UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS

iUAI
UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS

MDS[2019] PROCESAMIENTO DE IMÁGENES

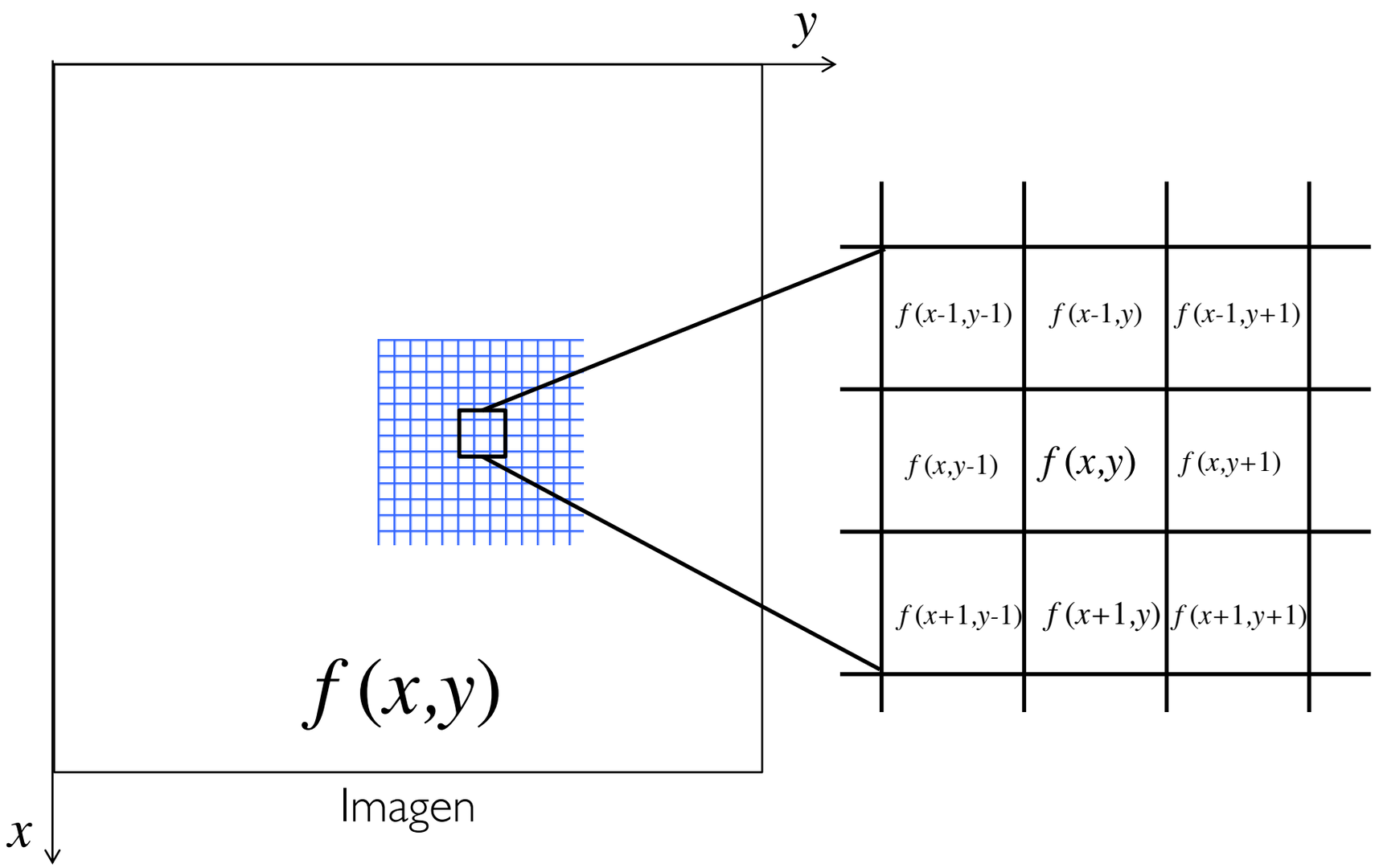
Mejoramiento en el espacio

Miguel Carrasco
miguel.carrasco@uai.cl
1er Semestre 2020

Temario

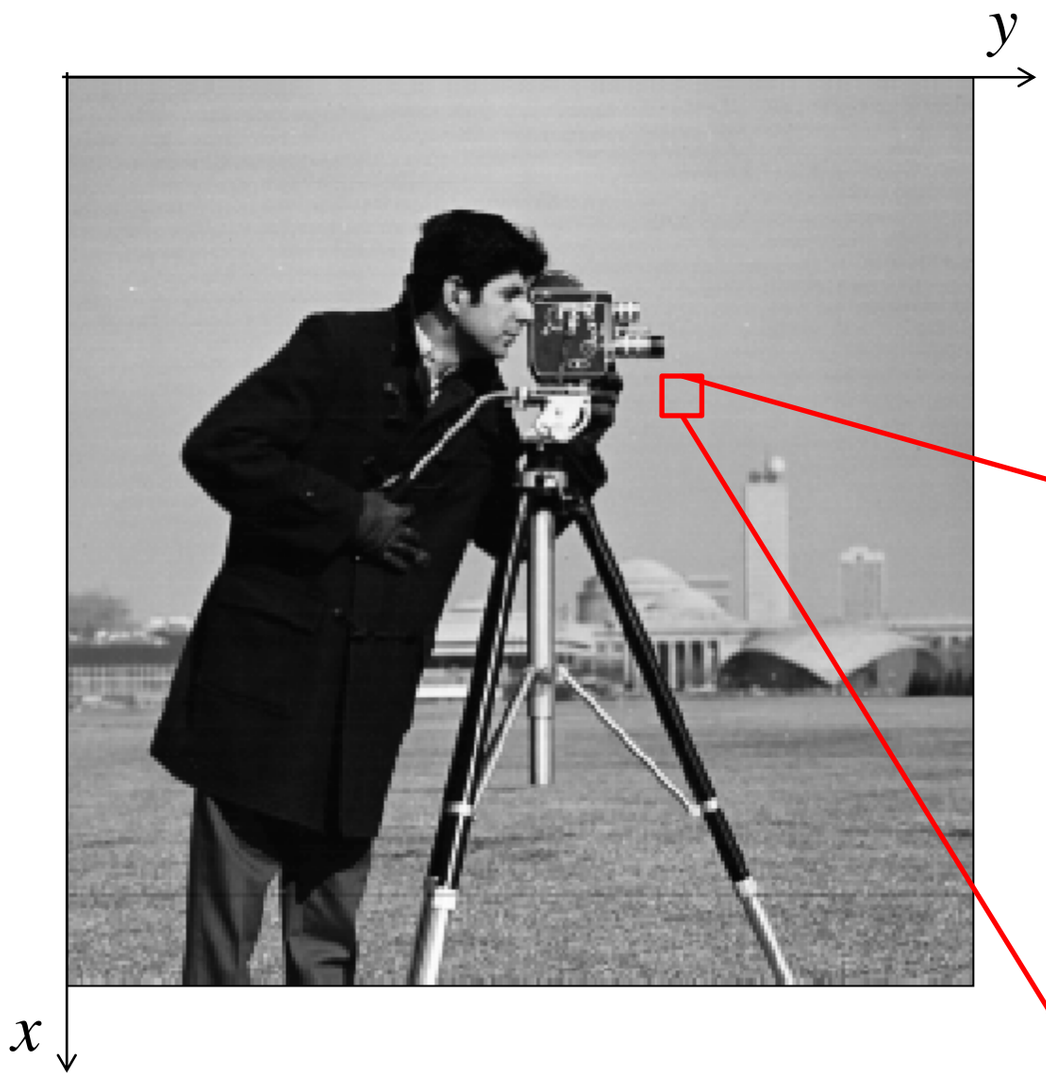
- Mejoramiento en el espacio
  - Filtros espaciales (lineal y no lineal)

