Explicando PCA con Gatos

¿Qué es PCA y Cómo se Aplica?

El Análisis de Componentes Principales (PCA) es un algoritmo de reducción de dimensionalidad que se utiliza para simplificar conjuntos de datos complejos, manteniendo la mayor cantidad de información (varianza) posible.

Fuente de Datos y Carga de Imágenes

Las imágenes utilizadas en este ejemplo provienen del conjunto de datos **Cat and dog face** de Kaggle. Específicamente, se utilizó la porción correspondiente a las imágenes de gatos.

Enlace al dataset en Kaggle

Carga de las Imágenes

Para realizar este análisis, primero se deben cargar las imágenes.

```
import os
import scipy
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        try:
            with Image.open(img_path) as img:
                # Convertir a escala de grises ('L') y redimensionar a 64x64
                img_gray = img.convert('L').resize((64, 64))
                images.append(np.array(img_gray))
        except IOError:
            pass
    return images

# Ruta a la carpeta que contiene las imágenes de gatos
data_folder = './data/'
imgs = load_images_from_folder(data_folder)
```

1. Aplanar los Datos

Cada imagen de 64x64 píxeles (una matriz) se convierte en un vector de 4096 píxeles (una fila). Esto transforma el conjunto de datos en una gran matriz de datos Y donde cada fila es una imagen y cada columna es una variable (un píxel).

```
# 'imgs' es una lista de matrices de imagen (64x64)
imgs_flatten = np.array([im.reshape(-1) for im in imgs])
```

2. Centrar los Datos

Se calcula la media de cada columna (cada píxel) y se resta de todos los valores de esa columna. Esto centra los datos con respecto al origen, lo cual es un requisito para calcular la covarianza correctamente.

Cálculo General:
Para cada columna j, la media μ_j se calcula como:

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} y_{i,j}$$

La matriz centrada X se obtiene restando la media de cada columna a la matriz original Y:

$$X_{i,j} = Y_{i,j} - \mu_j$$

```
def center_data(Y):
    # Calcular el vector de medias para cada columna (axis=0)
    mean_vector = np.mean(Y, axis=0)
    # Restar el vector de medias de la matriz de datos original
    X = Y - mean_vector
    return X, mean_vector

# Aplicar la función
X, mean_vector = center_data(imgs_flatten)
```

3. Calcular la Matriz de Covarianza

A partir de los datos centrados X, se calcula la matriz de covarianza (Σ). Esta matriz describe la relación y varianza entre los diferentes píxeles del conjunto de datos.

Cálculo General:

$$\Sigma = \frac{1}{m-1} X^T X$$

Donde X^T es la transpuesta de la matriz centrada y m es el número de observaciones (imágenes).

```
def get_cov_matrix(X):
    # Número de observaciones
    m = X.shape[0]
    # Calcular la matriz de covarianza
    cov_matrix = np.dot(X.T, X) / (m - 1)
    return cov_matrix

# Aplicar la función
cov_matrix = get_cov_matrix(X)
```

4. Calcular Autovalores y Autovectores

Se calculan los autovalores (λ) y autovectores (v) de la matriz de covarianza. Los autovectores, ordenados por su autovalor correspondiente de mayor a menor, son los **componentes principales**.

Cálculo General:
Resolver la ecuación característica para la matriz de covarianza Σ:

$$\Sigma v = \lambda v$$

```
# 'k' es el número de los mayores autovalores/vectores a encontrar
eigenvals, eigenvecs = scipy.sparse.linalg.eigsh(cov_matrix, k=55)
# Opción más robusta pero menos performante:
# eigenvals, eigenvecs = np.linalg.eigh(cov_matrix)

# Ordenarlos de mayor a menor
eigenvals = eigenvals[::-1]
eigenvecs = eigenvecs[:,::-1]
```

