T4_Experimentando_con_Datos_Faltantes

May 27, 2023

1 Tarea 4 - Experimentando con Datos Faltantes

1.0.1 Héctor Hibran Tapia Fernández - A01661114

Hecho en Google Colab:

[ ]:
```
!pip install numpy
!pip install scipy
!pip install matplotlib
```

Importamos las librerías:

[2]:
```
import numpy as np
import scipy.interpolate
import numpy.linalg as la
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy.interpolate import griddata
```

1.0.2 1. Selecciona dos pinturas de tu interés. Estos deben ser de tamaño mediano (alrededor de 300 x 400 pixeles) y de formato PNG. Deben ser muy diferentes (una contiene estructuras, la otra no).

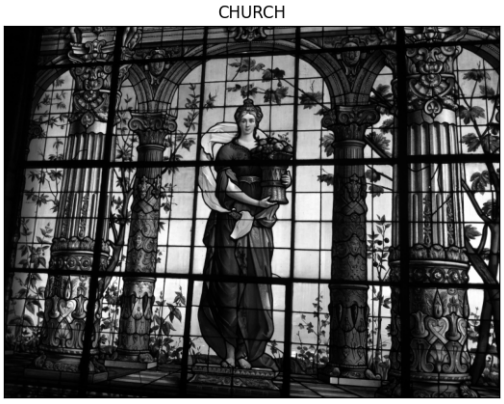
[3]:
```
# Lee las imágenes
image1 = mpimg.imread('/content/BIRD_007.png')  # 800x600 = 480,000pxls
image2 = mpimg.imread('/content/CHURCH_001.png') # 800x598 = 478,400pxls

# Crea una figura con dos subplots
fig, axs = plt.subplots(1, 2, figsize = (15, 5))

# Muestra las imágenes
axs[0].imshow(image1)
axs[0].set_title('BIRD')

axs[1].imshow(image2)
```

```
axs[1].set_title('CHURCH')

for ax in axs:
    ax.set_xticks([])
    ax.set_yticks([])

plt.show()
```

1.0.3 2. Ábrelas en Python usando. Recuerda que el formato es RBG, haz un promedio para convertirlo en escala de grises.

[4]:
```
# Comprueba si las imágenes tienen canal alfa y de ser así lo descarta
if image1.shape[2] == 4:
    image1 = image1[:, :, :3]
if image2.shape[2] == 4:
    image2 = image2[:, :, :3]

# Calcula el promedio de los canales RGB
grayscale_img1 = np.mean(image1, axis = 2)
grayscale_img2 = np.mean(image2, axis = 2)

# Crea una figura con dos subplots
fig, axs = plt.subplots(1, 2, figsize = (15, 5))

# Enseña las imagenes en escala de grises
axs[0].imshow(grayscale_img1, cmap = plt.cm.gray)
axs[0].set_title('BIRD')

axs[1].imshow(grayscale_img2, cmap = plt.cm.gray)
axs[1].set_title('CHURCH')

for ax in axs:
```

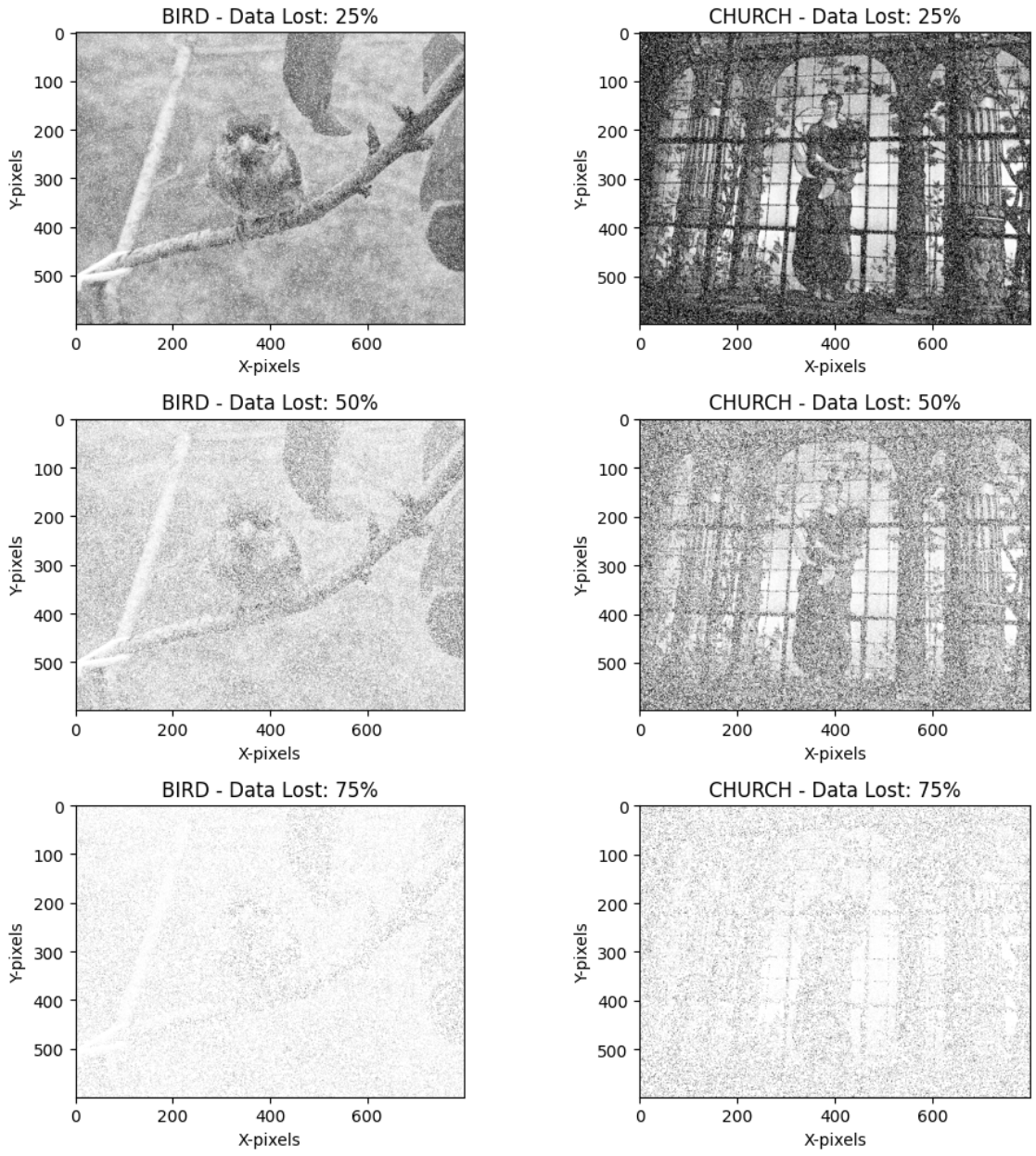
```
    ax.set_xticks([])
    ax.set_yticks([])

plt.show()
```



1.0.4 3. Ahora vamos a simular perder 25%, 50% y 75% de los datos. Para esto crea una matriz aleatoria de selección que solo conserve las posiciones de los pixeles que van a sobrevivir. Los que mueren, reemplázalos por np.NaN.

