Procesamiento Digital de Imágenes

Claudio Delrieux
Laboratorio de Ciencias de las Imágenes – UNS - CONICET
cad@uns.edu.ar

Procesamiento por convolución.

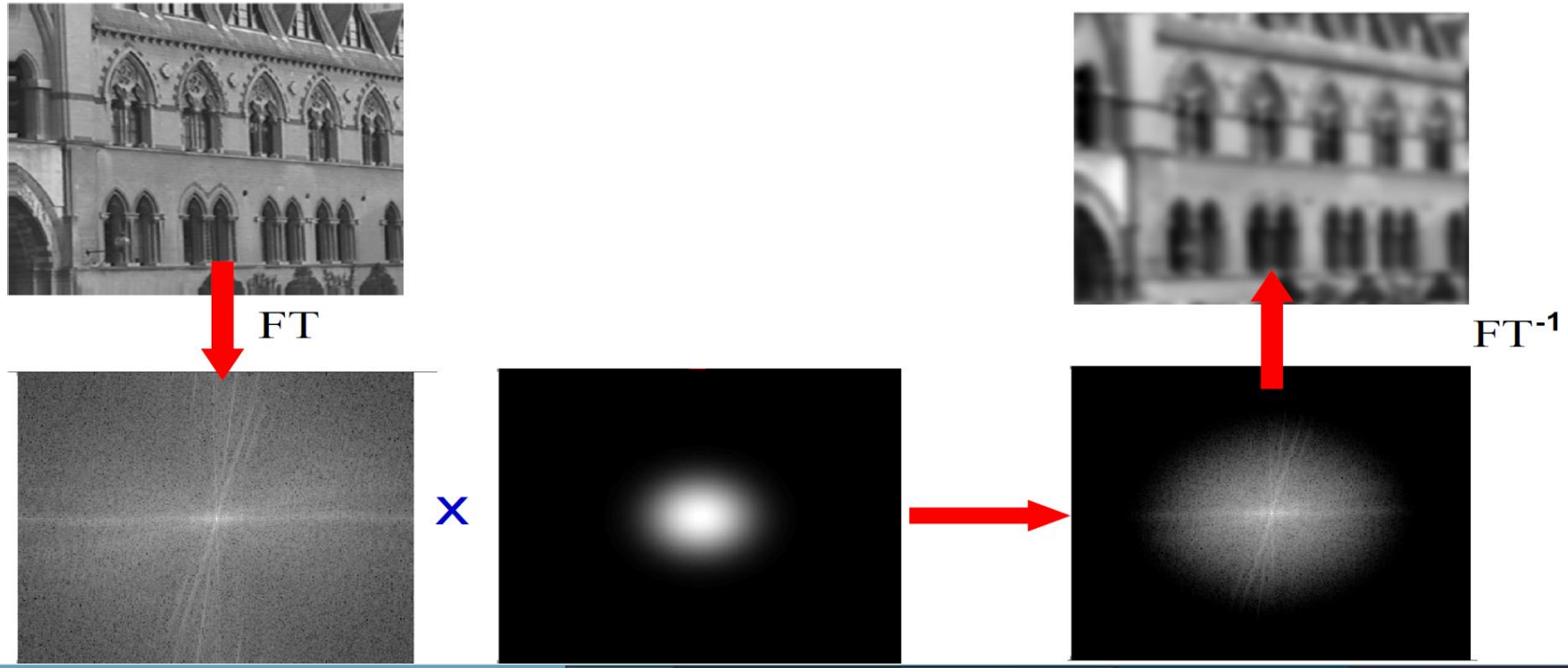
PDI – Procesamiento por convolución

Como vimos en el tema anterior, el procesamiento espectral de imágenes es extremadamente poderoso dado que permite filtrar elementos que en el dominio de los pixels son imposibles de separar, pero en el dominio frecuencial es posible discriminar.

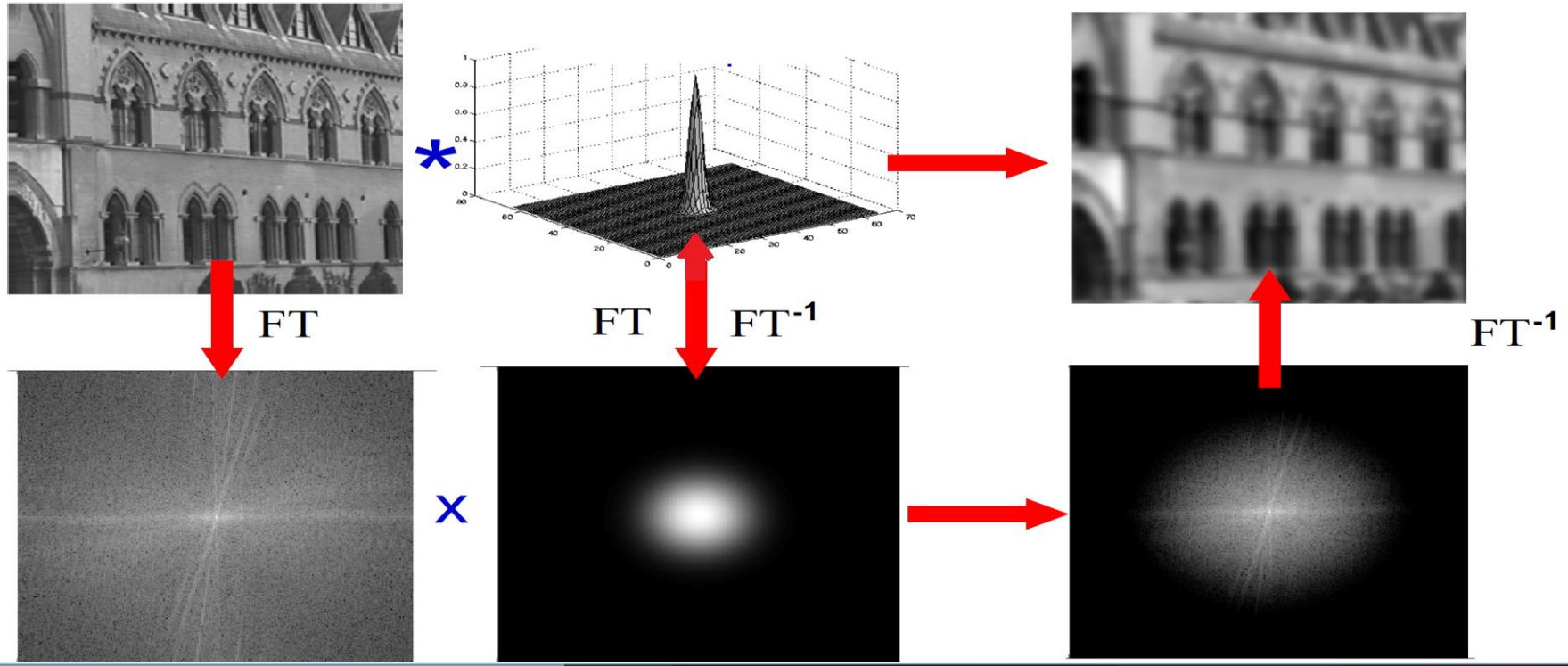
El workflow para dicho procesamiento es el que vimos:

1. Obtener la imagen espectro de la imagen original (por medio de la FFT)
2. Manipular el espectro de acuerdo a algún criterio
3. Generar la imagen resultado a través de la transformada inversa.