5. Reducir la Dimensionalidad (Proyección)

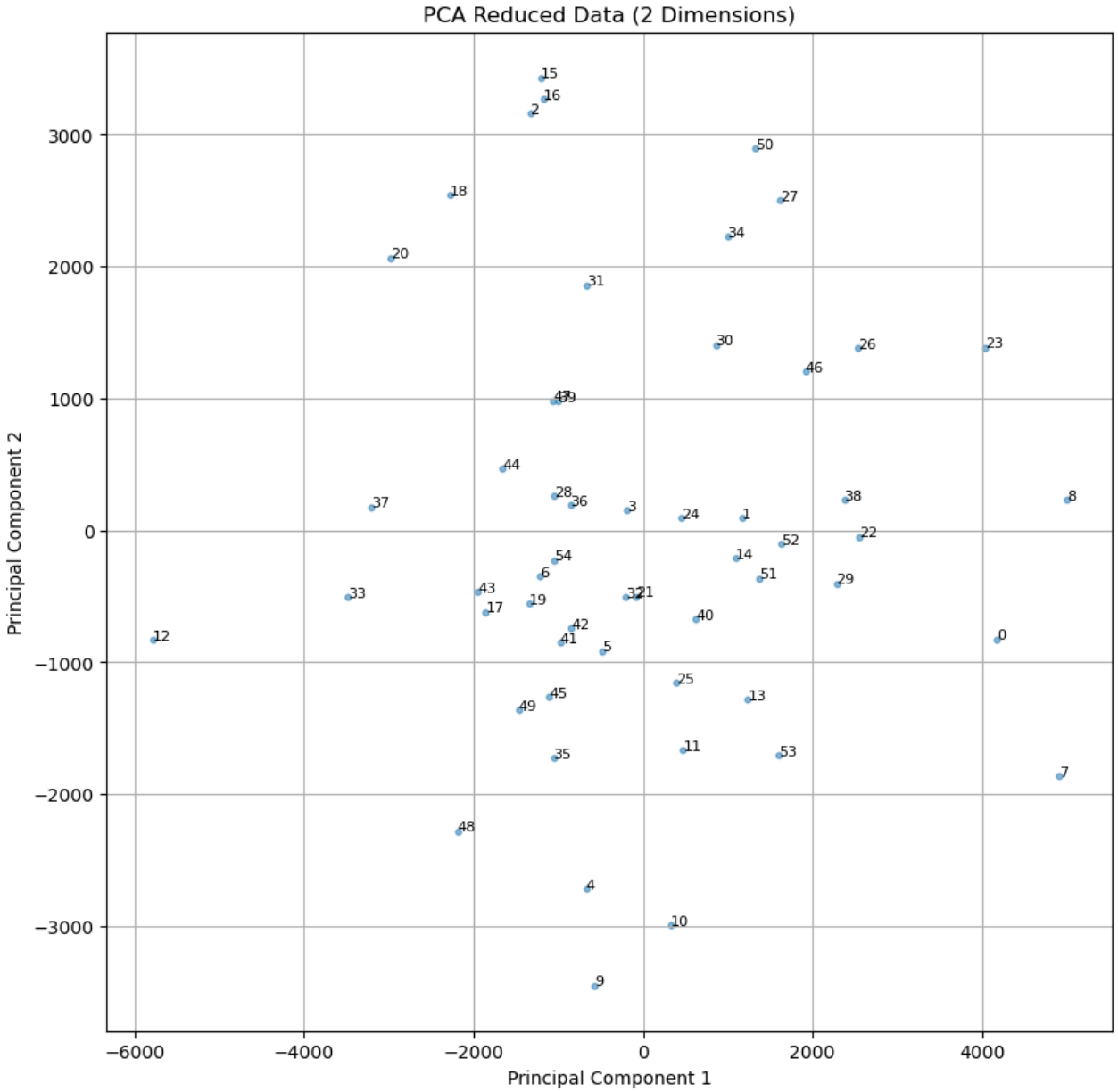
Para reducir la dimensionalidad de 4096 variables a un número menor *k* (por ejemplo, a 2 dimensiones para visualización), se proyectan los datos centrados X sobre los primeros *k* componentes principales (autovectores).

Cálculo General:

$$X_{red} = X \cdot V_k$$

Donde V_k es la matriz formada por los primeros *k* autovectores.

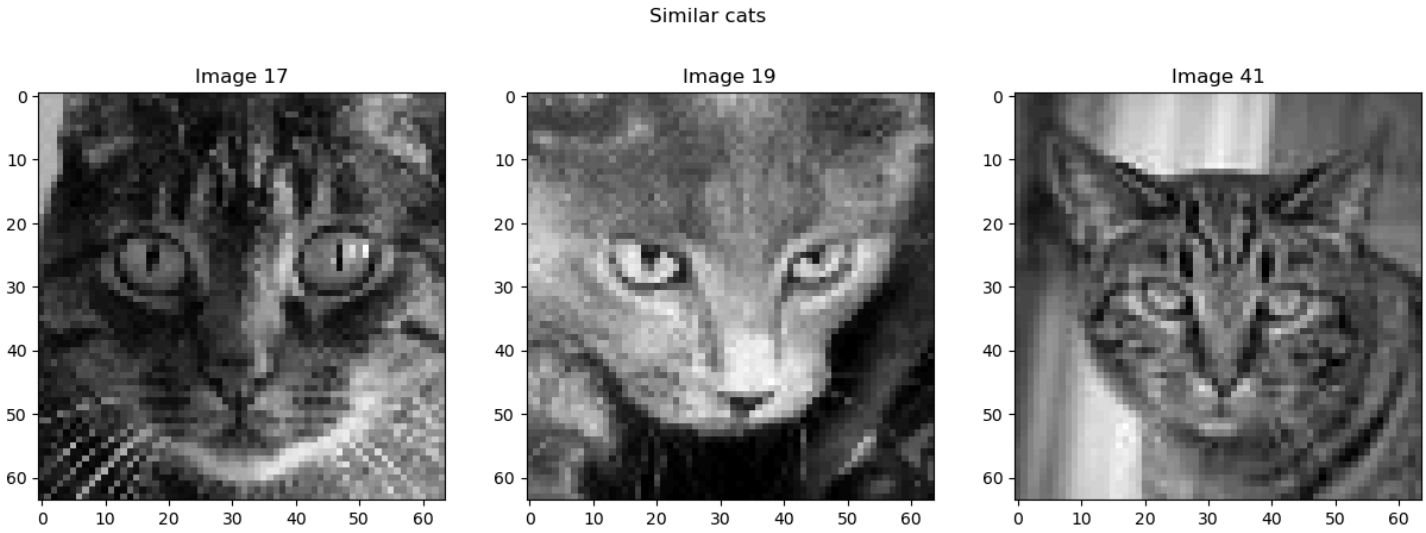
```
def perform_PCA(X, eigenvecs, k):
    # Seleccionar los primeros k autovectores
    V_k = eigenvecs[:, :k]
    # Proyectar los datos
    X_red = np.dot(X, V_k)
    return X_red

# Reducir a 2 dimensiones
X_red_2 = perform_PCA(X, eigenvecs, 2)

# Graficando los datos reducidos con el número de imagen en cada punto
plt.figure(figsize=(10, 10))
plt.scatter(X_red_2[:, 0], X_red_2[:, 1], s=10, alpha=0.5)
for i in range(len(X_red_2)):
    plt.annotate(str(i), (X_red_2[i, 0], X_red_2[i, 1]), fontsize=8)
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Datos Reducidos con PCA (2 Dimensiones)')
plt.grid()
plt.show()
```



En este gráfico, cada punto representa una imagen de gato, reducida a solo 2 dimensiones. Los puntos cercanos corresponden a imágenes de gatos con características similares.

