HW0

October 5, 2018

1 CSE 252A Computer Vision I Fall 2018 - Assignment 0

1.0.1 Instructor: David Kriegman

1.0.2 Assignment Published On: Tuesday, October 2, 2018

1.0.3 Due On: Tuesday, October 9, 2018 11:59 pm

1.1 Instructions

- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- All solutions must be written in this notebook
- Programming aspects of this assignment must be completed using Python in this notebook.
- If you want to modify the skeleton code, you can do so. This has been provided just to provide you with a framework for the solution.
- You may use python packages for basic linear algebra (you can use numpy or scipy for basic operations), but you may not use packages that directly solve the problem.
- If you are unsure about using a specific package or function, then ask the instructor and teaching assistants for clarification.
- You must submit this notebook exported as a pdf. You must also submit this notebook as .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- It is highly recommended that you begin working on this assignment early.
-

1.2 Late policy - 10% per day late penalty after due date.

Welcome to CSE252A Computer Vision I! This course gives you a comprehensive introduction to computer vison providing broad coverage including low level vision, inferring 3D properties from images, and object recognition. We will be using a variety of tools in this class that will require some initial configuration. To ensure smooth progress, we will setup the majority of the tools to be used in this course in this assignment. You will also practice some basic image manipulation techniques. Finally, you will need to export this Ipython notebook as pdf and submit it to Gradescope along with .ipynb file before the due date.

1.2.1 Piazza, Gradescope and Python

Piazza

Go to Piazza and sign up for the class using your ucsd.edu email account. You'll be able to ask the professor, the TAs and your classmates questions on Piazza. Class announcements will be made using Piazza, so make sure you check your email or Piazza frequently.

Gradescope

Every student will get an email regarding gradescope signup once enrolled in this class. All the assignments are required to be submitted to gradescope for grading. Make sure that you mark each page for different problems.

Python

We will use the Python programming language for all assignments in this course, with a few popular libraries (numpy, matplotlib). Assignments will be given in the format of browser-based Jupyter/Ipython notebook that you are currently viewing. We expect that many of you have some experience with Python and Numpy. And if you have previous knowledge in Matlab, check out the numpy for Matlab users page. The section below will serve as a quick introduction to Numpy and some other libraries.

1.3 Getting started with Numpy

Numpy is the fundamental package for scientific computing with Python. It provides a powerful N-dimensional array object and functions for working with these arrays.

1.3.1 Arrays

In [2]:
```
import numpy as np

v = np.array([1, 0, 0])       # a 1d array
print("1d array")
print(v)
print(v.shape)                # print the size of v
v = np.array([[1], [2], [3]]) # a 2d array
print("\n2d array")
print(v)
print(v.shape)                # print the size of v, notice the difference
v = v.T                       # transpose of a 2d array

m = np.zeros([2, 3])          # a 2x3 array of zeros
v = np.ones([1, 3])           # a 1x3 array of ones
```

```
        m = np.eye(3)                     # identity matrix
        v = np.random.rand(3, 1)          # random matrix with values in [0, 1]
        m = np.ones(v.shape) * 3          # create a matrix from shape
```

```
1d array
[1 0 0]
(3,)

2d array
[[1]
 [2]
 [3]]
(3, 1)
```

1.3.2 Array indexing

```
In [3]: import numpy as np

        m = np.array([[1, 2, 3], [4, 5, 6]])  # create a 2d array with shape (2, 3)
        print("Access a single element")
        print(m[0, 2])                        # access an element
        m[0, 2] = 252                         # a slice of an array is a view into the same da
        print("\nModified a single element")
        print(m)                              # this will modify the original array

        print("\nAccess a subarray")
        print(m[1, :])                        # access a row (to 1d array)
        print(m[1:, :])                       # access a row (to 2d array)
        print("\nTranspose a subarray")
        print(m[1, :].T)                      # notice the difference of the dimension of resu
        print(m[1:, :].T)                     # this will be helpful if you want to transpose

        # Boolean array indexing
        # Given a array m, create a new array with values equal to m
        # if they are greater than 0, and equal to 0 if they less than or equal 0

        m = np.array([[3, 5, -2], [5, -1, 0]])
        n = np.zeros(m.shape)
        n[m > 0] = m[m > 0]
        print("\nBoolean array indexing")
        print(n)
```

```
Access a single element
3

Modified a single element
[[  1   2 252]
```

```
 [  4   5   6]]

Access a subarray
[4 5 6]
[[4 5 6]]

Transpose a subarray
[4 5 6]
[[4]
 [5]
 [6]]

Boolean array indexing
[[3. 5. 0.]
 [5. 0. 0.]]
```

