Project # 1: Image Filtering
CSC 391/691: Selected Topics - Computer Vision

Instructor: V. Paúl Pauca

due date: Monday September 18

---

1 Part 1: Vectorized Operations (45%)

Algorithmic efficiency is a critical component in the application of computer vision. Efficient code takes advantage of Numpy's *vectorized* operations, which perform on blocks of data at the time, rather than on individual elements. This is done by delegating the actual manipulation of the arrays to optimized, pre-compiled code written in a low-level language such as C. See, e.g., Python Like You Mean It for more detail.

In this part, you will learn to use Numpy to develop efficient Python code for the following basic image processing operations.

1. **Histogram equalization.** Write a Python function called `histEq` that given an image, as a Numpy ND-array, applies histogram equalization (textbook, page 44) and returns the filtered image.
2. **Adaptive histogram equalization.** (Grad Only) Write a Python function called `adaptiveHistEq` that implements the adaptive histogram equalization method. Alternatively, you can implement the contrast-limited adaptive histogram equalization (CLHE) method. Wikipedia provides good information and a list of references.
3. **Convolution with 3×3 and 5×5 filters.** Write a Python function called `conv` that given an image and a 3×3 or a 5×5 filter, applies convolution (textbook, page 47-48) to filter the image with the given filter and returns a filtered image.

Guidelines

- Undergrad: Your code should work on grayscale input images.
- Grad: Your code should work on grayscale or RGB input images (hint: use the YCbCr colorspace).
- The resulting image should be of the same shape as the input image. You can choose any boundary condition.
- Write a small Python script that compares the performance of your code against the corresponding OpenCV code. Use the `tic` and `toc` operations shown in the sample code to estimate computation time.

2 Part 2: Design your own Image Filter (45%)

In this part, you will use what you have learned about point and local operations on images to develop an image filter of your own. This filter is not meant for a specific task, such as edge detection. Rather, it is meant to generate a "fun" output image, similar to those obtained using apps like SnapChat.

Guidelines

- You can use OpenCV functions for point and local operations, including corner detection. You are welcome to use your own implementations as well.
- You can use other concepts from chapter two, such as residual images.
- You can use compositions of different filters to transform the input image.

- **Don't** use techniques not found in chapter two, such as filters that require object or face detection or other supervised learning methods.
- Undergrad: Your functions should work on grayscale input images.
- Grad: Your functions should work on grayscale or RGB input images.

3 Part 3: Video Frame Processing (10%)

It is much more interesting when image processing operations are applied on real time to video frames, rather than to an image read from a file. Efficiency of the code does play an important factor in this case.

OpenCV provides functions for capturing video frames. Basic examples for how to do this are found in this Github repository.

For this part, you will develop a simple Python application allowing real-time filtering of video frames. Your implementation should show the unfiltered and filtered frames simultaneously side by side in the same window or separately in two windows. The user should interact with the application via the keyboard, as follows:

- `h` - selects histogram equalization
- `a` - selects adaptive histogram equalization (grad case)
- `s` - selects a 5×5 smoothing filter
- `u` - selects a 5×5 unsharp filter
- `e` - selects an edge detector filter of your choice

Guidelines

- Undergrad: you can develop your own Python script for implementing this application using the sample code provided in the Github repository. You can also base your code on the code provide under `cameo` in the repo.
- Grad: your Python app should use the `cameo` code as your code base.

4 Expectations and Grading

- **Documentation**. Your code should be well-structured and include brief and well-placed comments. Use comments by the Github repo authors as guidance.
- **Code quality**. We need to be writing code that follows good software development practices. This site at InterviewBit provides a nice short list of key software engineering principles. The expectation for this project is that your code is as DRY as possible.
- `Performance.` The relative performance of your code in Part 1, versus the corresponding OpenCV functions, is important. There should not be more than one order of magnitude difference for full credit.
- `Summary.` Include a `Readme` file with any comments and explanations of your work. It should not exceed one page (if printed).
- Grading. The project will be graded with a letter grade overall: A, A-, B+, B, etc.