A Fast Scheme for Altering Resolution in the Compressed Domain

Rakesh Dugad and Narendra Ahuja *
Department of Electrical and Computer Engineering
Beckman Institute, University of Illinois, Urbana, IL 61801.
dugad@vision.ai.uiuc.edu

Abstract

Given a video frame or image in terms of its 8×8 block-DCT coefficients we wish to obtain a downsized (lower resolution) or upsized (higher resolution) version of this frame also in terms of 8×8 block-DCT coefficients. We propose an algorithm for achieving this directly in the compressed domain which is computationally much faster, produces visually sharper images and gives significant improvements in PSNR (typically 4 dB better compared to other compressed domain methods based on bilinear interpolation). The downsampling and upsampling schemes combined together preserve all *the low-frequency DCT coefficients of the original signal. This implies tremendous savings for coding the difference between the original (unsampled image) and its prediction (the upsampled image). This is desirable for many applications based on scalable encoding of video.*

1 Introduction

The notion of hierarchical representation of a signal has been used in many contexts. For example in scale space theory [1, 2, 3, 4], the signal is represented from coarse to fine levels of detail to extract important structure at a continuum of scales, by convolving it with a Gaussian kernel whose standard deviation plays the role of scale. During the last few decades a number of other approaches to multi-scale representation like pyramids and wavelets have also gained popularity. The main difference between scale-space representation and the pyramid or wavelets representation is that the scale space representation uses the same spatial sampling density at all scales whereas in pyramid representations the main objective is to decrease the sampling density with coarser and coarser scales (or approximations). To reduce aliasing filtering is performed before subsampling. Hence with proper choice of filters (e.g. orthogonal wavelet representation) we can say that the multi-resolution (or pyramid) representation is non-redundant whereas the scale-space representation is maximally redundant [1]. Moreover the rapidly decreasing image size in the pyramid representation allows computationally efficient coarse-to-fine processing. Due to such reasons the pyramid structure using wavelet decomposition [5] is very suitable for compression or progressive encoding of the signal where as the scale-space approach is more suitable for immediate access to structure within the signal at all levels of scale.

However, the standards currently in practice do not use wavelet decompositions. Block-DCT is by far the most popular method for image compression. Keeping this in mind, we shall present in this paper a method for decreasing and increasing the resolution of images or video frames stored in terms of their block-DCT coefficients. For many applications one needs to produce a compressed bitstream containing the video at a different resolution than the original bitstream. For example for browsing a remote video database it would be more economical to send low-resolution versions of the video clips to the user and depending on his or her interest progressively enhance the resolution. A typical video conferencing application will require compositing video bitstreams at different resolutions into one bitstream containing the individual video frames at possibly lower resolution. Similarly for transmitting video over dual-priority networks we can transmit a low-resolution version of the video over high-priority channel and an enhancement layer over the low-priority channel. Similarly one solution to transmitting video over bandwidth-constrained channels is to transmit a low-resolution version of the video. Spatial scalability is also used in HDTV for maintaining compatibility with standards other than MPEG-2 (e.g. MPEG-1) and to allow for varied bitrate and computational capabilities of different users.

The straightforward approach of decompressing, carrying out the downsampling in spatial domain and then recompressing involves unnecessary work and is computationally too intensive to be feasible in real-time on currently available workstations. Hence recently there has been much work [6, 7, 8] in carrying out downsampling as well as other operations on video sequences directly in the compressed domain without requiring decompression and recompression.

Most previous approaches for downsampling in the compressed domain (say in the DCT domain) rely on the fact that the DCT, being a linear orthonormal transform, is distributive over matrix multiplication [9, 6, 7]. Let c_1, c_2, c_3, c_4 denote four adjacent 8×8

*The support of the office of Naval Research under grant N00014-96-1-0502 is gratefully acknowledged.

