DS2030 Data Structures and Algorithms for Data Science
Lab 1 (In Person) Due on August 13, 5.00pm

Instructions

- You are to use Python as the programming language. Use may use Visual Studio Code (or any other editor you are comfortable with) as the IDE.
- You have to work individually for this lab.
- You are not allowed to share code with your classmates nor allowed to use code from the internet. You are encouraged engage in high level discussions with your classmates; however ensure to include their names in the report/code documentation. If you refer to any source on the Internet, include the corresponding citation in the report/code documentation. If we find that you have copied code from your classmate or from the Internet, you will get a straight fail grade in the course.
- The submission must be a zip file with the following naming convention - rollnumber.zip.
- Include appropriate comments to document the code. Include a `read me` file containing the instructions on for executing the code. The code should run on institute linux machines.
- Upload your submission to moodle by the due date and time. Do not email the submission to the instructor or the TA.

This lab will improve your understanding of working with arrays, function calls, switch statements, and file i/o.

1 Image Manipulations (10 points)

In this lab, we will explore the fascinating world of image processing and learn how to apply various data structures and algorithms to manipulate images. Images play a crucial role in data science, and understanding how to perform operations on them is essential for analyzing and extracting meaningful information. Throughout this lab, we will focus on three fundamental image operations: flip, scale, and translation. By implementing these operations using Python and working with arrays, we will gain insights into how to mirror, resize, and shift images. We will use numpy arrays to store and manipulate image matrices and PIL for Image I/O operations.
The attached skeleton code should help you get started quickly.

```python

import numpy as np
from PIL import Image

def translate_image(image, x_offset , y_offset):
    # Translate the image by the given offsets
    # Implement the translation logic using numpy arrays
    # Return the translated image as a numpy array

def flip_image(image, axis):
    # Flip the image along the specified axis (horizontal or vertical)
    # Implement the flipping logic using numpy arrays
    # Return the flipped image as a numpy array

def scale_image(image, scale_factor):
    # Scale the image by the given factor
    # Implement the scaling logic using numpy arrays
    # Return the scaled image as a numpy array

# Read the image file names and the operations to perform from the input file.
# Each line contains the image file name and the operation separated by a space.
# The operation is followed its parameters.
# Process each line to read the image, perform the operation and store the output image.
# Name the output file after the input file name appended with the string 'op'.
```

Code Fragment 1: Queue Interface

1.1 Image I/O operations

When working with images in Python, the Python Imaging Library (PIL) is a powerful tool that provides a wide range of functionalities, including image input and output (I/O) operations. PIL allows us to read and write various image file formats, making it easy to load images into our programs and save the results of image manipulations. By using PIL, we can read images from files, manipulate them using data structures and algorithms, and then save the modified images back to disk. With its intuitive and straightforward syntax, PIL simplifies the process of handling image I/O tasks, enabling us to focus on the core aspects of image processing and analysis.
In the code snippet below, we first use Image.open() from the PIL library to open the image file and then convert it into a numpy array using np.array() [2]. We can then perform various image manipulations using numpy array operations. Finally, to save the modified image, we use Image.fromarray() to create a PIL Image object from the numpy array, and then call the save() method to save it as an image file.

```
import numpy as np
from PIL import Image

# Read an image file into a numpy array
image_file = 'input_image.jpg'
image = np.array(Image.open(image_file))

# Perform image manipulations using numpy array operations
# ...

# Write a numpy array back to an image file
output_file = 'output_image.jpg'
result_image = Image.fromarray(image)
result_image.save(output_file)
```

Code Fragment 2: PIL Image I/O

1.2 Translate

(3 points) The translate operation is an image transformation technique that involves shifting the image by given x and y offsets. Translation is commonly used to reposition an image within a coordinate system. By applying a translation, we can effectively move the image horizontally and vertically. The implementation logic for translation typically consists of iterating over the pixels in the image and calculating the new coordinates for each pixel based on the specified x and y offsets. The pixel values are then copied to their new positions in the resulting image. Translation allows us to adjust the spatial positioning of the image, such as moving it within a frame or aligning it with other elements. This operation is relatively straightforward to implement. Here are the steps to perform the translate operation.

1. Create a function `translate_image(image, x_offset, y_offset)` that takes the image array, x offset, and y offset as input.

2. Create a new numpy array of the same dimensions as the original image to store the translated image and initialize it to some constant value (say 0).

3. Iterate over each pixel in the original image and copy the corresponding pixel to the translated image array based on the specified offsets. If x and y represent the coordinates of a pixel in the original image, then the coordinates for the translated pixel are given by the following equations

$$translated\_x = x + \text{x\_offset} \tag{1}$$
$$translated\_y = y + \text{y\_offset} \tag{2}$$

   Remember to check if the translated coordinates fall within the translated image bounds.

4. Return the translated image as a numpy array.

1.3 Flip

(3 points) The flip operation is a common image manipulation technique that involves mirroring the image along a specified axis, either horizontally or vertically. Flipping an image can be useful for various purposes, such as correcting orientation or creating symmetrical visual effects. The implementation logic for flipping typically consists of identifying the desired axis of flip, iterating over the pixels in the image, and swapping the corresponding pixels to achieve the mirroring effect. When performing a horizontal flip, the pixels on the left side of the image are swapped with their corresponding pixels on the right side. Similarly, in a vertical flip, the pixels on the top side of the image are swapped with their corresponding pixels on the bottom side. Use the following steps to create the code for flipping an image.

1. Create a function `flip_image(image, axis)` that takes the image array and axis (horizontal - 0 or vertical - 1) as input.
2. Create a new numpy array of the same dimensions as the original image to store the flipped image.
3. Iterate over each pixel in the original image and copy the corresponding pixel value to the flipped image array based on the specified axis. If x and y represent the coordinates of a pixel in the original image of size *width* and *height*, then the equations for vertical and horizontal flips are

$$horizontal\_flip\_x = width - 1 - x \tag{3}$$
$$horizontal\_flip\_y = y \tag{4}$$
$$vertical\_flip\_x = x \tag{5}$$
$$vertical\_flip\_y = height - 1 - y \tag{6}$$

4. Return the flipped image as a numpy array.

1.4 Scale

(2 points) The scale operation is a fundamental image transformation technique that involves resizing the image by a given factor. Scaling is commonly used to adjust the size of an image while preserving its aspect ratio. By applying scaling, we can enlarge or reduce the dimensions of the image. The implementation logic for scaling typically involves iterating over the pixels in the original image and mapping them to new positions in the resulting image based on the scaling factor. Each pixel's coordinates are multiplied by the scaling factor to determine its new position. The pixel values are then interpolated or extrapolated to fill the new image dimensions. The following steps will assist in implementing the scaling function.

1. Create a function `scale_image(image, scale_factor)` that takes the image array and scale factor as input.
2. Calculate the new dimensions of the scaled image by multiplying the original dimensions with the scale factor.

3. Create a new numpy array of the calculated dimensions to store the scaled image.

4. Iterate over each pixel in the new array and apply the scaling transformation to obtain the corresponding pixel from the original image. If x and y represent the coordinates of a pixel in the original image, then the coordinates for the scaled pixel are given by the following equations

$$scaled\_x = \text{scale\_factor} \cdot x \tag{7}$$

$$scaled\_y = \text{scale\_factor} \cdot y \tag{8}$$

5. Return the scaled image as a numpy array.

1.5 Input and Output

The input to your code will be an input.txt (check the file in the lab folder). Each line of this file will provide the image and the operations to be performed. You will have to store the transformed image using this naming format - original image name-transformation-value.jpg

2 References

1. M Goodrich, R Tamassia, and M. Goldwasser, “Data Structures and Algorithms in Python”, 1^{st} edition, Wiley, 2013.

2. https://numpy.org/doc/stable/reference/generated/numpy.array.html