Full Name: Çağdaş Güven

Student ID: 2738938

Try to Tell Apples, Oranges and else apart

Images for this assignment are provided at github and they will be automatically downloaded right after imports. When the files are unzip there should be a folder: `AppleOrange` . These are sample images, you can test with more if you want. While testing just for fun, I can try other stuff as well.

Your objective is to convert **Apples to magenta** , **Oranges to Blue** , end blur everything else! While changing color, try to keep the original shading (i.e. try not provide a flat single color if possible, keep the shadings to the best you can).

Do not be **too picky** and lose too much time. Variety of images are provided so that you get the idea that generalizable / perfect filters are not easy to build. Yet your function is expected to work on more than one image at an acceptable level.

By the same token, you can use the `fakes` , also for for fun to see how your algorighm works on unrealted images, and why a general filter is not that easy...

At the end as usual you are expected to **clear all outputs** and then save this file as **Week10_student_id.ipynb** and upload to the assignment at ODTU Class.

imports as usual

You are only allowed to use concepts related to what we have seen in class and use only the following imports. You can import sub-libraries with new names etc. but NO NEW LIBRARIES

In [1]:
```
# not that all of them are necessary, but you are not allowed to import a
# yet as before, you can import sub libraries: i.e.:
#       from skimage import measure
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as pimg # check this out this is new
from numpy import cos, arccos, sin, pi, round
from numpy.linalg import matrix_rank as rank
from numpy.linalg import svd, eig
from scipy.linalg import orth
import cv2 as cv
from PIL import Image # good old pillow
import sklearn as skl # famous sci-kit learn
import skimage as ski # equally famous sci-kit image
!rm bug_numpy_utils.py 2>/dev/null  # at the first run file does not exit
```

```
!wget https://raw.githubusercontent.com/bugrakoku/bug_python_utils/main/b
from bug_numpy_utils import MatPrint, CData, text2mat # note that once th
!rm me536utils.py 2>/dev/null  # at the first run file does not exits but
!wget https://raw.githubusercontent.com/bugrakoku/bug_python_utils/main/m
from me536utils import RotMat
```

```
--2024-12-15 15:19:12--  https://raw.githubusercontent.com/bugrakoku/bug_p
ython_utils/main/bug_numpy_utils.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199
.108.133, 185.199.109.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18456 (18K) [text/plain]
Saving to: ‘bug_numpy_utils.py’

bug_numpy_utils.py  100%[===================>]  18.02K  --.-KB/s    in 0.0
09s

2024-12-15 15:19:13 (1.90 MB/s) - ‘bug_numpy_utils.py’ saved [18456/18456]

--2024-12-15 15:19:13--  https://raw.githubusercontent.com/bugrakoku/bug_p
ython_utils/main/me536utils.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199
.110.133, 185.199.109.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.19
9.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3130 (3.1K) [text/plain]
Saving to: ‘me536utils.py’

me536utils.py       100%[===================>]   3.06K  --.-KB/s    in 0.0
01s

2024-12-15 15:19:14 (5.48 MB/s) - ‘me536utils.py’ saved [3130/3130]
```

get images

In [2]:
```
!rm AppleOrange.zip 2>/dev/null # just in case
!wget https://github.com/bugrakoku/data4all/raw/main/AppleOrange.zip # ge
!unzip AppleOrange.zip
```

```
--2024-12-15 15:19:14--  https://github.com/bugrakoku/data4all/raw/main/AppleOrange.zip
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/bugrakoku/data4all/main/AppleOrange.zip [following]
--2024-12-15 15:19:14--  https://raw.githubusercontent.com/bugrakoku/data4all/main/AppleOrange.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2938722 (2.8M) [application/zip]
Saving to: ‘AppleOrange.zip’

AppleOrange.zip     100%[===================>]   2.80M  2.64MB/s    in 1.1s

2024-12-15 15:19:16 (2.64 MB/s) - ‘AppleOrange.zip’ saved [2938722/2938722]

Archive:  AppleOrange.zip
   creating: AppleOrange/
  inflating: AppleOrange/AorO18.jpg
  inflating: __MACOSX/AppleOrange/._AorO18.jpg
  inflating: AppleOrange/.DS_Store
  inflating: __MACOSX/AppleOrange/._.DS_Store
  inflating: AppleOrange/test1.jpg
  inflating: __MACOSX/AppleOrange/._test1.jpg
  inflating: AppleOrange/AorO8.jpg
  inflating: __MACOSX/AppleOrange/._AorO8.jpg
  inflating: AppleOrange/test3.jpg
  inflating: __MACOSX/AppleOrange/._test3.jpg
  inflating: AppleOrange/test2.jpg
  inflating: __MACOSX/AppleOrange/._test2.jpg
  inflating: AppleOrange/AorO9.jpg
  inflating: __MACOSX/AppleOrange/._AorO9.jpg
  inflating: AppleOrange/AorO2.jpeg
  inflating: __MACOSX/AppleOrange/._AorO2.jpeg
  inflating: AppleOrange/AorO4.jpg
  inflating: __MACOSX/AppleOrange/._AorO4.jpg
  inflating: AppleOrange/AorO5.jpg
  inflating: __MACOSX/AppleOrange/._AorO5.jpg
  inflating: AppleOrange/AorO7.jpg
  inflating: __MACOSX/AppleOrange/._AorO7.jpg
  inflating: AppleOrange/AorO6.jpg
  inflating: __MACOSX/AppleOrange/._AorO6.jpg
  inflating: AppleOrange/AorO3.png
  inflating: __MACOSX/AppleOrange/._AorO3.png
  inflating: AppleOrange/AorO1.jpg
  inflating: __MACOSX/AppleOrange/._AorO1.jpg
  inflating: AppleOrange/AorO10.jpg
  inflating: __MACOSX/AppleOrange/._AorO10.jpg
  inflating: AppleOrange/AorO11.jpg
  inflating: __MACOSX/AppleOrange/._AorO11.jpg
  inflating: AppleOrange/AorO13.jpg
  inflating: __MACOSX/AppleOrange/._AorO13.jpg
  inflating: AppleOrange/AorO12.jpg
```

```
 inflating: __MACOSX/AppleOrange/._AorO12.jpg
 inflating: AppleOrange/AorO16.jpg
 inflating: __MACOSX/AppleOrange/._AorO16.jpg
 inflating: AppleOrange/AorO17.jpg
 inflating: __MACOSX/AppleOrange/._AorO17.jpg
 inflating: AppleOrange/AorO15.jpg
 inflating: __MACOSX/AppleOrange/._AorO15.jpg
 inflating: AppleOrange/AorO14.jpg
 inflating: __MACOSX/AppleOrange/._AorO14.jpg
```

Get prepared

Below you can check images, perform tests runs, find critical color values etc.
Leave all your preparation code so that I can see how you have reached the final implementation of the function. This part I will NOT run for evaluation! I will only run the `AorO()` and `AorO2()` functions in my tests!

You can add as many code and text cells as you like below. But at the end, as I said above, I will just call the `AorO()` or `AorO2()` function.
And please **DO CLEAR ALL OUTPUTS**!

Recall that you are not restricted to R G B at all!

Part I: Manual threshold determination.

You can determine as many thresholds as you like.

Add explanations of what you do and why regarding the code(s) below

In [3]:
```
# your prep code
```

In [4]:
```
# potentially more code
```

`AorO()` function

In [5]:
```
'''
You are to properly implement the following function so that
Apples become magenta, orange blue in color, and rest is blurred where bl

'''

def AorO(img):
    '''
    this function takes the path of an image, loads the image
    converts the color of apples to magenta, color of oranges to blue and
    you are to use some hand crafted thresholds here
    returns the modified image
    '''
```

```
    # following is just a stub, you can totally get rid of it!
    oi = pimg.imread(img) # note that i is a numpy array
    # do your magic below




    i = oi # at the end delete this line and return the proper 'i' that h
    # finally return the process image
    return i
```

Part II: Use some clustering methods to find thresholds automatically

Assuming that there are apples and oranges in the image, analyze the colors in the image (such as histograms, or anything else you find fit), run some clustering algorithms on them and generate the image.

Add explanations of what you do and why regarding the code(s) below

In [6]:
```
# your prep code
```

In [7]:
```
# potentially more code
```

AorO2() function

In [8]:
```
'''
You are to properly implement the following function so that
Apples become magenta, orange blue in color, and rest is blurred where bl

'''

def AorO2(img):
    '''
    this function takes the path of an image, loads the image
    converts the color of apples to magenta, color of oranges to blue and
    you are to use some clustering methods to find thresholds here
    returns the modified image
    '''

    # following is just a stub, you can totally get rid of it!
    oi = pimg.imread(img) # note that i is a numpy array
    # do your magic below




    i = oi # at the end delete this line and return the proper 'i' that h
    # finally return the process image
    return i
```

My test will be performed below

Using my own code either at the end of this file or separately on my computer, I will just call the `AorO()` or ```AorO2() functions in different ways and enjoy the outcomes :)