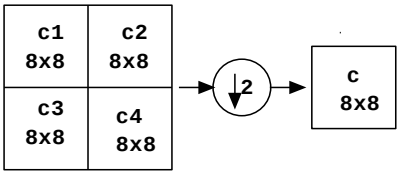

Figure 1: Downsampling 4 consecutive blocks to get a single block.

blocks in the spatial domain as shown in Figure 1 then the downsampled 8×8 block c can be written as: $c = \sum_{i=1}^{4} h_i c_i g_i$ where the downsampling filters h_i, g_i are usually taken as

$$h_1 = \begin{bmatrix} .5 & .5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .5 & .5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & .5 & .5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .5 & .5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

and similarly for the other h_i's and g_i's. Let T denote the 8×8 DCT operator matrix. Since $TT^t = T^tT = I$ we have

$$\mathrm{DCT}(c) = TcT^t = \sum_{i=1}^{4} \mathrm{DCT}(h_i)\mathrm{DCT}(c_i)\mathrm{DCT}(g_i) \quad (2)$$

$\mathrm{DCT}(h_i)$ and $\mathrm{DCT}(g_i)$ for i= 1 to 4 can be precomputed. Hence given $\mathrm{DCT}(c_i)$ for i= 1 to 4, $\mathrm{DCT}(c)$ can be computed as matrix multiplication. However even though the filter matrices h_i and g_i are sparse, $\mathrm{DCT}(h_i)$ and $\mathrm{DCT}(g_i)$ are not sparse at all. Hence the matrix multiplication in (2) can have complexity comparable to downsampling in spatial domain (by first taking inverse DCT, filtering and taking DCT) using fast algorithms for computation of the DCT. In [6] a fast algorithm has been developed for the computation of (2) based on factorization of the DCT matrix corresponding to the Winograd algorithm. However their approach is mainly aimed at the downsampling matrix in Eq. (1) and it has been commented in that paper that it is not guaranteed that for every reasonable anti-aliasing filter the method would have lesser complexity than spatial domain methods.

Hence we see that the previous approaches view the downsampling operation as lowpass filtering followed by downsampling and try to implement this in the DCT domain. The lowpass filter is chosen independent of the DCT transform. In our approach we shall show that it is possible to design a low-pass filter so that DCT of the filter matrix is sparse rather than the filter matrix itself.

We shall first give an outline of our scheme for 1-D signals. Let $\mathbf{B}_1$ and $\mathbf{B}_2$ denote the 8-point DCT of two consecutive 8-sample blocks $\mathbf{b}_1$ and $\mathbf{b}_2$. We wish to generate the 8-point DCT $\mathbf{B}$ of the 8-point block got by downsampling ($\mathbf{b}_1$, $\mathbf{b}_2$) with an appropriate filter. The downsampling has to carried out as far as possible in the compressed domain. In principle the proposed scheme can be viewed as follows: Take 4-point inverse DCT of the 4 lowpass coefficients in $\mathbf{B}_1$ and similarly for $\mathbf{B}_2$. Concatenate these two 4-point blocks and then take its 8-point DCT. The resulting block is the desired block $\mathbf{B}$. In section 2 we shall describe an algorithm for implementing this operation and show that it is computationally inexpensive compared to matrix multiplication in Eq. (2). The scheme works because taking 4 point inverse DCT of the 4 lowpass coefficients of 8-point DCT of a block gives a low passed and downsampled version of the original 8-point block [10].

2 Downsampling in the DCT domain

In this section we shall elaborate on the downsampling scheme and give a computationally efficient algorithm for it. We shall derive the relevant equations in 1-D and then extend them to 2-D. As in the previous section let $\mathbf{b}_1$ and $\mathbf{b}_2$ denote two consecutive 8-pixel blocks in the spatial domain. Let $\mathbf{B}_1$ and $\mathbf{B}_2$ be their 8-point DCTs respectively. Let $\mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}$. Let $\hat{\mathbf{B}}_1$ and $\hat{\mathbf{B}}_2$ denote the first 4 (low-pass) components of $\mathbf{B}_1$ and $\mathbf{B}_2$. Let $\hat{\mathbf{b}}_1$ and $\hat{\mathbf{b}}_2$ denote the 4-point inverse DCT of $\hat{\mathbf{B}}_1$ and $\hat{\mathbf{B}}_2$. Hence $\hat{\mathbf{b}}_1$ and $\hat{\mathbf{b}}_2$ are low-passed and downsampled versions of $\mathbf{b}_1$ and $\mathbf{b}_2$ respectively. Let $\hat{\mathbf{b}} = \begin{bmatrix} \hat{\mathbf{b}}_1 \\ \hat{\mathbf{b}}_2 \end{bmatrix}$ and let $\hat{\mathbf{B}}$ be the 8-point DCT of $\hat{\mathbf{b}}$. Hence $\hat{\mathbf{b}}$ is a low-pass filtered downsampled version of $\mathbf{b}$. We need to compute $\hat{\mathbf{B}}$ directly from $\mathbf{B}_1$ and $\mathbf{B}_2$ (i.e. from $\hat{\mathbf{B}}_1$ and $\hat{\mathbf{B}}_2$). Let T (8×8) denote the 8-point DCT operator matrix and let T_4 (4×4) denote the 4-point DCT operator matrix. Then we have the following equation:

