A Novel Approach for Deblocking JPEG Images

Multidimensional DSP – Final Report

Eric Heinen

5/9/08

***Abstract* –** This paper presents a novel approach for deblocking JPEG images. First, original-image pixels are related to corresponding JPEG-image windows via multiple linear regression models. Then, the regression-model coefficients are used to filter the decoded JPEG image. A particular implementation of this approach was tested, and its performance was quantified using peak signal-to-noise ratio (PSNR) and the Structural Similarity Index (SSIM). Numerical results indicated general improvements in visual quality, and visual evidence indicates both reductions in blocking artifacts and improvements in contrast.

INTRODUCTION & OBJECTIVES

One of the main drawbacks of the JPEG standard is the introduction of blocking artifacts at high compression ratios. This is part of the reason why the JPEG2000 standard was developed. However, JPEG2000 is not completely immune to blocking, especially if many tiles are used. In addition, JPEG remains the most widely used algorithm for lossy compression of still images. So, there is still ongoing research into different methods for deblocking decoded JPEG (and JPEG2000 to a lesser degree) images in order to improve overall image quality [1].

My objectives for this project changed significantly from those stated in the Literature Survey report for two key reasons. First, I was unable to obtain code for the published work I had referenced. So, instead of implementing those algorithms myself, I concentrated on developing a novel approach for deblocking JPEG images and thus improving visual quality. Second, I had some trouble working with the open-source JPEG2000 codecs I was able to find. Therefore, I decided to try to make my approach general enough so that it might also be applied to JPEG2000.

APPROACH

First, let us assume that we have an original image and a JPEG encoded/decoded version with significant blocking artifacts. One very simple approach to reducing these blocking artifacts might be to smooth block-to-block transitions by equations (1) and (2). In equation (1), pixels adjacent horizontal block boundaries are updated with a weighted sum of the original value and the values of pixels above and below. Similarly, in equation (2), pixels adjacent vertical block boundaries are updated with a weighted sum of the original value and values of pixels left and right. This was exactly the first approach I tried, and my experiments showed that in most cases it improved image quality, but not very significantly. Also, I was unsure about how to determine the best values for the coefficients. However, this all gave me inspiration for developing the following, more sophisticated approach.

$$x_{i,j}{}' = a_0 \cdot x_{i,j} + a_1 \cdot x_{i+1,j} + a_2 \cdot x_{i-1,j} \ , \ \forall x_{i,j} \ s\,t \ (i \,\%\, 8) \in \{0, 1\} \ and \ i \notin \{1, M\} \qquad (1)$$

$$x_{i,j}{}' = b_0 \cdot x_{i,j} + b_1 \cdot x_{i,j+1} + b_2 \cdot x_{i,j-1} \ , \ \forall x_{i,j} \ s\,t \ \left(j \,\%\, 8\right) \in \{0, 1\} \ and \ j \notin \{1, N\} \qquad (2)$$

Now, consider the set of all original-image pixels such that the column number modulo eight is some constant. In other words, each of these pixels has the same horizontal relation to the left and right vertical block-boundaries. Also, for each of these original-image pixels, consider a window of jpeg-image pixels centered at the same location. Then, we can attempt to relate the original-image pixels to jpeg-image windows with a linear regression model, described by equation (3). In this equation, y_i represents a particular original-image pixel value, the x_{ij}'s represent the corresponding jpeg-image window's pixel values, and we try to find coefficients (β_j's) that minimize all of the error terms (ε_i's). Next, we can modify the jpeg-image by replacing appropriate pixels with their coefficient-weighted window sums, which can be seen as a type of spatially-dependent filter. The goal is to hopefully make each pixel values closer to its corresponding original-image value.

$$y_i = \sum_{j=1}^{n} x_{ij} \cdot \beta_j + \varepsilon_i \qquad (3)$$

The process I just outlined should then be extended to all eight column numbers and all eight row numbers in order to develop sixteen sets of coefficients. An obvious question that needs to be answered is how does the filtering process acquire the coefficients values? There are two options. First, the JPEG encoder could incorporate the regression modeling to determine coefficient values, and then encode them with the image. This technique requires a constant amount of memory overhead, and so does not really affect scaling that much. The other option would be to develop coefficients that work well for every image.

IMPLEMENTATION

In the implementation of my algorithm, I used 1x9 windows (four pixels to left and right of center) for column-wise regression and 9x1 windows (four pixels above and below center) for row-wise regression. The sixteen sets of nine coefficients were computed sequentially, meaning that one set of coefficients was applied (i.e. appropriate pixels updated) before the next set of coefficients was computed. I determined experimentally that in this type of sequential implementation, the order of computation affects performance. I did not test every possible order, but found that progressing outwards towards block boundaries works well. Therefore, the order I used was: col_5, col_4, row_5, row_4, col_6, col_3, row_6, row_3, col_7, col_2, row_7, row_2, col_8, col_1, row_8, row_1.

TESTING FRAMEWORK

In order to test the performance of my algorithm, I first collected a set of 20 test images from [6]. The images that I selected depicted a variety of subjects (people, animals, buildings, scenery, etc.) and included the well-known mandrill, peppers, and Lena images. Each of these images was converted to (if not already) 256 by 256 pixels, 8-bit precision grayscale images. JPEG compression and decompression was performed using freely available Matlab code from [5]. For quantization, I used the common quantization matrix given in equation (4), but scaled it in order to achieve relatively high compression ratios and significant blocking artifacts. Finally, image qualities were assessed using both PSNR and the Structural Similarity Index (SSIM) [7]. I chose to use SSIM because Sheikh et. al. determined that it is a pretty good image quality metric when compared to others [4]. Also, the code was freely available from [2].

$$\mathbf{Q} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \qquad (4)$$

RESULTS & DISCUSSION

For this project I tested four different approaches, the results of which are listed in Table 1, and summarized in Table 2. The algorithm titled “Simple Smooth” is the simple block-to-block smoothing described by equations (1) and (2) in a previous section. The algorithm titled “My Algo.” is the implementation of my approach described in the previous sections, and using coefficients computed per-image. Since this approach would require the encoder to pass additional information (i.e. the coefficients), it is sort of cheating. Therefore, I also tested my algorithm by using a single set of coefficients that was developed by training on all of the test images (“My Algo.*”). Finally, I tested the Shape-Adaptive Discrete Cosine Transform (SA-DCT) deblocking algorithm [1], which was implemented in Matlab and made freely available [3].

		Percentage Increase in PSNR				**Percentage Increase in SSIM**			
Im	**BPP**	**Simple Smooth**	**My Algo.**	**My Algo.***	**SA-DCT**	**Simple Smooth**	**My Algo.**	**My Algo.***	**SA-DCT**
1	0.274	0.243%	2.447%	1.937%	-5.581%	2.511%	2.895%	2.282%	0.167%
2	0.277	-0.899%	2.190%	1.532%	-2.543%	1.835%	2.990%	2.258%	-2.492%
3	0.271	-1.087%	2.434%	1.807%	-4.688%	1.520%	2.650%	2.182%	4.171%
4	0.469	-1.939%	2.717%	2.067%	-8.589%	0.284%	2.713%	2.302%	1.046%
5	0.280	-2.069%	2.681%	1.198%	5.619%	2.138%	3.306%	2.428%	3.593%
6	0.569	-2.720%	1.227%	0.723%	-0.195%	-14.54%	-2.919%	-4.204%	-24.41%
7	0.386	-2.298%	1.914%	1.428%	-0.758%	0.126%	2.207%	1.950%	-0.129%
8	0.421	-4.091%	2.004%	1.210%	-2.453%	-0.196%	2.258%	1.942%	2.072%
9	0.520	-3.683%	1.909%	1.212%	-0.287%	-2.104%	2.036%	1.576%	-4.143%
10	0.398	-1.918%	2.115%	1.502%	-8.556%	2.112%	3.170%	2.840%	3.893%
11	0.292	-2.363%	2.955%	1.984%	-10.94%	1.056%	1.898%	1.600%	2.924%
12	0.235	0.756%	1.620%	1.366%	-3.550%	0.934%	2.032%	1.881%	-7.712%
13	0.586	-2.913%	1.502%	1.063%	-2.582%	-9.111%	-0.829%	-1.563%	-23.82%
14	0.462	-1.855%	2.702%	1.924%	-8.627%	-1.738%	2.403%	1.603%	-10.30%
15	0.508	-2.976%	1.310%	0.765%	-0.056%	-3.843%	1.283%	0.492%	-8.876%
16	0.443	-3.474%	1.928%	1.526%	-0.662%	-3.918%	1.469%	1.064%	-10.27%
17	0.355	0.138%	2.433%	2.056%	0.467%	2.609%	3.445%	2.726%	1.959%
18	0.475	-3.281%	1.825%	1.222%	0.919%	-3.968%	1.425%	0.585%	-5.226%
19	0.454	-0.761%	1.653%	1.341%	-35.68%	-4.970%	-0.174%	-0.225%	-9.499%
20	0.282	0.435%	1.481%	1.320%	-38.39%	0.698%	0.540%	1.213%	-13.86%

Table 1. Results by test image number.

	Improvement in PSNR				**Improvement in SSIM**			
	Num	**Avg %**	**Best %**	**Worst %**	**Num**	**Avg %**	**Best %**	**Worst %**
Simple Smooth	4	-1.8378%	0.756%	-4.091%	11	-1.428%	2.609%	-14.541%
My Algo.	20	2.052%	2.955%	1.227%	17	1.740%	3.445%	-2.919%
My Algo.*	20	1.459%	2.067%	0.723%	17	1.247%	2.840%	-4.204%
SA-DCT	3	-6.357%	5.619%	-38.391%	8	-5.045%	4.17%	-24.41%

Table 2. Summary of experimental results.

As you can see in Tables 1 and 2, there a few test images for which the SA-DCT deblocking algorithm performed well, but in many cases it significantly hurt image quality as measured by both PSNR and SSIM. As shown in Figure 1, the SA-DCT algorithm causes a significant amount of blurring, and many of the image features (hair, teeth, hat, etc.) are completely lost. I believe that this algorithm may perform better with different parameters or at higher compression ratios, although I did not test these hypotheses on the entire image set.

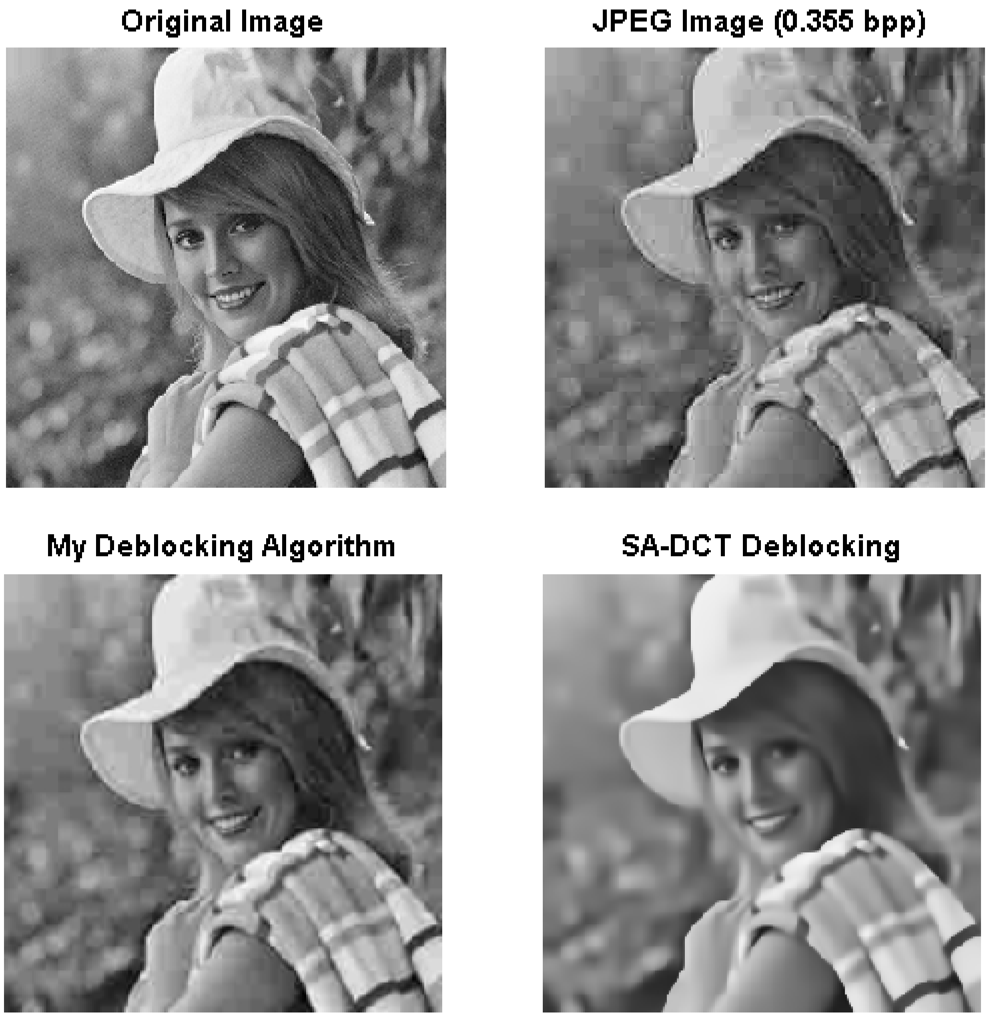

Figure 1. A visual comparison of deblocking algorithms.

For the particular quantization under test, my algorithm performed significantly better. And, as shown in Figure 1, my algorithm was able to reduce some of the blocking artifacts without blurring. Also, it is interesting to note that in the JPEG image, you can see that the woman’s hat is darker than in the original image. However, my algorithm helps to increase the luminance there and in other parts of the image, thus restoring the overall image contrast. Finally, although the numerical results listed in Tables 1 and 2 may not look too impressive, I ran a few tests with higher compression ratios and the actual percentage improvements were in general more significant.

FUTURE WORK & CONCLUSION

In conclusion, my novel approach for deblocking JPEG images looks promising when compared to an existing technique. However, there are many possibilities for future work. First, my approach is a very general framework and I fully tested only one particular type of implementation. Other implementations using different windows could be explored. Also, instead of computing coefficients with respect to row or column, 64 sets of coefficients might be computed, one for each pixel location within a block. Next, for this project I performed testing with one particular selection of the quantization matrix. In the future, testing should be done with different levels of compression. I think that coefficients computed using one particular quantization should work for other levels of compression. However, better performance might be achieved by using coefficients that are dependent on the actual level of compression.

REFERENCES

[1] A. Foi, V. Katkovnik, and K. Egiazarian, "Pointwise Shape-Adaptive DCT for High-Quality Denoising and Deblocking of Grayscale and Color Images", *IEEE Trans. Image Process.*, vol. 16, no. 5, pp. 1395-1411, May 2007.

[2] Department of Electrical and Computer Engineering, University of Waterloo, "The SSIM Index for Image Quality Assessment," May 2008, http://www.ece.uwaterloo.ca/~z70wang/research/ssim/ssim_index.m.

[3] Department of Signal Processing, Tampere University of Technology, "Pointwise Shape-Adaptive DCT Demobox," May 2008, http://www.cs.tut.fi/~foi/SA-DCT/SA-DCT_Demobox_v140.zip.

[4] H. R. Sheikh, M. F. Sabir, and A. C. Bovik, "A Statistical Evaluation of Recent Full Reference Image Quality Assessment Algorithms," *IEEE Trans. on Image Processing*, vol. 15, no. 11, pp. 3441-3452, November 2006.

[5] M. A. Azim, "JPEG Encoder Decoder," Matlab Central File Exchange, May 2008, http://www.mathworks.com/matlabcentral/fileexchange/.

[6] Signal and Image Processing Institute, University of Southern California, “The USC-SIPI Image Database,” May 2008, http://sipi.usc.edu/database/.

[7] Z. Wang, E. P. Simoncelli, and A. C. Bovik, “Multi-scale Structural Similarity for Image Quality Assessment,” *Proc. IEEE Asilomar Conf. Signals, Systems, and Computers*, November 2003.