1.3.3 Operations on array

Elementwise Operations

```
In [4]: import numpy as np

        a = np.array([[1, 2, 3], [2, 3, 4]], dtype=np.float64)
        print(a * 2)                                               # scalar multiplication
        print(a / 4)                                               # scalar division
        print(np.round(a / 4))
        print(np.power(a, 2))
        print(np.log(a))


        b = np.array([[5, 6, 7], [5, 7, 8]], dtype=np.float64)
        print(a + b)                                               # elementwise sum
        print(a - b)                                               # elementwise difference
        print(a * b)                                               # elementwise product
        print(a / b)                                               # elementwise division
```

```
[[2. 4. 6.]
 [4. 6. 8.]]
[[0.25 0.5  0.75]
 [0.5  0.75 1.  ]]
[[0. 0. 1.]
 [0. 1. 1.]]
[[ 1.  4.  9.]
 [ 4.  9. 16.]]
[[0.         0.69314718 1.09861229]
 [0.69314718 1.09861229 1.38629436]]
[[ 6.  8. 10.]
```

```
 [ 7. 10. 12.]]
[[-4. -4. -4.]
 [-3. -4. -4.]]
[[ 5. 12. 21.]
 [10. 21. 32.]]
[[0.2        0.33333333 0.42857143]
 [0.4        0.42857143 0.5       ]]
```

Vector Operations

```
In [5]: import numpy as np

        a = np.array([[1, 2], [3, 4]])
        print("sum of array")
        print(np.sum(a))                # sum of all array elements
        print(np.sum(a, axis=0))        # sum of each column need a row result
        print(np.sum(a, axis=1))        # sum of each row need a column result
        print("\nmean of array")
        print(np.mean(a))               # mean of all array elements
        print(np.mean(a, axis=0))       # mean of each column
        print(np.mean(a, axis=1))       # mean of each row
```

```
sum of array
10
[4 6]
[3 7]

mean of array
2.5
[2. 3.]
[1.5 3.5]
```

Matrix Operations

```
In [6]: import numpy as np

        a = np.array([[1, 2], [3, 4]])
        b = np.array([[5, 6], [7, 8]])
        print("matrix-matrix product")
        print(a.dot(b))                 # matrix product
        print(a.T.dot(b.T))

        x = np.array([1, 2])
        print("\nmatrix-vector product")
        print(a.dot(x))                 # matrix / vector product
```

```
matrix-matrix product
[[19 22]
```

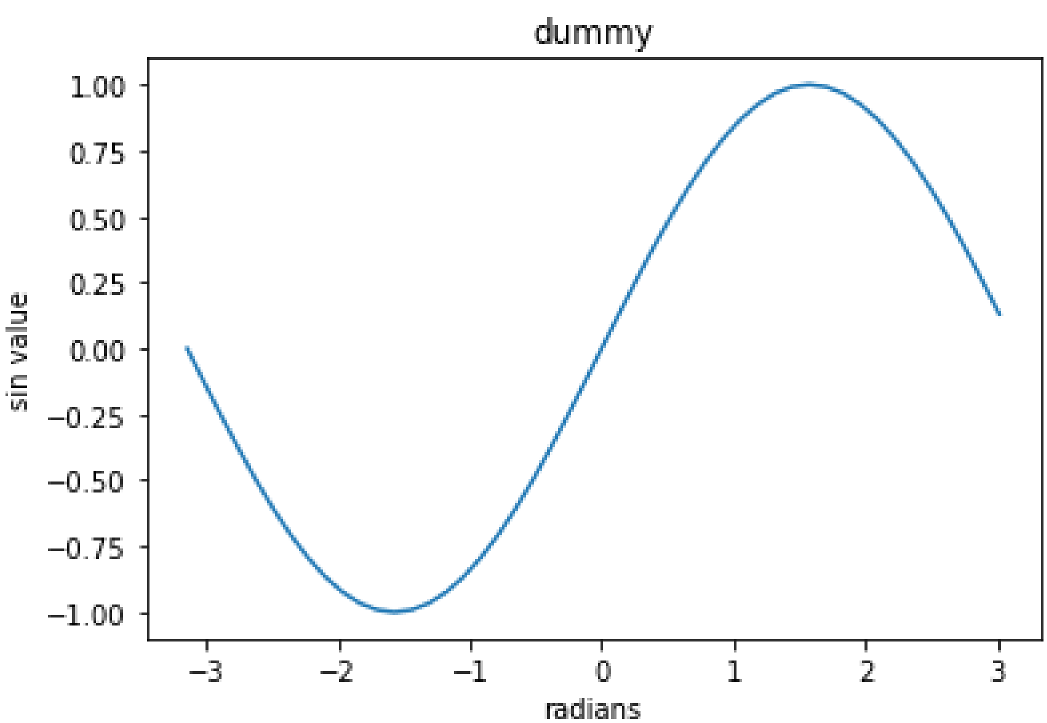
 [43 50]]
[[23 31]
 [34 46]]

