Due on Wednesday, January 31st, 2018 at 11:59 pm
10% grade reduction per late day submission
Answer all problems following the instructions. Please do NOT email the instructors your solutions, but please submit the written portion typset with LaTeX or Microsoft Word with Math Editor. Scanned work will be accepted, but must be of high-quality. If we can't read your work, you get no credit. Please include the written and programming portions in a zipped folder titled "Lastname_firstname_hw0.zip"

Problem 1. *Setting up Coding Environment*
We will utilize the Python programming language along with OpenCV bindings to do most of the image processing required in the homework assignments. In addition, we will utilize several helper libraries for visualizing images and displaying output.

1. Install Anaconda Python `https://docs.anaconda.com/anaconda/install/`. Make sure you follow the instructions specific to your operating system. I used the graphical interface to install, although you can also use the command line version if you are comfortable with that. To test your Python is working, create a file **helloworld.py**, and write the following line:

```python
print('Hello World!')
```

   To run the code, you can open a terminal shell and run the following command:

```
python helloworld.py
```

   and should get the terminal output of "Hello, World!". If you run into errors, check your installation of Anaconda Python (Google is your friend for debugging errors! and Stack-Overflow).

2. Install OpenCV via Anaconda using the following command:

```
conda install -c conda-forge opencv
```

3. Install Matplotlib (a useful library for visualizing graphical output)

```
conda install matplotlib
```

4. Install ipdb (a useful debugger for Python)

```
conda install -c conda-forge ipdb
```

5. Make sure you can import all the libraries. Write the following file **testimport.py** with the following lines:

```python
import cv2
import matplotlib
import ipdb
```

If you get no errors while running this, then you have successfully installed these packages. **NOTE:** If you are stuck on any step, Google is your friend! Just type "How do I install X on a Mac/Windows/Linux?" and experiment till you find something that works!

Problem 2. *Messing around with OpenCV and Images*
The point of this problem is to mess around with images, and explore the OpenCV framework.

1. Write all your code in a file entitled **HW0.py**. First import all the dependencies you will need as follows:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import ipdb
```

For every section, you should block it off with a set of comments as follows:

```
########## Section 2.X -<insert description>########################
# load image
<your code here>
# display image
<your code here>
##############################################
```

This makes it easy to read, and legible.

2. **Loading and displaying images:**
a. Read in the image *elephant.jpeg* that is provided in the homework assignment. Use the OpenCV commands *cv2.imread*,*cv2.imshow*, *cv2.waitKey*, and *cv2.destroyAllWindows*. Make the code so that you can press any button to destroy the window (otherwise the code will remain stuck on displaying the image).

b. Next, let's display the image using Matplotlib. Use the command *plt.imshow* and *plt.show* to show the image. **Notice how the colors seem inverted**. You have to close the window (hit the red close button) to allow the code to proceed. Write the image file using the command *cv2.imwrite* with the filename "elephant_opencv.png".

c. OpenCV stores images in a BGR format, but Matplotlib expects RGB images. To load the elephant image with correct colors for Matplotlib, use the following command *cv2.cvtColor* to convert the loaded image to a RGB space, then plot with Matplotlib. Write out this image using *cv2.imwrite* with filename "elephant_matplotlib.png". The two images you have written out in this section should have different colors.

d. Finally, let's read in the image in grayscale. Read in the image as gray (or convert the existing image to grayscale), plot the image using Matplotlib in grayscale (hint: look at the *cmap* option), and write out the image as "elephant_gray.png".

3. **Cropping:**
Read in the elephant image in color. Crop out the small elephant by figuring out the x,y coordinates needed. Note that the first coordinates in Python correspond to image rows,

and the second coordinates correspond to image columns, and the last set of coordinates are the color channels. Display the image of the small elephant (correctly in RGB) using Matplotlib, and then write out the image as "babyelephant.png".

4. **Pixel-wise Arithmetic Operations:**
   a. Read in the image in color, and convert it to RGB space.

   b. Add the value 256 to every single pixel in the image using the following command:

   ```
   image = image + 256
   ```

   After doing this, check the image's datatype using *image.dtype* command. To display the image, convert the image back to uint8 by using *np.uint8* command, and then display the image. **Describe what you see here. Why does it look like there is no change in the image? Hint: look at the pixel values before and after casting it into np.uint8()**.

   c. Now we will add 256 using opencv. First use the *cv2.split* command to split the image into R,G,B channels separately. Then apply *cv2.add* for each channel, adding 256 to each one. Merge the channels back together, and display the image. **Describe the image you see. Why was this different from step b? Feel free to Google the difference between opencv's add and numpy's add**.

   The moral of the story: be careful with your image formats and pixel arithmetic! Major source of error in doing image processing.

5. **Resizing images:**
   a. Read in the image in color again, and convert it to RGB space.

   b. Downsample the image by 10x in width and height. Use the *cv2.resize* command. If you have problems with this command, look at examples online and use the keyword "None" for the optional parameters. Display the image, and write out the image as "elephant_10xdown.png".

   c. Using the resize command, upsample the same downsampled image from part b by 10x back to its original resolution. This time, try two different interpolation methods: nearest neighbor and bicubic. Write out both images to files as "elephant_10xup_method.png" where method is the method you used for interpolation.

   d. Calculate the absolute difference between the ground truth image and the two upsampled images with the two methods (find the appropriate OpenCV command, should be one line of code for each image). Write out the difference images for both methods. **Sum all the pixels in the difference image for the two methods, and report the number. Which method caused less error in upsampling?**.

1 Handing in the Assignment

Create a written report for Problem 2, where you print out your code, embed the output images you generate after every *cv2.imwrite* for what you got, and answer any bolded question in the document. Feel free to ask me in class or office hours how to present your code in a legible and straightforward way.

Grading: Your written report will be graded 80% on correctness, and 20% on presentation (lots of comments, easy to read code, written report quality). The easier you make my life to grade, the easier for you! Submit your report, as well as your python files and output images in a zipped folder through Blackboard.