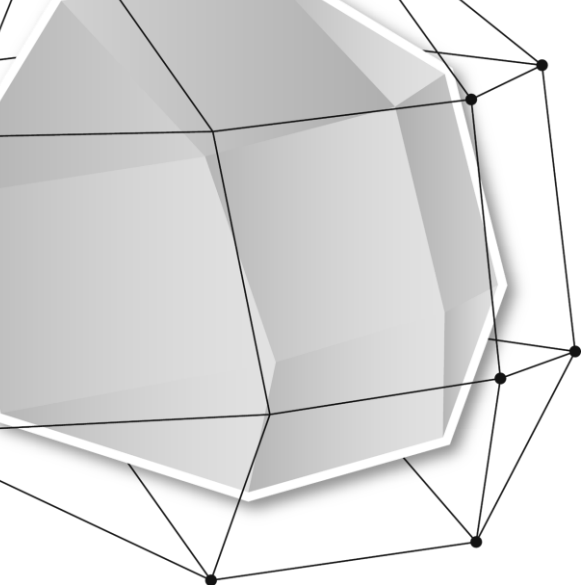
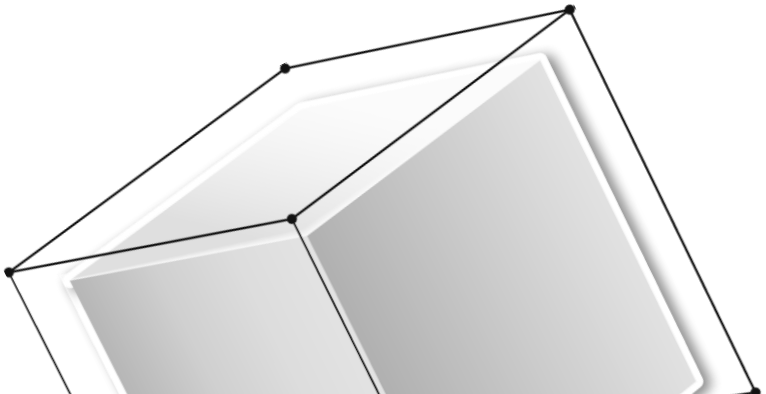




Optimización de fases de hologramas mediante UNET: Una Arquitectura de Aprendizaje Profundo en Acción

César Antonio Hoyos Peláez

Aprendizaje estadístico

01

¿Qué es la holografía?

02

Generación del conjunto de datos

03

Arquitectura UNET

04

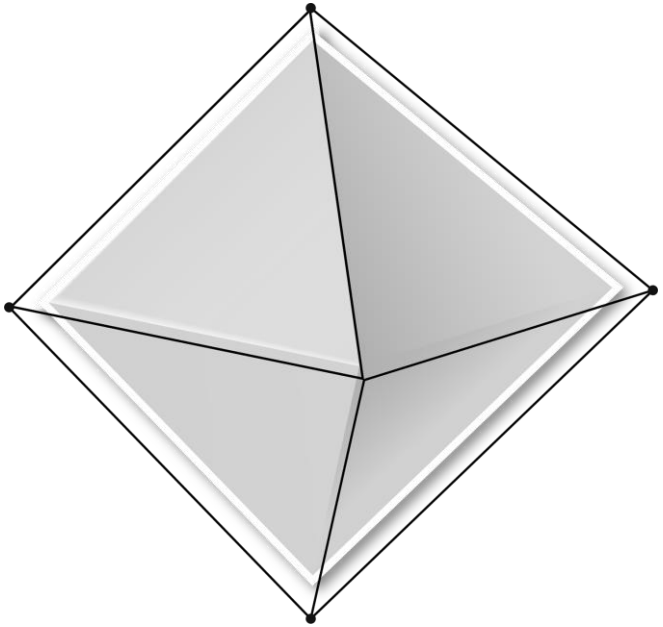
Como usar la red UNET para optimizar fases de hologramas

05

Resultados y conclusiones

01

¿Qué es la holografía?



Holografía

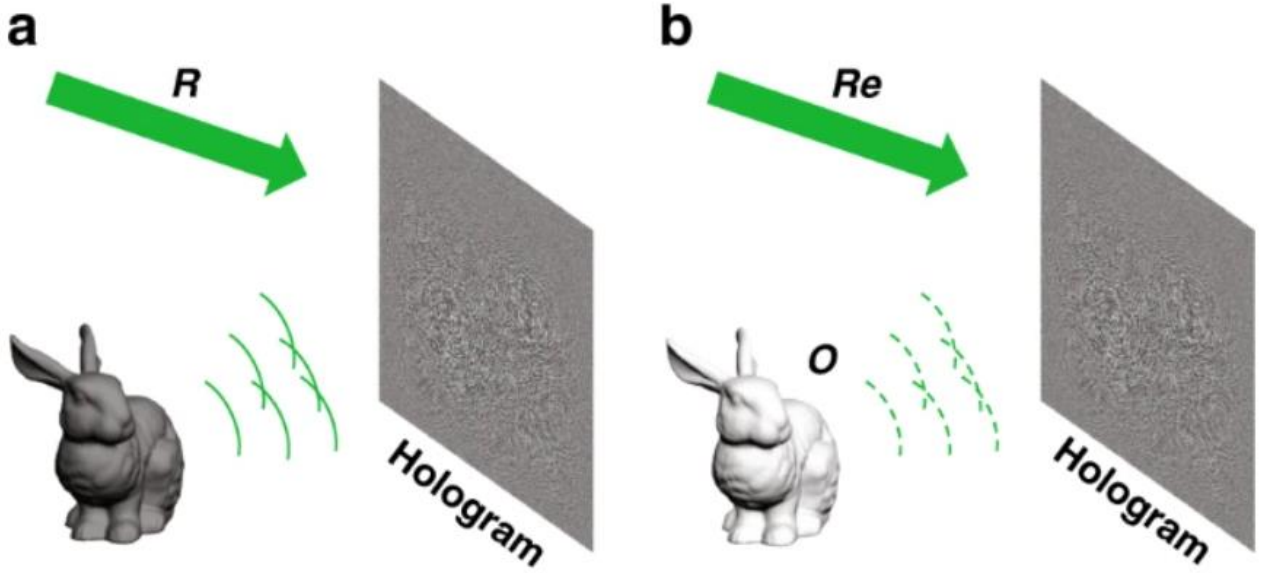


Figura 1. Esquema del proceso de registro y reconstrucción de un holograma. (a) registro, (b) reconstrucción[1].

