# Practica 12: Compresión DFFT

Procesamiento digital de imágenes

González Cano Daphne Sofia
2022710039
dgonzalezc2104@alumno.ipn.mx

Castro Elvira Diego
2022710168
dcastroe2100@alumno.ipn.mx

UPIIT: Unidad Profesional Interdisciplinaria en Ingeniería Campus Tlaxcala Instituto Politécnico Nacional, Tlaxcala, Tlaxcala, México 9000

Ingeniera en Inteligencia Artificial

31 de mayo de 2023

***Resumen*— *La compresión de imágenes consiste en reducir los datos redundantes e irrelevantes para un almacenamiento o transmisión eficiente. Puede ser sin pérdida, preservando la información original, o con pérdida, sacrificando cierta información. La Transformada de Fourier Discreta (DFT) transforma una función en el dominio del tiempo en una representación en el dominio de la frecuencia. La DFT se utiliza en métodos de compresión como JPEG, donde se aplica la Transformada de Coseno Discreta (DCT) y la DFT para cuantizar los coeficientes de frecuencia. La compresión fractal divide la imagen en bloques y utiliza la DFT para aproximarlos. La compresión basada en señales dispersas aprovecha la dispersión de coeficientes de frecuencia cercanos a cero en la DFT.***

***Palabras clave* — *Transformada de Fourier, compresión de imágenes, Transformada inversa de Fourier***

## I. OBJETIVOS

- Comprende los principios básicos en lo que se basa la transformada de Fourier y su utilidad en el campo del procesamiento de imágenes.
- Analiza la utilidad de la transformada de Fourier como mecanismo para lograr la compresión de imágenes.
- Comprende la relación entre el tamaño del espectro de la imagen y la calidad de la imagen.
- Analiza y aplica el procedimiento de cálculo de la transformada de Fourier.
- Comprueba de forma experimental el efecto de reducir el espectro de la imagen para lograr una compresión de una imagen y como afecta a la calidad de dicha imagen.

## II. MARCO TEÓRICO

### A. Compresión de imagen

Comprimir una imagen es reducir los datos redundantes e irrelevantes de la imagen con la menor pérdida posible para permitir su almacenamiento o transmisión de forma eficiente. La compresión de imagen puede ser con pérdida (Lossy) o sin pérdida (LossLess). La compresión sin pérdida se reduce el tamaño del archivo de imagen sin perder información esto garantiza que la imagen original se pueda restaurar completamente después de la descompresión. A diferencia de la compresión con pérdida la cual implica una pérdida irreversible de datos, lo que significa que la imagen original no se puede recuperar exactamente después de la descompresión.

### B. Transformada de Fourier discreta

La transformada discreta de Fourier o DFT es un tipo de transformada discreta que transforma una función matemática en otra, obteniendo una representación en el dominio de la frecuencia, siendo la función original una función en el dominio del tiempo. Esta transformación únicamente evalúa suficientes componentes frecuenciales para reconstruir el segmento finito que se analiza.
La secuencia de $N$ números complejos $x_0$, ..., $x_{N-1}$ se transforma en la secuencia de $N$ números complejos $X_0$, ..., $X_{N-1}$ mediante la DFT con la fórmula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

C. Trasformada inversa de Fourier

La trasformada inversa de Fourier se trata de recuperar la función original a partir de su transformada de Fourier, para ello se requiere que ambas funciones sean totalmente integrables

La transformada inversa de Fourier discreta (IDFT) viene dada por:

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N}kn} \qquad n = 0, \ldots, N-1.$$

D. Trasformada rápida de Fourier

La transformada rápida de Fourier, es un algoritmo eficiente que permite calcular la transformada de Fourier discreta (DFT) y su inversa. Cuando se habla del tratamiento digital de señales, el algoritmo FFT impone algunas limitaciones en la señal y en el espectro resultante ya que la señal muestreada y que se va a transformar debe consistir en un número de muestras igual a una potencia de dos.