- Posiciones del píxel en la imagen



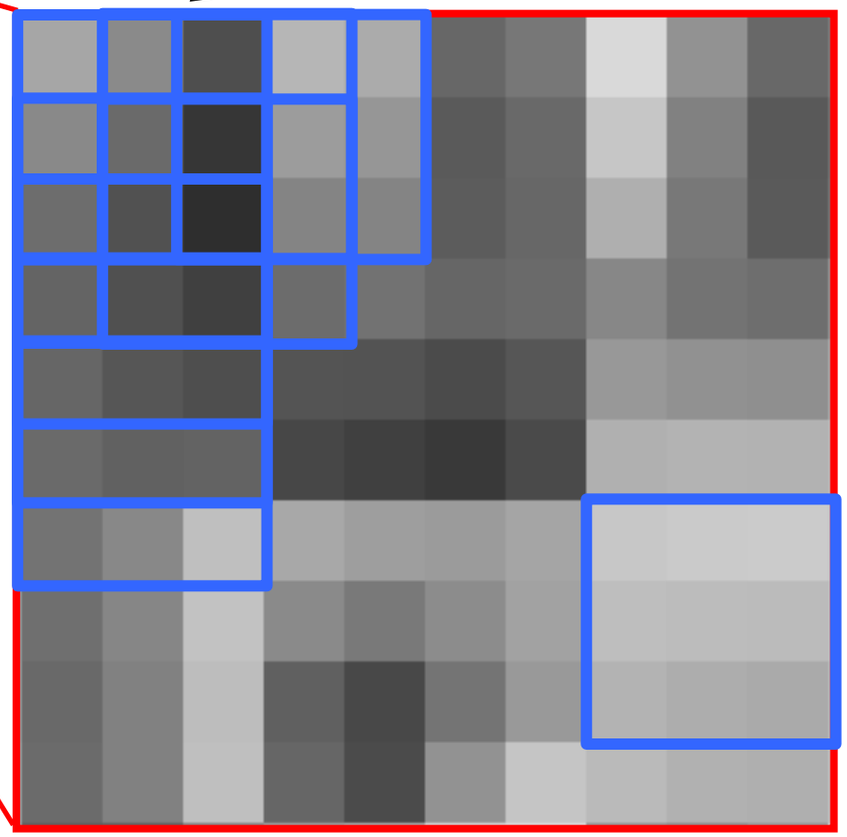
Filtros espaciales

- Proceso de la máscara

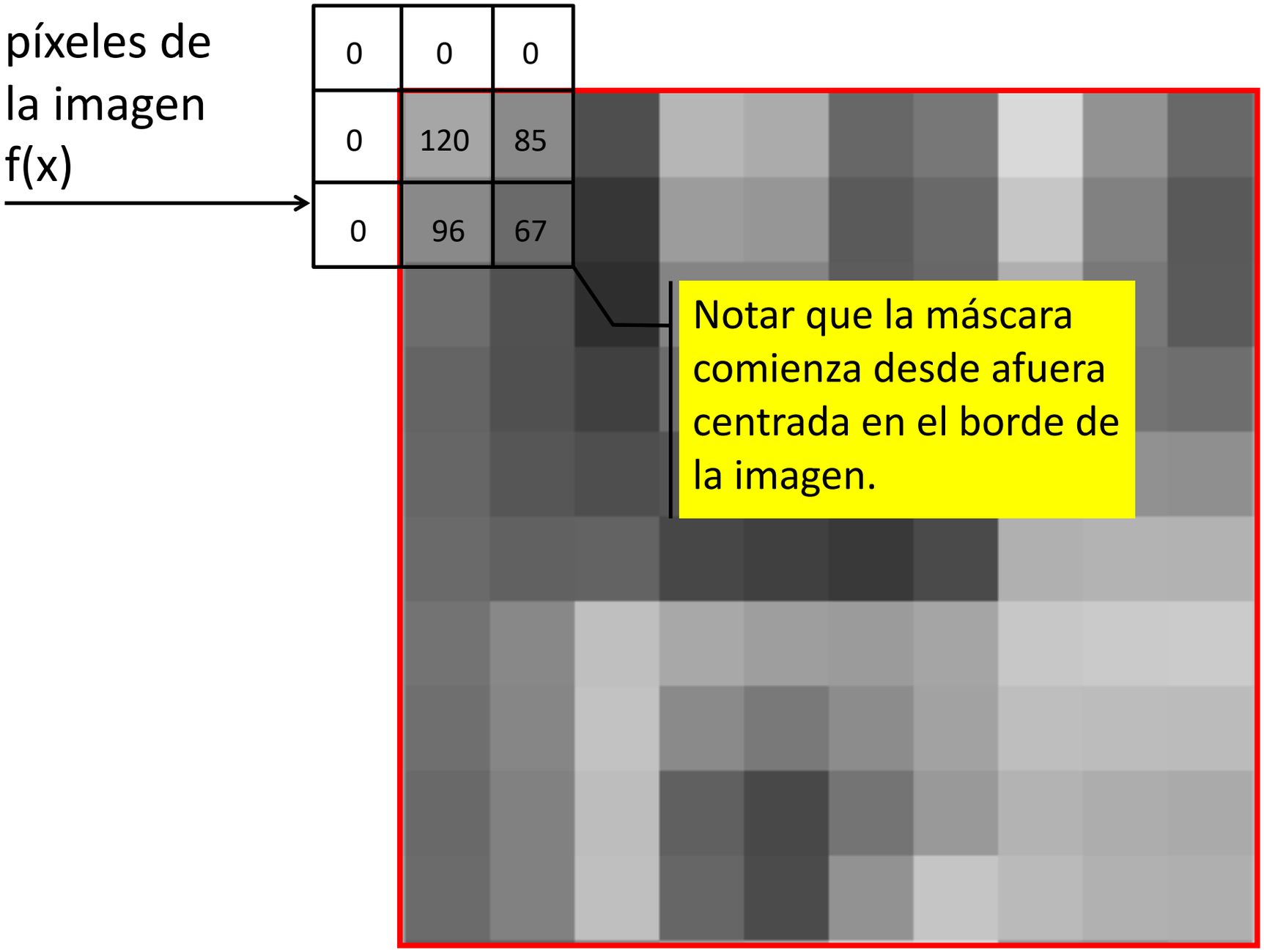


w (-1,-1)	w (-1,0)	w (-1,1)
w (0,-1)	w (0,0)	w (0,1)
w (1,-1)	w (1,0)	w (1,1)

Máscara de [3x3] aplicada sobre una sección de la imagen



- Proceso de la máscara



- Tipos de máscara
  - El proceso de filtrar con una máscara se denomina operación de ventana deslizante. Este consume mucho proceso ya que debe ser aplicado a toda la imagen.

constante que conserva el promedio

$1/9 \times$

1	1	1
1	1	1
1	1	1

Promedio

$1/16 \times$

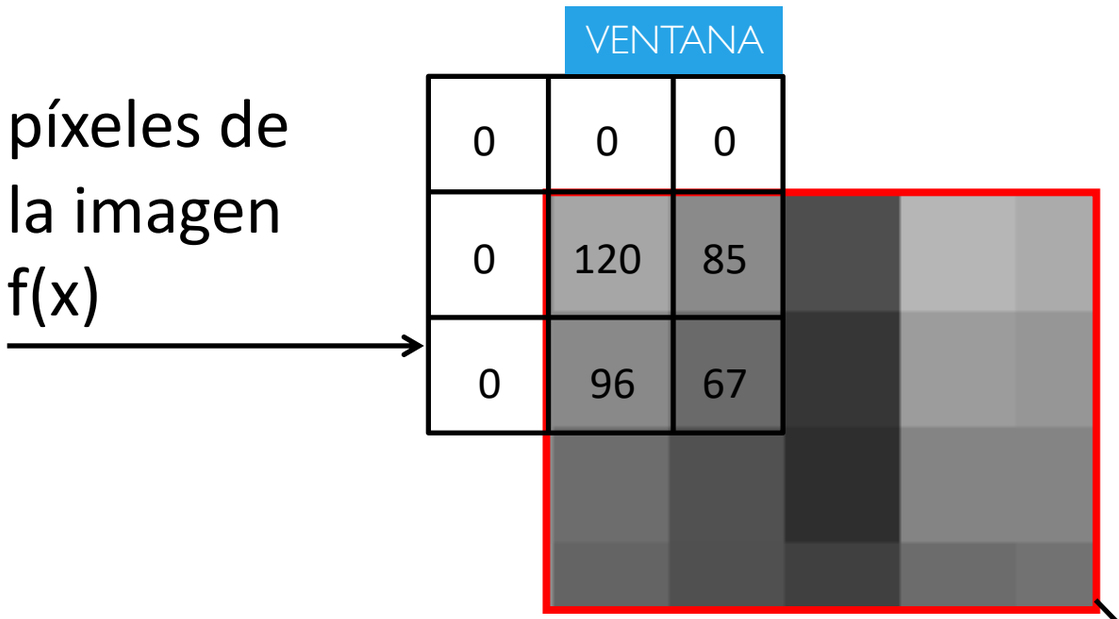
1	2	1
2	4	2
1	2	1

Promedio ponderado

FILTRO LINEAL

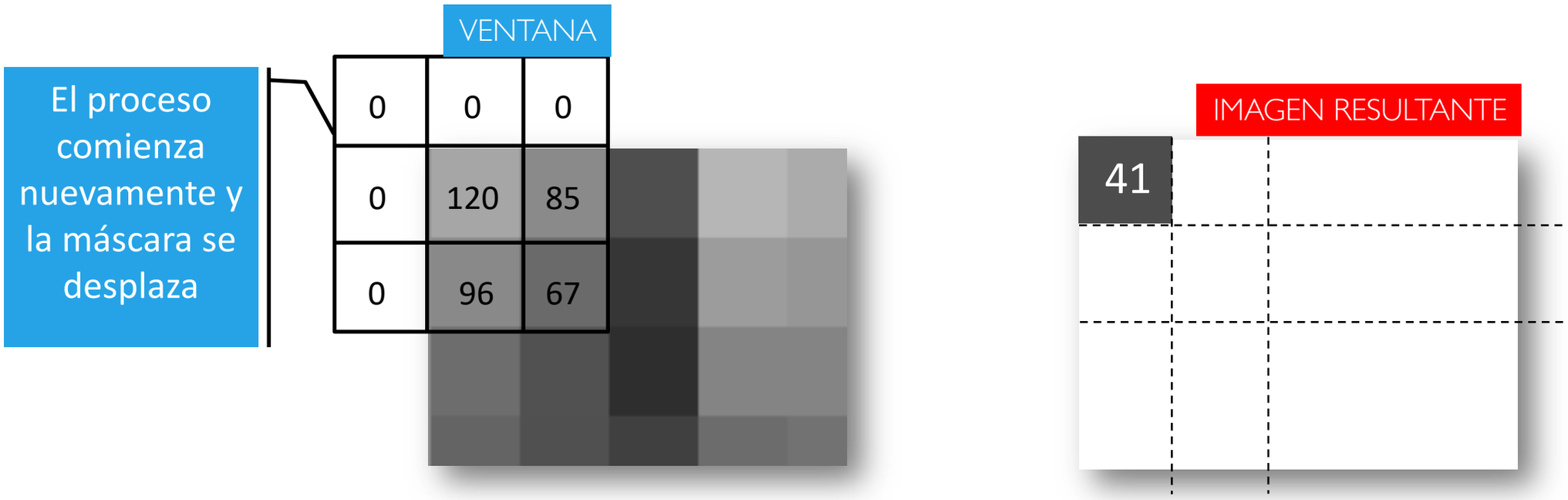
Filtros espaciales

- Proceso de la máscara
  - Primer paso: Extraer una parte de la imagen (ventana deslizante)



El proceso siempre se aplica sobre la imagen original

- Proceso de la máscara
  - Resultado: Se reemplaza solo el valor central de la máscara y luego movemos la máscara a una nueva posición. Recordar que el nuevo valor está en nueva imagen.



$$f(1,1) = 0 \cdot \frac{1}{9} + 0 \cdot \frac{1}{9} + 0 \cdot \frac{1}{9} + 0 \cdot \frac{1}{9} + 120 \cdot \frac{1}{9} + 96 \cdot \frac{1}{9} + 0 \cdot \frac{1}{9} + 85 \cdot \frac{1}{9} + 67 \cdot \frac{1}{9} = 41$$

$$f(1,1) = 41$$

- Paso de máscara

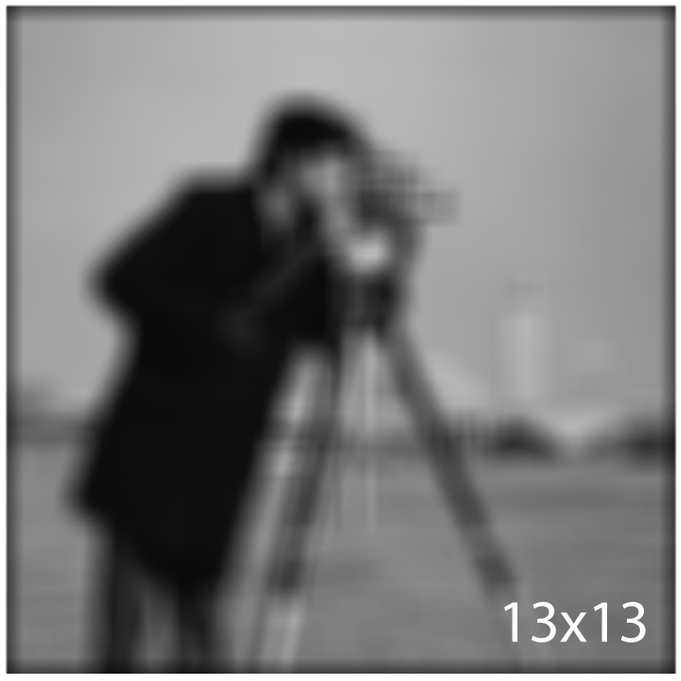
```python
import cv2

img = cv2.imread('cameraman.png')

imgBlur = cv2.blur(img,(5,5))

cv2.imshow('image',imgBlur)
cv2.waitKey(0)
```

resultado ← `cv2.blur(img, (5,5) )`

imagen

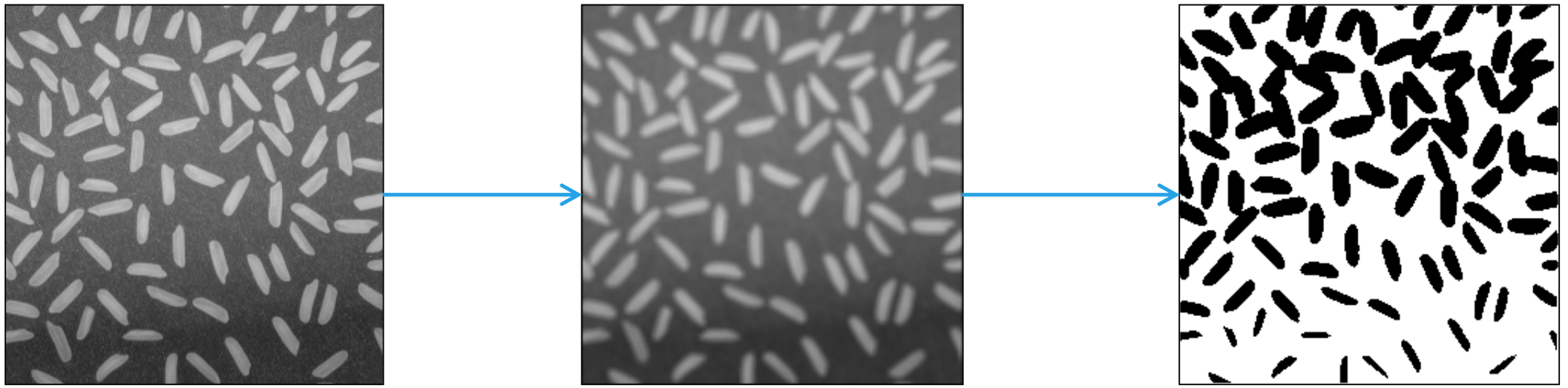
dimensión máscara

- Máscara promedio



Filtros espaciales

- Máscara promedio + umbralización




```python
import cv2

I = cv2.imread('rice.png', cv2.IMREAD_GRAYSCALE)

J = cv2.blur(I,(5,5))

ret, BJ= cv2.threshold(J, 120, 255, type=cv2.THRESH_BINARY)

cv2.imshow('image',BJ)
cv2.waitKey(0)
```