PDI – Procesamiento por convolución



PDI – Procesamiento por convolución



PDI – Procesamiento por convolución

Si el procesamiento que realizamos en el paso 2 es, como en casi todos los casos, eliminar la parte no deseada de la imagen original, entonces puede pensarse a dicho paso como un *producto* entre el espectro original y otra "imagen máscara" de unos y ceros que deja pasar la información de las frecuencias deseadas y filtra las indeseadas.

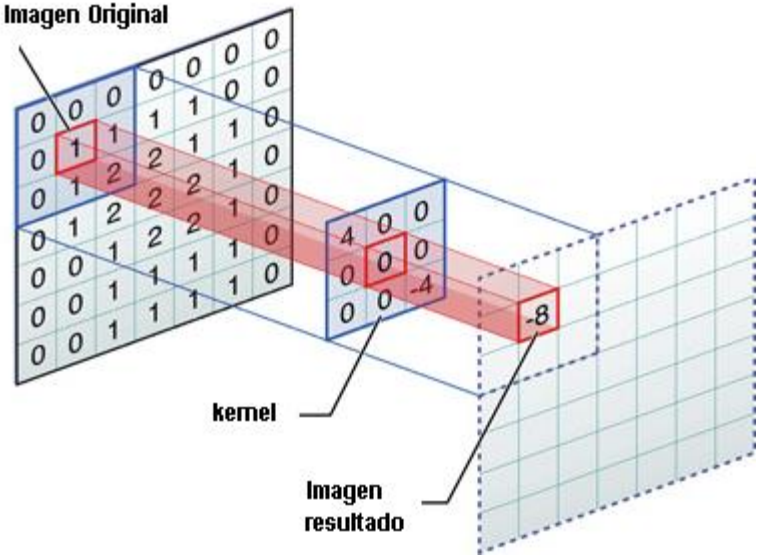
Si recordamos la propiedad de convolución de la TF, este procesamiento (obtener un espectro como producto de otros dos) debe poder obtenerse también a través de la convolución de dos imágenes.

Si la primera imagen a convolucionar es la que queremos procesar, y la segunda (denominada kernel en la bibliografía) es pequeña, esta forma de proceder es más eficiente y rápida. Nace así el procesamiento por convolución.

PDI – Procesamiento por convolución

La convolución entre imágenes (o una imagen y un pequeño *kernel*) puede entenderse con esta imagen:

1. Para cada pixel (i, j) de la imagen original depositamos el centro del *kernel* en dicho pixel.
2. Se multiplica cada pixel de la imagen original por cada celda correlativa del *kernel*.
3. Dichos productos se suman, y el valor final es el pixel correlativo (i, j) en la imagen resultado.



PDI – Procesamiento por convolución

Si bien hay casos excepcionales, en la gran mayoría de los casos los kernel que se utilizan son cuadrados y de tamaño impar, y por lo tanto se “insertan” en su celda central.

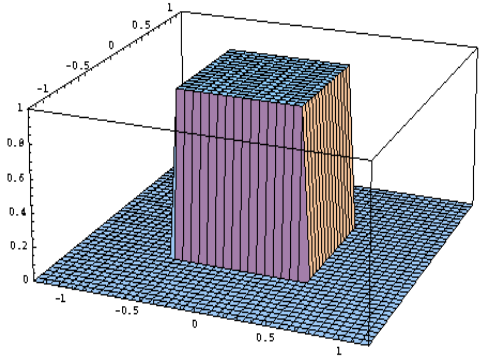
El procesamiento por convolución, por lo tanto, desde el punto de vista computacional es relativamente sencillo: se implementa por medio de un doble ciclo for doble: recorremos la imagen original por (i, j), y para cada pixel (i, j) recorremos el kernel por fila y columna para calcular la suma de productos.

Es, por lo tanto, menos complejo y más sencillo que el procesamiento espectral. Al igual que en dicho caso, veremos aquí convolución de imágenes en niveles de gris, aunque es sencillo llevar los conceptos a YIQ o RGB.

PDI – Procesamiento por convolución

Veamos ahora un caso sencillo de kernel, para interpretar el efecto de su procesamiento. Posiblemente el más sencillo (no trivial) que podamos imaginar sea un kernel de 3x3 con valores constantes (por ahora todos unos).

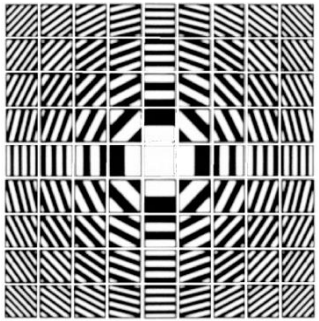
Nuestro kernel ahora es una matriz de 3x3 con unos en todas sus celdas. Si lo pensamos así, su representación es idéntica a nuestro "rectángulo 2D", y por lo tanto el efecto de convolucionar una imagen por dicho kernel será como el de multiplicar la TF de la imagen por la TF del kernel (el cual es la función sinc 2D).



PDI – Procesamiento por convolución

Antes de proseguir vamos a hacer un poco de matemática que nos permita afianzar los conceptos y hacer pie con lo que está por ocurrir.

El primer concepto que nos conviene entender es, cuál es la escala de la TF de una imagen? En nuestra cartografía hablamos de que movernos un pixel fuera del centro representaba una oscilación por pantalla, y así sucesivamente. Por ejemplo, en una imagen de 101x101, su TF será también de 101x101, y por lo tanto habrán 50 pixels arriba, abajo, a la izq. y a la der. del pixel central, y por lo tanto, las frecuencias máximas serán 50 oscilaciones por pantalla (horizontales o verticales).



PDI – Procesamiento por convolución

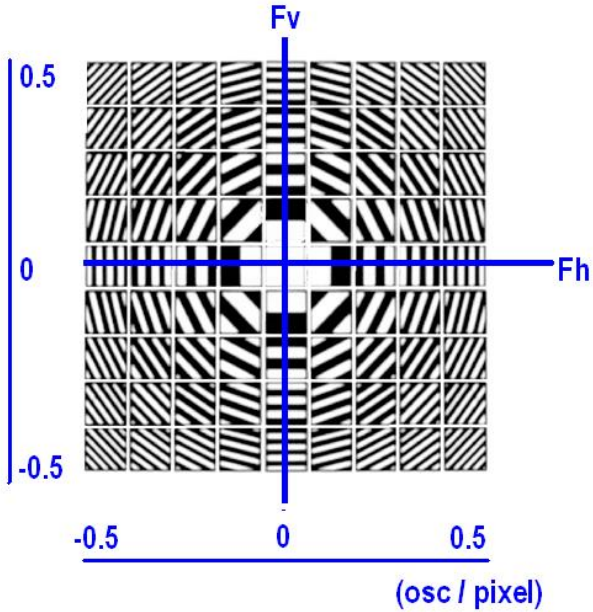
Si la imagen fuese de 1001x1001, entonces la máxima frecuencia sería 500 oscilaciones por pantalla, etc. Es decir, no tenemos una escala uniforme, dado que los valores de las frecuencias en la imagen TF dependen de la resolución de la imagen original.

Sin embargo, si lo pensamos un segundo, vemos que hay una relación entre la máxima frecuencia y el tamaño de la imagen (la primera es la mitad de la segunda). Es decir, la máxima frecuencia es *siempre 0.5 oscilaciones por pixel*, o bien una oscilación cada dos pixel.

PDI – Procesamiento por convolución

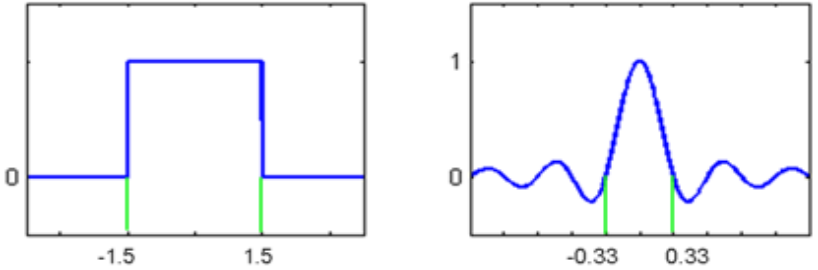
Esto tiene sentido: lo más pequeño que pudiésemos representar en una imagen es un pixel negro, luego uno blanco, etc., y por lo tanto la distancia entre picos o valles es 2.

De esa manera, podemos representarnos el rango de frecuencias de la TF de una imagen en *oscilaciones por pixel*, sabiendo que en cualquier caso los valores estarán entre -0.5 y 0.5.



PDI – Procesamiento por convolución

Volvamos ahora a nuestra amiga la función sinc. La teoría predice que la TF de una sinc de ancho 3 tendrá los cruces por cero a frecuencias 1/3.



Es decir, nuestro filtrado dejará pasar mayoritariamente las frecuencias menores a 1/3 y filtrará mayoritariamente las mayores. Frecuencia 1/3 representa una oscilación cada tres pixels. Es decir, el resultado será un flitro *pasa bajo*.

Y esto es así con nuestro kernel con todos unos, salvo por un pequeño detalle.

PDI – Procesamiento por convolución

El problema que tenemos ahora es que un kernel con todos unos actualmente multiplicará la luminancia por tantos valores como tenga el kernel. En nuestro caso, si un pixel tiene un valor de luminancia 1 y así ocurre con sus vecinos, el resultado del procesamiento por convolución producirá en la imagen resultado un pixel de luminancia 9.

Si deseamos que las luminancias no se multipliquen, tenemos que ponderar los valores dentro del kernel por un factor inversamente proporcional a la cantidad de celdas del kernel. En nuestro caso, el kernel tendrá todos valores 1/9.

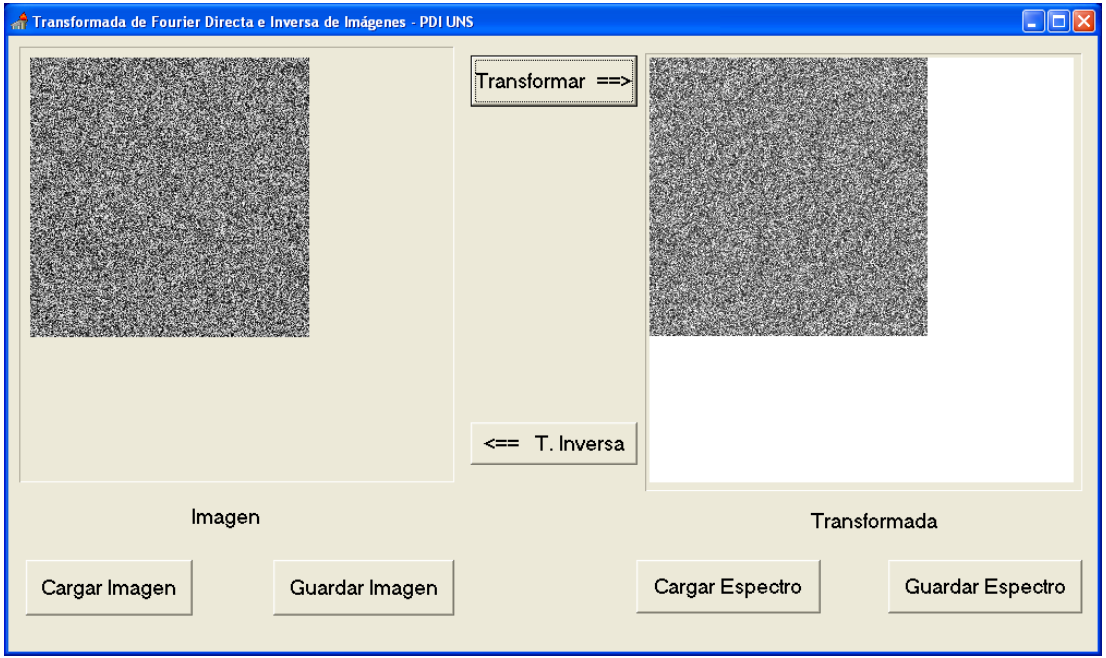
Ver que de esa manera, el filtrado que estamos haciendo es el *promedio* de un pixel con sus vecinos.

PDI – Procesamiento por convolución

Vemos en otra aplicación de referencia el efecto de aplicar este procesamiento sobre una imagen fotográfica. Se observa que el efecto se puede caracterizar como un borroneo o difuminado (blur). Los detalles finos se pierden, las aristas dejan de ser tan claras, etc.

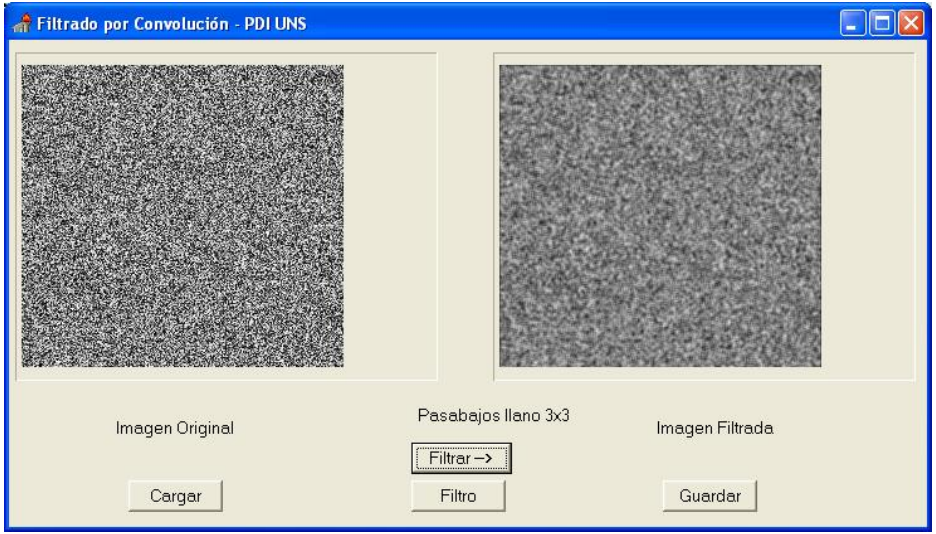
PDI – Procesamiento por convolución

Cómo estar seguros que estos resultados coinciden con la teoria? Vemos la TF de una imagen con “ruido blanco” (cada pixel tiene una luminancia aleatoria independiente de los demás) y vemos que su TF es similar (es otro caso de imagen invariante frente a TF).



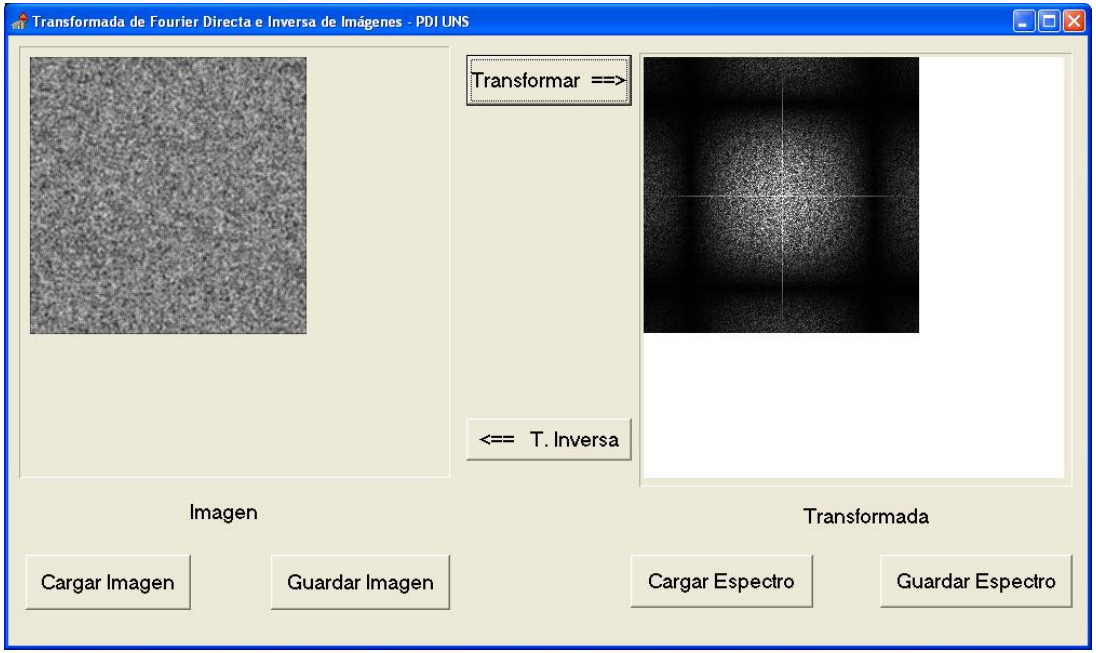
PDI – Procesamiento por convolución

Volvemos al aplicativo que computa convoluciones, y convolucionamos esa imagen, obteniendo una versión filtrada del ruido blanco.



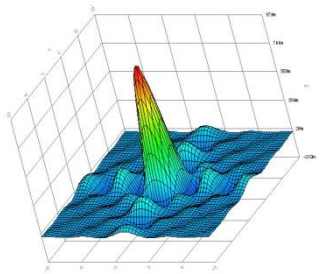
PDI – Procesamiento por convolución

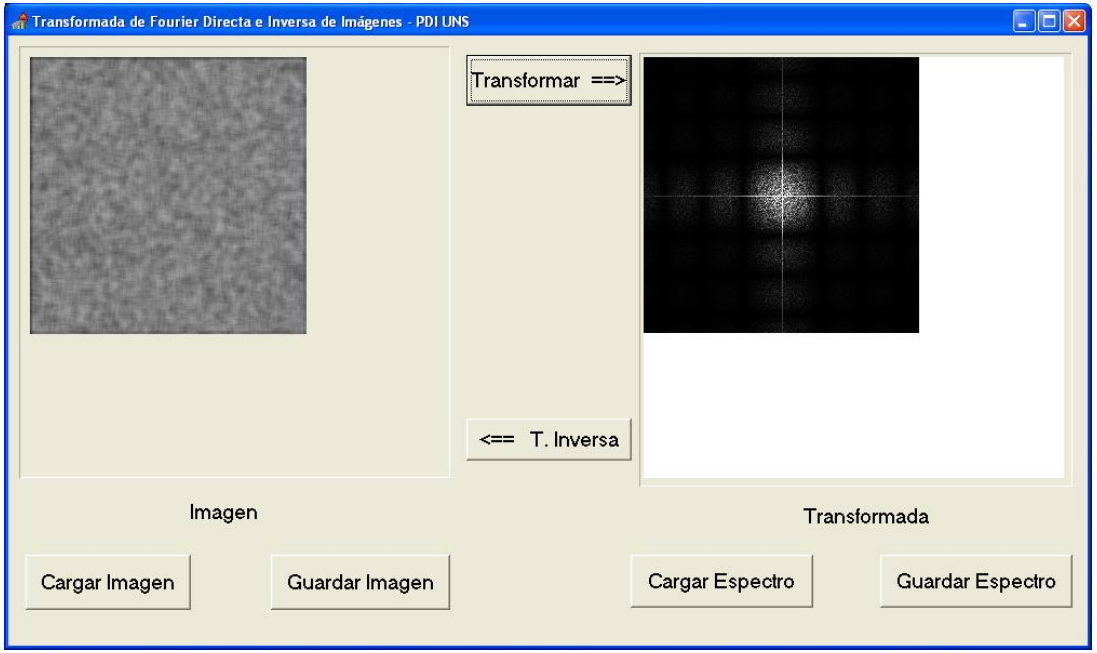
Cargamos esta imagen y encontramos su TF con el otro utilitario, y vemos que, en efecto, alrededor de la zona de frecuencias 1/3 toda la energía ha desaparecido (hay un poco de energía a frecuencias mayores, porque la sinc no cae a cero sino que sigue siendo no nula luego de este primer cero).



