AI VIETNAM
AI Insight Course

Image Retrieval Using Pretrained Models

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2020

Outline

- Vector and Matrix
- Cosine Similarity
- Implementation
- Case Studies

Vector & Matrix

Vector	Matrix
n is a natural number	Matrix A has the shape of rectangle
$\mathcal{R}$ is a set of real numbers	Has m rows and n columns
$\vec{v}$ has a length of n and contain real numbers $\vec{v} \in \mathcal{R}^n$	Use capital letter $A \in \mathcal{R}^{m \times n}$
$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \in \begin{bmatrix} \mathcal{R} \\ \mathcal{R} \\ \mathcal{R} \end{bmatrix} = \mathcal{R}^3$	$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \in \begin{bmatrix} \mathcal{R} & \mathcal{R} \\ \mathcal{R} & \mathcal{R} \\ \mathcal{R} & \mathcal{R} \end{bmatrix} = \mathcal{R}^{3 \times 2}$

Vector Operations

Addition

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_3 \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_3 \end{bmatrix}$$

$$\vec{v} + \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_3 \end{bmatrix} + \begin{bmatrix} u_1 \\ \dots \\ u_3 \end{bmatrix} = \begin{bmatrix} v_1 + u_1 \\ \dots \\ v_3 + u_3 \end{bmatrix}$$

```python
def add_vectors(vector1, vector2):
    '''
    Add corrosponding elements between two vectors
    vector1 and vector2 are with list type
    output is a vector (list)
    '''

    return [v1+v2 for v1, v2 in zip(vector1, vector2)]

# Test case
vector1 = [1, 2, 3]
vector2 = [4, 5, 6]

output = add_vectors(vector1, vector2)
print(output)
```

```
[5, 7, 9]
```

Vector Operations

Subtraction

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} + \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} - \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 - u_1 \\ \dots \\ v_n - u_n \end{bmatrix}$$

Multiply with a number

$$\alpha \vec{u} = \alpha \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} \alpha u_1 \\ \dots \\ \alpha u_n \end{bmatrix}$$

Length of a vector

$$||\vec{u}|| = \sqrt{u_1^2 + \dots + u_n^2}$$

Vector Operations

Dot product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} \cdot \vec{u} = v_1 \times u_1 + \cdots + v_n \times u_n$$

```
def dot_product(vector1, vector2):
    '''
    Compute dot product between two vectors
    Output is a floating-point number
    '''

    return sum([v1*v2 for v1, v2 in zip(vector1, vector2)])

# test case
vector1 = [1, 2, 3]
vector2 = [2, 3, 4]

ouptut = dot_product(vector1, vector2)
print(ouptut)
```

```
20
```

Vector Operations

Hadamard product

$$\vec{v} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \qquad \vec{u} = \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix}$$

$$\vec{v} \odot \vec{u} = \begin{bmatrix} v_1 \\ \dots \\ v_n \end{bmatrix} \odot \begin{bmatrix} u_1 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} v_1 \times u_1 \\ \dots \\ v_n \times u_n \end{bmatrix}$$

```python
def Hadamard_product(vector1, vector2):
    '''
    Compute Hadamard product between two vectors
    Output is a vector
    '''
    return [v1*v2 for v1, v2 in zip(vector1, vector2)]

# test case
vector1 = [1, 2]
vector2 = [3, 4]

output = Hadamard_product(vector1, vector2)
print(output)
```

```
[3, 8]
```

Matrix Operations

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1n} \\ \dots & \dots & \dots \\ b_{m1} & \dots & b_{mn} \end{bmatrix}$$

Addition

$$A + B = \begin{bmatrix} (a_{11} + b_{11}) & \dots & (a_{1n} + b_{1n}) \\ \dots & \dots & \dots \\ (a_{m1} + b_{m1}) & \dots & (a_{mn} + b_{mn}) \end{bmatrix}$$

Subtraction

$$A - B = \begin{bmatrix} (a_{11} - b_{11}) & \dots & (a_{1n} - b_{1n}) \\ \dots & \dots & \dots \\ (a_{m1} - b_{m1}) & \dots & (a_{mn} - b_{mn}) \end{bmatrix}$$

Matrix Operations

Multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}$$

$A \in \mathcal{R}^{m\times n}$ $\quad$ $B \in \mathcal{R}^{n\times k}$

$$C = AB$$

$$C \in \mathcal{R}^{m\times k}$$

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}$$

$$\begin{bmatrix} \boxed{a_{11} \quad a_{12}} \\ a_{21} \quad a_{22} \\ a_{31} \quad a_{32} \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} b_{11} \\ b_{21} \end{matrix}} & \begin{matrix} b_{12} \\ b_{22} \end{matrix} \end{bmatrix} = \begin{bmatrix} \boxed{a_{11}b_{11} + a_{12}b_{21}} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

Matrix Operations

Multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \qquad \vec{x} = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}$$

$$A \in \mathcal{R}^{m \times n}$$

$$C = A\vec{x}$$

$$c_i = \sum_{l=1}^{n} a_{il} x_l$$

Example

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \\ a_{31}x_1 + a_{32}x_2 \end{bmatrix}$$

Matrix Operations

Multiplication

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}$$

$A \in \mathcal{R}^{m \times n}$ $\quad$ $B \in \mathcal{R}^{n \times k}$

$$C = AB$$

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}$$

```python
def matrix_multiplication(matrix1, matrix2):
    '''
    This function does the multiplication between two matrices.
    #columns of matrix1 == #rows of matrix2
    '''
    matrix1_nrows = len(matrix1)
    matrix1_ncols = len(matrix1[0])

    matrix2_nrows = len(matrix2)
    matrix2_ncols = len(matrix2[0])

    # tạo matrix kết quả
    result = [[0]*matrix2_ncols for i in range(matrix1_nrows)]

    for i in range(matrix1_nrows):
        for j in range(matrix2_ncols):
            for k in range(matrix2_nrows):
                result[i][j] += matrix1[i][k] * matrix2[k][j]

    return result

# test case
# 3x3 matrix
matrix1 = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]

# 3x4 matrix
matrix2 = [[1, 1, 2, 1],
           [1, 2, 1, 1],
           [1, 1, 1, 2]]

result = matrix_multiplication(matrix1, matrix2)
print(result[0])
print(result[1])
print(result[2])
```

```
[6, 8, 7, 9]
[15, 20, 19, 21]
[24, 32, 31, 33]
```

Matrix Operations

Transpose

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

$$A^T = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \dots & \dots & \dots \\ a_{1n} & \dots & a_{mn} \end{bmatrix}$$

Example

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \end{bmatrix}^T = \begin{bmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \end{bmatrix}$$

Applications

Phép nhân giữa ma trận và vector

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$A\boldsymbol{x} = b$$

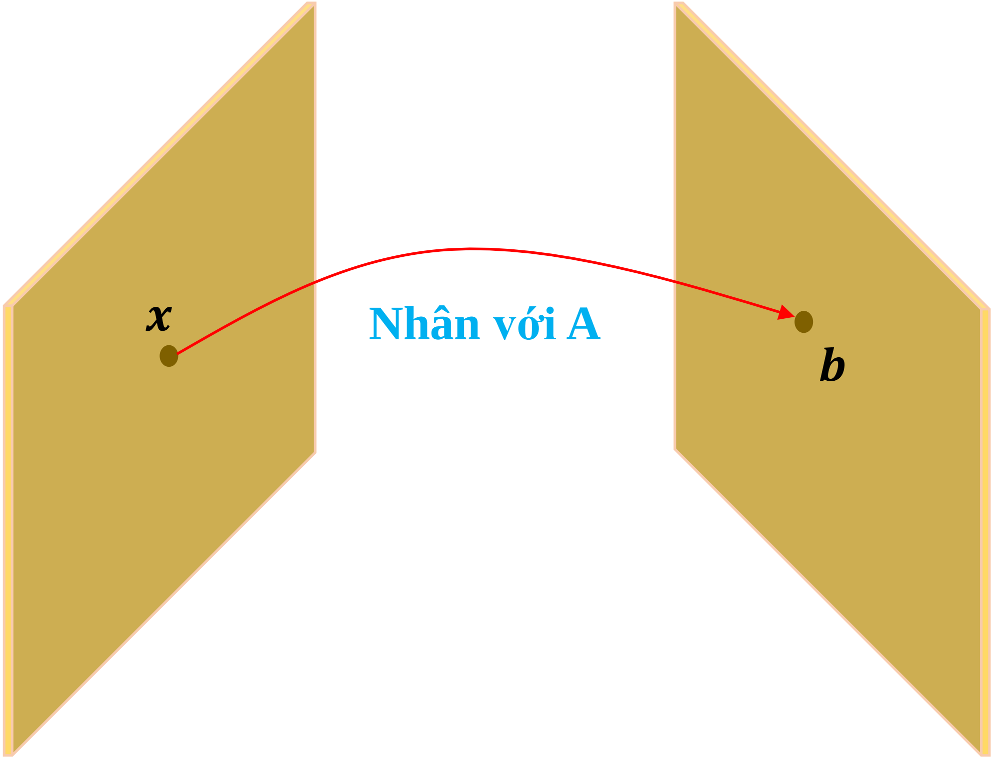
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Ma trận A biến đổi/dịch chuyển x sang b

Giá trị từng phần tử của b là tổ hợp tuyến tính của tất cả các phần tử của x

$$b_1 = a_{11}x_1 + a_{12}x_2$$

$$b_2 = a_{21}x_1 + a_{22}x_2$$



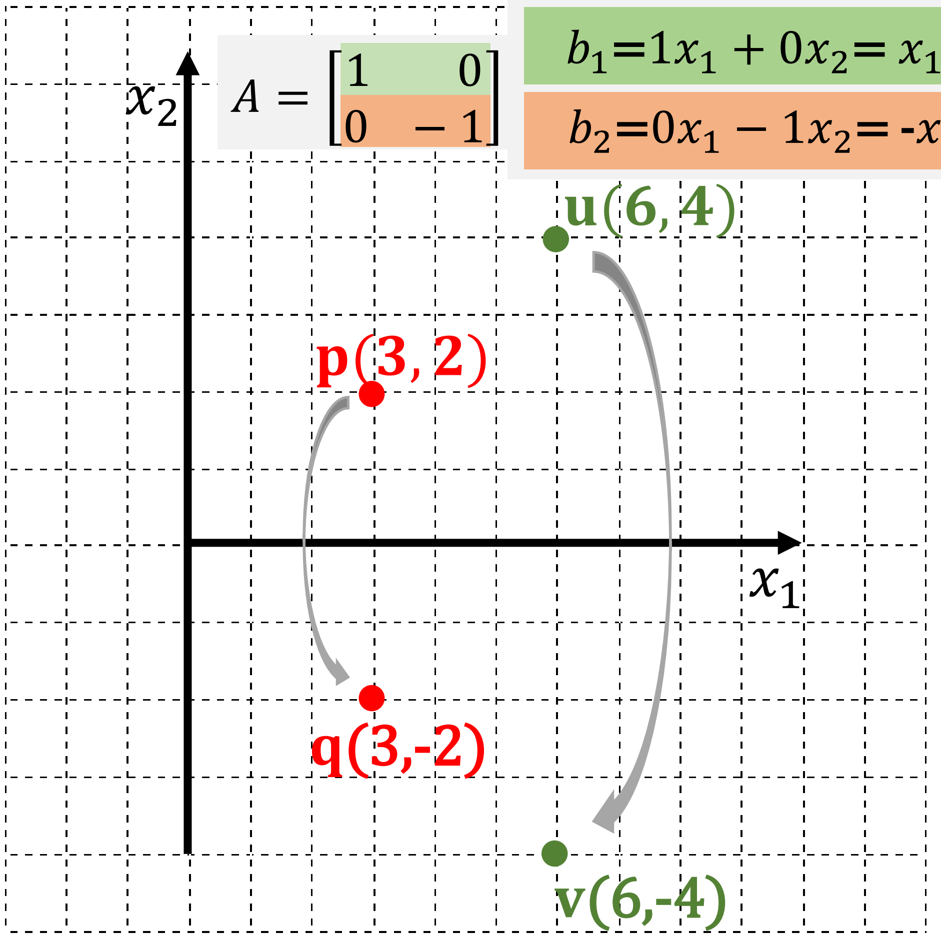
Applications

Ma trận A dịch chuyển điểm x đối xứng qua trục x_1

$$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$b_1 = 1x_1 + 0x_2 = x_1$$

$$b_2 = 0x_1 - 1x_2 = -x_2$$



Ứng dụng lật ảnh đối xứng qua trục ngang

Các thuộc tính của pixel $\mathbf{p}(\underbrace{x_1, x_2}_{\text{Tọa độ điểm } \mathbf{p}}, \overbrace{r, g, b}^{\text{Giá trị màu (red, green, blue) của điểm } \mathbf{p}})$