- Máscara desviación estándar

Desviación estándar

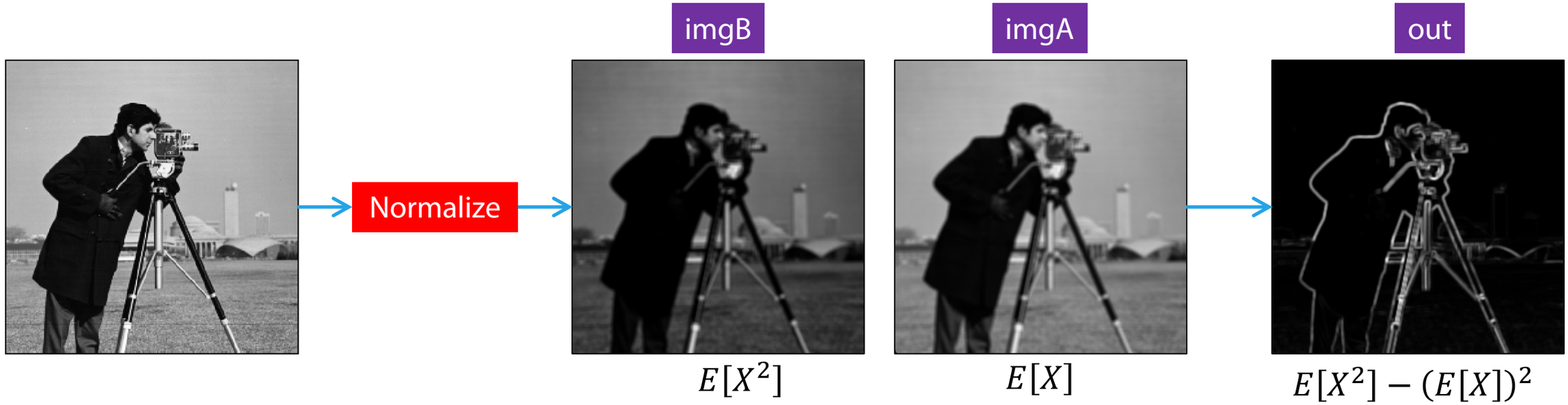
$$
\begin{aligned}
\sigma &= \sqrt{\mathrm{E}[(X-\mu)^2]} \\
&= \sqrt{\mathrm{E}[X^2] + \mathrm{E}[-2\mu X] + \mathrm{E}[\mu^2]} \\
&= \sqrt{\mathrm{E}[X^2] - 2\mu\,\mathrm{E}[X] + \mu^2} \\
&= \sqrt{\mathrm{E}[X^2] - 2\mu^2 + \mu^2} \\
&= \sqrt{\mathrm{E}[X^2] - \mu^2} \\
&= \sqrt{\mathrm{E}[X^2] - (\mathrm{E}[X])^2}
\end{aligned}
$$

$E[X]$ → operador Blur

↑ imagen

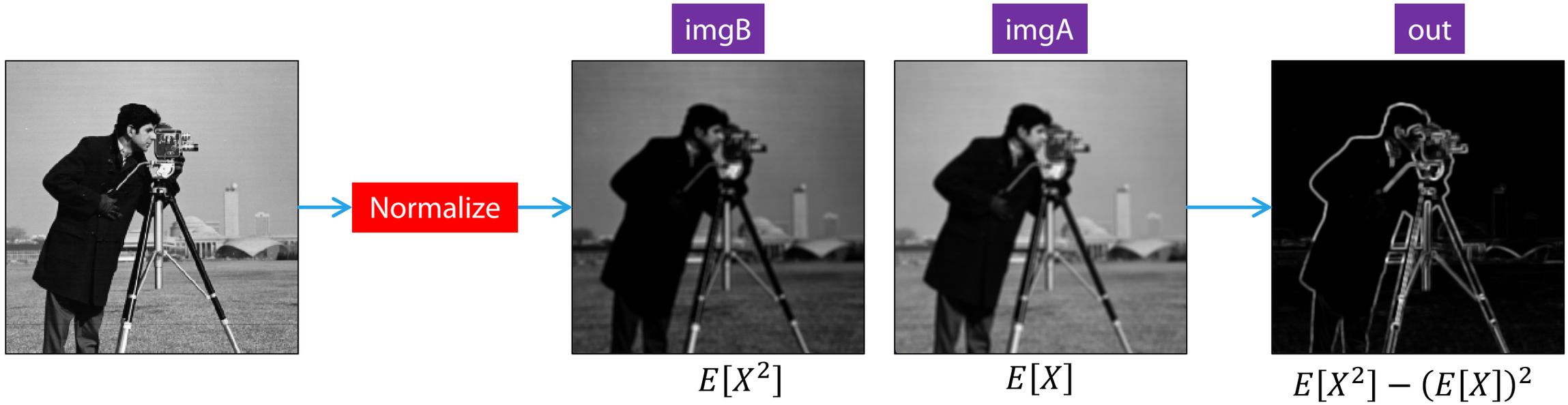
fuente: https://en.wikipedia.org/wiki/Standard_deviation#Definition_of_population_values

- Máscara desviación estándar




```
import cv2

img = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)
imgN = cv2.normalize(img.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)

imgA = cv2.blur(imgN,(3,3))
imgB = cv2.blur(imgN*imgN,(3,3))
```

- Máscara desviación estándar




```
import cv2

img = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)
imgN = cv2.normalize(img.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)

imgA = cv2.blur(imgN,(3,3))
imgB = cv2.blur(imgN*imgN,(3,3))

out = imgB - imgA**2

out = cv2.normalize(out.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)
cv2.imshow('image',out)
cv2.waitKey(0)
```

$E[X^2] - (E[X])^2$

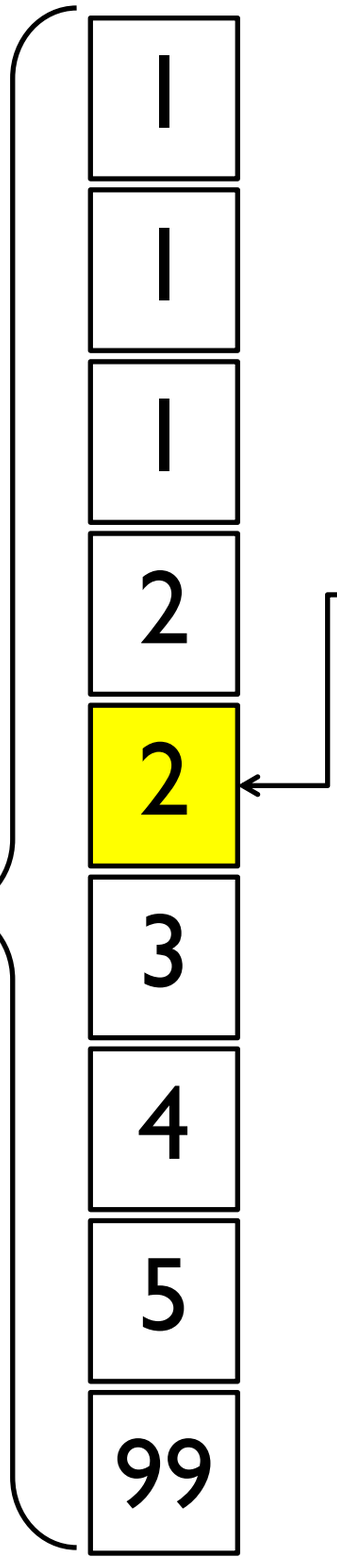
- Máscara desviación estándar

FILTRO NO LINEAL

- Filtro de la mediana

1ro. Determinar la mediana

4	2	3
5	99	2
1	1	1

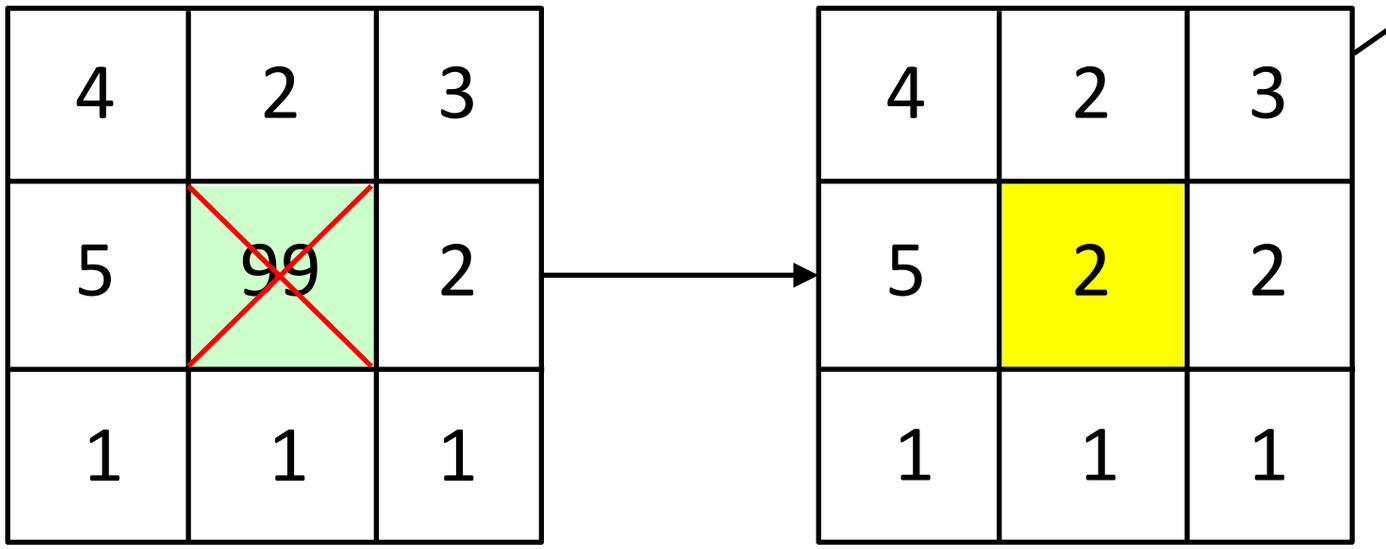


En una lista impar corresponde al valor en la posición media.

