03-SPRINT3-T02-EstructuraMATRIU

October 10, 2021

1 S03 T02: Estructura d'una Matriu

Descripció Anem a practicar i a familiaritzar-nos amb l'estructura de Matrius, dimensió, forma, vectorització i Broadcasting .

1.1 Nivell 1

Treballem els conceptes de l'estructura d'una matriu, dimensió, eixos i la vectorització que ens permet reduir l'ús de for loops en operacions aritmètiques o matemàtiques..

- **Exercici 1**
  Crea un np.array d'una dimensió, que inclogui l'almenys 8 nombres sencers, data type int64. Mostra la dimensió i la forma de la matriu. .
- **Exercici 2**
  De la matriu de l'exercici 1, calcula el valor mitjà dels valors introduïts i resta la mitjana resultant de cada un dels valors de la matriu.
- **Exercici 3**
  Crea una matriu bidimensional amb una forma de 5 x 5. Extreu el valor màxim de la matriu, i els valors màxims de cadascun dels seus eixos.

1.2 Nivell 2

Treballem els conceptes de l'estructura d'una matriu, Broadcasting, indexació, Mask..

- **Exercici 4**
  Mostreu-me amb exemples de diferents matrius, la regla fonamental de Broadcasting que diu : "les matrius es poden transmetre / broadcast si les seves dimensions coincideixen o si una de les matrius té una mida d'1".
- **Exercici 5**
  Utilitza la Indexació per extreure els valors d'una columna i una fila de la matriu. I suma els seus valors.
- **Exercici 6**
  Mask la matriu anterior, realitzeu un càlcul booleà vectoritzat, agafant cada element i comprovant si es divideix uniformement per quatre.
  Això retorna una matriu de mask de la mateixa forma amb els resultats elementals del càlcul.
- **Exercici 7**
  A continuació, utilitzeu aquesta màscara per indexar a la matriu de números original. Això

fa que la matriu perdi la seva forma original, reduint-la a una dimensió, però encara obteniu les dades que esteu cercant.

1.3 Nivell 3

Manipulació d'imatges amb Matplotlib.

Carregareu qualsevol imatge (jpg, png ..) amb Matplotlib. adoneu-vos que les imatges RGB (Red, Green, Blue) són realment només amplades × alçades × 3 matrius (tres canals Vermell, Verd i Blau), una per cada color de nombres enters int8,

manipuleu aquests bytes i torneu a utilitzar Matplotlib per desar la imatge modificada un cop hàgiu acabat.

Ajuda:Importeu, import matplotlib.image as mpimg. estudieu el metodde mpimg.imread(()

- **Exercici 8**
  Mostreu-me a veure que passa quan eliminem el canal G Verd o B Blau.
  Mostreu-me a veure què passa quan eliminem el canal G Verd o B Blau. Hauries d'utilitzar la indexació per seleccionar el canal que voleu anul · lar.
  Utilitzar el mètode, mpimg.imsave () de la llibreria importada, per guardar les imatges modificades i que haureu de pujar al vostre repositori a github.

1.4 Level 1

1.4.1 Exercice 1

Creat an np.array of 1 dimension, it should include 8 integers, data type int64. Show dimension and shape of the matrix

1- I import NumPy library to work with matrix
2- then I creat matrix with np.array and specify that the integers are int64

[45]:
```python
import numpy as np

matrixA = np.array([1,2,3,4,5,6,7,8], dtype = np.int64)
print(matrixA)
print(type(matrixA))
type(matrixA[0])
```

```
[1 2 3 4 5 6 7 8]
<class 'numpy.ndarray'>
```

[45]:
```
numpy.int64
```

We can see checking the type of the matrix and it corresponds to numpy.ndarray Similarly, the type of index 0 of the matrix is an int64

To obtain the length of the array or matrix across all dimensions, we use what is known as the array **shape in NumPy**. The shape returns the array size regardless of the number of dimensions.

