AI VIET NAM – COURSE 2023

Module 2 – Excercise 5

25th June 2023

Giới thiệu về bài tập: Ở bài tập này các bạn sẽ được ôn tập và hiện thực các phép toán giữa vector với vector, vector với ma trận, và ma trận với ma trận sử dụng thư viện numpy. Cũng như cách thức tìm ma trận nghịch đảo, eigenvalues và eigenvectors. Ngoài ra các bạn cũng được tìm hiểu làm thể nào để đo lường sự giống (hoặc khác nhau) của 2 vector, cũng như của 2 ma trận. Cuối cùng là tìm hiểu ứng dụng (tuỳ chọn) background subtraction vào bài toán thay đổi hình nền.

Bài tập 1: Các phép toán trên vector và ma trận

1.1. Length of a vector

- Vector: $\mathbf{v} = [v_1, v_2, ...v_n]^T$
- Length of a vector: $\| \mathbf{v} \| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$

Hãy hoàn thiện function **computeVectorLength()** để tính độ dài của vector sử dụng thư viện numpy:

```
import numpy as np
def computeVectorLength(vector):
  # ***********Your code here **********
  return len_of_vector
```

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

```
vector = np.array([-2, 4, 9, 21])
result = computeVectorLength([vector])
print(round(result,2))
```

```
a) 43.28

b) 33.28

c) 13.28

d) 23.28
```

1.2. Dot product

- Vector: $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ ... \\ v_n \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ ... \\ u_n \end{bmatrix}$
- Dot Product: $\mathbf{v} \cdot \mathbf{u} = v_1 * u_1 + v_2 * u_2 + ... + v_n * u_n$

Hãy hoàn thiện function **computeDotProduct()** sử dụng thư viện numpy:

```
def computeDotProduct(vector1, vector2):
  # ***********Your code here **********
  return result
```

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

```
v1 = np.array([0, 1, -1, 2])
v2 = np.array([2, 5, 1, 0])
result = computeDotProduct(v1, v2)
print(round(result,2))
```

a) 3

b) 4

c) 5

d) 6

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

```
x = np.array([[1, 2],
              [3, 4]])
k = np.array([1, 2])
print('result \n', x.dot(k))
```

a) [5 11]

b) [6 11]

c) [7 11]

d) [5 12]

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

```
x = np.array([[-1, 2],
```

```
            [3, -4]])
k = np.array([1, 2])
print('result \n', x@k)
```

```
a) [ 5 -5]

b) [ 3 -5]

c) [ 4 -5]

d) [ 7 -5]
```

1.3. Multiplying a vector by a matrix

- Matrix: $\mathbf{A} = \begin{bmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ a_{m1} & ... & a_{mn} \end{bmatrix}$, $\mathbf{A} \in R^{m*n}$
- Vector: $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ ... \\ v_n \end{bmatrix}$, $\mathbf{v} \in R^n$
- $\mathbf{c} = \mathbf{Av} = \begin{bmatrix} a_{11} * v_1 + ... + a_{1n} * v_n \\ ... \\ a_{m1} * v_1 + ... + a_{mn} * v_n \end{bmatrix}$, $\mathbf{c} \in R^n$

Hãy hoàn thiện function **matrix_multi_vector()** sử dụng thư viện numpy:

```
def matrix_multi_vector(matrix, vector):
  # ***********Your code here **********
  return result
```

Question 5: Kết quả của đoạn code sau đây là gì:

```
m = np.array([[-1, 1, 1], [0, -4, 9]])
v = np.array([0, 2, 1])
result = matrix_multi_vector(m, v)
print(result)
```

```
a) [3 1]

b) [1 3]

c) [2 3]
```

```
d) [3 2]
```

1.4. Multiplying a matrix by a matrix

- Matrix A: $\mathbf{A} = \begin{bmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ a_{m1} & ... & a_{mn} \end{bmatrix}$, $\mathbf{A} \in R^{m*n}$
- Matrix B: $\mathbf{B} = \begin{bmatrix} b_{11} & ... & b_{1k} \\ ... & ... & ... \\ b_{n1} & ... & b_{nk} \end{bmatrix}$, $\mathbf{B} \in R^{n*k}$
- $\mathbf{C} = \mathbf{AB} = \begin{bmatrix} a_{11} * b_{11} + ... + a_{1n} * b_{n1} & ... & a_{11} * b_{1k} + a_{1n} * b_{nk} \\ ... & ... & ... \\ a_{m1} * b_{11} + ... + a_{mn} * b_{n1} & ... & a_{m1} * b_{1k} + a_{mn} * b_{nk} \end{bmatrix}$, $\mathbf{C} \in R^{m*k}$