PDI – Procesamiento por convolución

Vemos un ejemplo más, con un kernel promediador de 7x7 (49 celdas con valor 1/49 cada una). La TF de dicha imagen pasa a parecerse cada vez más a la del producto seprarable de sincs, en este caso de un cuadrado de 7x7,

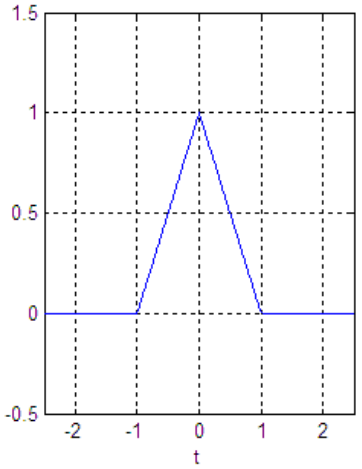


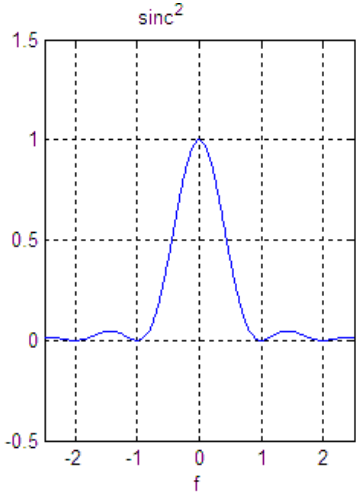


PDI – Más y mejores filtros por convolución

Si por ejemplo convolucionamos el pulso consigo mismo, el resultado es una función lineal a trozos o función triangular (recordar la animación en el artículo de wikipedia: https://en.wikipedia.org/wiki/Convolution#/media/File:Convolution_of_box_signal_with_itself2.gif).

La TF del pulso consigo mismo es el producto de la sinc consigo misma, la cual tiene mucho mejor calidad como pasabajos.





PDI – Pasabajos Bartlett

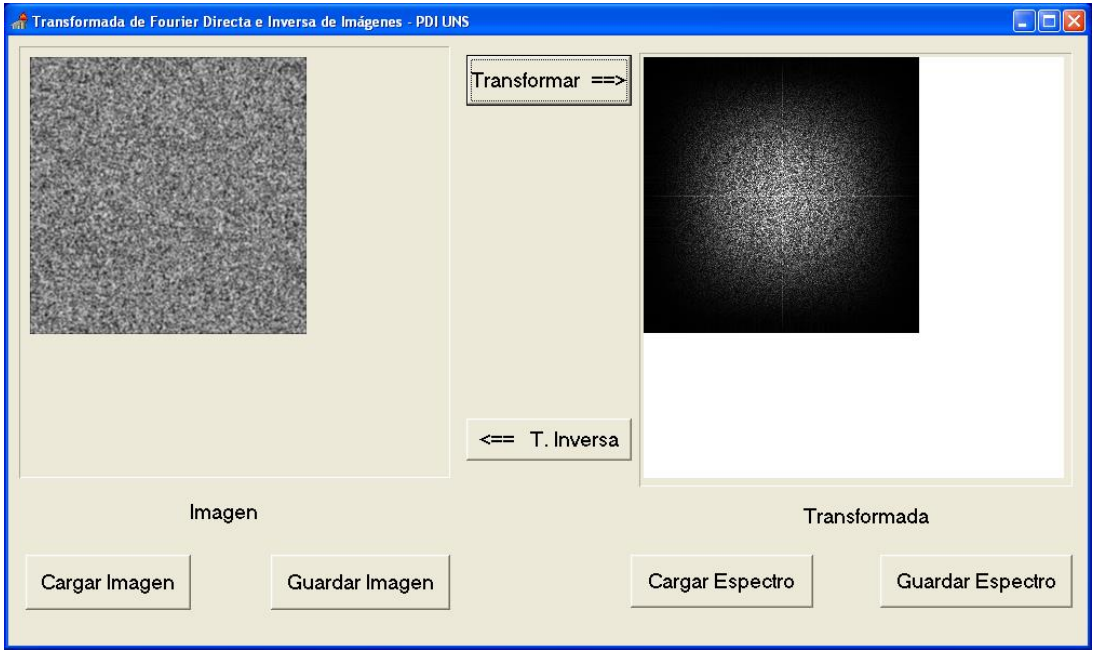
La forma más sencilla de implementar filtros de Bartlett consiste en armar una secuencia creciente hasta el centro y luego decreciente, y luego decreciente (1-2-1 para kernel de tamaño 3, 1-2-3-2-1 para kernel de tamaño 5, etc.) y luego armar el kernel multiplicando por fila y por columna dicha secuencia (se muestra el caso 3x3). Luego, para tener ganancia 1 como corresponde a un pasabajos, hay que dividir por la sumatoria de los coeficientes (en este caso 16).

1	2	1
2	4	2
1	2	1

Luego, el kernel Bartlett 3x3 queda 1/16, 2/16, 1/16, etc.

PDI – Pasabajos Bartlett

Probamos ahora filtrar nuestro ruido blanco con este filtro y ver su espectro, para constatar que, en efecto, la cantidad de energía en la zona de paso es mayor, y en la zona de rechazo es menor.