To check the dimension and shape I will use matrixName.ndim and matrixName.shape

[46]:
```
print(matrixA.ndim)
print(matrixA.shape)
```

```
1
(8,)
```

So, since I did a matrix with just one vector, or row it is 1 dimensions. The vector itsefs has 8 integers, so the shape for the 1 dimension matrix is 8.

1.4.2 Exercice 2

From the matrix you already built in Exercice 1, calculate the avarage value of the values in the matrix and substract the median.

I will use function mean from numpy to calculate the array mean. I also could use statistics module.

[54]:
```
print("Matrix A mean is: ", np.mean(matrixA))
```

```
Matrix A mean is:  4.5
```

The median is the number in the middle [1 2 3 4 5 6 7 8] if we have this vector and since its 8 numbers we will take 4 and 5 and do 4+5/2 which is 4,5 [1 2 3 4 5 6 7] if we had 7 numbers it would be 4

[52]:
```
print(np.median(matrixA))
```

```
4.5
```

The result of exercice 1 is to substract the median to the mean, we do this with a simple mathematical operation

[55]:
```
result = np.mean(matrixA) - np.median(matrixA) #I use both ways of calling
print("The substraction of the median to the mean is: " + str(result))
```

```
The substraction of the median to the mean is: 0.0
```

1.4.3 Exercice 3

Creat a bidimensional matrix with a 5 x 5 shape. Extract the maximum value, and the maximum value of each of its axes.

That means: Bidemensional, 2 dimensions, 2 arrays Shape is the size/lenght of each vector, so each vector should have 5 elements

[62]:
```
matrix5x5 = np.random.randint(1,11,(5,5)) #random integers in the range of 1 to␣
 ↪10 in an array of shape (2,3)
print(matrix5x5)
print(matrix5x5.ndim) #Checking dimension
print(matrix5x5.shape) #checking shape
```

```
[[9 5 1 8 2]
 [8 7 6 8 9]
 [3 5 6 9 8]
 [8 6 8 4 6]
 [8 7 1 6 8]]
2
(5, 5)
```

The matrix5x5 has 2 dimensions and 5x5 as asked.

To extract the maximum value of the matrix we use matrixName.max() function of numpy

[63]:
```
maxValue5x5 = matrix5x5.max()
print(maxValue5x5)
```

```
9
```

to do so with each axis we use the same, indicating the axis we want b.max(axis=0)

[64]:
```
print(matrix5x5.max(axis = 0))
print(matrix5x5.max(0)) #2 different ways of writting it
print(matrix5x5.max(axis = 1))
```

```
[9 7 8 9 9]
[9 7 8 9 9]
[9 9 9 8 8]
```

for **0 axis (horizontal)** we obtain the maximum numbers in the horizontal way in two different ways (we get the same).
for **1 axis (vertical)** we also get the maximum numbers in the vertical way.

1.5 Level 2

We will work the concepts of the matrix structure, broadcasting, indexaton, mask.

1.5.1 Exercice 4

Show with diferent matrix examples the first rule of bradcasting (matrices can be transmeted/broadcasted if their dimensions coincide or if one of the matrices has shape 1.)
- Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading (left) side.
- Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
- Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

Exemple 1:

[88]:
```
M = np.ones((2,3))
a = np.arange(3)
print(M)
print(a)
print(M.shape)
print(a.shape)
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
[0 1 2]
(2, 3)
(3,)
```

- So, M is 2x3 and a 1x3.
- Following rule 1 the shape of the array with fewer dimensions (so a) will be padded with ones in the LEFT SIDE, meaning that shape of a will become (1,3).
  - Now, M is (2,3)
  - and a is (1,3)
- Since the shape of the 2 arrays does not match we follow rule 2.
- In that case, the array with shape equal to 1 (a) is streched in that dimension to match the other shape. Sooo, a shape will be stratched from (1,3) to (2,3) to match the 2 in M shape.