[5]:
```
# Convierte las imágenes a floats
grayscale_img1 = grayscale_img1.astype(float)
grayscale_img2 = grayscale_img2.astype(float)

# Crea la figura para que entren los subplots
fig, axs = plt.subplots(3, 2, figsize = (10, 10))

probabilities = [(0.25, 0.75), (0.5, 0.5), (0.75, 0.25)] # Tiene que dar igual␣
 ↪a 1 la suma.

for i, (p_lost, p_win) in enumerate(probabilities):

    # Crea una matriz de selección aleatoria con el 25%, 50% y 75% de los␣
 ↪píxeles
    selection_matrix1 = np.random.choice([0, 1], size = grayscale_img1.shape, p␣
 ↪= [p_lost, p_win]) # Tiene que dar igual a 1 la suma.
    selection_matrix2 = np.random.choice([0, 1], size = grayscale_img2.shape, p␣
 ↪= [p_lost, p_win]) # Tiene que dar igual a 1 la suma.

    # Reemplaza los píxeles "muertos" con np.NaN
    grayscale_img1 = np.where(selection_matrix1, grayscale_img1, np.NaN)
    grayscale_img2 = np.where(selection_matrix2, grayscale_img2, np.NaN)
```

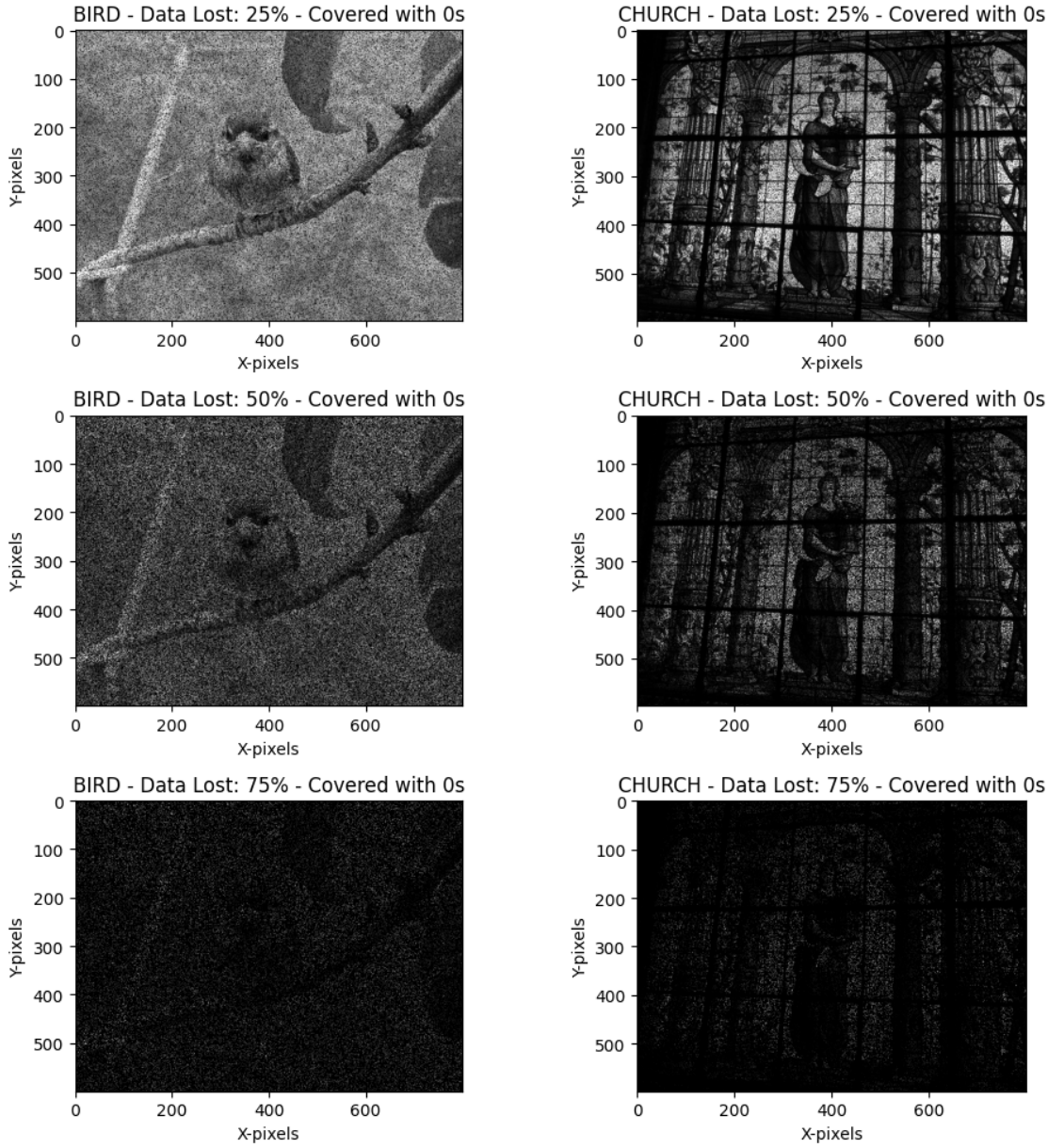
```
    # Reemplaza np.NaN con un valor específico (por ejemplo, 0 para negro, 1␣
 ↪para blanco)
    grayscale_img1 = np.nan_to_num(grayscale_img1, nan = 1.0)
    grayscale_img2 = np.nan_to_num(grayscale_img2, nan = 1.0)

    # Subplots y formato
    axs[i, 0].imshow(grayscale_img1, cmap = plt.cm.gray)
    axs[i, 0].set_title('BIRD - Data Lost: {:.0%}'.format(p_lost))
    axs[i, 0].set_xlabel('X-pixels')
    axs[i, 0].set_ylabel('Y-pixels')

    axs[i, 1].imshow(grayscale_img2, cmap = plt.cm.gray)
    axs[i, 1].set_title('CHURCH - Data Lost: {:.0%}'.format(p_lost))
    axs[i, 1].set_xlabel('X-pixels')
    axs[i, 1].set_ylabel('Y-pixels')

plt.tight_layout()
plt.show()
```



1.0.5 4. Reemplaza todos los valores faltantes con ceros y grafica.

