

# AI VIET NAM – COURSE 2023

## Module 2 – Final Project

25th June 2023

**Giới thiệu project:** Đây là project tổng kết của module này. Ở phần project này, đầu tiên các bạn sẽ được ôn tập và hệ thống lại kiến thức về Singular Value Decomposition (SVD). Cũng như cách hiện thực SVD sử dụng thư viện numpy. Sau đó các bạn sẽ được tìm hiểu 2 project thực tế ứng dụng SVD trong lĩnh vực xử lý ảnh: **image compression** và **image denoising**.

**Ôn tập: Singular Value Decomposition**

### 4.1. Singular Value Decomposition

Có ma trận  $\mathbf{A}_{m \times n}$  so sánh m và n

- Nếu  $m > n$ : có nghĩa dim của  $\mathbf{U}$  sẽ lớn hơn  $\mathbf{V}$ . Cần tìm ma trận  $\mathbf{U}$  trước bằng cách tìm eigenvectors (sắp xếp các eigenvector theo singular value tương ứng từ lớn đến bé) của  $\mathbf{A}\mathbf{A}^T$ . Tiếp theo, sắp xếp các singular values (căn bậc 2 eigenvalue của  $\mathbf{A}\mathbf{A}^T$ ) theo thứ tự từ lớn đến bé. Hình thành ma trận đường chéo vuông có kích thước là số lượng singular values khác không, (đường chéo là các singular values vị trí 00, 11, 22, ... các vị trí còn lại = 0). Sau đó padding = 0 để ma trận này có kích thước = ma trận  $\mathbf{A}$ . Thu được  $\Sigma$ . Sau đó tìm ma trận  $\mathbf{V}$  bằng công thức cho từng vector trong  $\mathbf{V}$ :  $\mathbf{v}_i = \frac{1}{\sigma_i} \mathbf{A}^T \mathbf{u}_i$
- Nếu  $m < n$ : có nghĩa dim của  $\mathbf{U}$  sẽ bé hơn  $\mathbf{V}$ . Cần tìm ma trận  $\mathbf{V}$  trước bằng cách tìm eigenvectors (sắp xếp các eigenvector theo singular value tương ứng từ lớn đến bé) của  $\mathbf{A}^T\mathbf{A}$ . Tiếp theo, sắp xếp các singular values (căn bậc 2 eigenvalue của  $\mathbf{A}^T\mathbf{A}$ ) theo thứ tự từ lớn đến bé. Hình thành ma trận đường chéo vuông có kích thước là số lượng singular values khác không, (đường chéo là các singular values vị trí 00, 11, 22, ... các vị trí còn lại = 0). Sau đó padding = 0 để ma trận này có kích thước = ma trận  $\mathbf{A}$ . Thu được  $\Sigma$ . Sau đó tìm ma trận  $\mathbf{U}$  bằng công thức cho từng vector trong  $\mathbf{U}$ :  $\mathbf{u}_i = \frac{1}{\sigma_i} \mathbf{A} \mathbf{v}_i$
- Cách tính determinant của ma trận 3x3

$$\begin{aligned} - \mathbf{C} &= \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \\ - \det(\mathbf{C}) &= a(ei - hf) - b(di - gf) + c(dh - eg) \end{aligned}$$

Dựa vào lược đồ trên, hãy hoàn thiện function **compute\_svd()** sử dụng thư viện numpy:

```
def compute_svd(matrix):
    # *****Your code here *****
    return U,V,V_T
```

Question 1: Kết quả của đoạn code sau đây là gì:

```
matrix = np.array([[7, 1], [0, 0], [5, 5]])
U, S, VT = compute_svd(matrix)
S = np.diag(S)
print(np.matrix.round(S,2))
```

- a) [7.49 3.16]
- b) [[9.49 0. ]  
[0. -3.16]]
- c) [[-9.49 0. ]  
[0. 3.16]]
- d) [[9.49 0. ]  
[0. 3.16]]

### Project 1: Ứng dụng SVD vào bài toán image compression

Cho trước ảnh đầu vào là ảnh màu (hình 1). Hãy phát triển chương trình ứng dụng SVD để compress ảnh đầu vào sử dụng các giá trị k khác nhau trên ảnh grayscale. Lưu ý rằng k chính là rank của ma trận ảnh đầu vào sau khi được xám hoá (hay chính xác là tổng số các phần tử khác không của ma trận  $\Sigma$ , sau khi tính SVD). Bên cạnh đó, bạn cũng cần đánh giá độ sai lệch của ảnh reconstruction ứng với từng k so với ảnh gốc sử dụng 2 metrics là cosine và root mean square error. Hình 2 thể hiện kết quả reconstruction sử dụng các giá trị k khác nhau. Hình 3 thể hiện kết quả cosine similarity của ảnh reconstruction và ảnh gốc với các giá trị k khác nhau. Hình 4 thể hiện kết quả rms error của ảnh reconstruction và ảnh gốc với các giá trị k khác nhau.