Now we could do any computation between these 2 arrays

```
[89]:
suma = M + a
print(suma)
print(suma.shape)
```

```
[[1. 2. 3.]
 [1. 2. 3.]]
(2, 3)
```

Exemple 2: Here I'll do two arrays with different shapes

```
[101]:
M1 = np.random.randint(1,11,(2,3)) #random integers in the range of 1 to 10 in␣
 ↪an array of shape (2,3)
M2 = np.random.randint(1,11,(2,1)) #shape (2,1)

print(M1.shape)
print(M1)
print(M2.shape)
print(M2)

Msuma = M1 + M2 #since M2
print(Msuma)
```

```
(2, 3)
[[8 2 8]
 [8 6 8]]
(2, 1)
[[7]
 [8]]
[[15  9 15]
 [16 14 16]]
```

We see that since M2 had same dimension but in one of them 1 element. M2 was broadcasted to match M1 so we can sum the matrices

Exemple 3: same dimension, different numbers of elements

[99]:
```
M1 = np.random.randint(1,11,(2,3)) #random integers in the range of 1 to 10 in␣
 ↪an array of shape (2,3)
M2 = np.random.randint(1,11,(2,5)) #shape (2,5)

print(M1.shape)
print(M1)
print(M2.shape)
print(M2)

Msuma = M1 + M2 #we get an error because the shape is different and neither is␣
 ↪equal to 1
print(Msuma)
```

```
(2, 3)
[[ 2 10  5]
 [ 2  4  2]]
(2, 5)
[[9 7 7 1 3]
 [1 7 9 4 3]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-99-7690020390d5> in <module>
      7 print(M2)
      8
----> 9 Msuma = M1 + M2 #we get an error because the shape is different and␣
 ↪neither is equal to 1
     10 print(Msuma)

ValueError: operands could not be broadcast together with shapes (2,3) (2,5)
```

The valueError explains the bradcasting problem, the shape are different and non of them is equal to 1, it cannot be broadcasted

1.5.2 Exercice 5

Use indexing to extract a row and a column of a matrix and sum their values.

[66]:
```
M = np.random.randint(1,101, size = (10, 10)) #creat a matrix with random␣
 ↪integers between 1 and 100 shape 10x10
print(M)
print(M.shape)
```

```
[[25 86 68 62 57 89 61 23 32  9]
 [48 56  5 81  7 65 65 54 65 34]
 [72  7 95 53 19  9 93 94 17 76]
 [53 12 73 97 70 52 43 59 40 42]
 [54 60 45 71 91 10  4 31 91 94]
 [85 71 12 70 48 15 28 56 56 43]
 [96 21 42 94 67 25 11 74 57  8]
 [69 44 42 55 67 75 86 65 13 12]
 [16 48 23 20 16 17 99 55 81 80]
 [ 4 64 14 35 10 88 70 35 18 81]]
(10, 10)
```

Extracting a row and a column

```
[67]: row9 = M[8, :] #row 9
column4 = M[:,3] #column 4
lastcolumn = M[:,-1] #column 10 (11-1)
print(row9)
print(column4)
print(lastcolumn)
```

```
[16 48 23 20 16 17 99 55 81 80]
[62 81 53 97 71 70 94 55 20 35]
[ 9 34 76 42 94 43  8 12 80 81]
```

Since I am not sure what they refer to, I will sum all elements of the row and columns I extract. Then I will sum the elements of 1 row and 1 column

```
[70]: sumRow9 = sum(M[8,:]) #sum() does the summation af the array you add inside, in␣
 ↪this case column 9
print(sumRow9)

sumR9C4 = row9 + column4
print(sumR9C4)
```

```
455
[ 78 129  76 117  87  87 193 110 101 115]
```

SumRow9 is the sumation of the elements of each row or column.
sumR9C4 is the sumation element by element of row 9 and column4

