

Situación Problema Entregable Final

Facundo Bautista Barbera^{1,*} and Luis Santiago Vargas Ochoa^{1,†}

¹*Escuela de Ingeniería y Ciencias, campus Monterrey.*

Tecnológico de Monterrey

(Dated: June 15, 2025)

Abstract

En este reporte se presentan los resultados de la experimentación realizada con un método de compresión de imágenes basado en la Transformada Discreta de Fourier en dos dimensiones (FFT 2D). Se introduce el problema y se explica la elección de la FFT 2D como técnica de compresión. En la sección de experimentación se detalla el flujo de trabajo implementado en Python. Finalmente, se hace una comparación de las tasas de compresión obtenidas en imágenes a color y en escala de grises.

INTRODUCCIÓN

En esta segunda entrega de la situación problema para la unidad de formación de Análisis Métodos Matemáticos Para La Física, se presentan los resultados de la investigación realizada en la primera etapa y los resultados de experimentación de compresión de imágenes.

Dando un poco de contexto adicional, hoy en día, enormes volúmenes de imágenes son generados y compartidos a diario, desde fotografías personales hasta imágenes satelitales o médicas. La compresión de imágenes es un proceso útil para poder disminuir el espacio de almacenamiento necesario para manejar este tipo de datos no estructurados. Un archivo de imagen sin comprimir (con algún formato de tipo RAW) almacena la información píxel por píxel y suele tener un tamaño considerable.

Utilizando métodos de compresión es posible reducir de forma impactante el tamaño de un archivo de imagen, logrando mantener una calidad visual aceptable. Un formato popular es el formato JPEG, el cual logra tasas de compresión cercanas al 90% (Es decir, comprime la imagen a un 10% de su tamaño original), manteniendo una calidad de imagen considerable. Esto se logra gracias a la redundancia de información visual y las limitaciones de percepción que tiene el ojo humano.

Existen dos tipos principales de compresión: sin pérdida (lossless), donde la imagen original se puede reconstruir exactamente igual, y con pérdida, donde se permite una leve degradación a cambio de mayores tasas de compresión. Para las experimentaciones realizadas para esta situación problema, se probará crear un script que cree una compresión con pérdida.

EXPERIMENTACIÓN

Entorno y herramientas

El proceso de experimentación se desarrolló usando Python 3.12, usando un entorno virtual (venv) para poder reproducir el proyecto fácilmente. Para el cómputo numérico y las transformadas se usó NumPy (v2.3.0). La lectura y escritura de imágenes utiliza Pillow (v11.2.1) para formatos estándar y rawpy (v0.25.0) para archivos RAW@.

Proceso de compresión

En esta sección se busca describir el flujo completo de compresión. Se implementa en una clase `Image`, definida en `image.py`, integrando la parte programática y matemática.

Carga de la imagen

Se abre un archivo (RAW, JPEG, u otros tipos) usando Pillow o rawpy, en caso de que la imagen sea de tipo RAW y se convierte en un array de NumPy de tipo Float.

Listing 1. Método `_load_image`

```
1 def _load_image(self, image_path, greyscale=False):
2     try:
3         img = PILImage.open(image_path)
4     except (UnidentifiedImageError, OSError):
5         with rawpy.imread(image_path) as raw:
6             rgb = raw.postprocess(no_auto_bright=True, output_bps
7                                   =8)
8             img = PILImage.fromarray(rgb)
9     img = img.convert('L' if greyscale else 'RGB')
9     return np.array(img, dtype=float)
```

Transformada de Fourier en 2D

Se aplica la FFT bidimensional y centramos el espectro para revelar las componentes de frecuencia (se usa numpy para estos procesos).

Listing 2. Método `_fast_fourier_transform`

```
1 def _fast_fourier_transform(self, image_array):
2     ft = np.fft.fft2(image_array, axes=(0,1))
3     return np.fft.fftshift(ft)
```

La FFT 2D parte de la idea de la FFT 1D, la cual normalmente se usa para descomponer señales unidimensionales en una suma de ondas sinusoidales de distintas frecuencias y amplitudes. En este caso nuestra ‘onda’ es una función bidimensional $f(x, y)$ que corresponde a la imagen.