Dịch chuyển pixel (x_1, x_2) **theo** $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Ảnh gốc

Ảnh kết quả

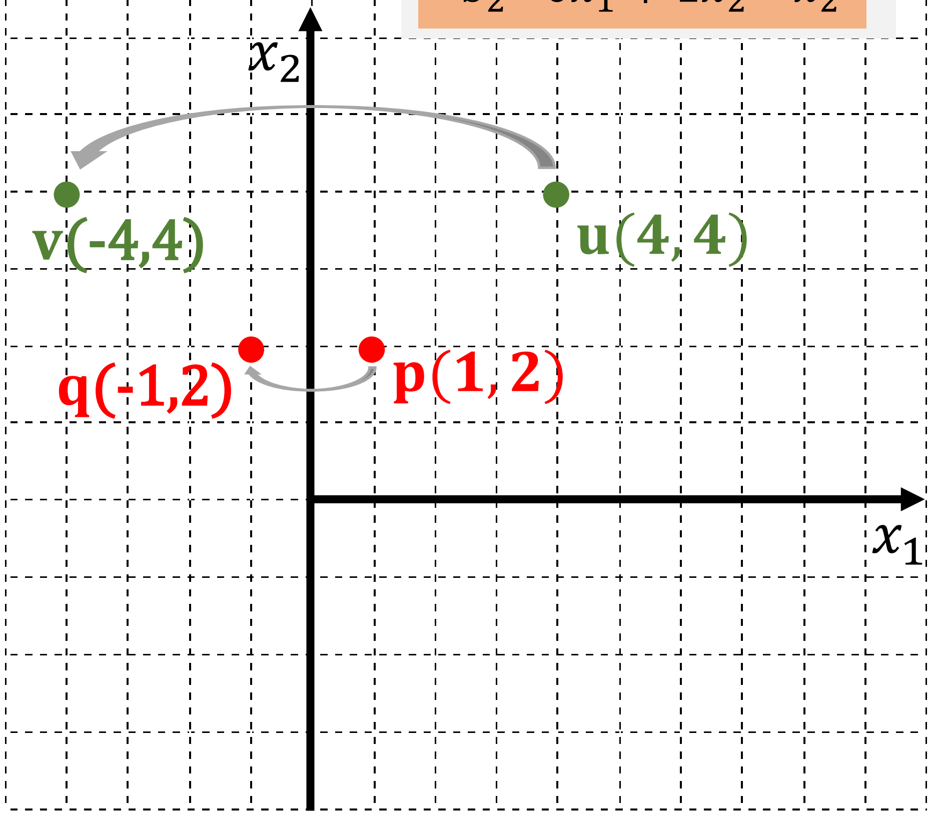
Applications

Ma trận A dịch chuyển điểm x đối xứng qua trục x_2

$$A = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$b_1 = -1x_1 + 0x_2 = -x_1$$

$$b_2 = 0x_1 + 1x_2 = x_2$$



Ứng dụng lật ảnh đối xứng qua trục đứng

Giá trị màu (red, green, blue) của điểm **p**

Các thuộc tính của pixel $\mathbf{p}(\underbrace{x_1, x_2}_{\text{Tọa độ điểm } \mathbf{p}}, \overbrace{\text{r, g, b}})$

Dịch chuyển pixel (x_1, x_2) **theo** $A = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Ảnh gốc

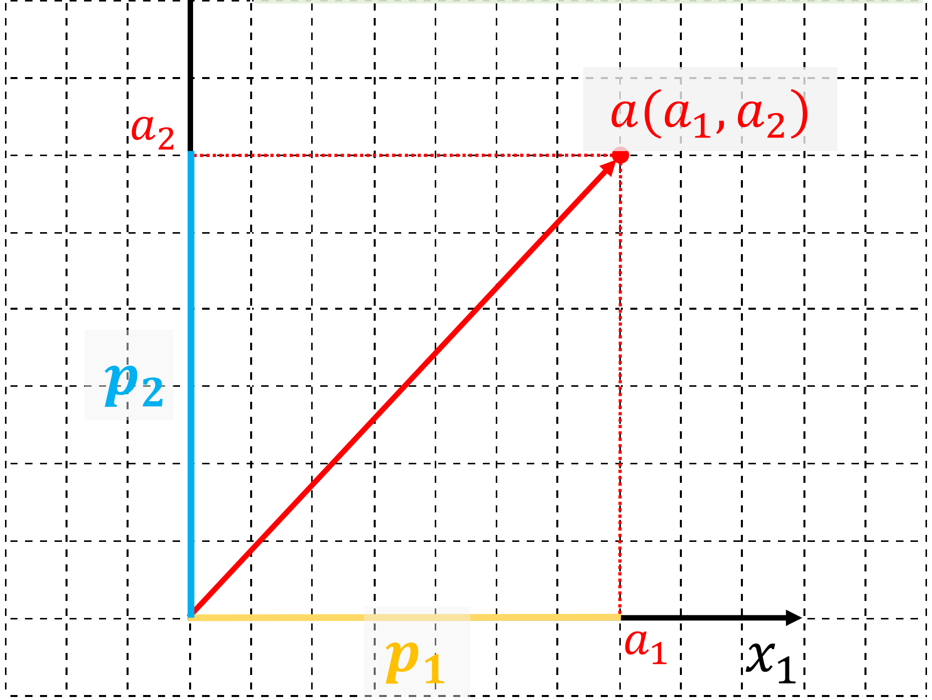
Ảnh kết quả

Dot Product

$$\overrightarrow{x_1} = \begin{pmatrix}1\\0\end{pmatrix} \quad \overrightarrow{x_2} = \begin{pmatrix}0\\1\end{pmatrix}$$

$$\mathrm{p}_1 = \vec{\mathrm{a}}.\overrightarrow{\mathrm{x}_1} = (\mathrm{a}_1 \ \ \mathrm{a}_2)^{\mathrm{T}}\begin{pmatrix}1\\0\end{pmatrix} = \mathrm{a}_1$$

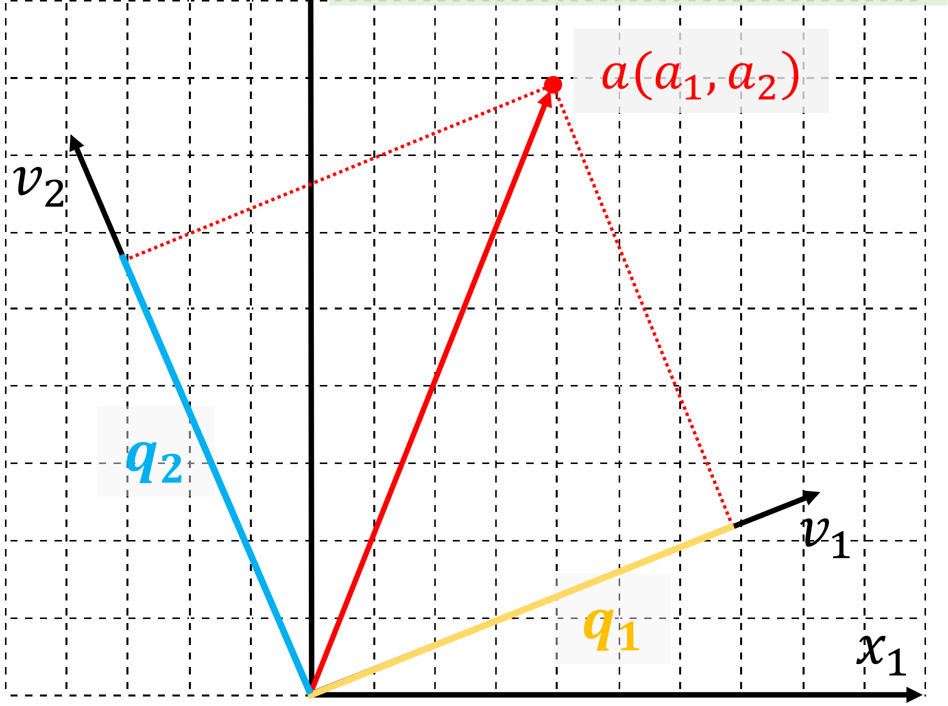
$$\mathrm{p}_1 = \vec{\mathrm{a}}.\overrightarrow{\mathrm{x}_2} = (\mathrm{a}_1 \ \ \mathrm{a}_2)^{\mathrm{T}}\begin{pmatrix}0\\1\end{pmatrix} = \mathrm{a}_2$$



$$\overrightarrow{v_1} = \begin{pmatrix}v_{1\_1}\\v_{1\_2}\end{pmatrix} \quad \overrightarrow{v_2} = \begin{pmatrix}v_{2\_1}\\v_{2\_2}\end{pmatrix}$$

$$\mathrm{q}_1 = \vec{\mathrm{a}}.\overrightarrow{\mathrm{v}_1} = (\mathrm{a}_1 \ \ \mathrm{a}_2)^{\mathrm{T}}\begin{pmatrix}v_{1\_1}\\v_{1\_2}\end{pmatrix}$$

$$\mathrm{q}_1 = \vec{\mathrm{a}}.\overrightarrow{\mathrm{v}_2} = (\mathrm{a}_1 \ \ \mathrm{a}_2)^{\mathrm{T}}\begin{pmatrix}v_{2\_1}\\v_{2\_2}\end{pmatrix}$$



Tìm độ dài hình chiếu của $\vec{a}$ lên $\overrightarrow{x_1}$ và $\overrightarrow{x_2}$

Tìm độ dài hình chiếu của $\vec{a}$ lên $\overrightarrow{v_1}$ và $\overrightarrow{v_2}$

Tách ma trận

Mục đích: Đưa ma trận Q về dạng UΣ sao cho các vector (cột) trong U có độ dài bằng 1.