**Hướng dẫn:** để hoàn thành bài tập trên, bạn cần hoàn thiện các function sau đây:

```
def main_application():
    color_image = cv2.imread("/content/cat.jpeg")
    color_image = cv2.resize(color_image, (300,300))
    ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
    gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)
    metric_list, approx_list, k =
        evaluate_svd_for_feature_compression_by_metric(gray_image, 1)
    print(len(approx_list))
    draw_metric_information(metric_list, k)
    draw_approx_image(approx_list, k)

main_application()
```

---

Hoàn thiện function **convertColorToGrayScale** để chuyển đổi ảnh màu thành ảnh xám (không dùng thư viện cv2.cvtColor()).

```
def convertColorToGrayScale(image, ratio):
    # ***** your code here *****
    return img_gray
```

Question 2: Kết quả của đoạn code sau đây là gì:

```
color_image = cv2.imread("/content/cat.jpeg")
color_image = cv2.resize(color_image, (300,300))
ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)
print(np.sum(gray_image))
```

- a) 6015657
- b) 7015657
- c) 8015657
- d) 9015657

Hoàn thiện function **compute\_cosine\_similarity** để tìm cosine similarity giữa 2 ma trận:

```
def compute_cosine_similarity(matrix1, matrix2):
    #***** Your code here *****
    return result
```

Question 3: Kết quả của đoạn code sau đây là gì:

```
A = np.array([[1,2,2],
             [3,2,2],
             [-2,1,-3]])
B = np.array([[4,2,4],
             [2,-2,5],
             [3,4,-4]])
result = compute_cosine_similarity(A,B)
print(round(result,2))
```

- a) 1.57
- b) 0.57
- c) 2.57

d) 3.57

Hoàn thiện function **compute\_rms\_error** để tìm rms error giữa 2 ma trận:

```
def compute_rms_error(matrix1, matrix2):
    ***** Your code here *****
    return result
```

**Question 4:** Kết quả của đoạn code sau đây là gì:

```
A = np.array([[1,2,2],
             [3,2,2],
             [-2,1,-3]])
B = np.array([[4,2,4],
              [2,-2,5],
              [3,4,-4]])
result = compute_rms_error(A,B)
print(round(result,2))
```

- a) 1.44
- b) 2.87
- c) 3.44
- d) 5.44

Hoàn thiện function bên dưới để tìm cosine similarity (hoặc rms error) giữa kết quả reconstruction và ảnh gốc tương ứng với từng giá trị k. Lưu ý kết quả trả về là danh sách cosine similarity (hoặc rms error), danh sách ảnh reconstruction, và k.

```
def evaluate_svd_for_feature_compression_by_metric(gray_image, metric):
    U,S,V_T = compute_svd(gray_image)
    rank_k = len(S)
    S = np.diag(S)

    approx_list = []
    metric_list = []
    for i in range(1,k+1,1):
        # compute reconstruction image by using U, S, and V_T
        approx = None
        # ***** Your code here *****
        approx_list.append(approx)
        if metric == 0:
            metric_list.append(compute_cosine_similarity(gray_image, approx))
        else:
            metric_list.append(compute_rms_error(gray_image, approx))

    return metric_list, approx_list, k
```

**Question 5:** Hãy chọn kết quả đúng của đoạn code sau đây:

```
color_image = cv2.imread("/content/cat.jpeg")
color_image = cv2.resize(color_image, (300,300))
ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)
_, _, k = evaluate_svd_for_feature_compression_by_metric(gray_image,0)
print(k)
```

- a) 400
- b) 200
- c) 300
- d) 100

Cuối cùng bạn có thể sử dụng function **draw\_metric\_information** để vẽ đồ thị theo metric và function **draw\_approx\_image** để visualize ảnh reconstruction tương ứng với K.

```
def draw_metric_information(metric_list, k):
    x_axis = list(range(1,k+1,1))
    plt.xlabel("k")
    plt.ylabel("RMS error")
    plt.plot(x_axis,metric_list, color="r")
    plt.show()

def draw_approx_image(approx_list, k):
    row = 6
    col = 4
    fig = plt.figure(figsize=(25, 25))

    fig_index = 1

    for i in range(0, k, 13):
        # Adds a subplot at the 1st position
        fig.add_subplot(row, col, fig_index)
        # showing image
        plt.imshow(approx_list[i], cmap="gray")

        plt.axis('off')
        plt.title("k = " + str(i+1))
        fig_index = fig_index + 1

    plt.show()
```

**Question 6:** Cho trước ảnh đầu vào có độ phân giải là (960, 1440). Giả sử mỗi phần tử được lưu bởi một số thực 4 byte. Hãy cho biết bạn đã tiếp kiệm được bao nhiêu % bộ nhớ, với k = 100

- a) 30.5%
- b) 40.5%
- c) 50.5%
- d) 60.5%

**Project 2: Ứng dụng SVD vào bài toán Image Denoising** Ở project này các bạn sẽ được tìm hiểu cách ứng dụng SVD vào giải quyết bài toán image denosing. Cho trước ảnh đầu vào có kích thước (2392, 2500), hãy phát triển chương trình sử dụng SVD để giải quyết bài toán image denosing (giảm nhiễu trên ảnh đầu vào). Hình 5 thể hiện các bước chính giải thuật mà các bạn cần hiện thực để giải quyết bài toán image denoising.

**Hướng dẫn:** để hoàn thành bài tập trên, bạn cần hoàn thiện các function sau đây:

```
def main_application():
    #Read an color image
    color_image = cv2.imread("/content/cat-551554_640.jpg")

    #Resize an image to (300,300)
    color_image = cv2.resize(color_image, (300,300))

    #Convert image to grayscale
    ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
    gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)

    #Save resized grayscale image to file gray_image.png
    cv2.imwrite("gray_image.png", gray_image)

    # Add Gaussian noise to gray image
    noise_gray_image = addNoiseToImage(gray_image, 0, 25)

    #Save noisy grayscale image to file noise_gray_image.png
    cv2.imwrite("noise_gray_image.png", noise_gray_image)

    #Perform image denois
    sigma_threshold = 300
    denoise_image = perform_image_denoise(noise_gray_image, sigma_threshold)

    cv2.imwrite("denoise_image.png", denoise_image)
```

Hoàn thiện function **addNoiseToImage()** để thêm nhiễu vào ảnh đầu vào grayscale:

```
# Add Gaussian noise to input image
def addNoiseToImage(image, mean = 0, stddev= 25):

    #uncomment the following three lines of codes for another project
    """noise = np.random.normal(mean, stddev, image.shape)
```

```

noise = np.where(noise > 0 , np.around(noise), 0)
noise = noise.astype(np.uint8)
np.save("noise.npy", noise)"""

noise = np.load("noise.npy")
# Add noise to image
noisy_img = cv2.add(image, noise)

return noisy_img

```

Question 7: Kết quả của đoạn code sau đây là gì:

```

from google.colab.patches import cv2_imshow
color_image = cv2.imread("/content/cat.jpeg")
color_image = cv2.resize(color_image, (300,300))
ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)

noise_image = addNoiseToImage(gray_image)
avg_org = np.sum(gray_image)/(gray_image.shape[0]*gray_image.shape[1])
avg_noise = np.sum(noise_image)/(gray_image.shape[0]*gray_image.shape[1])

print("avg_org: ", round(avg_org,2))
print("avg_noise: ", round(avg_noise,2))

```

- a) avg\_org: 96.9  
avg\_noise: 76.8
- b) avg\_org: 66.84  
avg\_noise: 76.73
- c) avg\_org: 86.9  
avg\_noise: 76.8
- d) avg\_org: 76.9  
avg\_noise: 76.8

Hoàn thiện function **perform\_image\_denoise()** sử dụng các tính chất của SVD để giảm nhiễu ảnh đầu vào grayscale đã thêm nhiễu ở bên trên.

```

def perform_image_denoise(noise_image, sigma_threshold):

    # Calculate U (u), (s) and V (vh)
    u, s, vh = # your code here ****

    # Remove sigma values below sigma_threshold
    s_cleaned = # your code here ****

    img_denoised = # your code here ****

```

```
    return img_denoised
```

Question 8: Kết quả của đoạn code sau đây là gì:

```
#Read an colori mage
color_image = cv2.imread("/content/cat.jpeg")
color_image = cv2.resize(color_image, (300,300))
ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)
noise_gray_image = addNoiseToImage(gray_image, 0, 25)

#Perform image denois
sigma_threshold = 200
denoise_image = perform_image_denoise(noise_gray_image, sigma_threshold)
cv2_imshow(denoise_image)
similarity = compute_rms_error(denoise_image,gray_image)
print("rms", round(similarity,2))
```

- a) 16.85
- b) 17.85
- c) 18.85
- d) 16.98

Question 9: Kết quả của đoạn code sau đây là gì:

```
#Read an colori mage
color_image = cv2.imread("/content/cat.jpeg")
color_image = cv2.resize(color_image, (300,300))
ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)
noise_gray_image = addNoiseToImage(gray_image, 0, 25)

#Perform image denois
sigma_threshold = 200
denoise_image = perform_image_denoise(noise_gray_image, sigma_threshold)
cv2_imshow(denoise_image)
similarity = compute_cosine_similarity(denoise_image,gray_image)
print("similarity", round(similarity,4))
```

- a) 0.7899
- b) 0.8899

c) 0.9898

d) 1.0

Hình 6 hiển thị kết quả của giải thuật denoise sử dụng SVD với các sigma threshold khác nhau. Từ kết quả hình 6 chúng ta thấy rằng cần phải phát triển giải thuật tìm thông số tối ưu cho sigma threshold. Do đó các bạn cần tiếp tục hoàn thiện function `find_best_sigma_image_denoise()`, dựa vào thước đo theo RMSE hoặc Cosine similarity. Tham số đầu vào của function này cần: noise\_image (ảnh đã thêm nhiễu), orignal\_image (ảnh grayscale gốc), metric (rmse hoặc cosine), sigma\_range (giá trị tối đa để loại bỏ sigma). Kết quả đầu ra, các bạn trả về sigma có threshold tốt nhất, và lịch sử đánh giá.

```
def find_best_sigma_image_denoise(noise_image, orignal_image, metric = 0, sigma_range = 1000):

    # Calculate U (u), (s) and V (vh)
    u, s, vh = # your code here *****

    # Select the best threshold that has mininum RMS error
    evaluation_history = []
    for i in range(1, sigma_range):

        # Remove sigma values below sigma_threshold
        s_cleaned = # your code here *****

        img_denoised = # your code here *****

        if (metric == 0):
            similarity = compute_cosine_similarity(orignal_image, img_denoised)
            evaluation_history.append(similarity)
        else:
            error = compute_rms_error(orignal_image, img_denoised)
            evaluation_history.append(error)

        if metric == 0:
            best_index = # your code here *****

        else:
            best_index = # your code here *****

    return best_index, evaluation_history
```

Question 10: Kết quả của đoạn code sau đây là gì nếu chúng ta tìm thông số tối ưu theo cosine:

```
def find_best_sigma_based_cosine():
    #Read an color image
    color_image = cv2.imread("/content/cat.jpeg")

    #Resize an image to (300,300)
    color_image = cv2.resize(color_image, (300,300))
```

```
#Convert image to grayscale
ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)

#Save resized grayscale image to file gray_image.png
cv2.imwrite("gray_image.png", gray_image)

# Add Gaussian noise to gray image
noise_gray_image = addNoiseToImage(gray_image, 0, 25)

#Save noisy grayscale image to file noise_gray_image.png
cv2.imwrite("noise_gray_image.png", noise_gray_image)

metric = 0 # for cosine similarity
best_threshold, evaluation_history = find_best_sigma_image_denoise(noise_gray_image,
    gray_image, metric)
print("best sigma: ", best_threshold)

find_best_sigma_based_cosine()
```

- a) best sigma: 503
- b) best sigma: 511
- c) best sigma: 513
- d) best sigma: 524

Question 11: Kết quả của đoạn code sau đây là gì nếu chúng ta tìm thông số tối ưu theo rmse:

```
def find_best_sigma_based_cosine():
    #Read an color image
    color_image = cv2.imread("/content/cat.jpeg")

    #Resize an image to (300,300)
    color_image = cv2.resize(color_image, (300,300))

    #Convert image to grayscale
    ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
    gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)

    #Save resized grayscale image to file gray_image.png
    cv2.imwrite("gray_image.png", gray_image)

    # Add Gaussian noise to gray image
    noise_gray_image = addNoiseToImage(gray_image, 0, 25)

    #Save noisy grayscale image to file noise_gray_image.png
    cv2.imwrite("noise_gray_image.png", noise_gray_image)
```

```

metric = 1 # for RMSE
best_threshold, evaluation_history = find_best_sigma_image_denoise(noise_gray_image,
    gray_image, metric)
print("best sigma: ", best_threshold)

find_best_sigma_based_cosine()

```

- a) best sigma: 525
- b) best sigma: 511
- c) best sigma: 513
- d) best sigma: 524

Các bạn có thể sử dụng đoạn code bên dưới để trực quan hóa quá trình tìm sigma threshold tối ưu theo metric Cosine (hình 7) như sau:

```

def plot_chart_based_rmse():
    #Read an color image
    color_image = cv2.imread("/content/cat.jpeg")

    #Resize an image to (300,300)
    color_image = cv2.resize(color_image, (300,300))

    #Convert image to grayscale
    ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
    gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)

    #Save resized grayscale image to file gray_image.png
    cv2.imwrite("gray_image.png", gray_image)

    # Add Gaussian noise to gray image
    noise_gray_image = addNoiseToImage(gray_image, 0, 25)

    #Save noisy grayscale image to file noise_gray_image.png
    cv2.imwrite("noise_gray_image.png", noise_gray_image)

    metric = 1 # for cosine similarity
    best_threshold, evaluation_history = find_best_sigma_image_denoise(noise_gray_image,
        gray_image, metric)

    plt.plot(evaluation_history)
    plt.xlabel("Sigma Threshold")
    plt.ylabel("Cosine Similarity")
    plt.show()

plot_chart_based_cosine()

```

Các bạn có thể sử dụng đoạn code bên dưới để trực quan hóa quá trình tìm sigma threshold tối

ưu theo metric RMSE (hình 8) như sau:

```
def plot_chart_based_rmse():
    #Read an color image
    color_image = cv2.imread("/content/cat.jpeg")

    #Resize an image to (300,300)
    color_image = cv2.resize(color_image, (300,300))

    #Convert image to grayscale
    ratio_for_grayscale = [0.2126, 0.7152, 0.0722]
    gray_image = convertColorToGrayScale(color_image, ratio_for_grayscale)

    #Save resized grayscale image to file gray_image.png
    cv2.imwrite("gray_image.png", gray_image)

    # Add Gaussian noise to gray image
    noise_gray_image = addNoiseToImage(gray_image, 0, 25)

    #Save noisy grayscale image to file noise_gray_image.png
    cv2.imwrite("noise_gray_image.png", noise_gray_image)

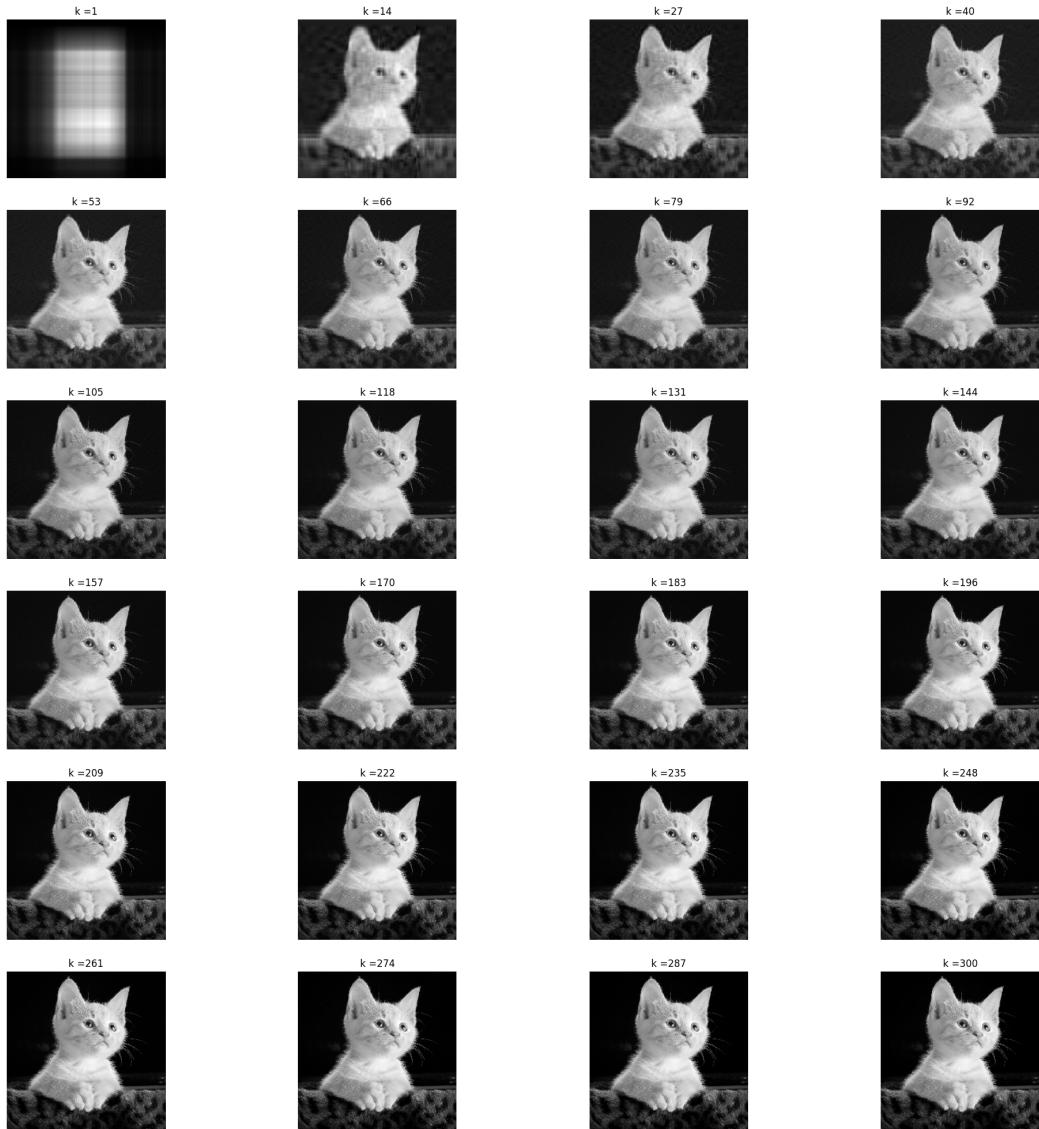
    metric = 1 # for cosine similarity
    best_threshold, evaluation_history = find_best_sigma_image_denoise(noise_gray_image,
        gray_image, metric)

    plt.plot(evaluation_history)
    plt.xlabel("Sigma Threshold")
    plt.ylabel("RMSE")
    plt.show()

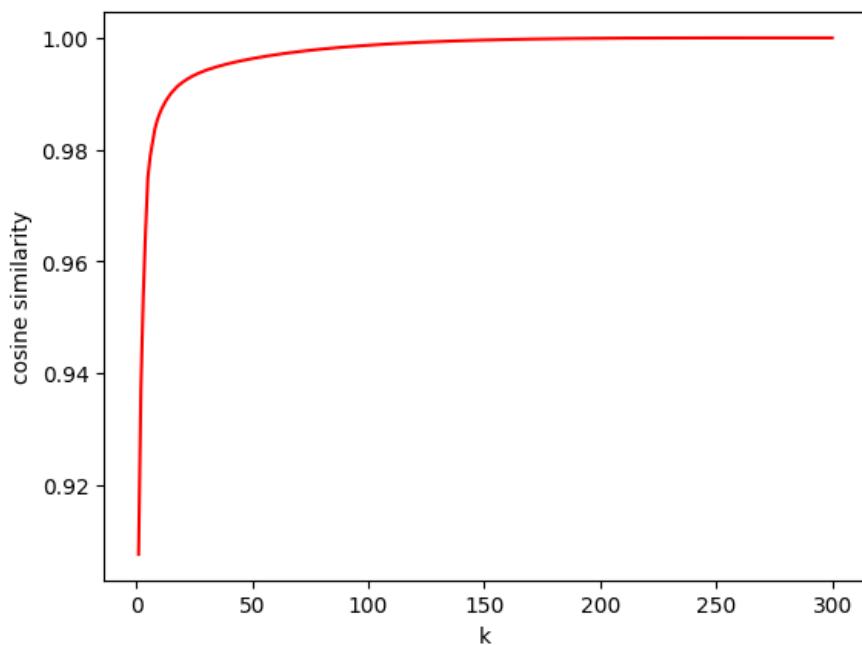
plot_chart_based_rmse()
```



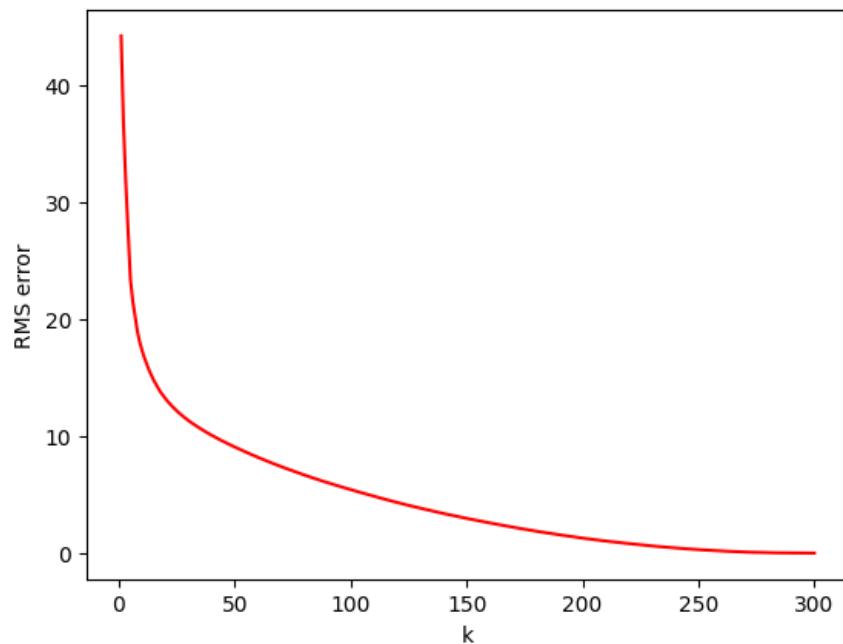
Hình 1: Ảnh đầu vào



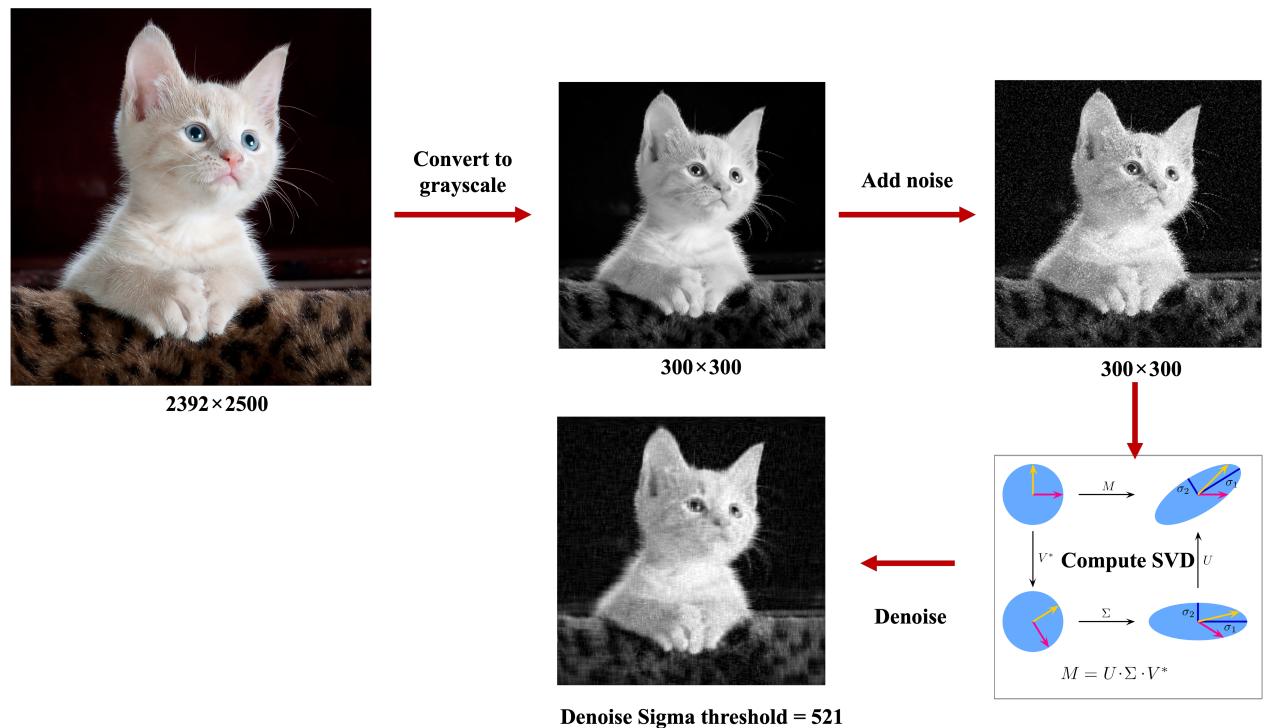
Hình 2: Kết quả visualization của ảnh reconstruction ứng với các giá trị  $k$  khác nhau



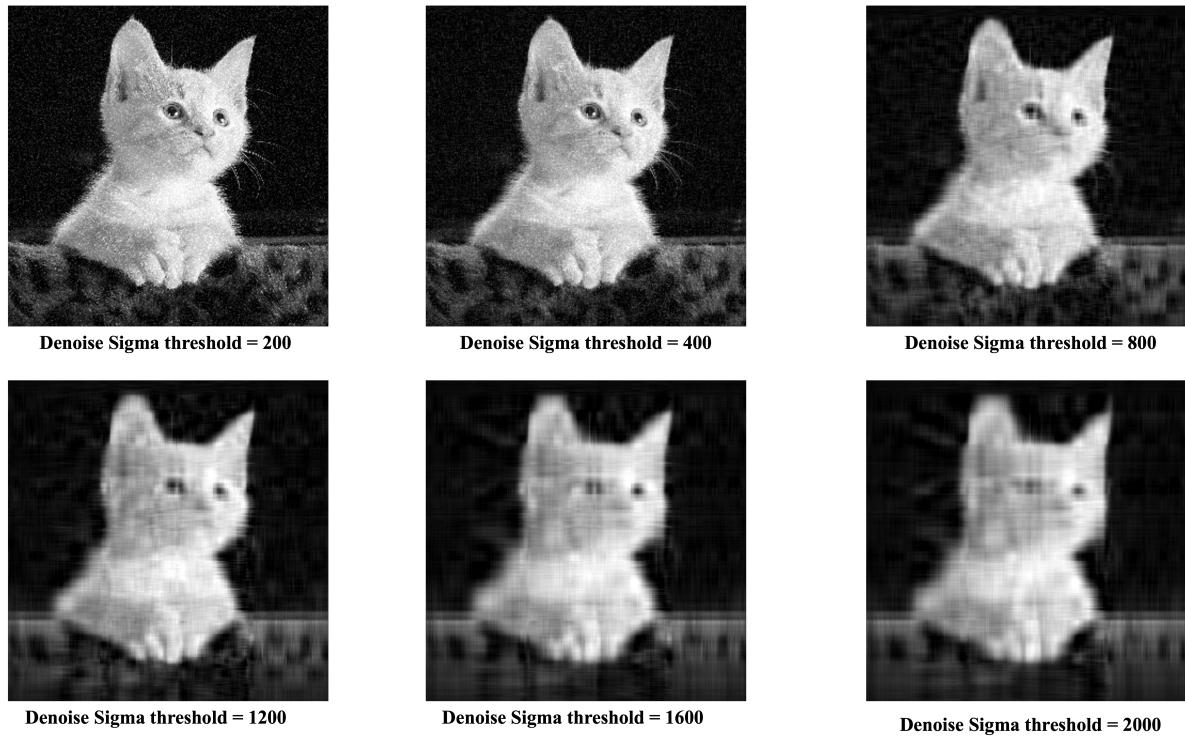
Hình 3: Consine similarity của ảnh reconstruction và ảnh gốc với các giá trị  $k$  khác nhau



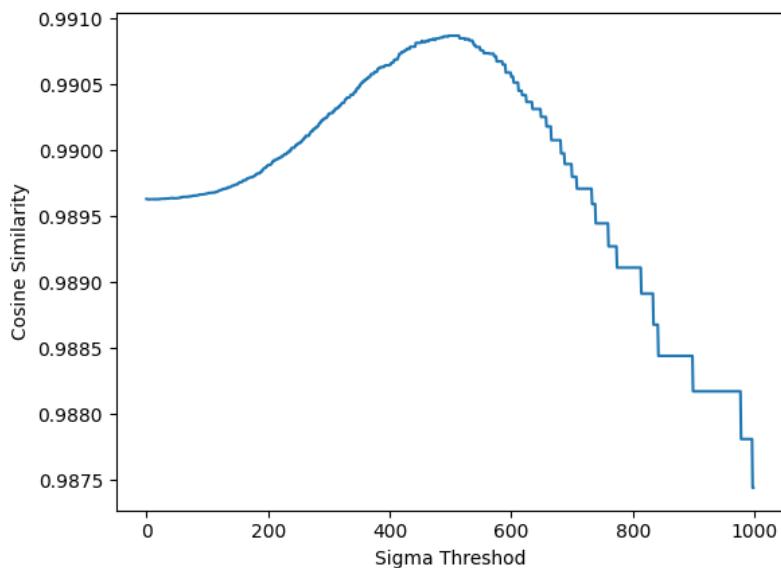
Hình 4: RMS error của ảnh reconstruction và ảnh gốc với các giá trị  $k$  khác nhau



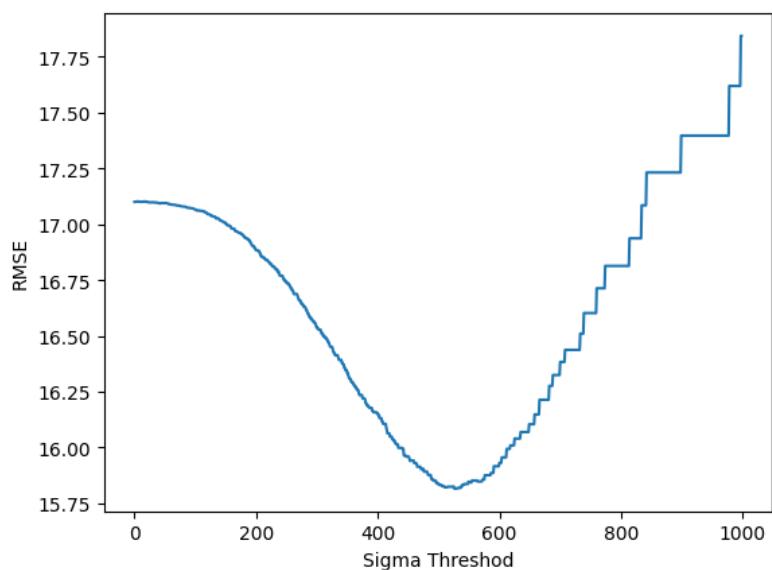
Hình 5: Flowchart ứng dụng SVD cho bài toán image denosing.



Hình 6: Kết quả image denoising bằng cách loại bỏ các giá sigma nhỏ hơn threshold cho trước



Hình 7: Kết quả Cosine similarity ứng với các giá trị sigma khác nhau



Hình 8: Kết quả RMSE ứng với các giá trị sigma khác nhau