CME 211: Homework 6

Due: Friday, November 18, 2022 at 5:00 PM Pacific.

This assignment was designed by Patrick LeGresley and modified for the purposes of this course.

Background

Image processing is an important category of computations utilized in many science and engineering fields. Some important computations in image processing include *focus detection* and various *filtering* operations to *smooth* or *sharpen* an image, *detect edges*, etc.

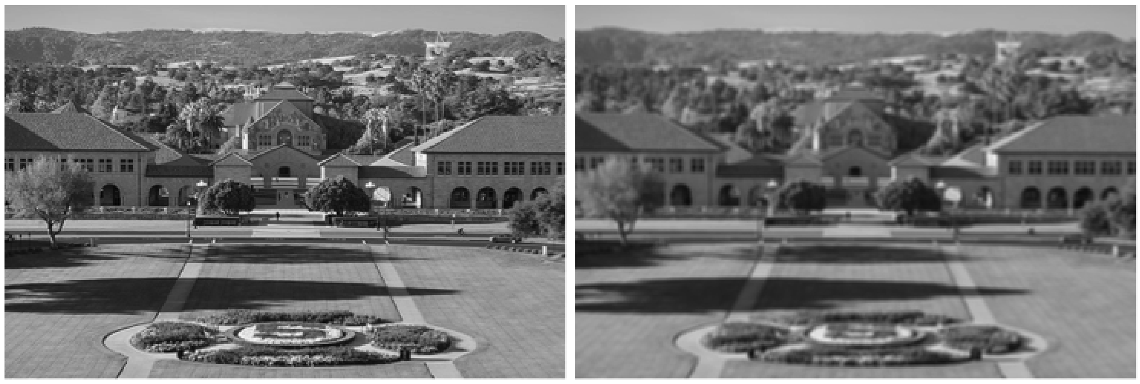


Figure 1: Original image (left) and box blurred image (right)

Many of these image processing operations can be implemented in terms of convolution between a kernel and the input image. A grayscale image can be stored in a 2D array of unsigned char values representing 256 shades of gray. The kernel is a small matrix used to compute the value of one pixel in the output from multiple pixel values in the input image. For example, the box blur in Figure 1 could be achieved by using convolution and the following 3 x 3 kernel:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{1}$$

The value of a pixel in the output image is calculated by summing the product of each kernel value and the corresponding input image pixel value as shown in Figure 2.

In this case the kernel effectively computes each output pixel as the average of the surrounding pixel values in the original input image.

For output pixels close to the boundary some of the surrounding input pixels will be outside of the image. To handle this the values can be extended from the image as shown in Figure 3.

Sharpness, or focus, of an image can be estimated in a simple way by first convolving the image with a kernel that approximates the *Laplacian* operator:

$$\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix} \tag{2}$$

The maximum value of the resulting output can be used to quantify the sharpness. Focused images will have a high maximum value, while out of focus images will have lower maximum values.

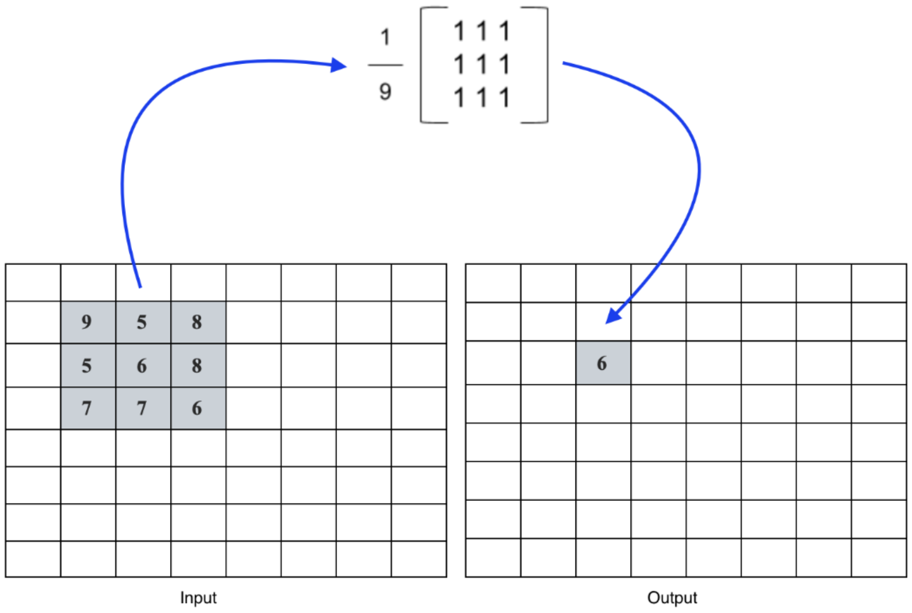


Figure 2: Convolution of the 3x3 kernel for box blur

Assignment (60 points functionality, 20 points design, 10 points style, 10 points writeup)

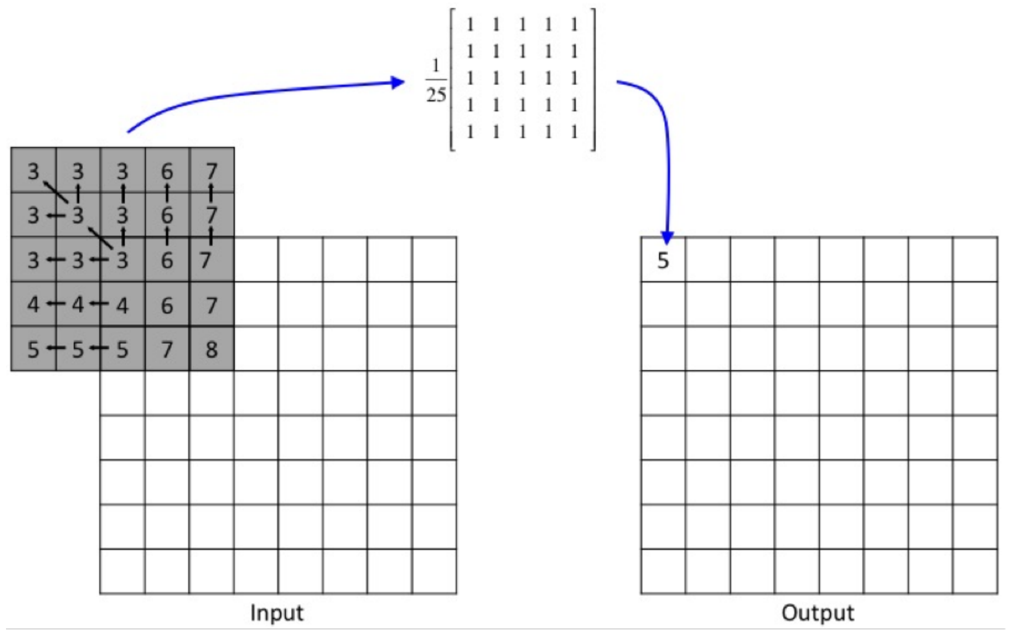
In this assignment you will be developing a C++ image class that can read and write JPEG files, and has methods to compute the sharpness of the image and smooth (blur) the image using a box blur kernel of a specified size. To demonstrate your class you will write a `main()` to read the provided grayscale image and compute the sharpness of the image. Then you will blur the image with successively larger box blur kernels and compute the sharpness of each increasingly blurry version of the image.

Summary of requirements

1. Implement in `image.hpp` / `image.cpp` an image class with a constructor that accepts a string containing the name of the JPEG file to be read. The constructor should read the file and store the image data as a data attribute. To assist you the included `hw6.hpp` / `hw6.cpp` files contain a `ReadGrayscaleJPEG()` function.
2. Add a `Save()` method to your class that writes the current version of the image to a JPEG file. The method should take a string containing the name of the JPEG file to be written, but if the string is empty the original filename should be used. To assist you the included `hw6.hpp` / `hw6.cpp` files contain a `WriteGrayscaleJPEG()` function.
3. In `image.cpp` write a `Convolution()` function. The recommended prototype / interface is:

```
void Convolution(boost::multi_array<unsigned char,2>& input,
                 boost::multi_array<unsigned char,2>& output,
                 boost::multi_array<float,2>&         kernel);
```

The input and output should be of the same size and you need only support odd size, square kernels of at least size 3. If the function is called and these requirements are not met you should output an error message and exit. When you are implementing this function be careful about your use of datatypes, particularly with regards to rounding and underflow / overflow.



$\frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$
Input
Output
5

Figure 3: Extending the pixel values for output pixels near the boundary

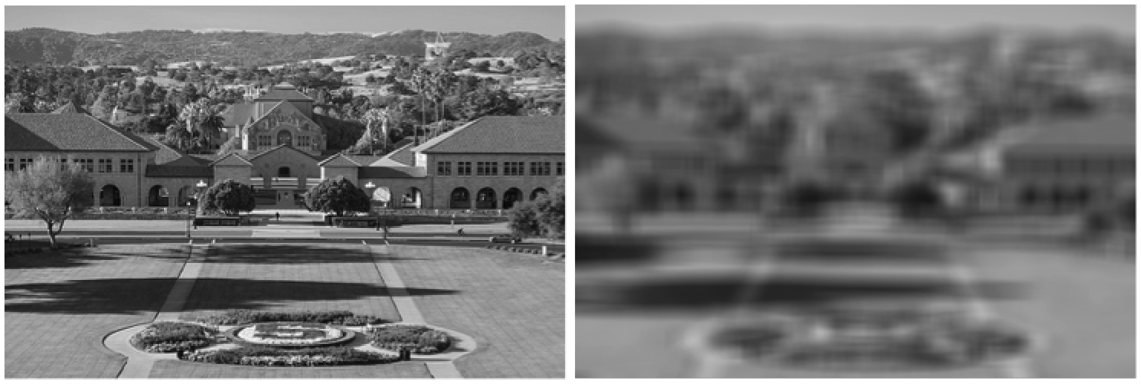


Figure 4: Box blur using a kernel size of 3 (left) and 27 (right)

4. Add a `BoxBlur()` method to your class that accepts one argument specifying the kernel size (e.g. 3 for equation 1) and uses the `Convolution()` function to smooth the image. The values in the kernel should always be scaled so that they sum to 1.

5. Add a `Sharpness()` method that returns an unsigned int representing the sharpness of the image. Use the kernel from equation 2 that approximates the Laplacian operator and the `Convolution()` function, and return the maximum of the convolution output. Note that returning an `unsigned int` rather than an `unsigned char` is for convenience, as stream insertion of an unsigned char defaults to the character interpretation rather than the number interpretation.

6. Write a `main()` function in `main.cpp` that:

   1. Loads the original image `stanford.jpg` and computes and outputs the sharpness
   2. For kernel sizes of 3, 7, . . . , 23, 27 reload the original image by instantiating a new instance of your image class, blur the image, and then compute and output the sharpness of the resulting image. Each blurred version of the image should be output to `BoxBlur03.jpg`, `BoxBlur07.jpg`, etc.

   The expected output of main should be: “‘ $ ./main Original image: 255 BoxBlur(3): 139 BoxBlur(7):

44 BoxBlur(11): 27 BoxBlur(15): 21 BoxBlur(19): 16 BoxBlur(23): 11 BoxBlur(27): 9

"'

7. Include a `makefile` that generates the executable main as the default target.
8. The code must compile without any warnings with following compiler flags: `-std=c++11 -Wall -Wconversion -Wextra -pedantic`.
9. Write a README file describing the functionality of your code and key implementation details.

Homework submission

Please be very careful with directory and filenames, as points will be deducted if you do not follow the directions. To be clear you should have the following files in the `hw6` directory you submit:

- `README`
- `makefile`
- `hw6.cpp`
- `hw6.hpp`
- `image.cpp`
- `image.hpp`
- `main.cpp`
- `stanford.jpg`

We will use a Git tag to mark the commit that you want graded. A Git tag is essentially a bookmark to a particular commit. After you have committed the version of the code you would like graded, create a hw6 tag with the following command:

```
$ git tag -am "HW6 Submission" hw6
```

or

```
git tag -m "HW6 Submission" -a hw6
```

The quotes contain a tag message. In the above command hw6 is the actual tag. You can push commits along with tags to the remote repo with:

```
$ git push --tags origin main
```

It is a good idea to check if the tags show up on GitHub. These appear in the "release" tab on the web interface.

It is possible to change your tag later, too, if you discover a bug after pushing:

```
$ git tag -d hw6 # delete the hw6 tag pointing to old commit

$ git push --delete origin hw6 # delete the remote hw6 tag

$ git tag -am "My HW6" hw6 # create hw6 tag pointing to current commit

$ git push --tags origin master # push changes to code and tags
```

We will use Git time-stamps on tags and the associated commits to check deadlines. Be careful with overwriting tags – you don't want to lose the submission tag.

Do not commit temporary files produced by your text editor. We will deduct points if these instructions are not followed. When working in teams and professional environments, it is important to keep your repositories free of extraneous files.