[6]:
```
# Crea nuevas medias, evita que se sobreescriban las gráficas, y así se generen␣
↪nuevas.
grayscale_img3 = np.mean(image1, axis = 2)
grayscale_img4 = np.mean(image2, axis = 2)

# Convierte las imágenes a floats
grayscale_img3 = grayscale_img3.astype(float)
```

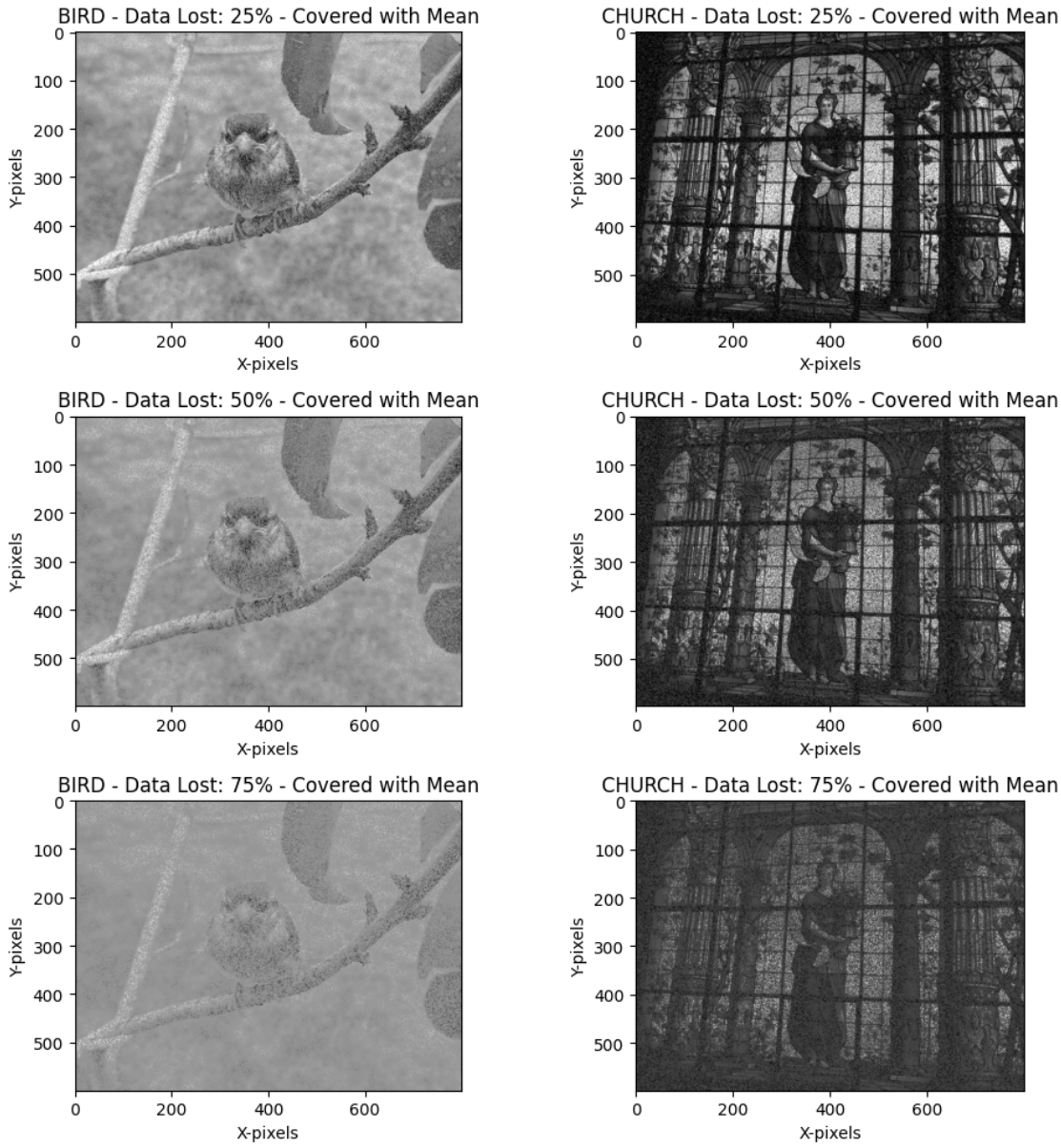
```
grayscale_img4 = grayscale_img4.astype(float)

# Crea la figura para que entren los subplots
fig, axs = plt.subplots(3, 2, figsize = (10, 10))

probabilities = [(0.25, 0.75), (0.5, 0.5), (0.75, 0.25)]

for i, (p_lost, p_win) in enumerate(probabilities):

    # Crea una matriz de selección aleatoria con el 25%, 50% y 75% de los␣
 ↪píxeles
    selection_matrix3 = np.random.choice([0, 1], size = grayscale_img3.shape, p␣
 ↪= [p_lost, p_win]) # Tiene que dar igual a 1 la suma.
    selection_matrix4 = np.random.choice([0, 1], size = grayscale_img4.shape, p␣
 ↪= [p_lost, p_win]) # Tiene que dar igual a 1 la suma.

    # Reemplaza los píxeles "muertos" con np.NaN
    grayscale_img3 = np.where(selection_matrix3, grayscale_img3, np.NaN)
    grayscale_img4 = np.where(selection_matrix4, grayscale_img4, np.NaN)

    # Reemplaza np.NaN con un valor específico (por ejemplo, 0 para negro, 1␣
 ↪para blanco)
    grayscale_img3 = np.nan_to_num(grayscale_img3, nan = 0.0)
    grayscale_img4 = np.nan_to_num(grayscale_img4, nan = 0.0)

    # Subplots y formato
    axs[i, 0].imshow(grayscale_img3, cmap = plt.cm.gray)
    axs[i, 0].set_title('BIRD - Data Lost: {:.0%} - Covered with 0s'.
 ↪format(p_lost))
    axs[i, 0].set_xlabel('X-pixels')
    axs[i, 0].set_ylabel('Y-pixels')

    axs[i, 1].imshow(grayscale_img4, cmap = plt.cm.gray)
    axs[i, 1].set_title('CHURCH - Data Lost: {:.0%} - Covered with 0s'.
 ↪format(p_lost))
    axs[i, 1].set_xlabel('X-pixels')
    axs[i, 1].set_ylabel('Y-pixels')

plt.tight_layout()
plt.show()
```



1.0.6 5. Cambia los valores faltantes por la media de los datos que no perdiste.

