CS 763/CS 764: Lab 01

Warmup

- Announced 15/01. Due 20/01 23:29
- Please write (only if true) the honor code. You can find the honor code on the web page. If you used any source (person or thing) explicitly state it.
- **This is an individual assignment**

1 Overview

In this lab, we will learn (or revise) programming in python and the use of numpy and openCV libraries. Each topic has a pointer to a relevant tutorial and some practice problems that you have to solve and turn in.
Be sure to read a companion document which lists the submission guidelines (among other things).

1.1 Python

Just to recap, we will use Python 3.

Tutorial

Find a tutorial on Python here. We agree that this is a long tutorial, but your interests are best served by reading a section in every lab.

Practice

Test your understanding by practising the below problems. Only use Python Standard Library. Use argparse for arguments to the script.

1. P-norm: The p-norm of a vector $v = [v_1, v_2, ..., v_n]$ in n-dimensional space is defined as

$$\|v\| = \sqrt[p]{|v_1|^p + |v_2|^p + ... + |v_n|^p}$$

Provide an implementation of a function named `norm` such that `norm(v, p)` returns the p-norm value of v and `norm(v)` returns the Euclidean norm (p = 2) of v. You may assume that v is a list of numbers.

Intention: We are expecting the program to run the program as:

```
$ python3 p_norm.py 2.3 21 4 1 --p 3
Norm of [2.3, 21.0, 4.0, 1.0] is 21.06

$ python3 p_norm.py 2.3 21 4 1
Norm of [2.3, 21.0, 4.0, 1.0] is 21.52
```

The input vector can have any number of components (obviously). While printing, output 2 decimal places (don't worry about rounding).
[Hint: Look up the documentation for argparse]

1.2 Numpy

Numpy is used to represent n-dimensional arrays. This is an understatement; it is ubiquitous in scientific programs, especially in deep learning.

Tutorial

Find a tutorial on Numpy here.

Practice

Test your understanding by solving the following problems. Use only Python Standard Library, Argparse and Numpy Library.

1. Row Manipulation Given a 1D array, we would like to be ready to convert it into another 1D array where the elements $(a_1, a_2, a_3, a_4, a_5, .. ., a_{2n}, a_{2n+1})$ are rearranged to form $(a_1, a_3, a_5, ..., a_{2n+1}, a_2, a_4, a_6, ..., a_{2n})$ i.e the numbers at odd position are transferred at the beginning and the even numbers are placed at the end.
We accomplish this using a permutation matrix (how?). An example for $N = 5$ is as shown below.

```
P =   [1, 0, 0, 0, 0]
      [0, 0, 1, 0, 0]
      [0, 0, 0, 0, 1]
      [0, 1, 0, 0, 0]
      [0, 0, 0, 1, 0]
```

1. Your task will be to create this matrix given the number N, (ideally without the use of for loops).
2. Implement a function `crop_array(arr_2d, offset_height, offset_width, target_height, target_width)` that cuts a rectangular part out of the 2-dimensional array. The top-left corner of the returned array is at `(offset_height, offset_width)` and its lower-right corner is at `(offset_height + target_height, offset_width + target_width)`.
3. Padding is a common operation in image processing. Pad the value 0.5 increasing the row size by 2, one each for the top and bottom. Similarly do this on the left and right. Pay attention to `ndarray.dtype`.
4. Concatenate the above padded `arr_2d` with its replica such that the two arrays are placed side-by-side.
5. Print the original, cropped, padded and concatenated arrays.

Intention: We are expecting the program to run as follows:

```
$ python3 row_manipulation.py --N 4
Original array:
 [[1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]]

Cropped array:
 [[0. 1.]
 [1. 0.]]

Padded array:
 [[0.5 0.5 0.5 0.5 0.5 0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5]
 [0.5 0.5 0.  1.  0.5 0.5]
```

```
 [0.5 0.5 1.  0.  0.5 0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5]]

Concatenated array: shape=(6, 12)
 [[0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]
 [0.5 0.5 0.  1.  0.5 0.5 0.5 0.5 0.  1.  0.5 0.5]
 [0.5 0.5 1.  0.  0.5 0.5 0.5 0.5 1.  0.  0.5 0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5]]
```

[Note: Try to match the output exactly as shown above]

2. k-means Naive k-means is a standard clustering algorithm for unsupervised learning. The observations are in $d >= 1$ dimensions.

Questions. With reference to the algorithm here here, but fixed k (write answers in `answers.pdf`)

1. How do you characterize the termination of the algorithm, if, say, $k = 3$? The question is deliberately open-ended since overspecifying the question gives the game away but we are referring to this statement The algorithm has converged when the assignments no longer change.
2. For the same input data, can two different valid implementations generate different solutions, assuming both converge?

Implementation You are given a stub for a k-means algorithm and you need to flesh out the stub without (obviously) using a library implementation of k-means. Here are the constraints:

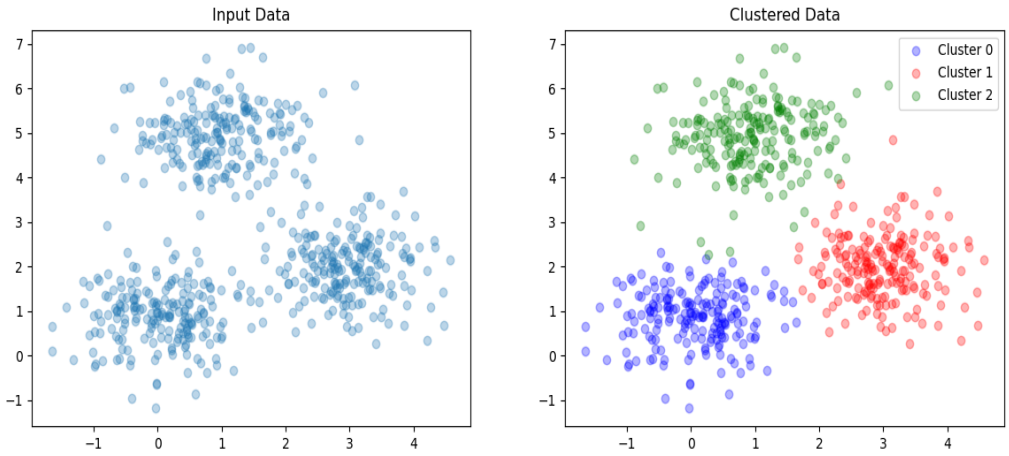

Figure 1: Expected results of the k-means algorithm for the input specified.

1. Use only `numpy` and for plotting `matplotlib` as python libraries
2. The stub has a function `fit()` which repeatedly performs both steps. Write your code for these between the (provided) "TODO" clauses.
3. An important goal of numpy is to vectorize loops. Your goal should be to avoid unnecessary explicit `for` loops.
4. Note that your code should work for data of any reasonable dimensionality d and not just 2D
5. The number of iterations in your implementation should be such that the program terminates a couple of seconds
6. Generate a dataset of 200 normally distributed random 2D points (aka 2D features) around each of three points $(3, 2)$, $(1, 5)$ and $(0, 1)$. The expected input is shown!

7. Plot the data points and the final 3 clusters (using a scatter plot) and save the plot to file in the data directory as '`../results/inputCluster.png`' and '`../results/outputCluster.png`' respectively as shown. [Saving should be automatically done by the code]. The expected output appears in Fig. 2 !

Intention: We are expecting the program to run the program as:

```
$ python3 kmeans_tester.py
```

The tester (not provided) will invoke the class shown in the stub.

1.3 OpenCV

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of Numpy.

Tutorial

Go through the following OpenCV tutorials:

1. Getting Started with Images
2. Getting Started with Videos

Practice

Test your understanding by practising the below problems. You can only use Python Standard Library, Numpy Library and OpenCV-Python (Version 3.4.2).

1. Image Conversion:

1. Read a color image into a numpy array. (Note: OpenCV uses BGR channel format for image input/output. It reads into a `uint8` type array.)
2. Normalize the pixel values so that they lie in `[0, 1]`.
3. Plot both images (original and normalized) using matplotlib, and show/save the image. (Matplotlib uses RGB channel format.)
4. Plot both images (original and normalized) using opencv, and show/save the image. (Note: OpenCV `cv.imshow` requires image array to be in `uint8` type.)

   Intention: We are expecting the program to run as follows:

   ```
   $ python3 image_conversion.py path_to_image
   ```

2. Display Images:

1. Download several (atleast 5) images of your choice and place them in the data directory. [Rename them as 'display00.png', 'display01.png'... and so on]
2. Read the images into a numpy array.
3. Display the first image in a window.
4. Press 'n' to display the next image and 'p' to display to the previous image in the same window. Pressing 'n' at the last image displays the first image, and similarly pressing 'p' at the first image displays the last image. Thus a wrap around.

Intention: We are expecting the program to run as follows:

```
$ python3 display_images.py path_to_directory_containing_images
```

(a) Plotting images using matplotlib

(b) Plotting images using opencv

Figure 2: Image conversion

3. Video Input/Output:

1. Read your webcam input. (If your webcam is not working, you can use play video from this file.)
2. While displaying any video in the further steps, annotate the videos with `"<Your_Name>"` bounded by a rectangular box and placed at the top right corner. (Optional: Try bounding it without explicitly entering the dimensions of the box)
3. Display it on a opencv window.
4. Also display the grayscale version of the same in another window.
5. Press 'q' to quit. Note: Both windows should be shown simultaneously beside each other. Use `cv.moveWindow` for window placement.

Intention: We are expecting the program to run as follows:

```
$ python3 display_video.py path_to_video_file
```

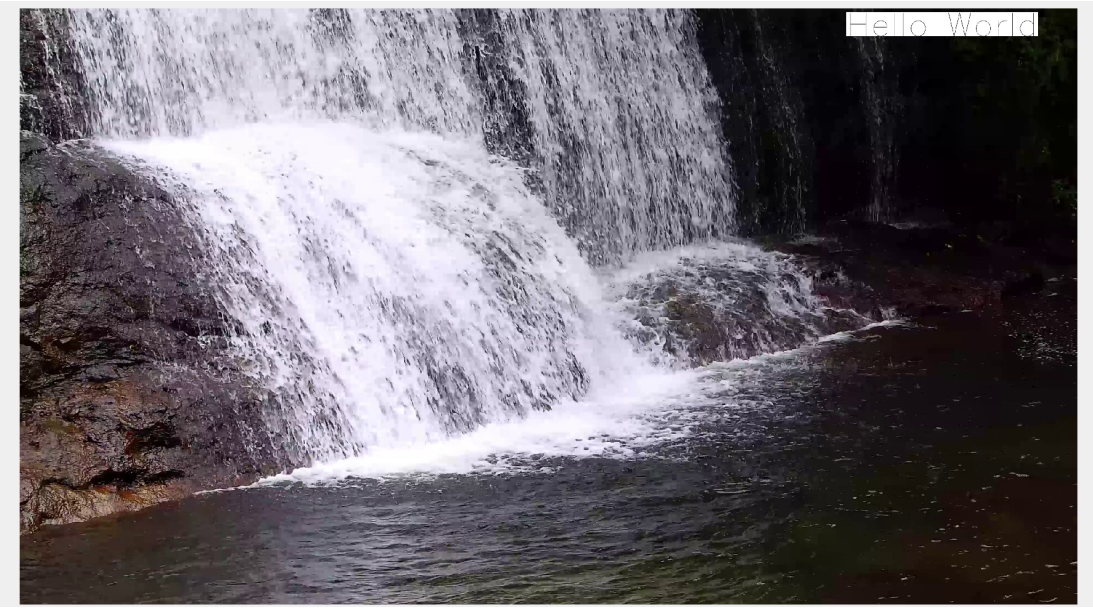

Figure 3: The annotation should be placed as shown in above figure

Submission Guidelines

1. The top assignment directory should contain the submission as detailed below.

(a) Do include a `readme.txt` (telling me whatever you want to tell me including any external help that you may have taken). Don't forget to include your honor code here . All members of a group are expected to (electronically) sign the honor code and the percentages (see below). This is a text file, not `pdf`, not `docx`. Do not change the name of the file to REadme.txt or README.txt or README and so on. This is not the place to show creativity – the attention to details might be minor for you but little drops make an ocean.

The `readme.txt` will contain individual contributions of each of the team members. If the assignment is finally worth 80% as graded by the TA, then a contribution of the form `80, 100, 60` (in sorted order of roll numbers) will result in (respectively) marks `64, 80, 48`. Do this for each question separately. A person claiming 100% is basically saying that (s)he can reproduce the entire assignment without the help of the other team members (after the discussions, if any, are over).

(b) `answers.txt`: This file should contain the answers to the questions posed.

(c) `ReflectionEssay.pdf`: This file does not contain answers, but meta-answers. It provides a human perspective of what you learned by doing this. It is not intended to be mechanical, but rather an introspection, like a blog post. Explain what you learnt in this assignment, and how this assignment improved your understanding. What will someone who read this pdf gain? Can this be a blog post, which if read end-to-end someone not in your class (but in your batch) will understand? (Not all of this is relevant to this particular assignment).

(d) A directory called `code` which contains all source files, and only source files (no output junk files). The mapping of code file to questions should be obvious and canonical.

(e) A directory called `data` on similar lines to code, whenever relevant. Note that your code should read your data in a relative manner.

(f) A directory called `results` on whenever relevant to store the concerned plots and images.

(g) Source files, and only source files (no output junk files). Mac users please don't include junk files such as `.DS_store`. We are not using MacOS.

(h) Create a directory called `convincingDirectory` which contains anything else you want to share to convince the grader that you have solved the problem. We don't promise to look at this (especially if the code passes the tests) but who knows? This is your chance.

2. Once you have completed all the questions and are ready to make a submission, prepend the roll numbers of all members in your group to the top assignment directory name and create a submission folder that looks like (for group but you get the idea) this `130010009_140076001_150D50001_lab0X.tar.gz`. Notice the numbers are sorted, and any letters in the roll number are in Upper Case.

   Please stick to `.tar.gz`. Do not use `.zip`. Do not use `.rar` Do not use `.tgz`

3. Your inlab submission folder should look something like:

```
130010009_140076001_150050001_lab01b/
├── ReflectionEssay.pdf
├── answers.pdf
├── python
│   ├── code
│   │   └── p_norm.py
│   ├── data
│   ├── results
│   └── convincingDirectory
├── numpy
│   ├── code
│   │   ├── row_manipulation.py
│   │   └── kmeans.py
│   ├── data
│   ├── results
│   │   ├── inputCluster.png
│   │   └── outputCluster.png
│   └── convincingDirectory
├── opencv
│   ├── code
│   │   ├── image_conversion.py
│   │   ├── display_images.py
│   │   └── display_video.py
│   ├── data
│   ├── results
│   └── convincingDirectory
└── readme.txt
```

4. Submission. Very very important.

   (a) The lexicographic smallest roll number in the group should submit the entire payload (with all the technical stuff).

   (b) All other roll numbers submit only `readme.txt` as discussed above.