Image Upscaling Using Bicubic Interpolation

Ellis Chen

ECPE 124

Abstract— **Image Upscaling is a common practice used in many display applications. Images are upscaled, where the number of pixels is increased by a linear factor. This increase in pixel count allows for higher resolution images to be displayed. When increasing the pixel count, there are unknown pixel intensity values that arise from newly created pixels. The problem lies in what intensity values will these new pixels take? One of the solutions to this common issue is using a technique called bicubic interpolation. In this project, I took a sample grayscale image called "cameraman.pgm" with dimensions of 256x256, and applied the bicubic interpolation algorithm on it in an attempt to upscale the image. The resulting image was a 512x512 image when scaled by a factor of 2, and a 767x767 image when scaled by a factor of 3. The 512x512 resulting image saw a more noticeable smoothing to diagonal lines and corners. The smoothing is more apparent with a 767x767 image, which lead to increasing sharpness of details. Scaling factors of 4 and beyond see diminishing differences in the smoothing of the images for small displays, to see the difference, the display has to be larger to accommodate the increase in pixels. For coloured images, the results are very similar, as the edges and corners are smoothed similar to the grayscale images.**

I. Bicubic Interpolation Introduction

Image upscaling is a common application that is widely used in TVs, monitors, phones, and many other devices that display images. An image is usually a static array of pixels with various colour or grayscale intensities, meaning that a 256x256 image will contain 256 pixels top to bottom and 256 pixels left to right. These dimensions are fixed, and the problem is displaying these fixed images on a display that has more pixels than the image itself, and the image is too small to see. Image upscaling resolves this issue by increasing the pixel count of the image. The increased pixel count provides a larger image to display, but the resolution of the image is dependent on the type of algorithm that upscales the image.

Increasing the pixel count of an image results in many of the new pixels being unassigned, meaning the application has to guess what pixel values these new pixels will take. The simplest method is to use the nearest neighbour method, in which the newly created pixels are determined by looking at its 4 nearest neighbours. Whichever value is closest to a neighbour, the new pixel will be assigned to that neighbour's value. This method will fill in any newly created pixels after the image is scaled, but diagonal edges and rounded corners will look unnecessarily jagged. Bilinear interpolation alleviates this problem slightly by providing more smooth details in the upscaled image. This method takes a line between the 4 nearest neighbours from left to right and top to bottom to find the most appropriate pixel value for that new pixel. This method makes edges and corners more defined because the method uses a linear fit to best approximate the best pixel values. Although, there is another algorithm that is able to provide even more smooth details when upscaling, and that is known as bicubic interpolation.

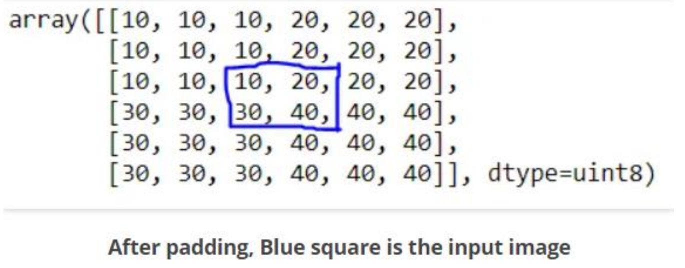
Bicubic interpolation uses the nearest 16 neighbours to figure out what is the best pixel value. As the name suggests, this method uses cubic splines that have gradients that are dependent on surrounding neighbours. This cubic approximation allows for more smooth transitions between edges of objects, which makes the resulting upscaled image larger without sacrificing a noticeable amount of detail in the images. Since this is also a common algorithm that is standardized in many commercial products like adobe photoshop, considering improving the algorithm was difficult and unordinary not feasible for my level during the time of this project.

This paper will cover my methodology in the implementation of this bicubic interpolation algorithm in section 2. The results from my implementation will be covered in section 3. Lastly, section 4 will consist of conclusions and future work.

II. Bicubic Interpolation Implementation

The implementation of the bicubic interpolation algorithm is follows:

1) The first step was to create a new output image that is dependent on the scale factor. With a scale factor of 2, a 256x256 input image will result in a 512x512 uninitialized output image. A scale factor of 3 will result in a 767x767 output image.
2) The pixel values of the original image are mapped to the corresponding pixel locations of the output image. After that, the output image is padded symmetrically along all 4 boundaries [1]



```
array([[10, 10, 10, 20, 20, 20],
       [10, 10, 10, 20, 20, 20],
       [10, 10, 10, 20, 20, 20],
       [30, 30, 30, 40, 40, 40],
       [30, 30, 30, 40, 40, 40],
       [30, 30, 30, 40, 40, 40]], dtype=uint8)
```

After padding, Blue square is the input image

Figure 1: example of padding the image

3) The difference between the output pixel and the floor of that value provides the distance to the top-left neighbours. This difference can be thought of the change in x and change in y as you move along the image.
4) Using the nearest 16 neighbours, the bicubic convolution algorithm requires solving a linear system shown in figure 2. This process is similar to applying a convolution using the kernel shown in figure 3, which was used in the implementation of the algorithm. [2]

$$p(x,y) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Figure 2: linear system required to find the unknown pixel value using 16 neighbours

$$W(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & \text{for } |x| \le 1, \\ a|x|^3 - 5a|x|^2 + 8a|x| - 4a & \text{for } 1 < |x| < 2, \\ 0 & \text{otherwise}, \end{cases}$$

Figure 3: Convolution formula for bicubic interpolation

$$\begin{aligned} b_{-1} &= p(t_x, f_{(-1,-1)}, f_{(0,-1)}, f_{(1,-1)}, f_{(2,-1)}), \\ b_0 &= p(t_x, f_{(-1,0)}, f_{(0,0)}, f_{(1,0)}, f_{(2,0)}), \\ b_1 &= p(t_x, f_{(-1,1)}, f_{(0,1)}, f_{(1,1)}, f_{(2,1)}), \\ b_2 &= p(t_x, f_{(-1,2)}, f_{(0,2)}, f_{(1,2)}, f_{(2,2)}), \\ p(x,y) &= p(t_y, b_{-1}, b_0, b_1, b_2). \end{aligned}$$

Figure 4: Visual representation of convolution for x-direction; also applied again for y-direction

5) The new pixel values are added to the output image for each neighbour that is visited. After interpolating the 16 neighbours, the output image should be an upscaled image with smoother edges and corners.

III. Results

The results of the bicubic interpolation did see a noticeable difference in the smoothness of edges. The original image that was used as a test sample was "cameraman.pgm". This grayscale image is 256x256 and is shown in figure 5.

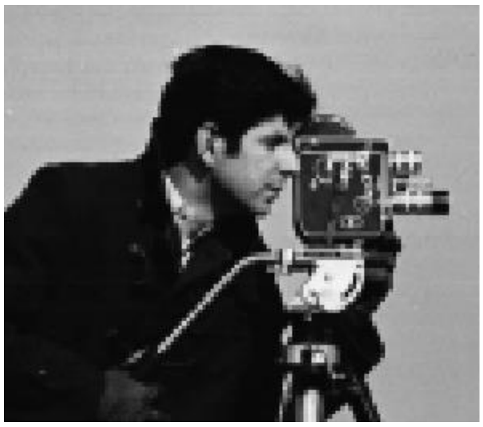


Figure 5: original 256x256 cameraman.pgm

The resulting image when using a scale factor of 2, resulted in a 512x512 grayscale image. Figure 6 shows the upscaled image and it clearly shows that the edges are more smoothed out. Although the image may look blurry, there are less jagged lines that are present in the original image.

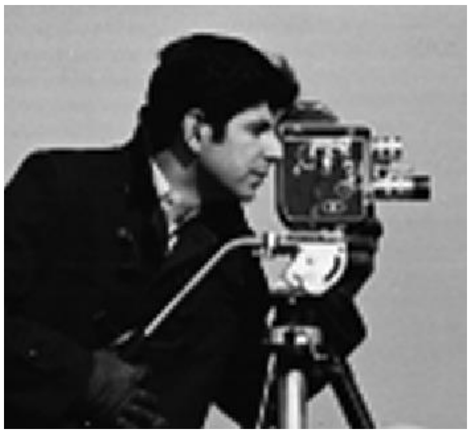


Figure 6: 512x512 upscaled image of cameraman.pgm

This implementation also works for coloured images. Figure 7 shows the original 256x256x3 "chalk.jpg" that is a coloured image. Since the interpolation algorithm works the same way for coloured images, just 2 more times because of the rbg channels, the resulting upscaled image was acceptable.

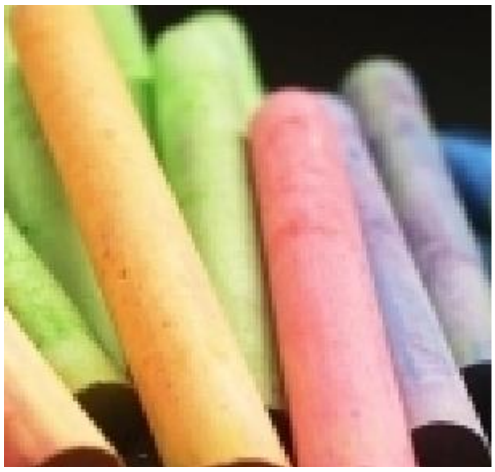


Figure 7: original 256x256x3 chalk.jpg

The upscaled image shown in figure 8 is a result from scaling the original 256x256x3 image by a factor of 3. So, the upscaled image is a 767x767x3 image. Similar to the grayscale upscaled image, the coloured upscaled image also shows signs of smoothening along edges. Nevertheless, the bicubic interpolation algorithm seems like it was correctly implemented.

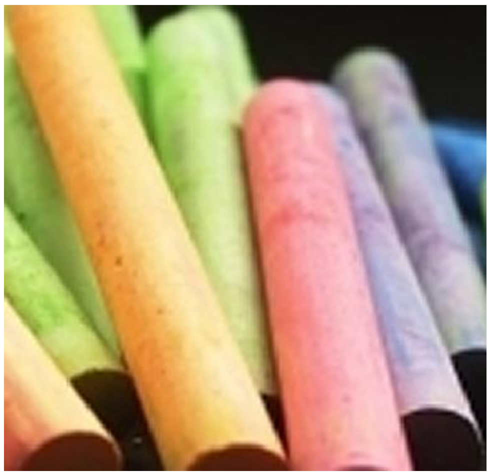


Figure 8: upscaled 767x767x3 chalk.jpg

IV. CONCLUSIONS

Overall, the bicubic interpolation algorithm was able to upscale images with fine detail. The method behind using cubic splines to approximate the best value using 16 nearest neighbours shows the efficiency of this implementation. Some noticeable potential problems with my implementation did arise during testing. Since the cubic splines are dependent on its 16 neighbours, it is possible that the cubic curves that travel from one extreme to another extreme may cause overshooting or undershooting. This means that any sharp or sporadic changes in pixel intensities may cause the approximation to be too high above the desired value or too low. This scenario can be visualized by looking at figure 9. Note that the data shown in figure 9 is not representative of any data resulting from my implementation, but is merely to serve as a visual example of a problem with bicubic interpolation.

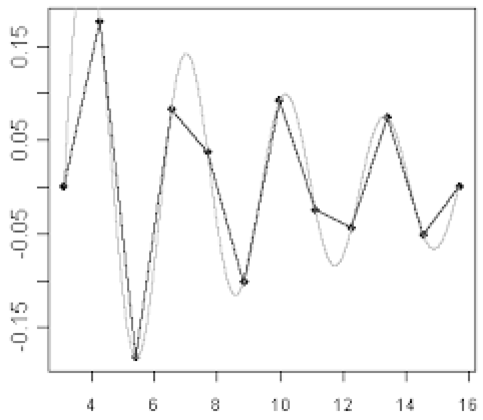


Figure 9: possible result of bicubic interpolation

One of the problems that may arise is overshooting; figure 9 shows some instances where the lobes are much greater than the neighbouring pixels, this overshooting is not considered as accurate if the difference is too great. Also, there is a potential to exceed the upper limit of 255 for grayscale images. The same case can be said for lobes that shoot too low; undershooting can cause the pixel to be too dark, or the intensity level may be under the lower limit of 0. In future work, this problem may be a suitable topic to investigate, and to improve. Also, there are many other algorithms that approach upscaling in various ways such as fourier-based interpolation that uses frequency domain, or edge-directed interpolation that is better suited for edge preservation.

Works Cited

[1] K. & Atul, K. & Atul, and Aastha Gupta 17 Jun 2020 at 6:09 pm, “Image Processing – Bicubic Interpolation,” *TheAILearner*, 29-Dec-2018. [Online]. Available: https://theailearner.com/2018/12/29/image-processing-bicubic-interpolation/. [Accessed: 09-Dec-2020].

[2] “Bicubic interpolation,” *Wikipedia*, 16-Apr-2020. [Online]. Available: https://en.wikipedia.org/wiki/Bicubic_interpolation. [Accessed: 09-Dec-2020].