Multiplication-Free Approximate Algorithms for Compressed Domain Linear Operations on Images

Neri Merhav
Computer Systems Laboratory and HP-ISC*

Keywords: image processing, compressed-domain processing, downsampling, scaling, translation, filtering, convolution, image compression, discrete cosine transform.

Abstract

We propose a method for devising approximate multiplication-free algorithms for compressed-domain linear operations on images, e.g., downsampling, translation, filtering, etc. We demonstrate that the approximate algorithms give output images that are nearly perceptually equivalent to those of the exact processing, while the computational complexity is significantly reduced.

*Current address: Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, U.S.A. Email: merhav@hpl.hp.com. Permanent address: HP Israel Science Center, Technion City, Haifa 32000, Israel. Email: merhav@hp.technion.ac.il

1 Introduction

Compressed-domain processing of digital images and video streams is a problem area of rapidly increasing interest in the last few years (see, e.g., [2], [8], [9], [10], [11], and references therein). Most of the research efforts thus far have been directed to *exact* algorithms, that provide the precise desired processing. Relatively little attention, on the other hand, has been devoted to the approach of devising *approximate* algorithms with significantly reduced computational complexity, and without, or almost without, sacrificing quality.

One example that falls in the category of the approximate approach, in the context of linear filtering, is the work of Bhaskaran *el al.* [1], who proposed a method of image sharpening by designing certain gain factors in the discrete cosine transform (DCT) domain. The main idea in [1] is that multiplication of each DCT coefficient by a constant, which roughly mimics the operation of linear filtering, can be implemented by modifying the de-quantization tables associated with the compression standard (JPEG, MPEG, and others). Consequently, this operation is computationally costless beyond the cost of the decompression itself. This idea has been further consolidated and extended in [4], where a method has been proposed to optimally design the DCT-domain gain factors, so as to best approximate the response of a given convolution kernel. It has been shown in [4] that the operation of certain 2D filters, often used in image processing, can be well approximated by this approach. For other useful filters, on the other hand, this method does not provide satisfactory results. Another example of a reported compressed-domain approximate algorithm is that of Natarajan and Bhaskaran [7], who proposed to approximate the compressed-domain operation of spatial domain downsampling, by quantizing each element of the matrices corresponding to this linear transformation, to the nearest integer power of 2. By doing this, multiplication by these matrices can be implemented by simple shifts and adds.

In this document, we combine the ideas of [1], [4], and [7], and further extend their scope to general linear operations in the compressed domain. For the sake of simplicity, and to fix ideas, consider the 1D case, where the 2D case will be obtained by just repeatedly applying the following to each row and then to each column. Suppose that the exact output DCT vector Y corresponding to some linear operation (e.g., downsampling, convolution, etc.) can be represented by a weighted sum

$$Y = \sum_i A_i X_i, \tag{1}$$

where $\{X_i\}$ are DCT input vectors and $\{A_i\}$ are certain fixed matrices. The approach proposed herein is to approximate Y by

$$\tilde{Y} = D_2 \sum_i \hat{A}_i D_1 X_i, \tag{2}$$

where $\{\hat{A}_i\}$ contain only 0, ± 1, $\pm\frac{1}{2}$, $\pm\frac{1}{4}$, and $\pm\frac{1}{8}$ as elements, and where D_1 and D_2 are optimally-designed diagonal matrices, so as to best approximate $\{A_i\}$ by $\{D_2 \hat{A}_i D_1\}$ in some reasonable sense. By using this form, multiplication by D_1 can be absorbed in the de-quantization step (by appropriately modifying the de-quantization table), multiplication by each A_i involves shifts and adds only, and finally, multiplication by D_2 can be made part of the re-quantization step. Since the de-quantization and the re-quantization are assumed to be performed anyhow, as ingredients of a partial decompression-recompression process, the only additional calculations that must be performed explicitly are the shifts and adds associated with each $\hat{A}_i$. Therefore, the implementation of eq. (2) is virtually multiplication-free.

We demonstrate the potential of this approach for three useful linear operations: downsampling by a factor of 2 in each dimension, translation (by 4 pixels in each direction), and convolution with the 5×5 uniformly weighted averaging kernel (which is not well-handled by the method of [4]). In all these cases, the approximate schemes provides output images that are perceptually equivalent, or almost equivalent (with PSNR = 35 − 45dB) to those of their exact counterparts, but with computational burden significantly reduced compared to the exact DCT-domain method and the spatial domain method. For downsampling and convolution, there is also a significant improvement in quality (3.5-5dB for downsampling, 13-18dB for convolution) over the approach proposed in [7], where $\hat{A}_i$ is just A_i quantized to the set $\{0, \pm 1, \pm\frac{1}{2} \pm \frac{1}{4}, \pm\frac{1}{8}\}$ and $D_1 = D_2 = I$. Therefore, the conclusion is that the optimization of D_1 and D_2 is very recommended.

2 The Approximation Method

Many useful linear row-column operations on images can be represented in the form of a weighted sum of input blocks. Consider an input image divided into a sequence of $N \times N$ spatial domain blocks (matrices) $x_1, x_2, \ldots$, and an output image with the same block structure $y_1, y_2, \ldots$, where

$$y_t = \sum_k a_k x_{f(t,k)} b_k^T, \tag{3}$$

$\{a_k\}$ and $\{b_k\}$ are fixed $N \times N$ matrices,[1] $\{f(t,k)\}$ are integers depending on t and k, and the superscript T denotes vector/matrix transposition. Common examples are those of downsampling, where $\{a_k\}$ and $\{b_k\}$ comprise the operations of anti-aliasing filtering and omission of every second sample, and filtering where the rows of $\{a_k\}$ and the columns of $\{b_k\}$ are shifted versions of the convolution kernel (see also [2] for more details and other examples.)

In the compressed (DCT) domain, the distributive property of the DCT operator w.r.t. matrix multiplication, translates eq. (3) into

$$Y_t = \sum_k A_k X_{f(t,k)} B_k^t, \tag{4}$$

where A_k, B_k, X_l and Y_l, are the 2D-DCT's of a_k, b_k, x_l, and y_l, respectively.

In order to simplify the description of our approximation approach, we confine attention to the 1D case, where it should be kept in mind that the 2D processing is given by applying the 1D operation for every row and every column. Thus, with a slight abuse of notation, from now on, $x_1, x_2, \ldots$ and $y_1, y_2, \ldots$ will denote sequences of non-overlapping N-dimensional column vectors of one-dimensional input and output signals, respectively, and $X_1, X_2, \ldots$ and $Y_1, Y_2, \ldots$ will denote the respective sequences of 1D-DCT vectors. We shall assume the 1D linear relation

$$y_t = \sum_k a_k x_{f(t,k)}, \tag{5}$$

[1] For compression standards like JPEG, MPEG, and H.261, $N = 8$.

and hence

$$Y_t = \sum_k A_k X_{f(t,k)}, \tag{6}$$

where A_k is again the 2D-DCT of the $N \times N$ matrix a_k.

We are interested in approximating each A_k by a product of three $N \times N$ matrices

$$\tilde{A}_k = D_2 \hat{A}_k D_1, \tag{7}$$

where D_1 and D_2 are diagonal matrices that are independent of k, and the elements of each $\hat{A}_k$ are all in the set $S \triangleq \{0, \pm 1, \pm\frac{1}{2} \pm \frac{1}{4}, \pm\frac{1}{8}\}$.

Since the (i,j)th element of $\tilde{A}_k$ is $\tilde{A}_k(i,j) = \hat{A}_k(i,j) D_1(i,i) D_2(j,j)$, a reasonable (though not necessarily optimal) choice of $\hat{A}_k$, is given by quantizing the elements of A_k to the above defined set S. The rationale behind this choice is two-fold: First, it guarantees that optimization w.r.t. D_1 and D_2 would yield results that cannot be worse than those provided by the approach of [7], where $\hat{A}_k$ is as above and $D_1 = D_2 = I$. Secondly, whenever possible, we would like to avoid situations where the diagonal elements of D_1 and D_2 are extremely far from unity since this might affect the quantization tables and hence also the compression ratio in an unexpected way.

Given $\{A_k\}$ and the above choice of $\{\hat{A}_k\}$, the next step will be to select D_1 and D_2 so that $\{\tilde{A}_k\}$ would be as 'close' as possible to $\{A_k\}$ in some sense. Let us assume that

$$R_{kl} = \mathbf{E} X_{f(t,k)} X^T_{f(t,l)} \tag{8}$$

is independent of t, and let us define the approximation criterion as

$$\begin{aligned}
\epsilon^2 &= \mathbf{E}||Y - \tilde{Y}||^2 \\
&= \mathrm{tr}\mathbf{E}(Y - \tilde{Y})(Y - \tilde{Y})^T \\
&= \mathrm{tr}\mathbf{E}[\sum_k (A_k - \tilde{A}_k) X_{f(t,k)}][\sum_l (A_l - \tilde{A}_l) X_{f(t,l)}]^T \\
&= \sum_k \sum_l \mathrm{tr}\left\{(A_k - D_2 \hat{A}_k D_1) R_{kl} (A_l - D_2 \hat{A}_l D_1)^T\right\}.
\end{aligned} \tag{9}$$

To minimize the last expression w.r.t. D_1 and D_2, we shall assume, for the sake of simplicity, a first order autoregressive model in the time domain. This means that the time domain autocorrelation between the ith and the jth samples of the 1D input signal is given by

$$r(i,j) = \sigma^2 \rho^{|i-j|}, \tag{10}$$

where $\sigma^2 > 0$ is a constant (whose value is immaterial for the purpose of optimizing D_1 and D_2), and $|\rho| \leq 1$. The DCT-domain cross-correlation matrix R_{kl} is, of course, the 2D-DCT of the respective time domain matrix $r_{kl} = \mathbf{E} x_{f(t,k)} x^T_{f(t,l)}$ created according to eq. (10).

We are not aware of a closed-form expression for the optimum matrices D_1^* and D_2^* that minimize ϵ^2. Therefore, for a given hypothesized ρ, we have used the Nedler-Meade simplex search algorithm for unconstrained optimization, which is implemented by the MATLAB library function `fmins`. A technical comment to observe is that in the minimization of ϵ^2 over the $(2N)$-dimensional space of the diagonal elements of D_1 and D_2, there is one superfluous degree of freedom. This follows from the simple fact that if D_1 is multiplied by an arbitrary real $\alpha \neq 0$ and D_2 is divided by α, then $\{\tilde{A}_k\}$ and hence also ϵ^2 are unaffected. Thus, without loss of optimality, we have fixed $D_1(1,1) = 1$ and optimized over the remaining $(2N-1)$ variables. The initial guess for the Nedler-Meade procedure was always chosen as $D_1 = D_2 = I$. The reasons for this choice are the same as those that were given above in the context of the choice of $\{\hat{A}_k\}$.

3 Applications and Implementation

We have applied the above described method to three different examples of linear transformations on images: downsampling by a factor of 2 in each dimension, convolution with the 5×5 uniform weight averaging filter, and image translation by 4 pixels north and 4 pixels west. In this section, we provide the details of the implementation of each one of these transformations, where in all of them, we have taken $N = 8$.

3.1 Downsampling

Downsampling by a factor of 2 in the 1D case, means that every two consecutive input vectors, x_{2t-1} and x_{2t}, $t = 1, 2, ...$, are mapped into one output vector given by $y_t = a_1 x_{2t-1} + a_2 x_{2t}$. For the simple case where the antialiasing filter performs uniformly weighted averaging of every two successive samples, the matrices a_1 and a_2 are given by

$$a_1 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad a_2 = \frac{1}{2}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}, \tag{11}$$

their 2D-DCT's are given by

$$A_1 = \begin{pmatrix} 0.5000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.4531 & 0.2039 & -0.0345 & 0.0095 & 0.0000 & -0.0064 & 0.0143 & -0.0406 \\ 0.0000 & 0.4904 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & -0.0975 \\ -0.1591 & 0.3879 & 0.2371 & -0.0406 & 0.0000 & 0.0271 & -0.0982 & -0.0772 \\ 0.0000 & 0.0000 & 0.4619 & 0.0000 & 0.0000 & 0.0000 & -0.1913 & 0.0000 \\ 0.1063 & -0.1728 & 0.3549 & 0.2039 & 0.0000 & -0.1362 & -0.1470 & 0.0344 \\ 0.0000 & 0.0000 & 0.0000 & 0.4157 & 0.0000 & -0.2778 & 0.0000 & 0.0000 \\ -0.0901 & 0.1362 & -0.1734 & 0.3599 & 0.0000 & -0.2405 & 0.0718 & -0.0271 \end{pmatrix},$$

and

$$A_2 = \begin{pmatrix} 0.5000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ -0.4531 & 0.2039 & 0.0345 & 0.0095 & 0.0000 & -0.0064 & -0.0143 & -0.0406 \\ 0.0000 & -0.4904 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0975 \\ 0.1591 & 0.3879 & -0.2371 & -0.0406 & 0.0000 & 0.0271 & 0.0982 & -0.0772 \\ 0.0000 & 0.0000 & 0.4619 & 0.0000 & 0.0000 & 0.0000 & -0.1913 & 0.0000 \\ -0.1063 & -0.1728 & -0.3549 & 0.2039 & 0.0000 & -0.1362 & 0.1470 & 0.0344 \\ 0.0000 & 0.0000 & 0.0000 & -0.4157 & 0.0000 & 0.2778 & 0.0000 & 0.0000 \\ 0.0901 & 0.1362 & 0.1734 & 0.3599 & 0.0000 & -0.2405 & -0.0718 & -0.0271 \end{pmatrix},$$

and finally, the quantized versions of A_1 and A_2 are

$$\hat{A}_1 = \begin{pmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & -0.125 \\ -0.125 & 0.5 & 0.25 & 0 & 0 & 0 & -0.125 & -0.125 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & -0.25 & 0 \\ 0.125 & -0.125 & 0.25 & 0.25 & 0 & -0.125 & -0.125 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & -0.25 & 0 & 0 \\ -0.125 & 0.125 & -0.125 & 0.25 & 0 & -0.25 & 0.125 & 0 \end{pmatrix},$$

and

$$\hat{A}_2 = \begin{pmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5 & 0.25 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0 & 0 & 0 & 0 & 0.125 \\ 0.125 & 0.5 & -0.25 & 0 & 0 & 0 & 0.125 & -0.125 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & -0.25 & 0 \\ -0.125 & -0.125 & -0.25 & 0.25 & 0 & -0.125 & 0.125 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0.25 & 0 & 0 \\ 0.125 & 0.125 & 0.125 & 0.25 & 0 & -0.25 & -0.125 & 0 \end{pmatrix}.$$

Consider now the implementation the equation

$$U = \hat{A}_1 V + \hat{A}_2 W, \tag{12}$$

where U, V, and W are 8-dimensional column vectors whose ith coordinates are denoted by u_i, v_i and w_i, $i = 1, ..., 8$, respectively. Since the corresponding elements of $\hat{A}_1$ and $\hat{A}_2$ differ at most by sign from each other (as can seen from elementary properties of the DCT), the implementation of eq. (12) can be made relatively efficient by applying the following steps:

$$\begin{aligned} u_1 &= (v_1 + w_1)/2 && (13) \\ z_1 &= (v_1 - w_1)/2 && (14) \\ z_2 &= (v_2 + w_2)/2 && (15) \\ u_2 &= z_1 + z_2/2 && (16) \\ u_3 &= [v_2 - w_2 + (w_8 - v_8)/4]/2 && (17) \end{aligned}$$

$$
\begin{aligned}
z_3 &= (v_3 - w_3)/4 && (18)\\
z_4 &= (v_7 - w_7)/8 && (19)\\
u_4 &= z_2 + z_3 - z_4 - [z_1 + (v_8 + w_8)/2]/4 && (20)\\
u_5 &= [v_3 + w_3 - (v_7 + w_7)/2]/2 && (21)\\
z_5 &= (v_4 + w_4)/4 && (22)\\
z_6 &= (v_6 + w_6)/4 && (23)\\
z_7 &= -z_4 + (z_1 - z_2)/4 && (24)\\
u_6 &= z_3 + z_5 + z_7 - z_6/2 && (25)\\
u_7 &= [v_4 - w_4 + (w_6 - v_6)/2]/2 && (26)\\
u_8 &= z_5 - z_6 - z_7 - z_3/2 && (27)
\end{aligned}
$$

These equations require altogether, 21 additions, 9 shift-and-adds, and 10 shifts. In the 2D case, every four 8×8 input blocks give one output block, by repeating the implementation of eq. (12) for every column, and then for every row, that is, 16 times. Therefore, the overall complexity in the 2D case is 336 additions, 144 shift-and-adds, and 160 shifts, i.e., a total of 640 basic operations per output block.

For comparison, the exact algorithm proposed in [3] requires 2824 operations and the spatial domain method needs 4512 operations. The approximate algorithm reported in [7] takes 696 adds and 184 shift, that is, 880 operations. However, it should be kept in mind that the matrices A_1 and A_2, and hence also $\hat{A}_1$ and $\hat{A}_2$ in [7] are somewhat different than the above.

3.2 Convolution

Convolution, or filtering, with a separable, noncausal, symmetric convolution kernel of maximum length 17, $\{h_8, h_7, ..., h_1, h_0, h_1, ..., h_7, h_8\}$, can be represented by $y_t = a_1 x_{t-1} + a_2 x_t + a_1^T x_{t+1}$, where a_1 and a_2 are Toeplitz matrices, with the ith row of a_1 given by $(i-1)$ zeros followed by $(h_8, ..., h_i)$, and the ith row of a_2 given by $(h_{i-1}, ..., h_0, ..., h_{8-i})$, $i = 1, ..., 8$.

If, for example, $h_0 = h_1 = h_2 = 0.2$, and $h_3 = h_4 = \cdots = h_8 = 0$, then the filter simply performs uniform averaging of 5 consecutive points. The 2D version of this filter (that averages all pixels in a 5×5 square around the current pixel) is useful in noise cleaning applications. Throughout this subsection, we confine attention to this example.[2] The DCT-domain version of the input-output relation is given by $Y_t = A_1 X_{t-1} + A_2 X_t + A_1^T X_{t+1}$, where

$$
A_1 = \begin{pmatrix}
0.0750 & -0.0987 & 0.0789 & -0.0519 & 0.0250 & -0.0046 & -0.0056 & 0.0058\\
0.0987 & -0.1296 & 0.1025 & -0.0658 & 0.0294 & -0.0022 & -0.0106 & 0.0096\\
0.0789 & -0.1025 & 0.0780 & -0.0453 & 0.0135 & 0.0090 & -0.0177 & 0.0129\\
0.0519 & -0.0658 & 0.0453 & -0.0183 & -0.0069 & 0.0231 & -0.0262 & 0.0169\\
0.0250 & -0.0294 & 0.0135 & 0.0069 & -0.0250 & 0.0347 & -0.0327 & 0.0196\\
0.0046 & -0.0022 & -0.0090 & 0.0231 & -0.0347 & 0.0391 & -0.0338 & 0.0196\\
-0.0056 & 0.0106 & -0.0177 & 0.0262 & -0.0327 & 0.0338 & -0.0280 & 0.0159\\
-0.0058 & 0.0096 & -0.0129 & 0.0169 & -0.0196 & 0.0196 & -0.0159 & 0.0089
\end{pmatrix},
$$

[2] While in principle, there are infinitely many possible kernels and each one yields a different approximate algorithm, in practice, a rather limited variety of kernels are commonly used in image processing.

and

$$A_2 = \begin{pmatrix} 0.8500 & 0.0000 & -0.1577 & 0.0000 & -0.0500 & 0.0000 & 0.0112 & 0.0000 \\ 0.0000 & 0.5931 & 0.0000 & -0.1315 & 0.0000 & -0.0045 & 0.0000 & 0.0191 \\ -0.1577 & 0.0000 & 0.3268 & 0.0000 & -0.0271 & 0.0000 & 0.0354 & 0.0000 \\ 0.0000 & -0.1315 & 0.0000 & 0.0335 & 0.0000 & 0.0462 & 0.0000 & 0.0338 \\ -0.0500 & 0.0000 & -0.0271 & 0.0000 & -0.1500 & 0.0000 & 0.0653 & 0.0000 \\ 0.0000 & -0.0045 & 0.0000 & 0.0462 & 0.0000 & -0.1578 & 0.0000 & 0.0392 \\ 0.0112 & 0.0000 & 0.0354 & 0.0000 & 0.0653 & 0.0000 & -0.0268 & 0.0000 \\ 0.0000 & 0.0191 & 0.0000 & 0.0338 & 0.0000 & 0.0392 & 0.0000 & 0.1312 \end{pmatrix}.$$

Thus, $\tilde{Y}_t = D_2(\hat{A}_1 D_1 X_{t-1} + \hat{A}_2 D_1 X_t + \hat{A}_1^T D_1 X_{t+1})/2$, [3] where

$$\hat{A}_1 = \begin{pmatrix} 0.125 & -0.25 & 0.125 & -0.125 & 0 & 0 & 0 & 0 \\ 0.25 & -0.25 & 0.25 & -0.125 & 0 & 0 & 0 & 0 \\ 0.125 & -0.25 & 0.125 & -0.125 & 0 & 0 & 0 & 0 \\ 0.125 & -0.125 & 0.125 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.125 & -0.125 & 0 \\ 0 & 0 & 0 & 0 & -0.125 & 0.125 & -0.125 & 0 \\ 0 & 0 & 0 & 0 & -0.125 & 0.125 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{28}$$

and

$$\hat{A}_2 = \begin{pmatrix} 2 & 0 & -0.250 & 0 & -0.125 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.25 & 0 & 0 & 0 & 0 \\ -0.25 & 0 & 0.5 & 0 & 0 & 0 & 0.125 & 0 \\ 0 & -0.25 & 0 & 0.125 & 0 & 0 & 0 & 0.125 \\ -0.125 & 0 & 0 & 0 & -0.25 & 0 & 0.125 & 0 \\ 0 & 0 & 0 & 0.125 & 0 & -0.25 & 0 & 0.125 \\ 0 & 0 & 0.125 & 0 & 0.125 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.125 & 0 & 0.125 & 0 & 0.125 \end{pmatrix}. \tag{29}$$

Consider next, the implementation of the equation

$$U = \hat{A}_1 V + \hat{A}_2 W + \hat{A}_1^T S, \tag{30}$$

where similarly as U, V, and W, S is also an 8-dimensional column vector $(s_1, ..., s_8)$. By using the specific structures of $\hat{A}_1$ and $\hat{A}_2$, eq. (30) can be implemented as follows:

$$z_1 = [s_2 - v_2 + (v_1 + s_1 + v_3 + s_3 + s_4 - v_4)/2]/4 \tag{31}$$
$$u_1 = w_1 + w_1 + z_1 - (w_3 + w_5/2)/4 \tag{32}$$
$$z_2 = (v_1 - s_1 - v_2 - s_2 + v_3 - s_3)/4 \tag{33}$$
$$u_2 = z_2 + w_2 - [w_4 + (v_4 + s_4)/2]/4 \tag{34}$$

[3] Here, we have chosen to quantize $2A_1$ and $2A_2$ rather than A_1 of A_2 as the latter does not turn out to provide satisfactory results.

$$
\begin{aligned}
u_3 &= z_1 + [w_3 - (w_1 - w_7/2)/2]/2 && (35)\\
z_3 &= w_4 + w_8 && (36)\\
u_4 &= [z_2 - (w_2 - z_3/2)/2]/2 && (37)\\
z_4 &= (v_6 - s_6)/8 && (38)\\
u_5 &= z_4 - [w_5 + (w_1 + v_7 + s_7 - w_7)/2]/4 && (39)\\
u_6 &= [-w_6 + (z_3 + s_5 - v_5 + v_6 + s_6 + s_7 - v_7)/2]/4 && (40)\\
u_7 &= z_4 + (w_3 + w_5 - v_5 - s_5)/8 && (41)\\
u_8 &= (z_3 + w_6)/8. && (42)
\end{aligned}
$$

The implementation of this set of equations requires 31 additions, 14 shift-and-adds, and 6 shifts. Thus, when extended to the 2D case, this amounts to 496 additions, 224 shift-and-adds, and 96 shifts, i.e., a total of 816 basic operations per output block. For comparison, the spatial domain implementation of convolution with this specific filter, and two fast exact algorithms for convolution in the compressed domain, that were recently reported in [5] and [6], all require more than 4000 operations.

3.3 Translation

Translation, or shifting, by k positions to the right ($1 \leq k \leq 7$), can be represented, in the 1D case, by $y_t = a_1 x_t + a_2 x_{t+1}$, $t = 1, 2, \ldots$, where now a_1 and a_2 have the block structure

$$
a_1 = \begin{pmatrix} 0 & I_k \\ 0 & 0 \end{pmatrix}, \quad a_2 = \begin{pmatrix} 0 & 0 \\ I_{8-k} & 0 \end{pmatrix}, \tag{43}
$$

and where I_n is the $n \times n$ identity matrix. If $k = 4$, for example, the 2D-DCT's of a_1 and a_2 are given by

$$
\hat{A}_1 = \begin{pmatrix}
0.5000 & -0.4531 & 0.0000 & 0.1591 & 0.0000 & -0.1063 & 0.0000 & 0.0901 \\
0.4531 & -0.3266 & -0.2079 & 0.3266 & -0.0373 & -0.1353 & -0.0114 & 0.1353 \\
0.0000 & 0.2079 & -0.5000 & 0.3955 & 0.0000 & -0.1762 & 0.0000 & 0.1389 \\
-0.1591 & 0.3266 & -0.3955 & 0.1353 & 0.2566 & -0.3266 & 0.0488 & 0.1353 \\
0.0000 & 0.0373 & 0.0000 & -0.2566 & 0.5000 & -0.3841 & 0.0000 & 0.1877 \\
0.1063 & -0.1353 & 0.1762 & -0.3266 & 0.3841 & -0.1353 & -0.2452 & 0.3266 \\
0.0000 & 0.0114 & 0.0000 & -0.0488 & 0.0000 & 0.2452 & -0.5000 & 0.4329 \\
-0.0901 & 0.1353 & -0.1389 & 0.1353 & -0.1877 & 0.3266 & -0.4329 & 0.3266
\end{pmatrix},
$$

and

$$
\hat{A}_2 = \begin{pmatrix}
0.5000 & 0.4531 & 0.0000 & -0.1591 & 0.0000 & 0.1063 & 0.0000 & -0.0901 \\
-0.4531 & -0.3266 & 0.2079 & 0.3266 & 0.0373 & -0.1353 & 0.0114 & 0.1353 \\
0.0000 & -0.2079 & -0.5000 & -0.3955 & 0.0000 & 0.1762 & 0.0000 & -0.1389 \\
0.1591 & 0.3266 & 0.3955 & 0.1353 & -0.2566 & -0.3266 & -0.0488 & 0.1353 \\
0.0000 & -0.0373 & 0.0000 & 0.2566 & 0.5000 & 0.3841 & 0.0000 & -0.1877 \\
-0.1063 & -0.1353 & -0.1762 & -0.3266 & -0.3841 & -0.1353 & 0.2452 & 0.3266 \\
0.0000 & -0.0114 & 0.0000 & 0.0488 & 0.0000 & -0.2452 & -0.5000 & -0.4329 \\
0.0901 & 0.1353 & -0.1389 & 0.1353 & 0.1877 & 0.3266 & 0.4329 & 0.3266
\end{pmatrix},
$$

whose quantized versions are given by

$$\hat{A}_1 = \begin{pmatrix} 0.5 & -0.5 & 0 & 0.125 & 0 & -0.125 & 0 & 0.125 \\ 0.5 & -0.25 & -0.25 & 0.25 & 0 & -0.125 & 0 & 0.125 \\ 0 & 0.25 & -0.5 & 0.5 & 0 & -0.125 & 0 & 0.125 \\ -0.125 & 0.25 & -0.5 & 0.125 & 0.25 & -0.25 & 0 & 0.125 \\ 0 & 0 & 0 & -0.25 & 0.5 & -0.5 & 0 & 0.25 \\ 0.125 & -0.125 & 0.125 & -0.25 & 0.5 & -0.125 & -0.25 & 0.25 \\ 0 & 0 & 0 & 0 & 0 & 0.25 & -0.5 & 0.5 \\ -0.125 & 0.125 & -0.125 & 0.125 & -0.25 & 0.25 & -0.5 & 0.25 \end{pmatrix}, \qquad (44)$$

and

$$\hat{A}_2 = \begin{pmatrix} 0.5 & 0.5 & 0 & -0.125 & 0 & 0.125 & 0 & -0.125 \\ -0.5 & -0.25 & 0.25 & 0.25 & 0 & -0.125 & 0 & 0.125 \\ 0 & -0.25 & -0.5 & -0.5 & 0 & 0.125 & 0 & -0.125 \\ 0.125 & 0.25 & 0.5 & 0.125 & -0.25 & -0.25 & 0 & 0.125 \\ 0 & 0 & 0 & 0.25 & 0.5 & 0.5 & 0 & -0.25 \\ -0.125 & -0.125 & -0.125 & -0.25 & -0.5 & -0.125 & 0.25 & 0.25 \\ 0 & 0 & 0 & 0 & 0 & -0.25 & -0.5 & -0.5 \\ 0.125 & 0.125 & 0.125 & 0.125 & 0.25 & 0.25 & 0.5 & 0.25 \end{pmatrix}. \qquad (45)$$

Returning to the case of a general displacement parameter k, then here, as in the case of downsampling and filtering, one can also devise an efficient implementation of the equation $U = \hat{A}_1 V + \hat{A}_2 W$, by taking advantage of the structure of $\hat{A}_1$ and $\hat{A}_2$ for every possible $1 \leq k \leq 7$. For the sake of brevity, we omit the details of the implementation and the operation count associated with each value of k.

4 Experimental Results

We have examined the approximate compressed-domain algorithms developed in Section 3 for $N = 8$ and for the optimally-designed diagonal matrices D_1 and D_2 corresponding to the first order autoregressive model, where the correlation coefficient ρ takes on the values $0, 0.9, 0.95$, and 0.99.

Each one of the algorithms was tested on several images and compared to the exact approach (corresponding to multiplication by $\{A_k\}$) and with the approximate approach corresponding to $D_1 = D_2 = I$ (as proposed in [7]), where the purpose of the latter comparison is to assess the usefulness of the optimization over D_1 and D_2. In the exact approach, every compressed DCT block was first de-quantized according to the recommended (default) JPEG quantization table for luminance, and after processing according to $\{A_k\}$, re-quantized using the same table. In the approximate version of the algorithm, the (i,j)th element $(i,j = 1, ..., 8)$ of this table was multiplied by $D_1(i,i)D_1(j,j)$ in the de-quantization phase, and divided by $D_2(i,i)D_2(j,j)$ in the re-quantization phase, whereas the processing in between was according to $\{\hat{A}_k\}$.

Seven black-and-white images were used to test the algorithms: "Barbara", "Boats", "Hotel", "Lena", "Gold", "Zelda", and "Einstein". The first six are quite commonly used 'natural' images, i.e., photographs of people, buildings, views, and so on. The latter is a portrait drawing of Albert Einstein that contains also some scanned text.

For each one of these images, and for each version of the approximate algorithm (depending on the value of ρ or on whether $D_1 = D_2 = I$), the PSNR (in dB) of the resulting processed image w.r.t. that of the exact algorithm has been computed. The results for downsampling, convolution, and translation are summarized in Tables 1, 2, and 3, respectively.

As can be seen, at least for the natural images, and particularly for downsampling and convolution, the optimization over D_1 and D_2 (for large ρ) provides considerable improvement (3.5-5dB for downsampling, 13-18dB for convolution) compared to the case where D_1 and D_2 are assumed to be the identity matrix. The improvement achieved for the case of translation is less dramatic (1-2dB). Thus, this optimization is very recommended for downsampling and convolution but less important for translation. The best choice of ρ for most natural images examined appears to be 0.95 for downsampling and translation, and 0.99 for convolution. However, the results are relatively insensitive to the value of ρ in the range $[0.95, 0.99]$, and therefore, the exact choice of ρ therein is not critical. In all three transformations, the performance achieved on the "Einstein" image was considerably inferior to these of the natural images. One possible explanation to this fact is that the autoregressive model is not suitable for this type of images, and therefore other models should be sought.

Besides the objective performance measure of PSNR, it turns out also that the approximate method, for large ρ, gives output images that look significantly cleaner and smoother than those that correspond to $D_1 = D_2 = I$. Also, blockiness artifacts associated with the quantization of $\{A_k\}$ are compensated and significantly reduced when D_1 and D_2 are optimized. This is demonstrated in Fig. 1 for the case of downsampling. As can be seen, the blocky artifacts for $D_1 = D_2 = I$, that appear especially near Lena's shoulder, are considerably smaller for $\rho = 0.95$.

5 Conclusion and Future Work

We have proposed a method for approximating linear operations on images in compressed domain using multiplication-free schemes and optimum modification of the de-quantization/re-quantization tables. We have demonstrated the performance for three examples of linear transformations: downsampling, convolution, and translation. In the first two of these transformations, the experimental results indicate that the ingredient of modifying the quantization tables, both before and after processing, is of substantial importance for improving quality.

A few future research directions are the following: (i) Examining the potential of this approach for other linear transformations, e.g., rotation, shearing, and other affine coordinate transformations. (ii) Extending the scope to linear operations that are not row-column separable, e.g., convolution with a nonseparable kernel. This might be useful for compressed domain motion estimation, where the current block serves as a matched (iii) Devising a more rigorous method to select the intermidiate matrices $\{\hat{A}_k\}$. (iv) Finding better models and optimization criteria that are suitable for unnatural images such as "Einstein."

Image	$D_1 = D_2 = I$	$\rho = 0.00$	$\rho = 0.90$	$\rho = 0.95$	$\rho = 0.99$
Barbara	32.5	34.8	35.9	36.0	35.5
Boats	33.7	35.6	37.4	37.7	37.5
Einstein	29.1	31.2	30.9	30.5	29.9
Hotel	31.9	34.4	36.0	36.2	35.9
Lena	32.5	34.6	37.3	37.4	37.3
Gold	34.9	37.1	39.2	39.4	39.2
Zelda	36.0	37.7	40.1	40.6	40.5

Table 1: PSNR[dB] results for downsampling by a factor of 2.

Image	$D_1 = D_2 = I$	$\rho = 0.00$	$\rho = 0.90$	$\rho = 0.95$	$\rho = 0.99$
Barbara	25.5	31.1	39.3	40.3	40.3
Boats	26.9	33.5	40.0	41.2	41.3
Einstein	25.8	26.7	32.9	34.4	35.0
Hotel	23.8	30.9	37.7	39.2	39.6
Lena	24.4	31.5	38.8	40.4	40.9
Gold	24.2	31.1	38.7	41.1	42.5
Zelda	26.0	33.6	41.2	43.5	44.6

Table 2: PSNR[dB] results for convolution with the 5×5 uniform averaging kernel.

Image	$D_1 = D_2 = I$	$\rho = 0.00$	$\rho = 0.90$	$\rho = 0.95$	$\rho = 0.99$
Barbara	31.5	27.6	32.2	32.3	32.2
Boats	33.7	28.5	34.7	35.0	35.0
Einstein	27.0	21.4	26.9	27.1	27.1
Hotel	31.4	25.9	32.2	32.5	32.5
Lena	32.7	26.9	34.0	34.5	34.5
Gold	35.1	26.4	35.4	36.6	36.6
Zelda	37.1	28.9	37.9	39.1	38.9

Table 3: PSNR[dB] results for translation by 4 pixels north and 4 pixels west.

Figure 1: "Lena" downsampled by 2: Top left - exact algorithm, top right - $D_1 = D_2 = I$, bottom left - $\rho = 0$, bottom right - $\rho = 0.95$.

6 References

[1] V. Bhaskaran, G. Beretta, and K. Konstantinides, "Text and image sharpening of JPEG compressed images in the frequency domain," HPL Technical Report, HPL-94-90, October 1994.

[2] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE J. Sel. Areas Comm.*, vol. 13, no. 1, pp. 1-11, January 1995.

[3] N. Merhav and V. Bhaskaran, "A transform domain approach to spatial domain image scaling," HPL Technical Report, HPL-94-116, December 1994.

[4] N. Merhav and R. Kresch, "Approximate convolution using DCT coefficient multipliers," HPL Technical Report, HPL-95-141, December 1995.

[5] R. Kresch and N. Merhav, "Fast DCT domain filtering using the DCT and the DST," HPL Technical Report, HPL-95-140, December 1995.

[6] R. Kresch and N. Merhav, "An implicit DST-based method for fast convolution in the DCT domain," HPL Technical Report, HPL-96-68, May 1996.

[7] B. K. Natarajan and V. Bhaskaran, "A fast approximate algorithm for scaling down digital images in the DCT domain," *Proc. ICIP '95*, 1995.

[8] B. C. Smith, "Fast software processing of motion JPEG video," Proc. ACM Multimedia, 1994.

[9] B. Shen and I. K. Sethi, "Inner-block operations on compressed images," ACM Multimedia '95, November 1995.

[10] B. Shen and I. K. Sethi, "Scanline algorithms in compressed domain," preprint.

[11] F. Arman, A. Hsu, and M.-Y. Chiu, "Image processing on encoded video sequences," Multimedia Systems, vol. 1, 211-219, 1994.