Công thức

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} \dfrac{a}{\sqrt{a^2+b^2}} & \dfrac{c}{\sqrt{c^2+d^2}} \\ \dfrac{b}{\sqrt{a^2+b^2}} & \dfrac{d}{\sqrt{c^2+d^2}} \end{pmatrix} \begin{pmatrix} \sqrt{a^2+b^2} & 0 \\ 0 & \sqrt{c^2+d^2} \end{pmatrix}$$

Ví dụ

$$\begin{pmatrix} 3 & 2 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} \dfrac{3}{\sqrt{3^2+4^2}} & \dfrac{2}{\sqrt{2^2+0}} \\ \dfrac{4}{\sqrt{3^2+4^2}} & 0 \end{pmatrix} \begin{pmatrix} \sqrt{3^2+4^2} & 0 \\ 0 & \sqrt{2^2+0} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{3}{5} & 1 \\ \frac{4}{5} & 0 \end{pmatrix} \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix}$$

Cách tính tương tự khi Q có kích thước khác

Singular Value Decomposition

Tìm độ dài hình chiếu của $\vec{a}$ lên $\overrightarrow{v_1}$ và $\overrightarrow{v_2}$

$$\mathrm{q}_1 = \vec{\mathrm{a}}.\overrightarrow{\mathrm{v}_1} = (\mathrm{a}_1 \ \ \mathrm{a}_2)^\mathrm{T}\begin{pmatrix} v_{1\_1} \\ v_{1\_2} \end{pmatrix}$$

$$\mathrm{q}_1 = \vec{\mathrm{a}}.\overrightarrow{\mathrm{v}_2} = (\mathrm{a}_1 \ \ \mathrm{a}_2)^\mathrm{T}\begin{pmatrix} v_{2\_1} \\ v_{2\_2} \end{pmatrix}$$

Viết lại dạng ma trận cho ngắn gọn

$$\vec{\mathrm{a}}.\mathrm{V} = (a_1 \ \ a_2)^T\begin{pmatrix} v_{1\_1} & v_{2\_1} \\ v_{1\_2} & v_{2\_2} \end{pmatrix}$$

$$= (\mathrm{q}_1 \ \ \mathrm{q}_2)$$

Nếu có thêm $\vec{\mathrm{b}}$

$$\underset{A}{\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}^T} \ \underset{V}{\begin{pmatrix} v_{1\_1} & v_{2\_1} \\ v_{1\_2} & v_{2\_2} \end{pmatrix}} = \underset{Q}{\begin{pmatrix} \mathrm{q}_{1\mathrm{a}} & \mathrm{q}_{2\mathrm{a}} \\ \mathrm{q}_{1\mathrm{b}} & \mathrm{q}_{2\mathrm{b}} \end{pmatrix}}$$

Ma trận các điểm — Ma trận các trục — Ma trận độ dài hình chiếu

Giả sử V là ma trận trực giao

$$AV = Q$$

$$A = QV^{-1} = QV^T \qquad \text{chuyển vế V}$$

$$A = QV^{-1} = U\Sigma V^T \qquad \text{tách Q}$$

Tổng quát

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

A là ma trận $n \times d$, có n điểm và mỗi điểm có d phần tử

U là ma trận $n \times n$ chứa độ dài hình chiếu, trong đó các vector cột có chiều dài bằng 1

Σ là ma trận đường chéo $(n \times d)$, xác định độ dài của các vector

V là ma trận $d \times d$ chứa các trục

Trong Python tính SVD với hàm **numpy.linalg.svd**

Ứng dụng SVD cho foreground removal

Idea: Mỗi hình trong video được xem như một vector. Từ video → A. Sau đó tính SVD cho A.

Chỉ dùng r giá trị lớn nhất trong Σ (r trục chứa thông tin quan trọng và phổ biến nhất).

Tính lại A với Σ mới

Hình gốc

Hình với Σ mới

Hình gốc

Hình với Σ mới

Hình gốc

Hình với Σ mới

Hình gốc

Hình với Σ mới

Outline

- Vector and Matrix
- Cosine Similarity
- Implementation
- Case Studies

Cosine similarity

Cosine similarity (cs) được dùng để đo mức độ giống nhau/tương đồng giữa hai vector

Gọi $\vec{x}$ và $\vec{y}$ là hai vector, cs được tính như sau

$$cs(\vec{x},\vec{y}) = \frac{\vec{x}.\vec{y}}{\|\vec{x}\|\|\vec{y}\|} = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2}\sqrt{\sum_1^n y_i^2}}$$

Tính chất 1: $cs(\vec{x},\vec{y}) = cs(a\vec{x}, b\vec{y})$

$$cs(a\vec{x}, b\vec{y}) = \frac{a\vec{x}.b\vec{y}}{\|a\vec{x}\|\|b\vec{y}\|} = \frac{\sum_1^n a x_i b y_i}{\sqrt{\sum_1^n a^2 x_i^2}\sqrt{\sum_1^n b^2 y_i^2}}$$

$$= \frac{ab\sum_1^n x_i y_i}{\sqrt{a^2\sum_1^n x_i^2}\sqrt{b^2\sum_1^n y_i^2}}$$

$$= \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2}\sqrt{\sum_1^n y_i^2}} = cs(\vec{x},\vec{y})$$

Tính chất 2: $cs(\vec{x},\vec{y}) \neq cs(\vec{x}+c, \vec{y}+d)$

Ví dụ:

$\vec{x} = [4, 2, 1, 2]^T$

$\vec{y} = [1, 2, 2, 0]^T$

$\vec{u} = 2\vec{x} = [8, 4, 2, 4]^T$

$\vec{v} = 3\vec{y} = [3, 6, 6, 0]^T$

$$cs(\vec{x},\vec{y}) = \frac{4*1+2*2+1*2+2*0}{\sqrt{4^2+2^2+1^2+2^2}\sqrt{1^2+2^2+2^2+0}}$$

$$= \frac{10}{\sqrt{25}\sqrt{9}} = \frac{10}{15} = 0.67$$

$$cs(\vec{u},\vec{v}) = \frac{8*3+4*6+2*6+4*0}{\sqrt{8^2+4^2+2^2+4^2}\sqrt{3^2+6^2+6^2+0}}$$

$$= \frac{60}{\sqrt{100}\sqrt{81}} = \frac{60}{90} = 0.67$$

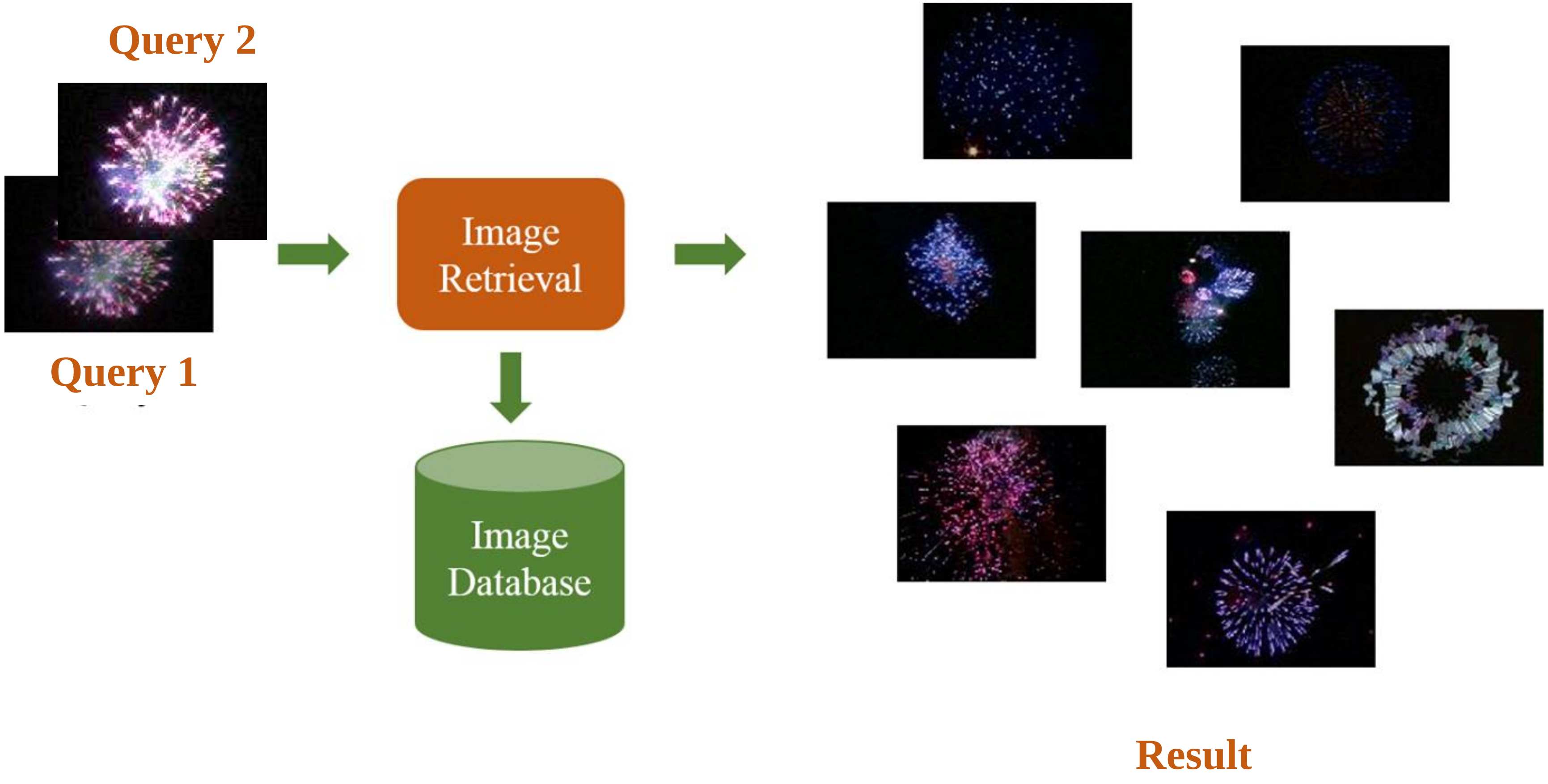
$$= cs(\vec{x},\vec{y})$$

Image Retrieval

Query 2

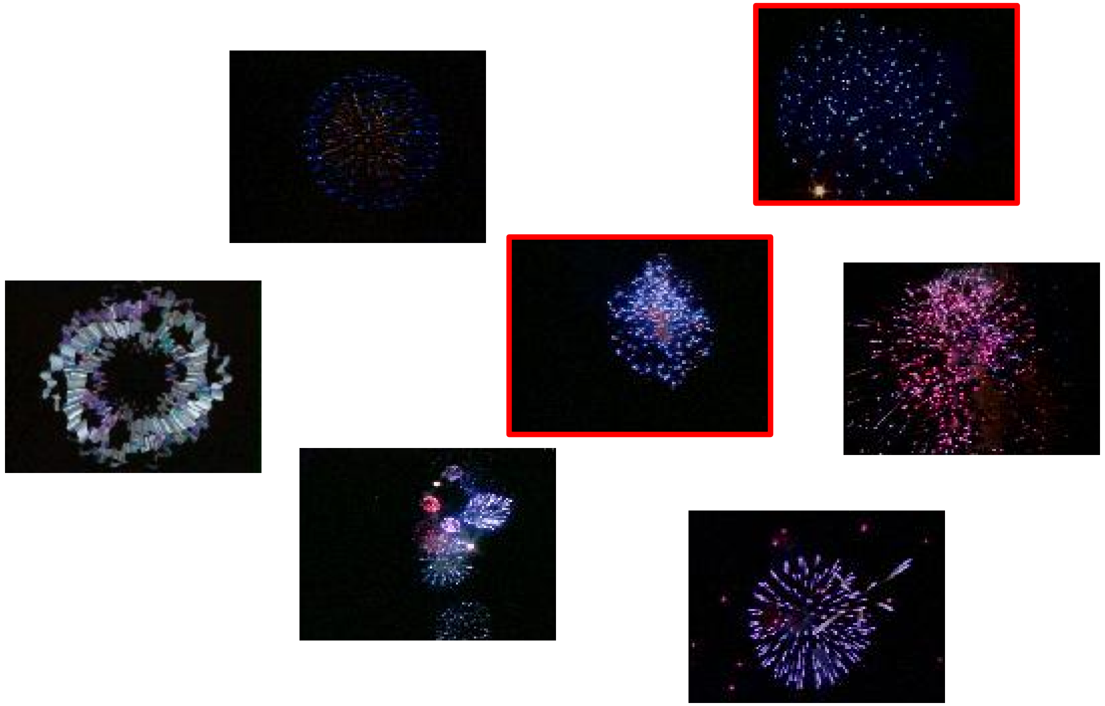
= 2 *

Query 1



Hai hình query cho cùng kết quả như nhau

Query 1

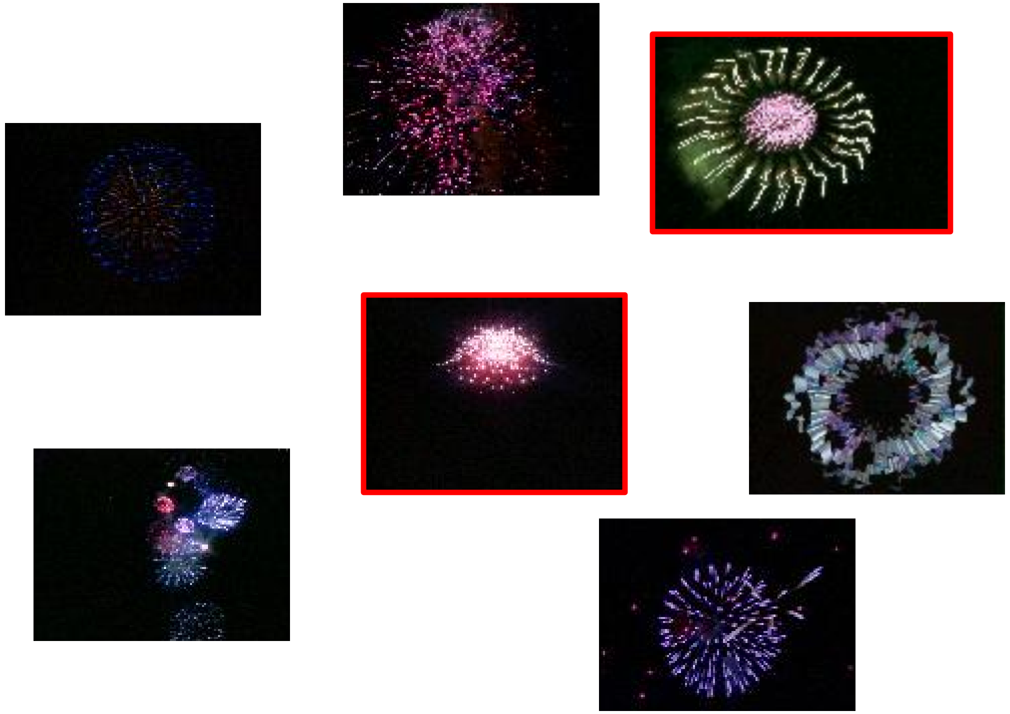


Result 1

Query 2

$= 50 +$

Query 1



Result 2

Result 1 ≠ Result 2

Cosine similarity

❖ Code

```
import math

def cosine_similarity(vector1, vector2):
    '''
    Compute dot product between two vectors
    Output is a floating-point number
    '''

    sumxy = sum([v1*v2 for v1, v2 in zip(vector1, vector2)])
    sumxx = sum([v1*v2 for v1, v2 in zip(vector1, vector1)])
    sumyy = sum([v1*v2 for v1, v2 in zip(vector2, vector2)])

    return sumxy/math.sqrt(sumxx*sumyy)

# test case
vector1 = [5, 3, 2, 7]
vector2 = [2, 9, 4, 1]

output = cosine_similarity(vector1,vector2)
print(output)
```

```
0.552005787925351
```

Outline

- Vector and Matrix
- Cosine Similarity
- Implementation
- Case Studies

Image Retrieval

❖ Database

Query Images

Database

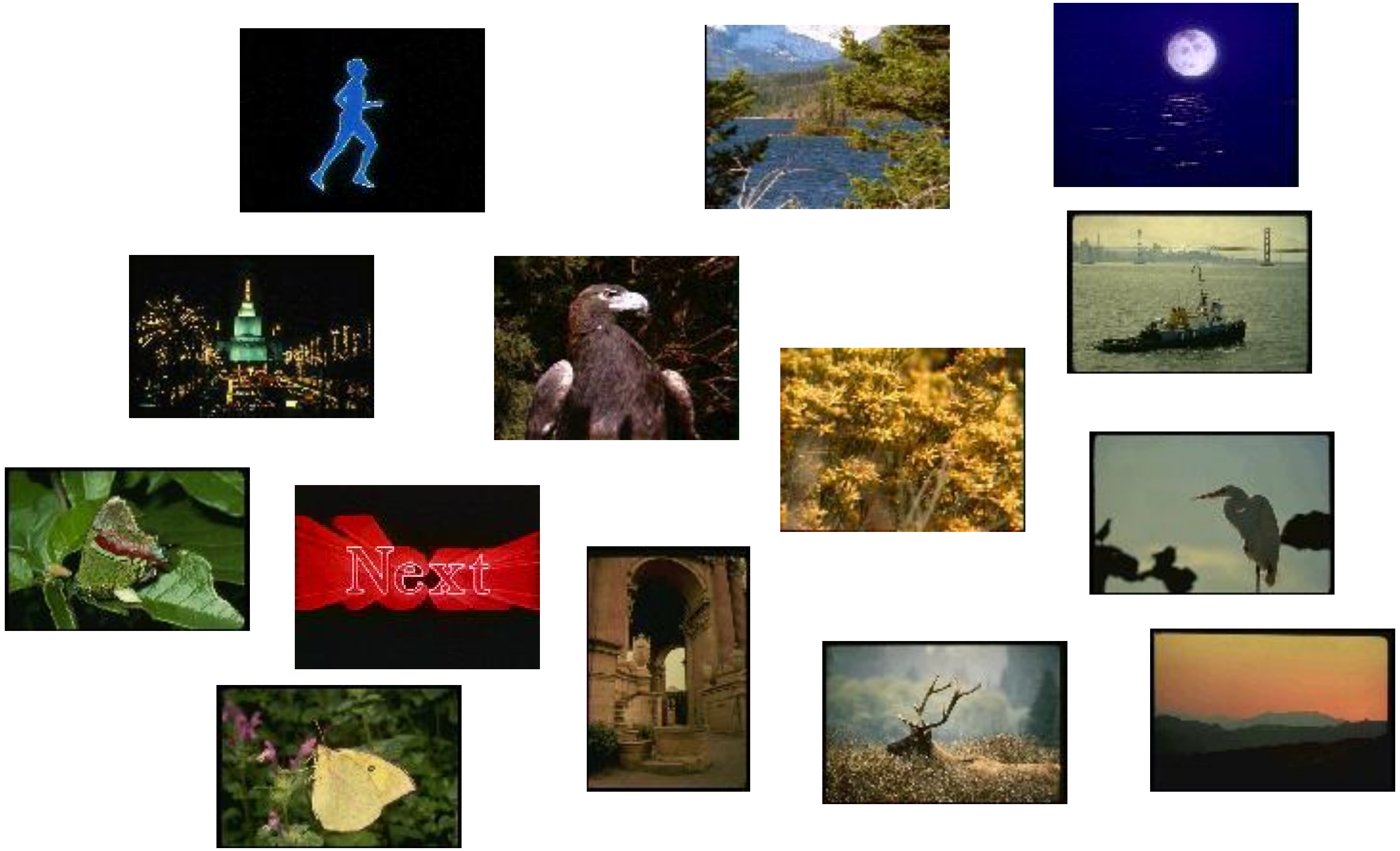


Image Retrieval

Ứng dụng Cosine Similarity để tính mức độ giống nhau giữa hai hình

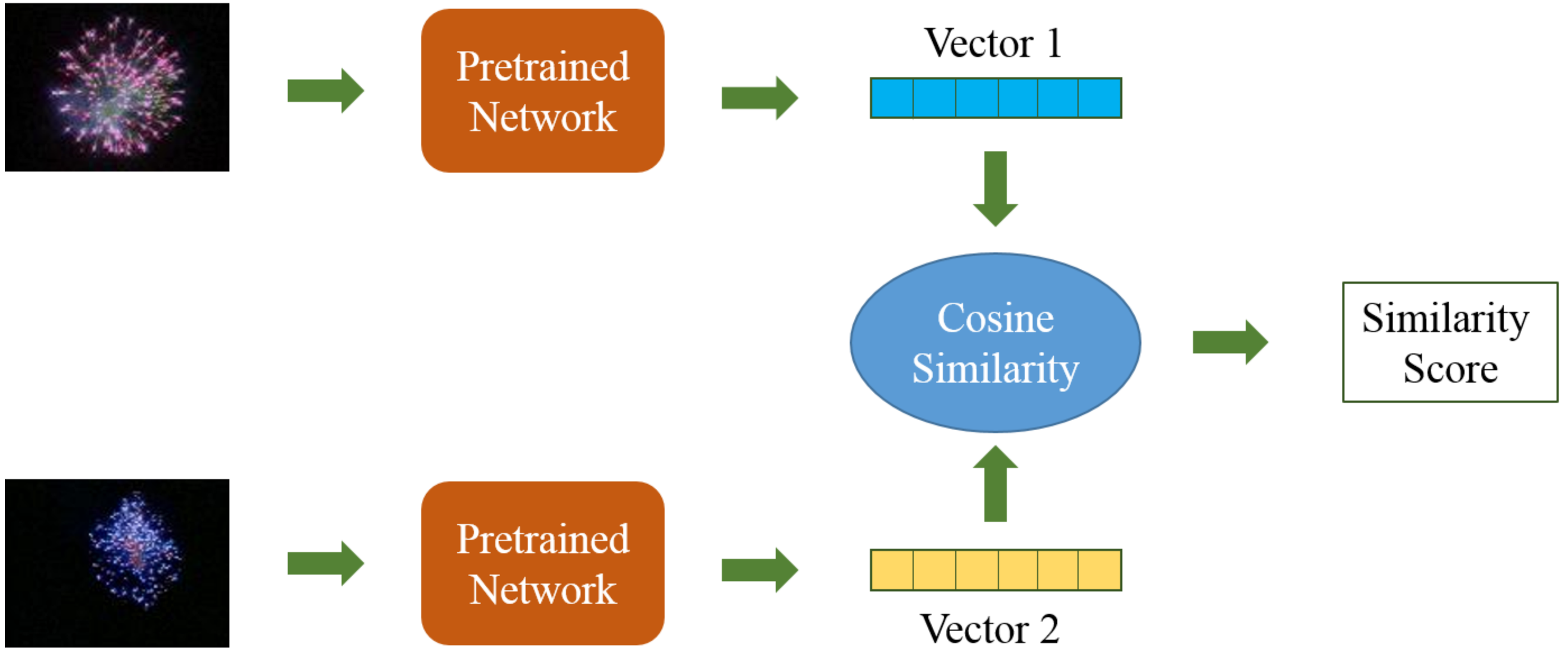


Image Retrieval

❖ Database Preparation

```
import numpy as np
from tensorflow.keras.preprocessing import image as kimage
from tqdm import tqdm

images = []
lists = [i for i in range(9908)]

for index in tqdm(lists):
    img = kimage.load_img('images_mr/%d.jpg' % (index), target_size=(86, 128))
    img_np = kimage.img_to_array(img)
    images.append(img_np)

images_np = np.array(images)
print(images_np.shape)
```

```
100%|██████████████████████████████████████████████████|
s]
```

```
(9908, 86, 128, 3)
```

Image Retrieval

❖ Database Preparation

$p \in [0, 255]$

Byte 1 Byte 2 Byte 3 Byte 4

float32

Byte 1

uint8

Image Retrieval

❖ Database Preparation

```python
import numpy as np
from tensorflow.keras.preprocessing import image as kimage
from tqdm import tqdm

images = []
lists = [i for i in range(9908)]
for index in tqdm(lists):
    img = kimage.load_img('images_mr/%d.jpg' % (index), target_size=(86, 128))
    img_np = kimage.img_to_array(img)

    images.append(img_np)

# convert to np.array
images_np = np.array(images)

# reduce memory used
images_np = images_np.astype(np.uint8)
print(images_np.shape)
```

```
100%|██████████████████████████████████████████████████|
s]

(9908, 86, 128, 3)
```

Image Retrieval

❖ Database Preparation

```python
import numpy as np

data = np.load('images_mr.npy', allow_pickle=True)
print(data.shape)
print(type(data[0,0,0,0]))

data = data.astype(np.float32)
print(type(data[0,0,0,0]))

print(np.amin(data))
print(np.amax(data))
```

```
(9908, 86, 128, 3)
<class 'numpy.uint8'>
<class 'numpy.float32'>
0.0
255.0
```

