Bootcamp Python

Bootcamp Python

Day03 - NumPy

Today you will learn how to use the Python library that will allow you to manipulate multidimensional arrays (vectors, matrices, tensors...) and perform complex mathematical operations on them.

Notions of the day

NumPy array, slicing, stacking, dimensions, broadcasting, normalization, etc...

General rules

- Use the NumPy Library: use NumPy's built-in functions as much as possible. Here you will be given no credit for reinventing the wheel.
- The version of Python to use is 3.7, you can check the version of Python with the following command: python -V
- The norm: during this bootcamp you will follow the Pep8 standards https://www.python.org/dev/peps/pep-0008/
- The function eval is never allowed.
- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- You can also ask questions in the dedicated channel in the 42 AI Slack: 42-ai.slack.com.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on Github: https://github.com/42-AI/bootcamp_python/issues.

Helper

For this day you will use the image provided in the `resources` folder

Ensure that you have the right Python version.

```
> which python
/goinfre/miniconda/bin/python
> python -V
Python 3.7.*
> which pip
/goinfre/miniconda/bin/pip
```

Exercise 00 - NumPyCreator

Exercise 01 - ImageProcessor

Exercise 02 - Basic manipulations

Exercise 03 - Color filters

Exercise 04 - Advanced modifications

Exercise 00 - NumPyCreator

Turn-in directory :	ex00
Files to turn in :	NumPyCreator.py
Allowed libraries :	numpy
Remarks :	n/a

You need to write a class named NumPyCreator, which will implement all of the following methods.

Each method receives as an argument a different type of data structure and transforms it into a NumPy array:

- from_list(lst) : takes in a list and returns its corresponding NumPy array.
- from_tuple(tpl) : takes in a tuple and returns its corresponding NumPy array.
- from_iterable(itr) : takes in an iterable and returns an array which contains all of its elements.
- from_shape(shape, value) : returns an array filled with the same value.

The first argument is a tuple which specifies the shape of the array, and the second argument specifies the value of all the elements. This value must be 0 by default.

- random(shape) : returns an array filled with random values.

It takes as an argument a tuple which specifies the shape of the array.

- identity(n) : returns an array representing the identity matrix of size n.

BONUS : Add to those methods an optional argument which specifies the datatype (dtype) of the array (e.g. if you want its elements to be represented as integers, floats, ...)

NOTE : All those methods can be implemented in one line. You only need to find the right NumPy functions.

```
>>> from NumPyCreator import NumPyCreator
>>> npc = NumPyCreator()

>>> npc.from_list([[1,2,3],[6,3,4]])
array([[1, 2, 3],
       [6, 3, 4]])

>>> npc.from_tuple(("a", "b", "c"))
array(['a', 'b', 'c'])

>>> npc.from_iterable(range(5))
array([0, 1, 2, 3, 4])

>>> shape=(3,5)
>>> npc.from_shape(shape)
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])

>>> npc.random(shape)
array([[0.57055863, 0.23519999, 0.56209311, 0.79231567, 0.213768 ],
       [0.39608366, 0.18632147, 0.80054602, 0.44905766, 0.81313615],
       [0.79585328, 0.00660962, 0.92910958, 0.9905421 , 0.05244791]])

>>> npc.identity(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Exercise 01 - ImageProcessor

Turn-in directory :	ex01
Files to turn in :	ImageProcessor.py
Forbidden functions :	None
Helpful libraries :	Matplotlib

Now you will build a tool to load and display images in the upcoming exercises.

Write a class named ImageProcessor which implements the following methods:

- **`load(path)`** : opens the .png file specified by the **`path`** argument and returns an array with the RGB values of the image pixels.
It must display a message specifying the dimensions of the image (e.g. 340 x 500).
- **`display(array)`** : takes a NumPy array as an argument and displays the corresponding RGB image.

`NOTE` : You can use the library of your choice for this exercise, but converting the image to a NumPy array is mandatory. The goal of this exercise is to dispense with the technicality of loading and displaying images, so that you can focus on array manipulation in the upcoming exercises.

