Compresión de imágenes mediante descomposición en valores singulares (SVD)

1.0 El problema del tamaño de la imagen

En esencia, una imagen digital es una matriz (una cuadrícula) de números. El tamaño de esta matriz se correlaciona directamente con el espacio de almacenamiento requerido. Sin compresión, las imágenes pueden consumir rápidamente grandes cantidades de almacenamiento y ancho de banda.

1.1 Imágenes en escala de grises

Una imagen en escala de grises es la forma más simple. Es una cuadrícula 2D donde cada celda (píxel) representa la intensidad de la luz.

Estructura de datos: Una única matriz de tamaño M*N.

Valor de píxel: normalmente un entero de 8 bits (0–255), donde 0 es negro y 255 es blanco.

Ejemplo: transmisión de cámara de seguridad en blanco y negro

- Imagine una transmisión de cámara de seguridad en blanco y negro **Full HD (1920 x 1080) estándar.**
- **Cálculo:** 1920*1080 píxeles * 1 byte/píxel = aproximadamente 2,07 MB por imagen.
- Parece poco, pero si esa cámara graba vídeo a 30 fotogramas por segundo, son **62 MB por segundo** o **223 GB por hora**. Sin compresión, almacenar un solo día de grabación requeriría casi **5,4 terabytes**.

1.2. Imágenes en color

Las imágenes en color son más complejas porque necesitan representar rojo, verde y azul (RGB). Esto triplica los datos.

Estructura de datos: Tres matrices (canales) de tamaño M*N apiladas. Una matriz para el rojo, otra para el verde y otra para el azul.

Valor de píxel: tres números enteros de 8 bits por píxel.

Ejemplo: fotografía de alta resolución tomada con un teléfono inteligente moderno

- Consideremos una fotografía de alta resolución tomada con un teléfono inteligente moderno, digamos de **12 megapíxeles (4000 x 3000)**.
- **Cálculo:** 4000*3000 píxeles * 3 bytes/píxel = 36 000 000 bytes = 34,3 MB.
- Si tomas 100 fotos durante tus vacaciones, eso equivale a **3,4 GB** de datos sin procesar. Si las subes a un servicio en la nube o a una red social, consumes una cantidad considerable de ancho de banda y almacenamiento del servidor.

1.3 Escenario de la vida real: ¿Por qué necesitamos la compresión?

Veamos un ejemplo práctico: **sitios web de comercio electrónico (como Amazon o eBay)**.

Contexto: Estos sitios muestran millones de imágenes de productos. Una sola página de producto puede mostrar entre 5 y 10 imágenes (miniaturas, vista principal y vista ampliada).

El desafío:

- **Experiencia del usuario:** Si cada imagen de producto fuera un archivo sin procesar de 34 MB, un usuario con una conexión móvil 4G esperaría casi 30 segundos solo para ver la foto de un zapato. Probablemente abandonaría el sitio (alta tasa de rebote).
- **Costo:** La empresa paga el ancho de banda. Servir petabytes de imágenes sin comprimir a millones de usuarios diariamente costaría una fortuna en tarifas de transferencia de datos.

El requisito:

- Necesitamos imágenes que se vean “suficientemente bien” al ojo humano pero que tengan un tamaño de archivo significativamente más pequeño.
- En lugar de 34 MB, necesitamos que la foto del producto tenga entre **100 KB y 200 KB** (una relación de compresión de casi 300:1).

Por eso, las técnicas de compresión de imágenes como SVD (Descomposición en Valores Singulares), JPEG y PNG son tecnologías fundamentales para internet. Nos

permiten descartar datos visuales "menos importantes" para ahorrar muchísimo espacio y tiempo.

2. Descomposición en valores singulares (SVD)

La descomposición en valores singulares (SVD) es una poderosa técnica matemática que nos permite dividir cualquier matriz en tres matrices distintas y más simples.

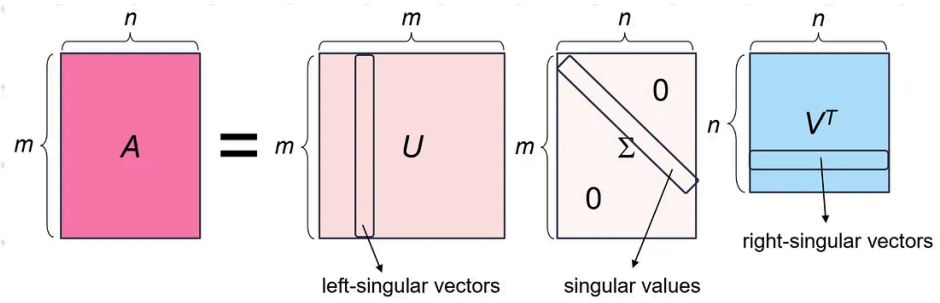
2.1 El concepto central

Imagina que tienes una receta compleja. La SVD consiste en identificar los ingredientes crudos (la esencia) que componen ese plato. Luego, puedes recrearlo usando solo los ingredientes más importantes, simplificando la receta y conservando el sabor principal.

2.2 La fórmula matemática

Cualquier matriz A (que representa nuestra imagen) se puede descomponer en:

$$A = U \times \Sigma \times V^T$$



Dónde:

- **A** : La matriz original (por ejemplo, nuestra imagen).
- **U** : Matriz de "Vectores Singulares Izquierdos". Considérelo como los "patrones verticales" o columnas de la imagen.
- **Σ** : Matriz de "Valores Singulares". Esta es una matriz diagonal (ceros en todas partes excepto en la diagonal). Estos valores representan la **fuerza** o **energía** de cada patrón. Siempre se ordenan de mayor a menor.
- **V^T** : La transpuesta de la matriz de "Vectores Singulares Rectos". Considérelo como los "patrones horizontales" o filas.

2.3 Un ejemplo sencillo

Tomemos una “imagen” muy pequeña representada por una matriz 2*2:

$$A = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$$

Cuando aplicamos SVD a esta matriz A, obtenemos tres matrices:

1. **Matriz U (Direcciones verticales):**

$$U = \begin{bmatrix} -0.707 & -0.707 \\ -0.707 & 0.707 \end{bmatrix}$$

2. Matriz Σ (Fuerzas/Energía):

$$\Sigma = \begin{bmatrix} \mathbf{5} & 0 \\ 0 & \mathbf{1} \end{bmatrix}$$

Observe los valores 5 y 1. El primer valor (5) es mucho mayor que el segundo (1). Esto indica que la primera capa de información es cinco veces más importante que la segunda.

3. Matriz V^T (Direcciones horizontales):

$$V^T = \begin{bmatrix} -0.707 & -0.707 \\ -0.707 & 0.707 \end{bmatrix}$$

3. Compresión en acción: "Aproximación de rangos"

Ahora, la magia de la compresión ocurre. Como los valores singulares en Σ están ordenados por importancia, podemos optar por **conservar solo los superiores** y descartar los pequeños.

En nuestro ejemplo, conservaremos solo el **primer** valor singular (el 5) y descartaremos el segundo (el 1). Esto se denomina **aproximación de rango 1**.

3.1 Reconstrucción original (calidad perfecta)

$$U \times \Sigma \times V^T = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$$

Esta es una combinación perfecta.

3.2 Reconstrucción comprimida (rango 1)

Efectivamente ponemos a cero el segundo valor:

$$\Sigma_{\text{new}} = \begin{bmatrix} \mathbf{5} & 0 \\ 0 & \mathbf{0} \end{bmatrix}$$

Ahora, si volvemos a multiplicar las matrices:

$$A_{\text{compressed}} = \begin{bmatrix} -0.707 \\ -0.707 \end{bmatrix} \times [5] \times \begin{bmatrix} -0.707 & -0.707 \end{bmatrix}$$

$$A_{\text{compressed}} = \begin{bmatrix} 2.5 & 2.5 \\ 2.5 & 2.5 \end{bmatrix}$$

El resultado:

$$\textbf{Original: } \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$$

$$\textbf{Compressed: } \begin{bmatrix} 2.5 & 2.5 \\ 2.5 & 2.5 \end{bmatrix}$$

¿Perdimos información?

Sí. Los números cambiaron ligeramente. ¿ **Ahorramos espacio?** En este pequeño ejemplo de 2 x 2, no. Pero en una imagen real con dimensiones como 1000 x 1000, conservar solo los 50 valores singulares superiores (en lugar de 1000) nos permite

descartar cantidades masivas de datos, mientras que la imagen reconstruida (la matriz de números) se mantiene visualmente muy similar a la original. La "energía" que conservamos (el 5) capturó la mayor parte de la estructura de la imagen.

4. Código Python para SVD

A continuación se muestra un código de muestra para calcular SVD para una imagen en escala de grises.

```
import numpy as np

# 1. Create a 10x10 matrix (our "image")
# A simple gradient pattern where value = row_index + col_index
A = np.zeros((10, 10))
for i in range(10):
    for j in range(10):
        A[i, j] = i + j

print("--- 1. Original Matrix A (10x10) ---")
print(A)
print("\nTotal numbers to store: 100")

# 2. Perform SVD
U, S, Vt = np.linalg.svd(A)

print("\n--- 2. SVD Components ---")
print("Singular Values (Sigma):")
print(np.round(S, 2))
# Note: SVD returns Sigma as a list of values, not a diagonal matrix,
# for efficiency.

# 3. Rank-1 Approximation (Compression)
# We keep only the first component (k=1)
k = 1

# Extract the top-k components
U_k = U[:, :k]        # First k columns of U (10 x 1)
S_k = np.diag(S[:k])  # First k singular values (1 x 1 diagonal matrix)
Vt_k = Vt[:k, :]      # First k rows of Vt (1 x 10)

# 4. Reconstruct the Compressed Matrix
# Formula: A_approx = U_k * S_k * Vt_k
A_compressed = np.dot(U_k, np.dot(S_k, Vt_k))

print(f"\n--- 3. Compressed Matrix (Rank-{k}) ---")
print(np.round(A_compressed, 1))

# 5. Analyze Savings
original_size = 10 * 10
# Compressed size: k columns of U + k singular values + k rows of Vt
```

```
compressed_size = (10 * k) + k + (k * 10)

print(f"\n--- 4. Storage Comparison (Rank-{k}) ---")
print(f"Original Size:   {original_size} numbers")
print(f"Compressed Size: {compressed_size}
numbers (10 from U + 1 from Sigma + 10 from Vt)")
print(f"Compression Ratio: {original_size / compressed_size:.2f}x")
print(f"Space Saved: {100 - (compressed_size/original_size)*100:.1f}%")

# Let's compare a single pixel to check accuracy
# Original A[0,0] was 0. Compressed is typically close but not exact.
print(f"\nCheck Pixel [0,0]: Original = {A[0,0]}, Compressed =
{A_compressed[0,0]:.2f}")
print(f"Check Pixel [9,9]: Original = {A[9,9]}, Compressed =
{A_compressed[9,9]:.2f}")
```

A continuación se muestra el resultado.

```
--- 1. Original Matrix A (10x10) ---
[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
 [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
 [ 2.  3.  4.  5.  6.  7.  8.  9. 10. 11.]
 [ 3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
 [ 4.  5.  6.  7.  8.  9. 10. 11. 12. 13.]
 [ 5.  6.  7.  8.  9. 10. 11. 12. 13. 14.]
 [ 6.  7.  8.  9. 10. 11. 12. 13. 14. 15.]
 [ 7.  8.  9. 10. 11. 12. 13. 14. 15. 16.]
 [ 8.  9. 10. 11. 12. 13. 14. 15. 16. 17.]
 [ 9. 10. 11. 12. 13. 14. 15. 16. 17. 18.]]

Total numbers to store: 100

--- 2. SVD Components ---
Singular Values (Sigma):
[98.39  8.39  0.    0.    0.    0.    0.    0.    0.    0.  ]

--- 3. Compressed Matrix (Rank-1) ---
[[ 2.7  3.2  3.7  4.2  4.7  5.2  5.7  6.2  6.7  7.2]
 [ 3.2  3.8  4.4  5.   5.5  6.1  6.7  7.3  7.9  8.5]
 [ 3.7  4.4  5.   5.7  6.4  7.1  7.8  8.5  9.2  9.9]
 [ 4.2  5.   5.7  6.5  7.3  8.1  8.9  9.6 10.4 11.2]
 [ 4.7  5.5  6.4  7.3  8.2  9.   9.9 10.8 11.7 12.5]
 [ 5.2  6.1  7.1  8.1  9.  10.  11.  11.9 12.9 13.9]
 [ 5.7  6.7  7.8  8.9  9.9 11.  12.  13.1 14.2 15.2]
 [ 6.2  7.3  8.5  9.6 10.8 11.9 13.1 14.3 15.4 16.6]
 [ 6.7  7.9  9.2 10.4 11.7 12.9 14.2 15.4 16.7 17.9]
 [ 7.2  8.5  9.9 11.2 12.5 13.9 15.2 16.6 17.9 19.3]]

--- 4. Storage Comparison (Rank-1) ---
Original Size:   100 numbers
Compressed Size: 21 numbers (10 from U + 1 from Sigma + 10 from Vt)
```

```
Compression Ratio: 4.76x
Space Saved: 79.0%

Check Pixel [0,0]: Original = 0.0, Compressed = 2.67
Check Pixel [9,9]: Original = 18.0, Compressed = 19.26
=== Code execution complete ===
```

5. ¿Cómo ahorramos espacio?

La distinción clave es entre **"Representación de almacenamiento"** (cómo guardamos el archivo en el disco) y **"Representación de visualización"** (cómo mostramos la imagen en la pantalla).

5.1. Qué se almacena en el disco (comprimido)

Al guardar esta imagen comprimida en un archivo, **no se guardan los 100 números** que se ven en el resultado. Solo se guardan los ingredientes necesarios para construirlos.

Guardamos exactamente estos **21 números** :

- **Columna U (10 números):** `[-0.21, -0.23, -0.26, ...]`
- **Valor Sigma (Σ) (1 número):** `[168.4]`
- **Fila V^T (10 números):** `[-0.17, -0.20, -0.23, ...]`

Total almacenado: 21 números de punto flotante.

Tamaño de almacenamiento: si cada flotante tiene 4 bytes, el tamaño del archivo es 21*4 = 84 bytes.

5.2. Lo que se muestra en la pantalla (reconstruido)

Cuando desea *ver* la imagen (por ejemplo, abrirla en un visor de fotografías), la computadora lee esos 21 números y realiza la multiplicación:

$$A_{\text{view}} = U_{\text{col}} \times \Sigma_{\text{val}} \times V^T_{\text{row}}$$

El resultado de esa multiplicación es la **matriz de 100 números** que ves en la salida.

La computadora debe generar estos 100 números en la memoria RAM para iluminar los 100 píxeles de la pantalla. Pero el *archivo en el disco duro* sigue siendo diminuto

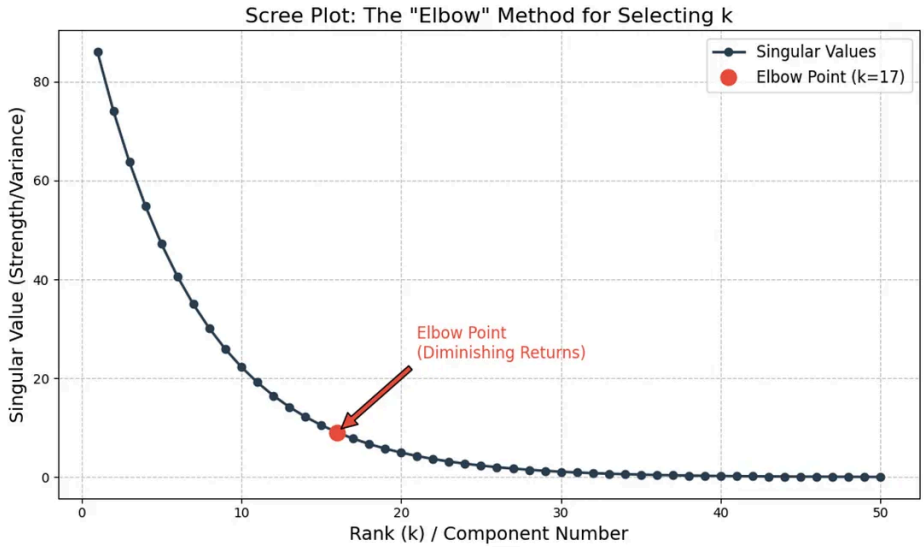
(21 números).

6. Elección del rango óptimo (k)

¿Cómo decidimos exactamente cuál k es el "mejor"? Rara vez hay una única respuesta correcta, pero los científicos de datos suelen usar uno de dos métodos para encontrar el equilibrio óptimo:

6.1 El método del “codo” (gráfico de pedregal)

Si se trazan los valores singulares (Σ) en un gráfico de mayor a menor, normalmente se observará una caída pronunciada seguida de una cola larga y plana. El k óptimo suele encontrarse en el "codo" de esta curva, el punto donde los rendimientos disminuyen y la adición de más valores singulares proporciona una ganancia de información insignificante.



6.2 Retención de energía (Energía acumulada)

Un enfoque más matemático consiste en definir un umbral de "energía" que se desea preservar (p. ej., 90 % o 95 %). La energía se calcula como la suma de los cuadrados de los valores singulares. Se suman los cuadrados de los primeros k valores y se divide entre la suma total de todos los valores singulares elevados al cuadrado.

$$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{N} \sigma_i^2} \geq 0.95$$

Simplemente se elige el valor k más pequeño que satisfaga esta desigualdad. Para imágenes con ruido, se podría optar por una retención menor (para filtrar el ruido); para imágenes médicas, se podría exigir una retención del 99 %.

7. Reflexiones finales sobre la SVD para la compresión de imágenes

Analizamos la mecánica de la Descomposición en Valores Singulares (SVD) utilizando una matriz simple de 10*10. Los resultados fueron claros: al conservar solo los valores singulares más significativos (aproximación de rango 1), reconstruimos una imagen casi perfecta utilizando solo aproximadamente el 20 % del almacenamiento de datos original.

Este sencillo ejemplo resalta varias conclusiones clave sobre la compresión de imágenes y el álgebra lineal:

7.1. Los datos no siempre están “llenos” de información

Aunque una imagen tenga 100 píxeles (o 12 millones), no cada píxel contiene información única y crucial. Patrones como degradados, colores sólidos o texturas repetidas son matemáticamente simples. La SVD aprovecha esta simplicidad y nos permite descartar la redundancia.

7.2. La compresión es un equilibrio

En nuestro ejemplo, la compresión presentaba pérdidas. Los valores de píxeles reconstruidos (p. ej., 17,98 en lugar de 18,0) no eran idénticos a los originales.

Alto rango (k **):** Mejor calidad, menor compresión.

Rango bajo (k **):** Peor calidad (más borroso), mayor compresión. Encontrar el punto óptimo es el arte de la compresión.

7.3. ¿Por qué SVD no es el estándar? (JPEG vs. SVD)

En la práctica, formatos como **JPEG** dominan la compresión de imágenes. Utilizan una técnica llamada **Transformada Discreta del Coseno (DCT)**, que es matemáticamente similar a la SVD (ambas son transformaciones de base), pero presenta ventajas cruciales:

1. **Velocidad computacional:** Calcular la descomposición en valores singulares (SVD) para matrices grandes es computacionalmente costoso ($O(N^3)$). La DCT es mucho más rápida ($O(N \log N)$), lo que permite tomar y guardar fotos al instante con el teléfono.

2. **Estandarización:** Almacenar una imagen comprimida en SVD requiere guardar tres matrices independientes (U, Sigma, V^T). JPEG cuenta con una forma estandarizada de almacenar coeficientes que es universalmente comprendida por navegadores y sistemas operativos.

3. **Optimización perceptual:** Las tablas de cuantificación JPEG están específicamente ajustadas para descartar información invisible para el ojo humano (cambios de color de alta frecuencia). La SVD descarta según la varianza matemática, que se correlaciona con la importancia visual, pero no está perfectamente ajustada a la biología humana.

7.4 ¿Dónde se utiliza realmente la SVD?

La SVD no es solo teórica. Es un caballo de batalla en otras áreas de la ciencia e ingeniería de datos:

- **Reducción de ruido (Denoising):** Al mantener los valores k superiores, se suele descartar el "ruido" (estática aleatoria) que suele residir en los valores singulares más pequeños. Esto limpia las señales de datos.

- **Sistemas de recomendación:** Netflix y Amazon utilizan algoritmos basados en SVD para predecir qué películas o productos te gustarán. Descomponen la enorme matriz "Usuario vs. Película" para encontrar patrones ocultos (características latentes) que conectan a usuarios con gustos similares.

- **Reducción de dimensionalidad (PCA):** el análisis de componentes principales, una técnica central en el aprendizaje automático para simplificar conjuntos de datos complejos, se basa fundamentalmente en SVD.

- **Motores de búsqueda (LSI):** la indexación semántica latente utiliza SVD para encontrar relaciones entre palabras y documentos, lo que ayuda a los motores de búsqueda a comprender que una búsqueda de “automóvil” también debe coincidir con páginas sobre “automóviles”.

Así que, aunque no guardes las fotos de tus vacaciones como `.svd` archivos, el algoritmo impulsa los motores de recomendación y las herramientas de búsqueda que usas a diario. Es un pilar del procesamiento de datos moderno.

7.5 Resumen

La SVD es una potente herramienta para visualizar datos. Nos enseña que una cuadrícula compleja de números a menudo puede describirse mediante unos pocos

vectores dominantes simples. Ya sea filtrando el ruido de una señal, comprimiendo una foto o analizando las relaciones en un conjunto de datos, el principio fundamental sigue siendo el mismo: **conservar la señal y descartar el ruido.**