Image Retrieval

❖ Using absolute difference

Database

Image Retrieval

❖ Using cosine similarity

Database

Image Retrieval

❖ **Feature extraction**

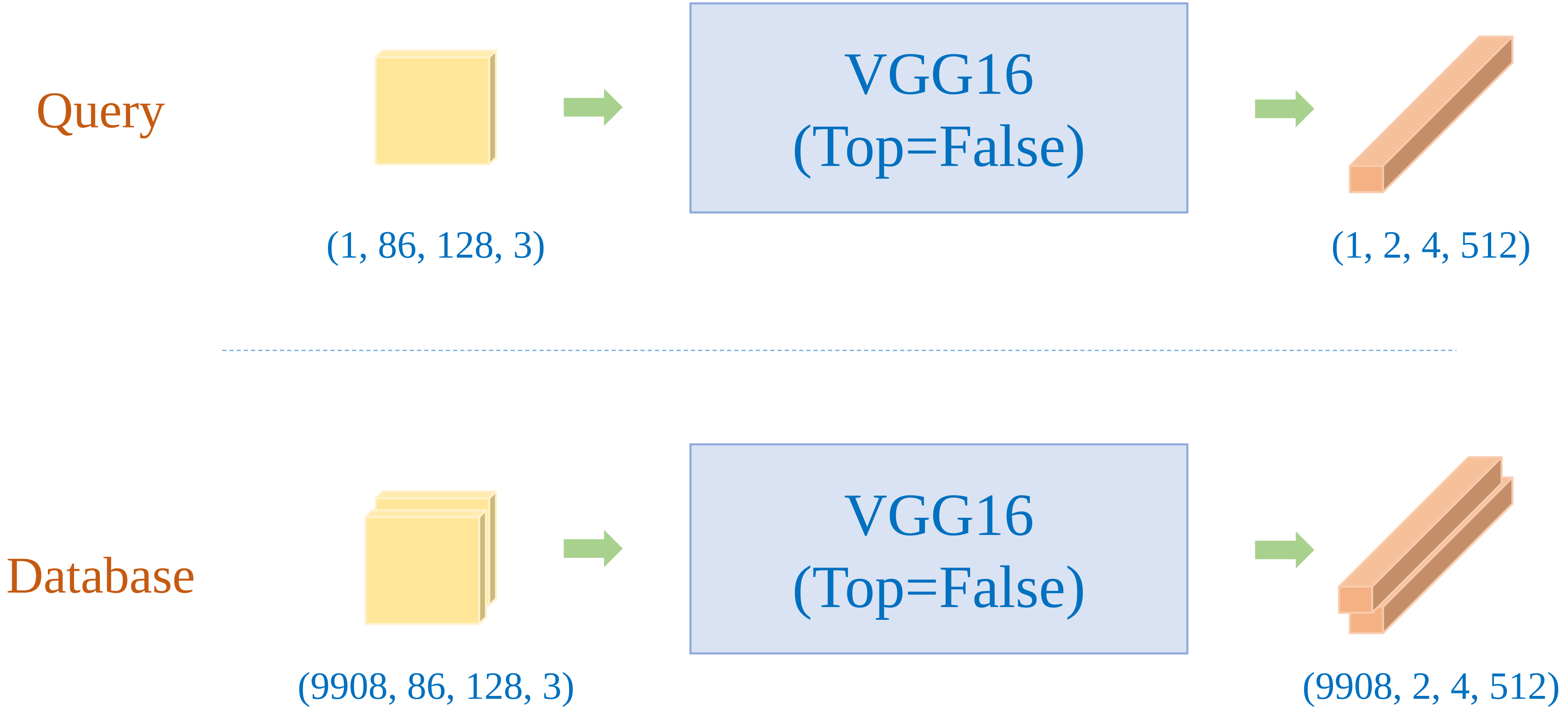


Image Retrieval

❖ Doing

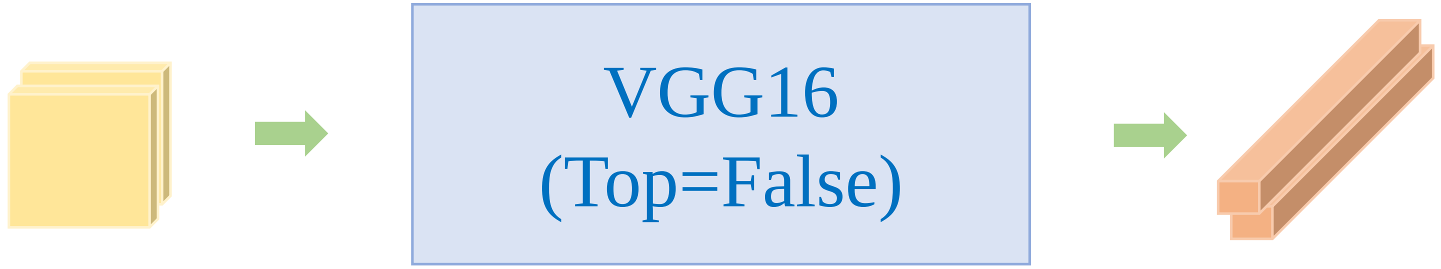

```python
# load query
query = kimage.load_img(PATH+'q2.jpg', target_size=(86, 128))
query_np = kimage.img_to_array(query)
query_np = np.expand_dims(query_np, axis=0)
query_np = preprocess_input(query_np)

pred_query = model.predict(query_np)
print(pred_query.shape)
```

```
(1, 2, 4, 512)
```

Image Retrieval

❖ Doing

Database



(9908, 86, 128, 3)

(9908, 2, 4, 512)

```
PATH = '/content/gdrive/My Drive/data/image_retrieval2/'
data = np.load(PATH+'images_mr.npy', allow_pickle=True)
data = data.astype(np.float32)

data = preprocess_input(data)
pred_data  = model.predict(data)
print(pred_data.shape)
```

```
(9908, 2, 4, 512)
```

Image Retrieval

❖ Cost Functions

Absolute Difference

$$cs(\vec{x}, \vec{y}) = \sum_1^n |x_i - y_i|$$

Cosine Similarity

$$cs(\vec{x}, \vec{y}) = \frac{\vec{x}.\vec{y}}{\|\vec{x}\|\|\vec{y}\|} = \frac{\sum_1^n x_i y_i}{\sqrt{\sum_1^n x_i^2}\sqrt{\sum_1^n y_i^2}}$$

```
data_abs = tf.math.abs(data1_tile - data2)
errors = tf.math.reduce_sum(data_abs, axis=1)
```

```
data1  = np.reshape(data1,(1, -1))
data2  = np.reshape(data2,(N, -1))
sims   = cosine_similarity(data1,  data2)
```

Image Retrieval

❖ Save features

Database

Image Retrieval

❖ **Pre-trained models with different sizes**

Database

Image Retrieval

❖ Pre-trained models with different sizes

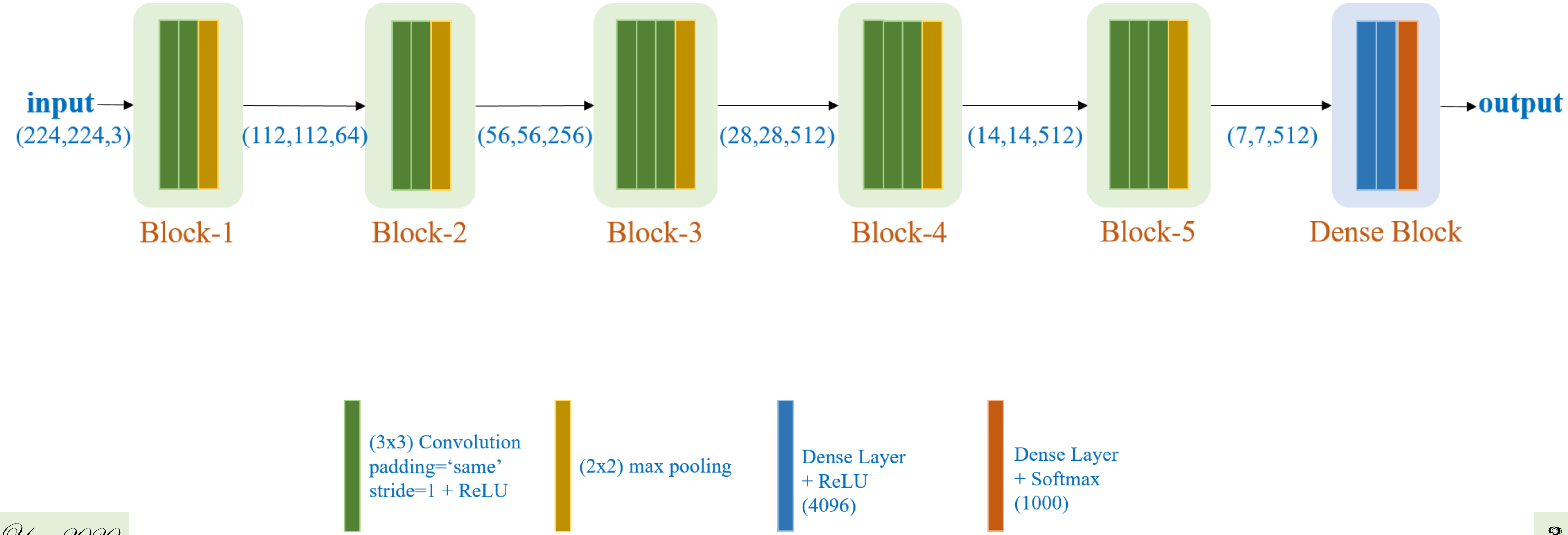


Image Retrieval

- **Pre-trained models with different sizes**
  - **Batch processing**

Outline

- Vector and Matrix
- Cosine Similarity
- Implementation
- Case Studies

Mean

Data

$$X = \{X_1, \dots, X_N\}$$

Formula

$$\mu = \frac{1}{N}\sum_{i-1}^{N} X_i$$

Given the data

$$X = \{2, 8, 5, 4, 1, 8\}$$

$$N = 6$$

$$\mu = \frac{1}{N}\sum_{i-1}^{N} X_i = \frac{1}{6}(2 + 8 + 5 + 4 + 1 + 8)$$

$$= \frac{18}{6} = 3$$

Mean

❖ Code

```python
def calculate_mean(numbers): #1
    s = sum(numbers)         #2
    N = len(numbers)         #3
    mean = s/N               #4
    return mean              #5

# Tạo mảng donations đại diện cho số tiền quyên góp trong 12 ngày
donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000,1200]

mean_value = calculate_mean(donations)
print('Trung bình số tiền quyên góp là: ', mean_value)
```

#1. Đặt tên là calculate_mean(), hàm này sẽ nhận đối số numbers, là chuỗi các số cần tính trung bình.

#2. Sử dụng hàm sum() để tính tổng dãy số cho trước.

#3. Sử dụng hàm len() để tính chiều dài của dãy số cần tính.

#4. Tính trung bình của dãy số trên bằng cách lấy tổng chia cho chiều dài.

#5. Cuối cùng ta cho hàm trả về giá trị mean tính được.

