Computational Photography

Assignment #2: Image I/O & Libraries

Ram Subramanian

Fall 2016

General Template Notes

The following notes apply to the entire write-up.

Note 1: *Several parts require you to use images you have taken, now or in the past. Make sure these are images you can share.*

Note 2: *For each function, you may include critical code lines, however code alone is not sufficient, explain it clearly using sentences.*

Note 3: *You may use smaller text (minimum size = 10), insert images and code, and choose your own layout for each page. More pages may be used whenever you find it necessary for your best presentation.*

Note 4: *Make sure that you answer all questions and discussion items!*

numberOfPixels

```
return image.size
```

Every numpy.ndarray has a ‘size’ variable that indicates how many entries (across all dimensions) are present in the array.

Since we are dealing with a grayscale image (1 channel), each pixel has exactly one entry in the image array. Therefore image.size will give you the number of pixels in the grayscale image ’image’.

averagePixel

```
return int(np.sum(image) / numberOfPixels(image))
```

The numpy function .sum() returns the sum of all the entries (across all dimensions) in an np-array. I simply divide the sum by the previously calculated number of pixels to get the average, which is casted to an int before returning.

How would your approach change if you used a color (BGR) image?
Depends on whether we want a single average for the whole image, or an average per pixel. For the former, we can simply reuse the solution above, while dividing the result by 3, one for each channel. For the latter, np.sum() takes an axis argument. Specifying axis=2 will sum each of the dimensions together along the third dimension (color channel). So, it would be:

```
return int(np.sum(image, axis=2) / image.shape[2])
```

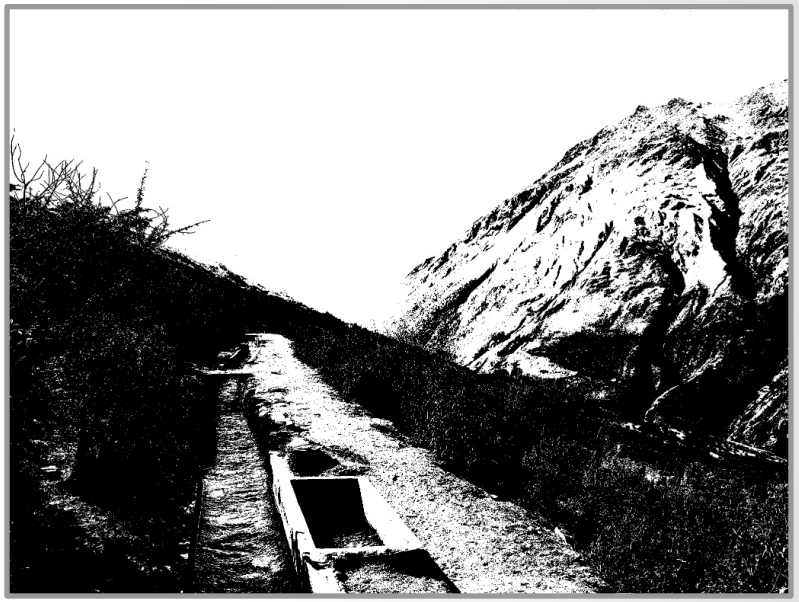
convertToBlackAndWhite

```
image[image>128] = 255
image[image<=128] = 0
```

Applying a comparison to a numpy array causes it to apply the comparison to every constituent element and return an array of the results, shaped in the same was as the original array. And conveniently, this result can serve as an index into said array. So putting the two together, we can convert to b&w easily.

convertToBlackAndWhite (continued)

Input image



Output image

averageTwoImages

```
avg = (image1 / 2.0 + image2 / 2.0)
return avg.astype(np.uint8)
```

cv2.imread() returns an image array of type uint8, i.e. each pixel has unsigned 8 bits to store its intensity. Since 8 bits may not be sufficient to hold the addition of two 8-bit integers, some information maybe lost – this is called overflow. For example, the following uint8's *a* and *b* are added to give *c*. *c* being 9 bits long cannot be stored back into a uint8 in its entirety.

```
1111 1111 + 0000 0001 = 1 0000 0000
    a           b             c
```

To solve this problem, I halve each image before adding them together. Since the types of the images are implicitly converted to float (by the python interpreter, because '2.0' is of float type), we don't lose the .5's we get from halving odd intensities. I convert the image back to uint8 before returning, which truncates any residual .5's remaining in the intensities.

averageTwoImages (continued)

Input 1

Input 2

Overflow Output image

Correct Output image

Notice that the overflow image is a lot darker in places where both constituent images were lighter. This is because of how numpy handles overflow – for any value greater than 255 (uint8), it simply loops back to 0 and continues counting. That is, (255 + 1 = 0), (255 + 2 = 1), (255 + 10 = 9), etc. And smaller values = darker image.

flipHorizontal

```
return image[:,::-1]
```

Since the image is single channel, it suffices to reverse the order of the columns in the numpy array to flip the image horizontally. ‘::-1’ is the python syntax to reverse the particular axis. ‘:’ is the python syntax for selecting all entries in that axis.

flipHorizontal (continued)

Input image

Output image

Resources

List the sources for your ideas.

The computer vision class I took during Spring ‘16 :)