The University of Saskatchewan
Saskatoon, Canada
Department of Computer Science

487/819– Computer Vision and Image Processing

Assignment 0

Date Due: September 26, 2018, 9:00pm

Total Marks: 27

1 Submission Instructions

- Assignments are to be submitted via the Moodle class page.
- Programs must be written in Python 3.
- No late assignments will be accepted. See the course syllabus for the full late assignment policy for this class.

2 Background

This section contains background information on Python, the modules we'll be commonly using, and some specific functions from those modules that you'll need to use to complete your assignment.

2.1 Python

If you don't already know Python, you are responsible for learning the basics on your own. The CMPT 141 course readings are provided which provide a very gentle introduction to Python. It is suggested you read chapters 2–9 and 13 at a minimum. It's not as much reading as it sounds as there are lots of examples and illustrations.

We will do our Python programming using the PyCharm IDE. This is available from jetbrains.com. The Community Edition is free. The Professional Edition is also free, but you have to create a jetbrains account and request a free educational license. You won't need the Professional Edition but you can get it if you want to. Instructions for how to install PyCharm on your own computer are on the course website. If installing your own, I recommend the Community Edition. If you want to use the Professional Edition, it is available pre-installed in the Spinks labs. There are special instructions for first-time setup of the Professional Edition on the lab machines on the course website.

For this course, you won't notice any difference between the Professional and Community editions.

2.2 iPython Notebooks

PyCharm allows you to load and use iPython notebooks in a convenient fashion. iPython notebooks are documents which allow one to interleave executable blocks of code and non-executable blocks of text (actually HTML written in a simplified markdown language). When you execute a code block, its output appears underneath the code block.

When you open `asn0-q1.ipynb` in PyCharm, you'll see that it has first a couple of text blocks, and then a code block with some inital code, and some comments that describe some things you have to do. We'll have a short demonstration in class of how to use iPython notebooks. In the Appendix A of this document you'll find a screenshot that points out important elements of an iPython notebook.

All of the code blocks of a notebook all share a persistent variable namespace, as if everything you do is run in the same Python command-line shell session. If you run a block of code, then all of the variables defined by that block are visible to both later and **earlier** blocks of code. If you run a code block that defines a variable, and you later remove that variable from the code entirely, that variable will remain defined until you either restart PyCharm, or restart the iPython kernel. Generally, the notebooks I provide you will be designed such that the code blocks are intended to be executed in order from top to bottom.

2.3 Frequently Used Python Modules

In this course we will frequently be using the following Python modules:

skimage (scikit-image) Provides functions that provide basic image I/O, manipulation, and common image processing algorithms.

matplotlib Provides basic image display and sophisticated creation of plots and figures.

numpy Python's de-facto standard array module. Scikit-image and matplotlib use images stored as numpy arrays.

As a convention we will almost always import parts of these modules as follows:

```
import matplotlib.pyplot as plt
import skimage.util as util
import skimage.io as io
import numpy as np
```

`matplotlib.pyplot` is a module within `matplotlib` that contains all of the functions for making figures and plotting graphs. There are other modules within matplotlib, but we'll use this one the most often.

`skimage` contains several submodules that we will use, and `io` is one of them. We'll typically import just the submodules of `skimage` that we need rather than the whole thing. We'll almost always need `skimage.io` because this defines the functions for reading and writing image files. The `skimage.util` module contains functions for converting between images of different data types. More about this shortly. In class we will talk/have talked about another of `skimage`'s submodules, namely `skimage.color` which contains the functions for colour space conversions and conversions from RGB to grayscale.

`skimage` and `matplotlib` already import `numpy` and so functions in these modules are aware of, and may return numpy arrays. However, if we want to call functions in `numpy` ourselves, then we need to import it.

2.4 `numpy` Arrays as Images

If you've taken CMPT 141, you already have had some practice with `numpy` arrays.[1] Anything you already know how to do to a `numpy` array, such as slicing, arithmetic on entire arrays, etc. can be done to images because images are just stored in `numpy` arrays. However, there are some conventions that `numpy` arrays that store images must adhere to.

Grayscale images must be 2D arrays (the `ndim` attribute of the array must be 2). The first dimension of the array is the row, the second is the column. Thus, `A[y,x]` addresses the pixel at row `y`, column `x`. The element at row `y`, column `x` of such an array is the intensity value (brightness) of the pixel (x, y). Since images are stored in arrays, the origin of the image pixel coordinate system is at the top-left of the image, and the positive y-axis points **down** (as opposed to up, as we normally view it in mathematics).

Colour RGB images are 3D arrays (the `ndim` attribute of the array must be 3). The first two dimensions are row and column, respectively, just as in grayscale images. The third dimension indexes the channel of

[1]If you have experience in MATLAB, you'll find that numpy arrays are extremely similar to MATLAB matrices in their functionality. Almost anything you can do with a MATLAB matrix you can do in a very syntactically similar (but not identical) fashion with a `numpy` array.

the image. Channel 0 is the red channel, channel 1 is the green channel, and channel 2 is the blue channel. Thus, `A[y,x,0]` is the value of the red channel at row `y`, column `x`.

`numpy` arrays, and therefore images, can have different data types, depending on the array's `dtype` attribute. Usually images are data types `uint8`, `uint16`, `float64` or `bool_`. The data type determines how the values are interpreted. For grayscale images, the following table summarizes this:

Image `dtype`	Actual Range of Type	Black Value	White Value
`uint8`	0–255	0	255
`uint16`	0–65535	0	65535
`float64`	$10^{-308} - 10^{308}$	0.0	1.0
`bool_`	`False` and `True` only.	`False`	`True`

For the unsigned integer types, the minimum value is black, the maximum value is white, and the grayscales are in between. For `float` type, black is 0.0, white is 1.0, and the grayscales are in between. Numbers larger than 1.0 are interpreted as white, numbers smaller than 0.0 are interpreted as black. A common mistake is to try to use an array whose values are floating point between 0.0 and 255.0. There's nothing stopping you from doing this, but when go to display it, such an image will appear completely white since most of its values will be larger than 1.0.

RGB images work exactly the same way. For integer types, 0 represents no R, G, or B, the maximum value represents 100% R, G, or B. For type `float`, 0.0 represents no R, G, or B, 1.0 represents 100% R, G, or B. Values smaller than 0.0 are interpreted the same as 0.0; values larger than 1.0 are interpreted as 1.0. RGB images cannot have the `bool_` data type.

Binary images are stored in numpy arrays with the data type `bool_`. Such images only have two intensities. The value `False` is interpreted as black, and the value `True` is interpreted as white.

Other data types, including signed integers can be used as images too, but they're interpreted differently yet again. For example, signed 8-bit grayscale image arrays are interpreted with -128 being black, 0 being 50% gray, and +127 being white.

2.5 Reading and Writing Image Files

Images can be read using the `skimage.io.imread()` function. In the most basic usage, the only argument required is a filename. The image file format will be determined by the filename extension, and read appropriately, and the image pixel data will be returned as an array. The data type of the array returned will depend on the file format and/or the particular variant of that file format. The dimensionality of the array returned will depend on whether the image file is grayscale or RGB. Unless your image is from an unusual source, the data type returned is almost always `uint8`, but watch out for exceptions!

Here's an example:

```python
# The skimage.io module defines imread()
import skimage.io as io

# Read the file myimage.png from the current folder.
# After executing this, the variable myimage refers to
# an array containing the image data.
myimage = io.imread('myimage.png')
```

The function `skimage.io.imsave()` function writes the pixel data in a `numpy` array to an file. The file name and the array must be provided as arguments. The file format written is determined by the extension of the given filename, and the file will be written with the pixel data stored as the same data type as that of the array if the file format supports that data type. Here's an example you can try out if you like:[2]

[2] `numpy.zeros()` requires one argument that is a two-element tuple or list containing the height and width of the array. A new

```
# The skimage.io module defines imsave()
import skimage.io as io
import numpy as np

# Create a 256 by 256 grayscale iamge
myimage = np.zeros((256,256), dtype=np.uint8)
for i in range(256):
    myimage[:,i] = i

# Save it to a file called gradient.png.
io.imsave('gradient.png', myimage)
```

2.6 Creating Figures

Figures are created using the function `matplotlib.pyplot.figure()`. When you create a figure, it becomes the *current figure*, and any subsequent "drawing" (image display, plotting) or figure modifications are applied to the current figure. `matplotlib.pyplot.figure()` returns a *handle* for the created figure which can be retained if you wish to make changes to the figure later when it is no longer the current figure.

To display an image in the current figure, use the `matplotlib.pyplot.imshow()` function. In its most basic usage, only the array containing the image to be displayed is provided as a argument. If you call `imshow()` and there is no current figure window (e.g. if no figures at all have been created yet), then a new figure is created (as if you called `matplotlib.pyplot.figure()`) and then the image is displayed in that figure. If there is already current figure (or subplot, see below) and you want to display an image in a new figure, you must call `matplotlib.pyplot.figure()` (or `matplotlib.pyplot.subplots()`, see below) explicility to make a new figure.

Here's an example:

```
# The skimage.io module defines imread()
import skimage.io as io
import matplotlib.pyplot as plt

# Read the file myimage.png from the current folder.
# After executing this, the variable myimage refers to
# an array containing the image data.
myimage = io.imread('myimage.png')

# display myimage
myfigure = plt.figure() # note: saving the handle for later if we need it
plt.imshow(myimage)
plt.title('This␣is␣My␣Image')

# display another image.  If we don't call plt.figure() first, the next
# call to plt.imshow() will overwrite the earlier image in the current
# figure becuase the current figure will not have changed.
secondfigure = plt.figure()
plt.imshow(myimage > 128)
```

If you are working in an iPython notebook, figures are automatically shown below the code block in which they are created as long as the `%matplotlib inline` directive has been issued. This is included for

array is returned having the specified height and width and containing entirely zeros. The optional `dtype` argument selects the data type of the returned array which defaults to `float`.

you in the given iPython notebooks. If you are using plain Python instead of an iPython notebook, you must call the `matplotlib.pyplot.show()` function in order for figures to be displayed.

There are many functions that can modify figures. We'll introduce just one now, which is the function `matplotlib.pyplot.title()`. This function requires one string argument. The provided string is displayed as the title of the figure, and appears above the figures axes. As you can see in the example above, the `title()` function is called to add the title "This is My Image" to the figure.

2.7 Multiple Images in Figure (Subplots)

You can display multiple images in one figure using the `matplotlib.pyplot.subplots()` function. The arguments to subplots are the number of rows and columns of subplots you want, e.g. `plt.subplots(1,2)` would create a pair of side-by-side figures (one row of two columns of figures). You can select a subplot to be the current figure using the `matplotlib.pyplot.subplot()` function. The `subplot()` function takes 3 arguments, the first two of which are the number of rows and columns of subplots in the figure (the same values you passed to `subplots()`). The third argument is the index of the subplot you want. Subplots are indexed starting from index 1, and are numbered in row-major order, thus, the top-right subplot is subplot #1, the subplot to the left is subplot #2, and so on. When the end of a row is reached, numbering continues along the next row. Thus, to select the bottom right subplot of a 2 by 2 grid of subplots, you would use `subplot(2,2,4)`. Here's an example of plotting two side-by-side images:

```
# The skimage.io module defines imread()
import skimage.io as io
import matplotlib.pyplot as plt

# Read the file myimage.png from the current folder.
# After executing this, the variable myimage refers to
# an array containing the image data.
myimage = io.imread('myimage.png')
yourimage = io.imread('yourimage.png')

# Create a 1 by 2 grid of subplots
plt.subplots(1,2)

# Select the left subplot and display myimage
plt.subplot(1,2,1)
plt.imshow(myimage)

# Select the right subplot and display yourimage
plt.subplot(1,2,2)
plt.imshow(yourimage)
```

Once again, if after displaying several subplots in a figure, we want to display a new image (or graph or whatever) in an entirely new figure or set of subplots, we have to call `matplotlib.pyplot.figure()` or `subplots()` again to make create a new figure and make it the current figure, otherwise, the new image will overwrite the most recently used subplot.

3 Problems

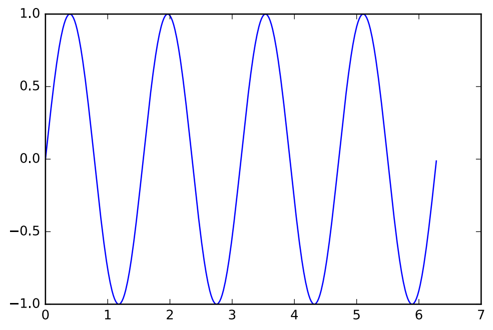
Question 1 (5 points):

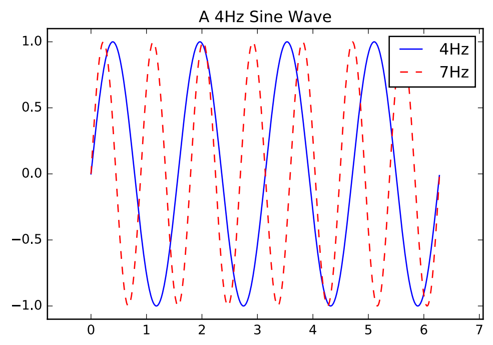
In this question you will practice basic image I/O, manipulation, and display. To complete the question, follow the instructions in the provided `asn0-q1.ipynb` iPython notebook. Notebooks can be loaded with PyCharm.

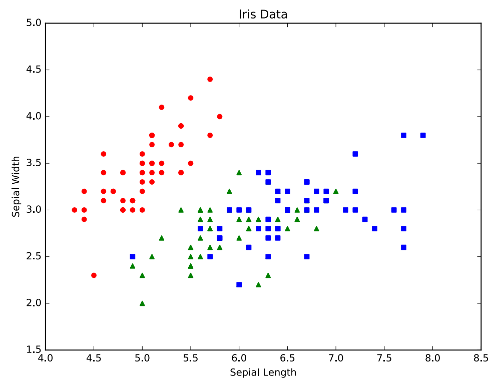
Question 2 (15 points):

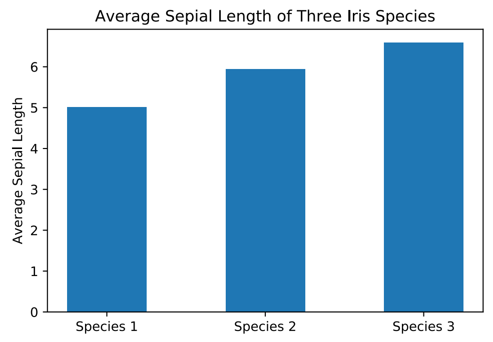
In this question you will learn how to produce some common types of plots using matplotlib. To complete the question, follow the instructions in the provided `asn0-q2.ipynb` iPython notebook. Notebooks can be loaded with PyCharm.

If you've done everything right, your completed notebook should contain four graphs which look like this:









Question 3 (7 points):

In this question, you'll practice using slicing and logical indexing of `numpy` arrays and use the subplots feature of matplotlib to display some of our results. To complete the question, follow the instructions in the provided `asn0-q3.ipynb` iPython notebook. Notebooks can be loaded with PyCharm.

The grading rubric for all questions can be found through the assignment handin link on Moodle.

4 Files Provided

asn0-qX.ipynb: These are iPython notebooks, one for each question, which includes instructions and in which you will do your assignment.

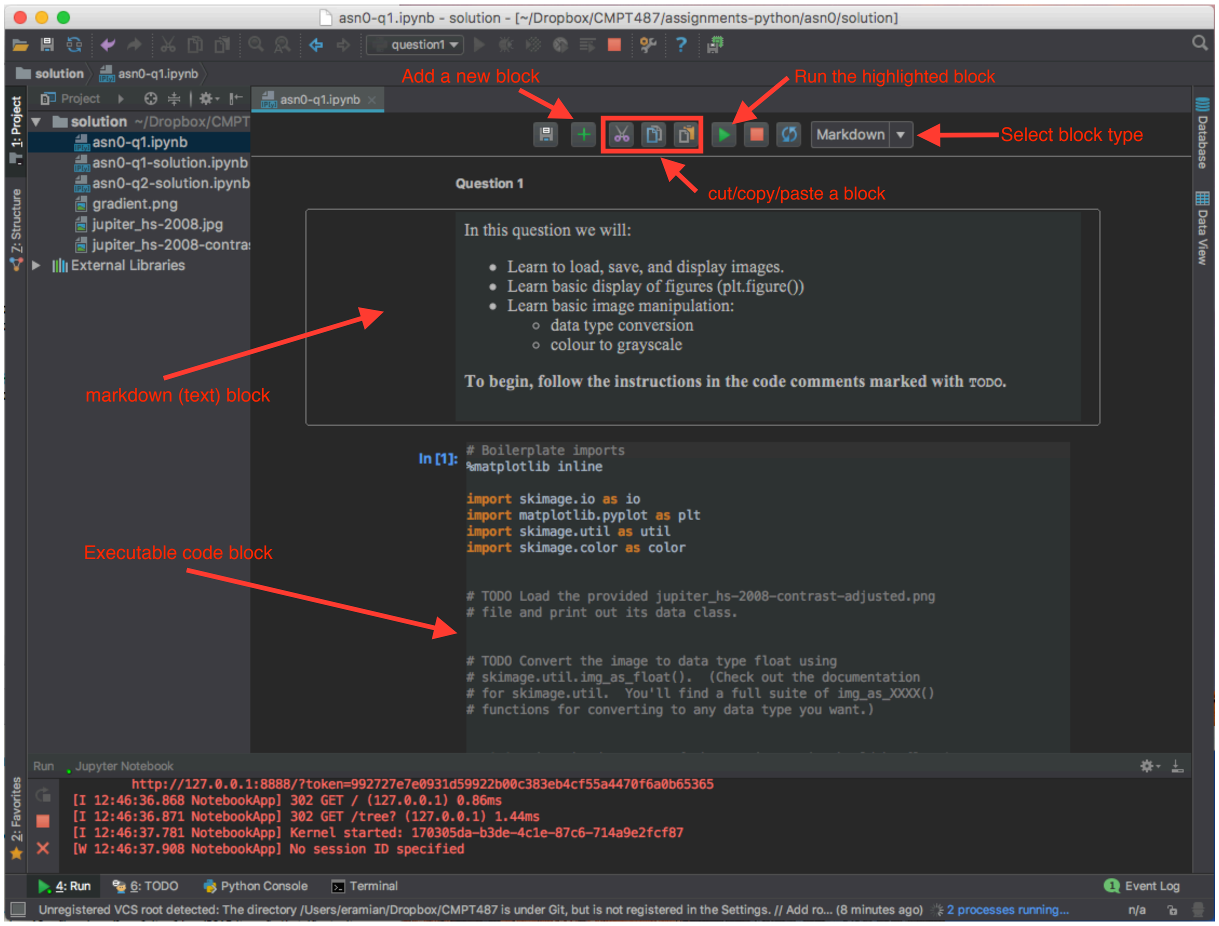
cmpt141readings.pdf: CMPT 141 course readings which contain a gentle introduction to Python.

Various Images: Image files are from NASA's online image gallery (`https://solarsystem.nasa.gov/galleries/`). They are not copyrighted and are used in accordance with their NASA's use policies: `https://www.nasa.gov/multimedia/guidelines/index.html`

5 What to Hand In

Hand in your completed iPython notebooks `asn0-q1.ipynb`, `asn0-q2.ipynb`, and `asn0-q3.ipynb`.

6 Appendix A — Important Elements of iPython Notebook



Question 1

In this question we will:

- Learn to load, save, and display images.
- Learn basic display of figures (plt.figure())
- Learn basic image manipulation:
  - data type conversion
  - colour to grayscale

To begin, follow the instructions in the code comments marked with TODO.

```
# Boilerplate imports
%matplotlib inline

import skimage.io as io
import matplotlib.pyplot as plt
import skimage.util as util
import skimage.color as color


# TODO Load the provided jupiter_hs-2008-contrast-adjusted.png
# file and print out its data class.


# TODO Convert the image to data type float using
# skimage.util.img_as_float().  (Check out the documentation
# for skimage.util.  You'll find a full suite of img_as_XXXX()
# functions for converting to any data type you want.)
```

```
        http://127.0.0.1:8888/?token=992727e7e0931d59922b00c383eb4cf55a4470f6a0b65365
[I 12:46:36.868 NotebookApp] 302 GET / (127.0.0.1) 0.86ms
[I 12:46:36.871 NotebookApp] 302 GET /tree? (127.0.0.1) 1.44ms
[I 12:46:37.781 NotebookApp] Kernel started: 170305da-b3de-4c1e-87c6-714a9e2fcf87
[W 12:46:37.908 NotebookApp] No session ID specified
```

Unregistered VCS root detected: The directory /Users/eramian/Dropbox/CMPT487 is under Git, but is not registered in the Settings. // Add ro... (8 minutes ago) 2 processes running...