E. Métodos de compresión basados en la DFFT

- Compresión JPEG: utiliza una combinación de la DCT (Transformada de Coseno Discreta) y la DFFT para comprimir imágenes. La DCT se aplica a bloques de píxeles en la imagen y, a continuación, se realiza una cuantización de los coeficientes de frecuencia resultantes.
- Compresión fractal: es un enfoque basado en la recursividad y la auto semejanza. En este método, se divide la imagen original en bloques más pequeños, y cada bloque se aproxima utilizando una transformación iterada y fractales. La DFFT se utiliza a menudo para calcular las similitudes entre los bloques y para mejorar la precisión de la aproximación.
- Compresión basada en señales dispersas: Este enfoque aprovecha el hecho de que muchas imágenes son "dispersas" en el dominio de la transformada de Fourier, lo que significa que la mayoría de los coeficientes de frecuencia son cercanos a cero.

## III. METODOLOGÍA

1. Aplicar el algoritmo de la DFFT a la imagen para obtener el espectro de la imagen.

```
def FFT(x):
    Y = x.shape[0]
    if Y == 1:
        return x
    else:
        X_even = FFT(x[::2])
        X_odd = FFT(x[1::2])
        C = np.exp(-2j * np.pi * np.arange(Y) / Y)
        even_c = C[:Y//2]
        odd_c = C[Y//2:]

        even_part = X_even + even_c * X_odd
        odd_part = X_even + odd_c * X_odd

        X = np.concatenate((even_part, odd_part))
    return X
```

La función FFT realiza la FFT en una dimensión donde:

1.1 Y = x.shape[0] obtiene el tamaño de la dimensión 0 del vector de entrada x, es decir, el número de elementos en el vector.

1.2 if Y == 1: verifica si el tamaño del vector es igual a 1. Si es así, significa que no se puede dividir más y se devuelve el vector original.

1.3 X_even = FFT(x[::2]) realiza una llamada recursiva a la función, con los elementos de x que tienen índices pares.

1.4 X_odd = FFT(x[1::2]) realiza lo mismo pero con los elementos de x que tienen índices impares.

1.5 C = np.exp(-2j * np.pi * np.arange(Y) / Y) calcula los coeficientes, utilizando la fórmula de la exponencial compleja.

1.6 even_c = C[:Y//2] obtiene la mitad de los coeficientes para los elementos pares.

1.7 odd_c = C[Y//2:] obtiene la mitad de los coeficientes para los elementos impares.

1.8 even_part = X_even + even_c * X_odd combina los resultados de los elementos y los coeficientes pares.

1.9 odd_part = X_even + odd_c * X_odd combina los resultados de los elementos y los coeficientes impares.

1.10 X = np.concatenate((even_part, odd_part)) concatena las partes pares e impares para obtener la transformada final.

Se requiere una segunda función FFT_2D la cual aplica la FFT en dos dimensiones a una matriz de entrada:

```
def FFT_2D(matriz):
    matriz = recortarImg(matriz)

    rows = np.zeros_like(matriz, dtype=np.complex128)
    for i in range(matriz.shape[0]):
        rows[i, :] = FFT(matriz[i, :])

    finalFFT2 = np.zeros_like(matriz,
dtype=np.complex128)
    for j in range(matriz.shape[1]):
        finalFFT2[:, j] = FFT(rows[:, j])

    return finalFFT2
```

Donde:

1.1. matriz = recortarImg(matriz) realiza un recorte de la matriz de entrada utilizando una función recortarImg la cual:

```
    def recortarImg(img):
        Xc, Yc = img.shape[1], img.shape[0]
        newH = powClose(img.shape[0])
        newW = powClose(img.shape[1])
        p1 = math.floor((Xc - newW) / 2)
        p2 = math.floor((Yc - newH) / 2)
        return img[p2:p2+newH, p1:p1+newW]
```

1.1.1. `newH = powClose(img.shape[0])` calcula la nueva altura de la imagen como el resultado de la función powClose aplicada a la altura original de la imagen (img.shape[0]). La función powClose se utiliza para redondear el valor de entrada a la potencia de dos más cercana:

```
def powClose(cantidad):
    potencia_menor = math.floor(math.log2(cantidad))
    potencia_dos_menor_cercana = pow(2, potencia_menor)
    return potencia_dos_menor_cercana
```

1.1.2. `newW = powClose(img.shape[1])` calcula la nueva anchura (newW) de la imagen de manera similar a newH, pero aplicando la función powClose a la anchura original de la imagen (img.shape[1]).