$$\begin{aligned} \hat{\mathbf{B}} = T\hat{\mathbf{b}} &= T \begin{bmatrix} \hat{\mathbf{b}}_1 \\ \hat{\mathbf{b}}_2 \end{bmatrix} = [T_L \quad T_R] \begin{bmatrix} T_4^t\hat{\mathbf{B}}_1 \\ T_4^t\hat{\mathbf{B}}_2 \end{bmatrix} \\ &= T_L T_4^t \hat{\mathbf{B}}_1 + T_R T_4^t \hat{\mathbf{B}}_2 \end{aligned} \quad (4)$$

where T_L and T_R are 8×4 matrices denoting the first and last four columns respectively of the 8-point DCT operator T . Let us have a look at $T_L T_4^t$ and $T_R T_4^t$ and see how about 50% of the terms in the product turn out to be zeros (see also Eq. (3) at top): [1]

$$T_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & -\cos\frac{3\pi}{8} & -\cos\frac{\pi}{8} \\ \cos\frac{2\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{6\pi}{8} & \cos\frac{2\pi}{8} \\ \cos\frac{3\pi}{8} & \cos\frac{9\pi}{8} & -\cos\frac{9\pi}{8} & -\cos\frac{3\pi}{8} \end{pmatrix} \quad (5)$$

The following points can be noted immediately from Eqs. (3) and (5) :

[1] The normalizing constants in these matrices have been omitted since they do not affect our conclusions.

$$T = [T_L \mid T_R] = \left(\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \cos\frac{\pi}{16} & \cos\frac{3\pi}{16} & \cos\frac{5\pi}{16} & \cos\frac{7\pi}{16} & -\cos\frac{7\pi}{16} & -\cos\frac{5\pi}{16} & -\cos\frac{3\pi}{16} & -\cos\frac{\pi}{16} \\ \cos\frac{\pi}{8} & \cos\frac{3\pi}{8} & -\cos\frac{3\pi}{8} & -\cos\frac{\pi}{8} & -\cos\frac{\pi}{8} & -\cos\frac{3\pi}{8} & \cos\frac{3\pi}{8} & \cos\frac{\pi}{8} \\ \vdots & & & & & & & \vdots \end{array} \right) \quad (3)$$

1. For $k = 0, 1, 2, 3$, the $(2k)$th row of T_L is same as the kth row of T_4 and the the $(2k)$th row of T_R is negative of the kth row of T_4 .

2. Since the rows of T_4 are orthogonal, point 1 above implies that *every* $2k$th row of T_L (and also of T_R) is orthogonal to *every* row of T_4 except the kth row for $k = 0, 1, 2, 3$. Hence in the 8×4 matrices $T_L T_4^t$ and $T_R T_4^t$ every $(2k)$th row has all its entries as zeros except the kth entry. Hence we see that about 50% of the entries of these matrices are zero.

3. Odd rows of T are anti-symmetric and even rows are symmetric. Also odd rows of T_4 are anti-symmetric and even rows are symmetric. These two facts together imply that all the corresponding entries of $T_L T_4^t$ and $T_R T_4^t$ are identical except possibly for a change of sign. Specifically $T_L T_4^t(i,j) = (-1)^{i+j} T_R T_4^t(i,j)$ for $i = 0, 1, \ldots, 7$ and $j = 0, 1, 2, 3$. This fact can be exploited to further reduce the computations in Eq. (4). Grouping the terms for which $i + j$ is even into one matrix C and the remaining terms into another matrix D we have $T_L T_4^t = C + D$ and $T_R T_4^t = C - D$. Hence from Eq. (4) we have:

$$\mathbf{B} = (C + D)\hat{\mathbf{B}}_1 + (C - D)\hat{\mathbf{B}}_2 \quad (6)$$

$$= C(\hat{\mathbf{B}}_1 + \hat{\mathbf{B}}_2) + D(\hat{\mathbf{B}}_1 - \hat{\mathbf{B}}_2) \quad (7)$$