[7]:
```
# Crea nuevas medias, evita que se sobreescriban las gráficas, y así se generen␣
↪nuevas.
grayscale_img5 = np.mean(image1, axis = 2)
grayscale_img6 = np.mean(image2, axis = 2)

# Convierte las imágenes a floats
grayscale_img5 = grayscale_img5.astype(float)
grayscale_img6 = grayscale_img6.astype(float)
```

```
# Crea la figura para que entren los subplots
fig, axs = plt.subplots(3, 2, figsize = (10, 10))

probabilities = [(0.25, 0.75), (0.5, 0.5), (0.75, 0.25)]

for i, (p_lost, p_win) in enumerate(probabilities):

    # Crea una matriz de selección aleatoria con el 25%, 50% y 75% de los␣
 ↪píxeles
    selection_matrix5 = np.random.choice([0, 1], size = grayscale_img5.shape, p␣
 ↪= [p_lost, p_win])
    selection_matrix6 = np.random.choice([0, 1], size = grayscale_img6.shape, p␣
 ↪= [p_lost, p_win])

    # Reemplaza los píxeles "muertos" con np.NaN
    grayscale_img5_with_nan = np.where(selection_matrix5, grayscale_img5, np.
 ↪NaN)
    grayscale_img6_with_nan = np.where(selection_matrix6, grayscale_img6, np.
 ↪NaN)

    # Calculate the mean of survived pixels
    mean5 = np.nanmean(grayscale_img5_with_nan)
    mean6 = np.nanmean(grayscale_img6_with_nan)

    # Reemplaza np.NaN con el valor medio de los píxeles sobrevivientes
    grayscale_img3_filled = np.nan_to_num(grayscale_img5_with_nan, nan = mean5)
    grayscale_img4_filled = np.nan_to_num(grayscale_img6_with_nan, nan = mean6)

    # Subplots y formato
    axs[i, 0].imshow(grayscale_img3_filled, cmap = plt.cm.gray)
    axs[i, 0].set_title('BIRD - Data Lost: {:.0%} - Covered with Mean'.
 ↪format(p_lost))
    axs[i, 0].set_xlabel('X-pixels')
    axs[i, 0].set_ylabel('Y-pixels')

    axs[i, 1].imshow(grayscale_img4_filled, cmap = plt.cm.gray)
    axs[i, 1].set_title('CHURCH - Data Lost: {:.0%} - Covered with Mean'.
 ↪format(p_lost))
    axs[i, 1].set_xlabel('X-pixels')
    axs[i, 1].set_ylabel('Y-pixels')

plt.tight_layout()
plt.show()
```

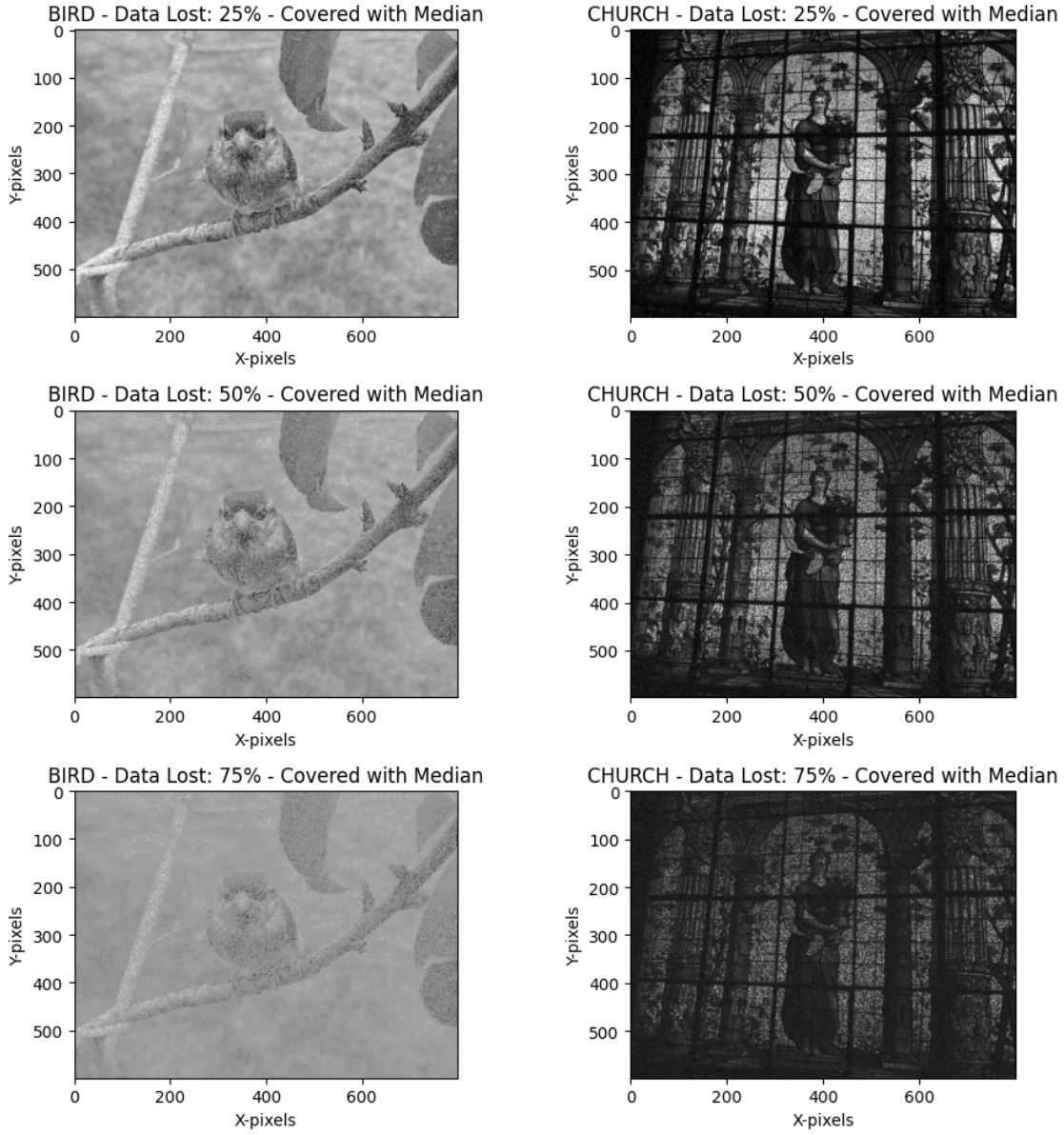


1.0.7 6. Cambia los valores faltantes por la mediana de los datos que no perdiste.