PDI – Filtro Gaussiano

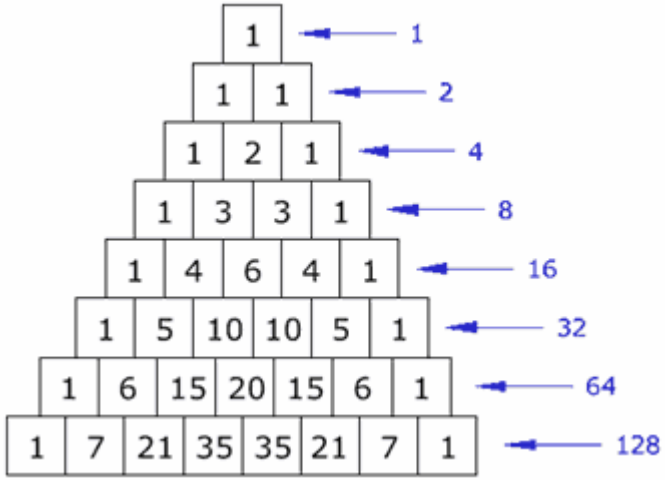
Por qué detenernos en una sola convolución del pulso consigo mismo? Por qué no hacerlo 10, 20 veces e ir mejorando cada vez más la calidad del filtro pasabajos?

Un corolario del teorema del límite central es que la convolución repetida del pulso consigo mismo tiende a la función Gaussiana. Por dicha razón, el kernel Gaussiano es el estandar de facto en filtrado pasabajos.

Una manera práctica de implementar kernels Gaussianos es aprovechar la relación que tiene el “triángulo de Pascal” con la función Gaussiana.

PDI – Filtro Gaussiano

En la figura vemos algunas filas de dicho triángulo y la suma total de sus coeficientes (ya veremos por qué).



Si vemos las filas con cantidad impar de elementos, vemos que la tercera coincide con Bartlett, así que veamos la quinta: 1-4-6-4-1.

Aplicando la misma idea que con Bartlett, armamos una matriz multiplicando esos coeficientes por fila y por columna.

PDI – Filtro Gaussiano

Tenemos luego que normalizar para obtener ganancia 1, por lo que hay que dividir dichos coeficientes por su sumatoria. Como la suma en la fila era igual a 16, y la matriz tiene el producto de dos de dichos vectores, el coeficiente de normalización es 256.

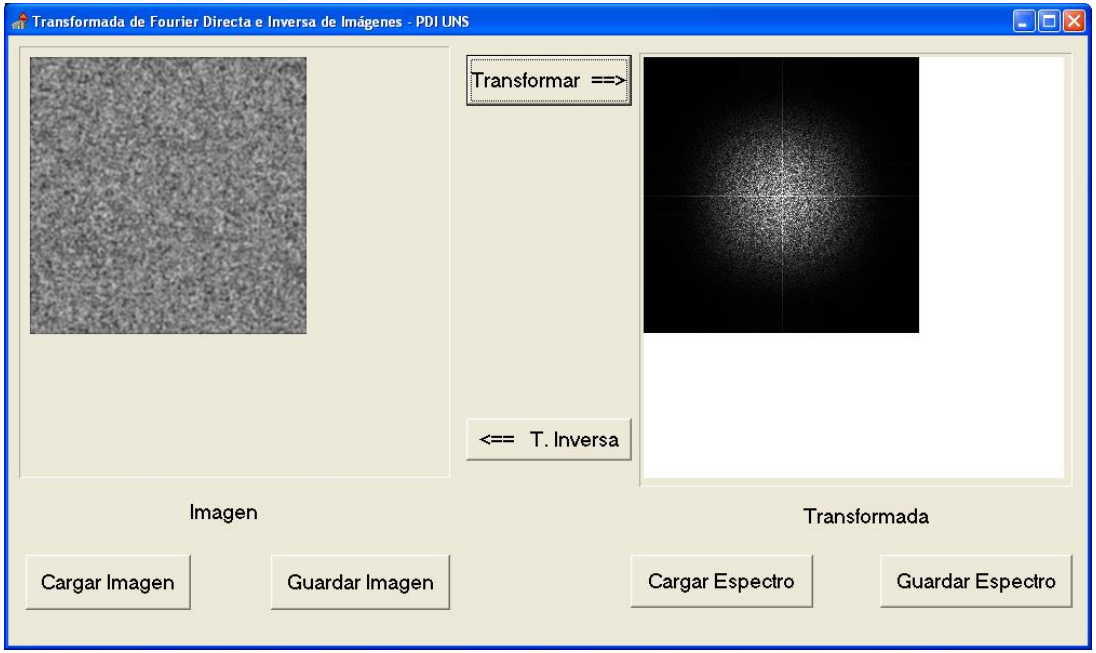
$$\frac{1}{256}\cdot\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix} = \frac{1}{256}\cdot\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix}\cdot\begin{bmatrix}1 & 4 & 6 & 4 & 1\end{bmatrix}$$

Variantes de este filtro son utilizadas multitudinariamente por todas las aplicaciones de procesamiento de imágenes y video.

PDI – Filtro Gaussiano

Vemos un ejemplo del filtrado Gaussiano en nuestro ruido blanco. Recordemos que como el kernel es de 5x5, la frecuencia de corte es 0.2.

Un dato importante: la TF de la Gaussiana es también una Gaussiana!!



PDI – Filtro Gaussiano

También para comparar, vemos en esta imagen el resultado de aplicar nuestro filtro Gaussiano a la imagen fotográfica. Si bien a través de este documento es difícil que se aprecien las sutiles diferencias con los primeros filtrados, el hecho es que si bien la imagen filtrada no posee alta frecuencia, igual sigue siendo mucho más inteligible y menos deteriorada que con el filtrado promedio o el Bartlett.

PDI – Filtros Pasa-altos y detección de bordes

Si existe un pasabajos, entonces seguramente existirá y será útil el pasa*altos*. Este será un filtro que eliminará en la imagen toda la energía excepto aquella que esté en frecuencias altas (es decir, cambios de luminancia muy locales).

Como sabemos, la mayor parte de la energía de las imágenes está en la zona baja del espectro, así que el filtrado pasaaltos retiene muy poca cantidad de energía.

El filtro más utilizado para estos fines es el Laplaciano.
https://en.wikipedia.org/wiki/Discrete_Laplace_operator.