```
>>> from ImageProcessor import ImageProcessor
>>> imp = ImageProcessor()
>>> arr = imp.load("../resources/42AI.png")
Loading image of dimensions 200 x 200
>>> arr
array([[[0.03529412, 0.12156863, 0.3137255 ],
        [0.03921569, 0.1254902 , 0.31764707],
        [0.04313726, 0.12941177, 0.3254902 ],
        ...,
        [0.02745098, 0.07450981, 0.22745098],
        [0.02745098, 0.07450981, 0.22745098],
        [0.02352941, 0.07058824, 0.22352941]],

       [[0.03921569, 0.11764706, 0.30588236],
        [0.03529412, 0.11764706, 0.30980393],
        [0.03921569, 0.12156863, 0.30980393],
        ...,
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098]],

       [[0.03137255, 0.10980392, 0.2901961 ],
        [0.03137255, 0.11372549, 0.29803923],
        [0.03529412, 0.11764706, 0.30588236],
        ...,
        [0.02745098, 0.07450981, 0.23137255],
        [0.02352941, 0.07450981, 0.22745098],
        [0.02352941, 0.07450981, 0.22745098]],

       ...,

       [[0.03137255, 0.07450981, 0.21960784],
        [0.03137255, 0.07058824, 0.21568628],
        [0.03137255, 0.07058824, 0.21960784],
        ...,
        [0.03921569, 0.10980392, 0.2784314 ],
        [0.03921569, 0.10980392, 0.27450982],
        [0.03921569, 0.10980392, 0.27450982]],

       [[0.03137255, 0.07058824, 0.21960784],
        [0.03137255, 0.07058824, 0.21568628],
        [0.03137255, 0.07058824, 0.21568628],
        ...,
        [0.03921569, 0.10588235, 0.27058825],
        [0.03921569, 0.10588235, 0.27058825],
        [0.03921569, 0.10588235, 0.27058825]],

       [[0.03137255, 0.07058824, 0.21960784],
        [0.03137255, 0.07058824, 0.21176471],
        [0.03137255, 0.07058824, 0.21568628],
        ...,
        [0.03921569, 0.10588235, 0.26666668],
        [0.03921569, 0.10588235, 0.26666668],
        [0.03921569, 0.10588235, 0.26666668]]], dtype=float32)
>>> imp.display(arr)
```

Exercise 02 - ScrapBooker

Turn-in directory :	ex02
Files to turn in :	ScrapBooker.py
Allowed libraries :	NumPy
Notions :	Slicing

Write a class named **`ScrapBooker`** which implements the following methods.
All methods take in a NumPy array and return a new modified one.
We are assuming that all inputs are correct, ie, you don't have to protect your functions against input errors.

- **`crop(array, dimensions, position)`** : crop the image as a rectangle with the given **`dimensions`** (meaning, the new height and width for the image), whose top left corner is given by the **`position`** argument. The position should be (0,0) by default. You have to consider it an error (and handle said error) if **`dimensions`** is larger than the current image size.
- **`thin(array, n, axis)`** : delete every n-th pixel row along the specified axis (0 vertical, 1 horizontal), example below.
- **`juxtapose(array, n, axis)`** : juxtapose **`n`** copies of the image along the specified axis (0 vertical, 1 horizontal).
- **`mosaic(array, dimensions)`** : make a grid with multiple copies of the array. The **`dimensions`** argument specifies the dimensions (meaning the height and width) of the grid (e.g. 2x3).

`NOTE` : In this exercise, when specifying positions or dimensions, we will assume that the first coordinate is counted along the vertical axis starting from the TOP, and that the second coordinate is counted along the horizontal axis starting from the left. Indexing starts from 0.

e.g.:

(1,3)

.....

...x.

.....

example for thin:

```
permorm thin with n=3 and axis=0:
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL ==> ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ
ABCDEFGHIJQL        ABDEGHJQ

permorm thin with n=4 and axis=1:
AAAAAAAAAAA
BBBBBBBBBBB         AAAAAAAAAAA
CCCCCCCCCCC         BBBBBBBBBBB
DDDDDDDDDDD         CCCCCCCCCCC
EEEEEEEEEEE         EEEEEEEEEEE
FFFFFFFFFFF ==> FFFFFFFFFFF
GGGGGGGGGGG         GGGGGGGGGGG
HHHHHHHHHHH         IIIIIIIIIII
IIIIIIIIIII         JJJJJJJJJJJ
JJJJJJJJJJJ         KKKKKKKKKKK
KKKKKKKKKKK
LLLLLLLLLLL
```

Exercise 03 - ColorFilter

Turn-in directory :	ex03
Files to turn in :	ColorFilter.py
Forbidden functions :	See each method
Notions :	broadcasting

Now you will build a tool that can apply a variety of color filters on images.
For this exercise, the authorized functions and operators are specified for each methods. You are not allowed to use anything else.

Write a class named `ColorFilter` which implements the following methods:

- `invert(array)` : Takes a NumPy array of an image as an argument and returns an array with inverted color.
Authorized function : None
Authorized operator: -
- `to_blue(array)` : Takes a NumPy array of an image as an argument and returns an array with a blue filter.
Authorized function : .zeros, .shape
Authorized operator: None
- `to_green(array)` : Takes a NumPy array of an image as an argument and returns an array with a green filter.
Authorized function : None
Authorized operator: *
- `to_red(array)` : Takes a NumPy array of an image as an argument and returns an array with a red filter.
Authorized function : green, blue
Authorized operator: -, +
- `celluloid(array)` : Takes a NumPy array of an image as an argument, and returns an array with a celluloid shade filter. The celluloid filter must display at least four thresholds of shades. Be careful! You are not asked to apply black contour on the object here (you will have to, but later...), you only have to work on the shades of your images.
Authorized function: arange, linspace

`Bonus`: add an argument to your method to let the user choose the number of thresholds.

Authorized function : .vectorize, (.arange?)

Authorized operator: None

• to_grayscale(array, filter) : Takes a NumPy array of an image as an argument and returns an array in grayscale. The method takes another argument to select between two possible grayscale filters. Each filter has specific authorized functions and operators.
'mean' or 'm' : Takes a NumPy array of an image as an argument and returns an array in grayscale created from the mean of the RBG channels.
Authorized function : .sum, .shape, reshape, broadcast_to, (as_type?)
Authorized operator: /'weighted' or 'w' : Takes a NumPy array of an image as an argument and returns an array in weighted grayscale. This argument should be selected by default if not given.
The usual weighted grayscale is calculated as : 0.299 * R_channel + 0.587 * G_channel + 0.114 * B_channel.
Authorized function : .sum, .shape, .tile
Authorized operator: *

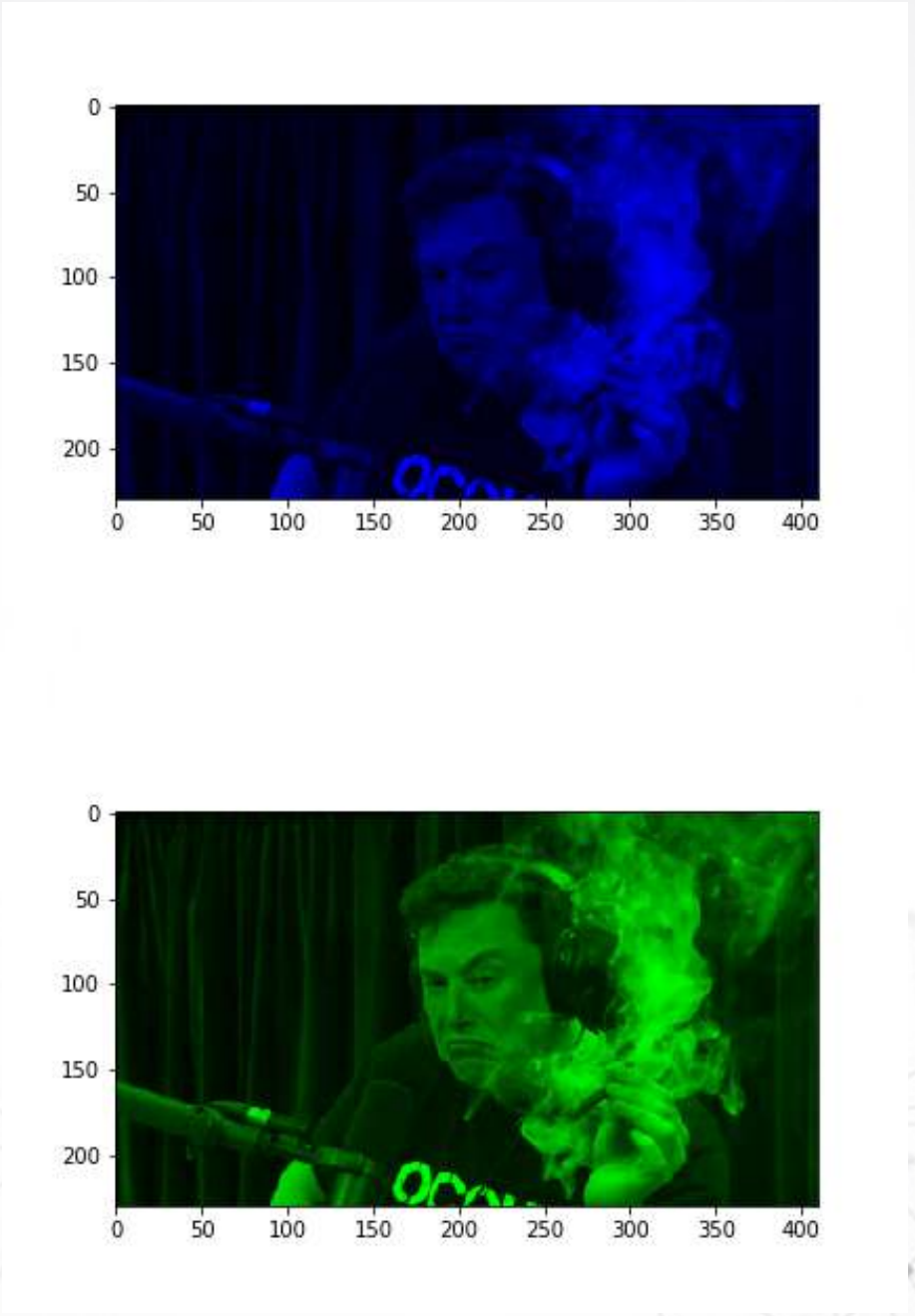
```
>>> from ImageProcessor import ImageProcessor
>>> imp = ImageProcessor()
>>> arr = imp.load("../42AI.png")
