CSE 252A Computer Vision I Fall 2021 - Assignment 0

Instructor: Ben Ochoa

- Assignment Published On: **Wed, September 29, 2021**.
- Due On: **Wed, October 6, 2021 11:59 PM (Pacific Time)**.

Instructions

Please answer the questions below using Python in the attached Jupyter notebook and follow the guidelines below:

- This assignment must be completed **individually**. For more details, please follow the Academic Integrity Policy and Collaboration Policy on Canvas.
- All the solutions must be written in this Jupyter notebook.
- After finishing the assignment in the notebook, please export the notebook as a PDF and submit both the notebook and the PDF (i.e. the `.ipynb` and the `.pdf` files) on Gradescope.
- You may use basic algebra packages (e.g. `NumPy`, `SciPy`, etc) but you are not allowed to use the packages that directly solve the problems. Feel free to ask the instructor and the teaching assistants if you are unsure about the packages to use.
- 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.

Introduction

Welcome to **CSE252A Computer Vision I**!

This course provides a comprehensive introduction to computer vision providing broad coverage including low level vision (image formation, photometry, color, image feature detection), inferring 3D properties from images (shape-from-shading, stereo vision, motion interpretation) and object recognition.

We will use a variety of tools (e.g. some packages and operations) in this class that may 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 0**. You will also practice some basic image manipulation techniques.

Piazza, Gradescope and Python

Piazza

All students are automatically added to the class in Piazza once enrolled in this class. You can get access to it from Canvas. 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

All students are automatically added to the class in Gradescope once enrolled in this class. You can also get access to it from Canvas. 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 web-based Jupyter 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. Some basic use of this packages is shown below. This is **NOT** a problem, but you are highly recommended to run the following code with some of the input changed in order to understand the meaning of the operations.

Arrays

In [ ]:

```
import numpy as np                # Import the NumPy package

v = np.array([1, 2, 3])           # A 1D array
print(v)
print(v.shape)                    # Print the size / shape of v
print("1D array:", v, "Shape:", v.shape)

v = np.array([[1], [2], [3]])     # A 2D array
print("2D array:", v, "Shape:", v.shape) # Print the size of v and check the difference.

# You can also attempt to compute and print the following values and their size.

v = v.T                           # Transpose of a 2D array
m = np.zeros([3, 4])              # A 2x3 array (i.e. matrix) of zeros
v = np.ones([1, 3])               # A 1x3 array (i.e. a row vector) of ones
v = np.ones([3, 1])               # A 3x1 array (i.e. a column vector) of ones
m = np.eye(4)                     # Identity matrix
m = np.random.rand(2, 3)          # A 2x3 random matrix with values in [0, 1] (sampled from u
```

Array Indexing

In [ ]:

```
import numpy as np

print("Matrix")
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print(m)

print("\nAccess a single element")
print(m[0, 1])                          # Access an element
```

```
m[1, 1] = 100                           # Modify an element
print("\nModify a single element")
print(m)

print("\nAccess a subarray")
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print(m[1, :])                          # Access a row (to 1D array)
print(m[1:2, :])                        # Access a row (to 2D array)
print(m[1:3, :])                        # Access a sub-matrix
print(m[1:, :])                         # Access a sub-matrix

print("\nModify a subarray")
m = np.array([[1,2,3], [4,5,6], [7,8,9]]) # Create a 3x3 array.
v1 = np.array([1,1,1])
m[0] = v1
print(m)
m = np.array([[1,2,3], [4,5,6], [7,8,9]]) # Create a 3x3 array.
v1 = np.array([1,1,1])
m[:,0] = v1
print(m)
m = np.array([[1,2,3], [4,5,6], [7,8,9]]) # Create a 3x3 array.
m1 = np.array([[1,1],[1,1]])
m[:2,:2] = m1
print(m)

print("\nTranspose a subarray")
m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # Create a 3x3 array.
print(m[1, :].T)                                # Notice the difference of the dimension 
print(m[1:2, :].T)
print(m[1:, :].T)
print(np.transpose(m[1:, :], axes=(1,0)))       # np.transpose() can be used to transpose

print("\nReverse the order of a subarray")
print(m[1, ::-1])                               # Access a row with reversed order (to 1D

# Boolean array indexing
# Given a array m, create a new array with values equal to m
# if they are greater than 2, and equal to 0 if they less than or equal to 2
m = np.array([[1, 2, 3], [4, 5, 6]])
m[m > 2] = 0
print("\nBoolean array indexing: Modify with a scaler")
print(m)

# Given a array m, create a new array with values equal to those in m
# if they are greater than 0, and equal to those in n if they less than or equal 0
m = np.array([[1, 2, -3], [4, -5, 6]])
n = np.array([[1, 10, 100], [1, 10, 100]])
n[m > 0] = m[m > 0]
print("\nBoolean array indexing: Modify with another array")
print(n)
```

Array Dimension Operation

In [ ]:

```
import numpy as np

print("Matrix")
m = np.array([[1, 2], [3, 4]]) # Create a 2x2 array.
print(m, m.shape)

print("\nReshape")
re_m = m.reshape(1,2,2)   # Add one more dimension at first.
print(re_m, re_m.shape)
re_m = m.reshape(2,1,2)   # Add one more dimension in middle.
print(re_m, re_m.shape)
```

```
re_m = m.reshape(2,2,1)   # Add one more dimension at last.
print(re_m, re_m.shape)

print("\nStack")
m1 = np.array([[1, 2], [3, 4]]) # Create a 2x2 array.
m2 = np.array([[1, 1], [1, 1]]) # Create a 2x2 array.
print(np.stack((m1,m2)))

print("\nConcatenate")
m1 = np.array([[1, 2], [3, 4]]) # Create a 2x2 array.
m2 = np.array([[1, 1], [1, 1]]) # Create a 2x2 array.
print(np.concatenate((m1,m2)))
print(np.concatenate((m1,m2), axis=0))
print(np.concatenate((m1,m2), axis=1))
```

Math Operations on Array

Element-wise Operations

In [ ]:

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float64)
print(a * 3)                                                # Scalar multiplication
print(a / 2)                                                # Scalar division
print(np.round(a / 2))
print(np.power(a, 2))
print(np.log(a))
print(np.exp(a))

b = np.array([[1, 1, 1], [2, 2, 2]], dtype=np.float64)
print(a + b)                                                # Elementwise sum
print(a - b)                                                # Elementwise difference
print(a * b)                                                # Elementwise product
print(a / b)                                                # Elementwise division
print(a == b)                                               # Elementwise comparison
```

Broadcasting

In [ ]:

```
# Note: See https://numpy.org/doc/stable/user/basics.broadcasting.html
#       for more details.
import numpy as np
a = np.array([[1, 1, 1], [2, 2, 2]], dtype=np.float64)
b = np.array([1, 2, 3])
print(a*b)
```

Sum and Mean

In [ ]:

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
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
```

Vector and Matrix Operations

In [ ]:

```
import numpy as np

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

x = np.array([3, 4])
print("\nMatrix-vector product")
print(a.dot(x))                 # Matrix-vector product

x = np.array([1, 2])
y = np.array([3, 4])
print("\nVector-vector product")
print(x.dot(y))                 # Vector-vector product
```

Matplotlib

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

In [ ]:

```
%config InlineBackend.figure_format = 'retina' # For high-resolution.
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-2., 2., 0.01) * np.pi
plt.plot(x, np.sin(x))
plt.xlabel('x')
plt.ylabel('$\sin(x)$ value') # '$...$' for a LaTeX formula.
plt.title('Sine function')

plt.show()
```

This brief overview introduces many basic functions from `NumPy` and `Matplotlib`, but is far from complete. Check out more operations and their use in documentations for NumPy and Matplotlib.

Problem 1: Image Operations and Vectorization (15 points)

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 `ucsd-triton-statue.png` that you will find in the assignment folder. Two functions are then provided as different approaches for doing an operation on the image.

The function `iterative()` demonstrates the image divided into 4 parts:

(Top Left) The original image.
(Top Right) Red channel image.
(Bottom Left) (B,G,R) colored image.
(Bottom Right) Grayscale image.

For your implementation:

(1) For the red channel image, write your implementation to extract a single channel from a colored image. This means that from the $H \times W \times 3$ shaped image, you'll get three matrices of the shape $H \times W$ (Note that it's 2-dimensional).

(2) For the (B,G,R) colored image, write a function to merge those single channel images back into a 3-dimensional colored image in the reversed channels order (B,G,R).

(3) For the grayscale image, write a function to conduct operations with the extracted channels.

Your task is to follow through the code and fill the blanks in `vectorized()` function and compare the speed difference between `iterative()` and `vectorized()`. Make sure your final generated image in the `vectorized()` is the same as the one generated from `iterative()`.

In [ ]:

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

img = plt.imread('ucsd-triton-statue.jpg') # Read an image
print("Image shape:", img.shape)           # Print image size and color depth. The shape s

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

In [ ]:

```
import copy
import time
def iterative(img):
    """ Iterative operation. """
    image = copy.deepcopy(img)              # Create a copy of the image matrix
    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            #Top Right
            if y < image.shape[0]/2 and x > image.shape[1]/2:
                image[y,x] = image[y,x] * np.array([1,0,0])    # Keep the red channel
            #Bottom Left
            elif y > image.shape[0]/2 and x < image.shape[1]/2:
                image[y,x] = [image[y,x][2], image[y,x][1], image[y,x][0]]  #(B,G,R) image
            #Bottom Right
            elif y > image.shape[0]/2 and x > image.shape[1]/2:
                r,g,b = image[y,x]
                image[y,x] = 0.2989 * r + 0.5870 * g + 0.1140 * b
    return image

def get_channel(img, channel):
    """ Function to extract 2D image corresponding to a channel index from a color image.
    This function should return a H*W array which is the corresponding channel of the inpu
    img = copy.deepcopy(img)     # Create a copy so as to not change the original image

    #### Write your code here. ####


def merge_channels(img0, img1, img2):
    """ Function to merge three single channel images to form a color image.
    This function should return a H*W*3 array which merges all three single channel images
    (i.e. img0, img1, img2) in the input."""
    # Hint: There are multiple ways to implement it.
    #       1. For example, create a H*W*C array with all values as zero and
    #          fill each channel with given single channel image.
    #          You may refer to the "Modify a subarray" section in the brief NumPy tutoria
    #       2. You may find np.stack() / np.concatenate() / np.reshape() useful in this pr

    #### Write your code here. ####
```

```python
def vectorized(img):
    """ Vectorized operation. """
    image = copy.deepcopy(img)
    a = int(image.shape[0]/2)
    b = int(image.shape[1]/2)
    # Please also keep the red / green / blue channel respectively in the corresponding pa
    # with the vectorized operations. You need to make sure your final generated image in
    # vectorized() function is the same as the one generated from iterative().

    #### Write your code here. ####
    #Top Right: keep the red channel
    image[:a,b:] =

    #Bottom Left: (B,G,R) image
    image[a:,:b] =

    #Bottom Right: Grayscale image
    image[a:,b:] =

    return image
```

Now, run the following cell to compare the difference between iterative and vectorized operation.

In [ ]:

```python
import time

def compare():
    img = plt.imread('ucsd-triton-statue.jpg')
    cur_time = time.time()
    image_iterative = iterative(img)
    print("Iterative operation (sec):", time.time() - cur_time)

    cur_time = time.time()
    image_vectorized = vectorized(img)
    print("Vectorized operation (sec):", time.time() - cur_time)

    return image_iterative, image_vectorized

# Test your implemented get_channel()
assert len(get_channel(img, 0).shape) == 2  # Index 0

# Run the function
image_iterative, image_vectorized = compare()

# Plotting the results in sepearate subplots.
plt.figure(figsize=(12,4))   # Adjust the figure size.
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)  # Iterative operations on the image.

plt.subplot(1, 3, 3)
plt.imshow(image_vectorized) # Vectorized operations on the image.

plt.show()                   # Show the figure.

# Note: The shown figures of image_iterative and image_vectorized should be identical!
```

Problem 2: More Image Manipulation (35 points)

In this problem you will use the image `bear.png`. Being a color image, this image has three channels, corresponding to the primary colors of red, green and blue.

(1) Read the image.

(2) Write a function to flip the original image from top to bottom. For this function, please only use **Array Indexing** to implement this function and **do not** directly use the functions (e.g. `np.flip()`) that directly flips the matrix.

(3) Next, write another function to rotate the original image 90 degrees counterclockwise. For this function, please only use **Array Indexing** to implement this function and **do not** directly use the functions (e.g. `np.rot90()`) that directly rotates the matrix. Try to apply the rotation function once (i.e. 90-degree rotation) and twice (i.e. 180-degree rotation)

(4) Read the `face-mask.png` image and the corresponding `face-mask-binary.png` binary mask image.

(5) Given the `start_x` and `start_y` on the bear image indicating the starting position(top-left corner) of the face mask, you need to write a function to help the bear put on the face mask. (Hints: **Mask** pixel values of 1 indicate **image** pixels to show.)

(6) Finally, consider **4 color images** you obtained: 1 original bear image, 1 from flipping (top to bottom), 1 from rotation (180-degree), and 1 for bear wearing the face mask. Using these 4 images, create one single image by tiling them together **without using loops**. The image will have 2×2 tiles making the shape of the final image $2H \times 2W \times 3$. The order in which the images are tiled does not matter. Show the tiled image.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import copy
```

In [ ]:

```
# (1) Read the image.
#### Write your code here. ####
img =

plt.imshow(img) # Show the image after reading.
plt.show()
```

In [ ]:

```
# (2) Flip the image from top to bottom.
def flip_img(img):
    """ Function to mirror image from top to bottom.
    This function should return a H*W*3 array which is the flipped version of original ima
    """
    #### Write your code here. ####


plt.imshow(img)
plt.show()
flipped_img = flip_img(img)
plt.imshow(flipped_img)
plt.show()
```

In [ ]:

```
# (3) Rotate image.
def rotate_90(img):
    """ Function to rotate image 90 degrees counter-clockwise.
    This function should return a W*H*3 array which is the rotated version of original ima
    #### Write your code here. ####
```

```python
plt.imshow(img)
plt.show()
rot90_img = rotate_90(img)
plt.imshow(rot90_img)
plt.show()
rot180_img = rotate_90(rotate_90(img))
plt.imshow(rot180_img)
plt.show()
```

In [ ]:

```python
# (4)Read the face mask image and the face mask binary image

#### Write your code here. ####

mask_img =
bi_mask_img =

print("Face Mask Image Size: ")
print(mask_img.shape)
print("Face Mask Binary Mask Image Size: ")
print(bi_mask_img.shape)

plt.imshow(mask_img)
plt.show()
plt.imshow(bi_mask_img)
plt.show()
```

In [ ]:

```python
# (5) Put the face mask on the bear's face
start_x = 565
start_y = 240

maskon_img = copy.deepcopy(img)

#### Write your code here. ####


plt.imshow(maskon_img)
```

In [ ]:

```python
# (6) Write your code here to tile the four images and make a single image.
# You can use the img, flipped_img, rot180_img, maskon_img to represent the four images.
# After tiling, please display the tiled image.
#### Write your code here. ####
```

Submission Instructions

Remember to submit **both** the Jupyter notebook file and the PDF version of this notebook to Gradescope. Please make sure the content in each cell is clearly shown in your final PDF file. To convert the notebook to PDF, you can choose one way below:

1. You can print the web page and save as PDF (e.g. Chrome: Right click the web page → Print... → Choose "Destination: Save as PDF" and click "Save").
2. You can find the export option in the header: File → Download as → "PDF via LaTeX"