

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM KHOA CÔNG NGHỆ THÔNG TIN**

\*\*\*\*\*\*\*\*

**BÁO CÁO GIỮA KỲ**

**MÔN HỌC: TOÁN CHO TRÍ TUỆ NHÂN TẠO**

**VẬN DUNG THUẬT TOÁN SVD CHO NÉN ẢNH**

**GVHD: TS. Bùi Mạnh Quân**

**Lớp: Chiều thứ 5 (tiết 7-10)**

**Sinh viên thực hiện:**

**Hồng Anh Khoa - 22110351**

**Tp. Hồ Chí Minh, tháng 12 năm 2021**

**MỤC LỤC**

[**PHẦN 1: GIỚI THIỆU** 1](#_Toc193335489)

[**PHẦN 2: NỘI DUNG** 1](#_Toc193335490)

[**2.1. Cơ sở lý thuyết của phân tích giá trị kỳ dị (SVD)** 1](#_Toc193335491)

[**2.2. Giải thích chi tiết về ma trận U, Σ, và V** 2](#_Toc193335492)

[**2.3. Ý nghĩa hình học của SVD** 2](#_Toc193335493)

[**2.4. Các tính chất quan trọng của SVD** 3](#_Toc193335494)

[**2.5. Các bước chi tiết để tính toán SVD của một ma trận (thuật toán)** 4](#_Toc193335495)

[**PHẦN 3: VẬN DỤNG** 6](#_Toc193335496)

[**3.1 Biểu diễn hình ảnh số dưới dạng ma trận** 6](#_Toc193335497)

[**3.2. Ảnh xám** 6](#_Toc193335498)

[**3.3. Ảnh màu** 7](#_Toc193335499)

[**3.4. Ứng dụng của SVD trong nén ảnh** 7](#_Toc193335500)

[**3.4.1. Nguyên lý cơ bản của nén ảnh bằng SVD** 7](#_Toc193335501)

[**3.4.2. Công thức toán học cho nén ảnh bằng SVD** 8](#_Toc193335502)

[**3.4.3. Các bước thực hiện nén ảnh bằng SVD** 9](#_Toc193335503)

[**3.4.4. Code Python minh họa việc nén ảnh** 10](#_Toc193335504)

[**3.4.5. Khôi phục ảnh sau khi nén** 16](#_Toc193335505)

[**3.3.6. Nhận xét bài toán** 19](#_Toc193335506)

[**3.5. Các vấn đề và Mở rộng** 20](#_Toc193335507)

[**PHẦN 4: KẾT LUẬN** 21](#_Toc193335508)

[**TÀI LIỆU THAM KHẢO** 21](#_Toc193335509)

# **PHẦN 1: GIỚI THIỆU**

Phân tích giá trị kỳ dị (Singular Value Decomposition), thường được viết tắt là SVD, là một phương pháp phân tích ma trận mạnh mẽ, có nguồn gốc từ hình học vi phân. Ban đầu, mục đích của phương pháp này là tìm ra một phép xoay không gian sao cho tích vô hướng của các vector không thay đổi, dẫn đến sự hình thành của khái niệm ma trận trực giao. SVD đã được phát triển dựa trên các tính chất của ma trận trực giao và ma trận đường chéo để tạo ra một ma trận xấp xỉ với ma trận gốc.

Ngày nay, SVD đã trở thành một công cụ cơ bản trong nhiều lĩnh vực, bao gồm xử lý tín hiệu, thống kê, học máy và đặc biệt là xử lý ảnh. Khả năng phân tích một ma trận phức tạp thành các thành phần đơn giản hơn đã mở ra nhiều ứng dụng quan trọng như giảm chiều dữ liệu, nén dữ liệu, hệ thống gợi ý và giải các bài toán hồi quy tuyến tính. Trong lĩnh vực xử lý ảnh, SVD được ứng dụng rộng rãi trong nén ảnh, giảm nhiễu, trích xuất đặc trưng và xác thực ảnh. Báo cáo này sẽ tập trung vào việc ứng dụng thuật toán SVD để nén ảnh, trình bày cơ sở lý thuyết, các bước thực hiện, ví dụ code và đánh giá kết quả.

# **PHẦN 2: NỘI DUNG**

## **2.1. Cơ sở lý thuyết của phân tích giá trị kỳ dị (SVD)**

Mọi ma trận **A** có kích thước m x n với các phần tử là số thực hoặc số phức đều có thể được phân tích thành tích của ba ma trận đặc biệt:

A = UΣVT

Trong đó:

* U là một ma trận trực giao kích thước m x m, các cột của nó là các vector riêng trái (left-singular vectors) của A. Ma trận này thỏa mãn UTU = UUT =Im, với Im là ma trận đơn vị kích thước m x m. Các cột của U cũng là các vector riêng trực chuẩn của ma trận AAT.
* Σ là một ma trận đường chéo kích thước m x n chứa các giá trị kỳ dị (singular values) trên đường chéo chính, được sắp xếp theo thứ tự giảm dần và không âm: σ1 ​≥ σ2 ​≥ ⋯ ≥ σr ​≥ 0 , trong đó (r) là hạng của ma trận A. Các phần tử còn lại của Σ (nếu có) đều bằng 0. Các giá trị kỳ dị σi là căn bậc hai của các trị riêng của cả AAT và ATA.
* V là một ma trận trực giao kích thước n x n, các cột của nó là các vector riêng phải (right-singular vectors) của ma trận . Ma trận này thỏa mãn VTV = VVT = In​, với In là ma trận đơn vị kích thước n x n. Các cột của ma trận V cũng là các vector riêng trực chuẩn của ma trận ATA. Trong biểu thức SVD, thường thấy sử dụng chuyển vị của V, tức là VT .

## **2.2. Giải thích chi tiết về ma trận U, Σ, và V**

Như đã đề cập ở trên, ma trận U và V là các ma trận trực giao, nghĩa là các cột của chúng tạo thành một hệ cơ sở trực chuẩn. Điều này có nghĩa là tích của ma trận với chuyển vị của chính nó (hoặc ngược lại) sẽ cho ra ma trận đơn vị. Về mặt hình học, các ma trận trực giao đại diện cho các phép xoay và/hoặc phản xạ trong không gian vector. Các cột của ma trận U chỉ ra các hướng trong không gian (m) chiều mà dữ liệu trong các hàng của ma trận A phân tán nhiều nhất, trong khi các cột của ma trận V chỉ ra các hướng trong không gian (n) chiều mà dữ liệu trong các cột của ma trận A phân tán nhiều nhất.

Ma trận **Σ** là một ma trận đường chéo mà các phần tử trên đường chéo chính là các giá trị kỳ dị, thường được sắp xếp theo thứ tự giảm dần. Các giá trị này cho biết "độ mạnh" hoặc "tầm quan trọng" của mỗi chiều trong việc biểu diễn ma trận A. Giá trị kỳ dị càng lớn thì chiều tương ứng càng quan trọng. Số lượng các giá trị kỳ dị khác không chính là hạng của ma trận A.

## **2.3. Ý nghĩa hình học của SVD**

Về mặt hình học, SVD có thể được hiểu là một sự phân tích một phép biến đổi tuyến tính thành ba bước: một phép xoay hoặc phản xạ VT, theo sau là một phép co giãn dọc theo các trục tọa độ **Σ**, và cuối cùng là một phép xoay hoặc phản xạ khác U. Nếu ta xem xét tác động của một ma trận A lên một hình cầu đơn vị trong không gian (n) chiều, kết quả sẽ là một hình elipsoid trong không gian (m) chiều. Các độ dài của các bán trục của elipsoid này chính là các giá trị kỳ dị của ma trận A. Các vector riêng phải V xác định hướng của các trục chính của hình cầu, và các vector riêng trái U xác định hướng của các trục chính của hình elipsoid sau khi biến đổi và co giãn theo các giá trị kỳ dị **Σ**.

## **2.4. Các tính chất quan trọng của SVD**

SVD có nhiều tính chất quan trọng, trong đó có một số tính chất liên quan trực tiếp đến ứng dụng nén ảnh:

* **Hạng của ma trận:** Hạng của ma trận A bằng số lượng các giá trị kỳ dị khác không trong ma trận **z**.
* **Biểu diễn dưới dạng tổng của các ma trận hạng 1:** Ma trận A có thể được biểu diễn dưới dạng tổng của các ma trận hạng 1:



\*Trong đó:

(r) là hạng của ma trận A,

là các giá trị kỳ dị,

ui là các vector riêng trái và vi là các vector riêng phải.

Tính chất này là nền tảng cho việc nén ảnh bằng cách chỉ giữ lại một số hữu hạn các thành phần đầu tiên trong tổng này.

* **Xấp xỉ ma trận hạng thấp tốt nhất:** **(Low-rank Approximation)**

Nếu ta chỉ giữ lại (k) giá trị kỳ dị lớn nhất và các vector riêng trái và phải tương ứng, ta sẽ thu được một ma trận:

Ak ​= Uk​ Σk​ VkT​

Trong đó:

* Uk là ma trận chứa **k** cột đầu tiên của U.
* Vk ​ là ma trận chứa **k** cột đầu tiên của V.
* Σk là ma trận đường chéo kích thước k×k, chỉ chứa k giá trị kỳ dị lớn nhất của Σ.

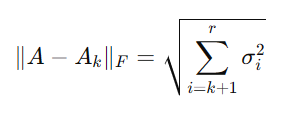
Σk =

Ma trận Ak ​ có hạng tối đa là **k**, trong khi ma trận A có hạng là r.

**\*Định lý Eckart-Young-Mirsky**:

Xấp xỉ hạng thấp Ak là xấp xỉ **tốt nhất** của ma trận A theo chuẩn Frobenius hoặc chuẩn bậc hai (Spectral norm).

\*Chuẩn Frobenius:



Sai số bình phương tổng của xấp xỉ chỉ phụ thuộc vào các giá trị kỳ dị bị bỏ qua.

\*Chuẩn bậc hai:

∥ A − Ak​ ∥2 ​= σk+1​

Sai số tối đa chính là giá trị kỳ dị lớn nhất bị bỏ qua.

**Vì thế, nếu ta bỏ qua bất kỳ giá trị kỳ dị lớn nào và giữ lại các giá trị nhỏ hơn, sai số sẽ lớn hơn so với việc giữ lại k giá trị kỳ dị lớn nhất.** Do đó, **không có cách nào tốt hơn cách này** để tìm một ma trận hạng k xấp xỉ A.

Điều này có nghĩa là nếu ta muốn tìm **ma trận hạng k tốt nhất** để xấp xỉ A, thì **không có cách nào tốt hơn cách lấy k giá trị kỳ dị lớn nhất**.

Tính chất này cho phép xấp xỉ một ma trận A bằng một ma trận hạng thấp hơn nhưng vẫn giữ được nhiều thông tin quan trọng nhất có thể. Đây là nền tảng của nhiều ứng dụng như **nén ảnh, giảm chiều dữ liệu và khử nhiễu**.

## **2.5. Các bước chi tiết để tính toán SVD của một ma trận (thuật toán)**

Mặc dù việc tính toán SVD thường được thực hiện bằng các thư viện toán học chuyên dụng, việc hiểu các bước cơ bản là hữu ích:

1. **Tính ma trận ATA và AAT**.
2. **Tìm các trị riêng và vector riêng của ATA.** Các trị riêng này sẽ là bình phương của các giá trị kỳ dị , và các vector riêng tương ứng là các cột của ma trận V (sau khi chuẩn hóa).
3. **Tính các giá trị kỳ dị σi bằng cách lấy căn bậc hai của các trị riêng**. Sắp xếp các giá trị kỳ dị theo thứ tự giảm dần để tạo thành ma trận đường chéo Σ.
4. **Tìm các vector riêng trái bằng cách sử dụng công thức cho các giá trị kỳ dị khác không**. Các vector này sẽ là các cột của ma trận U (sau khi chuẩn hóa).
5. **Xây dựng ma trận từ U, Σ, và V**

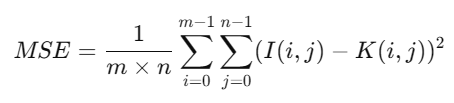
Đối với các trường hợp mà m n, hoặc khi có các giá trị kỳ dị bằng không, cần tìm thêm các vector trực giao để hoàn thiện cơ sở trực chuẩn cho không gian cột của U và không gian hàng của.

**2.6. Đánh giá chất lượng ảnh nén**

**2.6.1. MSE (Mean Squared Error - Sai số bình phương trung bình)**

SE đo lường sự khác biệt trung bình bình phương giữa ảnh gốc và ảnh nén. Nó giúp đánh giá mức độ mất mát thông tin khi nén ảnh.

Công thức MSE:



Trong đó:

* m,n là kích thước của ảnh (chiều cao và chiều rộng).
* I(i,j) là giá trị pixel tại vị trí (i, j) trong ảnh gốc.
* K(i,j) là giá trị pixel tại vị trí (i,j) trong ảnh đã nén.

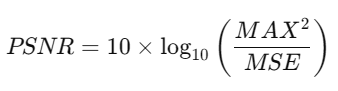
**Ý nghĩa:**

* Nếu MSE = 0, hai ảnh hoàn toàn giống nhau.
* MSE càng lớn thì ảnh bị biến dạng nhiều hơn sau khi nén.

**2.6.2. PSNR (Peak Signal-to-Noise Ratio - Tỉ số tín hiệu trên nhiễu đỉnh)**

PSNR sử dụng MSE để tính toán mức độ trung thực của ảnh nén so với ảnh gốc. Nó cho biết ảnh nén có giữ được độ trung thực so với ảnh gốc hay không.

Công thức PSNR:

****

Trong đó:

* MAX là giá trị pixel tối đa của ảnh (thường là 255 đối với ảnh 8-bit).
* MSE là sai số bình phương trung bình.

Ý nghĩa:

* PSNR cao nghĩa là ảnh nén gần giống ảnh gốc (tín hiệu cao, nhiễu thấp).
* PSNR thấp nghĩa là ảnh bị mất thông tin nhiều khi nén.
* Giá trị PSNR thường nằm trong khoảng:
* 30 - 50 dB: Ảnh nén có chất lượng tốt.
* 20 - 30 dB: Ảnh bị méo đáng kể.
* Dưới 20 dB: Ảnh bị mất chất lượng nghiêm trọng.

# **PHẦN 3: VẬN DỤNG**

## **3.1 Biểu diễn hình ảnh số dưới dạng ma trận**

Để áp dụng SVD cho nén ảnh, trước tiên chúng ta cần hiểu cách biểu diễn một hình ảnh số dưới dạng ma trận. Một hình ảnh số có thể được xem như một mảng hai chiều (ma trận) các giá trị, trong đó mỗi phần tử của ma trận tương ứng với một điểm ảnh (pixel). Giá trị của mỗi pixel đại diện cho độ sáng hoặc màu sắc tại vị trí đó.

## **3.2. Ảnh xám**

Trong ảnh xám (grayscale image), mỗi pixel thường được biểu diễn bằng một giá trị duy nhất, cho biết cường độ sáng của pixel đó. Thông thường, giá trị này là một số nguyên nằm trong khoảng từ 0 đến 255, trong đó 0 tương ứng với màu đen, 255 tương ứng với màu trắng và các giá trị trung gian đại diện cho các sắc độ xám khác nhau. Do đó, một ảnh xám có kích thước m x n pixel có thể được biểu diễn bằng một ma trận kích thước m x n, trong đó phần tử ở hàng (i) và cột (j) chứa giá trị độ sáng của pixel tại vị trí đó.

## **3.3. Ảnh màu**

Ảnh màu thường sử dụng mô hình RGB (Red, Green, Blue), trong đó mỗi pixel được biểu diễn bằng ba giá trị, tương ứng với cường độ của ba màu cơ bản: đỏ, xanh lá cây và xanh dương. Tương tự như ảnh xám, các giá trị này thường là các số nguyên từ 0 đến 255. Một ảnh màu có kích thước m x n pixel có thể được biểu diễn bằng một mảng ba chiều (tensor) kích thước m x n x 3, trong đó mỗi phần tử ((i, j, k)) chứa giá trị cường độ của kênh màu (k) (với (k = 0) cho đỏ, (k = 1) cho xanh lá cây và (k = 2) cho xanh dương) tại pixel ((i, j)). Trong trường hợp này, để áp dụng SVD, chúng ta thường xử lý từng kênh màu (đỏ, xanh lá cây, xanh dương) một cách riêng biệt như các ma trận hai chiều.

## **3.4. Ứng dụng của SVD trong nén ảnh**

### **3.4.1. Nguyên lý cơ bản của nén ảnh bằng SVD**

Nguyên lý cơ bản của việc nén ảnh bằng SVD dựa trên tính chất xấp xỉ ma trận hạng thấp tốt nhất. Như đã đề cập ở phần lý thuyết, ma trận ảnh A có thể được phân tích thành:

A = UΣVT

Các giá trị kỳ dị trong ma trận Σ được sắp xếp theo thứ tự giảm dần, cho thấy mức độ quan trọng của các thành phần tương ứng trong việc tái tạo lại ma trận gốc. Các giá trị kỳ dị lớn hơn mang nhiều thông tin quan trọng hơn về hình ảnh, trong khi các giá trị nhỏ hơn thường chứa thông tin ít quan trọng hoặc nhiễu.

Để nén ảnh, chúng ta có thể chọn một số lượng (k) nhỏ các giá trị kỳ dị lớn nhất (với (k < rank(A)) và các vector riêng trái và phải tương ứng để xây dựng một ma trận xấp xỉ

Ak ​= Uk​ Σk​ VkT​.

Trong đó: Uk bao gồm (k) cột đầu tiên của ma trận U, Σk​ là ma trận đường chéo k x k chứa (k) giá trị kỳ dị lớn nhất, và VkT​ bao gồm (k) hàng đầu tiên của ma trận VT​. (tương ứng với (k) cột đầu tiên của ma trận V. Ma trận Ak sẽ là một xấp xỉ hạng (k) của ma trận A, và nếu (k) được chọn đủ nhỏ, chúng ta có thể giảm đáng kể dung lượng lưu trữ cần thiết để biểu diễn hình ảnh.

### **3.4.2. Công thức toán học cho nén ảnh bằng SVD**

Cho ma trận ảnh A có kích thước m x n. Phân tích SVD của A là:

A​= Um x m ​ Σm x n​ (Vn x n​)T.

Để nén ảnh, chúng ta chọn (k) giá trị kỳ dị lớn nhất σ1, σ2, σ3,…, σk và các vector riêng tương ứng. Ma trận xấp xỉ Ak được tính như sau:

Ak ​= Um x k ​ Σk x k (Vn x k)T.

Trong đó:

* Um x k là ma trận bao gồm (k) cột đầu tiên của ma trận U.
* Σk x k là ma trận đường chéo kích thước k x k với các phần tử trên đường chéo là (σ1​, σ2​, ... , σk​).
* Vn x k *là ma trận bao gồm (k) cột đầu tiên của ma trận V, và* (Vn x k)T là chuyển vị của nó.

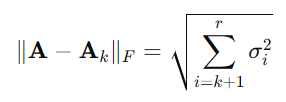
Dung lượng lưu trữ cho ma trận gốc A là m x n phần tử. Để lưu trữ ma trận nén Ak, chúng ta cần lưu trữ ma trận Um x k  (có m x k phần tử), ma trận Σk x k (chỉ cần lưu (k) phần tử trên đường chéo), và ma trận (Vn x k)T (có n x k phần tử). Tổng số phần tử cần lưu trữ cho ảnh nén là m \* k + k + n \* k = k(m + n + 1) .

Tỷ lệ nén (Compression Ratio - CR) được tính bằng tỷ lệ giữa số phần tử của ma trận gốc và ma trận nén:

Giá trị CR càng lớn cho thấy mức độ nén càng cao, nhưng đồng thời chất lượng ảnh có thể giảm.

\*Chất lượng ảnh sau khi nén

* Khi k nhỏ, Ak ​ chỉ giữ lại các thành phần chính, làm mất nhiều chi tiết của ảnh.
* Khi k lớn, Ak ​ giữ lại nhiều thông tin hơn, làm giảm sai số xấp xỉ nhưng tăng kích thước lưu trữ.

Để đánh giá chất lượng ảnh nén, ta có thể dùng **sai số Frobenius** giữa ảnh gốc và ảnh nén:

Trong đó r là hạng của ma trận A. Sai số này giảm khi k tăng, nghĩa là càng giữ nhiều giá trị kỳ dị, ảnh nén càng giống ảnh gốc.

### **3.4.3. Các bước thực hiện nén ảnh bằng SVD**

Để nén một hình ảnh bằng SVD, chúng ta có thể thực hiện theo các bước sau:

1. **Đọc hình ảnh và chuyển nó thành một ma trận số**. Đối với ảnh xám, ma trận sẽ có kích thước m x n. Đối với ảnh màu, chúng ta có thể xử lý từng kênh màu (R, G, B) một cách riêng biệt, tạo ra ba ma trận có cùng kích thước.
2. **Tính toán phân tích giá trị kỳ dị (SVD) của ma trận ảnh (hoặc từng ma trận kênh màu)** . Sử dụng các thư viện toán học như NumPy trong Python để thực hiện SVD, thu được các ma trận U, Σ (thường trả về dưới dạng một vector các giá trị trên đường chéo) và VT (hoặc (V)).
3. **Chọn số lượng (k) giá trị kỳ dị lớn nhất để giữ lại**. Giá trị của (k) sẽ quyết định mức độ nén và chất lượng của ảnh nén. (k) càng nhỏ thì mức độ nén càng cao nhưng chất lượng ảnh có thể giảm.
4. **Tạo ma trận đường chéo** Σk **kích thước k x k từ (k) giá trị kỳ dị lớn nhất**.
5. **Lấy (k) cột đầu tiên của ma trận U (Uk) và (k) cột đầu tiên của ma trận V (Vk)**.
6. **Tái tạo ma trận ảnh nén Ak bằng cách nhân ba ma trận Uk,** Σk **và** VkT .
7. **Hiển thị hoặc lưu trữ ma trận Ak dưới dạng hình ảnh**. Đối với ảnh màu, cần kết hợp các kênh màu đã nén lại để tạo thành ảnh màu nén cuối cùng.

### **3.4.4. Code Python minh họa việc nén ảnh**

**\*Code Python thực hiện nén ảnh xám bằng SVD:**

|  |
| --- |
| pip install numpy opencv-python matplotlib imageio |

→ Cài đặt thư viện cần thiết

|  |
| --- |
| import cv2  import numpy as np  import os  import matplotlib.pyplot as plt  # Đọc ảnh và chuyển thành ảnh xám  image\_path = r'C:\Users\anhkh\OneDrive\Desktop\ToanTriTueNhanTao\img.jpg'  image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE).astype(np.uint8) |

→ Đọc ảnh và chuyển thành ảnh xám

|  |
| --- |
| # Kích thước ảnh gốc  m, n = image.shape  original\_size = os.path.getsize(image\_path)  # Dung lượng file gốc (bytes)  print(f"\n📂 Dung lượng ảnh gốc: {original\_size / 1024:.2f} kB")  print(f"📏 Kích thước ảnh gốc: {m} x {n} pixels")  # Lưu ảnh gốc với tên "origin.jpg"  origin\_path = r'C:\Users\anhkh\OneDrive\Desktop\ToanTriTueNhanTao\baocao\origin.jpeg'  cv2.imwrite(origin\_path, image, [cv2.IMWRITE\_JPEG\_QUALITY, 90])  # Hiển thị ảnh gốc  plt.figure(figsize=(6, 6))  plt.imshow(image, cmap='gray')  plt.title("Ảnh Gốc")  plt.axis('off')  plt.show() |

→Lấy kích thước, dung lượng và hiện, lưu ảnh gốc sau khi chuyển thành ảnh xám

|  |
| --- |
| # Tính SVD  U, S, Vt = np.linalg.svd(image, full\_matrices=False) |

→ Thực hiện **phân rã SVD** để tách ảnh thành 3 ma trận.

|  |
| --- |
| # Hiển thị giá trị kỳ dị quan trọng nhất  print("\n 10 Giá trị kỳ dị đầu tiên:")  print(S[:10])  # Tính toán tổng giá trị kỳ dị để đánh giá mức độ nén  total\_energy = np.sum(S)  energy\_ratios = np.cumsum(S) / total\_energy  # Vẽ biểu đồ tích lũy giá trị kỳ dị  plt.figure(figsize=(8, 5))  plt.plot(energy\_ratios, marker='o', linestyle='--')  plt.xlabel("Số lượng giá trị kỳ dị giữ lại (k)")  plt.ylabel("Tỉ lệ thông tin được bảo toàn")  plt.title("Phân tích mức độ nén dựa trên số giá trị kỳ dị")  plt.grid()  plt.show() |

→ Vẽ biểu đồ phân tích mức độ nén

|  |
| --- |
| # Hàm tính toán MSE  def calculate\_mse(original, compressed):      return np.mean((original - compressed) \*\* 2)  # Hàm tính toán PSNR  def calculate\_psnr(original, compressed):      mse = calculate\_mse(original, compressed)      if mse == 0:          return float('inf')  # Trường hợp ảnh không có lỗi      max\_pixel = 255.0      return 10 \* np.log10((max\_pixel \*\* 2) / mse)  def compress\_image(U, S, Vt, k):      """Hàm tái tạo ảnh từ SVD với k giá trị kỳ dị"""      Uk = U[:, :k]  # Chỉ lấy k cột đầu của U      Sk = np.diag(S[:k])  # Chỉ lấy k phần tử đầu của Sigma      Vtk = Vt[:k, :]  # Chỉ lấy k hàng đầu của Vt      return np.dot(Uk, np.dot(Sk, Vtk))  # A\_k = U\_k \* S\_k \* V\_k^T |

→ Xây dụng hàm nén ảnh: Uk​Sk​VkT​

|  |
| --- |
| # Chọn các mức k để kiểm tra mức độ nén  k\_values = [5, 20, 50, 100]  # Hiển thị ảnh với các mức nén khác nhau  fig, axes = plt.subplots(1, len(k\_values) + 1, figsize=(15, 6))  # Ảnh gốc  axes[0].imshow(image, cmap='gray')  axes[0].set\_title("Ảnh Gốc")  axes[0].axis('off')  # Ảnh sau khi nén  for i, k in enumerate(k\_values):      compressed\_img = compress\_image(U, S, Vt, k)      axes[i + 1].imshow(compressed\_img, cmap='gray')      axes[i + 1].set\_title(f"k = {k}")      axes[i + 1].axis('off')  plt.show() |

→ cho các giá trị k để kiểm tra mức độ nén và hiển thị ảnh sau khi nén

|  |
| --- |
| # Kiểm tra dung lượng ảnh trước và sau nén  for k in k\_values:      # Tính số phần tử lưu trữ sau nén      compressed\_size\_pixels = (m \* k) + k + (n \* k)        # Tính dung lượng ảnh sau nén (giả sử mỗi phần tử lưu trữ 1 byte)      compressed\_size\_bytes = compressed\_size\_pixels  # Đơn vị bytes (xấp xỉ)      compression\_ratio = original\_size / compressed\_size\_bytes      # Lưu ảnh nén ra file JPG để kiểm tra dung lượng thực tế      compressed\_image = compress\_image(U, S, Vt, k)      compressed\_path = f'compressed\_k{k}.jpeg'       # Đảm bảo ảnh sau nén chỉ có 1 kênh grayscale trước khi lưu      compressed\_image = np.clip(compressed\_image, 0, 255).astype(np.uint8)  # Đảm bảo dữ liệu hợp lệ      # Không cần chuyển về BGR nữa, giữ nguyên ảnh xám      cv2.imwrite(compressed\_path, compressed\_image, [cv2.IMWRITE\_JPEG\_QUALITY, 90])        # Tính MSE và PSNR      mse\_value = calculate\_mse(image, compressed\_image)      psnr\_value = calculate\_psnr(image, compressed\_image)      # Dung lượng file ảnh sau khi nén      actual\_compressed\_size = os.path.getsize(compressed\_path)      print(f"\n📉 Đối với k = {k}:")      print(f"  🔹 Kích thước ảnh gốc: {m \* n} pixels")      print(f"  🔹 Kích thước ảnh nén (ma trận): {compressed\_size\_pixels:.0f} pixels")      print(f"  🔹 Dung lượng ảnh sau nén (ước lượng): {compressed\_size\_bytes / 1024:.2f} kB")      print(f"  🔹 Dung lượng file thực tế: {actual\_compressed\_size / 1024:.2f} kB")      print(f"  🔹 Tỉ lệ nén (ước lượng): {compression\_ratio:.2f}x")      print(f"  🔹 Tỉ lệ nén (thực tế): {original\_size / actual\_compressed\_size:.2f}x")      print(f"  🔹 MSE: {mse\_value:.4f}")      print(f"  🔹 PSNR: {psnr\_value:.2f} dB") |

→ Hàm kiểm tra thông tin ảnh sau khi nén:

Ví dụ:

Đối với k = 5:

🔹 Kích thước ảnh gốc: 1638400 pixels

🔹 Kích thước ảnh nén (ma trận): 12805 pixels

🔹 Dung lượng ảnh sau nén (ước lượng): 12.50 kB

🔹 Dung lượng file thực tế: 80.03 kB

🔹 Tỉ lệ nén (ước lượng): 6.08x

🔹 Tỉ lệ nén (thực tế): 0.95x

🔹 MSE: 54.0903

🔹 PSNR: 30.80 dB

**\*Code Python thực hiện nén ảnh màu bằng SVD:**

|  |
| --- |
| import cv2  import numpy as np  import os  import matplotlib.pyplot as plt |

→ Cài các thư viện cần thiết

|  |
| --- |
| def mse(original, compressed):      return np.mean((original - compressed) \*\* 2)  def psnr(original, compressed):      mse\_value = mse(original, compressed)      if mse\_value == 0:          return float('inf')      max\_pixel = 255.0      return 20 \* np.log10(max\_pixel / np.sqrt(mse\_value))  def compress\_channel(channel, k):      U, S, Vt = np.linalg.svd(channel, full\_matrices=False)      Uk = U[:, :k]      Sk = np.diag(S[:k])      Vtk = Vt[:k, :]      return np.dot(Uk, np.dot(Sk, Vtk))  def compress\_image\_color(image, k):      compressed\_channels = [compress\_channel(image[:, :, i], k) for i in range(3)]      compressed\_image = np.stack(compressed\_channels, axis=2)      return np.clip(compressed\_image, 0, 255).astype(np.uint8) |

→ Định nghĩa các hàm:

MSE (Mean Squared Error): Đo lường mức độ khác biệt giữa ảnh gốc và ảnh nén

PSNR (Peak Signal-to-Noise Ratio): Đánh giá chất lượng ảnh theo thang dB (decibel).

compress\_channel: Dùng **SVD** để phân rã ma trận ảnh

compress\_image\_color: Nén toàn bộ ảnh màu

|  |
| --- |
| #Đọc ảnh màu  image\_path = r'C:\Users\anhkh\OneDrive\Desktop\ToanTriTueNhanTao\img.jpg'  # Cập nhật đường dẫn ảnh của bạn  image = cv2.imread(image\_path)  image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  # Chuyển sang RGB  original\_size = os.path.getsize(image\_path)  # Dung lượng ảnh gốc (bytes)  m, n, \_ = image.shape  # Kích thước ảnh gốc |

→ Đọc và hiển thị ảnh

|  |
| --- |
| # Chọn các mức nén  k\_values = [5, 20, 50, 100]  fig, axes = plt.subplots(1, len(k\_values) + 1, figsize=(15, 6))  axes[0].imshow(image)  axes[0].set\_title("Ảnh Gốc")  axes[0].axis('off')  for i, k in enumerate(k\_values):      compressed\_img = compress\_image\_color(image, k)      axes[i + 1].imshow(compressed\_img)      axes[i + 1].set\_title(f"k = {k}")      axes[i + 1].axis('off')  plt.show() |

→ Nén ảnh với các giá trị k khác nhau và xuất các ảnh đã nén theo (k)

|  |
| --- |
| # Đánh giá chất lượng ảnh  for k in k\_values:      compressed\_img = compress\_image\_color(image, k)      # Tính MSE & PSNR      mse\_value = mse(image, compressed\_img)      psnr\_value = psnr(image, compressed\_img)      # Tính toán kích thước ảnh nén      compressed\_size\_pixels = (m \* k) \* 3 + k \* 3 + (n \* k) \* 3  # Tổng số phần tử lưu trữ      compressed\_size\_bytes = compressed\_size\_pixels  # Giả sử mỗi phần tử chiếm 1 byte      compression\_ratio\_estimated = original\_size / compressed\_size\_bytes      # Lưu ảnh nén để kiểm tra dung lượng thực tế      compressed\_path = f'compressed\_k{k}.jpeg'      cv2.imwrite(compressed\_path, cv2.cvtColor(compressed\_img, cv2.COLOR\_RGB2BGR), [cv2.IMWRITE\_JPEG\_QUALITY, 90])      actual\_compressed\_size = os.path.getsize(compressed\_path)  # Dung lượng thực tế (bytes)      compression\_ratio\_actual = original\_size / actual\_compressed\_size      print(f"\n📉 Đối với k = {k}:")      print(f"  🔹 Kích thước ảnh gốc: {m} x {n} pixels")      print(f"  🔹 Kích thước ảnh nén (ước lượng): {compressed\_size\_pixels:.0f} pixels")      print(f"  🔹 Dung lượng ảnh gốc: {original\_size / 1024:.2f} kB")      print(f"  🔹 Dung lượng ảnh nén (ước lượng): {compressed\_size\_bytes / 1024:.2f} kB")      print(f"  🔹 Dung lượng file thực tế: {actual\_compressed\_size / 1024:.2f} kB")      print(f"  🔹 Tỉ lệ nén (ước lượng): {compression\_ratio\_estimated:.2f}x")      print(f"  🔹 Tỉ lệ nén (thực tế): {compression\_ratio\_actual:.2f}x")      print(f"  🔹 MSE: {mse\_value:.2f}")      print(f"  🔹 PSNR: {psnr\_value:.2f} dB") |

→ In ra màn hình các giá trị thu được:

Ví dụ:

Đối với k = 20:

🔹 Kích thước ảnh gốc: 1280 x 1280 pixels

🔹 Kích thước ảnh nén (ước lượng): 153660 pixels

🔹 Dung lượng ảnh gốc: 76.06 kB

🔹 Dung lượng ảnh nén (ước lượng): 150.06 kB

🔹 Dung lượng file thực tế: 138.41 kB

🔹 Tỉ lệ nén (ước lượng): 0.51x

🔹 Tỉ lệ nén (thực tế): 0.55x

🔹 MSE: 36.53

🔹 PSNR: 32.50 dB

→ In ra màn hình các giá trị thu được:

### 

### **3.4.5. Khôi phục ảnh sau khi nén**

Ảnh sau khi nén bằng phương pháp SVD (Singular Value Decomposition) có thể phục hồi một phần nhưng không thể khôi phục hoàn toàn như ảnh gốc, trừ khi k (số giá trị kỳ dị giữ lại) bằng đúng với số lượng gốc của nó.

**Nguyên nhân:**

* Mất dữ liệu: Khi giảm số lượng k, ta loại bỏ một số thành phần chứa thông tin chi tiết của ảnh. Những thông tin này không thể được khôi phục đầy đủ.
* Nén tổn hao: Quá trình chỉ giữ lại một số giá trị kỳ dị lớn nhất làm mất đi các chi tiết nhỏ của ảnh.
* Làm tròn số: Khi tính toán SVD và lưu ảnh, có sai số làm ảnh bị biến đổi nhẹ.

**Ý tưởng khôi phục ảnh nén:**

* Khi nén ảnh bằng SVD, ta chỉ giữ lại **k** giá trị số hạng kỳ dị đầu tiên.
* Nếu tăng **k**, ta có thể phục hồi ảnh gần với ảnh gốc hơn.
* Tuy nhiên, nếu **k ban đầu quá nhỏ**, nhiều thông tin bị mất và **không thể khôi phục hoàn toàn**.

Nếu **k rất lớn** (gần bằng kích thước gốc của ma trận), ảnh nén có chất lượng gần giống ảnh ban đầu.

**Kết luận:** Nén ảnh bằng SVD giúp giảm kích thước nhưng là một dạng **nén tổn hao**, nên không thể khôi phục ảnh 100% như ban đầu nếu đã loại bỏ dữ liệu.

\*Demo đoạn code khôi phục ảnh: (sư dụng nội suy)

|  |
| --- |
| import numpy as np  import cv2  import os  from scipy.interpolate import interp1d  def compress\_channel(channel, k):      """Nén 1 kênh màu bằng SVD với k thành phần chính"""      U, S, Vt = np.linalg.svd(channel, full\_matrices=False)      return U[:, :k] @ np.diag(S[:k]) @ Vt[:k, :], U, S, Vt  # Trả về ảnh nén và các ma trận SVD  def compress\_image(image, k):      """Nén ảnh màu bằng SVD"""      compressed\_channels = []      svd\_data = []  # Lưu U, S, Vt cho việc khôi phục      for i in range(3):          compressed, U, S, Vt = compress\_channel(image[:, :, i], k)          compressed\_channels.append(compressed)          svd\_data.append((U, S, Vt))      return np.clip(np.stack(compressed\_channels, axis=2), 0, 255).astype(np.uint8), svd\_data  def interpolate\_singular\_values(S, k\_target):      """Nội suy giá trị kỳ dị để tăng k lên k\_target"""      k\_existing = len(S)      x\_existing = np.arange(k\_existing)      x\_target = np.arange(k\_target)        f\_interp = interp1d(x\_existing, S, kind='quadratic', fill\_value='extrapolate')      S\_interpolated = f\_interp(x\_target)      S\_interpolated[S\_interpolated < 0] = 0  # Đảm bảo không có giá trị âm        return S\_interpolated  def recover\_channel(U, S, Vt, k\_existing, k\_target):      """Khôi phục một kênh màu với số k giá trị kỳ dị, có nội suy"""      S\_new = interpolate\_singular\_values(S[:k\_existing], k\_target)      return np.clip(U[:, :k\_target] @ np.diag(S\_new) @ Vt[:k\_target, :], 0, 255)  def recover\_image(svd\_data, k\_existing, k\_target):      """Khôi phục ảnh từ dữ liệu SVD có nội suy"""      recovered\_channels = [recover\_channel(U, S, Vt, k\_existing, k\_target) for U, S, Vt in svd\_data]      return np.clip(np.stack(recovered\_channels, axis=2), 0, 255).astype(np.uint8)  # Đọc ảnh và chuyển sang RGB  image\_path = os.path.join("img.jpg")  image = cv2.imread(image\_path)  image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  # Bước 1: Nén ảnh với k = 20  k\_compressed = 20  compressed\_img, svd\_data = compress\_image(image, k\_compressed)  # Bước 2: Từ ảnh đã nén, cố gắng khôi phục với k = 50 bằng nội suy  k\_recovered = 50  recovered\_img = recover\_image(svd\_data, k\_compressed, k\_recovered)  # Lưu ảnh khôi phục  cv2.imwrite("recovered.jpg", cv2.cvtColor(recovered\_img, cv2.COLOR\_RGB2BGR))  # Hiển thị kết quả  fig, axes = plt.subplots(1, 3, figsize=(15, 6))  axes[0].imshow(image)  axes[0].set\_title("Ảnh Gốc")  axes[0].axis('off')  axes[1].imshow(compressed\_img)  axes[1].set\_title(f"Ảnh Nén (k={k\_compressed})")  axes[1].axis('off')  axes[2].imshow(recovered\_img)  axes[2].set\_title(f"Ảnh Phục Hồi (k={k\_recovered})")  axes[2].axis('off')  plt.show() |



### **3.3.6. Nhận xét bài toán**

**Ảnh màu**

* Khi giá trị k tăng, chất lượng ảnh được cải thiện đáng kể, thể hiện qua chỉ số PSNR tăng và MSE giảm. Ví dụ, với k = 5, PSNR chỉ đạt 30.73 dB, trong khi với k = 100, PSNR đạt 38.49 dB.
* Dung lượng ảnh sau nén không hoàn toàn tuân theo ước lượng lý thuyết. Dù về mặt lý thuyết, tỉ lệ nén có thể đạt mức cao (2.03x với k = 5), nhưng trong thực tế, tỉ lệ nén thực tế thường thấp hơn (chỉ 0.75x với k = 5). Điều này có thể do ảnh hưởng của định dạng lưu trữ và mức độ phức tạp của ảnh gốc.
* Khi k tăng lên, sự khác biệt giữa dung lượng ước lượng và dung lượng thực tế thu hẹp lại. Điều này cho thấy rằng đối với ảnh màu, việc chọn k hợp lý cần cân nhắc cả chất lượng hình ảnh và mức độ giảm dung lượng thực tế.

**Ảnh xám**

* Tương tự ảnh màu, khi k tăng, chất lượng ảnh xám cũng cải thiện với PSNR tăng và MSE giảm.
* Tuy nhiên, ảnh xám có tỉ lệ nén tốt hơn so với ảnh màu. Ví dụ, với k = 5, tỉ lệ nén ước lượng đạt 6.08x, cao hơn nhiều so với ảnh màu. Điều này hợp lý vì ảnh xám chỉ có một kênh thay vì ba kênh màu.
* Mặc dù tỉ lệ nén ước lượng khá cao, dung lượng file thực tế không giảm đáng kể, chỉ đạt mức 0.95x với k = 5. Điều này có thể do các yếu tố liên quan đến cách lưu trữ dữ liệu của ảnh.

**Tổng kết chung**

* Phương pháp nén ảnh sử dụng SVD mang lại hiệu quả cao, đặc biệt trong việc giảm kích thước ảnh mà vẫn giữ được chất lượng hình ảnh ở mức chấp nhận được.
* Đối với ảnh màu, k = 50 hoặc k = 100 có vẻ là lựa chọn tối ưu giữa dung lượng và chất lượng.
* Đối với ảnh xám, giá trị k thấp hơn có thể mang lại tỉ lệ nén tốt hơn mà không làm giảm chất lượng ảnh quá nhiều.
* Mặc dù phương pháp này giúp nén ảnh hiệu quả, nhưng dung lượng thực tế vẫn bị ảnh hưởng bởi cách mã hóa và lưu trữ, dẫn đến sự chênh lệch giữa tỉ lệ nén lý thuyết và thực tế.

## **3.5. Các vấn đề và Mở rộng**

Phương pháp nén ảnh bằng SVD có một số ưu điểm. Thứ nhất, nó tương đối đơn giản và dễ hiểu về mặt toán học. Thứ hai, nó không gây ra hiệu ứng khối (blocking artifacts) thường thấy ở các phương pháp nén dựa trên khối như JPEG khi nén ở mức độ cao.

Tuy nhiên, SVD cũng có một số nhược điểm. Hiệu suất nén của SVD có thể không cao bằng các phương pháp nén chuyên dụng như JPEG ở cùng mức chất lượng. Điều này là do JPEG tận dụng các đặc tính cụ thể của ảnh (ví dụ: sự nhạy cảm khác nhau của mắt người đối với các tần số không gian khác nhau) để đạt được tỷ lệ nén tốt hơn.

Hiệu suất nén của SVD cũng phụ thuộc vào đặc điểm của ảnh. Các ảnh có cấu trúc lặp lại hoặc ma trận có hạng thấp sẽ nén tốt hơn so với các ảnh phức tạp hoặc có ma trận hạng cao.

Ngoài ứng dụng nén ảnh, SVD còn có nhiều ứng dụng khác trong xử lý ảnh. **giảm nhiễu ảnh (Image Denoising)**. Sử dụng trong **watermarking**, bằng cách nhúng dấu bản quyền vào các thành phần SVD của ảnh. Các ứng dụng khác bao gồm **phục hồi ảnh bị hỏng (Image Inpainting)** và **trích xuất đặc trưng ảnh (Image Feature Extraction)**.

Lý thuyết SVD cũng có nhiều mở rộng. **Tensor SVD** là một mở rộng của SVD cho dữ liệu đa chiều như ảnh màu (xem như một tensor 3 chiều) và video (xem như một tensor 4 chiều). **Randomized SVD** là các phương pháp tính toán SVD hiệu quả hơn cho các ma trận lớn, thường được sử dụng trong các ứng dụng học máy và phân tích dữ liệu lớn. SVD cũng có thể được áp dụng cho các cấu trúc dữ liệu khác như đồ thị.

# **PHẦN 4: KẾT LUẬN**

Phân tích giá trị kỳ dị (SVD) là một công cụ toán học mạnh mẽ với nhiều ứng dụng trong xử lý ảnh, đặc biệt là trong nén ảnh. Bằng cách tận dụng tính chất xấp xỉ ma trận hạng thấp tốt nhất của SVD, chúng ta có thể giảm đáng kể dung lượng lưu trữ cần thiết để biểu diễn một hình ảnh bằng cách chỉ giữ lại một số lượng nhỏ các giá trị kỳ dị quan trọng nhất và các vector riêng tương ứng. Mức độ nén và chất lượng ảnh nén có thể được điều chỉnh thông qua việc lựa chọn số lượng giá trị kỳ dị được giữ lại. Mặc dù hiệu suất nén của SVD có thể không sánh bằng các phương pháp chuyên dụng trong mọi trường hợp, nhưng tính đơn giản và khả năng tránh được các hiệu ứng khối làm cho nó trở thành một lựa chọn hữu ích trong nhiều ứng dụng. Hơn nữa, SVD còn là một công cụ nền tảng cho nhiều kỹ thuật xử lý ảnh và phân tích dữ liệu khác, cho thấy tầm quan trọng và tính linh hoạt của nó trong lĩnh vực trí tuệ nhân tạo và các lĩnh vực liên quan.

# **TÀI LIỆU THAM KHẢO**

1. Singular Value Decomposition - Kaggle, truy cập vào tháng 3 19, 2025, <https://www.kaggle.com/code/phamdinhkhanh/singular-value-decomposition>
2. Phân Tích SVD | PDF - Scribd, truy cập vào tháng 3 19, 2025, <https://id.scribd.com/document/723276770/PHAN-TICH-SVD>
3. Phân tích SVD và ứng dụng trong học máy | PDF - Scribd, truy cập vào tháng 3 19, 2025, <https://www.scribd.com/document/792306792/Phan-tich-SVD-va-%E1%BB%A9ng-d%E1%BB%A5ng-trong-h%E1%BB%8Dc-may>
4. Phép nhúng trong máy học là gì? - AWS, truy cập vào tháng 3 19, 2025, <https://aws.amazon.com/vi/what-is/embeddings-in-machine-learning/>
5. Bài 26: Singular Value Decomposition - Machine Learning cơ bản, truy cập vào tháng 3 19, 2025, <https://machinelearningcoban.com/2017/06/07/svd/>
6. SINGULAR VALUE DECOMPOSITION - Tennessee Tech University, truy cập vào tháng 3 19, 2025, <https://www.tntech.edu/cas/pdf/math/techreports/TR-2018-2.pdf>
7. Singular Value Decomposition: Applications to Image Processing, truy cập vào tháng 3 19, 2025, <https://www.lagrange.edu/academics/undergraduate/undergraduate-research/_images/18-Citations2020.Compton.pdf>
8. Differentiate Between Grayscale and RGB Images - GeeksforGeeks, truy cập vào tháng 3 19, 2025, <https://www.geeksforgeeks.org/differentiate-between-grayscale-and-rgb-images/>
9. SVD Image Compression - Frank Cleary, truy cập vào tháng 3 19, 2025, <https://www.frankcleary.com/svdimage/>
10. SVD image reconstruction in Python - Stack Overflow, truy cập vào tháng 3 19, 2025, <https://stackoverflow.com/questions/62832360/svd-image-reconstruction-in-python>