Contexto y motivación.

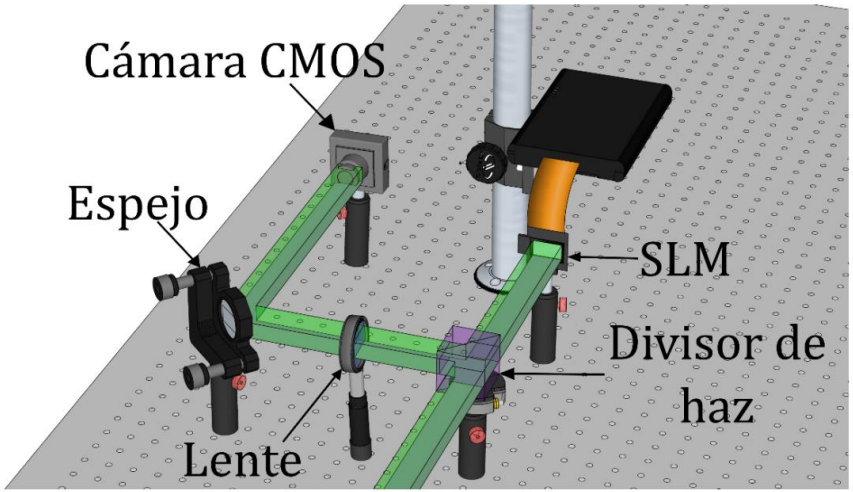


Figura 2. Esquema experimental para reconstruir hologramas de Fourier haciendo uso de un modulador espacial de luz (SLM) por reflexión [2].

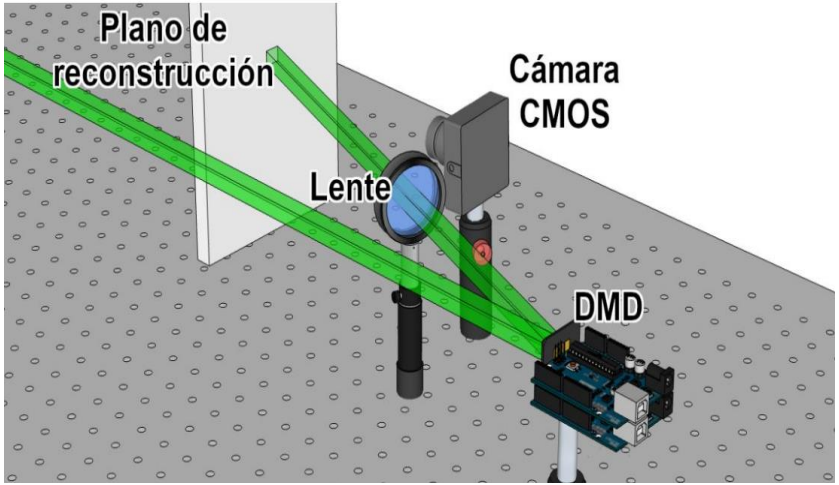
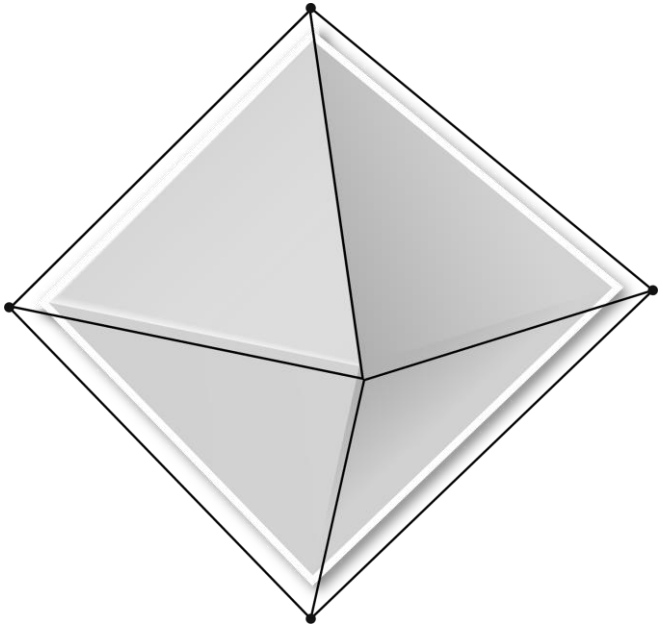