Eq. (7) is computationally much faster compared to Eq. (6) because each of the matrices C and D have only half the number of non-zero entries compared to $C + D$ or $C - D$. Hence the number of multiplications involved is reduced drastically. Another way to look at this is that instead of computing expressions like $\alpha\beta + \alpha\gamma$ in Eq. (6) we are computing $\alpha(\beta + \gamma)$ in Eq. (7).

4. Since T and T_4 are DCT matrices it is our belief that much faster procedures (based on fast DCT algorithms [11]) could be designed for the scheme (cf. Eq. (4)) presented here. But we shall not pursue this in the current paper.

Using the facts mentioned above it can be shown that for the 2-D case our downsampling scheme requires 1.25 multiplications and 1.25 additions per pixel of the original image.

2.1 The Downsampling Filter

In this section we shall derive the downsampling filter which corresponds to the downsampling operation mentioned above. We have the following equation:

$$\hat{\mathbf{b}}_1 = T_4^t \hat{\mathbf{B}}_1 = T_4^t [I \;\; O] \mathbf{B}_1 = T_4^t [I \;\; O] T \mathbf{b}_1 = T_4^t M^t T \mathbf{b}_1 \quad (8)$$

where I and O denote 4×4 identity and zero matrices respectively and $M \stackrel{\text{def}}{=} \left[\begin{array}{c} I \\ O \end{array} \right]$. Hence we have

$$\left[\begin{array}{c} \hat{\mathbf{b}}_1 \\ \mathbf{0} \end{array} \right] = M T_4^t M^t T \mathbf{b}_1 \stackrel{\text{def}}{=} h \mathbf{b}_1 \quad (9)$$

where $\mathbf{0}$ is a 4×1 zero vector. Hence we see that the downsampling filter matrix $h(8 \times 8)$ is given by

$$h = M T_4^t M^t T \quad (10)$$

Compare this with the filter matrix h_1 in Eq. (1). h_1 is chosen without considering the fact that finally in Eq. (2) it is the DCT of h_1 that is used. It is more desirable to have DCT(h_1) to be sparse than to have h_1 as sparse. In our approach the downsampling filter h in Eq. (10) is derived from the DCT basis functions so that DCT(h) is guaranteed to be very sparse:[2]

$$\text{DCT}(h) = T h T^t = [T_L \;\; T_R] M T_4^t M^t T T^t = T_L T_4^t M^t \quad (11)$$

We have already seen above that $T_L T_4^t$ is very sparse. The M^t at the end makes sure that only the first four low-pass components of DCT$(\mathbf{b}_1)$ are used in the computation of $\hat{\mathbf{b}}_1$. The product $T_R T_4^t$ appears in the computation of $\left[\begin{array}{c} \mathbf{0} \\ \hat{\mathbf{b}}_2 \end{array} \right]$ which when added to $\left[\begin{array}{c} \hat{\mathbf{b}}_1 \\ \mathbf{0} \end{array} \right]$ gives the desired low-pass downsampled vector $\hat{\mathbf{b}}$.