Hãy hoàn thiện function **matrix_multi_matrix()** sử dụng thư viện numpy:

```
import numpy as np
def matrix_multi_matrix(vector):
  # ***********Your code here **********
  return len_of_vector
```

Question 6: Kết quả của đoạn code sau đây là gì:

```
m1 = np.array([[0, 1, 2], [2, -3, 1]])
m2 = np.array([[1, -3],[6, 1], [0, -1]])
result = matrix_multi_matrix(m1, m2)
print(result)
```

```
a)
[[ 9  -1]
 [-16 -10]]

b)
[[ 8  -1]
 [-16 -10]]

c)
[[ 6  -1]
 [-16 -10]]

d)
[[ 7  -1]
 [-16 -10]]
```

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

```
m1 = np.eye(3)
m2 = np.array([[1, 1, 1],[2, 2, 2], [3, 3, 3]])
result = m1@m2
print(result)
```

```
a)
[[1. 1. 1.]
 [2. 2. 2.]
 [3. 3. 3.]]

b)
 [[1. 1. 1.]
 [1. 1. 2.]
 [3. 3. 3.]]

c)
[[1. 1. 1.]
 [2. 2. 2.]
 [1. 1. 3.]]

d)
[[1. 1. 1.]
 [2. 2. 2.]
 [2. 3. 2.]]
```

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

```
m1 = np.eye(2)
m1 = np.reshape(m1,(-1,4))[0]
m2 = np.array([[1, 1, 1, 1],[2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]])
result = m1@m2
print(result)
```

```
a) [6. 6. 6. 6.]

b) [4. 4. 4. 4.]

c) [3. 3. 3. 3.]

d) [5. 5. 5. 5.]
```

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

```
m1 = np.array([[1, 2], [3, 4]])
m1 = np.reshape(m1,(-1,4), "F")[0]
m2 = np.array([[1, 1, 1, 1],[2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]])
result = m1@m2
print(result)
```

```
a) [30 30 30 30]

b) [29 29 29 29]

c) [31 31 31 31]

d) [28 28 28 28]
```

1.5 Matrix inverse

- Matrix A: $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\mathbf{A} \in R^{2*2}$
- Determinant of $\mathbf{A} \in R^{2*2}$: $det(\mathbf{A}) = ad - bc$
- if $det(\mathbf{A}) \neq 0$ $\mathbf{A}$ is invertible
- Inverse Matrix: $\mathbf{A^{-1}} = \frac{1}{det(\mathbf{A})} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

Cho trước A $= \begin{bmatrix} -2 & 6 \\ 8 & -4 \end{bmatrix}$ Tìm $\mathbf{A^{-1}}$ (Trình bày chi tiết theo lược đồ 1.5 (Có thể viết bằng latex sau đó gửi file, hoặc viết bằng markdown trên google colab)

Dựa vào công thức ở trên, hãy hoàn thiện function **inverse_matrix()** sử dụng thư viện numpy:

```
import numpy as np
def inverse_matrix(matrix):
  # ***********Your code here **********
  return result
```

Question 10: Kết quả của đoạn code sau đây là gì:

```
m1 = np.array([[-2, 6], [8, -4]])
result = inverse_matrix(m1)
print(result)
```

```
a)
[[0.1  0.15]
 [0.2  0.05]]

b)
[[1.1  0.15]
 [0.2  0.05]]

c)
[[0.1  0.15]
 [1.2  0.05]]
```

```
d)
[[2.1  0.15]
 [0.2  0.05]]
```

Bài tập 2: Eigenvector and eigenvalues

2.1 Eigenvector and eigenvalue

- $\mathbf{A} \in R^{n*n}, \mathbf{I}(\text{identity matrix}) \in R^{n*n}, \mathbf{v} \in R^n$
- Eigenvalue (λ): $det(\mathbf{A} - \lambda\mathbf{I}) = 0$
- Eigenvector ($\mathbf{v}$): $\mathbf{Av} = \lambda\mathbf{v} \iff (\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = 0$
- Normalize vector: $\frac{\mathbf{v}}{\| \mathbf{v} \|}, v_i = \frac{v_i}{\sqrt{\Sigma_1^n v_i^2}}$

Cho trước $\mathbf{A} = \begin{bmatrix} 0.9 & 0.2 \\ 0.1 & 0.8 \end{bmatrix}$ Tìm Eigenvector ($\mathbf{v}$) đã được normalize và eigenvalue λ của $\mathbf{A}$ (Trình bày chi tiết theo lược đồ 2.1 (Có thể viết bằng latex sau đó gửi file, hoặc viết bằng markdown trên google colab)?

Dựa vào công thức ở trên, hãy hoàn thiện function **compute_eigenvalues_eigenvectors** sử dụng thư viện numpy:

```
def compute_eigenvalues_eigenvectors(matrix):
 # ***********Your code here **********
 return eigenvalues,eigenvectors
```

Question 11: Kết quả của đoạn code sau đây là gì:

```
matrix = np.array([[0.9, 0.2], [0.1, 0.8]])
eigenvalues, eigenvectors = compute_eigenvalues_eigenvectors(matrix)
print(eigenvectors)
```

```
a)
[[ 0.89442719 -0.70710678]
 [ 0.4472136  0.70710678]]

b)
[[ 1.89442719 -0.70710678]
 [ 0.4472136  0.70710678]]

c)
[[ 2.89442719 -0.70710678]
 [ 0.4472136  0.70710678]]

d)
```

```
[[ 3.89442719 -0.70710678]
 [ 0.4472136  0.70710678]]
```

Bài tập 3: Cosine Similarity

3.1. Cosine Similarity

- Data (vector $\mathbf{x}$, $\mathbf{y}$): $\mathbf{x} = \{x_1, ..., x_N\}$ $\mathbf{y} = \{y_1, ..., y_N\}$
- Cosine Similarity: $cs(\mathbf{x}, \mathbf{y}) = \dfrac{\mathbf{x} \cdot \mathbf{y}}{\| \mathbf{x} \| \| \mathbf{y} \|} = \dfrac{\Sigma_1^n x_i y_i}{\sqrt{\Sigma_1^n x_i^2}\sqrt{\Sigma_1^n y_i^2}}$

Cho trước $\mathbf{x} = \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$ $\mathbf{y} = \begin{bmatrix}1\\0\\3\\0\end{bmatrix}$ Tìm Cosine similarity cs(**x**,**y**) (Trình bày chi tiết theo lược đồ 3.1 (Có thể viết bằng latex sau đó gửi file, hoặc viết bằng markdown trên google colab)
Dựa vào trên quả tính tay ở trên, hãy hoàn thiện function **compute_cosine** sử dụng thư viện numpy:

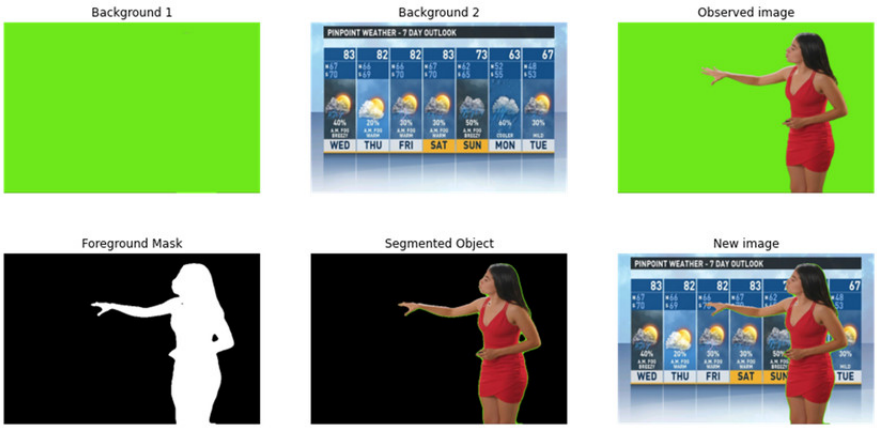
```
def compute_cosine(v1, v2):
  # ***********Your code here **********
  return cos_sim
```

Question 12: Kết quả của đoạn code sau đây là gì:

```
x = np.array([1, 2, 3, 4])
y = np.array([1, 0, 3, 0])
result = compute_cosine(x,y)
print(round(result, 3))
```

```
a) 1.577

b) 2.577

c) 0.577

d) 3.577
```

Bài tập 4 (tuỳ chọn):
Viết chương trình sử dụng kỹ thuật background substraction để trích xuất forgeground (object mong muốn) và dán vào background mới. (Hình 1)
- Input: 3 ảnh Background 1, Background 2, Observed image
- Output: là ảnh mới khi trích xuất object từ Observed image và dán vào Background 2
- Gợi Ý: Đưa cả 3 ảnh về cùng kích thước Dùng background substraction với Observed image và Background 1 để lấy mask object (object ở đây là người MC mặc áo đỏ) Mask object là ảnh binary (Foreground Mask) sẽ gồm 2 giá trị: 0 là background, 1 các vùng pixel chứa object. Tạo



Hình 1: Kết quả trích xuất forgeground (object mong muốn) và dán vào background mới

ra ảnh output (New image) vị trí pixel nào = 1 thì lấy giá trị của Observed image và vị trí nào = 0 thì lấy giá trị của Background 2

