CS 763/CS 764: Lab 01

Warmup

- Announced 26/01. Due 02/02 9AM

Overview

The lab consists of two parts. Setting up the environment, and then learning via doing.

1 Setup

We will setup a Python-based development virtual environment. The purpose of this is to isolate your changes from other things going on in your system.
As an aside, you might have heard about `anaconda` which also supports a python-based virtual environment. We won't need the entire kitchen sink that `anaconda` offers (see, e.g., this page for the differences). You can always install it later. Note that lab submissions may be checked in a vanilla (non-Conda) environment, so it is to your advantage to know what you are using and doing at the basic level. We are also not discussing Python coding environments (e.g. spyder, ipython, pycharm and so on).
Choose the installation steps according to your operating system.

- MacOS
- Ubuntu
- Windows

1.1 MacOS

The following steps are for MacOS 10.12.6 (Sierra) or later (64-bit). [Not fully tested]

Install Python and Virtualenv

The following steps will install Python, pip package manager and Virtualenv. Install using the Homebrew package manager.

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
    master/install)"
$ export PATH="/usr/local/bin:/usr/local/sbin:$PATH"
$ brew update
$ brew install python  # Python 3
$ sudo pip3 install -U virtualenv
```

Create a Virtual Environment

Python virtual environments are used to isolate package installation from the system.
Create a new virtual environment by choosing a Python interpreter and making a venv directory to hold it:

```
$ virtualenv --system-site-packages -p python3 venv
```

Activate the virtual environment:

```
$ source venv/bin/activate
```

When virtualenv is active, your shell prompt is prefixed with (venv).
Install packages within a virtual environment without affecting the host system setup. Start by upgrading pip. :

```
$ pip install --upgrade pip
```

And to exit `virtualenv` later:

```
$ deactivate # only exit when not using the environment
```

Install Libraries

Install Numpy, Matplotlib and OpenCV-Python :

```
$ pip install numpy matplotlib opencv-python==3.4.2.16 opencv-contrib-python
    ==3.4.2.16
```

The `SIFT` algorithm is a popular algorithm that we might use but is patented. It is not included in newer versions of opencv-python.

1.2 Ubuntu

The following steps are for Ubuntu 16.04 or later (64-bit).

Install Python and Virtualenv

The following steps will install Python, pip package manager and Virtualenv.

```
$ sudo apt update
$ sudo apt install python3-dev python3-pip
$ sudo pip3 install -U virtualenv
```

Create a Virtual Environment

Python virtual environments are used to isolate package installation from the system. Create a new virtual environment by choosing a Python interpreter and making a venv directory to hold it:

```
$ virtualenv --system-site-packages -p python3 venv
```

Activate the virtual environment :

```
$ source venv/bin/activate
```

When `virtualenv` is active, your shell prompt is prefixed with (venv).
Install packages within a virtual environment without affecting the host system setup. Start by upgrading pip.

```
$ pip install --upgrade pip
```

And to exit virtualenv later:

```
$ deactivate # only exit when not using the environement
```

Install Libraries

Install Numpy, Matplotlib and OpenCV-Python:

```
$ pip install numpy matplotlib opencv-python==3.4.2.16  opencv-contrib-python
    ==3.4.2.16
```

The `SIFT` algorithm is a popular algorithm that we might use but is patented. It is not included in newer versions of opencv-python.

1.3 Windows

The following steps are for Windows 7 or later (64-bit). [Not fully tested]

Install Python and Virtualenv

The following steps will install Python, pip package manager and Virtualenv. Install the *Microsoft Visual C++ 2015 Redistributable Update 3*. This comes with Visual Studio 2015 but can be installed separately:

1. Go to the Visual Studio Downloads
2. Select Redistributables and Build Tools
3. Download and install the *Microsoft Visual C++ 2015 Redistributable Update 3*

Make sure long paths are enabled on Windows. Install the 64-bit Python 3 release for Windows (select pip as an optional feature).

```
$ pip3 install -U virtualenv
```

Create a Virtual Environment

Python virtual environments are used to isolate package installation from the system. Create a new virtual environment by choosing a Python interpreter and making a venv directory to hold it:

```
$ virtualenv --system-site-packages -p python3 venv
```

Activate the virtual environment.

```
$ venv\Scripts\activate
```

When virtualenv is active, your shell prompt is prefixed with (venv).
Install packages within a virtual environment without affecting the host system setup. Start by upgrading pip.

```
$ pip install --upgrade pip
```

And to exit virtualenv later:

```
$ deactivate # only exit when not using the environement
```

Install libraries

Install Numpy, Matplotlib and OpenCV-Python:

```
$ pip install numpy matplotlib opencv-python==3.4.2.16  opencv-contrib-python
    ==3.4.2.16
```

The `SIFT` algorithm is a popular algorithm that we might use but is patented. It is not included in newer versions of opencv-python.

2 Learn

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.

2.1 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]

2.2 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 $(a_1, a_3, a_5, ..., a_{2n+1}, a_2, a_4, a_6$ 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. Shape Manipulation: Given a numpy array `arr_2d` of size $H * W$ (to be read from a file) and two additional numbers M and N (to be read from stdin) we have to create an array in which

all the individual elements of the array `arr_2d` are replaced by an $M * N$ sized matrix with that element repeated $M * N$ times.

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

```
# Let arr_2d be [[0,1],[2,3],[4,5]]

$ python3 shape_manipulation.py path_to_grid_file
M = # Ask for input M
N = # Ask for input N
[[[[0 0 0]
   [0 0 0]]
  [[1 1 1]
   [1 1 1]]]
 [[[2 2 2]
   [2 2 2]]
  [[3 3 3]
   [3 3 3]]]
 [[[4 4 4]
   [4 4 4]]
  [[5 5 5]
   [5 5 5]]]]
```

[Note: Newline characters generated while printing multi-dmensional numpy arrays are acceptable; do not try to remove them]

3. Principal Component Analysis: Principal Component Analysis, or PCA, is a popular dimensionality reduction technique. [Although we believe you have studied PCA earlier, you can revise by visiting here]

1. Read a D dimensional dataset of N datapoints from a .txt file and project it onto a 2-dimensional space (N, D can be variable). The file will have N rows, with each row containing D comma-separated values. There will be no whitespaces in this file.
2. Plot the projected data (using a scatter plot) and save the plot to file in the data directory as 'out.png' [This saving to file should be automatically done by the code]. While plotting, ensure both the x and y axis have the same aspect, and show values from [-15,15].

Notes: In addition to numpy and argparse you may use matplotlib in this question. Do not use `scikit` or other similar packages.
Look up eigenvalue decomposition functions in `numpy`. Are there multiple functions that can perform such a decomposition? What is the difference between them?

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

```
$ python3 pca.py path_to_data_file
```

2.3 OpenCV

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of Numpy. We will use OpenCV 3.4.2.

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
```

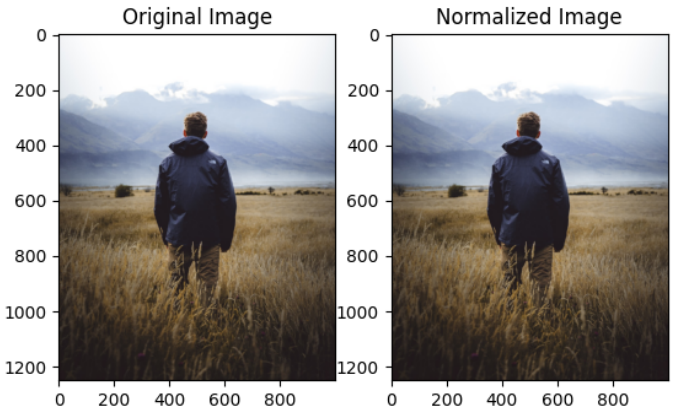

Figure 1: Plotting images using matplotlib

Figure 2: Plotting images using opencv

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
```

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
```

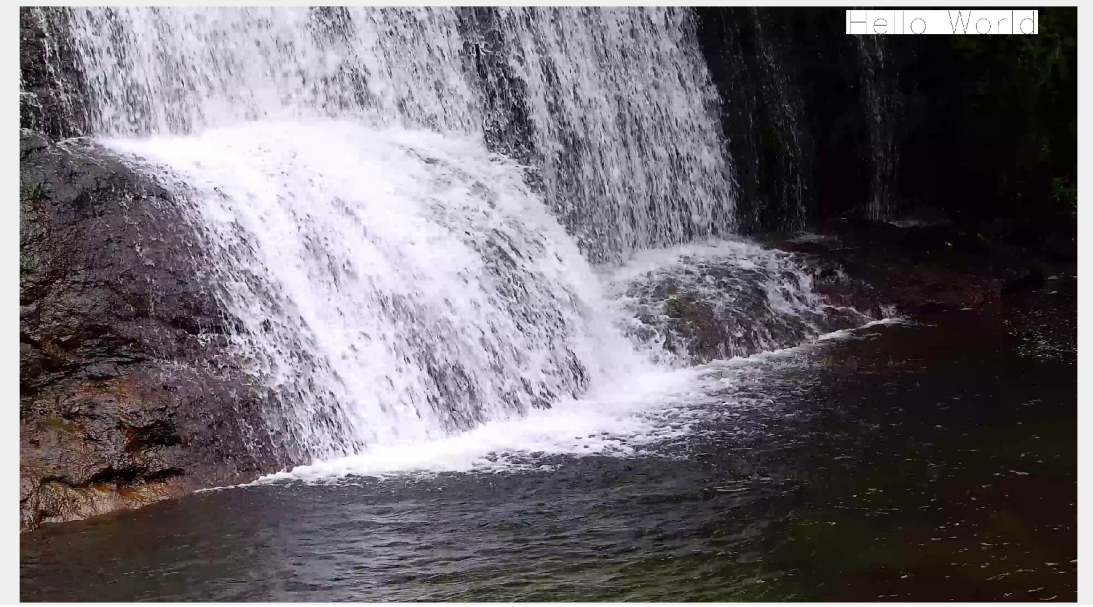

Figure 3: The annotation should be placed as shown in above figure

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`.

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.

(b) `ReflectionEssay.pdf`: Should contain the explanation for all the questions implicitly and explicitly raised. Provide an output of a sample run. Explain what you learnt in this assignment, and how this assignment improved your understanding. What will someone who reads this gain? Can this be a blog post, which if read end-to-end someone not in your class (but in your batch) will understand?

(c) 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.

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

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

(f) 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.

(g) 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 `150050001_130010009_140076001_lab0X_description.tar.gz`

   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:

```
150050001_130010009_140076001_lab01/
├── ReflectionEssay.pdf
├── python
│   ├── code
│   │   └── p_norm.py
│   ├── data
│   ├── results
│   └── convincingDirectory
├── numpy
│   ├── code
│   │   ├── row_manipulation.py
│   │   ├── shape_manipulation.py
│   │   └── pca.py
│   ├── data
│   ├── results
│   │   └── out.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 smalles roll number (on google classroom) 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.