Mediana

- Filtro de la mediana

2ro. Reemplazar del valor central el valor mediana



Al reemplazar el centro por el valor de la mediana, eliminamos el ruido

FILTRO NO LINEAL

- Máscara mediana

```python
import cv2

img = cv2.imread('cameraman.png')

img_median = cv2.medianBlur(img,7)

cv2.imshow('image',img_median)
cv2.waitKey(0)
```

resultado ← `cv2.medianBlur(img, 7)`

imagen

número impar

- Máscara mediana

FILTRO NO LINEAL

- Máscara mediana

img

```
import cv2
import numpy as np

img = cv2.imread('cameraman.png', cv2.IMREAD_GRAYSCALE)

m,n = img.shape
mat_noise=np.random.rand(m,n); #Agregamos ruido aleatorio entre 0 y 1

sp_noise_white= np.uint8(np.where(mat_noise>=0.99, 255,0))
sp_noise_black= np.uint8(np.where(mat_noise>=0.01,   1,0))
```

Máscara mediana

img

noise_img_sp

img_median

```
noise_img    = cv2.multiply(img,sp_noise_black)
noise_img_sp = cv2.add(noise_img,sp_noise_white)

img_median   = cv2.medianBlur(noise_img_sp,3)

cv2.imshow('image',img_median)
cv2.waitKey(0)
```

Temario

- Mejoramiento en el espacio
  - Filtros espaciales (lineal y no lineal)
  - Gradientes

Filtros de gradientes

- Gradiente

$$G\{f(i,j)\} = \begin{bmatrix} \dfrac{\partial f(i,j)}{\partial i} \\ \dfrac{\partial f(i,j)}{\partial j} \end{bmatrix}$$

Gradiente de una función vectorial

$$\left|G\{f(i,j)\}\right| = \sqrt{\left(\frac{\partial f(i,j)}{\partial i}\right)^2 + \left(\frac{\partial f(i,j)}{\partial i}\right)^2} \quad (1)$$

Módulo del gradiente

$$\left(\frac{\partial f(i,j)}{\partial i}\right)^2 \approx \left(f(i,j) - f(i,j+1)\right)^2$$

$$\left(\frac{\partial f(i,j)}{\partial i}\right)^2 \approx \left(f(i,j) - f(i+1,j)\right)^2 \quad (2)$$

Aproximación digital al gradiente

Filtros de gradientes

- Reemplazando (2)en (1)

$$\left(\frac{\partial f(i,j)}{\partial i}\right)^2 \approx \left(f(i,j)-f(i,j+1)\right)^2$$

$$\left(\frac{\partial f(i,j)}{\partial i}\right)^2 \approx \left(f(i,j)-f(i+1,j)\right)^2$$

$$\left|G\{f(i,j)\}\right| = \sqrt{\left(\frac{\partial f(i,j)}{\partial i}\right)^2 + \left(\frac{\partial f(i,j)}{\partial i}\right)^2}$$

$$\Rightarrow \left|G\{f(i,j)\}\right| \approx \sqrt{\left(f(i,j)-f(i,j+1)\right)^2 + \left(f(i,j)-f(i+1,j)\right)^2}$$

Separando términos obtenemos una aproximación digital del gradiente

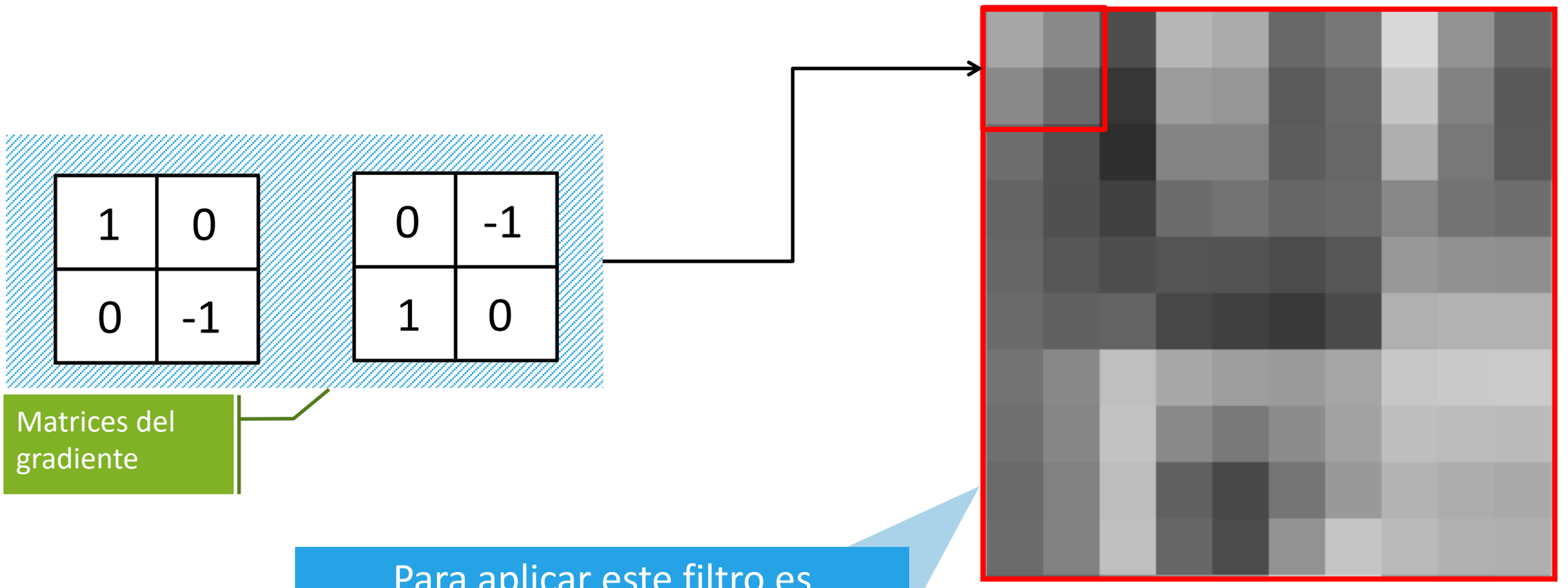
$$\left|G\{f(i,j)\}\right| \approx \left|f(i,j)-f(i,j+1)\right| + \left|f(i,j)-f(i+1,j)\right|$$

Filtros de gradientes

- Filtro de Roberts

Gradiente de Roberts

$$\left|G\left\{f\left(i,j\right)\right\}\right| \approx \left|f\left(i,j\right)-f\left(i+1,j+1\right)\right|+\left|f\left(i+1,j\right)-f\left(i,j+1\right)\right|$$



Matrices del gradiente

Para aplicar este filtro es necesario multiplicar la submatriz por cada matriz de Robert

Filtro de Roberts

```python
import cv2
from skimage import filters

img = cv2.imread('cameraman.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

roberts = filters.roberts(gray)

cv2.imshow('roberts',roberts)
cv2.waitKey(0)
```

esta biblioteca posee muchísimos filtros, cuec

Abre *una ventana de terminal*

Escribe: pip install scikit-image

ver más en: https://scikit-image.org/docs/dev/api/skimage.filters.html

Filtros de gradientes

- Filtro de Roberts

El filtro de Robert permite determinar los bordes, sin embargo, es muy sensible al ruido

Filtros de gradientes

- Filtro Laplaciano (de segunda derivada)

1	1	1
1	-8	1
1	1	1

0	1	0
1	-4	1
0	1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

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

Observe que la suma de los valores de cada matriz es cero.