2.2 Extension to 2-D

Let $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ and $\mathbf{b}_4$ denote four consecutive 8×8 blocks numbered in the same way as c_1, c_2, c_3 and c_4 in Fig. 1. Following the same notation as in 1-D case, let $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$ and $\mathbf{B}_4$ denote the DCTs of these blocks respectively. Let $\hat{\mathbf{B}}_1$ etc. denote the 4×4 matrices containing the low-pass coefficients of $\mathbf{B}_1$ etc. Let $\hat{\mathbf{b}}_1$ etc. denote the 4×4 inverse DCT of $\hat{\mathbf{B}}_1$ etc. Then $\hat{\mathbf{b}} \stackrel{\text{def}}{=} \left[\begin{array}{cc} \hat{\mathbf{b}}_1 & \hat{\mathbf{b}}_2 \\ \hat{\mathbf{b}}_3 & \hat{\mathbf{b}}_4 \end{array} \right]$ denotes the low-pass and downsampled version of $\mathbf{b} \stackrel{\text{def}}{=} \left[\begin{array}{cc} \mathbf{b}_1 & \mathbf{b}_2 \\ \mathbf{b}_3 & \mathbf{b}_4 \end{array} \right]$. Let $\hat{\mathbf{B}} \stackrel{\text{def}}{=} \text{DCT}(\hat{\mathbf{b}})$. We need to compute $\hat{\mathbf{B}}$ directly from $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$ and $\mathbf{B}_4$ (i.e. from $\hat{\mathbf{B}}_1$, $\hat{\mathbf{B}}_2$, $\hat{\mathbf{B}}_3$ and $\hat{\mathbf{B}}_4$). We have

$$\hat{\mathbf{B}} = T \hat{\mathbf{b}} T^t \quad (12)$$

$$= [T_L \;\; T_R] \left[\begin{array}{cc} \hat{\mathbf{b}}_1 & \hat{\mathbf{b}}_2 \\ \hat{\mathbf{b}}_3 & \hat{\mathbf{b}}_4 \end{array} \right] \left[\begin{array}{c} T_L \\ T_R \end{array} \right] \quad (13)$$

$$= [T_L \;\; T_R] \left[\begin{array}{cc} T_4^t \hat{\mathbf{B}}_1 T_4 & T_4^t \hat{\mathbf{B}}_2 T_4 \\ T_4^t \hat{\mathbf{B}}_3 T_4 & T_4^t \hat{\mathbf{B}}_4 T_4 \end{array} \right] \left[\begin{array}{c} T_L \\ T_R \end{array} \right] \quad (14)$$

$$= (T_L T_4^t) \hat{\mathbf{B}}_1 (T_L T_4^t)^t + (T_L T_4^t) \hat{\mathbf{B}}_2 (T_R T_4^t)^t + (T_R T_4^t) \hat{\mathbf{B}}_3 (T_L T_4^t)^t + (T_R T_4^t) \hat{\mathbf{B}}_4 (T_R T_4^t)^t \quad (15)$$

[2] Note that DCT$(h\mathbf{b}_1) = Th\mathbf{b}_1 = ThT^tT\mathbf{b}_1 =$ DCT(h)DCT$(\mathbf{b}_1)$.

We have already seen that $(T_L T_4^t)$ and $(T_R T_4^t)$ are very sparse. We know from 1-D case that $(T_L T_4^t) = C + D$ and $(T_R T_4^t) = C - D$ where each of C and D contain only half as many non-zero entries as $(T_L T_4^t)$ or $(T_R T_4^t)$. Hence it can be shown that

$$\hat{\mathbf{B}} = (X+Y)C^t + (X-Y)D^t \quad (16)$$

where

$$X = C(\hat{\mathbf{B}}_1 + \hat{\mathbf{B}}_3) + D(\hat{\mathbf{B}}_1 - \hat{\mathbf{B}}_3) \quad (17)$$

$$Y = C(\hat{\mathbf{B}}_2 + \hat{\mathbf{B}}_4) + D(\hat{\mathbf{B}}_2 - \hat{\mathbf{B}}_4) \quad (18)$$

We know from the 1-D case that Eqs. (17) and (18) are computationally inexpensive. Again from 1-D case we know that Eq. (16) is also computationally inexpensive. Hence for the 2-D case the procedure is:[3] compute X and Y as in Eqs. (17) and (18) and then compute $\hat{\mathbf{B}}$ as in Eq. (16). As mentioned before, it can be shown that our downsampling scheme requires 1.25 multiplications and 1.25 additions per pixel of the original image. Compare this to 4 multiplications and 4.75 additions (after assuming that the DCT of any 8×8 image block is 75% sparse) per pixel of original image required by the method in [7].

3 Upsampling

In many applications it is required to obtain an upsampled version from a downsampled version of an original image. For example for spatially scalable encoding of video [12, 13] a prediction of the original frame is obtained by upsampling the lower resolution frame and the difference is encoded in the enhancement or low-priority layer. A natural question is can we design an upsampling scheme that works in the compressed domain and can keep all the low-frequency components of the original image in the upsampled image. In terms of our notation in the previous section the problem is: can we get back $\hat{\mathbf{B}}_1$, $\hat{\mathbf{B}}_2$, $\hat{\mathbf{B}}_3$ and $\hat{\mathbf{B}}_4$ from $\hat{\mathbf{B}}$ (see Eq. (15)). Since the matrices T and T_4 are unitary the following inverse relationships can be easily derived from Eq. (15):[4]

$$\hat{\mathbf{B}}_1 = (T_L T_4^t)^t \hat{\mathbf{B}} (T_L T_4^t) \quad (19)$$

$$\hat{\mathbf{B}}_2 = (T_L T_4^t)^t \hat{\mathbf{B}} (T_R T_4^t) \quad (20)$$

$$\hat{\mathbf{B}}_3 = (T_R T_4^t)^t \hat{\mathbf{B}} (T_L T_4^t) \quad (21)$$

$$\hat{\mathbf{B}}_4 = (T_R T_4^t)^t \hat{\mathbf{B}} (T_R T_4^t) \quad (22)$$

[3] Actually due to the different sizes of the DCTs there would be a factor of half that we have not accounted for so far. Hence in practice the actual equations to be used for X and Y would be: $X = \frac{1}{2}[C(\hat{\mathbf{B}}_1 + \hat{\mathbf{B}}_3) + D(\hat{\mathbf{B}}_1 - \hat{\mathbf{B}}_3)]$ and $Y = \frac{1}{2}[C(\hat{\mathbf{B}}_2 + \hat{\mathbf{B}}_4) + D(\hat{\mathbf{B}}_2 - \hat{\mathbf{B}}_4)]$. Note that the factor of $\frac{1}{2}$ can be absorbed in C and D which would be pre-computed.

[4] As mentioned in footnote 3 we have to account for a factor of half due to different sizes of the DCTs involved and hence in practice the equations used for upsampling would be $\hat{\mathbf{B}}_1 = 2(T_L T_4^t)^t \hat{\mathbf{B}} (T_L T_4^t)$ etc.

Then

$$\tilde{\mathbf{B}}_1 \stackrel{\text{def}}{=} \begin{bmatrix} \hat{\mathbf{B}}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{bmatrix} \quad (23)$$

etc. give us the four blocks in the upsampled image corresponding to the block $\hat{\mathbf{B}}$ in the downsampled image. Note that the process of downsampling and then upsampling has the effect of truncating to zero all the high frequency components in each 8×8 DCT block in the original image while preserving all the low-frequency components of such blocks. Also note the upsampling scheme can be applied to any given image irrespective of whether or not the given image is obtained by the downsampling scheme outlined in Section 2. It can be shown that our upsampling scheme requires 1.25 multiplications and 1.50 additions per pixel of the upsampled image.

4 Results

We have already shown the computational efficiency of our method to be superior to existing methods for downsampling in the DCT domain. That is the main contribution of this paper. Moreover, since we preserve most of the significant energy (the low-pass coefficients of each block) of the image, the image obtained after downsampling and upsampling looks much sharper compared to the image obtained after bilinear interpolation. The later looks blurred. Comparing the PSNR shows that the image obtained by our downsampling and upsampling scheme is typically about 4 dB better than the image obtained by bilinear interpolation. This is an additional advantage of our method.

Fig. 3 shows the lena image after processing by our scheme and the bilinear interpolation scheme. We see that the image obtained by our method is much sharper compared to the bilinearly interpolated image which looks blurred. This is particularly noticeable in the hair and the eye regions. The PSNR values are given in Table 1 for different sets of operations on various original images. We see an improvement of 4.65 dB with our method compared to the conventional bilinear interpolation method [9, 6, 7]. The downsampled images using our method and the method using bilinear downsampling filter look equally good and are not shown.

Figure 3 also shows similar results for a watch image. The PSNR values for these images are given in Table 1. Note that the numbers on the three circular dials are much clearer in our method than the one obtained by bilinear interpolation. Similarly the numbers on the rim of the watch appear much sharper. Similar results are obtained for the cap image shown in Figure 2. It is seen that more texture is retained by our method than by bilinear interpolation.

Comparing the columns of dct-dct and con-con in Table 1 we see that our scheme gives about 3 to 4

dB improvement compared to using bilinear scheme. Note that bilinear scheme is extensively used in papers reporting downsampling in the compressed domain [9, 6, 7]. Also note that the performance of con-dct is very close to that of dct-dct and that of dct-con is very close to that of con-con. This implies that in terms of PSNR the upsampling scheme is the deciding factor. This in turn implies that we could choose the downsampling scheme that is faster in the domain in which the original images (video frames) are available. For example if the original frames are available in compressed format (e.g. motion JPEG) we should use our proposed scheme which is faster in the compressed domain (since the DCT of the downsampling filter matrix is sparse). On the other hand, if the original frames are available in the raw format we should use the bilinear scheme which is faster in the spatial domain. In either case the upsampling scheme described in this paper should be used to minimize the energy in the residual image.

References

[1] T. Lindeberg, *Scale-space theory in computer vision.* Boston: Kluwer Academic, 1994.

[2] A. P. Witkin, "Scale-space filtering," in *Proc. 8th IJCAI*, pp. 1019–1022, Aug. 1983.

[3] A. L. Yuille and T. A. Poggio, "Scaling theorems for zero crossings," *PAMI*, vol. 8, pp. 15–25, Jan. 1986.

[4] J. Babaud, A. P. Witkin, M. Baudin, and R. O. Duda, "Uniqueness of the gaussian kernel for scale-space filtering," *PAMI*, vol. 8, pp. 26–33, Jan. 1986.

[5] A. Tran and K.-M. Liu, "An efficient pyramid image coding system," in *ICASSP*, pp. 744–747, 1987.

[6] N. Merhav and V. Bhaskaran, "Fast algorithms for DCT-domain image down-sampling and for inverse motion compensation," *IEEE Trans. CSVT*, vol. 7, pp. 468–476, June 1997.

[7] S.-F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE JSAC*, vol. 13, pp. 1–11, Jan. 1995.

[8] B. C. Smith and L. Rowe, "Algorithms for manipulating compressed images," *IEEE Comput. Graph. Applicat. Mag.*, vol. 13, pp. 34–42, Sept. 1993.

[9] Q. Hu and S. Panchanathan, "Image/video spatial scalability in compressed domain," *IEEE Transactions on Industrial Electronics*, vol. 45, pp. 23–31, Feb. 1998.

[10] K. N. Ngan, "Experiments on two-dimensional decimation in time and orthogonal transform domains," *Signal Processing*, vol. 11, pp. 249–263, 1986.

[11] W. B. Pannebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard.* New York: Van Nostrand Reinhold, 1993.

[12] A. Puri and A. Wong, "Spatial domain resolution scalable video coding," in *Proc. SPIE Visual Communications and Image Processing*, (Boston MA), pp. 718–729, Nov. 1993.

[13] ISO/IEC 13818-2 — Rec. ITU-T H.262, *Generic coding of moving pictures and associated audio*, Nov. 1994.

Image	dct-dct	con-con	con-dct	dct-con
Lena	34.69	30.04	34.06	30.09
Watch	29.09	25.15	28.63	25.25
Cap	34.22	32.08	33.73	32.17

Table 1: PSNR values with respect to the original image and images obtained by downsampling the original image and upsampling the downsampled image. There are 4 possible ways of doing this depending on whether we use our scheme or bilinear scheme for downsampling and upsampling respectively. dct-dct (con-con) refers to the case in which our (bilinear) scheme is used for both downsampling and upsampling. con-dct (dct-con) refers to using bilinear (our) scheme for downsampling and our (bilinear) scheme for upsampling.

(a)

(b)

Figure 2: (a) Cap downsampled and upsampled by our method (b) Cap downsampled and upsampled using bilinear interpolation [9, 7]. We see that the image in (a) is sharper compared to that in (b) which is blurred. For example the letters on the white cap are much sharper in (a). Also (a) has more texture on the left side of the image.

Figure 3: Lena downsampled and upsampled using (a) our method (b) using bilinear interpolation [9, 7]. We see that the image in (a) is sharper compared to that in (b) which is blurred. Watch downsampled and upsampled using (c) our method (d) using bilinear interpolation. We see that the number appear sharper in (c). Results for all these images have been tabulated in Table 1