**CMP SCI 4420/5420** **Image Processing and Computer Vision**
Assignment # 3 Due Date: October 11, 2022

---

# Image Sampling and Quantization

### Purpose

The goal of this assignment is to study the interplay and tradeoff between the *sampling* and *quantization* parameters used in the image acquisition process. You are given an image of size $M \times N$ pixels, with $k$ bits used for the intensity resolution. How do you *resample* the image? For example, you want to shrink the image size to $M/2 \times N/2$ or enlarge the image to $2M \times 2N$ size. Also, you want to reduce the image storage requirements by decreasing the number of bits used for intensity resolution. How do you accomplish these tasks without losing the information contents of the image?

### Sampling and Quantization

In the older days, cameras used a film to capture images. Those images were modeled by continuous functions that needed to be converted to discrete values for processing by a computer using a process called *digitization*. Modern digital cameras capture the images directly by projecting continuous signal onto a discrete sensor array. Image *sampling* and *quantization* are the two techniques that control the image quality during the *digitization* process.

Sampling leads to digitization of the coordinate values in a signal/image. Sampling affects the resolution of an image in terms of the number of pixels in $x$ and $y$ directions in the image.

Quantization is the number of distinct grayscale/color levels achieved by digitizing the amplitude values in the signal/image. Typically, we have 256 grayscale levels or $2^{24}$ different color values in an image though all the possible color values are not used.

Consider a digital image $f(x, y)$ represented by an $M \times N$ matrix. Each element in this matrix is referred to as a *pixel*. For a 2D image, sampling refers to the total number of pixels in the image, which in turn is determined by the number of rows and the number of columns.

$$f(x,y) = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,N) \\ f(2,1) & f(2,2) & \cdots & f(2,N) \\ \vdots & \vdots & \vdots & \vdots \\ f(M,1) & f(M,2) & \cdots & f(M,N) \end{bmatrix}$$

There are $N$ samples in each row of $f(x, y)$. We have $M$ rows, and the total number of pixels is $M \times N$. In other words, the $2D$ image is sampled at $M \times N$ *spatial locations*.

A *quantization* scheme determines the range of values for $f(x, y)$. For example, if we use a single bit for quantization, we can only store two ($2^1$) different values for any $f(x, y)$ – 0 or 1. On the other hand, if we use 4 bits for quantization, we can store sixteen ($2^4$) different values for any $f(x, y)$. The number of bits used for quantization is referred to as the *intensity resolution*.

In general, image quality increases as we increase the number of samples (i.e., *sampling rate* or *spatial resolution*) and the intensity resolution. However, increasing the spatial resolution and the intensity resolution increases the storage requirements. The storage required for an $M \times N$ image with $k$ bits for intensity resolution is $M \times N \times k$ bits.

### Image Interpolation

Image interpolation is the process of estimating pixel values at unknown locations using the pixel values at known locations. Image interpolation is a widely used technique for tasks such as zooming, shrinking, rotating, and geometric corrections.

Image interpolation techniques include *nearest neighbor interpolation*, *linear interpolation*, *bilinear interpolation*, and *bicubic interpolation*.

**Task**

This project requires you to visually assess the change in quality of an image at different sampling and quantization level. You are required to vary the spatial and intensity resolution of an image and assess the changes qualitatively; for example you can specify when you start to lose some object in the image, or when you start to see some bands in the image.

You will select the method for changing the sampling rate through a command line option and display the resulting images, with downsampling and upsampling.

Your program should provide the following functions:

- Downsample the images by removing alternate columns and rows (say, by removing all odd-numbered columns and rows). Use the projection $I'(x, y) \leftarrow I(2x, 2y)$
- Upsample the image to twice the number of rows and columns by copying the pixels. $I'(2x, 2y) \leftarrow I'(2x + 1, 2y) \leftarrow I'(2x, 2y + 1) \leftarrow I'(2x + 1, 2y + 1) \leftarrow I(x, y)$
- Downsample the images by averaging pixels. $I'(x, y) \leftarrow \frac{1}{4} \sum_{c=2x}^{2x+1} \sum_{r=2y}^{2y+1} I(r, c)$
- Upsample the image by interpolation.

$$\begin{aligned} I'(2x, 2y) &\leftarrow I(x, y) \\ I'(2x + 1, 2y) &\leftarrow \frac{1}{2}(I(x, y) + I(x + 1, y)) \\ I'(2x, 2y + 1) &\leftarrow \frac{1}{2}(I(x, y) + I(x, y + 1)) \\ I'(2x + 1, 2y + 1) &\leftarrow \frac{1}{4}(I(x, y) + I(x + 1, y) + I(x, y + 1) + I(x + 1, y + 1)) \end{aligned}$$

- You are welcome to add other methods to downsample and upsample.
- You can perform intensity downsampling by $n$ bits ($0 \leq n \leq 7$) by $I'(x, y) \leftarrow (I(x, y) >> n) << n$. You should be able to do it on both color and grayscale images.

**Invoking the solution**

Your solution will be invoked using the following command:

```
sample -h -s sampling_method -d depth -i intensity imagefile
```

| | |
|---|---|
| `sample` | Name of your executable |
| `-s` | Sampling method; 1 for pixel deletion/replication, 2 for pixel averaging/interpolation [default: 1] |
| `depth` | Number of levels for downsampling [default: 1] |
| `intensity` | Intensity levels, between 1 and 7 [default: 1] |

The parameters prefixed with - are optional. You are free to use long parameter names such as `--depth` for `-d`. Display the image at each level of depth when you downsample and upsample. If both spatial and intensity sampling is asked for, apply intensity sampling after downsampling before display. Use an image from your corpus that you have developed. It may be a good idea to use an image of size $512 \times 512$ pixels as input.

**Suggested implementation steps**

1. Parse the command line. You can create your own parser or use the class `CommandLineParser` provided by OpenCV. Each of the optional arguments may have a default value. If the user specifies the option `-h`, print a help message and exit.

2. Apply downsampling followed by upsampling.
3. Apply intensity sampling.
4. Display each sampling step in a separate window. Display downsampled and upsampled image separately and include the level information in window name.
5. Make sure that you catch any exceptions thrown by OpenCV functions. They will help to save you a lot of trouble.

### Criteria for Success

Your code should successfully achieve all the downsampling/upsampling goals in both spatial and intensity sampling. Your code should handle both color and grayscale images appropriately. Describe your observations and conclusions in the file `README`.

### Grading

I'll use the following rubric to assess your submission.

1. *Overall submission; 30pts* Program compiles and upon reading, seems to be able to solve the assigned problem.
2. *Command line parsing; 10pts* Program is able to parse the command line appropriately, assigning defaults as needed; issues help if needed.
3. *Downsampling/upsampling by pixel deletion/replication; 10pts* Program is able todownsample/upsamle by pixel deletion/replication.
4. *Downsampling/upsampling by averaging/interpolation; 10pts* Program is able to downsample/upsample by pixel averaging/interpolation.
5. *Intensity sampling; 10pts* Program is able to sample inetnsity levels as specified.
6. *Code readability; 10pts* The code must be readable, with appropriate comments. Author and date should be identified.
7. *Code reusability; 10pts* The code is structured in the form of functions, in separate files with their own headers, to facilitate reusability.
8. *Exception handling; 5pts* The code should handle exceptions, thrown from within the code or OpenCV.
9. *Meta-files, 5pts* Meta-files such as `README`, `Makefile`, and revision history are provided and are appropriately written.

### Submission

Submit an electronic copy of all the sources, `README`, Makefile(s), and results. Create your programs in a directory called *username*.3 where *username* is your login name on delmar. This directory should be located in your `$HOME`. Once you are done with everything, *remove the executables and object files*, and issue the following commands:

```
% cd
% chmod 755 ~
% ~bhatias/bin/handin cs5420 3
% chmod 700 ~
```

Do not copy-paste these commands from the PDF; type in those commands.