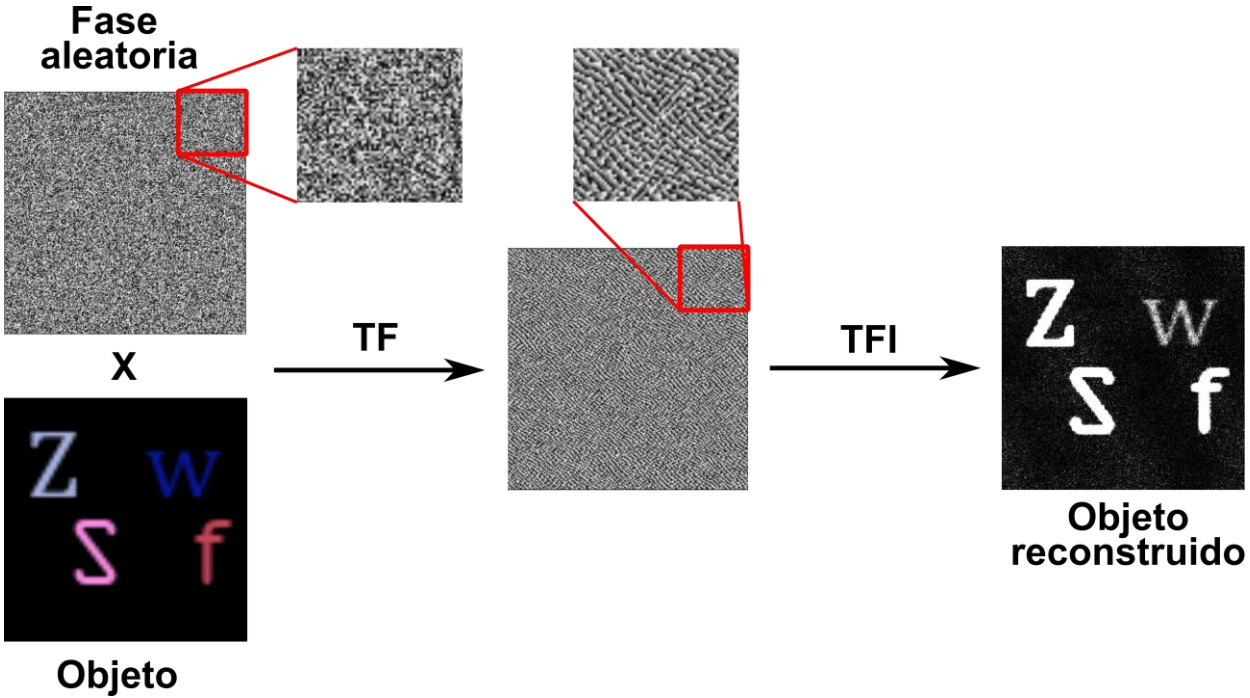
Figura 3. Esquema experimental para reconstruir hologramas de Fourier haciendo uso de un dispositivo digital de microespejos (DMD) [3].

02

Generación del conjunto de datos



Hologramas de fase sin optimizar



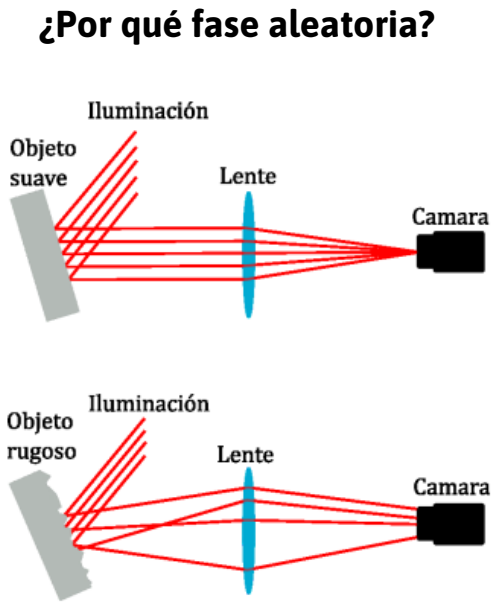
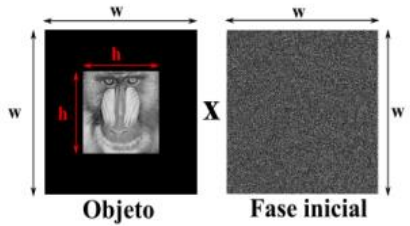


Figura 4. Esquema de algoritmo para generar un holograma a partir de una fase aleatoria.

Algoritmo de Gerchberg–Saxton



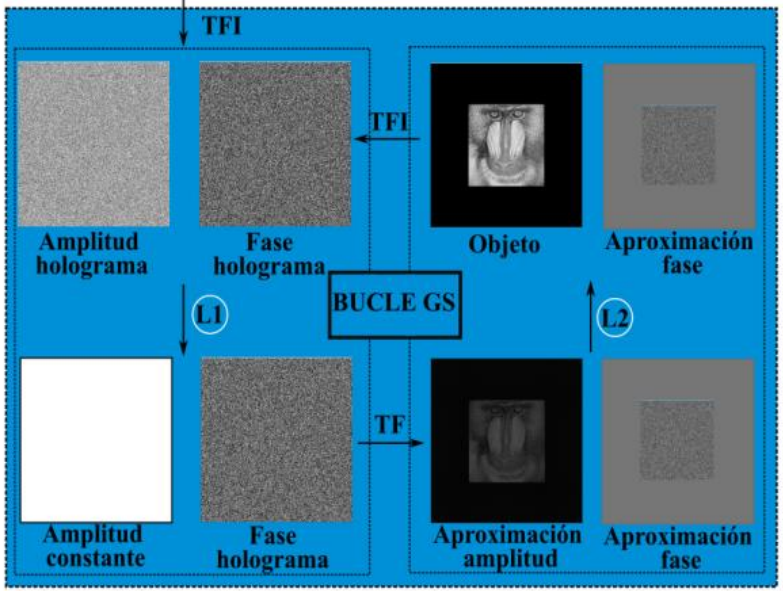
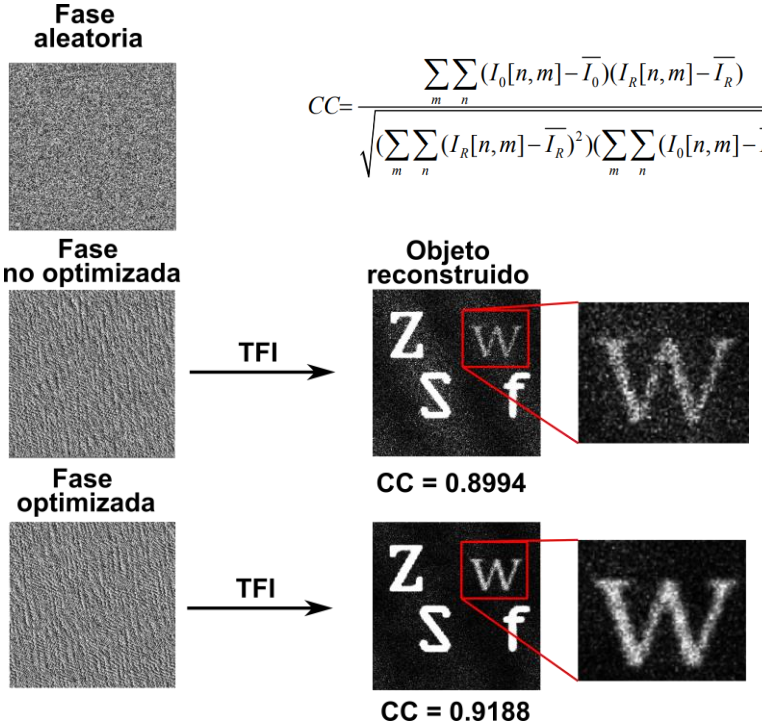


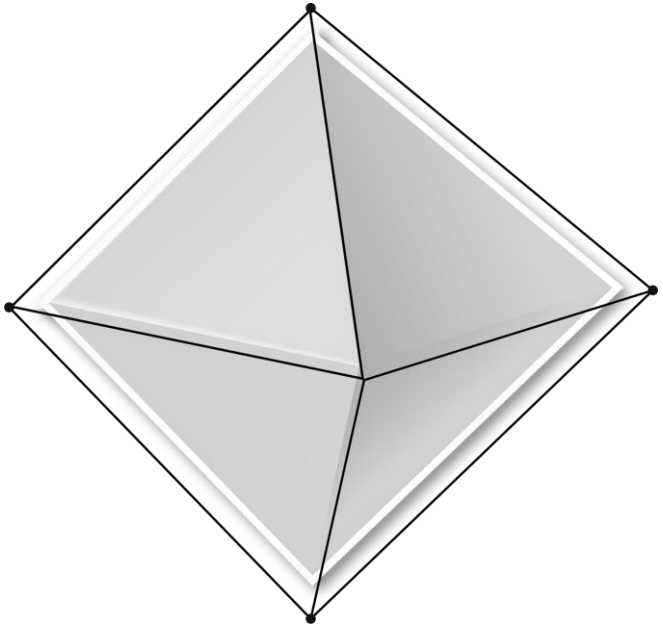
Figura 5. Esquema para realizar el algoritmo de Gerchberg-Saxton en el dominio de Fourier[4].



$$CC = \frac{\sum_m \sum_n (I_0[n,m] - \overline{I_0})(I_R[n,m] - \overline{I_R})}{\sqrt{(\sum_m \sum_n (I_R[n,m] - \overline{I_R})^2)(\sum_m \sum_n (I_0[n,m] - \overline{I_0})^2)}}$$

03

Arquitectura UNET



UNET

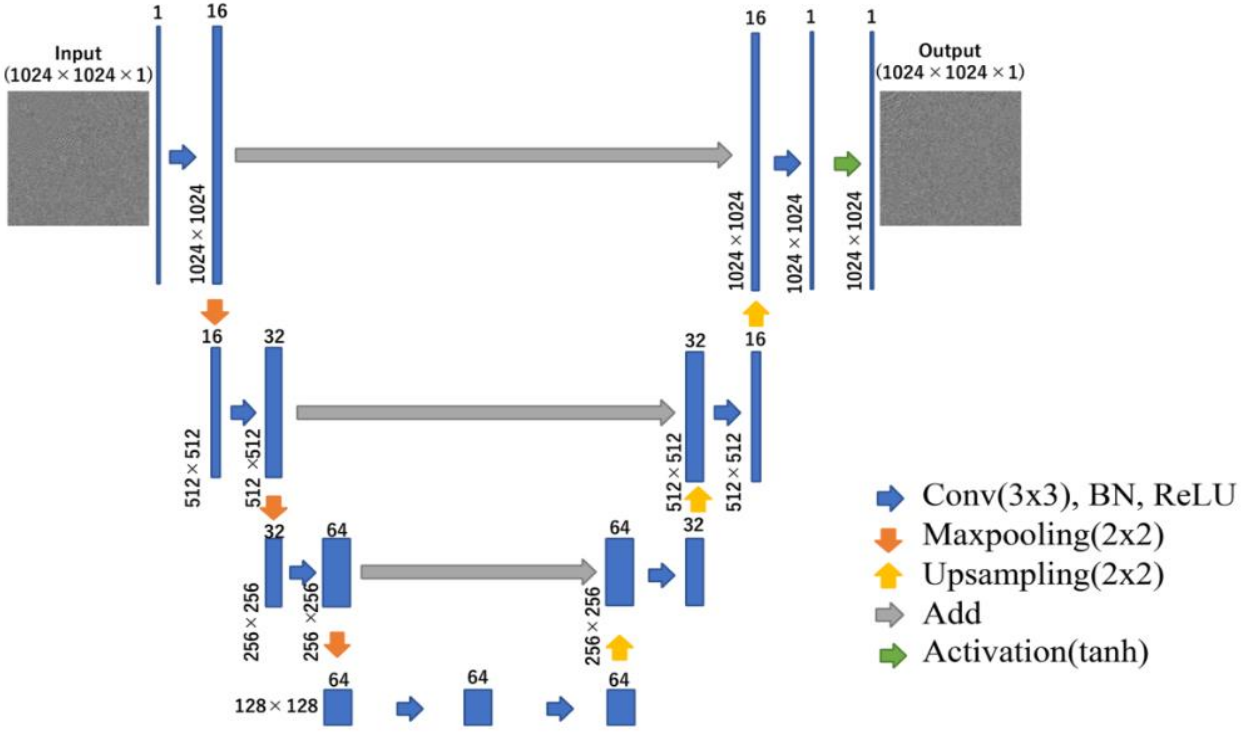
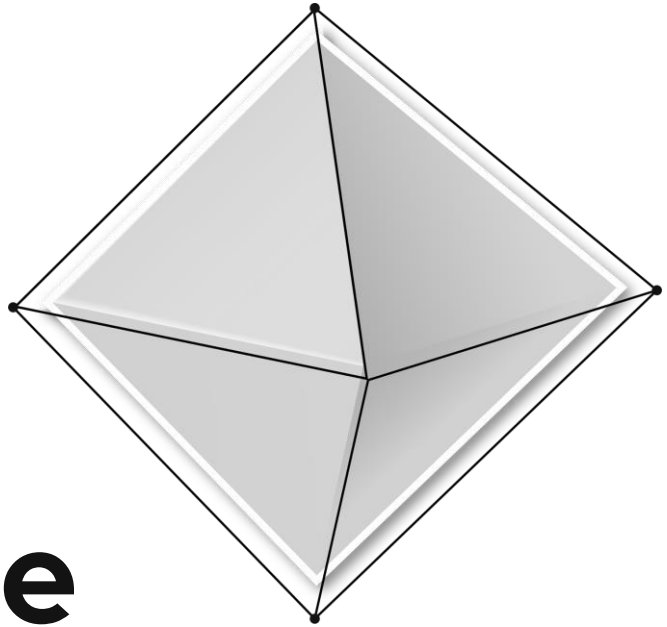


Figura 6. Arquitectura de la red UNET aplicada a hologramas[5].

04

Como usar la red UNET para optimizar fases de hologramas



Idea de funcionamiento

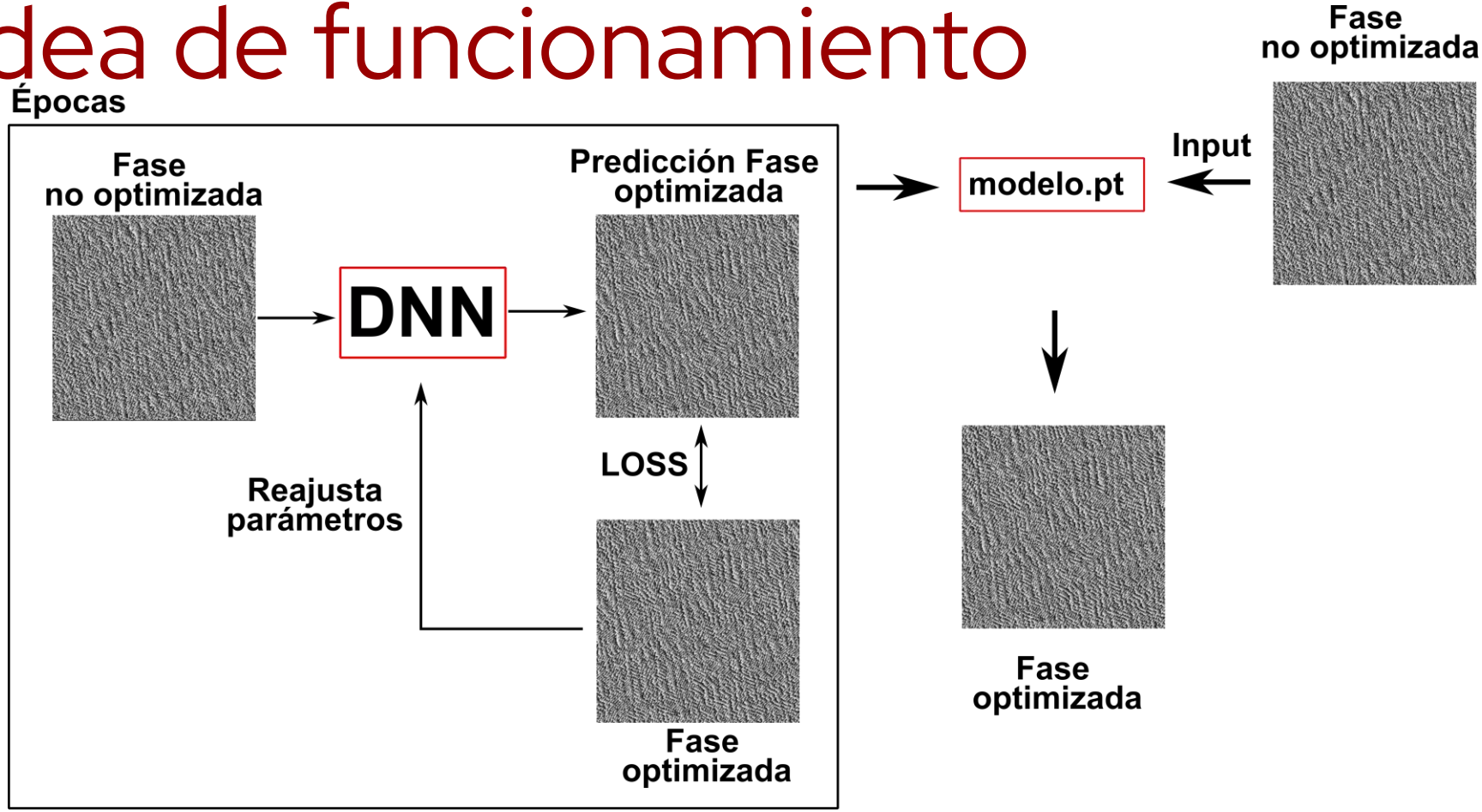
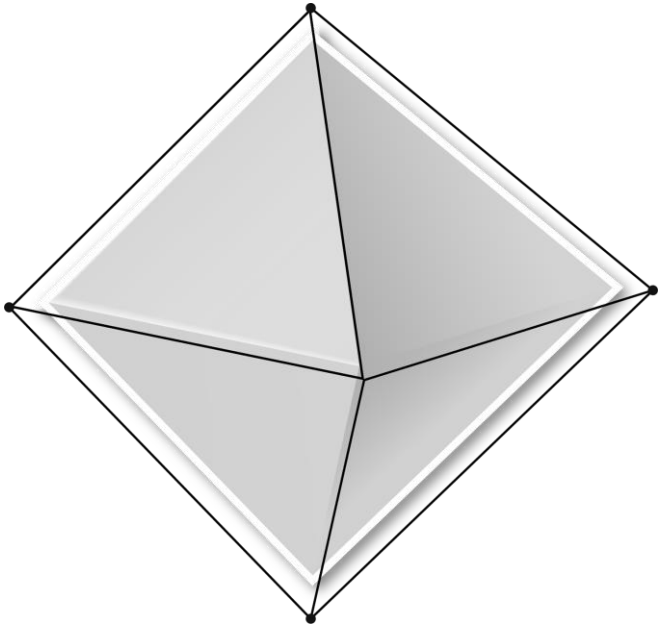


Figura 7. Funcionamiento del algoritmo para extraer el modelo y ejecución del modelo.

05

Resultados



Prueba #1: solo una imagen

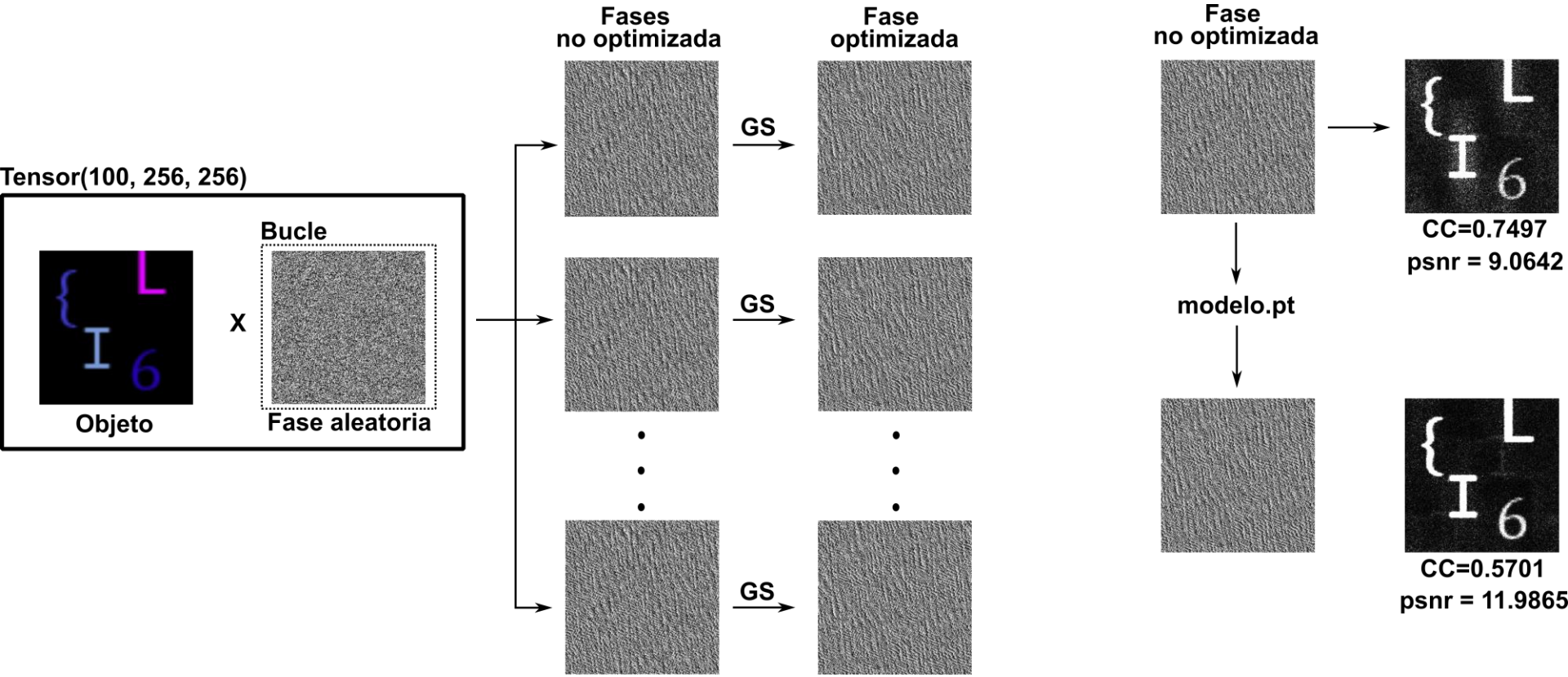


Figura 8. Resultados para una sola imagen, se obtiene overfitting.

Prueba #1: solo una imagen

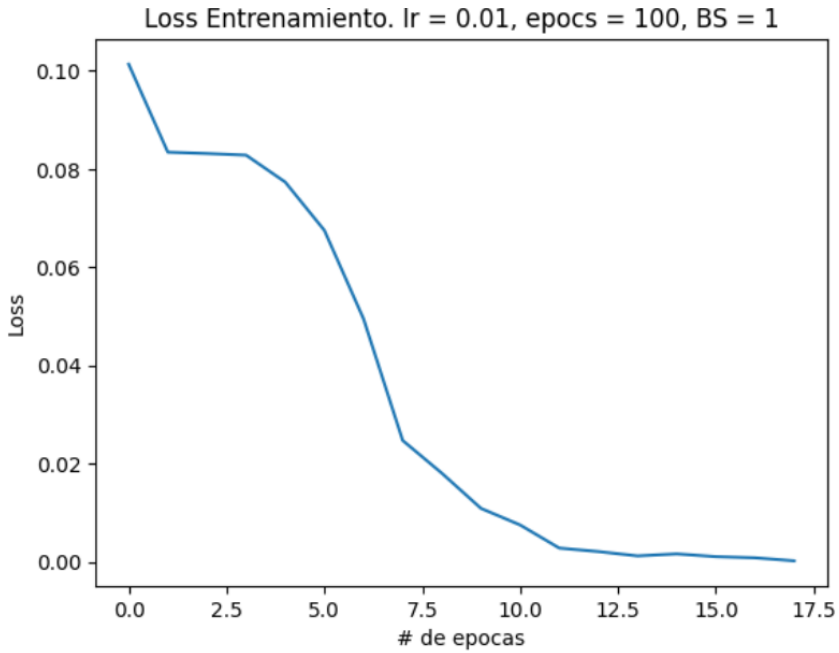


Figura 9. Función de pérdida para los datos de entrenamiento.

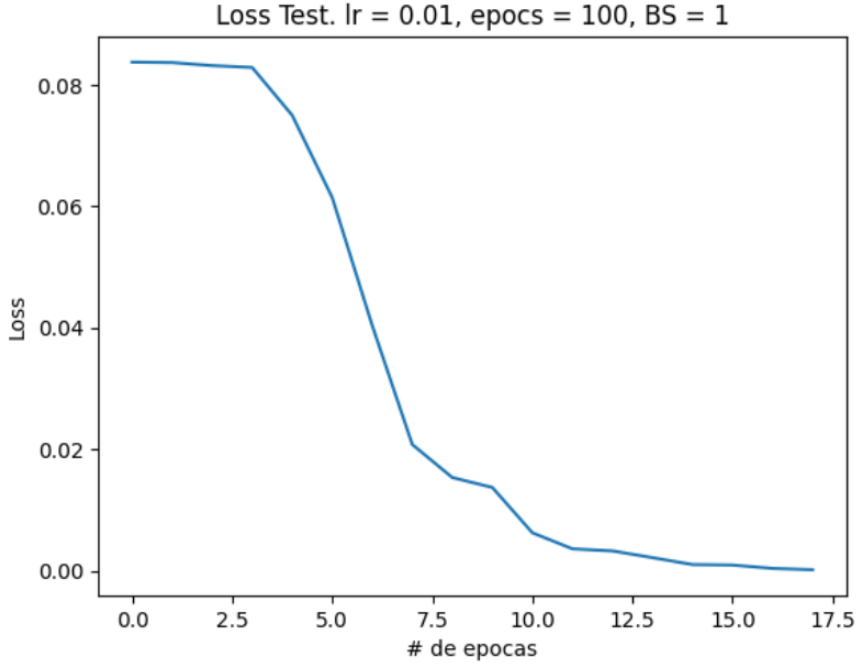


Figura 10. Función de pérdida para los datos de test.

Prueba #2: 100 imágenes

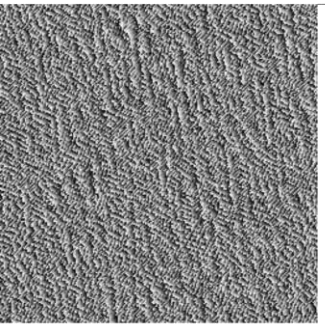
```python
tensor_ima = torch.zeros(dimIma * cantidadMascaras, size, size) # Dimension de 1000x256x256
tensor_fase = torch.zeros(dimIma * cantidadMascaras, size, size) # Dimension de 1000x256x256
j = 0
for k in range(0, dimIma):
    for i in range(0, cantidadMascaras):
        R = im.open(path+ f"{k}.bmp") # Se lee la misma imagen
        R = R.convert('L') # Se asegura que este en escala de grises
        R = R.resize((size, size)) # Se redimensiona al tamaño de la red
        R = np.asarray(R) # Se convierte en array de numpy
        R = R/255 # Se normaliza
        tensor_ima[j,:,:] = torch.from_numpy(R) # Se empieza a llenar el tensor con cada una de las imagenes
        MA = np.exp(1j*2*np.pi*np.random.rand(int(size), int(size))) # Se genera 100 fases aleatorias
        tensor_fase[j,:,:] = torch.from_numpy(MA)
        print(j)
        j += 1
```

```python
Input = tensor_ima * tensor_fase

U = torch.fft.ifftshift(torch.fft.ifft2(torch.fft.fftshift(Input)))
U = torch.exp(1j*torch.angle(U)) # Se extrae la fase de la operacion anterior y se hace cte la amplitud
```

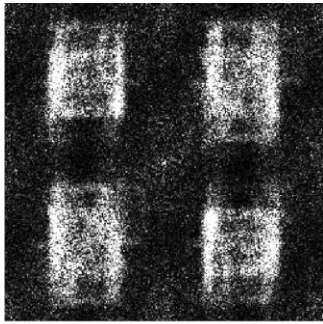
```python
MA = torch.from_numpy(np.exp(1j*2*np.pi*np.random.rand(int(size), int(size)))) # mascara aleatoria

Input = tensor_ima * MA # Se multiplica cada una de las imagenes por la mascara aleatoria


# GS
for i in range(0, iterations):
    U = torch.fft.ifftshift(torch.fft.ifft2(torch.fft.fftshift(Input)))
    U = torch.exp(1j*torch.angle(U)) # Se extrae la fase de la operacion anterior y se hace cte la amplitud
    ug = torch.fft.ifftshift(torch.fft.fft2(torch.fft.fftshift(U))) # Se hace TF
    Input = tensor_ima*torch.exp(1j*torch.angle(ug));
    print(i)
```

Figura 11. Fragmento de código, parte izquierda fases sin optimizar, parte derecha fases introducidas al algoritmo GS.

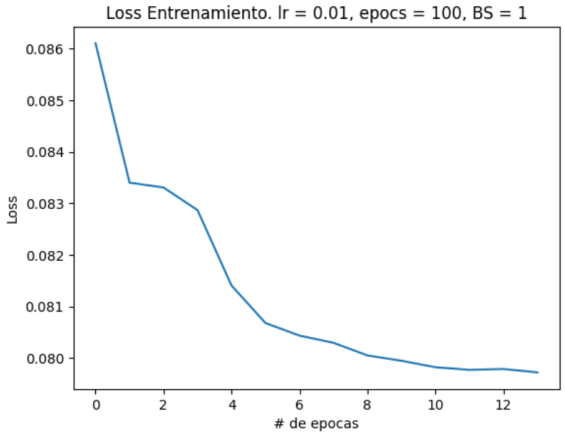
Prueba #2: 100 imágenes



input → modelo.pt → output



Objeto reconstruido



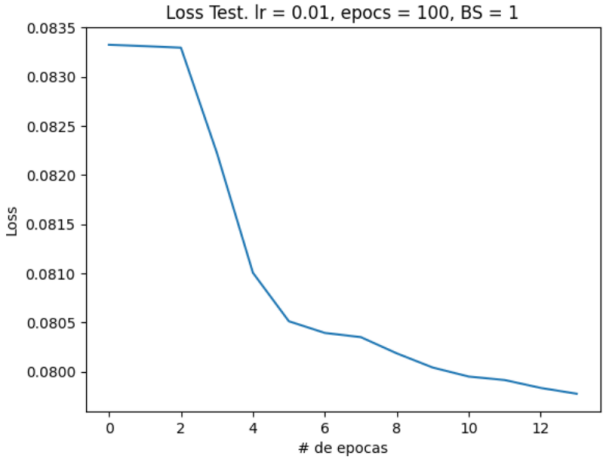
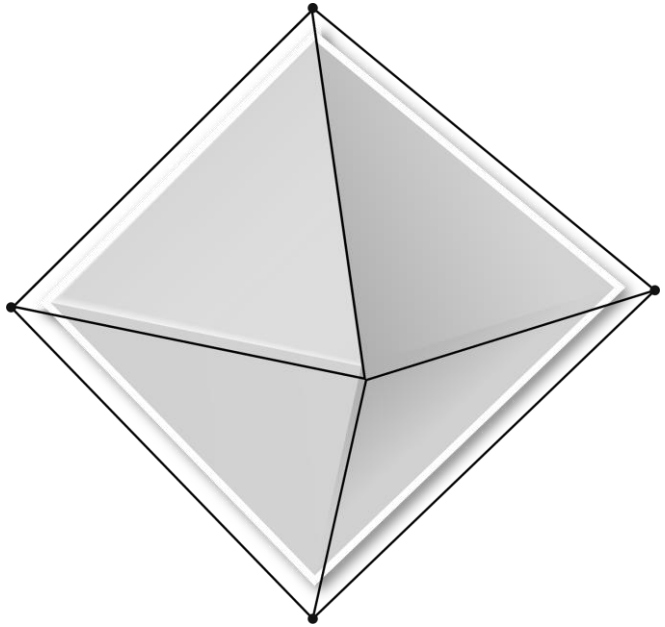


Figura 12. Gráficas de la función de pérdida en el caso de 100 imágenes cada una con 20 fases aleatorias.

05

Conclusiones



Conclusiones

En este estudio, se ha implementado la red neuronal UNET con el objetivo de optimizar fases de hologramas. Los datos empleados para el entrenamiento consisten en fases no optimizadas, mientras que para la clasificación se utilizan fases derivadas del algoritmo de Gerchberg-Saxton. Los hallazgos indican que, al aplicar la red a una sola imagen utilizando 100 máscaras aleatorias como datos de entrada, se logra un desempeño satisfactorio, aunque se observa un problema de sobreajuste (overfitting). Por otro lado, al incrementar el conjunto de datos a 100 imágenes, cada una con 20 máscaras aleatorias, se identifica un problema de subajuste (underfitting), resultando en un rendimiento deficiente de la red.

Referencias

[1] Pi, D., Liu, J., & Wang, Y. (2022). Review of computer-generated hologram algorithms for color dynamic holographic three-dimensional display. Light: Science & Applications, 11(1), 231.
[2] Zea, A. V. (2018). Holografía digital 3D y su extension a la encriptación óptica, la compresibilidad y la visualización de la información. Tesis doctoral. Universidad Nacional de la Plata, Argentina.
[3] Quinchia, S. B. (2021). Hologramas binarios de amplitud con mascaras aleatorias de fase optimizadas. Tesis de pregrado. Universidad de Antioquia, Colombia.
[4] Zalevsky, Z., Mendlovic, D., & Dorsch, R. G. (1996). Gerchberg–Saxton algorithm applied in the fractional Fourier or the Fresnel domain. Optics Letters, 21(12), 842-844.
[5] Ishii, Y., Shimobaba, T., Blinder, D., Birnbaum, T., Schelkens, P., Kakue, T., & Ito, T. (2022). Optimization of phase-only holograms calculated with scaled diffraction calculation through deep neural networks. Applied Physics B, 128(2), 22.

GRACIAS!

Do you have any questions?

cantonio.hoyos@udea.edu.co