1.6 Exercici 6

Creat a mask of the previous matrix al do a vectorized boolean calculus taking each element and checking if it's modulus divided by 4 is 0.
This will return a masked matrix with the same shape with the elemental results of the calculus

```
[71]: print(M)
```

```
[[25 86 68 62 57 89 61 23 32  9]
 [48 56  5 81  7 65 65 54 65 34]
 [72  7 95 53 19  9 93 94 17 76]
 [53 12 73 97 70 52 43 59 40 42]
 [54 60 45 71 91 10  4 31 91 94]
 [85 71 12 70 48 15 28 56 56 43]
 [96 21 42 94 67 25 11 74 57  8]
 [69 44 42 55 67 75 86 65 13 12]
 [16 48 23 20 16 17 99 55 81 80]
 [ 4 64 14 35 10 88 70 35 18 81]]
```

[72]:
```
print(M%4 ==  0 )
Mmodulus4Exact = M%4 ==  0
print(Mmodulus4Exact)
```

```
[[False False  True False False False False False  True False]
 [ True  True False False False False False False False False]
 [ True False False False False False False False False  True]
 [False  True False False False  True False False  True False]
 [False  True False False False False  True False False False]
 [False False  True False  True False  True  True  True False]
 [ True False False False False False False False False  True]
 [False  True False False False False False False False  True]
 [ True  True False  True  True False False False False  True]
 [ True  True False False False  True False False False False]]
[[False False  True False False False False False  True False]
 [ True  True False False False False False False False False]
 [ True False False False False False False False False  True]
 [False  True False False False  True False False  True False]
 [False  True False False False False  True False False False]
 [False False  True False  True False  True  True  True False]
 [ True False False False False False False False False  True]
 [False  True False False False False False False False  True]
 [ True  True False  True  True False False False False  True]
 [ True  True False False False  True False False False False]]
```

1.7 Exercici 7

Next, use the masc tin indexate in the original matrix. This will make the matrix to loose its original shape, changing it to 1 dimension, but will give you the data you are looking for.

Now to select these values from the array, we can simply index on this Boolean array; this is known as a masking operation

[73]:
```
maskedM =  M[M%4 ==  0] #this is creating the mask with all the elements that␣
 ↪satisfy the condition
print(maskedM)
```

```
[68 32 48 56 72 76 12 52 40 60  4 12 48 28 56 56 96  8 44 12 16 48 20 16
 80  4 64 88]
```

What is returned is a one-dimensional array filled with all the values that meet this condition; in other words, all the values in positions at which the mask array is True.

1.8 Level 3

Image manipulation with Matplotlib

Upload any image(jpg, png,...)with Matplotlib. Be awar that RGB imatges are just heights x widths x 3 matrixs (3 channels red, green and blue), one for each color of integers int8.

manipulate this bytes and use again matplotlib to save the modify images ones finished.

Help: import, import matplotlib.imgage as mimg. And study the methode mpimg.imread()

Exercice 8

Show what is happening when g or B channels are eliminated Use the indexation to choose the channel that you want to get nul.

Use the method, mpimg.imsave() from the imported library, to save the modified images and that you will need to upload to your github repository.

First we import all the necessari libraries

[ ]:
```
import numpy as np
from IPython.display import Image
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

We print the selected image

[30]:
```
Image('IMG_2745.jpg')
```

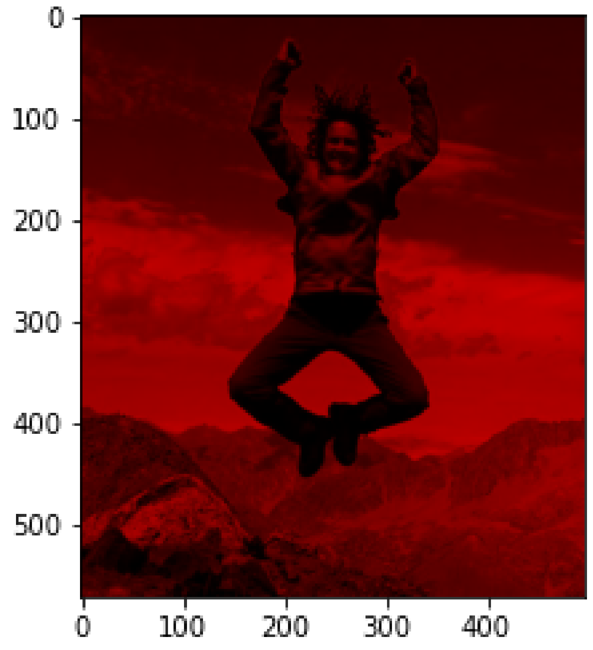
[30]:

