CSE 252A Computer Vision I Fall 2019 - Homework 1

Instructor: Ben Ochoa

Assignment Published On: Tuesday, October 1, 2019

Due On: Tuesday, October 8, 2019 11:59 pm

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.
- **Late policy** - Assignments submitted late will receive a 15% grade reduction for each 12 hours late (i.e., 30% per day). Assignments will not be accepted 72 hours after the due date. If you require an extension (for personal reasons only) to a due date, you must request one as far in advance as possible. Extensions requested close to or after the due date will only be granted for clear emergencies or clearly unforeseeable circumstances.

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.

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

See Piazza post on how to add CSE 252A in Gradescope. 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.

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.

Arrays

In [1]:

```
import numpy as np

array1d = np.array([1,0,0])           # a 1d array

print("1d array :")
print(array1d)
print("Shape :", array1d.shape)      # print the shape of array

array2d = np.array([[1], [2], [3]])  # a 2d array
print("\n2d array :")
print(array2d)
print("Shape :", array2d.shape)     # print the size of v, notice the difference

print("\nTranspose 2d :", array2d.T)      # Transpose of a 2d array
print("Shape :", array2d.T.shape)

print("\nTranspose 1d :", array1d.T)      # Notice how 1d array did not change after transpose (Thoughts?)
print("Shape :", array1d.T.shape)

allzeros = np.zeros([2, 3])            # a 2x3 array of zeros
allones =  np.ones([1, 3])             # a 1x3 array of ones
identity = np.eye(3)                   # identity matrix
rand3_1 = np.random.rand(3, 1)        # random matrix with values in [0, 1]
arr = np.ones(allones.shape) * 3        # create a matrix from shape
```

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

2d array :
[[1]
 [2]
 [3]]
Shape : (3, 1)

Transpose 2d : [[1 2 3]]
Shape : (1, 3)

Transpose 1d : [1 0 0]
Shape : (3,)
```

Array Indexing

In [2]:

```
import numpy as np
array2d = np.array([[1, 2, 3], [4, 5, 6]])  # create a 2d array with shape (2, 3)
print("Access a single element")
print(array2d[0, 2])                    # access an element
array2d[0, 2] = 252                            # a slice of an array is a view into the same data;
print("\nModified a single element")
print(array2d)                                 # this will modify the original array

print("\nAccess a subarray")
print(array2d[1, :])                           # access a row (to 1d array)
print(array2d[1:, :])                          # access a row (to 2d array)
print("\nTranspose a subarray")
print(array2d[1, :].T)                         # notice the difference of the dimension of resulting array
print(array2d[1:, :].T)                        # this will be helpful if you want to transpose it later

# 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

array2d = np.array([[3, 5, -2], [50, -1, 0]])
arr = np.zeros(array2d.shape)
arr[array2d > 0] = array2d[array2d > 0]
print("\nBoolean array indexing")
print(arr)
```

```
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.]
 [50.  0.  0.]]
```

Operations on array

Elementwise Operations

In [3]:

```
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 [4]:

```
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
print(np.sum(a, axis=1))        # sum of each row
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 [5]:

```
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
print(a@x)       # Can also make use of the @ instad of .dot()
```

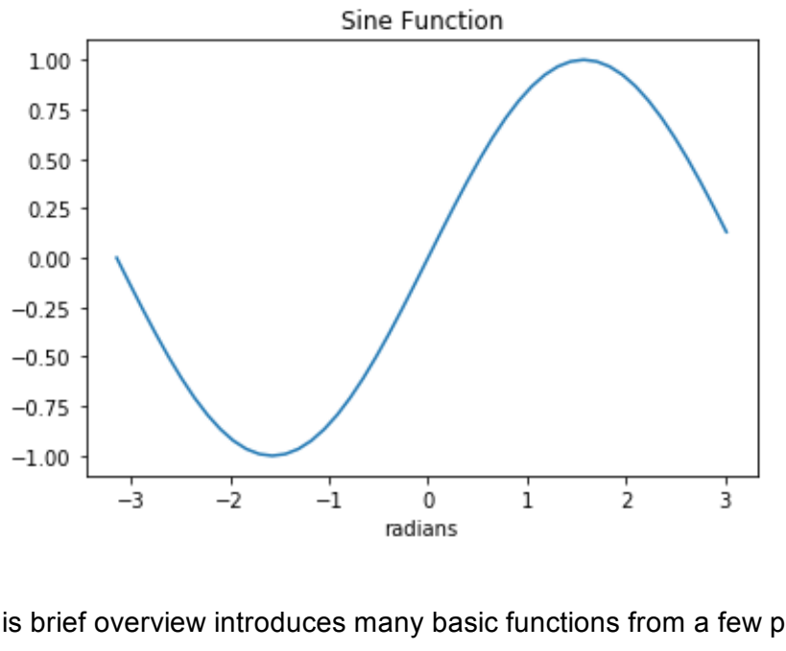
```
matrix-matrix product
[[19 22]
 [43 50]]
[[23 31]
 [34 46]]

matrix-vector product
[ 5 11]
[ 5 11]
```

Matplotlib

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

In [6]:

```
# 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))
plt.xlabel('radians')
plt.ylabel('sin value')
plt.title('Sine Function')

plt.show()
```



This brief 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.

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.

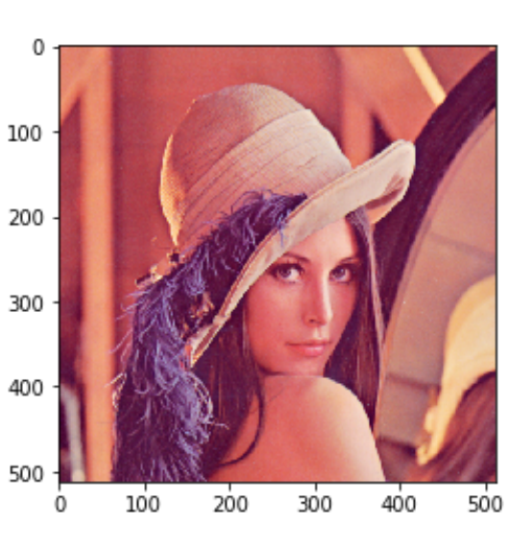
The problem reads an image "Lenna.png" that you will find in the assignment folder. Two functions are then provided as different approaches for doing an operation on the image.

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

In [7]:

```
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 [8]:

```
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
```

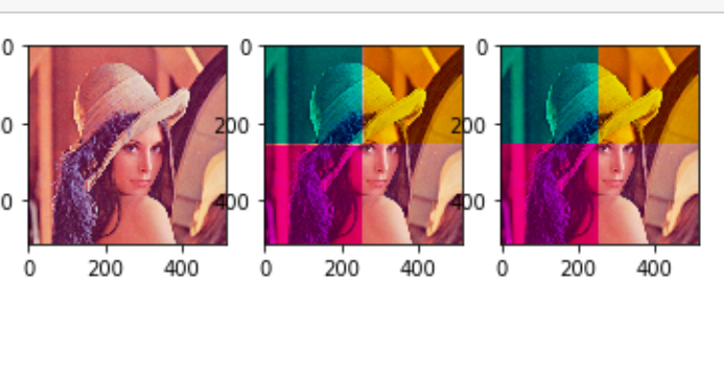
In [9]:

```
# # The code for this problem is posted on Piazza. Sign up for the course if you have not. Then find
# # the function definition included in the post 'Welcome to CSE252A' to complete this problem.
# # 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 2.140733242034912 seconds
Vectorized method took 0.018352031707763672 seconds
```

In [10]:

```
# 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
```



Problem 2 Further Image Manipulation (7pts)

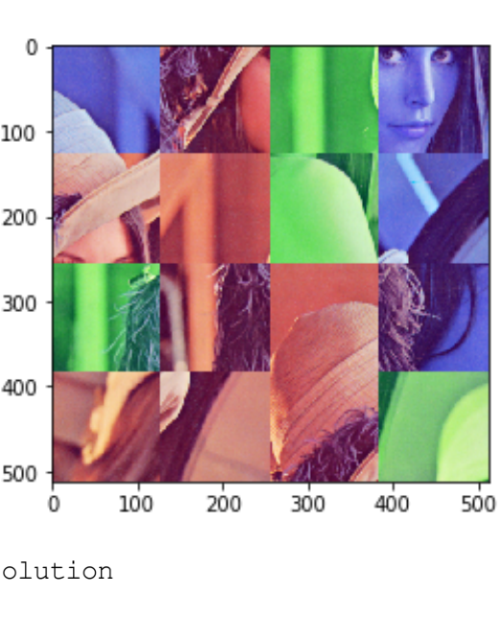
In this problem you will solve a jigsaw puzzle using the 'jigsaw.png' provided with the homework. The solution of this jigsaw is the Lenna image we used above. There are a total of 16 jigsaw pieces of size 128x128x3 which together make up the 512x512x3 image. Not only is Lemma jumbled spatially, but some of the channels in jigsaw pieces are also permuted i.e. **RGB** to **BGR** and **GRB**.

Your task is to put all the pieces to their respective locations and correct the channel permutations. To achieve this task, you are required to complete the three helper functions that will be used to solve this puzzle. You are **NOT** allowed to use any function other than the three provided here. Also, the code needs to be *vectorised* i.e. you are **NOT** allowed to use *for* loops to achieve this task.

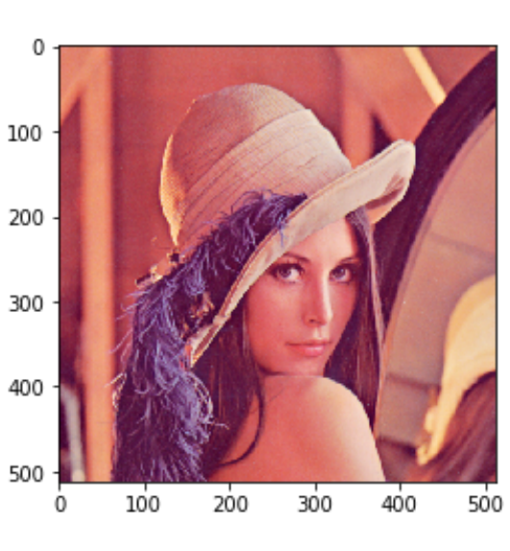
In [66]:

```
def getTile(jigsaw, tile_idx):

    '''
    This function returns a particular jigsaw piece

    jigsaw      : 512x512x3 np.ndarray
    tile_idx : tuple containing the (i,j) location of the piece

    piece    : 128x128x3 np.ndarray
    '''
    assert isinstance(tile_idx,tuple), 'tile index must be a tuple'
    assert len(tile_idx) == 2, 'tile index must specify the row and column index of the jigsaw'


    # Write your code here
    piece = np.zeros((128,128,3))   # modify piece
    piece = jigsaw[tile_idx[0]*128: (tile_idx[0]+1)*128, tile_idx[1]*128: (tile_idx[1]+1)*128,:3]

    return piece.copy()

def permuteChannels(tile, permutation):

    '''
    This function performs a permutation on channel

    tile          : 128x128x3 np.ndarray
    permutation   : tuple containing (i,j,k) channel indices
    tile_permuted : 128x128x3 np.ndarray
    '''

    assert tile.shape == (128,128,3), 'tile size should be 128x128x3'
    assert isinstance(permutation, tuple), 'permutation should be a tuple'
    assert len(permutation) == 3, 'There are only 3 channels'

    #Write your code here
    tile[:,:,[permutation[0],permutation[1],permutation[2]]]=tile[:,:,[0,1,2]]
    tile_permuted = tile.copy()

    return tile_permuted.copy()

def putTile(board, tile, tile_idx):

    '''
    This function put a jigsaw piece at a particular location on the board

    board : 512x512x3 np.ndarray
    tile : 128x128x3 np.ndarray
    tile_idx : tuple containing the (i,j) location of the piece
    img : 512x512x3 np.ndarray
    '''

    assert board.shape == (512,512,3), 'canvas size should be 512x512x3'
    assert tile.shape == (128,128,3), 'tile size should be 128x128x3'
    assert isinstance(tile_idx,tuple), 'tile index must be a tuple'
    assert len(tile_idx) == 2, 'tile index must specify the row and column index of the jigsaw'

    # Write your own code here
    img = board.copy()   # modify img
    img[tile_idx[0]*128:(1+tile_idx[0])*128, tile_idx[1]*128:(1+tile_idx[1])*128, 0:] = tile

    return img

TILE_SIZE = 128
source = [(0,0),(0,1),(0,2),(0,3),
          (1,0),(1,1),(1,2),(1,3),
          (2,0),(2,1),(2,2),(2,3),
          (3,0),(3,1),(3,2),(3,3)]

# Fill in the target list with the corresponding piece locations
target = [(0,2),(2,1),(1,0),(2,2),
          (1,2),(0,0),(3,2),(0,3),
          (2,0),(3,0),(0,1),(3,1),
          (3,3),(1,3),(1,1),(2,3)]

#Fill in the respective channel permutations
channelPermutation = [(2,1,0),(0,1,2),(1,0,2),(2,1,0),
                      (0,1,2),(0,1,2),(1,0,2),(2,1,0),
                      (1,0,2),(0,1,2),(0,1,2),(2,1,0),
                      (0,1,2),(0,1,2),(0,1,2),(1,0,2)]

jigsaw = plt.imread('jigsaw.png')
board = np.ones(jigsaw.shape)

for i in range(16):
    tile = getTile(jigsaw, source[i])
    tile = permuteChannels(tile, channelPermutation[i])
    board = putTile(board, tile, target[i])

print("Jigsaw Puzzle")
plt.imshow(jigsaw)
plt.show()
print("Solution")
plt.imshow(board)
plt.show()
```

```
Jigsaw Puzzle
```



```
Solution
```



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. Upload to Gradescope. **NOTE**: You need to have XeTex installed on your machine to generate PDFs