[8]:
```
# Crea nuevas medias, evita que se sobreescriban las gráficas, y así se generen␣
↪nuevas.
grayscale_img7 = np.mean(image1, axis = 2)
grayscale_img8 = np.mean(image2, axis = 2)

# Convierte las imágenes a floats
grayscale_img7 = grayscale_img7.astype(float)
grayscale_img8 = grayscale_img8.astype(float)
```

```
# Crea la figura para que entren los subplots
fig, axs = plt.subplots(3, 2, figsize = (10, 10))

probabilities = [(0.25, 0.75), (0.5, 0.5), (0.75, 0.25)]

for i, (p_lost, p_win) in enumerate(probabilities):

    # Crea una matriz de selección aleatoria con el 25%, 50% y 75% de los␣
 ↪píxeles
    selection_matrix7 = np.random.choice([0, 1], size = grayscale_img7.shape, p␣
 ↪= [p_lost, p_win])
    selection_matrix8 = np.random.choice([0, 1], size = grayscale_img8.shape, p␣
 ↪= [p_lost, p_win])

    # Reemplaza los píxeles "muertos" con np.NaN
    grayscale_img7_with_nan = np.where(selection_matrix7, grayscale_img7, np.
 ↪NaN)
    grayscale_img8_with_nan = np.where(selection_matrix8, grayscale_img8, np.
 ↪NaN)

    # Calcula la mediana de píxeles que sobrevivieron
    median7 = np.median(grayscale_img7_with_nan[~np.
 ↪isnan(grayscale_img7_with_nan)])
    median8 = np.median(grayscale_img8_with_nan[~np.
 ↪isnan(grayscale_img8_with_nan)])

    # Reemplaza np.NaN con el valor mediano de los píxeles sobrevivientes
    grayscale_img7_filled = np.nan_to_num(grayscale_img7_with_nan, nan =␣
 ↪median7)
    grayscale_img8_filled = np.nan_to_num(grayscale_img8_with_nan, nan =␣
 ↪median8)

    # Subplots y formato
    axs[i, 0].imshow(grayscale_img7_filled, cmap = plt.cm.gray)
    axs[i, 0].set_title('BIRD - Data Lost: {:.0%} - Covered with Median'.
 ↪format(p_lost))
    axs[i, 0].set_xlabel('X-pixels')
    axs[i, 0].set_ylabel('Y-pixels')

    axs[i, 1].imshow(grayscale_img8_filled, cmap = plt.cm.gray)
    axs[i, 1].set_title('CHURCH - Data Lost: {:.0%} - Covered with Median'.
 ↪format(p_lost))
    axs[i, 1].set_xlabel('X-pixels')
    axs[i, 1].set_ylabel('Y-pixels')
```

```
plt.tight_layout()
plt.show()
```



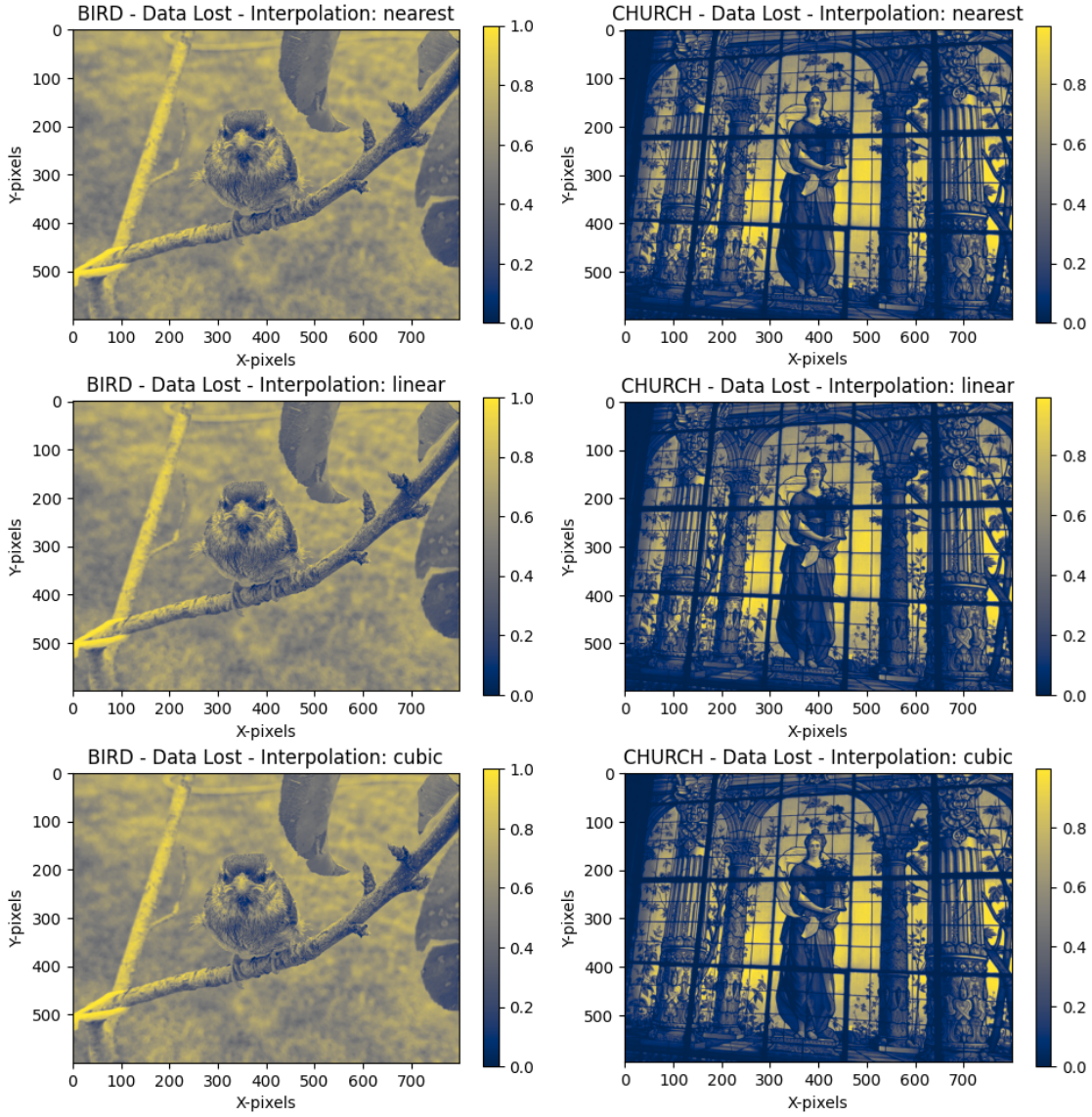
1.0.8 7. Estos datos tienen una estructura espacial.

