UMBC
CMSC 491/691 Computer Vision

Homework 1
491: 100 points. 691: 120 points

Due: February 23, 2024 23:59 Eastern

Answer all problems according to the instructions provided.
Written problems may be typeset or handwritten.
Submit one `.zip` file named `LastName_FirstName_hw1.zip` containing a single `pdf` named `hw1.pdf` with the answers to both the written and coding sections. The pdf **must** contain relevant code snippets, outputs and results, and other requested information for each task. Put all other files (code, data, outputs, etc.) in a single directory named "`code`"

1 Written Problems (15)

Problem 1. *Image Convolution by Hand* 5 points
Show the method and answer for convolving image $x(m,n)$ with filter $h(m,n)$. Assume zero-padding outside of x.

$$x(m,n) = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 2 & 1 \end{bmatrix}, \qquad h(m,n) = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix}.$$

Problem 2. *Properties of Convolution*
1D Convolution between two signals $f, g \in \mathbb{R}^N$ is given by:

$$(f * g)[n] = \sum_{k=0}^{N-1} f[n-k]g[k].$$

(a) Prove that convolution is commutative and associative. 10 points

(b) **[Extra Credit, 5 points]** Prove that cross-correlation is NOT commutative.

2 Basic Operations in OpenCV (30)

Task 1 *Setting up your coding environment* 0 points
We will use Python along with OpenCV and additional helper libraries to do most of the image processing required in homework assignments. You will not need anything other than your laptop or desktop for this homework assignment. No GPUs are necessary.

1. Install Anaconda Python https://docs.anaconda.com/anaconda/install/ and make sure you follow the instructions specific to your operating system. To test if your Python is working, create a file `helloworld.py` with the following line:

```
1        print("Hello World")
```

and run it using the following command in a terminal shell / command prompt:

```
python helloworld.py
```

2. Install OpenCV via Anaconda using the following command:

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

3. Install Matplotlib (a useful library for visualizing graphical outputs and for making plots)

```
conda install matplotlib
```

4. Make sure your libraries work by importing them. Create a file named `testimport.py` with the following lines:

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib.pyplot as plt
```

If you get no errors at this point, you have successfully set up your environment. **NOTE:** if you are stuck on any step, Google is your friend! Just search something like "How do I install X on a ¡Mac/Windows/Linux/...¿" or you can even copy your error message and google it.

5. Note for documenting code: for every section, you should block it off with a set of comments. This makes it easy to read and legible. For example:

```
import cv2 # this imports OpenCV
import numpy as np # this imports numpy
import matplotlib.pyplot as plt

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

```

Task 2 *Loading and Displaying Images* 5 points

1. Read in the image `elephant.jpeg` that is provided in the homework assignment. Useful functions: `cv2.imread, cv2.imshow, cv2.waitKey, cv2.destroyAllWindows`. The latter makes sure that pressing a button will close all windows (otherwise the code will remain stuck on displaying the image).

2. Display the image using matplotlib. Useful functions: `plt.imshow, plt.show`. Write the image to a file named `elephant_opencv.png` by using the command `cv2.imwrite`. Do you notice anything weird about the colors?

3. OpenCV stores images in BGR format, but Matplotlib expects RGB! This is the reason why the image `elephant_opencv.png` had inverted colors. Let's fix this. To load the elephant image with correct colors for Matplotlib, use `cv2.cvtColor` to convert the loaded image to a RGB space, then plot with Matplotlib. Write this image to a file named `elephant_matplotlib.png`.

Task 3 *Basic Image Processing Operations* 15 points

1. Convert the image to grayscale. Read (or convert the existing image) as grayscale, plot it (hint: look up the `cmap` option) and write the output to `elephant_gray.png`.

2. **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 `elephant_baby.png`.

3. **Resizing:**

   (a) Read in the image in color again, and convert it to RGB space. Downsample the image by 10× 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`.

   (b) Using the resize command, upsample the same downsampled image from part b by 10× 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.*

   (c) 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?**

Task 4 *Edge Detection and Image Blurring by Convolution* 10 points

1. Write a function using `Python` and `OpenCV` to implement edge detection using 2D convolution. (Hint: look up `cv2.filter2D`. Show the filter that you used for this operation, written as a matrix.

2. Similarly, implement image blurring (also known as smoothing) using the same OpenCV function you used for edge detection. Show the filter that you used for this operation, written as a matrix.

3. Display the original elephant image, the edge detection output, and the blurring output, side by side.

4. **[Extra Credit, 5 points]** Implement convolution without using any OpenCV or Numpy, but purely with Python.

3 Fourier Domain Image Blending (20)

Task 1. *Phase Swapping* 10 points

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.

Task 2. *Hybrid Images* 10 points

Take two images of your choice (be creative!) and implement the "Hybrid Images" approach from Oliva et al. (SIGGRAPH 2006) [2] (Section 2) to blend the two images. PDF of the paper is in the footnote[1].

[1] https://stanford.edu/class/ee367/reading/OlivaTorralb_Hybrid_Siggraph06.pdf

4 Multiresolution Image Blending (35)

In this problem we will implement multiresolution image blending using Gaussian/Laplacian pyramids. This approach is from Burt and Adelson [1]. PDF of the paper is in the footnote[2]. In this problem, we are going to make the famous *"orapple"*, a fruit that is a blend of an orange and an apple!

Task 1. *Direct and Alpha blending* 10 points
Blend the two images by using an appropriate mask M and blending the two images as $I_{blend} = (1-M)I_1 + M * I_2$. For alpha blending, blur the mask edge with a filter. Display $I_1, I_2, I_{blend}^{direct}, I_{blend}^{alpha}$ side-by-side.

Task 2. *Gaussian and Laplacian Pyramids* 10 points
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. **cv2.pyrUp, cv2.pyrDown cannot be used except to check/unit test whether your pyramid code is accurate.**

Task 3. *Multiresolution blending* 10 points
Write a `Python` function `multiblend` that takes two images as input and returns a blended image as output. Use the functions that you wrote in Task 2. 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 algorithm shown in class. Display the output and compare it with outputs of Task 1.

Task 4. *Repeat Task 1 and 3 with maximum grit.* 5 points
Find one image of True Grit (our mascot) and one image of a real Chesapeake Bay Retriever. Apply the blending techniques from this problem. Display the outputs. Comment on why "realistic" blending is hard to achieve in this case but easier for orapple.

5 Required for 691 Section. Optional for 491. (20)

Prompt 1. 2 points
In the coding problem we blended two images into one and created an `orapple`. Could this technique be used to create deepfakes? Explain why or why not.

Prompt 2. 3 points
What is the oldest instance of a deepfake image that you know of? Show us that image, the year that it was created, and briefly describe why it is fake / altered / manipulated.

Prompt 3. 15 points
Conduct a literature review on deepfakes. List and describe:

1. at least 2 research papers that propose image altering / editing / manipulation techniques without any use of machine learning
2. at least 2 state-of-the-art research papers on the same topics (published 2020 or later)
3. at least 2 research papers on detecting deepfakes.

References

[1] P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. 4

[2] Aude Oliva, Antonio Torralba, and Philippe G Schyns. Hybrid images. *ACM Transactions on Graphics (TOG)*, 25(3):527–532, 2006. 3

[2] http://persci.mit.edu/pub_pdfs/pyramid83.pdf