matrix-vector product
[5 11]

1.3.4 Matplotlib

Matplotlib is a plotting library. We will use it to show the result in this assignment.

```
In [7]: # this line prepares IPython for working with matplotlib
        %matplotlib inline

        import numpy as np
        import matplotlib.pyplot as plt
        import math

        x = np.arange(-24, 24) / 24. * math.pi
        plt.plot(x, np.sin(x)) #x and y value
        plt.xlabel('radians')
        plt.ylabel('sin value')
        plt.title('dummy')

        plt.show()
```



This breif overview introduces many basic functions from a few popular libraries, but is far from complete. Check out the documentations for Numpy and Matplotlib to find out more.

1.4 Problem 1 Image operations and vectorization (1pt)

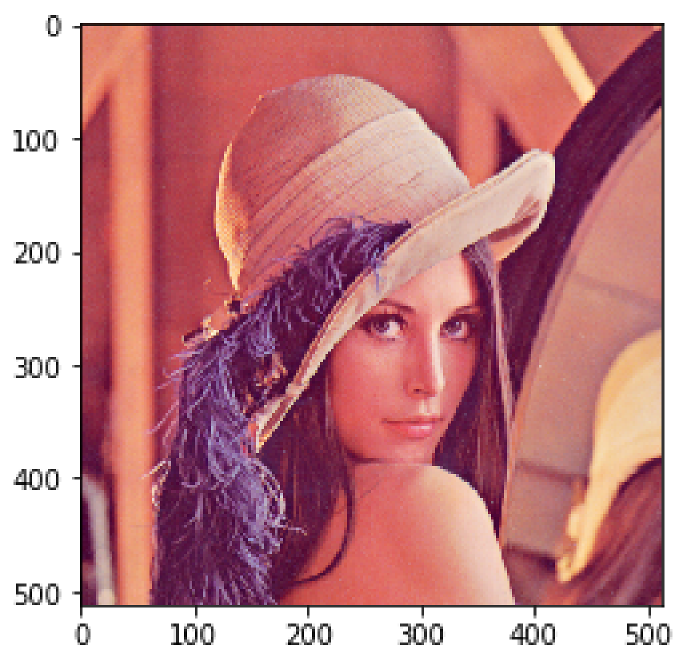
Vector operations using numpy can offer a significant speedup over doing an operation iteratively on an image. The problem below will demonstrate the time it takes for both approaches to change the color of quadrants of an image.

The problem reads an image "Lenna.png" that you will find in the assignment folder. Two functions are then provided as different approaches for doing an operation on the image.