Usa interpolación: del vecino más cercano, lineal, cúbica.

[9]:
```
# Crea nuevas medias, evita que se sobreescriban las gráficas, y así se generen␣
↪nuevas.
grayscale_img_9 = np.mean(image1, axis = 2)
grayscale_img_10 = np.mean(image2, axis = 2)
```

```
# Convierte las imágenes a floats
grayscale_img_9 = grayscale_img_9.astype(float)
grayscale_img_10 = grayscale_img_10.astype(float)

# Crea una cuadrícula de coordenadas para los datos existentes
coords_9 = np.array(np.nonzero(~np.isnan(grayscale_img_9))).T
values_9 = grayscale_img_9[~np.isnan(grayscale_img_9)]
coords_10 = np.array(np.nonzero(~np.isnan(grayscale_img_10))).T
values_10 = grayscale_img_10[~np.isnan(grayscale_img_10)]

# Crea una cuadrícula de coordenadas para los puntos donde desea interpolar
grid_coords_9 = np.array(np.nonzero(np.isnan(grayscale_img_9))).T
grid_coords_10 = np.array(np.nonzero(np.isnan(grayscale_img_10))).T

# Crea la figura para que entren los subplots
fig, axs = plt.subplots(3, 2, figsize = (10, 10))

for i, method in enumerate(['nearest', 'linear', 'cubic']):

    # Interpola los datos usando los métodos
    filled_pixels_9 = griddata(coords_9, values_9, grid_coords_9, method =␣
 ↪method)
    filled_pixels_10 = griddata(coords_10, values_10, grid_coords_10, method =␣
 ↪method)

    # Cree una copia de las imágenes originales y rellena los píxeles NaN con␣
 ↪los valores interpolados
    grayscale_img9_filled = grayscale_img_9.copy()
    grayscale_img9_filled[np.isnan(grayscale_img_9)] = filled_pixels_9

    grayscale_img_10_filled = grayscale_img_10.copy()
    grayscale_img_10_filled[np.isnan(grayscale_img_10)] = filled_pixels_10

    # Subplots para BIRD.
    im_9 = axs[i, 0].imshow(grayscale_img9_filled, cmap = 'cividis')
    axs[i, 0].set_title('BIRD - Data Lost - Interpolation: {}'.format(method))
    axs[i, 0].set_xlabel('X-pixels')
    axs[i, 0].set_ylabel('Y-pixels')
    fig.colorbar(im_9, ax=axs[i, 0])

    # Subplots para CHURCH.
    im_10 = axs[i, 1].imshow(grayscale_img_10_filled, cmap = 'cividis')
    axs[i, 1].set_title('CHURCH - Data Lost - Interpolation: {}'.format(method))
    axs[i, 1].set_xlabel('X-pixels')
    axs[i, 1].set_ylabel('Y-pixels')
    fig.colorbar(im_10, ax = axs[i, 1])
```

```
plt.tight_layout()
plt.show()
```



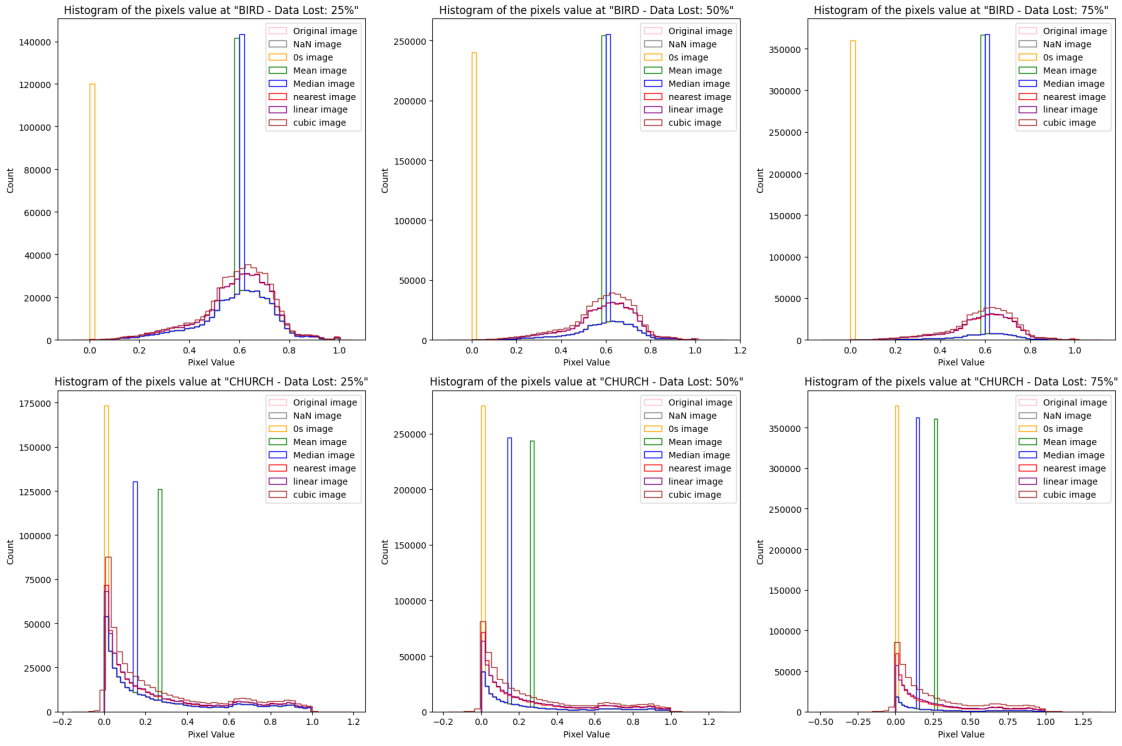
1.0.9 8. Calcula los histogramas de los valores pixelares para los datos originales, y para todas las interpolaciones.

