UPV, Facultad de Informática

Dpto. de Arquitectura y Tecnología de Computadores

PROCESADO DIGITAL DE SEÑAL

PROYECTO ESPECÍFICO - LABORATORIO 5 (FIm)

ANALISIS FRECUENCIAL: IMAGEN

OBJETIVOS

- Comprender el uso de la DFT con señales bidimensionales, en concreto con imágenes.
- Obtener la DFT de distintas imágenes y comprender la influencia de las diferentes componentes frecuenciales en la imagen.
- Comprender el funcionamiento de la DCT y su aplicación en técnicas de compresión.

DFT PARA SEÑALES BIDIMENSIONALES

La DFT de una imagen M x N se define como:

$$X(k,l)=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}x(m,n)e^{-j2\pi\left(\frac{k}{M}m+\frac{l}{N}n\right)} \qquad k=0,1,\ldots,M-1 \qquad l=0,1,\ldots,N-1$$

En realidad, la transformada es separable, es decir, se puede obtener aplicando la DFT unidimensional a todas las filas y a continuación a las columnas del resultado, ya que también puede escribirse:

$$X(k,l)=\sum_{m=0}^{M-1}\left(\sum_{n=0}^{N-1}x(m,n)e^{-j2\pi\frac{l}{N}n}\right)e^{-j2\pi\frac{k}{M}m} \qquad k=0,1,\ldots,M-1 \;\; l=0,1,\ldots,N-1$$

La función MATLAB utilizada para calcular la DFT bidimensional es **`fft2`**, cuya sintaxis es la siguiente:

```
X = fft2(x);
```

donde `X` es la matriz obtenida al aplicar la DFT a la matriz `x` (`X` y `x` tienen el mismo tamaño). La función `fft2` utiliza distintos algoritmos según los valores de M y N. La velocidad es máxima cuando son potencias de 2. Por ello, la función **`fft2`** admite otros dos parámetros:

```
X = fft2(x,MM,NN);
```

En ese caso se amplia `x` añadiendo ceros para que tenga (`MM * NN`) elementos.

Ejemplo 1:

Vamos a generar una imagen monocroma artificial con una trama sinusoidal de brillos, utilizando la siguiente función:

$$x(m,n)=0.5+0.5\sin\left[2\pi\left(\frac{3}{17}m+\frac{1}{25}n\right)\right] \qquad 0\le m\le 127,\ 0\le n\le 127$$

en MATLAB

```
[m, n] = ndgrid(0:127, 0:127);        %genera todos los pares (m,n) del rango dado
x = 0.5+0.5*sin(2*pi*(3/17*m+1/25*n));
figure, imshow(x)
```

Para calcular y visualizar la DFT de la imagen basta hacer:

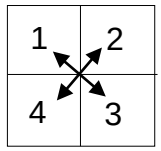
```
X = fft2(x);
figure, subplot(1,2,1), imshow(abs(X),[]),axis on;
subplot(1,2,2), imshow(angle(X),[]);
```

Hay que tener en cuenta que, aunque estamos visualizando el espectro como si fuera una imagen, sus valores no están dentro del rango [0, 1]. Por ello, especificamos en la función `imshow` la opción [ ] mediante la cual la función interpreta los valores mínimo y máximo encontrados en la matriz, como los brillos correspondientes al negro y al blanco respectivamente. Para visualizar la escala se utiliza el comando `axis on`.
Puedes observar que:

- En el espectro de magnitud el valor máximo aparece en k=0, l=0 que corresponde al brillo medio de la imagen. Esto es típico de las imágenes, ya que siempre tienen un brillo medio positivo.
- Aparece también un pico muy claro en el punto correspondiente a unas frecuencias vertical y horizontal cercanas a 3/17 y 1/25 (ciclo/m) respectivamente, y como la señal es real aparece duplicado para una frecuencia vertical de aproximadamente 14/17 y horizontal de 24/25 (ciclo/m). Esto se debe a que hemos generado la imagen precisamente con una sinusoide de esas frecuencias.
- En el resto los valores son prácticamente nulos (ausencia de brillo, negro).