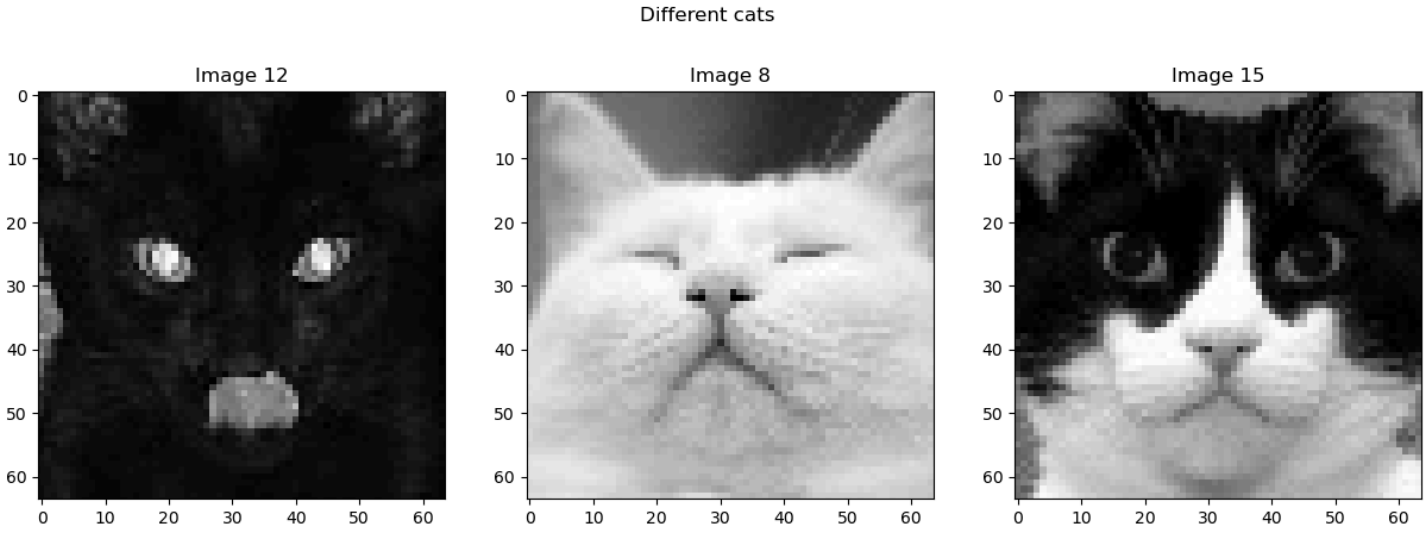
Para verificar esto, podemos visualizar las imágenes 19, 21 y 41, que aparecen muy juntas en el gráfico. Como se puede ver, los tres gatos tienen rasgos muy similares, como el hocico blanco y el pelaje oscuro alrededor de los ojos.

```
fig, ax = plt.subplots(1,3, figsize=(15,5))
ax[0].imshow(imgs[17], cmap='gray')
ax[0].set_title('Imagen 17')
ax[1].imshow(imgs[19], cmap='gray')
ax[1].set_title('Imagen 19')
ax[2].imshow(imgs[41], cmap='gray')
ax[2].set_title('Imagen 41')
plt.suptitle('Gatos similares')
```



De manera similar, si tomamos puntos que están muy alejados en el gráfico, como las imágenes 18, 41 y 51, vemos que los gatos son visualmente muy diferentes entre sí.

```
fig, ax = plt.subplots(1,3, figsize=(15,5))
ax[0].imshow(imgs[12], cmap='gray')
ax[0].set_title('Imagen 12')
ax[1].imshow(imgs[8], cmap='gray')
ax[1].set_title('Imagen 8')
ax[2].imshow(imgs[15], cmap='gray')
ax[2].set_title('Imagen 15')
plt.suptitle('Gatos diferentes')
```



Reconstrucción de la Imagen a partir de los Componentes

Una de las aplicaciones más visuales de PCA es ver cómo los componentes principales capturan las características esenciales de las imágenes.

Proceso de Reconstrucción de la Imagen

Podemos revertir la proyección de PCA para reconstruir la imagen original a partir de los datos reducidos. Este proceso nos permite visualizar cuánta información se ha conservado después de la compresión. Un paso crucial es sumar de nuevo el vector de medias que se restó inicialmente, para devolver la imagen a su espacio de color/brillo original.

Cálculo General:

$$X_{rec\_centrado} = X_{red} \cdot V_k^T$$

$$Y_{rec} = X_{rec\_centrado} + media$$

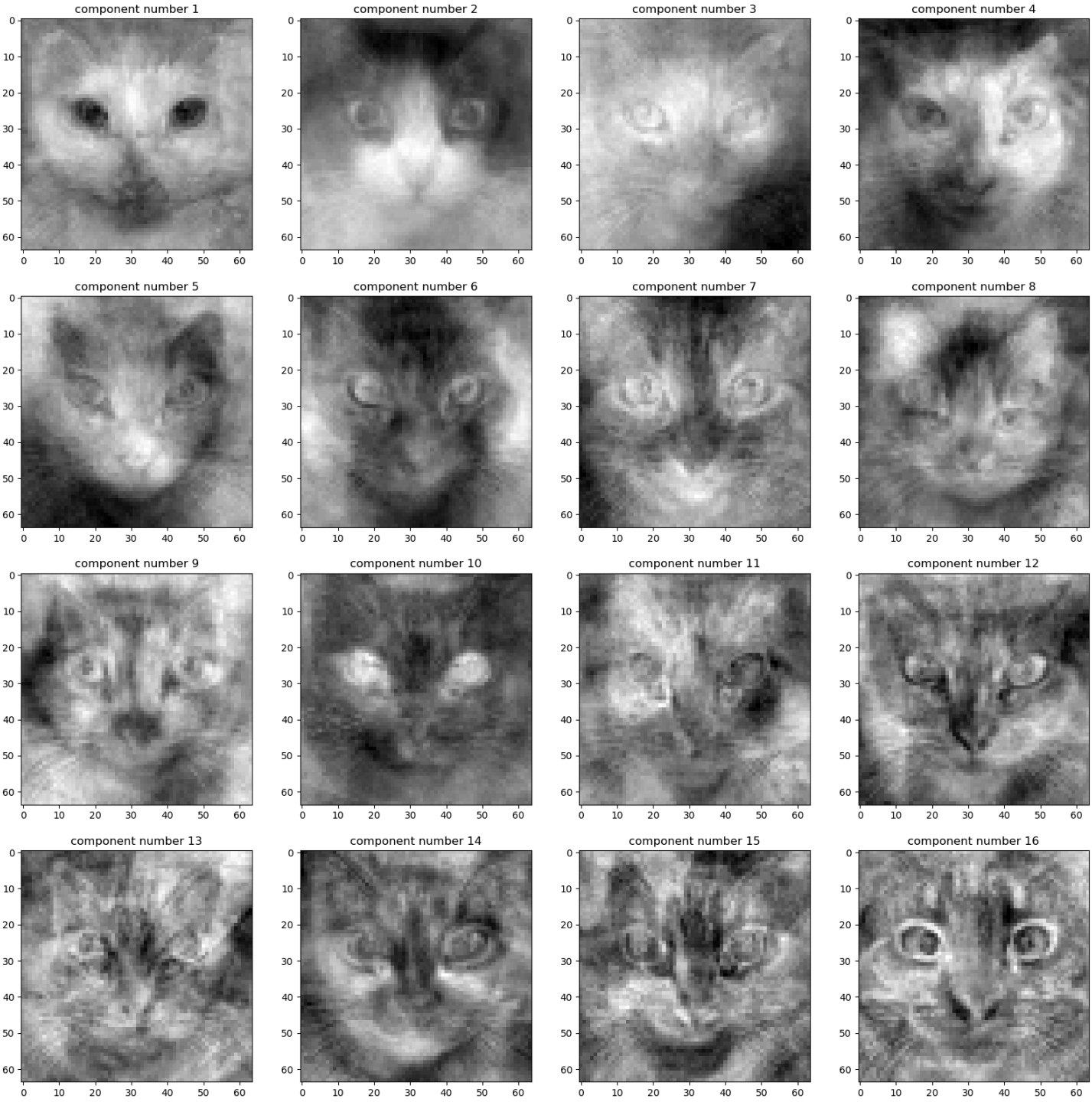
Donde X_{red} son los datos en el espacio reducido y V_k^T es la transpuesta de la matriz con los *k* autovectores utilizados.

```python
def reconstruct_image(Xred, eigenvecs, mean_vector):
    # El número de componentes k está implícito en el shape de Xred
    k = Xred.shape[1]
    # Reconstruir la imagen proyectando de vuelta al espacio original
    X_reconstructed = Xred.dot(eigenvecs[:,:k].T)
    # Sumar la media para devolverla al espacio original
    X_reconstructed = X_reconstructed + mean_vector
    return X_reconstructed
```

Visualización de los Componentes Individuales

Cada componente principal puede ser redimensionado a 64x64 para visualizarse como una imagen. Estos "eigen-gatos" representan patrones fundamentales encontrados en el conjunto de datos.

```
height, width = imgs[0].shape
fig, ax = plt.subplots(4,4, figsize=(20,20))
for n in range(4):
    for k in range(4):
        ax[n,k].imshow(eigenvecs[:,n*4+k].reshape(height,width), cmap='gray')
        ax[n,k].set_title(f'componente número {n*4+k+1}')
```

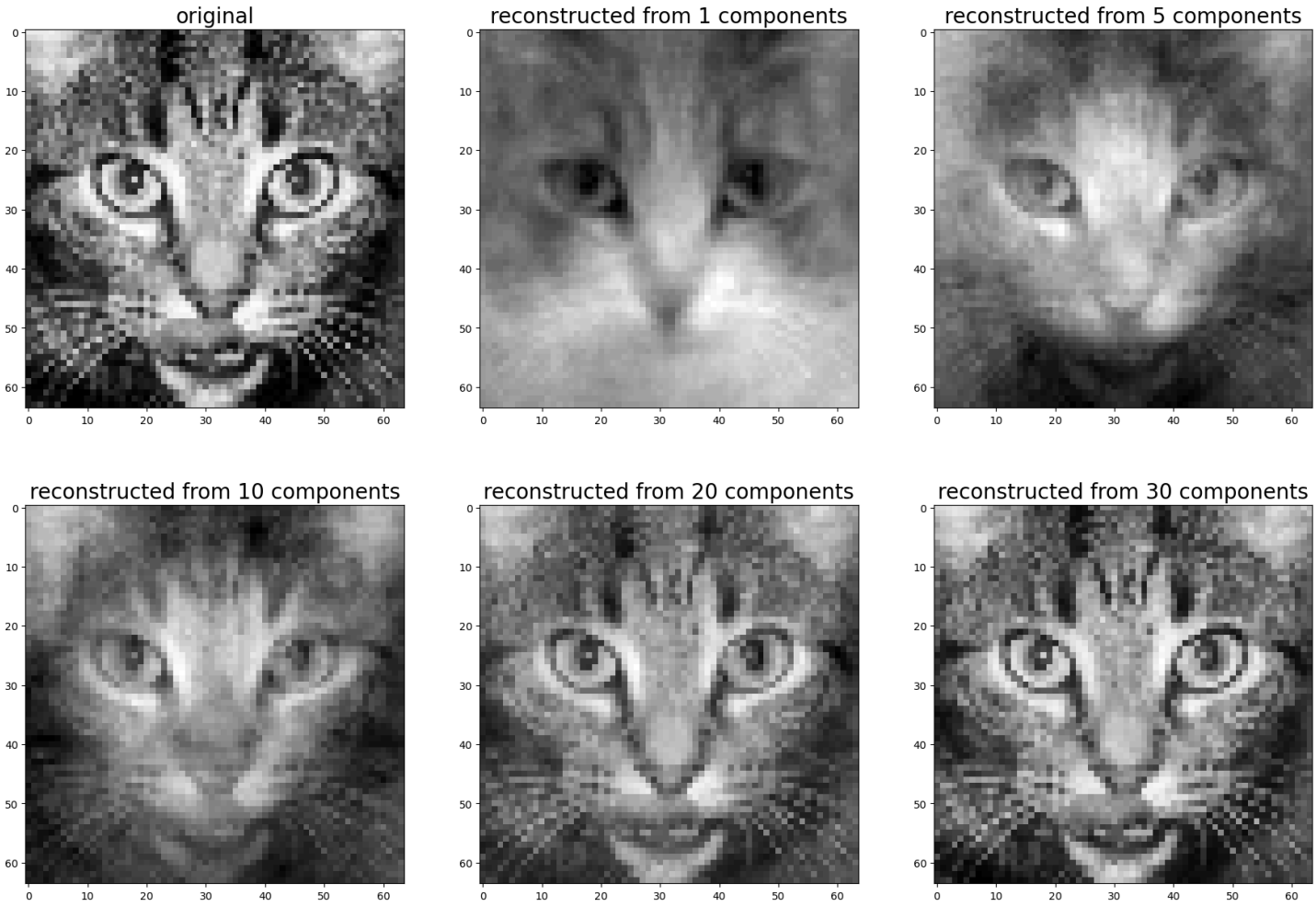


Visualización de los primeros 16 componentes principales. Los primeros componentes capturan la forma general de la cara de un gato, mientras que los posteriores capturan detalles más finos.

Reconstrucción con un Número Creciente de Componentes

A medida que usamos más componentes principales para la reconstrucción, la imagen recupera más detalles y se parece más a la original, demostrando el equilibrio entre compresión y fidelidad.

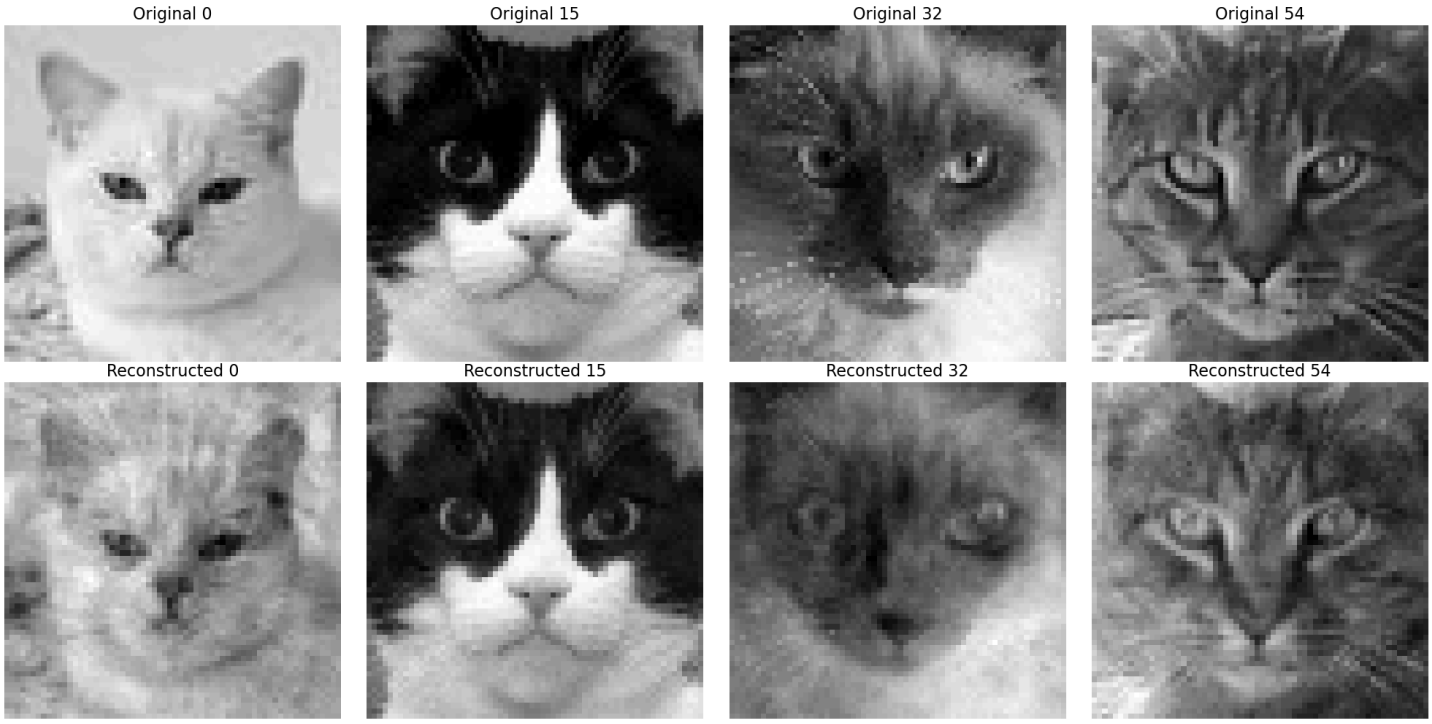
```
Xred1 = perform_PCA(X, eigenvecs,1) # reducir dimensiones a 1 componente
Xred5 = perform_PCA(X, eigenvecs, 5) # reducir dimensiones a 5 componentes
Xred10 = perform_PCA(X, eigenvecs, 10) # reducir dimensiones a 10 componentes
Xred20 = perform_PCA(X, eigenvecs, 20) # reducir dimensiones a 20 componentes
Xred30 = perform_PCA(X, eigenvecs, 30) # reducir dimensiones a 30 componentes
Xrec1 = reconstruct_image(Xred1, eigenvecs, mean_vector) # reconstruir imagen con 1 componente
Xrec5 = reconstruct_image(Xred5, eigenvecs, mean_vector) # reconstruir imagen con 5 componentes
Xrec10 = reconstruct_image(Xred10, eigenvecs, mean_vector) # reconstruir imagen con 10 componentes
Xrec20 = reconstruct_image(Xred20, eigenvecs, mean_vector) # reconstruir imagen con 20 componentes
Xrec30 = reconstruct_image(Xred30, eigenvecs, mean_vector) # reconstruir imagen con 30 componentes

fig, ax = plt.subplots(2,3, figsize=(22,15))
ax[0,0].imshow(imgs[21], cmap='gray')
ax[0,0].set_title('original', size=20)
ax[0,1].imshow(Xrec1[21].reshape(height,width), cmap='gray')
ax[0,1].set_title('reconstruido con 1 componente', size=20)
ax[0,2].imshow(Xrec5[21].reshape(height,width), cmap='gray')
ax[0,2].set_title('reconstruido con 5 componentes', size=20)
ax[1,0].imshow(Xrec10[21].reshape(height,width), cmap='gray')
ax[1,0].set_title('reconstruido con 10 componentes', size=20)
ax[1,1].imshow(Xrec20[21].reshape(height,width), cmap='gray')
ax[1,1].set_title('reconstruido con 20 componentes', size=20)
ax[1,2].imshow(Xrec30[21].reshape(height,width), cmap='gray')
ax[1,2].set_title('reconstruido con 30 componentes', size=20)
```



Comparación de una imagen original con sus reconstrucciones utilizando un número creciente de componentes. Con solo 30-35 componentes, la imagen reconstruida es casi idéntica a la original.

A continuación, se muestran varios ejemplos de imágenes originales junto a su versión reconstruida utilizando 35 componentes principales, demostrando la alta fidelidad que se puede lograr con una fracción de la información original.

```
Xred35 = perform_PCA(X, eigenvecs, 35) # reducir dimensiones a 35 componentes
Xrec35 = reconstruct_image(Xred35, eigenvecs, mean_vector) # reconstruir imagen con 35 componentes

fig, ax = plt.subplots(2, 4, figsize=(20, 10))

indices = [0, 15, 32, 54]
for i, idx in enumerate(indices):
    # Fila superior: originales
    ax[0, i].imshow(imgs[idx], cmap='gray')
    ax[0, i].set_title(f'Original {idx}', size=16)
    ax[0, i].axis('off')
    # Fila inferior: reconstruidas
    ax[1, i].imshow(Xrec35[idx].reshape(height, width), cmap='gray')
    ax[1, i].set_title(f'Reconstruido {idx}', size=16)
    ax[1, i].axis('off')

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



Referencias:

- **Curso (Intuitivo y Práctico)**: *Linear Algebra for Machine Learning and Data Science* (deeplearning.ai)
- **Curso (Matemático y Formal)**: *Mathematics for Machine Learning: PCA* (Imperial College London)
- **Libro (Avanzado):** *Mathematics for Machine Learning*, M. P. Deisenroth, A. A. Faisal, y C. S. Ong., Capítulo 10, "Dimensionality Reduction with Principal Component Analysis". (Libro Gratuito)