[10]:
```
# Crea nuevas medias, evita que se sobreescriban las gráficas, y así se generen␣
↪nuevas.
grayscale_img1 = np.mean(image1, axis = 2)
grayscale_img2 = np.mean(image2, axis = 2)
```

```
# Convierte las imágenes a floats
grayscale_img1 = grayscale_img1.astype(float)
grayscale_img2 = grayscale_img2.astype(float)

# Define las probabilidades para la pérdida de datos
probabilities = [(0.25, 0.75), (0.5, 0.5), (0.75, 0.25)]

# Crea una lista de métodos para completar datos perdidos
fill_methods = ['Original', 'NaN', '0s', 'Mean', 'Median', 'nearest', 'linear',␣
 ↪'cubic']

# Crear una lista de colores para plotear con los métodos
colors = ['pink', 'gray', 'orange', 'green', 'blue', 'red', 'purple', 'brown']

# Crea la figura para que entren los subplots
fig, axs = plt.subplots(2, 3, figsize = (18, 12))

for i, (p_lost, p_win) in enumerate(probabilities):

    selection_matrix1 = np.random.choice([0, 1], size = grayscale_img1.shape, p␣
 ↪= [p_lost, p_win])
    selection_matrix2 = np.random.choice([0, 1], size = grayscale_img2.shape, p␣
 ↪= [p_lost, p_win])

    grayscale_img1_with_nan = np.where(selection_matrix1, grayscale_img1, np.
 ↪NaN)
    grayscale_img2_with_nan = np.where(selection_matrix2, grayscale_img2, np.
 ↪NaN)

    for j, method in enumerate(fill_methods):

        if method == 'Original':
            filled_img1 = grayscale_img1
            filled_img2 = grayscale_img2

        elif method == 'NaN':
            filled_img1 = grayscale_img1_with_nan
            filled_img2 = grayscale_img2_with_nan

        elif method == '0s':
            filled_img1 = np.nan_to_num(grayscale_img1_with_nan, nan = 0.0)
            filled_img2 = np.nan_to_num(grayscale_img2_with_nan, nan = 0.0)

        elif method == 'Mean':
            mean1 = np.nanmean(grayscale_img1_with_nan)
            mean2 = np.nanmean(grayscale_img2_with_nan)
            filled_img1 = np.nan_to_num(grayscale_img1_with_nan, nan = mean1)
```

```
            filled_img2 = np.nan_to_num(grayscale_img2_with_nan, nan = mean2)

        elif method == 'Median':
            median1 = np.nanmedian(grayscale_img1_with_nan)
            median2 = np.nanmedian(grayscale_img2_with_nan)
            filled_img1 = np.nan_to_num(grayscale_img1_with_nan, nan = median1)
            filled_img2 = np.nan_to_num(grayscale_img2_with_nan, nan = median2)

        else:  # PARA LOS MÉTODOS DE INTERPOLACIÓN ('más cercano', 'lineal' o␣
↪'cúbico')
            coords1 = np.array(np.nonzero(~np.isnan(grayscale_img1_with_nan))).T
            values1 = grayscale_img1_with_nan[~np.
↪isnan(grayscale_img1_with_nan)]
            grid_coords1 = np.array(np.nonzero(np.
↪isnan(grayscale_img1_with_nan))).T
            filled_pixels1 = griddata(coords1, values1, grid_coords1, method =␣
↪method)
            filled_img1 = grayscale_img1_with_nan.copy()
            filled_img1[np.isnan(grayscale_img1_with_nan)] = filled_pixels1

            coords2 = np.array(np.nonzero(~np.isnan(grayscale_img2_with_nan))).T
            values2 = grayscale_img2_with_nan[~np.
↪isnan(grayscale_img2_with_nan)]
            grid_coords2 = np.array(np.nonzero(np.
↪isnan(grayscale_img2_with_nan))).T
            filled_pixels2 = griddata(coords2, values2, grid_coords2, method =␣
↪method)
            filled_img2 = grayscale_img2_with_nan.copy()
            filled_img2[np.isnan(grayscale_img2_with_nan)] = filled_pixels2

        # Aplana las imágenes a matrices 1D poder plotear los histogramas
        flattened_img1 = filled_img1.flatten()
        flattened_img2 = filled_img2.flatten()

        # Plotea los histogramas
        axs[0, i].hist(flattened_img1, bins = 50, alpha = 1, label = '{} image'.
↪format(method), color = colors[j], histtype = 'step')
        axs[1, i].hist(flattened_img2, bins = 50, alpha = 1, label = '{} image'.
↪format(method), color = colors[j], histtype = 'step')

    axs[0, i].set_title('Histogram of the pixels value at "BIRD - Data Lost: {:.
↪0%}"'.format(p_lost))
    axs[0, i].set_xlabel('Pixel Value')
    axs[0, i].set_ylabel('Count')
    axs[0, i].legend()
```

```
    axs[1, i].set_title('Histogram of the pixels value at "CHURCH - Data Lost:␣
 ↪{:.0%}"'.format(p_lost))
    axs[1, i].set_xlabel('Pixel Value')
    axs[1, i].set_ylabel('Count')
    axs[1, i].legend()

plt.tight_layout()
plt.show()
```



1.0.10 9. Errores puntuales

Calcula la norma de Frobenius para los 3 métodos de llenado y los 3 de interpolación.

[11]:
```
# Abre el diccionario para almacenar las normas de Frobenius
frobenius_norms = {}

# Loop que pasa por cada método, para cada imagen, para cada porcentaje, es un␣
 ↪poco largo.

for i, (p_lost, p_win) in enumerate(probabilities):

    selection_matrix1 = np.random.choice([0, 1], size = grayscale_img1.shape, p␣
 ↪= [p_lost, p_win])
```

```
  selection_matrix2 = np.random.choice([0, 1], size = grayscale_img2.shape, p␣
↪= [p_lost, p_win])

  grayscale_img1_with_nan = np.where(selection_matrix1, grayscale_img1, np.
↪NaN)
  grayscale_img2_with_nan = np.where(selection_matrix2, grayscale_img2, np.
↪NaN)

  for j, method in enumerate(fill_methods):

      if method == 'Original':
          filled_img1 = grayscale_img1
          filled_img2 = grayscale_img2

      elif method == 'NaN':
          filled_img1 = grayscale_img1_with_nan
          filled_img2 = grayscale_img2_with_nan

      elif method == '0s':
          filled_img1 = np.nan_to_num(grayscale_img1_with_nan, nan = 0.0)
          filled_img2 = np.nan_to_num(grayscale_img2_with_nan, nan = 0.0)

      elif method == 'Mean':
          mean1 = np.nanmean(grayscale_img1_with_nan)
          mean2 = np.nanmean(grayscale_img2_with_nan)
          filled_img1 = np.nan_to_num(grayscale_img1_with_nan, nan = mean1)
          filled_img2 = np.nan_to_num(grayscale_img2_with_nan, nan = mean2)

      elif method == 'Median':
          median1 = np.nanmedian(grayscale_img1_with_nan)
          median2 = np.nanmedian(grayscale_img2_with_nan)
          filled_img1 = np.nan_to_num(grayscale_img1_with_nan, nan = median1)
          filled_img2 = np.nan_to_num(grayscale_img2_with_nan, nan = median2)

      else:   # PARA LOS MÉTODOS DE INTERPOLACIÓN ('más cercano', 'lineal' o␣
↪'cúbico')
          coords1 = np.array(np.nonzero(~np.isnan(grayscale_img1_with_nan))).T
          values1 = grayscale_img1_with_nan[~np.
↪isnan(grayscale_img1_with_nan)]
          grid_coords1 = np.array(np.nonzero(np.
↪isnan(grayscale_img1_with_nan))).T
          filled_pixels1 = griddata(coords1, values1, grid_coords1, method =␣
↪method)
          filled_img1 = grayscale_img1_with_nan.copy()
          filled_img1[np.isnan(grayscale_img1_with_nan)] = filled_pixels1
```

```
            coords2 = np.array(np.nonzero(~np.isnan(grayscale_img2_with_nan))).T
            values2 = grayscale_img2_with_nan[~np.
 ↪isnan(grayscale_img2_with_nan)]
            grid_coords2 = np.array(np.nonzero(np.
 ↪isnan(grayscale_img2_with_nan))).T
            filled_pixels2 = griddata(coords2, values2, grid_coords2, method =␣
 ↪method)
            filled_img2 = grayscale_img2_with_nan.copy()
            filled_img2[np.isnan(grayscale_img2_with_nan)] = filled_pixels2

        # Calcula la norma de Frobenius
        frobenius_norm1 = la.norm(grayscale_img1 - filled_img1, 'fro')
        frobenius_norm2 = la.norm(grayscale_img2 - filled_img2, 'fro')

        # Almacena las normas de Frobenius en el diccionario
        frobenius_norms[(p_lost, method)] = (frobenius_norm1, frobenius_norm2)

for (p_lost, method), (norm1, norm2) in frobenius_norms.items():
    print(f'Frobenius norm for {method} method, with {p_lost*100}% data loss.␣
 ↪BIRD = {norm1} and CHURCH = {norm2}')
```

```
Frobenius norm for Original method, with 25.0% data loss. BIRD = 0.0 and CHURCH
= 0.0
Frobenius norm for NaN method, with 25.0% data loss. BIRD = nan and CHURCH = nan
Frobenius norm for 0s method, with 25.0% data loss. BIRD = 209.7585585242732 and
CHURCH = 135.57018245208832
Frobenius norm for Mean method, with 25.0% data loss. BIRD = 51.18828789507367
and CHURCH = 98.527709611157
Frobenius norm for Median method, with 25.0% data loss. BIRD = 51.70052540601792
and CHURCH = 107.4933965549627
Frobenius norm for nearest method, with 25.0% data loss. BIRD =
13.613991846308865 and CHURCH = 38.704396127425035
Frobenius norm for linear method, with 25.0% data loss. BIRD = nan and CHURCH =
nan
Frobenius norm for cubic method, with 25.0% data loss. BIRD = nan and CHURCH =
nan
Frobenius norm for Original method, with 50.0% data loss. BIRD = 0.0 and CHURCH
= 0.0
Frobenius norm for NaN method, with 50.0% data loss. BIRD = nan and CHURCH = nan
Frobenius norm for 0s method, with 50.0% data loss. BIRD = 297.05855561378104
and CHURCH = 191.76631099267755
Frobenius norm for Mean method, with 50.0% data loss. BIRD = 72.13997833681242
and CHURCH = 139.44017240767684
Frobenius norm for Median method, with 50.0% data loss. BIRD = 72.81328774100193
and CHURCH = 151.82663679943866
Frobenius norm for nearest method, with 50.0% data loss. BIRD =
19.61497259609547 and CHURCH = 55.62933576560822
```

```
Frobenius norm for linear method, with 50.0% data loss. BIRD = nan and CHURCH =
nan
Frobenius norm for cubic method, with 50.0% data loss. BIRD = nan and CHURCH =
nan
Frobenius norm for Original method, with 75.0% data loss. BIRD = 0.0 and CHURCH
= 0.0
Frobenius norm for NaN method, with 75.0% data loss. BIRD = nan and CHURCH = nan
Frobenius norm for 0s method, with 75.0% data loss. BIRD = 363.8614123325962 and
CHURCH = 235.2293430253534
Frobenius norm for Mean method, with 75.0% data loss. BIRD = 88.25775036657241
and CHURCH = 170.91844474085266
Frobenius norm for Median method, with 75.0% data loss. BIRD = 89.07851476431847
and CHURCH = 186.53549868785566
Frobenius norm for nearest method, with 75.0% data loss. BIRD =
26.177746140018222 and CHURCH = 75.97655806367716
Frobenius norm for linear method, with 75.0% data loss. BIRD = nan and CHURCH =
nan
Frobenius norm for cubic method, with 75.0% data loss. BIRD = nan and CHURCH =
nan
```