Por otro lado, normalmente se visualiza el espectro utilizando los rangos de frecuencia (–0.5,0.5), esto es desplazado, mediante la función MATLAB `fftshift`:

```
XX = fftshift(X);
figure, imshow(abs(XX),[]),axis on;
```



Para apreciar mejor los detalles es habitual utilizar una escala logarítmica, y en ese caso conviene especificar a `imshow` los límites de brillo (ya que el logaritmo de 0 tiende a menos infinito):

```
Xdb = 20*log10(abs(XX));
dbmax = max(Xdb(:));
figure, imshow(Xdb,[dbmax-60 dbmax]), axis on;
```

Hemos especificado para el brillo máximo (blanco) el valor máximo del espectro en decibelios, y para el brillo mínimo (negro) un valor 60 decibelios por debajo (un factor de 1000 en el espectro original).

Finalmente podemos comprobar que obtenemos el mismo resultado aplicando la `fft` unidimensional dos veces:

```
X = fft(fft(x).').';
XXdb = fftshift(20*log10(abs(X)));
figure, imshow(XXdb,[dbmax-60 dbmax]),axis on;
```

Ejercicio 1:

A continuación, calcula y visualiza los espectros de las siguientes imágenes

$$x_1(m,n) = \frac{1}{24} resto\left(\frac{3m+3n}{25}\right) \quad 0 \le m \le 255, 0 \le n \le 255$$ (Usar la función **`rem`** de MATLAB)

$$x_2(m,n) = \begin{cases} 1, & 0 \le m \le 24 \ldots y \ldots 0 \le n \le 24 \\ 0, & 25 \le m \le 255 \ldots o \ldots 25 \le n \le 255 \end{cases}$$

Para ello:

1) Sigue los pasos efectuados en el ejemplo.
2) Edita un fichero con los comandos ejecutados y visualiza los espectros obtenidos en cada caso.
3) Para la función x1 trata de estimar las frecuencias asociadas a los puntos de máximo brillo (al menos 3 armónicos). Para visualizar las escalas se puede utilizar el comando `axis on`.

Ejercicio 2:

A continuación, calcula y visualiza los espectros de las fotografías de los ficheros **`arboles.jpg`** y **`vias.jpg`**.

P5_1_dft_bidim_ej1_ej2.m

IDFT

La IDFT de un espectro MxN se calcula mediante la fórmula siguiente:

$$x(m,n)=\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}X(k,l)e^{j2\pi\left(\frac{k}{M}m+\frac{l}{N}n\right)} \quad m=0,1,\ldots,M-1, \quad n=0,1,\ldots,N-1$$

implementada en MATLAB mediante la función **`ifft2`** Esta operación nos permite reconstruir la señal original a partir de su espectro.

Ejemplo 2:

Supongamos que el espectro de la imagen **`arboles.jpg`** se encuentra en la variable **`X`** Podemos recuperar la imagen original simplemente aplicando `ifft2` (ten en cuenta que si la transformada se ha calculado con `fftshift`(), la transformada inversa se calcula con `ifftshift`( )):

```
xx = uint8(real(ifft2(ifftshift(X))));
figure, imshow(xx)
```

El resultado de **`ifft2()`** debe ser una señal real, pero, es posible que debido a los errores de redondeo MATLAB siga considerando el resultado complejo. Por eso, conviene aplicar **`real()`**. Por otra parte, se aplica **`uint8()`** porque era el tipo de la imagen original.

RECUERDA que la recuperación se hace a partir del espectro completo (magnitud y fase). El espectro de fase en imágenes es muy importante. Si lo modificamos la imagen cambia ya que estamos desplazando las componentes frecuenciales de la imagen original.

Ejercicio 3:

Para comprobarlo aplica la transformada inversa solo al espectro de magnitud y comprueba el resultado.

P5_2_ifft2_magnitud_ej3.m

Ejemplo 3:

Otro proceso que podemos hacer en el dominio de la frecuencia, y que puede ser muy instructivo, es eliminar determinadas frecuencias de la señal. Esto se consigue anulando (poniendo a cero) los elementos correspondientes del espectro y aplicando la transformada inversa. Para ello, se propone usar la función **`mascara`** siguiente:

```
function MA = mascara(tam, Fc, tipo)
M = tam(1);
N = tam(2);
MA = zeros(M, N);
[m, n] = ndgrid(ceil(-M/2):ceil(M/2-1), ceil(-N/2):ceil(N/2-1));
Fm = m/M;
Fn = n/N;
if tipo=='bajo'
   MA = sqrt(Fm.^2+Fn.^2)<Fc;
elseif tipo=='alto'
   MA = sqrt(Fm.^2+Fn.^2)>Fc;
end
```

Esta función genera una matriz formada por unos y ceros: la región correspondiente a los unos (de forma “circular”) se especifica mediante una frecuencia Fc (entre 0 y 0.5) que se pasa en el segundo argumento y un parámetro (‘bajo’ o ‘alto’) que indica si los unos están en el interior o en el exterior de esa región, respectivamente.