Enmascaramiento del espectro

Listing 3. Enmascaramiento del espectro

```
1 magnitude = np.abs(fourier_transformed)
2 threshold = np.percentile(magnitude, 100 * (1 - ratio))
3 mask = magnitude > threshold
4 fourier_compressed = fourier_transformed * mask
```

Al aplicar un umbral sobre las magnitudes de este espectro, conservamos únicamente los componentes más importantes. Este umbral se calibró con un ratio de pérdida de 0.1, lo que significa que se descarta hasta el 90% de las características menos relevantes durante la compresión, sin afectar significativamente la calidad visual.

Transformada inversa

Listing 4. Transformada inversa

```
1 img_back = np.fft.ifft2(np.fft.ifftshift(fourier_compressed)).real
2 img_back = np.clip(img_back, 0, 255).astype(np.uint8)
```

Se reconstruye la imagen utilizando IFFT2D, es decir la transformada inversa después de haber recortado los valores fuera de rango.

Guardado del resultado

Listing 5. Guardado de la imagen comprimida

```
1 img = PILImage.fromarray(img_back)
2 img.save(output_path, quality=100, subsampling=0)
```

Se utiliza Pillow para exportar la imagen comprimida con calidad máxima, evitando pérdidas adicionales de compresión.

COMPARACIÓN

Para poder ilustrar mejor las diferencias entre las imágenes originales y las comprimidas, tanto en escala de grises como en color, se creó un notebook de Jupyter que presenta los resultados visuales. Esta notebook fue generada con apoyo de herramientas con inteligencia artificial y cumple únicamente el propósito de demostración gráfico y no afecta directamente en el desarrollo técnico de proyecto.

Imagenes de comparación



FIG. 1. Resultados de compresión imagen 1

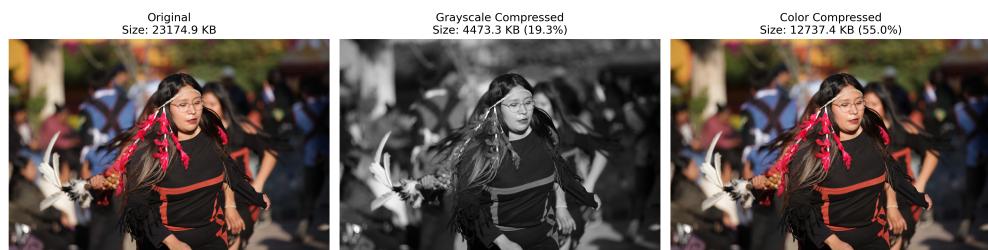


FIG. 2. Resultados de compresión imagen 2



FIG. 3. Resultados de compresión imagen 3

Observaciones sobre las imágenes

Con la implementación del método basado en FFT 2D se logró reducir el tamaño de los archivos sin sacrificar visualmente los detalles esenciales. Al transformar las imágenes se mantiene intacta una estructura general, contornos, gradaciones suaves y las texturas continuas.

Para las imágenes de color, el método FFT 2D logra reducir las imágenes entre un 50% y un 60% de su tamaño original. Esto se debe a que los 3 canales de color (rojo, verde y azul) comparten redundancias y el FFT 2D logra descartar muchas frecuencias sin que aparezca perceptibles.

Para las imágenes en escala de grises se puede observar que la compresión es menor, se logra reducir entre un 15% y un 25%, lo cual sigue siendo un tamaño considerable.

Visualmente se puede observar que las imágenes son perfectamente perceptibles. Al ampliar la imagen comprimida, se puede percibir una ligera cantidad de ruido y algunos bordes ligeramente más duros, pero cumple con la función de compresión.

Es importante destacar que aunque la compresión es consistente en la mayor parte de los casos, aunque si existe la posibilidad de que la imagen no se comprima adecuadamente si esta contiene muchos detalles granulares o ruido visual (como partículas).

CONCLUSIONES

Este reporte presentó un método de compresión de imágenes con base en la Transformada Discreta de Fourier en dos dimensiones (FFT 2D). Los resultados de la experimentación demuestran que la técnica implementada permite alcanzar tasas de compresión considerables, lo cual es aún más notable en imágenes comprimidas con color, gracias a las redundancias de los canales cromáticos. A pesar de la eficiencia que tiene el modelo, este también puede tener sus limitaciones, como producir ruido visible en zonas con mucho detalles o texturas granuladas, y un umbral fijo no adapta automáticamente a variaciones locales de contenido (es decir, ajuste de enmascaramiento por región de imagen).

Nota de Facundo

Considero que este reporte y, en general, el proceso de experimentación fue bastante satisfactorio. El desarrollar el código que implementa un método matemático para hacer algo real y útil fue realmente entretenido. Aplicar las transformadas de Fourier para algo que no sea simplemente resolver un problema matemático cualquiera realmente ayuda a centrar más la utilidad del concepto. Desde un inicio me pareció increíble como es que las series y transformada de Fourier logran capturar elementos ‘escondidos’ en las cosas y creo que este conocimiento me será realmente útil en el futuro para entender cosas como redes neuronales.

REFERENCIAS

* a01066843@tec.mx

† a01563708@tec.mx

- [1] OpenAI ChatGPT, Response generated by chatgpt on image compression and jpg standard, <https://chatgpt.com/share/68253a67-1958-800a-a3f1-a8be67be8ef1> (2025), accessed on May 14, 2025.
- [2] N. Gupta, R. Vijay, and H. K. Gupta, EAI Endorsed Transactions on Energy Web **7**, 10.4108/EAI.13-7-2018.163976 (2020).
- [3] N. L. Sugara, T. W. Purboyo, and A. L. Prasasti, Journal of Engineering and Applied Sciences **13**, 10.3923/jeasci.2018.4447.4452 (2018).
- [4] I. A. Ismaili, S. A. Khowaja, and W. J. Soomro, in *Proceedings of the 2013 International Conference on Image Processing, Computer Vision, and Pattern Recognition, IPCV 2013*, Vol. 2 (CSREA Press, 2013) pp. 962–965.
- [5] R. A. Hamzah, M. M. Roslan, A. F. B. Kadmin, S. F. B. A. Gani, and K. A. A. Aziz, Telkomnika (Telecommunication Computing Electronics and Control) **19**, 10.12928/TELKOM-NIKA.v19i3.14758 (2021).

Anexo: Código Python

Listing 6. Código de compresión de imagen en Python

```
1 import numpy as np
2 import rawpy
3 import os
4 from PIL import (
5     Image as PILImage,
6     UnidentifiedImageError
7 )
8
9
10 class Image:
11
12     def __init__(self, image_path=None):
13         self.image_path = image_path
14         self.image_array = None
15
16         if image_path:
17             self.image_array = self._load_image(image_path)
18
19     def _load_image(self, image_path, greyscale=False):
20         img = None
21
22         try:
23             img = PILImage.open(image_path)
24         except (UnidentifiedImageError, OSError):
25             with rawpy.imread(image_path) as raw:
26                 rgb = raw.postprocess(
27                     no_auto_bright=True,
28                     output_bps=8
29                 )
30                 img = PILImage.fromarray(rgb)
31
32         if img is None:
33             raise ValueError(f"Could not load image from {image_path}")
34
35         if greyscale:
36             img = img.convert('L')
37         else:
38             img = img.convert('RGB')
39
40         image_array = np.array(img, dtype=float)
41         return image_array
42
43     def _fast_fourier_transform(self, image_array):
44         if len(image_array.shape) == 2:
45             fourier_transformed = np.fft.fft2(image_array)
46             fourier_transformed_shifted = np.fft.fftshift(
```

```

        fourier_transformed)
    return fourier_transformed_shifted
else:
    height, width, channels = image_array.shape
    fourier_transformed_shifted = np.zeros((height, width,
                                             channels), dtype=complex)

for c in range(channels):
    fourier_transformed = np.fft.fft2(image_array[:, :, c])
    fourier_transformed_shifted[:, :, c] = np.fft.
        fftshift(fourier_transformed)

return fourier_transformed_shifted

def _mask(self, fourier_transformed, ratio=0.5):
    if len(fourier_transformed.shape) == 2:
        magnitude = np.abs(fourier_transformed)
        threshold = np.percentile(magnitude, 100 * (1 - ratio))

        mask = magnitude > threshold
        fourier_transformed_compressed = fourier_transformed *
            mask

    return fourier_transformed_compressed
else:
    height, width, channels = fourier_transformed.shape
    fourier_transformed_compressed = np.zeros_like(
        fourier_transformed)

    for c in range(channels):
        magnitude = np.abs(fourier_transformed[:, :, c])
        threshold = np.percentile(magnitude, 100 * (1 -
            ratio))

        channel_mask = magnitude > threshold
        fourier_transformed_compressed[:, :, c] =
            fourier_transformed[:, :, c] * channel_mask

    return fourier_transformed_compressed

def _inverse_fast_fourier_transform(self, fourier_transformed):
    if len(fourier_transformed.shape) == 2:
        img_back = np.fft.ifft2(np.fft.ifftshift(
            fourier_transformed)).real
        img_back = np.clip(img_back, 0, 255).astype(np.uint8)

    return img_back
else:
    height, width, channels = fourier_transformed.shape
    img_back = np.zeros((height, width, channels), dtype=np

```

```

        .uint8)

89
90     for c in range(channels):
91         channel_back = np.fft.ifft2(np.fft.ifftshift(
92             fourier_transformed[:, :, c])).real
93         img_back[:, :, c] = np.clip(channel_back, 0, 255).\
94             astype(np.uint8)

95
96     return img_back

97
98
99     def _save_image(self, image_array, output_path):
100        img = PILImage.fromarray(image_array)

101
102        ext = os.path.splitext(output_path)[1].lower()
103        save_kwargs = {}

104        if ext in ('.jpg', '.jpeg'):
105            # JPEG: set maximum quality, no subsampling
106            save_kwargs['quality'] = 100
107            save_kwargs['subsampling'] = 0
108        elif ext == '.png':
109            # PNG: no compression
110            save_kwargs['compress_level'] = 0

111        img.save(output_path, **save_kwargs)

112
113    def compress(self, output_path, ratio=0.5, greyscale=False):
114        if self.image_array is None or greyscale != (len(self.
115            image_array.shape) == 2):
116            self.image_array = self._load_image(self.image_path,
117                greyscale)

118            fourier_transformed = self._fast_fourier_transform(self.
119                image_array)
120            fourier_transformed_compressed = self._mask(
121                fourier_transformed, ratio)
122            compressed_image = self._inverse_fast_fourier_transform(
123                fourier_transformed_compressed)

124            self._save_image(compressed_image, output_path)

125
126    return compressed_image

```

Listing 7. Código para procesar todas las imágenes

```
1 import os
2 from pathlib import Path
3 from src import Image
4
5 RAW_EXTS = {
6     ".arw",
7     ".cr2",
8     ".nef",
9     ".dng",
10    ".raf",
11    ".rw2"
12 }
13 IMAGE_EXTS = RAW_EXTS | {
14     ".jpg",
15     ".jpeg",
16     ".png",
17     ".tif",
18     ".tiff",
19     ".bmp",
20     ".gif",
21     ".webp",
22 }
23
24
25 def compress_with_image_class(input_path, output_path,
26     compression_ratio=0.1, greyscale=False):
27     img = Image(input_path)
28
29     img.compress(output_path, ratio=compression_ratio,
30                 greyscale=greyscale)
31
32
33 if __name__ == "__main__":
34     input_dir = "images/raw"
35     output_dir = "images/compressed"
36     output_color_dir = "images/compressed_color"
37
38     os.makedirs(output_dir, exist_ok=True)
39     os.makedirs(output_color_dir, exist_ok=True)
40
41     for file in Path(input_dir).iterdir():
42         if not file.is_file():
43             continue
44         if file.suffix.lower() not in IMAGE_EXTS:
45             continue
46
47         out_suffix = '.jpg' if file.suffix.lower() in
48             RAW_EXTS else file.suffix.lower()
```

```
48     out_file = f"{output_dir}/{file.stem}_compressed{  
49         out_suffix}"  
50     out_file_color = f"{output_color_dir}/{file.stem}  
51         _compressed{out_suffix}"  
52  
53     try:  
54         compress_with_image_class(str(file), str(  
55             out_file), compression_ratio=0.01,  
56             greyscale=True)  
57         compress_with_image_class(str(file), str(  
58             out_file_color), compression_ratio=0.01,  
59             greyscale=False)  
60     except Exception as e:  
61         print(f"Error processing {file.name}: {e}")  
62         continue  
63  
64     print("Test completed successfully!")
```