Práctica 4: Operadores locales I		Grupo	
		Puesto	
Apellidos, nombre		**Fecha**	
Apellidos, nombre			

El objetivo de esta práctica es presentar al alumno las técnicas para realizar operadores locales sobre imágenes en escala de grises.

Desarrolle cada ejercicio en un fichero de comandos 'ejercicio_X.m' separado. Para conocer el funcionamiento preciso de los comandos que se introducen en este guión, utilice la ayuda de MATLAB. Para evitar posibles interferencias con otras variables o ventanas recuerde incluir siempre las instrucciones **`clear all`** y **`close all`** al principio de cada fichero de comandos.

Justo antes de finalizar la práctica, comprima el guión y los ficheros '.m' generados en un único fichero con el nombre '**Practica_4_ApellidosNombre.zip**', conéctese al sistema de entrega de prácticas de la Intranet y entréguelo en el grupo de prácticas (grupo único).

1 Aproximaciones a la realización de operadores locales

La aplicación de un operador sobre una imagen que denominaremos *original* ($\psi[n,m]$), da lugar a una nueva imagen que denominaremos *procesada* ($\theta[n,m]$). Un operador local toma como entrada el valor de un entorno de píxeles en el entorno de uno dado y general un valor resultante en el píxel homólogo de la imagen resultante. Por simplicidad asumiremos que el entorno es rectangular, de dimensiones $M'xN'$ según indica la siguiente figura:

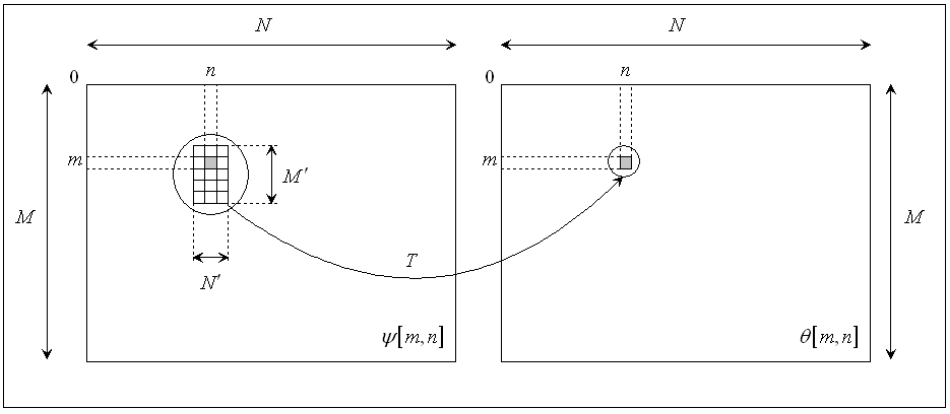

Figura 1: Esquema genérico de aplicación de un operador local

Según se ha visto en la parte teórica de la asignatura, si la operación que se efectúa sobre los píxeles del entorno es lineal, aplica la Teoría de Sistemas Lineales multidimensionales. MatLab ofrece herramientas específicas para esta situación. Sin embargo, si la operación no es lineal, no existe una herramienta que permita obtener el resultado de una manera genérica. En este caso, MatLab ofrece herramientas para aplicar los operadores no lineales más usuales.

1.1 Ejercicio 1: filtrado lineal

Si un operador local efectúa una transformación lineal, su comportamiento puede definirse completamente por su respuesta al impulso, $h[n,m]$. Según se ha visto en las clases de teoría, si la

respuesta al impulso es rectangular y simétrica respecto de su origen (para lo cual M' y N' han de ser impares) su aplicación sobre la imagen (es decir, la operación de convolución) puede efectuarse de manera sencilla aplicando la máscara $w[n,m] = h[-n,-m]$ sobre cada píxel de la imagen original, según muestra la Figura 2.

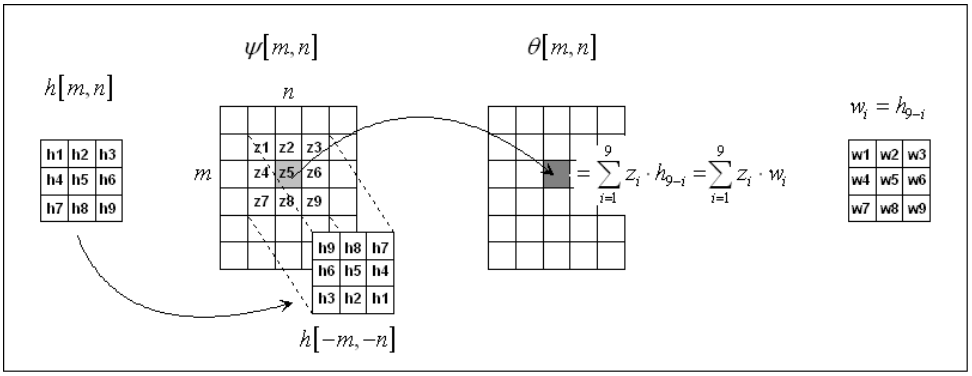

Figura 2: Esquema de aplicación de un operador local lineal

MatLab ofrece la función **`imfilter`** para llevar a cabo esta operación. Su modo de operación por defecto toma como parámetro la imagen original, **`ima`** y la máscara **`mask`** (una matriz de $M'xN'$) y arroja la imagen resultante, **`ima_res`** :

```
>> ima_res=imfilter(ima,mask);
```

La imagen **`ima_res`** resultante es del mismo tipo que la imagen original (**`double`**, **`uint8`**, **`uint16`**, etc.). Para evitar problemas de truncamiento, si la aplicación de la máscara hace que los píxeles de la imagen procesada puedan tener un rango de valores mayor que los de la imagen original, es conveniente convertir la imagen a **`double`** antes de operar. Si no, no es necesario.

Para comprobar su funcionamiento, cargue y visualice la imagen en escala de grises **`edificio_bw_512.bmp`**[1] (salvo que se le indique lo contrario, visualice todas las imágenes en figuras independientes y a tamaño real utilizando **`imshow`** con la opción **`'InitialMagnification',100`**). Defina a continuación la siguiente máscara:

$$\mathbf{W} = \frac{1}{C} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 6 & 4 & 1 & 4 & 6 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A partir del modo de aplicación mostrado en la Figura 2, indique el máximo valor, C, que puede arrojar este operador para una imagen con valores normalizados (valores en $[0,1]$):

$C =$

Su inverso es el valor que ha de ponderar al operador para garantizar que no se modifica el rango dinámico de la imagen.

Aplique esta máscara y la máscara rotada 90º (es decir $\mathbf{W}^{\mathbf{T}}$) sobre la imagen dada y visualice simultáneamente las dos imágenes procesadas. Indique de forma cualitativa el efecto que produce cada una de estas dos máscaras:

[1] Descárguese esta imagen de la página *web* de la asignatura.

Observe las bandas oscuras que aparecen en algunos extremos de las imágenes procesadas. Se deben al modo en que la función **`imfilter`** calcula el resultado cuando la máscara se aplica cerca de los extremos de la imagen: el comportamiento por defecto consiste en considerar que los píxeles que *faltan* tienen valor nulo. Explique, siendo así, la presencia y ausencia de las bandas oscuras en cada caso y su mayor o menor oscuridad (incluya los diagramas que considere necesario):

Pruebe los otros tres modos predefinidos de operar en los extremos que tiene esta función, observe los resultados e indique cuál le parece el más adecuado para esta máscara y porqué:

1.2 Ejercicio 2: filtrado no lineal

Este ejercicio pretende mostrar en detalle el modo de aplicación de un operador local. El objetivo es desarrollar una función MatLab (denomínela **`imfilter_no_lineal_X`**) que tome como parámetros de entrada una imagen y las dimensiones del entorno (M', N', ambos impares) en que ha de aplicarse el operador. La función deberá tener en cuenta:

- Se supone que punto de aplicación del operador es su centro.
- Para evitar los efectos del *padding* en los extremos de la imagen, la función aplicará la operación sólo en los píxeles que sea posible, es decir, devolverá una imagen procesada más pequeña que la original.
- La imagen procesada será del mismo tipo que la original (es decir, **`unit8`**).
- El rango dinámico de la imagen procesada estará ajustado a $[0, 255]$. Para ello considere el proceso de desplazamiento y escalado descrito en el ejercicio 2 de la práctica 1.
- La función llevará a cabo una determinada operación X, distinta en cada caso (habrá, por tanto, tantas funciones como operadores).

Siguiendo esta aproximación, desarrolle una función de nombre **`imfilter_no_lineal_1`** que asigne a cada píxel de la imagen destino el número de píxeles del entorno de la original que compartan el mismo valor del píxel sobre el que se aplica (incluido este mismo). Dado que en MatLab no existen funciones para facilitarle esta labor, deberá crear los bucles necesarios para explorar la imagen original y aplicar la operación indicada.

Para comprobar que la función que ha realizado funciona correctamente, verifique que invocándola con los siguientes parámetros obtiene el siguiente resultado (recuerde que el resultado de la operación ha de escalarlo en el rango $[0,255]$):

Parámetros	Resultado
$ima = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 3 & 2 & 3 & 2 & 3 & 2 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 4 & 3 & 4 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 4 & 5 & 4 & 5 & 4 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}$, $M' = N' = 3$	$res = \begin{bmatrix} 64 & 64 & 64 & 0 & 0 \\ 64 & 96 & 32 & 0 & 0 \\ 0 & 64 & 64 & 0 & 0 \\ 0 & 32 & 96 & 64 & 0 \\ 0 & 64 & 64 & 64 & 64 \end{bmatrix}$

Cargue y visualice la imagen de prueba en escala de grises **`ima_test.bmp`**[2] . Aplique el operador que ha diseñado para valores de $M'xN'$ de 7x7, 5x5 y 3x3. Visualice las tres imágenes procesadas e indique cuál es el efecto del operador no lineal diseñado y qué tamaño del operador resulta más adecuado para una imagen con detalle fino:

Aplique a continuación el operador más adecuado sobre la imagen **`edificio_bw_512.bmp`** y observe el resultado obtenido. Indique si el resultado era o no el esperado. Trate de interpretar un posible uso del operador en este caso:

Observe que el comportamiento de un operador no lineal en general no es predecible: depende en gran medida de la imagen o tipo de imagen sobre la que aplique.

[2] Descárguese esta imagen de la página *web* de la asignatura.

2 Aplicaciones de filtros lineales

2.1 Ejercicio 3: suavizado de imágenes sintéticas

La generación directa de imágenes sintéticas genera contornos que, salvo que sean horizontales o verticales, exhiben un escalonado claramente visible, fruto de la resolución finita de la imagen. Un filtro de suavizado elimina este molesto efecto con eficacia.

Para comprobar este efecto genere una imagen de tamaño 256x256 consistente en bandas diagonales blancas y negras inclinadas entre 60º y 75º con respecto a la horizontal. Para ello, siguiendo la técnica vista en la Práctica 1, genere esta imagen a partir de la expresión:

$$\psi(x,y) = \mathrm{sgn}\left(\sin\left(40\pi x + 13\pi y\right)\right)$$

NOTA: en MatLab, la función 'sgn' se invoca con la función **`sign`** .

Utilice la función **`imfilter(ima,mask)`** descrita en el ejercicio 1. Visualice la imagen obtenida y observe el efecto indicado. A continuación fíltrela con filtros de media de orden 3, 5 y 7 y con filtros binomiales equivalentes*. Indique, numerando de peor calidad (0) a mejor calidad (5), en qué orden clasificaría las seis imágenes así obtenidas.

Media orden 3	Media orden 5	Media orden 7	Binomial orden 2	Binomial orden 4	Binomial orden 6

*Recuerde que las máscaras de los filtros binomiales de orden N se generan aplicando el producto externo a dos vectores iguales de longitud N+1 cuyos valores siguen la ley de formación de un triángulo de Pascal. Como resultado se obtiene una matriz (N+1)x(N+1) que define el filtro. A continuación se normaliza el filtro de modo que mantenga el rango dinámico. Para calcular el triángulo de Pascal y el vector necesario para generar el filtro, utilice el siguiente código:

```
>> orden = 2; %orden del filtro a obtener
>> triang = cell(1,orden);
>> triang{1} = [1 1]; %vector inicial (Nivel 1 del triangulo de Pascal)
>> for i=2:orden
>>    v_ant = triang{i-1};
>>    v=ones(1, i+1);
>>    v(1)=1;
>>    v(end)=1;
>>    for j=2:length(v)-1
>>        v(j)=v_ant(j-1) + v_ant(j);
>>    end
>>    triang{i}=v;
>> end
>> v= triang{orden};%vector para generar el filtro
```

3 Aplicaciones de filtros no lineales

3.1 Ejercicio 4: eliminación de ruido impulsivo

Uno de los filtros no lineales más utilizados es el filtro de mediana. Su principal uso es la eliminación de ruido impulsivo.

Cargue y visualice la imagen en escala de grises **`edificio_bw_512.bmp`**. A continuación añádale ruido impulsivo (función **`imnoise`**) de tipo **`'salt & pepper'`** y distancia **`0.10`**, y visualice la imagen obtenida.

Intente suavizar el ruido aplicando filtrado lineal binomial de orden 2, 4 y 6. Observe el resultado y puntúe de 0 a 10 el grado de mejora obtenido con esta técnica en cada caso:

Binomial orden 2	Binomial orden 4	Binomial orden 6

A continuación acuda a un filtro no lineal. El filtro de mediana asigna a cada píxel de la imagen procesada la mediana de los valores de los píxeles situados en un entorno definido de la imagen original. Partiendo del código creado para el ejercicio 2, desarrolle una función **`imfilter_no_lineal_2`** que lleve a cabo esta operación (utilice la función **`median`** en el entorno del operador). Aplique a continuación este filtro no lineal sobre la imagen con ruido impulsivo, para entornos de 3x3, 5x5 y 7x7, y puntúe de nuevo la calidad de las imágenes obtenidas:

Mediana 3x3	Mediana 5x5	Mediana 7x7

4 Localización de contornos

4.1 Ejercicio 5: Aplicación del operador de Sobel

En este ejercicio se propone utilizar el operador de *Sobel* para la localización de contornos en imágenes sintéticas y naturales, con el objetivo adicional de ilustrar las precauciones que se deben tomar al utilizar funciones avanzadas proporcionadas por MatLab. Primeramente se presenta la función de MatLab que implementa el operador de *Sobel* (función **`edge`**) y posteriormente se propone la implementación de dicho operador con la función básica **`imfilter`** para comparar los resultados obtenidos con ambas aproximaciones.

Para la localización de contornos, MatLab ofrece la función **`edge`**. Su modo de operación por defecto toma como parámetro la imagen original, **`ima`**, el método deseado para el realce de contornos, **`'sobel'`** (utilice la ayuda de MatLab para consultar los métodos disponibles) y la dirección de cálculo de contornos, **`direction`** (con valores **`'horizontal'`**, **`'vertical'`** o **`'both'`**). Posteriormente a la aplicación del operador, la función **`edge`** realiza una umbralización para decidir qué niveles de gradiente se consideran contornos. Finalmente, la función devuelve una imagen binaria que indica con valor '1' la existencia de contornos y '0' la no-existencia de contornos.

```
>> ima_res = edge(ima,'sobel', direction)
```

Cargue y visualice las imágenes de prueba en escala de grises **`ima_test.bmp`**. Aplique el operador de *Sobel* en la dirección horizontal, vertical y ambas direcciones. Visualice los resultados en ventanas separadas y observe el resultado obtenido. Indique si el resultado es o no el esperado.

A continuación repita la misma operación sobre la imagen en escala de grises **`edificio_bw_512.bmp`**. Visualice los resultados en ventanas separadas y observe el resultado obtenido. Indique si el resultado es o no el esperado. Observe que la aplicación de una función proporcionada por MatLab puede producir resultados inesperados debido a las operaciones que se realizan internamente.

En la segunda parte del ejercicio, se propone analizar en detalle el operador de *Sobel* previamente realizado mediante la función **edge**. Para ello, implemente el filtro de *Sobel* en la dirección horizontal y vertical utilizando la función **imfilter**[3]. Después obtenga la aplicación del filtro en ambas dimensiones mediante la suma de ambos resultados (utilice la función de MatLab **imadd**). Recuerde que para este caso tiene que operar con imágenes de tipo **double** y en el rango $[0,1]$ (para ello utilice la función **im2double()**). Aplique el filtro diseñado a las imágenes **ima_test.bmp** y **edificio_bw_512.bmp**. Visualice los resultados en ventanas separadas y comente el resultado obtenido.

Finalmente, la imagen binaria de contornos se obtiene mediante la umbralización de la imagen filtrada obtenida tras la aplicación de la función **imfilter**. Para obtener dicha imagen binaria utilice el siguiente código:

```
>> ima_res=imfilter(ima,mask);
>> umbral = 0.15; %umbral para binarizacion
>> ima_b = ima_res > umbral;
```

En esta parte se pide obtener la imagen de contornos para las imágenes **ima_test.bmp** y **edificio_bw_512.bmp**. Para ello seleccione el umbral más adecuado a utilizar en el proceso de umbralización. Visualice, observe los resultados obtenidos y razone la diferencia obtenida con la función **edge** de MatLab.

[3] Recuerde que el operador no debe modificar el rango dinámico de la imagen $[0,1]$ y que debe tomar el valor absoluto del resultado de la aplicación del operador (ya que puede presentar valores negativos).