Como ejemplo vamos a eliminar del espectro de la imagen **arboles.jpg** las componentes de frecuencias fuera del círculo de radio 0.2:

```
M = mascara(size(X), 0.2, 'bajo');
Xf = M.*X;
xf = uint8(real(ifft2(ifftshift(Xf))));
figure, imshow(xf)
```

Ejercicio 4:

Prueba ahora a eliminar frecuencias por encima de 0.1 y por debajo de 0.1. Prueba también a filtrar todas las frecuencias fuera del rango comprendido entre 0.05 y 0.1.

P5_3_mascaras_ej4.m

Ejemplo 4:

Basándonos en esto podemos hacer cosas más sofisticadas. En el fichero **`ejemplo4.m`**, primero se contamina artificialmente la imagen **`arboles.jpg`** con rayas verticales espaciadas 3 pixels. Al visualizar el espectro aparece claramente el efecto de la alteración alrededor de los puntos $k = 0, l = \pm 213$ (la imagen tiene 640 columnas). A continuación, se modifica el espectro, anulando los valores alrededor de estos puntos (filtrado en el dominio de la frecuencia) y se aplica la transformada inversa.

DCT y compresión de imágenes

La transformada del coseno o DCT, es una transformada que se usa mucho en compresión de imágenes porque la información perceptible de la imagen se concentra en los coeficientes correspondientes a valores bajos de la frecuencia. Esta transformada está emparentada con la DFT pero trabaja sólo con números reales. En MATLAB para obtener la DCT de una imagen, utilizaremos una función del toolbox de procesamiento de imágenes:

```
X = dct2(x);
```

Y la que reconstruye la imagen original a partir de la DCT es:

```
x = idct2(X);
```

Para comprobar su interés en compresión, vamos a aplicarla a la imagen 'autumn.tif'. Es una imagen en formato RGB que la transformamos a matriz de intensidades en blanco y negro:

```
x = imread('autumn.tif');
figure,imshow(x);
I = rgb2gray(x);
figure,imshow(I,[0 255])
J = dct2(I);
figure,imshow(20*log10(abs(J)),[]), colormap(jet(64)), colorbar
```

Dado el espectro si anulamos en la transformada todos los términos de valor absoluto menor que 10 y reconstruimos la imagen con la función inversa idct2:

```
J(abs(J) < 10) = 0;
K = idct2(J);
figure, imshow(K,[0 255])
```

Comprobamos que la nueva imagen K es prácticamente igual a la imagen en blanco y negro I.

Otro ejercicio consiste en aplicar esta transformada un fragmento 8x8 cualquiera de una imagen:

```
x = imread('julia.jpg');
figure, imshow(x);
frag = x(250:257, 95:102);
figure, subplot(1,2,1),imshow(frag,'InitialMagnification','fit')
FRAG = dct2(frag);
FRAGdb = 20*log10(abs(FRAG));
dbmax = max(FRAGdb(:));
subplot(1,2,2),imshow(FRAGdb,[dbmax-40 dbmax])
```

Puede verse que los valores más altos corresponden a las frecuencias bajas además las frecuencias altas tienen menos importancia en la percepción. El parámetro `'InitialMagnification','fit'` lo damos porque la imagen es muy pequeña y no queremos que la visualice a tamaño real.

Si de los 64 valores anulamos todos menos los 16 del cuadrante superior izquierdo, e invertimos la transformada:

```
COMP = zeros(8,8);
COMP(1:4,1:4) = FRAG(1:4,1:4);
comp = uint8(idct2(COMP));
figure, imshow(comp, 'InitialMagnification','fit')
```

Puede verse que el aspecto del fragmento no ha cambiado mucho. Se han suavizado las variaciones de brillo, pero esto lo percibimos porque estamos viendo una imagen muy ampliada.

Un modo simple de compresión de imágenes con pérdida de información, pero poco perceptible consiste en:

- Dividir toda la imagen en fragmentos de 8x8 píxeles y calcular la DCT de cada fragmento.
- Truncar todos los fragmentos obtenidos en esta operación anulando valores correspondientes a frecuencias altas. A más valores anulados, más compresión y menos calidad.
- Para regenerar la imagen basta aplicar lDCT a los fragmentos almacenados.

Para comprobar la bondad del método puedes utilizar una demo que existe en MATLAB **`dctdemo.m`** que ilustra este tema.