```
[32]: img =  mpimg.imread('IMG_2745.jpg') # with imread module I get an array of the␣
       ↪image
      print(img) #I print it to
      print(img.shape)
```

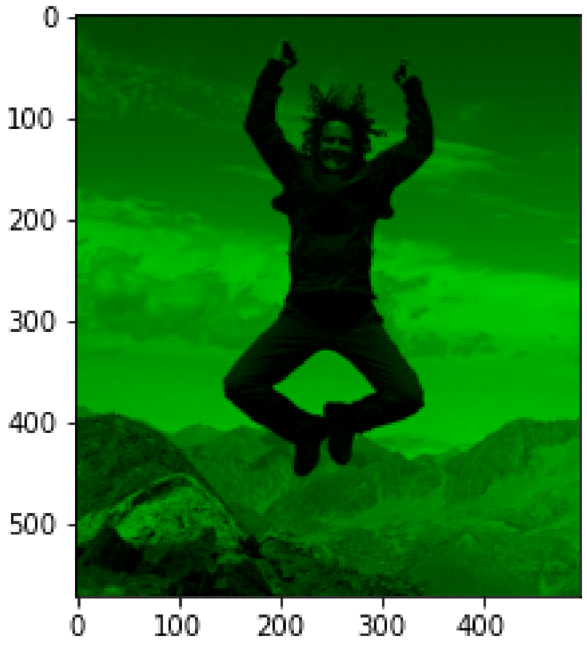
```
[[[ 53  69 129]
  [ 53  69 129]
  [ 53  69 128]
  …
  [ 51  71 124]
  [ 51  71 124]
  [ 51  71 124]]

 [[ 53  69 129]
  [ 53  69 129]
  [ 53  69 128]
  …
  [ 51  71 124]
  [ 51  71 124]
  [ 51  71 124]]

 [[ 53  69 129]
  [ 53  69 129]
  [ 53  69 128]
  …
  [ 51  70 126]
  [ 51  70 126]
  [ 51  70 126]]

 …

 [[ 24  15  16]
  [ 69  60  61]
  [ 71  65  65]
  …
  [ 88  89  93]
  [ 63  64  68]
  [ 46  47  51]]

 [[ 33  24  25]
  [ 68  59  60]
  [ 62  56  56]
  …
  [ 98  99 103]
  [ 72  73  77]
  [ 80  81  85]]