1.1.3. `p1 = math.floor((Xc - newW) / 2)` calcula la posición de inicio en el eje X para el recorte de la imagen. Resta la nueva anchura de la imagen (newW) de la anchura original de la imagen (Xc), divide el resultado entre 2 y toma el valor entero inferior utilizando la función math.floor().

1.1.4. `p2 = math.floor((Yc - newH) / 2)` calcula la posición de inicio en el eje Y para el recorte de la imagen de manera similar a p1, pero en el eje Y.

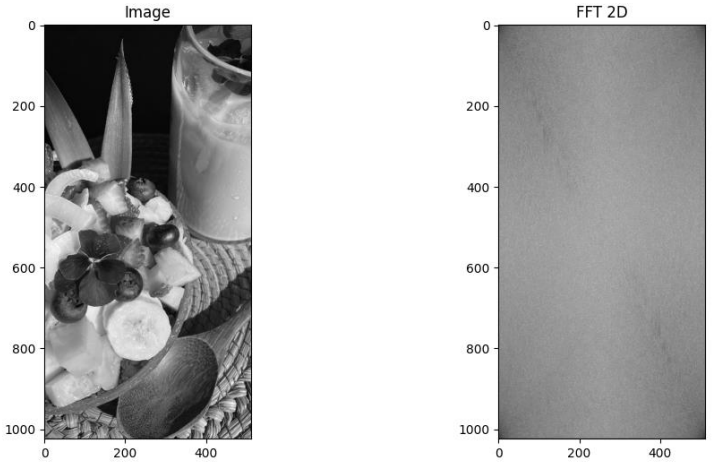
1.1.5. `return img[p2:p2+newH, p1:p1+newW]` devuelve la porción recortada de la imagen utilizando los índices calculados p1, p2, newW y newH. Esto recorta la imagen desde la posición (p2, p1) hasta (p2+newH, p1+newW)

1.2. El primer bucle for itera sobre las filas de la matriz y aplica la función FFT a cada fila

1.3. El segundo bucle for itera sobre las columnas de la matriz y aplica la función FFT a cada columna

1.4. `return finalFFT2` devuelve la matriz que representa la transformada de Fourier bidimensional de la matriz de entrada

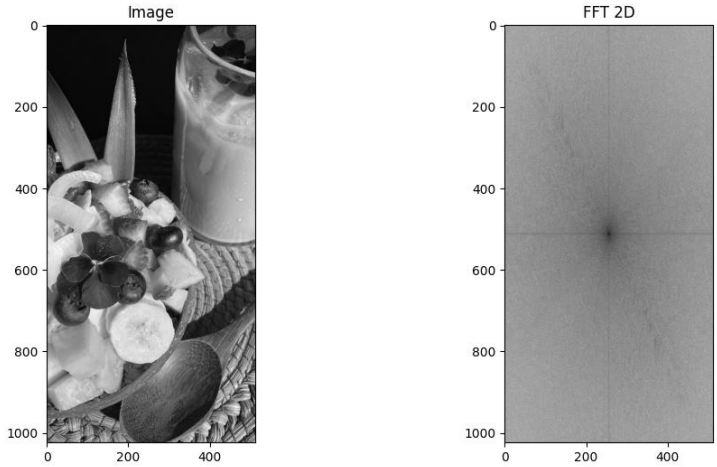
Si adelantamos los resultados, obtenemos:




Donde en el resultado podemos observar las frecuencias altas que se ven representadas en el centro y las frecuencias bajas que son lo negro de las orillas, tenemos que realizar una función que centre el resultado, es decir, colocar el píxel (0,0) en el centro de la imagen, por lo que:

```
def centerInfoFFT(matrix):
    filas, columnas = matrix.shape
    Xc = filas // 2
    Yc = columnas // 2
    matrixRes = np.zeros_like(matrix)
    matrixRes[:Xc, :Yc] = matrix[-Xc:, -Yc:]
    matrixRes[-Xc:, -Yc:] = matrix[:Xc, :Yc]
    matrixRes[:Xc, -Yc:] = matrix[-Xc:, :Yc]
    matrixRes[-Xc:, :Yc] = matrix[:Xc, -Yc:]
    return matrixRes
```

El resultado que obtenemos es mas claro:




2 Aplicar un recorte al espectro de 50%, 80% y 95%.

```
def recortarMask(mask, percent):
    height, width = mask.shape[:2]
    center_x = width // 2
    center_y = height // 2

    newW = int(width * (1 - percent / 100) / 2)
    newH = int(height * (1 - percent / 100) / 2)

    MaskRes = mask[center_y - newH:center_y + newH,
              center_x - newW:center_x + newW]

    return recortarImg(MaskRes)
```

2.1 `height, width = mask.shape[:2]` obtiene las dimensiones de la máscara en altura y ancho.

2.2 `center_x = width // 2` & `center_y = height // 2` calculan la posición central

2.3 `newW` y `newH` calculan la cantidad de píxeles a recortar en cada borde de la máscara. El cálculo se basa en el porcentaje dado como entrada. Se utiliza la fórmula `(1 - percent / 100) / 2` para obtener la fracción del ancho y altura de la máscara que se recortará, y luego se convierte a enteros

2.4 `MaskRes` realiza el recorte central en la máscara utilizando los valores calculados center_x, center_y, newW y newH.

3 Aplicar el algoritmo de IDFFT y visualizar la calidad resultante como producto de la compresión.

Para el cálculo de la Transformada Discreta Inversa de Fourier (IDFFT), se realiza un proceso similar al de la función DFFT, donde primero se hace en una matriz unidimensional:

```
def IFFT(x):
    Y = x.shape[0]
    if Y == 1:
        return x
    else:
        X_even = IFFT(x[::2])
        X_odd = IFFT(x[1::2])
        C = np.exp(2j * np.pi * np.arange(Y) / Y)
        even_c = C[:Y//2]
        odd_c = C[Y//2:]

        even_part = X_even + even_c * X_odd
        odd_part = X_even + odd_c * X_odd

        X = np.concatenate((even_part, odd_part))
        return X
```

3.1 `Y = x.shape[0]` obtiene la longitud de la matriz x en el eje 0 (número de filas) y la asigna a la variable Y.

3.2 `if Y == 1`: verifica si Y es igual a 1, lo cual indica que la matriz x es un único elemento, en cuyo caso se devuelve x sin realizar ninguna transformación.

3.3 En caso contrario, `X_even = IFFT(x[::2])` & `X_odd = IFFT(x[1::2])` realizan la IFFT de la mitad de los elementos de la matriz x, el primero solo toma los elementos en las posiciones pares y el segundo en los impare

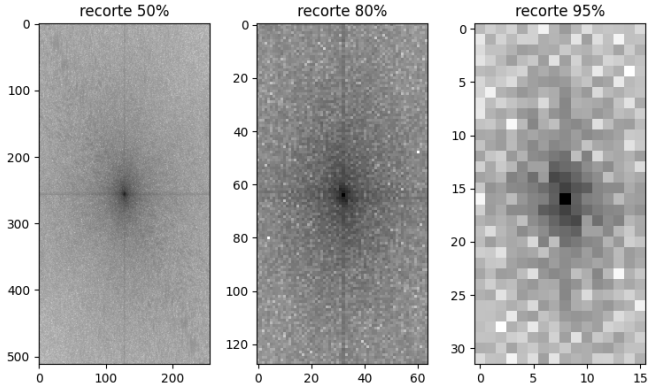
3.4 `C = np.exp(2j * np.pi * np.arange(Y) / Y)` calcula los coeficientes de la transformada inversa.