Loading image of dimensions 200 x 200
>>> from ColorFilter import ColorFilter
>>> cf = ColorFilter()
>>> cf.invert(arr)
>>>
>>> cf.to_green(arr)
>>>
>>> cf.to_red(arr)
>>>
>>> cf.to_blue(arr)
>>>
>>> cf.to_celluloid(arr)
>>>
>>> cf.to_grayscale(arr, 'm')
>>>
>>> cf.to_grayscale(arr, 'weigthed')
>>>
```

Examples:

From this base image:

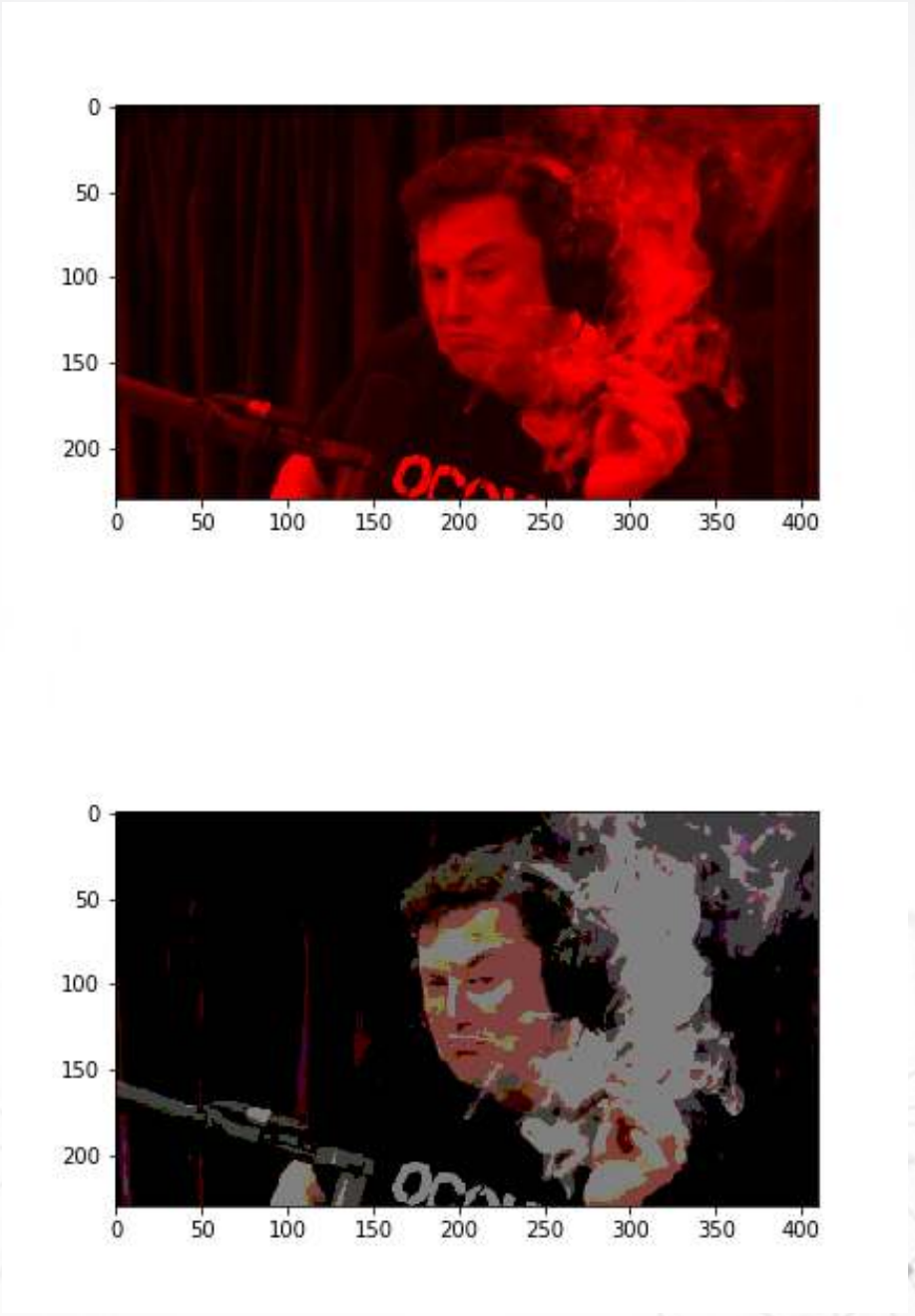
- invert:

- to_blue:



- to_green:

- to_red:


0
50
100
150
200
0
50
100
150
200
250
300
350
400

- celluloid:

0
50
100
150
200
0
50
100
150
200
250
300
350
400

Exercise 04 - AdvancedFilter

Turn-in directory :	ex04
Files to turn in :	AdvancedFilter.py
Allowed libraries :	NumPy
Notions :	Slicing, matrix operations

Write a class named `AdvancedFilter` which implements the following methods.

All methods take in a 3D NumPy array (as in, a tensor of rank 3) and return a modified copy of the array.

The following video should be used as a resource for completing the exercise:

https://www.youtube.com/watch?v=C_zFhWdM4ic

- `mean_blur()` : This method receives an image, performs a mean blur on it and returns a blurred copy. In a mean blur, each pixel becomes the average of its neighboring pixels.
- `gaussian_blur()` : This method receives an image, performs a gaussian blur on it and returns a blurred copy. In a gaussian blur, the weighting of the neighboring pixels is adjusted so that closer pixels are more heavily counted in the average.

`BONUS` : You can add an optional argument to those methods to choose the kernel size.

Remember, you can add helper methods to your class!