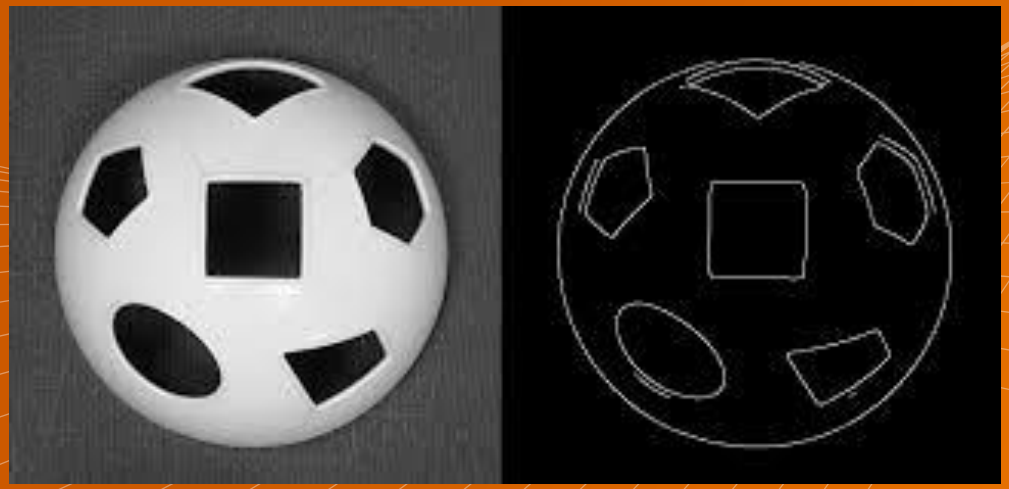
تشخیص لبه
(edge detection)



مقدمه

یکی از متداولترین اعمال در تحلیل تصویر **تشخیص لبه** می باشد به این دلیل که لبه مرز میان یک شی و زمینهء آن است به عبارت دیگر لبه تغییر دو سطح خاکستری یا مقادیر مربوط به روشنایی دو پیکسل مجاور است که در مکان خاصی از تصویر رخ می دهد. هر چه این تغییر در سطح بیشتر باشد تشخیص لبه ساده تر خواهد بود.

نقاطی از تصویر که دارای **تغییرات روشنایی ناگهانی** هستند اغلب لبه نامیده می شوند.

انسان می تواند بسیاری از اشیاء را از روی تصویر خطوط آنها شناسایی کند. بهترین مثال برای آن تصاویر کارتنی است. سیستم بینایی انسان قبل از بازشناسی رنگ یا شدت روشنایی نوعی کشف لبه انجام می دهد. بنابراین انجام کشف لبه قبل از تفسیر تصاویر در سیستمهای خودکار منطقی به نظر می رسد. انجام عملیات کشف لبه پردازش مهمی در بسیاری از سیستمهای بینایی مصنوعی محسوب می شود. هدف اصلی لبه یابی **کاهش حجم داده ها در تصویر به همراه حفظ ساختار و شکل اصلی تصویر** است.

انواع الگوریتم های لبه یابی

۱– الگوریتم sobel

تصویر اصلی:

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

پیش از پیاده‌سازی کد مربوط به تشخیص لبه در پایتون، ابتدا ریاضیات نهفته در پس انجام این کار مورد بررسی قرار می‌گیرد. انسان‌ها در کار تشخیص لبه‌های یک تصویر عملکرد بسیار خوبی دارند، اما چطور می‌توان به یک کامپیوتر یاد داد که کار مشابهی را انجام دهد؟ ابتدا، می‌توان یک تصویر که در آن یک مربع سیاه در میان یک پس زمینه سفید قرار گرفته است، فرض کرد.

در این مثال، در نظر گرفته می‌شود که هر پیکسل مقداری بین ۰ (سیاه) و ۱ (سفید) دارد. بنابراین، در حال حاضر فقط با تصویر سیاه و سفید کار خواهد شد. نظریه کاملا مشابهی، روی تصاویر رنگی نیز قابل اعمال است. اکنون، فرض می‌شود که کاربر در تلاش است تا مشخص کند که آیا پیکسل مشخص شده با رنگ سبز، جزوی از این تصویر است یا خیر. انسان‌ها به سادگی پاسخ این پرسش را می‌دانند و فورا پاسخ مثبت خواهند داد. اما چطور می‌توان از پیکسل‌های موجود در همسایگی استفاده کرد تا به کامپیوتر کمک شود که به نتیجه‌ای مشابه آنچه انسان رسیده است، برسد. **یک مربع کوچک ۳ × ۳ از پیکسل‌های محلی که پیکسل سبز رنگ در وسط آن قرار دارد، در نظر گرفته می‌شود.** این مربع، به رنگ قرمز نمایش داده شده است. سپس، فیلتری روی این مربع کوچک اعمال می‌شود

فیلتری که اعمال می‌شود، در تصویر روبرو نمایش داده شده است و در نگاه اول، کمی عجیب به نظر می‌رسد. اما در ادامه، عملکرد آن مورد بررسی قرار خواهد گرفت. در ادامه این مطلب، هر گاه گفته شود که «فیلتر روی مربع محلی کوچک متشکل از پیکسل‌ها اعمال شود» بدین معنا است که هر پیکسل موجود در مربع محلی قرمز رنگ در هر پیکسل موجود در فیلتر به صورت «درایه‌ای» **ضرب** می‌شود. بنابراین، پیکسل موجود در سمت چپ بالا در مربع قرمز، برابر با ۱ است. این در حالی است که پیکسل موجود در سمت چپ بالا در فیلتر، برابر با ۱- است؛ بنابراین، حاصل ضرب این دو با یکدیگر، برابر با ۱- است که می‌توان آن را در بالا و سمت چپ مربع نتیجه، مشاهده کرد. سایر پیکسل‌های موجود در مربع نتیجه نیز به همین ترتیب به دست آمده‌اند.

تصویر اصلی:

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

فیلتر عمودی:

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

اعمال فیلتر روی پیکسل‌ها:

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

مجموع = ۴- که مقدار کمینه است. نگاشت می‌شود به ۰.

گام بعدی، جمع کردن پیکسل‌های موجود در مربع نتیجه است که خروجی ۴– را به دست می‌دهد. شایان توجه است که **۴– در واقع کوچک‌ترین مقداری است که می‌توان با اعمال این فیلتر به دست آورد** (زیرا پیکسل‌ها فقط می‌توانند ۰ و ۱ باشند). بنابراین، پیکسل مورد پرسش، بخشی از لبه عمودی بالایی است. زیرا مقدار کمینه ۴– حاصل شده است.

فیلتر عمودی:

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

اعمال فیلتر روی پیکسل‌ها:

0	0	0
0	0	0
1	2	1

مجموع = ۴ که مقدار بیشینه است. نگاشت می‌شود به ۱.

تصویر اصلی:

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

فیلتر عمودی:

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

اعمال فیلتر روی پیکسل‌ها:

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

مجموع = ۴- که مقدار کمینه است. نگاشت می‌شود به ۰.

تصویر اصلی:

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

می‌توان مشاهده کرد که نتایج نسبتا مشابهی حاصل شده است، با این تفاوت که مجموع مقادیر نتیجه، برابر با ۴ است. **۴ بیشترین مقداری است که می‌توان با اعمال این فیلتر به دست آورد**. بنابراین، مشخص می‌شود که پیکسل در لبه عمودی پایینی قرار دارد؛ زیرا پس از اعمال فیلتر، بیشترین مقدار ۴ به دست آمده است. برای نگاشت مجدد این مقادیر به بازه ۰ و ۱، هر یک از ارقام با ۴ جمع و سپس بر ۸ تقسیم می‌شوند. بدین شکل، ۴– به ۰ (مشکی) و ۴ به ۱ (سفید) نگاشت می‌شود. بنابراین، با استفاده از این فیلتر که به آن «فیلتر عمودی سوبل» گفته می‌شود، می‌توان به سادگی لبه‌های عمودی موجود در تصویر را پیدا کرد.

برای لبه‌های افقی چه کاری می‌توان انجام داد؟ به سادگی، **ترانهاده فیلتر عمودی** اعمال می‌شود و این فیلتر جدید برای شناسایی مرزهای افقی روی تصویر اعمال می‌شود. اکنون، اگر هدف تشخیص مرزهای افقی، مرزهای عمودی و مرزهایی است که در میان این‌ها قرار دارند، می‌توان امتیازهای افقی و عمودی را محاسبه کرد.

برای مشخص شدن کلیه لبه ها:

اگر Gxو Gyتصاویر فیلتر شده به وسیله ماسک افقی و عمودی باشند، آنگاه تصویر $\sqrt{[G_x]^2+[G_y]^2}$ لبه های تصویر را بهتر نشان میدهد .روال فوق به عملگر یا الگورریتم سوبل موسوم است.

در عمل، به منظور کاهش هزینه محاسبات، به جای بالا میتوان از تقریب [Gx] + [Gy]استفاده میشود .توجه شود که نتیجه این دو فرمول تقریبا یکسان است ولی فرمول دوم با هزینه کمتری قابل محاسبه است.

```
import cv2
import numpy as np
#define the vertical filter
vertical_filter = [[-1,-2,-1], [0,0,0], [1,2,1]]
#define the horizontal filter
horizontal_filter = [[-1,0,1], [-2,0,2], [-1,0,1]]
#read image
img = cv2.imread(r'C:\Users\MOSALAS\Pictures\elephant.jpg')
#get the dimensions of the image
n,m,d = img.shape
#initialize the edges image
edges_img = img.copy()
#loop over all pixels in the image
for row in range(3, n-2):
    for col in range(3, m-2):
        #create little local 3x3 box
        local_pixels = img[row-1:row+2, col-1:col+2, 0]
        #apply the vertical filter
        vertical_transformed_pixels = vertical_filter*local_pixels
        #remap the vertical score
        vertical_score = vertical_transformed_pixels.sum()/4
        #apply the horizontal filter
        horizontal_transformed_pixels = horizontal_filter*local_pixels
        #remap the horizontal score
        horizontal_score = horizontal_transformed_pixels.sum()/4
        #combine the horizontal and vertical scores into a total edge score
        edge_score = (vertical_score**2 + horizontal_score**2)**.5
        #insert this edge score into the edges image
        edges_img[row, col] = [edge_score]
cv2.imshow('sobel',edges_img)
cv2.waitkey(0)
```

نتیجه:

۲– الگوریتم Canny

لبه یاب کنی توسط جان اف کنی در سال ۱۹۸۶ ایجدداد شد و هنوز یک لبه یاب استاندارد و با دقت و کیفیت بالا میباشد.الگوریتم لبه یابی کنی یکی از بهترین لبه یابها تا به امروز است. این الگوریتم لبه یابی از سه بخش اصلی زیر تشکیل شده است:

- تضعیف نویز
- پیدا کردن نقاطی که بتوان آنها را به عنوان لبه در نظر گرفت
- حذف نقاطی که احتمال لبه بودن آنها کم است

معیارهایی که در لبه یابی کنی مطرح است:

۱ –پایین آوردن نرخ خطا– یعنی تا حد امکان هیچ لبه ای در تصویر نباید گم شود و هم چنین هیچ چیزی که لبه نیست نباید به جای لبه فرض شود. لبه هان پیدا شده تا حد ممکن به لبه ها اصلی نزدیک باشند.

۲ –لبه در مکان واقعی خود باشد– یعنی تا حد ممکن لبه ها کمترین فاصله را با مکان واقعی خود داشته باشند.

۳ –لبه ها کمترین ضخامت را داشته باشند– (در صورت امکان یک پیکسل).

لبه یاب کنی بخاطر توانایی در تولید لبه های نازک تا حد یک پیکسل برای لبه های پیوسته معروف شده است. این لبه یاب شامل چهار مرحله و چهار ورودی زیر است:

یک تصویر ورودی

یک پارامتر به نام سیگما جهت مقدار نرم کنندگی تصویر

یک حد آستانه بالا (Th)

یک حد آستانه پایین (Tl)

و مراحل شامل

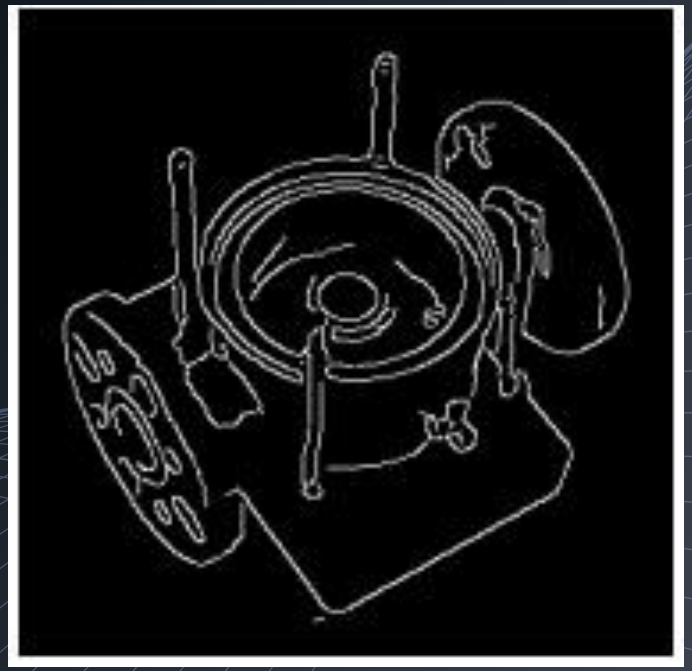
۱– در ابتدا باید تصویر رنگی را به جهت لبه یابی بهتر به یک تصویر سطح خاکستری تبدیل کرد.

۲– نویز را از تصویر دریافتی حذف کرد. بدلیل اینکه فیلتر گاوسین از یک ماسک ساده برای حذف نویز استفاده می کند لبه یاب کنی در مرحله اول برای حذف نویز آن را بکار میگیرد.

۳– در یک تصویر سطح خاکستر جایی را که بیشترین تغییرات را داشته باشند به عنوان لبه در نظر گرفته می شوند و این مکانها با گرفتن گرادیان تصویر با استفاده عملگر سوبل بدست می آیند. سپس لبه های مات یافت شده به لبه های تیزتر تبدیل می شوند.

۴– برخی از لبه های کشف شده واقعا لبه نیستند و در واقع نویز هستند که باید آنها توسط حد آستانه فیلتر شوند.آستانه از دو حد آستانه بالاتر (Th)و حد آستانه پایین تر (Tl) استفاده کرده و کنی پیشنهاد می کند که نسبت استانه بالا به پایین سه به یک باشد.

این روش بیشتر به کشف لبه های ضعیف به درستی می پردازد و کمتر فریب نویز را می خورد و از بقیه روش ها بهتر است.

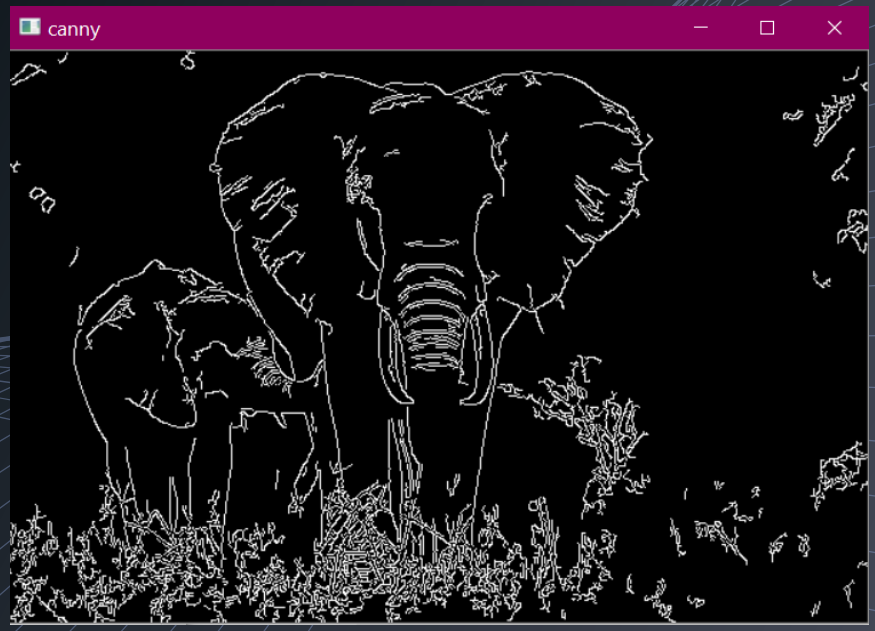


```python
import cv2
import numpy as np

img = cv2.imread(r'C:\Users\MOSALAS\Pictures\elephant.jpg')
edges_img = cv2.Canny(img,100,300)

cv2.imshow('canny',edges_img)
cv2.waitkey(0)
```