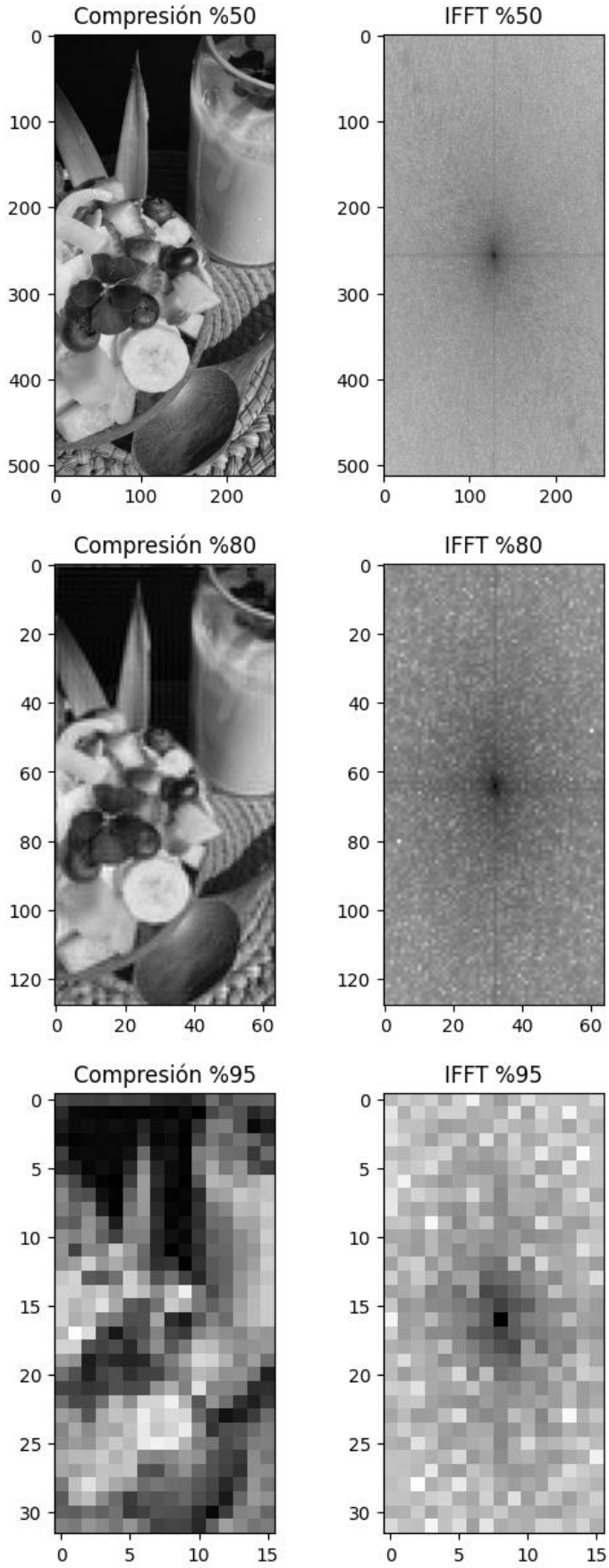
3.5 `even_c = C[:Y//2]` & `odd_c = C[Y//2:]` oma la otra mitad de los coeficientes, en el primer caso solo los pares y el segundo los impares

3.6 `even_part = X_even + even_c * X_odd` & `odd_part = X_even + odd_c * X_odd` realizan la combinación de las pares e impares de la IFFT

3.7 `X = np.concatenate((even_part, odd_part))` concatena las partes pares e impares para formar el resultado final de la IFFT.

## IV. RESULTADOS






## V. CONCLUSIONES Y OBSERVACIONES

Con los resultados obtenidos podemos decir que la compresión mediante la Transformada Discreta Rápida de Fourier (DFFT) es una técnica eficiente para reducir el tamaño de las imágenes sin perder información esencial. La DFFT permite transformar la imagen al dominio de la frecuencia, donde se pueden eliminar componentes redundantes o de baja importancia. Al comprimir la imagen, se pueden lograr reducciones significativas en el tamaño del archivo, lo que facilita su almacenamiento y transmisión. Sin embargo, es importante tener en cuenta que la compresión basada en esta técnica puede introducir cierta pérdida de calidad en la imagen debido a la eliminación selectiva de componentes de frecuencia, esto lo podemos visualizar comparando los resultados obtenidos aplicando la compresión en diferentes porcentajes. Por lo tanto, es necesario encontrar un equilibrio entre la compresión y la preservación de la calidad de la imagen para satisfacer las necesidades específicas de cada caso de uso. Una posible mejora es encontrar de forma automática esta compresión, donde encontremos un balance adecuado entre la compresión y la calidad de la imagen

## VI. REFERENCIAS

[1] Compresión de imagen. (2023, 1 de mayo). Wikipedia, La enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Compresi%C3%B3n_de_imagen&oldid=150901162.

[2] Transformada rápida de Fourier. (2022, 23 de diciembre). Wikipedia, La enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Transformada_r%C3%A1pida_de_Fourier&oldid=148099216.

[3] Transformada de Fourier discreta. (2023, 9 de enero). *Wikipedia, La enciclopedia libre*. https://es.wikipedia.org/w/index.php?title=Transformada_de_Fourier_discreta&oldid=148495486.