Project 1: Digital Image Quantization

Austin Huyett, Kadri Nizam, Hayden Weber

September 12, 2022

Objective

We approached this project with several main objectives in mind:

- To familiarize ourselves with and implement algorithms related to downsampling, upsampling, and grayscale quantization
- To learn how to code in MATLAB
- To learn how to use LaTeXto typeset a professional looking report

Methods

Using MATLAB, we read in the reference image `walkbridge.tif` allowing for easy manipulation of the image. The image contained and extra channel which we suspect to be the *transparency* or *alpha* of the image which was not pertinent to this project, so we discarded the channel and only extracted the first channel which contained grayscale values. Throughout this report and in the code base, we hold the restriction that all image sizes are restricted to powers of 2.

Downsampling Algorithm

Sampling at intervals

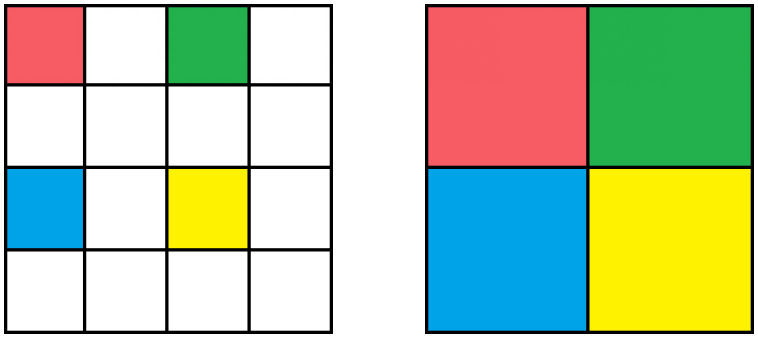
Our first task was to implement a downsampling algorithm to spatially downsample the original 512×512 image to an $N \times N$ image. Our initial approach was to sample at a constant interval. Let $p = (x, y)$ be a pixel at coordinate (x, y) in the original image and $f(p)$ is the grayscale value at that coordinate. Furthermore, let $M = 512/N$. Then we define the set

$$\mathcal{P} = \{p : x \bmod M = 0,\, y \bmod M = 0\}$$

where mod is the modulo operator with M as the modulus. We can then simply impose the requirement that

$$f(p_{\text{new}}) = f(p) \iff p \in \mathcal{P}$$

In essence, we sample the value of every M pixels in the original image and map them into adjacent pixels in a new image. We demonstrate this with an example in Figure 1.



(a) Sampled pixels from image (b) Final image

Figure 1: Example of sampling at intervals on a 4×4 grid with $M = 2$. Notice how only every top left pixel in a 2×2 grid from the original image, Figure 1(a), is chosen to be represented in the new image in Figure 1(b)

Pooling

An alternative implementation to the aforementioned sampling method is to perform a mathematical operation on a block of pixels instead. Consider downsampling the 512×512 image to $N \times N$ again. We can approach the problem by discretizing the original image into $M \times M$ pixel blocks where each pixel block maps to a single pixel in the new image. We can then decide on a mathematical function that maps values in the pixel block to a single value for its associated pixel. `max-pool` simply takes the maximal value from each pixel block as the new pixel value. `min-pool` takes the minimal value instead. `mean-pool` calculates the average pixel value within the pixel block and uses it as the new pixel value. This is the approach that we decided to implement in our code base.

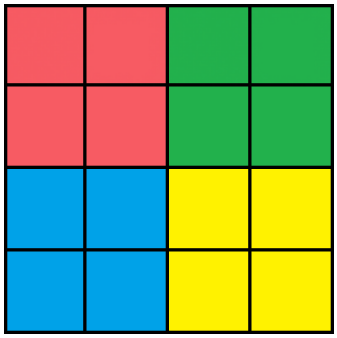


Figure 2: In contrast to Figure 1(a), all values in each 2×2 grid (colour block) is considered. A mathematical function is applied to pixel values in each colour block to form the new final image. By default, we used `max-pool` so the maximal value in each colour block is used in the new image

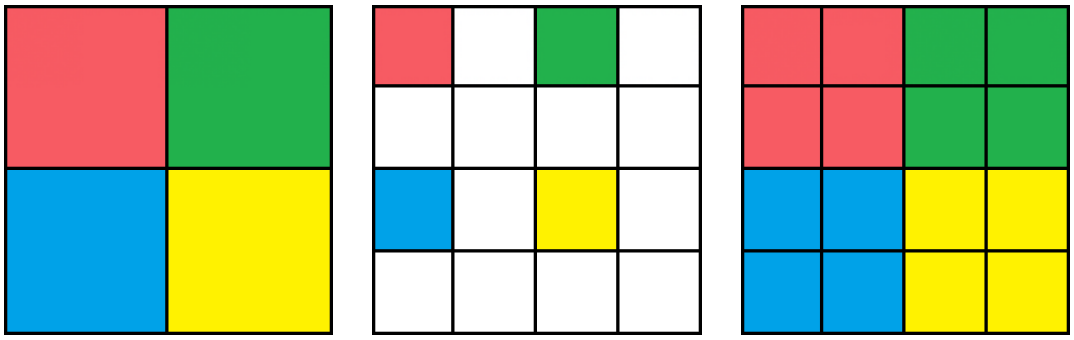
Upsampling Algorithm

We've implemented two upsampling algorithms in our project namely "Nearest-Neighbour" and "Bilinear Interpolation". We plan to add more functionality into the project in the future for

our own curiosity. We discuss the two algorithms here briefly.

Nearest-Neighbour

The nearest-neighbour algorithm feels somewhat like a "reverse" to that of the interval sampling method detailed in the **Downsampling Algorithm** section. Without loss of generality, assume we are upsampling from an $N \times N$ image to 512×512. We again define $M = 512/N$. Each pixel value from the original $N \times N$ image gets replicated into an $M \times M$ grid. The grid is combined with arrangements similar to that of the original image forming the final upsampled image. Figure 3 demonstrates this idea.



(a) Original image (b) Grid expanded (c) Final image

Figure 3: Starting from a 2×2 image in 3(a), we insert pixels in between all pixels and expand the grid to the desired dimensions (4×4 in this example) as in 3(b). The value from the original pixel is then replicated to the empty pixel forming the final image in 3(c)

Bilinear Interpolation

Consider two pixels on the same row of an image; $p_1 = (x_1, y)$ and $p_2 = (x_2, y)$ with grayscale values $f(p_1)$ and $f(p_2)$ respectively. We can create sub-integer pixels in between p_1 and p_2 by interpolating the x coordinate:

$$p_{\text{sub}} = (x_1 + \gamma(x_2 - x_1),\, y), \quad \gamma \in [0, 1] \subset \mathbb{R}.$$

Notice that when $\gamma = 0$ we recover p_1 and when $\gamma = 1$ we recover p_2. Equivalently, the grayscale values at these interpolated sub-pixels can also be interpolated using the same method

$$f(p_{\text{sub}}) = f(p_1) + \gamma[f(p_2) - f(p_1)], \quad \gamma \in [0, 1] \subset \mathbb{R}.$$

With this simple yet powerful idea in mind, bilinear interpolation is performed by interpolating between the existing pixels from the original image. The interpolation is performed in two steps: first along the rows, then along the columns **or** first along the columns, then along the rows. The value γ is dependent on the ratio of the dimensions in the original image and the dimensions of the target image. More concretely, using the $N \times N$ to 512×512 example again, we can define

$$\Delta x = \frac{N - 1}{512 - 1}.$$

The reason for subtracting 1 from both numerator and denominator is because we are dividing $N-1$ segments into $512-1$ indices (γ starts at 0). Then any pixel in the image can be found by

$$p_i = x_1 + (i-1)\Delta x, \quad i \in [1, 512] \subset \mathbb{Z}.$$

Thus, while performing interpolation, γ will be some multiple of Δx.

Grayscale Quantization

We've experimented with multiple methods of altering the bit-depth of the image. We present here the two methods.

Modulo Operation

The modulo operation returns the remainder of a division between two numbers. We are interested in lowering the bit-depth of the image to N bits where $N \in (0, 8) \subset \mathbb{Z}$. This corresponds to having 2^N grayscale intensities in the range $[0, 255]$. More concretely, if $f(p)$ is a grayscale value for some pixel p, we can perform the operation:

$$f_{\text{new}}(p) = f(p) - f(p) \bmod 2^{8-N} \tag{1}$$

Any values that are not divisible by 2^{8-N} will return a remainder in the range $[1, 2^{8-N} - 1]$. When subtracted from the original $f(p)$ value, we have constrained all values in the original image to be one of the 2^N possible grayscale intensities.

Domain Transformation

The alternative approach is by transforming the domain of the grayscale values. First, we normalize the image so its grayscale intensities lie in the range $[0, 1]$. It is important to note that the image must first be converted into a higher precision type (in our case a double) so information is not lost after normalization. Then we perform a scaling operation

$$f_{\text{new}}(p) = 255 \left(\frac{\lfloor f(p) \cdot (2^N - 1) \rceil}{2^N - 1} \right) \tag{2}$$

where $\lfloor \cdot \rceil$ is the rounding operation to the nearest integer. The numerator in Eqn. 2 ensures that all values in the image is converted to an integer in the range $[0, N-1]$. Note that the rounding operation can be substituted with a flooring or ceiling operation if so desired.

Code Structure and Execution Detail

All source code and relevant input files are located in the root directory of the zipped folder. The main source code file is labeled `main.m` and should be the starting point to exploring our code base. For general overview of our project, use

```
help main
```

in the MATLAB IDE console. Similarly, we've written docstrings for all functions, thus getting help on usage of any function can be found by simply typing

```
help function_name
```

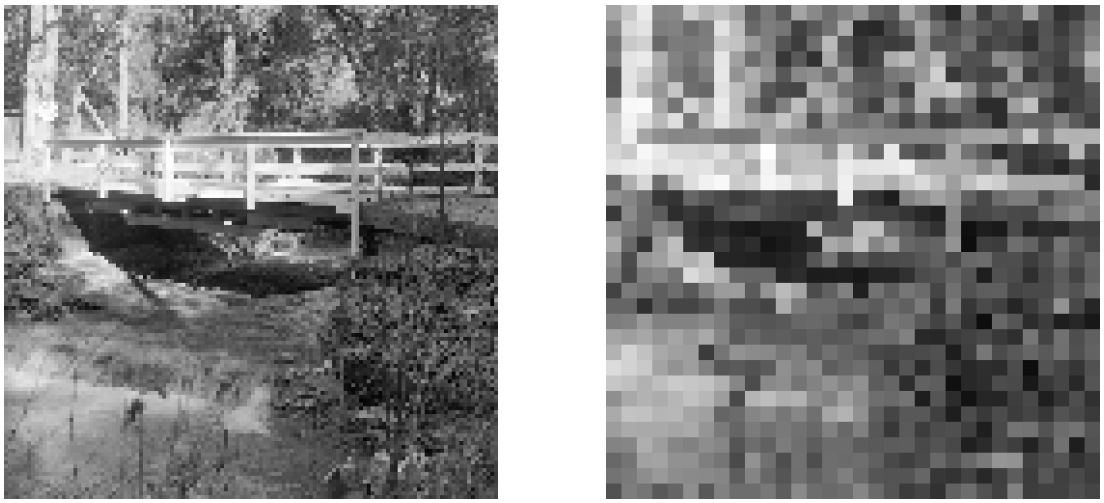
Executing our project can be done by typing `main` in the MATLAB IDE console or simply hitting the "Run" button on the IDE toolbar with `main.m` open and active. All output files can be found in the `output` folder.

Results

Downsampling, Nearest-Neighbour

(a) (b)

Figure 4: Results after downsampling from the original 512×512 image 4(a) to a 256×256 image 4(b). Deterioration of image quality is visible but minimal in the downsampled image especially in areas of highly contrasted pixel values.



(a) (b)

Figure 5: Results after downsampling the original 512×512 image 4(a) to a 128×128 image 5(a) and to a 32×32 image 5(b). Deterioration of image quality continues until the original image can no longer be perceived. The downsampled images look pixelated and details have been reduced. This phenomenon is especially observed when downsampled to 32×32 5(b).

Quantizing the Grayscale Values

(a) (b)

Figure 6: The original 512×512 `walkbridge.tif` image 4(a) reduced to a 7-bit grayscale 6(a) and a 6-bit grayscale 6(b).

(a) (b)

Figure 7: The original 512×512 `walkbridge.tif` image 4(a) reduced to a 5-bit grayscale 7(a) and a 4-bit grayscale 7(b).

(a)

(b)

Figure 8: The original 512×512 `walkbridge.tif` image 4(a) reduced to a 3-bit grayscale 8(a) and a 2-bit grayscale 8(b).

Figure 9: The original 512×512 `walkbridge.tif` image 4(a) reduced to a 1-bit (binary) grayscale.

Combining Downsampling and Grayscale Quantization

To study the combined affects of grayscale quantization and downsampling on an image, we used MATLAB to generate a new image of the `walkbridge.tif` Figure 4(a) that is downsampled to 256×256 spatial resolution and has a 6 bits/pixel grayscale resolution in Figure 10. When comparing these to images, it is clear that the new image, Figure 10, lacks the visual clarity of the original, Figure 4(a). The edges of the boards on the bridge are blurry, the edges of

Figure 10: The results of reducing the original 512×512 `walkbridge.tif` image 4(a) to a 256×256 spatial resolution and a 6 bits/pixel grayscale resolution.

the shadow under the bridge is jagged, and the trees and bushes in the background are very pixelated, grainy, and noisy. These are examples of artifacts that appeared in the new image that do not appear in the original high-resolution `walkbridge.tif` image.

Conclusion

We were able to utilize our new understanding of digital image processing concepts in order to implement all of the required algorithms for this project. We were able to utilize MATLAB and LaTeXsufficiently to meet the objectives stated at the start of this report. It was interesting to view alternate implementation of the same idea and to compare code within the group. With the methods we've implemented, it's interesting to see the pros and cons of each method. Upsampling with nearest-neighbour is easy, but the end result looks pixelated and blocky whereas bilinear looks better although blurry at an increased computation need. Altering gray-levels did not show too much of a degradation to image quality until below 6-bits to our surprise. For the future, we plan to implement the remaining upsampling methods such as cubic and inverse-distance to further compare the results.