PDI – Filtrado Laplaciano

El Laplaciano (discreto) en 1D queda expresada como la diferencia finita entre dos diferencias finitas (o también, *segundas diferencias*). En 2D queda según esta ecuación:

$$\Delta f(x,y) \approx \frac{f(x-h,y) + f(x+h,y) + f(x,y-h) + f(x,y+h) - 4f(x,y)}{h^2}$$

En pocas palabras, utilizamos el pixel central y sus 4 vecinos por fila y columna, o bien sus 8 vecinos. Veamos que la ganancia de un pasaaltos debe ser cero, por lo que la suma de todos los coeficientes del kernel debe ser cero.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

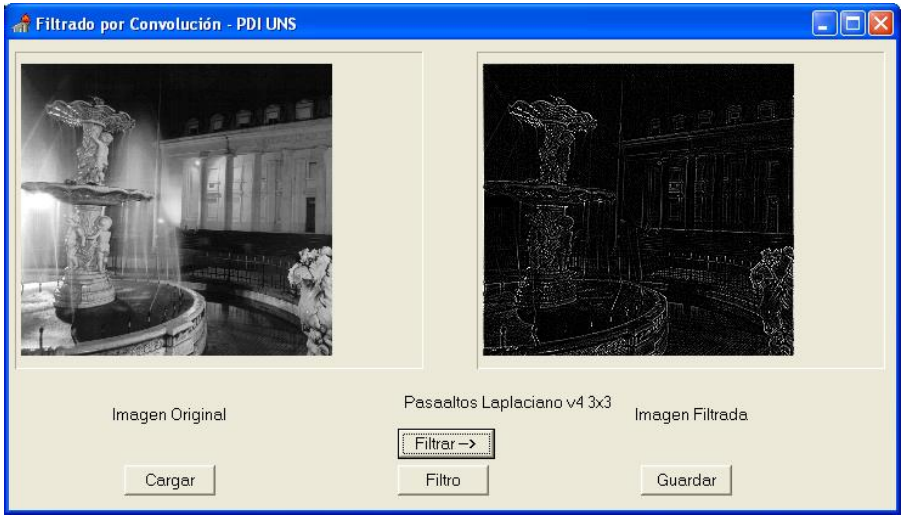
PDI – Filtrado Laplaciano

Uno de los problemas con los que nos enfrentamos al implementar el Laplaciano es que en el pixel resultado podemos toparnos con valores mayores que 1 o menores que 0, al aplicar Laplaciano a un pixel de luminancia alta rodeado de pixels de luminancia baja o a la inversa, respectivamente.

Para ello tenemos que recurrir a nuestras funciones de coerción que vimos en el segundo tema de la primera clase, es decir, luego de aplicar la suma de productos, si la luminancia es menor que cero, se coerciona a cero, o si es mayor que uno se coerciona a uno.

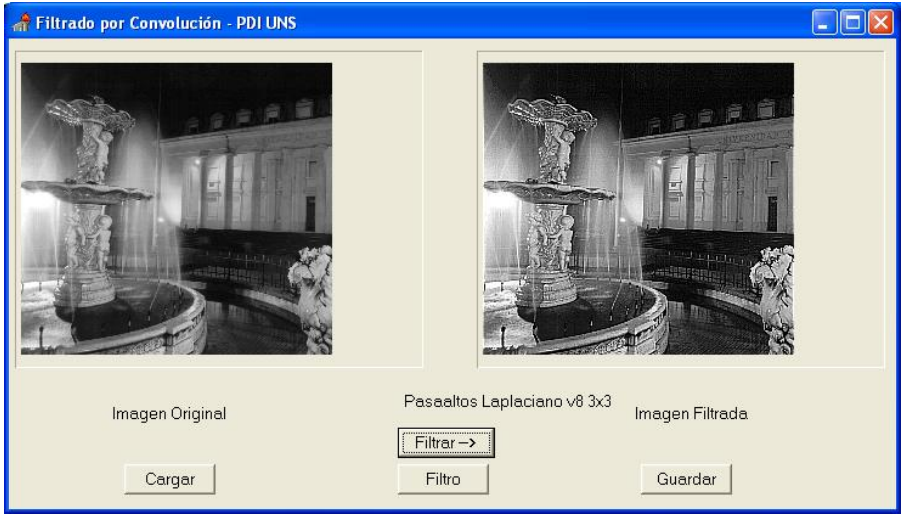
PDI – Filtrado Laplaciano

Volvemos a nuestra anterior imagen fotográfica y vemos que el resultado de aplicar el Laplaciano (en este caso con 4 vecinos) selecciona la parte de la imagen donde se concentra el mayor detalle que era borroneado por el filtro pasabajos. Estos detalles están en general en las aristas o bordes de los objetos.



PDI – Filtro de mejora

Otra de las propiedades sobresalientes del filtrado por convolución es la posibilidad de sumar dos o más kernel y de esa forma computar en un solo paso operaciones más complejas. En este ejemplo le sumamos a la imagen original un 20% de filtro Laplaciano (aumentamos artificialmente la intensidad de la alta frecuencia), lo cual vuelve a la imagen mucho más vívida. Esto se logra sumando el kernel identidad al Laplaciano multiplicado por 0.2.



PDI – Filtros direccionales

Una característica más de la convolución es que su cómputo permite encontrar correlaciones geométricas con partes de la imagen. Por ejemplo, un kernel que localmente se parezca a una arista vertical generará correlaciones más altas con cada parte de la imagen que localmente se parezcan a aristas verticales.

Nacen así los filtros direccionales. Vemos los kernels de Sobel, que permiten computar el gradiente horizontal (cuando la imagen transiciona localmente de baja a alta luminancia de izquierda a derecha) y vertical (de abajo a arriba).

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

Gx

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

Gy

PDI – Filtros direccionales

El primer kernel nos permite detectar aristas que “miren al oeste” si el objeto es negro sobre fondo blanco (y al este a la inversa), mientras que el segundo detecta aristas que miren “al sur” si el objeto es negro sobre blanco.

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

Gx

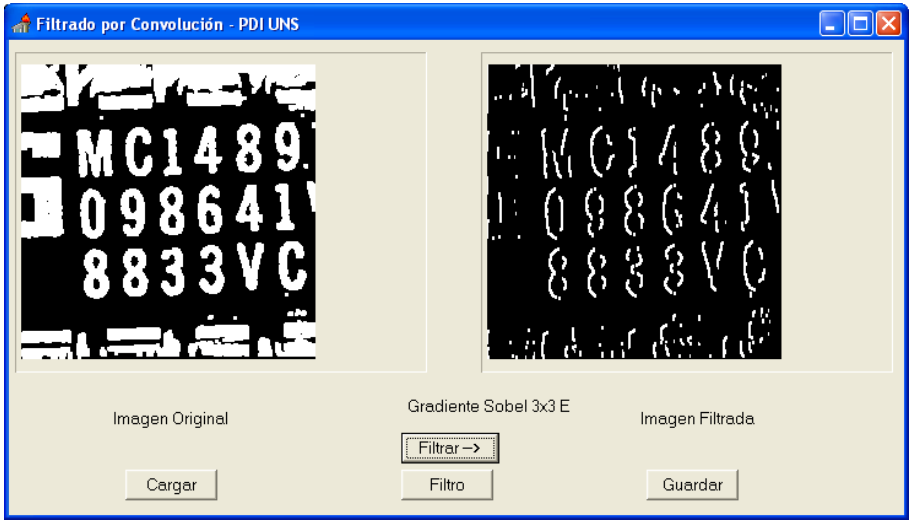
+1	+2	+1
0	0	0
-1	-2	-1

Gy

Es fácil ver que ambos kernels son la rotación 90 grados uno del otro, por lo que se pueden encontrar fácilmente los kernels de Sobel para las 8 orientaciones (es decir, se pueden encontrar aristas que apunten en diagonal).

PDI – Filtros direccionales

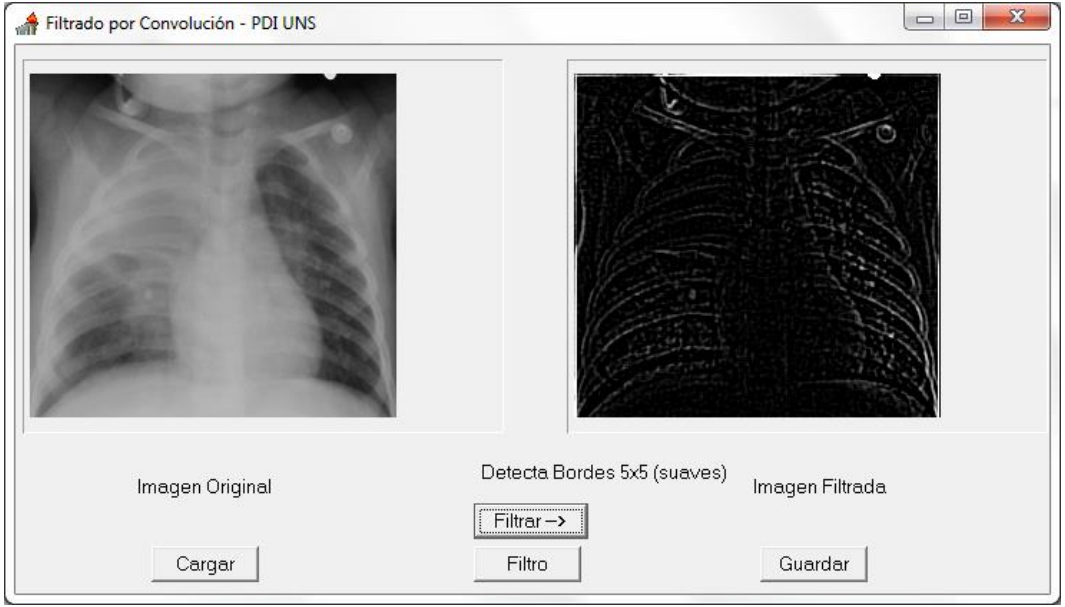
Vemos en este ejemplo una imagen y su filtrado por Sobel “Este”. Se puede observar cómo esta última retiene las aristas que apuntan “hacia el Oeste” (dado que la figura está en blanco y el fondo en negro, o sea, se invierte la orientación del filtro.



PDI – Pasa-altos de diferentes frecuencias de corte

Los pasaaltos que vimos (Laplaciano con kernel de 3x3) tienen una frecuencia de corte 0,33, la cual en general es muy alta para determinado tipo de imágenes.

Un kernel de 5x5 permite adoptar dos frecuencias de corte posibles (0,2 y 0,4).



PDI – Pasa-banda

Vimos hasta ahora (de manera más o menos implícita) dos formas de diseñar kernels:

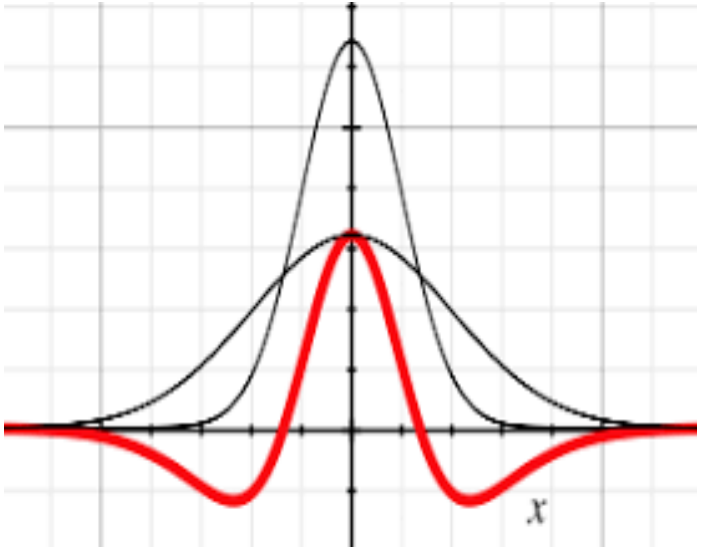
1. A través de su DFT, viendo la respuesta en frecuencia como combinación de la función constante y de cosenos de F/3 (para kernel 3x3) o 2F/5 y 4F/5 (para kernel 5x5, etc.)
2. Aprovechando la propiedad asociativa de la convolución, convolucionando el pulso consigo mismo (Bartlett, Gaussiano).

Ahora aprovechamos otra propiedad de la convolución, y es su linealidad. Por lo tanto, la suma o resta de kernels producirá suma o resta de frecuencias en el espacio transformado. Eso permite diseñar por ejemplo un filtro pasabanda como la diferencia entre dos pasabajos, típicamente el filtro DOG.

PDI – Difference of Gaussians

El kernel de diferencia entre Gaussianas puede obtenerse simplemente restando el kernel 5x5 al kernel 3x3. Ver que ambas áreas (ganancias) son iguales, por lo que su valor en el (0,0) es diferente.

Sin embargo, en frecuencia ambas Gaussianas tienen el mismo alto en la frecuencia 0, por lo que su resta no deja pasar bajas frecuencias. Por dicha razón es que este filtro es pasabanda.



PDI – Actividad práctica

Implementar un aplicativo o notebook que levante una imagen en nivel de gris (o que la convierta a YIQ y retenga solo el Y) y aplique el filtrado por convolución aquí visto:

1. Pasabajos: Plano, Bartlett 3x3, Gaussiano 5x5.
2. Detectores de bordes: Laplaciano v4, Sobel 4 orientaciones.
3. Pasabanda.

Respecto del “cierre” de la imagen cerca de los bordes, implementar la técnica no matemáticamente perfecta de “repetir” artificialmente filas o columnas cuando estas hagan falta. Por ejemplo, si el kernel es 3x3 y estoy en la primera columna, y no hay ninguna columna a la izquierda, entonces repito la primera columna como si estuviese en la imagen original.