نتیجه:



۳– الگوریتم Roberts

این الگوریتم به نویز حساسیت زیادی دارد وپیکسل های کمتری را برای تقریب گرادیان بکار می برد،درضمن نسبت به الگوریتم canny هم قدرت کمتری دارد.

+1	0
0	-1

G_x

0	+1
-1	0

G_y

```
import cv2
import numpy as np
#define the vertical filter
vertical_filter = [[1,0], [0,-1]]
#define the horizontal filter
horizontal_filter = [[0,1], [-1,0]]
#read image
img = cv2.imread(r'C:\Users\MOSALAS\Pictures\elephant.jpg')
#get the dimensions of the image
n,m,d = img.shape
#initialize the edges image
edges_img = img.copy()
#loop over all pixels in the image
for row in range(2, n-1):
    for col in range(2, m-1):
        #create little local 2x2 box
        local_pixels = img[row-1:row+1, col-1:col+1,0]
        #apply the vertical filter
        vertical_transformed_pixels = vertical_filter*local_pixels
        #remap the vertical score
        vertical_score = vertical_transformed_pixels.sum()
        #apply the horizontal filter
        horizontal_transformed_pixels = horizontal_filter*local_pixels
        #remap the horizontal score
        horizontal_score = horizontal_transformed_pixels.sum()
        #combine the horizontal and vertical scores into a total edge score
        edge_score = (vertical_score**2 + horizontal_score**2)**.5
        #insert this edge score into the edges image
        edges_img[row, col] = [edge_score]
cv2.imshow('roberts',edges_img)
cv2.waitkey(0)
```

نتیجه:

۴– الگوریتم Prewitt

این الگوریتم شباهت زیادی با الگوریتم sobel دارد با این تفاوت که ضرایب ماسک آنها با هم فرق می کند.