 [[ 53  44  47]
  [ 78  69  72]
  [ 62  56  58]
  …
  [102 103 107]
  [ 86  87  91]
  [105 106 110]]]
(572, 496, 3)
```

We see the image is a 3D array with shape (572,492,3) (height, width, channels) To be able to plot later the image in the selected channels, we first do an array, the same size as the image array. Secondly, we put in the same location of the 3D array created of 0's, so third place, the red channel by indexing the red channel.

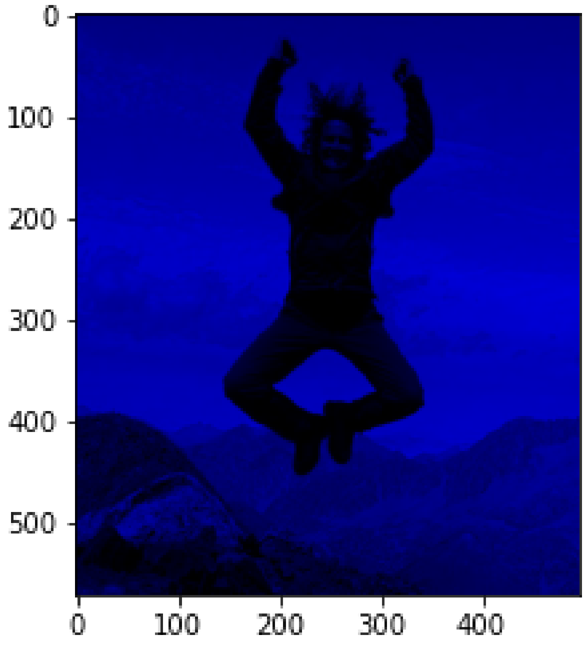
[36]:
```
R = np.zeros(img.shape, np.uint8)    # create array of 0-s with same dimensions␣
 ↪as image
R[:,:,0]=img[:,:,0]           # copy red channel values into array of 0-s
plt.imshow(R)
plt.show()     # display new image
```



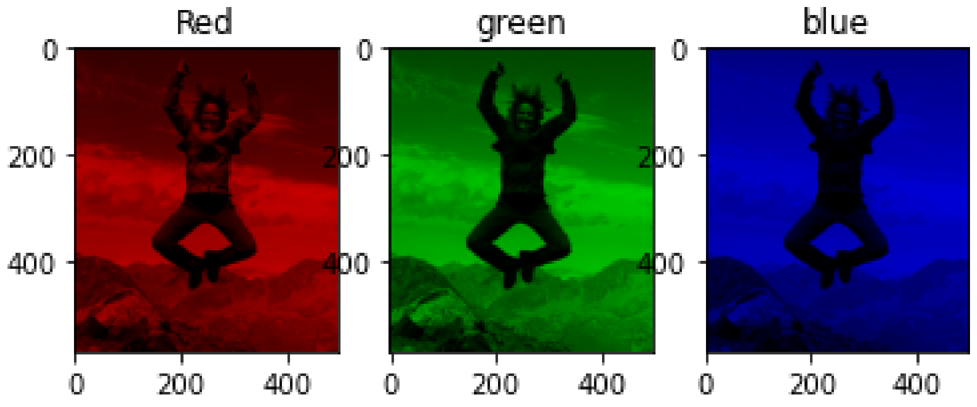
[37]:
```
G = np.zeros(img.shape, np.uint8)    # create array of 0-s with same dimensions␣
 ↪as image
G[:,:,1]=img[:,:,1]           # copy red channel values into array of 0-s
plt.imshow(G)
plt.show()     # display new image
```



```
[38]: B = np.zeros(img.shape, np.uint8)     # create array of 0-s with same dimensions␣
       ↪as image
      B[:,:,2]=img[:,:,2]             # copy red channel values into array of 0-s
      plt.imshow(B)
      plt.show()      # display new image
```



[75]:
```
fig = plt.figure()
ax = fig.add_subplot(1, 3, 1)#Three integers (nrows, ncols, index).
imgplot = plt.imshow(R)
ax.set_title('Red')
#plt.colorbar(ticks=[0.1, 0.3, 0.5, 0.7], orientation='horizontal')
ax = fig.add_subplot(1, 3, 2)
imgplot = plt.imshow(G)
#imgplot.set_clim(0.0, 0.7)
ax.set_title('green')
#plt.colorbar(ticks=[0.1, 0.3, 0.5, 0.7], orientation='horizontal')
ax = fig.add_subplot(1, 3, 3)
imgplot = plt.imshow(B)
imgplot.set_clim(0.0, 0.7)
ax.set_title('blue')
#plt.colorbar(ticks=[0.1, 0.3, 0.5, 0.7], orientation='horizontal')

plt.savefig('Infiernos_R_G_B.jpg')
```



[ ]: