Due on Wednesday, September 18, 2018 at 11:59 pm
10% grade reduction per late day submission
Answer all problems following the instructions. Please include the written and programming portions in a zipped folder titled "Lastname_firstname_hw1.zip"

1 Written Problems

Problem 1. *(Image convolution by hand)* Compute the convolution of an image $x(m,n)$ and a filter $h(m,n)$, given as follows

$$x(m,n) = \begin{array}{c|cc} & n & \rightarrow \\ \hline m & \mathbf{1} & 2 \\ \downarrow & 3 & 4 \\ & 2 & 1 \end{array} \qquad h(m,n) = \begin{array}{c|ccc} & n & \rightarrow & \\ \hline m & \mathbf{2} & 1 & 0 \\ \downarrow & 1 & 0 & -1 \end{array}$$

where the boldfaced number indicates the sample at the origin. Assume zero padding outside of x so that the convolution is well-defined. (Hint: Note in 1D convolution, $h[n-k] = h[-(k-n)]$ means you flip about $k=0$, then shift by n samples before multiplying to calculate $y[n]$. Extend this to 2D)

2 Coding Assignments

Problem 2. *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:

```
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 StackOverflow).

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:

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

1. 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?**.

Problem 4. *(Convolution as Matrix Multiplication)*

2D Convolution: We will implement 2D convolution as a matrix multiplication. In fact, this is used often in machine learning algorithms such as convolutional neural networks where one needs to perform several convolutions for each layer/filter bank of the algorithm. We will now implement 2D convolution as a matrix operation. Define h as follows:

$$h = \begin{bmatrix} 1 & 0 & -1 \\ 1 & \mathbf{0} & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

.

(a) First use image

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

. Assume zero-padding and no flipping of the convolution kernal (i.e. the center of the 3x3 kernal is the pivot point, and since the filter is symmetric, there is no reason for flipping). First, vectorize $\vec{I}$ where each row of the image is stacked vertically into a column vector, i.e. $\vec{I} = [1, 2, 3, 4, 5, 6, 7, 8, 9]^T$. Then write by hand a matrix H such that $H * \vec{I}$ is the vectorized output image. H should be a 25 x 9 matrix in this case. The output should be a $25x1$ vector, taht corresponds to a 5x5 output image that is row stacked. **This part should be calculated by hand on paper**.

(b) Now write a function *conv2dmatrix* that takes inputs $[image, H]$ and outputs $[image * H, time, error]$. This code should compute the same result as (a) which you did calculate numerically.

(c) Generalize your code so the image is the elephant picture (grayscaled), and the new function can still compute the convolution as a matrix multiplication. The output should be an edge filtered image.

Problem 5. *(Fourier Domain Fun)*

(a) Read in the elephant picture. Plot the Fourier transform (magnitude and phase) of the picture.

(b) Implement both a low-pass, high-pass filter, and a diagonal bandpass filter (of your choice) on the elephant picture in the frequency domain. Make sure to plot the Fourier magnitudes of the image before and after filtering. Show the resulting filtered images.

(c) Phase Swapping: Select two images, compute their Fourier transform, and display their magnitude and phase images. Then swap their phase images, perform the inverse Fourier transform, and reconstruct the images (display these). Comment on how this looks perceptually.

(d) Hybrid Images: Take two images (of your choice), and try the hybrid images approach from the paper from Olivia et al. I will show the best results in class!

Problem 6. *(Multiresolution Blending using Gaussian/Laplacian Pyramids)*

In this problem, we are going to make the famous "orapple", a fruit that is a blend of an orange and an apple! To do so, we are going to use multiresolution blending. Use the paper by Burt and Adelson (see Canvas website) as reference for this assignment.

(a) First write a code to generate the Gaussian and Laplacian pyramids of an image. You should confirm that your Laplacian pyramid can be collapsed back into the original image. **Note for this example, you cannot use the functions cv2.pyrUp, cv2.pyrDown except to check/unit test whether your pyramid code is accurate.**

(b) Read in the images for apple and orange, display them, and convert them to double precision.

(c) Make a mask that will act as the transition between the apple and orange image by keeping one half of the image as 1 and the other half at zero.

(d) Direct Blending: Try to directly blend the two images by using $I = (1 - M) * I_1 + M * I_2$, where M is the mask and $I_{1,2}$ are the two fruit images. Display this image.

(e) Alpha Blending: Try to blur the mask edge with a Gaussian, and then $I = (1-M)*I_1+M*I_2$ (i.e. alpha blending or feathering). How does this look (does it look like one fruit?)

(f) Multiresolution blending: Write a function *multiblend* that takes in as inputs $[image1, image2]$ and outputs $[blendedimage]$. Construct a Gaussian pyramid of the Mask, and Laplacian pyramids of the images (depth is your choice, but keep it fixed for all pyramids). Blend the images using the multiresolution blending algorithm shown in class. Show your "orapple" to the world! **Note for this example, you cannot use the functions cv2.pyrUp, cv2.pyrDown except to check/unit test whether your Gaussian pyramid code is accurate.** Try to play around with parameters, etc until you find a blending you like.