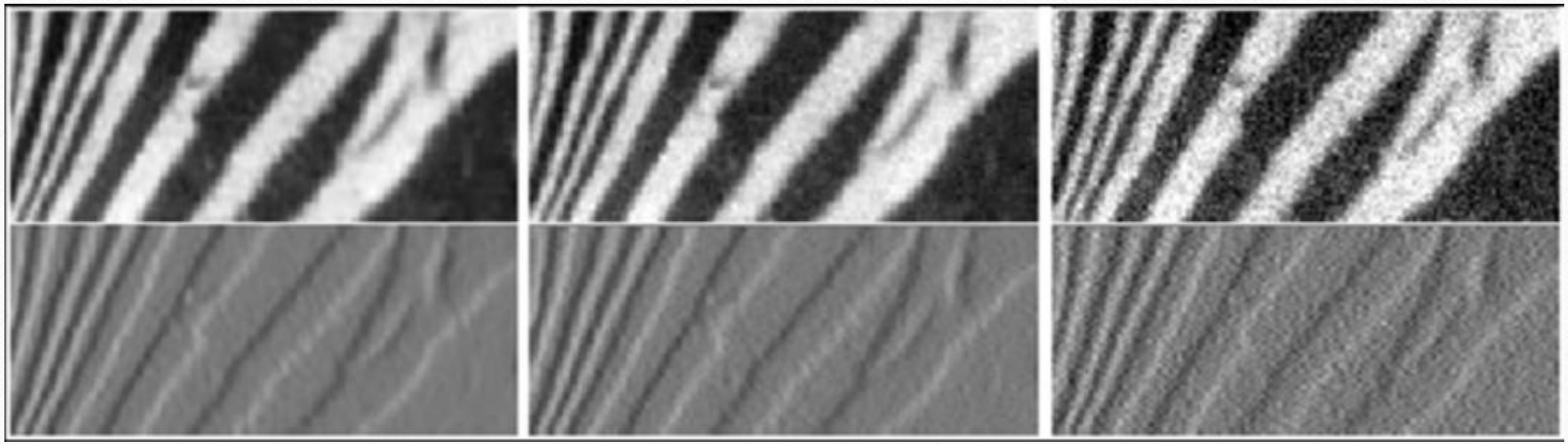


$\frac{d}{dx}(f(x))$



لبه کجاست ؟!

تاثیر حضور نویز بر لبه یابی



افزایش نویز گاوسی

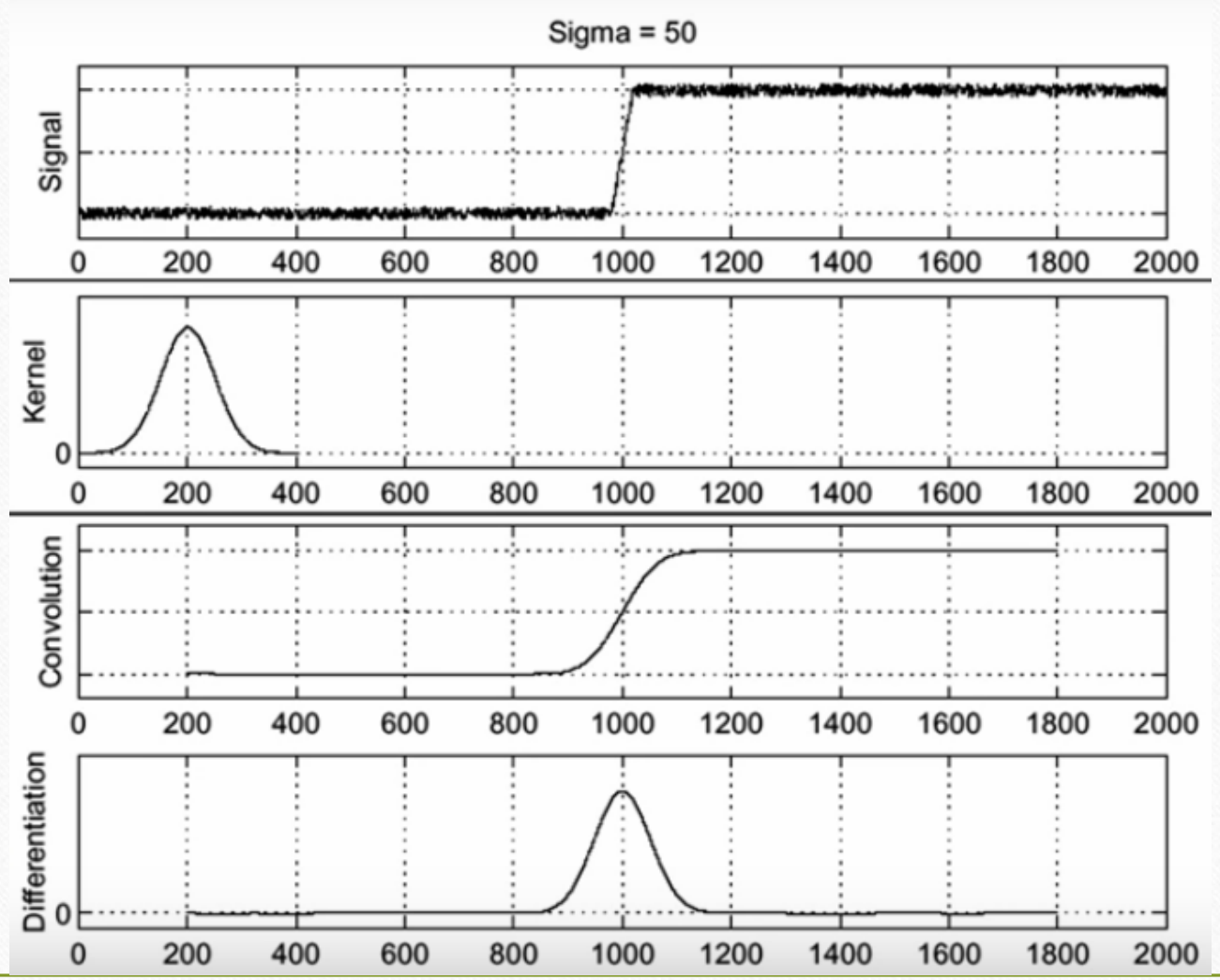
برای حل این مشکل چه کار کنیم ؟

$$f$$

$$h$$

$$f * h$$

$$\frac{\partial}{\partial x}(f * h)$$



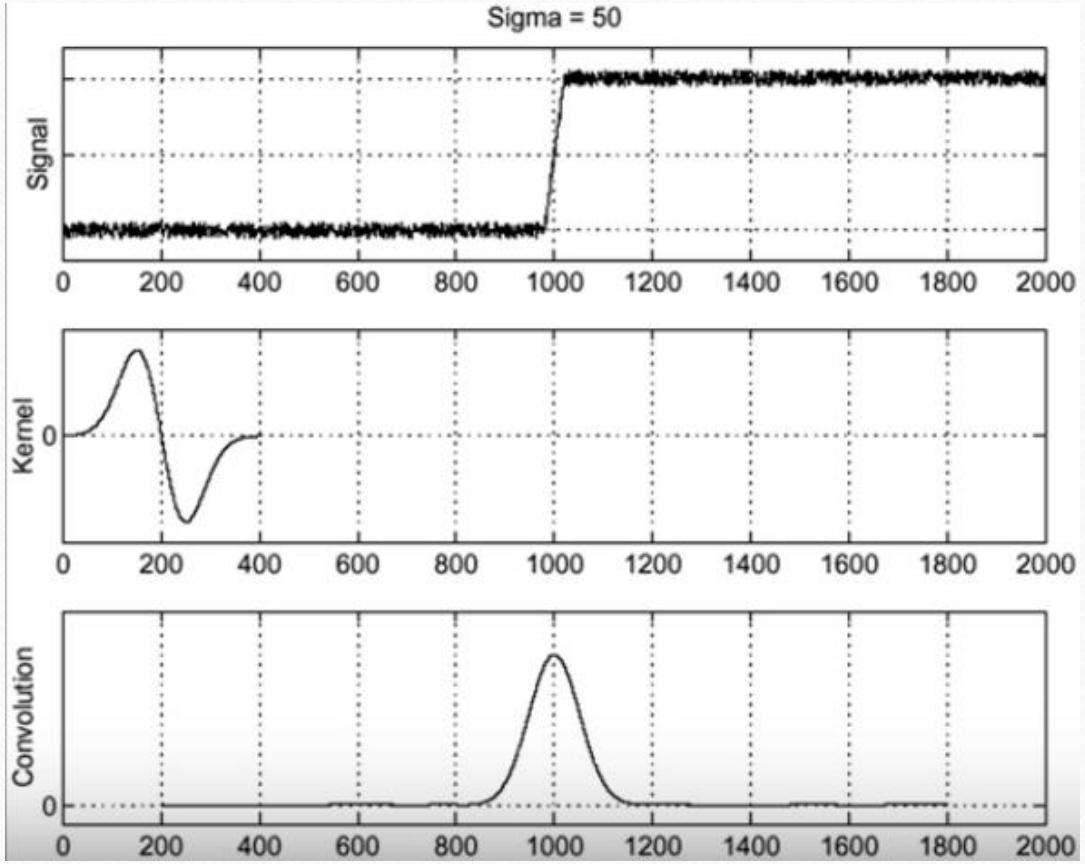
$$\frac{\partial}{\partial x}(h * f) = \frac{\partial}{\partial x}(h) * \mathbf{f}$$

میتوانیم از خواص ریاضی کانولوشن استفاده کنیم.

$$f$$

$$\frac{\partial}{\partial x}(h)$$

$$\frac{\partial}{\partial x}(f * h)$$



الگوریتم کنی برای لبه یابی

ارایه شده توسط جان کنی مهندس کامپیوتر استرالیایی

در سال 1986 و به عنوان پایان نامه ارایه شد

به صورت گسترده در بینایی ماشین ارایه می شود.

مراحل الگوریتم Canny

1. حذف نویز با فیلتر گاوسین
2. محاسبه گرادیان تصویر
3. Non Maximum Suppression
4. hysteresis thresholding

1 حذف نویز با فیلتر گاوسین

لبه یابی بسیار حساس به نویز است. به همین دلیل یک فیلتر گاوسین در ابتدا اعمال می شود.

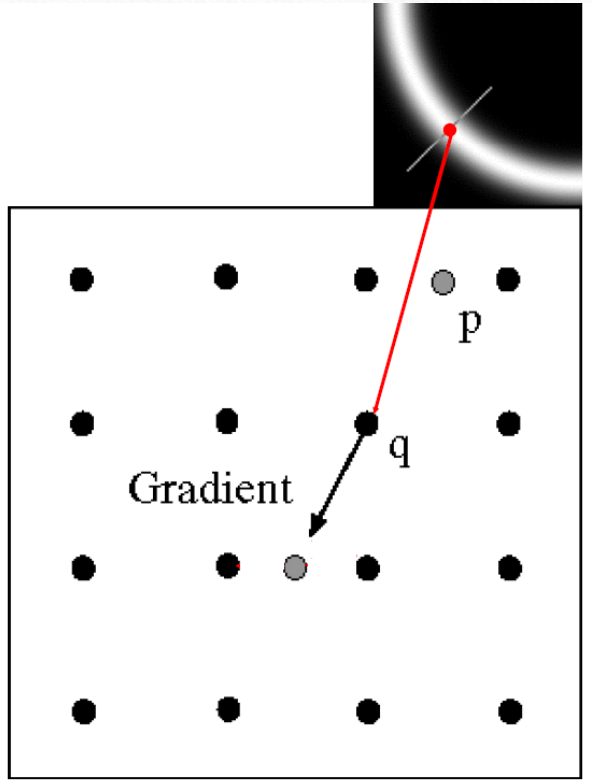
2 محاسبه گرادیان تصویر

به کمک عملگر Sobel گرادیان تصویر را در دو جهت x و y محاسبه می کنیم و magnitude را بدست می آوریم.

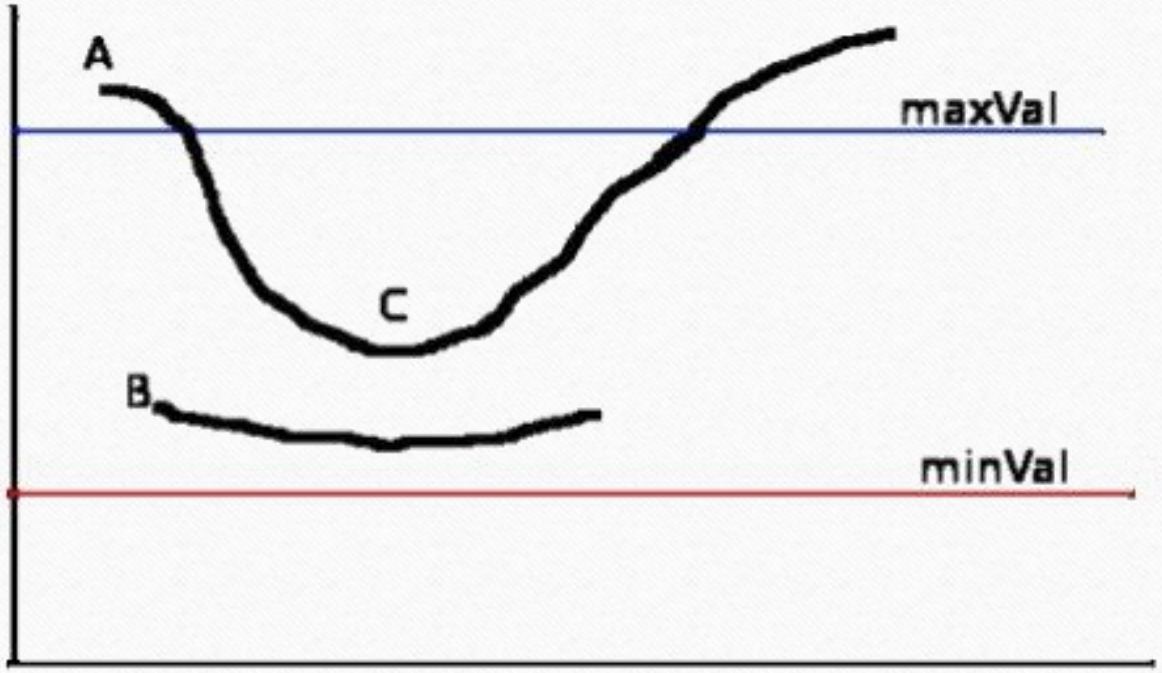
Non maximum supression

3

برای لبه ای که چند پیکسل نماینده آن هستند، باید پیکسلی با بیشترین Magnitude را نگه داشت و بقیه را حذف کرد.

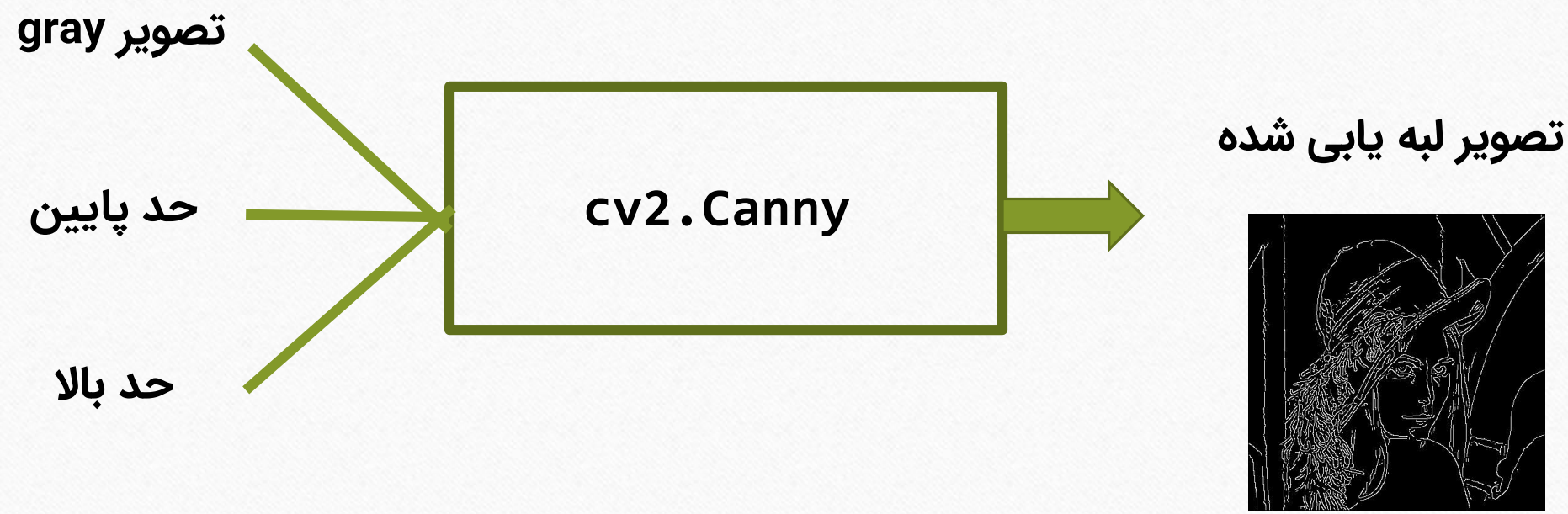


4 معرفی hysteresis thresholding



- نقطه A چون مقدار گرادیانش از مقدار maximum بیشتر است. پس لبه است.
- نقطه B مقدارش بین minVal و maxVal می باشد ولی چون به لبه ای قوی متصل نشده است ، لبه در نظر گرفته نمی شود.
- نقطه C مقدارش بین minVal و maxVal می باشد ولی چون به لبه ای قوی متصل شده است ، لبه در نظر گرفته می شود.

دستور 33 : الگوریتم Canny



مثال :

```
Canny_img = cv2.Canny(gray, 100, 200)
```

تمرین : به کمک عملگر Sobel و آنچه تا کنون آموخته اید در تصویر زیر بارکد را پیدا کنید.

الگوریتم های تشخیص ویژگی

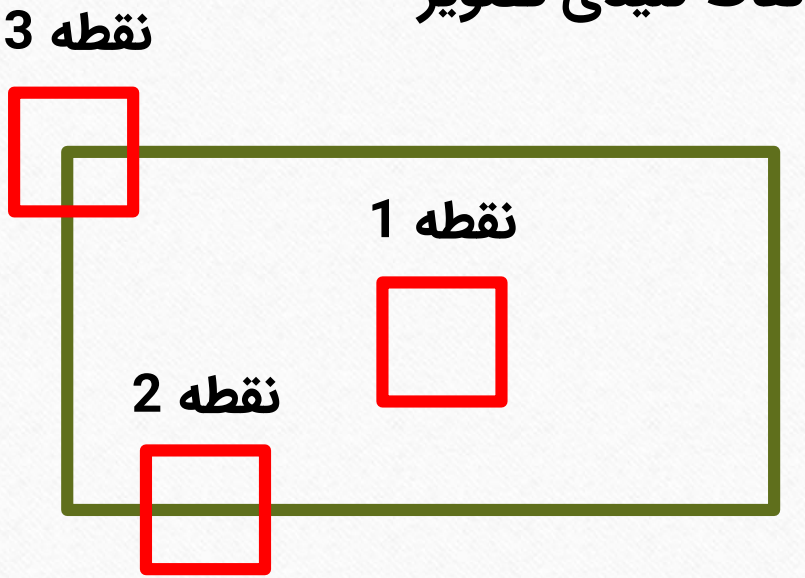
آشنایی با الگوریتم های تشخیص ویژگی

با کمک KeyPoint ها و Descriptor ها

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-3} \\ x_{n-2} \\ x_{n-1} \\ x_n \end{bmatrix} \quad \cdots \quad \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \cdot \\ \cdot \\ \cdot \\ u_{n-3} \\ u_{n-2} \\ u_{n-1} \\ u_n \end{bmatrix}$$

KeyPoint چیست ؟

- ✓ نقاط کلیدی تصویر

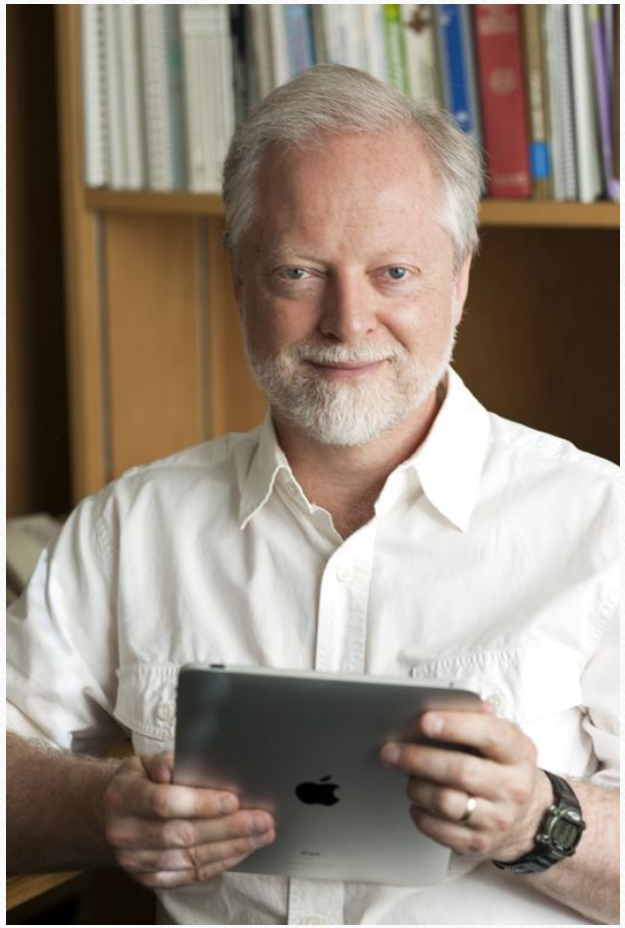


- ✓ مکان هایی که تغییر گرادیان در اطراف آنها زیاد است.

Descriptor چیست ؟

- ✓ توصیفی از نقاط کلیدی
- ✓ از گرادیان نقاط در اطراف نقط کلیدی میتوان برای ایجاد descriptor استفاده کرد.
- ✓ معمولا به صورت برداری از اعداد ارایه می شود.

آشنایی با الگوریتم SIFT

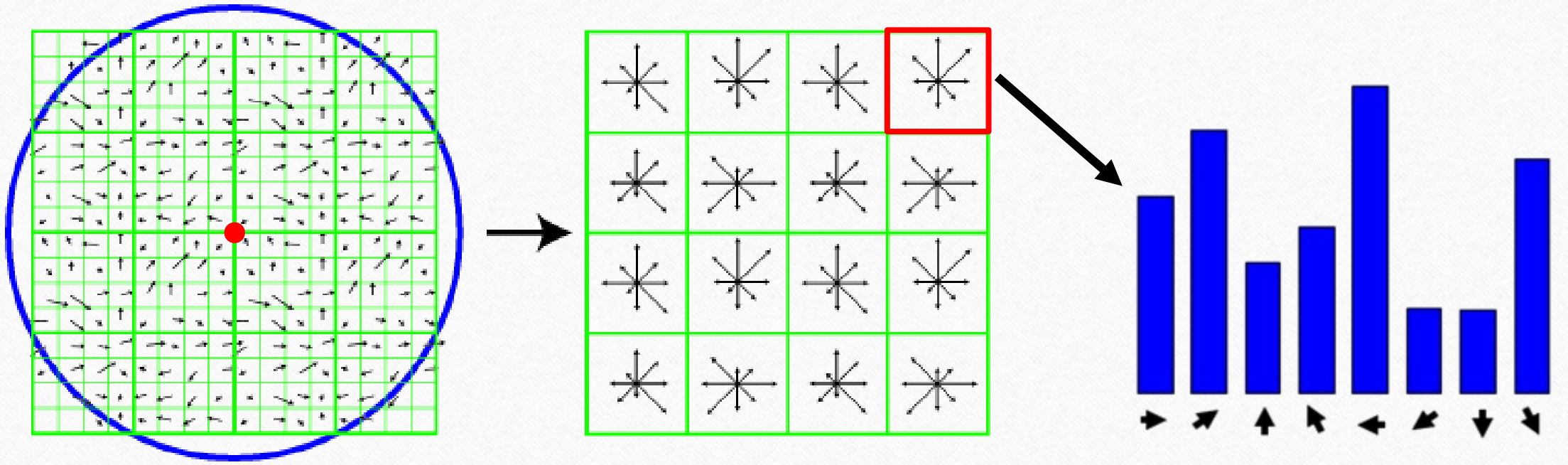


کلمه SIFT مخفف عبارت Scale Invarient Feature Transform می باشد.

توسط david lowe و در 2004 ارایه شد.

مستقل از Scale و rotation و نور و نقطه مشاهده است.

Descriptor در الگوریتم SIFT



برای تک تک بلوک ها، این نمودار استخراج می شود و در نهایت یک بردار 16 * 8 یعنی 128 تایی از اعداد تولید می شود.

دستور 32 : تعریف یک Object از SIFT

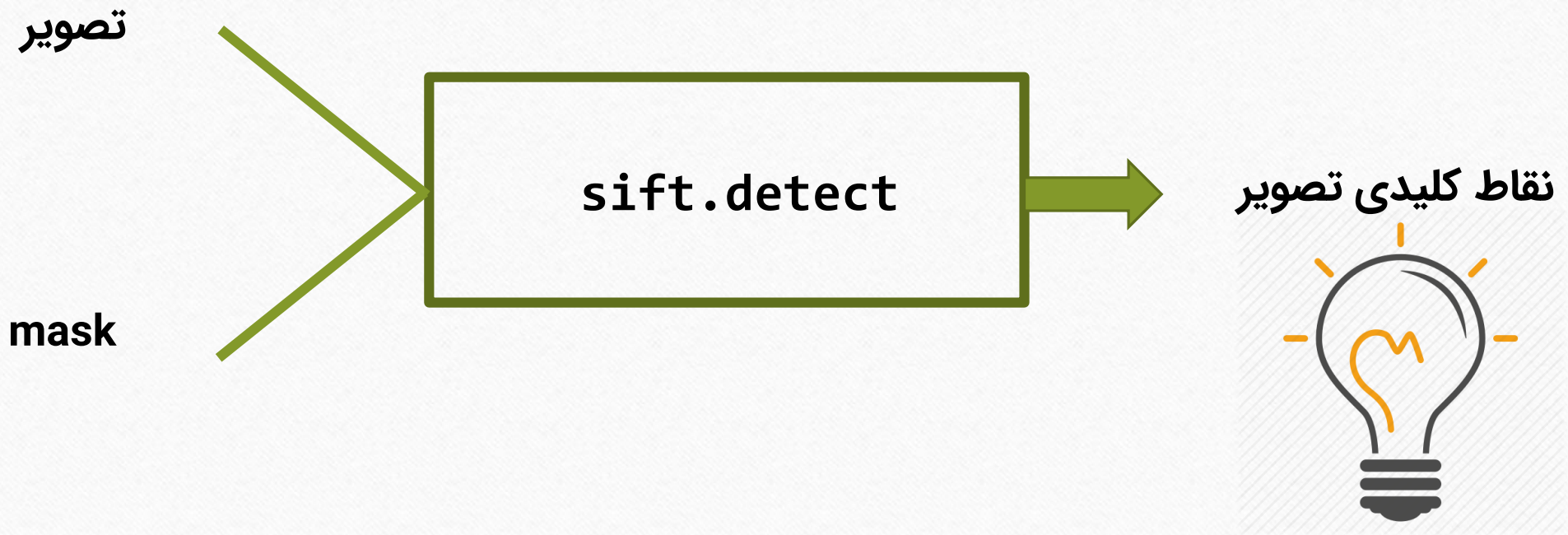
```
cv2.xfeatures2d.SIFT_create
```

یک Object از کلاس SIFT

مثال :

```
sift = cv2.xfeatures2d.SIFT_create()
```

دستور 32 : پیدا کردن KeyPoint ها



مثال :

```
kp = sift.detect(img, None)
```

دستور 32 : رسم KeyPoint ها

تصویر ورودی

KP

تصویر خروجی

flag (اختیاری)

cv2.drawKeypoints

رسم نقاط کلیدی بر روی تصویر

```
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS
```

مثال :

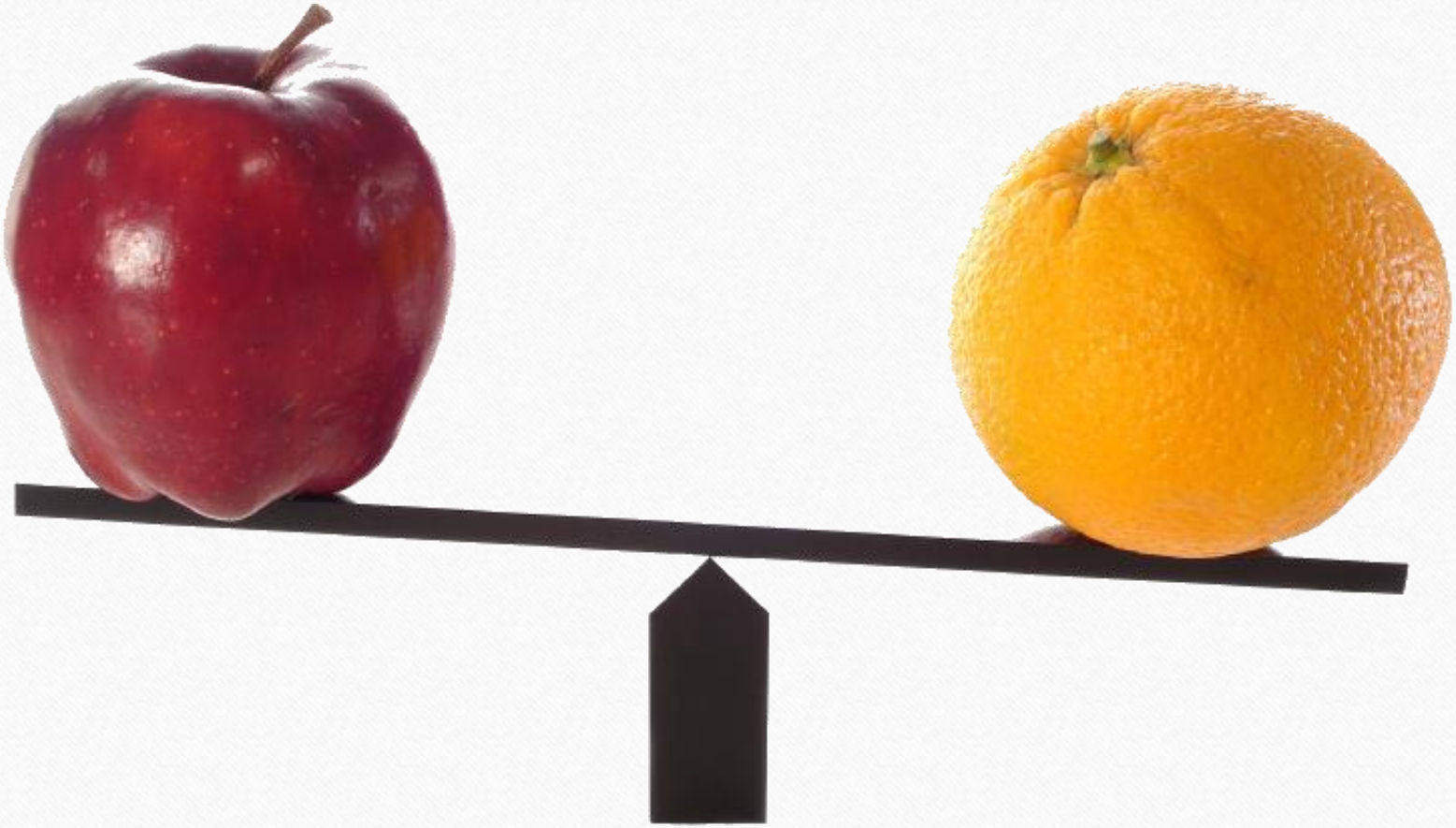
```
cv2.drawKeypoints(img, kp, img)
```

دستور 32 : پیدا کردن KeyPoint ها و descriptor ها

مثال :

```
kp, des= sift.detectAndCompute(gray, None)
```

مقایسه بین descriptor ها و match کردن آنها



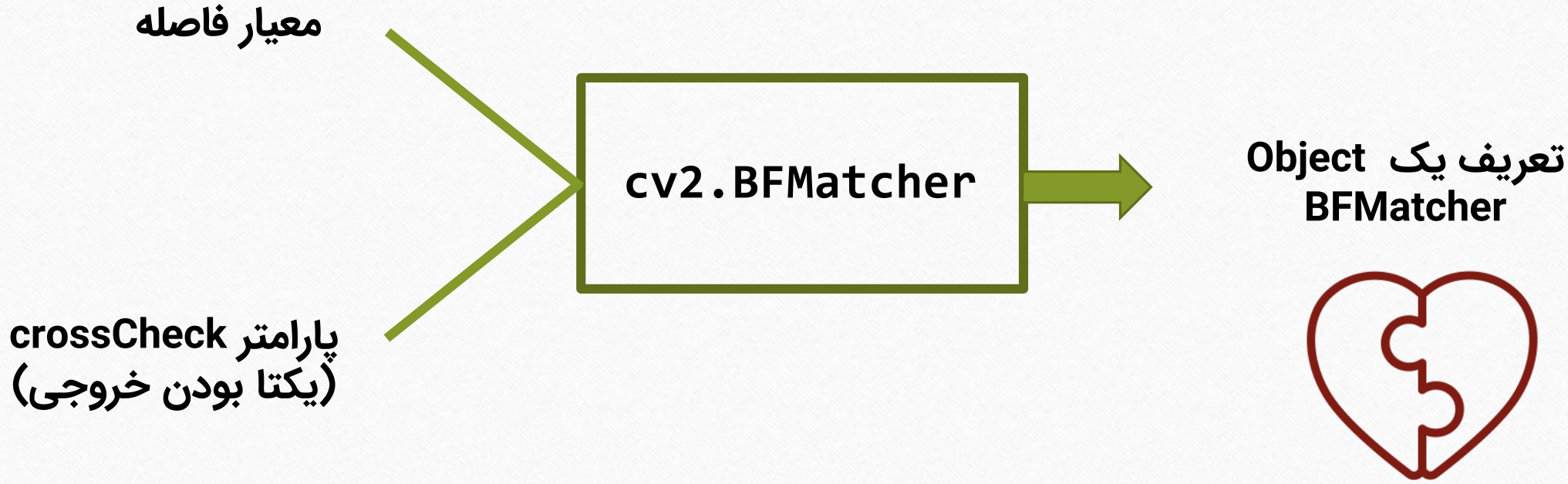
الگوریتم BFMatcher :

یک descriptor از تصویر اول را می گیرد و با تمام descriptor های تصویر دوم مقایسه می کند و نزدیکترین descriptor را بر میگرداند.

مقایسه بین اعداد با روش های مختلفی از جمله NORM_L1 و NORM_L2 و ... انجام می گیرد.

در کنار FLANN based matcher از الگوریتم های اصلی مقایسه می باشد.

دستور 32 : تعریف یک Object از BFMatcher



مثال :

```
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
```

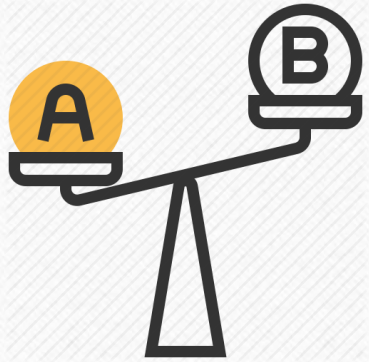
دستور 32 : مقایسه بین descriptor ها

descriptor اول

descriptor دوم

bf.match

بهترین descriptor هایی
که با هم match هستند.



مثال :

```
match = bf.match(des1, des2)
```

دستور 32 : رسم نقاطی که match هستند

تصویر اول

KP تصویر اول

تصویر دوم

KP تصویر دوم

Match ها

None

cv2.drawMatches

تصویری از نقاطی که match هستند.

مثال :

```
matching_result = cv2.drawMatches(book1, kp1, book2, kp2, match, None)
```

اشکالات را چگونه رفع کنیم ؟

روش 1 : sort کردن

Sort کردن نقاط match بر حسب فاصله

روش 2 : ratio test

روش پیشنهادی david lowe بر مقاله SIFT

روش 1 : sort کردن

```
match = sorted(match, key = lambda x:x.distance)
```

تنظیم معیار بر حسب فاصله

```
matching_result = cv2.drawMatches(book1, kp1, book2, kp2,  match[:40], None)
```

قانون lowe Ratio test

به کمک این قانون می توانیم برخی از feature هایی که مناسب نیستند را حذف کنیم.

این قانون در واقع مقایسه بین دو feature ای است که به feature مورد نظر ما نزدیکتر هستند.

اگر دو feature فاصله تقریبا یکسانی با یک feature دیگر داشته باشند، احتمالا گزینه های مناسبی نیستند.

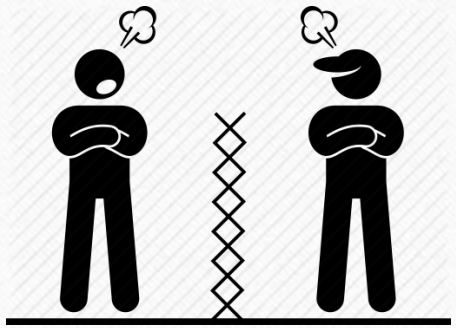
دستور 32 : مقایسه بین descriptor ها با KNN

descriptor اول

descriptor دوم

پارامتر K

K تا از بهترین descriptor
برای هر descriptor



نکته : CrossCheck باید حتما false شود.

مثال :

```
matches = bf.knnMatch(des1,des2, k=2)
```