+۱	+۱	+۱
۰	۰	۰
−۱	-۱	−۱

G_y

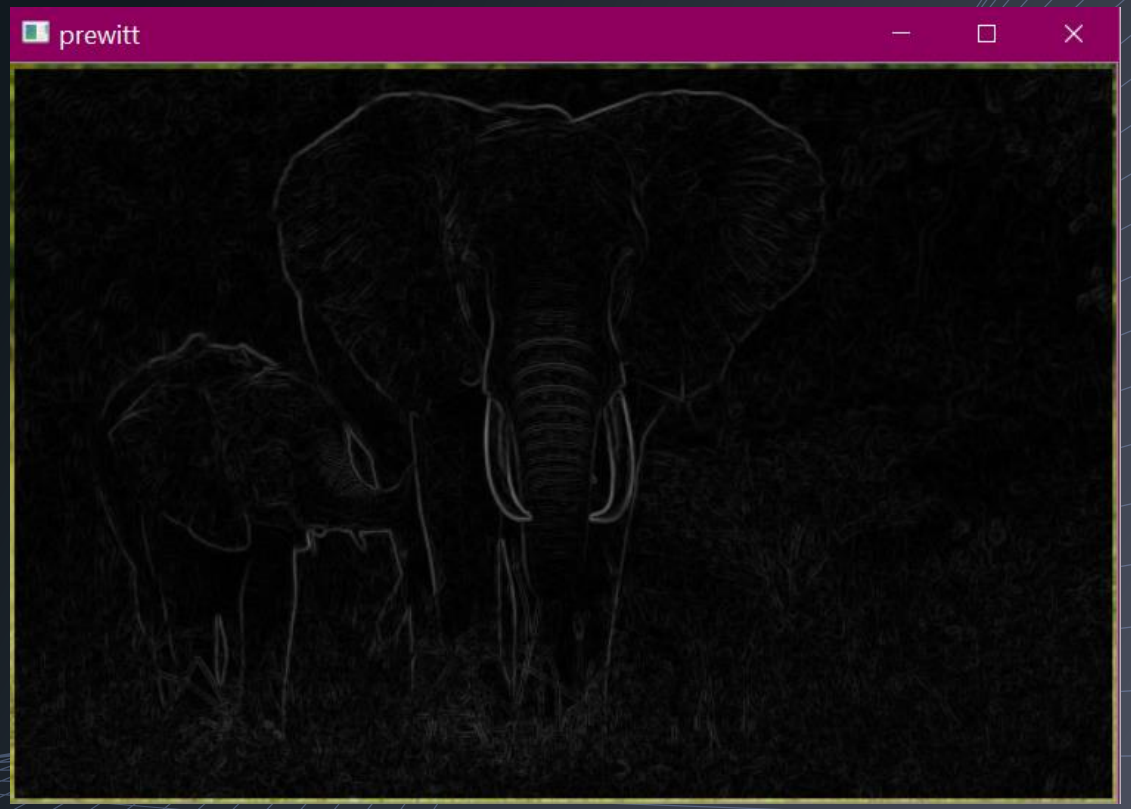
−۱	۰	+۱
-۱	۰	+۱
−۱	۰	+۱

G_x

شکل۵. ماسک آشکارساز لبه prewitt برای G_x ، G_y

```
import cv2
import numpy as np
#define the vertical filter
vertical_filter = [[-1,-1,-1], [0,0,0], [1,1,1]]
#define the horizontal filter
horizontal_filter = [[-1,0,1], [-1,0,1], [-1,0,1]]
#read image
img = cv2.imread(r'C:\Users\MOSALAS\Pictures\elephant.jpg')
#get the dimensions of the image
n,m,d = img.shape
#initialize the edges image
edges_img = img.copy()
#loop over all pixels in the image
for row in range(3, n-2):
    for col in range(3, m-2):
        #create little local 3x3 box
        local_pixels = img[row-1:row+2, col-1:col+2, 0]
        #apply the vertical filter
        vertical_transformed_pixels = vertical_filter*local_pixels
        #remap the vertical score
        vertical_score = (vertical_transformed_pixels.sum()+3)/6
        #apply the horizontal filter
        horizontal_transformed_pixels = horizontal_filter*local_pixels
        #remap the horizontal score
        horizontal_score = (horizontal_transformed_pixels.sum()+3)/6
        #combine the horizontal and vertical scores into a total edge score
        edge_score = (vertical_score**2 + horizontal_score**2)**.5
        #insert this edge score into the edges image
        edges_img[row, col] = [edge_score]
cv2.imshow('prewitt',edges_img)
cv2.waitkey(0)
```

نتیجه:



۵– الگوریتم laplacian

0	-1	0
-1	4	-1
0	-1	0

The laplacian operator

بر خلاف آشکارساز لبه Sobel، آشکارساز لبه Laplacian فقط از یک کرنل استفاده می کند. همچنین مشتقات مرتبه دوم را در یک گذر محاسبه می کند.

```
import cv2
import numpy as np
#define the lap filter
lap_filter = [[0,1,0], [1,-4,1], [0,1,0]]
#read the image
img = cv2.imread(r'C:\Users\MOSALAS\Pictures\elephant.jpg')
#get the dimensions of the image
n,m,d = img.shape
#initialize the edges image
edges_img = img.copy()
#loop over all pixels in the image
for row in range(3, n-2):
    for col in range(3, m-2):
        #create little local 3x3 box
        local_pixels = img[row-1:row+2, col-1:col+2, 0]
        #apply the lap filter
        lap_transformed_pixels = lap_filter*local_pixels
        #remap the lap score
        lap_score = (lap_transformed_pixels.sum()**2)**0.5
        edges_img[row, col] = [lap_score]
cv2.imshow('lap',edges_img)
cv2.waitkey(0)
```

نتیجه: