CS484 - Homework Tutorial

1 Installation

Please install anaconda. Anaconda is a powerful package manager and environment manager that you use with command line commends at the Anaconda Prompt for Windows, or in a terminal window for macOS or Linux. Here is documentation about anaconda installation for each OS: https://docs.anaconda.com/anaconda/install/

When the installation is finished, create a new environment for the homework. We will name the environment 'cs484_hw' and install Python (v3.8) package. Note that we use Python in this course, but you don't have to install it directly. Then, activate the new environment with 'conda activate'.
At the anaconda prompt or in your terminal window, type line by line as follows:

```
conda create --name cs484_hw python=3.8
conda activate cs484_hw
pip install opencv-contrib-python
pip install opencv-python
conda install matplotlib
```

To see a list of all your packages in the environment, type:

```
conda list
```

The result should contain Matplotlib, Numpy, Opencv-python, Opencv-contrib-python, and Python.

2 Official Python Introduction

If you are not familiar with Python, we recommend reading the official Python tutorial: https://docs.python.org/3/tutorial/

The rest of this document contains concepts we will assume you know. Please become familiar with them, try them out, and let us know if you have any questions.

3 Arrays

3.1 Indexing

NumPy arrays are pretty similar to the Python list. To see how the NumPy array works, try the following exercise.

To create a 5-element NumPy array A, with integers 1 through 5 as follows:

```
import numpy as np
A = np.arange(1,6)
```

A should now be equal to the NumPy array $[1, 2, 3, 4, 5]$.

We can access each element with an index same as the python list. When we print $A[1]$, 2 is the return value; A[1] is the second element in the array A.

Now try modifying the first element of the array by setting its value to 5; $A[0] = 5$. The value of the array should now be $[5, 2, 3, 4, 5]$.

4 Reading and Displaying Images with OpenCV

We import OpenCV library and NumPy as follows:

```
import cv2
import numpy as np
```

The $cv2.imread$ function is used to load images as NumPy array with OpenCV.

The $cv2.imwrite$ function takes in a filename and a matrix representation of an image as parameters and saves the image to a specified file.

The $cv2.imshow$ function displays an image in the specified window. The input array should have value range [0,255]. Note that if the type of the input array is a floating-point format, the value range [0,1] is mapped to [0,255].

However, OpenCV uses different order of the color channels: not RGB, but BGR for the image processing functions ($cv2.imread$, $cv2.imwrite$, $cv2.imshow$). For example, to get the red channel value of a specific pixel, you have to see index 2, not 0. So, you should pay attention when you are using those functions.

4.1 Types

We can freely handle the type of variable in Python. And so it is for the NumPy array.

Try downloading an image from the internet and reading it into the Python environment like so:

```
image = cv2.imread('yourimage.jpg')
```

You can check the type of NumPy array with $image.dtype$ since the array has a $dtype$ parameter. If you print it, you should notice that the 'image' variable is a *uint8* NumPy array, which means that each value in the array is an 8-bit unsigned integer. Thus, if you examine the array, you should notice that it consists of integers from 0 to 255.

At times, you will want to alter the image in ways such that some of the entries become non-integer values, and thus you can convert the image to floating-point format by using the NumPy array method $astype$. However, in general, $cv2.imwrite$ function only gets 8-bit single-channel or 3-channel images as input array (see documents). If the conversion is not done, an entry is

automatically rounded to an integer if it is set to a non-integer value. This means when you want to save $.jpg$ or $.png$ images, you should convert numpy array type as $np.uint8$ for the safe.

Note that we can freely convert the image array to floating-point format as follows:

```
image = image.astype(np.float32)
```

The candidates of the parameter for the method $astype$ are in $Arraytypes$.

4.2 Multidimensional Arrays / Matrices

The multidimensional array in NumPy is an extension of the two-dimensional matrix. And they are also called N-dimensional arrays. One of the most common usages of multidimensional arrays in computer vision is to represent images with multiple channels. For instance, an RGB image has three channels and can be represented as a 3-D array. Each of these channels can be accessed independently.

Let us create an 'RGB' image. To begin, let us create a 300x400x3 array and initialize it to zeros. This can be done as follows:

```
image = np.zeros((300, 400, 3), dtype=np.uint8)
```

Now, we assign a mid red to the first hundred columns and a bright red to the following hundred columns:

```
image[:,:100,0] = 128 # 'half' red
image[:,100:200,0] = 255 # 'full' red
```

The colon ':' indexes all elements in a particular dimension.

Finally, we can assign green randomly to the first 100 rows:

```
image[:100,:,1] = np.random.randint(255, size=(100,400)) # Green
```

To view the image, use $cv2.imshow$ to display target image in the specified window. Then, save image as $result.png$:

```
# Convert RGB image to BGR for imwrite function
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
cv2.imshow('result', image)
cv2.waitKey(0)
cv2.imwrite('result.png',image)
```

4.3 Color images vs. Grayscale

Color images are often built of several stacked color channels, each of them representing the value levels of the given channel. For example, RGB images are composed of three independent channels for red, green, and blue primary color components. In contrast, a grayscale image (aka

black and white image) is one in which the value of each pixel is a single sample, that is, it carries only intensity information.

In OpenCV, it is easy to convert an RGB image to grayscale. This can be achieved using the *cvtColor* function and the parameter 'cv2.COLOR_RGB2GRAY'.

You can also access individual color channels of a color image. This is illustrated in the code snippet below.

```
# Read in original RGB image.
rgbImage = cv2.imread('gigi.jpg')
m,n,o = rgbImage.shape

# Extract color channels.
blueChannel = rgbImage[:,:,0] # Blue channel
greenChannel = rgbImage[:,:,1] # Green channel
redChannel = rgbImage[:,:,2] # Red channel

# Create an all black channel.
allBlack = np.zeros((m, n), 'uint8')

# Create color versions of the individual color channels.
justBlue = np.stack((blueChannel, allBlack, allBlack), axis=2)
justGreen = np.stack((allBlack, greenChannel, allBlack), axis=2)
justRed = np.stack((allBlack, allBlack, redChannel), axis=2)

# Recombine the individual color channels to create the original RGB
    image again.
recombinedRGBImage = np.stack((blueChannel, greenChannel,
    redChannel), axis=2)
```

Try to view the various results using *cv2.imwrite*.

5 Performance Improvements

When using the high-dimensional array, it is best to avoid using for loops whenever possible; one can attain significant performance improvements through vectorization and logical indexing.

5.1 Vectors as function parameters

Most NumPy functions support passing vectors or matrices as parameters. This prevents you from having to apply the function to individual elements as a way of improving performance. It is best illustrated with a few examples:

Suppose you have a 10-element array A. You want to take the sine of each element and store the results in another array B. A naive method would use a for loop as follows:

```
A = np.arange(10, dtype=np.float32)
B = np.zeros((10))
for i in range(10):
   B[i] = np.sin(A[i])
```

The same operation can be accomplished as follows:

```
B = np.sin(A)
```

Similar operations can be completed if one wishes to raise every element in A to a certain power. For example, suppose we want to square every element in A and store the result in B. This can be done as follows:

```
B = np.power(A, 2)
```

5.2 Logical Indexing

Suppose we have an $m \times n$ 2D array, and we want to set every element in the array that has a value greater than 100 to 255. This can be done as follows with a for loop:

```
m = 400
n = 400
A = np.random.randint(256, size=(m,n))
for i in range(m):
   for j in range(n):
      if A[i,j] > 100:
         A[i,j] = 255
cv2.imwrite('non_logical.png',A)
```

A more efficient method uses logical indexing:

```
B = A > 100
A[B] = 255
```

B is now a binary logical array, where for all i, j, $B[i, j] = 1$ if and only if $A[i, j] > 100$; otherwise, $B[i, j] = 0$. Then we do the following: $A[B] = 255$. An element-wise assignment is then performed; the result of A the same as it would be using the for loop method. A appears brighter, as more pixels are set to their maximum value.

5.3 Evaluating Performance

We can evaluate time performance using the python standard library 'time'. It is used as follows:

```
import time
start_time = time.time() # Get current time
# Perform some operation
time_spent = time.time() - start_time # Calculate elapsed time
```

The elapsed time during some operation is the subtraction of start-time from end-time. You should try doing several of the examples above, and note the performance differences between using for loops and using the more efficient methods.

6 Debugging

Python provides a Python Debugger module called pdb for debugging. Please see the following Python page for information on how to enter the debugger, and how to set and navigate breakpoints: Python Debugger

7 Homework 1 questions

Made it this far? Good. Now, please attempt the Homework 1 questions.