Filtros de gradientes

0	1	0
1	-4	1
0	1	0

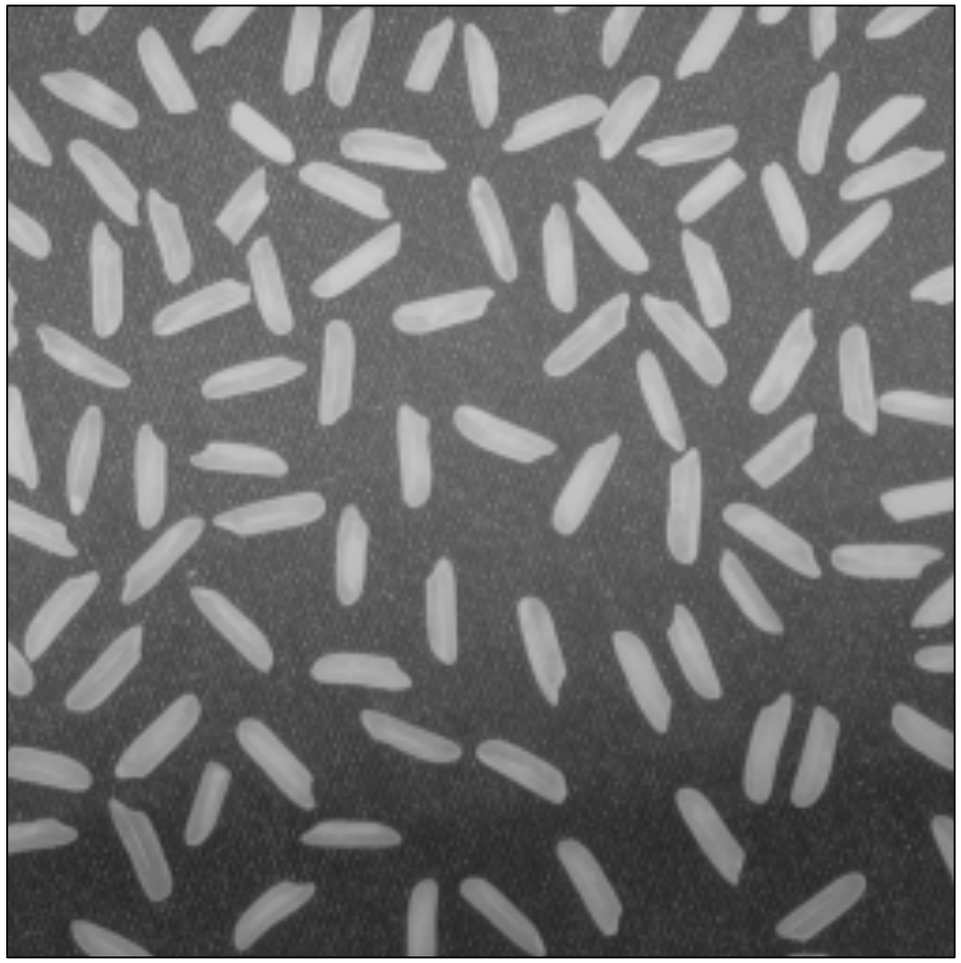
0	-1	0
-1	4	-1
0	-1	0

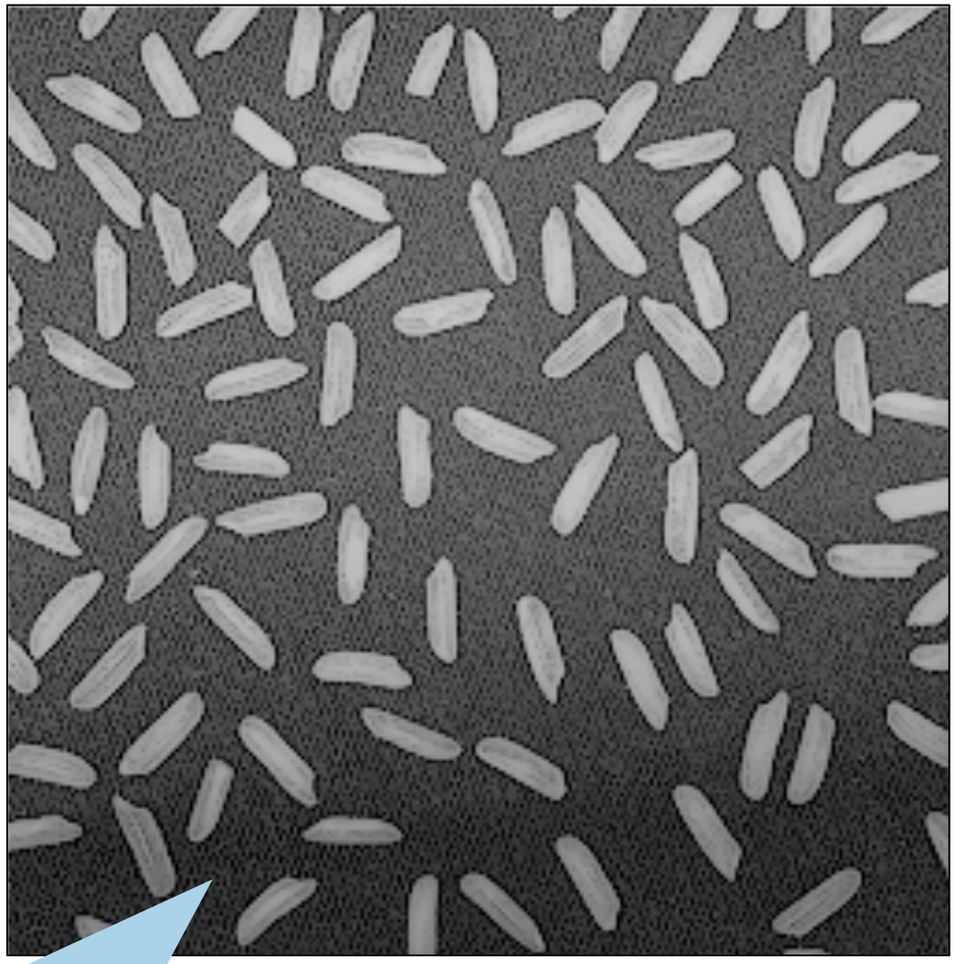
1	1	1
1	-8	1
1	1	1

-1	-1	-1
-1	8	-1
-1	-1	-1

Filtros de gradientes

- Unsharp mask

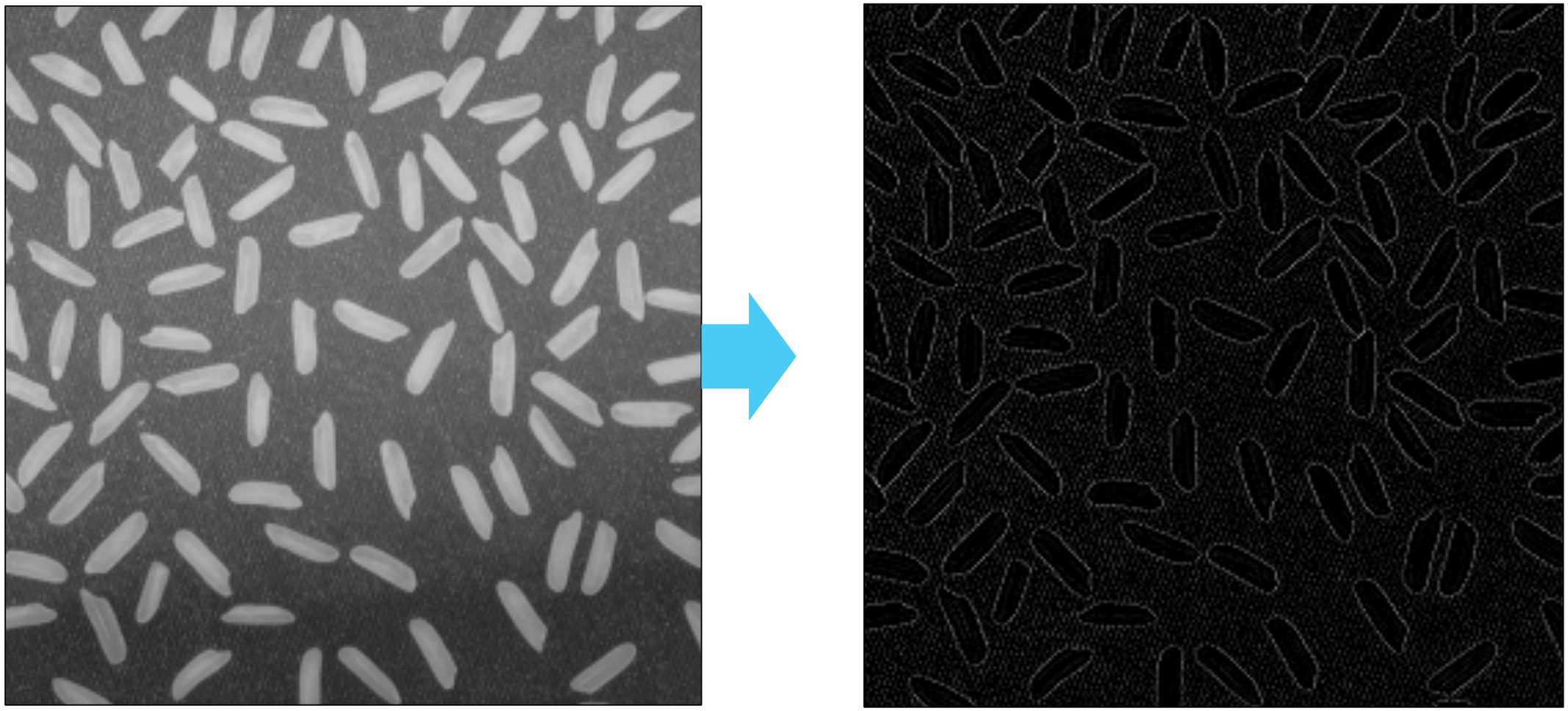




Esta imagen se ve más nítida, ¿por qué?

Filtros de gradientes

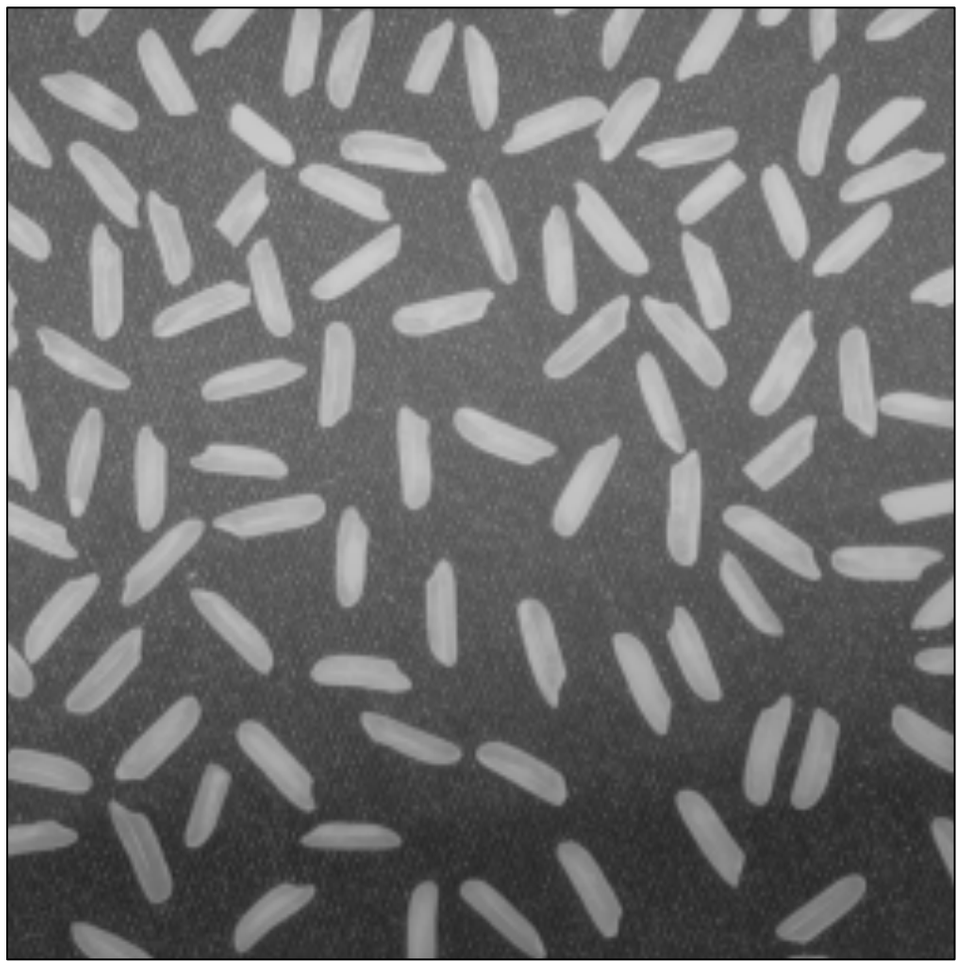
- Unsharp mask

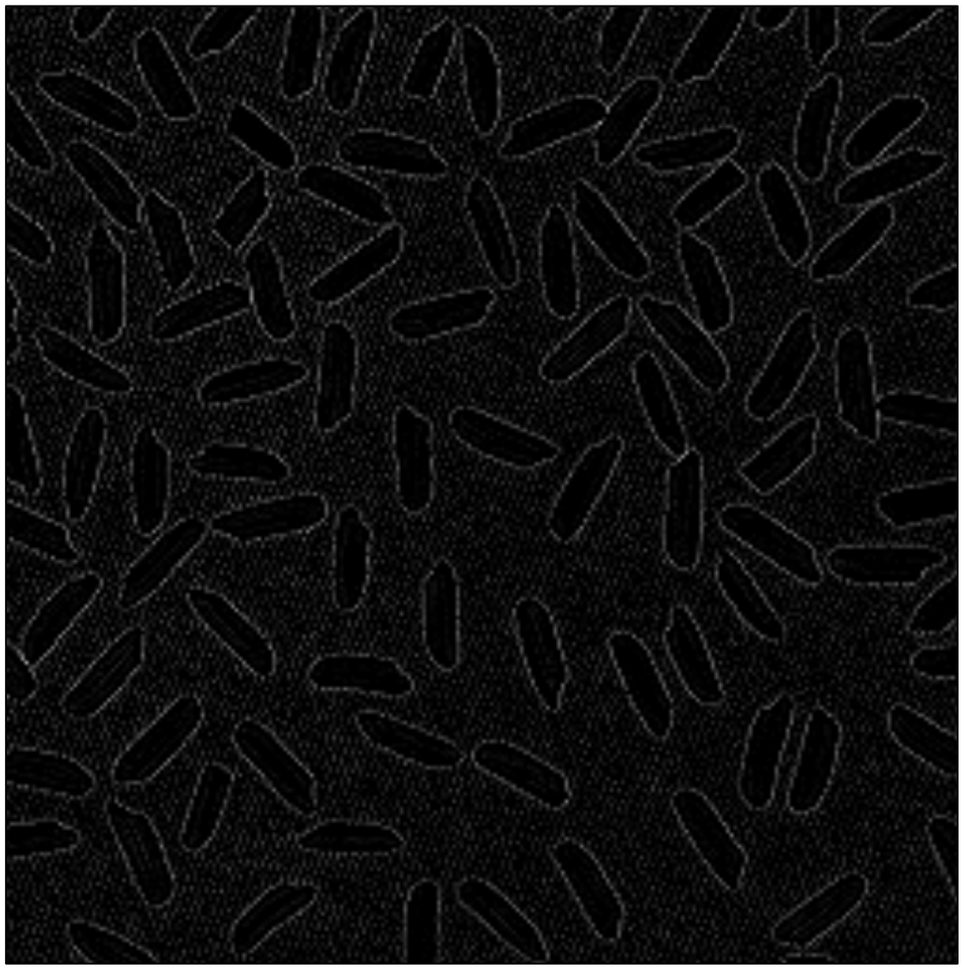


PASO 1:
Calcular el Laplaciano

Filtros de gradientes

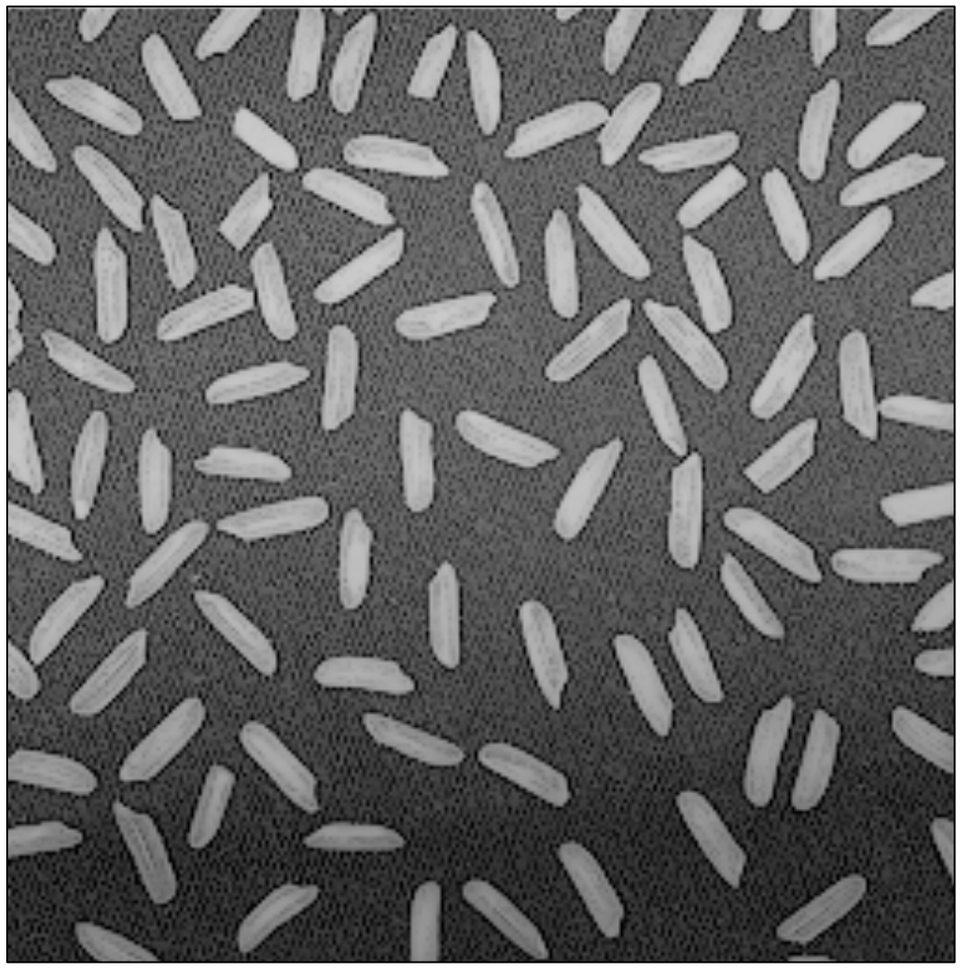
- Unsharp mask





PASO 2:
Restar ambas imágenes

- Unsharp mask



```
import cv2

img = cv2.imread('rice.png')

lpl = cv2.Laplacian(img,cv2.CV_8U)

uSharp= cv2.subtract(img, lpl)

cv2.imshow('img',img)
cv2.imshow('usharp',uSharp)
cv2.waitKey(0)
```

Este es un clásico del procesamiento de imágenes

Filtros de gradientes

- Operador de Sobel

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

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

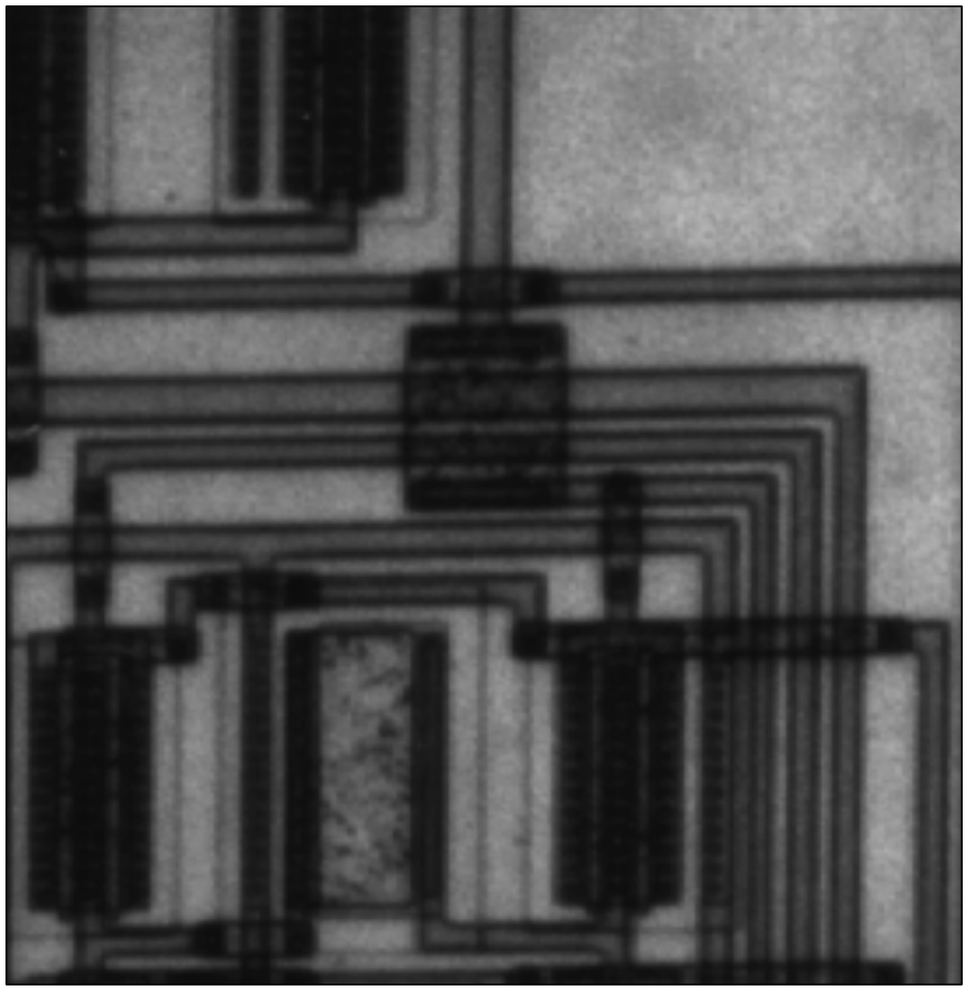
- Operador de Prewitt

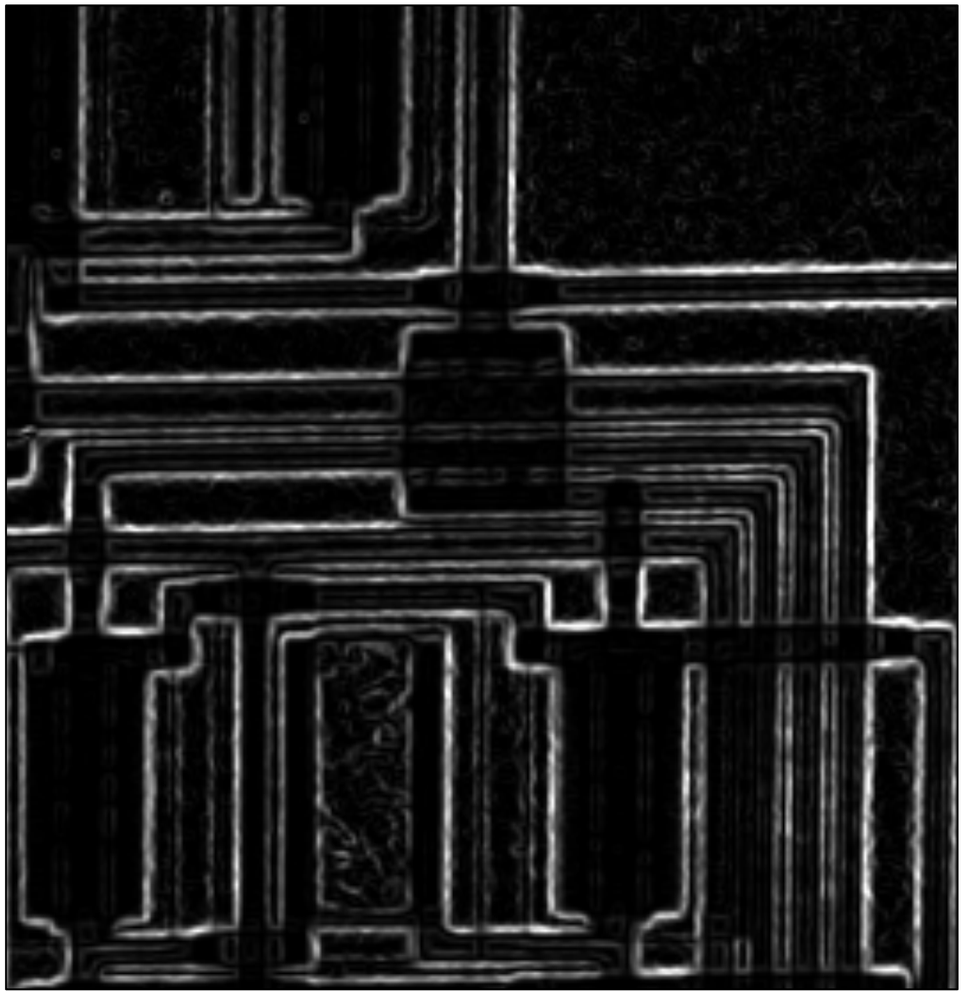
1	1	1
0	0	0
-1	-1	-1

1	0	-1
1	0	-1
1	0	-1

Filtros de gradientes

- Operador de Sobel



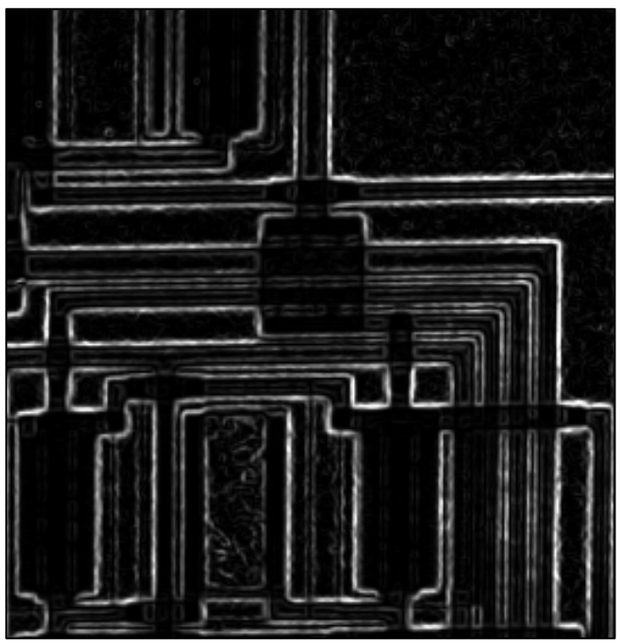


Resultado del operador SOBEL

Filtros de gradientes

- Operador de Sobel

```python
import cv2
from skimage import filters

img = cv2.imread('circuit.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

edges= filters.sobel(gray)

out = cv2.normalize(edges.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)
cv2.imshow('Sobel',out)
cv2.waitKey(0)
```

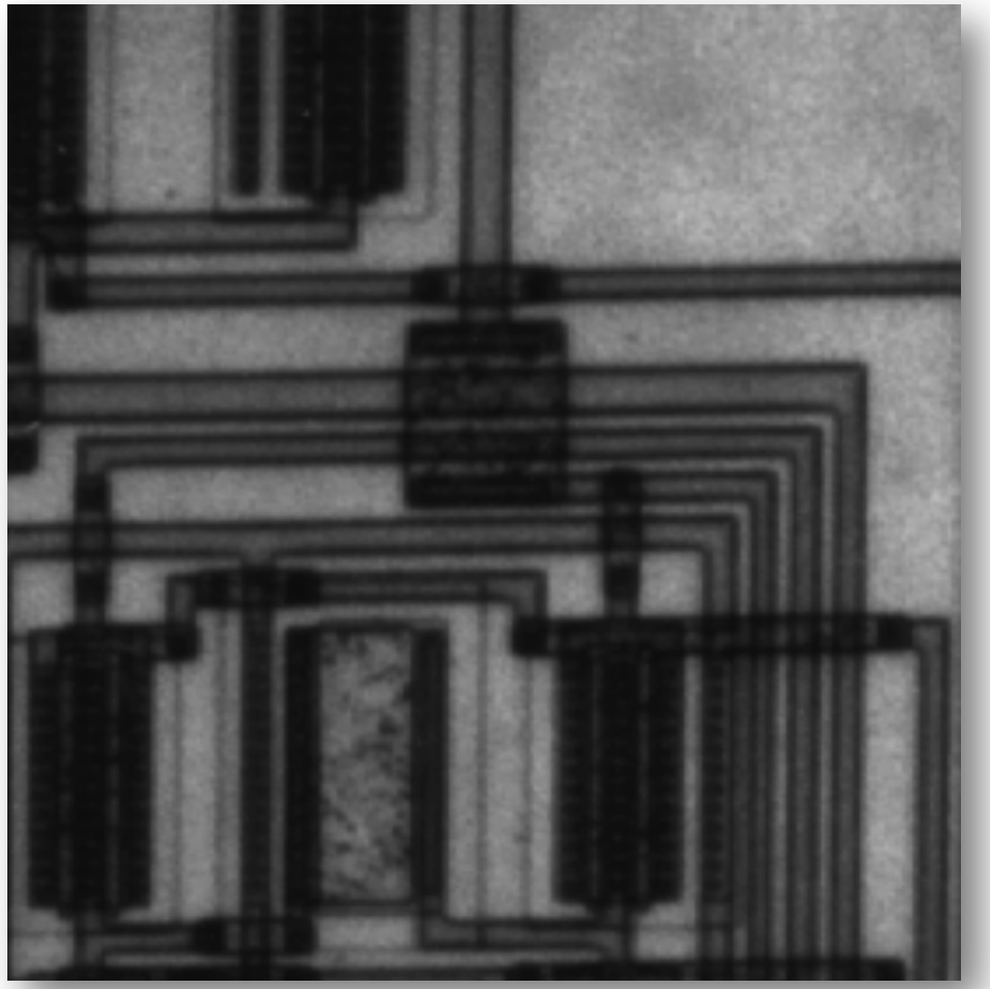


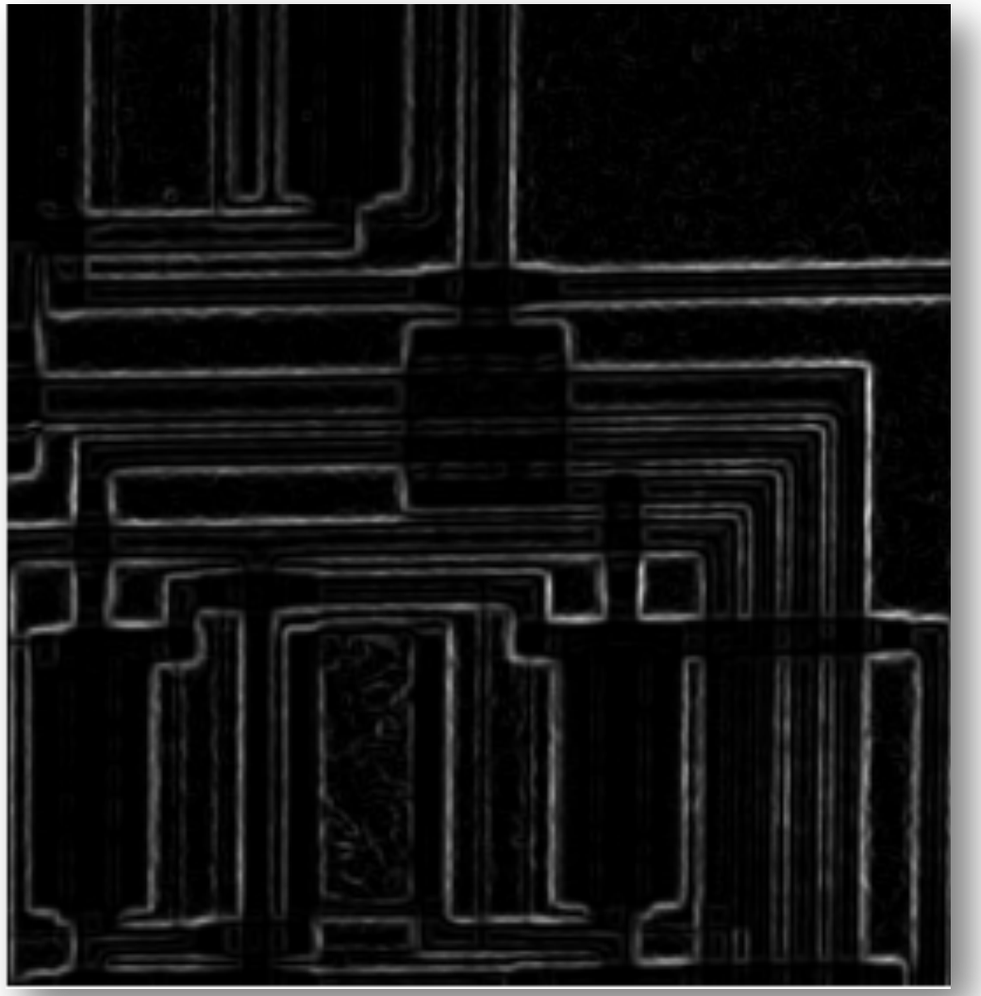
Una vez que obtenemos los bordes, normalizamos la imagen

ver más en: https://scikit-image.org/docs/dev/auto_examples/edges/plot_edge_filter.html#sphx-glr-auto-examples-edges-plot-edge-filter-py

Filtros de gradientes

- Operador de Prewitt



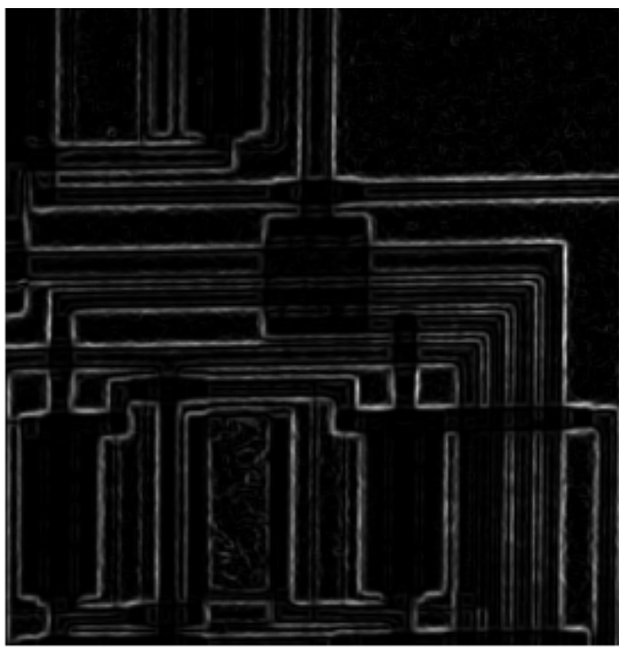


Resultado del operador PREWITT

Filtros de gradientes

- Operador de Prewitt

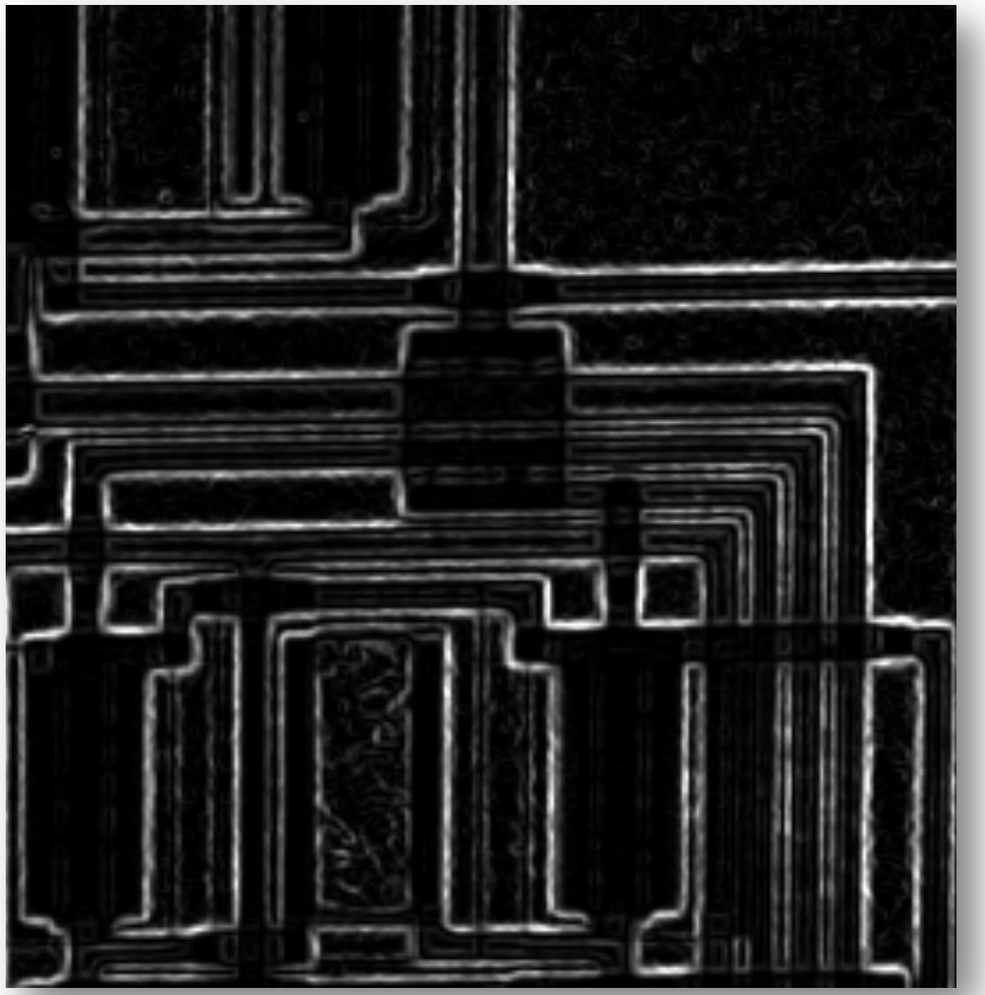
```
import cv2
from skimage import filters

img = cv2.imread('circuit.png')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

edges= filters.prewitt(gray)

out = cv2.normalize(edges.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)
cv2.imshow('Sobel',out)
cv2.waitKey(0)
```



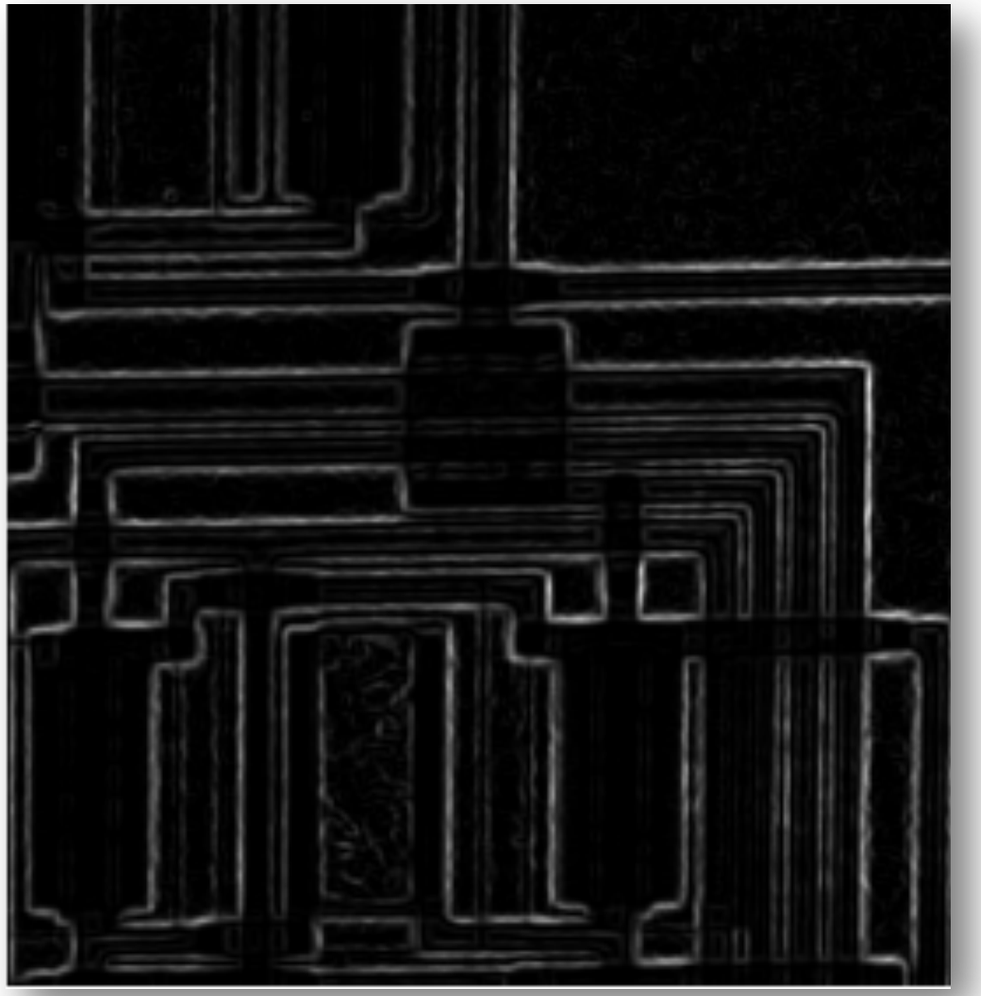
Una vez que obtenemos los bordes, normalizamos la imagen

Filtros de gradientes

- Comparación entre SOBEL y PREWITT



SOBEL



PREWITT

Filtros de gradientes

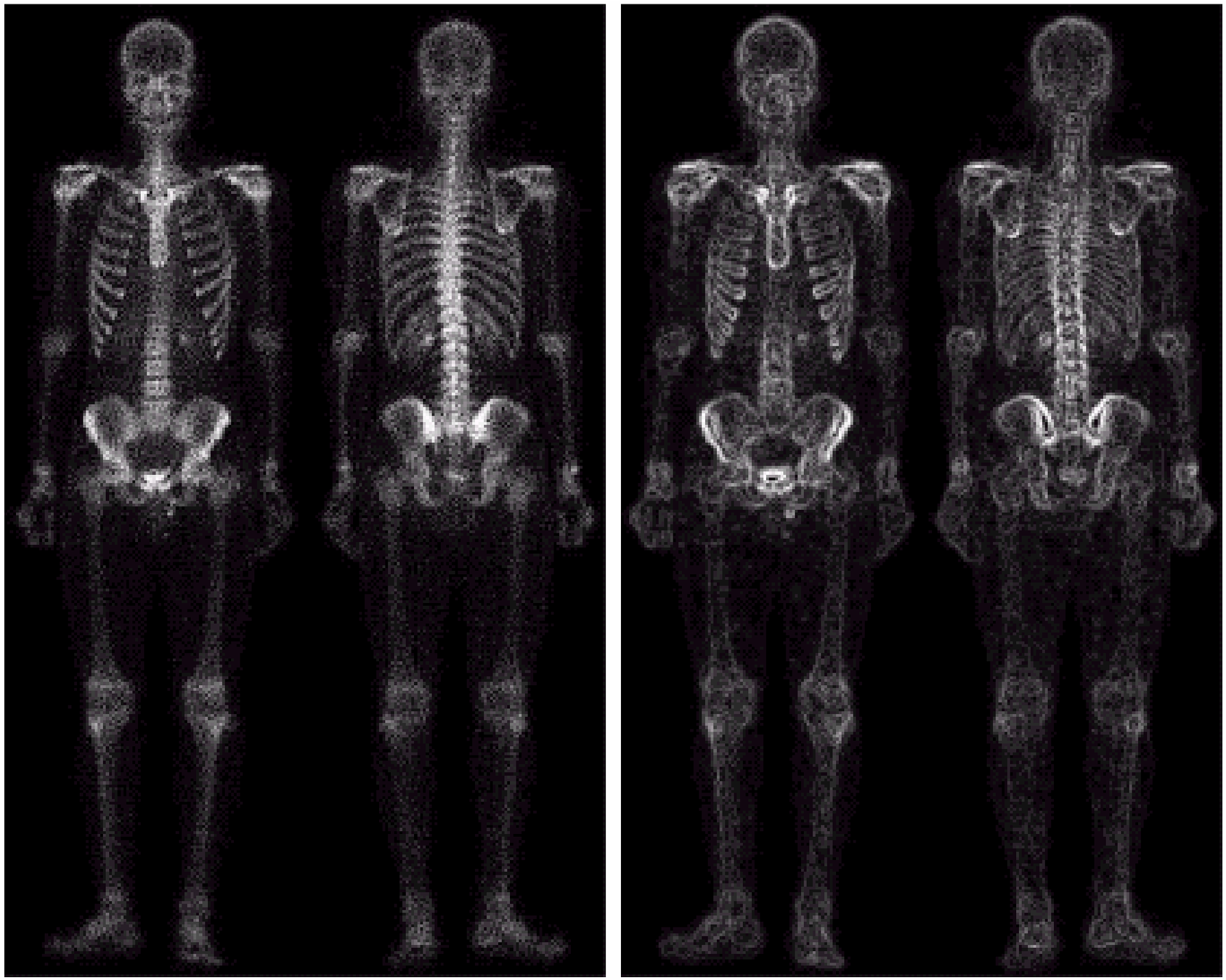
- Ejemplo de un proceso

original

laplaciano

Filtros de gradientes

- Ejemplo de un proceso

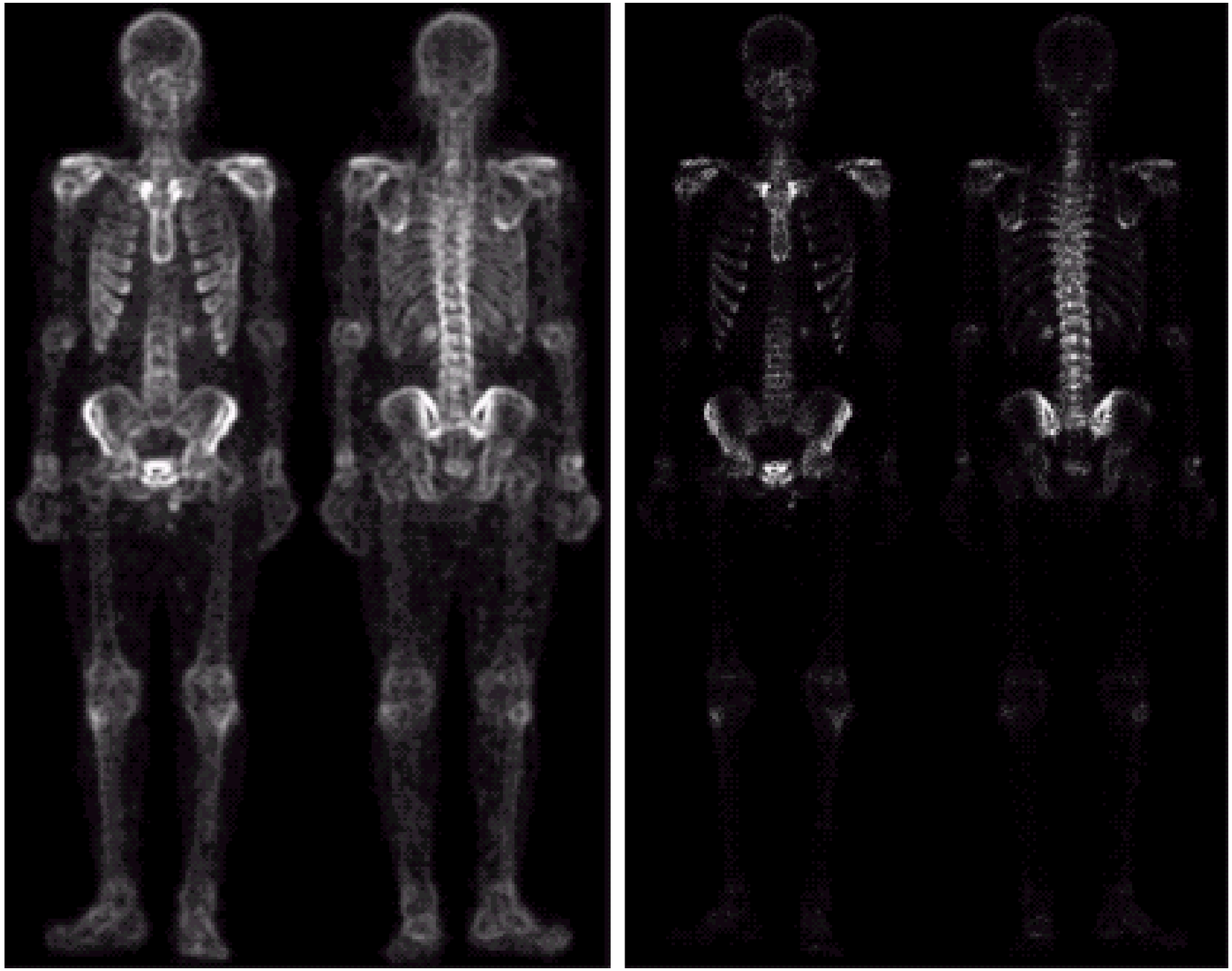


original + laplaciano

sobel

Filtros de gradientes

- Ejemplo de un proceso

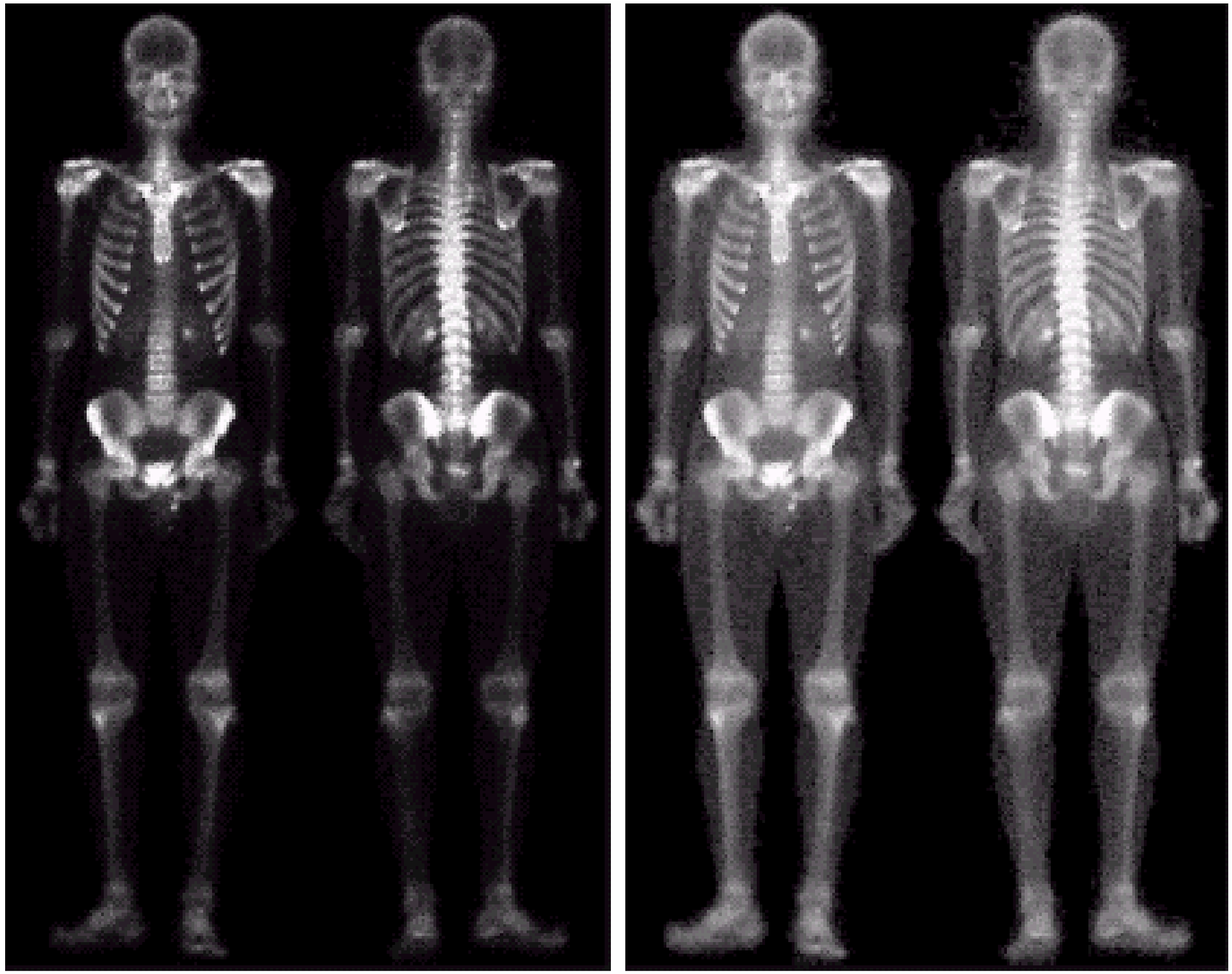


sobel+
promedio

unsharp

Filtros de gradientes

- Ejemplo de un proceso



original + unsharp

gama