Hướng dẫn: để hoàn thành bài tập trên, bạn cần hoàn thiện các functions sau đây (các bạn chú ý chỉnh lại đường dẫn đến ảnh cho phù hợp):

```
import numpy as np
from google.colab.patches import cv2_imshow
import cv2

bg1_image = cv2.imread('background.png', 1)
bg1_image = cv2.resize(bg1_image, (640, 480))

ob_image = cv2.imread('StillImage.png', 1)
ob_image = cv2.resize(ob_image, (640, 480))

bg2_image = cv2.imread('FakeBackground.png')
bg2_image = cv2.resize(bg2_image, (640, 480))

output = replaceBackGround(bg1_image, bg2_image, ob_image)

cv2_imshow(output)
```

Hoàn thiện function **replaceBackGround()**:

```
def replaceBackGround(bg1_image, bg2_image, ob_image):
  difference_single_channel = computeDifference(bg1_image,ob_image)

  binary_mask = computeBinaryMask(difference_single_channel)

  output = np.where(binary_mask==255, ob_image, bg2_image)
  return output
```

Hoàn thiện function **computeDifference()**:

```
def computeDifference(bg_img, input_img):
  # *********your coce here ************************************
  return difference_single_channel
```

Các bạn có thể sử dụng đoạn code bên dưới để hiển thị kết quả (hình 2) của function **computeDifference()** như sau:

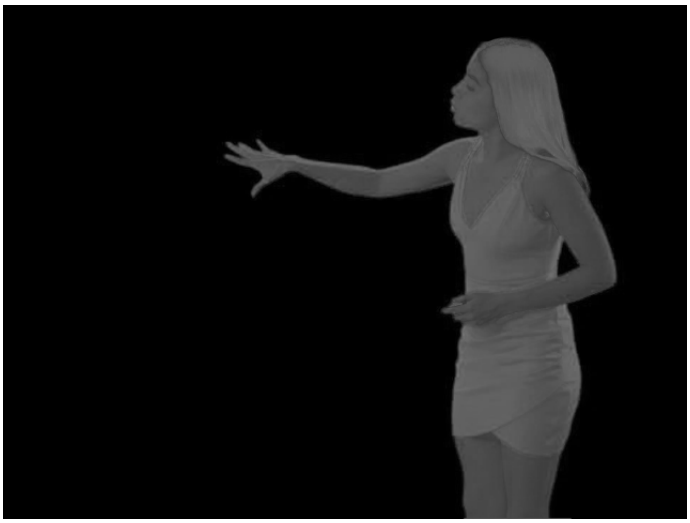
```
bg1_image = cv2.imread('background.png', 1)
bg1_image = cv2.resize(bg1_image, (640, 480))

ob_image = cv2.imread('StillImage.png', 1)
ob_image = cv2.resize(ob_image, (640, 480))

difference_single_channel = computeDifference(bg1_image,ob_image)
cv2_imshow(difference_single_channel)
```

Hoàn thiện function **computeBinaryMask()**:

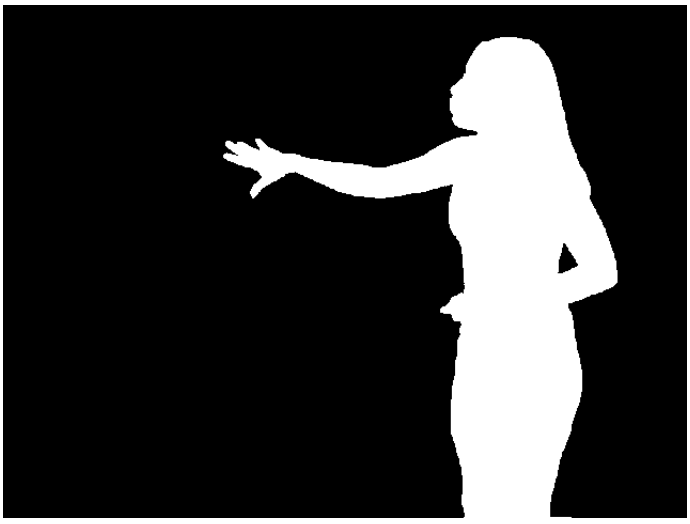
```
def computeBinaryMask(difference_single_channel):
    # *********your coce here ************************************

  return difference_binary
```

Các bạn có thể sử dụng đoạn code bên dưới để hiển thị kết quả (hình 3) của function **computeBinaryMask()** như sau:

```
bg1_image = cv2.imread('background.png', 1)
bg1_image = cv2.resize(bg1_image, (640, 480))

ob_image = cv2.imread('StillImage.png', 1)
ob_image = cv2.resize(ob_image, (640, 480))

difference_single_channel = computeDifference(bg1_image,ob_image)

binary_mask = computeBinaryMask(difference_single_channel)
cv2_imshow(binary_mask)
```



Hình 2: Kết quả của function **computeDifference()**



Hình 3: Kết quả của function **computeBinaryMask()**