Your task is to follow through the code and fill in the "piazza" function using instructions on Piazza.

```
In [8]: import numpy as np
        import matplotlib.pyplot as plt
        import copy
        import time

        img = plt.imread('Lenna.png')                  # read a JPEG image
        print("Image shape", img.shape)                # print image size and color depth

        plt.imshow(img)                                # displaying the original image
        plt.show()
```

```
Image shape (512, 512, 3)
```

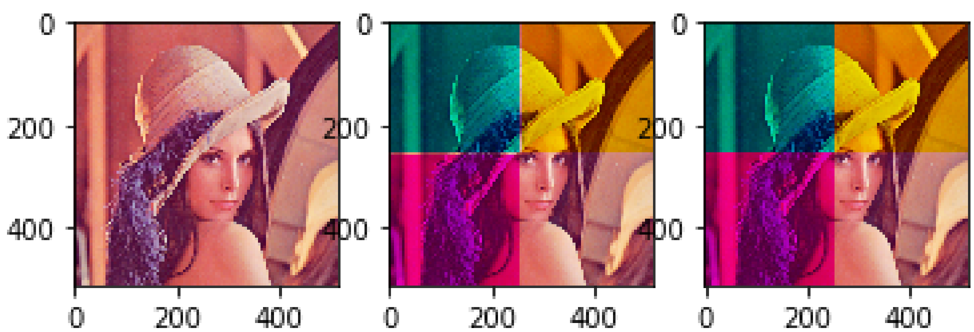


```
In [9]: def iterative(img):

            image = copy.deepcopy(img)              # create a copy of the image matrix
            for x in range(image.shape[0]):
                for y in range(image.shape[1]):
                    if x < image.shape[0]/2 and y < image.shape[1]/2:
                        image[x,y] = image[x,y] * [0,1,1]    #removing the red channel
                    elif x > image.shape[0]/2 and y < image.shape[1]/2:
                        image[x,y] = image[x,y] * [1,0,1]    #removing the green channel
                    elif x < image.shape[0]/2 and y > image.shape[1]/2:
                        image[x,y] = image[x,y] * [1,1,0]    #removing the blue channel
                    else:
                        pass
            return image

        def vectorized(img):

            image = copy.deepcopy(img)
            a = int(image.shape[0]/2)

            b = int(image.shape[1]/2)
            image[:a,:b] = image[:a,:b]*[0,1,1]

            image[a:,:b] = image[a:,:b]*[1,0,1]
            image[:a,b:] = image[:a,b:]*[1,1,0]

            return image
```

```
In [10]: # The code for this problem is posted on Piazza. Sign up for the course if you have n
         # the function definition included in the post 'Welcome to CSE252A' to complete this
         # This is the only cell you need to edit for this problem.
         def piazza():
             start = time.time()
             image_iterative = iterative(img)
             end = time.time()
             print("Iterative method took {0} seconds".format(end-start))
             start = time.time()
             image_vectorized = vectorized(img)
             end = time.time()
             print("Vectorized method took {0} seconds".format(end-start))
             return image_iterative, image_vectorized


         # Run the function
         image_iterative, image_vectorized = piazza()
```

```
Iterative method took 0.7529640197753906 seconds
Vectorized method took 0.003984212875366211 seconds
```

```
In [11]: # Plotting the results in sepearate subplots

         plt.subplot(1, 3, 1)  # create (1x3) subplots, indexing from 1
         plt.imshow(img)       # original image

         plt.subplot(1, 3, 2)
         plt.imshow(image_iterative)

         plt.subplot(1, 3, 3)
         plt.imshow(image_vectorized)

         plt.show()           #displays the subplots

         plt.imsave("multicolor_Lenna.png",image_vectorized)    #Saving an image
```



1.5 Problem 2 Further Image Manipulation (5pts)

In this problem you will reuse the image "Lenna.png". Being a colored image, this image has three channels, corresponding to the primary colors of red, green and blue. Import this image and write your implementation for extracting each of these channels separately to create 2D images. This means that from the nxnx3 shaped image, you'll get 3 matrices of the shape nxn (Note that it's two dimensional).

Now, write a function to merge all these images back into a colored 3D image. The orginal image has a warm color tone, being more reddish. What will the image look like if you exchange the reds with the blues? Merge the 2D images first in original order of channels (RGB) and then with red swapped with blue (BGR).

Finally, you will have **six images**, 1 original, 3 obtained from channels, and 2 from merging. Using these 6 images, create one single image by tiling them together **without using loops**. The image will have 2x3 tiles making the shape of the final image (2*512)x(3*512)x3. The order in which the images are tiled does not matter. Display this image.