Median

Data

$$X = \{X_1, \dots, X_N\}$$

Formula

Step 1: Sort X → S

Step 2

If N is odd, then $m = S_{\left(\frac{N+1}{2}\right)}$

If N is even, then $m = \left(S_{\left(\frac{N}{2}\right)} + S_{\left(\frac{N}{2}+1\right)}\right)/2$

Given the data

$$X = \{2, 8, 5, 4, 1\}$$

$$N = 5$$

Step 1

$$S = \{1, 2, 4, 5, 8\}$$

1 2 3 4 5

Step 2; $N = 5$

$$k = \frac{N+1}{2} = 3$$

$$m = S_k = 4$$

Median

Data

$$X = \{X_1, \dots, X_N\}$$

Formula

Step 1: Sort X → S

Step 2

If N is odd, then $m = S_{\left(\frac{N+1}{2}\right)}$

If N is even, then $m = \left(S_{\left(\frac{N}{2}\right)} + S_{\left(\frac{N}{2}+1\right)}\right)/2$

Given the data

$$X = \{2, 8, 5, 4, 1, 8\}$$

$$N = 6$$

Step 1

$$S = \{1, 2, 4, 5, 8, 8\}$$

1 2 3 4 5 6

Step 2; $N = 6$

$$m = \frac{S_3 + S_4}{2}$$

$$= \frac{4+5}{2} = 4.5$$

Median

❖ Code

```
def calculate_median(numbers):                           #1
    N = len(numbers)                                     #2
    numbers.sort()                                       #3
    if N%2 == 0:                                         #4
        m1 = N/2
        m2 = (N/2) + 1
        m1 = int(m1)-1
        m2 = int(m2)-1
        median = (numbers[m1] + numbers[m2])/2
    else:                                                #5
        m = (N+1)/2
        m = int(m)-1
        median = numbers[m]
    return median                                        #6
```

Mean and Median

❖ Comparison

❖ Noise

Data

$$X = \{X_1, \dots, X_N\}$$

Formula

$$\mu = \frac{1}{N}\sum_{i-1}^{N} X_i$$

Formula

Step 1: Sort X → S

Step 2

If N is odd, then $m = S_{\left(\frac{N+1}{2}\right)}$

If N is even, then $m = \left(S_{\left(\frac{N}{2}\right)} + S_{\left(\frac{N}{2}+1\right)}\right)/2$

Mean and Median

- **Comparison**
  - **Image denoising**

Data

$$X = \{X_1, \dots, X_N\}$$

Formula

$$\mu = \frac{1}{N}\sum_{i-1}^{N} X_i$$

Formula

Step 1: Sort X → S

Step 2

If N is odd, then $m = S_{\left(\frac{N+1}{2}\right)}$

If N is even, then $m = \left(S_{\left(\frac{N}{2}\right)} + S_{\left(\frac{N}{2}+1\right)}\right)/2$

Mean and Median

Làm mờ ảnh
dựa vào mean

Khử nhiễu
dựa vào median

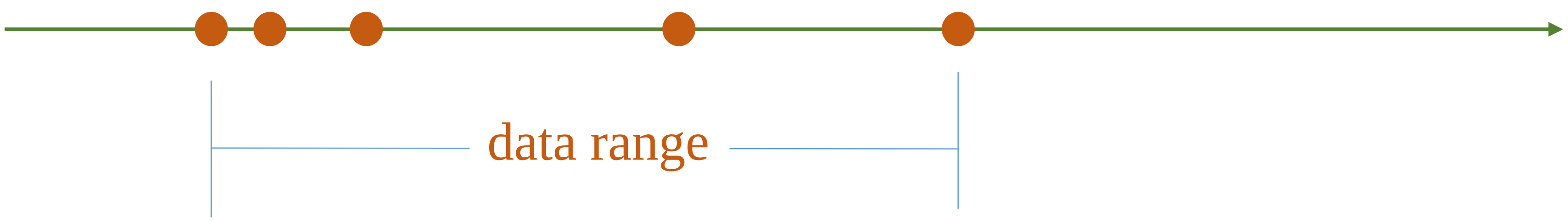
Mode

❖ Code

```
# import packages Counter để đếm số lần xuất hiện của mỗi giá trị trong chuỗi
from collections import Counter

# data
points = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10]

def calculate_mode(numbers): #1
    c = Counter(numbers)     #2
    mode = c.most_common(1)  #3
    return mode[0][0]        #4

print('Mode của chuỗi số đã cho: ', calculate_mode(points))
```

Range

❖ Procedure




```python
def find_range(numbers):            #1
    lowest  = min(numbers)          #2
    highest = max(numbers)          #3
    r = highest-lowest              #4
    print('Lowest: {0}\tHighest: {1}\tRange: {2}'.format(lowest, highest, r))

# data
points = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10, 6, 6]
find_range(points)
```

Variance

Formula: $X_{norm} = \dfrac{X - \mu}{\sigma}$

mean $\mu = \dfrac{1}{n}\sum_{k=1}^{n} x_i$

variance $var(X) = \dfrac{1}{n}\sum_{k=1}^{n} (x_i - \mu)^2$

Standard deviation $\sigma = \sqrt{var(X)}$

Example: $X = \{5, 3\ 6, 7, 4\}$

$$\mu = \frac{1}{5}\sum_{k=1}^{n}(5+3+6+7+4) = \frac{25}{5} = 5$$

$$var(X) = \frac{1}{5}[(5-5)^2 + (3-5)^2 + (6-5)^2 + (7-5)^2 + (4-5)^2]$$

$$= \frac{1}{5}(0+4+1+4+1)=2$$

$$\sigma = \sqrt{var(X)} = 1.41$$

$$X_{norm} = \frac{X-5}{1.41} = \{5, 3\ 6, 7, 4\}$$

Variance $var(X)$ xác định độ phân tán dữ liệu so với giá trị trung bình μ

mean $\mu = \frac{1}{n}\sum_{k=1}^{n} x_i$

variance $var(X) = \frac{1}{n}\sum_{k=1}^{n} (x_i - \mu)^2$

standard deviation $std(X) = \sqrt{var(X)}$

Ví dụ: $X = \{5, 3\ 6, 7, 4\}$

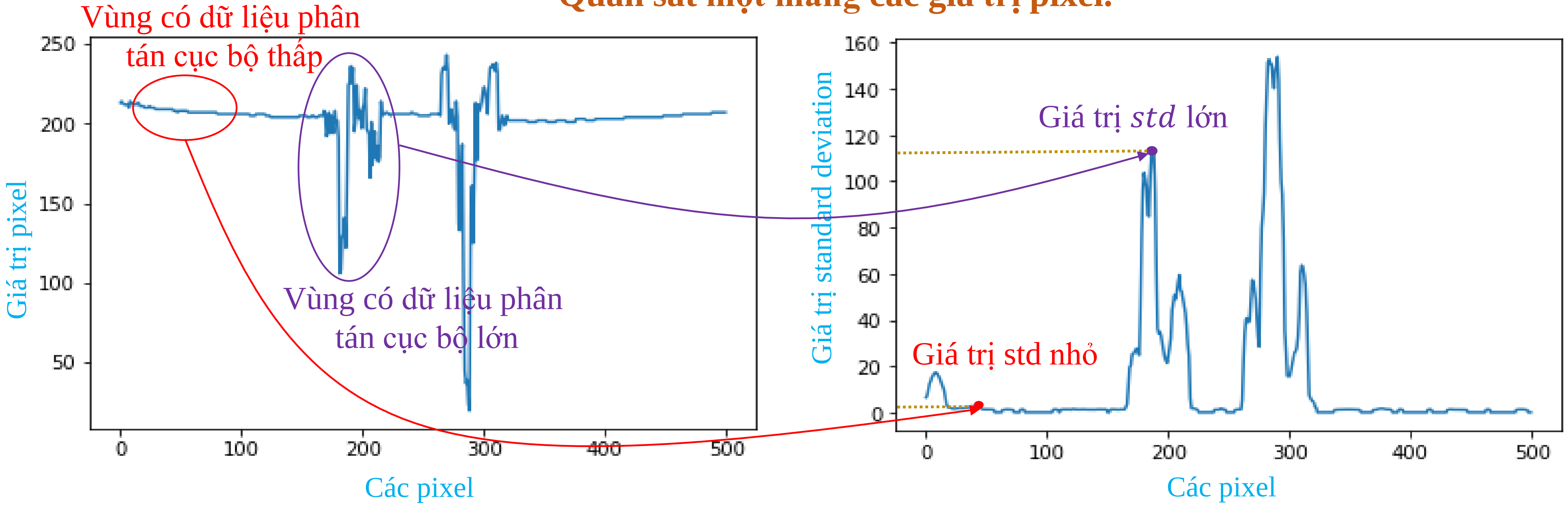
$$\mu = \frac{1}{5}\sum_{k=1}^{n}(5+3+6+7+4) = \frac{25}{5} = 5$$

$$var(X) = \frac{1}{5}[(5-5)^2 + (3-5)^2 + (6-5)^2 + (7-5)^2 + (4-5)^2]$$

$$= \frac{1}{5}(0+4+1+4+1)=2$$

$$std(X) = \sqrt{var(X)} = 1.41$$

Quan sát một mảng các giá trị pixel.



Variance

Ứng dụng tính chất của variance (~standard deviation) để tìm texture cho một hình

Ảnh gốc

Tính standard deviation cục bộ cho từng pixel.

Ảnh thông tin texture

Variance

❖ Code

```
def calculate_mean(numbers):                        #1
    s = sum(numbers)
    N = len(numbers)
    mean = s/N
    return mean

def caculate_variance(numbers):                     #2
    mean = calculate_mean(numbers)                  #3

    diff = []                                       #4
    for num in numbers:
         diff.append(num-mean)

    squared_diff = []                               #5
    for d in diff:
        squared_diff.append(d**2)
        sum_squared_diff = sum(squared_diff)
        variance = sum_squared_diff/len(numbers)

    return variance
```

Hệ số tương quan (correlation coefficient)

Công thức: Gọi x,y là hai biến ngẫu nhiên

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$

$$= \frac{n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)}{\sqrt{n\sum_i x_i^2 - (\sum_i x_i)^2}\sqrt{n\sum_i y_i^2 - (\sum_i y_i)^2}}$$

Tính chất 1

$$-1 \quad \leq \quad \rho_{xy} \quad \leq \quad 1$$

Tương quan nghịch

Tương quan thuận

Tính chất 2

$$\rho_{xy} = \rho_{uv}$$

trong đó

$$u = ax + b$$

$$v = cy + d$$

Ví dụ 1

x = [7, 18, 29, 2, 10, 9, 9]

y = [1, 6, 12, 8, 6, 21, 10]

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$

$$= \frac{n*818 - 84*64}{\sqrt{n*1480 - 7056}\sqrt{n*822 - 4096}} = 0.149$$

Ví dụ 2

u=2*x-14 = [0, 22, 44, -10, 6, 4, 4]

v=y+2 = [3, 8, 14, 10, 8, 23, 12]

$$\rho_{uv} = \frac{E[(u-\mu_u)[(v-\mu_v)]}{\sqrt{var(u)}\sqrt{var(v)}}$$

$$= \frac{n*880 - 70*78}{\sqrt{n*2588 - 4900}\sqrt{n*1106 - 6084}} = 0.149$$

Correlation Coefficient

Công thức: Gọi x,y là hai biến ngẫu nhiên

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$

$$= \frac{n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)}{\sqrt{n\sum_i x_i^2 - (\sum_i x_i)^2}\sqrt{n\sum_i y_i^2 - (\sum_i y_i)^2}}$$

```python
def find_corr_x_y(x,y):                                          #1
    n = len(x)                                                   #2
    prod = []
    for xi,yi in zip(x,y):                                       #3
        prod.append(xi*yi)

    sum_prod_x_y = sum(prod)                                     #4

    sum_x = sum(x)
    sum_y = sum(y)

    squared_sum_x = sum_x**2
    squared_sum_y = sum_y**2

    x_square = []
    for xi in x:
        x_square.append(xi**2)
    x_square_sum = sum(x_square)

    y_square=[]
    for yi in y:
        y_square.append(yi**2)
    y_square_sum = sum(y_square)

    # Use formula to calculate correlation                       #5
    numerator = n*sum_prod_x_y - sum_x*sum_y
    denominator_term1 = n*x_square_sum - squared_sum_x
    denominator_term2 = n*y_square_sum - squared_sum_y
    denominator = (denominator_term1*denominator_term2)**0.5
    correlation = numerator/denominator

    return correlation
```

Image Retrieval

❖ Cost Functions

Công thức: Gọi x,y là hai biến ngẫu nhiên

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$
$$= \frac{n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)}{\sqrt{n\sum_i x_i^2 - (\sum_i x_i)^2}\sqrt{n\sum_i y_i^2 - (\sum_i y_i)^2}}$$

```
from scipy.stats.stats import pearsonr

sim_list = []
for i in range(9908):
  sim = pearsonr(pred_query[0], pred_data[i])
  sim_list.append(sim)
print(len(sim_list))
```

```
9908
```

Outline

- Introduction to Numpy
- Numpy Array Indexing
- Numpy Array Operations
- Broadcasting
- Data Processing

Template Matching

Hệ số tương quan (correlation coefficient)

Công thức: Gọi x,y là hai biến ngẫu nhiên

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$

$$= \frac{n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)}{\sqrt{n\sum_i x_i^2 - (\sum_i x_i)^2}\sqrt{n\sum_i y_i^2 - (\sum_i y_i)^2}}$$

Tính chất 1

$$-1 \quad \leq \quad \rho_{xy} \quad \leq \quad 1$$

Tương quan nghịch ← → Tương quan thuận

Tính chất 2

$$\rho_{xy} = \rho_{uv}$$

trong đó

$$u = ax + b$$

$$v = cy + d$$

Ví dụ 1

x = [7, 18, 29, 2, 10, 9, 9]
y = [1, 6, 12, 8, 6, 21, 10]

$$\rho_{xy} = \frac{E[(x-\mu_x)[(y-\mu_y)]}{\sqrt{var(x)}\sqrt{var(y)}}$$

$$= \frac{n*818 - 84*64}{\sqrt{n*1480 - 7056}\sqrt{n*822 - 4096}} = 0.149$$

Ví dụ 2

u=2*x-14 = [0, 22, 44, -10, 6, 4, 4]
v=y+2 = [3, 8, 14, 10, 8, 23, 12]

$$\rho_{uv} = \frac{E[(u-\mu_u)[(v-\mu_v)]}{\sqrt{var(u)}\sqrt{var(v)}}$$

$$= \frac{n*880 - 70*78}{\sqrt{n*2588 - 4900}\sqrt{n*1106 - 6084}} = 0.149$$

Correlation Coefficient

```python
def find_corr_x_y(x,y):                                     #1
    n = len(x)                                              #2
    prod = []
    for xi,yi in zip(x,y):                                  #3
        prod.append(xi*yi)

    sum_prod_x_y = sum(prod)                                #4

    sum_x = sum(x)
    sum_y = sum(y)

    squared_sum_x = sum_x**2
    squared_sum_y = sum_y**2

    x_square = []
    for xi in x:
        x_square.append(xi**2)
    x_square_sum = sum(x_square)

    y_square=[]
    for yi in y:
        y_square.append(yi**2)
    y_square_sum = sum(y_square)

    # Use formula to calculate correlation                  #5
    numerator = n*sum_prod_x_y - sum_x*sum_y
    denominator_term1 = n*x_square_sum - squared_sum_x
    denominator_term2 = n*y_square_sum - squared_sum_y
    denominator = (denominator_term1*denominator_term2)**0.5
    correlation = numerator/denominator

    return correlation
```

Ứng dụng cho patch matching

P_1 P_2 P_3 P_4

$\rho_{P_1P_2} = 0.55$

$\rho_{P_1P_3} = 0.23$ ➧ Ảnh P_2 giống với ảnh P_1 hơn so với P_3 và P_4

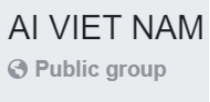
$\rho_{P_1P_4} = 0.30$

P_1 $P_2 = P_1 + 50$ $P_3 = 1.2P_1 + 10$

$\rho_{P_1P_2} = 0.9970$

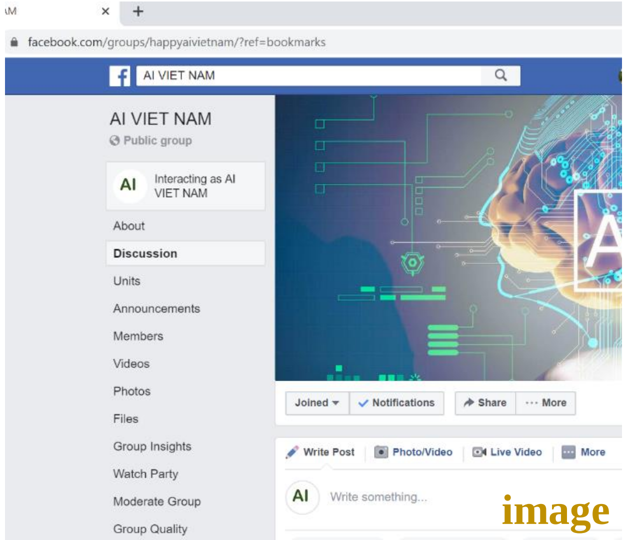
$\rho_{P_1P_3} = 0.9979$ ➧ ρ hoạt động tốt dưới sự thay đổi tuyến tính

Ứng dụng vào template matching



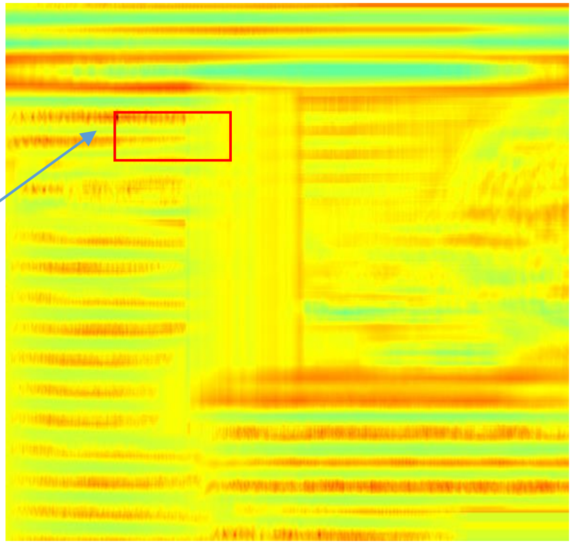
template

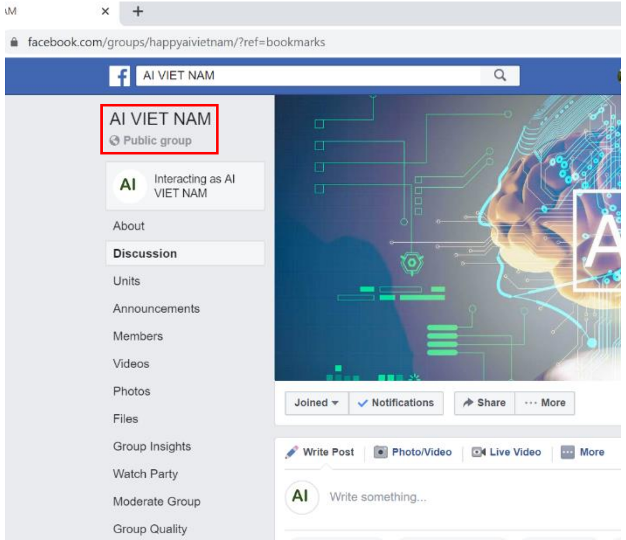
Tìm template có trong hình image



image

max





Map ρ cho từng pixel trong ảnh image

Kết quả

Ứng dụng cho patch matching

P_1 P_2 P_3 P_4

$\rho_{P_1P_2} = 0.55$

$\rho_{P_1P_3} = 0.23$ ➡ Ảnh P_2 giống với ảnh P_1 hơn so với P_3 và P_4

$\rho_{P_1P_4} = 0.30$

P_1 $P_2 = P_1 + 50$ $P_3 = 1.2P_1 + 10$

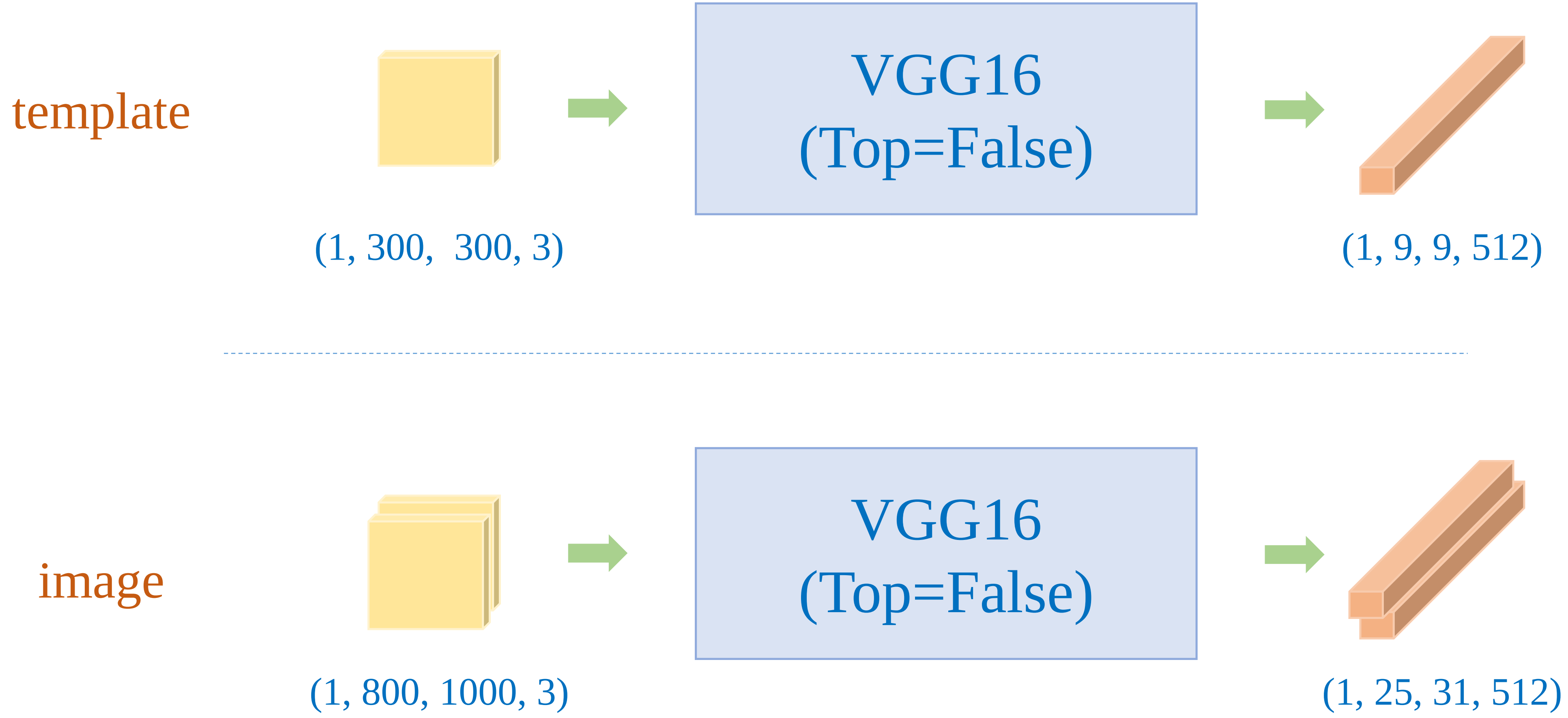
$\rho_{P_1P_2} = 0.9970$

$\rho_{P_1P_3} = 0.9979$ ➡ ρ hoạt động tốt dưới sự thay đổi tuyến tính

```python
# aivietnam.ai

import numpy as np
from PIL import Image

# load ảnh và chuyển về kiểu list
image1 = Image.open('images/img1.png')
image2 = Image.open('images/img2.png')
image3 = Image.open('images/img3.png')
image4 = Image.open('images/img4.png')

image1_list = np.asarray(image1).flatten().tolist()
image2_list = np.asarray(image2).flatten().tolist()
image3_list = np.asarray(image3).flatten().tolist()
image4_list = np.asarray(image4).flatten().tolist()


# tính correlation coefficient
corr_1_2 = find_corr_x_y(image1_list, image2_list)
corr_1_3 = find_corr_x_y(image1_list, image3_list)
corr_1_4 = find_corr_x_y(image1_list, image4_list)

print('corr_1_2:', corr_1_2)
print('corr_1_3:', corr_1_3)
print('corr_1_4:', corr_1_4)
```

Image Retrieval

❖ Feature extraction



Template Matching

❖ **Get the VGG16 model**

```
model = tf.keras.applications.VGG16(include_top=False,
                                    weights='imagenet',
                                    input_shape=(None, None, 3))

model.summary()
```

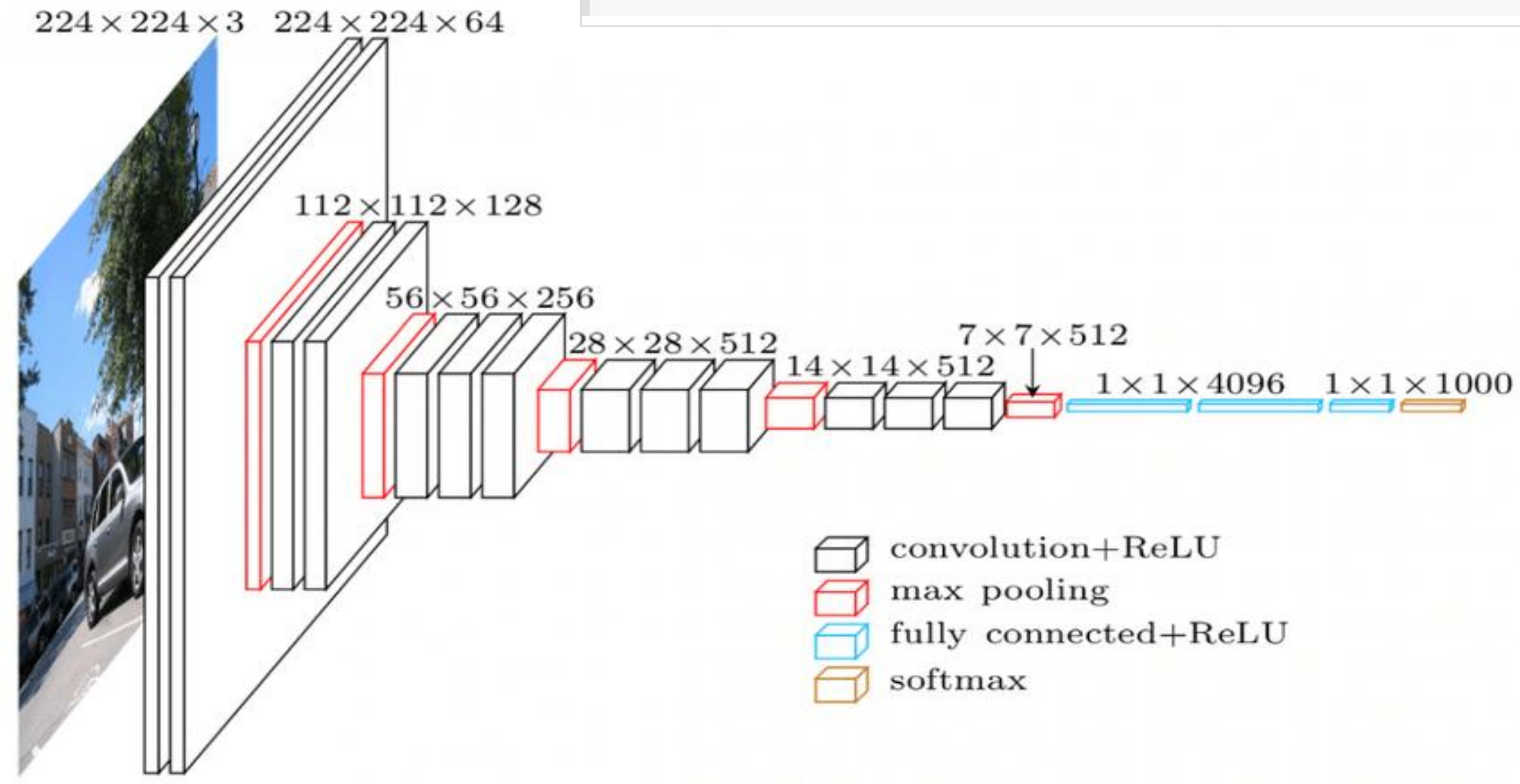


Image Retrieval

❖ Feature extraction


```python
from tensorflow.keras.preprocessing import image as kimage

# load template
template = kimage.load_img(PATH+'template.jpg', target_size=(300, 300))

# add one more dim
template_dim = np.expand_dims(template, axis=0) # (1, 300, 300, 3)

# compute features
template_feature = model.predict(template_dim)  # (1, 9, 9, 3)
```

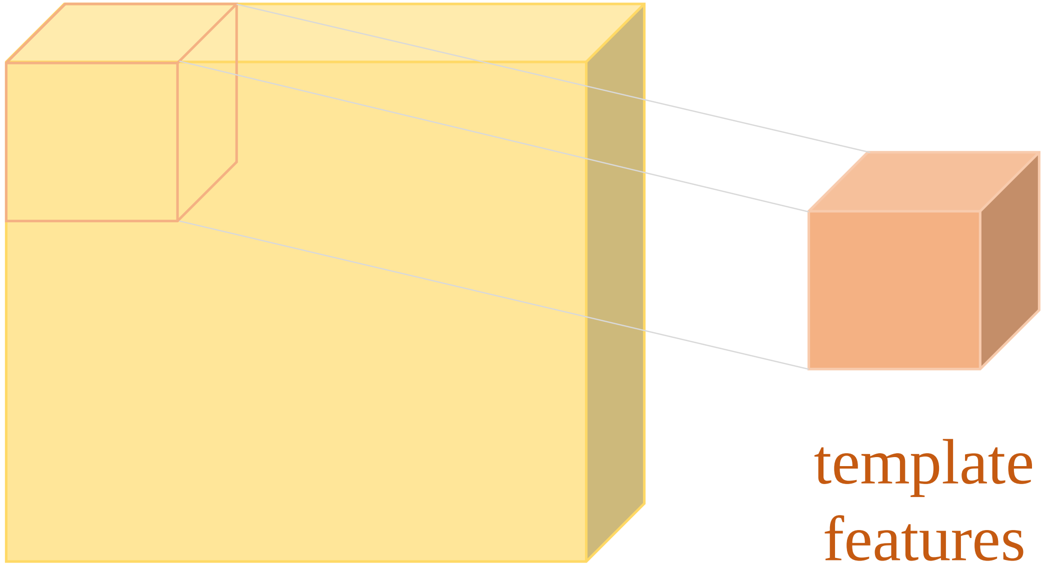
Image Retrieval

❖ Feature extraction


```python
from tensorflow.keras.preprocessing import image as kimage

# load template
image = kimage.load_img(PATH+'image2.jpg', target_size=(800, 1000))

# add one more dim
image_dim = np.expand_dims(image, axis=0) # (1, 800, 1000, 3)

# compute features
image_feature = model.predict(image_dim)  # (1, 25, 31, 3)
```

Template Matching

❖ Compute similarity




```python
# some parameters
side = 9
height_fm = 25
width_fm  = 31

# to store similarity values
sim_data = []
for i in range(height_fm-side+1):
    for j in range(width_fm-side+1):
        # get patch at (i,j)
        patch = image_feature[0,i:i+side,j:j+side,:]

        # reshape
        patch = np.reshape(patch, (1, -1))
        template_feature = np.reshape(template_feature, (1,-1))

        # compute cosine similarity
        sim = cosine_similarity(patch, template_feature)

        # save to a list
        sim_data.append((sim[0][0], i, j))
```

Template Matching

❖ Object Location

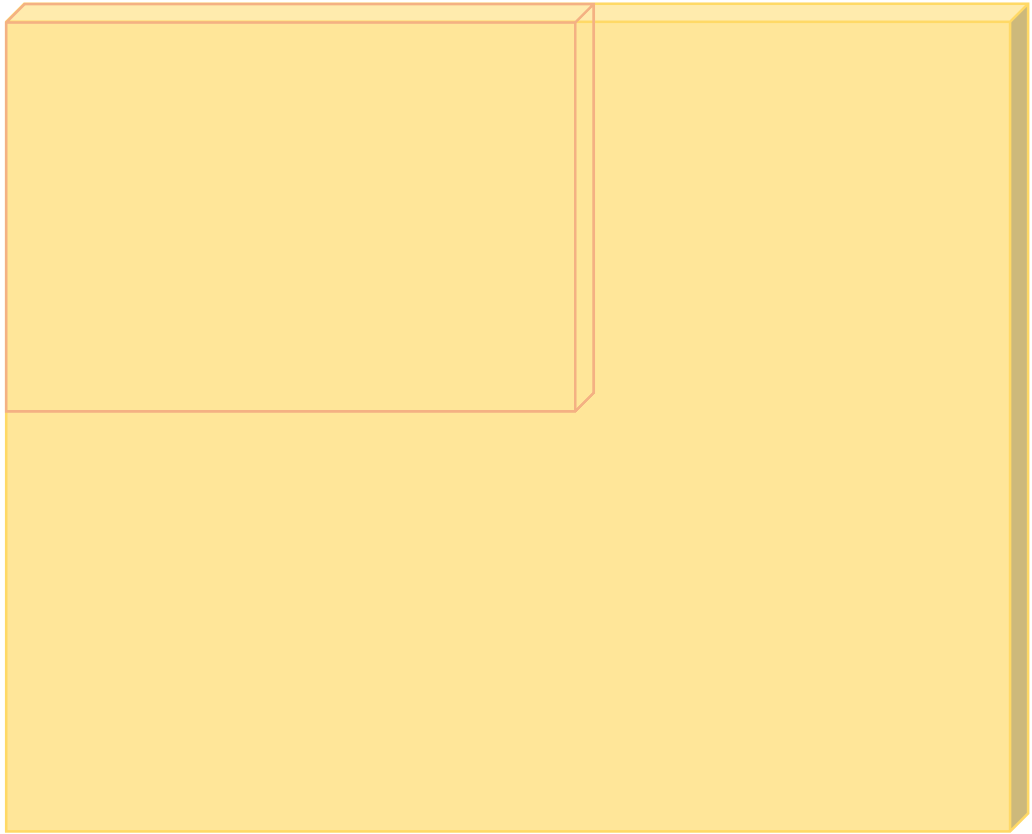


Image
(1, 800, 1000, 3)

```
scale_height = 800//25
scale_width  = 1000//31
```

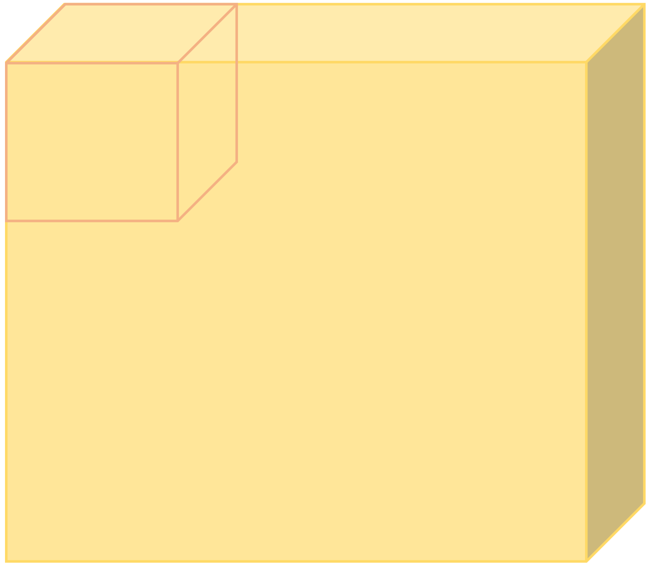


Image features
(1, 25, 31, 512)


Thank you!