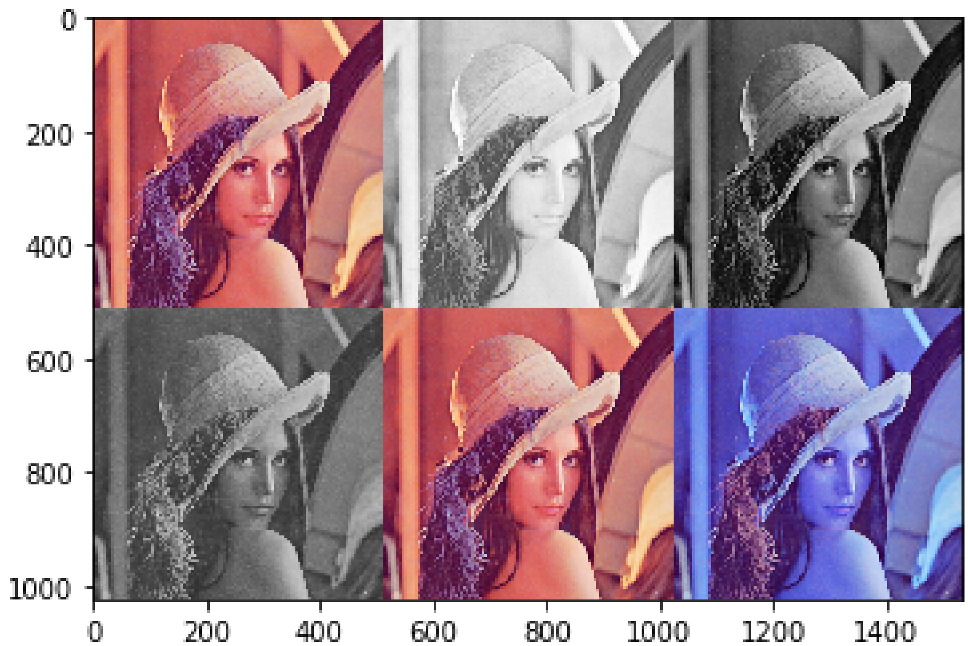
In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
import copy
plt.rcParams['image.cmap'] = 'gray'   # Necessary to override default matplot behavio
```

In [13]:
```
# Write your code here. Import the image and define the required funtions.

image = None
#Import image here
image = plt.imread('Lenna.png')

def getChannel(img,channel):
    '''Function for extracting 2D image corresponding to a channel number from a colo
    image = copy.deepcopy(img)     #Create a copy so as to not change the original ima

    # Write your code here
    return image[:,:, channel -1]

def mergeChannels(image1,image2,image3):
    '''Function for merging three single channels images to form a color image'''
    # Write your code here
    # input 2d matrix
    return np.dstack((image1, image2, image3))
    # np.dstack()
```

In [14]:
```
# Test your function

# getChannel returns a 2d image
assert len(getChannel(image,1).shape) == 2
# mergeChannels returns a 3d image
assert len(mergeChannels(getChannel(image,1),getChannel(image,2),getChannel(image,3))
```

In [15]:
```
# Write your code here for tiling the six images to make a single image and displayin
# Notice that the images returned by getChannel will be 2 dimensional,
# To tile them together with RGB images, you might need to change it to a 3 dimension
# This can be done using np.expand_dims and specifying the axis as an argument.
red_channel = getChannel(image,1)
green_channel = getChannel(image,2)
blue_channel = getChannel(image,3)
zero_img = np.zeros_like(blue_channel)


RGB_img = mergeChannels(red_channel, green_channel, blue_channel)
red_img = np.expand_dims(red_channel, 2) * [1,1,1]
green_img = np.expand_dims(green_channel, 2) * [1,1,1]
blue_img = np.expand_dims(blue_channel, 2) * [1,1,1]
BGR_img = mergeChannels(blue_channel, green_channel, red_channel)
```

```
horizontal_img1 = np.concatenate((image, red_img, green_img), axis = 1)
horizontal_img2 = np.concatenate((blue_img, RGB_img, BGR_img), axis = 1)

result_img = np.concatenate((horizontal_img1, horizontal_img2), axis = 0)

plt.imshow(result_img)
plt.show()
```



** Submission Instructions**
Remember to submit a pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX